UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación
Máster Universitario en Ingeniería y Tecnología de Sístemas Software

Iberian Odyssey. Diseño y desarrollo de una aplicación de Realidad Aumentada con Geolocalización

Trabajo Fin de Master

Autor : Vicente Murgui Sanchis

Tutor: Francisco Javier Jaén Martinez

Curso: 2020 - 2021

RESUMEN

La rápida evolución tecnológica que se ha producido en los últimos años ha desarrollado una gran variedad de nuevas tecnologías informáticas y de la comunicación, entre ellas se encuentra la Realidad Aumentada. La utilización de esta tecnología es el punto de partida de este trabajo.

El presente proyecto tiene como objetivo el desarrollo de una aplicación de realidad aumentada en dispositivo móvil que emplee las coordenadas de geolocalización del GPS para poder integrar en nuestro mundo distintos elementos con los que el usuario pueda interactuar. El proyecto también hace hincapié en como el usuario puede interactuar con estos elementos o determinadas acciones que impliquen la detección de superficies planas como paredes o suelos, añadiéndose esta información al contexto sobre el que trabaja la aplicación. Para esta aplicación se ha escogido un enfoque educativo que permita a los usuarios introducirse a la mitología ibérica mediante actividades de entretenimiento.

AGRADECIMIENTOS

Mi agradecimiento a mi tutor de Trabajo Fin de Máster D. Francisco Javier Jaén Martínez por su guía y ayuda.

A Pere Andreu Gómez por componer la música, a Laia Fuster Corral por su ayuda en el modelado 3D y a todos los testers por su colaboración en el desarrollo de este proyecto.

A mi familia y amigos por su apoyo incondicional y animarme a alcanzar mis objetivos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Capítulo 1. INTRODUCCIÓN

El rápido avance de las tecnologías informáticas hace que estas sean desarrolladas y perfeccionadas continuamente con el fin de optimizar la experiencia de los usuarios. Este avance afecta tanto a las aplicaciones que son creadas como a la forma en la que los usuarios interactúan con dichas aplicaciones.

Este Trabajo de Fin de Máster titulado "Iberian Odyssey. Diseño y desarrollo de una aplicación de Realidad Aumentada con Geolocalización" es un proyecto realizado con el objetivo inicial de desarrollar una aplicación de realidad aumentada que emplee distintas mecánicas con las que explorar las posibilidades de interacción y combinando con la tecnología de geolocalización que permitirá integrar elementos digitales en lugares definidos con el fin de optimizar el efecto de inmersión del usuario en la cultura mitológica de la Península Ibérica.

Además, se busca dotar a la aplicación de un enfoque al sector del entretenimiento dirigido a áreas de la educación, introduciendo a los usuarios en la cultura de la mitología de la Península Ibérica para así dar a conocer y preservar estos conocimientos.

En este trabajo también se propone evaluar el grado de satisfacción por parte de los usuarios de las funcionalidades presentadas mediante pruebas de usabilidad y la carga mental que la propia aplicación ejerce sobre dichos usuarios.

Para llevar esto a cabo, se ha desarrollado la aplicación para dispositivos móviles que permite al usuario interactuar de distintas formas con los elementos que se encuentre dentro de la dimensión digital de la aplicación.

1.1. Contexto

La Realidad Aumentada (RA) es una tecnología relativamente reciente, a principios del siglo XX se le consideraba un término de ciencia-ficción, a través de los años esta tecnología ha ido evolucionando gracias a la investigación y desarrollo hasta llegar al momento actual.

Entre las numerosas definiciones de RA que existen se puede concluir que es un conjunto de tecnologías que hacen posible que el usuario pueda visualizar el mundo real a partir de un dispositivo que añade información virtual a la realidad. Esto permite que los elementos físicos que hay en el mundo real se mezclen con los elementos virtuales que el dispositivo o conjuntos de dispositivos.

La realidad aumentada es una tecnología emergente que está alcanzando un gran protagonismo en diversos aspectos de la vida cotidiana debido a que combina el mundo real y el virtual, la interacción se realiza en tiempo real, se adapta al entorno en que se trabaja y la imagen se presenta mediante modelos 3D.
Esta tecnología permite la inserción de imágenes virtuales en espacios reales, lo que permite que los usuarios consigan interactuar con elementos digitales combinando la

dimensión virtual representada por todos los elementos digitales que provee la aplicación y la física que está constituida por los elementos del mundo real los cuales son captados por dispositivos como cámaras.

La Realidad Aumentada ha experimentado un gran auge durante los últimos años gracias a la creación de diversas aplicaciones que han ganado mucha fama debido a la originalidad de sus métodos de empleo y de cómo sumerge al usuario en un entorno gracias al tipo de interacción que la Realidad Aumentada propone. Aun con todo, todavía le queda un largo recorrido a esta tecnología para que pueda alcanzar su máximo potencial.

1.2. Objetivos del proyecto

Con este Trabajo Final de Máster se pretende alcanzar los siguientes objetivos:

- **Desarrollar** un prototipo de una aplicación de realidad aumentada para dispositivos Android que emplee geolocalización para posicionar los elementos digitales de la aplicación. entretenimiento
- **Implementar** una serie de mecánicas distintas que permitan al usuario interactuar de diferentes formas con los elementos digitales presentados en la aplicación desarrollada.
- **Crear** una aplicación con la que introducir a los usuarios conocimientos básicos sobre la mitología ibérica con el fin de preservar dicho conocimiento aportándole a la aplicación un enfoque educativo para usuarios de distintas edades.
- **Evaluar** si las mecánicas presentadas en la aplicación son satisfactorias para los usuarios mediante una prueba de usabilidad y la cantidad de carga mental que la propia aplicación ejerce sobre los usuarios empleando el test de Carga Mental NASA-TLX.

1.3. Estructura del proyecto

El proyecto está organizado en capítulos enumerados seguidos de un anexo:

- ***Capítulo 1*, Introducción**: presenta el proyecto que se ha desarrollado.
- ***Capítulo 2*, Estado del arte**: detalla conocimientos imprescindibles para el entendimiento del proyecto.
- ***Capítulo 3*, Análisis**: se definen los requisitos de usuario, tanto funcionales como no funcionales, y el diagrama de clases del proyecto.
- ***Capítulo 4*, Diseño e Implementación**: muestra las distintas tecnologías y conocimientos empleados durante el desarrollo del proyecto, así como el diseño de la interfaz de usuario y la implementación de la propia experiencia de realidad aumentada.
- ***Capítulo 5*, Pruebas y resultados**: describe los distintos tipos de evaluaciones a las que se ha sometido la aplicación y se detallan los resultados de estas.
- ***Capítulo 6*, Conclusiones y trabajos futuros**: muestra las conclusiones a las que se llegan tras la realización de las evaluaciones anteriores y se hace una mención a las posibles mejoras que se podrían implementar en el proyecto.

- ***Capítulo 7*, Bibliografía**: un listado con las distintas fuentes bibliográficas empleada durante el desarrollo del proyecto.
- ***Anexo 1*, Scripts implementados**: Se muestran los distintos scripts que han sido implementados para desarrollar la aplicación.
- ***Anexo 2, Test realizados por los usuarios***: Se muestran los distintos test que han realizado los usuarios durante la prueba de usabilidad y la prueba NASA-TLX.

Capítulo 2. ESTADO DEL ARTE

2.1. Realidad aumentada en dispositivos móviles

El escultor y pintor Josep María Subirach esculpió en un mural “Lo que se ve es una porción de lo invisible”, donde apuntaba a la existencia de una realidad perceptible y otra no visible [1]. Esta cita es oportuna para aplicarla a la RA, cuando se define como “Aquella tecnología que permite agregar información adicional a una imagen del mundo real cuando ésta se visualiza a través de un dispositivo”. [2]

El uso del RA en dispositivos móviles permite crear aplicaciones innovadoras que plantean una serie de funcionalidades con las que una aplicación estándar no puede rivalizar, esto es debido a que las aplicaciones de realidad aumentada permiten al usuario explorar su entorno moviéndose por distintos espacios en donde interactuar con los elementos digitales que proporciona la aplicación. Tim Cook, de Apple manifestó: "considero que la RA puede ser el teléfono inteligente para todos. Creo que la RA es así de grande, es enorme"[3].

La evolución de los dispositivos móviles ha sido muy rápida y su uso se ha generalizado transformándose una herramienta indispensable para los individuos en su actividad profesional, social, de entretenimiento, etc. En este sentido, cobra especial importancia la universalización del uso de los teléfonos móviles para el acceso a internet, el concepto de geolocalizacion y la existencia de aplicaciones que pueden ser usadas en estos dispositivos, ya que la realidad aumentada se puede visualizar a través de dispositivos portátiles como smartphones o tablets.

La facilidad de acceso y el uso habitual de teléfonos, tablets, etc., ha originado que los usuarios utilicen estas plataformas tanto o más que los ordenadores, lo que ha hecho que el desarrollo del software se está orientando cada vez más a estas plataformas.

La evolución de RA se ha producido principalmente en los últimos 100 años. En 1916 Albert B.Pratt creó el primer HMD (Heas Mounted- display) , patentó un sistema de periscopio situado en un casco para poder ver las trincheras. Mucho más tarde Morton Heiling en 1956, desarrollo un simulador multimodal, Sensorama, donde el usuario logra una experiencia través de sensaciones visuales, sonoras y olfativas. Cinco años más tarde en 1961 Comeau y Brian de Philco crearon un HMD binocular, que se podía utilizar como cámara de video remota mediante un sensor de orientación electromagnético, ya en 1973 Myron W. Krueger creó el laboratorio de realidad artificial Videoplace que permitía a los usuarios interactuar con objetos virtuales, esta fue la primera instalación de realidad aumentada que combinaba la utilización de cámaras de video con un sistema de proyección que reconocía a los movimientos de los usuarios por medio de sombras y movimiento.

En tiempos más recientes, Tom Caudell en 1990 acuño el término RA y describió una aplicación de apoyo al montaje de cableados eléctricos complejos en la empresa

Boeing. En 1992 Louis Rosenberg diseñó un sistema de inmersión total Virtual Fixture donde se combinaba RA y RV. Algunos años después en 1998 la Nasa utilizó por primera vez la RA en la nave espacial X-38.

Ya en el siglo actual se han realizado muchos grandes avances en la RA entre los cuales se encuentran el realizado en el año 2000 por B. Thomas creó el juego de móvil con realidad aumentada llamado ARQuake, donde los jugadores usaban una pantalla montada en la cabeza, llevaban una mochila con giroscopios y un ordenador. En los años posteriores se desarrollaron aplicaciones con fines comerciales y de entretenimiento, IKEA desarrollo una aplicación que usaba la RA para poder situar mobiliario virtual en una casa para ver cómo quedaba y en 2016 con lanzamiento del juego Pokemon GO se dio a conocer y hacerse famosa la tecnología de la RA, lo que oriento el desarrollo de la realidad aumentada principalmente al sector del ocio y los videojuegos.

Con cada año que pasa, la realidad aumentada gana más protagonismo en distintos ámbitos debido a su versatilidad y las múltiples posibilidades que ofrece en diversos sectores de la vida. Los campos de aplicación son muy variados, como ejemplo de ellos están: el sanitario, el turístico, el marketing, la comunicación, la arquitectura, la manufactura, el militar el ocio, etc. destacamos entre todos ellos el ámbito educativo por tener este proyecto una orientación lúdico-educativa.

A continuación, se detalla como la utilización de la realidad aumentada puede ser útil para la educación y para algunas investigaciones realizadas en dicho ámbito. También se explicará un aspecto muy importante de la realidad aumentada para este proyecto que es la sensibilidad al contexto, la cual es aplicada muchas veces en los conocidos "location-based games" o juegos basados en la ubicación, en donde se mostrarán varias investigaciones que abordan este punto.

2.1.1. RA en el ámbito educativo

La educación es una de las áreas que más se ha beneficiado de la RA, ha demostrado ser una herramienta con un enorme potencial, tanto para los alumnos como para los profesores en el proceso de enseñanza-aprendizaje. El alumno estudia de forma diferente ya que la RA le proporciona una formación más personalizada, originando experiencias inmersivas, interactivas y visuales.

Una de las aplicaciones de RA empleadas en la educación más conocidas actualmente es Magic Book [4] del grupo HIT de Nueva Zelanda. Esta aplicación consiste en un libro real que, al emplear un visualizador de mano sobre las páginas de éste, el usuario puede visualizar una escena de realidad aumentada que le permite experimentarla de forma más inmersiva. Estas escenas de carácter educativo pueden aplicarse para enseñar distintas materias a los alumnos, como el funcionamiento de los volcanes, las partes de un vehículo, etc.

Ilustración 1: Dos usuarios usando Magic Book para instruirse sobre volcanes

A continuación, se muestran una serie de investigaciones sobre el empleo de la realidad aumentada en el campo educativo:

- **Realidad Aumentada en la Educación: una tecnología emergente** [5]: presenta la RA como una tecnología emergente que puede ser empleada en la enseñanza y muestra distintas implementaciones de esta aplicada a distintas áreas de la educación.
- **Realidad aumentada y educación: análisis de experiencias prácticas** [6]: detalla varios proyectos que han sido llevados a cabo durante los últimos años en distintos centros educativos con el fin de obtener un punto de vista general del estado del arte del uso de aplicaciones de realidad aumentada en el ámbito de la educación en España.
- **Realidad Aumentada en Educación Primaria: efectos sobre el aprendizaje** [7]**:** realiza un experimento en el que durante 5 semanas se impartían a dos grupos de alumnos de 6º de primaria un determinado temario. En uno de los grupos se empleaban herramientas de RA durante las clases para comprobar si estas influían en la adquisición de conocimientos por parte del alumno. El experimento concluyó que había una mejora en las calificaciones del grupo que empleó la tecnología RA
- **Augmented reality in education: current technologies and the potential for education** [8]**:** proporciona una introducción a la tecnología de realidad aumentada y las posibilidades que esta ofrece en el ámbito de la enseñanza.
- **Current status, opportunities and challenges of augmented reality in education** [9]**:** analiza las tecnologías de la RA identificando sus características y prestaciones y sacando a la luz no solo las oportunidades que esta ofrece, sino también los desafíos que la realidad aumentada puede crear a los educadores. El artículo trata también de aportar posibles soluciones a los desafíos de la RA.
- **Augmented Reality in education** [10]**:** presenta el concepto de realidad aumentada y da algunos ejemplos de su uso en cursos universitarios. También describe un experimento realizado con el fin de ver como el uso de la realidad

aumentada puede elevar las calificaciones de los estudiantes mostrando unos resultados muy positivos.

- **Realidad Aumentada aplicada a la enseñanza de Ciencias Naturales** [11]**:** muestra los resultados obtenidos de un experimento realizado en 2014 en donde por medio de una experiencia de realidad aumentada, se intenta que los alumnos comprendan más fácilmente conceptos y temas que son dados en Ciencias Naturales en primaria.

2.2. Realidad aumentada sensible al contexto y juegos basados en la ubicación

El objetivo de emplear sensibilidad al contexto en aplicaciones de realidad aumentada es aportar una mayor calidad a la experiencia del usuario mientras emplea la aplicación.

El contexto hace referencia a aquella información que permite a la aplicación conocer en qué situación se encuentra una entidad y que pueda ser relevante para mejorar la interacción entre el usuario y la aplicación. Esta información puede traducirse en información sobre que objetos hay en una estancia y como están distribuidos, las coordenadas geográficas en las que se encuentra una determinada entidad o incluso la cantidad de luz hay en una habitación.

Los dispositivos móviles actualmente poseen una gran cantidad de sensores (cámaras, giroscopio ...) que permiten a éstos obtener mucha información del entorno, que puede ser empleada para enriquecer las experiencias que la realidad aumentada puede ofrecer. Esto ha permitido el desarrollo de aplicaciones conocidas como "location-based games" o juegos basados en la ubicación. Estos juegos emplean la ubicación del usuario obtenida por medio del GPS para saber dónde se encuentra el usuario, estos juegos también son conocidos como "street games" o "urban game" debido a que generalmente son juegos pensados para ser jugados en las calles.

Un ejemplo muy conocido es el Pokémon GO [12] que en los últimos años ha sido un juego que ha dado a conocer al público general la existencia de la realidad aumentada. La aplicación tiene en cuenta donde se encuentra el usuario por medio de su posición GPS para permitirle visualizar a los diferentes pokemons que se encuentran en su zona y así poder cazarlos. La aplicación también tiene en cuenta puntos de interés dentro del mundo real para que los usuarios puedan ir a esos lugares y conseguir dentro del juego objetos que les faciliten la caza y cría de los pokemons. El hecho de forzar al usuario a desplazarse por las calles para encontrar a estos seres permite un mayor nivel de inmersión ya que se consigue mezclar el mundo real con el universo digital del videojuego, lo que puede resultar atractivo para los jugadores.

Ilustración 2: Aplicación de Pokemon Go

Una vez conocido el concepto de sensibilidad al contexto y su uso en los juegos basados en la localización, lo siguiente es mostrar una serie de investigación que tratan de profundizar en el uso y desarrollo de las aplicaciones de realidad aumentada que empleen sensibilidad al contexto:

- **Un modelo para integrar la sensibilidad al contexto en aplicaciones de realidad aumentada para dispositivos móviles** [13]**:** propone un modelo que sirva de guía para incorporar la sensibilidad al contexto en aplicaciones de realidad aumentada para dispositivos móviles.
- **Adaptive object placement for augmented reality use in driver assistance systems** [14]**:** presenta un enfoque para la colocación de objetos adaptables que puedan ser usados en aplicaciones de realidad aumentada con el objetivo de mejorar los sistemas de asistencias a los conductores.
- **Urban games: application of augmented reality** [15]**:** detalla el concepto de "Urban Games" como la combinación de juegos, dispositivos móviles y entornos urbanos, explicando cómo los usuarios interactúan con estos sistemas y como este tipo de aplicaciones suponen una nueva forma de entretenimiento.
- **On the Development of Context-Aware Augmented Reality Applications** [16]**:** introduce conceptos referentes a la RA cómo la Realidad Aumentada Pervasiva (RAP). También se muestran los principales desafíos a la hora de desarrollar una aplicación de realidad aumentada sensible al contexto y proponen una solución para un entorno de desarrollo basado en modelos mostrando la eficacia de dicha solución con dos ejemplos en donde es aplicada en aplicaciones AR móviles sensibles al contexto.
- **Retargetable AR: Context-aware Augmented Reality in Indoor Scenes based on 3D Scene Graph** [17]**:** presenta la realidad aumentada retargeteable, un novedoso entorno de RA cuyas experiencias que produce tienen en cuenta el contexto de la escena en el que se establece, pudiendo producir una experiencia de RA con una interacción natural entre el mundo real y el virtual en distintos entornos reales.

Es indudable que la realidad aumentada proporciona un sinfín de posibilidades a la hora de elaborar una aplicación, sin contar los múltiples ámbitos que dicha aplicación puede abarcar y del enfoque que se le puede dar. Muchas empresas trabajan con el fin de conseguir que la RA sea cada vez más inmersiva, consiguiendo integrar de forma cada vez más convincente los elementos digitales dentro del mundo real.

Este proyecto tiene como objetivo el desarrollo de una aplicación de realidad aumentada en dispositivo móvil que emplee como contexto las coordenadas de geolocalización del GPS para poder integrar en nuestro mundo distintos elementos con los que el usuario pueda interactuar. El proyecto también hace hincapié en como el usuario puede interactuar con estos elementos o determinadas acciones que impliquen la detección de superficies planas como paredes o suelos, añadiéndose esta información al contexto sobre el que trabaja la aplicación. Para esta aplicación se ha escogido una perspectiva educativa que permita a los usuarios introducirse a la mitología ibérica mediante actividades de entretenimiento.

Capítulo 3. ANÁLISIS

El capítulo de Análisis recoge la fase de desarrollo del software, es donde se define de forma detallada cómo será el sistema que se desea crear y cuáles serán sus requerimientos. Esto implica definir las especificaciones de la aplicación, detallar los requisitos de usuario y elaborar un diagrama de clases.

Con toda esta información conocida de antemano, la dificultad de construir correctamente el sistema del proyecto se verá reducido de forma considerable.

3.1. Especificaciones de la aplicación

Iberian Odyssey es una aplicación móvil que genera un entorno lúdico que permita a los usuarios de todas las edades conocer algunos entes ficticios de la mitología ibérica a través de la realización de distintas actividades que utilizan la ubicación en coordenadas GPS de los objetos digitales de la aplicación.

La aplicación presenta un componente narrativo en el cual el usuario se ve sumergido en una aventura en la que, guiado por la bestia “Anjana”, deberá superar una serie de misiones en las que se encontrará con distintos seres pertenecientes a la cultura mitológica de la Península Ibérica con el objetivo final de derrotar a la bestia mitológica de nombre “Tarasca”.

Al contrario que muchos juegos actuales de realidad aumentada como Pokemon GO o Harry Potter: Wizards Unite [18] que apuestan por un modelo de juego en el que el jugador observa a su avatar moverse por un mapa en función de la localización real del jugador para encontrar eventos que inicien la experiencia de realidad aumentada, Iberian Odyssey apuesta por una experiencia más inmersiva en donde el jugador se encuentra constantemente en dicha experiencia RA en la cual suceden varios eventos en función de las acciones del jugador.

La experiencia que presenta la aplicación está planteada para que participe un solo usuario. Éste deberá realizar, como se ha dicho previamente, una serie de pruebas donde deberá de usar un conjunto de habilidades que le serán proporcionadas al comienzo de la prueba. El objetivo de dichas pruebas es que el usuario conozca y se instruya sobre los seres mitológicos que aparecen a lo largo de esta serie de pruebas.

Las pruebas que el usuario tendrá que realizar en orden son las siguientes y consisten en:

- **Primera prueba**: encontrar a cuatro seres llamados “Trentis” que se desplazan en una zona determinada y devolverlos con su hermano mayor.
- **Segunda prueba**: seguir las indicaciones del último “Trenti” recogido para encontrar al ser mitológico llamado “Musgosu”.
- **Tercera prueba**: obtener el tesoro que guarda el “Cuélebre”. Mientras dicha bestia esté cerca del tesoro, éste no se podrá recuperar, por lo que será necesario atraerlo a algún lugar lejos del cofre.

- **Cuarta y última prueba**: emplear la reliquia que había en el cofre para derrotar a la "Tarasca". Si se usa esa acción cerca de ella, está será eliminada y el usuario tendrá que invocar a "Anjana" para que la aplicación finalice.

Durante las distintas pruebas, el usuario se encontrará con una serie de objetos digitales que representan distintos seres de la mitología Ibérica, cuya implementación puede ser vista en el capítulo 4.3.7. Las bestias que el usuario encontrará son:

- **Anjana:** es una muchacha de cabellos dorados que servirá de guía al usuario cada vez que éste la invoque con la acción "Ayuda", lo que hará que le dé un consejo útil sobre lo que el usuario debe hacer a continuación.
- **Trenti:** es un pequeño duende de los bosques. Durante la aplicación el usuario encontrará a varios perdidos a los que tendrá que reunir para continuar con su aventura.
- **Musgosu:** es un anciano que vive en los bosques y que comparte su sabiduría con aquellos que respetan la naturaleza. Se encargará de desbloquearle al usuario la acción "Dejar Cebo".
- **Cuélebre:** se trata de una gigantesca serpiente alada que en la aplicación protege un cofre que el usuario debe recoger. Para ello, el usuario deberá emplear la acción "Dejar Cebo" para atraer al "Cuélebre" lejos del cofre y así poder desbloquear la acción "Cruz de Santa Marta".
- **Tarasca:** es el villano final de la aventura. Se trata de un gigantesco león de seis patas con cuerpo de tortuga y cola de dragón. El usuario deberá emplear en esta bestia la acción "Cruz de Santa Marta" para poder derrotarla y así terminar la aventura que ofrece esta experiencia RA.

Para resolver cada una de las pruebas el usuario podrá interactuar con el mundo digital que presenta la aplicación de distintas formas:

- **Ayuda:** permite al usuario invocar en un lugar del espacio que le rodea un ser mitológico digital llamado "Anjana" que le da una pista sobre el siguiente paso que debe realizar a continuación.
- **Interactuar:** esta acción le permite al usuario interactuar con los distintos objetos digitales que encontrará durante la experiencia para seguir adelante con cada una de las pruebas.
- **Analizar:** con esta acción el usuario podrá recopilar información de los distintos seres mitológicos que aparezcan durante las pruebas. Dicha información podrá ser vista en el menú de bestiario en donde se guardará toda la información obtenida.
- **Brújula:** al activar esta acción, el usuario podrá visualizar una flecha que le indica la dirección en la que debe dirigirse.
- **Poner cebo:** el usuario desbloquea esta habilidad al finalizar la segunda prueba. Esta acción le permite colocar un objeto digital en algún lugar del espacio que le rodea para atraer al "Cuélebre" a la posición en la que se ha dejado el objeto.

- **Cruz de Santa Marta:** es la habilidad que se desbloquea al finalizar la tercera prueba de la aplicación. Permite al usuario emplear el objeto llamado “Cruz de Santa Marta” para eliminar a la “Tarasca” del espacio digital de la aplicación.

La aplicación termina cuando la “Tarasca” es eliminada del mundo digital, tras esto se le indica al jugador que debe invocar una vez más a la “Anjana” con la acción de “Ayuda” para que ésta le dé un último mensaje con el que concluir la aplicación.

3.2. Requisitos de usuario

Los requisitos de usuario describen de forma detallada las funcionalidades que se esperan que cumpla un sistema. Estas funcionalidades se clasifican en requisitos funcionales y requisitos no funcionales:

3.2.1. Requisitos funcionales

Tabla 1: Requisitos funcionales

Número	Definición de requisitos funcionales
1	Permitir al usuario comenzar una nueva partida con la que reiniciar todas las misiones y empezar desde el principio.
2	Permitir al usuario continuar con la última prueba que estaba haciendo antes de cerrar el juego la última vez.
3	Permitir al usuario poder ver que personas han trabajado en el desarrollo del juego apretando el botón de “Créditos”.
4	Permitir al usuario cerrar la aplicación desde el menú principal.
5	Permitir al usuario poder moverse por el entorno para poder buscar a los distintos seres mitológicos que hay por la zona.
6	Permitir al usuario apretar el botón “Menú” para abrir el menú desplegable desde el cual acceder al menú de acciones, el menú “Bestiario”, el menú de “Ayuda” y el botón de salir de la aplicación.
7	Permitir al usuario poder salir en cualquier momento de la partida clicando el botón “Salir” que le llevará al menú principal.
8	Permitir al usuario emplear el menú de “Ayuda” para colocar en un espacio determinado del entorno una señal en la que aparecerá la criatura mitológica “Anjana” que le dará indicaciones sobre qué debe hacer a continuación mediante mensajes de texto.
9	Permitir al usuario abrir el menú “Bestiario” para leer toda la información disponible de los seres mitológicos sobre los que el usuario ha empleado la acción “Analizar”.
10	Permitir al usuario usar la acción “Interactuar” del menú de acciones para poder comunicarse con los seres mitológicos del entorno o interactuar con objetos cuando se encuentra a una determinada distancia de ellos.
11	Permitir al usuario usar la acción “Analizar” del menú de acciones para obtener información de los seres mitológicos que hay alrededor al apuntarlos con el visor de análisis y estar a una determinada distancia.
12	Permitir al usuario usar la acción “Brujula” del menú de acciones para indicar mediante una flecha que aparecerá en pantalla la dirección en la que se encuentra el siguiente objetivo de la prueba.
13	Permitir al usuario usar la acción “Poner Cebo” del menú de acciones para dejar

	un cebo en un espacio determinado del entorno que atraiga al ser mitológico de nombre “Cuélebre” siempre y cuando no haya otro cebo colocado en la escena.
14	Permitir al usuario usar la acción “Cruz de Santa Marta” del menú de acciones para invocar el objeto de la cruz de Santa Marta con el que se puede vencer al ser mitológico de nombre “Tarasca” si la cruz apunta hacia este a partir de una determinada distancia.

3.2.2. Requisitos no funcionales

Tabla 2: Requisitos no funcionales

Número	Definición de requisitos no funcionales
1	Debido a la experiencia que se tiene a la hora de usar el programa Unity, el lenguaje de programación que se empleará será C#.
2	La aplicación se compone de un menú principal, una pantalla de tutorial en donde se explican las bases para emplear la aplicación, la pantalla principal en donde se lleva a cabo la experiencia de realidad aumentada y una pantalla de despedida para indicar que se han completado todas las pruebas.
3	Los menús de la interfaz de usuario serán intuitivos, sencillos y respetará el espacio de visión del usuario para no romper la inmersión de la experiencia de realidad aumentada.
4	La aplicación presentará un argumento que dote de contexto al porqué el usuario debe completar cada una de las pruebas.
5	La aplicación guardará de forma automática los datos de en qué prueba se encuentra el usuario actualmente, las acciones que tiene desbloqueadas y los seres mitológicos que ha analizado, cada vez que termine una prueba.
6	La aplicación no permitirá al usuario continuar una partida si no hay datos guardados de ninguna partida previa.
7	Al iniciar una nueva partida, la aplicación mostrará a modo de tutorial una escena en donde se describen las distintas acciones que puede realizar el usuario y como emplearlas.
8	El usuario irá desbloqueando nuevas acciones conforme vaya avanzando en las pruebas con el fin de evitar que experiencia lúdica sea monótona.
9	Una vez analizado un ser mitológico, la aplicación mostrará un mensaje indicando si ya se tenía información de ese ser en el menú “Bestiario” o si se ha añadido información nueva al mismo.
10	Cuando el usuario emplea la acción “Poner Cebo” o el menú “Ayuda”, la aplicación mostrará un indicador que avise al usuario donde será colocado el objeto que va a invocar.
11	Cada vez que el usuario avance a la siguiente prueba, se mostrará un icono al lado del botón de “Ayuda” para indicar que hay nueva información que le puede ser útil al usuario.
12	Al finalizar todas las pruebas, la aplicación mostrará una pantalla en donde se le agradece al usuario por completar todas las pruebas y desde la que el usuario puede regresar al menú principal.

3.3 Diagrama de casos de uso

Considerando los requisitos de usuario vistos previamente (tablas 1 y 2) y las especificaciones del sistema, se ha desarrollado un diagrama de casos de uso. Un caso

de uso representa una serie de acciones que producen un resultado observable de valor para los usuarios que intervienen en un sistema.

El siguiente diagrama representará de forma gráfica, empleando los distintos casos de uso que se pueden dar en el proyecto, el funcionamiento de la aplicación.

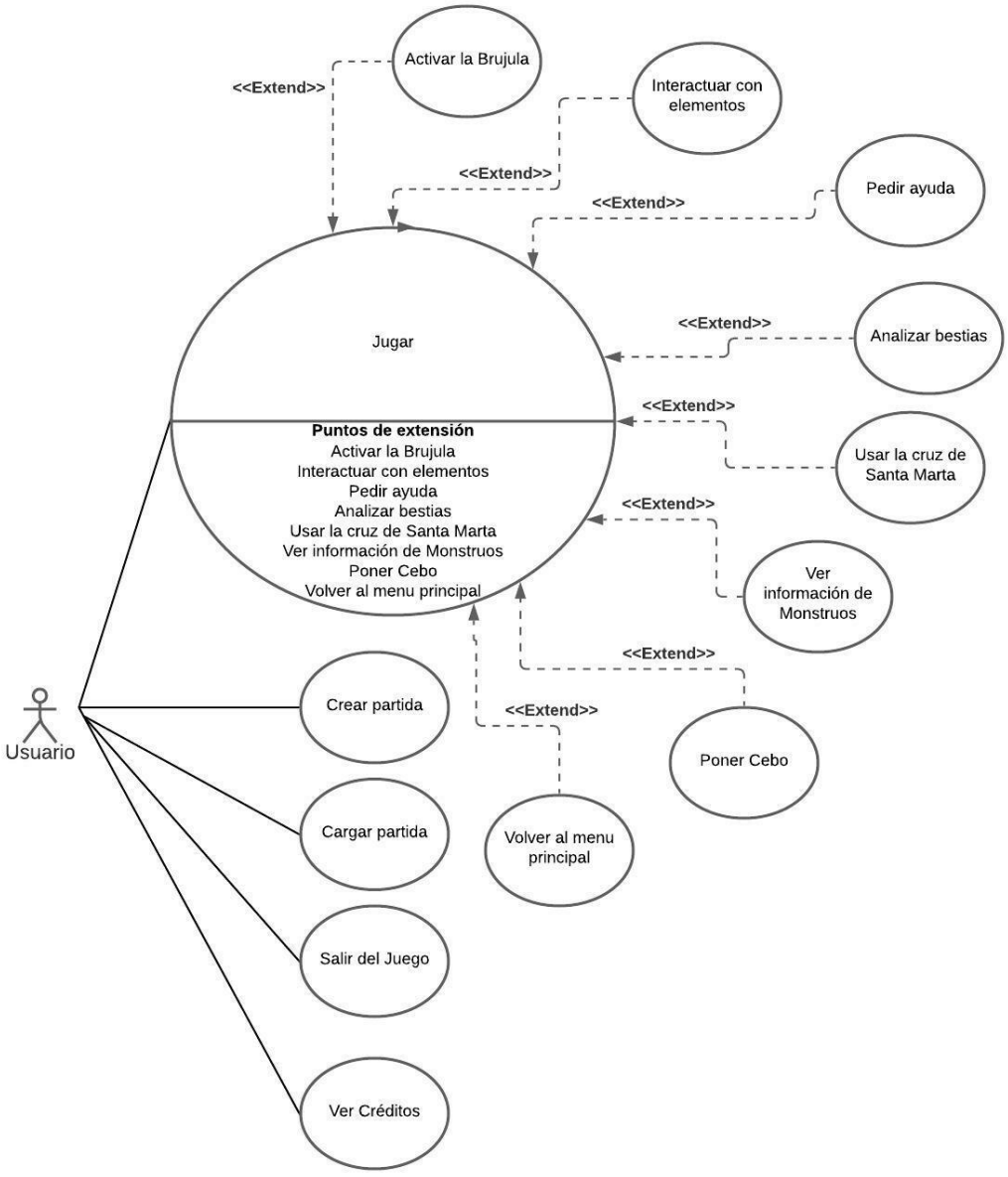


Ilustración 3: Diagrama de Casos de Uso

A continuación, se explican los distintos casos de uso:

- **Jugar**: este caso de uso comienza después de que el usuario haya realizado la operación “Crear partida” o “Cargar partida”. Con la escena iniciada el usuario podrá realizar los distintos casos de uso que extienden de este. La experiencia terminará cuando el usuario termine todas las pruebas que se le presenten.

- **Crear partida**: el siguiente caso de uso permite al usuario crear una experiencia nueva iniciado un breve tutorial en donde se le expliquen las distintas acciones que puede realizar.
- **Cargar partida**: el usuario podrá emplear este caso de uso en caso de que ya haya una experiencia que se haya iniciado previamente mediante la operación "Crear partida". Este caso de uso permite al usuario seguir con una experiencia ya iniciada desde el punto en el que se dejó la última vez.
- **Salir del Juego**: esta acción hará que se cierre la aplicación desde el menú de inicio.
- **Ver Créditos**: esta acción permitirá al usuario poder ver los nombres de las personas involucradas en el desarrollo de la aplicación.
- **Activar la Brújula**: esta acción permite al usuario activar la brújula que ofrecerá información sobre donde se encuentra el objetivo del usuario.
- **Interactuar con elementos**: esta acción permite al usuario interactuar con los distintos objetos digitales para obtener información de ellos.
- **Pedir ayuda**: esta acción permite al usuario invocar a la bestia "Anjana" para que ofrezca información sobre qué hacer a continuación.
- **Analizar bestia**: este caso de uso es empleado para desbloquear información de las bestias que el usuario encuentra en el bestiario.
- **Usar la cruz de Santa Marta**: este caso de uso es empleado para invocar el objeto Cruz de Santa Marta que se usa para derrotar a la bestia "Tarasca".
- **Ver información de monstruos**: esta acción solo puede usarse si se ha empleado la acción "Analizar bestia" sobre la bestia de la cual se quiere ver información. Este caso de uso permite al usuario ver información sobre una bestia ya analizada por medio de un texto y una imagen.
- **Poner cebo**: esta acción es empleada por el usuario para colocar un objeto digital que sirva para atraer a la bestia de nombre "Cuélebre".
- **Volver al menú principal**: esta acción permite al usuario abandonar la experiencia de realidad aumentada para regresar al menú inicial.

3.4. Diagrama de clases

Teniendo en cuenta la información descrita en los apartados de las especificaciones del sistema y los requisitos de usuario, se ha procedido a elaborar un diagrama de clases conceptual de alto nivel que muestre un modelo conceptual de la aplicación, las distintas entidades que hay y las relaciones existentes entre ellas.

La elaboración de este diagrama que puede ser visto en la ilustración 4 se ha llevado a cabo con el fin de facilitar la implementación de la aplicación en la plataforma de Unity.

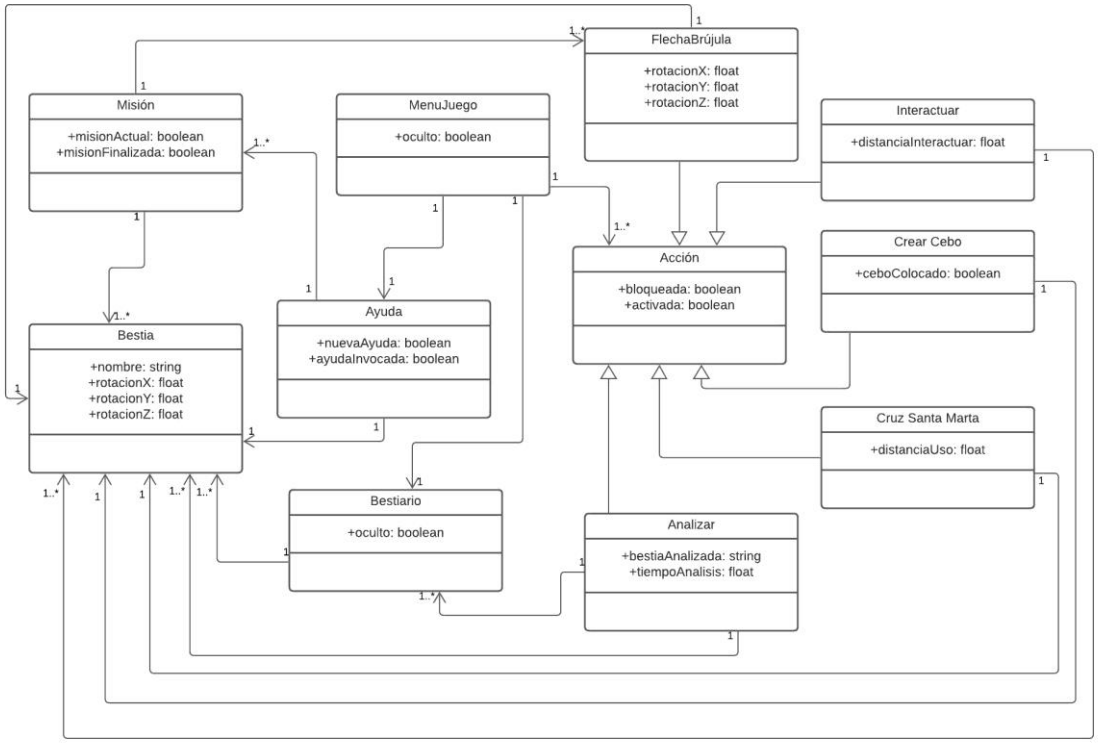


Ilustración 4: Diagrama de clases conceptual

Con este diagrama de clases se pueden apreciar los conceptos clave con los que trabajará la aplicación y las relaciones entre ellos. A continuación, se profundiza en el propósito de cada una de las clases a través de la función que cumplen sus atributos:

- **MenuJuego**: representa el menú con el que el jugador podrá realizar las distintas funcionalidades de la aplicación. El campo booleano oculto refleja si el menú se encuentra en el estado oculto o no, de esa forma cuando el menú se encuentra oculto no perturba la experiencia de juego ocupando gran parte de la pantalla. Si por el contrario no se encuentra oculto, el menú mostrará las distintas funcionalidades que el usuario puede emplear. El menú de la aplicación contará con acceso a un bestiario, al menú de ayuda y a un listado de acciones que el usuario podrá realizar.
- **Acción**: esta clase hace referencia a las distintas formas que tiene el usuario de interactuar con los elementos digitales de la experiencia de realidad aumentada. Pese a que cada una de estas funcionalidades difiere considerablemente de las demás, todas heredan de esta clase dos atributos:
    - bloqueada: indica si el usuario posee esta acción, en caso de no poseerla el usuario no podrá emplearla. Algunas acciones están desbloqueadas por defecto desde el principio de la experiencia, mientras que otras se irán desbloqueando conforme el usuario vaya superando las pruebas.
    - activada: define si una acción está siendo usada. En caso de ser así, las demás acciones no podrán ser usadas mientras se esté realizando la acción actual.

- **Interactuar**: se trata de una clase que hereda de la clase Acción. Interactuar se encarga de permitir al usuario comunicarse con los distintos objetos virtuales de la aplicación cuando el usuario se encuentra a una determinada distancia de ellos. La variable distanciaInteractuar es la que representa esa distancia mínima que se ha de cumplir para que el usuario pueda emplear esta acción. La acción interactuar podrá ser realizada sobre las distintas bestias que tiene asociada.
- **Analizar**: es una clase que hereda de la clase Acción. La clase se encarga de recopilar información sobre las bestias que el usuario pueda encontrar y almacenar dicha información en el bestiario. Esta acción tendrá asociada las distintas bestias sobre las que puede usarse. La clase Analizar emplea dos atributos:
    - bestiaAnalizada: representa el nombre de la bestia que se acaba de analizar. Si en el inventario no existe una entrada de esa bestia, se crea una nueva entrada con información sobre la bestia analizada. En caso contrario se avisa al usuario de que ya existe una entrada con información de la bestia que se acaba de analizar.
    - tiempoAnalisis: para que el análisis no sea inmediato, se emplea un tiempo de espera para la realización de la acción. Esto para darle un efecto a la acción de que la aplicación está escaneando a la bestia y necesita recopilar primero la información antes de actualizar el bestiario.
- **FlechaBrujula**: se trata de una clase que hereda de la clase Acción. Cuando es activada, se muestra por pantalla unas flechas que apuntan al objetivo actual de la misión guiando al usuario en la dirección a la que debe dirigirse, dicho objetivo será una única bestia que tendrá asociada. Los atributos de rotacionX, rotacionY y rotacionZ sirven para orientar esas flechas en la dirección en la que se encuentra el objetivo de la misión, y sus valores variarán en función de la posición del usuario con respecto a dicho objetivo.
- **CrearCebo**: se trata de una clase que hereda de la clase Acción. Esta acción permite al usuario invocar en un lugar de su alrededor un cebo que atraiga la bestia de nombre “Cuélebre”, por lo tanto, la acción de crear un cebo estará asociada únicamente a la bestia de ese nombre. Esta acción solo podrá realizarse mientras esté activa o no haya ningún cebo colocado. Por lo que el atributo ceboColocado servirá para detectar si el usuario ya ha colocado un cebo y no permitirá colocar otro cebo hasta que el ya existente desaparezca.
- **CruzSantaMarta**: se trata de una clase que hereda de la clase Acción. Esta acción permite derrotar a la bestia de nombre “Tarasca” si se emplea la acción a una determinada distancia dada por el atributo distanciaUso. Esta acción no tiene efecto sobre ninguna otra bestia que no sea la “Tarasca”, de forma que la acción de la cruz de Santa Marta estará asociada a la bestia con dicho nombre.
- **Mision**: representa la prueba que tiene que realizar el usuario. La misión tiene asociada una flecha de la brújula que apunta hacia el objetivo de la prueba y una bestia que será el objetivo de esa misma prueba. Cada misión trabaja con dos atributos:

    - misionActual: indica si la misión es la que el usuario está realizando en este momento.
    - misionFinalizada: hace referencia a si la misión ya ha sido completada.

  Las distintas misiones son en su mayoría encontrar una bestia determinada e interactuar con ella con alguna de las acciones que posee el jugador.
- **Ayuda**: se trata de una funcionalidad que permite al usuario invocar a una bestia de nombre "Anjana" para que le dé información sobre lo que el usuario debe hacer a continuación en función de la prueba en la que se encuentre. La acción Ayuda tendrá por tanto asociada a la bestia de nombre "Anjana" y una lista de las distintas misiones que el usuario deberá de superar. La funcionalidad de ayuda contará con los siguientes atributos:
    - nuevaAyuda: indica si hay un nuevo texto de ayuda que el usuario no ha consultado aún. Esta indicación se traducirá en una indicación visual que haga que el usuario sea consciente de que aún no ha leído el texto de ayuda.
    - ayudaInvocada: define si la "Anjana" ha sido invocada, de forma que, si ya hay una "Anjana" invocada, el usuario no podrá invocar otra hasta que la actual haya desaparecido.
- **Bestiario**: es una funcionalidad a la que se accede desde el menú de la aplicación que permite a los usuarios ver la información de aquellos seres mitológicos sobre los que ha usado la acción de analizar. Para tener esta información de cada ser, el bestiario tiene asociado una lista de las distintas bestias que el usuario ha analizado. El atributo oculto refleja si el bestiario se encuentra oculto o no para que el usuario pueda ocultarlo a placer y no le estorbe durante la experiencia RA.
- **Bestia**: son los seres mitológicos con los que el usuario se encontrará durante la experiencia de realidad aumentada. Aunque cada bestia sea distinta de la anterior, el funcionamiento de cada una es bastante similar, por lo que todas las bestias comparten los siguientes atributos:
    - nombre: representa el nombre de la bestia.
    - rotacionX, rotacionY, rotacionZ: representan cada una de las rotaciones en los distintos ejes de coordenadas de la bestia. Estos valores son importantes porque cuando el usuario interactúa con las bestias, la orientación de estas cambia para que miren al usuario. La orientación de la bestia dependerá de estos parámetros, sobre todo de la rotación en el eje Y.

Con este diagrama de clases se consigue una primera aproximación detallada de cómo será la implementación de la lógica de la aplicación de forma resumida y fácil de entender. En el capítulo 4.3 se muestra el diagrama de clases de implementación en el que se ve cómo ha sido la implementación final de la lógica de la aplicación, observando algunas variaciones en la estructura mostrada en el diagrama de clases de la ilustración 4.

Capítulo 4. DISEÑO E IMPLEMENTACIÓN

El desarrollo de esta etapa de Diseño e Implementación de la aplicación se basa en lo expuesto en la fase de análisis, pues en ella se han detallado los requerimientos y especificaciones que el sistema necesita para que su funcionamiento sea tal y como se ha descrito.

En este capítulo, se presentarán las distintas tecnologías y programas empleados para la realización del proyecto, el diseño y funcionamiento de la interfaz y la implementación de los aspectos más relevantes de la aplicación móvil.

4.1. Tecnologías y programas empleados

Para la realización de este proyecto ha sido necesario el uso de distintos programas y tecnologías, los cuales se pueden diferenciar en el empleo de un motor software con los assets que éste pueda ofrecer para la crear la aplicación, los distintos programas para modelar y animar los distintos objetos que se vayan a necesitar y la plataforma hardware en donde se empleará la aplicación.

4.1.1. Motor

Dado que el proyecto se trata de una aplicación de carácter lúdico, un motor enfocado en el desarrollo de videojuegos es una buena opción para el desarrollo de la aplicación. Los motores de videojuegos se pueden definir como un conjunto de librerías de programación que ayudan a los desarrolladores durante la creación del videojuego.

Estas librerías facilitan enormemente el trabajo de los desarrolladores a la hora de escribir código ya que muchos de los cálculos que deberían implementarse mediante código son realizados de forma automática por las librerías y son empleados cuando el programador los necesita. Esta ayuda implica una gran reducción de trabajo para el desarrollador, quien puede centrarse en implementar los aspectos de la aplicación que las librerías no pueden realizar.

Dos de las funcionalidades más importantes que puede ofrecer un motor y que más se han de tener en cuenta a la hora de escoger uno son:

- El motor de físicas: se encarga de realizar los cálculos de las magnitudes físicas necesarias para el desarrollo de un videojuego, esto permite que los comportamientos de la gravedad, el peso o el volumen resulten naturales a ojos del usuario.
- La capacidad gráfica: permite al motor realizar los dibujos de los gráficos en la pantalla (tanto en 3D como en 2D) y calcular varios aspectos de una aplicación relacionados con los gráficos, como la iluminación y el texturizado.

Para poder realizar el desarrollo de Iberian Odyssey, el motor escogido ha sido Unity [19], en este caso la versión 2021.1.6. Una de las grandes ventajas que tiene este motor es que su curva de aprendizaje es muy accesible para los iniciados en el

desarrollo de aplicaciones, a esto se le añade la comunidad que tiene en la que la gran mayoría de dudas pueden ser fácilmente resueltas con una búsqueda rápida en internet.

Uno de los motivos principales a la hora de escoger Unity es que permite la creación de aplicaciones en múltiples plataformas distintas como dispositivos móviles (Android, iOS), PC (Windows, Mac, Linux) e incluso consolas (PS4, Xbox One) y dispositivos de realidad virtual (PlayStation VR, Oculus Rift).

4.1.1.1. Assets empleados

Además de la capacidad de Unity para desarrollar aplicaciones multiplataforma, hay un motivo con mayor peso que lo convierte en la opción idónea para el desarrollo de este proyecto. Esta es la amplia variedad de assets creados por los miembros de la comunidad y que se pueden instalar en Unity por medio de la Asset Store. Estos assets proporcionan nuevas librerías y componentes que pueden ayudar de forma considerable a reducir la carga de trabajo que el desarrollador sufre.

Los assets más relevantes que han sido empleados durante el desarrollo de este proyecto han sido los siguientes:

- Fungus [20]: este asset ofrece a los usuarios de forma gratuita una herramienta de código abierto que permita crear juegos interactivos de carácter narrativo. La curva de aprendizaje de esta herramienta es bastante accesible y ofrece muchas posibilidades, tanto para desarrolladores sin experiencia a la hora de escribir código como a los más experimentados en este ámbito. Este asset resultó de vital importancia para el desarrollo del proyecto a la hora de crear cuadros de dialogo asociados a cada uno de los personajes que aparezcan en la aplicación, sirviendo estos cuadros de texto como método para transmitir la narrativa que presenta la aplicación y las directrices que el usuario ha de seguir para superar las distintas pruebas.
- AR Foundation [21]: se trata de un paquete que ofrece de forma gratuita un marco de trabajo que unifica las mejores características de ARCore [22] y ARKit [23] para crear experiencias robustas de realidad aumentada que puedan implementarse en dispositivos móviles. AR Foundation fue una pieza crucial para el desarrollo de Iberian Odyssey ya que aporta todas las bases para crear una aplicación de realidad aumentada funcional.
- AR+GPS Location [24]: es el único de los assets mencionados en esta lista que no es gratuito. Se trata de un paquete que permite al usuario mediante el uso de coordenadas GPS posicionar objetos 3D en localizaciones geográficas del mundo real para posteriormente ser visualizados mediante aplicaciones de realidad aumentada. Este asset es compatible con AR Foundation de Unity, lo que hace que trabajar con ambos assets sea viable. El uso de este paquete ha sido necesario para geoposicionar los distintos elementos de la aplicación en los lugares exactos en donde deben el usuario deberá interactuar con ellos.

4.1.2. Programas de modelado y animación 3D

El modelado en 3D es un proceso en el cual se crean objetos virtuales dotados de tridimensionalidad por medio de un software especializado.

Por otro lado, animar en 3D es dar movimiento a objetos tridimensionales empleando software especializado.

La mayoría de los modelos 3D empleados para este proyecto han sido extraídos de Models Resource [25], ha sido necesario emplear el modelado 3D para modificar algunos modelos ya sea tanto su forma como el texturizado de los mismos y la animación 3D para crear animaciones para estos modelos que aporten mayor credibilidad y sensación de inmersión a la experiencia de realidad aumentada.

Los dos programas empleados para realizar el modelado y las animaciones 3D han sido:

- Blender [26]: es un programa dedicado al modelado 3D, iluminación, renderizado, texturizado y animación. Se trata de un programa gratuito, pero bastante completo. Pese a que su interfaz no resulta tan intuitiva, esta puede ser personalizada a gusto del usuario. Para este proyecto, el uso de Blender ha sido vital para el modelado de algunos objetos y el texturizado de varios modelos. También se ha empleado Blender para realizar algunas animaciones de modelos que no fuese humanoides.
- Mixamo [27]: se trata de una página web en la que los usuarios pueden crearse una cuenta de forma totalmente gratuita. Mixamo sirve exclusivamente para descargar animaciones asociadas a modelos 3D humanoides, de forma que los usuarios pueden importar sus propios modelos 3D y añadirles la animación que más les convenga para luego exportarlas. Mixamo también ofrece unos pocos modelos 3D que pueden ser usados. El uso de esta página web ha permitido ahorrar mucho tiempo y trabajo a la hora de animar varios de los modelos 3D empleados.

4.1.3. Plataforma

La plataforma hace referencia al dispositivo electrónico en el cual será reproducida la aplicación. Debido a que Iberian Odyssey se trata de una aplicación “urban game”, tal y como se ha visto antes en el capítulo 2.1.2, este tipo de aplicaciones están diseñadas para ser empleadas en la calle, por lo que una consola de sobremesa o un ordenador no serían las opciones adecuadas para este proyecto.

Es por eso por lo que la plataforma destino del proyecto son los dispositivos móviles, los cuales permiten al usuario poder desplazarse por el entorno con total libertad y poseen una gran cantidad de sensores que aportan contexto para el desarrollo de aplicaciones de realidad aumentad sensibles al contexto de buena calidad.

Dentro de todas las opciones de móviles, se ha escogido solamente Android 7.0 o superior. Debido a la falta de tiempo, Iberian Odyssey no ha podido ser implementada para iOS.

4.2. Diseño e implementación de la interfaz de usuario

En este apartado se verá el diseño de los distintos menús que componen la interfaz de usuario de la aplicación y como ese diseño permite que se cumplan varios de los requisitos de funcionales vistos en el capítulo 3.2.1. Cada uno de los menús del proyecto se encuentran en una escena distinta, el proyecto de Unity en el cual se ha desarrollado la aplicación se compone de cuatro escenas distintas: “Inicio”, “NuevaPartida”, “Juego” y “Final”.

4.2.1. Menú de inicio

El menú de inicio abarca todos los menús que se encuentran en la escena “Inicio” del proyecto de Unity. Tal y como se ve en la ilustración 5, se trata de un menú muy simple que presenta el título completo de la aplicación “Iberian Odyssey, el retorno de la Tarasca”. Debajo del título se pueden ver cuatro botones alineados con los siguientes nombres: “Juego Nuevo”, “Continuar”, “Créditos” y “Salir”.

Ilustración 5: Interfaz gráfica del menú de inicio

El botón “Continuar” de la ilustración 5 aparece desactivado (representado por la opacidad que muestra frente a los demás botones de la interfaz) debido a que al iniciar la aplicación no hay datos guardados, es por ello por lo que se le impide al usuario continuar una partida dado que no existe tal partida que continuar. En caso de que hubiese una partida guardada el botón “Continuar” luciría como los demás, tal y como se puede apreciar en la ilustración 6.

Ilustración 6: Interfaz gráfica del menú de inicio con datos guardados

También cabe mencionar que cuando el usuario pulsa sobre el botón "Créditos" se abre un panel mostrando a las personas que han formado parte del desarrollo de la aplicación y las tareas de las que se han encargado.

Ilustración 7: Interfaz gráfica menú de inicio con panel de créditos desplegado

Habiendo visto todo lo que ofrece la interfaz del menú principal, se puede comprobar que satisface los requisitos funcionales 1, 2, 3 y 4.

4.2.2. Nueva partida

En este apartado se abarca la interfaz desarrollada para la escena "NuevaPartida" del proyecto de Unity. Se trata de una interfaz muy simple en la que aparece la bestia "Anjana" en el centro

sobre un fondo negro y un cuadro de texto en la parte inferior en donde se podrá leer el diálogo del personaje a modo de tutorial para aprender el funcionamiento de la aplicación.

Ilustración 8: Interfaz gráfica del menú de partida nueva

Conforme se va avanzando en el texto se añade a la interfaz una serie de imágenes para que el jugador conozca los iconos de las acciones que puede realizar y comience a familiarizarse con los mismos.

Ilustración 9: Interfaz gráfica menú de partida nueva con los iconos de acciones

4.2.3. Interfaz de la aplicación

Para la interfaz que acompañará al usuario durante toda la experiencia de realidad aumentada, se ha pensado en un diseño de interfaz desplegable que trate de ocupar el menor espacio en pantalla para que sea lo menos intrusivo posible. Para explicar en detalle la interfaz de usuario en la escena “Juego” de Unity, se dividirá la interfaz en capas que irán añadiendo más contenido conforme se vayan mostrando nuevas partes del menú.

Como primera capa se muestra al menú en su estado de menor despliegue. Lo único que se puede observar de dicho menú es un recuadro azul con un botón en el centro en donde se lee “Menú” en la esquina inferior izquierda de la pantalla. Esto permite que cuando el menú no esté desplegado, el área de visión del usuario sea lo más amplia posible.

Ilustración 10: Interfaz gráfica del menú de juego sin desplegar

La ilustración 10 satisface el requisito funcional 5.

La segunda capa muestra el primer despliegue del menú que ocupa el lateral izquierdo de la pantalla sin restar mucho campo de visión al usuario. Este menú muestra cuatro botones: “Acciones”, “Bestiario”, “Ayuda” y “Salir”. Esto satisface el requisito funcional de usuario 6 tal y como se muestra en la ilustración 11.

El botón “Acciones” desplegará el menú de las distintas acciones que puede emplear el usuario. Por otro lado, el botón “Bestiario” desplegará el menú con las distintas entradas del bestiario. Se puede apreciar como en la ilustración 12, la interfaz puede mostrar un icono de exclamación sobre el botón “Ayuda”, este icono aparecerá siempre que haya nueva información para el usuario en el menú de ayuda. Y finalmente, el botón “Salir” permitirá a los usuarios salir de la aplicación.

Ilustración 11: Interfaz gráfica del menú de juego desplegada

Ilustración 12: Interfaz gráfica menú de juego desplegada con nueva información de ayuda

Si se acciona el botón de “Acciones” se desplegará un menú horizontal que muestra las distintas acciones que el usuario puede emplear, representando cada acción con un icono. En función de cuantas acciones tenga desbloqueadas el usuario, habrá más o menos espacios vacíos en este menú desplegable.

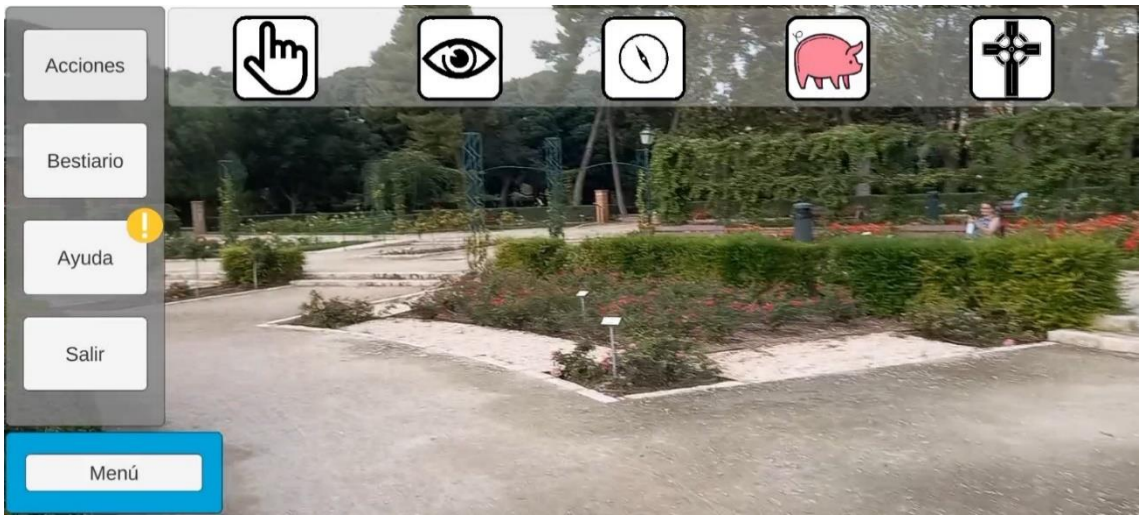


Ilustración 13: Interfaz gráfica menú de juego con menú de acciones desplegado

En caso de que se use el botón de “Bestiario” se mostrará el menú mostrado en la ilustración 14 con las distintas bestias que el usuario tiene registradas.

Ilustración 14: Interfaz gráfica menú de juego con menú del bestiario desplegado

El menú que se muestra al pulsar el botón ayuda se compone simplemente de un botón cuadrado con una imagen de la “Anjana”. La interfaz también muestra un símbolo si el usuario apunta la cámara al suelo que sirve como señal para indicar en donde será invocada la bestia “Anjana”, la cual ofrecerá ayuda al usuario mediante una caja de texto con la que le dará indicaciones. Mientras la “Anjana” esté invocada, el usuario no podrá usar el menú desplegable, el cual estará deshabilitado”. Tras haber sido leído el cuadro de texto, la “Anjana” desaparecerá dejando usar el menú al usuario.

Ilustración 15: Interfaz gráfica menú de juego con menú de Ayuda desplegado

Ilustración 16: Interfaz gráfica menú de juego con menú de Ayuda desplegado y Anjana invocada

Las dos ilustraciones anteriores muestran el cumplimiento del requisito de usuario 8.

Y finalmente, dentro de estos cuatro botones que hay en el primer despliegue del menú de juego, al apretar el botón “Salir” se muestra un sencillo panel que le indica al usuario si está seguro de querer salir de la experiencia de realidad aumentada, cumpliendo así el requisito de usuario 7

Ilustración 17: Interfaz gráfica menú de juego con menú de Salir desplegado

Una vez mostradas las interfaces del menú desplegable, es momento de mostrar las distintas interfaces de cada una de las acciones.

La interfaz de la acción “Interactuar” muestra un botón cuadrado en la esquina inferior derecha con el icono de la acción “Interactuar”, el cual se encuentra deshabilitado si el usuario no se encuentra cerca de un objeto con el que se pueda interactuar. La interfaz también muestra en el centro de la pantalla un puntero para que el usuario sepa que debe de apuntar con él a los elementos con los que se quiera interactuar.

Ilustración 18: Interfaz gráfica del menú de Interactuar

Ilustración 19: Interfaz gráfica del menú de Interactuar alejado

Cuando el usuario se encuentre a una determinada distancia de una bestia y pulse sobre el botón de “Interactuar” mientras el puntero apunta a la bestia, se abrirá en la parte inferior de la pantalla un cuadro de texto con el nombre de la bestia con la que se está interactuando y con el dialogo de la bestia. Para pasar al siguiente diálogo, el usuario simplemente debe pulsar la pantalla táctil.

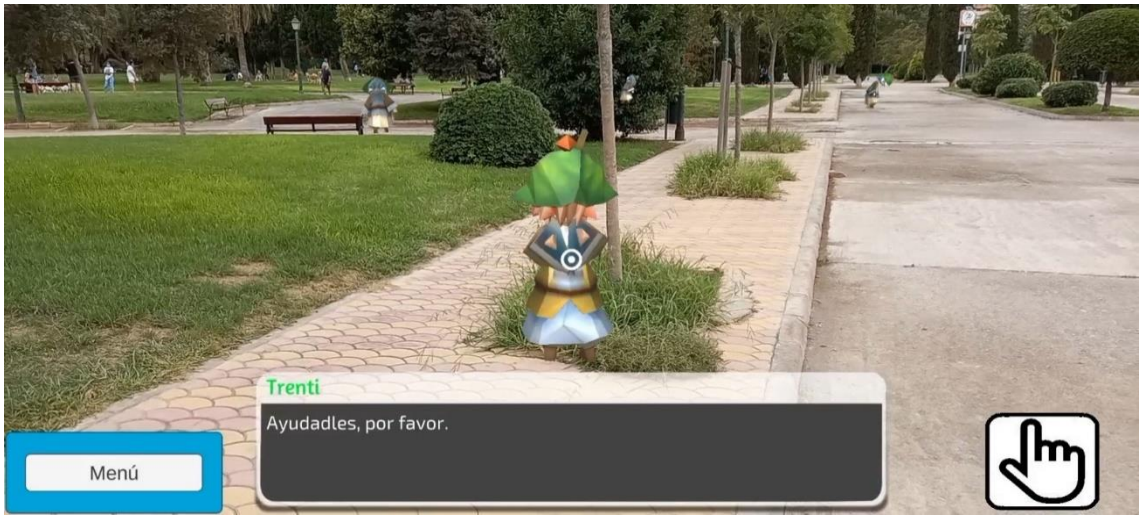


Ilustración 20: Interfaz gráfica del menú de Interactuar con interactuación con bestia

Las ilustraciones 18, 19 y 20 muestran cómo se satisface el requerimiento de usuario 10.

La acción “Analizar” muestra una interfaz con una retícula en el centro de la pantalla y una barra que se rellena si el usuario apunta con la retícula a una bestia y se encuentra a una determinada distancia de esta. Cuando se termina de rellenar la barra, está se vacía al instante y muestra un mensaje confirmando que hay una nueva entrada en el inventario o en caso de que dicha bestia ya haya sido analizada previamente, el mensaje indicara que ya existe una entrada de esa bestia en el inventario.

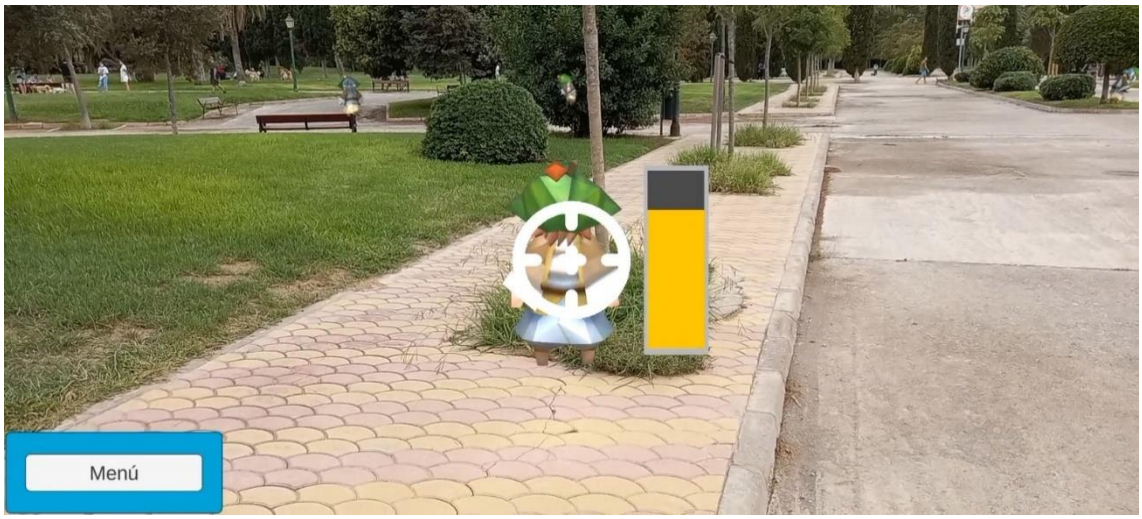


Ilustración 21: Interfaz gráfica del menú de Analizar

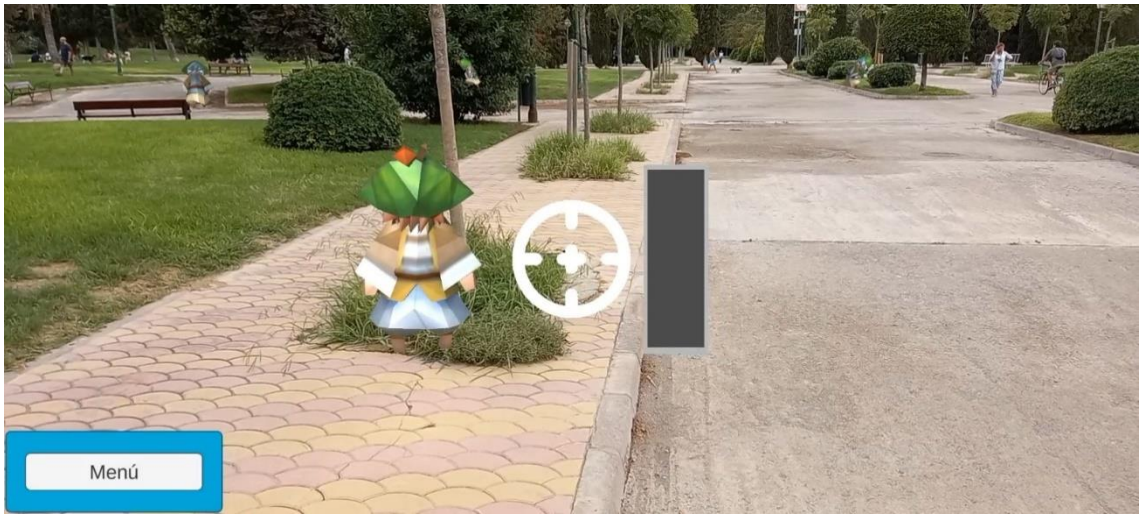


Ilustración 22: Interfaz gráfica del menú de Analizar sin contacto

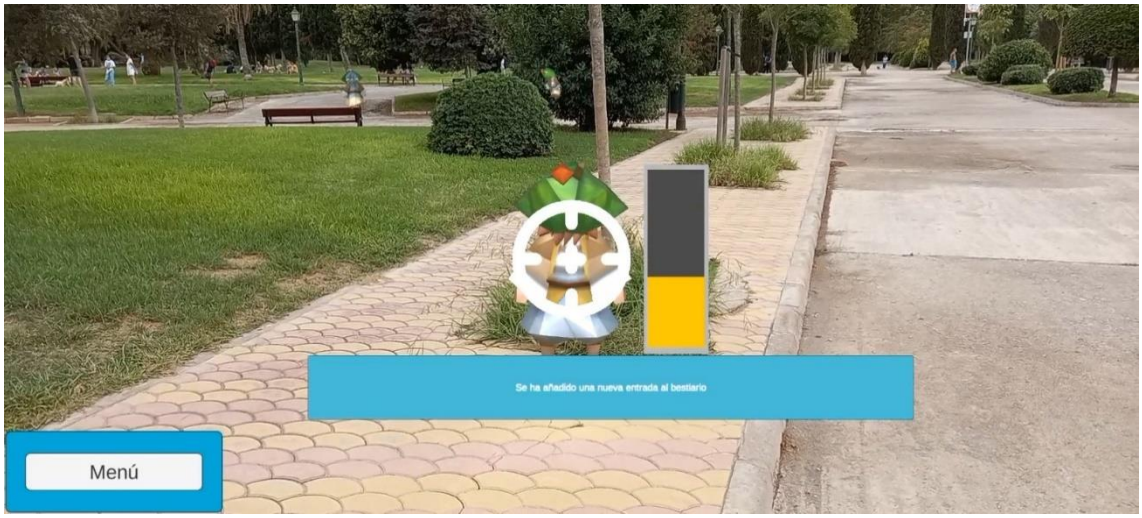


Ilustración 23: Interfaz gráfica del menú de Analizar con mensaje

Las ilustraciones 21 y 22 satisfacen el requisito funcional de usuario 11.

La interfaz de la acción “Brújula” simplemente muestra por pantalla una flecha que apunta a la dirección del siguiente objetivo del usuario. En caso de que el usuario se halle cerca del objetivo la flecha desaparecerá y saldrá un mensaje indicando que el objetivo se encuentra cerca.

Ilustración 24: Interfaz gráfica del menú de Brújula

Ilustración 25: Interfaz gráfica menú de Brújula mostrando mensaje de proximidad

Las ilustraciones 24 y 25 cumplen con el requisito funcional de usuario 12.

La acción de “Poner Cebo” muestra una interfaz idéntica a la mostrada en la ilustración 15, pero el icono del botón es el de la acción “Poner Cebo” y en vez de invocar a la “Anjana” invoca al objeto “Cebo” con el que se puede atraer a la bestia de nombre “Cuélebre”, la cual permanecerá sobre el cebo durante un minuto. Mientras haya un cebo colocado, el usuario no podrá colocar otro, por lo que el botón para dejar el cebo estará desactivado.

Ilustración 26: Interfaz gráfica del menú de Poner Cebo

Ilustración 27: Interfaz gráfica menú de Poner Cebo con el objeto Cebo invocado

Las imágenes 26 y 27 muestran en detalle como el requisito funcional de usuario 13 se cumple.

Y la última acción llamada “Cruz de Santa Marta” es una interfaz vacía en la que, si el usuario mantiene pulsada la pantalla con su dedo, el objeto “Cruz” aparece enfrente suya. Si se encuentra cerca de la bestia de nombre “Tarasca”, se activará un cuadro de texto, tras terminar de ser leído por parte del usuario, la “Tarasca” desaparecerá y aparecerá un mensaje indicando que hay que invocar a “Anjana” para terminar con la experiencia AR.

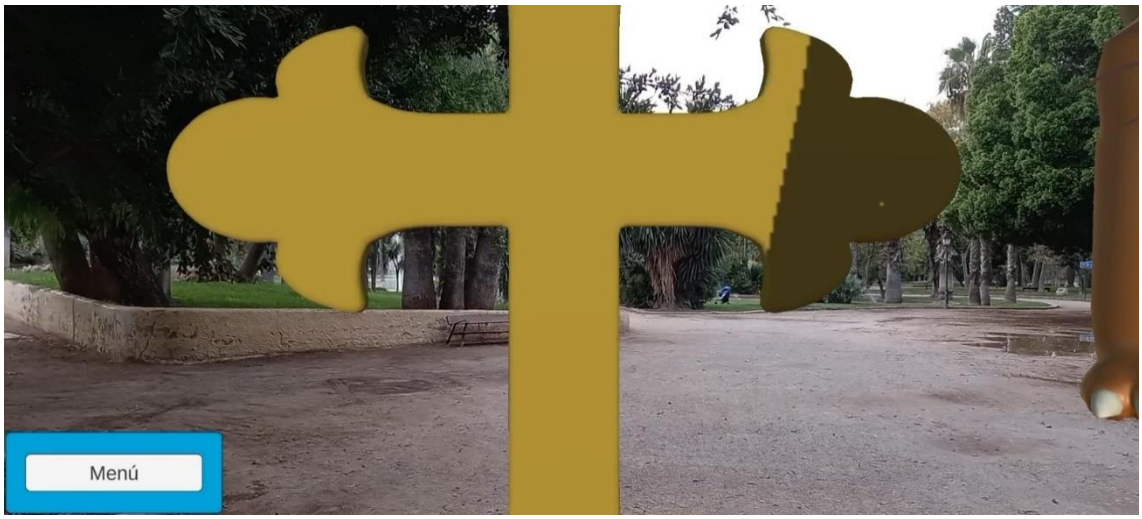


Ilustración 28: Interfaz gráfica menú de la Cruz de Santa Marta con el objeto Cruz invocada

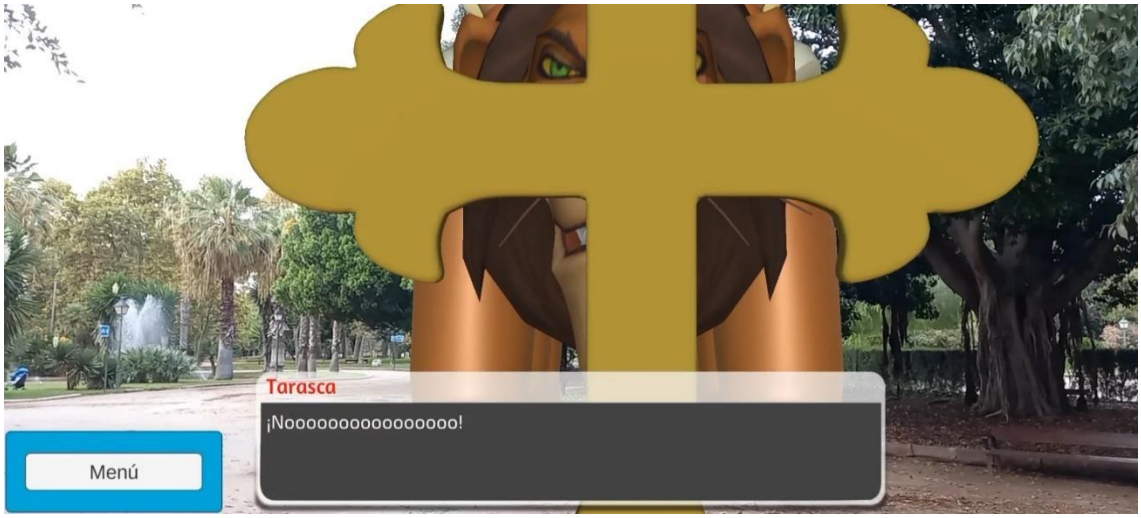


Ilustración 29: Interfaz gráfica menú de la Cruz de Santa Marta usada en la Tarasca

Ilustración 30: Interfaz gráfica con mensaje final

Las ilustraciones 28, 29 y 30 que detallan el funcionamiento de la acción “Cruz de Santa Marta”, muestran como el requisito funcional de usuario 14 se cumple.

Por último, la última interfaz que es mostrada durante la experiencia de realidad aumentada es la entrada al bestiario, a la que se accede desde el menú desplegado del bestiario al pulsar sobre cualquiera de los botones de éste. La interfaz muestra un panel que ocupa toda la pantalla en la que aparece el nombre de la bestia que se ha escogido, una imagen de la bestia e información sobre esta. En función de la bestia escogida, la imagen, el nombre y la información de la misma variarán.

Las siguientes ilustraciones muestran las distintas entradas de las bestias de las que el usuario puede obtener información que luego puede consultar, cumpliendo así el requisito funcional de usuario 9.

Ilustración 31: Interfaz gráfica menú de la entrada del bestiario sobre Anjana

Ilustración 32: Interfaz gráfica menú de la entrada del bestiario sobre Trenti

Ilustración 33: Interfaz gráfica menú de la entrada del bestiario sobre Musgosu

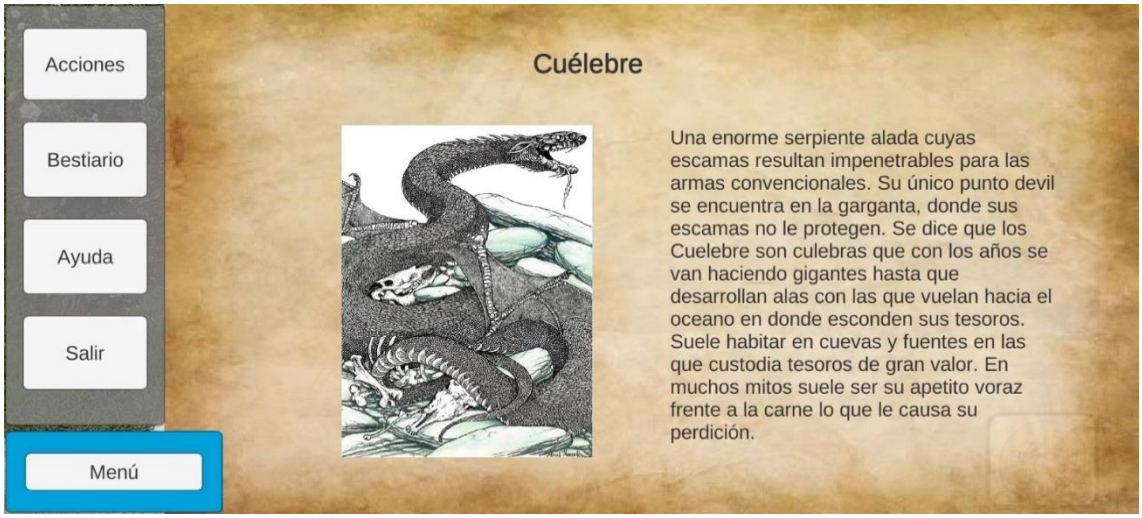


Ilustración 34: Interfaz gráfica menú de la entrada del bestiario sobre Cuélebre

Ilustración 35: Interfaz gráfica menú de la entrada del bestiario sobre Tarasca

4.2.4. Menú de victoria

La última escena de Unity con el nombre “Final”, presenta una interfaz muy simple, con un letrero que da las gracias al usuario por haber llegado hasta el final y un botón en el centro que devuelve al usuario al menú de inicio.

Ilustración 36: Interfaz gráfica del menú de victoria

Cabe mencionar que la aplicación tiene una música ambiental durante la experiencia de realidad aumentad y que cuando aparecen los cuadros de texto suena un sonido conforme aparecen las letras para captar la atención del usuario.

4.3. Implementación de la experiencia de realidad aumentada

En la ilustración 4 del capítulo 3.4 se ha presentado un diagrama de clases conceptual con el que se explicaba de forma resumida los conceptos más importantes que se daban en este proyecto y las relaciones que había entre cada uno de ellos. Más a la hora de realizar la implementación de la aplicación este diagrama no es lo suficientemente detallado. Es por ello por lo que basándose en el diagrama conceptual de la ilustración 4, se ha diseñado un diagrama de clases para la implementación en donde se muestran todas las clases empleadas y todos los atributos que hay en ellas.

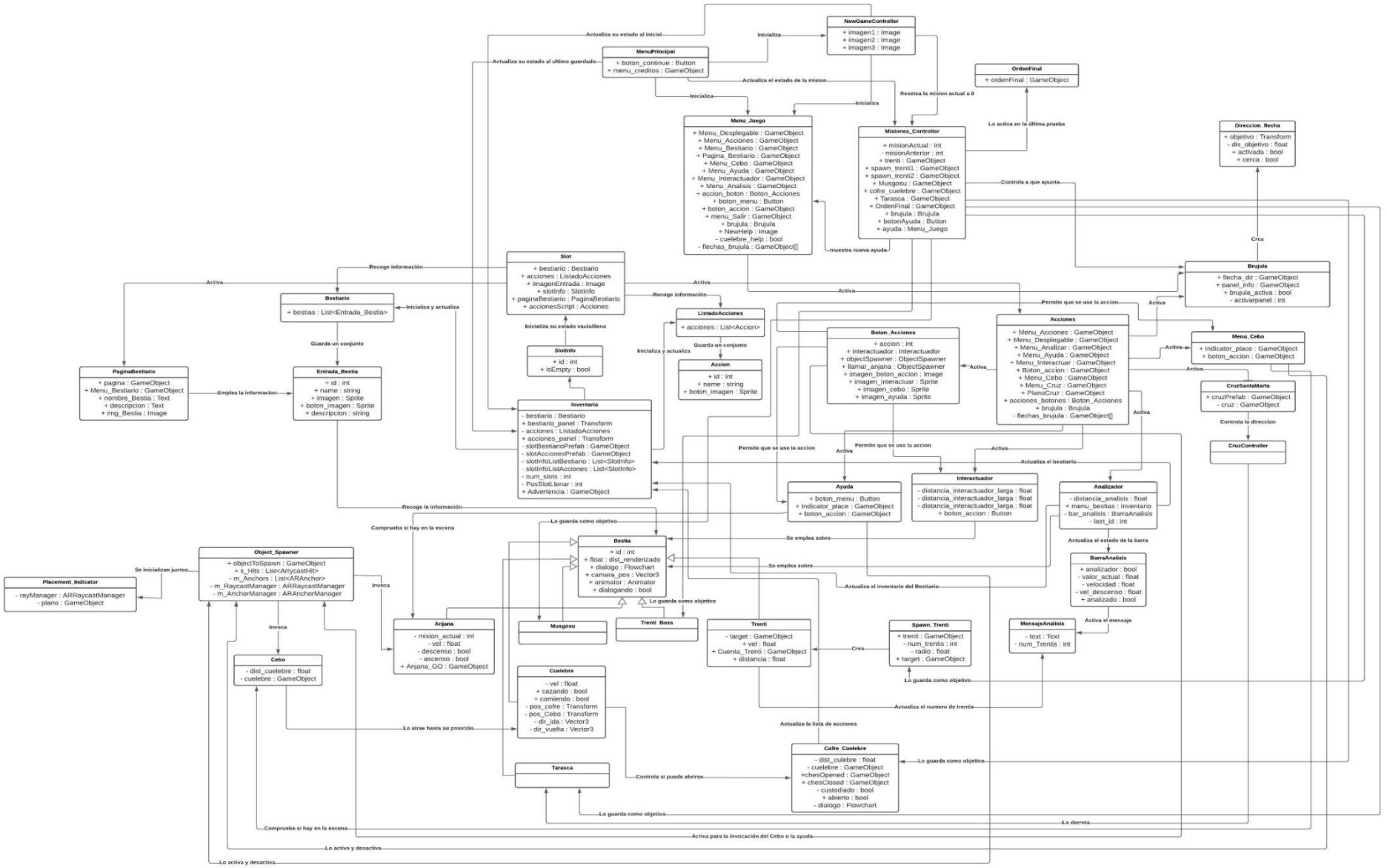


Ilustración 37: Diagrama de Clase de implementación

Como se puede apreciar en la ilustración 37, el diagrama final se ha complicado bastante, a la vez que se han realizado algunos cambios en su estructura y en las relaciones entre clases. En este capítulo se abordarán los motivos de estos cambios y como se han implementado los objetos y entidades más relevantes de la aplicación.

Antes de abordar esos apartados, cabe mencionar de forma muy resumida como se han estructurado las distintas escenas de Unity y las entidades que hay en ellas. Todo lo mencionado a continuación será profundizado en los siguientes apartados.

Escena "Inicio"

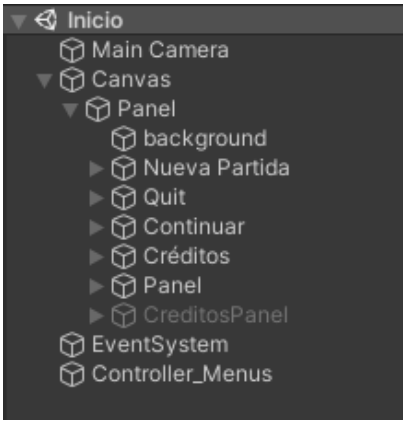


Ilustración 38: Escena Inicio

La imagen anterior muestra como la escena está compuesta por los elementos MainCamera que mostrará qué se verá por pantalla, Canvas que almacena todos los elementos UI del menú, EventSystem que administra el envío de eventos a objetos basado en input y el elemento Controller_Menus que se encarga de controlar el funcionamiento del menú.

Escena "NuevaPartida"

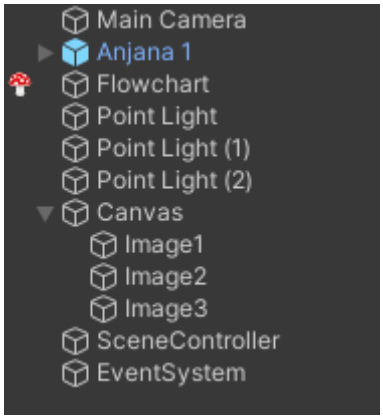


Ilustración 39: Escena NuevaPartida

En la ilustración 39, la escena está compuesta por diversos elementos. Al igual que en la anterior escena, esta tiene una MainCamera y un EventSystem. También hay un Canvas que en este caso solo almacena las tres imágenes que habrá en la escena. Los elementos que más destacan son un PlayerPref llamado Anjana_1 que representa a la bestia Anjana que hay en esta escena, un Flowchart que se encarga de crear el cuadro de dialogo de la Anjana y almacenar el texto que habrá en dicho cuadro, tres elementos Point_Light para iluminar la escena y un SceneController que se encarga de controlar el funcionamiento de la escena.

Escena "Juego"

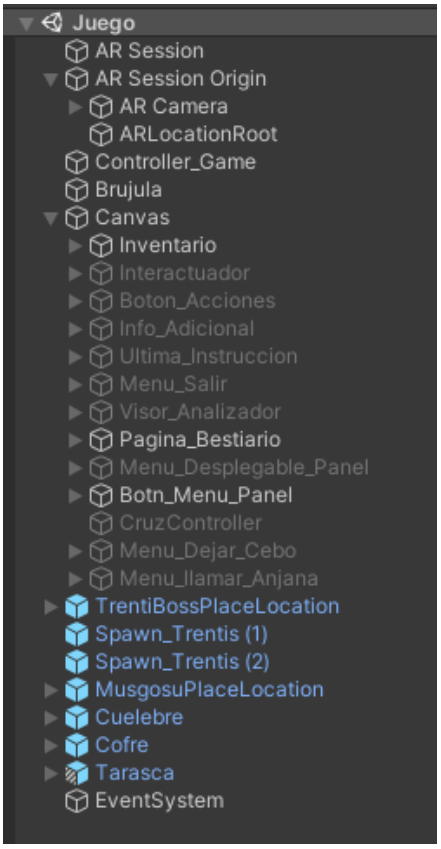


Ilustración 40: Escena Juego

En la ilustración 40 se muestra toda la estructura de la escena en la que se desarrollará la experiencia de realidad aumentada, lo que la convierte en la escena más relevante a la hora de explicar la implementación del proyecto.

Se puede observar que en este caso se tienen dos elementos llamados AR_Session y AR_Session_Origin, el cual tiene dentro los objetos AR_Camera y ARLocationRoot. Estos elementos son esenciales para que la experiencia de realidad aumentada funcione correctamente y se profundizará sobre ellos en el capítulo 4.3.2.

La escena posee un Controller_Game que se encarga de manejar el funcionamiento principal de la experiencia, centrándose en el comportamiento del menú, las pruebas que debe realizar el usuario y que acción está activada en cada momento.

El Canvas almacena los distintos menús de la aplicación y sus elementos UI, mientras que el elemento Brújula se encarga de controlar el funcionamiento de la acción Brújula que podrá emplear el usuario. En el capítulo 4.3.5 se detalla el funcionamiento de las distintas acciones que puede realizar el usuario.

Finalmente, todos los elementos que se pueden ver en color azul son los distintos Prefabs de las bestias que aparecen durante la experiencia. El comportamiento e implementación de las bestias se especifíca en el apartado 4.3.6.

Escena "Final"

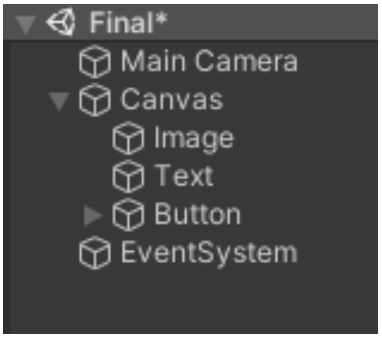


Ilustración 41: Escena Final

Esta última escena es la más sencilla de todas, pues solo posee una Main_Camera, un Canvas que almacena una imagen para el fondo, un texto y un botón para regresar al menú principal, y el EventSystem.

Como un aspecto relevante dentro de la implementación del proyecto, el paso entre cada una de las escenas a otra en Unity se ha realizado mediante la librería 'UnityEngine.SceneManagement'.

Con todas las estructuras de las escenas conocidas, se procede a detallar los aspectos más importantes de la implementación del proyecto Iberian Odyssey. Estos aspectos son: el uso y almacenamiento de datos, la configuración del funcionamiento de la experiencia RA, la implementación de los menús, la implementación del inventario, la implementación de las acciones, la implementación de las bestias y la implementación de las misiones.

4.3.1. Almacenamiento y uso de datos

Debido a que esta es la primera toma de contacto que se tiene con la tecnología de realidad aumentada y geolocalización por GPS, se ha optado por añadir un sistema de guardado automático que permita en caso de que hubiese algún tipo de malfuncionamiento por parte de la aplicación. De esta forma, en caso de que sucediese algún tipo de malfuncionamiento con el GPS que desviase la posición de los objetos digitales a lugares a los que el usuario no pudiese acceder, el usuario podrá cerrar la aplicación y reiniciarla sin perder nada del progreso realizado durante la experiencia de realidad aumentada.

Debido a que la cantidad de datos que deben ser guardados no es muy elevada y el tipo de dato a guardar no es complejo, se decide emplear los PlayerPrefs.

PlayerPrefs es una clase que permite almacenar el valor de variables de tipo string, float e integer entre sesiones de uso de la aplicación. Esto facilita enormemente la implementación, pues permite emplear dichos datos almacenados simplemente llamando al PlayerPref que los tiene guardados.

Para guardar los datos que se quieren almacenar simplemente se ha de llamar a la clase PlayerPrefs y luego a la función SetFloat, SetString o SetInt en función del tipo de dato que se quiera guardar. Dentro de la función escogida se ha de pasar como para metros el nombre de la variable en donde se vaya a guardar el valor y el propio valor que vaya a ser guardado.

La ilustración 42 muestra un ejemplo en la función EmpezarPartida de la clase NewGameControlle. Esta función le da el valor 1 al PlayerPref 'Juego_Nuevo' para indicar a la aplicación de que ya hay una partida empezada.

```
public void EmpezarPartida()
{
    PlayerPrefs.SetInt("Juego_Nuevo",1);
    SceneManager.LoadScene("Juego");
}
```

Ilustración 42: Ejemplo de uso de PlayerPrefs para guardar un valor tipo integer

Una vez guardada la variable 'Juego_Nuevo' de tipo integer con valor 1 en el PlayerPref, esta se puede usar posteriormente cuando sea necesaria. En este proyecto, esta variable es usada para habilitar o deshabilitar el botón Continuar en el menú de inicio. De esta forma si no hay ninguna partida previamente iniciada, el usuario no podrá emplear el botón Continuar y solo podrá emplear el botón de Nueva_Partida para iniciar la experiencia AR desde el principio.

La imagen 43 detalla como en la función Awake de la clase Menu_Principal se habilita o deshabilita el botón de continuar en función del valor del PlayerPref 'Juego_Nuevo'.

```
void Awake()
{
    if(PlayerPrefs.GetInt("Juego_Nuevo",0) == 0)
    {
        boton_continue.interactable = false;
    }

    else{
        boton_continue.interactable = true;
    }
}
```

Ilustración 43: Ejemplo uso de un PlayerPrefs para deshabilitar o habilitar un botón

Como puede apreciarse en la imagen anterior, en caso de que quiera usarse un PlayerPref se ha de emplear las funciones GetInt, GetFloat o GetString en función del tipo de valor que se quiera emplear. A dicha función se le pasarán dos variables como parámetros: el nombre de la variable que se vaya a emplear y el valor por defecto de la variable en caso de que no haya sido inicializada. Como se puede ver en la imagen anterior, el valor por defecto de 'Juego_Nuevo' es 0, si ese valor se mantiene la propiedad *interactable* del botón de continuar será false, por lo que el funcionamiento del botón estará desactivado. En caso de que sea distinta a 0, dicha propiedad tendrá un valor true y por tanto la funcionalidad del botón estará activa.

Las ilustraciones 42 y 43 son un ejemplo del uso que se les puede dar a los PlayerPrefs.

En este proyecto hay una serie de PlayerPrefs que será empleados para guardar determinados valores imprescindibles para que el usuario pueda comenzar de nuevo la experiencia RA desde el punto en donde lo dejó la última vez. Los PlayerPrefs empleados se pueden ver en la siguiente tabla:

Tabla 3: PlayerPrefs empleados en Iberian Odyssey

Nombre	Tipo	Valor por defecto	Finalidad
Juego_Nuevo	Integer	0	Si el valor es 0 implica que no hay ninguna partida guardada previamente, en caso contrario sí que la hay.
num_mision	Integer	1	Indica en que misión se encuentra el usuario en la partida que hay actualmente. El valor de esta variable puede variar del 1 al 7.
registro_Tarasca	Integer	0	Si el valor está en 1, significa que el usuario ha analizado a la Tarasca y sus datos están almacenados en el inventario. En caso contrario no se encontrará ningún registro de la bestia en el bestiario.
registro_Trenti	Integer	0	Si el valor está en 1, significa que el usuario ha analizado al Trenti y sus datos están almacenados en el inventario. En caso contrario no se encontrará ningún registro de la bestia en el bestiario.
registro_Musgosu	Integer	0	Si el valor está en 1, significa que el usuario ha analizado al Musgosu y sus datos están almacenados en el inventario. En caso contrario no se encontrará ningún registro de la bestia en el bestiario.

registro_Cuelebre	Integer	0	Si el valor está en 1, significa que el usuario ha analizado al Cuélebre y sus datos están almacenados en el inventario. En caso contrario no se encontrará ningún registro de la bestia en el bestiario.
registro_PonerCebo	Integer	0	Si el valor es 1, significa que el usuario ha desbloqueado la acción "Poner Cebo" y ahora se encuentra almacenada en la lista de acciones que puede realizar. En caso contrario dicha acción se encontrará bloqueada.
registro_Cruz	Integer	0	Si el valor es 1, significa que el usuario ha desbloqueado la acción "Cruz Santa Marta" y ahora se encuentra almacenada en la lista de acciones que puede realizar. En caso contrario dicha acción se encontrará bloqueada.

El PlayerPref 'Juego_Nuevo' adquiere el valor 0 cuando se pulsa el botón de "Nueva_Partida" en el menú inicial, y se le da el valor 1 cuando se pasa del menú de Nueva_Partida a la experiencia RA de la escena "Juego" empleando la función EmpezarPartida de la ilustración 42.

Al iniciar una nueva partida a todos los PlayerPrefs menos 'Juego_Nuevo' se modifica su valor para que el usuario comience la experiencia de realidad aumentada desde la primera misión, que no tenga registrada ninguna bestia en el bestiario a excepción de la Anjana y que tenga bloqueadas las acciones de "Poner Cebo" y la de "Cruz Santa Marta". Todo esto sucede en la función Awake de la clase NewGameController, tal y como se muestra en la ilustración 44.

```
private void Awake() {
    PlayerPrefs.SetInt("num_mision",1);
    PlayerPrefs.SetInt("registro_Tarasca",0);
    PlayerPrefs.SetInt("registro_Trenti",0);
    PlayerPrefs.SetInt("registro_Musgosu",0);
    PlayerPrefs.SetInt("registro_Cuelebre",0);

    PlayerPrefs.SetInt("registro_PonerCebo",0);
    PlayerPrefs.SetInt("registro_Cruz",0);

    imagen1.enabled = false;
    imagen2.enabled = false;
    imagen3.enabled = false;
}
```

Ilustración 44: Asignación de valores de PlayerPrefs para una nueva partida

Para cambiar el valor del PlayerPref 'num_mision' se emplea la función CambiarMision de la clase Misiones_Controller. Esta clase se encarga de controlar el funcionamiento de cada misión y en qué misión se encuentra el usuario en cada momento. Tal y como

se muestra en la siguiente imagen, la función CambiarMision comprueba que el usuario se encuentra en una nueva misión y acto seguido guarda el número de la misión actual en el PlayerPref para posteriormente activar la misión correspondiente.

```
public void Cambiar_Mision()
{
    misionActual++;
    if(misionActual != misionAnterior)
    {
        misionAnterior = misionActual;
        PlayerPrefs.SetInt("num_mision",misionActual);
        brujula.deleteObjetivos();
        switch(misionActual)
        {
            case 1:
                ActivarMision1();
                break;
            case 2:
                ActivarMision2();
                break;
            case 3:
                ActivarMision3();
                break;
            case 4:
                ActivarMision4();
                break;
            case 5:
                ActivarMision5();
                break;
            case 6:
                ActivarMision6();
                break;
            case 7:
                ActivarMisionFinal();
                break;
        }
        ayuda.MostrarNuevaAyuda();
    }
}
```

Ilustración 45: Uso de la función CambiarMision para guardar la misión en que se encuentra el usuario

Para alterar los valores de los PlayerPref que gestionan que bestias han sido analizadas en el bestiario, la clase Analizador una vez ha terminado de analizar la bestia, extrae el identificador de dicha bestia, añade la entrada de la bestia al inventario mediante la función AddEntradaBestiario y en función del identificador extraído, altera el valor del PlayerPref correspondiente a la bestia analizada. Todo este procedimiento puede verse en la ilustración 46.

```
last_id = hit.collider.GetComponent<Bestia>().id;
menu_bestias.AddEntradaBestiario(last_id);
switch(last_id)
{
    case 1:
        PlayerPrefs.SetInt("registro_Trenti",1);
        break;
    case 2:
        PlayerPrefs.SetInt("registro_Musgosu",1);
        break;
    case 3:
        PlayerPrefs.SetInt("registro_Cuelebre",1);
        break;
    case 4:
        PlayerPrefs.SetInt("registro_Tarasca",1);
        break;
}
```

Ilustración 46: Modificación de los PlayerPref en función de la bestia analizada

Finalmente, a la hora de cambiar el valor de los dos PlayerPrefs que se encargan de mantener activas las acciones de “Poner Cebo” y “Curz Santa Marta” cuando se inicia una partida ya comenzada, es necesario comentar que los valores de los PlayerPref son modificados cuando se realizan determinadas acciones.

En el caso de la modificación de ‘registro_PonerCebo’, esta es realizada cuando el usuario termina de interactuar con la bestia Musgosu. Durante esta acción, se emplea la función SiguienteMision de la clase Musgosu para activar en el inventario la acción “Poner Cebo” y pasar a la siguiente misión destruyendo al Musgoso al final. Es ahí cuando se modifica el valor de ‘registro_PonerCebo’ a 1, tal y como se puede ver en la ilustración 47.

```
public void SiguienteMision()
{
    GameObject.FindGameObjectWithTag("Inventario").GetComponent<Inventario>().AddEntradaAccion(2);
    PlayerPrefs.SetInt("registro_PonerCebo",1);
    FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
    Destroy(this.gameObject);
}
```

Ilustración 47: Modificación del PlayerPref registro_PonerCebo

Para la modificación del PlayerPref ‘registro_Cruz’, esta acción es realizada por la clase Cofre_Cuelebre, la cual emplea la función Activar_Cruz, que funciona de forma muy similar a la función SiguienteMision de la clase Musgosu pero sin destruir el GameObject en el que se encuentra la clase. Esto puede verse en la siguiente imagen.

```
public void Activar_Cruz()
{
    GameObject.FindGameObjectWithTag("Inventario").GetComponent<Inventario>().AddEntradaAccion(4);
    PlayerPrefs.SetInt("registro_Cruz",1);
    FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
}
```

Ilustración 48: Modificación del PlayerPref registro_Cruz

A la hora de emplear los distintos PlayerPrefs vistos para continuar con una experiencia RA comenzada previamente, se han de tener en cuenta los tres aspectos que se deben alterar: la misión en la que se encuentra el usuario, las acciones que tiene desbloqueadas y las entradas al bestiario que tiene desbloqueadas.
Para cargar la misión en la que se encuentra, el PlayerPref 'num_mision' será cargado en la función Awake de la clase Misión_Controller. Esta función se ejecuta cuando la clase es creada dentro de la escena, por lo que al iniciarse la clase se ejecutará esta función. Dependiendo del valor del PlayerPref, se activará una misión u otra y se destruirán algunos objetos que se destruirán en misiones previas a en la que se encuentra el usuario actualmente.

```
private void Awake() {
    misionActual = PlayerPrefs.GetInt("num_mision",1);
    switch(misionActual)
    {
            case 1:
                Debug.Log("Activada mision 1");
                ActivarMision1();
                break;
            case 2:
                Debug.Log("Activada mision 2");
                ActivarMision2();
                break;
            case 3:
                Destroy(spawn_trenti1);
                Destroy(spawn_trenti2);
                ActivarMision3();
                break;
            case 4:
                Destroy(spawn_trenti1);
                Destroy(spawn_trenti2);
                Destroy(trenti);
                ActivarMision4();
                break;
            case 5:
                Destroy(spawn_trenti1);
                Destroy(spawn_trenti2);
                Destroy(trenti);
                Destroy(Musgosu);
                ActivarMision5();
                break;
            case 6:
                Destroy(spawn_trenti1);
                Destroy(spawn_trenti2);
                Destroy(trenti);
                Destroy(Musgosu);
                cofre_cuelebre.GetComponent<Cofre_Cuelebre>().abierto = true;
                ActivarMision6();
                break;
            case 7:
                Destroy(spawn_trenti1);
                Destroy(spawn_trenti2);
                Destroy(trenti);
                Destroy(Musgosu);
                cofre_cuelebre.GetComponent<Cofre_Cuelebre>().abierto = true;
                Destroy(Tarasca);
                ActivarMisionFinal();
                break;
    }
}
```

Ilustración 49: Función Awake de Mision_Controller para cargar la misión actual

En el caso de cargar los elementos del bestiario y el listado de acciones, se emplea la función LoadInventario de la clase Inventario. La clase Inventario será detallada en el capítulo 4.3.3, pero en esta sección es importante ver que cuando se ejecuta la función LoadInventario al inicializarse la clase Inventario, dependiendo del valor de los PlayerPrefs que se encargan de las entradas al bestiario y acciones a las que puede acceder el usuario, la función añadirá determinadas entradas al bestiario y acciones.

```
AddEntradaBestiario(0);
if(PlayerPrefs.GetInt("registro_Trenti",0)==1)
    AddEntradaBestiario(1);
if(PlayerPrefs.GetInt("registro_Musgosu",0)==1)
    AddEntradaBestiario(2);
if(PlayerPrefs.GetInt("registro_Cuelebre",0)==1)
    AddEntradaBestiario(3);
if(PlayerPrefs.GetInt("registro_Tarasca",0)==1)
    AddEntradaBestiario(4);

AddEntradaAccion(0);
AddEntradaAccion(1);
AddEntradaAccion(3);
if(PlayerPrefs.GetInt("registro_PonerCebo",0)==1)
    AddEntradaAccion(2);
if(PlayerPrefs.GetInt("registro_Cruz",0)==1)
    AddEntradaAccion(4);
```

Ilustración 50: Uso de los PlayerPref para cargar entradas del inventario

Se puede apreciar como hay algunas entradas que no dependen de ser añadidas por ningún PlayerPref. Esto se debe a que esas entradas siempre estarán activadas al principio de cada experiencia RA.

4.3.2. Configuración del funcionamiento de la experiencia RA

Para la implementación de un proyecto en Unity que emplee realidad aumentada, es necesario configurar Unity previamente para que permita la creación de este tipo de aplicaciones. Para ello es necesaria la instalación de los assets AR Foundation y el plugin de ARCore para poder emplear sus funcionalidades. Tal y como se muestra en la ilustración 51.

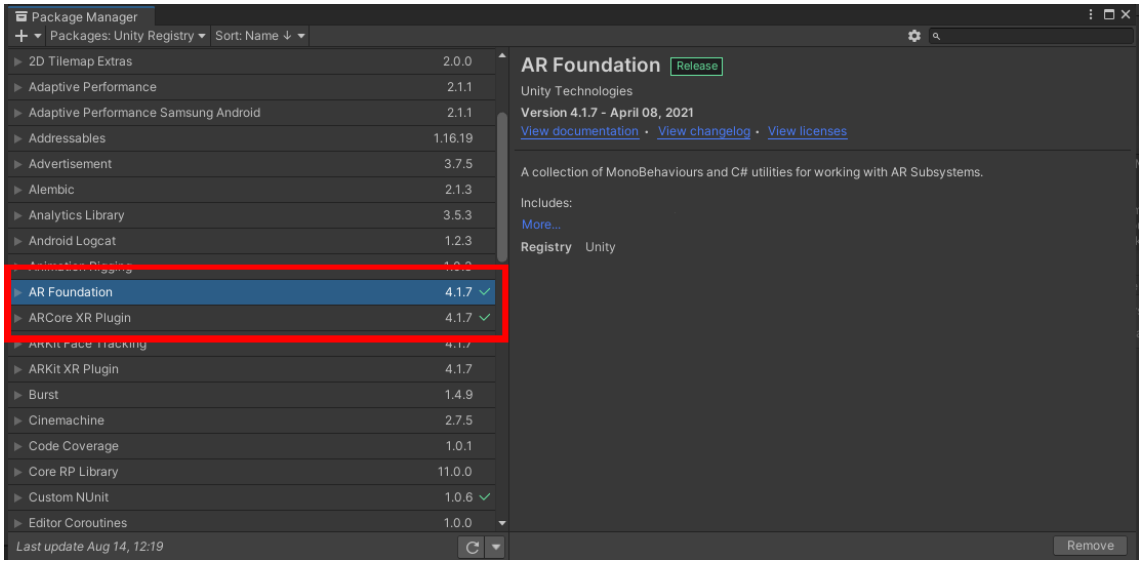


Ilustración 51: Lista de assets instalados para trabajar con RA en Unity

Tal y como se menciona en el capítulo 4.1.1.1, AR Foundation es un framework de Unity que ofrece una serie de características que permiten la creación de aplicaciones de realidad aumentada. De entre estas características se puede destacar la capacidad de detectar planos, la cual será empleada para realizar algunas funcionalidades en Iberian Odyssey (ver capítulo 4.3.5).

Una vez instalados los assets, se han de crear dos objetos en la escena que serán fundamentales para crear la experiencia RA. El AR Session y el AR Session Origin.

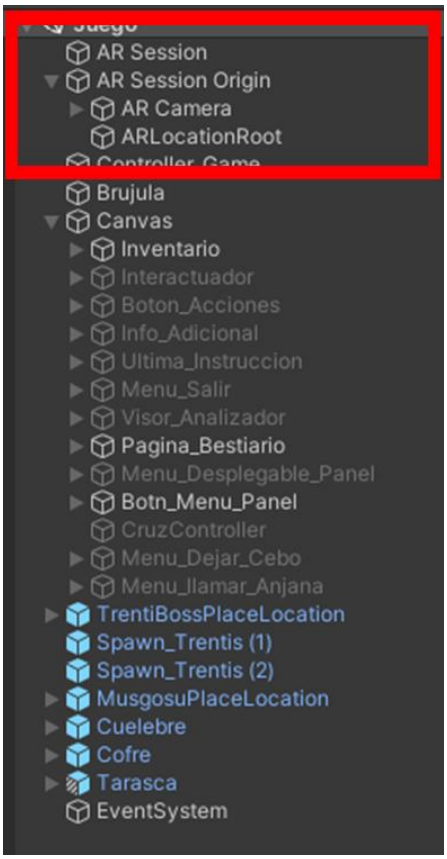


Ilustración 52: Elementos de control de RA en la escena

- AR Session [28]: se encarga de habilitar y deshabilitar las propiedades de la realidad aumentada en la plataforma en la que se emplea mediante los dos componentes que se pueden ver en la imagen 53: el script AR Session y el script AR Input Manager.

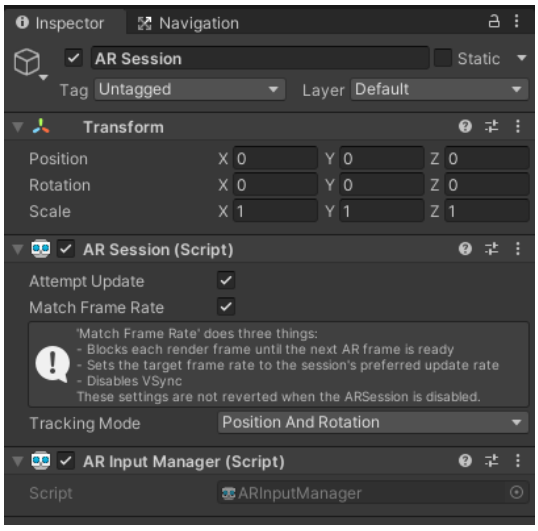


Ilustración 53: Configuración del objeto AR Session

- AR Session Origin [29]: este elemento tiene asociado la cámara principal de la experiencia de realidad aumentada, tal y como se puede ver en la ilustración 49 con el nombre AR Camera. AR Session Origin capta información de distintos elementos rastreables del entorno (como superficies planas o puntos de anclaje) para convertir el espacio que conforman esos elementos en el espacio de la escena de Unity, permitiendo escalar, orientar y posicionar la cámara que hay en dicha escena para que se integre de forma natural en la experiencia RA. Los componentes que permiten el funcionamiento de AR Session Origin son: AR Session Origin (se encarga de recopilar la cámara y cualquier GameObjetc como planos o puntos de anclaje para posicionar la cámara en la experiencia RA), AR Plane Manager (se encarga de detectar superficies planas), AR Anchor Manager (se encarga de detectar puntos de anclaje en la escena), AR Point Cloud Manager (se encarga de detectar nubes de puntos en la escena para detectar formas de objetos del entorno y medir la profundidad de los mismos) y el AR Raycast Manager (que permite emplear la funcionalidad de los Raycast en elementos de ARFoundation).

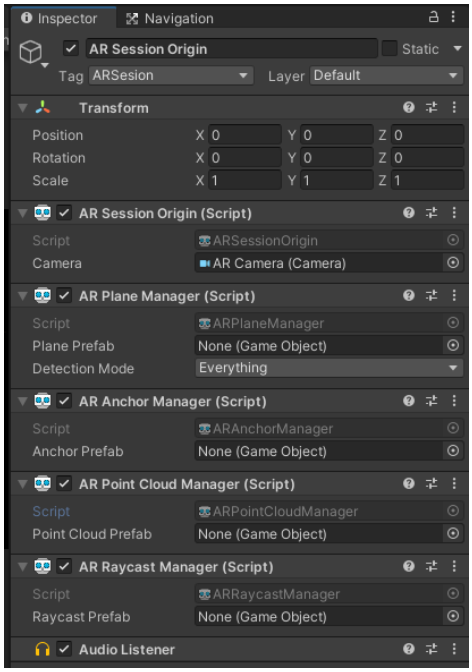


Ilustración 54: Componentes de AR Session Origin

El objeto AR Camera que hay dentro del AR Session Origin es una cámara común de Unity, pero con componentes añadidos que le permiten adaptarse al entorno RA. También se le ha añadido un AudioSource para que pueda generar la banda sonora que se escuchará durante la experiencia RA.

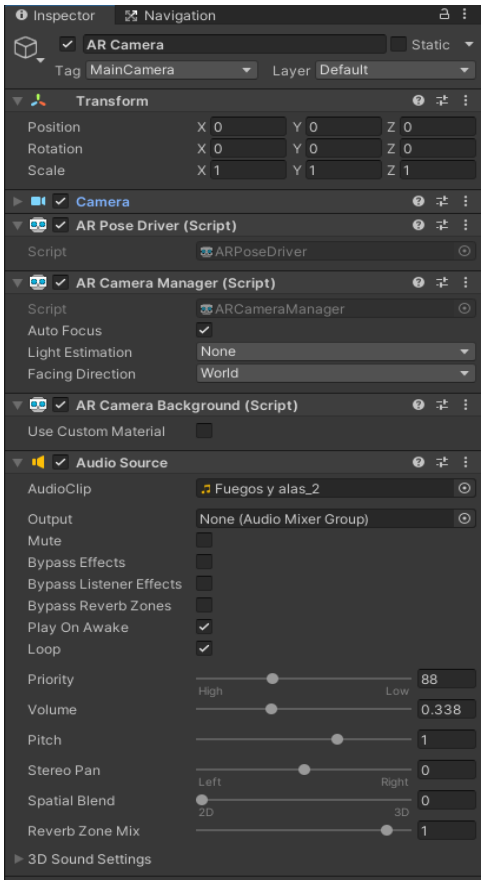


Ilustración 55: Componentes de AR Camera

A parte de AR Foundation, se ha empleado el asset AR+GPS Location, que permite colocar objetos de Unity en lugares del mundo real empleando las coordenadas GPS, tal y como se menciona en el capítulo 4.1.1.1. El elemento que se encarga de manejar todos los cálculos y la información del GPS para colocar los objetos en el lugar correcto es el objeto ARLocationRoot [30], el cual debe ser colocado como junto al objeto AR Camera.

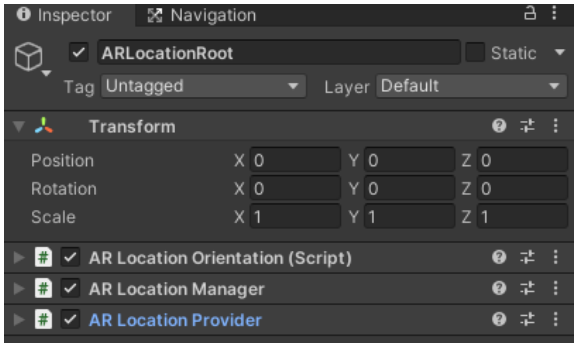


Ilustración 56: Componentes de ARLocationRoot

Hay tres componentes que definen el funcionamiento de ARLocationRoot. ARLocationManager y ARLocationOrientation permiten que ARLocationRoot se alinee con las direcciones geográficas, permitiendo colocar los objetos correctamente. Por otra parte, está el componente ARLocationProvider que se encarga de manejar toda la información proveniente del GPS del dispositivo.

Por último, para colocar objetos digitales en posiciones geográficas del GPS, se debe de añadir el componente PlaceAtLocation en el objeto que se quiera geoposicionar, en dicho componente se colocan las coordenadas de latitud y longitud. En la ilustración 57 se puede observar cómo se emplea el componente PlaceAtLocation para posicionar el GameObject TrentiBossPlaceLocation en unas coordenadas específicas.

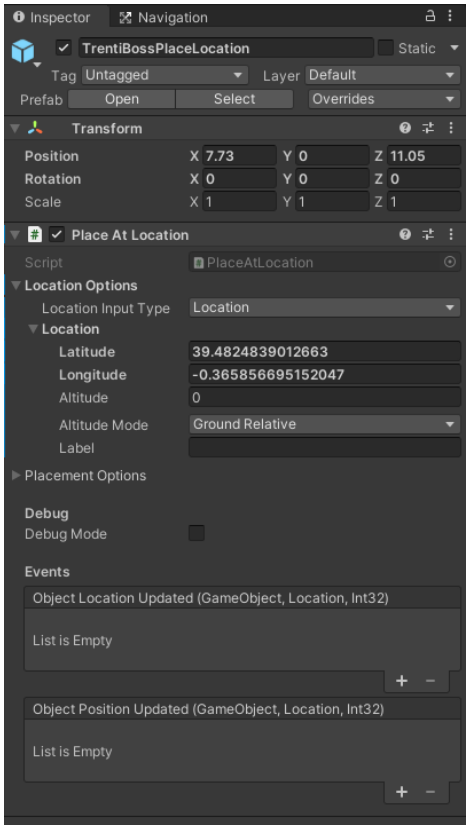


Ilustración 57: Uso del componente PlaceAtLocation

4.3.3. Implementación del inventario

Como puede verse en el diagrama de clases de la ilustración 37, existe una clase en el diagrama llamada Inventario. Esta clase se encarga de inicializar y actualizar las entradas al bestiario y la lista de acciones a las que tiene acceso el usuario. Previo a describir la implementación del funcionamiento de Inventario, hay que entender qué va a ser almacenado en los menús del bestiario y de acciones que pueden verse en las ilustraciones 13 y 14 del capítulo 4.2.3.

En el primer caso, el bestiario almacenará una serie de entradas al bestiario de aquellas bestias que el usuario haya analizado, permitiéndole al usuario visualizar una imagen de la bestia analizada y una descripción de la misma. Para representar la entrada al bestiario se ha creado la clase **Entrada_Bestia** (ver anexo 1 apartado 33), en la imagen 58 se pueden ver los distintos atributos que almacena esta clase:

- **id**: un identificador de la entrada al bestiario.

- **name**: el nombre de la bestia que ocupa la entrada.
- **imagen**: la imagen que acompañará la entrada al bestiario.
- **boton_imagen**: la imagen del botón que da a esta entrada del bestiario.
- **descripcion**: la descripción de la bestia.

```
public class Entrada_Bestia
{
    1 reference
    public int id;
    1 reference
    public string name;
    1 reference
    public Sprite imagen;
    1 reference
    public Sprite boton_imagen;
    1 reference
    public string descripcion;
}
```

Ilustración 58: Clase Entrada_Bestia

Por otro lado, el listado de acciones almacena las distintas acciones que el usuario puede realizar. Cada una de estas acciones es representada por la clase **Acción** (ver anexo 1 apartado 32) la cual podemos ver en la ilustración 59 y almacena los siguientes atributos:

- **id:** un identificador de la acción almacenada.
- **name:** el nombre de la acción.
- **boton_imagen:** la imagen del botón que permite emplear la acción.

```
public class Accion
{
    2 references
    public int id;
    0 references
    public string name;
    1 reference
    public Sprite boton_imagen;
}
```

Ilustración 59: Clase Acción

Una vez implementadas las clases en donde se almacenará la información de los elementos que habrá en el bestiario y en el listado de acciones, es necesario rellenar estos elementos y almacenarlos para que luego la clase Inventario pueda recoger cada una de las distintas entradas al inventario y acciones y actualizar a que elementos tiene acceso el usuario. Para ellos se emplea un tipo de clase llamada ScriptableObject [31],

la cual permite almacenar grandes cantidades de datos independientes de scripts ya instanciados.

En este proyecto se han creado dos clases de este tipo: **Bestiario** (ver anexo 1 apartado 33) para almacenar los distintos elementos del tipo Entrada_Bestia y **ListadoAcciones** (ver anexo 1 apartado 32) para almacenar los distintos elementos del tipo Accion. Como se pueden ver en las ilustraciones 60 y 61, ambas clases tienen una función que les permite buscar uno de los elementos que almacenan por medio de su identificador.

```
[CreateAssetMenu(menuName = "Inventory System/Bestiario")]
2 references
public class Bestiario : ScriptableObject
{
    1 reference
    public List<Entrada_Bestia> bestias = new List<Entrada_Bestia>();

    5 references
    public Entrada_Bestia FindEntrada_Bestia(int id) {
        foreach (Entrada_Bestia b in bestias)
        {
            if (b.id == id)
            {
                return b;
            }
        }
        return null;
    }
}
```

Ilustración 60: Clase Bestiario

```
[CreateAssetMenu(menuName = "Inventory System/Listado de acciones")]
2 references
public class ListadoAcciones : ScriptableObject
{
    1 reference
    public List<Accion> acciones = new List<Accion>();

    3 references
    public Accion FindAcciones(int id) {
        foreach (Accion a in acciones)
        {
            if (a.id == id)
            {
                return a;
            }
        }
        return null;
    }
}
```

Ilustración 61: Clase ListadoAcciones

La primera línea de código que se muestra en las imágenes anteriores permite que se puedan crear instancias de un ScriptableObjetc, en este proyecto se ha creado una instancia de Bestiario y otra de ListadoAcciones. En las siguientes imágenes se muestran ambas instancias y como se han rellenado las distintas Entrada_Bestiario y Acción que hay en las instancias de Bestiario y ListadoAcciones respectivamente.

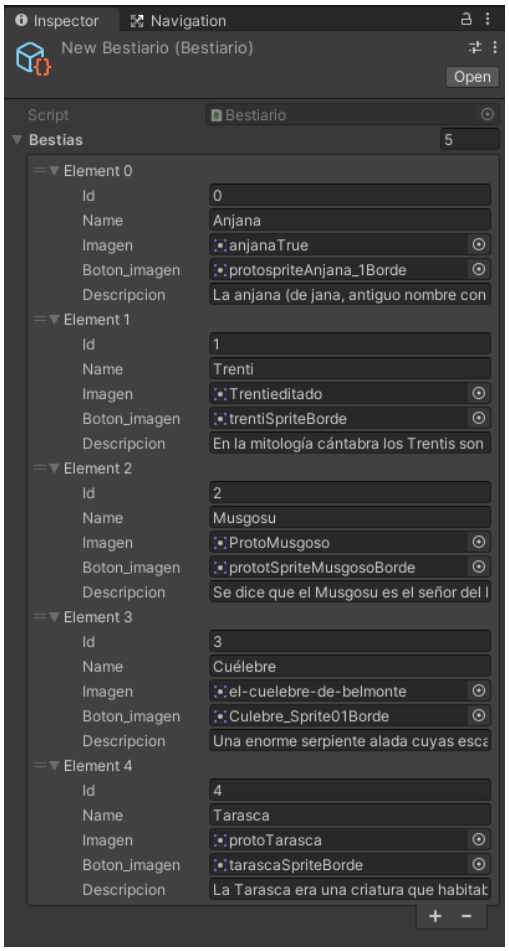


Ilustración 62: Instancia del Bestiario

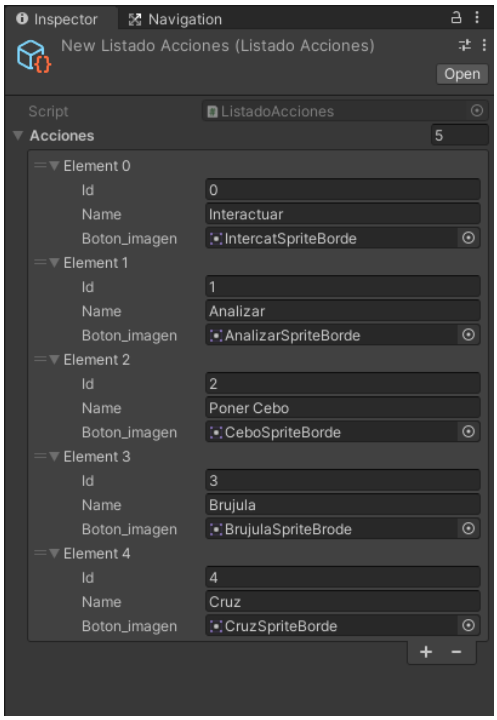


Ilustración 63: Instancia del ListadoAcciones

Una vez explicado qué información va a ser empleada por los menús del bestiario y de las acciones de la ilustración 63, es momentos de explicar cómo van a ser utilizada esta información para que el usuario pueda emplearla.

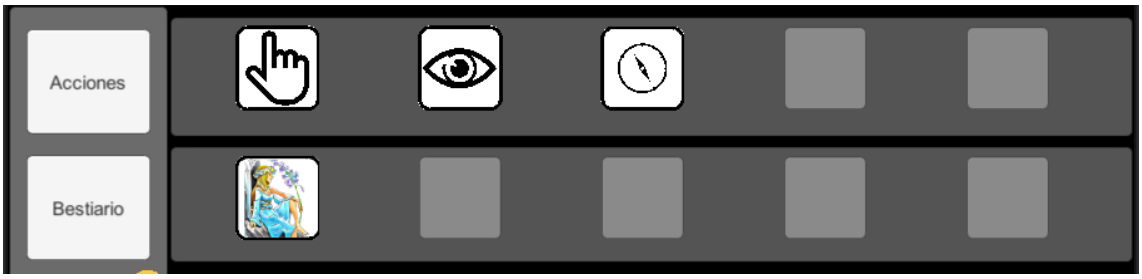


Ilustración 64: Menús desplegables del listado de acciones y del bestiario con botones vacíos y llenos

Como se puede ver en la ilustración anterior, ambos menús desplegables tienen una serie de botones que representan las acciones y las entradas al bestiario a los que el usuario tiene acceso. Aquellos botones con ilustraciones son los que el usuario puede emplear y los que tienen un fondo grisáceo son aquellos que el usuario tiene bloqueados.

Cada uno de estos botones es un objeto prefabricado de Unity (conocidos como Prefabs [32]). Los botones empleados en el menú de acciones son Prefabs del tipo **SlotAcciones**, mientras que los empleados en el bestiario son del tipo **SlotBestiario**. Ambos Prefabs poseen los mismos componentes, salvo que se diferencian en que los scripts de la clase **Slot** de cada uno de los Prefabs tiene algunas de sus variables vacías y otras llenas, tal y como se ve en la ilustración 65.

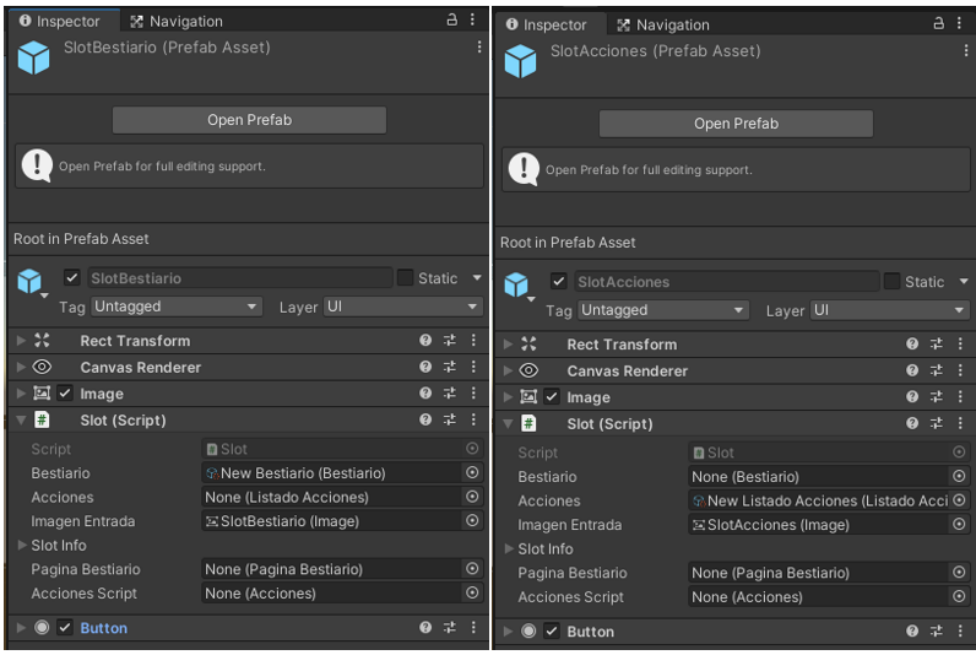


Ilustración 65: Prefabs SlotBestiario y SlotAcciones

La clase Slot puede verse al completo en el anexo 1 apartado 30, dicha clase presenta los siguientes atributos:

- **bestiario**: En caso de estar en el Prefab SlotBestiario, esta variable le permite conectar el Slot a la instancia del Bestiario, permitiéndole tener acceso a la información del elemento Entrada_Bestia asociada a ese Slot. En caso de estar en el Prefab SlotAcciones, esta variable permanecerá vacía.
- **acciones**: En caso de estar en el Prefab SlotAcciones, esta variable le permite conectar el Slot a la instancia del ListadoAcciones, permitiéndole tener acceso a la información del elemento Accion asociada a ese Slot. En caso de estar en el Prefab SlotBestiario, esta variable permanecerá vacía.
- **imagenEntrada:** Es la imagen asociada a la acción o a la entrada del bestiario que el Slot empleará en caso de no estar vacío.
- **slotInfo:** Se trata de una variable que hace referencia a la clase **SlotInfo** (ver anexo 1 apartado 30), la cual se encarga de controlar si el botón del Slot no tiene ninguna entrada al bestiario o acción asociada, y el identificador del elemento al que está asociado el Slot.

```
public class SlotInfo
{
    14 references
    public int id;
    11 references
    public bool isEmpty;

    3 references
    public void EmptySlot()
    {
        isEmpty = true;
    }


}
```

Ilustración 66: Clase SlotInfo

Estos son los atributos de la clase SlotInfo:

- **id**: se trata del identificador del Slot al que está asociada la instancia de SlotInfo, en caso de que el Slot tenga asociado algún elemento del tipo Entrada_Bestia o Accion, esta variable será igual al identificador del elemento asociado.
- **isEmpty**: esta variable booleana tendrá un valor true si hay un elemento del tipo Entrada_Bestia o Accion asociado al Slot.

La clase también posee una función para cambiar el valor de la variable isEmpty.

- **paginaBestiario:** En caso de estar en el Prefab SlotBestiario, esta variable hace referencia a la clase **PaginaBestiario** del objeto "Pagina_Bestiario", del cual hablaremos más adelante en profundidad. Dicha clase permite mostrar por pantalla los elementos de la Entrada_Bestia asociada al Slot (nombre de la

Bestia, imagen y descripción). Para mostrar dicha información mediante la clase PaginaBestiario, se emplea la función OpenPageBest de dicha clase, la cual recoge toda la información de la bestia asociada al SlotBestiario y la muestra por pantalla. Esta función se ejecutará cuando el usuario pulse el botón del menú del bestiario asociado al SlotBestiario. En caso de estar en el Prefab SlotAcciones, esta variable permanecerá vacía.

- **accionesScript:** En caso de estar en el Prefab SlotAcciones, esta variable hace referencia la clase **Acciones** del objeto Controller_Game (del cual se hablará en profundidad en el capítulo 4.3.5). Dicha clase permite controlar que acción puede realizar el usuario está activada en el momento mediante las distintas funciones que tiene, por lo que la variable 'accionesScript' permite asociar el SlotAcciones con una acción asociada a la función de la clase Acciones que permite activar dicha acción. La forma de relacionar dicha función con la acción a la que hacer referencia el Slot se puede ver en la imagen 67 mediante la función ActiveActions que se ejecutará cuando se pulse el botón del menú de acciones asociado al SlotAcciones. En caso de estar en el Prefab SlotBestiario, esta variable permanecerá vacía.

```
public void ActiveActions()
{
    switch(acciones.FindAcciones(slotInfo.id).id)
    {
        case 0:
            accionesScript.Onclick_Interactuar();
            break;
        case 1:
            accionesScript.Onclick_Analizar();
            break;
        case 2:
            accionesScript.Onclick_PonerCebo();
            break;
        case 3:
            accionesScript.Onclick_Brujula();
            break;
        case 4:
            accionesScript.Onclick_Cruz();
            break;
    }
}
```

Ilustración 67: Función ActiveActions que asocia una función de activación de una acción al Slot

Tal y como se ha mencionado durante la explicación de las variables 'paginaBestiario' y 'accionesSript', se asocia una función a la pulsación del botón del Slot.

En el caso de los SlotBestiario, la función asociada es OpenPageBest que permite emplear la función AbrirPaginaBestiario de la clase PaginaBestiario por medio de la variable 'paginaBestiario' para mostrar por pantalla la información de la entrada del bestiario asociada al SlotBestiario.

```
public void OpenPageBest()
{
    paginaBestiario.AbrirPaginaBestiario(bestiario.FindEntrada_Bestia(slotInfo.id).name,
                                         bestiario.FindEntrada_Bestia(slotInfo.id).descripcion,
                                         bestiario.FindEntrada_Bestia(slotInfo.id).imagen);
}
```

Ilustración 68: Función OpenPageBest de la clase Slot

Tal y como se muestra en la imagen 69, la clase PaginaBestiario posee dos variables del tipo GameObject: página que hace referencia al objeto de la escena de Unity que muestra la información de las bestias por pantalla y Menu_bestiario que hace referencia al menú desplegable del bestiario.

```
public class PaginaBestiario : MonoBehaviour
{
    // Start is called before the first frame update
    1 reference
    public GameObject pagina;
    1 reference
    public GameObject Menu_Bestiario;
    1 reference
    public Text nombre_Bestia;
    1 reference
    public Text descripcion;
    1 reference
    public Image img_Bestia;

    1 reference
    public void AbrirPaginaBestiario(string name, string descr, Sprite img)
    {
        pagina.SetActive(true);
        Menu_Bestiario.SetActive(false);
        nombre_Bestia.text = name;
        descripcion.text = descr;
        img_Bestia.sprite = img;

    }

}
```

Ilustración 69: Clase PaginaBestiario

La función AbrirPaginaBestiario simplemente abre el panel en donde se mostrará la información mientras cierra el panel del menú desplegable del bestiario y le asocia a los distintos elementos (dos textos y una imagen) de la página del bestiario la información de la entrada del bestiario. El resultado puede verse en las imágenes 31, 32, 33, 34 y 35 del capítulo 4.2.3, en donde en función de la información del elemento Entrada_Bestia asociada al SlotBestiario que ha sido pulsado, la imagen y los textos del título y la descripción de la bestia cambian.

Por otro lado, en el caso de los SlotAcciones, la función asociada a la pulsación del Slot es ActiveActions, la cual se puede ver en la imagen 67. Esta función permite emplear la variable 'accionesScript' que hace referencia a la clase Aciones para emplear las funciones de dicha clase que activan las acciones que el usuario puede realizar en función del 'id' asociado al SlotAcciones. El funcionamiento de cada una de las funciones que pueden ser usadas para activar una acción se verá en detalle en el capítulo 4.3.4.

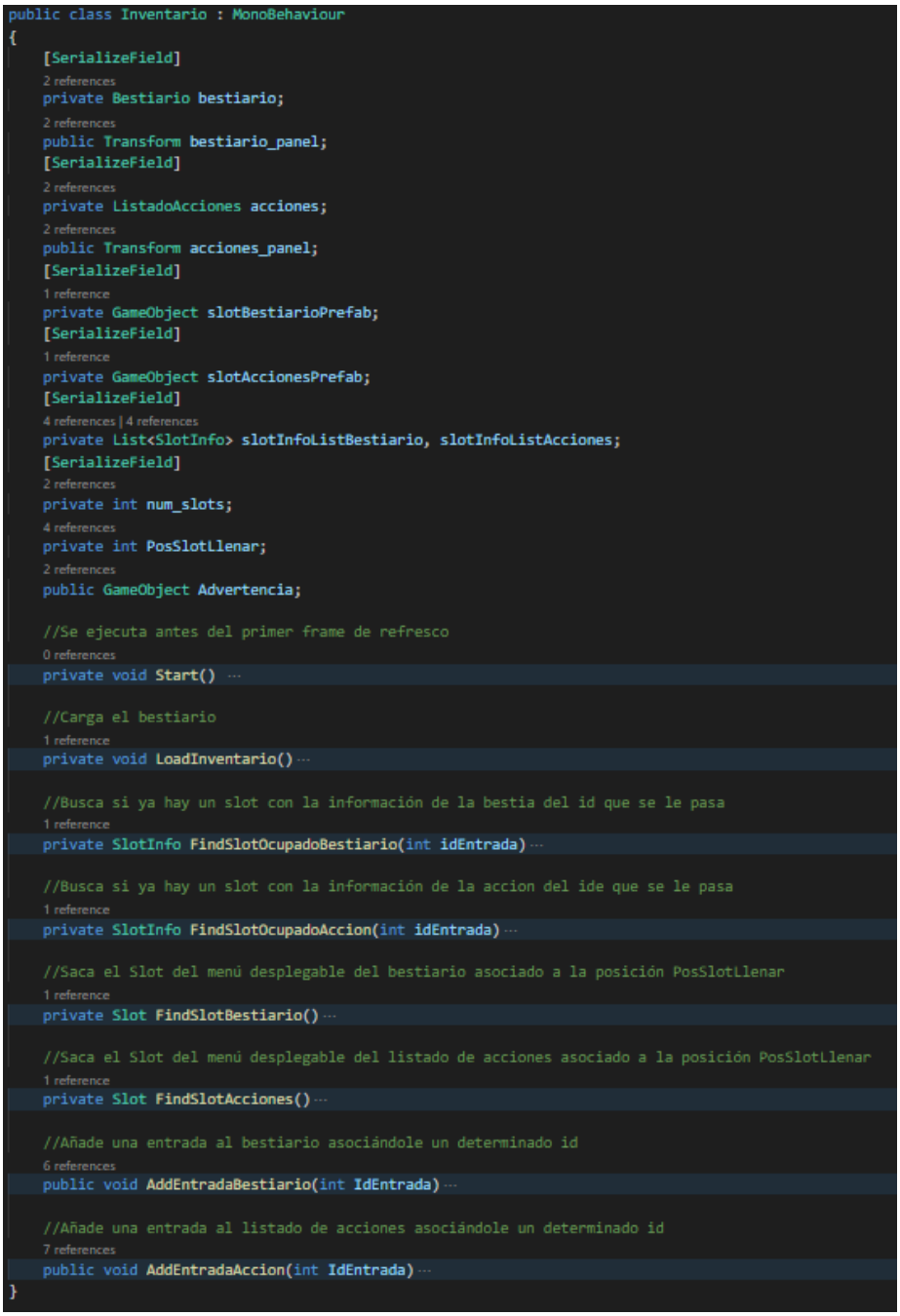
Para asociar tanto un elemento Entrada_Bestiario como un elemento Accion a un Slot, se emplea la clase UpdateUI, que como se ve en la imagen 69, asocia la imagen del Slot

a la imagen que guarda el elemento Entrada_Bestia o Acción en la variable Boton_Imagen, asocia la variable 'paginabestiario' a la clase PaginaBestiario y la variable 'accionesScript' a la clase Acciones, y asocia al evento Onclick del botón del Slot la función OpenPageBest en caso de ser un SlotBestiario y en caso de ser un SlotAcciones lo asociará a la función ActiveActions.

```
public void UpdateUI()
{
    if(slotInfo.isEmpty)
    {
        imagenEntrada.sprite = null;
        imagenEntrada.enabled = false;
        this.GetComponent<Button>().interactable = false;
    }

    else
    {
        Button boton_slot = this.GetComponent<Button>();
        boton_slot.interactable = true;

        if(bestiario != null)
        {
            imagenEntrada.sprite = bestiario.FindEntrada_Bestia(slotInfo.id).boton_imagen;
            paginaBestiario = (PaginaBestiario) FindObjectOfType(typeof(PaginaBestiario));
            boton_slot.onClick.AddListener(OpenPageBest);
        }
        else
        {
            imagenEntrada.sprite = acciones.FindAcciones(slotInfo.id).boton_imagen;
            accionesScript = (Acciones) FindObjectOfType(typeof(Acciones));
            boton_slot.onClick.AddListener(ActiveActions);
        }


        imagenEntrada.enabled = true;


    }
}
```

Ilustración 70: Función UpdateUI

Ya con todos los elementos mencionados, solo falta explicar la implementación de la clase **Inventario**, la cual se encarga de carga y actualiza los dos menús del bestiario y las acciones. Esta clase emplea las distintas clases que se han visto en este apartado para garantizar un funcionamiento correcto tanto en el menú del bestiario como en el del listado de acciones. Dado que es la clase más importante a la hora de manejar todo lo referente al sistema de inventario de la aplicación, habrá que explicar de forma clara que es lo que hace esta clase.



```
public class Inventario : MonoBehaviour
{
    [SerializeField]
    2 references
    private Bestiario bestiario;
    2 references
    public Transform bestiario_panel;
    [SerializeField]
    2 references
    private ListadoAcciones acciones;
    2 references
    public Transform acciones_panel;
    [SerializeField]
    1 reference
    private GameObject slotBestiarioPrefab;
    [SerializeField]
    1 reference
    private GameObject slotAccionesPrefab;
    [SerializeField]
    4 references | 4 references
    private List<SlotInfo> slotInfoListBestiario, slotInfoListAcciones;
    [SerializeField]
    2 references
    private int num_slots;
    4 references
    private int PosSlotLlenar;
    2 references
    public GameObject Advertencia;

    //Se ejecuta antes del primer frame de refresco
    0 references
    private void Start() ···

    //Carga el bestiario
    1 reference
    private void LoadInventario() ···

    //Busca si ya hay un slot con la información de la bestia del id que se le pasa
    1 reference
    private SlotInfo FindSlotOcupadoBestiario(int idEntrada) ···

    //Busca si ya hay un slot con la información de la accion del ide que se le pasa
    1 reference
    private SlotInfo FindSlotOcupadoAccion(int idEntrada) ···

    //Saca el Slot del menú desplegable del bestiario asociado a la posición PosSlotLlenar
    1 reference
    private Slot FindSlotBestiario() ···

    //Saca el Slot del menú desplegable del listado de acciones asociado a la posición PosSlotLlenar
    1 reference
    private Slot FindSlotAcciones() ···

    //Añade una entrada al bestiario asociándole un determinado id
    6 references
    public void AddEntradaBestiario(int IdEntrada) ···

    //Añade una entrada al listado de acciones asociándole un determinado id
    7 references
    public void AddEntradaAccion(int IdEntrada) ···
}
```

Ilustración 71: Clase Inventario

El script Inventario hace referencia a la clase Inventario que se muestra en la imagen anterior puede ser visto en el anexo 1 apartado 31 para ver en detalle cómo se han implementado los métodos que hay en dicha clase. Las variables que emplea esta clase son las siguientes:

- **bestiario**: hace referencia a la clase Bestiario vista en la imagen 60.
- **bestiario_panel**: hace referencia al objeto de la escena de Unity que equivale al menú desplegable del bestiario.
- **acciones**: hace referencia a la clase ListadoAcciones vista en la imagen 61.
    - **acciones_panel**: hace referencia al objeto de la escena de Unity que equivale al menú desplegable del listado de acciones.

- **slotBestiarioPrefab**: hace referencia al objeto prefabricado de Unity SlotBestiario.
- **slotBestiarioPrefab**: hace referencia al objeto prefabricado de Unity SlotAcciones.
- **slotInfoListBestiario**: será el listado de InfoSlots de los SlotBestiario que indicarán que 'id' tienen y si dichos slots del menú del bestiario estarán vacíos o llenos.
- **slotInfoListAcciones**: será el listado de InfoSlots de los SlotAcciones que indicarán que 'id' tienen y si dichos slots del menú del listado de acciones estarán vacíos o llenos.
- **num_slots**: este número indicará cuantos Slots habrá en el menú del listado de acciones y en el del bestiario.
- **PosSlotLlenar**: esta variable es empleada para guardar cual es la posición del Slot que va a usarse para añadir nueva información tanto en el menú del listado de acciones como en el bestiario.
- **Advertencia**: hace referencia al objeto prefabricado de nombre Advertencia, el cual es un simple elemento UI que muestra un mensaje por pantalla.

Tras haber visto los distintos atributos que emplea la clase, es momento de explicar el funcionamiento los distintos métodos que emplea.

- **Start:** Esta función se ejecuta al iniciarse el primer frame en el que la clase está activada. La función inicializa dos listas de los slots que serán los SlotBestiario del bestiario y los SlotAcciones del listado de acciones y llama al método LoadInventario.
- **LoadInventario:** Este método carga todo el contenido de los menús del bestiario y el listado de acciones. Primero creando en el menú del bestiario cinco instancias de SlotBestiario vacíos y en el menú de acciones otras cinco instancias de SlotAcciones también vacíos. Después, tal y como se ha visto en la imagen 50, se añaden a esos Slots la información de entradas al bestiario para los SlotBestiario o la información de acciones para SlotAcciones, llamando a la función AddEntradaBestiario y AddEntradaAccion respectivamente, en función del valor de los PlayerPref que indican que bestias han sido analizadas y que acciones desbloqueadas.
- **FindSlotOcupadoBestiario:** Se emplea para averiguar si existe ya un Slot en el menú del Bestiario que tenga almacenada la información de una determinada bestia en función del 'id' que se le pasa como parámetro a la función. En caso de que exista ese Slot, se devuelve el componente SlotInfo del mismo, en caso de que no exista se devuelve el componente SlotInfo del primer Slot que no tenga asociada ninguna entrada al bestiario. En caso de que todos los Slots estén ocupados y no haya ninguno con la información asociada al id, no devolverá nada. En esta función también se le da valor a la variable

PosSlotLlenar, que es la posición del Slot del que se coge su SlotInfo dentro del listado de Slots.

- **FindSlotOcupadoAccion:** Se emplea para averiguar si existe ya un Slot en el menú del listado de acciones que tenga almacenada la información de una determinada acción en función del 'id' que se le pasa como parámetro a la función. En caso de que exista ese Slot, se devuelve el componente SlotInfo del mismo, en caso de que no exista se devuelve el componente SlotInfo del primer Slot que no tenga asociada ninguna acción. En caso de que todos los Slots estén ocupados y no haya ninguno con la información asociada al id, no devolverá nada. En esta función también se le da valor a la variable PosSlotLlenar, que es la posición del Slot del que se coge su SlotInfo dentro del listado de Slots.
- **FindSlotBestiario:** Saca el Slot de la lista de SlotBestiario del menú de bestiario en la posición asociada a la variable PosSlotLlenar.
- **FindSlotAcciones:** Saca el Slot de la lista de SlotAcciones del menú de listado de acciones en la posición asociada a la variable PosSlotLlenar.
- **AddEntradaBestiario:** Añade la entrada al bestiario asociada al 'id' que se le pasa como parámetro a esta función a uno de los SlotBestiario del menú desplegable del bestiario. Primero se comprueba que el 'id' corresponde con alguna de las entradas al bestiario existentes en la instancia del Bestiario de la imagen 62. Luego, se ha de asegurar que hay espacio en el menú y que no hay ningún otro Slot con la misma entrada al bestiario que la que se quiere colocar (empleando la función FindSlotOcupadoBestiario). En caso de que la ningún otro Slot tiene esa información, se coge el Slot escogido para asociarlo a esta información (con la función FindSlotBestiario) y se le asocia la información de la entrada al bestiario igualando su 'id' al del elemento Entrada_Bestia que se desea asociar y seguidamente empleando la función UpdateUI de la clase Slot que puede ser vista en la imagen 70. Por último, muestra un mensaje mediante la variable Advertencia indicando que el bestiario ha sido actualizado. En caso de que ya haya un SlotBestiario con la misma información que se quería añadir, el mensaje de la variable Advertencia indicará que ya se tenía esa información en el bestiario.
- **AddEntradaAccion:** Añade la acción asociada al 'id' que se le pasa como parámetro a esta función a uno de los SlotAcciones del menú desplegable del bestiario. Primero se comprueba que el 'id' corresponde con alguna de las acciones existentes en la instancia del ListadoAcciones de la imagen 63. Luego, se ha de asegurar que hay espacio en el menú y que no hay ningún otro Slot con la misma acción que la que se quiere colocar (empleando la función FindSlotOcupadoAccion). En caso de que la ningún otro Slot tiene esa información, se coge el Slot escogido para asociarlo a esta información (con la función FindSlotBestiario) y se le asocia la información de la acción igualando su 'id' al del elemento Accion que se desea asociar seguidamente empleando la función UpdateUI de la clase Slot que puede ser vista en la imagen 70.

4.3.4. Implementación de los menús de la aplicación

En el capítulo 4.2.3 se ha hablado sobre el diseño de la interfaz de la aplicación. Este capítulo se apoyará en lo visto en el diseño para explicar cómo se ha realizado la implementación de este.

Para hablar de la implementación de los menús, cabe mencionar que se han empleado distintos elementos de interfaz de usuario (botones, paneles, textos, etc.) conocidos también como elementos UI. Para poder manipular el comportamiento de dichos elementos mediante los scripts, se ha hecho uso de la librería 'UnityEngine.UI'.

Los dos elementos de la escena de Unity que se encargan del funcionamiento del menú son: el Controller_Game que se encargará de abrir y cerrar los menús que correspondan, y el Canvas que almacena todos los elementos UI que componen los distintos menús.

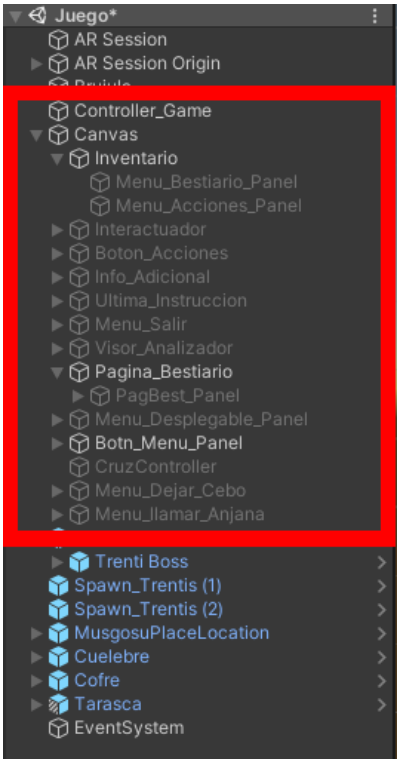


Ilustración 72: Controller_Game y Canvas en la escena Juego

Antes de comenzar a detallar el funcionamiento, es necesario conocer los elementos del menú que van a ser manipulados. Todos estos elementos que componen los distintos menús se encuentran dentro del Canvas y los más relevantes son los siguientes:

- **Menu_Bestiario_Panel**: este elemento se trata del menú desplegable del bestiario. Se activará cuando el usuario pulse el botón "Bestiario" de Menu_Desplegable_Panel. (Este menú puede verse ilustrado en la imagen 14).
- **Menu_Acciones_Panel**: este elemento se trata del menú desplegable del listado de las acciones. Se activará cuando el usuario pulse el botón "Acciones"

de Menu_Desplegable_Panel. (Este menú puede verse ilustrado en la imagen 13).

- **Interactuador**: este elemento hace referencia a todos los elementos del menú empleados durante el uso de la acción de interactuar. Este objeto se activará cuando se pulse el botón de Menu_Acciones_Panel asociado a la acción de interactuar. (Este menú puede verse ilustrado en la imagen 18).
- **Boton_Acciones**: este elemento es un botón que aparece cuando se activan los menús de la acción interactuar, poner cebo o pedir ayuda. Su funcionalidad varía dependiendo de la acción que se tenga activada. (Este botón puede verse ilustrado en la esquina inferior derecha de las imágenes 15, 18 y 24).
- **Info_Adicional**: este elemento es un cartel con un mensaje que aparece exclusivamente cuando el jugador está empleando la acción de usar la brújula y se encuentra cerca del objetivo. (Este cartel puede verse ilustrado en la imagen 23).
- **Ultima_Instruccion**: este elemento es un cartel que solo aparece cuando terminas la última misión, indicándote que debes emplear el menú de ayuda para finalizar el último paso de la experiencia. (Este cartel puede verse en la imagen 30).
- **Menu_Salir**: este elemento es el menú que aparece al pulsar el botón 'Salir' de Se activará cuando el usuario pulse el botón "Bestiario" de Menu_Desplegable_Panel. El menú sirve para confirmar si el usuario quiere abandonar la experiencia de realidad aumentada. (Este menú puede verse ilustrado en la imagen 17).
- **Visor_Analizador**: este elemento hace referencia a todos los elementos del menú empleados durante el uso de la acción de analizar. Este objeto se activará cuando se pulse el botón de Menu_Acciones_Panel asociado a la acción de analizar. (Este menú puede verse ilustrado en la imagen 21).
- **Pagina_Bestiario**: este elemento hace referencia a todos los elementos UI que componen la entrada del bestiario que se abre al pulsar sobre los botones desbloqueados que hay en el menú desplegable del bestiario, variando los componentes de texto e imagen de la entrada al bestiario en función de la información de la bestia asociada al botón pulsado. (Pueden verse las distintas variaciones de la página del bestiario en las imágenes 31, 32, 33, 34 y 35).
- **Menu_Desplegable_Panel:** este elemento es el menú desplegable que contiene los botones "Acciones" (para activar Menu_Acciones_Panel), "Bestiario" (para activar Menu_Bestiario_Panel), "Ayuda" (para activar Menu_llamar_Anjana) y "Salir" (para activar Menu_Salir). (Este menú puede verse ilustrado en la imagen 11).
- **NewHelp:** se trata de un elemento UI dentro del botón "Ayuda" de Menu_Desplegable_Panel. Este elemento es una imagen que aparecerá sobre el botón "Ayuda" cuando haya un nuevo consejo para el jugador. (Puede verse el icono de NewHelp sobre el botón "Ayuda" en la imagen 12).

Ilustración 73: El elemento NewHelp en Menu_Desplegable_Panel

- **Boton_Menu_Panel:** este elemento es el panel en el que se encuentra el botón que abre y cierra el elemento Menu_Desplegable_Panel. (Este menú puede verse ilustrado en la imagen 10).
- **CruzController:** este elemento controla el funcionamiento de la acción de emplear la cruz de Santa Marta. No aporta ningún elemento visual, solo se centra en la funcionalidad de la acción, por lo que este elemento será visto en profundidad en el capítulo 4.3.5. Este objeto se activará cuando se pulse el botón de Menu_Acciones_Panel asociado a la acción de emplear la cruz de Santa Marta.
- **Menu_Dejar_Cebo:** este objeto recopila los elementos del menú empleados durante el uso de la acción de dejar cebo. Este objeto se activará cuando se pulse el botón de Menu_Acciones_Panel asociado a la acción de dejar cebo. (Este menú puede verse ilustrado en la imagen 24).
- **Menu_llamar_Anjana:** este objeto recopila los elementos del menú empleados durante el uso de la acción de pedir ayuda tras pulsar el botón “Ayuda” de Menu_Desplegable_Panel. (Este menú puede verse ilustrado en la imagen 15).

También cabe mencionar el objeto Brújula que por motivos de funcionamiento no se encuentra en el Canvas, pero es importante tenerlo en cuenta.

- **Brujula:** se trata del elemento que controla el funcionamiento de la acción de usar la brújula. Su funcionamiento será visto en detalle en el capítulo 4.3.5. Este objeto se activará cuando se pulse el botón de Menu_Acciones_Panel asociado a la acción de usar la brújula.

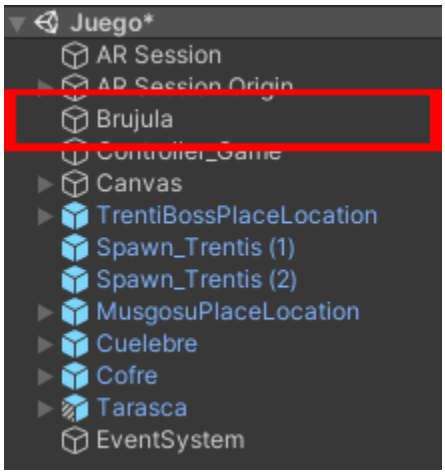


Ilustración 74: Objeto Brujula en la escena Juego

Conocidos ya todos los elementos que componen los distintos menús de la experiencia de realidad aumentada, es momento de ver como las clases que se encargan del funcionamiento del menú manejan estos elementos.

En la imagen 75 se puede apreciar como dentro del objeto Controller_Game, este tiene cuatro elementos que hacen referencia a cuatro clases vistas en la imagen 37, tres de esas clases son útiles para la implementación de los menús: **Menu_Juego**, **Acciones** y **Boton_Acciones**. La clase Misiones_Controller será vista en detalle en el capítulo 4.3.7.

Ilustración 75: Elementos del Controller_Game

La primera de las clases de Controller_Game que merece la pena verse es Boton_Acciones (ver anexo 1 apartado 9), que se encarga de cambiar la imagen y la funcionalidad del botón Boton_Acciones dependiendo de la acción que se tenga activada en ese momento. En este caso el botón solo será funcional si se está empleando las acciones de interactuar o dejar cebo, o en el caso de que el usuario quiera usar el menú de pedir ayuda.

La clase emplea la variable de tipo integer llamada 'accion' que indica cuál de las acciones que requiere Boton_Acciones está activada en ese momento. Por una parte, cuando el usuario pulsa el botón de Menu_Acciones_Panel asociado a la acción "Interactuar" o "Dejar Cebo", o el botón de Ayuda de Menu_Desplegable_Panel, se llamará a la función setAccionSprite y se le pasará como parámetro el número asociado a la acción activada. La función setAccionSprite se encargará de actualizar el valor de la variable 'accion' y de cambiar la imagen del botón, tal y como se ve en la ilustración 76.

```
public void setAccionSprite(int a)
{
    accion = a;
    switch(accion)
    {
        case 1:
            imagen_boton_accion.sprite = imagen_interactuar;
            break;
        case 3:
            imagen_boton_accion.sprite = imagen_cebo;
            break;
        case 6:
            imagen_boton_accion.sprite = imagen_ayuda;
            break;
    }
}
```

Ilustración 76: Función setAccionSprite de la clase Boton_Acciones

Por otro lado, la función AccionBoton de la imagen 77 se encarga de cambiar la funcionalidad del botón en función del valor de la variable 'accion'. Esto se consigue asociando dicha función al componente Button del objeto Boton_Acciones, como se muestra en la imagen 78.

```
public void AccionBoton()
{
    switch(accion)
    {
        case 1:
            interactuador.Interactuar();
            break;
        case 3:
            if(!GameObject.FindGameObjectWithTag("Cebo"))
            {
                objectSpawner.DejarElemento();
            }
            break;
        case 6:
            llamar_anjana.DejarElemento();
            break;
    }
}
```

Ilustración 77: Función AccionBoton de la clase Boton_Acciones

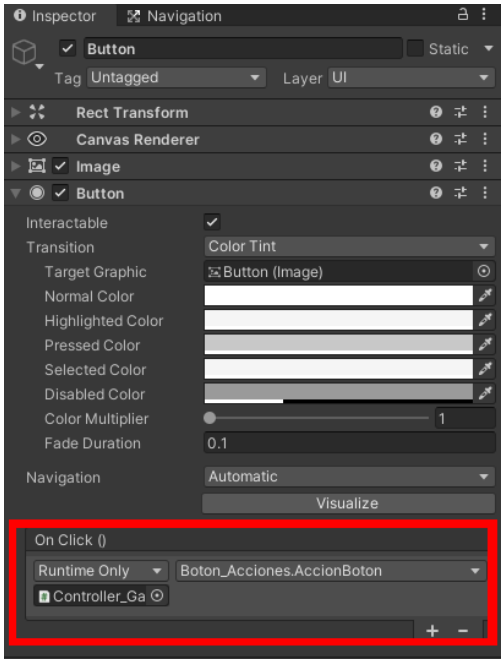


Ilustración 78: Función AccionBotno de la clase Boton_Acciones asociada a un botón

Cabe destacar en la función AccionBoton de la imagen 77, que la función DejarCebo que se le asocia al botón solo podrá realizarse en caso de que no haya ningún objeto en la escena con el tag “Cebo”. Esto se hace para evitar que el usuario pueda añadir más de un cebo y que no haya varios cebos en la misma escena.

Pasando ahora a hablar de la clase Menu_Juego, esta clase se encarga activar y desactivar los distintos elementos que conforman los menús de la escena en función de qué botones se pulsen del menú desplegable Menu_Desplegable_Panel. Los botones que se pueden encontrar en este menú son: el botón “Acciones”, el botón “Bestiario”, el botón “Ayuda”, el botón “Salir” y el botón “Menú”. La clase puede verse en su totalidad en el anexo 1 apartado 33, pero a continuación se mostrarán los distintos métodos que tiene y el funcionamiento de cada uno.

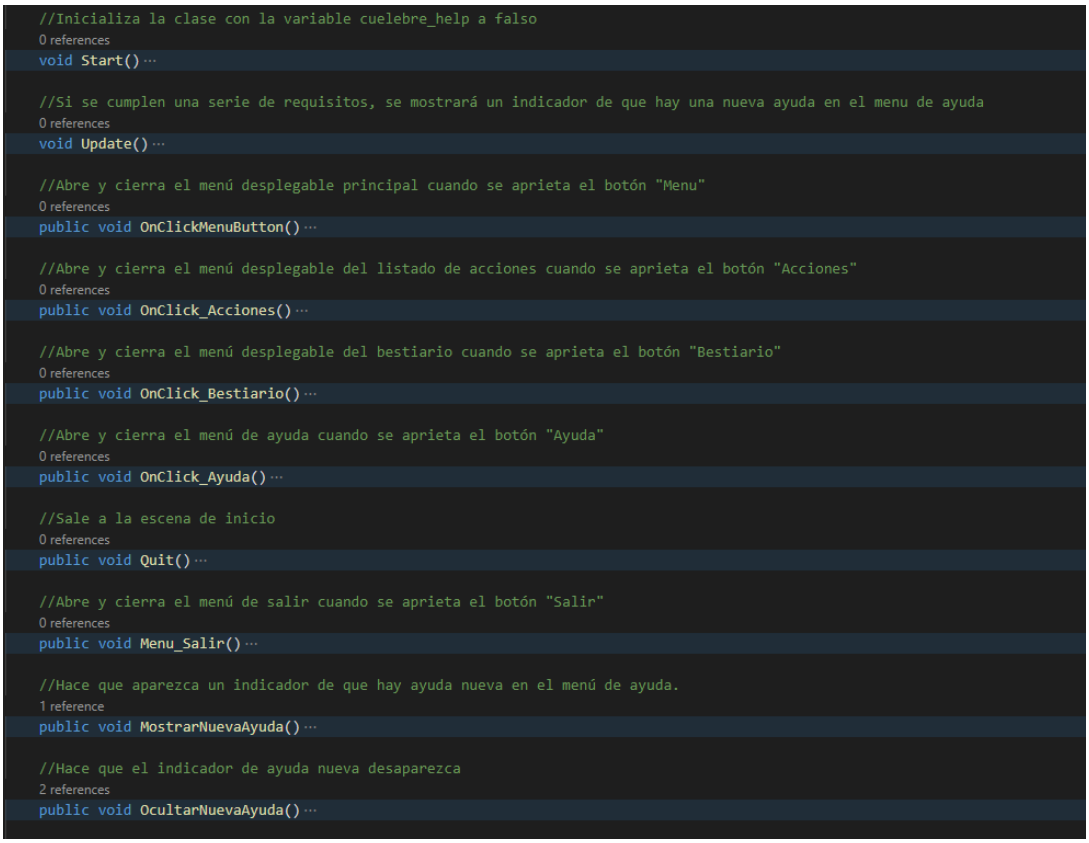


Ilustración 79: Métodos de la clase Menu_Juego

- **Start**: esta función se ejecuta al iniciarse la clase Menu_Juego. Lo único relevante que hay es que inicia a false el valor de la variable “cuelebre_help”. Esta variable sirve para poder activar el componente NewHelp en un momento dado que de otra forma no se podría activar.
- **Update**: esta función se ejecuta cada frame que está activa la clase Menu_Juego. Se encarga de en caso de que se cumplan las condiciones de que el usuario haya analizado a la bestia “Cuelebre”, el PlayerPref “num_mision” tenga el valor 5 y que la variable “cuelebre_help” siga en false, se active el componente NewHelp para avisar al usuario de que hay una ayuda nueva.
- **OnClickMenuButton**: esta función está asociada al botón “Menu” de Boton_Menu_Panel. Cuando el botón es pulsado se activará el menú Menu_Desplegable_Panel en caso de que no esté activado o lo desactivará en caso de que sí lo esté. La función desactivará los demás menús que estén activados, a excepción de los menús de las acciones o de ayuda una vez sea ejecutada.
- **OnClick_Acciones**: esta función está asociada al botón “Acciones” de Menu_Desplegable_Panel. Cuando el botón es pulsado se activará el menú desplegable Menu_Acciones_Panel, en caso de estar ya abierto, dicho menú se cerrará. Todos los menús a excepción del menú desplegable Menu_Desplegable_Panel y los menús de acciones o ayuda se cerrará una vez se ejecute el método.

- **OnClick_Bestiario**: esta función está asociada al botón “Bestiario” de Menu_Desplegable_Panel. Cuando el botón es pulsado se activará el menú desplegable Menu_Bestiario_Panel, en caso de estar ya abierto, dicho menú se cerrará. Todos los menús a excepción del menú desplegable Menu_Desplegable_Panel y los menús de acciones o ayuda se cerrará una vez se ejecute el método.
- **OnClick_Ayuda**: esta función está asociada al botón “Ayuda” del menú desplegable Menu_Desplegable_Panel. Cuando se pulsa el botón cerrará todos los menús abiertos a excepción del propio botón “Menú” de Boton_Menu_Panel y abrirá el menú de ayuda Menu_llamar_Anjana y el botón Boton_Acciones con la imagen y la funcionalidad del menú de ayuda. En el caso de que el menú de ayuda ya esté abierto previamente, simplemente se cerrarán todos los menús abiertos a excepción de Boton_Menu_Panel. Independientemente de si el menú ayuda está abierto o cerrado, esta función llamará al método OcultarNuevaAyuda.
- **Quit**: esta función está asociada al botón que sirve para confirmar que el usuario quiere salir de la experiencia RA en el menú Menu_Salir.
- **Menu_Salir**: esta función está asociada al botón “Salir” de Menu_Desplegable_Panel. Abre el menú Menu_Salir en caso de que no esté abierto, en caso contrario lo cierra. Al ejecutarse esta función cierra todos los menús abiertos a excepción de Boton_Menu_Panel.
- **MostrarNuevaAyuda**: esta función es empleada por la clase Misiones_Controller (ver el capítulo 4.3.6) cada vez que el usuario supera una misión. La función se encarga de activar el elemento NewHelp para que aparezca sobre el botón de “Ayuda” el indicador de que hay un nuevo mensaje en el menú de ayuda.
- **OcultarNuevaAyuda**: esta función es llamada cuando se ejecuta la función OnClick_Ayuda. La función desactiva el elemento NewHelp para que no aparezca en la pantalla. En caso de que se cumplan las condiciones de que el usuario haya analizado a la bestia “Cuelebre”, el PlayerPref “num_mision” tenga el valor 5 y que la variable “cuelebre_help” siga en false, esta función convertirá la variable “cuelebre_help” a true. Permitiendo de esa forma que el elemento NewHelp que debía salir cuando se cumplieran esas condiciones por la función Update desaparezca.

Habiendo visto la clase Menu_Juego, se ha abarcado todo el comportamiento del menú concerniente a los botones del menú Menu_Desplegable_Panel y el de Boton_Menu_Panel. Para terminar con la implementación del comportamiento de los menús falta ver como la clase Acciones (ver anexo 1 apartado 5) controla el comportamiento de los menús de las distintas acciones que puede hacer el usuario sin contar la acción de “Ayuda”, estas acciones son: “Interactuar”, “Analizar”, “Brújula”, “Poner_Cebo” y “Cruz Santa Marta”.

Se ha hablado en el capítulo 4.3.3 de que cada uno de los botones activos dentro del menú desplegable del listado de acciones Menu_Acciones_Panel, lleva asociada la

información de una acción y por medio de la clase Slot era capaz de asociar al evento OnClik de dicho botón a uno de los métodos de la clase Acciones que permitía activar el menú de la acción que tenía asociada empleando la función ActiveActions (imagen 67). Esto permite que al apretar cada botón activado de Menu_Acciones_Panel, se active el menú de funcionamiento de la acción a la que está asociado.

A continuación, se muestran los métodos que son asociados a cada uno de los botones del listado de acciones y su funcionamiento:

```
//Activa el menú de la acción Poner Cebo
1 reference
public void Onclick_PonerCebo()···

//Activa el menú de la acción Análisis
1 reference
public void Onclick_Analizar()···

//Activa el menú de la acción Interactuar
1 reference
public void Onclick_Interactuar()···

//Activa el menú de la acción Brújula
1 reference
public void Onclick_Brujula()···

//Activa el menú de la acción Cruz Santa Marta
1 reference
public void Onclick_Cruz()···
```

Ilustración 80: Métodos de la clase Acciones

- **Onclick_PonerCebo**: se encarga de activar el menú de la acción “Dejar Cebo”. Cuando la función es ejecutada se ocultan todos los elementos del menú, dejando solo el elemento Boton_Menu_Panel y los elementos del menú de “Dejar Cebo” que serían Menu_Dejar_Cebo y el botón Boton_Acciones con la imagen y la funcionalidad asociadas a la acción “Dejar Cebo”.
- **Onclick_Analizar**: se encarga de activar el menú de la acción “Analizar”. Cuando la función es ejecutada se ocultan todos los elementos del menú, dejando solo el elemento Boton_Menu_Panel y los elementos del menú de “Analizar” que sería el elemento Visor_Analizador.
- **Onclick_Interactuar**: se encarga de activar el menú de la acción “Interactuar”. Cuando la función es ejecutada se ocultan todos los elementos del menú, dejando solo el elemento Boton_Menu_Panel y los elementos del menú de “Interactuar” que serían Interactuador y el botón Boton_Acciones con la imagen y la funcionalidad asociadas a la acción “Interactuar”.
- **Onclick_Brujula**: se encarga de activar el menú de la acción “Brújula”. Cuando la función es ejecutada se ocultan todos los elementos del menú, dejando solo el elemento Boton_Menu_Panel y los elementos del menú de “Brújula”. En este caso peculiar, la función busca unos elementos que son hijos del elemento Brujula, estos son llamados Flecha_Direccion y los encuentra mediante su etiqueta “Flecha_Brujula”. El funcionamiento y la creación de estos elementos serán abordados en el capítulo 4.3.5, de momento lo que se ha de saber es que estos elementos son buscados por esta función y activados para que el usuario

los pueda ver por pantalla. Cuando se emplea cualquiera de las otras funciones de la clase Acciones, estos elementos son desactivados.

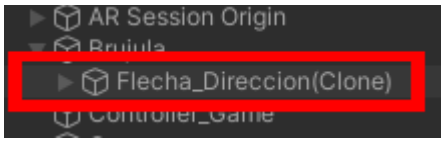


Ilustración 81: Objeto Flecha_Direccion hijo de Brujula

- **Onclick_Cruz**: se encarga de activar el menú de la acción "Cruz Santa Marta". Cuando la función es ejecutada se ocultan todos los elementos del menú, dejando solo el elemento Boton_Menu_Panel y los elementos del menú de "Cruz Santa Marta". En este caso el elemento que se activa es un elemento hijo de AR Camera llamado Plane, este elemento se trata de un plano en el que aparecerá el objeto CruzSantaMarta. El cómo se emplea el elemento Plane para invocar a ese objeto se detallará en el capítulo 4.3.5.

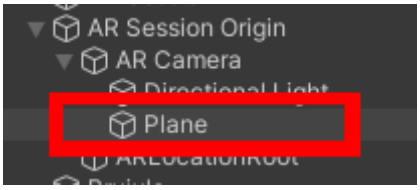


Ilustración 82: Objeto Plane hijo de AR Camera

4.3.5. Implementación de las acciones

En el diagrama de clases conceptual de la ilustración 4 se mostraba una clase padre llamada Acción la cual era padre de las clases Interactuar, Analizar, Crear Cebo, FlechaBrújula y Cruz Santa Marta. Cada una de estas clases representaban las distintas acciones que el usuario podría hacer y todas ellas heredarían atributos y métodos comunes de la clase Acción.

Sin embargo, la implementación del funcionamiento de las acciones ha resultado más complicada de lo que parecía en un principio, ya que el funcionamiento de cada acción es bastante distinto del de las demás. Es por eso por lo que se ha optado por crear distintas clases que se encarguen del funcionamiento de las distintas acciones que puede hacer el usuario sin que estén relacionadas de ningún modo entre ellas, como se aprecia en la ilustración 37.

En este apartado se repasarán las distintas acciones mencionadas en el capítulo 3.1 que el usuario puede realizar y se explicará en cada una de ellas cómo funcionan y qué clases permiten que las distintas acciones se lleven a cabo.

Ayuda

Pese a que esta acción no se encuentra en la instancia de ListadoAcciones en donde se almacenan las acciones que aparecerán en el menú desplegable de las acciones, se incluirá "Ayuda" en este apartado dado que es una acción que permite al usuario interactuar con el entorno.

Lo que permite la acción de “Ayuda” al usuario es invocar en algún punto de su alrededor al personaje “Anjana” para que le de algún consejo sobre qué debe hacer a continuación.

Los objetos que se encargan de manejar la funcionalidad de esta acción se encuentran dentro de los elementos que componen el menú de ayuda. Estos elementos pueden ser vistos en la ilustración 83 y se encuentran dentro del objeto Menu_llamar_Anjana, el cual tiene como hijos los objetos ObjectSpawner que se encarga de crear el objeto de la “Anjana”, y Indicator_Place que se encarga de indicar en qué lugar del entorno va a ser invocada la “Anjana” en función del posicionamiento y rotación del elemento Plane.

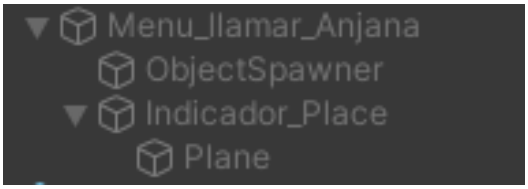


Ilustración 83: Elementos de Menu_llamar_Anjana

Los componentes Menu_llamar_Anjana, ObjectSpawner e Indicador_Place tienen asociados respectivamente las siguientes clases fundamentales para el funcionamiento de la acción “Ayuda”: **Ayuda**, **ObjectSpawner** y **PlacementIndicator**.

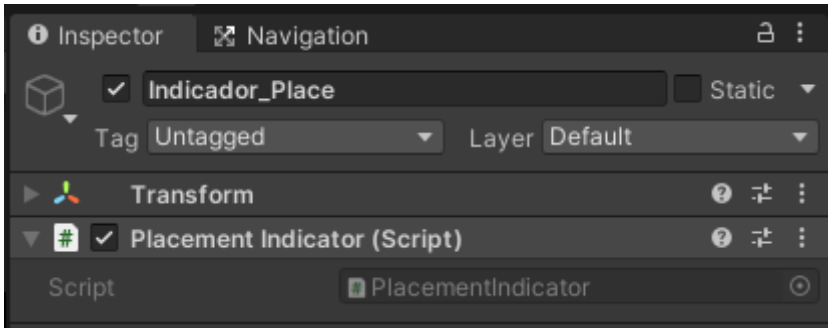


Ilustración 84: Componentes del objeto Indicador_Place

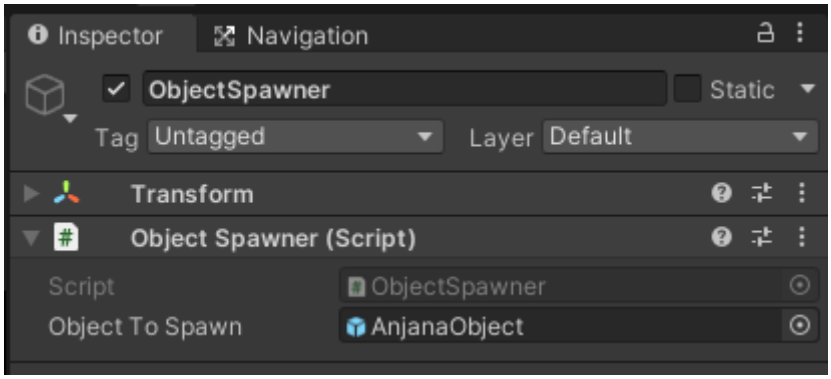


Ilustración 85: Componentes del objeto ObjectSpawner

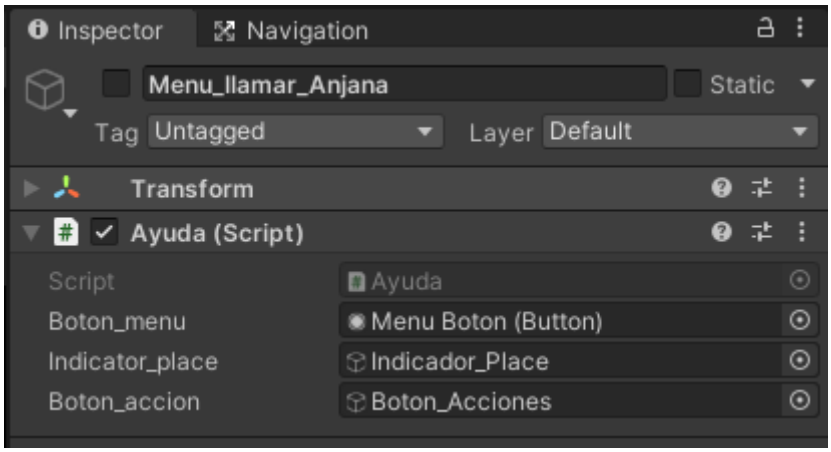


Ilustración 86: Componentes del objeto Menu_llamar_Anjana

La clase PlacementIndicator se encarga de posicionar en una superficie plana del mundo real que haya alrededor del usuario el elemento Plane que es una imagen 2D que sirve para indicarle al usuario donde se posicionará el objeto que se va a invocar. Tal y como se puede ver en el ejemplo de la imagen 15. La clase PlacementIndicator puede ser vista al completo en el anexo 1 apartado 7, pero lo más relevante de la clase es el atributo "rayManager" del tipo ARRaycastManager. Este elemento se encarga que se pueda emplear la función Raycast (función que lanza un rayo desde el centro de la cámara y puede colisionar con los objetos que tengan un elemento Collider, pudiendo identificar con que ha colisionado el rayo) con funcionalidades de AR Foundation, permitiendo que la función Raycast pueda detectar elementos del mundo real como superficies planas.

En la función Update de la imagen 88 se puede apreciar como la lista de ARRaycastHit se va llenando conforme la función Raycast del atributo "rayManager" va detectando planos. En el momento en el que detecta uno, todo el objeto Indicador_Place se posiciona en la posición en donde ha colisionado el rayo del Raycast y activa el elemento Plane (nombrado plano en esta función) para que se vea por pantalla.

```
void Update ()
{
    List<ARRaycastHit> hits = new List<ARRaycastHit>();
    rayManager.Raycast(new Vector2(Screen.width / 2, Screen.height / 2), hits, TrackableType.Planes);

    if(hits.Count > 0)
    {
        transform.position = hits[0].pose.position;
        transform.rotation = hits[0].pose.rotation;

        if(!plano.activeInHierarchy)
        {
            plano.SetActive(true);
        }
    }
}
```

Ilustración 87: Función Update de la clase PlacementIndicator

Por otra parte, se tiene la clase ObjectSpawner (ver anexo 1 apartado 8) que se encarga de invocar el elemento que tenga asignado. En el caso de la acción "Ayuda"

será la “Anjana”, pero más adelante se verá como esta función también es usada para invocar al objeto “Cebo”.

Para ello también emplea como en la clase PlacementIndicator un elemento del tipo ARRAycastManager para poder detectar superficies planas. También cabe destacar el empleo de la clase ARAnchorManager para poder colocar elementos “Anchor” en la escena. Previamente se ha mencionado que los “Anchor” son puntos que describen una posición y rotación fijas en el mundo real. De tal forma que, si se le asocia a un “Anchor” un objeto digital, este será visto en la posición y rotación del “Anchor” dentro del mundo real.

Es por ello por lo que la función CreateAnchor de la clase ObjectSpawner, recoge la colisión de un Raycast que haya colisionado con un plano y crea un “Anchor” al que le asocia el objeto que se va a crear y lo coloca en la posición de la colisión.

```
ARAnchor CreateAnchor(in ARRaycastHit hit)
{
    ARAnchor anchor = null;

    if(hit.trackable is ARPlane plane){
        var planeManager = FindObjectOfType<ARPlaneManager>();
        if (planeManager)
        {
            var oldPrefab = m_AnchorManager.anchorPrefab;
            m_AnchorManager.anchorPrefab = objectToSpawn;
            anchor = m_AnchorManager.AttachAnchor(plane, hit.pose);
            m_AnchorManager.anchorPrefab = oldPrefab;
            return anchor;
        }
    }

    return anchor;
}
```

Ilustración 88: Función CreateAnchor de ObjectSpawner

La función CreateAnchor es ejecutada cuando Raycast detecta una superficie plana en la función DejarElemento, la cual como se ha visto en la imagen 77, es la función asociada al botón de acciones cuando se activa la acción “Ayuda”.

La clase también tiene la función RemoveAllAnchors que elimina los “Anchor” pertenecientes a la lista “m_Anchors” y los objetos digitales asociados a estos.

Por último, la clase Ayuda (ver anexo 1 apartado 6) se encarga de que cuando haya una “Anjana” ya invocada en la escena, desactivar los botones de acción y el del menú para evitar que se pueda realizar otra acción mientras la “Anjana” habla o que se puedan invocar otras “Anjanas”, y de desactivar el elemento Indicador_Place para que no aparezca la imagen 2D de Plane mientras está la “Anjana” invocada. Esto puede verse en la imagen 16 en la que los botones de menú y de acción se encuentran desactivados.

```
public class Ayuda : MonoBehaviour
{
    2 references
    public Button boton_menu;
    2 references
    public GameObject Indicator_place;
    2 references
    public GameObject boton_accion;

    0 references
    void Update()
    {
        var anjana = FindObjectOfType<Anjana>();
        if(anjana != null)
        {
            boton_menu.interactable = false;
            Indicator_place.SetActive(false);
            boton_accion.SetActive(false);
        }

        else{
            boton_menu.interactable = true;
            Indicator_place.SetActive(true);
            boton_accion.SetActive(true);
        }
    }
}
```

Ilustración 89: Clase Ayuda

Interactuar

Para la funcionalidad de la acción "Interactuar" simplemente es necesaria la clase Interactuador (ver anexo 1 apartado 19) que se encuentra en el objeto Interactuador del Canvas.

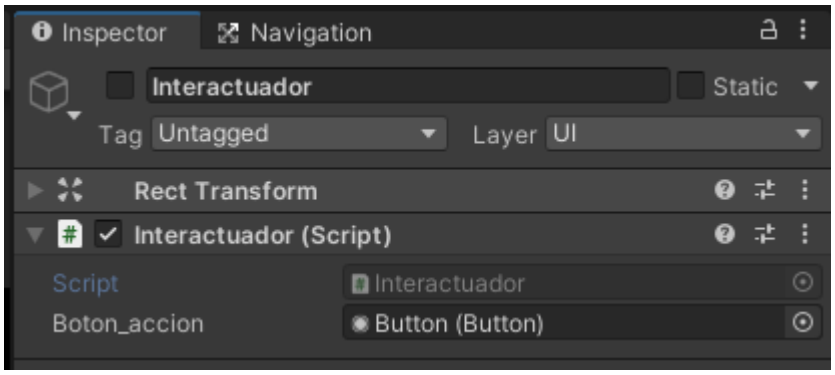


Ilustración 90: Componentes del elemento Interactuador

Al igual que con las acciones de "Ayuda" y "Dejar Cebo" en la imagen 77, parte de la funcionalidad de la acción recae en la pulsación del botón de acciones. La funcionalidad que se le asocia a ese botón es la función Interactuar de la clase Interactuador.

```
public void Interactuar()
{
    RaycastHit hit;
    if(Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, distancia_interactuador_larga))
    {
        if(hit.collider.tag == "Trenti_Boss")
        {
            Debug.Log("Interctuar con Trenti_Boss");
            hit.collider.GetComponent<Trenti_Boss>().Dialogo();
        }
        if(hit.collider.tag == "Trenti")
        {
            Debug.Log("Interctuar con Trenti");
            hit.collider.GetComponent<Trenti>().Dialogo();
        }
        if(hit.collider.tag == "Musgosu")
        {
            Debug.Log("Interctuar con Musgosu");
            hit.collider.GetComponent<Musgosu>().Dialogo();
        }
        if(hit.collider.tag == "Cuelebre")
        {
            Debug.Log("Interactuar con Cuelebre");
            hit.collider.GetComponent<Cuelebre>().Dialogo();
        }
        if(hit.collider.tag == "Tarasca")
        {
            Debug.Log("Interactuar con Tarasca");
            hit.collider.GetComponent<Tarasca>().Dialogo();
        }
        if(hit.collider.tag == "Cofre")
        {
            hit.collider.GetComponent<Cofre_Cuelebre>().Abrir();
        }
    }
}
```

Ilustración 91: Función Interactuar de la clase Interactuado

En la imagen anterior se muestra como en función de con qué tipo de bestia u objeto colisiona el rayo del Physics.Raycast, se ejecutará una función de dicha bestia u objeto (el cofre). Esas funciones asociadas a las bestias serán vistas en el capítulo 4.3.7, pero cabe entender que esas funciones se encargan de sacar por pantalla un texto que representa un dialogo de la bestia con la que se está interactuando o una descripción de lo que se está haciendo con el objeto con el que se interactúa. Este texto variará en función de otros factores externos a la acción “Interactuar”.

La dirección del rayo lanzado por el Raycast se representa mediante la imagen de una pequeña retícula en el centro de la pantalla que le indique al usuario hacia donde debe de apuntar para interactuar con los objetos digitales, tal y como se ve en la imagen 18.

La clase del Interactuador también tiene en cuenta que no todos los objetos con los que va a interactuar son igual de grandes, por lo que la distancia a la que podrá interactuar con ellos deberá ser distinta. No es lo mismo interactuar con un objeto igual de grande que un humano que con un objeto igual de grande que un camión. Es por ello que la clase Interactuador emplea tres variables del tipo float que indican tres distancias a las que puede interactuar el usuario con los objetos digitales: “distancia_interactuador_larga”, “distancia_interactuador_media” y “distancia_interactuador_corta”.

La función FixedUpdate se encarga de activar o desactivar el botón al que está asociado la función Interactuar para que solo pueda usarse cuando el usuario esté apuntando a un objeto digital y esté a una distancia apropiada para interactuar con este. El método emplea la función Raycast y lanza tres rayos desde el centro de la cámara. Cada uno de estos rayos tiene una de las tres distancias anteriores asociadas a

la longitud del rayo. Si uno de estos rayos colisiona con un objeto digital asociado a la distancia con la que se puede interactuar con él, el botón de acciones se activará pudiendo usarse la función Interactuar, en caso contrario el botón permanecerá desactivado.

```
private void FixedUpdate()
{
    RaycastHit hitlargo;
    RaycastHit hitmedio;
    RaycastHit hitcorto;
    if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hitlargo, distancia_interactuador_larga))
    {
        //Lanzamiento del Raycast de larga distancia
        if(hitlargo.collider.tag == "Cuelebre" || hitlargo.collider.tag == "Tarasca")
        {
            boton_accion.interactable = true;
        }

        else if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hitmedio, distancia_interactuador_media))
        {
            //Lanzamiento del Raycast de media distancia
            if(hitmedio.collider.tag == "Cofre" || hitmedio.collider.tag == "Trenti")
            {
                boton_accion.interactable = true;
            }

            else if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hitcorto, distancia_interactuador_corta))
            {
                //Lanzamiento del Raycast de corta distancia
                if(hitcorto.collider.tag == "Trenti_Boss" || hitcorto.collider.tag == "Musgosu")
                {
                    boton_accion.interactable = true;
                }
            }

            else
            {
                boton_accion.interactable = false;
            }
        }

        else
        {
            boton_accion.interactable = false;
        }
    }

    else
    {
        boton_accion.interactable = false;
    }
}
```

Ilustración 92: Función FixedUpdate de la clase Interactuador

Analizar

La acción de analizar se encarga de recopilar la información de una bestia tras observarla y cargar una barra de análisis al completo. Para que el funcionamiento de la acción se realice correctamente, son necesarios dos elementos que pueden ser vistos en la imagen 20: la imagen de una retícula con la que el usuario podrá apuntar a las bestias y una barra de análisis que se encargue que indique cuanto falta para que se complete el análisis de la bestia a la que se está observando.

Estos dos elementos se encuentran representados respectivamente en la escena de Unity con los elementos hijos del elemento Visor_Analizador: Imagen_Visor y Barra_Analisis.

Ilustración 93: Elementos de Visor_Analizador

Para que la acción funciones correctamente, se han necesitado dos clases: **Analizador** (ver anexo 1 apartado 20) para detectar a las bestias y añadir nuevas entradas al bestiario, y **BarraAnalisis** (ver anexo 1 apartado 21) para controlar el funcionamiento de la barra de análisis.

Por un lado, la clase BarraAnalisis tiene varias funciones y atributos para controlar cómo se va llenando y vaciando. Las variables más importantes para tener en cuenta son las siguientes:

- **analizando**: se trata de una variable de tipo booleano que indica si se está analizando a alguna bestia en este momento.
- **analizado**: se trata de una variable de tipo booleano que indica si se acaba de terminar de analizar a una bestia.

A partir del valor de estas dos variables, controlar el comportamiento de la barra de análisis resulta muy sencillo. Para ello se emplean las siguientes funciones.

- **Iniciar**: inicia la barra de análisis a un estado en la que esta se encuentra vacía, con las variables “analizando” y “analizado” en false. Esta función se ejecuta al pulsar sobre el botón del listado de acciones asociado a la acción “Analizar”.
- **Analizar**: cambia el valor de la variable “analizando” a true. Este método es llamado por la clase Analizador para que comience a llenarse la barra.
- **StopAnalizar**: cambia el valor de la variable “analizando” a true. Este método es llamado por la clase Analizador para que cuando no se esté apuntando a ninguna bestia o se haya terminado de analizar una bestia, la barra deje de llenarse.
- **Update**: esta función es llamada cada frame que pasa en la aplicación. El funcionamiento del método es sencillo, si la variable “analizando” tiene el valor true y la variable “analizado” tiene un valor false, significa que la barra se está llenado, por lo que se rellenará hasta sobrepasar un determinado valor, momento en el que el valor de la variable “analizado” pasará a true. En caso de que alguna de las variables tenga un valor distinto al mencionado previamente, significará que la barra no se está llenando y por tanto si se estaba llenando, el valor que hubiese en la barra comenzará a disminuir hasta llegar a 0. Este funcionamiento puede verse en la siguiente imagen.

```
void Update()
{
    if(analizando && !analizado)
    {

        valor_actual = (valor_actual + (velocidad*Time.deltaTime));
        if(valor_actual >= 100)
        {
            valor_actual = 100;
            analizado = true;
        }
        transform.localScale = new Vector3(1,valor_actual/100,1);
    }
    else{
        if(valor_actual <= 0)
        {
            analizado = false;
            valor_actual = 0f;
        }
        else{
            valor_actual = (valor_actual - (vel_descenso*Time.deltaTime));
        }
        transform.localScale = new Vector3(1,valor_actual/100,1);
    }
}
```

Ilustración 94: Función Update de la clase BarraAnalisis

Por otro lado, la clase Analizador solo posee una función, FixedUpdate que se ejecuta cada frame que la clase Analizador permanece activada. Esta función emplea la función Raycast para crear un rayo que colisionará con las distintas bestias que vayan a ser analizadas. Si se detecta una colisión con una de las bestias que haya alrededor, FixedUpdate llama a la función Analizar vista anteriormente para que comience el análisis de la bestia con la que ha colisionado el rayo.

Una vez la variable “analizado” de BarraAnalisis tenga el valor true, significará que el análisis habrá terminado, por lo que se recogerá el “id” de la bestia analizada y se añadirá una entrada al bestiario mediante la función AddEntradaBestiario de la clase Inventario pasando como parámetro el “id” recogido. En caso de que el rayo no haya colisionado con ninguna bestia, se llamará a la función StopAnalizar para que la barra de análisis deje de llenarse en caso de no estar vacía o que se mantenga completamente vacía.

```
private void FixedUpdate()
{
    RaycastHit hit;
    if(Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, distancia_analisis))
    {
        Debug.DrawRay(Camera.main.transform.position, Camera.main.transform.forward*distancia_analisis, Color.red);
        if(hit.collider.tag == "Trenti_Boss" || hit.collider.tag == "Trenti" || hit.collider.tag == "Musgosu" ||
            hit.collider.tag == "Cuelebre" || hit.collider.tag == "Tarasca")
        {
            bar_analisis.Analizar();
            if(bar_analisis.analizado)
            {
                last_id = hit.collider.GetComponent<Bestia>().id;
                menu_bestias.AddEntradaBestiario(last_id);
                switch(last_id)
                {
                    case 1:
                        PlayerPrefs.SetInt("registro_Trenti",1);
                        break;
                    case 2:
                        PlayerPrefs.SetInt("registro_Musgosu",1);
                        break;
                    case 3:
                        PlayerPrefs.SetInt("registro_Cuelebre",1);
                        break;
                    case 4:
                        PlayerPrefs.SetInt("registro_Tarasca",1);
                        break;
                }
            }
        }

        else
        {
            bar_analisis.StopAnalizar();
        }
    }
    else
    {
        bar_analisis.StopAnalizar();
    }
}
```

Ilustración 95: Función FixedUpdate de la clase Analizador

Brújula

La acción “Brújula” permite al usuario ver a su alrededor unas flechas que apuntan hacia el objetivo al que debe dirigirse. Cuando el usuario está próximo a dicho objetivo, la flecha desaparece y aparece un mensaje indicando que se encuentra cerca del objetivo. Estas flechas creadas son objetos prefabricados de Unity con el nombre Flecha_Direccion.

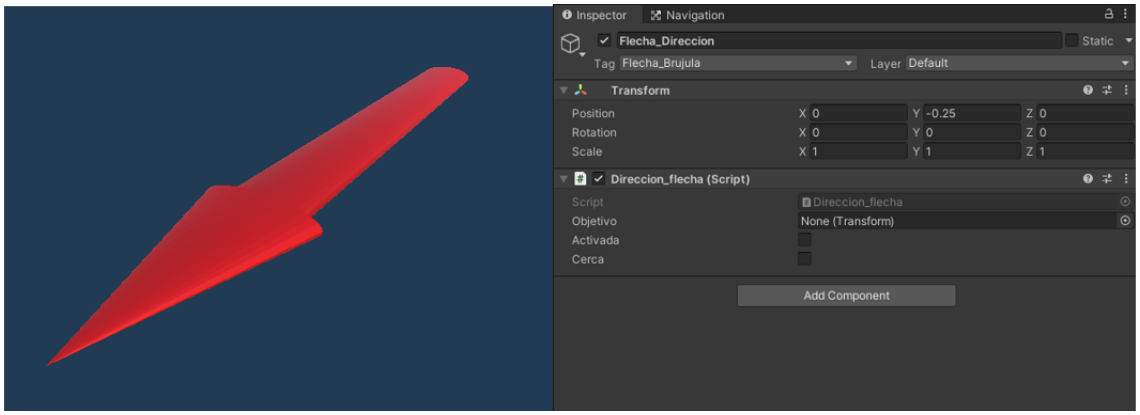


Ilustración 96: Prefab de Flecha_Direccion

Para que la acción “Brújula” funcione, son necesarias dos clases. Por un lado, está la clase **Brujula** (ver anexo 1 apartado 23) que se encarga de activar y desactivar las flechas cuando se pulsa sobre el botón del listado de acciones asociado a la acción “Brújula”, activar y desactivar el mensaje de objetivo cercano y crear o eliminar las flechas existentes.

```
0 references
void Awake() {···

// Update is called once per frame
0 references
void Update()···

7 references
public void addObjetivo(GameObject target)···

1 reference
public void deleteObjetivos()···

1 reference
public void ActivarPanelInfo()···

2 references
public void DesactivarPanelInfo()···
```

Ilustración 97: Métodos de la clase Brújula

Las distintas funcionalidades de esta clase se llevan a cabo por las siguientes funciones:

- **Awake**: inicializa la clase con un indicador de que el cartel de proximidad esta desactivado y que la brújula esta desactivada.
- **Update**: se encarga de comprobar si alguna de las flechas está cerca del objetivo, en caso de estarlo activa el cartel de proximidad mediante la función ActivaPanelInfo. En caso contrario mantiene el cartel desactivado mediante la función DesactivarPanellInfo.

```
void Update()
{
    activarpanel = 0;
    this.transform.position = Camera.main.transform.position;

    if(brujula_activa)
    {
        foreach(GameObject f in GameObject.FindGameObjectsWithTag("Flecha_Brujula"))
        {
            if(f.GetComponent<Direccion_flecha>().cerca)
            {

                activarpanel++;
            }
        }

        if(activarpanel > 0)
        {
            ActivarPanelInfo();
        }

        else
        {
            DesactivarPanelInfo();
        }
    }
    else{
        DesactivarPanelInfo();
    }
}
```

Ilustración 98: Función Update de la clase Brujula

- **addObjetivo**: a este método crea una nueva flecha y se le pasa por parámetro el objeto digital que será el objetivo de dicha flecha. Esta función llama al método SetObjetivo de la clase Direccion_flecha para darle a esa flecha la posición de su objetivo.

```
public void addObjetivo(GameObject target)
{
    GameObject flecha = Instantiate(flecha_dir, new Vector3(0,0,0), Quaternion.identity) as GameObject;
    flecha.transform.parent = this.transform;
    flecha.transform.localPosition = new Vector3(0, -0.25f, 0);
    flecha.GetComponent<Direccion_flecha>().SetObjetivo(target.transform);
}
```

Ilustración 99: Función addObjetivo de la clase Brújula

- **deleteObjetivos**: busca todas las flechas que hay en la escena y las elimina.
- **ActivarPanelInfo**: activa el cartel de proximidad.
- **DesactivarPanelInfo**: desactiva el cartel de proximidad.

Por el otro, está la clase **Direccion_flecha** (ver anexo 1 apartado 24) que se encarga de controlar la dirección a la que apunta la flecha en función de la posición del objetivo, de desactivar o activar el renderizado del objeto cuando está muy cerca del objetivo y de destruirse a sí mismo en caso de que no haya ningún objetivo.

```
public class Direccion_flecha : MonoBehaviour
{
    5 references
    public Transform objetivo;

    2 references
    private float dis_objetivo;

    9 references
    public bool activada;

    4 references
    public bool cerca;

    0 references
    void Awake() {···

    // Update is called once per frame
    0 references
    void Update()···

    1 reference
    public void SetObjetivo(Transform t)···

}
```

Ilustración 100: Clase Direccion_flecha

Esta clase solo está compuesta por tres métodos que realizan todo el funcionamiento de la clase:

- **Awake**: inicializa la variable “activada” y “cerca” a false.
- **Update**: en caso de que la clase Brujula haya indicado que las flechas estén activadas, se averigua si la flecha tiene algún objetivo al que apuntar, en caso de no tenerlo la flecha es eliminada. En caso de tenerlo, se calcula la proximidad entre la flecha y dicho objetivo. Si la distancia es inferior a 50, la variable “cerca” pasa a true y los renders de la flecha se desactivan. En caso

contrario los renders se mantienen activados y se actualiza la rotación de la flecha para que apunte al objetivo. Si la flecha no estaba activada desde un inicio, los renders también serán desactivados.

```
void Update()
{
    if(activada)
    {
        if(!objetivo)
            Destroy(this.gameObject);

        else
        {
            dis_objetivo = Mathf.Sqrt(Mathf.Pow(this.transform.position.x - objetivo.position.x,2) +
                                      Mathf.Pow(this.transform.position.z - objetivo.position.z,2));

            if(dis_objetivo <= 50)
            {
                cerca = true;
                var meshes = this.GetComponentsInChildren<MeshRenderer>();
                foreach(MeshRenderer m in meshes)
                {
                    m.enabled = false;
                }
            }
            else{
                cerca = false;
                var meshes = this.GetComponentsInChildren<MeshRenderer>();
                foreach(MeshRenderer m in meshes)
                {
                    m.enabled = true;
                }
                var rotacion = objetivo.position - transform.position;
                rotacion.y = 0;
                var rotacion_Quaternion = Quaternion.LookRotation(rotacion);
                transform.rotation = Quaternion.Slerp(transform.rotation, rotacion_Quaternion, 1);
            }
        }
    }

    else{
        var meshes = this.GetComponentsInChildren<MeshRenderer>();
        foreach(MeshRenderer m in meshes)
        {
            m.enabled = false;
        }
    }

}
```

Ilustración 101: Función Update de la clase Direccion_flecha

- **SetObjetivo**: se le pasa a la variable “objetivo” el valor tipo Transfor que se le pasó como pará metro a esta función. Esta función es ejecutas después de que se haya creado el objeto flecha en la función addObjetivo de la clase Brujula.

Dejar Cebo

El funcionamiento de la acción “Dejar Cebo” es idéntico al de la acción “Ayuda” pero con algunas variaciones.

Al igual que en la acción “Ayuda”, el menú de la acción “Dejar Cebo” está compuesto por un elemento ObjectSpawner con una clase ObjectSpawner y un elemento Indicator_Place con una clase Placement_Indicator.

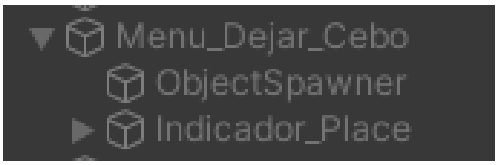


Ilustración 102: Elementos de Menu_Dejar_Cebo

Tanto la clase ObjectSpawner como Placement_Indicator realizan el mismo funcionamiento que en la acción "Ayuda". La única diferencia es que, en este caso, la clase ObjectSpawner en vez de invocar a la bestia "Anjana" invoca al objeto prefabricado de Unity llamado "Cebo". Tal y como se ve en la imagen 104.

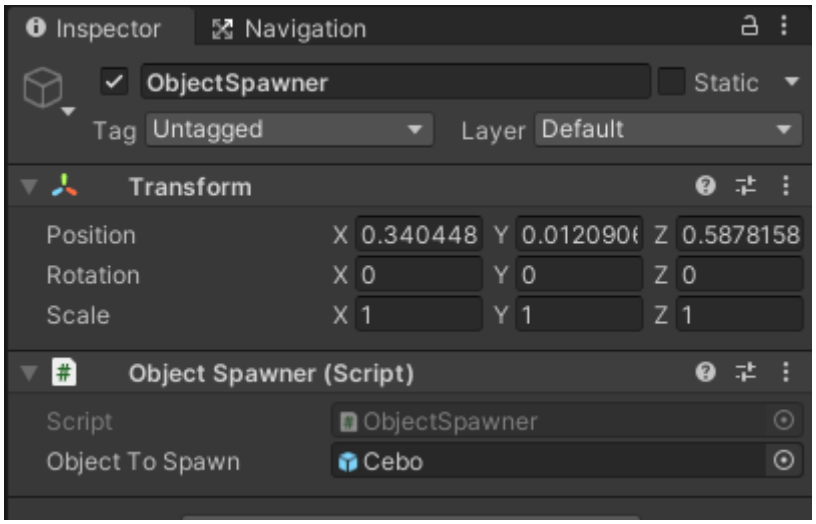


Ilustración 103: Elemento ObjectSpawner de Menu_Dejar_Cebo

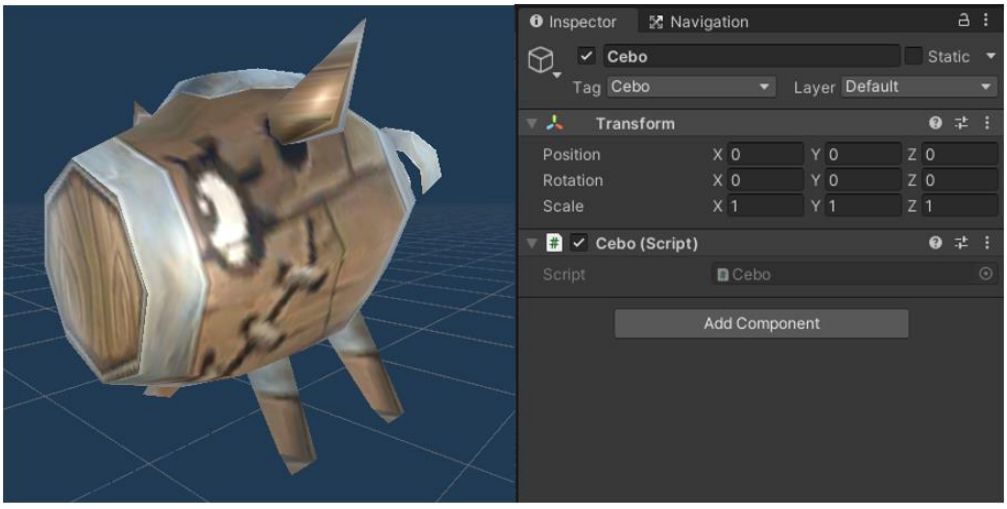


Ilustración 104: Prefab de Cebo

El objeto "Cebo" lleva asociado la clase **Cebo** (ver anexo 1 apartado 26), la cual cuando aparece en la escena se ejecutará el método Update que llama a la función SeguirCebo de la clase Cuelebre. Esta clase será vista en el capítulo 4.3.7, simplemente decir que esta función hará que la bestia llamada "Cuelebre" se acerque a la posición del "Cebo".

En el momento en el que la bestia “Cuelebre” se encuentre a una determinada distancia del “Cebo”, comenzará una cuenta atrás que al terminar destruirá el propio “Cebo”.

```
void Update()
{
    cuelebre = GameObject.FindGameObjectWithTag("Cuelebre");
    cuelebre.GetComponent<Cuelebre>().SeguirCebo();

    if(cuelebre != null)
    {
        dist_cuelebre = Mathf.Sqrt(Mathf.Pow(cuelebre.transform.position.x - this.transform.position.x,2) +
                        Mathf.Pow(cuelebre.transform.position.z - this.transform.position.z,2));
        if(dist_cuelebre < 5f)
        {
            cuelebre.GetComponent<Cuelebre>().comiendo = true;
            Destroy(this.gameObject,60f);
        }
    }

}
```

Ilustración 105: Función Update de la clase Cebo

Por último, dentro del elemento Menu_Dejar_Cebo estará la clase **Menu_Cebo** (ver anexo 1 apartado 25) que se encarga de ver si hay algún objeto “Cebo” ya instanciado en la escena. De ser así, el botón de acción que permite invocar el objeto digital y el elemento Indicador_Place que permite ver dónde colocar el “Cebo” serán desactivado para evitar que pueda haber más de un objeto “Cebo” en la escena. En caso contrario, ambos elementos estarán activos.

```
public class Menu_Cebo : MonoBehaviour
{
    // Start is called before the first frame update
    2 references
    public GameObject Indicator_place;
    2 references
    public Button boton_accion;


    0 references
    void Update()
    {
        var cebo_colocado = FindObjectOfType<Cebo>();
        if(cebo_colocado != null)
        {
            Indicator_place.SetActive(false);
            boton_accion.interactable = false;
        }

        else{
            Indicator_place.SetActive(true);
            boton_accion.interactable = true;
        }


    }
}
```

Ilustración 106: Clase Menu_Cebo

Cruz de Santa Marta

La acción de “Cruz de Santa Marta” permite al usuario invocar el objeto “CruzSantaMarta” cuando este pulsa sobre la pantalla. El objeto aparecerá delante del

jugador y desaparecerá si este separa el dedo de la pantalla táctil. En caso de estar cerca de la bestia “Tarasca” y apuntarle con el objeto “CruzSantaMarta” invocado, hará que la “Tarasca” sea derrotada.
Hay dos elementos en la escena que se encargan del funcionamiento de esta acción: el elemento CruzController del Canvas y el elemento Plane hijo de la AR Camera.

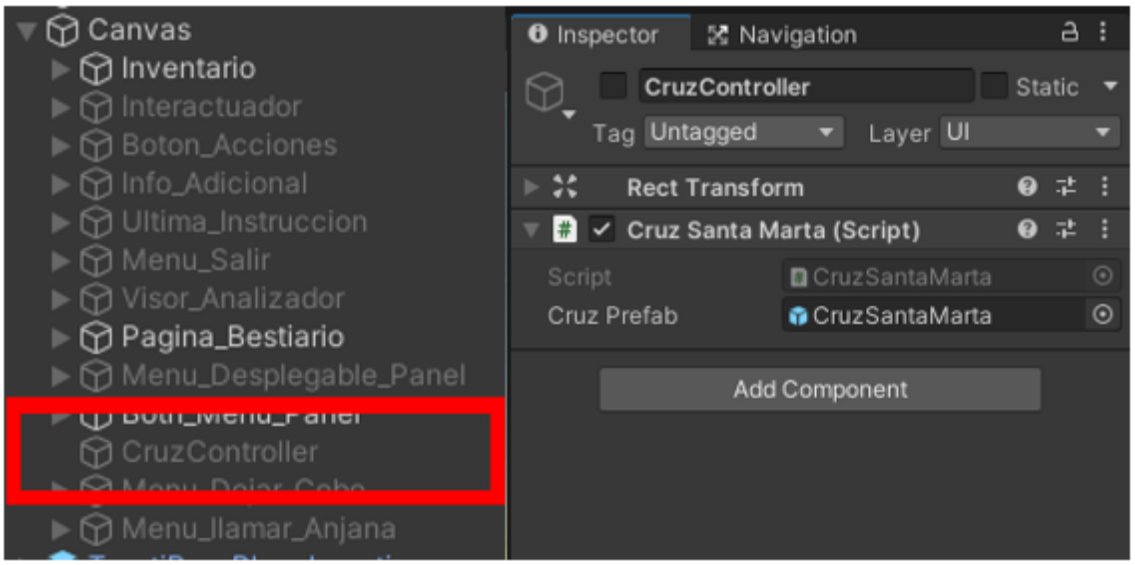


Ilustración 107: Elemento CruzController

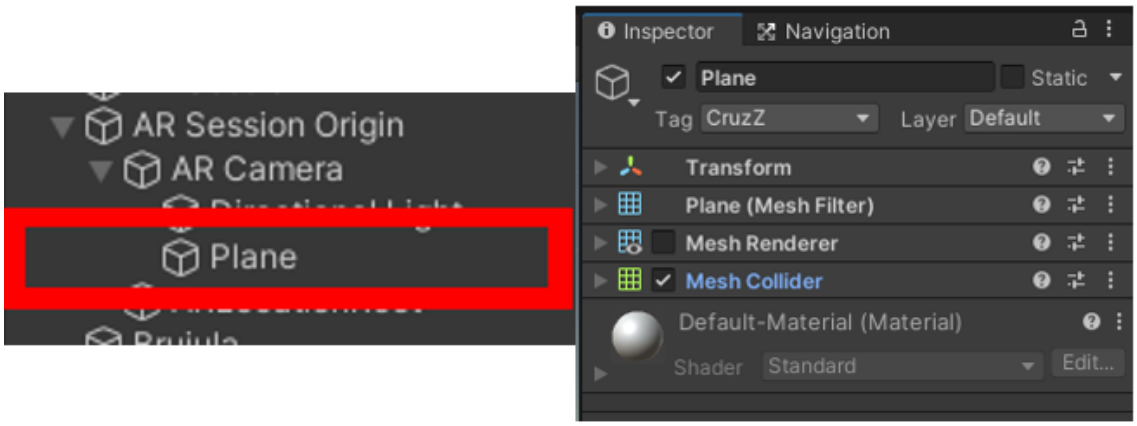


Ilustración 108: Elemento Plane hijo de AR Camera

La clase **CruzSantaMarta** (ver anexo 1 apartado 27) se encarga de controlar el funcionamiento de la acción, permitiendo al jugador crear una instancia del objeto predefinido de “CruzSantaMarta”.

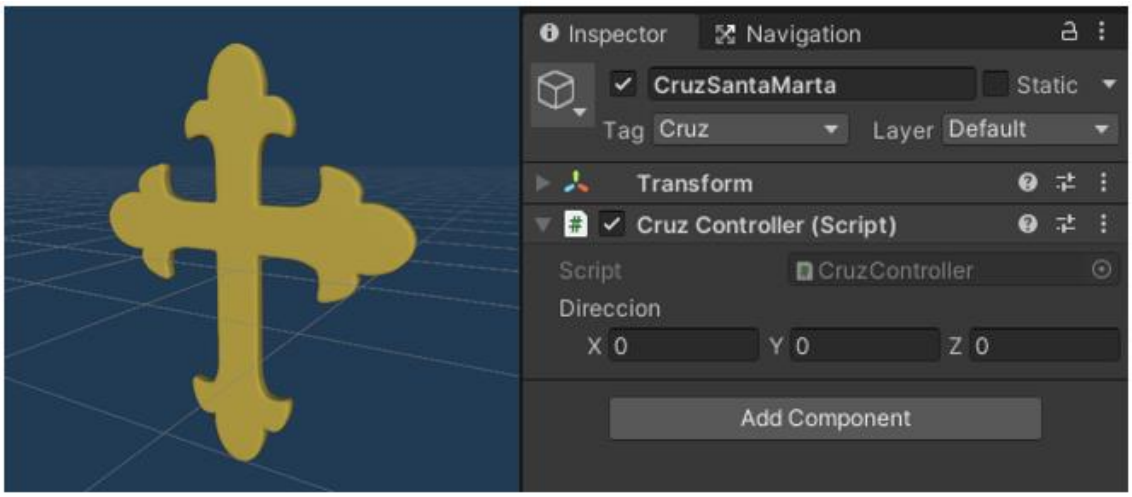


Ilustración 109: Prefab CruzSantaMarta

La clase de CruzSantaMarta posee dos métodos que se encargan de todo el funcionamiento de la acción: la función Update y DestruirCruces.

- **Update**: se encarga de controlar el funcionamiento de las pulsaciones sobre la pantalla. Cuando se pulsa sobre la pantalla se creará una instancia de "CruzSantaMarta" (imagen 109), en caso de que el dedo del usuario se mueva por la pantalla, el objeto de la cruz se moverá a la posición del dedo y si el dedo es retirado de la pantalla la cruz desaparecerá. Para la creación de la cruz delante del usuario se emplea la función Raycast para crear un rayo que salga desde la posición del dedo y colisione con el elemento Plane (con el tag "CruzZ") de la imagen 108, el cual es un plano que permanece delante de la cámara en el cual se crea el objeto prefabricado en la posición en donde colisiona el rayo.

```
void Update()
{
    if(Input.GetMouseButtonDown(0))
    {
        var crucesRezagadas = GameObject.FindGameObjectsWithTag("Cruz");
        foreach(GameObject c in crucesRezagadas)
        {
            Destroy(c);
        }
        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

        if(Physics.Raycast(ray,out hit))
        {
            if(hit.collider.tag == "CruzZ")
            {
                cruz = Instantiate(cruzPrefab, hit.point, new Quaternion(0,0,0,0)) as GameObject;
                cruz.transform.parent = Camera.main.transform;
                cruz.transform.localRotation = new Quaternion(0,0,0,0);
            }
        }
    }

    else if(Input.GetMouseButton(0))
    {
        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

        if(Physics.Raycast(ray,out hit))
        {
            if(hit.collider.tag == "CruzZ")
            {
                cruz.transform.position = hit.point;
                cruz.transform.localRotation = new Quaternion(0,0,0,0);
            }
        }
    }

    else if(Input.GetMouseButtonUp(0))
    {
        Destroy(cruz);
    }
}
```

Ilustración 110: Función Update de la clase CruzSantaMarta

- **DestruirCruces**: la función busca cualquier objeto “CruzSantaMarta” y lo destruye. Esta función ha sido necesaria por un problema en el que, al cambiar de acción en el menú, el objeto “CruzSantaMarta” se mantenía en pantalla. Es por eso por lo que esta función se ejecuta siempre que se cambia de acción o se pulsa cualquier botón en el menú.

Como se puede ver en la imagen 109, el objeto “CruzSantaMarta” tiene asociado el script CruzController que hacer referencia a la clase **CruzController** (ver anexo 1 apartado 28). Esta clase se encarga de emplear la función Raycast para crear un rayo que salga de la posición del objeto “CruzSantaMarta”. Si este rayo colisiona con la bestia “Tarasca” activa la función Amenazado de la misma, esta función hace que la bestia sea derrotada. La función Amenazado se verá en profundidad en el capítulo 4.3.7.

```
public class CruzController : MonoBehaviour
{
    0 references
    void Update()
    {
        RaycastHit hit;
        if(Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward),out hit, 45f))
        {
            if(hit.collider.tag == "Tarasca")
            {
                FindObjectOfType<Tarasca>().Amenazado();
            }
        }
    }

}
```

Ilustración 111: Clase CruzController

4.3.6. Implementación de las misiones

Tal y como se ha comentado en el capítulo 3.1, durante la experiencia de realidad aumentada el usuario deberá realizar una serie de misiones para finalmente completarla. Pese a que en dicho capítulo se han mencionado que hay cuatro pruebas, algún de estar tareas requieren varios pasos para realizarse, es por ello por lo que en la implementación se han extraído de las cuatro tareas iniciales las distintas acciones que han de realizarse en cada una de ellas, consiguiendo un total de siete pasos a realizar para completar la aplicación.

Estos siete pasos serán las misiones que la clase **Misiones_Controller** (ver anexo 1 apartado 4) tendrá en cuenta a la hora de controlar que acciones debe realizar el usuario para avanzar en la aplicación.

Dividir las cuatro pruebas en siete misiones más simples resulta de gran utilidad no solo para tener en cuenta el progreso del usuario, sino también para que en caso de que haya un error en la aplicación, esta guarde en que parte de la prueba actual se encuentra el usuario y de esa forma que el usuario no deba repetir toda la prueba.

Por ejemplo, durante la primera prueba el usuario debe de encontrar a cuatro bestias de nombre “Trenti” y reunirlos con su hermano “TrentiBoss”. Sin embargo, esta prueba se divide en tres partes que en implementación se han considerado misiones distintas: hablar con “TrentiBoss” para dar al usuario contexto de la misión, recoger a los cuatro “Trentis” interactuando con cada uno de ellos y regresar con “TrentiBoss” para que actualice la dirección de la brújula y de por finalizada la prueba. En caso de que surjan problemas y la aplicación se interrumpiera antes de que el usuario pudiese hablar con “TrentiBoss”, dividiendo esta prueba entre misiones se consigue que la aplicación guarde el progreso del usuario en el momento en donde termina la fase de recoger “Trentis”, de esa forma al reiniciar la aplicación el usuario no tendrá que volver a realizar la primera prueba desde el principio.

Ya se ha visto en la imagen 75 del capítulo 4.3.4 que la clase Misiones_Controller se encuentra asociada al objeto Controller_Game de la escena.

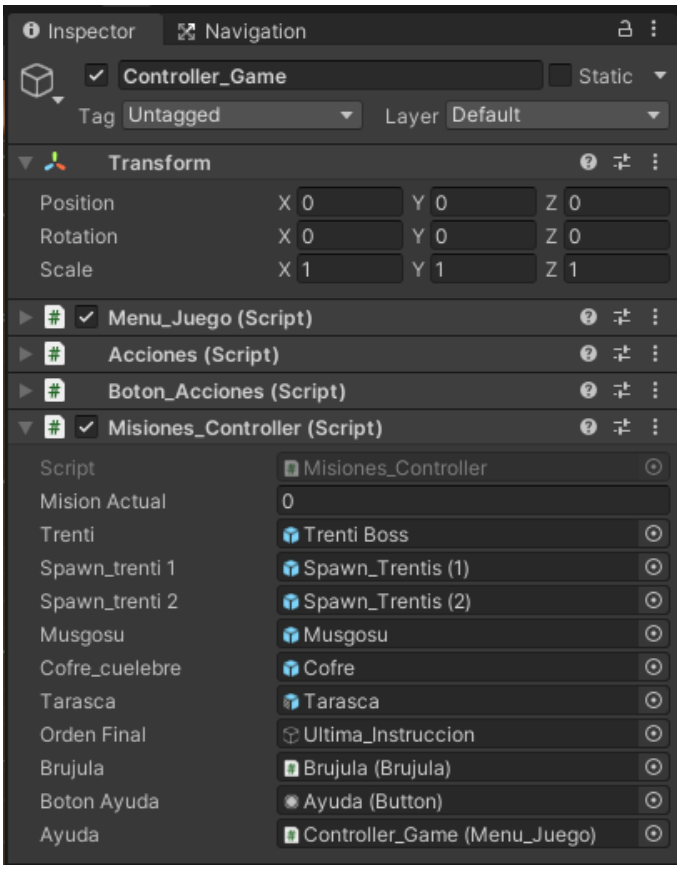


Ilustración 112: Componente Mision_Controller del objeto Controller_Game

Como se puede ver en la imagen 112, la clase Misión_Controller recoge por referencia todos los objetos digitales de las bestias (estos objetos serán vistos en profundidad en el capítulo 4.3.7) que serán en su mayoría los objetivos a los que apuntarán las flechas que creará la clase Brujula vista en el capítulo 4.3.5, la cual será empleada en esta clase para crear nuevas flechas que apunten a los objetivos de la misión. También se puede apreciar como coge otros elementos de la escena como Ultima_Instrucción que hace referencia al cartel con la última instrucción para finalizar la aplicación, el botón de la acción “Ayuda” y la clase “Menu_Juego” para activar en ella el indicador de que hay un nuevo consejo en el menú ayuda una vez se haya completado una misión.

```
public class Misiones_Controller : MonoBehaviour
{

    12 references
    public int misionActual;
    2 references
    private int misionAnterior;
    6 references
    public GameObject trenti;
    6 references | 6 references
    public GameObject spawn_trenti1, spawn_trenti2;
    5 references
    public GameObject Musgosu;
    3 references
    public GameObject cofre_cuelebre;
    2 references
    public GameObject Tarasca;
    1 reference
    public GameObject OrdenFinal;
    8 references
    public Brujula brujula;
    1 reference
    public Button botonAyuda;
    1 reference
    public Menu_Juego ayuda;


    0 references
    private void Awake() {···

    6 references
    public void Cambiar_Mision()···

    2 references
    public void ActivarMision(int id_mision)···
}
```

Ilustración 113: Clase Misiones_Controller

La ilustración 113 muestra las distintas funciones que permiten que la aplicación pueda pasar de una misión a otra controlando el progreso del usuario y activar la misión que corresponda en cada momento.

- **Awake**: se encarga de inicializar la misión que debe hacer el usuario en función del valor del PlayerPref “num_mision” visto en el capítulo 4.3.1. Al ejecutarse, Awake llama a la función ActivarMision pasándole como parámetro el valor del “num_mision”. Tras ello, en función del valor del PlayerPref, se destruirán determinados elementos de la escena que no deberían de existir durante la misión que se está realizando.

```
private void Awake() {
    misionActual = PlayerPrefs.GetInt("num_mision",1);
    ActivarMision(misionActual);
    switch(misionActual)
    {
        case 3:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            break;
        case 4:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            break;
        case 5:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            Destroy(Musgosu);
            break;
        case 6:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            Destroy(Musgosu);
            cofre_cuelebre.GetComponent<Cofre_Cuelebre>().abierto = true;
            break;
        case 7:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            Destroy(Musgosu);
            cofre_cuelebre.GetComponent<Cofre_Cuelebre>().abierto = true;
            Destroy(Tarasca);
            break;
    }
}
```

Ilustración 114: Función Awake de la clase Misiones_Controller

- **Cambiar_Mision**: esta función se encarga de eliminar las flechas ya existentes de la brújula empleando la función deleteObjetivos de la clase Brujula y activar la siguiente misión empleando la función ActivarMision. Se puede apreciar en la imagen 115 como emplea la función MostrarNuevaAyuda de la clase Menu_Juego para que aparezca un indicador en el botón de ayuda indicando que hay un nuevo consejo.

```
public void Cambiar_Mision()
{
    misionActual++;
    if(misionActual != misionAnterior)
    {
        misionAnterior = misionActual;
        PlayerPrefs.SetInt("num_mision",misionActual);
        brujula.deleteObjetivos();
        ActivarMision(misionActual);
        ayuda.MostrarNuevaAyuda();
    }
}
```

Ilustración 115: Función Cambiar_Mision de la clase Misiones_Controller

- **ActivarMision**: se encarga de emplear la función addObjetivo de la clase Brujula para crear flechas de dirección que apuntarán a los objetivos de la misión. En función del valor de la variable “id_mision” de tipo integer, crear

una flecha que apuntará a un objetivo o a otro. En caso de que se le pase el "id_mision" correspondiente a la última misión lo único que hará será activar el cartel que indica el último paso para completar la aplicación.

```
public void ActivarMision(int id_mision)
{
    switch(id_mision)
    {
        case 1:
            brujula.addObjetivo(trenti);
            break;
        case 2:
            brujula.addObjetivo(spawn_trenti1);
            brujula.addObjetivo(spawn_trenti2);
            break;
        case 3:
            brujula.addObjetivo(trenti);
            break;
        case 4:
            Musgosu.SetActive(true);
            brujula.addObjetivo(Musgosu);
            break;
        case 5:
            brujula.addObjetivo(cofre_cuelebre);
            break;
        case 6:
            brujula.addObjetivo(Tarasca);
            break;
        case 7:
            botonAyuda.interactable = true;
            OrdenFinal.SetActive(true);
            break;
    }


}
```

Ilustración 116: Función ActivarMision de la clase Misiones_Controller

4.3.7. Implementación de las bestias

Durante todo el proyecto se ha estado explicando que los usuarios se encontrarán con distintos seres mitológicos provenientes de la mitología ibérica. Todas estas criaturas han sido creadas como objetos predefinidos de Unity y asociadas cada una a una de las clases herederas de la clase **Bestia** (ver anexo 1 apartado 10) vistas en la imagen 37.

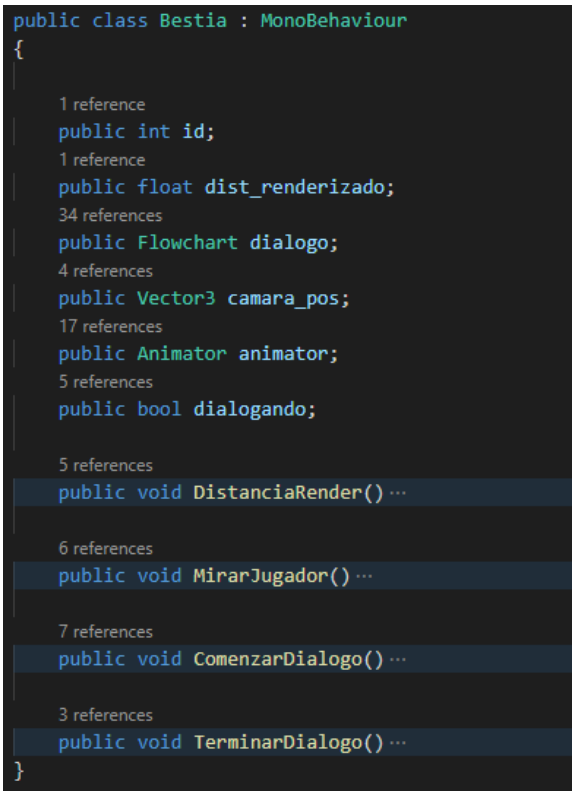


Ilustración 117: Clase Bestia

La clase Bestia se encarga de recoger todos los atributos y métodos comunes a las bestias. Los atributos que almacena esta clase son los siguientes:

- **id**: se trata del identificador de la bestia, este atributo coincidirá con la variable "id" de la clase Entrada_Bestia de la misma bestia a la que haga referencia.
- **dist_renderizado**: debido a que no se le ha podido añadir a la experiencia de RA la propiedad "depth tracking" (seguimiento de profundidad) que permite detectar si un objeto del mundo real está más próximo que un objeto digital, muchos de los objetos digitales podrían verse desde muy lejos arruinando la sensación de profundidad y dificultando la inmersión del usuario dentro de la experiencia RA.

  Es por ello que, para evitar este problema se ha decidido que solo se renderizarán aquellos objetos digitales que se encuentren a una determinada distancia de la cámara. Haciendo que estos objetos se vean en pantalla cuando el usuario este lo suficientemente cerca de ellos como para que la integración de los mismos en la escena parezca natural.

  Sin embargo, ya se ha comentado que no todos los objetos son igual de grandes, por lo que para evitar que un objeto muy grande se integre demasiado pronto o un objeto muy pequeño se integre demasiado tarde, se ha decidido asignarle a cada bestia una distancia de renderizado en función de lo grandes que sean con respecto a la cámara.

- **dialogo**: este elemento pertenece a la clase Flowchart, que se trata de un objeto perteneciente a la librería Fungus, cuyo asset ya ha sido mencionado en el capítulo 4.1.1.1.

  Flowchart hace referencia a un objeto asociado a los Prefabs de las bestias que se encarga de almacenar los distintos diálogos de las bestias. Una misma bestia puede tener distintos diálogos que están separados en distintos bloques, los cuales contienen el texto que aparecerá en los cuadros de dialogo y en los que se puede llamar a algunas funciones. Cada uno de estos bloques se puede ejecutar empleando la función ExecuteBlock y en ellos no solo se guardan diálogos que saldrán en el cuadro de texto, también se pueden invocar métodos de clases que estén en asociadas al mismo objeto al que está asociado el Flowchart.

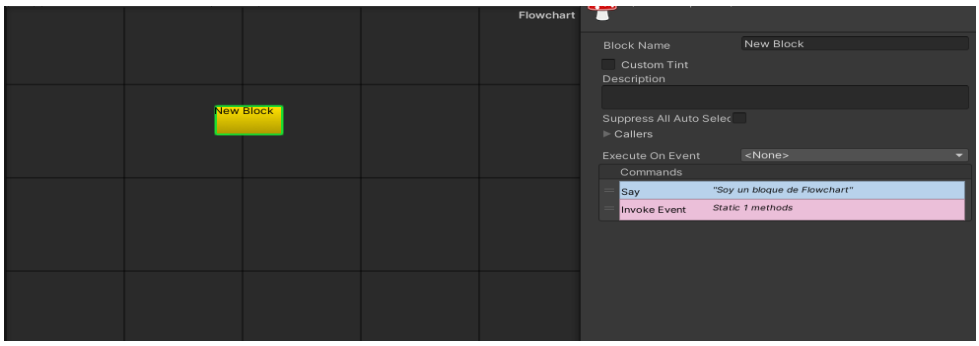


Ilustración 118: Ejemplo Flowchart

  En el ejemplo de la imagen 118 se puede ver un Flowchart con un único bloque llamado “New Block”. En dicho bloque se muestra un elemento Say que será un texto que aparecerá en cuadro de texto que invocará el Flowchart y luego una llamada a un método mediante el Invoke Event, que permitirá llamar a un evento de algún script que esté asociado al objeto al que está asociado el Flowchart.

  Como detalle a mencionar, para que en el cuadro de texto aparezca un nombre que indique quien está hablando en ese momento, el Flowchart se asocia con el script Character en el que se indica el nombre que saldrá en el cuadro de texto y el color en el que saldrá dicho nombre. Más adelante se mostrarán los distintos Flowchart asociados a cada una de las bestias.

- **camara_pos**: se trata de un vector que indica la posición en la que se encuentra la cámara. Esta variable es empleada para poder calcular la distancia que hay entre la cámara y la bestia.
- **animator**: esta variable hace referencia al elemento Animator que se encarga de controlar cuando se ejecutan las distintas animaciones de las bestias y en qué orden. Cada bestia tiene un árbol de animaciones propio que será visto más adelante.

- **dialogando**: esta variable booleana se mantiene con el valor false hasta que el usuario emplea la acción “Interactuar”, es entonces cuando la variable pasa a ser true. Esta variable se emplea para saber en qué momento la bestia está dándole algún mensaje al usuario.

Una vez vistos los atributos que comparten todas las bestias es momento de ver qué métodos les son comunes:

- **DistanciaRender**: este método calcula la distancia que hay entre la bestia y la cámara por medio de la variable “camara_pos”. Si el valor que se obtiene es inferior al valor de la variable “dist_renderizado” asociada a la bestia, la mesh de la bestia será renderizada permitiendo que sea vista por el usuario. En caso contrario la mesh permanecerá desactivada y sin renderizar hasta que la distancia entre el usuario y la bestia sea lo suficientemente corta.
- **MirarJugador**: esta función se encarga de cambiar la rotación en el eje Y de la bestia para que mire al usuario cuando esté interactuando con ella.
- **ComenzarDialogo**: esta función modifica el valor de la variable “dialogando” a true.
- **TerminarDialogo**: esta función modifica el valor de la variable “dialogando” a false.

Una vez conocidas todas las funciones y variables que comparten las distintas bestias, es momento de explorarlas una a una y entender su comportamiento.

Anjana

La “Anjana” es la bestia que el usuario podrá invocar mediante la acción “Ayuda” para que le dé consejos y pistas sobre qué hacer a continuación. Sirve como la guía del usuario durante la experiencia de realidad aumentada y se le asocia la clase **Anjana** (ver anexo 1 apartado 11) que se encarga de manejar su funcionamiento.

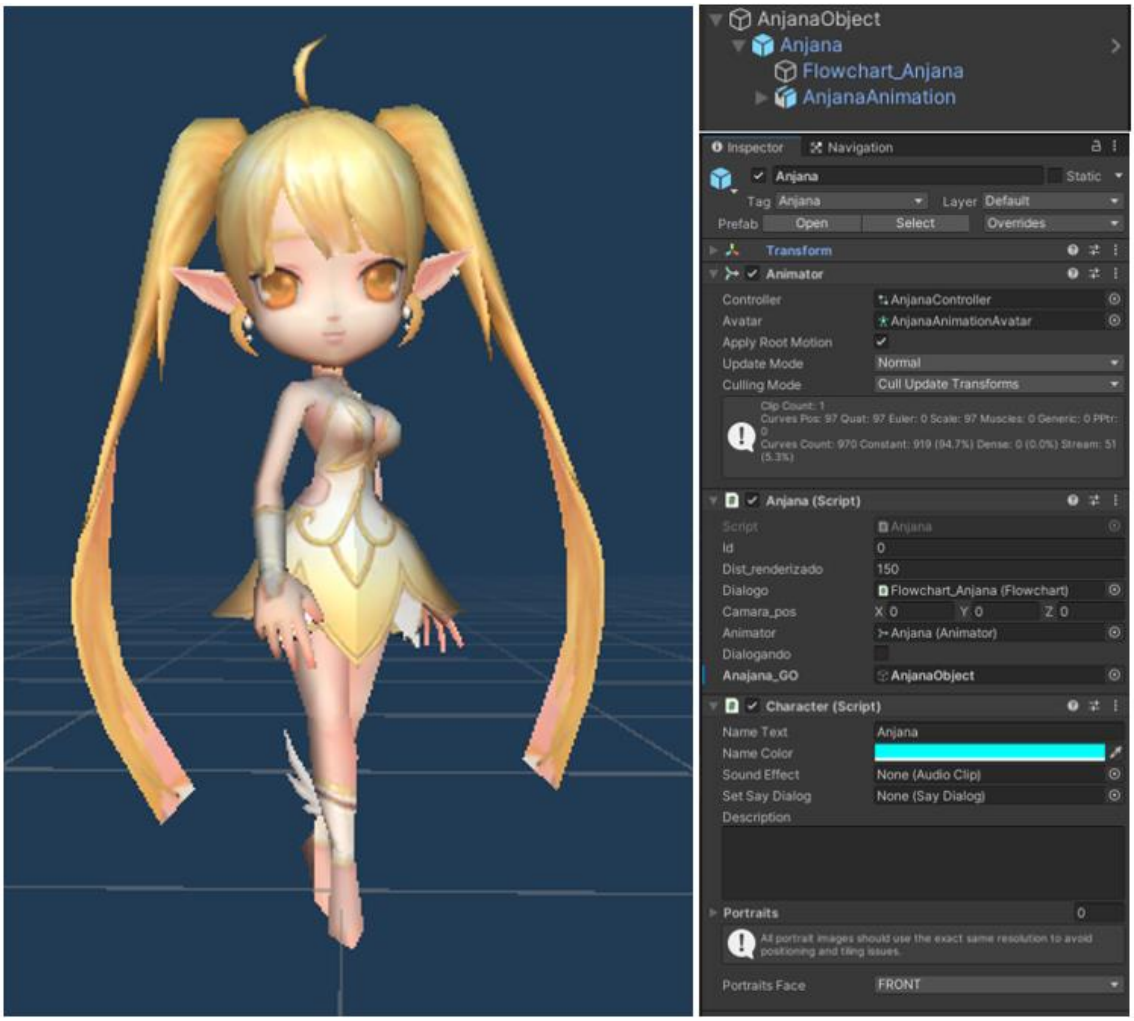


Ilustración 119: Prefab Anjana

Un dato característico sobre el Prefab de Anjana es que se encuentra dentro del Prefab AnjanaObject, que es un objeto sin ningún elemento dentro. Esto se debe a que el objeto Anjana va a realizar una animación en la que se mueve en el eje Y al aparecer y al desaparecer para dar la sensación de levitar. Para que no sucedieran problemas de movimiento, se decidió poner un objeto vacío padre que mantuviese su posición en un punto y colocar el Prefab de Anjana como hijo para que se moviese en el eje Y con respecto a la posición del objeto padre AnjanObject sin perjudicar de esa forma el posicionamiento del objeto digital durante la experiencia.

En este caso la “Anjana” tiene un árbol de animaciones muy simple, con una única animación que se repite en bucle que funciona de Idle.

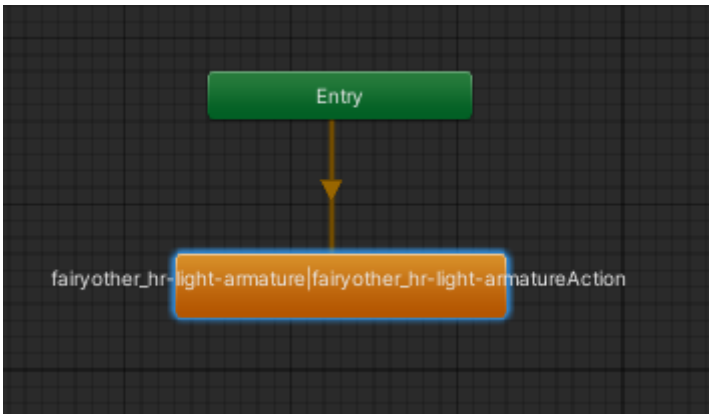


Ilustración 120: Árbol de animación de Anjana

Entrando en lo referente a la clase Anjana, esta se encarga de controlar el comportamiento de la bestia mientras esta se encuentre en la escena.

```
public class Anjana : Bestia
{
    2 references
    private int mision_actual;

    2 references
    private float vel = 0.25f;

    3 references | 4 references
    private bool descenso, ascenso;

    1 reference
    public GameObject Anajana_GO;

    // Start is called before the first frame update
    0 references
    void Start() ···

    // Update is called once per frame
    0 references
    void Update() ···

    0 references
    public void Irse() ···

    0 references
    public void Finalizar() ···
}
```

Ilustración 121: Clase Anjana

La clase Anjana posee algunos atributos propios muy interesantes:

- **mision_actual**: debido a que en función de la misión en la que se encuentre el usuario se ejecutará un determinado bloque del Flowchart, es necesario para esta clase tener una variable en la que guarde la misión en la que se encuentra el usuario actualmente.
- **vel**: antes se ha mencionado que el objeto Anjana realiza una animación de forma manual que hace desplazarse al objeto en el eje Y cuando aparece y cuando se va. Esta variable define la velocidad de ese desplazamiento.
- **descenso**, **ascenso**: son los valores booleanos que se encargan de indicar si la "Anjana" debe desplazarse hacia abajo en el eje Y o hacia arriba.

- **Anjana_Go**: hace referencia al objeto padre que se ha mencionado previamente. Esto se debe a que cuando le Prefab Anjana deba ser destruido, se deberá destruir desde su padre.

Los métodos que emplea la clase son:

- **Start**: inicializa las variables "descenso" y "ascenso" a true y false respectivamente. Tras esto llama a la función ComenzarDialogo vista anteriormente y en función de la misión actual en la que se encuentre el usuario ejecutará un bloque determinado de su Flowchart.

```
void Start()
{
    dialogo = GetComponentInChildren<Flowchart>();
    ComenzarDialogo();
    descenso = true;
    ascenso = false;
    camara_pos = Camera.main.transform.position;
    this.transform.localPosition = new Vector3(0,0.75f,0);
    mision_actual = FindObjectOfType<Misiones_Controller>().misionActual;

    switch(mision_actual)
    {
        case 1:
            dialogo.ExecuteBlock("Mision1");
            break;
        case 2:
            dialogo.ExecuteBlock("Mision2");
            break;
        case 3:
            dialogo.ExecuteBlock("Mision3");
            break;
        case 4:
            dialogo.ExecuteBlock("Mision4");
            break;
        case 5:
            if(PlayerPrefs.GetInt("registro_Cuelebre",0) == 1)
            {
                dialogo.ExecuteBlock("Mision5_2");
            }
            else{
                dialogo.ExecuteBlock("Mision5");
            }
            break;
        case 6:
            dialogo.ExecuteBlock("Mision6");
            break;
        case 7:
            dialogo.ExecuteBlock("Final");
            break;
    }
}
```

Ilustración 122: Función Start de la clase Anjana

Ilustración 123: Flowchart con los distintos bloques de texto de Anjana

Como dato particular, hay un bloque llamado Mision5_2 que solo se activará si el usuario se encuentra en la misión 5 y ha analizado a la bestia "Cuelebre". Se trata de un bloque de texto que da información adicional sobre qué hacer en la misión 5.

- **Update**: esta función llama al método MirarJugador para que la "Anjana" siempre esté orientada hacia el usuario. Al mismo tiempo revisa si el movimiento de ascenso o descenso en el eje Y a finalizado. En el caso de que el movimiento de ascenso finalice, significará que la Anjana ya se ha ido, por lo que destruirá el objeto AnjanaObject destruyendo todo el Prefab Anjana.
- **Irse**: modifica el valor de la variable "ascenso" a true, lo que hará que en el update se active la condición de ascender, que en cuanto finalice destruirá el objeto Anjana. Esta variable se ejecuta en los bloques del Flowchart después de terminar de mostrar todo el texto.
- **Finalizar**: esta función solo se ejecutará cuando se finalice el bloque del Flowchart llamado "Final". Una vez se ejecute esta función, se cerrará la escena de Unity actual y cargará la escena de nombre "Final".

TrentiBoss

"TrentiBoss" es la primera bestia con la que el usuario deberá interactuar. Se trata de un "Trenti" que ha perdido a sus hermanos, por lo que el usuario deberá interactuar con él para que le informe de la misión y cuando haya recogido a los demás "Trentis" regresar e informarle de que todos han sido recogidos para dar la misión por finalizada.

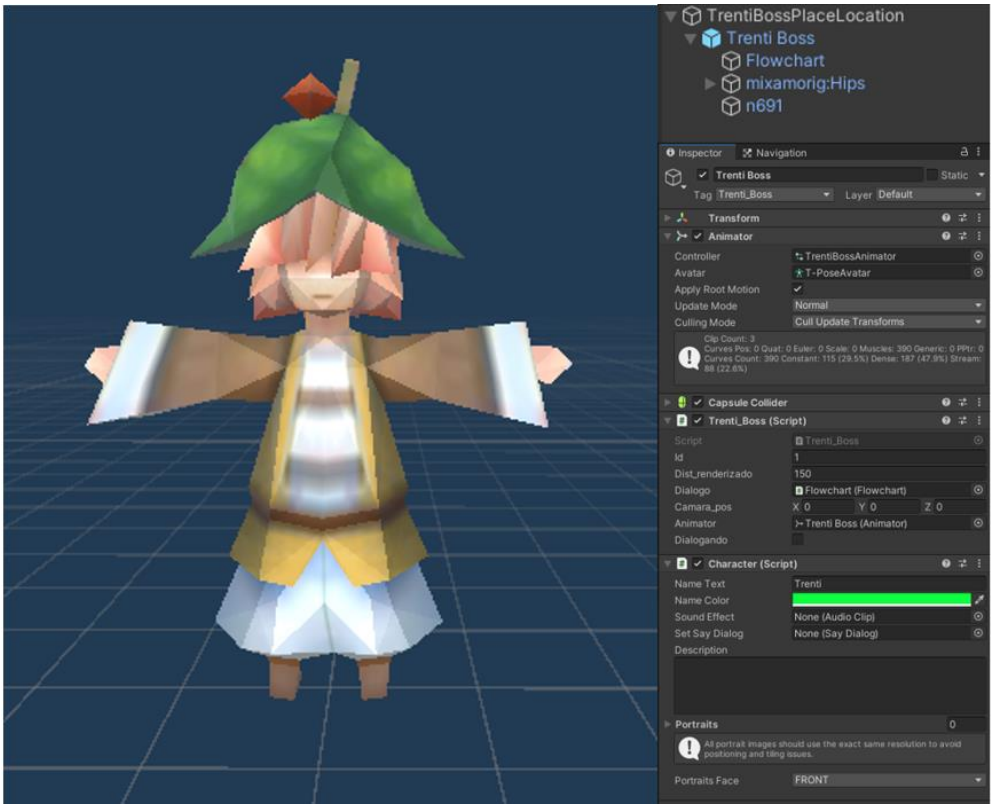


Ilustración 124: Prefab TrentiBoss

Al igual que el Prefab Anjana, TrentiBoss también es hijo de otro objeto, en este caso TrentiBossPlaceLocation. Esto se debe a que el script PlaceAtLocation (que se encarga de colocar el objeto digital en una zona empleando las coordenadas del GPS) colocaba al Trenti demasiado arriba en el eje Y, por lo que se colocó todo el Prefab TrentiBoss

como hijo de TrentiBossPlaceLocation y se le disminuyó su posición en el eje Y para que acabase a una altura adecuada para el jugador.

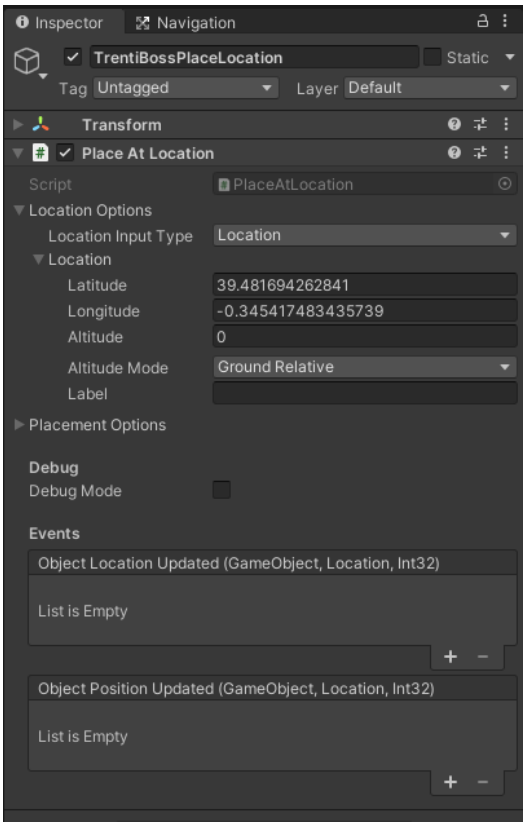


Ilustración 125: Elementos del objeto TrentiBossPlaceLocation

En el caso de "TrentiBoss" el árbol de animación es un poco más complicado, teniendo una animación central llamada "Sad" (triste) que puede cambiar a "cry" (llorar) o a "happy" (alegre).

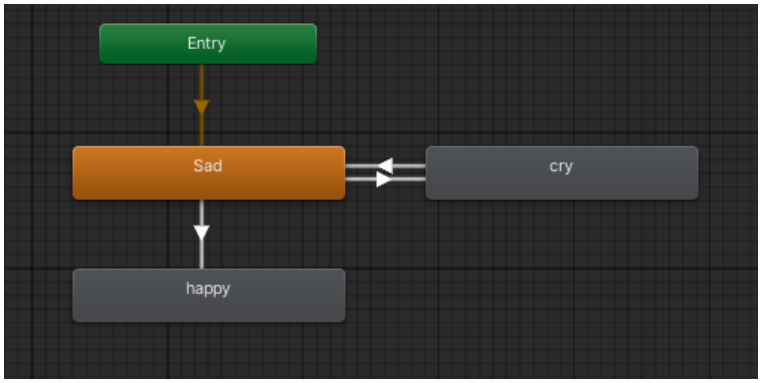


Ilustración 126: Árbol de animación de TrentiBoss

El objeto TrentiBoss tiene asociado la clase **Trenti_Boss** (ver anexo 1 apartado 12) que se encarga del comportamiento del Trenti. En este caso la clase no añade nuevos atributos, pero sus métodos son los siguientes:

```
public class Trenti_Boss : Bestia
{
    0 references
    void Start() ···

    // Update is called once per frame
    0 references
    void Update() ···

    1 reference
    public void Dialogo() ···

    0 references
    public void Irse() ···

    0 references
    public void CerrarDialogoCry() ···
}
```

Ilustración 127: Clase Trenti_Boss

- **Start**: se encarga de inicializar las variables "animator" y "dialogo" para que coincidan respectivamente con los elementos Animator y Flowchart asociados al objeto TrentiBoss.
- **Update**: ejecuta las funciones DistanciaRenderer para controlar cuando es visible el "TrentiBoss" y MirarJugador para que cuando esté dialogando con el usuario esté mirando hacia éste.
- **Dialogo**: se ejecuta cuando el usuario realiza la acción "Interactuar" sobre el "TrentiBoss", lo que hace que ejecute ComenzarDialogo para indicar que se está dialogando con la bestia. En función de en qué misión se encuentre el usuario ejecutará uno de los tres bloques de Flowchart. En caso en que el usuario ya haya reunido a los hermanos de "TrentiBoss" cambiará la animación predefinida "Sad" (triste) a "happy" (alegre), en caso contrario se activará la acción "cry" (llorar).

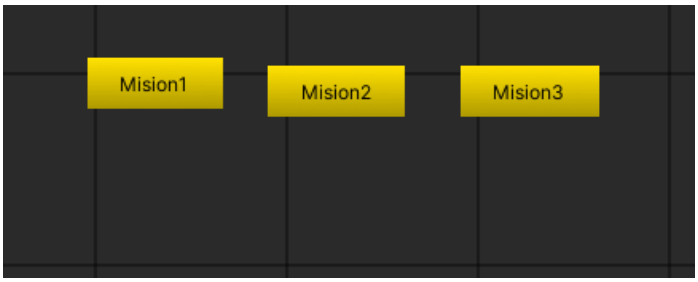


Ilustración 128: Flowchart con los distintos bloques de texto de TrentiBoss

- **Irse**: esta función se ejecutará al terminar de mostrar todo el texto del bloque de nombre "Mision3". Al terminar, la función destruye el Prefab para hacer desaparecer al "Trenti".
- **CerrarDialogoCry**: esta función se ejecuta al terminar los bloques "Mision1" y "Mision2". Al terminar, llama a la función TerminarDialogo para indicar que se ha terminado de dialogar con la criatura y desactivará la animación "cry".

Trenti

Los “Trentis” serán los objetos que representen a los hermanos de “TrentiBoss”. Estos serán recogidos cuando el usuario interactúe con ellos y haya recibido la misión de buscarlos por parte del “TrentiBoss”.

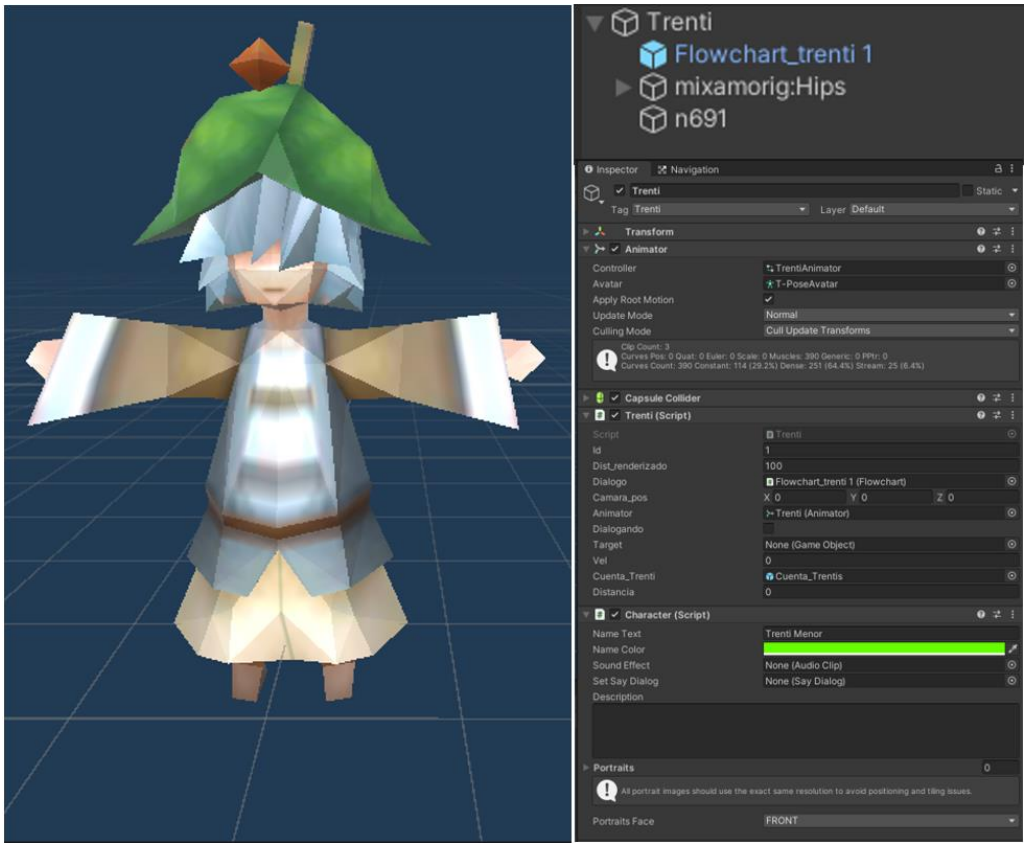


Ilustración 129: Prefab de Trenti

Antes de profundizar en el Prefab de Trenti, cabe mencionar que estos son creados como hijos del objeto llamado Spawn_Trenti, al que se le asocia la clase **SpawnTrenti** (ver anexo 1 apartado 13) que se encarga de crear instancias del Prefab Trenti en la escena, y el script PlaceAtLocation que se encargará de posicionar el Spawn_Trenti en una localización real mediante las coordenadas del GPS que se le asocien. De esta forma el objeto Spawn_Trenti permanecerá en la posición del GPS y los “Trentis” hijos se moverán a su alrededor.

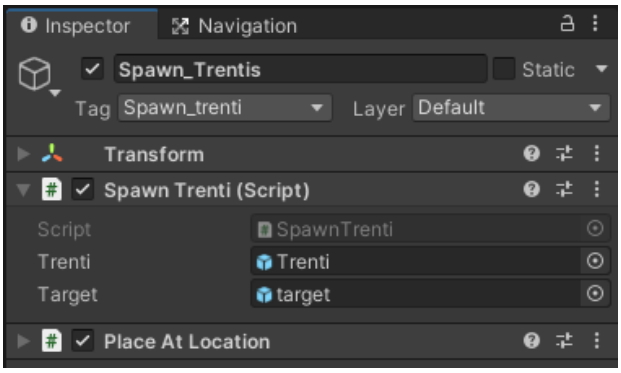


Ilustración 130: Elementos del objeto Spawn_Trentis

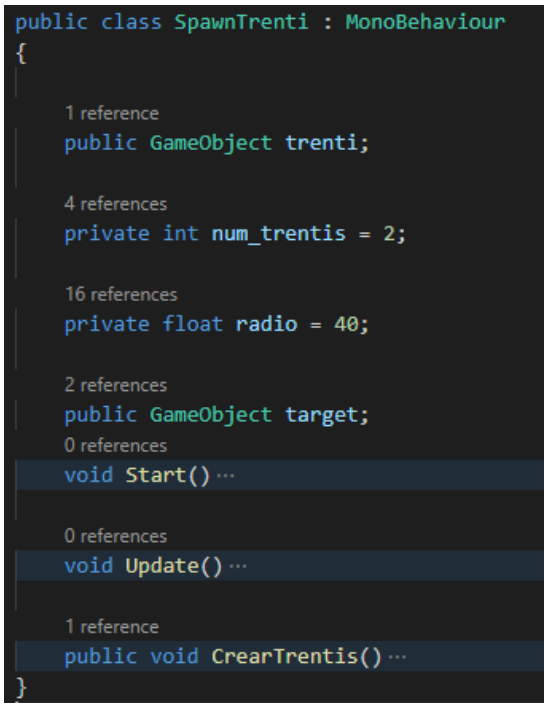


Ilustración 131: Clase SpawnTrenti

La clase SpawnTrenti muestra los siguientes atributos:

- **trenti**: hace referencia al objeto prefabricado de Unity Trenti visto en la imagen 129.
- **num_trentis**: esta variable tipo integer detalla cuantos objetos tipo Trenti creará la clase SpawnTrenti. En este caso cada SpawnTrenti creará dos instancias del objeto Trenti.
- **radio**: esta variable indica el radio máximo del SpawnTrenti, delimitando donde se crearán los "Trentis" y los objetos targets que servirán para indicar la dirección en la que se mueven los "Trentis".
- **target**: se trata de un objeto vació que sirve para guiar un "Trenti" a un punto cercano al Spawn_Trenti, haciendo que la bestia se mueva hacia la posición en la que se encuentra el objeto "target".

Los métodos que emplea esta clase son los siguientes:

- **Start**: inicializa el funcionamiento de la clase SpawnTrenti llamando a la función CrearTrentis.
- **Update**: por cada "Trenti" creado por la clase SpawnTrenti, se creará una instancia de un objeto "target". Si la distancia entre el "Trenti" y el "target" es muy reducida, se destruirá el objeto "target" y se creará uno nuevo dentro del radio definido por la variable "radio" atrayendo al "Trenti" a una nueva posición.

- **CrearTrentis**: este método crea un número de instancias del Prefab “Trenti” igual al valor de la variable “num_trenti” en una posición aleatoria dentro del radio definido por el valor de la variable “radio”.

Ahora que se conoce como son creados los “Trenti”, es momento de profundizar el Prefab mostrado en la ilustración 129.

En el caso del componente Animator, los “Trentis” presentan un árbol de animaciones idéntico al de “TrentiBoss” con la única excepción de que la animación predefinida de los “Trentis” es “sad_Walk” (caminar triste), ya que al contrario que el “TrentiBoss”, los “Trentis” se mueven por la zona definida por la variable “radio” de SpawnTrenti.

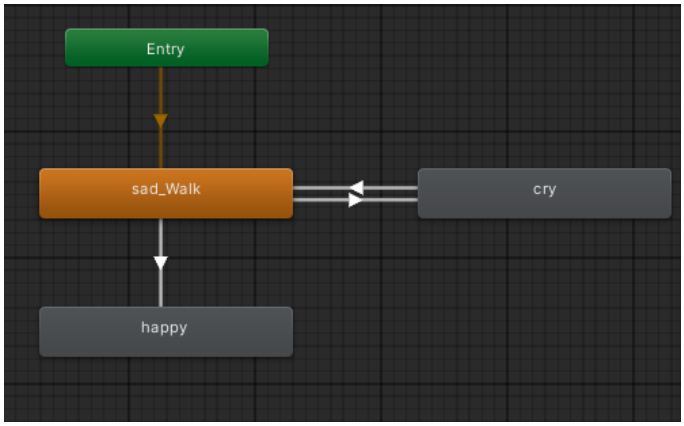


Ilustración 132: Árbol de animaciones de Trenti

El objeto Trenti se encuentra asociado a la clase **Trenti** (ver anexo 1 apartado 14) que se encarga de controlar el comportamiento de los “Trentis”.

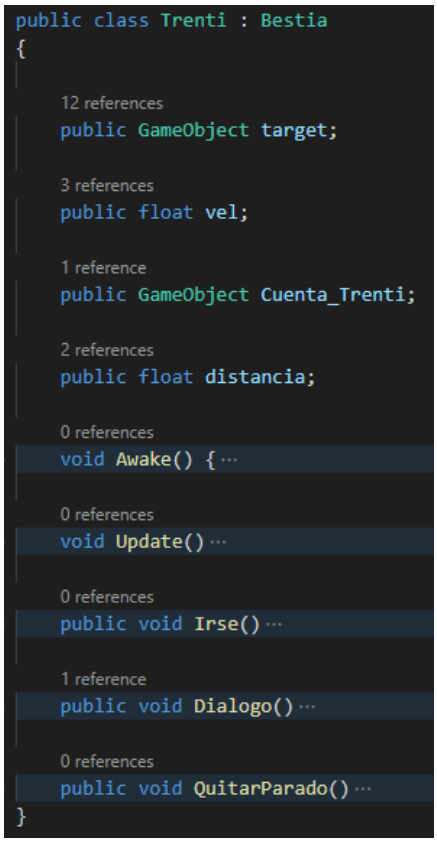


Ilustración 133: Clase Trenti

Como se puede observar, la clase Trenti presenta cuatro atributos, los cuales son los siguientes:

- **target**: hace referencia al objeto que le asocia la clase SpawTrenti al "Trenti" para indicar hacia qué posición debe moverse.
- **vel:** este valor integer define la velocidad a la que se mueve el "Trenti".
- **Cuenta_Trenti**: hace referencia al cartel que aparecerá cada vez que el usuario recoja un "Trenti". El cartel mostrará cuantos "Trentis" tiene recogidos el usuario y cuantos le falta por recoger.
- **distancia**: es la distancia que hay entre la posición del objeto Trenti y el "target" que tiene asociado.

Ya conocidos los atributos de la clase, hay que ver como son empleados mediante los distintos métodos que tiene la clase Trenti:

- **Awake**: inicializa las variables vistas anteriormente y las que hereda de la clase bestia.
- **Update**: se encarga de ejecutar las funciones DistanciaRender y MirarJugador que hereda de la clase bestia, calcular la distancia actual entre el objeto Trenti y la variable "target" y, en caso de que no se haya interactuado con el "Trenti", se encargará de calcular su desplazamiento aplicando la variable "vel" para manejar su velocidad.

- **Irse**: esta función se ejecuta al finalizar un bloque de texto del Flowchart del objeto Trenti que tenga en el nombre "Respuesta". La función crea una instancia del cartel "Cuenta_Trenti" que desaparecerá a los tres segundos de ser creado y destruye tanto al objeto Trenti como el "target" asociado al mismo.

  Cabe mencionar que el objeto Cuenta_Trenti tiene la clase **MensajeAnalisis** (ver anexo 1 apartado 22) asociada que se encarga de maneja que mensaje se verá en dicho cartel. Para esta clase se llamará a la función CuentaTrenti que sacará un mensaje que indique cuantos "Trentis" hay en total y cuantos ha recogido el usuario.

- **Dialogo**: esta función se ejecuta cuando el usuario interactúa con el "Trenti". El método llama a la función ComenzarDialogo y en función de la misión en la que se encuentre el usuario, ejecutará un bloque aleatorio asociado a un comentario triste (si al usuario aun no le han asignado la misión de recoger a los "Trentis") que tiene en su nombre la palabra "Comentario" o un bloque aleatorio asociado a un comentario alegre (si el usuario tiene la misión de recoger "Trentis") que tiene en su nombre la palabra "Respuesta". En función de en qué misión esté también activará la animación "happy" o "cry" del árbol de animaciones.

Ilustración 134: Flowchart con los bloques de texto en Trenti

- **QuitarParado**: se encarga de terminar el dialogo del "Trenti" mediante la función TerminarDialogo y desactivando la animación "cry".

Musgosu

El "Musgosu" es la bestia objetivo durante la misión número cuatro. Esta bestia se encarga de desbloquear la acción "Dejar Cebo" en el listado de acciones del usuario.

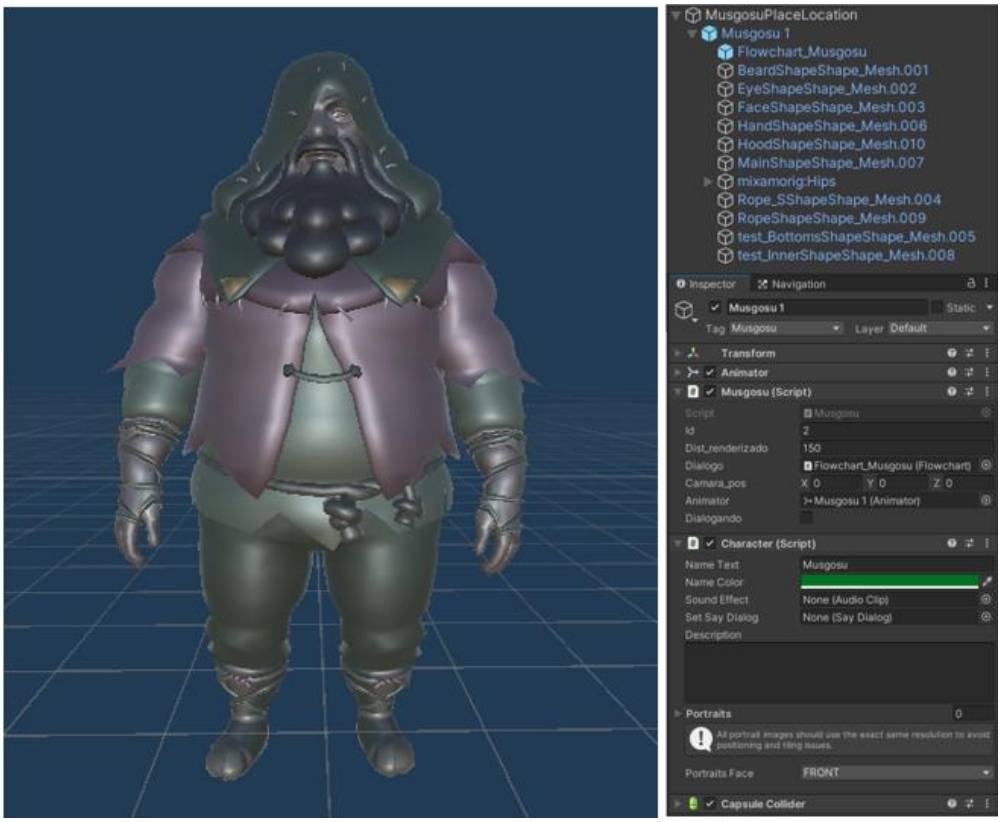


Ilustración 135: Prefab Musgosu

Al igual que con el Prefab TrentiBoss, el Prefab Musgosu es hijo de un objeto vacío llamado MusgosuPlaceLocation, el cual tiene asociado el script PlaceAtLocation para poder posicionarse en una ubicación del mundo real mediante las coordenadas GPS.

Esta decisión se debe al mismo problema que había con “TrentBoss”, el objeto Musgosu aparecía demasiado arriba en el eje Y, es por eso por lo que se ha decidido poner el Prefab Musgosu como hijo y modificarle la posición en el eje Y con respecto a la posición del objeto padre MusgosuPlaceLocation para corregir la desviación.

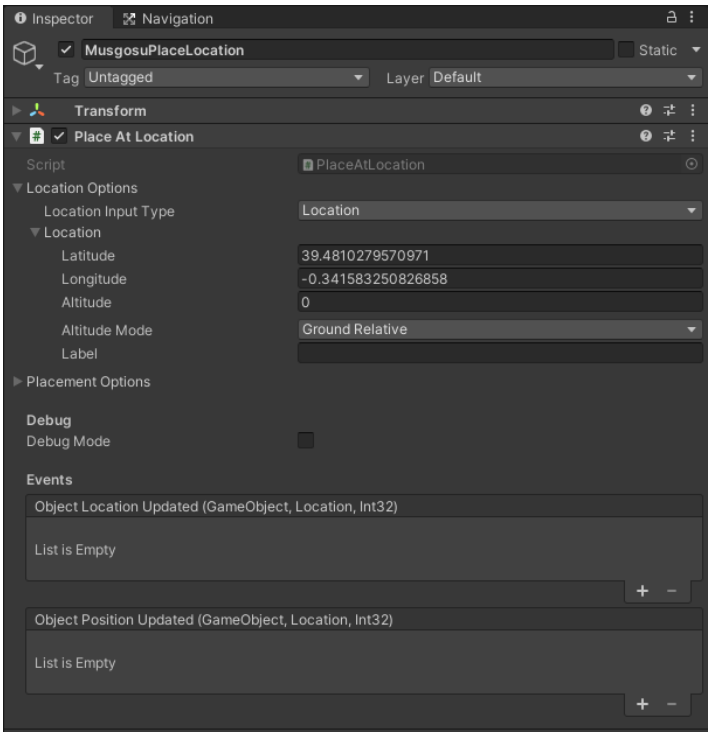


Ilustración 136: Elementos del objeto MusgosuPlaceLocation

El árbol de animaciones del "Musgosu" solo consta de dos animaciones: "Breathing idle" que es la animación predefinida del "Musgosu" cuando está quieto, y "Talking" (hablando) que se activa para que haga un determinado gesto mientras interactúa con el usuario.

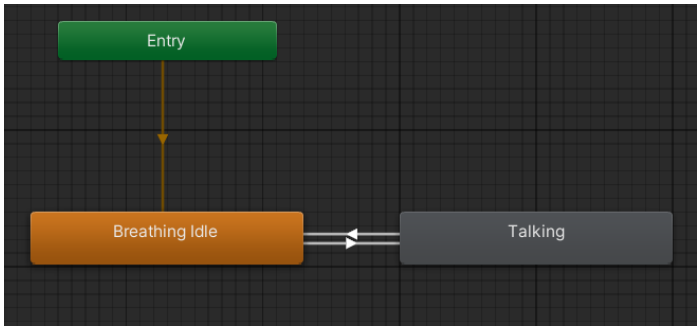


Ilustración 137: Árbol de animaciones de Musgosu

Al igual que las bestias anteriores, el "Musgosu" tiene asociada la clase **Musgosu** (ver anexo 1 apartado 15) que se encarga de controlar el comportamiento del objeto.

```
public class Musgosu : Bestia
{

    0 references
    private void Awake() {···

    0 references
    void Update()···

    1 reference
    public void Dialogo()···

    0 references
    public void SiguienteMision()···

    0 references
    public void Gesticular()···

    0 references
    public void NoGesticular()···
}
```

Ilustración 138: Clase Musgosu

Esta clase no contiene ningún atributo. Sin embargo, emplea una serie de métodos para controlar el funcionamiento del "Musgosu":

- **Awake**: inicializa las variables que hereda de la clase bestia.
- **Update**: ejecuta las funciones DistanciaRender y MirarJugador heredadas de la clase Bestia.
- **Dialogo**: este método se ejecuta cuando el usuario interactúa con el "Musgosu". Si el usuario se encuentra en la misión cuatro, se reproducirá el bloque "Dialogo_Musgosu" con el que se desbloqueará la acción "Dejar Cebo", en caso contrario reproducirá el bloque "No_Preparado".

Ilustración 139: Flowchart con bloques de texto de Musgosu

- **SiguienteMision**: esta función se ejecuta al terminar de reproducirse el texto del bloque "Dialog_Musgosu". La función añade la acción "Dejar Cebo" en el listado de acciones mediante el método AddEntradaAccion de la clase Inventario y carga la siguiente misión empleando la función Cambiar_Mision de la clase Misiones_Controller. Tras esto, el objeto Musgosu es destruido.

```
public void SiguienteMision()
{
    GameObject.FindGameObjectWithTag("Inventario").GetComponent<Inventario>().AddEntradaAccion(2);
    PlayerPrefs.SetInt("registro_PonerCebo",1);
    FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
    Destroy(this.gameObject);
}
```

Ilustración 140: Función SiguienteMision de la clase Musgosu

- **Gesticular**: activa la animación "Talking".
- **NoGesticular**: desactiva la animación "Talking".

Cuelebre

El "Cuelebre" es la bestia que custodia un cofre que será el objetivo de la misión número cinco y que desbloquea la acción "Cruz Santa Marta". Mientras el "Cuelebre" se encuentre cerca del cofre será imposible para el usuario acceder a este y obtener la acción que le ayudará a derrotar a la "Tarasca". Es por ello por lo que el usuario deberá emplear la acción "Dejar Cebo" para colocar un cebo lo suficientemente lejos del "Cuelebre" y así hacer que éste abandone la zona y deje el cofre desprotegido para que el usuario lo abra y desbloque la última acción.

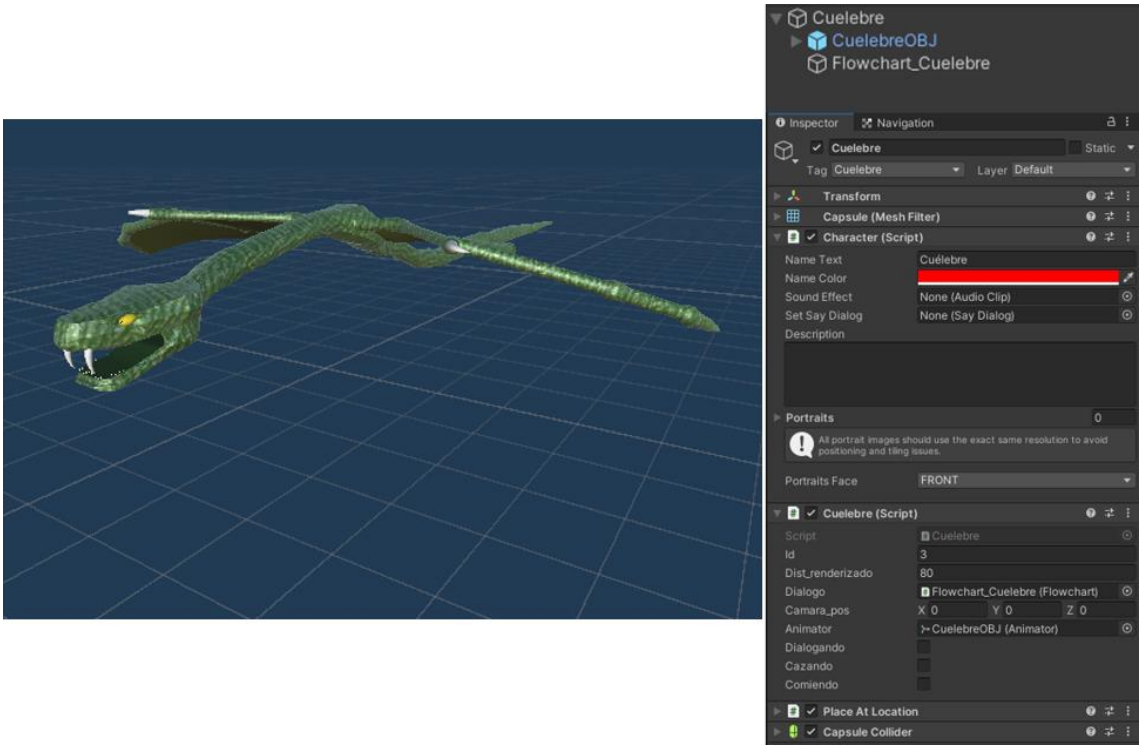


Ilustración 141: Prefab de Cuelebre

El árbol de animaciones del "Cuelebre" consta de dos animaciones: "Idle" que es la animación predeterminada y "fly" (volar) que es la animación que se emplea cuando el objeto Cuelebre se desplaza.

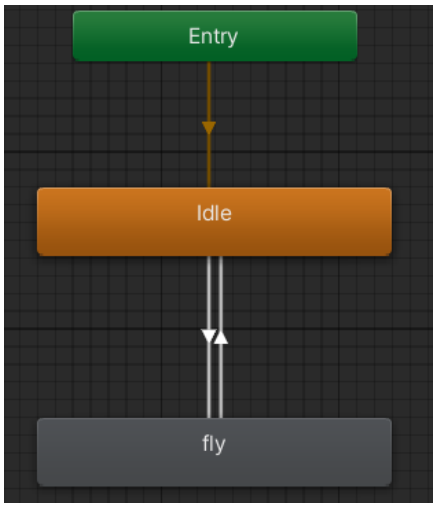


Ilustración 142: Árbol de animación del objeto Cuelebre

El objeto Cuelebre tiene asociado la clase **Cuelebre** (ver anexo 1 apartado 16) que se encarga de controlar el comportamiento del "Cuelebre".

```
public class Cuelebre : Bestia
{

    5 references
    private float vel;
    5 references
    public bool cazando;
    3 references
    public bool comiendo;
    8 references
    private Transform pos_cofre;
    7 references
    private Transform pos_Cebo;
    3 references
    private Vector3 dir_ida;
    3 references
    private Vector3 dir_vuelta;

    0 references
    private void Awake() {···

    0 references
    void Update()···

    1 reference
    public void SeguirCebo()···

    1 reference
    public void Dialogo()···
}
```

Ilustración 143: Clase Cuelebre

Los atributos que almacena la clase Cuelebre a parte de los que hereda de la clase Bestia son los siguientes:

- **vel**: indica la velocidad de desplazamiento del objeto.

- **cazando**: un atributo booleano que informa si el “Cuelebre” está abandonando su posición para ir en busca del objeto “Cebo”.
- **comiendo**: un atributo booleano que informa si el “Cuelebre” ha llegado a la posición del “Cebo” y se lo está comiendo.
- **pos_cofre**: el siguiente atributo indica la posición en la que se encuentra el objeto Cofre en el mundo digital.
- **pos_Cebo**: el siguiente atributo indica la posición en la que se encuentra el objeto Cofre en el mundo digital.
- **dir_ida**: es el vector de la dirección que debe tomar el objeto “Cuelebre” para llegar a la posición del “Cebo” desplegado por el usuario.
- **dir_vuelta**: es el vector de la dirección que debe tomar el objeto “Cuelebre” para regresar a la posición del objeto Cofre tras haber devorado el “Cebo” desplegado por el usuario.

A continuación, se presentan los métodos empleados para controlar el funcionamiento del objeto Cuelebre:

- **Awake**: inicializa la clase dándole el valor adecuado a los atributos “animator” (con el componente Animator del objeto Cuelebre), “pos_cofre” (con la posición del objeto Cofre que hay en la escena), “dialogo” (con el componente Flowchart del objeto Cuelebre), “cazando” (dándole el valor false) y “vel” (iniciándola con el valor 5).
- **Update**: esta función se ejecuta durante cada frame que la clase Cuelebre permanezca activa, ejecutando los métodos DistanciaRender y MirarJugador que hereda de la clase Bestia. A parte de esto, se encarga de controlar si se tiene la posición de un “Cebo” con la variable “pos_Cebo”. En caso de que esa variable no esté vacía, el objeto Cuelebre se moverá siguiendo la dirección del vector “dir_ida” hasta llegar a dicha posición activando la animación “fly”. En cuanto llegue a la posición del “Cebo”, se desactivará la animación “fly” y el “Cuelebre” se mantendrá en esa posición mientras siga existiendo el objeto “Cebo”, durante ese tiempo la variable “comiendo” tendrá el valor true. Una vez haya desaparecido el objeto Cebo, se volverá a activar la animación “fly” y el objeto Cuelebre regresará a la posición del cofre siguiendo la dirección del vector “dir_vuelta”. Con el “Cuelebre” habiendo regresado a su posición inicial, la variable “cazando” volverá al valor false.
- **SeguirCebo**: este método lo ejecuta el objeto Cebo (tal y como se ve en la imagen 106) en la función Update. De esta forma, la clase Cuelebre va recogiendo constantemente la posición del objeto Cebo almacenándola en la variable “pos_Cebo”, mantiene el valor de la variable “cazando” a true y calcula los vectores de “dir_ida” y “dir_vuelta”. Con estos cambios se podrá entrar en el if de la función Update de la clase Cuelebre permitiendo al objeto empezar a moverse en dirección al “Cebo”.

- **Dialogo**: este método se ejecuta cuando el usuario interactúa con el "Cuelebre", en caso de que la bestia no tenga la variable "cazando" con el valor true, se ejecutará el único bloque del Flowchart que tiene el objeto Cuelebre.

Ilustración 144: Flowchart con bloques de texto del objeto Cuelebre

En este apartado, cabe mencionar también el funcionamiento de objeto Cofre, ya que es un elemento de vital importancia para la aplicación y está relacionado directamente con el comportamiento de "Cuelebre".

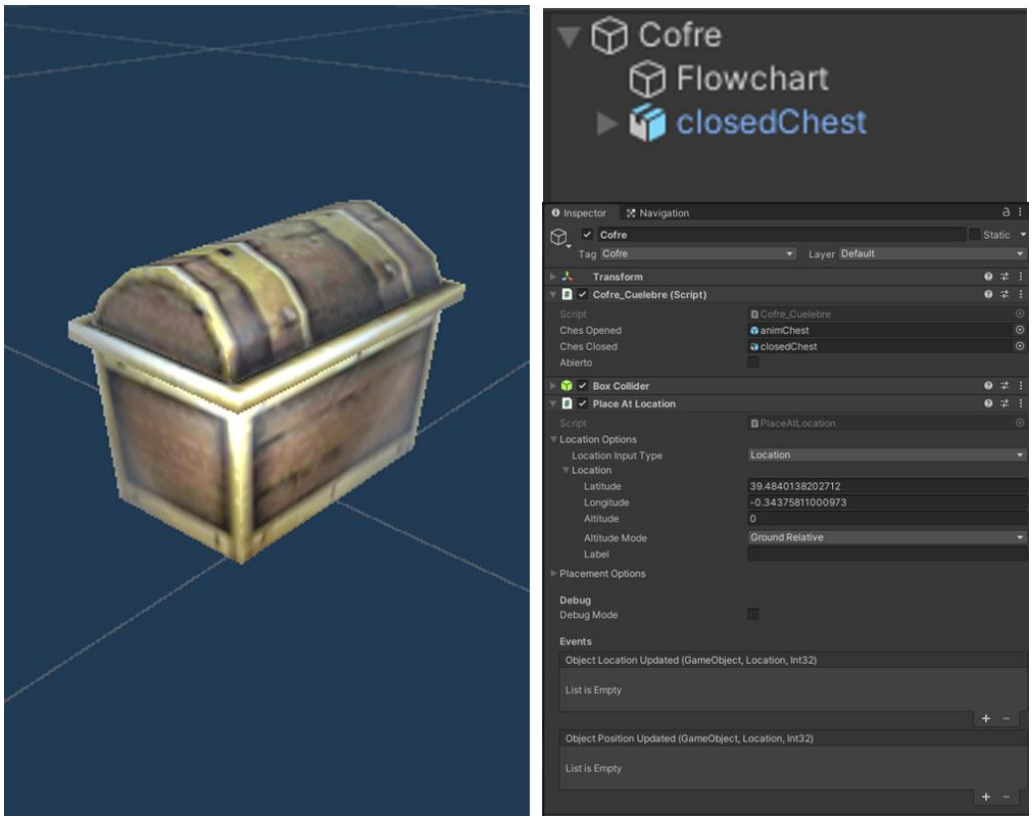


Ilustración 145: Prefab de Cofre

Como se puede ver en la imagen 145, el Prefab Cofer tiene asociado el script que hace referencia a la clase **Cofre_Cuelebre** (ver anexo 1 apartado 17).

```
public class Cofre_Cuelebre : MonoBehaviour
{
    2 references
    private float dist_culebre;

    3 references
    private GameObject cuelebre;

    1 reference
    public GameObject chesOpened;

    7 references
    public GameObject chesClosed;

    4 references
    private bool custodiado;

    6 references
    public bool abierto;

    4 references
    private Flowchart dialogo;

    0 references
    private void Awake() {···

    0 references
    void Update()···

    1 reference
    public void Abrir()···

    0 references
    public void Activar_Cruz()···
}
```

Ilustración 146: Clase Cofre_Cuelebre

Los atributos que presenta la clase son los siguientes:

- **dist_cuelbre**: esta variable guarda la distancia que hay entre el objeto Cofre y el objeto Cuelebre.
- **cuelebre**: esta variable hace referencia al objeto Cuelebre.
- **chesOpened**: esta variable guarda en objeto openedChes que es el objeto 3D que representa el cofre después de haber sido abierto por el usuario.

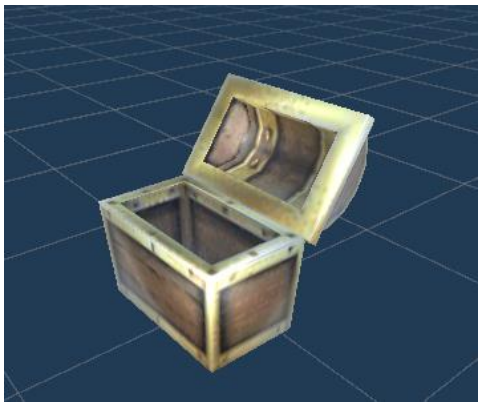


Ilustración 147: Objeto predefinido openedChes

- **chesClosed**: esta variable guarda en objeto closedChes que es el objeto 3D que representa el cofre antes de haber sido abierto por el usuario. La imagen de este objeto puede verse en la ilustración 145.

- **custodiado**: se trata de una variable booleana que indica si el cofre está siendo custodiado por el "Cuelebre" y por tanto es accesible para el usuario.
- **abierto**: se trata de un variable booleana que indica si el cofre ya ha sido abierto por el usuario.
- **dialogo**: esta variable guarda el Flowchart del objeto Cofre.

Para controlar el funcionamiento del objeto Cofre, se emplean los siguientes métodos:

- **Awake**: inicializa las variables "cuélebre" con el objeto Cuelebre que hay en la escena, "custodiado" a true, "abierto" a false y "dialogo" con el componente Flowchart asociado al objeto Cofre.
- **Update**: cambia el valor de la variable "custodiado" a false en caso de que el objeto Cuelebre se aleje y lo devuelve a true cuando regresa. También maneja que el objeto Cofre solo se renderice cuando el jugador se encuentre a una determinada distancia.
- **Abrir**: esta función se ejecuta cuando el usuario interactúa con el cofre, lo cual solo puede suceder si el objeto Cuelebre no se encuentra cerca del cofre. En caso de que el cofre no esté abierto, se ejecutará el bloque del Flowchart "Abierto", se cambiará el valor de la variable "abierto" a true y se cambiará el objeto de closedChes por openedChes para que el usuario vea el cofre abierto. En caso de ya estar abierto, simplemente se ejecutará el bloque del Flowchart "Vacio".

Ilustración 148: Flowchart con los bloques de texto del objeto Cofre

- **Activar_Cruz**: esta función se ejecuta al finalizar el texto del bloque "Abierto" del Flowchart. Se encarga de añadir a la lista de acciones la acción "Cruz Santa Marta" mediante la función AddEntradaAccion de la clase Inventario, cambiar el valor del PlayerPref "registro_Cruz" a 1 y cargar la siguiente misión mediante el método Cambiar_Mision de la clase Misiones_Controller.

Tarasca

La "Tarasca" es el enemigo final en esta experiencia de realidad aumentada. Se trata del villano dentro de la historia de la aplicación y a quién el usuario debe derrotar para completar la aventura. Para ello tendrá que desbloquear la acción "Cruz Santa Marta" emplearla contra la bestia para que esta sea destruida de la escena.

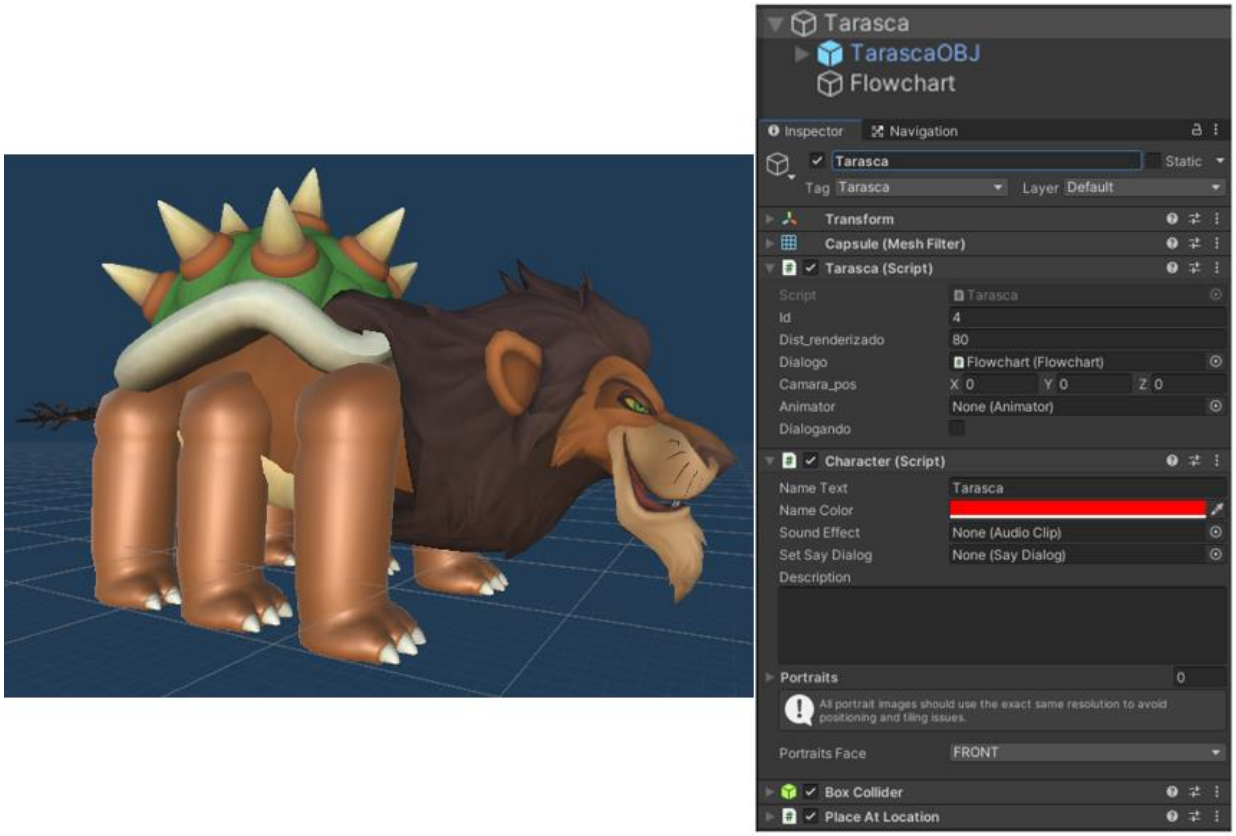


Ilustración 149: Prefab de Tarasca

En este caso, pese a tener al igual que las demás bestias un elemento PlaceAtLocation para indicar en que coordenadas GPS del mundo real debe aparecer, la “Tarasca” no tiene asociada ningún elemento Animator. Esto se debe a que debido a la complegidad del modelo de la Tarasca, la creación de un esqueleto para el mismo era una tarea complicada en la que habría que invertir mucho tiempo. Es por ello que esta es la única bestia que no posee un árbol de animaciones.

Para controlar el comportamiento del objeto Tarasca, a este se le asocia la clase **Tarasca** (ver anexo 1 apartado 18) que se encarga de cumplir esta función mediante los múltiples métodos que posee.

```
public class Tarasca : Bestia
{
    0 references
    void Start() ···

    0 references
    void Update() ···

    1 reference
    public void Dialogo() ···

    1 reference
    public void Amenazado() ···

    0 references
    public void Eliminacion() ···
}
```

Ilustración 150: Clase Tarasca

- **Start**: se encarga de inicializar la variable dialogo heredada de la clase Bestia añadiéndole como valor el Flowchart asociado al objeto Tarasca.
- **Update**: se encarga de ir ejecutando las funciones DistanciaRender y MirarJugador heredadas de la clase Bestia.
- **Dialogo**: esta función se ejecuta cuando el usuario emplea la acción "Interactuar" sobre el objeto Tarasca. La función ejecuta el método ComenzarDialogo y el bloque de Flowchart "Dialogo_Tarasca".
- **Amenazado**: esta función se ejecuta cuando el rayo generado por el Raycast del objeto CruzSantaMarta colisiona con el objeto Tarasca (tal y como se aprecia en la clase CruzController de la imagen 111). La función ejecuta el método ComenzarDialogo y el bloque de Flowchart "Derrota".

Ilustración 151: Flowchart con los bloques de texto del objeto Tarasca

- **Eliminacion**: esta función se ejecuta al final de bloque de Flowchart "Derrota". La función carga la siguiente misión mediante la función Cambiar_Mision de la clase Misión_Controller y destruye el objeto Tarasca.

Capítulo 5. PRUEBAS Y RESULTADOS

Una vez terminada la aplicación se ha puesto a prueba con un grupo de usuarios con el fin de concluir si la funcionalidad de la aplicación y la carga de trabajo que ejerce sobre los usuarios es la adecuada.

Se ha escogido una ubicación en donde colocar mediante coordenadas GPS los distintos objetos digitales vistos en el capítulo 4.3.7 y se ha llevado a los usuarios para que prueben la aplicación desarrollada.

Tras esto, se han realizado dos tipos de pruebas que son comunes durante los testeos de las aplicaciones. En primer lugar, se ha empleado el Sistema de Escalas de Usabilidad [33] para medir la usabilidad de la aplicación, después se ha realizado la segunda que ha utilizado el método NASA TLX [34] para evaluar el nivel de carga de trabajo que la aplicación ejerce sobre los usuarios.

Se es consciente que, dado que el proyecto realizado tiene fines educativos, se debería haber realizado algún test que probará como de efectivo es el enfoque pedagógico dado. Más por la falta de recursos ha sido imposible efectuar tests que valorasen este aspecto de la aplicación.

Se han realizado estos test a un total de 22 usuarios de edades comprendidas entre los 20 y 26 años. El grado de conocimiento en el uso de aplicaciones de realidad aumentada de carácter lúdico variaba, habiendo usuarios que conocían a la perfección como emplear este tipo de aplicaciones y otros usuarios para los que era su primera experiencia con una aplicación de realidad aumentada. Esto ha permitido que los resultados de los test sean muy variados, lo cual se ha considerado adecuado debido a que de esta forma se han podido obtener opiniones de usuarios ya versados en el uso de aplicaciones de realidad aumentada y de principiantes que desconocían completamente el medio, lo que ha enriquecido de sobre manera las conclusiones obtenidas de los resultados de los tests.

5.1 Preparación para el testeo de la aplicación

Para realizar la prueba del funcionamiento de la aplicación, ha sido necesario escoger una localización en donde la realización de la prueba se desarrollase de forma segura y con el menor número de contratiempos e inconvenientes posibles.

De entre todas las localizaciones estudiadas, se ha escogido los Jardines del Real de la ciudad de Valencia porque reúne las características óptimas para probar la aplicación como son:

- Se trata de un lugar cerrado delimitado por un vallado que permite identificar fácilmente los límites de la zona.
- Es un emplazamiento público de libre acceso durante un amplio horario para los usuarios.

- Se han detectado algunos problemas de desviación con el posicionamiento GPS de los objetos digitales. Es por ello que este lugar resulta muy ventajoso por el hecho de que esta ubicación ofrece una zona muy amplia que permite que dicha posible desviación posicione al objeto en un lugar que siga siendo accesible para el usuario.
- La zona está cerrada al tráfico, lo que elimina la posibilidad de cualquier accidente con vehículos.
- Al ser un parque, los usuarios se encontrarán en un entorno rodeado de árboles y vegetación, lo que ayuda en la inmersión de la experiencia que ofrece la aplicación, dado el carácter silvestre y relacionado con la naturaleza que tienen los seres mitológicos que aparecen en la aplicación.

La siguiente imagen muestra rodeada por el círculo rojo la zona del parque que se ha utilizado durante la prueba.

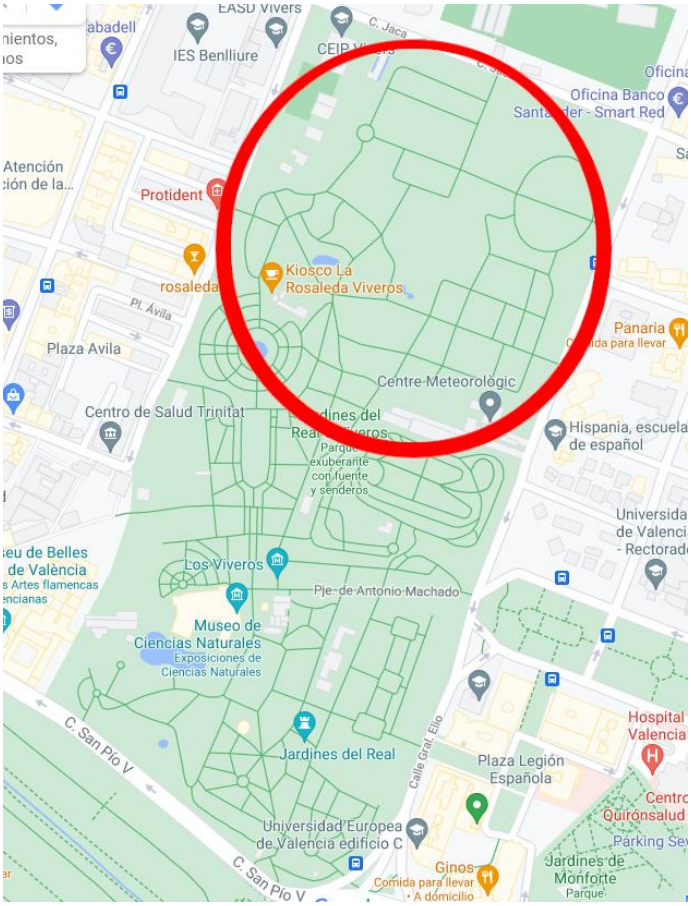


Ilustración 152: Zona empleada de los Jardines Reales para la prueba

Finalmente, una vez decidido en donde se van a realizar las pruebas, se ha de elegir las coordenadas GPS en donde se deben posicionar los distintos objetos digitales y pasárselas a los scripts PlaceAtLocation que tiene asociados cada uno de los objetos vistos en el apartado 4.3.7. La siguiente tabla muestra los distintos objetos digitales y las coordenadas en donde han sido posicionados.

Tabla 4: Coordenadas de los objetos digitales

	Coordenadas GPS	
Objetos digitales	**Latitud**	**Longitud**
TrentiBossPlaceLocation	39,4824839012663	-0.365856695152047
Spawn_Trentis (1)	39,4830180017438	-0.366356645823942
Spawn_Trentis (2)	39,4826368379455	-0.366548700045371
MusgosuPlaceLocation	39,4812774472605	-0.368239960979309
Cuelebre	39,4809376659974	-0.367756405849804
Cofre	39,4809376659974	-0.367756405849804
Tarasca	39,4819009670429	-0.367398580579874

5.2 Sistema de Escalas de Usabilidad

El sistema de escalas de usabilidad se trata de un test conocido como SUS (System Usability Scale) que es empleado para medir la sencillez de uso de una aplicación, indicando lo cómodo que les resulta a los usuarios el empleo de dicha aplicación.

El test está compuesto de 10 afirmaciones que los usuarios deberán valorar con una escala del 1 al 5 cada una de ellas. Dando el valor 1 cuando el usuario está en completo desacuerdo con la cuestión y un valor de 5 cuando el usuario está totalmente de acuerdo.

Una vez obtenidas las respuestas de los usuarios, se obtendrá un valor para cada afirmación en función de esas respuestas dadas: las afirmaciones impares tendrán un valor equivalente a la respuesta del usuario restándole 1, las afirmaciones pares tendrán un valor de 5 restándoles el valor de la respuesta dada.

Con los valores obtenidos, de cada una de las afirmaciones, se suma el total de ellos y se le multiplica por el coeficiente 2’5.

Las afirmaciones que se plantearon a los usuarios son las siguientes:

- Creo que usuaria este sistema frecuentemente.
- Encuentro este sistema innecesariamente complejo.
- Creo que el sistema fue fácil de usar.
- Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar este sistema.
- Las funciones de este sistema están bien integradas.
- Creo que el sistema es muy inconsistente.
- Imagino que la mayoría de la gente aprendería a usar este sistema de forma muy rápida.
- Encuentro que el sistema es muy difícil de usar. Me siento confiado al usar este sistema.
- Necesité aprender muchas cosas

La hoja del test que rellenaron los usuarios se expone en el anexo

Tabla 5: Resultados del test SUS

	Usuario 1		Usuario 2		Usuario 3		Usuario 4		Usuario 5		Usuario 6		Usuario 7		Usuario 8	
Pregunta 1	3	2	3	2	3	2	3	2	1	0	4	3	2	1	4	3
Pregunta 2	1	4	2	3	1	4	1	4	1	4	1	4	1	4	1	4
Pregunta 3	5	4	4	3	4	3	4	3	2	1	4	3	5	4	4	3
Pregunta 4	1	4	1	4	1	4	1	4	3	2	1	4	1	4	2	3
Pregunta 5	4	3	4	3	4	3	5	4	4	3	2	1	5	4	5	4
Pregunta 6	2	3	1	4	3	2	2	3	1	4	2	3	1	4	3	2
Pregunta 7	4	3	5	4	5	4	5	4	4	3	4	3	5	4	4	3
Pregunta 8	2	3	1	4	1	4	2	3	1	4	1	4	1	4	1	4
Pregunta 9	5	4	5	4	4	3	5	4	3	2	5	4	5	4	5	4
Pregunta 10	1	4	1	4	1	4	1	4	3	2	1	4	1	4	1	4
Total		34		35		33		35		25		33		37		34
Resultado		85		87,5		82,5		87,5		62,5		82,5		92,5		85

	Usuario 9		Usuario 10		Usuario 11		Usuario 12		Usuario 13		Usuario 14		Usuario 15		Usuario 16	
Pregunta 1	3	2	5	4	3	2	3	2	1	0	4	3	4	3	2	1
Pregunta 2	2	3	1	4	1	4	2	3	2	3	2	3	2	3	1	4
Pregunta 3	5	4	5	4	2	1	4	3	3	2	4	3	4	3	4	3
Pregunta 4	1	4	1	4	1	4	2	3	2	3	2	3	2	3	1	4
Pregunta 5	3	2	5	4	2	1	4	3	3	2	3	2	5	4	4	3
Pregunta 6	2	3	1	4	1	4	2	3	1	4	3	2	1	4	1	4
Pregunta 7	5	4	3	2	5	4	5	4	3	2	5	4	4	3	4	3
Pregunta 8	1	4	1	4	1	4	2	3	3	2	1	4	1	4	1	4
Pregunta 9	5	4	5	4	3	2	4	3	3	2	4	3	5	4	4	3
Pregunta 10	1	4	2	3	1	4	5	0	2	3	1	4	1	4	1	4
Total		34		37		30		27		23		31		35		33
Resultado		85		92,5		75		67,5		57,5		77,5		87,5		82,5

	Usuario 17		Usuario 18		Usuario 19		Usuario 20		Usuario 21		Usuario 22		Media
Pregunta 1	3	2	3	2	3	2	4	3	3	2	5	4	
Pregunta 2	2	3	4	1	1	4	2	3	1	4	2	3	
Pregunta 3	4	3	4	3	4	3	5	4	5	4	4	3	
Pregunta 4	3	2	4	1	1	4	1	4	1	4	3	2	
Pregunta 5	3	2	3	2	4	3	5	4	4	3	3	2	
Pregunta 6	4	1	1	4	1	4	1	4	1	4	2	3	
Pregunta 7	2	1	4	3	4	3	3	2	4	3	5	4	
Pregunta 8	2	3	2	3	2	3	1	4	1	4	1	4	
Pregunta 9	3	2	2	1	3	2	5	4	5	4	4	3	
Pregunta 10	2	3	4	1	1	4	1	4	1	4	1	4	
Total		22		21		32		36		36		32	
Resultado		55		52,5		80		90		90		80	78,977273

La tabla anterior muestra las distintas respuestas dadas por los usuarios en cada pregunta del test. La primera columna de cada usuario muestra la respuesta del usuario dada en el test, mientras que la segunda es la respuesta calculada para obtener finalmente el resultado del test en la última fila realizando los cálculos mostrados previamente.

En base a las respuestas y resultados obtenidos de los usuarios, se ha podido realizar una gráfica indicando los tipos de resultados obtenidos:

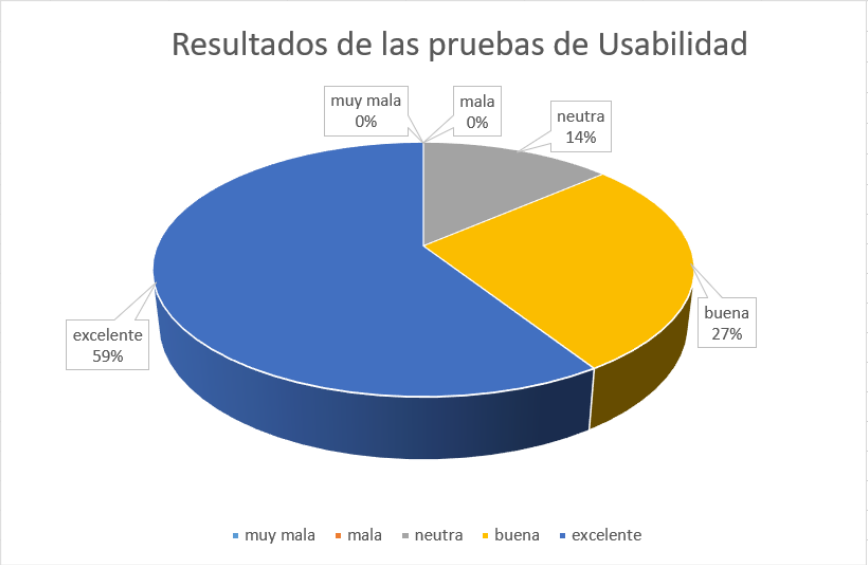


Ilustración 153: Resultados de las pruebas de usabilidad

Los resultados de los usuarios se han dividido en cinco grupos en función del valor de los mismos. Esta clasificación puede ser vista en la siguiente tabla:

Tabla 6: Clasificación de los valores obtenidos en el test SUS

Valor del resultado	**Clasificación**
0 a 20	muy mala
20 a 40	mala
40 a 60	neutra
60 a 80	buena
80 a 100	excelente

5.3 Método NASA TLX

El método Nasa TLX mide el nivel de carga de trabajo que una aplicación, sistema o actividad tiene sobre los usuarios.

La prueba se realiza en dos etapas:

1ª Tiene como objetivo obtener el nivel de importancia que el usuario le da a los conceptos de demanda mental, demanda física, demanda temporal, rendimiento, esfuerzo y frustración, como causa de carga de trabajo. Para ello se comparan cada uno de los conceptos entre ellos formando 15 comparaciones binarias entre todos. El usuario deberá escoger, para cada combinación, aquel concepto que cree que ofrece una mayor carga de trabajo.

En función del número de veces que se haya escogido cada concepto, se obtendrá un peso asociado al concepto equivalente al número de veces que este haya sido seleccionado. Dicho peso variará entre 0 (en caso de que el concepto no haya sido escogido ninguna vez) y 5 (si el concepto ha sido escogido en todas las comparaciones en donde ha aparecido).

La hoja de pruebas que respondieron los usuarios para realizar esta fase del test con las distintas comparaciones de conceptos se muestra en el anexo X.

2ª En esta fase los usuarios deberán haber completado previamente la aplicación. Después tendrán que marcar la cantidad de carga de trabajo que consideran que han sufrido durante el uso de la aplicación debido a cada uno de los seis conceptos mencionados previamente en una escala con valores de 0 a 100 divididos en segmentos de 5 unidades.

La hoja que los usuarios han rellenado para esta fase del test puede ser vista en el anexo X.

Con los datos recopilados de los pesos de la primera fase y las escalas de la segunda, se calcula el índice de carga de trabajo global mediante la suma de cada uno de los valores de las escalas multiplicado por su respectivo peso y dividido todo entre 15.

Tabla 7: Resultados del test NASA TLX

	D. Mental		D. Física		D. Temporal		Rendimiento		Esfuerzo		Frustación		Índice Globla	Índice Global
	Peso/Valor	Puntuación	Peso/Valor	Posición	Peso/Valor	Posición	Peso/Valor	Posición	Peso/Valor	Posición	Peso/Valor	Puntuación		33,90909091
Usuario 1	2	20	0	50	3	70	1	25	4	5	5	15	24,66666667	
Usuario 2	2	40	2	40	3	25	2	15	5	5	1	5	19,66666667	
Usuario 3	2	65	0	25	4	15	4	45	3	40	3	75	47,66666667	
Usuario 4	1	25	0	5	5	20	2	5	3	10	4	10	13,66666667	
Usuario 5	2	85	5	85	4	5	2	55	0	90	2	40	53,66666667	
Usuario 6	1	70	2	80	0	15	5	0	3	80	3	5	32,33333333	
Usuario 7	1	15	0	15	5	85	3	0	2	15	4	0	31,33333333	
Usuario 8	2	55	1	30	4	65	2	10	1	50	4	80	52,66666667	
Usuario 9	3	50	4	35	0	40	1	10	2	35	5	10	28	
Usuario 10	2	20	0	60	1	50	3	15	3	30	5	10	18,33333333	
Usuario 11	1	20	3	40	4	10	1	30	1	25	5	15	20,66666667	
Usuario 12	1	10	0	35	4	60	3	30	2	25	5	30	36	
Usuario 13	1	15	1	50	1	50	3	50	3	25	5	35	34,33333333	
Usuario 14	2	20	0	25	4	50	3	70	1	30	5	30	42	
Usuario 15	2	15	1	10	1	50	4	35	2	10	4	40	27,33333333	
Usuario 16	5	35	0	15	4	65	3	0	1	20	2	5	31	
Usuario 17	0	60	5	50	2	20	1	30	3	50	4	40	42	
Usuario 18	0	65	5	20	2	70	1	65	3	75	4	50	48,66666667	
Usuario 19	1	35	0	30	5	0	3	80	2	50	4	40	35,66666667	
Usuario 20	2	45	1	30	3	30	0	35	4	30	5	15	27	
Usuario 21	1	55	0	55	4	35	3	20	2	50	5	10	27	
Usuario 22	2	50	4	70	0	40	5	40	3	60	1	25	52,33333333	

La tabla anterior muestra los resultados de cada usuario durante la prueba NASA TLX, mostrando para cada uno de los seis conceptos dos columnas, una con los pesos asociados a cada concepto en la primera fase del test y la otra con el valor dado en las escalas de la segunda fase. Realizando los cálculos mencionados previamente, se obtiene el índice global de carga de trabajo para cada uno de los usuarios.

Con los resultados obtenidos de los usuarios, se ha realizado la siguiente gráfica:

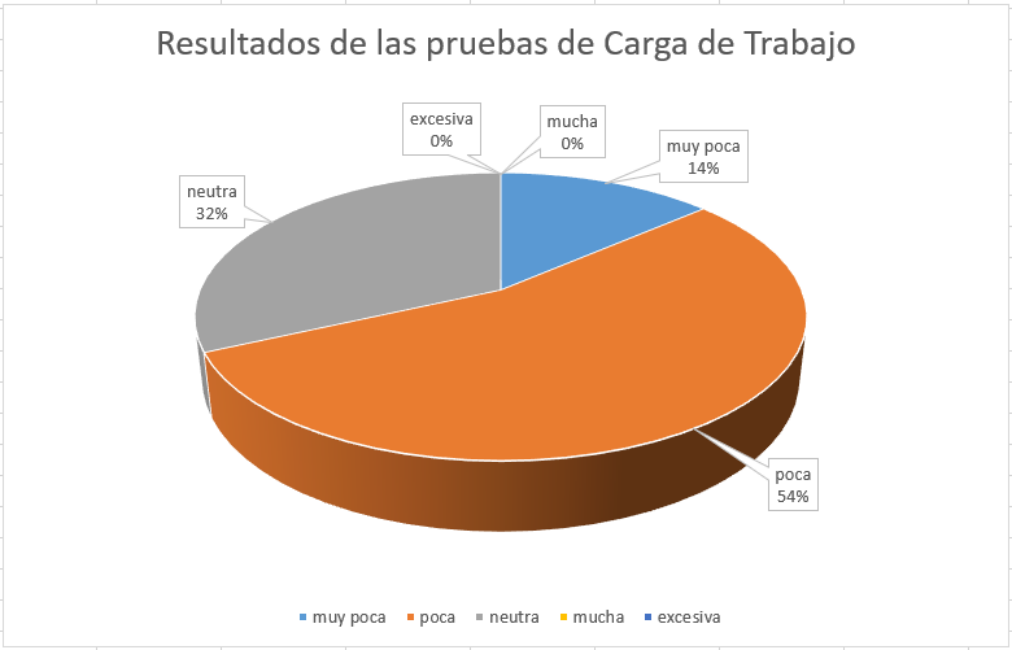


Ilustración 154: Resultados de las pruebas de Carga de Trabajo

Los resultados de los usuarios se han dividido en cinco grupos en función del valor de los mismos como se puede observar en la siguiente tabla:

Tabla 8: Clasificación de los resultados del test NASA TLX

Valor del resultado	Clasificación
0 a 20	muy poca
20 a 40	poca
40 a 60	neutra
60 a 80	mucha
80 a 100	excesiva

5.4 Comentarios de los usuarios

Tras la prueba se les preguntó a los usuarios que era lo que pensaban sobre la aplicación que acababan de usar:

Tabla 9: Opiniones de los usuarios sobre la aplicación

Usuario 1	La app es bastante curiosa, contar una historia mientras te obliga a desplazarte por diferentes lugares es bastante satisfactorio porque te hace querer ir a ese otro lugar que te marca para continuar la historia, aunque a veces habían cosas no muy intuitivas en general me pareció bastante bien hecha.
Usuario 2	Ha sido una experiencia entretenida, pues las pruebas que te ofrecía el propio juego eran divertidas, sobre todo por ver a los personajes en un plano real. Por otra parte si he de añadir que era un poco incómodo ir con la mano en alto y sujetando el móvil, pero sin duda volvería a jugar. Y quizás uno de los defectos destacables fue el posicionamiento de cada uno de los personajes, pues a la vez que te movías, el personaje también lo hacía sin mantenerse éste en un lugar fijo, que intuyo que eso se buscaba. En cuanto a la app, creo que tiene potencial.
Usuario 3	Me gustó mucho la experiencia, era la primera vez que utilizaba un programa de realidad aumentada y me sorprendió, las misiones eran sencillas y entretenidas. Por el contrario el móvil se recalentaba mucho mientras estaba ejecutando el programa y había veces que se paraba o no respondía, tenía que cerrar y volver a abrir. Pero quitando ese pequeño error fue entretenido y divertido.
Usuario 4	Un juego interesante y con potencial. La geolocalización funciona correctamente y la interfaz es muy cómoda de usar. Algunas funciones necesitan de un cuadro de dialogo que explique o recuerde como usarlas correctamente, para los usuarios más novatos. No obstante, se nota el trabajo en el desarrollo del lore y los NPC son visualmente atractivos.
Usuario 5	En cuanto al videojuego, decir que es mi primer contacto con algún juego de este estilo. A pesar de ello no me ha resultado demasiado difícil, gracias a las indicaciones que he ido recibiendo por parte de la pestaña de ayuda que ha sido de gran utilidad. Una vez acabado me ha parecido más sencillo y más corto de lo esperado. Así como último apunte decir que la experiencia ha sido positiva aunque no me acabo de ver usando este tipo de juegos ya que requieren de una superficie bastante amplia y no muy concurrida y aun así me parece no muy difícil tropezarse.
Usuario 6	La aplicación me ha parecido muy original, obviamente falta pulir y mejorar mucho. Pero para una sola persona es obvio que es demasiado trabajo. Habría que corregir o hacer un poco más intuitivo el generador de planos para invocar y corregir los posibles fallos que pueda haber con la brújula. Conclusión: He disfrutado de la experiencia y no me ha resultado nada pesado ni el manejar la aplicación ni el juego en sí. Cosas a mejorar pero tiene mucho potencial latente.
Usuario 7	El programa funcionó sin problema alguno, todo fue muy intuitivo y no hubo momento alguno en el que tuviese dudas sobre qué hacer. La historia es simple, pero adecuada para la longitud del juego. Haciendo una buena experiencia. Lo destacable del juego es el bestiario, que, aunque no necesario para finalizar el juego es una introducción a una mitología desconocida para la mayoría; Anjana como ayudante y el encontrar el villano final antes de estar en el final de la historia ayuda a darle vida al mundo. Como negativas debería comentar la falta de menú de configuración que permita desactivar la música sin desactivar los sonidos, el modelo de Tarasca que no funciona bien como villano, pareciendo más una abominación y por último un problema de los juegos de este género: la demanda temporal. Como conclusión, el programa es intuitivo, el juego es fácil, corto y funciona como pequeña introducción a una mitología poco conocida, pero personalmente no podría dedicarle el tiempo necesario para jugarlo.
Usuario 8	Para mí ha sido una experiencia bastante positiva, ya que en lo personal me pareció una aplicación muy entretenida y divertida. Es una aplicación muy interesante tanto para utilizarla en solitario como para utilizarla en grupos, sacando el lado más “competitivo” de uno mismo sin dejar de ser una experiencia divertida. Las mecánicas del juego son fáciles y

	la propia aplicación te va dando pistas de cómo avanzar si te quedas atascado. En lo personal destacaría todo lo que es la introducción y cómo te introduces en el juego, ya que desde el principio se crea una atmósfera bastante concreta y que te dan ganas de disfrutar del juego. Sin duda es un juego que en algún momento de mi vida tendría en el móvil.
Usuario 9	Ha sido muy entretenido utilizar la app. No tuve ningún problema de uso con mi dispositivo, aunque aún le faltaba un poco de fluidez y cuando te acercabas a la geolocalización de los objetos estos se alejaban de ti y hacía difícil interactuar con ellos y además tenías de caminar más. El botón de ayuda también daba algún fallo. Por lo general resolví los retos fácilmente gracias a que la aplicación era muy intuitiva. A mi parecer tiene potencial. La temática de mitología española es poco conocida, pero con aplicaciones como esta puede volverse más popular utilizando la RA como material pedagógico.
Usuario 10	He tenido una experiencia muy agradable. El juego me ha parecido dinámico ya que haces un objetivo y obtienes una recompensa. Si tienes dudas siempre puedes usar el botón "ayuda" y te explica lo que necesitas hacer para avanzar, lo cual se agradece. Me ha gustado mucho la historia y la información que aportaba cuando identificaba algún individuo. No creo que requiera mucho esfuerzo físico, ya que puedes ir a tu ritmo, ni mental, ya que recibes la "ayuda" antes mencionada. Me ha gustado mucho probarlo y me encantaría ver cómo evoluciona, ya que las misiones eran divertidas y me hacían querer más y más. Lo único que diría de perfeccionar sería el hecho de llevar el móvil en alto. Las flechas se podrían seguir como si te grabaras por delante de los pies para poder ver lo que tienes delante al caminar. O incluso poder poner el móvil en vertical antes que horizontal. Todo lo demás, las interacciones y las ayudas, me han parecido útiles y necesarias.
Usuario 11	Podemos determinar la experiencia completa como una aventura de realidad aumentada muy amena, algunas de las cosas más destacables de la aplicación son la relación espacio-realvirtual. No exige unas grandes habilidades físicas ni una gran implicación a nivel intelectual ya que la propia aplicación te guía. Uno de los puntos más destacables son la interfaz sencilla que ayuda a que esta aventura sea más amena. Por otro lado, es cierto que hay algunos aspectos que podrían pulirse un poco más relacionados con la integración virtual-realidad. Valoraría esta experiencia como positiva, en mi opinión es un juego con un gran potencial abierto a un gran público.
Usuario 12	Como puntos a destacar veo la facilidad de uso de la misma, pues sin mucha ayuda, pude desenvolverme bien durante la experiencia, y la expansión de la narración gracias a la herramienta del bestiario, ya que el folclore ibérico es una temática no muy conocida y esto es un buen refuerzo para aquellos usuarios que quieran ahondar en la inmersión narrativa. Por otra parte, como punto negativo señalaría la sensación de distancia en relación a los seres del juego, que en ocasiones puede resultar confusa, como fue el caso de la Tarasca, pues, durante la experiencia, en el momento de ir a interactuar con ella, parecía estar cerca del visor cuando aún había que recorrer bastante distancia para poder interactuar con ella.
Usuario 13	El juego está muy bien hecho, los personajes son muy bonitos y acertados para el tipo de juego que es, me ha gustado la trama de la historia y los diferentes hechizos que hay. Aun así la aplicación me sacaba del parque donde se realizaba la prueba y tuve que reiniciar varias veces. A modo de conclusión, el juego es muy divertido, creo que a la gente que le gusten los juegos de realidad aumentada podría entretenerle bastante, solo faltaría corregir unos pequeños fallos.

Usuario 14	Esta aplicación me ha parecido algo innovador, ya que mucho se oye hablar de la realidad aumentada y tampoco se aprecia en el día a día. Me ha parecido bastante completa, el simple hecho de salir a un sitio abierto y poder disfrutar del teléfono de una forma distinta a la habitual me fascina ya que esta app junta ejercicio físico ya que hay que desplazarse, ejercicio mental siguiendo los pasos propuestos, y en cierto modo te introduce en una aventura en la que tú mismo eres el protagonista. Algunos de los puntos positivos que he experimentado han sido el menú de ayuda que me ha permitido avanzar cuando tenía dudas sobre cómo hacerlo. La interfaz se veía limpia y era sencilla de utilizar. La música ambiente era capaz de introducirme en el juego. La utilización de objetos me ha parecido buena idea para hacerlo más interesante. Como contras puedo mencionar algún que otro fallo por la localización con el GPS, de forma que la brújula en el juego me sacaba fuera de la zona. Un segundo problema era al acercase a los personajes dinámicos que forman parte de la historia, ya que, a pesar de acercarme a ellos, estos se alejaban. Me parece una idea muy interesante que con un poco más de trabajo puede llegar a ser algo yo mismo llegaría a utilizar para pasar el día. Ha sido toda una experiencia poder utilizarla, ya que me ha sumergido la app hasta el punto de visitar zonas por donde no pasaría normalmente.
Usuario 15	Está muy chula la idea, tiene algunos Bugs, y es mejorable en cuanto a Gameplay (controles, sistema de ayuda...) pero el tema de la mitología está muy guay, el sistema de Quests y buscar seres mitológicos también. En general muy bien.
Usuario 16	Contras -Interfaz poco atractiva -Fallo en la geolocalización -La pantalla del móvil se apaga si no hay actividad (si no tocas la pantalla) -Demasiado texto -Hay que tener mucho cuidado con el entorno para no tener ningún accidente -Problemas a la hora de poner el señuelo (el cerdo) al Cuelebré Pros: -Historia interesante y me he quedado con ganas de conocer más. -Entretenido -Actividades amenas
Usuario 17	No soy público objetivo para la aplicación porque no me gusta estar pendiente del móvil mientras camino. Dicho esto, conocer la mitología de nuestro país es interesante pero faltan cosas por pulir. La localización de las bestias es poco precisa pues cada vez que bloqueas la aplicación/la reinicias pueden cambiar. El diseño de los modelados también debería mejorar para hacer más atractivo el juego.
Usuario 18	Una vez utilizado el juego puedo concluir que es fácil de manejar aunque no hayas tenido experiencias previas con videojuegos de ningún estilo. Me parecen muy interesantes como se han plasmados y la buena calidad de los personajes en especial el de la guía. Sólo hay dos problemas desde mi perspectiva. Uno es que el vocabulario empleado en algunos momentos es demasiado culto sobre todo si tenemos en cuenta que está dirigido a niños/as pequeños. Y por otro lado, el segundo problema que encuentro es que al haber rango de desplazamiento en algún momento puede aparecer algún objetivo en zonas donde hay niños/as jugando y puede generar algún problema ya que vas con un móvil que parece que estés grabando en una zona con menores de edad. Por lo demás me parece un juego bastante entretenido y al que puede jugar todo aquel que quiera.
Usuario 19	La aplicación que se me ha dado a probar presenta un concepto interesante donde se usa la realidad aumentada para contar una historia interactiva. Si bien quizás la geolocalización de la que se sirve presenta algunas desviaciones, en general me parece una buena aplicación, con un diseño accesible para gente acostumbrada a jugar y gente que acabe de empezar

Usuario 20	Se trata de una experiencia interesante que no exige demasiado puesto que es una aventura corta y sencilla. Sin embargo, se pueden predecir algunos elementos debido a la interfaz. También me hubiese gustado utilizar la aplicación con orientación vertical en vez de horizontal. También me gustaría una mayor fluidez en algunos elementos como las desapariciones de personajes de manera más natural en vez de desapariciones tan bruscas. En general la funcionalidad está bien hecha pero se podría pulir más a nivel artístico.
Usuario 21	Me ha parecido una experiencia muy interesante. Como cosas a mejorar destacaría la interfaz de usuario, puesto que en mi opinión las acciones están un poco escondidas. A pesar de esto, la experiencia es muy buena e integra muy bien los elementos de realidad virtual.
Usuario 22	La experiencia que ofrece me parece una premisa genial: utilizando la realidad aumentada y la geolocalización, se consigue que el usuario se sienta atraído por una parte tan importante de la historia como es la mitología. La integración de esta en un escenario real como lo es el parque de Viveros, me parece la mejor elección posible. Como contras, solo cabe destacar algunos ajustes gráficos, y posiblemente un aumento de la dificultad en la parte inicial del juego. En conclusión, el juego me parece una buena idea, que espero que evolucione y mejore.

5.5 Observaciones a los resultados obtenidos en las pruebas

- Viendo los resultados de la tabla 5 y los porcentajes de la imagen 153, se puede concluir que la recepción de la aplicación en cuanto a lo accesible que resultan sus mecánicas para los usuarios ha sido bastante positiva. El valor medio de los resultados obtenidos es de 78’97 en general la acogida ha sido notable
- También se puede apreciar en la imagen 153 como los valores más bajos de usabilidad no han superado el rango de valores neutrales, y dado que el valor más pequeño de los resultados de la tabla 5 es de 52’5, se puede concluir que la aplicación es notablemente accesible para los usuarios.
- Viendo los resultados de la tabla 7 se llega a la conclusión de que la media de la carga de trabajo que produce la aplicación no es elevada, el valor de la media del índice global de la carga de trabajo ha sido de 33’9%. De los valores de la imagen 154 se deduce que la mayoría de los usuarios consideran que la carga ejercida es baja, Aunque un número relevante de usuarios han tenido una carga considerada neutra, siendo el porcentaje más alto de carga conseguido de 53’66%.
- Como se puede ver en varias opiniones de la tabla 9, varios usuarios tuvieron problemas con que la aplicación reposicionaba algunos objetos durante su ejecución. Esto unido a algunas desviaciones que sufrían algunos objetos con respecto a la posición que se le había asignado durante algunas pruebas da a entender que el algoritmo de ubicación mediante coordenadas GPS no es del todo preciso y puede cometer errores o también se debe considerar la posibilidad de que el problema se produjera por fallos en el GPS de los dispositivos móviles.

- Otro punto en el que hacen hincapié varios usuarios es en la calidad de los modelos 3D. Pese a que la aceptación de los mismos no ha sido negativa, opinan que debería ser mejorado en futuras versiones.
- Una opinión que se repite mucho es lo satisfechos que están los usuarios con tener una forma de recibir indicaciones sobre lo que tienen que hacer a continuación mediante el menú ayuda.
- Todos los usuarios han sido capaces de completar la aplicación, por lo que se puede llegar a la conclusión de que esta no ofrece un grado muy elevado de complejidad.
- El usuario 18 posee amplios conocimientos en el ámbito educativo, opina que el lenguaje empleado es demasiado complejo para los niños. Lo que hay que tomar en consideración si se quiere tener a este sector de la población como público objetivo.
- Los usuarios han mostrado su aprobación y han dado una buena acogida tanto a la temática mitológica ibérica como a la narrativa de la aplicación.

Capítulo 6. TRABAJO FUTURO Y CONCLUSIONES

6.1 Trabajo futuro

Pese a haber presentado a los usuarios una aplicación completamente funcional y haber alcanzado los requisitos del proyecto, se puede ver, en el apartado visual, que este todavía se encuentra en una fase muy temprana. Por lo tanto, para llegar a la versión final habrá que seguir trabajando en diferentes aspectos del mismo.

Entre los aspectos que hay que ampliar y mejorar de Iberian Odyssey se encuentran los siguientes:

- Mejorar la parte visual de la aplicación empleando ilustraciones propias y más atrayentes para los menús de inicio y de aplicación completada. También cambiar los sprites de los distintos elementos UI que ofrece Unity por sprites más vistosos y originales.
- Sustituir los modelos 3D que han sido extraídos de páginas externas por modelos 3D propios con el fin de evitar cualquier tipo de demanda por infringir derechos de copyright.
- Agregar varias funcionalidades de AR Foundation que mejorarían la sensación de inmersión en los usuarios. Estas funcionalidades serían:
    - Detector de profundidad: se encarga de detectar la nube de puntos que compone un lugar y detectar si dicha nube se encuentra entre el objeto 3D. En caso de que se encuentre en medio, la nube de puntos irá ocultando progresivamente el objeto 3D, esto permite dar la sensación de que objetos del mundo real pueden ocultar detrás objetos del mundo digital, mejorando enormemente la integración de los objetos digitales en el mundo real.
    - Estimación lumínica: se encarga de calcular cuanta intensidad lumínica hay en el ambiente y la aplica a los modelos 3D de la aplicación.
- Dada la popularidad de las funcionalidades online y la cantidad de posibilidades que esto ofrece, sería interesante emplear la funcionalidad de los Cloud Anchor que ofrece AR Foundation para marcar lugares en donde los usuarios puedan colocar mensajes que otros usuarios pudieran leer, incentivando a que los usuarios se ayudaran mediante consejos puestos en determinados lugares.
- Ampliar la experiencia haciéndola más grande, abarcando toda una ciudad y añadiendo más bestias de las que se pueda extraer información y misiones que el usuario pueda realizar.
- Realizar tests para probar la efectividad de la aplicación en usuarios con edades comprendidas entre 12 y 17 años a la hora de introducir conocimientos de la mitología ibérica.
- Desarrollar una versión de la aplicación para dispositivos iOS.

También se han de tener en cuenta los problemas encontrados en las pruebas realizadas por los usuarios. Los fallos y errores detectados durante las pruebas se pueden tratar de corregir mediante las siguientes acciones:

- Corregir un error que ha surgido durante las pruebas con los usuarios 9, 12 y 18. En dicho error, los frames de la aplicación disminuían de forma considerable obligando al usuario a reiniciar la aplicación para que funcionase. Se han de realizar algunas pruebas vigilando la tasa de frames y comprobar cuál ha sido la cauda de este fallo.
- Implementar la funcionalidad que permita al usuario cerrar la pantalla de la aplicación sin que esto afecte al posicionamiento de las coordenadas GPS de los objetos digitales. Esta corrección también trata de solucionar los problemas de desviaciones que han surgido. Para ello se tendrá que revisar como se han implementado las funcionalidades del GPS de asset AR+GPS Location.
- Mejorar la acción "Ayuda" para que el uso del detector de planos en el que se indica donde invocar a la "Anjana" o al cebo para que sea más intuitivo de usar.
- Mejorar la implementación del "Interactuador" para no depender de tantos "else if" cuando se emplee la función Raycast para detectar las distintas bestias en función de tres distancias fijas. Esto ayudará a disminuir el trabajo a realizar si se añade una nueva distancia.

6.2 Conclusiones

Las conclusiones que se han obtenido de la realización de la aplicación Iberian Odyssey, han sido las siguientes:

- Se ha comprobado la viabilidad del proyecto al aplicar la geolocalización, la temática de la mitología ibérica y un tipo de experiencia más inmersiva a las aplicaciones del tipo "urban game".
- Pese a sus fallos y encontrarse en una etapa muy temprana, se ha conseguido una buena acogida entre los usuarios.
- Se ha demostrado la viabilidad de las distintas funcionalidades que ofrece la aplicación.
- Se han adquirido con éxito nuevos conocimientos sobre el uso del asset AR Foundation del motor Unity para la creación de aplicaciones de realidad aumentada.
- Se han mejorado habilidades en otras disciplinas como el modelado y animación 3D.
- Se ha aprendido sobre la gestión de un proyecto, incluyendo su diseño y ejecución. Concluyendo que un proyecto de esta magnitud es más factible si es desarrollado por un equipo multidisciplinar con el fin de aprovechar todo su potencial.

A modo de conclusión final, la realización de este Trabajo Fin de Master me ha servido para confirmar que trabajar en el desarrollo de aplicaciones que empleen tecnologías que exploren nuevas formas de que el usuario interactúe con los dispositivos y su entorno sería una excelente opción para mi futura vida laboral.

Capítulo 7. BIBLIOGRAFIA

1 Ciencia y Salud. Consultado el 15 de julio de 2021

https://www.um.es/lafem/DivulgacionCientifica/CienciaySalud/Portalyblog/cienciaysalud.laverdad.es/7_1_78.html

2 Ciencia y Salud. Consultado el 20 de agosto de 2021.

https://www.um.es/lafem/DivulgacionCientifica/CienciaySalud/Portalyblog/cienciaysalud.laverdad.es/7_1_78.html

3 realovirtual. Consultado el 15 de julio de 2021.

https://www.realovirtual.com/noticias/3259/realidad-aumentada-es-gran-idea-como-smartphone

4 ScienceDirect. Consultado el 8 de agosto de 2021.

https://www.sciencedirect.com/science/article/abs/pii/S0097849301001170

5 Google Académico. Consultado el 8 de agosto de 2021.

http://files.trendsandissues.webnode.com/200000010-3884839004/educamadrid-2007.pdf

6 idUS. Consultado el 8 de agosto de 2021.

https://idus.us.es/bitstream/handle/11441/45413/realidad%20aumentada%20y%20educacion.pdf?sequence=1&isAllowed=y

7 Dialnet. Consultado el 10 de agosto de 2021.

https://dialnet.unirioja.es/servlet/articulo?codigo=6046929

8 ScienceDirect. Consultado el 11 de agosto de 2021.

https://www.sciencedirect.com/science/article/pii/S1877042812023907

9 ScienceDirect. Consultado el 11 de agosto de 2021.

https://www.sciencedirect.com/science/article/abs/pii/S0360131512002527

10 Google Académico. Consultado el 11 de agosto de 2021.

https://www.solomonalexis.com/downloads/ar_edu.pdf

11 Sedici. Consultado el 11 de agosto de 2021.

http://sedici.unlp.edu.ar/bitstream/handle/10915/50745/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

12 Pokemon GO. Consultado el 12 de agosto de 2021.

https://pokemongolive.com/es/

13 progmat. Consultado el 13 de agosto de 2021.

http://www.progmat.uaem.mx:8080/Vol7num3/vol7num3art3.pdf

14 Semantic Scholar. Consultado el 13 de agosto de 2021.

https://breckon.org/toby/publications/papers/bordes11ar.pdf

15 ResearchGate. Consultado el 14 de agosto de 2021.

https://www.researchgate.net/publication/262220278_Urban_games_application_of_augmented_reality

16 ResearchGate. Consultado el 14 de agosto de 2021.

https://www.researchgate.net/publication/341116694_On_the_Development_of_Context-Aware_Augmented_Reality_Applications

17 Cornell University. Consultado el 14 de agosto de 2021.

https://arxiv.org/pdf/2008.07817.pdf

18 Harry Potter Wizards Unite. Consultado el 16 de agosto de 2021.

https://www.harrypotterwizardsunite.com/es/

19 Unity. Consultado el 16 de agosto de 2021.

https://unity.com/es

20 Fungus. Consultado el 16 de agosto de 2021.

https://fungusgames.com/

21 Unity. Consultado el 16 de agosto de 2021.

https://unity.com/es/unity/features/arfoundation

22 Developer Google. Consultado el 16 de agosto de 2021.

https://developers.google.com/ar

23 Developer Apple. Consultado el 16 de agosto de 2021.

https://developer.apple.com/augmented-reality/

24 Unity AR+GPS Location. Consultado el 16 de agosto de 2021.

https://docs.unity-ar-gps-location.com/

25 Models Resource. Consultado el 17 de agosto de 2021.

https://www.models-resource.com/

26 Blender. Consultado el 17 de agosto de 2021.

https://www.blender.org/

27 Mixamo. Consultado el 17 de agosto de 2021.

https://www.mixamo.com/#/

28 Emiliusvgs. Consultado el 17 de agosto de 2021.

https://emiliusvgs.com/ar-foundation-arcore-arkit/

29 Emiliusvgs. Consultado el 17 de agosto de 2021.

https://emiliusvgs.com/ar-foundation-arcore-arkit/

30 Unity AR+GPS Location. Consultado el 17 de agosto de 2021.

https://docs.unity-ar-gps-location.com/guide/#ar-foundation

31 Unity. Consultado el 18 de agosto de 2021.

https://unity.com/es/how-to/architect-game-code-scriptable-objects

32 Academia Android. Consultado el 18 de agosto de 2021.

https://academiaandroid.com/que-son-los-prefab-en-unity-3d/

33 UXpañol. Consultado el 26 de agosto de 2021.

https://uxpanol.com/teoria/sistema-de-escalas-de-usabilidad-que-es-y-para-que-sirve/

34 SciELO. Consultado el 26 de agosto de 2021.

https://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1576-59622010000300003

ANEXO 1: SCRIPTS IMPLEMENTADOS

1. Script Menu_Principal.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Menu_Principal : MonoBehaviour
{

    public Button boton_continue;
    public GameObject menu_creditos;


    void Awake()
    {
        if(PlayerPrefs.GetInt("Juego_Nuevo",0) == 0)
        {
            boton_continue.interactable = false;
        }

        else{
            boton_continue.interactable = true;
        }
    }


    public void Play()
    {
        PlayerPrefs.SetInt("Juego_Nuevo", 1);
        SceneManager.LoadScene("Juego");
    }

    public void NewGame()
    {
        PlayerPrefs.SetInt("Juego_Nuevo",0);
        SceneManager.LoadScene("NuevaPartida");
    }

    public void Quit()
    {
        Application.Quit();
    }

    public void Credits()
```

```
    {
        if(menu_creditos.activeSelf)
        {
            menu_creditos.SetActive(false);
        }

        else{
            menu_creditos.SetActive(true);
        }
    }

    public void Continuar()
    {
        SceneManager.LoadScene("Juego");
    }

    public void Regresar()
    {
        SceneManager.LoadScene("Inicio");
    }
}
```

2. Script NewGameController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class NewGameController : MonoBehaviour
{
    public Image imagen1;
    public Image imagen2;
    public Image imagen3;
    private void Awake() {
        PlayerPrefs.SetInt("num_mision",1);
        PlayerPrefs.SetInt("registro_Tarasca",0);
        PlayerPrefs.SetInt("registro_Trenti",0);
        PlayerPrefs.SetInt("registro_Musgosu",0);
        PlayerPrefs.SetInt("registro_Cuelebre",0);

        PlayerPrefs.SetInt("registro_PonerCebo",0);
        PlayerPrefs.SetInt("registro_Cruz",0);

        imagen1.enabled = false;
        imagen2.enabled = false;
```

```
        imagen3.enabled = false;
    }

    public void EmpezarPartida()
    {
        PlayerPrefs.SetInt("Juego_Nuevo",1);
        SceneManager.LoadScene("Juego");
    }

    //Activa la primera imagen
    public void EncenderImagen1()
    {
        imagen1.enabled = true;
    }

    //Activa la segunda imagen
    public void EncenderImagen2()
    {
        imagen2.enabled = true;
    }

    //Activa la tercera imagen
    public void EncenderImagen3()
    {
        imagen3.enabled = true;
    }
}
```

3. Script Menu_Juego.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class Menu_Juego : MonoBehaviour
{
    public GameObject Menu_Desplegable;
    public GameObject Menu_Acciones;
    public GameObject Menu_Bestiario;
    public GameObject Pagina_Bestiario;
    public GameObject Menu_Cebo;
    public GameObject Menu_Ayuda;
    public GameObject Menu_Analisis;
    public GameObject Menu_Interactuador;

    public Boton_Acciones accion_boton;
```

```
public Button boton_menu;
public GameObject boton_accion;

public GameObject menu_Salir;
public GameObject Cruz_Controller;
public Brujula brujula;
public Image NewHelp;
private bool cuelebre_help;

private GameObject[] flechas_brujula;

//Inicializa la clase con la variable cuelebre_help a falso
void Start()
{
    cuelebre_help = false;
}

//Si se cumplen una serie de requisitos, se mostrará un indicador de que hay una nueva ayuda
//en el menu de ayuda
void Update()
{
    if(PlayerPrefs.GetInt("registro_Cuelebre",0) == 1 &&
        PlayerPrefs.GetInt("num_mision",1) == 5 && !cuelebre_help)
    {
        NewHelp.enabled = true;
    }
}

//Abre y cierra el menú desplegable principal cuando se aprieta el botón "Menu"
public void OnClickMenuButton()
{
    Cruz_Controller.GetComponent<CruzSantaMarta>().DestruirCruces();
    Menu_Acciones.SetActive(false);
    Menu_Bestiario.SetActive(false);
    Pagina_Bestiario.SetActive(false);
    if(Menu_Desplegable.activeSelf)
    {
        Menu_Desplegable.SetActive(false);
    }
    else{
        Menu_Desplegable.SetActive(true);
    }
}

//Abre y cierra el menú desplegable del listado de acciones cuando se aprieta el
//botón "Acciones"
public void OnClick_Acciones()
{
    Cruz_Controller.GetComponent<CruzSantaMarta>().DestruirCruces();
```

```
        Menu_Bestiario.SetActive(false);
        Pagina_Bestiario.SetActive(false);
        if(Menu_Acciones.activeSelf)
        {
            Menu_Acciones.SetActive(false);
        }

        else{
            Menu_Acciones.SetActive(true);
        }
    }

    //Abre y cierra el menú desplegable del bestiario cuando se aprieta el botón "Bestiario"
    public void OnClick_Bestiario()
    {
        Cruz_Controller.GetComponent<CruzSantaMarta>().DestruirCruces();
        Menu_Acciones.SetActive(false);
        Pagina_Bestiario.SetActive(false);
        if(Menu_Bestiario.activeSelf)
        {
            Menu_Bestiario.SetActive(false);
        }

        else{
            Menu_Bestiario.SetActive(true);
        }
    }

    //Abre y cierra el menú de ayuda cuando se aprieta el botón "Ayuda"
    public void OnClick_Ayuda()
    {
        Cruz_Controller.GetComponent<CruzSantaMarta>().DestruirCruces();
        OcultarNuevaAyuda();
        Cruz_Controller.SetActive(false);
        Menu_Bestiario.SetActive(false);
        Menu_Cebo.SetActive(false);
        Menu_Interactuador.SetActive(false);
        Menu_Acciones.SetActive(false);
        Pagina_Bestiario.SetActive(false);
        Menu_Interactuador.SetActive(false);
        Menu_Desplegable.SetActive(false);
        Menu_Analisis.SetActive(false);
        flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
        foreach(GameObject flecha in flechas_brujula)
        {
            flecha.GetComponent<Direccion_flecha>().activada = false;
        }
        if(Menu_Ayuda.activeSelf)
        {
```

```
            Menu_Ayuda.SetActive(false);
            boton_accion.SetActive(false);
        }

        else{
            Menu_Ayuda.SetActive(true);
            boton_accion.SetActive(true);
            boton_accion.GetComponentInChildren<Button>().interactable = true;
            accion_boton.setAccionSprite(6);
        }
    }

    //Sale a la escena de inicio
    public void Quit()
    {
        SceneManager.LoadScene("Inicio");
    }

    //Abre y cierra el menú de salir cuando se aprieta el botón "Salir"
    public void Menu_Salir()
    {
        Cruz_Controller.GetComponent<CruzSantaMarta>().DestruirCruces();
        Menu_Bestiario.SetActive(false);
        Menu_Cebo.SetActive(false);
        Menu_Acciones.SetActive(false);
        Pagina_Bestiario.SetActive(false);
        Menu_Ayuda.SetActive(false);
        Menu_Desplegable.SetActive(false);
        Menu_Analisis.SetActive(false);
        boton_accion.SetActive(false);
        brujula.brujula_activa = false;
        flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
        foreach(GameObject flecha in flechas_brujula)
        {
            flecha.GetComponent<Direccion_flecha>().activada = false;
        }
        if(menu_Salir.activeSelf)
        {
            menu_Salir.SetActive(false);
        }
        else{
            menu_Salir.SetActive(true);
        }
    }

    //Hace que aparezca un indicador de que hay ayuda nueva en el menú de ayuda.
    public void MostrarNuevaAyuda()
    {
        NewHelp.enabled = true;
```

```
    }

    //Hace que el indicador de ayuda nueva desaparezca
    public void OcultarNuevaAyuda()
    {
        NewHelp.enabled = false;
        if(PlayerPrefs.GetInt("registro_Cuelebre",0) == 1 &&
            PlayerPrefs.GetInt("num_mision",1) == 5 && !cuelebre_help)
        {
            cuelebre_help = true;
        }
    }

}
```

4. Script Misiones_Controller.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Misiones_Controller : MonoBehaviour
{

    public int misionActual;
    private int misionAnterior;
    public GameObject trenti;
    public GameObject spawn_trenti1, spawn_trenti2;
    public GameObject Musgosu;
    public GameObject cofre_cuelebre;
    public GameObject Tarasca;
    public GameObject OrdenFinal;
    public Brujula brujula;
    public Button botonAyuda;
    public Menu_Juego ayuda;


    private void Awake() {
        misionActual = PlayerPrefs.GetInt("num_mision",1);
        ActivarMision(misionActual);
        switch(misionActual)
        {
            case 3:
                Destroy(spawn_trenti1);
                Destroy(spawn_trenti2);
                break;
            case 4:
                Destroy(spawn_trenti1);
```

```
            Destroy(spawn_trenti2);
            Destroy(trenti);
            break;
        case 5:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            Destroy(Musgosu);
            break;
        case 6:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            Destroy(Musgosu);
            cofre_cuelebre.GetComponent<Cofre_Cuelebre>().abierto = true;
            break;
        case 7:
            Destroy(spawn_trenti1);
            Destroy(spawn_trenti2);
            Destroy(trenti);
            Destroy(Musgosu);
            cofre_cuelebre.GetComponent<Cofre_Cuelebre>().abierto = true;
            Destroy(Tarasca);
            break;
    }
}

public void Cambiar_Mision()
{
    misionActual++;
    if(misionActual != misionAnterior)
    {
        misionAnterior = misionActual;
        PlayerPrefs.SetInt("num_mision",misionActual);
        brujula.deleteObjetivos();
        ActivarMision(misionActual);
        ayuda.MostrarNuevaAyuda();
    }
}

public void ActivarMision(int id_mision)
{
    switch(id_mision)
    {
        case 1:
            brujula.addObjetivo(trenti);
            break;
        case 2:
            brujula.addObjetivo(spawn_trenti1);
```

```
                brujula.addObjetivo(spawn_trenti2);
                break;
            case 3:
                brujula.addObjetivo(trenti);
                break;
            case 4:
                Musgosu.SetActive(true);
                brujula.addObjetivo(Musgosu);
                break;
            case 5:
                brujula.addObjetivo(cofre_cuelebre);
                break;
            case 6:
                brujula.addObjetivo(Tarasca);
                break;
            case 7:
                botonAyuda.interactable = true;
                OrdenFinal.SetActive(true);
                break;
        }
    }
}
```

5. Script Acciones.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Acciones : MonoBehaviour
{
    public GameObject Menu_Acciones;
    public GameObject Menu_Desplegable;
    public GameObject Menu_Analizar;
    public GameObject Menu_Interactuar;
    public GameObject Menu_Ayuda;
    public GameObject Boton_accion;
    public GameObject Menu_Cebo;
    public GameObject Menu_Cruz;
    public Boton_Acciones accion_botones;
    public GameObject PlanoCruz;
    public Brujula brujula;
    private GameObject[] flechas_brujula;

    //Activa el menú de la acción Poner Cebo
    public void Onclick_PonerCebo()
```

```
{
    PlanoCruz.SetActive(false);
    flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
    foreach(GameObject flecha in flechas_brujula)
    {
        flecha.GetComponent<Direccion_flecha>().activada = false;
    }
    Destroy(GameObject.FindGameObjectWithTag("Cruz"));
    Menu_Interactuar.SetActive(false);
    Menu_Ayuda.SetActive(false);
    Boton_accion.SetActive(true);
    Boton_accion.GetComponentInChildren<Button>().interactable = true;
    Menu_Acciones.SetActive(false);
    Menu_Desplegable.SetActive(false);
    Menu_Cruz.GetComponent<CruzSantaMarta>().DestruirCruces();
    Menu_Cruz.SetActive(false);
    brujula.brujula_activa = false;

    Menu_Analizar.SetActive(false);

    Menu_Cebo.SetActive(true);

    accion_botones.setAccionSprite(3);

}

//Activa el menú de la acción Análisis
public void Onclick_Analizar()
{
    PlanoCruz.SetActive(false);
    flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
    foreach(GameObject flecha in flechas_brujula)
    {
        flecha.GetComponent<Direccion_flecha>().activada = false;
    }
    Destroy(GameObject.FindGameObjectWithTag("Cruz"));
    Menu_Interactuar.SetActive(false);
    Menu_Ayuda.SetActive(false);
    Boton_accion.SetActive(false);
    Menu_Acciones.SetActive(false);
    Menu_Desplegable.SetActive(false);
    Menu_Cruz.GetComponent<CruzSantaMarta>().DestruirCruces();
    Menu_Cruz.SetActive(false);
    brujula.brujula_activa = false;

    Menu_Cebo.SetActive(false);

    Menu_Analizar.SetActive(true);
    Menu_Analizar.GetComponentInChildren<BarraAnalisis>().Iniciar();
```

```
}

//Activa el menú de la acción Interactuar
public void Onclick_Interactuar()
{
    PlanoCruz.SetActive(false);
    flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
    foreach(GameObject flecha in flechas_brujula)
    {
        flecha.GetComponent<Direccion_flecha>().activada = false;
    }
    Destroy(GameObject.FindGameObjectWithTag("Cruz"));

    Menu_Ayuda.SetActive(false);
    Boton_accion.SetActive(true);
    Menu_Interactuar.SetActive(true);
    Menu_Acciones.SetActive(false);
    Menu_Desplegable.SetActive(false);
    Menu_Cruz.GetComponent<CruzSantaMarta>().DestruirCruces();
    Menu_Cruz.SetActive(false);
    brujula.brujula_activa = false;

    Menu_Analizar.SetActive(false);

    Menu_Cebo.SetActive(false);

    accion_botones.setAccionSprite(1);

}

//Activa el menú de la acción Brújula
public void Onclick_Brujula()
{
    PlanoCruz.SetActive(false);
    brujula.brujula_activa = true;
    flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
    foreach(GameObject flecha in flechas_brujula)
    {
        flecha.GetComponent<Direccion_flecha>().activada = true;
    }
    Destroy(GameObject.FindGameObjectWithTag("Cruz"));
    Menu_Interactuar.SetActive(false);
    Menu_Ayuda.SetActive(false);
    Boton_accion.SetActive(false);
    Menu_Acciones.SetActive(false);
    Menu_Desplegable.SetActive(false);
    Menu_Cruz.GetComponent<CruzSantaMarta>().DestruirCruces();
    Menu_Cruz.SetActive(false);
```

```
        Menu_Analizar.SetActive(false);

        Menu_Cebo.SetActive(false);
    }

    //Activa el menú de la acción Cruz Santa Marta
    public void Onclick_Cruz()
    {
        PlanoCruz.SetActive(true);
        flechas_brujula = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
        foreach(GameObject flecha in flechas_brujula)
        {
            flecha.GetComponent<Direccion_flecha>().activada = false;
        }

        Menu_Ayuda.SetActive(false);
        Menu_Interactuar.SetActive(false);
        Boton_accion.SetActive(false);
        Menu_Acciones.SetActive(false);
        Menu_Desplegable.SetActive(false);
        Menu_Cruz.GetComponent<CruzSantaMarta>().DestruirCruces();
        Menu_Cruz.SetActive(true);
        brujula.brujula_activa = false;
        Menu_Analizar.SetActive(false);

        Menu_Cebo.SetActive(false);
    }
}
```

6. Script Ayuda.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Ayuda : MonoBehaviour
{
    public Button boton_menu;
    public GameObject Indicator_place;
    public GameObject boton_accion;

    void Update()
    {
        var anjana = FindObjectOfType<Anjana>();
        if(anjana != null)
        {
            boton_menu.interactable = false;
            Indicator_place.SetActive(false);
```

```
            boton_accion.SetActive(false);
        }

        else{
            boton_menu.interactable = true;
            Indicator_place.SetActive(true);
            boton_accion.SetActive(true);
        }
    }
}
```

7. Script PlacementIndicator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

public class PlacementIndicator : MonoBehaviour
{
    // Start is called before the first frame update
    private ARRaycastManager rayManager;
    private GameObject plano;

    void Start ()
    {
        rayManager = FindObjectOfType<ARRaycastManager>();
        plano = transform.GetChild(0).gameObject;
        plano.SetActive(false);
    }

    void Update ()
    {
        List<ARRaycastHit> hits = new List<ARRaycastHit>();
        rayManager.Raycast(new Vector2(Screen.width / 2, Screen.height / 2),
                           hits, TrackableType.Planes);
        if(hits.Count > 0)
        {
            transform.position = hits[0].pose.position;
            transform.rotation = hits[0].pose.rotation;

            if(!plano.activeInHierarchy)
            {
                plano.SetActive(true);
            }
        }
    }
```

```
}
```

8. Script ObjectSpawner.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARSubsystems;

namespace UnityEngine.XR.ARFoundation.Samples
{
    public class ObjectSpawner : MonoBehaviour
    {
        // Start is called before the first frame update

        public GameObject objectToSpawn;
        static List<ARRaycastHit> s_Hits = new List<ARRaycastHit>();

        List<ARAnchor> m_Anchors = new List<ARAnchor>();

        ARRaycastManager m_RaycastManager;

        ARAnchorManager m_AnchorManager;

        void Awake()
        {
            m_RaycastManager = FindObjectOfType<ARRaycastManager>();
            m_AnchorManager = FindObjectOfType<ARAnchorManager>();
        }

        public void DejarElemento ()
        {
            if(m_RaycastManager.Raycast(new Vector2(Screen.width/2, Screen.height/2),
                                        s_Hits,TrackableType.Planes))
            {
                var hit = s_Hits[0];
                var anchor = CreateAnchor(hit);

                if(anchor)
                {
                    m_Anchors.Add(anchor);
                }

                else{
                    Debug.Log("Problema al crear anchor");
                }
            }
```

```
        }

        public void RemoveAllAnchors()
        {
            foreach (var anchor in m_Anchors)
            {
                Destroy(anchor.gameObject);
            }
            m_Anchors.Clear();
        }

        ARAnchor CreateAnchor(in ARRaycastHit hit)
        {
            ARAnchor anchor = null;

            if(hit.trackable is ARPlane plane){
                var planeManager = FindObjectOfType<ARPlaneManager>();
                if (planeManager)
                {
                    var oldPrefab = m_AnchorManager.anchorPrefab;
                    m_AnchorManager.anchorPrefab = objectToSpawn;
                    anchor = m_AnchorManager.AttachAnchor(plane, hit.pose);
                    m_AnchorManager.anchorPrefab = oldPrefab;
                    return anchor;
                }
            }
            return anchor;
        }
    }
}
```

9. Script Boton_Acciones.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.ARFoundation.Samples;

public class Boton_Acciones : MonoBehaviour
{
    public int accion;

    public Interactuador interactuador;
    public ObjectSpawner objectSpawner, llamar_anjana;
    public Image imagen_boton_accion;

    public Sprite imagen_interactuar, imagen_cebo, imagen_ayuda;
```

```
    void Awake()
    {
        accion = 0;
    }

    public void AccionBoton()
    {
        switch(accion)
        {
            case 1:
                interactuador.Interactuar();
                break;
            case 3:
                if(!GameObject.FindGameObjectWithTag("Cebo"))
                {
                    objectSpawner.DejarElemento();
                }
                break;
            case 6:
                llamar_anjana.DejarElemento();
                break;
        }
    }

    public void setAccionSprite(int a)
    {
        accion = a;
        switch(accion)
        {
            case 1:
                imagen_boton_accion.sprite = imagen_interactuar;
                break;
            case 3:
                imagen_boton_accion.sprite = imagen_cebo;
                break;
            case 6:
                imagen_boton_accion.sprite = imagen_ayuda;
                break;
        }
    }
}
```

10. Script Bestia.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using Fungus;

public class Bestia : MonoBehaviour
{

    public int id;
    public float dist_renderizado;
    public Flowchart dialogo;
    public Vector3 camara_pos;
    public Animator animator;
    public bool dialogando;

    public void DistanciaRender()
    {
       if(Mathf.Sqrt(Mathf.Pow(this.transform.position.x -
 Camera.main.transform.position.x,2)+Mathf.Pow(this.transform.position.z -
 Camera.main.transform.position.z,2)) >= dist_renderizado)
       {
            if(this.GetComponentInChildren<SkinnedMeshRenderer>())
            {

                var meshes = this.GetComponentsInChildren<SkinnedMeshRenderer>();
                foreach (var m in meshes)
                {
                    m.enabled = false;
                }
            }
            else
            {
                var meshes = this.GetComponentsInChildren<MeshRenderer>();
                foreach (var m in meshes)
                {
                    m.enabled = false;
                }
            }
       }

       else{
           if(this.GetComponentInChildren<SkinnedMeshRenderer>())
           {
               var meshes = this.GetComponentsInChildren<SkinnedMeshRenderer>();
                foreach (var m in meshes)
                {
                    m.enabled = true;
                }
           }
           else
            {
                var meshes = this.GetComponentsInChildren<MeshRenderer>();
```

```
                foreach (var m in meshes)
                {
                    m.enabled = true;
                }
            }
        }
    }

    public void MirarJugador()
    {
        if(dialogando)
        {
            camara_pos = Camera.main.transform.position;
            var rotacion = camara_pos - transform.position;
            rotacion.y = 0;
            var rotacion_Quaternion = Quaternion.LookRotation(rotacion);
            transform.rotation = Quaternion.Slerp(transform.rotation,
                                                  rotacion_Quaternion, 1);
        }
    }

    public void ComenzarDialogo()
    {
        dialogando = true;
    }

    public void TerminarDialogo()
    {
        dialogando = false;
    }
}
```

11. Script Anjana.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using Fungus;

public class Anjana : Bestia
{
    //private Vector3 camara_pos;

    private int mision_actual;

    private float vel = 0.25f;
```

```
private bool descenso, ascenso;

public GameObject Anajana_GO;

// Start is called before the first frame update
void Start()
{
    dialogo = GetComponentInChildren<Flowchart>();
    ComenzarDialogo();
    descenso = true;
    ascenso = false;
    camara_pos = Camera.main.transform.position;
    this.transform.localPosition = new Vector3(0,0.75f,0);
    mision_actual = FindObjectOfType<Misiones_Controller>().misionActual;

    switch(mision_actual)
    {
        case 1:
            dialogo.ExecuteBlock("Mision1");
            break;
        case 2:
            dialogo.ExecuteBlock("Mision2");
            break;
        case 3:
            dialogo.ExecuteBlock("Mision3");
            break;
        case 4:
            dialogo.ExecuteBlock("Mision4");
            break;
        case 5:
            if(PlayerPrefs.GetInt("registro_Cuelebre",0) == 1)
            {
                dialogo.ExecuteBlock("Mision5_2");
            }
            else{
                dialogo.ExecuteBlock("Mision5");
            }
            break;
        case 6:
            dialogo.ExecuteBlock("Mision6");
            break;
        case 7:
            dialogo.ExecuteBlock("Final");
            break;
    }
}

// Update is called once per frame
void Update()
```

```
    {
        if(descenso)
        {
            this.transform.localPosition = new Vector3(0,
                       this.transform.localPosition.y + -(vel)*Time.deltaTime, 0);
            if(this.transform.localPosition.y <= 0)
                descenso = false;
        }

        else if(ascenso)
        {
            this.transform.localPosition = new Vector3(0,
                       this.transform.localPosition.y + (vel)*Time.deltaTime, 0);
            if(this.transform.localPosition.y >= 1)
            {
                ascenso = false;
                Destroy(Anajana_GO);
            }
        }
        MirarJugador();
    }

    public void Irse()
    {
        ascenso = true;
    }

    public void Finalizar()
    {
        SceneManager.LoadScene("Final");
    }
}
```

12. Script Trenti_Boss.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Fungus;

public class Trenti_Boss : Bestia
{
    void Start()
    {
        animator = GetComponent<Animator>();
        dialogo = GetComponentInChildren<Flowchart>();
    }
```

```
    // Update is called once per frame
    void Update()
    {
        DistanciaRender();
        MirarJugador();
    }

    public void Dialogo()
    {
        ComenzarDialogo();

        int mision = FindObjectOfType<Misiones_Controller>().misionActual;

        switch(mision)
        {
            case 1:
                animator.SetBool("cry",true);
                dialogo.ExecuteBlock("Mision1");
                FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
            break;
            case 2:
                animator.SetBool("cry",true);
                dialogo.ExecuteBlock("Mision2");
            break;
            case 3:
                animator.SetBool("cry",false);
                animator.SetBool("happy",true);
                dialogo.ExecuteBlock("Mision3");
                FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
            break;
        }
    }

    public void Irse()
    {
        TerminarDialogo();
        Destroy(this.gameObject);
    }

    public void CerrarDialogoCry()
    {
        TerminarDialogo();
        animator.SetBool("cry",false);
    }
}
```

13. Script SpawnTrenti.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnTrenti : MonoBehaviour
{

    public GameObject trenti;

    private int num_trentis = 2;

    private float radio = 40;

    public GameObject target;
    void Start()
    {
        CrearTrentis();
    }

    void Update()
    {
        foreach(Trenti t in this.GetComponentsInChildren<Trenti>())
        {
            if(t.distancia < 0.5f)
            {
                Destroy(t.target);
                Vector3 pos_trenti = new Vector3(Random.Range(
                                                this.transform.position.x - radio,
                                                this.transform.position.x + radio),
                                                this.transform.position.y,
                                                Random.Range(this.transform.position.z - radio,
                                                this.transform.position.z + radio));

                GameObject target_obj = Instantiate(target,pos_trenti,
                                                 Quaternion.identity) as GameObject;

                target_obj.transform.parent = this.transform;
                t.target = target_obj;
            }
        }


        if(this.GetComponentsInChildren<Trenti>().Length != num_trentis)
        {
            num_trentis = this.GetComponentsInChildren<Trenti>().Length;
            if(FindObjectOfType<Misiones_Controller>().misionActual == 2 &&
               num_trentis == 0)
            {
                Destroy(this.gameObject);
```

```
            }
        }
    }

    public void CrearTrentis()
    {
        for(int i=1; i<=num_trentis;i++)
        {
            Vector3 pos_trenti = new Vector3(Random.Range(this.transform.position.x - radio,
                                                          this.transform.position.x + radio),
                                             -3f, Random.Range(this.transform.position.z - radio,
                                                          this.transform.position.z + radio));

            GameObject trenti_obj = Instantiate(trenti, pos_trenti,
                                                Quaternion.identity) as GameObject;

            trenti_obj.transform.parent = this.transform;
            GameObject target_obj = Instantiate(target, this.transform.position,
                                                Quaternion.identity) as GameObject;

            target_obj.transform.parent = this.transform;
            trenti_obj.transform.position = new Vector3(Random.Range(
                                            this.transform.position.x -radio,
                                            this.transform.position.x + radio),
                                            0, Random.Range(this.transform.position.z - radio,
                                                            this.transform.position.z + radio));

            target_obj.transform.position = new Vector3(Random.Range(
                                                        this.transform.position.x - radio,
                                                        this.transform.position.x + radio),
                                             0, Random.Range(this.transform.position.z - radio,
                                                        this.transform.position.z + radio));

            trenti_obj.GetComponent<Trenti>().target = target_obj;
        }
    }
}
```

14. Script Trenti.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
using Fungus;

public class Trenti : Bestia
{
```

```
public GameObject target;
public float vel;
public GameObject Cuenta_Trenti;
public float distancia;

void Awake() {
    animator = GetComponent<Animator>();
    dialogo = GetComponentInChildren<Flowchart>();
    vel = 2;
    this.transform.position = new Vector3(this.transform.position.x,
                                          -3, this.transform.position.z);
}

void Update()
{
    DistanciaRender();
    distancia = Mathf.Sqrt(Mathf.Pow(
                this.transform.position.x - target.transform.position.x,2) +
                Mathf.Pow(this.transform.position.z - target.transform.position.z,2));

    MirarJugador();
    if(!dialogando){
        if(target != null)
        {
            float mod = Mathf.Sqrt(Mathf.Pow(target.transform.position.x -
                                             this.transform.position.x,2) +
                                             Mathf.Pow(target.transform.position.z -
                                             this.transform.position.z,2));

            Vector3 dir_target = new Vector3(((target.transform.position.x -
                                               this.transform.position.x)/mod),-3,
                                               ((target.transform.position.z -
                                               this.transform.position.z)/mod));

            this.transform.position = new Vector3(this.transform.position.x +
                                                  (vel * dir_target.x)*Time.deltaTime,
                                                  -3, this.transform.position.z + (
                                                  vel * dir_target.z)*Time.deltaTime);

            var rotacion = target.transform.position - transform.position;
            rotacion.y = 0;
            var rotacion_Quaternion = Quaternion.LookRotation(rotacion);
            transform.rotation = Quaternion.Slerp(transform.rotation,rotacion_Quaternion,1);
        }
    }

}

public void Irse()
```

```
{
    GameObject cuenta = Instantiate(Cuenta_Trenti, new Vector3(0,0,0),
                                    Quaternion.identity);

    cuenta.GetComponent<MensajeAnalisis>().CuentaTrenti();
    Destroy(target);
    Destroy(this.gameObject);
}

public void Dialogo()
{
    ComenzarDialogo();
    int comentario = Random.Range(0,6);
    if(FindObjectOfType<Misiones_Controller>().misionActual == 2)
    {
        animator.SetBool("happy",true);
        switch(comentario)
        {
            case 0:
                dialogo.ExecuteBlock("Respuesta0");
            break;
            case 1:
                dialogo.ExecuteBlock("Respuesta1");
            break;
            case 2:
                dialogo.ExecuteBlock("Respuesta2");
            break;
            case 3:
                dialogo.ExecuteBlock("Respuesta3");
            break;
            case 4:
                dialogo.ExecuteBlock("Respuesta4");
            break;
            case 5:
                dialogo.ExecuteBlock("Respuesta5");
            break;
        }
    }

    else
    {
        animator.SetBool("cry",true);
        switch(comentario)
        {
            case 0:
                dialogo.ExecuteBlock("Comentario0");
            break;
            case 1:
                dialogo.ExecuteBlock("Comentario1");
```

```
                break;
                case 2:
                    dialogo.ExecuteBlock("Comentario2");
                break;
                case 3:
                    dialogo.ExecuteBlock("Comentario3");
                break;
                case 4:
                    dialogo.ExecuteBlock("Comentario4");
                break;
                case 5:
                    dialogo.ExecuteBlock("Comentario5");
                break;
            }
        }
    }

    public void QuitarParado()
    {
        TerminarDialogo();
        animator.SetBool("cry",false);
    }
}
```

15. Script Musgosu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Fungus;

public class Musgosu : Bestia
{

    private void Awake() {
        animator = this.GetComponent<Animator>();
        dialogo = this.GetComponentInChildren<Flowchart>();
        camara_pos = Camera.main.transform.position;
        dialogando = false;
    }

    void Update()
    {
        DistanciaRender();
        MirarJugador();
    }

    public void Dialogo()
```

```
    {
        if(PlayerPrefs.GetInt("num_mision",1) == 4)
        {
            dialogo.ExecuteBlock("Dialogo_Musgosu");

        }

        else{
            dialogo.ExecuteBlock("No_Preparado");
        }
        ComenzarDialogo();
    }

    public void SiguienteMision()
    {
        GameObject.FindGameObjectWithTag("Inventario").GetComponent<Inventario>().
                                                    AddEntradaAccion(2);

        PlayerPrefs.SetInt("registro_PonerCebo",1);
        FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
        Destroy(this.gameObject);
    }

    public void Gesticular()
    {
        animator.SetBool("gesto",true);
    }

    public void NoGesticular()
    {
        animator.SetBool("gesto",false);
    }
}
```

16. Script Cuelebre.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Fungus;

public class Cuelebre : Bestia
{

    private float vel;
    public bool cazando;
    public bool comiendo;
```

```
private Transform pos_cofre;
private Transform pos_Cebo;
private Vector3 dir_ida;
private Vector3 dir_vuelta;

private void Awake() {
    animator = this.GetComponentInChildren<Animator>();
    pos_cofre = GameObject.FindGameObjectWithTag("Cofre").transform;
    dialogo = this.GetComponentInChildren<Flowchart>();
    cazando = false;
    vel = 5;
}

void Update()
{
    DistanciaRender();
    MirarJugador();
    if(cazando)
    {
        if(pos_Cebo != null)
        {
            if(!comiendo)
            {
                animator.SetBool("fly", true);
                var rotacion = pos_Cebo.position - transform.position;
                rotacion.y = 0;
                var rotacion_Quaternion = Quaternion.LookRotation(rotacion);
                transform.rotation = Quaternion.Slerp(transform.rotation,
                                                      rotacion_Quaternion, 1);

                this.transform.position = new Vector3(this.transform.position.x +
                                                     (vel * dir_ida.x) *Time.deltaTime,
                                                     this.transform.position.y,
                                                     this.transform.position.z +
                                                     (vel * dir_ida.z) * Time.deltaTime);
            }

            else
            {
                animator.SetBool("fly", false);
            }
        }

        else
        {
            animator.SetBool("fly", true);
            comiendo = false;
            var rotacion = pos_cofre.position - transform.position;
            rotacion.y = 0;
```

```
            var rotacion_Quaternion = Quaternion.LookRotation(rotacion);
            transform.rotation = Quaternion.Slerp(transform.rotation,
                                                  rotacion_Quaternion, 1);

            this.transform.position = new Vector3(this.transform.position.x +
                                                 (vel * dir_vuelta.x)*Time.deltaTime,
                                                  this.transform.position.y,
                                                  this.transform.position.z +
                                                  (vel * dir_vuelta.z)*Time.deltaTime);

            if((Mathf.Sqrt(Mathf.Pow(this.transform.position.x - pos_cofre.position.x,2) +
                           Mathf.Pow(this.transform.position.z - pos_cofre.position.z,2)))
                            <= 10f)
            {
                cazando = false;
            }
        }
    }
}

public void SeguirCebo()
{
    cazando = true;
    pos_Cebo = GameObject.FindGameObjectWithTag("Cebo").transform;
    float mod_ida = Mathf.Sqrt(Mathf.Pow(
                    pos_Cebo.position.x - this.transform.position.x,2) +
                    Mathf.Pow(pos_Cebo.position.z - this.transform.position.z,2));

    dir_ida = new Vector3(((pos_Cebo.position.x - this.transform.position.x)/mod_ida),
                          this.transform.position.y,((pos_Cebo.position.z -
                          this.transform.position.z)/mod_ida));

    float mod_vuelta = Mathf.Sqrt(Mathf.Pow(pos_cofre.position.x -
                                  this.transform.position.x,2) +
                                  Mathf.Pow(pos_cofre.position.z -
                                  this.transform.position.z,2));

    dir_vuelta = new Vector3(-((this.transform.position.x -
                             pos_cofre.position.x)/mod_vuelta),
                             this.transform.position.y,
                             -((this.transform.position.z -
                             pos_cofre.position.z)/mod_vuelta));
}

public void Dialogo()
{
    if(!cazando)
    {
        ComenzarDialogo();
```

```
            dialogo.ExecuteBlock("Dialogo_Cuelebre");
        }

    }
}
```

17. Script Cofre_Cuelebre.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Fungus;

public class Cofre_Cuelebre : MonoBehaviour
{
    private float dist_culebre;
    private GameObject cuelebre;
    public GameObject chesOpened;
    public GameObject chesClosed;
    private bool custodiado;
    public bool abierto;
    private Flowchart dialogo;

    private void Awake() {
        cuelebre = GameObject.FindGameObjectWithTag("Cuelebre");
        custodiado = true;
        abierto = false;
        dialogo = GetComponentInChildren<Flowchart>();
    }

    void Update()
    {
        dist_culebre = Mathf.Sqrt(Mathf.Pow(cuelebre.transform.position.x -
                                            this.transform.position.x,2) +
                                  Mathf.Pow(cuelebre.transform.position.z -
                                            this.transform.position.z,2));

        if(dist_culebre > 5f)
        {
            custodiado = false;
        }
        else{
            custodiado = true;
        }


        if (Mathf.Sqrt(Mathf.Pow(this.transform.position.x -
                       Camera.main.transform.position.x, 2) +
```

```
                    Mathf.Pow(this.transform.position.z -
                    Camera.main.transform.position.z, 2)) >= 80f)
    {
        if (this.GetComponentInChildren<MeshRenderer>())
        {
            var meshes = this.GetComponentsInChildren<MeshRenderer>();
            foreach (var m in meshes)
            {
                m.enabled = false;
            }
        }
    }
    else
    {
        if (this.GetComponentInChildren<MeshRenderer>())
        {
            var meshes = this.GetComponentsInChildren<MeshRenderer>();
            foreach (var m in meshes)
            {
                m.enabled = true;
            }
        }
    }
}

public void Abrir()
{
    if(!abierto)
    {
     GameObject opend = Instantiate(chesOpened, new Vector3(chesClosed.transform.position.x,
                                                            chesClosed.transform.position.y,
                                                           chesClosed.transform.position.z),
                                    Quaternion.Euler(0,0,0)) as GameObject;

        opend.transform.parent = this.gameObject.transform;
        opend.transform.position = new Vector3(chesClosed.transform.position.x,
                                               chesClosed.transform.position.y,
                                               chesClosed.transform.position.z);
        Destroy(chesClosed);
        opend.transform.localScale = new Vector3(0.15f, 0.15f, 0.15f);
        dialogo.ExecuteBlock("Abierto");
        abierto = true;
    }
    else{
        dialogo.ExecuteBlock("Vacio");
    }
}

public void Activar_Cruz()
```

```
    {
        GameObject.FindGameObjectWithTag("Inventario").GetComponent<Inventario>().
                                                        AddEntradaAccion(4);

        PlayerPrefs.SetInt("registro_Cruz",1);
        FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
    }
}
```

18. Script Tarasca.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Fungus;

public class Tarasca : Bestia
{
    void Start()
    {
        dialogo = this.GetComponentInChildren<Flowchart>();
    }

    void Update()
    {
        DistanciaRender();
        MirarJugador();
    }

    public void Dialogo()
    {
        ComenzarDialogo();
        dialogo.ExecuteBlock("Dialogo_Tarasca");
    }

    public void Amenazado()
    {
        ComenzarDialogo();
        dialogo.ExecuteBlock("Derrota");
    }

    public void Eliminacion()
    {
        FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
        Destroy(this.gameObject);
    }
}
```

19. Script Interactuador.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;


public class Interactuador : MonoBehaviour
{
    private float distancia_interactuador_larga = 45;
    private float distancia_interactuador_media = 40;
    private float distancia_interactuador_corta = 15;
    public Button boton_accion;


    private void FixedUpdate()
    {
        RaycastHit hitlargo;
        RaycastHit hitmedio;
        RaycastHit hitcorto;
        if(Physics.Raycast(Camera.main.transform.position,
                           Camera.main.transform.forward,
                           out hitlargo, distancia_interactuador_larga))

        {
            //Lanzamiento del Raycast de larga distancia
            if(hitlargo.collider.tag == "Cuelebre" || hitlargo.collider.tag == "Tarasca")
            {
                boton_accion.interactable = true;
            }

            else if (Physics.Raycast(Camera.main.transform.position,
                                     Camera.main.transform.forward,
                                     out hitmedio, distancia_interactuador_media))
            {
                //Lanzamiento del Raycast de media distancia
                if(hitmedio.collider.tag == "Cofre" || hitmedio.collider.tag == "Trenti")
                {
                    boton_accion.interactable = true;
                }

                else if (Physics.Raycast(Camera.main.transform.position,
                                         Camera.main.transform.forward,
                                         out hitcorto, distancia_interactuador_corta))
                {
                    //Lanzamiento del Raycast de corta distancia
```

```
            if(hitcorto.collider.tag == "Trenti_Boss" || hitcorto.collider.tag == "Musgosu")
                {
                    boton_accion.interactable = true;
                }
            }

            else
            {
                boton_accion.interactable = false;
            }
        }

        else
        {
            boton_accion.interactable = false;
        }
    }

    else
    {
        boton_accion.interactable = false;
    }
}

public void Interactuar()
{
    RaycastHit hit;
    if(Physics.Raycast(Camera.main.transform.position,
                       Camera.main.transform.forward,
                       out hit, distancia_interactuador_larga))
    {
        if(hit.collider.tag == "Trenti_Boss")
        {
            Debug.Log("Interctuar con Trenti_Boss");
            hit.collider.GetComponent<Trenti_Boss>().Dialogo();
        }
        if(hit.collider.tag == "Trenti")
        {
            Debug.Log("Interctuar con Trenti");
            hit.collider.GetComponent<Trenti>().Dialogo();
        }
        if(hit.collider.tag == "Musgosu")
        {
            Debug.Log("Interctuar con Musgosu");
            hit.collider.GetComponent<Musgosu>().Dialogo();
        }
        if(hit.collider.tag == "Cuelebre")
        {
            Debug.Log("Interactuar con Cuelebre");
```

```
                hit.collider.GetComponent<Cuelebre>().Dialogo();
            }
            if(hit.collider.tag == "Tarasca")
            {
                Debug.Log("Interactuar con Tarasca");
                hit.collider.GetComponent<Tarasca>().Dialogo();
            }
            if(hit.collider.tag == "Cofre")
            {
                hit.collider.GetComponent<Cofre_Cuelebre>().Abrir();
            }
        }
    }
}
```

20. Script Analizador.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Analizador : MonoBehaviour
{

    private float distancia_analisis = 50;
    public Inventario menu_bestias;
    private BarraAnalisis bar_analisis;
    private int last_id;
    // Start is called before the first frame update
    void Start()
    {
        bar_analisis = FindObjectOfType<BarraAnalisis>();
    }

    // Update is called once per frame
    private void FixedUpdate()
    {
        RaycastHit hit;
        if(Physics.Raycast(Camera.main.transform.position,
                           Camera.main.transform.forward,
                           out hit, distancia_analisis))
        {
            if(hit.collider.tag == "Trenti_Boss" || hit.collider.tag == "Trenti" ||
                hit.collider.tag == "Musgosu" || hit.collider.tag == "Cuelebre" ||
                hit.collider.tag == "Tarasca")
            {
                bar_analisis.Analizar();
```

```
                if(bar_analisis.analizado)
                {
                    last_id = hit.collider.GetComponent<Bestia>().id;
                    menu_bestias.AddEntradaBestiario(last_id);
                    switch(last_id)
                    {
                        case 1:
                            PlayerPrefs.SetInt("registro_Trenti",1);
                            break;
                        case 2:
                            PlayerPrefs.SetInt("registro_Musgosu",1);
                            break;
                        case 3:
                            PlayerPrefs.SetInt("registro_Cuelebre",1);
                            break;
                        case 4:
                            PlayerPrefs.SetInt("registro_Tarasca",1);
                            break;
                    }
                }
            }

            else
            {
                bar_analisis.StopAnalizar();
            }
        }
        else
        {
            bar_analisis.StopAnalizar();
        }
    }
}
```

21. Script BarraAnalisis.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BarraAnalisis : MonoBehaviour
{

    public bool analizando;
    private float valor_actual;
    private float velocidad = 25f;
    private float vel_descenso = 100f;
    public bool analizado;
```

```
private void Awake() {
    valor_actual = 0;
    analizando = false;
    analizado = false;
    transform.localScale = new Vector3(1,0,1);
}

// Update is called once per frame
void Update()
{
    if(analizando && !analizado)
    {
        valor_actual = (valor_actual + (velocidad*Time.deltaTime));
        if(valor_actual >= 100)
        {
            valor_actual = 100;
            analizado = true;
        }
        transform.localScale = new Vector3(1,valor_actual/100,1);
    }
    else{
        if(valor_actual <= 0)
        {
            analizado = false;
            valor_actual = 0f;
        }
        else{
            valor_actual = (valor_actual - (vel_descenso*Time.deltaTime));
        }
        transform.localScale = new Vector3(1,valor_actual/100,1);
    }
}

public void Analizar()
{
    analizando = true;
}

public void StopAnalizar()
{
    analizando = false;
}

public void Iniciar()
{
    valor_actual = 0;
    analizando = false;
    analizado = false;
```

```
        transform.localScale = new Vector3(1,0,1);
    }
}
```

22. Script MensajeAnalisis.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MensajeAnalisis : MonoBehaviour
{
    private Text text;
    private int num_Trentis;

    private void Awake() {
        text = this.GetComponentInChildren<Text>();
        num_Trentis = 4;
    }

    // Start is called before the first frame update
    void Start()
    {
        Destroy(this.gameObject, 4);
    }

    public void BestiarioActualizado()
    {
        text.text = "Se ha añadido una nueva entrada al bestiario";
    }

    public void EntradaExistente()
    {
        text.text = "Ya hay una entrada en el bestiario de este ser";
    }

    public void CuentaTrenti()
    {
        text.text = "Trentis " + (num_Trentis -
                    ((GameObject.FindGameObjectsWithTag("Trenti").Length)-1)) +
                     " / " + num_Trentis;

        if((num_Trentis - ((GameObject.FindGameObjectsWithTag("Trenti").Length)1))
           == num_Trentis)
        {
            GameObject.FindObjectOfType<Misiones_Controller>().Cambiar_Mision();
```

```
        }
    }

}
```

23. Script Brujula.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Brujula : MonoBehaviour
{



    public GameObject flecha_dir;

    public GameObject panel_info;

    public bool brujula_activa;
    public int activarpanel;

    void Awake() {
        activarpanel = 0;
        brujula_activa = false;
    }

    // Update is called once per frame
    void Update()
    {
        activarpanel = 0;
        this.transform.position = Camera.main.transform.position;

        if(brujula_activa)
        {
            foreach(GameObject f in GameObject.FindGameObjectsWithTag("Flecha_Brujula"))
            {
                if(f.GetComponent<Direccion_flecha>().cerca)
                {

                    activarpanel++;
                }
            }

            if(activarpanel > 0)
            {
                ActivarPanelInfo();
```

```
            }

            else
            {
                DesactivarPanelInfo();
            }
        }
        else{
            DesactivarPanelInfo();
        }
    }

    public void addObjetivo(GameObject target)
    {
        GameObject flecha = Instantiate(flecha_dir, new Vector3(0,0,0),
                                        Quaternion.identity) as GameObject;

        flecha.transform.parent = this.transform;
        flecha.transform.localPosition = new Vector3(0, -0.25f, 0);
        flecha.GetComponent<Direccion_flecha>().SetObjetivo(target.transform);
    }

    public void deleteObjetivos()
    {
        GameObject[] flechas = GameObject.FindGameObjectsWithTag("Flecha_Brujula");
        foreach( GameObject f_go in flechas)
        {
            Destroy(f_go);
        }
    }

    public void ActivarPanelInfo()
    {
        panel_info.SetActive(true);
    }

    public void DesactivarPanelInfo()
    {
        panel_info.SetActive(false);
    }
}
```

24. Script Direccion_flecha.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class Direccion_flecha : MonoBehaviour
{
    public Transform objetivo;

    private float dis_objetivo;

    public bool activada;

    public bool cerca;

    void Awake() {
        cerca = false;
        activada = false;
    }

    // Update is called once per frame
    void Update()
    {
        if(activada)
        {
            if(!objetivo)
                Destroy(this.gameObject);

            else
            {
                dis_objetivo = Mathf.Sqrt(Mathf.Pow(this.transform.position.x -
                                                    objetivo.position.x,2) +
                                          Mathf.Pow(this.transform.position.z -
                                                    objetivo.position.z,2));

                if(dis_objetivo <= 50)
                {
                    cerca = true;
                    var meshes = this.GetComponentsInChildren<MeshRenderer>();
                    foreach(MeshRenderer m in meshes)
                    {
                        m.enabled = false;
                    }
                }
                else{
                    cerca = false;
                    var meshes = this.GetComponentsInChildren<MeshRenderer>();
                    foreach(MeshRenderer m in meshes)
                    {
                        m.enabled = true;
                    }
                    var rotacion = objetivo.position - transform.position;
                    rotacion.y = 0;
                    var rotacion_Quaternion = Quaternion.LookRotation(rotacion);
```

```
                    transform.rotation = Quaternion.Slerp(transform.rotation,
                                                          rotacion_Quaternion, 1);
                }
            }
        }

        else{
            var meshes = this.GetComponentsInChildren<MeshRenderer>();
            foreach(MeshRenderer m in meshes)
            {
                m.enabled = false;
            }
        }
    }

    public void SetObjetivo(Transform t)
    {
        objetivo = t;
    }

}
```

25. Script Menu_Cebo.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Menu_Cebo : MonoBehaviour
{
    public GameObject Indicator_place;
    public Button boton_accion;

    void Update()
    {
        var cebo_colocado = FindObjectOfType<Cebo>();
        if(cebo_colocado != null)
        {
            Indicator_place.SetActive(false);
            boton_accion.interactable = false;
        }

        else{
            Indicator_place.SetActive(true);
            boton_accion.interactable = true;
        }
```

```
    }
}
```

26. Script Cebo.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Cebo : MonoBehaviour
{
    private float dist_cuelebre;
    private GameObject cuelebre;

    void Update()
    {
        cuelebre = GameObject.FindGameObjectWithTag("Cuelebre");
        cuelebre.GetComponent<Cuelebre>().SeguirCebo();

        if(cuelebre != null)
        {
            dist_cuelebre = Mathf.Sqrt(Mathf.Pow(cuelebre.transform.position.x -
                                                 this.transform.position.x,2) +
                            Mathf.Pow(cuelebre.transform.position.z -
                                      this.transform.position.z,2));

            if(dist_cuelebre < 5f)
            {
                cuelebre.GetComponent<Cuelebre>().comiendo = true;
                Destroy(this.gameObject,60f);
            }
        }
    }
}
```

27. Script CruzSantaMarta.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CruzSantaMarta : MonoBehaviour
{
    public GameObject cruzPrefab;
    private GameObject cruz;

    private void Awake() {
```

```
        cruz = null;
    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetMouseButtonDown(0))
        {
            var crucesRezagadas = GameObject.FindGameObjectsWithTag("Cruz");
            foreach(GameObject c in crucesRezagadas)
            {
                Destroy(c);
            }
            RaycastHit hit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if(Physics.Raycast(ray,out hit))
            {
                if(hit.collider.tag == "CruzZ")
                {
                    cruz = Instantiate(cruzPrefab, hit.point,
                                       new Quaternion(0,0,0,0)) as GameObject;
                    cruz.transform.parent = Camera.main.transform;
                    cruz.transform.localRotation = new Quaternion(0,0,0,0);
                }
            }
        }

        else if(Input.GetMouseButton(0))
        {
            RaycastHit hit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if(Physics.Raycast(ray,out hit))
            {
                if(hit.collider.tag == "CruzZ")
                {
                    cruz.transform.position = hit.point;
                    cruz.transform.localRotation = new Quaternion(0,0,0,0);
                }
            }
        }

        else if(Input.GetMouseButtonUp(0))
        {
            Destroy(cruz);
        }
    }
```

```
    public void DestruirCruces()
    {
        GameObject[] cruces = GameObject.FindGameObjectsWithTag("Cruz");
        foreach (GameObject cruz in cruces)
        {
            Destroy(cruz);
        }
    }
}
```

28. Script CruzController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CruzController : MonoBehaviour
{
    void Update()
    {
        RaycastHit hit;
        if(Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward),
                           out hit, 45f))
        {
            if(hit.collider.tag == "Tarasca")
            {
                FindObjectOfType<Tarasca>().Amenazado();
            }
        }
    }

}
```

29. Script OrdenFinal.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OrdenFinal : MonoBehaviour
{
    public GameObject ordenFinal;

    // Update is called once per frame
    void Update()
    {
        var anjana = FindObjectOfType<Anjana>();
        if(anjana != null)
```

```
        {
            ordenFinal.SetActive(false);
        }
    }
}
```

30. Script Slot.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Slot : MonoBehaviour
{
    public Bestiario bestiario;
    public ListadoAcciones acciones;
    public Image imagenEntrada;
    public SlotInfo slotInfo;
    public PaginaBestiario paginaBestiario;
    public Acciones accionesScript;

    public void SetUp(int id)
    {
        slotInfo = new SlotInfo();
        slotInfo.id = id;
        slotInfo.EmptySlot();
    }

    public void UpdateUI()
    {
        if(slotInfo.isEmpty)
        {
            imagenEntrada.sprite = null;
            imagenEntrada.enabled = false;
            this.GetComponent<Button>().interactable = false;
        }

        else
        {
            Button boton_slot = this.GetComponent<Button>();
            boton_slot.interactable = true;

            if(bestiario != null)
            {
                imagenEntrada.sprite = bestiario.FindEntrada_Bestia(slotInfo.id).boton_imagen;
                paginaBestiario = (PaginaBestiario) FindObjectOfType(typeof(PaginaBestiario));
                boton_slot.onClick.AddListener(OpenPageBest);
```

```
            }
            else
            {
                imagenEntrada.sprite = acciones.FindAcciones(slotInfo.id).boton_imagen;
                accionesScript = (Acciones) FindObjectOfType(typeof(Acciones));
                boton_slot.onClick.AddListener(ActiveActions);
            }
            imagenEntrada.enabled = true;
        }
    }

    public void OpenPageBest()
    {
        paginaBestiario.AbrirPaginaBestiario(bestiario.FindEntrada_Bestia(slotInfo.id).name,
                                             bestiario.FindEntrada_Bestia(slotInfo.id).descripcion,
                                             bestiario.FindEntrada_Bestia(slotInfo.id).imagen);
    }

    public void ActiveActions()
    {
        switch(acciones.FindAcciones(slotInfo.id).id)
        {
            case 0:
                accionesScript.Onclick_Interactuar();
                break;
            case 1:
                accionesScript.Onclick_Analizar();
                break;
            case 2:
                accionesScript.Onclick_PonerCebo();
                break;
            case 3:
                accionesScript.Onclick_Brujula();
                break;
            case 4:
                accionesScript.Onclick_Cruz();
                break;
        }
    }
}

    [System.Serializable]
public class SlotInfo
 {
    public int id;
    public bool isEmpty;

    public void EmptySlot()
    {
```

```
        isEmpty = true;
    }
}
```

31. Script Inventario.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Inventario : MonoBehaviour
{
    [SerializeField]
    private Bestiario bestiario;
    public Transform bestiario_panel;
    [SerializeField]
    private ListadoAcciones acciones;
    public Transform acciones_panel;
    [SerializeField]
    private GameObject slotBestiarioPrefab;
    [SerializeField]
    private GameObject slotAccionesPrefab;
    [SerializeField]
    private List<SlotInfo> slotInfoListBestiario, slotInfoListAcciones;
    [SerializeField]
    private int num_slots;
    private int PosSlotLlenar;
    public GameObject Advertencia;

    //Se ejecuta antes del primer frame de refresco
    private void Start()
    {
        slotInfoListBestiario = new List<SlotInfo>();
        slotInfoListAcciones = new List<SlotInfo>();
        LoadInventario();
    }

    //Carga el bestiario
    private void LoadInventario()
    {
        //Crear slots bestiario
        for(int i = 0; i < num_slots; i++)
        {

            GameObject slot = Instantiate<GameObject>(slotBestiarioPrefab, bestiario_panel);
            Slot newSlot = slot.GetComponent<Slot>();
```

```
            Button buttonSlot = slot.GetComponent<Button>();
            newSlot.SetUp(i);
            newSlot.bestiario = bestiario;
            SlotInfo newSlotInfo = newSlot.slotInfo;
            slotInfoListBestiario.Add(newSlotInfo);
            buttonSlot.interactable = false;
        }

        //Crear slots Acciones
        for(int i = 0; i < num_slots; i++)
        {

            GameObject slot = Instantiate<GameObject>(slotAccionesPrefab, acciones_panel);
            Slot newSlot = slot.GetComponent<Slot>();
            Button buttonSlot = slot.GetComponent<Button>();
            newSlot.SetUp(i);
            newSlot.acciones = acciones;
            SlotInfo newSlotInfo = newSlot.slotInfo;
            slotInfoListAcciones.Add(newSlotInfo);
            buttonSlot.interactable = false;
        }

        AddEntradaBestiario(0);
        if(PlayerPrefs.GetInt("registro_Trenti",0)==1)
            AddEntradaBestiario(1);
        if(PlayerPrefs.GetInt("registro_Musgosu",0)==1)
            AddEntradaBestiario(2);
        if(PlayerPrefs.GetInt("registro_Cuelebre",0)==1)
            AddEntradaBestiario(3);
        if(PlayerPrefs.GetInt("registro_Tarasca",0)==1)
            AddEntradaBestiario(4);

        AddEntradaAccion(0);
        AddEntradaAccion(1);
        AddEntradaAccion(3);
        if(PlayerPrefs.GetInt("registro_PonerCebo",0)==1)
            AddEntradaAccion(2);
        if(PlayerPrefs.GetInt("registro_Cruz",0)==1)
            AddEntradaAccion(4);
    }

    //Busca si ya hay un slot con la información de la bestia del id que se le pasa
    private SlotInfo FindSlotOcupadoBestiario(int idEntrada)
    {
        //Si hay un slot con la información de la bestia devuelve ese slot
        foreach(SlotInfo slotInfo in slotInfoListBestiario)
        {
            if(slotInfo.id == idEntrada && !slotInfo.isEmpty)
            {
```

```
                return slotInfo;
            }
        }

        int contador = 0;
        //Si no lo hay devuelve el primer slot vacio
        foreach(SlotInfo slotInfo in slotInfoListBestiario)
        {
            if(slotInfo.isEmpty)
            {
                PosSlotLlenar = contador;
                slotInfo.EmptySlot();
                return slotInfo;
            }
            contador++;
        }

        //Si no hay slot disponibles no devuelve nada
        return null;
    }

    //Busca si ya hay un slot con la información de la accion del id que se le pasa
    private SlotInfo FindSlotOcupadoAccion(int idEntrada)
    {
        //Si hay un slot con la información de la bestia devuelve ese slot
        foreach(SlotInfo slotInfo in slotInfoListAcciones)
        {
            if(slotInfo.id == idEntrada && !slotInfo.isEmpty)
            {
                return slotInfo;
            }
        }

        int contador = 0;
        //Si no lo hay devuelve el primer slot vacio
        foreach(SlotInfo slotInfo in slotInfoListAcciones)
        {
            if(slotInfo.isEmpty)
            {
                PosSlotLlenar = contador;
                slotInfo.EmptySlot();
                return slotInfo;
            }
            contador++;
        }

        //Si no hay slot disponibles no devuelve nada
        return null;
    }
```

```
//Saca el Slot del menú desplegable del bestiario asociado a la posición PosSlotLlenar
private Slot FindSlotBestiario()
{
    return bestiario_panel.GetChild(PosSlotLlenar).GetComponent<Slot>();
}

//Saca el Slot del menú desplegable del listado de acciones asociado a la posición
//PosSlotLlenar
private Slot FindSlotAcciones()
{
    return acciones_panel.GetChild(PosSlotLlenar).GetComponent<Slot>();
}

//Añade una entrada al bestiario asociándole un determinado id
public void AddEntradaBestiario(int IdEntrada)
{
    Entrada_Bestia bestia = bestiario.FindEntrada_Bestia(IdEntrada);
    if(bestia != null)
    {
        SlotInfo slotInfo = FindSlotOcupadoBestiario(IdEntrada);
        if(slotInfo != null)
        {

            //Si el slot está vacio, lo rellenamos con la entrada y
            //alertamos de que se a añadido una nueva entrada al bestiario
            if(slotInfo.isEmpty)
            {
                slotInfo.id = IdEntrada;
                slotInfo.isEmpty = false;

                FindSlotBestiario().UpdateUI();
                GameObject ad = Instantiate(Advertencia,new Vector3(0,0,0),
                                            Quaternion.identity);
                ad.GetComponent<MensajeAnalisis>().BestiarioActualizado();
            }
            //Si la bestia ya tiene una entrada en el bestiario,
            //alertar de que ya existe una entrada
            else{
                if(!GameObject.FindGameObjectWithTag("Advertencia"))
                {
                    GameObject ad = Instantiate(Advertencia,new Vector3(0,0,0),
                                                Quaternion.identity);
                    ad.GetComponent<MensajeAnalisis>().EntradaExistente();
                }
            }
        }
    }
}
```

```
    //Añade una entrada al listado de acciones asociándole un determinado id
    public void AddEntradaAccion(int IdEntrada)
    {
        Accion accion = acciones.FindAcciones(IdEntrada);
        if(accion != null)
        {
            SlotInfo slotInfo = FindSlotOcupadoAccion(IdEntrada);
            if(slotInfo != null)
            {
                //Si el slot está vacio, lo rellenamos con la entrada y
                //alertamos de que se a añadido una nueva entrada al bestiario
                if(slotInfo.isEmpty)
                {
                    slotInfo.id = IdEntrada;
                    slotInfo.isEmpty = false;

                    FindSlotAcciones().UpdateUI();

                }
            }
        }
    }
}
```

32. Script ListadoAcciones.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "Inventory System/Listado de acciones")]
public class ListadoAcciones : ScriptableObject
{
     public List<Accion> acciones = new List<Accion>();

    public Accion FindAcciones(int id) {
        foreach (Accion a in acciones)
        {
            if (a.id == id)
            {
                return a;
            }
        }
        return null;
    }
}
```

```
[System.Serializable]
public class Accion
{
    public int id;
    public string name;
    public Sprite boton_imagen;
}
```

33. Script Bestiario.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "Inventory System/Bestiario")]
public class Bestiario : ScriptableObject
{
    public List<Entrada_Bestia> bestias = new List<Entrada_Bestia>();

    public Entrada_Bestia FindEntrada_Bestia(int id) {
        foreach (Entrada_Bestia b in bestias)
        {
            if (b.id == id)
            {
                return b;
            }
        }
        return null;
    }
}

[System.Serializable]
public class Entrada_Bestia
{
    public int id;
    public string name;
    public Sprite imagen;
    public Sprite boton_imagen;
    public string descripcion;
}
```

34. Script PaginaBestiario.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.UI;

public class PaginaBestiario : MonoBehaviour
{
    // Start is called before the first frame update
    public GameObject pagina;
    public GameObject Menu_Bestiario;
    public Text nombre_Bestia;
    public Text descripcion;
    public Image img_Bestia;

    public void AbrirPaginaBestiario(string name, string descr, Sprite img)
    {
        pagina.SetActive(true);
        Menu_Bestiario.SetActive(false);
        nombre_Bestia.text = name;
        descripcion.text = descr;
        img_Bestia.sprite = img;
    }

}
```

ANEXO 2: TEST REALIZADOS POR LOS USUARIOS

1. Prueba de usabilidad (SUS)

Prueba de Usabilidad para Iberian Odyssey

TFM del Master en Ingeniería del Software

de Vicente Murgui Sanchis

Apellidos: Nombre:

Edad:

En esta prueba se muestra 10 afirmaciones sobre la usabilidad de la aplicación. El usuario deberá rodear un numero en función de cuan de acuerdo esté con dicha afirmación, donde 1 equivale a “completamente en desacuerdo” y 5 a “completamente de acuerdo”.

-1. Creo que usaría este sistema frecuentemente.

1 2 3 4 5

-2. Encuentro este sistema innecesariamente complejo.

1 2 3 4 5

-3. Creo que el sistema fue fácil de usar.

1 2 3 4 5

-4. Creo que necesitaría ayuda de una persona con conocimientos técnicos

para usar este sistema.

1 2 3 4 5

-5. Las funciones de este sistema están bien integradas.

1 2 3 4 5

-6. Creo que el sistema es muy inconsistente.

1 2 3 4 5

-7. Imagino que la mayoría de la gente aprendería a usar este sistema en

forma muy rápida.

1 2 3 4 5

-8. Encuentro que el sistema es muy difícil de usar.

1 2 3 4 5

-9. Me siento confiado al usar este sistema.

1 2 3 4 5

-10. Necesité aprender muchas cosas antes de ser capaz de usar este sistema.

1 2 3 4 5

Una vez obtenidas las respuestas de los usuarios, se obtendrá un valor para cada pregunta en función de esas respuestas dadas: las preguntas **impares** tendrán un valor equivalente a la respuesta del usuario restándole 1, las preguntas **pares** tendrán un valor de 5 restándoles el valor de la respuesta dada. Con los **valores** de cada una de las preguntas obtenidos, se suma el total de ellos y se le multiplica por 2’5.

2. Prueba NASA-TLX

2.1 Test de evaluación de importancia

TEST DE EVALUACIÓN DE IMPORTANCIA COMO FUENTE POTENCIAL DE CARGA DE TRABAJO

De los pares de conceptos elige en cada uno de ellos cuál consideras que provoca una mayor carga de trabajo cuando un usuario emplea una aplicación.

Rodea el concepto seleccionado.

1.1 Demanda mental o Demanda temporal

1.2 Demanda mental o Demanda física

1.3 Demanda mental o Rendimiento

1.4 Demanda mental o Esfuerzo

1.5 Demanda mental o Frustración

2.1 Demanda física o Demanda temporal

2.2 Demanda física o Rendimiento

2.3 Demanda física o Esfuerzo

2.4 Demanda física o Frustración

3.1 Demanda temporal o Rendimiento

3.2 Demanda temporal o Esfuerzo

3.3 Demanda temporal o Frustración

4.1 Rendimiento o Esfuerzo

4.2 Rendimiento o Frustración

5.1 Esfuerzo o Frustración

2.2 Prueba de evaluación de carga de trabajo

Método NASA-TLX

para Iberian Odyssey TFM del Master de Ingeniería del Software

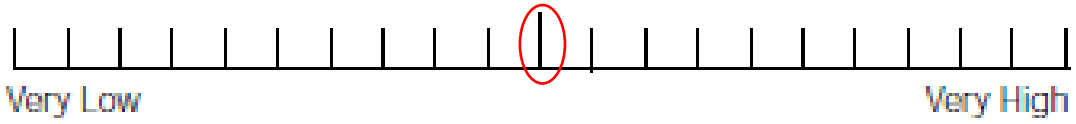
de Vicente Murgui Sanchis

Apellidos: Nombre:

Edad:

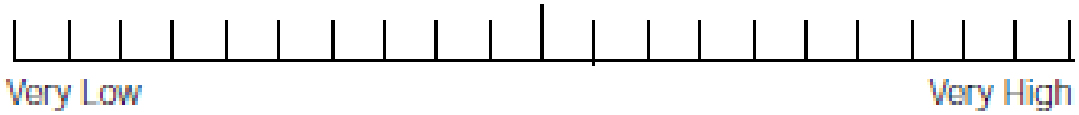
En las siguientes reglas, el usuario deberá rodear la línea vertical de la regla en función de la sensación que le haya transmitido la aplicación para cada uno de los siguientes conceptos:

Ejemplo



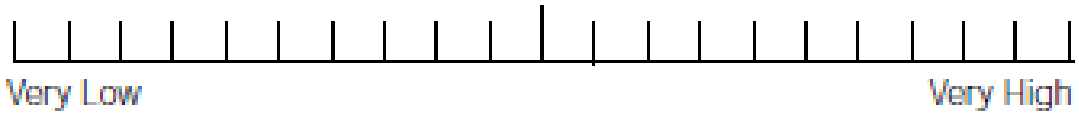
Demanda Mental

Cantidad de actividad mental y perceptiva que requiere la tarea (p. e.: pensar, decidir, calcular, recordar, mirar, buscar, etc.).



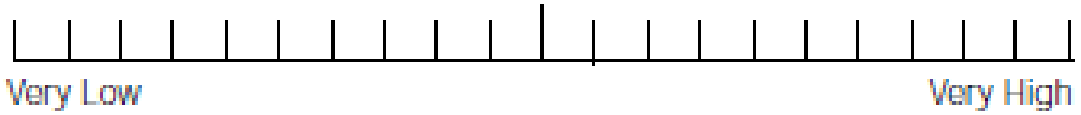
Demanda Física

Cantidad de actividad física que requiere la tarea (p.e.: pulsar, empujar, girar, etc.).



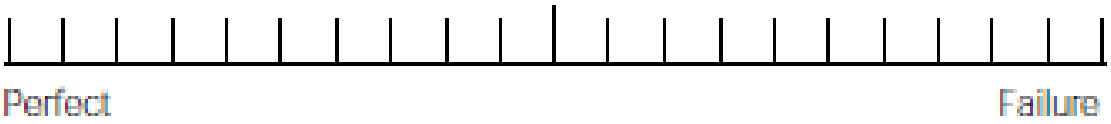
Demanda Temporal

Nivel de presión temporal sentida. Razón entre el tiempo requerido y el disponible.



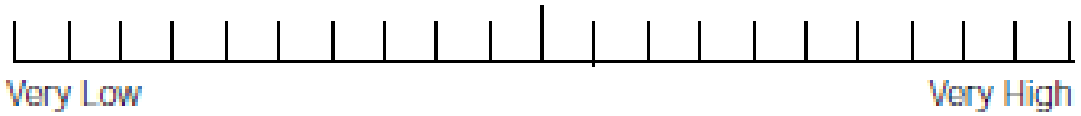
Rendimiento

Hasta qué punto el individuo se siente satisfecho con su nivel de rendimiento.



Esfuerzo

Grado de esfuerzo mental y físico que tiene que realizar el sujeto para obtener su nivel de rendimiento.



Frustración

Hasta qué punto el sujeto se siente inseguro, estresado, irritado, descontento, etc. durante la realización de la tarea.