



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

# DESARROLLO DE UN VIDEOJUEGO SERIO COMO HERRAMIENTA DE ACERCAMIENTO AL TEMA DE LA HISTORIA ESPACIAL Y LA ASTRONOMÍA

# DEVELOPMENT OF A SERIOUS GAME AS A TOOL OF APPROACHING THE SUBJECT OF SPACE HISTORY AND ASTRONOMY

Realizado por
**Pablo García López**

Tutorizado por
**Antonio José Fernández Leiva**

Departamento
**Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2023

Fecha defensa: septiembre de 2023

# Resumen

El objetivo de este Trabajo de Fin de Grado es la creación de un videojuego serio que sirva como herramienta de aprendizaje y acercamiento a la temática de la historia espacial.

Este proyecto consiste en una serie de pruebas que abarcan tanto preguntas como minijuegos, y que ponen a prueba la capacidad de aprendizaje del jugador, así como su habilidad. De esta forma, al acertar preguntas y superar los minijuegos, el jugador se verá recompensado y fomentado a seguir aprendiendo sobre el tema.

Estas pruebas se proponen en un contexto narrativo en el que el jugador tiene el papel de un habitante de una especie que proviene de los humanos, que tiene que llevar a cabo un rito por el que recorre una serie de pruebas para aprender más sobre sus antecesores. Aquí aparece la figura de otro miembro de la misma especie, que fue dejando unas notas por los escenarios en las que cuenta su experiencia y sirven de motor narrativo para el jugador.

Se utiliza tanto el componente narrativo como los minijuegos y preguntas para motivar al jugador a terminar el juego, así como para despertar o aumentar su interés en la historia de espacial y su relación con la humanidad.

**Palabras clave: Videjouego, juego serio, puzle, minijuegos, aprendizaje.**

# Abstract

The main goal for this End of Degree Project is the creation and development of a serious videogame that serves as a tool for learning and an approach to the subject of space history.

This Project consists of a series of trials that cover both quizzes and minigames, and put to the test the player's learning capabilities and ability. Thus, when getting the questions right and successfully overcoming the minigames, the player will get rewarded and encouraged to keep learning about the subjects.

These trials are presented in a narrative context in which the player has the role of an inhabitant of a species that comes from the humans, who has to perform a rite in which they have go over a series of trials to learn more about their predecessors. Here the figure of another member of the same species appears, who kept leaving some notes across the levels in which he narrates his own experience to the player, and it is used as a narrative hook.

The narrative component, the minigames and the quizzes are all used to motivate the player to finish the game, as well as to awake or increase their interest in the subject of space history and its relationship with humankind.

**Key words: Videogame, serious game, puzzle, minigames, learning.**

# Índice de contenido

# Índice de Figuras

# Índice de tablas

# Capítulo 1

# Introducción

En este capítulo se describen el contenido del proyecto, las motivaciones y objetivos, y por último la estructura de la memoria.

## 1.1 Motivación

El sector de los videojuegos es uno de los que ha experimentado un mayor crecimiento en los últimos años, llegando a superar a otros sectores como el cine o la música en cuanto a beneficios económicos se refiere. En España está reconocido desde 2009 por el Parlamento como una Industria Cultural, lo que implica una mayor cobertura a todos los niveles. Además, es un área que va más allá del ocio, ya que provoca efectos positivos en otros ámbitos, tales como el personal o el socioeconómico, pudiendo llegar a contribuir al progreso y mejora de la sociedad.

En este contexto empiezan a aparecer videojuegos que buscan no solo entretener, sino que tienen objetivos adicionales como el aprendizaje de nuevos conceptos e ideas o reforzar otras ya conocidas. Esta clase de videojuegos se conocen como videojuegos serios [1].

Esta clase de videojuegos pueden ser unas herramientas muy útiles a la hora de enseñar ciertos temas. Así, con la idea de acercar un tema como la historia espacial en mente surge este proyecto, con el objetivo no solo de evaluar al jugador, sino de que descubra el campo o aumente su conocimiento sobre este, así como divertirse por el camino.

## 1.2 Objetivo

El principal objetivo de este proyecto es el de desarrollar un videojuego serio que sirva para fomentar al jugador el aprendizaje sobre la historia espacial.

Para ello se ofrece un sistema de evaluación mediante preguntas tipo test, cuya información se expone a lo largo del nivel, y de varios minijuegos que persiguen lo mismo, pero añadiéndole variedad, ya que se busca que el jugador no solo termine el videojuego, sino que también disfrute y aprenda por el camino.

Estas pruebas se encuentran en un contexto narrativo que también avanzará conforme lo haga el jugador, añadiendo así otro aliciente para completarlo.

Estas pruebas, minijuegos e historia han pasado en primer lugar por una fase de desarrollo, seguido de una fase de implementación, para terminar con una de evaluación personal y refinado de mecánicas y errores. En todo este proceso también se ha hecho uso de la metodología ágil SCRUM [2] para planificar e implementar mejor dicho proceso.

## 1.3 Estructura de la memoria

A continuación se exponen los próximos capítulos de la memoria, así como un breve resumen de cada uno:

- *Antecedentes:* Se explican los principales conceptos involucrados en este proyecto, las tecnologías utilizadas y la metodología de trabajo seguida a modo de contexto para el lector de esta memoria.
- *Diseño:* En este capítulo se explica todo el proceso que se ha seguido para diseñar el videojuego, tanto los principales niveles como las mecánicas que más adelante acabarían siendo implementadas.
- *Implementación:* Capítulo donde se hace una explicación de cómo se ha implementado lo diseñado en el capítulo anterior, es decir, de cómo se ha generado el código, de cómo se ha ido adaptando conforme surgían necesidades y de cómo se han terminado de montar las distintas pantallas o niveles.
- *Conclusiones:* En este último capítulo se exponen las principales conclusiones sacadas de realizar este proyecto, así como algunas opiniones personales, comentar algunos errores e ideas que se han quedado por el camino y qué más se podría implementar en una futura continuación de este proyecto.

# Capítulo 2

# Antecedentes

Este capítulo sirve de contexto al lector acerca de los distintos términos, tecnologías utilizadas y métodos de trabajo. Es decir, este capítulo servirá para que el lector no tenga que depender en exceso de conocimientos previos a la lectura de esta memoria.

## 2.1 Términos

- **Juegos serios:**

Es el término que se utiliza para referirse a los juegos cuyo propósito no es solamente entretener, sino cuyo objetivo puede ser también el de aprender, concienciar, ayudar, etc. *Clark Abt.* acuñó este término en 1970, en su libro *Serious Games*. En el caso de este proyecto, se utiliza el entretenimiento proporcionado por el videojuego para aprender acerca de un tema concreto.

- **Itch.io:**

Es una plataforma especializada en la publicación de contenido relacionado con los videojuegos independientes. Esto abarca desde videojuegos completos hasta recursos como *sprites* (imágenes pixeladas estáticas o en movimiento), audio y música [3]. Su uso en este proyecto ha sido el de facilitador de recursos de uso libre para poderlos incorporar al propio videojuego.

## 2.2 Tecnologías utilizadas

- **Unity:**

*Unity* [4] es un motor de juego gratuito, es decir, una herramienta que proporciona todos los recursos necesarios para poder desarrollar un videojuego. Pese a que *Unity* permite desarrollar tanto videojuegos en 3D como en 2D, para este proyecto se ha decidido utilizar la vertiente 2D. *Unity* destaca por ser un motor de juego ampliamente extendido a lo largo de la industria, y esto implica la existencia de numeroso material de documentación que se ha consultado para el desarrollo de este proyecto [5].

- **C#:**

Es un lenguaje de programación multiparadigma creado por *Microsoft* para su plataforma *.NET*, y su lenguaje deriva de *C* y *C++* [6]. Fue escogido como

lenguaje de programación para este proyecto debido a que es un lenguaje que *Unity* soporta de manera nativa y la propia herramienta recomienda su uso para el desarrollo. Para las consultas relacionadas con este lenguaje, *Microsoft* proporciona una página con documentación [7].

- **Visual Studio Code:**

Es el *IDE* utilizado para escribir el código. Fue elegido principalmente por su directa integración con *Unity*, además de ser recomendado durante la instalación de este.

- **GIMP:**

Programa de software libre que sirve como herramienta de edición de imágenes. En este proyecto se ha utilizado para editar o recortar algunos *sprites* ligeramente cuando ha sido necesario.

- **Piskel:**

Programa de software libre que sirve como herramienta de edición de *sprites*. Se ha utilizado con la misma intención que *GIMP*, ya que este programa está más focalizado en *sprites*.

- **Draw.io:**

Plataforma online gratuita que sirve como herramienta para diseñar distintos tipos de diagramas o esquemas. Para este proyecto, se ha utilizado para hacer los diagramas de flujos y de clase.

- **Krita:**

Programa gratuito de edición de imágenes centrado en el dibujo. En este proyecto se ha utilizado para hacer los bocetos de distintas partes del videojuego durante la fase de diseño.

- **Audacity:**

Programa gratuito de edición de archivos de audio. En este proyecto se ha utilizado en un caso muy concreto para alterar la música de un nivel a partir de la música de otro.

## 2.3 Metodología de trabajo

Para este proyecto se ha seguido la metodología ágil SCRUM. En ella, se divide el tiempo de desarrollo del proyecto en Sprints más pequeños para poder abordar las distintas etapas de una forma más eficiente. Los roles más típicos a la hora de abordar un proyecto mediante SCRUM son:

- *Scrum Master*: Es la persona encargada de gestionar el equipo de desarrollo del producto, organizar y dirigir las reuniones, y de ser el principal enlace con el cliente.
- *Equipo*: Es el conjunto de personas encargadas del desarrollo del producto.
- *Cliente o Product Owner*: Es la persona o conjunto de personas que demandan los servicios del equipo de desarrollo. Aportan las principales guías o pautas que debe tener el producto y se reúnen con el equipo para hacer seguimientos.

# Capítulo 3

# Diseño

En este capítulo se detalla el proceso por el cual se ha diseñado el videojuego, pasando por distintas fases, tales como la definición de requisitos, la realización de un *High Concept* [8] o un *Game Design Document* (GDD), entre otros.

## 3.1 Análisis de requisitos iniciales

Esta primera fase del desarrollo es de las más importantes, ya que se fijarán unas bases (aunque luego hayan sufrido cambios a lo largo del proceso) sobre las que se asentará el resto del proyecto. En pocas palabras, es una lluvia de ideas donde se fijan unos puntos básicos para el comienzo del desarrollo. Estos son:

- El videojuego deberá ser principalmente perteneciente al género de *puzles*.
- Se debe incluir un componente ligero de juego de *plataformas* para el desplazamiento del personaje.
- Los recursos utilizados (imágenes, música, etc.) serán *assets* gratuitos sacados de internet, para agilizar la implementación.
- Se incluirá un *sistema de evaluación con preguntas* tipo test para poner a prueba lo aprendido a lo largo del juego.
- Las preguntas planteadas serán de la temática *historia espacial*.
- Se ofrecerá un *componente narrativo* que servirá como gancho para que el jugador avance en los niveles.
- Debe ser un juego *accesible* para el mayor público posible, así que señalizarán lo mínimo los fallos.
- Los controles no deben ser muy complejos.
- Tendrá que haber un *Non-Player Character (NPC)* al menos, que sirva de guía al jugador.
- Los niveles se dividirán en dos tipos: el principal con los puzles y la historia, y otro que sirva de transición entre estos.
- Las preguntas guardadas se podrán modificar, para que el jugador pueda ampliar su experiencia.

## 3.2 High Concept

Una vez terminados los requisitos, se empieza con documentos como el *High Concept* (también llamado *Game Concept Document o GCD*), donde se suelen describir de forma general las principales características e inspiraciones de lo que acabará siendo el producto final. También se detallan otros aspectos como el público objetivo o la plataforma a la que estará destinado.

A continuación se incluye la versión final del *High Concept* creado para este proyecto:

### TÍTULO

*Rocket to Home*

### DECLARACIÓN DEL HIGH CONCEPT (HIGH CONCEPT STATEMENT)

*Follow the path. Solve the mistery. Get home. (Sigue el sendero. Resuelve el misterio. Vuelve a casa.)*

El jugador debe avanzar los niveles respondiendo preguntas y superando los puzles para poder conseguir la pieza de la nave y avanzar de planeta. También podrá disfrutar de una historia que se va desarrollando conforme va avanzando en los niveles.

### CARACTERÍSTICAS (FEATURES)

- Explora las mazmorras, que se expanden conforme avanzas (estilo metroidvania) y resuelve puzles de temática espacial y basados en físicas, así como recorre el mapa con un ligero componente de plataformas.
- Aprende sobre la historia de la humanidad en el espacio, así como de la cultura en torno a este, con preguntas estilo Trivial.
- Mejora tu nave, con las piezas que necesitas para avanzar al siguiente planeta.
- Pilota tu nave, en un sencillo minijuego para ir de un planeta a otro.
- Disfruta de la historia. Descubre por lo que han pasado tus antecesores en esta aventura que ahora te toca a ti emprender.

## MOTIVACIÓN DEL JUGADOR (PLAYER MOTIVATION)

Explorar el nivel, completar la historia y aprender datos sobre la historia del espacio (carrera espacial, mitología sobre algunos planetas, etc) por el camino.

## GÉNERO (GENRE)

Puzzle acompañado de mecánicas de juego de plataformas.

## PÚBLICO OBJETIVO (TARGET CUSTOMER)

Público joven, con o sin experiencia previa jugando a un videojuego.

## PUNTOS DE VENTA ÚNICOS (UNIQUE SELLING POINTS)

Mezclar historia real con la historia ficticia del propio juego, así como intercalarlo con puzles y elementos de juegos de plataformas.

## OBJETIVOS DE DISEÑO (DESIGN GOALS)

- Despertar el interés sobre el tema en quien lo juegue, tanto con las preguntas tipo trivial como con la ambientación.
- Enganchar con la historia propia, aunque esta tenga menor importancia que lo otro.
- Que sea sencillo de jugar, que cualquier persona del público objetivo sea capaz de poder jugarlo sin problemas.

## HARDWARE OBJETIVO (TARGET HARDWARE)

PC (Windows)

## PERSONAJES

En un principio habrá solamente 3 personajes principales, que todavía no tienen nombre:

- **Protagonista:** Es quien controla el jugador. Va a realizar el rito por el cuál recorrerá el mismo sendero que sus antecesores, cuyo objetivo es volver al planeta natal de tu especie.

- **Anterior:** Es el último que hizo el rito antes que el protagonista, y fue dejando fragmentos de su diario para narrar lo que le sucedió durante su viaje. Es el principal motor narrativo para el jugador.
- **Tutora:** Es quien te explica todo lo que vas a tener que hacer, así como introducirte a las mecánicas del juego y explicártelas en caso de duda.

## MÁS DETALLES

En cuanto al estilo artístico, el diseño de los planetas estará muy influenciado por la estética de juegos como *Dome Keeper* [9], *Risk Of Rain* [10] (sobre todo en la paleta de colores) u *Hyper Light Dirfter* [11] (para dar una sensación hogareña o misteriosa a los distintos planetas), y las mazmorras por otro lado beberán de un estilo futurista y ancestral de juegos como Dungeon of The Endless [12] e *Hyper Light Drifter (de nuevo)* o de películas como Alien: El Octavo Pasajero. Para la parte donde se maneja la nave, se busca hacer un "sistema solar" simplificado o en miniatura, al estilo de juegos como *Outer Wilds* [13] o *Astroneer* [14], así como representar visualmente los campos de gravedad al estilo de, por ejemplo, *Angry Birds Space* [15], y usar una interfaz al estilo de *Papers Please* [16].

## REFERENCIAS PARA EL DISEÑO

A nivel artístico:

Figura 3.1: Captura del videojuego Dome Keeper

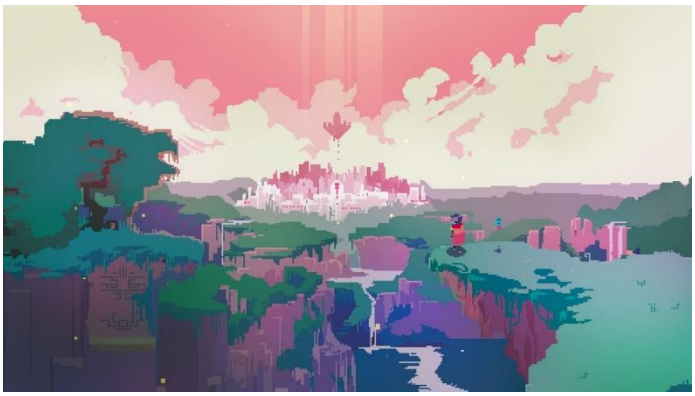

Figura 3.2: Captura del videojuego Hyper Light Drifter

Se buscará un contraste de colores para la superficie de los planetas, y de una forma parecida a la imagen de *Dome Keeper*, la superficie supondrá una pequeñísima parte que el jugador podrá recorrer, puesto que la chicha estará en las mazmorras. Por ejemplo, el jugador bajará de su nave, y al avanzar un poco hacia su derecha, encontrará la entrada a dichas ruinas.

Figura 3.3: Arte conceptual del videojuego Dungeon Of The Endless

Figura 3.4: Captura del videojuego Hyper Light Drifter

En cuanto a las mazmorras se busca dar la sensación de que han sido usadas por alguna otra especie que habitaba el sistema en algún momento muy distante, ya sean ruinas, laboratorios (esto se podría usar para justificar algunos puzles), etc.


Figura 3.5: Arte conceptual del videojuego Astroneer

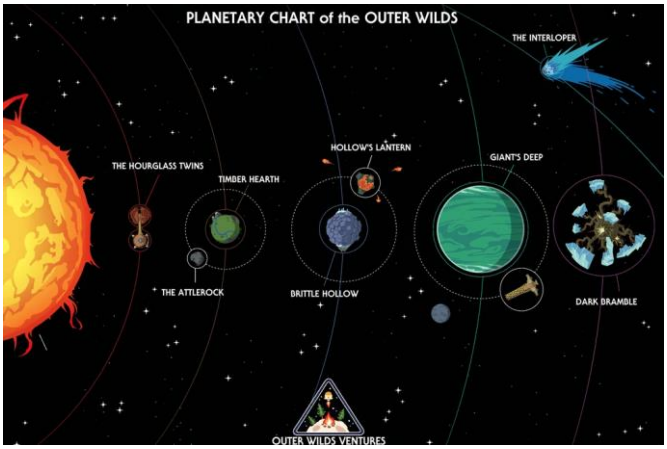


Figura 3.6: Arte conceptual del videojuego Outer Wilds

Para el sistema solar se busca hacerlo con unos planetas que presenten un diseño muy sencillo, pero al mismo tiempo fácil de reconocer. Como si fuera un punto intermedio entre usar distintos colores (como en el caso de *Astroneer*) y darle características únicas (como en el caso de *Outer Wilds*).

Para hacer más legible e interesante la mecánica de pilotar la nave, se buscará implementar un sistema sencillo de campos de gravedad, al estilo de Angry Birds Space, de forma que cuando entre con la nave, esta se verá atraída poco a poco por el planeta, hasta acercarse lo suficiente como para cambiar de pantalla y aterrizar. Para darle un poco más de variedad, se podría jugar con variedades de esto, para justificar narrativamente el uso de las nuevas piezas de la nave, como, por ejemplo, que la gravedad esté invertida.

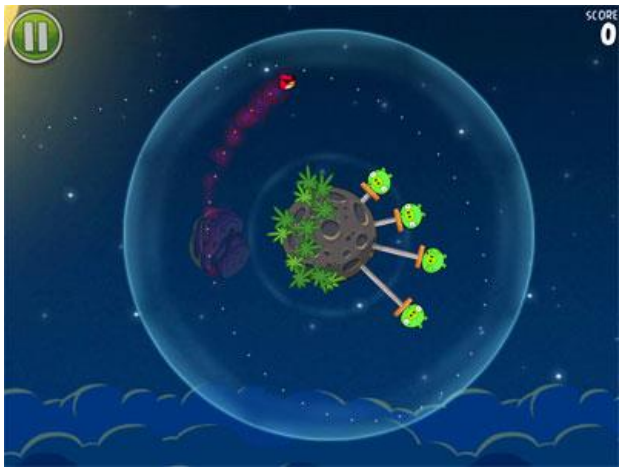

Figura 3.7: Captura de pantalla del videojuego Angry Birds Space

Para la parte de la interfaz de la nave, he pensado que se podría implementar algo del estilo de *Papers Please*, es decir, que se use el ratón para interactuar con los botones, que pueden consistir en impulsar para arriba y abajo, y girar para un lado u otro sobre sí mismo.

Figura 3.8: Captura de pantalla del videojuego Papers Please

## 3.3 Evolución de los requisitos y del GDD

A lo largo de la fase de diseño, así como durante el proceso de implementación, los requisitos indicados anteriormente han ido cambiando, ya fuera para ser modificados, borrados o añadidos. Al mismo tiempo, se ha creado y completando un *GDD* o *Game Design Document*, que recoge aspectos más concretos que lo encontrado en un *High Concept*. A continuación hay una muestra con las partes más importantes:

### 3.3.1 Mecánicas del juego

- **Movimiento:** En la fase del planeta, el jugador puede desplazarse lateralmente (a la izquierda o a la derecha) con las flechas o con las teclas A y D y saltar con

la barra espaciadora. En la fase del tipo Espacio el jugador puede desplazarse mediante las flechas (arriba o W para avanzar, abajo o S para retroceder, derecha o D para girar hacia la derecha e izquierda o A para girar a la derecha) o unos botones sobre los que puede hacer clic. En cualquiera de los dos tipos de fases se puede acceder al menú pulsando la tecla escape.

- **Interacción:** El jugador puede interactuar con distintos elementos siempre que se le indique por pantalla con la tecla E.
    - *Diálogos o paneles de información:* estos diálogos o paneles muestran información pertinente mediante texto por pantalla. Se puede avanzar con la tecla E, como se indica.
    - *Paneles de preguntas:* se accede a ellos con la tecla E, y para responder se deberá clicar con el ratón alguna de las posibles respuestas. Una vez hecho, para salir o continuar, se indica que hay que pulsar la tecla E.
    - *Cajas:* se cogen con la tecla E y se sueltan de nuevo con la tecla E.
    - *Portales:* se pulsa la tecla E para cruzar el portal.
    - *Cañón:* se utilizan las teclas A y D, o bien las flechas izquierda o derecha para aumentar o disminuir la potencia respectivamente, y las teclas W y S, o bien las flechas arriba y abajo para cambiar el ángulo de tiro, la barra espaciadora para disparar y Q para salir.
    - *Laberinto:* se utilizan las teclas de las flechas izquierda o derecha para girar la estructura para un lado o para otro, y la tecla Q para dejar de interactuar con el laberinto.
    - *Objeto esencial:* una vez interactuado se añade al inventario del jugador.
    - *Nave:* sirve para salir del nivel. Solo se puede utilizar una vez obtenido el objeto correspondiente del nivel y tener una puntuación de al menos 50%.
- **Guardar y Cargar:** El jugador puede guardar desde la propia partida y cargar dicha partida desde el menú principal. También se guarda automáticamente la partida al salir de un nivel o entrar en otro.
- **Menú:** Hay dos tipos de menú: el principal y el de juego.
    - Menú principal: aquí el jugador puede elegir si quiere empezar una nueva partida, cargar la partida o salir del juego haciendo clic con el ratón en el botón correspondiente.
    - Menú del juego: este menú se podrá abrir dentro de cualquier nivel siempre que no se esté interactuando con nada, y permitirá al jugador guardar la partida y salir al menú principal, o salir del menú y continuar.
- **Pruebas Principales:** Aquí se explica en mayor profundidad en qué consisten las principales pruebas que tiene que superar el jugador.
    - *Paneles de preguntas:* una de las formas de poner a prueba el conocimiento adquirido. Aquí se mostrará una pregunta y cuatro posibles repuestas, de

las cuales el jugador tendrá que elegir una y hacer clic con el ratón. Si el jugador acierta, le aumentará la puntuación media y podrá avanzar a la siguiente pregunta, o bien si ya ha respondido todas las de ese panel, lo habrá completado y conseguirá abrir la puerta. Si falla, entonces se le recomienda revisar las notas que ha tomado o la información directamente del panel. Cuando vuelva a intentar resolver el panel, se encontrará siempre con una pregunta distinta.

  - *Cubos:* esta prueba consiste en llevar un cubo con la respuesta correcta a una plataforma que muestra una pregunta. Cada caja tiene un color que corresponde con una respuesta. Si el jugador coloca la caja correcta en la plataforma, activará un mecanismo (este puede ser una plataforma móvil o una puerta). Si coloca una caja incorrecta, no pasará nada.
  - *Portales:* aquí se muestra una pregunta al principio, y cuatro posibles respuestas, cada una con un portal. Si se interactúa con el portal que representa la respuesta correcta, avanzará a la siguiente pregunta, y si falla, retrocederá a la pregunta anterior.
  - *Cañón:* esta prueba consiste en disparar una bola con el cañón para encestarla. Tiene tres fases, y al completarlas se abre la puerta correspondiente. En la primera fase la gravedad es normal, en la segunda es menor (la bola pesará menos) y en la tercera la gravedad es mayor (la bola pesará más).
  - *Laberinto:* esta es el último tipo de prueba, y en ella el jugador tiene que conseguir sacar el objeto esencial para poderlo recoger. Para ello tiene que girarlo y guiar la bola por el camino correcto.

- **Puntuación:** Para gamificar el progreso del jugador a lo largo del juego, existe un sistema de puntuación para los paneles de preguntas. Esta puntuación consiste en una media entre las preguntas acertadas y las preguntas contestadas. De esta forma, se fomenta que el jugador no vaya respondiendo al tuntún y tenga un objetivo en mente, además de avanzar en el nivel o la historia. Para poder salir del nivel, es necesario que el jugador haya respondido bien al menos un 50% de todas las preguntas que se le han mostrado. En caso contrario, al jugador habrá perdido la partida y tendrá que empezar desde cero, saliendo al menú principal y empezando una Nueva Partida.

### 3.3.2 Diagramas de estados

- General:

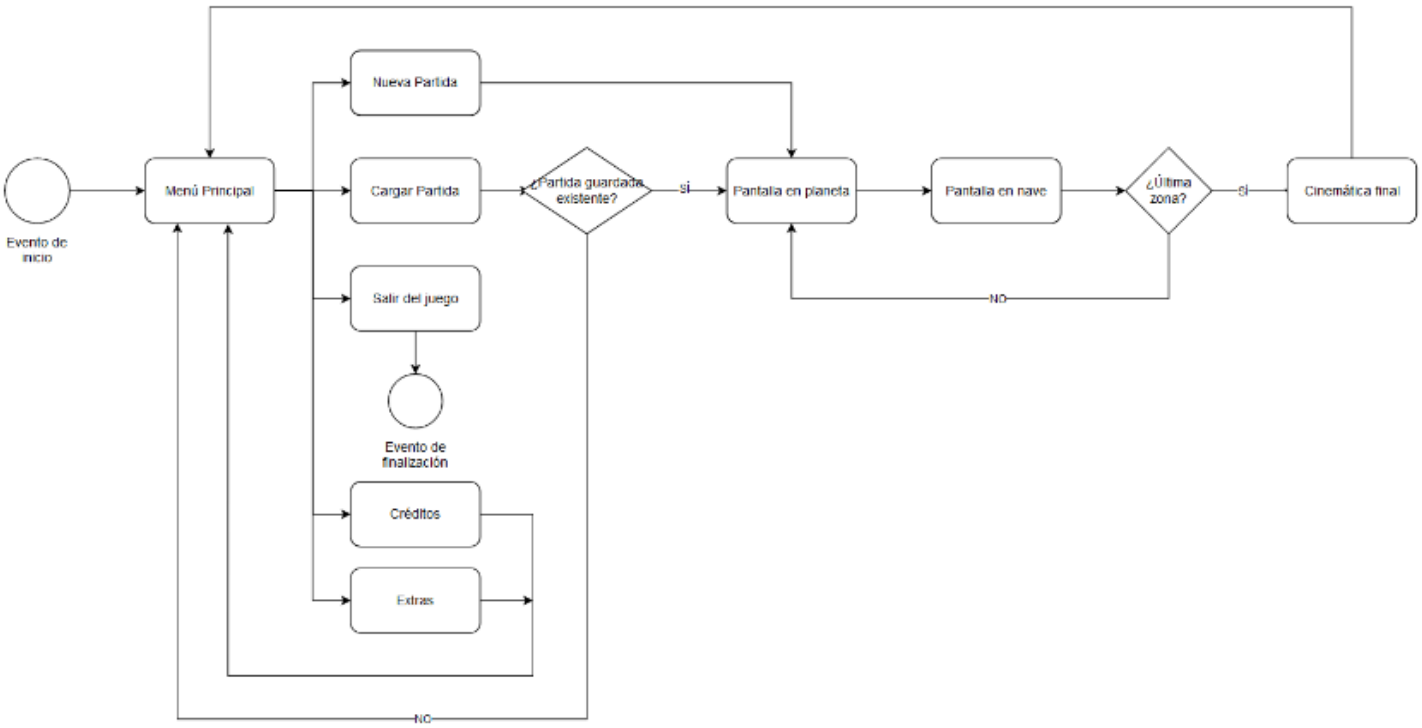


Figura 3.9: Diagrama de flujo general

- Fase de planeta 1:

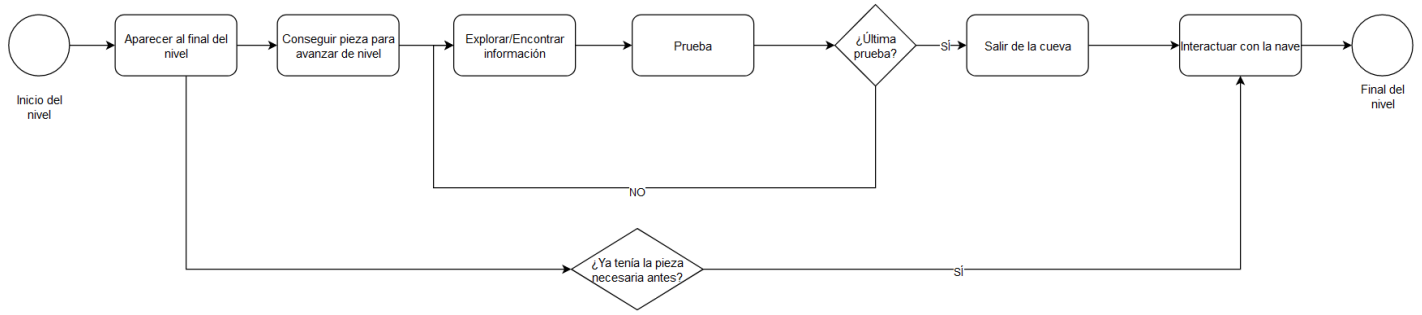


Figura 3.10: Diagrama de flujo del primer nivel

- Fase de planeta 2:

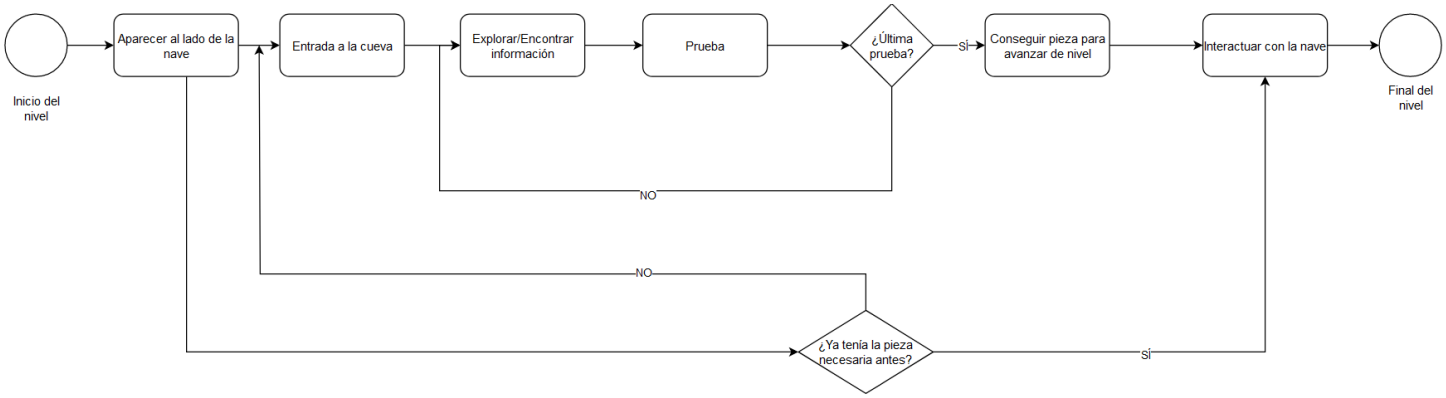


Figura 3.11: Diagrama de flujo del segundo nivel

- Fase de nave:

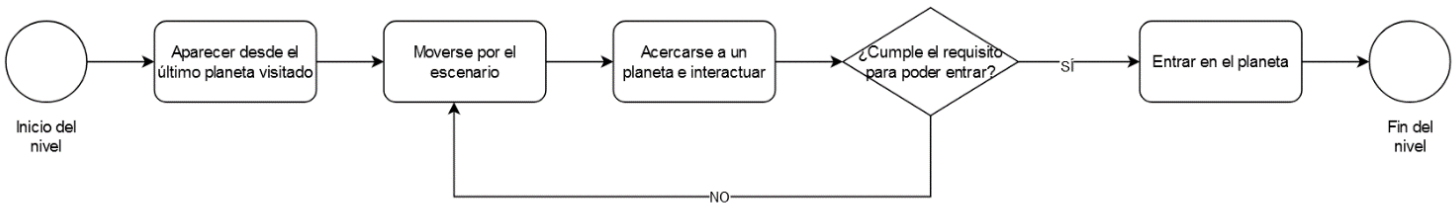


Figura 3.12: Diagrama de flujo de la fase del tipo Espacio

- Último nivel:

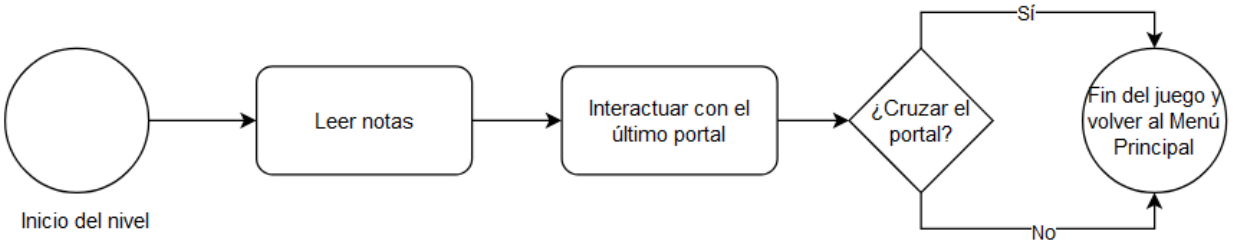


Figura 3.13: Diagrama de flujo del último nivel

Otras partes importantes del diseño, así como el resto del *GDD* se pueden encontrar en el *Apéndice A*.

Asimismo, la lista de requisitos finales es la siguiente:

- El videojuego deberá ser principalmente perteneciente al género de *puzles*.
- Se debe incluir un componente ligero de juego de *plataformas* para el desplazamiento del personaje.
- Los recursos utilizados (imágenes, música, etc.) serán *assets* gratuitos sacados de internet, para agilizar la implementación.
- Se incluirá un sistema de *evaluación con preguntas tipo test* para poner a prueba lo aprendido a lo largo del juego.
- Se incluirán varios tipos de *minijuegos*, para no depender en todo momento de las preguntas tipo test.
- Se ofrecerá un *componente narrativo* que servirá como gancho para que el jugador avance en los niveles.
- Debe ser un juego *accesible* para el mayor público posible, así que señalizarán lo mínimo los fallos.
- Al mismo tiempo, se busca *gamificarlo* añadiendo un *sistema de puntuación* de las preguntas, que será un requisito para poder avanzar.

- Los controles no deben ser muy complejos.
- Tendrá que haber un *NPC* al menos, que sirva de guía al jugador.
- Los niveles se dividirán en dos tipos: el principal con los puzles y la historia, y otro que sirva de transición entre estos.
- El juego tendrá que tener un *sistema de guardado de datos*, que permita al jugador reanudar la partida.
- Las preguntas guardadas se podrán modificar, para que el jugador pueda ampliar su experiencia.
- Se añadirá un segundo nivel, muy parecido al primero, pero con algún giro inesperado.
- Las preguntas serán sobre la *carrera espacial* para un nivel y la *astronomía y mitología* para el otro.

## 3.4 Sprints

En este proyecto los *sprints* han tenido una duración de unas dos semanas cada uno y han tratado los siguientes objetivos:

| SPRINT | OBJETIVOS |
|---|---|
| 12/06/2023 – 25/07/2023 | Análisis de requisitos, búsqueda de assets, documentación, implementación de elementos básicos. |
| 26/07/2023 – 09/08/2023 | Implementación de sistema de generación de preguntas, transición entre niveles, diseño e implementación de los principales minijuegos, sistema de guardado y menú. Se comienza el diseño de un primer nivel prototipo. |
| 10/08/2023 – 23/08/2023 | Diseño e implementación de un primer nivel prototipo e implementación del audio. |
| 24/08/2023 – 06/08/2023 | Periodo de testeo, detección de errores y pulido de estos. |
| 07/08/2023 – 20/08/2023 | Diseño e implementación de un segundo y de un último nivel, y escritura del guion del primer y último nivel. |
| 21/08/2023 – 30/08/2023 | Escritura del guion del segundo nivel, y periodo de testeo del segundo nivel y del juego completo. |

Tabla 3.1: Tabla de sprints

## 3.5 Cronograma

El siguiente cronograma muestra el progreso de las distintas tareas planificadas a lo largo de los sprints y del tiempo dedicado a cada una de ellas:


Figura 3.14: Cronograma del primer sprint

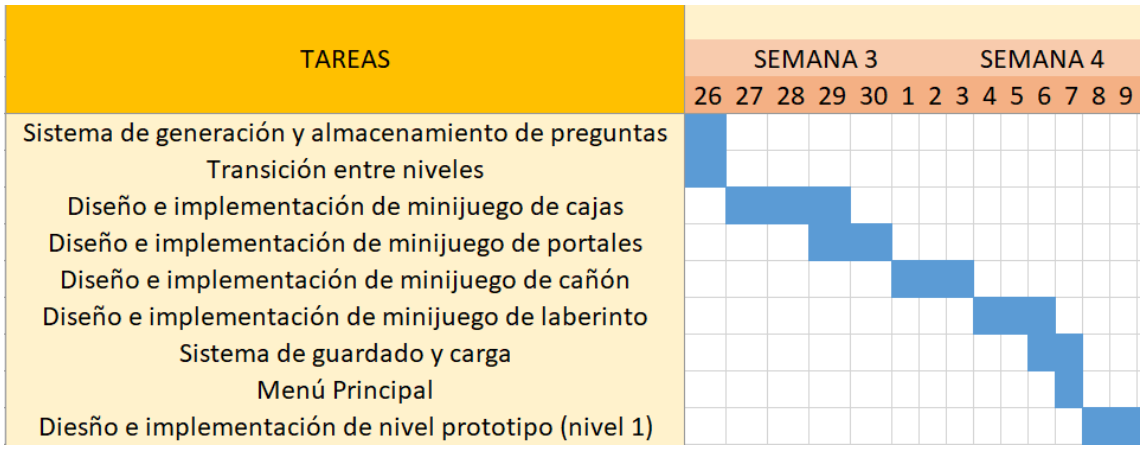


Figura 3.15: Cronograma del segundo sprint

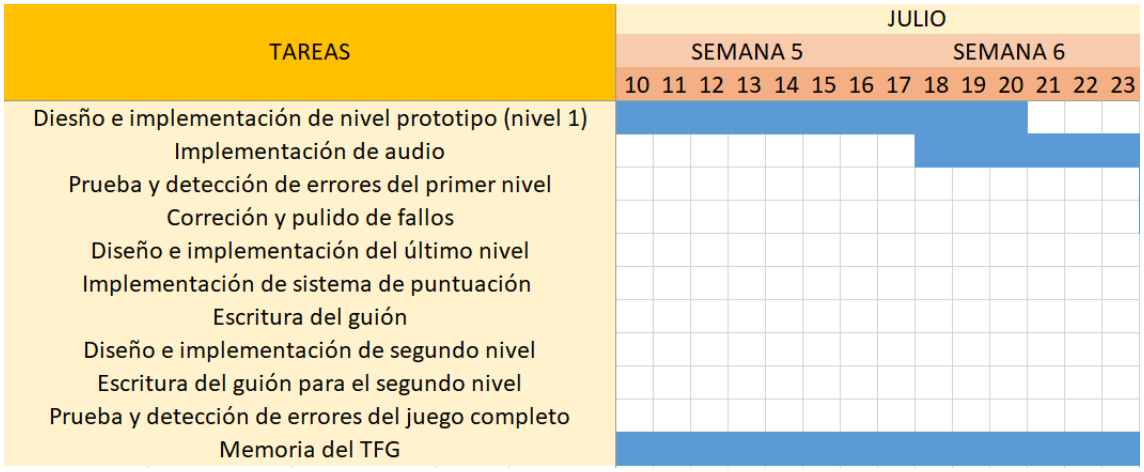


Figura 3.16: Cronograma del tercer sprint

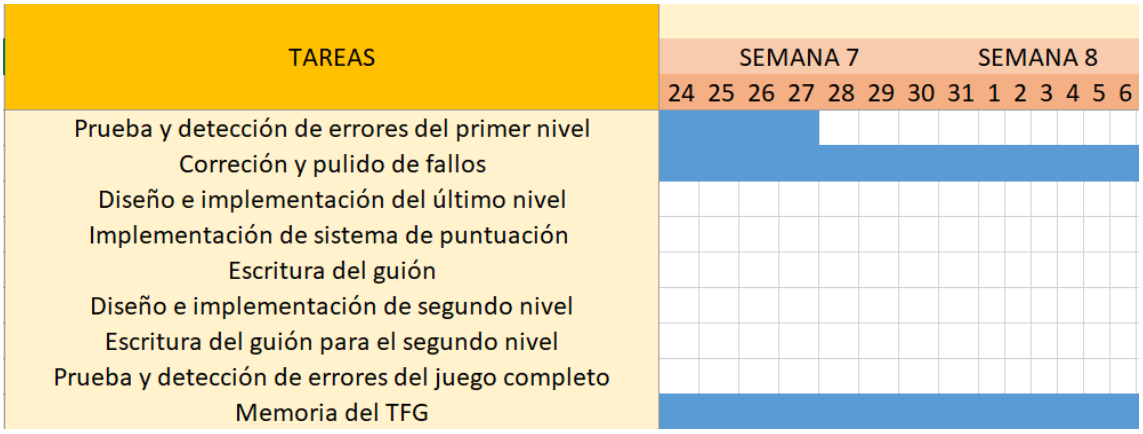


Figura 3.17: Cronograma del cuarto sprint

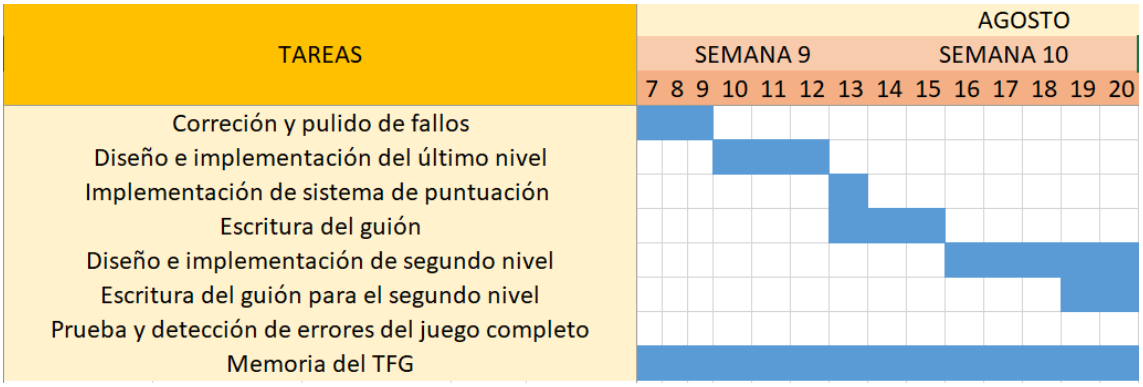


Figura 3.18: Cronograma del quinto sprint

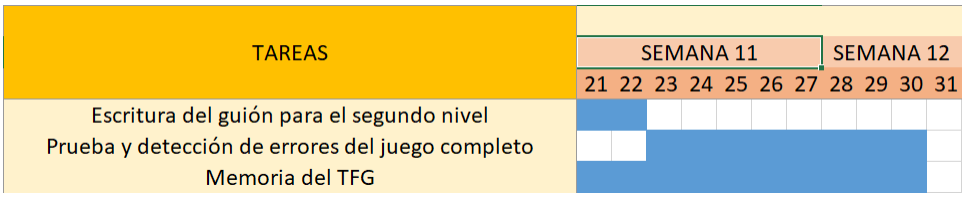


Figura 3.19: Cronograma del sexto sprint

# Capítulo 4

# Implementación

Una vez se ha terminado de diseñar los aspectos básicos del videojuego, comienza la fase de implementación de los elementos del juego y del código. Estos aspectos serán los que se detallarán en mayor profundidad en este capítulo, pasando por cada una de las distintas mecánicas y los distintos elementos que componen el producto final.

## 4.1 Personajes controlados por el jugador

Dentro del videojuego, al jugador se le permite controlar a dos personajes, siendo uno el principal y el segundo una nave espacial. En este bloque se irán detallando las distintas funcionalidades que tienen sendos personajes.

### 4.1.1 Personaje principal

El personaje principal es el primero de los dos principales elementos que el jugador controla directamente, y en este caso, solo se puede controlar dentro de los Niveles de tipo Planeta.

Figura 4.1: Sprite del personaje principal

En una primera instancia, y para facilitar el desarrollo, el *sprite* o imagen del jugador consistía en un cubo. Más adelante se acabó cambiando por la animación y *sprite* finales. El jugador se compone de un *Rigidbody2D* para su movimiento y físicas, así como de un *BoxCollider2D* para chocar con otros elementos de forma normal.

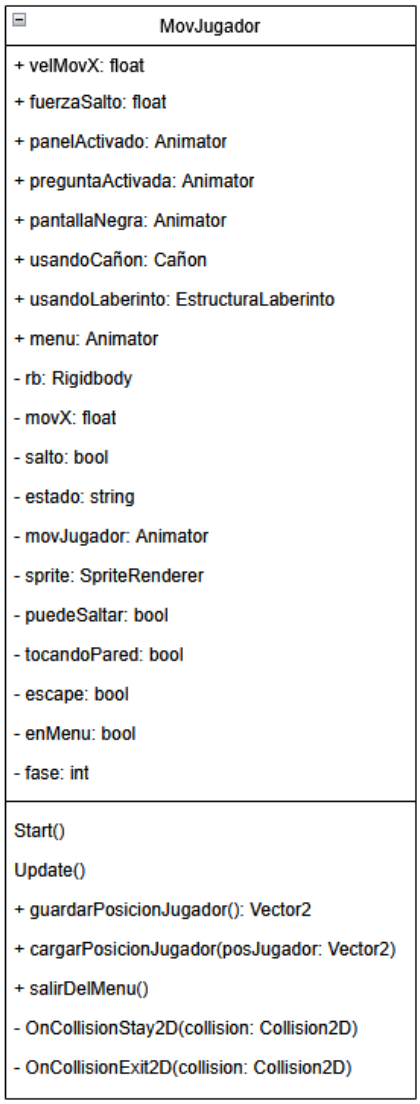


Figura 4.2: Diagrama de clase del movimiento del personaje principal

El código relacionado con su movimiento se encuentra en la clase *MovJugador*. Para dicho movimiento se hace uso del sistema de físicas proporcionado por *Unity*, en concreto, *velocity*, que añade una velocidad fija en un vector dirección a una velocidad que se le indique. Esta parte del código se activa controlando la pulsación de las teclas asociadas con el eje horizontal (flecha derecha, flecha izquierda, W y A) para el movimiento lateral, y la barra espaciadora para el movimiento vertical, es decir, el salto.

En este script también se controlan otros aspectos que determinan si el personaje puede moverse, así como otras condiciones en función de la pantalla en la que se encuentre. Estas condiciones son principalmente:

- Si el jugador está en el aire y toca una pared, no podrá moverse. Esto se hace para evitar que el jugador se quede atascado caminando hacia la pared.
- Si el jugador está interactuando con alguna prueba o panel, no podrá moverse. Esto se hace para evitar que el jugador se vaya cuando no debería.
- Si el jugador está en el menú, no puede moverse. Normalmente, en un videojuego, toda acción se pausa al entrar en un Menú.

También se incluye al final de la clase un código que permite guardar y cargar los datos cuando se llamen a los correspondientes métodos o funciones públicas, que se comentarán en la parte correspondiente al sistema de guardado.

El jugador contiene un inventario sencillo, que tiene en cuenta si el jugador posee el objeto esencial correspondiente al nivel o no, siendo estos dos objetos en total. Este código corresponde a la clase *InventarioJugador*, que además incluye métodos y funciones públicas que permiten a otras clases consultar o modificar el estado de dicho objeto mediante llamadas a *obtenerBola()* o *getBola()*. También se incluye, para evitar un fallo que surgió (se explica más en la sección *Problemas técnicos encontrados y dificultades*), una variable de control de posesión de cajas (*cajaEnMano()*), para evitar que el jugador pudiera tener en la mano más de una caja al mismo tiempo.

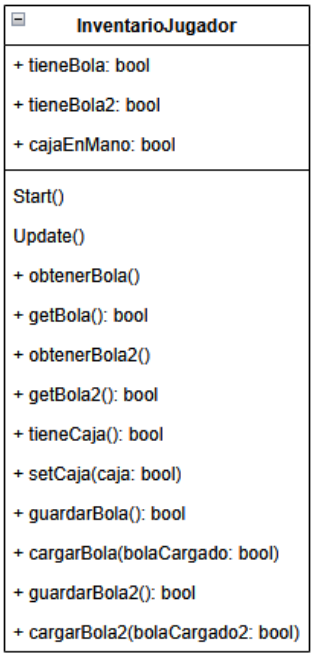


Figura 4.3: Diagrama de clase del inventario del jugador

La animación del personaje se controla con el componente *Animator*, y se ha seguido un proceso que también se puede aplicar al resto de elementos con componentes de animación que se incluyen en el proyecto [17]. En un primer lugar, se importaron los *sprite sheets* (hojas de *sprites*) correspondientes a distintos estados en los que se puede encontrar al personaje (siendo estos *Idle* o quieto, corriendo, saltando y cayendo). En segundo lugar, se recortan usando la herramienta *Sprite Editor* integrada en *Unity* para separarlas por fotogramas (en este caso usando el recorte automático). Después se crea una animación (*Animation*), que será correspondiente a un estado de la animación, y se le añaden los fotogramas correspondientes.

Aquí es donde se abre la ventana *Animator*, que es la encargada de organizar visualmente las transiciones a las diversas animaciones. En esta ventana se pueden añadir distintas condiciones por variables y transiciones para pasar de una animación a otra, aunque esas variables se acaben gestionando dentro del script de una clase (en este caso, *MovJugador*).

Ahora, dentro de la clase *MovJugador*, se comprueba si:

- El jugador está quieto, entonces mostrará la animación *Idle*.
- El jugador está moviéndose, mostrará la animación *Correr* mientras lo haga. Esta animación controla si está dirigiéndose hacia la derecha o hacia la izquierda, en cuyo caso activa o desactivará la variable *FlipX* del

componente *Sprite Renderer* (el componente encargado de gestionar la imagen que tiene el jugador).

- El jugador salta, mostrará la animación *Saltar* hasta que toque el suelo de nuevo.
- El jugador está cayendo (es decir, su valor de *velocity* en el eje Y es negativo), mostrará la animación *Caer*.

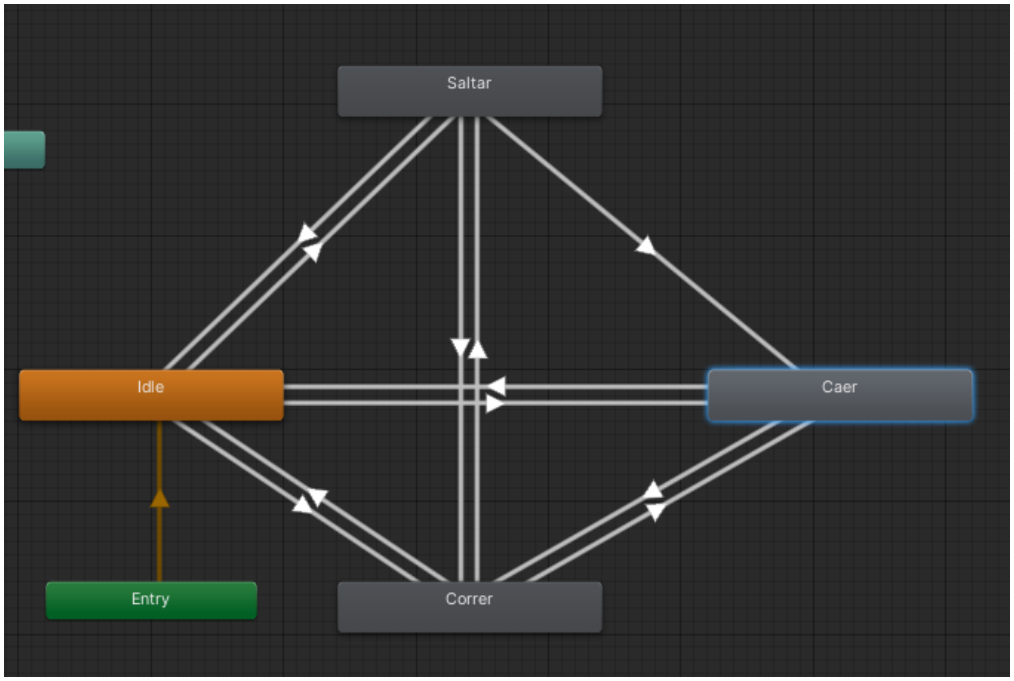


Figura 4.4: Animación del personaje principal

### 4.1.2 Nave espacial

La nave espacial es el otro elemento que el jugador controla directamente, y solo lo hace en las fases del tipo Espacio. Al igual que con el personaje principal, cuenta con un *Rigidbody2D* para su comportamiento físico, un *CapsuleCollider2D* para sus colisiones con otros elementos. En esta ocasión también cuenta con dos elementos hijos, los propulsores, que se explican más adelante.

Figura 4.5: Sprite de la nave espacial

El jugador puede mover la nave espacial de dos formas distintas: mediante las teclas *WASD* para avanzar y girar, o mediante unos botones que aparecen por pantalla en los cuales se puede hacer clic.

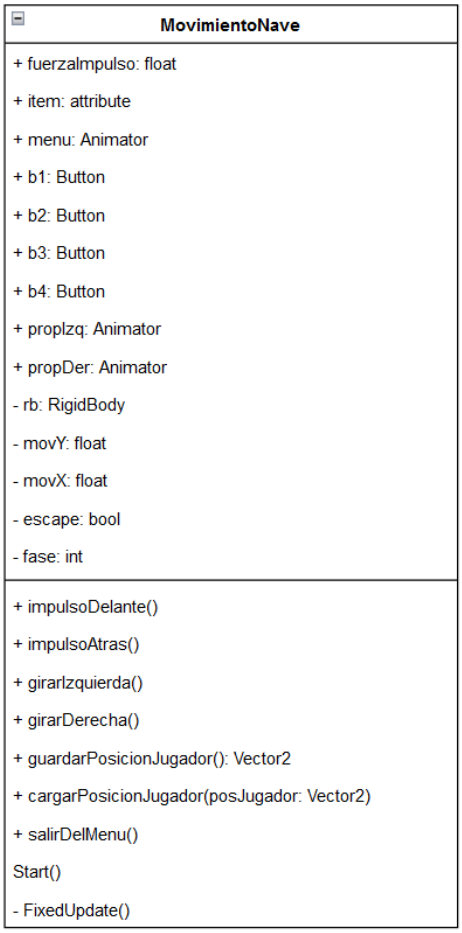


Figura 4.6: Diagrama de clase del movimiento de la nave espacial

Sea cual sea la forma que elija el jugador de moverse, ambas se recogen en la clase *MovimientoNave*. Los controles por teclado se recogen de una forma similar al del jugador principal, solo que, en este caso, para hacer más sencilla la

implementación de los botones, se llama al método público correspondiente al movimiento. Con *impulsoDelante()*, la nave avanza hacia delante aplicando *velocity*, al igual que con el jugador principal. Con *impulsoAtras()*, hace lo mismo, pero en dirección contraria. Con *girarIzquierda()*, se aplica torque (fuerza de giro de forma cinética) en sentido anti horario. Y, por último, con *girarDerecha*(), se hace lo mismo, pero en sentido horario.

Para los controles por botones, se recurre a una serie de botones (elementos *Button*) que forman parte de la interfaz mostrada en la pantalla en todo momento. Estos botones están hechos para que al hacer clic sobre ellos y mantenerlos pulsados [18] llamen a una función en concreto, que en este caso son las cuatro mencionadas antes asociadas al movimiento.

Figura 4.7: Botones de control de la nave

Para las animaciones asociadas a la nave espacial, se hace control de los dos elementos propulsores de la nave. Estos elementos tienen dos estados: *PropulsorNave_Apagado* y *PropulsorNave_Encendido*. En el mismo script *MovimientoNave* se controla que:

- Si la nave está quieta, entonces ningún propulsor está encendido y se desactivan sus variables de transición de estados.
- Si la nave avanza, ambos propulsores se encienden y se activan las dos variables.
- Si la nave gira hacia la derecha, el propulsor izquierdo se enciende y se activa la variable asociada al propulsor izquierdo.
- Si la nave gira hacia la izquierda, el propulsor derecho se enciende y se activa la variable asociada al propulsor derecho.

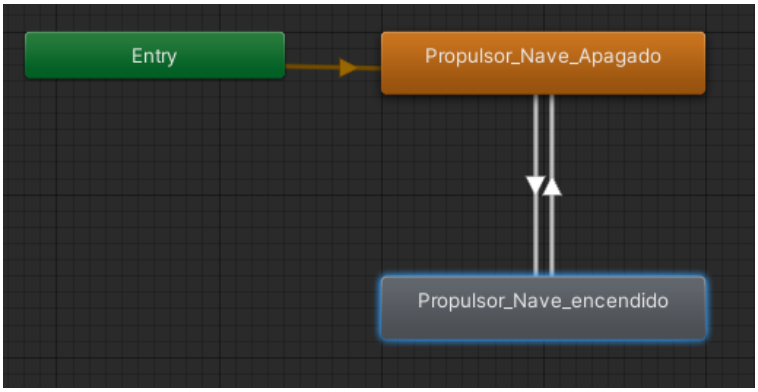

Figura 4.8: Animación de los propulsores

Por último, la nave espacial también incluye un script *ControlCamaraEspacio*, que le permite al jugador alejar la vista de la cámara para tener una idea más clara de a dónde puede ir. Se explica con más detalle en la siguiente sección.

## 4.2 Cámara

Uno de los elementos más esenciales de un videojuego es la cámara, puesto que sin ella no se vería nada en la pantalla. Pese a que el juego se divide en dos tipos de niveles (de *Planeta* y de *Espacio*), ambos utilizan la misma clase *ComportamientoCamara*.

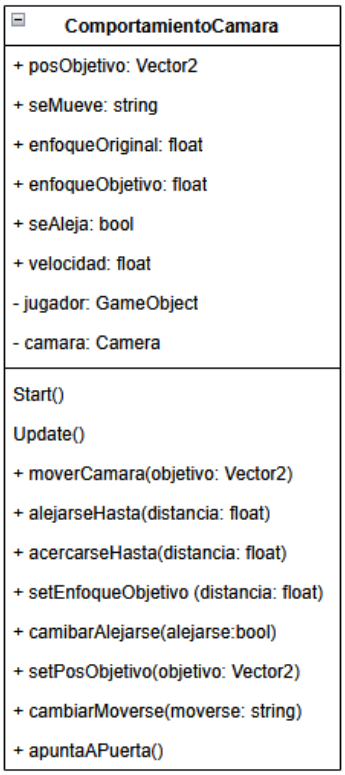

Figura 4.9: Diagrama de clase de la cámara

Como comportamiento inicial y básico, la cámara está programada para seguir en todo momento al jugador, a no ser que se le indique lo contrario (con la variable de estado *seMueve*, de tipo *string*), con una amplitud del encuadre que se le pasa inicialmente con una variable pública. Esta, al igual que la posición, puede cambiar en cualquier momento (con una variable *seAleja* del tipo *bool*).

La clase incluye una serie de funciones públicas que son accedidas por otros elementos (que se exponen a lo largo de este capítulo) las cuales controlan que la cámara se aleje o se acerque, o bien apunte a una posición concreta:

- La función pública *moverCamara*() mueve la cámara a una posición que esté a medio camino entre la posición que se le pasa por entrada y el jugador. Esto se consigue mediante *Lerp*, que devuelve una posición proporcional a dos posiciones que se le pasan, siendo el tercer valor la proporción deseada. En este caso, la proporción es 0'5, es decir, la mitad. También está programado para que este movimiento de la cámara sea progresivo, es decir, el valor de la proporción que se le pasa al *Lerp* empieza en 0 y va aumentando gradualmente hasta que llega a 0'5.
- Las funciones *alejarseHasta()* y *acercarseHasta()* utilizan el valor *orthographicSize* del componente *Camera* para indicarle la amplitud del encuadre a la que se tiene que alejar o acercar. Esto lo hace gradualmente hasta que llega al número objetivo.
- Las funciones públicas *setEnfoqueObjetivo()* y *setPosObjetivo()* son las encargadas de proporcionar los nuevos valores del enfoque de la cámara y de la nueva posición que va a tomar.
- Las funciones públicas *cambiarAlejarse()* y *cambiarMoverse()* cambian respectivamente los valores de *seMueve* y *seAleja*. Normalmente esto tiene repercusiones en el bucle principal de la cámara (es decir, en *Update()*).
- Por último, la función pública *apuntaAPuerta()* es solamente utilizada por el elemento *Puerta* y hace una función similar a *moverCamara()*, solo que en este caso no se queda a la mitad, sino que se aleja casi del todo del jugador para centrarse en la puerta objetivo.

Como ya se ha mencionado al principio de esta sección, normalmente la cámara seguirá al jugador allá donde vaya, ya esté controlando el personaje principal o la nave espacial. En este último caso, se añade un control, mediante la pulsación de la tecla E, en la que se hace una llamada mientras dicha tecla esté

mantenida a la función alejarseHasta, para permitir al jugador tener una vista mejor de sus alrededores. En caso de que deje de pulsar la tecla, la cámara irá lentamente volviendo a su amplitud original. Esto se controla en la clase *ControlCamaraEspacio*.

## 4.3. Elementos interactuables

En esta sección se irán viendo uno por uno los distintos elementos con los que el jugador puede interactuar, tanto del tipo de nivel de planeta como del espacio, así como algunos elementos que interactúan indirectamente con este.

### 4.3.1 Cuadro de Diálogos, Panel interactivo, Notas y NPC

En el juego la información, así como la historia, se trasmiten a través de cuadros de diálogos, que son unos cuadros de texto que muestran por pantalla, por partes, una serie de frases deseadas [19]. Para conseguir esto se hace uso de tres clases: *Dialogo*, *VentanaDialogos* y *TriggerDialogo*.

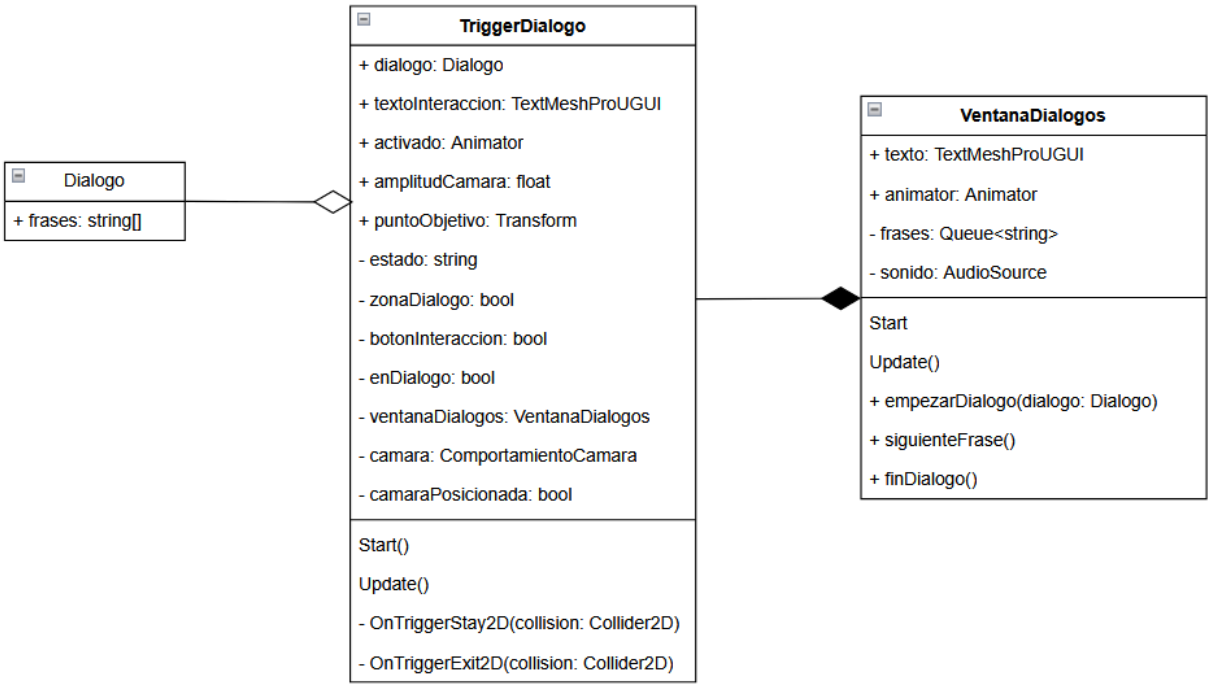


Figura 4.10: Diagrama de clases de los diálogos

Dialogo es una clase pública que contiene un array de frases, que son las distintas frases que incluye un diálogo a mostrar. Esta clase tiene la peculiaridad de no ser *MonoBehaviour*, por lo que se tiene que hacer *Serializable* para poderla

usar en otras clases de forma correcta. La principal función de esta clase es la de ser un esqueleto, una base, para simplificar el trabajo al utilizarlos en la clase *TriggerDialogo* y *VentanaDialogos*.

*VentanaDialogos* es la clase que se encarga de mostrar por pantalla el texto correspondiente del diálogo siempre que se le dé la orden. Esta clase tiene tres métodos públicos muy importantes, y son *empezarDialogo()*, *siguienteFrase()* y *finDialogo()*. La forma en la que funciona esta clase es la siguiente:

- La clase *TriggerDialogo* le pasa un *array* con las frases que *VentanaDialogos* tiene que mostrar. Esta las guarda en una pila vacía en *empezarDialogo* e inmediatamente llama a *siguienteFrase()*. También hace que empiece la animación correspondiente a la aparición por pantalla del cuadro de diálogos, que consiste en un sencillo desplazamiento para que aparezca en pantalla.
- *siguienteFase()* lo que hace es sacar la primera frase que encuentra en la pila de *string frases* y mostrarla por pantalla. Hará esto siempre que se llame a esta clase. En el caso de que se llame a esta función y a la pila ya no le queden más frases, entonces se llama a *finDialogo*().
- *finDialogo()* lo que hace es encargarse de llamar a la animación pertinente que hace que el cuadro de diálogos desaparezca de la pantalla.

*VentanaDialogos* también incluye un componente de audio [20], es decir, un *AudioSource*, el cual se utiliza dentro del script también (esto se aplica también a la mayoría de componentes de audio del resto de elementos). Este *AudioSource* tiene un *AudioClip*, que es el sonido o música a reproducir, *Volume*, para controlar el volumen, y *Spatial Blend*, que sirve para indicar si queremos que el audio sea en 2D (a 0) o 3D (a 1, con posibilidad de cambiarlo a algún número mayor que 0 y menor que 1). Este clip se reproduce cada vez que hace una llamada a *siguienteFase()*, y está configurado para que sea en 2D.

La clase pública *TriggerDialogo* se encarga de enviar las frases de diálogo a *VentanaDialogos*, así como indicarle cuando debe avanzar en este. Este diálogo se escribe en el inspector de *Unity*, puesto que se utiliza una variable de tipo *Dialogo*, y se pueden escribir libremente las frases correspondientes. Para controlar qué tiene que hacer en cada momento, existe la variable *estado* de tipo *string*, que cambia en función de si se está interactuando por primera vez, si no es así, o si el diálogo ya ha terminado. Esta clase suele estar asociada a los paneles

de información y al NPC, así que solo se puede activar e interactuar si el jugador está próximo. Si lo está y pulsa la tecla E, el diálogo da comienzo y se le retira momentáneamente el control al jugador, llamando entonces a *empezarDialogo()*, y luego cambiando *estado* a *"Conversación"*. Si se vuelve a interactuar, como ya está en mitad del diálogo, entonces llama a *siguienteFrase()*. Por último, para controlar si sigue o no en la conversación, se fija en la variable de control (*Animator.getBool()*) de la animación de VentanaDialogos, y si está a false, quiere decir que ya ha terminado el diálogo y entonces, le devuelve el control al jugador y reinicia el estado a *"PrimeraInteracción"*. En el caso de que el jugador no interactúe con este diálogo, este se reinicia automáticamente al salir de su *Trigger* para evitar que cuando el jugador vuelva a empezar la conversación, esta esté ya empezada. Esta clase también se encarga de llamar a la función *moverCamara()* y *cambiarMoverse()* para apuntar la cámara a una zona concreta que se le pasa por variable pública, así como *cambiarAlejarse()* y *alejarseHasta()*. Cuando termina el diálogo, le devuelve la cámara al jugador.

También incluye una llamada a otro texto que aparece por pantalla, llamado *TextoInteracción*. Este texto lo que hace es mostrar por pantalla las indicaciones para que el jugador pueda y sepa cómo interactuar con esta clase. No se vuelve a mostrar hasta que el diálogo haya terminado y el jugador vuelva a entrar a la zona en la que puede interactuar. Este texto también aparece en el resto de elementos interactuables.

Pasamos ahora al elemento *PanelInteractivo*. La función de este elemento es permitir que el jugador interactúe con él y mostrar la información pertinente por pantalla. Estos paneles se encontrarán en unas salas concretas del mapa y, desde el punto de vista de la gestión de la información, parten esta en bloques (por ejemplo, en el primer nivel de planeta, dividen la historia de la carrera espacial en los bloques de *"Principios y lanzamiento de satélites"*, *"Seres vivos en el espacio"* y *"Aterrizaje en la Luna"*). Este elemento hace uso de la clase *TriggerDialogo* para poder decirle qué frases debe enviar a *VentanaDialogos*.

Figura 4.11: Sprite del panel interactivo

Existen dos variantes del panel de diálogos, es decir, funcionan de la misma manera, pero varían en su aspecto y función en el juego. Estos son las notas dejadas por Anterior y el *NPC* de la Tutora.

Las notas dejadas por Anterior son el principal reclamo narrativo de este proyecto, en el sentido de que van exponiendo conforme el jugador avanza en los niveles, la historia de este y de su experiencia haciendo el mismo recorrido que el jugador.

Figura 4.12: Sprite de la nota dejada por Anterior

El *NPC* principal, la Tutora, corresponde a la guía del jugador. Este único *NPC* aparece justo al principio del videojuego y se encarga de hacer un resumen de los controles, de lo que tiene que hacer el jugador para avanzar, y de darle algún que otro consejo. Toda esta información siempre está presente todas las veces que el jugador la necesite. La otra principal diferencia con los paneles de información es que este *NPC* tiene animación. Esta animación es única, es decir, tiene un solo estado, por lo que no hace falta controlarla en ninguna clase, y es una sencilla animación *Idle*, que se repite en bucle.

Figura 4.13: Sprite del NPC Tutora

Por último, en el último nivel, se encuentra una variante algo más compleja de diálogo, que además añade una elección en la que al jugador se le muestra por pantalla una elección con dos botones, y en función de cuál pulse, se mostrará un diálogo u otro, es decir, pasará uno de dos elementos *Dialogos* a *TriggerDialogos*. Esto se recoge en la clase *PortalFinal*.

### 4.3.2 Cuadro de preguntas, Panel de preguntas, Base de datos de preguntas y Puntuación

Una de las principales pruebas que se encuentra el jugador a lo largo de su aventura son unos paneles que pueden contener una serie de preguntas que, por norma general, se corresponden con la zona en la que se encuentren, es decir, con los bloques de información que se han expuesto hasta ese momento. Estos paneles de preguntas hacen uso de una base de datos de preguntas, que luego muestran por pantalla para que el jugador las responda, y actúan en consecuencia a la respuesta.

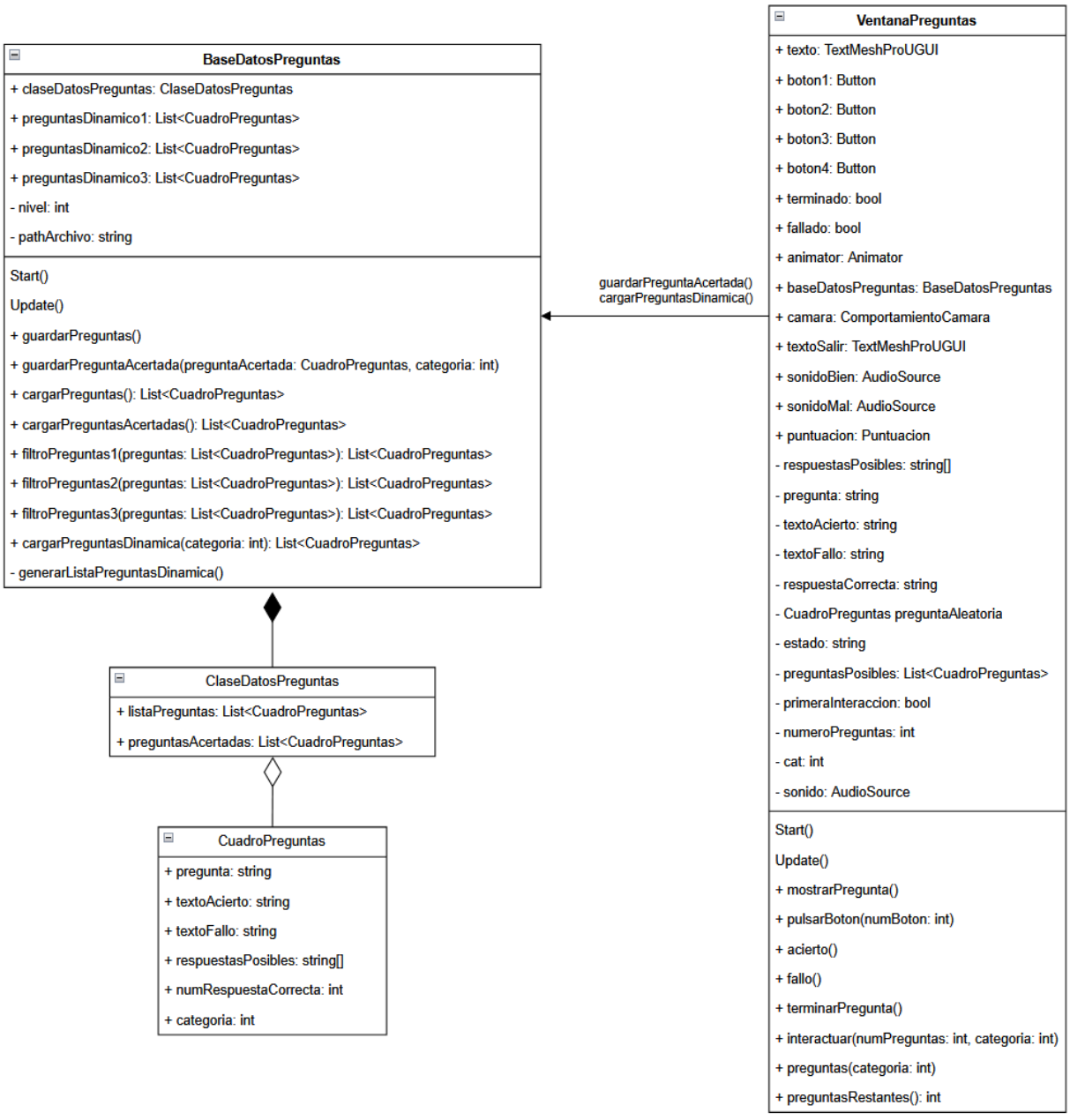


Figura 4.14: Primer diagrama de clases de las preguntas

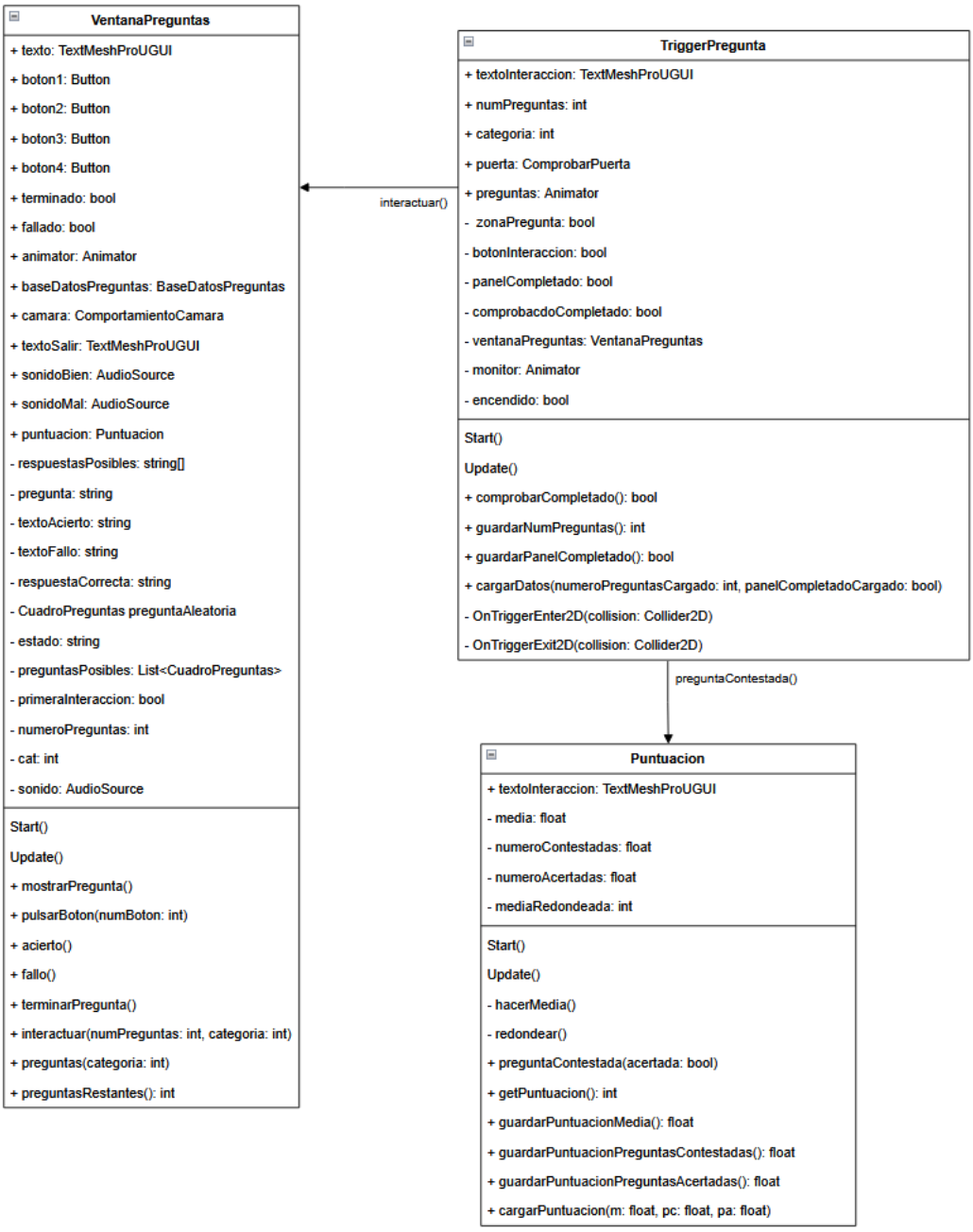


Figura 4.15: Segundo diagrama de clases de las preguntas

El principal elemento de esta sección es la base de datos de preguntas, que se encuentra en la clase pública *BaseDatosPreguntas*. Esta clase, en pocas palabras, genera una serie de preguntas con el esqueleto de otra clase pública *CuadroPreguntas*, que se guarda en un archivo JSON junto con una lista de las preguntas acertadas, las separa y clasifica en función de su categoría o bloque de información al que pertenecen, las suministra cuando se le requiere, y las modifica de forma dinámica conforme se van respondiendo. Una ventaja de guardarlo en JSON es la posibilidad de modificar la base de datos de las preguntas, es decir,

modificar, añadir o quitar las preguntas que se quieran. Esto se indica en la pestaña *Extras* del Menú Principal.

Empezamos por la clase pública *CuadroPreguntas*, que, de una forma similar a lo visto en la clase *Dialogo*, guarda la estructura básica que tendrán las preguntas, siendo esta el texto de la pregunta, las posibles respuestas, la respuesta correcta, la categoría, así como unos textos que se muestran en función de si se acierta o se falla al responder. Esta clase se llama *CuadroPreguntas*, y se utiliza en la clase pública *ClaseDatosPreguntas* de forma que contiene una lista de preguntas para poder incluir varias de golpe, y otra de preguntas acertadas. Esta clase entonces se utiliza en la clase *BaseDatosPreguntas* (en el mismo script) de forma pública y con *SerializeField* (al ser una clase creada dentro de otra, hace falta declararla así) para poder añadir de una forma más rápida las distintas preguntas desde el inspector de Unity, en lugar de añadirlas de una en una dentro del código.

Ahora pasamos al funcionamiento de la base de datos. Para empezar, se declaran otras tres listas de tipo *CuadroPreguntas*, que serán las que se vayan generando y modificando de forma dinámica conforme se requiera. Las principales funciones y métodos de esta clase son:

- *guardarPreguntas()*: este método comprueba si existe un archivo donde se tengan guardadas las preguntas y, en caso de que no sea así, lo crea y guarda las preguntas así como las ya acertadas mediante la clase *ClaseDatosPreguntas* ya mencionadas anteriormente. Este guardado se realiza cogiendo el *path* que *Unity* suministra con *Application.persistentDataPath* (para evitar confusiones en caso de que el programa se utilice en un ordenador distinto al mío) seguido de un nombre para el archivo, que en este caso es *"Preguntas.json"* (o *"Preguntas2.json"* si es el segundo nivel).
- *cargarPreguntas()*: esta función lee del archivo creado con el método anterior, guarda las preguntas de tipo *CuadroPreguntas* en una lista de *CuadroPreguntas* y la devuelve.
- *cargarPreguntasAcertadas()*: hace los mismo que la función anterior, solo que en este caso lee la lista de preguntas acertadas y las devuelve.
- *generarListaPreguntasDinamica()*: este es de los métodos más importantes de esta clase, pues solo se ejecuta una sola vez al empezar el nivel, y filtra

las listas *preguntasDinámico1/2/3* en función de las categorías de las preguntas y de si ya están acertadas o no. Este método llama a la función *filtroPreguntas1, 2 y 3*.

- *filtroPreguntas (1, 2 y 3)*: esta función lo que hace es generar una lista en las que se guardan las preguntas, sin contar las ya acertadas, en función de la categoría que estas tengan. Recorre la lista de preguntas una por una, empezando por quitar las que ya están acertadas. Entonces comprueba la categoría de las restantes, y si corresponden, las guarda en otra lista que devuelve. El funcionamiento de las categorías en estas funciones consiste en, si es de la primera categoría, solo guardará preguntas de categoría 1, si es de categoría 2, devolverá aquellas de categoría 1 y 2. Y así respectivamente, ya que se entiende que, cuanto más avance el jugador, mayor será el abanico de información que pueden abarcar las preguntas.
- *cargarPreguntasDinamica*(): esta función lo único que hace es devolver la lista de preguntas correspondiente a la categoría que se pide, siendo estas listas las ya mencionadas al principio en *preguntasDinamico 1, 2 y 3*.
- *guardarPreguntaAcertada()*: este método es llamado cuando se acierta una pregunta, y sirve para eliminar de las listas correspondientes la pregunta acertada y actualizar el archivo de guardado de las preguntas. Esta categoría que se le pasa por entrada, si es 1, elimina de la primera lista solamente, si es 2, entonces de las listas 1 y 2, y si es 3, de todas.

La forma de funcionar de esta clase es la siguiente. Al empezar la ejecución, en un primer lugar se comprueba el nivel y se llama a *guardarPreguntas()* para asegurarse de que existe un archivo de guardado, y actuar en consecuencia, y a *generarListaPreguntasDinamica()*, para tener lista la base de datos y filtrada para cuando se la necesite en el gestor de preguntas. Cuando esta es consultada, lo único que hace es devolver la correspondiente, y cuando una pregunta es acertada, modifica las listas correspondientes para que estas reflejen las preguntas disponibles.

*VentanaPreguntas* es, en pocas palabras, el gestor de preguntas que muestra por pantalla las preguntas correspondientes y hace las llamadas necesarias cuando estas son respondidas. También controla la animación de las preguntas y los botones que incluyen las respuestas de cada pregunta.

Esta clase contiene las siguientes funciones y métodos:

- *interactuar()*: es la encargada de gestionar si tiene que empezar o terminar la pregunta, así como continuar si el número de preguntas restantes no es cero (esto influye en *TriggerPregunta*), y también de gestionar la cámara para este caso y la animación de las preguntas (si deben aparecer en pantalla o no, al igual que con los diálogos). Su principal método de control es con una variable del tipo *string estado*, que comprueba si está disponible o no (esto se utiliza para evitar interacciones innecesarias con esta función), en caso afirmativo llama al método *preguntas()*, y en caso negativo llama a *terminarPregunta()* y restablece la disponibilidad. Se le pasan por entrada el número de preguntas restantes y la categoría de estas. También activa un sonido cada vez que se interactúa con éxito, procedente del componente *AudioSource* que tiene.
- *preguntas()*: a este método se le pasa por entrada la categoría de la pregunta, entonces consulta la lista de preguntas correspondiente de la base de datos de preguntas, y en base a esta lista, elige una pregunta al azar, para pasársela al método *mostrarPregunta()*. Justo antes de esto, asigna los valores correspondientes a unas variables globales que se corresponden con los distintos elementos de un *CuadroPreguntas* y le pasa a los botones de la interfaz los textos de las respectivas posibles respuestas.
- *mostrarPregunta()*: lo único que hace es asignar al texto que se muestra por pantalla en la interfaz la pregunta de ese instante.
- *pulsarBoton()*: a este método solamente lo llaman los botones de la interfaz una vez son pulsados, y estos le pasan por entrada un valor correspondiente con la pregunta asignada a dicho botón. Entonces se comprueba si el número que tiene asignado el botón pulsado era el mismo que el asociado a la respuesta correcta, y se llama a *acierto()* o *fallo()* en cada caso.
- *acierto()*: este método inhabilita los botones nada más ser llamado, muestra el texto de acierto por pantalla en lugar de la pregunta, resta en 1 el número de preguntas restantes del panel y llama al método *guardarPreguntaAcertada()* de la base de datos con la pregunta acertada y la categoría de esta. Entonces, muestra por pantalla un texto

que indica cómo continuar con el panel y devuelve la disponibilidad a la función *interactuar()*. Aquí también se hace una llamada a *preguntaContestada()*, de la clase *Puntuacion*, con el valor a *true*. También tiene un sonido asociado al acierto, procedente de un elemento hijo de *VentanaPregunta*, ya que en un mismo objeto no puede haber varios *AudioSource* sin que de problemas.

- *fallo()*: funciona de forma similar a *acierto()*, pero muestra otro texto y en lugar de devolver la disponibilidad, pasa al estado *"Terminar"*. En esta ocasión, se llama a *preguntaContestada()*, pero con el valor a *false*. Y, al igual que con el acierto, se activa un sonido, pero esta vez asociado al fallo.
- *terminarPregunta()*: se llama desde *interactuar()* una vez se detecta con *estado* que, o bien no quedan más preguntas por responder, o se ha fallado alguna. En este caso, termina la animación de las preguntas y devuelve la cámara y el control al jugador.

La forma en la que funciona esta clase es que solamente se entra por dos vías: o se la llama a *interactuar()* desde *TriggerPregunta*, o se llama a *pulsarBoton()* cuando se hace clic en alguno.


Figura 4.16: Captura de una pregunta en funcionamiento

La clase *TriggerPregunta* es la clase con la que el jugador puede interactuar y se comunica con *VentanaPreguntas*, de una forma similar a *TriggerDialogos* con *VentanaDialogos*. Esta clase funciona solo dentro del *Update*. Tiene una serie de

variables de control, para actuar en función de si le quedan preguntas restantes (variable que devuelve actualizada o no la llamada a *interactuar()*), o si se está interactuando con *VentanaPreguntas* (como hemos visto antes, por mucho que se interactúe, si esta no está disponible, no hará nada más) por medio de su *Animator*. Su funcionamiento consiste en, si el jugador está cerca del panel de pregunta y pulsa el botón de interacción (tecla E), este se activa y llama a *VentanaPreguntas*, que hará su trabajo. Si esta, al devolver el número de preguntas restantes es igual a cero, querrá decir que el panel ha sido completado, así que hará una llamada a la función *comprobarPaneles()*, de la clase *Puerta,* correspondiente a su puerta asignada. Al mismo tiempo, se activará una pequeña animación estética en el panel mientras se esté usando este.

El elemento *PanelPregunta* que acabamos de mencionar es el que hace uso de la clase *TriggerPregunta*, y representa un panel que muestra las preguntas al jugador y lo pone a prueba. Contiene un *Trigger*, que es el que detecta si el jugador está cerca.

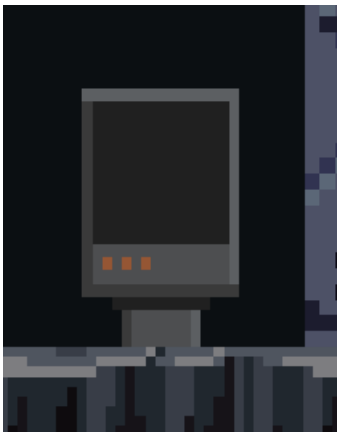

Figura 4.17: Sprite del panel de preguntas

Por último, los principales niveles del tipo *Planeta* (es decir, menos el último del juego) cuentan con un sencillo sistema de *Puntuación* que se apoya en los paneles de preguntas. Este sistema calcula en tiempo real una media entre las preguntas acertadas y las contestadas, para luego sacar el porcentaje. Este sistema está para dar otro aliciente al jugador para que juegue bien (es decir, para que no intente contestar al tuntún), así como conceder o evitar el progreso y que este continúe en función de esta puntuación (el jugador tiene que intentar volver a la nave con al menos un 50% de preguntas acertadas). Su principales funciones son *hacerMedia()*, que hace justo lo descrito, y *preguntaContestada()*, que es llamado desde *VentanaPreguntas* cada vez que se responde una pregunta.

### 4.3.3 Puertas

Las puertas son, dentro de la estructura del juego, uno de los principales obstáculos que impiden que el jugador avance físicamente, y están directamente relacionados con los paneles de preguntas y, en algunos casos, con las cajas.

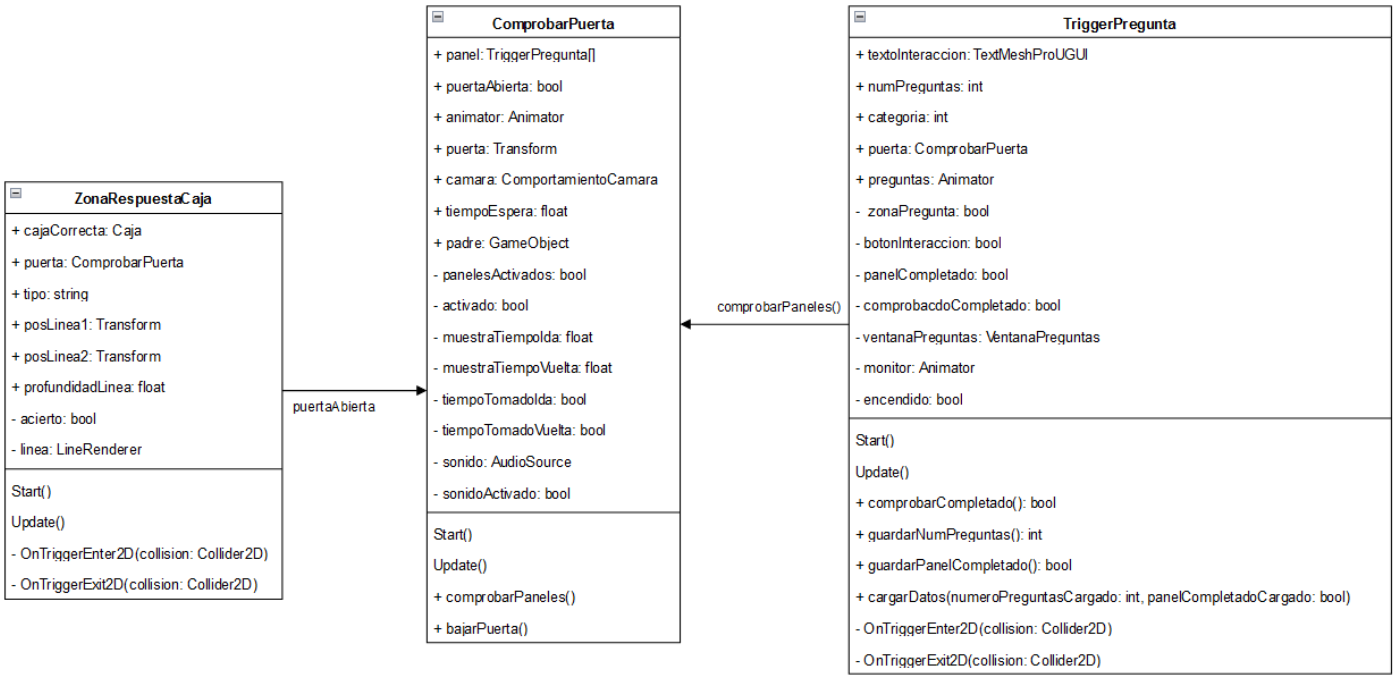


Figura 4.18: Diagrama de clases de las puertas

El principal método del que disponen las puertas en la clase *ComprobarPuerta* es *comprobarPaneles()*, que es llamada por cada *TriggerPregunta* cada vez que un panel se completa. Cada puerta tiene asignado uno o varios paneles de pregunta (se meten en un *array* público *panel* del tipo *TriggerPregunta*). El método comprueba panel por panel si están completados todos los que tiene asignados, y en caso de ser así, la variable *booleana puertaAbierta*, que es la encargada de activar el proceso de apertura de la puerta, pasa a ser true permanentemente. En el caso de las cajas, al activarse solo se pone a *true* el valor de *puertaAbierta*.

Entonces, en *Update()*, empieza el proceso. Se toma una muestra del tiempo (con *Time.time*) y se cambia la cámara para que apunte a dicha puerta (haciendo una llamada a *setPosObjetivo()* y a *cambiarMoverse()*). Una vez pasados unos segundos, se ejecuta la animación que baja la puerta, y se vuelve a tomar una muestra del tiempo para esperar a devolver la cámara. En el caso de

*bajarPuerta()*, es el mismo proceso pero más simplificado, para no coger la cámara al hacer la animación.

Figura 4.19: Sprite de la puerta

### 4.3.4 Cajas

Las cajas son parte de unas pruebas en las que al jugador se le presenta una pregunta, representando las cajas las posibles respuestas con distintos colores o números. Entonces, el jugador deberá coger la caja con la respuesta correcta y colocarla sobre una plataforma próxima. Esta prueba, así como otras que también aparecen más adelante en este capítulo, surgieron como medida para evitar la repetición y monotonía asociadas a tener que estar respondiendo preguntas constantemente en los paneles.

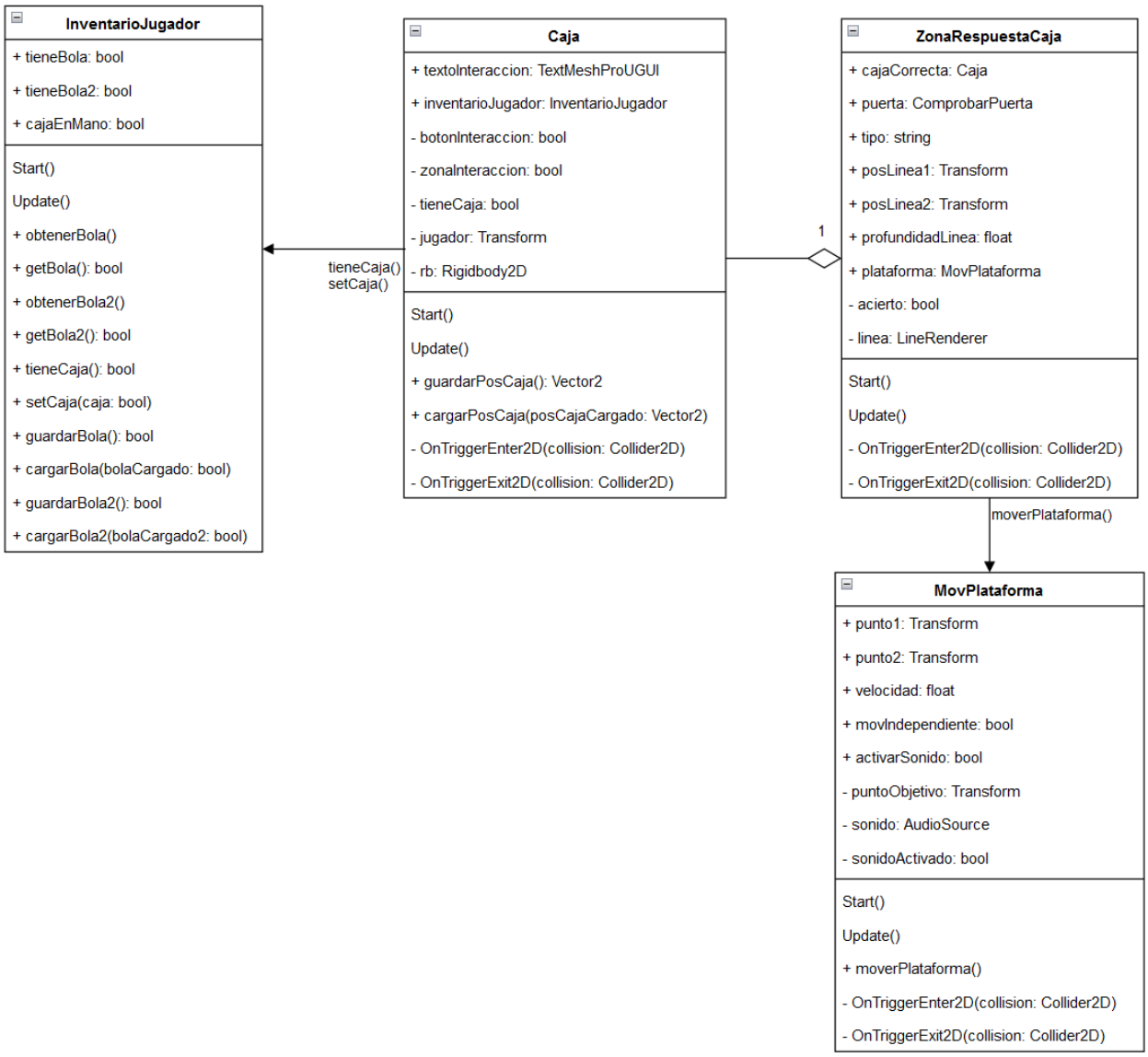


Figura 4.20: Diagrama de clases de las cajas

El funcionamiento de las cajas recae en la clase pública *Caja*. Si el jugador está muy cerca de la caja, en primer lugar, se muestra el *TextoInteracción* y, si pulsa el botón E, entonces la caja pasa a ser hija del elemento *Jugador* (para que

así esta se mueva con él), se le desactivan las físicas para que la gravedad no le afecte (poniendo el valor de *simulated*, de su *Rigidbody2D,* a *false*), y se muestra un texto para indicar cómo soltarla (de la misma forma que al cogerla). Cuando el jugador decide soltar la caja, esta deja de ser hija del *Jugador* para no ser hija de nadie, y se le reactivan las físicas, volviendo a sus condiciones originales.

Como se mencionó en el apartado del inventario del jugador, se hace una comprobación del inventario del jugador antes de coger cada caja para ver si este ya está sosteniendo alguna.

El elemento *ZonaRespuestaCaja*, como su nombre indica, es una zona donde el jugador debe depositar la caja con la respuesta correcta. El funcionamiento de este elemento se encuentra dentro de la clase pública *ZonaRespuestaCaja*. Esta clase tiene asociada una de las cajas correspondientes a la pregunta, siendo esta la única correcta. Su funcionamiento es muy sencillo, si la caja que se coloca encima es la correcta, activa el mecanismo correspondiente, y si se quita, lo detiene. En el caso de que se coloque encima una caja que no sea la que tiene asociada como correcta, no hace nada.

Estos mecanismos que se activan pueden ser una plataforma móvil o una puerta, haciendo una llamada al método *movIndependiente()* en *MovPlataforma,* o bien *puertaAbierta* en *Puerta* pasa a ser *true*. En cualquier caso, se activa también un *LineRenderer* [21], que pinta una línea desde la zona de respuesta hasta el mecanismo correspondiente, para darle una pista visual al jugador de qué es lo que ha activado la zona.

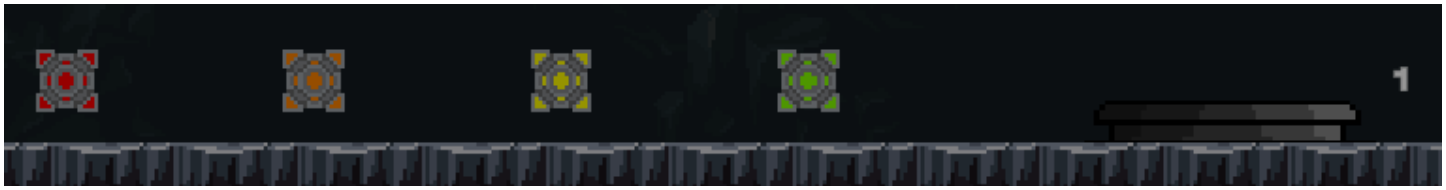

Figura 4.21: Sprites de las cajas del primer planeta y de su zona de respuesta

### 4.3.5 Plataformas móviles

Las plataformas móviles son un elemento directamente relacionado con las zonas de respuestas de las cajas, que permiten al jugador acceder a zonas que de otra forma no podría alcanzar.

Su funcionamiento, mediante la clase *MovPlataforma*, es muy sencillo. La plataforma tiene asociada dos puntos en el espacio, a los que se irá dirigiendo de

forma alterna. Luego tiene una variable *booleana* que, al activarse, hará que se llame continuamente al método *moverPlataforma()*. Este método lo único que hace es acercar la plataforma al punto que le corresponde vía *Vector2.MoveTowards* [22], que mueve el objeto que se le pasa como primer valor a la posición que se le pase como segundo. Si la plataforma detecta en *FixedUpdate()* que ya ha llegado a la posición de destino, la posición objetivo cambia al otro punto que tiene asociado.

Para evitar comportamientos inesperados con las físicas por el desplazamiento y el jugador, contiene una zona con un *Trigger* que detecta cuándo el jugador está sobre la plataforma, y entonces lo hace hijo suyo. Cuando el jugador salta o se separa lo suficiente como para salir del *Trigger*, deja de serlo.

Las plataformas móviles también cuentan con un componente de audio, que emite siempre que se mueve un sonido en 3D para que no se solape con los demás y sea progresivo conforme el jugador se acerca a estas.

Figura 4.22: Objeto PlataformaMovil y Triggers

### 4.3.6 Teletransportadores

Los teletransportadores o portales son la tercera clase de prueba que se encuentra el jugador a lo largo de su aventura. Estos, al igual que las cajas, exponen una pregunta y cada uno de ellos representa una respuesta. Se disponen varias preguntas seguidas, y si el jugador acierta, pasará a la siguiente, pero si falla, retrocederá una. La idea es que consiga acertar variar preguntas seguidas.

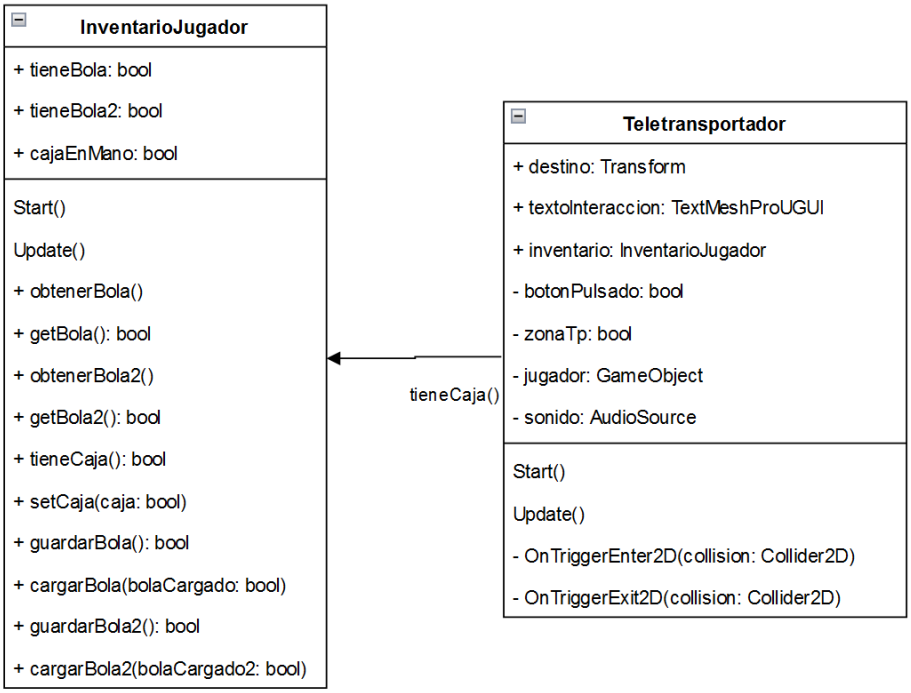


Figura 4.23: Diagrama de clases de los teletransportadores

Su funcionamiento recae en la clase *Teletransportador*. Esta tiene como variable pública un único destino objetivo, que será a donde el jugador será desplazado. Su funcionamiento se basa en que, si el jugador está lo suficientemente cerca de este y entra en el *Trigger*, por pantalla se mostrará el texto de interacción y, si es pulsado, se le llevará a la posición objetivo. También se incluye una comprobación del inventario del jugador, para ver si este está sosteniendo una caja en el momento de cruzar el portal. Esto se hace para intentar evitar que el jugador se lleve cajas a sitios que no están pensados para ello.

Cada pregunta de los portales tiene cuatro posibles respuestas, y solamente una de ellas es la correcta. Para ayudar con la lectura, al pasar cerca de la pregunta la cámara se alejará un poco y apuntará a esta, hasta que el jugador se aleje lo suficiente como para volver a su funcionamiento normal.

Consta de una animación en bucle, que siempre estará activa y nunca cambiará de estado. Al final del primer nivel, se abren unos portales que llevan al principio de este. Estos portales sí tienen animaciones algo más complejas, en el sentido de que, se tiene que subir a una posición visible, y entonces pasar a la animación en la que se abre, y por último mantenerse en *Idle*.

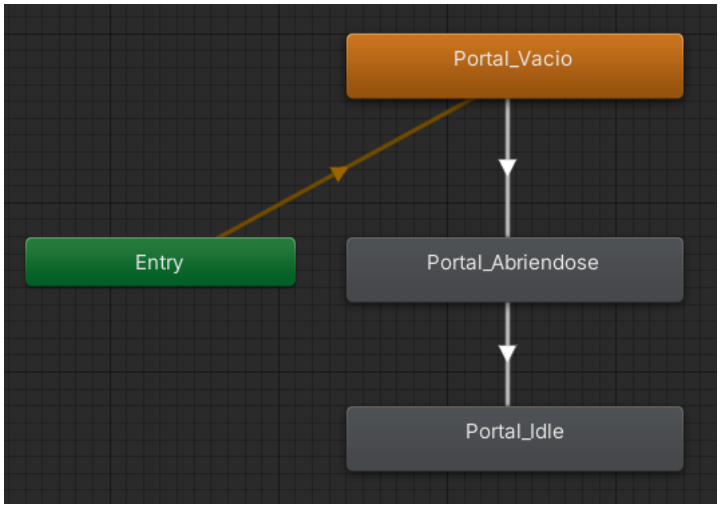

Figura 4.24: Animación del último portal del primer nivel

Los portales también tienen un componente de audio, y se activa un sonido cuando el jugador interactúa con estos.


Figura 4.25: Sprite del portal

### 4.3.7 Cañón

El cañón es la penúltima gran prueba que se encuentra el jugador, y en ella deberá disparar una bola desde dicho cañón, hasta una cesta en movimiento. Esto sucede en tres fases, que cambian cada vez que se encesta y hacen que la gravedad afecte de distintas formas al proyectil disparado. En esta parte se hablará del cañón, de la canasta y de la bola.

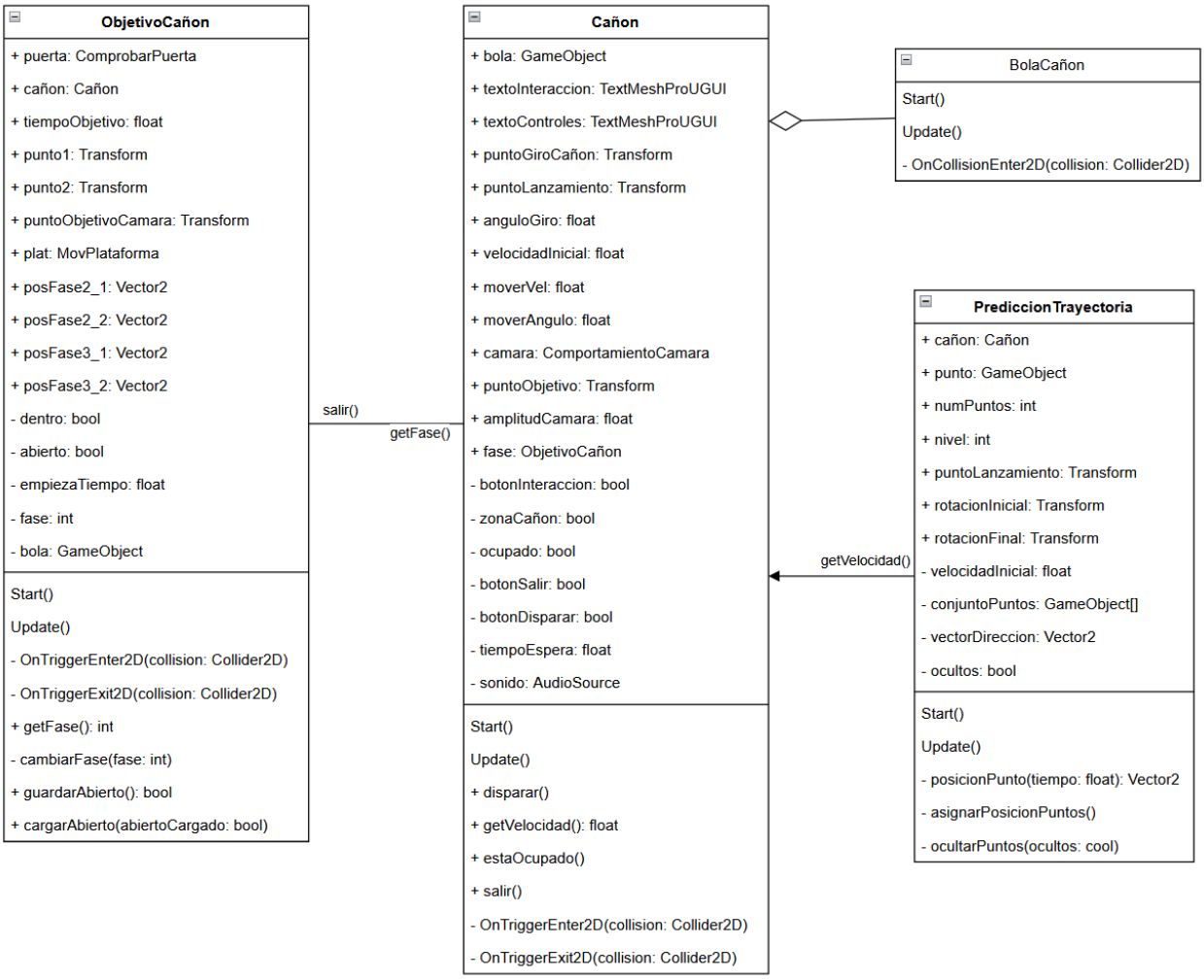


Figura 4.26: Diagrama de clases del cañón

Está formado por una cabina, un cañón (con su correspondiente punto de giro) y un punto de salida del proyectil, que mantiene el ángulo del cañón. Asimismo, tiene una zona formada por un *Trigger* en la que el jugador podrá tomar el control del cañón si interactúa con este y un *AudioSource* que se activará cada vez que se dispare.

Luego está la bola o proyectil, que es una esfera básica con su *CircleCollider2D* y su *Rigidbody2D*, salvo por el hecho de que tiene un material creado específicamente para que rebote más [23] (su valor de *Bounciness* se pone a 0'5). Este material se llama *ReboteBola*. La bola también está programada en la clase *BolaCañón* para que, al tocar cualquier cosa que no sea la canasta por primera vez, desaparezca a los dos segundos, y esto se hace con una llamada al método *Destroy()*, que viene dado por *Unity*.

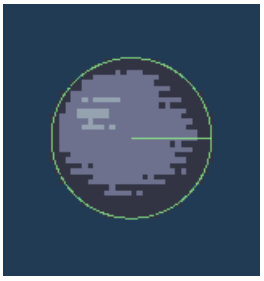

Figura 4.27: Sprite de la bola y su RigidBody2D

Por último, la canasta (o como se le ha llamado su clase, *ObjetivoCañon*), está formado por una estructura que asemeja a una cesta, y dentro contiene una zona con un *Trigger* que solo detecta si entran *Bolas*. Su funcionamiento principal consiste en que, si una bola entra en la *ZonaObjetivo* y está un tiempo que se le pasa por una variable pública, entonces avanzará de fase. Si es la última fase y se avanza, entonces se completa la prueba y se abre la puerta correspondiente poniendo a *true* la variable *puertaAbierta*, lo que activa también su animación. Esta canasta se mueve entre dos puntos de la misma forma que la vista en las plataformas móviles, aunque estos puntos cambian de lugar con cada fase.

Figura 4.28: Objeto ObjetivoCañon

Ahora pasamos a explicar el funcionamiento principal del cañón. En primer lugar, no estará operativo hasta que el jugador esté lo suficientemente próximo y, al igual que con otras pruebas, interactúe con esta. Entonces, evita que el jugador se siga moviendo y empieza la prueba, y la cámara toma una posición predeterminada en la que se puede ver la totalidad de la prueba. También aparecen en la esquina superior izquierda de la pantalla, en la interfaz, los controles que explican al jugador en qué fase está y cuáles son los controles. Este texto es un elemento de la interfaz específico del cañón y del laberinto. En cualquier momento podrá dejar de utilizarlo si pulsa el botón Q, y entonces se le devolverá al jugador el control y la cámara, y el texto desaparecerá.

Dentro de *Update()*, si se pulsan las flechas izquierda y derecha, se podrá aumentar o disminuir la potencia a la que saldrá el proyectil (la variable *velocidadInicial*), y si se pulsan las de arriba y abajo, se rotará el cañón cambiándole el ángulo local al cañón desde su punto de giro (estas medidas no podrán pasar un umbral por ambos extremos, para evitar que se dispare muy flojo o muy fuerte, o bien que se dispare para atrás). Si se pulsa la barra espaciadora, se llamará a *disparar()* entonces el cañón creará un objeto *Bola* (con *Instantiate()*) en el punto de salida del proyectil (que recordemos tiene el ángulo del cañón), le aplicará la fuerza determinada con los controles y esta saldrá despedida en esa dirección, con el componente *velocity* determinado por *velocidadInicial*. El cañón no podrá disparar más de una bola cada segundo y medio. Todo esto se recoge en la clase *Cañon*.

Un añadido muy importante es la clase *PrediccionTrayectoria*, que ofrece al jugador la posibilidad de ver una serie de puntos que le indican la trayectoria parabólica que se predice que va a seguir el proyectil lanzado [24]. Esta clase hace aparecer una serie de puntos rojos, cuyas posiciones son calculadas en primer lugar siguiendo la siguiente fórmula**:**

$$p + (dir * velocidadInicial * t) + (g * t^2)/2$$

donde $p$ es un *Vector2* con la posición del punto de lanzamiento, *dir* es un *Vector2* del vector dirección con el ángulo de lanzamiento apuntando hacia la derecha (*rotación * Vector2.Right*), $t$ es el tiempo y $g$ la aceleración de la gravedad dada por *Physics2D.Gravity*.

Estas posiciones son luego guardadas en un *array* y son dispuestas en función del tiempo, es decir, la posición que tendría la bola pasada dicho tiempo, dando la impresión de que la bola sigue una trayectoria determinada. Es decir, primero se llama a *asignarPosicionPuntos()*, que a su vez llama varias veces a *posicionPuntos()* pasándole el tiempo en aumento, y este devuelve la posición predicha. Estas posiciones se van actualizando sobre la marcha conforme se aumente o disminuya la potencia, y conforme el ángulo de disparo cambie. En el segundo nivel se cambia el valor de $g$ a negativo para simular gravedad invertida.

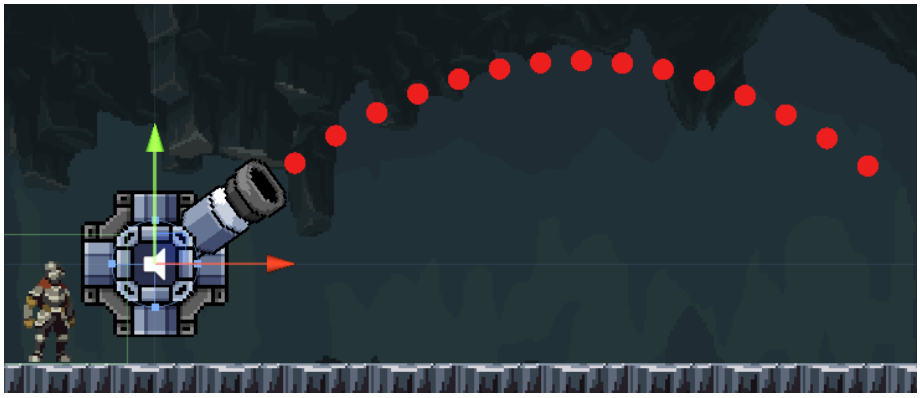

Figura 4.29: Cañón en funcionamiento con la predicción

Por último, las fases consisten en una zona delimitada en la que la gravedad afectará a los proyectiles en mayor o menor medida. En la primera fase la gravedad será normal, en la segunda será menor (la bola irá más arriba de lo predicho) y en la tercera será mayor (la bola irá por debajo de lo predicho). Todo esto se recoge en la clase *ZonaAlteracionBola*, y si la bola está dentro de la zona, esta le aplicará una fuerza determinada por la fase en la que se encuentre. En el segundo nivel, al igual que con la predicción de la bola, se simula una gravedad invertida, y para ello al objeto *Bola* se le cambia la escala de gravedad (cómo le afecta la gravedad en proporción a la aceleración) a -1.

### 4.3.8 Laberinto

Esta es la última gran prueba a la que se enfrenta el jugador para poder obtener el objeto esencial, y dicho objeto está dentro del laberinto. Esta prueba consistirá en sacar el objeto (que se asemeja al *Bola* visto antes) inclinando y rotando el laberinto hasta conseguir sacarlo.

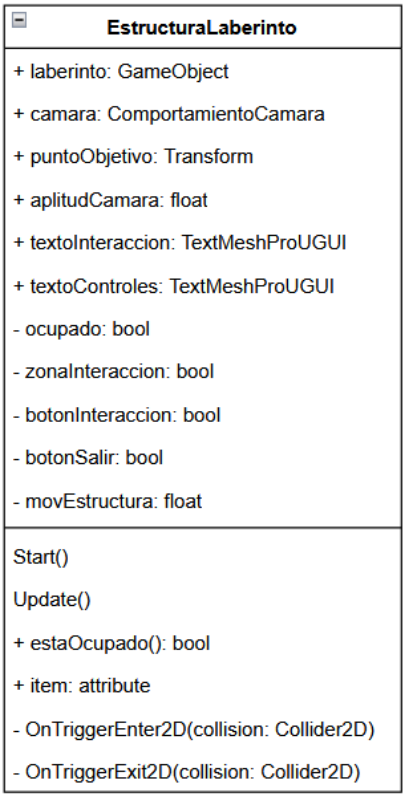


Figura 4.30: Diagrama de clase del laberinto

El laberinto consta de una estructura formada por muchos rectángulos, y dispuestos uno por uno de forma que hacen de paredes para evitar que la bola se salga con sus *BoxColliders2D,* asemejándose a un laberinto, y un punto de rotación justo en el centro, que será en torno al cual el jugador pueda moverlo una vez lo esté utilizando. Una peculiaridad de este elemento es tener que haber incluido la bola dentro del objeto *Laberinto*, puesto que, de no ser así, al igual que lo mencionado con las plataformas móviles, las físicas resultantes de girar el laberinto y las de colisión de la bola hacían que esta se comportara de forma errática. Al cambiar esto, este fallo se solucionó. Esto también provocó el añadido de un *Trigger* en la salida del laberinto que hiciera que la bola dejara de ser hija de este, para que así no siguiera dependiendo del giro del laberinto una vez sacada.

El objeto entero *Laberinto* también incluye una plataforma que permite la interacción del jugador con este, con el mismo aspecto que una *ZonaRespuestaCaja*, pero con un comportamiento similar al del *Cañón*.

Todo esto se recoge en la clase *EstructuraLaberinto*. Si el jugador está sobre la zona de activación e interactúa, entonces se le retira el control, la cámara apunta a la estructura y se amplía para que se pueda ver todo, y se muestran las instrucciones de la misma manera que con el *Cañón*. El jugador entonces puede pulsar las flechas izquierda y derecha para que se aplique una rotación a la estructura principal (mediante *transform.Rotate*) en sentido horario o anti horario. En cualquier momento el jugador puede pulsar el botón Q para salir y que se le devuelva el control y la cámara.

Figura 4.31: Estructura del laberinto, con el objeto esencial en una esquina

### 4.3.9 Objeto esencial del nivel

Este objeto es el objetivo del jugador en los niveles del tipo *Planeta*, y este tendrá que interactuar cuando esté lo suficientemente cerca para añadirlo a su inventario y cumplir una de las dos condiciones para poder salir del nivel. Al interactuarse, la bola habrá cumplido su función y desaparecerá.

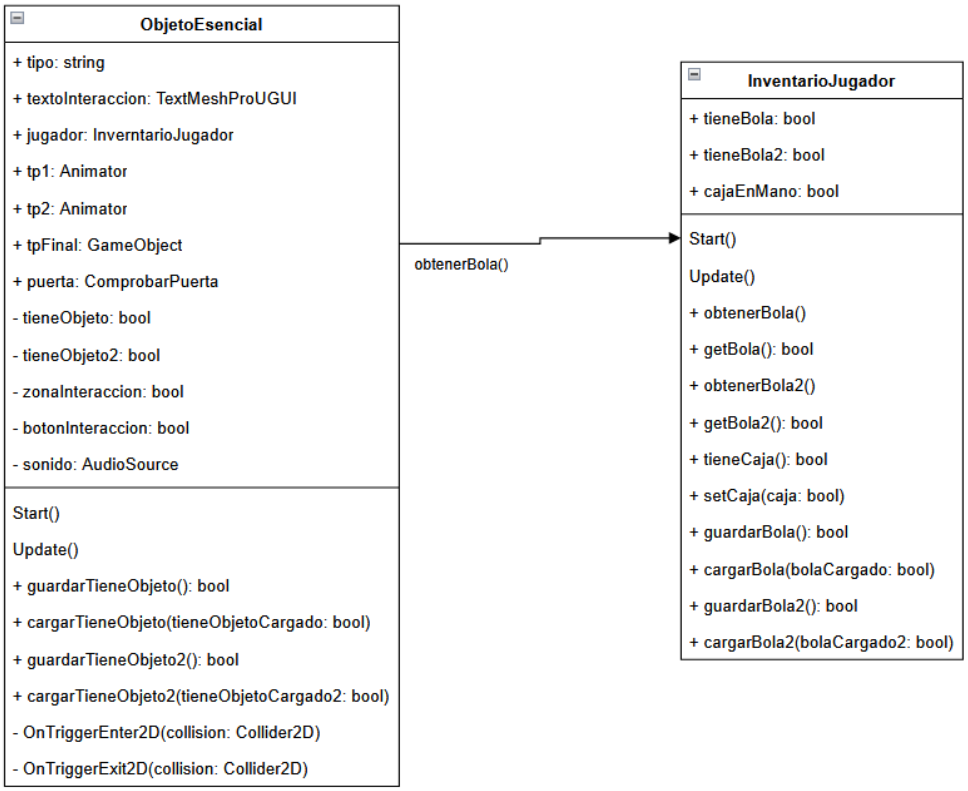


Figura 4.32: Diagrama de clases del objeto esencial

Este elemento consiste en una esfera con dos *CircleColliders2D*, siendo el segundo un *Trigger* para detectar si el jugador está cerca. Esto se encuentra en la clase *ObjetoEsencial*. Cuando el jugador interactúa con la bola, se llama al método *obtenerBola()* de *InventarioJugador* y, en el primer planeta, se activan las animaciones de dos portales que llevan al principio del nivel. Inmediatamente después, el objeto se deshabilita para que el jugador no pueda interactuar más, ya que está añadido.

También tiene un componente de audio que se activa al ser recogido por el jugador.

Figura 4.33: Objeto esencial

### 4.3.10 Zona de salida del nivel

Una vez que el jugador ha encontrado lo que buscaba, deberá abandonar el planeta para seguir con su aventura. Y esto quiere decir que deberá dirigirse a su nave espacial e interactuar con ella para salir del nivel. Pero esto no es posible mientras el jugador no tenga en su inventario dicho objeto ni tenga más de una puntuación concreta, y se le avisará por pantalla de ello.

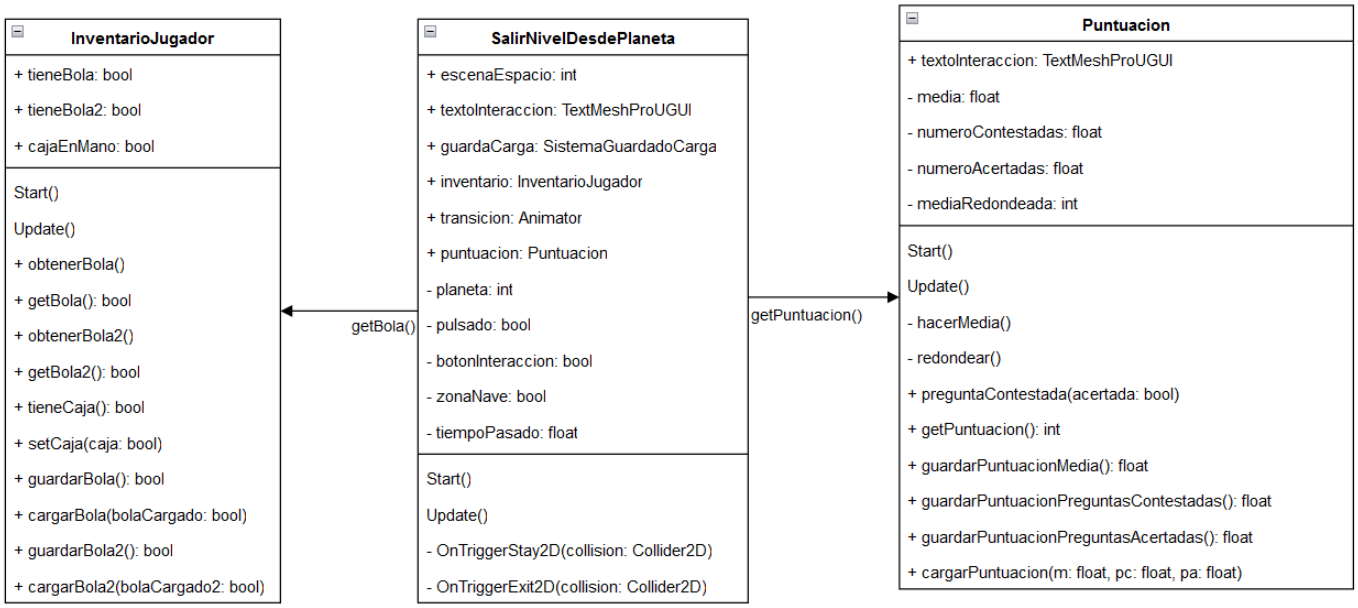


Figura 4.34: Diagrama de clases de la zona de salida del nivel

Al entrar dentro de la zona del *Trigger*, y en caso de que el jugador ya tenga el objeto y puntuación necesarios, se le permitirá interactuar con la nave y salir del nivel. Esto se puede ver en la clase *SalirNivelDesdePlaneta*.

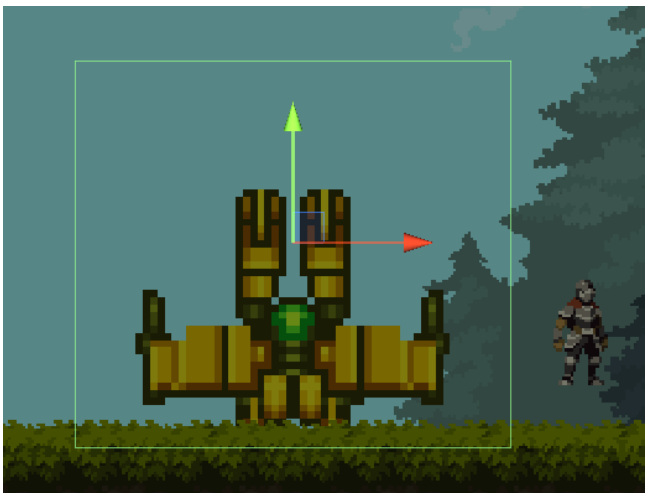

Figura 4.35: Objeto ZonaNave

### 4.3.11 Planetas

Estos planetas se encuentran en el nivel del espacio, y representan los distintos niveles a los que puede acceder el jugador con su nave. Para simplificar el proceso aún más, el jugador solo tiene que acercarse y tocar dicho planeta para que empiece el proceso de cambio de nivel. Todo esto se encuentra en la clase *CampoGravedad* y *EntrarNivelDesdePlaneta*.

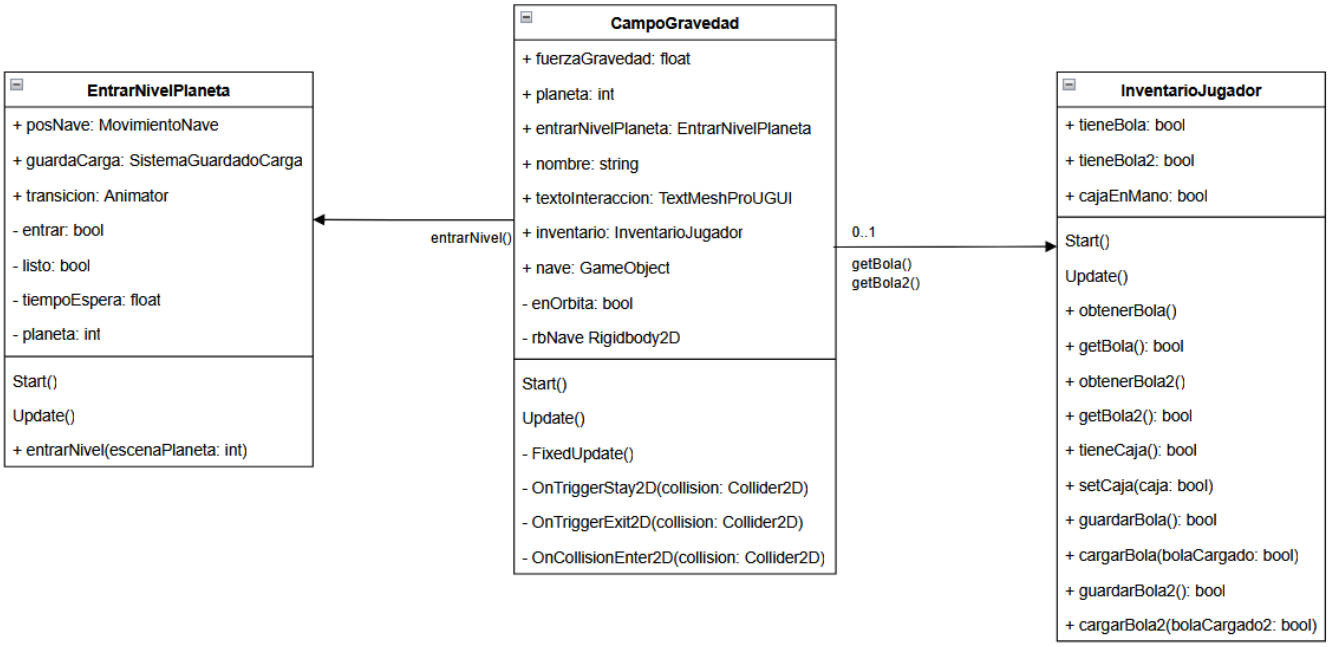


Figura 4.36: Diagrama de clases de los planetas

Estos tienen un campo de gravedad que hace que la nave se vaya acercando (o alejando en el caso del Sol) a este [25]. Esto se hace con un *Trigger* más grande que el planeta, en el que, si entra la nave, entonces se toma un vector de dirección de la nave y el centro del planeta, y entonces a la nave se le aplica una fuerza determinada por variable pública con *AddForce* a su *RigidBody2D* en dicho sentido y teniendo en cuenta la masa de dicha nave. Al entrar dentro de este campo de gravedad, se muestra por pantalla mediante *TextoInteraccion* un mensaje indicando el nombre del plantea al que van a entrar.

Una vez la nave entra en contacto con el RigidBody2D del planeta, se hace una llamada a *entrarNivel()* de *EntrarNivelPlaneta*, que hace el proceso pertinente para cargar el nivel que corresponde. Este nivel se le pasa por una variable pública.

Los planetas también cuentan con un componente de animación sencillo, en donde el *sprite* asemeja la rotación del planeta.

Por último, en el caso del nivel final (la estación espacial misteriosa), no tiene campo de gravedad propio, y antes de poder permitir el acceso a este, se comprueba el inventario del jugador con *getBola()* (1 y 2) para ver si tiene los dos objetos esenciales necesarios. En caso negativo, al entrar en contacto no ocurre nada.

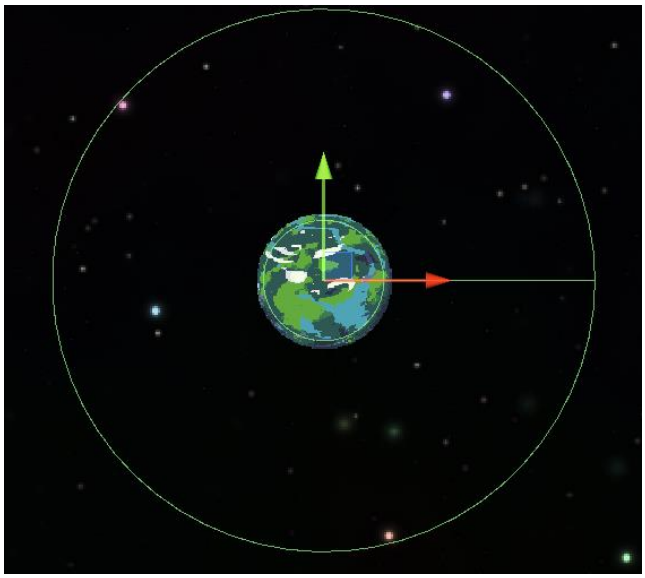

Figura 4.37: Objeto Planeta

## 4.4 Gráficos y escenario

Los gráficos asociados a este juego se pueden separar en dos categorías: en escenarios y en elementos con o sin animación. Sea cual sea, en todos los casos se tratan de *sprites* gratuitos adquiridos desde la página *Itch.io*.

En cuanto a los escenarios, se sigue un proceso para implementarlos [26]. En primer lugar, se guardan todas las imágenes en una carpeta. A continuación, se accede al *Sprite Editor*, en la parte izquierda del *Inspector* de *Unity*, y este nos lleva a una ventana nueva. Aquí se recorta la imagen haciendo clic en *Slice* y se escoge la opción para hacerlo por tamaño de celda en píxeles (en el caso de algunos de los *sprites* de algunos personajes, en lugar de esto se ha hecho mediante la opción *Automatic*). Según la imagen, este número podrá variar, pero para la mayoría de casos suele bastar con 16 x 16 píxeles para tener unas celdas perfectas. Estas celdas representarán un cuadro con una parte de la imagen, que se podrá poner en el escenario.

Una vez tenemos cortada la imagen, se crea un *Tilemap (Create-> 2DObject->Tilemap->Rectangular)*, que a su vez crea automáticamente un *Grid*, que es el espacio dentro del juego que muestra dichos *sprites* estáticos. Una vez se ha hecho y elegido un nombre, en *Tile Palette*, justo al lado del *Inspector*, aparecerá una ventana en la que se puede crear una nueva paleta, que es donde irán las imágenes recortadas. Esta herramienta permite elegir los trozos y "pintarlos" en el escenario del suelo, pudiendo gestionar las capas e incluso crear o añadir otras paletas nuevas.

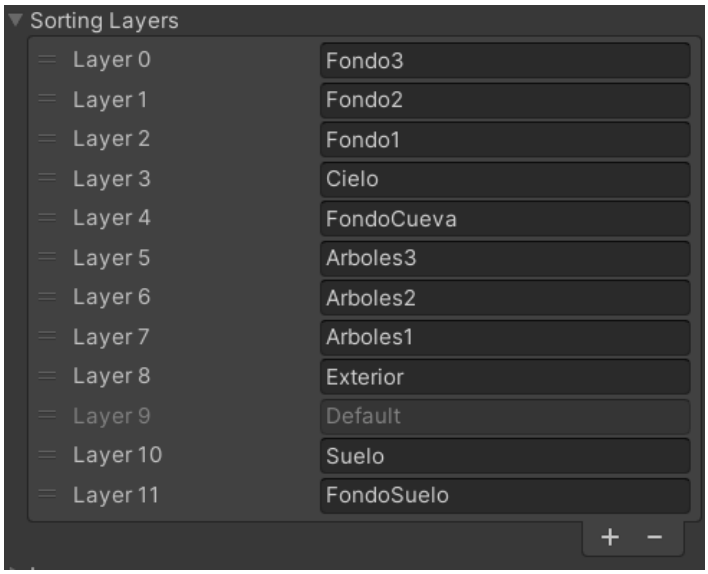


Figura 4.38: Organización de las capas del Grid

En el caso de los *sprites* asignados al suelo y las paredes, estos también contienen componentes de *Rigidbody2D* junto con *TilemapCollider2D* (para que el jugador pueda colisionar con este), y *CompositeCollider2D* (que hace que todo lo pintado tenga un mismo colisionador, en lugar de uno por cada celda ocupada).

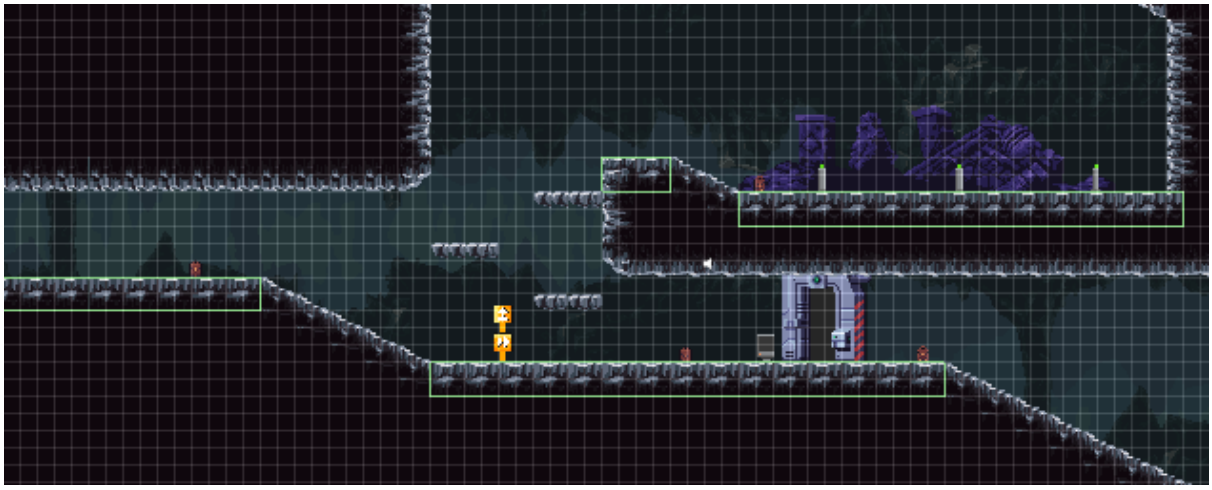

Figura 4.39: Ejemplo de disposición del suelo en el Grid

En cuanto al resto de elementos consisten en acceder al componente de *SpriteRenderer*, y cambiarlo por la imagen deseada.

Los únicos casos en los que se han tenido que añadir a mano los colisionadores en el escenario fueron con las rampas y con las plataformas pasables. El caso de las rampas se dio por el hecho de que el *TilemapCollider* no daba como resultado una rampa totalmente uniforme, sino con las imperfecciones que se asemejan a las de una roca, como en su imagen. En este caso se crearon unas cajas (Box), se les desactivó el *SpriteRenderer* y se colocaron justo donde estaban los *sprites* de las rampas, dando la ilusión de una colisión normal.

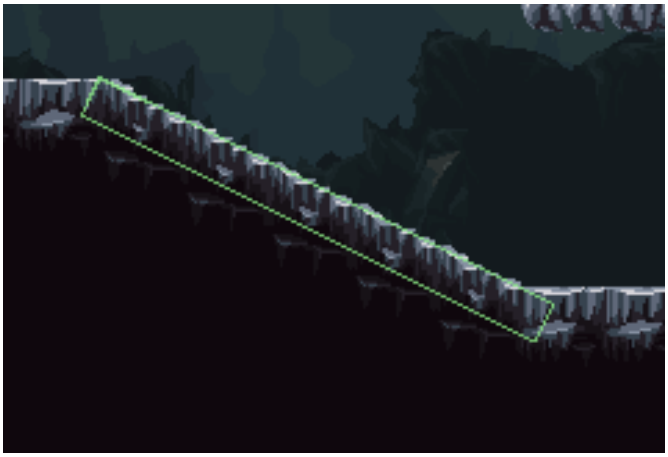

Figura 4.40: Ejemplo de rampa

En el caso de las plataformas pasables, ocurre algo similar, pero como se buscaba que el jugador además pudiera saltar a estas plataformas desde una posición inferior y que se pudiera quedar encima [27], se recurrió a un componente *PlatformEffector2D*, haciendo clic en *"Use One Way"* (usar una vía).

Figura 4.41: Ejemplos de plataformas pasables

Por último, para evitar que el jugador pueda salirse de las zonas designadas, existen una serie de paredes invisibles, que son simples cajas con el *SpriteRenderer* desactivado, colocadas estratégicamente donde el jugador pudiera intentar acceder a algún sitio no previsto.

Cada uno de los escenarios cuenta con su propia música, que es un *AudioSource* en bucle a un volumen que, ni molesta, ni impide escuchar el resto de sonidos.

## 4.5 Gestión de datos

La gestión de los datos es uno de los elementos más importantes que se puede encontrar en un videojuego, y que se encuentra prácticamente en todos los videojuegos modernos, ya que, sin este, el progreso del jugador a lo largo de una sesión muy larga podría perderse, y con ello el interés de este. En este proyecto se ha gestionado mediante un archivo JSON, al igual que con las preguntas, pero con una estructura distinta [28].

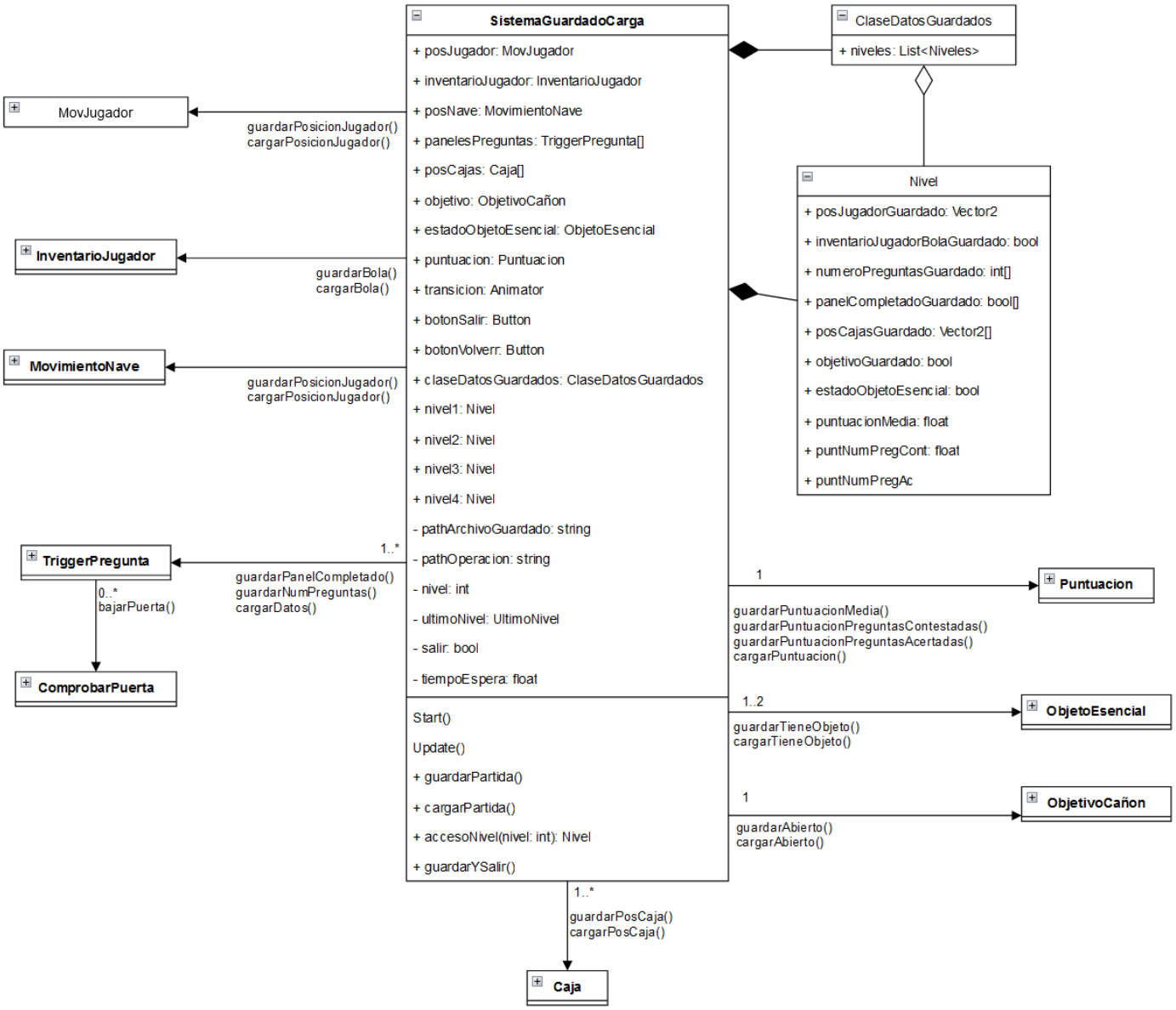


Figura 4.42: Diagrama de clases del sistema de guardado

En primer lugar, esta estructura consiste en un objeto básico *Nivel* [29], que guarda la posición del jugador en un *Vector2*, el inventario del jugador con respecto a los objetos esenciales, los números de preguntas restantes y el estado

de compleción de los paneles mediante *arrays* de *int* y *bool* respectivamente, las posiciones de todas las cajas mediante un *array* de *Vector2*, el estado de compleción del objetivo del cañón en un *bool*, y el estado del objeto esencial del nivel en un *bool*, para saber si tiene que aparecer en el escenario o no. Estas variables, dependiendo del nivel, se usarán en mayor o menor medida, como ahora se explicará más adelante.

Esta clase *Nivel* luego se utiliza en otra llamada *ClaseDatosGuardados*, que consistirá en una lista de *Niveles*, para poder gestionar cada uno individualmente en función de su posición en la lista, siendo el de la primera posición el de tipo *Espacio*, y el resto los de tipo *Planeta* en orden.

También existen una serie de llamadas a funciones y métodos dentro de los distintos elementos que han visto hasta ahora que almacenan o asignan sus respectivos valores según se estén guardando o cargando los datos. Estas funciones y métodos son:

- *guardar/cargarPosicionJugador*(): dentro de *MovJugador* y *MovNave*, que devuelve la posición del jugador en un *Vector2*, o bien asigna los valores que recibe a su variable de posición dentro de la clase.
- *guardar/cargarBola() (1 y 2)*: dentro de *InventarioJugador*, que devuelve o asigna el valor del *bool* asociado al objeto esencial correspondiente.
- *guardarNumPreguntas()*, *guardarPanelCompletado()* y *cargarDatos()*: dentro de *TriggerPregunta*, que devuelven el número de preguntas restantes del panel, si está completo o no, o cargan en sus variables estos mismos datos. En caso de estar completado, abre directamente la puerta llamando a *bajarPuerta()* para que la abra sin utilizar la cámara.
- *guardar/cargarPosCaja():* dentro de *Caja*, que devuelven en un *Vector2* la posición de la caja, o bien asignan a su variable esta posición que se le pasa.
- *guardar/cargarAbierto()*: dentro de *ObjetivoCañon*, que devuelve o asigna el estado del *bool* asignado a la compleción del objetivo del cañón. En caso de estar completado, llama a *bajarPuerta()*.
- *guardar/cargarTieneObjeto() (1 y 2)*: en *ObjetoEsencial*, se devuelve o carga el estado del objeto esencial, es decir, la variable *tieneObjeto*, para que, en caso de haberse cogido, se deshabilite el objeto inmediatamente, ya que el jugador ya lo tiene en su inventario.

- *guardarPuntuacionMedia()*, *guardarPuntuacionPreguntasContestadas()*, *guardarPuntuacionPreguntasAcertadas()* y *cargarPuntuacion()*: dentro de *Puntuación*, que devuelven la media, el número de preguntas contestadas y el número de preguntas acertadas, y carga todo esto en sus variables cuando se llama.

Ahora, el funcionamiento al empezar cada nivel es el siguiente:

Se obtienen el número del nivel (mediante *SceneManager.GetActiveScene().builIndex*, que devuelve el número de la escena en la *build*), el *path* de los archivos de guardado y el de comprobación del último nivel accedido (se guarda el último nivel accedido para cuando se quiera cargar la partida), y se hace una llamada al método *cargarNivel()*.

Este método comprueba si existe un archivo en el *path*, si no lo encuentra, lo crea, y si existe, carga los datos desde el JSON a una *claseDatosGuardados* mediante *FromJson*, asigna los distintos niveles dentro de esa lista a unos niveles de forma individual, y hace las llamadas correspondientes a los métodos correspondientes con los datos que se necesiten para ese nivel (en el caso de los *arrays* se hace una llamada para cada elemento de este). Por ejemplo, para un nivel del tipo *Planeta* (salvo el último) hacen falta todos los datos que se han visto, pero para el caso del nivel del tipo *Espacio*, solamente le hacen falta la posición del jugador y el inventario.

Si hablamos de la forma en la que se guardan los datos, esta solo se activa cuando se sale del nivel o cuando se sale del juego, y su funcionamiento es el siguiente:

En primer lugar, se comprueba el nivel en el que se está, puesto que se supone que los datos de otros niveles en los que se ha estado ya han sido guardados anteriormente. Entonces, se crea una variable de tipo *Nivel* y se hacen las llamadas a funciones correspondientes a los datos que se necesitan. Este nivel se añade a la lista de niveles *claseDatosGuardados* en la posición que le corresponde. Como a una lista no se le pueden añadir y quitar elementos sin alterar el orden deseado, entonces se hacen llamadas a una función *accesoNivel()* de los niveles restantes, que simplemente devuelve un elemento *Nivel* con los datos del nivel que se le pide, ya que estos fueron cargados al principio de la ejecución de la clase. Esta lista se reconstruye añadiendo los niveles en el orden deseado junto con el que se quiere

guardar, y entonces se crea el JSON, se le escribe la lista de niveles, y se guarda en el *path* descrito al principio. También se guarda en otro archivo distinto, llamado *Operación*, que contiene el número del último nivel visitado para que, cuando se carguen los datos, se hagan solamente los de dicho nivel.

## 4.6 Transiciones entre niveles

Este proyecto consta de varias pantallas o escenas, así que es necesario que exista una forma de poder cambiar de una a otra, y de que este proceso vaya acompañado de una transición [30].

Para ello se recurre a la clase *SalirNivelDesdePlaneta*, integrado en el elemento *ZonaNave* (la zona de salida del nivel), a la clase *EntrarNivelPlaneta*, integrado en el elemento *Planeta*, y al método *guardarYSalir()* de la clase *SistemaGuardadoCarga*.

Estas clases y métodos lo que hacen es que, cuando se interactúa con ellas, hace que empiece un proceso por el cual se guardan los datos de la partida y se inicie una animación que consiste en una pantalla en negro difuminándose hasta que está totalmente negra. Durante este proceso, al jugador se le retira el control para evitar que se desplace mientras no pueda ver. Este elemento, llamado *PantallaTransición*, se encuentra en el *Canvas* y consta de una animación con tres estados: uno para desaparecer (que siempre se activa nada más entrar en cada nivel), uno para moverse de sitio (ya que de otra forma estorba para pulsar los distintos botones como los de las preguntas) y uno para moverse a la posición inicial y aparecer (cuando se sale del nivel). Estas animaciones se controlan con una variable *bool salir*, que solo se activa cuando se quiere salir del nivel, y por tanto pasa del segundo al tercer estado.

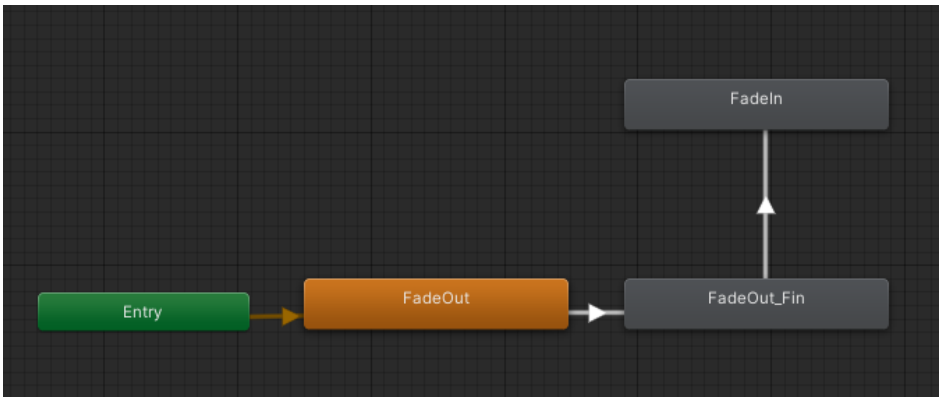

Figura 4.43: Animación de la pantalla de transición

El cambio de nivel como tal se hace mediante el *SceneManager*, que es la herramienta de *Unity* encargada de gestionar las distintas escenas que están creadas y su orden en la *build*. Con este orden nos referimos a una ventana, la de *Build Settings*, que nos permite ver el número que ocupa una escena u otra en el orden de escenas. Esto nos puede servir para poner el orden que necesitemos, o bien para actuar en el código en el nivel con el número que se requiera.

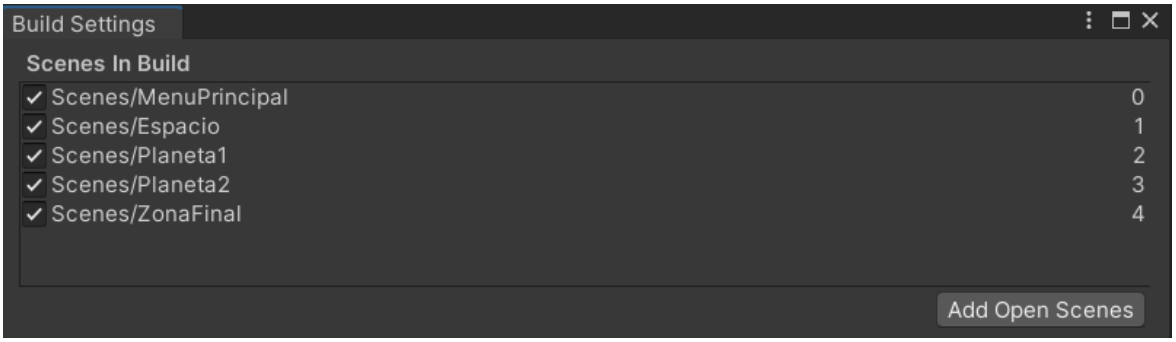


Figura 4.44: Disposición de los niveles en la build

Para cambiar de nivel, se hace una llamada a *SceneManager.LoadScene()*, pasándole por entrada el número de la escena objetivo. Este número también se puede obtener haciendo una llamada a *SceneManager.GetActiveScene().buildIndex*, que devuelve el número de la escena en el momento de la ejecución. En el caso de transitar de un *Planeta* al *Espacio*, siempre se carga la escena 1, pero si es desde el *Espacio* a un *Planeta*, antes de cargar la escena correspondiente, y cuando ya está la pantalla en negro, se mueve la nave espacial a una posición que dependerá del planeta, para que cuando se vuelva al *Espacio*, aparezca cerca del planeta, pero no tanto como para verse arrastrado por su campo de gravedad. Si es para salir al *Menú Principal*, se llama a *guardarYSalir()*, que guarda la partida, activa la transición y, pasados unos segundos, carga la escena 0 en *Update()*.

## 4.7 Menús

Los menús son otro de los muchos elementos esenciales de un videojuego, puesto que sin ellos los jugadores no tienen forma de acceder a diversas opciones, o directamente empezar el juego. En el caso de este proyecto, hay dos tipos de menús: uno principal y otro dentro del propio juego.

Figura 4.45: Captura de pantalla del menú principal

El menú principal es la primera pantalla que aparece cuando se abre el juego, y sus elementos principales son una serie de botones en el *Canvas* con un fondo detrás. Cada uno de estos botones hace una función distinta, y son:

- *Nueva partida*: este botón borra los archivos guardados en caso de que existan (con *File.Delete()* y los *paths*) y carga la escena correspondiente al primer nivel (no hace falta que se cree un archivo de guardado ya que es lo primero que se hace al cargar un nivel si no se detecta ninguno).
- *Cargar partida*: este botón abre el archivo *Operación* y consulta qué nivel es al que tiene que dirigirse. En caso de que no exista, no hace nada.
- *Extras*: este botón deshabilita el resto para mostrar por pantalla un texto que indica el *path* de los archivos de las preguntas, por si el jugador deseara modificarlos. Aquí, pulsando la tecla E, se vuelve al estado original de la pantalla.
- *Salir*: este botón hace una llamada a *Application.Quit()*, que cierra el juego.
- *Créditos*: este botón desactiva el resto para presentar un texto que se desplaza hacia arriba y muestra los créditos correspondientes a todos los archivos utilizados para llevar a cabo el proyecto. Si se pulsa el botón *escape* (que se indica con un texto en la esquina inferior derecha) durante el proceso o se espera a que pasen, entonces vuelve a habilitar todos los botones y vuelve a su estado inicial.

Los métodos para *Nueva Partida, Cargar Partida, Extras* y *Salir* se encuentran en la clase *MenuPrincipal*, los créditos están en la clase *Creditos*.


Figura 4.46: Captura de pantalla del menú del juego

Ahora, dentro del propio juego (es decir, en cualquier pantalla que no sea la del menú principal), el menú funciona de una forma algo diferente. Para empezar, se tiene que acceder a este pulsando la tecla *escape*. Entonces, aparece por pantalla dicho menú, se detiene el tiempo (haciendo que la escala del tiempo sea cero con *Time.timeScale*) y aquí se muestran dos botones:

- *Guardar y salir*: esto lo que hace es hacer una llamada a *guardarYSalir()* de *SistemaGuardadoCarga*, que como recordemos, guarda los datos y activa la pantalla de transición para volver a la escena asociada al menú principal.
- *Volver*: este botón hace que se salga de este menú, volviendo el tiempo a la normalidad y devolviendo el control al jugador.

El comportamiento asociado a este menú se encuentra en el script *MovJugador*, ya que es más fácil tenerlo controlado junto a las comprobaciones de pulsaciones del resto de botones, así como que dicho comportamiento no es tan invasivo o complejo como para tener que haber recurrido a otra clase nueva.

# Capítulo 5

# Conclusiones

Para este Trabajo de Fin de Grado se ha llevado a cabo el desarrollo de un videojuego serio cuyo objetivo ha sido el de acercar o dar a conocer el tema de la historia espacial y la astronomía a personas de todas las edades. Este proyecto se ha dividido en dos fases principales:

- Una primera fase de diseño, en la que se realizaron tanto el *High Concept* como se empezó el *GDD*. Esta fase también incluía la designación de una serie de requisitos que, a lo largo del proyecto, han ido cambiando, siendo añadidos o quitados en función de las necesidades y las dificultades encontradas.
- Una segunda fase de implementación, en la que se utilizó el motor de juegos gratuito *Unity*, así como otras herramientas gratuitas para añadir e implementar el código, hacer las modificaciones necesarias a las imágenes o crear y alterar los diagramas creados en la fase de diseño. En esta fase también se incluye la consulta de material para una mayor facilidad de implementación de las distintas mecánicas expuestas a lo largo de esta memoria.

## 5.1. Mejoras y trabajo futuro

Una de las principales mejoras a plantear, teniendo en cuenta que en el apartado audiovisual se ha recurrido a *assets* gratuitos es, el diseño y desarrollo de los *sprites* y los archivos de audio necesarios para que el producto final tenga un aspecto más similar a la idea original.

Otro de los aspectos en los que se busca mejorar sería el de las animaciones, ya que, con el debido tiempo y documentación, se podría conseguir un resultado más satisfactorio y más refinado. Para este proyecto se ha recurrido a animaciones sencillas, basadas principalmente en *sprites* que ya traen los fotogramas hechos y en desplazar objetos de una posición a otra, normalmente en línea recta. Con este trabajo futuro se buscaría hacer estas animaciones más complejas y eficientes.

En cuanto al trabajo futuro, lo principal sería expandir el producto, añadiendo más niveles, más minijuegos y mecánicas, así como desarrollar aún más

el apartado narrativo, para tener un producto más grande que pudiera intentar conseguir lo que se ha buscado con este proyecto de una forma más eficiente y efectiva. También se usaría la experiencia y soltura que me ha proporcionado este proyecto para expandir mis conocimientos, tanto en desarrollo como en diseño y planificación de videojuegos.

## 5.2 Aprendizaje personal

En este aspecto, este Trabajo de Fin de Grado me ha servido principalmente para asentar los conocimientos aprendidos este año en una asignatura previa, así como aprender aspectos nuevos y muy interesantes. Principalmente he expandido mi conocimiento a hacer varias pantallas o niveles, a trabajar con interfaces, con animaciones, con sonidos y audio y con sistemas de carga y guardado de datos.

Este trabajo también me ha ayudado a empezar a aprender en mayor profundidad acerca del diseño de videojuegos, así como a estructurar y planificar este en las dos fases que se han visto, algo muy importante a la hora de desarrollar un proyecto como este.

## 5.3 Problemas técnicos encontrados y dificultades

Los principales problemas encontrados a lo largo de la etapa de implementación han sido los siguientes:

- *Fallo gráfico en los caracteres*: Este fallo hace que las letras con tilde, la letra "ñ" o abrir interrogación salgan sin la fuente de texto elegida. Esto se debe a que estos caracteres no se incluyen en la fuente de texto, seguramente debido a que el creador no es nativo en el idioma castellano.
- *Sprites borrosos (solucionado)*: Este fallo provocaba que los sprites se vieran borrosos, fuera cual fuera su tamaño. Para esto se siguió un método con cada una de las imágenes utilizadas en este proyecto. El *Mesh Type* se establecía a *Full Rect*, el *Filter Mode* a *Point(no filter)* y la compresión a *None*.
- *Fallo gráfico en los sprites del escenario (solucionado)*: Este fallo provocaba que entre las distintas celdas del *Grid* existiese un hueco minúsculo pero visible, que lastraba la experiencia al cargarse el aspecto del escenario. Este se ha solucionado haciendo uso de un *Sprite Atlas* [31]. Un *Sprite Atlas* junta todos los *sprites* que tenga y los une en una sola

imagen. Una vez creado, se le añaden los *sprites* asociados al escenario y, al igual que con el problema anterior, el *Filter Mode* se establece a *Point* y la compresión a *None*.

- *Fallo gráfico en la animación de la nave*: Este fallo provoca que al hacer la llamada a *girarDerecha()* en la clase *MovimientoNave*, la animación del propulsor izquierdo se active en el caso de que se llame si se pulsa en el teclado, pero no si se pulsa por el botón en la pantalla. Ambas formas de controlar la nave llaman al mismo método.
- *Caracteres borrosos en la pantalla (solucionado)*: Este fallo provocaba que los caracteres que salían en el *Canvas* aparecieran muy borrosos, hasta el punto en que costaba distinguir lo que se mostraba. Este fallo se solucionó forzando la resolución en el editor a 1920 por 1080 píxeles, para luego en la *Build* indicarle que se iniciara en modo ventana con dicha resolución.
- *Cámara apuntando a puertas abriendo al cargar la partida*: Este fallo provoca que, hecho cierto avance en el juego y cuando se carga la partida, se active la animación en la que se abre la cámara, cuando no tendría que ser así. Se piensa al haber intentado recrear el fallo, que este puede ser un fallo provocado por la lectura del propio motor de variables de control en un orden que no es el establecido. Es decir, la puerta que se tendría que abrir al cargar la partida tendría que ser llamada al método *bajarPuerta()*, en la clase *ComprobarPuerta*, que bajaba la puerta sin reclamar la cámara para ello. Pero se piensa que el motor hace un control de la variable activado erróneo en esas puertas concretas.
- *Pasar por un portal sosteniendo una caja*: Este fallo provoca que el jugador sea capaz de atravesar un portal de pregunta mientras sostiene una caja. Pese a que se hicieron cambios para que se detectara si el jugador sostiene una caja al entrar el en *Trigger* de un portal, si se deja una caja lo suficientemente cerca, el portal detecta que no se está sosteniendo una caja, pero al mismo tiempo, la caja detecta que tiene al jugador al lado, permitiéndole cogerla. Con esto se puede pasar esa guarda y, en ocasiones, pasar un portal con una caja en la mano.

Las dificultades más notables encontradas en el desarrollo de este proyecto han sido las de, pese a tener una experiencia previa con *Unity* al haber cursado una asignatura optativa relacionada con el tema, que esta no fuera suficiente y se necesitara una cantidad de documentación, soltura y experiencia mayores. El

tiempo limitado para realizar este proyecto, así como la escala de algunas de las ideas plantadas durante la fase de diseño supusieron que el proyecto no fuera tan grande o que algunos aspectos estuvieran más pulidos que otros. Con el suficiente tiempo y un mayor conocimiento, como se mencionó en *Mejoras y trabajo futuro*, estoy seguro de que los fallos conocidos podrían haberse pulido aún más.

# Bibliografía y referencias web


[1] "Juegos Serios" (Accedido en 10 de junio de 2023) https://es.wikipedia.org/wiki/Juego_serio

[2] "Scrum (desarrollo de software)" (Accedido en 30 de agosto de 2023) https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

[3] "Itch.io" (Accedido en 10 de junio de 2023) https://es.wikipedia.org/wiki/Itch.io

[4] "Página principal de Unity" (Accedido en 10 de junio de 2023) https://unity.com/es

[5] "Página principal de Unity Documentation" (Accedido en 10 de junio de 2023) https://docs.unity.com/

[6] "C Sharp" (Accedido en 10 de junio de 2023) https://es.wikipedia.org/wiki/C_Sharp

[7] "Documentación de C#" (Accedido en 10 de junio de 2023) https://learn.microsoft.com/es-es/dotnet/csharp/

[8] "Cómo escribir el Game Concept" (Accedido en 30 de agosto de 2023) https://www.fsansberro.com/post/c%C3%B3mo-escribir-el-game-concept

[9] "Dome Keeper – Página principal" (Accedido en 30 de agosto de 2023) https://rawfury.com/games/dome-keeper/

[10] "Risk Of Rain" (Accedido en 30 de agosto de 2023) https://en.wikipedia.org/wiki/Risk_of_Rain

[11] "Hyper Light Drifter" (Accedido en 30 de agosto de 2023) https://es.wikipedia.org/wiki/Hyper_Light_Drifter

[12] "Dungeon of The Endless" (Accedido en 30 de agosto de 2023) https://en.wikipedia.org/wiki/Dungeon_of_the_Endless

[13] "Outer Wilds – Página principal" (Accedido en 30 de agosto de 2023) https://www.mobiusdigitalgames.com/outer-wilds.html

[14] “Astroneer – Página principal” (Accedido en 30 de agosto de 2023) https://astroneer.space/

[15] “Angry Birds Space” (Accedido en 30 de agosto de 2023) https://es.wikipedia.org/wiki/Angry_Birds_Space

[16] “Papers, Please” (Accedido en 30 de agosto de 2023) https://es.wikipedia.org/wiki/Papers,_Please

[17] “Animation & Animator | Build a 2D Platformer Game in Unity #4” (Accedido en 13 de junio de 2023) https://www.youtube.com/watch?v=GChUpPnOSkg&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=8

[18] “How Do I Detect When A Button Is Being Pressed & Held On – EventType” (Accedido en 21 de junio de 2023) https://forum.unity.com/threads/how-do-i-detect-when-a-button-is-being-pressed-held-on-eventtype.352368/

[19] “How to make a Dialogue System in Unity” (Accedido en 15 de junio de 2023) https://www.youtube.com/watch?v=_nRzoTzeyxU&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=4

[20] “Introduction to Audio in Unity” (Accedido en 21 de julio de 2023) https://www.youtube.com/watch?v=6OT43pvUyfY&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=18

[21] “Line Renderer component” (Accedido en 19 de julio de 2023) https://docs.unity3d.com/Manual/class-LineRenderer.html

[22] “Documentación de Unity – Vector3.MoveTowards” (Accedido en 10 de junio de 2023) https://docs.unity3d.com/ScriptReference/Vector3.MoveTowards.html

[23] “How to Create Bouncy Balls, Icy Surfaces and More using 2D Physics Materials in Unity!?” (Accedido en 3 de julio de 2023) https://www.youtube.com/watch?v=UuzmU_lwk2s&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=19

[24] “How to make Predicted Trajectory of Projectiles in Unity – Bow and Arrow 2D” (Accedido en 1 de agosto de 2023) https://www.youtube.com/watch?v=3DUmpVi82q8&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=11

[25] “Mario Galaxy Unique Planet Gravity | UnityTutorial” (Accedido en 14 de junio de 2023) https://www.youtube.com/watch?v=aZOyZJhreSU&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=1

[26] “Tilemap & Tile Palette | Build a 2D Platformer Game in Unity #2” (Accedido en 13 de junio de 2023) https://www.youtube.com/watch?v=QkbGr1rAya8&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=2&t=1448s

[27] “How To Make 2D One-Way Platforms in Unity (Accedido en 8 de junio de 2023) https://www.youtube.com/watch?v=7rCUt6mqqE8&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=12

[28] “Save & Load System in Unity” (Accedido el 15 de junio de 2023) https://www.youtube.com/watch?v=XOjd_qU2Ido&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=3

[29] “Saving Data as JSON in Unity” (Accedido el 15 de junio de 2023) https://prasetion.medium.com/saving-data-as-json-in-unity-4419042d1334

[30] “Adding Multiple Levels | Build a 2D Platformer Game in Unity #12” (Accedido en 13 de junio de 2023) https://www.youtube.com/watch?v=dO5BzWYqEdY&list=PL1ArKGExDGl5fopbIQRySsHlHoSNy2lGo&index=6

[31] “Unity Tilemap: Fix Gaps Between Tiles” (Accedido en 11 de julio de 2023) https://www.youtube.com/watch?v=pXc-H0pb668

# Apéndice A

# *Game Design Document (GDD)*

## A.1 CONCEPTO

- **Título:** Rocket To Home
- **Diseñador:** Pablo García López
- **Plataforma:** PC (Windows)
- **Sinopsis, Historia y Jugabilidad:**

  El juego busca el acercamiento al usuario de la historia del espacio y el papel de la humanidad en ella, principalmente con los temas de la carrera espacial (que acabó con el aterrizaje en la Luna), la astronomía y la mitología a lo largo de la historia.

  La historia tratará de cómo el protagonista (el jugador) debe seguir el mismo camino que otros antecesores suyos (principalmente se centrará en la historia de uno concreto) para aprender acerca de la historia del planeta original de su especie y, finalmente llegar hasta allí. Aquí se presentará una última decisión en la que el propio jugador podrá elegir si quedarse con los de su especie o continuar e ir a dicho planeta.

  Esto se desarrolla a lo largo de varios niveles, que principalmente varían entre controlar al protagonista dentro de un planeta, o bien la nave en el espacio. En los niveles a pie se expondrá la correspondiente información, se ofrecerán varias preguntas estilo trivial o test y se resolverá una prueba final que consistirá en un puzle basado en físicas. Con estas preguntas existe un sistema de puntuación, con el cual se fomenta que el jugador intente acertar siempre que pueda.

  En el nivel de la nave se moverá por un mapa que consiste en un sistema solar en miniatura y el jugador usará la nave para desplazarse entre los niveles (planetas).

- **Categoría:** El juego principalmente se clasifica dentro del género de puzles, y estará enmarcado dentro del género de plataformas para avanzar por los niveles, aunque con un menor énfasis.
- **Mecánicas:** Las principales mecánicas consistirán en:

    - En la fase del tipo *Planeta*, poder moverse por los escenarios en 2D, principalmente consistiendo en desplazamiento lateral y salto.
    - Interacción con paneles, que mostrarán la información pertinente, o bien darán acceso a las correspondientes pruebas, ya sean preguntas o las notas con la historia.
    - Interacción con los distintos minijuegos, o "pruebas", que consistirán en una serie de preguntas o un puzle basado en físicas.
    - En la fase del tipo *Espacio*, desplazamiento de la nave interactuando con una interfaz que muestra unos botones para controlarla. Estos controles serán sencillos (hacia delante, hacia atrás y girar sobre sí mismo para la derecha o izquierda).
- **Público:** El juego está pensado para que lo puedan disfrutar personas de todas las edades, pero quizás al estar más enfocado a enseñar y mostrar un tema, esté más enfocado a un público más joven, tengan o no experiencia previa con un videojuego.

## A.2 PERSONAJES

- **Protagonista:**
    - Es el personaje que controla el jugador. En un principio no aporta nada al diálogo, es un mero espectador de la historia. En cierto modo refleja lo que el jugador va aprendiendo a lo largo del juego.
    - Su papel en la historia es la de un miembro de la raza del planeta en el que vive, que ha alcanzado la edad en la que debe pasar por el mismo *Rito* que otros anteriores a él. Conforme avanza el juego, irá descubriendo la historia tanto de los Natales (los antecesores de la especie del protagonista) y de uno de los que recorrieron el mismo camino que él tiempo atrás.
    - En un principio puede moverse lateralmente, saltar e interactuar con los diferentes elementos.
- **Anterior:**
    - Es un miembro de la misma especie que el protagonista.
    - Su papel en la historia es la de contar su experiencia haciendo el *Rito*, dejando por el camino unas notas. En estas se puede ver que en un principio no le hacía mucha gracia hacer el *Rito*, pero poco a poco se dará cuenta de que es lo que su pueblo espera de él, así que

tiene que terminarlo. Esto puede llegar a sembrar dudas al jugador de cara a la decisión final.

- **Tutora:**
    - Otro miembro de la misma especie que el protagonista.
    - Su papel en la historia es la de explicar los controles y poner en contexto de la historia al jugador. Aparece en el primer planeta.
- **Los Natales:**
    - Son los antepasados de la especie del protagonista.
    - Su papel en la historia es la de dar la información pertinente al jugador por medio de paneles de información en unas "Salas del saber" (la carrera espacial hasta el aterrizaje en la Luna en el caso del primer planeta y la astronomía y mitología en el caso del segundo).

## A.3 MECÁNICAS DEL JUEGO

- **Movimiento:** En la fase del planeta, el jugador puede desplazarse lateralmente (a la izquierda o a la derecha) con las flechas o con las teclas A y D, y saltar con la barra espaciadora. En la fase del espacio el jugador puede desplazarse mediante las flechas (arriba o W para avanzar, abajo o S para retroceder, derecha o D para girar hacia la derecha e izquierda o A para girar a la derecha) o unos botones sobre los que puede hacer clic. En cualquiera de los dos tipos de fases se puede acceder al menú pulsando la tecla escape.
- **Interacción:** El jugador puede interactuar con distintos elementos siempre que se le indique por pantalla con la tecla E.
    - *Diálogos o paneles de información:* estos diálogos o paneles muestran información pertinente mediante texto por pantalla. Se puede avanzar con la tecla E, como se indica.
    - *Paneles de preguntas:* se accede a ellos con la tecla E, y para responder se deberá clicar con el ratón alguna de las posibles respuestas. Una vez hecho, para salir o continuar, se indica que hay que pulsar la tecla E.
    - *Cajas:* se cogen con la tecla E y se sueltan de nuevo con la tecla E.
    - *Portales:* se pulsa la tecla E para cruzar el portal.
    - *Cañón:* se utilizan las teclas AD, o bien las flechas izquierda o derecha para aumentar o disminuir la potencia respectivamente, y

las teclas WS, o bien las flechas arriba y abajo para cambiar el ángulo de tiro, la barra espaciadora para disparar y Q para salir.

  - *Laberinto:* se utilizan las teclas de las flechas izquierda o derecha para girar la estructura para un lado o para otro, y la tecla Q para dejar de interactuar con el laberinto.
  - *Objeto esencial:* una vez interactuado se añade al inventario del jugador.
  - *Nave:* sirve para salir del nivel. Solo se puede utilizar una vez obtenido el objeto correspondiente del nivel y tener una puntuación de al menos 50%.

- **Guardar y Cargar:** El jugador puede guardar desde la propia partida y cargar dicha partida desde el menú principal. También se guarda automáticamente la partida al salir de un nivel o entrar en otro.
- **Menú:** Hay dos tipos de menú: el principal y el de juego.
  - Menú principal: aquí el jugador puede elegir si quiere empezar una nueva partida, cargar la partida o salir del juego haciendo clic con el ratón en el botón correspondiente.
  - Menú del juego: este menú se podrá abrir dentro de cualquier nivel siempre que no se esté interactuando con nada, y permitirá al jugador guardar la partida y salir al menú principal, o salir del menú y continuar.
- **Pruebas Principales:** Aquí se explica en mayor profundidad en qué consisten las principales pruebas que tiene que superar el jugador.
  - *Paneles de preguntas:* una de las formas de poner a prueba el conocimiento adquirido. Aquí se mostrará una pregunta y cuatro posibles repuestas, de las cuales el jugador tendrá que elegir una y hacer clic con el ratón. Si el jugador acierta, su puntuación media mejorará y podrá avanzar a la siguiente pregunta, o bien si ya ha respondido todas las de ese panel, lo habrá completado y conseguirá abrir la puerta. Si falla, entonces se le recomienda revisar las notas que ha tomado o la información directamente del panel. Cuando vuelva a intentar resolver el panel, se encontrará siempre con una pregunta distinta.
  - *Cubos:* esta prueba consiste en llevar un cubo con la respuesta correcta a una plataforma que muestra una pregunta. Cada caja

tiene un color que corresponde con una respuesta. Si el jugador coloca la caja correcta en la plataforma, activará un mecanismo (este puede ser una plataforma móvil o una puerta). Si coloca una caja incorrecta, no pasará nada.

  - *Portales:* aquí se muestra una pregunta al principio, y cuatro posibles respuestas, cada una con un portal. Si se interactúa con el portal que representa la respuesta correcta, avanzará a la siguiente pregunta, y si falla, retrocederá a la pregunta anterior.
  - *Cañón:* esta prueba consiste en disparar una bola con el cañón para encestarla. Tiene tres fases, y al completarlas se abre la puerta de abajo. En la primera fase la gravedad es normal, en la segunda es menor (la bola pesará menos) y en la tercera la gravedad es mayor (la bola pesará más).
  - *Laberinto:* esta es la última prueba, y en ella el jugador tiene que conseguir sacar el objeto esencial para poderlo recoger. Para ello tiene que girarlo y guiar la bola por el camino correcto.

- **Puntuación:** Para gamificar el progreso del jugador a lo largo del juego, existe un sistema de puntuación para los paneles de preguntas. Esta puntuación consiste en una media entre las preguntas acertadas y las preguntas contestadas. De esta forma, se fomenta que el jugador no vaya respondiendo al tuntún y tenga un objetivo en mente, además de avanzar en el nivel o la historia. Para poder salir del nivel, es necesario que el jugador haya respondido bien al menos un 50% de todas las preguntas que se le han mostrado. En caso contrario, al jugador habrá perdido la partida y tendrá que empezar desde cero, saliendo al menú principal y empezando una Nueva Partida.

## A.4 DIAGRAMAS DE ESTADOS

- General:

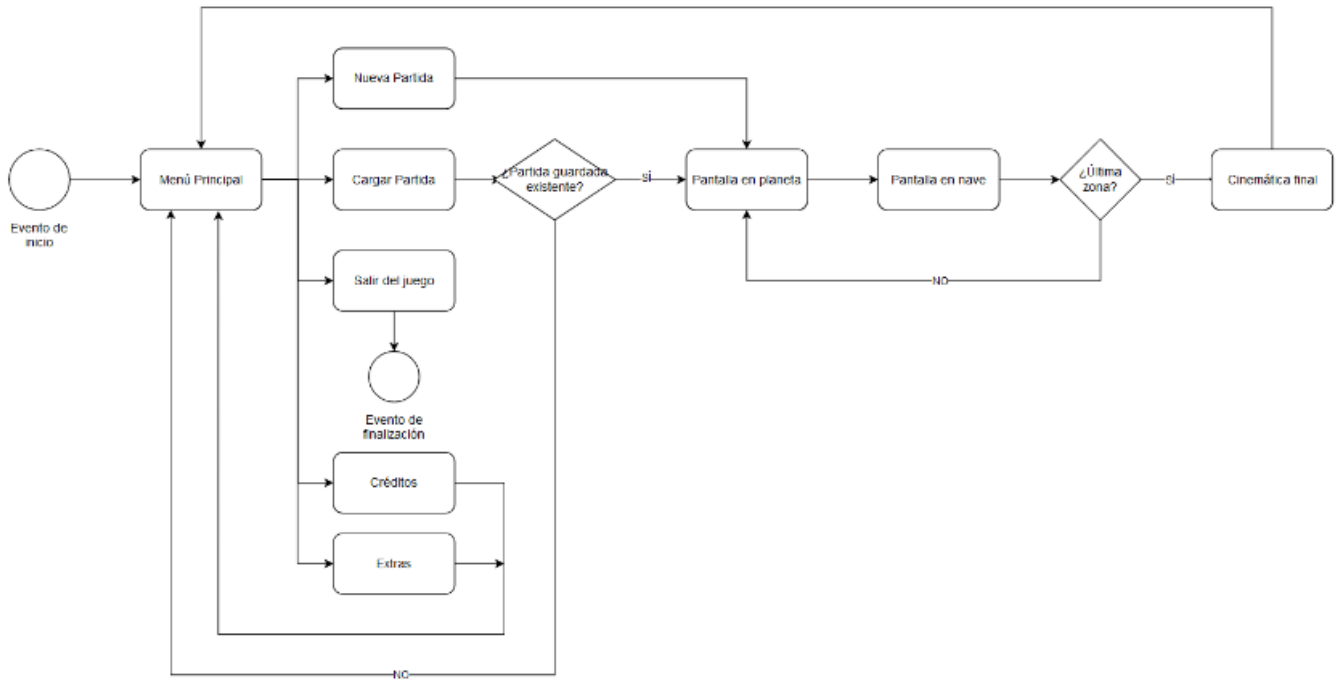


Figura A.1: Diagrama de flujo general

- Fase de planeta 1:

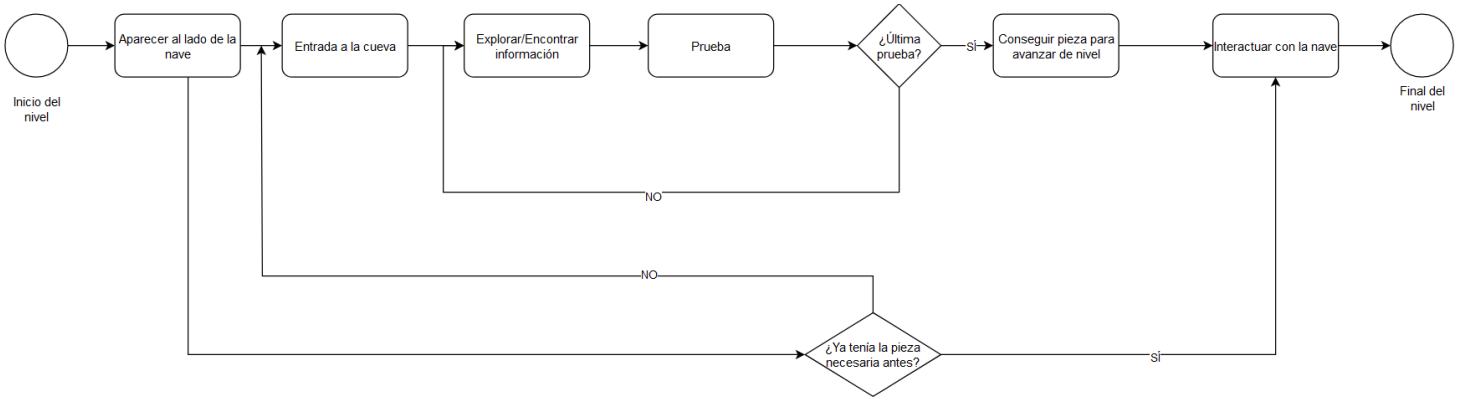


Figura A.2: Diagrama de flujo del primer nivel

- Fase de planeta 2:

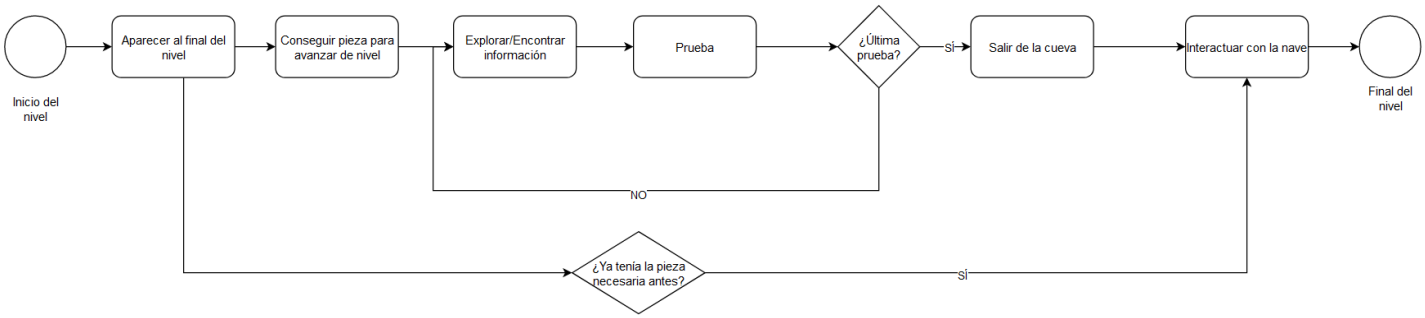


Figura A.3: Diagrama de flujo del segundo nivel

- Fase de nave:

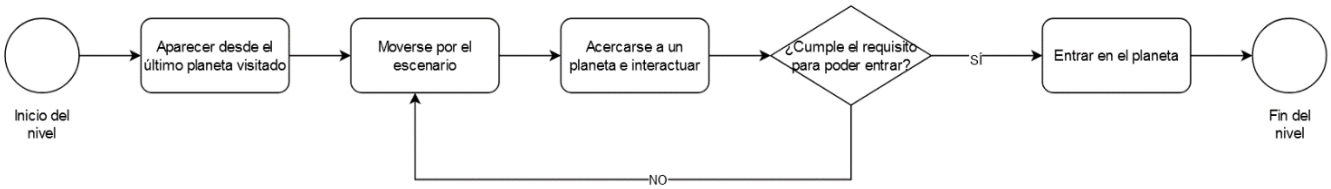


Figura A.4: Diagrama de flujo de la fase del tipo Espacio

- Último nivel:

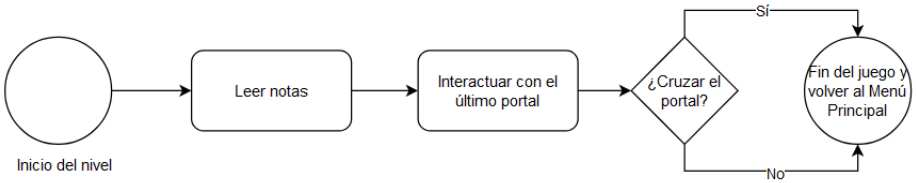


Figura A.5: Diagrama de flujo del último nivel

## A.5 INTERFACES

- **Menú principal:** desde aquí se puede empezar una nueva partida, cargar una existente, ir a la pantalla de extras, salir, o ver los créditos.


Figura A.6: Interfaz del menú principal

- **Menú del juego:** desde aquí se puede guardar la partida y salir al menú principal, o reanudar la partida.


Figura A.7: Interfaz del menú del juego

- **Interacción:** aparece cada vez que se pueda interactuar con algún elemento.


Figura A.8: Interfaz de interacción

- **Diálogo/Información:** esta ventana aparece cada vez que empieza un diálogo, y se bajará cuando termine.

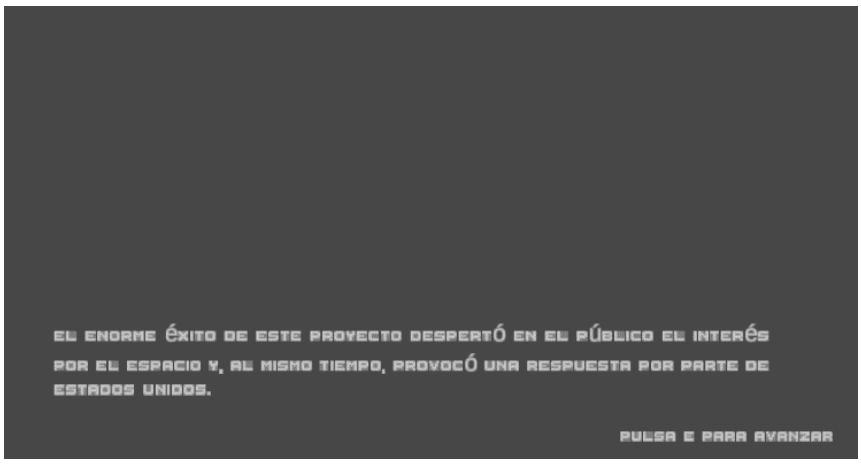


Figura A.9: Interfaz de diálogo

- **Preguntas:** aparece cada vez que se interactúa con un panel de pregunta. Cada botón representa una respuesta, y habrá que hacer clic en la correcta. Desaparece cuando se termina de responder.


Figura A.10: Interfaz de pregunta

- **Controles:** informan al jugador de los controles más específicos de la prueba. Solo aparecen cuando se ha interactuado con dicha prueba.


Figura A.11: Interfaz de controles

- **Controles de la nave:** aparecen como alternativa a controlar la nave con el teclado. Para controlar la nave, solo hay que hacer clic en los botones.

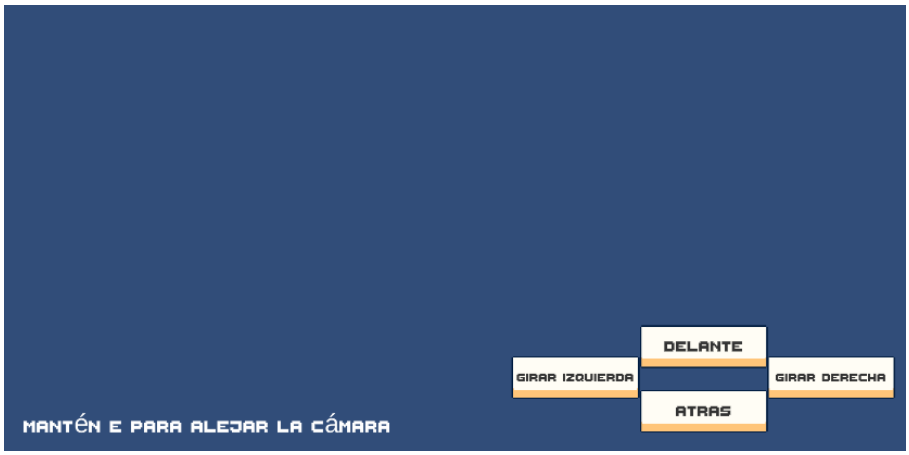


Figura A.12: Interfaz de controles de nave espacial

## A.6 NIVELES

- *Planeta 1: Gaia*

Este es el primer nivel en el que aparecerá el jugador. Aparecerá en el exterior, junto a su nave, y su objetivo es adentrarse en las instalaciones y conseguir la pieza que necesita su nave para salir del planeta. Esto se consigue respondiendo las preguntas y completando los minijuegos. En este nivel las preguntas abarcan la historia de la *Carrera Espacial*, desde que empezó hasta que terminó con el aterrizaje del hombre en la Luna. Una vez conseguida la pieza, se abrirá un portal que le lleva a la entrada de las instalaciones, para que no tenga que ir andando hasta el principio. En este nivel aparecen tanto la Tutora al principio del nivel, como el Anterior a lo largo de las notas que ha ido dejando. La función de la primera es la de explicar los controles y principales cosas que puede hacer el jugador, así como indicarle el objetivo principal. La función del segundo es la de ir contando su historia, que consiste en su experiencia teniendo que hacer el mismo *Rito* que el protagonista.

- *Planeta 2: Kara-B*

Este es el segundo planeta que puede acceder el jugador, y consiste en una suerte de *cara B* del primer nivel. Esto quiere decir que el diseño del mapa es muy parecido al del primero, pero en esta ocasión se tiene que completar a la inversa, es decir, se empieza al principio y se consigue la pieza de la nave en la primera prueba, solo para hacer el mismo recorrido pero adaptado para tener que hacerlo al revés. En este nivel las preguntas abarcan los temas de la historia de

la *astronomía* y la *mitología*, así como su influencia en esta. Y en esta ocasión el único personaje *NPC* que aparece es el Anterior, ya que la Tutora cumplió su objetivo al principio del primer planeta.

- *Zona final: Estación espacial misteriosa*

Este es el último nivel del juego, y solo se puede acceder desde el espacio si se cumple el requisito de tener las dos piezas de los dos planetas anteriores. Este nivel no contiene ni preguntas ni pruebas, ya que supone un cierre narrativo para el jugador. En este nivel se encuentran las últimas notas que dejó el Anterior, además del portal con el que se termina el juego una vez interactuado. Aquí, habiendo seguido la historia del Anterior, se presentan dos opciones: atravesar el portal e irse, o no hacerlo y quedarse. La historia está hecha de forma que ninguna de las dos opciones es mala, y ambas tienen su propia conclusión, para que el jugador elija en función de lo que piense.

## A.7 GUION

Cada línea representa una interacción en el diálogo.

- **Primer Nivel**
  - *Tutora*

Hola.
Estoy aquí para guiarte por las pruebas que te aguardan más adelante, así como las que te esperan más allá.
Para moverte por el planeta:
Puedes moverte con las flechas, saltar con la barra espaciadora e interactuar con el botón E.
Los controles más específicos se te explicarán en las propias pruebas.
Eso que ves en la esquina superior derecha es tu puntuación.
Irá aumentando o disminuyendo según contestes bien a las preguntas que te irás encontrando.
Como una suerte de media. ¡Intenta sacar más del 50%!
Para moverte con tu nave:
Puedes utilizar las flechas para avanzar o girar, o bien los botones que aparecerán en la pantalla.
Y puedes mantener pulsado el botón E para tener una vista más amplia de tus alrededores.

Por último, puedes pulsar el botón Escape para acceder al menú.
Si necesitas recordar lo que te acabo de contar, estaré aquí mismo.
Buena suerte.
Y recuerda, si te sientes perdido, no dudes en explorar más. ¿Quién sabe qué te puedes encontrar por ahí?

- *Anterior*
  - *Primer diálogo, antes de entrar en las instalaciones*

Si encuentras esto, es que he muerto.
¡Es broma! Estoy dejando unas notas grabadas por ahí.
Como una especie de cuaderno de bitácora.
Yo, al igual que tú, desconocido oyente, estoy empezando este viaje.
¡Qué nervios! ¿Qué cosas me esperarán dentro de las ruinas?
En fin, solo hay una forma de saberlo...

- *Segundo diálogo, nada más entrar en las instalaciones*

¡Wooooooow!¡Mira todo esto!
¿Cuántos años pueden tener estas instalaciones? ¡Es fascinante!
Me pregunto cómo de avanzada sería aquella civilización.

- *Tercer diálogo, al lado del primer panel de pregunta*

Hmm... acabo de encontrar un dispositivo con una pantalla táctil.
Parece que cuando interactúo con la pantalla, esta automáticamente muestra una especie de prueba.
Un momento... ¿Esto es una pregunta? Qué curioso.
Desconozco totalmente la respuesta a lo que me pregunta... tendré que seguir buscando.
El camino antes se bifurcaba... Es un buen punto de partida.

- *Cuarto diálogo, en la primera sala del saber*

Parece que estos cacharros son capaces de almacenar información y mostrarla dentro de mi cabeza cuando interactúo.
¿Qué clase de tecnología es capaz de hacer eso?
Será mejor que apunte lo que me dice. Tengo la sensación de que está relacionado con ese panel de ahí abajo.

- *Quinto diálogo, detrás de la primera puerta*

¡Ja! ¡Chúpate esa! ¡Lo conseguí!
¡Nada puede pararme ahora!

- *Sexto diálogo, al principio de la prueba de las cajas*

Pues parece que sí hay algo que pudiera parame ahora...
En fin, parece que estas cajas activan aquel mecanismo del fondo.
Sí que eran creativos estos natales.

- *Séptimo diálogo, en la segunda sala del saber*

Me pregunto qué le pasaría a una civilización tan avanzada para desaparecer así sin dejar rastro.
Tanta información almacenada.
Tanta tecnología dejada atrás.
¿Dónde estarán?
¿Qué les pasó?
Tengo tantas preguntas...
Solo hay una forma de responderlas, terminar esto.

- *Octavo diálogo, antes de la segunda puerta*

Las pruebas se están volviendo cada vez más creativas.
Y la emoción por ver que habrá detrás de cada puerta no deja de crecer
Un momento... uno, dos, tres...
¡Aquí falta una caja!

- *Noveno diálogo, detrás de la puerta con la caja blanca*

(suspiro) Al fin, un sitio donde poder descansar un poco.
Pienso en todo lo que me queda por ver y se me reponen las energías.
Pero, en el fondo, creo que una parte de mí no pensó en las consecuencias de todo esto
No pensó en todo lo que dejo atrás.
Al igual que ellos.
...
Bueno, será mejor que vuelva a lo mío, antes de que me lo piense demasiado
¡Natales, allá voy!

- *Décimo diálogo, antes de la prueba de los portales*

¡¡¿¿TIENEN PORTALES??!!
¿Hasta dónde les llegaba la genialidad?

- *Undécimo diálogo, después del primer portal*

Vale, he cruzado bien el portal.
Y sigo de una pieza.
Un momento... ¿sigo siendo yo o soy una reconstrucción de mi otro yo que cruzó por el otro lado?

Meh, supongo que no importa.

- *Duodécimo diálogo, antes de la prueba del cañón*

Heh, supongo que los natales también se divertían.

Parece que tengo que intentar encestar en aquella estructura de allí

Esto va a estar bien.

- *Decimotercer diálogo, en la tercera sala del saber*

Sigo pensando que hay algo raro en todo esto...

Pero al mismo tiempo hay algo fascinante.

Algo desconocido.

Algo intrigante.

Que me obliga a continuar.

A ver dónde termina esto.

- *Decimocuarto diálogo, antes de los tres últimos paneles de preguntas*

Hum... parece como si detrás de esa puerta hubiera algo importante.

Como si solo quien pasase estas pruebas pudiera encontrarlo.

Espero estar listo para ello. ¡Deséame suerte!

- *Decimoquinto diálogo, antes de la prueba del laberinto*

¡Lo conseguí!

Ahora bien, ¿qué es eso de ahí?

Hay algo brillante dentro, voy a intentar sacarlo. Parece importante.

- *Decimosexto diálogo, antes del último portal de salida*

¡Era la pieza de la que me hablaron! Ahora puedo ponérsela a mi nave.

Se acaba de abrir un portal cuando la he tocado.

Por la luz que emana diría que me lleva... ¿a la entrada? ¡Genial!

En fin, creo que esta va a ser mi última nota aquí.

Si estás leyendo esto, significa que tú también has llegado hasta aquí ¡Enhorabuena!

Nos vemos dondequiera que nos lleve este viaje.

- **Segundo Nivel**
  - *Primer diálogo, al principio del nivel*

(Escuchas una estática seguida de una voz familiar)

¿Qué? ¿Dónde estoy? ¿Qué ha pasado?

Un momento... esto me suena, esa estructura ya la he visto antes...

Pero hay algo que no encaja.

- *Segundo diálogo, después de la primera puerta*

Vale, vale, no te asustes.
Ya tienes la pieza de la nave.
Tal vez estés atrapado aquí.
pero esto ya lo has vivido antes, ya te imaginas lo que puedes hacer...
Coge aire. Tú puedes.

- *Tercer diálogo, en la primera sala del saber*

Hmmm... misma estructura y tecnología que en otro planeta, pero esta sala no estaba aquí antes.
Sin embargo, es como si estuviera diseñada para que fuera la primera en encontrarse.
Pese a que en la otra ocasión era la última.
¿Por qué?

- *Cuarto diálogo, después del primer portal*

¿He hecho bien en continuar? Mira donde me ha llevado...
Las comunicaciones no funcionan, y mira que lo he intentado.
(suspiro) Tengo que salir de aquí como sea.

- *Quinto diálogo, después del último portal*

Jé, hasta el mecanismo de las cajas están cambiadas.
Veamos... ahora parece que van por números.

- *Sexto diálogo, en la segunda sala del saber*

Esta sala sí que coincide con las otras instalaciones, pero su contenido es distinto.
¿Sería todo esto parte de una prueba todavía más grande?

- *Séptimo diálogo, en la tercera sala del saber*

Por lo menos este rincón sigue aquí, supongo que me tomaré un descanso.
...
¿Qué estoy haciendo? ¿Cómo no he pensado que esto podía pasar?
...
No lo sé...creo que voy a tumbarme un rato, necesito reflexionar un poco...

- *Octavo diálogo, antes de la última puerta*

Si estás leyendo esto, significa que he decidido continuar.
No puedo parar ahora.
No debo parar ahora.
No cuando me queda tan poco.
¡Ánimo, querido oyente desconocido!

- *Noveno diálogo, al salir de las instalaciones*

¡Mi nave! ¡Sigue ahí, de una pieza! ¿Cómo es eso posible? ¿Todo esto estaba planeado?

En fin, creo que no hay mucho tiempo, me largo pitando de aquí.

¡Nos vemos en el próximo destino!

- **Último Nivel**
  - *Anterior*
    - *Primer diálogo*

Aquí estamos.

El final de nuestro viaje.

Las vistas son impresionantes.

- *Segundo diálogo*

Llegados a este punto, la pregunta surge.

¿Hago bien en continuar?

¿En dejar todo atrás?

- *Tercer diálogo*

Pero al mismo tiempo, pienso en las maravillas que me aguardan más allá de aquí.

Y entonces lo tengo claro. Tengo que continuar.

Pero tú, mi querido amigo desconocido, quizás no lo tengas tan claro.

- *Cuarto diálogo*

En fin, última parada. Ya no hay marcha atrás.

¿Continuarás con la tradición y te unirás a nosotros?

¿Te quedarás y te forjarás tu propio futuro?

Elijas lo que elijas, ha sido un placer.

Nos vemos (o no)

- *Elegir irse*

Cruzas el portal mientras piensas en todo los que dejas aquí. Tu hogar, tu familia, todo.

Pero miras al futuro, fascinado por todo aquello que te ha traído hasta aquí.

Das un último vistazo a lo que te rodea. Entonces das el paso.

FIN

- *Elegir quedarse*

Justo antes de cruzar el portal, te das cuenta de que no eres capaz de dejarlo todo atrás.

Comprendes que esto es una tradición, pero prefieres crear tu propio futuro.

Junto a tu familia, junto a tus amigos, en tu hogar.

Das un último vistazo a lo que te rodea, a la aventura que has vivido, y vuelves a la nave.

De vuelta a casa.

FIN

### A.8 FUENTES

Todos los recursos utilizados en este apartado han sido descargados de manera gratuita desde la página itch.io.

- *Sprites utilizados:*
    - Space Station Generator by Norma2D: https://norma-2d.itch.io/space-station-generator
    - Pixel Fantasy Caves by Szadi art: https://szadiart.itch.io/pixel-fantasy-caves
    - Warped: Super Grotto Escape Pack by ansimuz: https://ansimuz.itch.io/super-grotto-escape-pack
    - Legacy Fantasy - High Forest by anokolisa (Free - Pixel Art Asset Pack - Sidescroller Fantasy - 16x16 Forest Sprites by Anokolisa): https://anokolisa.itch.io/sidescroller-pixelart-sprites-asset-pack-forest-16x16
    - Seamless Space Backgrounds by Screaming Brain Studios: https://screamingbrainstudios.itch.io/seamless-space-backgrounds
    - Complete GUI Essential Pack [Paper, Wood, Metal, Hologram, Font] by Crusenho: https://crusenho.itch.io/complete-gui-essential-pack
    - Shikashi's Fantasy Icons Pack (FREE) by shikashipx: https://shikashipx.itch.io/shikashis-fantasy-icons-pack
    - Sideview Sci-Fi - Patreon Collection by ansimuz: https://ansimuz.itch.io/sideview-sci-fi
    - Warped Caves by ansimuz: https://ansimuz.itch.io/warped-caves
    - Pixel Planet Generator by Deep-Fol: https://deep-fold.itch.io/pixel-planet-generator
    - Pixel Art Wallpaper by Norma2D: https://norma-2d.itch.io/pixel-art-wallpaper
    - 16x16+ Robot Tileset by 0x72: https://0x72.itch.io/16x16-robot-tileset?download

    - Pirate Bomb by Pixel Frog: https://pixelfrog-assets.itch.io/pirate-bomb
    - Cannon Gun by BDragon1727: https://bdragon1727.itch.io/cannon-gun
    - UI Buttons by Kicked-in-Teeth: https://kicked-in-teeth.itch.io/button-ui
    - 2D Pixel Art Portal Sprites by Elthen's Pixel Art Shop: https://elthen.itch.io/2d-pixel-art-portal-sprites
    - Animated 2d Portal Spritesheet by Cookiscuit: https://cookiscuit.itch.io/animated-2d-portal-spritesheet
    - Patreon's Top Down Collection by ansimuz: https://ansimuz.itch.io/patreons-top-down-collection
    - Fantasy Knight - Free Pixelart Animated Character by aamatniekss: https://aamatniekss.itch.io/fantasy-knight-free-pixelart-animated-character
- *Audio utilizado:*
    - Shapeforms Audio Free Sound Effects by Shapeforms: https://shapeforms.itch.io/shapeforms-audio-free-sfx
    - DOS-88 Synthwave Music Library by DOS88: https://dos88.itch.io/dos-88-music-library
- *Fuente de texto:* Free Pixel Font - Thaleah by Tiny Worlds: https://tinyworlds.itch.io/free-pixel-font-thaleah
- *Fuente de las preguntas:*
    - "Wikipedia - Carrera Espacial" https://es.wikipedia.org/wiki/Carrera_espacial
    - "Wikipedia – Año Geofísico Internacional" https://es.wikipedia.org/wiki/A%C3%B1o_Geof%C3%ADsico_Internacional
    - "GT Historia de la Astronomía" https://astroelda.com/wp-content/22.html
    - "Astronomía en Babilonia" https://www.astromia.com/historia/astrobabilonia.htm
    - "La Astronomía en el antiguo Egipto" https://www.astromia.com/historia/astroegipto.htm

- “Dioses y Planetas” https://truttafario.com/category/2-astronomia/dioses-y-planetas/

# Apéndice B

# *MANUAL DE ROCKET TO HOME*

A continuación se explican los controles y las principales pruebas que se encontrará el jugador a lo largo de la historia de *"Rocket to Home"*.

## B.1 Instalación y ejecución

Para instalar el juego, es necesario descargar la carpeta denominada como *"TFG – Ejecutable"*. Entonces, dentro se encontrará el ejecutable *"TFG.exe"*. Para iniciarlo, solo hay que hacer doble clic.

## B.2 Historia

La historia principal trata sobre el personaje que el jugador controla, que está empezando un *Rito Tradicional* en tu especie, por el que tienes que recorrer un sendero ancestral a lo largo de varios planetas de tu Sistema para conocer más sobre tus antecesores, los *Natales* y, al final, llegar a un portal que te lleve a su *Sistema Solar*. Para poder llevar esto a cabo, tendrás que conseguir una pieza que permitirá a tu nave abandonar el planeta y continuar con tu aventura.

También se cuenta la experiencia que vivió tu Anterior, un miembro de tu misma especie que hizo el mismo recorrido que tú estás empezando y fue dejando notas a lo largo de los niveles, a modo de diario.


Tu historia comienza en el planeta Gaia. ¿A dónde te llevará tu aventura?

Figura B.1: Captura de pantalla del primer nivel

## B.3 Controles

En el Menú Principal, salvo que se indique lo contrario por pantalla, los botones se pulsan haciendo clic con el botón izquierdo del ratón. Para empezar una partida, haz clic en "Nueva Partida", para cargarla haz clic en "Cargar Partida", y para salir del juego haz clic en "Salir".

El juego cuenta con dos tipos de niveles: los de tipo *Planeta* y los de tipo *Espacio*. Así que hay dos esquemas de controles de movimiento.

Para los niveles de tipo *Planeta*:

- A o flecha izquierda para desplazarse a la izquierda
- D o flecha derecha para desplazarse a la derecha
- Barra Espaciadora para saltar (te puedes mover en el aire)
- E para interactuar, siempre que se indique por pantalla que se puede hacer
- Escape para acceder al Menú, y desde ahí se puede hacer clic en "Guardar y Salir" para volver al Menú Principal, y en "Volver" para reanudar la partida.

Para los niveles de tipo *Espacio*:

- W o flecha arriba para mover la nave hacia delante
- S o flecha abajo para mover la nave hacia atrás
- A o flecha izquierda para girar la nave hacia la izquierda
- D o flecha derecha para girar la nave hacia la derecha
- También se pueden utilizar estos controles haciendo clic sobre los botones que aparecen por pantalla.
- Mantén E para alejar la vista de la cámara, y así poder ver mejor el Sistema.

Si en algún momento te sientes perdido. El *NPC* Tutora puede ayudarte y explicarte todo esto. Se encuentra al principio del primer nivel.

Cada prueba y minijuego tiene sus propios controles, y se explican en sus respectivas secciones.


Para entrar en un planeta, acércate y deja que su gravedad te haga entrar. Un momento, ¿qué es eso a tu derecha?

Figura B.2: Captura de pantalla del nivel del tipo Espacio

## B.4 Preguntas

Responder preguntas es la principal forma de avanzar en este juego. A lo largo de los niveles de tipo *Planeta* te encontrarás varios paneles con los que se puede interactuar, y que te presentarán una pregunta de tipo test, con 4 posibles respuestas, pero solo 1 será la correcta. La información necesaria para responder estos paneles se encuentra en unas *"Salas del Saber"* que se encuentran dispersas por los niveles. El primer nivel, *Gaia*, contendrá preguntas sobre la historia de la *Carrera Espacial*, mientras que el segundo nivel, *Kara-B*, contendrá preguntas acerca de la historia de la *Astronomía y la Mitología*.

Para hacer la cosa un poco más interesante, existe una puntuación que se ve afectada por estos paneles, y que irá cambiando conforme aciertes o falles tus respuestas. ¡Intenta sacar más de un 50 % para poder avanzar!

Como un añadido extra, puedes modificar el archivo que contiene las preguntas de cada planeta para poder añadir las que quieras. En el apartado *Extras* del menú principal se te indica el lugar donde se encuentran los archivos correspondientes. Cada pregunta deberá seguir la siguiente estructura para funcionar correctamente dentro de "listaPreguntas":

- pregunta: Aquí escribes la pregunta que quieras.
- textoAcierto: Este es el texto que aparecerá si aciertas la pregunta.

- textoFallo: Este es el texto que aparecerá si fallas la pregunta.
- respuestasPosibles: Una lista de cuatro posibles respuestas. Deben estar todas entre unos corchetes y separados con comas.
- numRespuestaCorrecta: El número del 1 al 4 que indica cuál es la respuesta correcta.
- categoría: Un número del 1 al 3 que indica en qué momento del nivel puede aparecer la pregunta. Normalmente el 1 significa que aparecerá desde el principio, mientras que el 3 solamente en la parte final.

¡Pásatelo bien modificando el juego a tu gusto!

## B.5 Cajas

Este sencillo minijuego presenta una serie de cajas asociadas a un cartel que contiene una pregunta. Cada caja representa una respuesta distinta. En el caso del primer nivel, van asociadas a unos colores, y en el segundo a unos números. El objetivo de este minijuego es llevar la caja correspondiente a la respuesta correcta a una plataforma emparejada con la pregunta por un número. Para interactuar con la caja, solo tienes que pulsar E estando cerca, y para soltarla, volver a pulsar E. Cuando cojas la caja, la llevarás sobre tu cabeza, así que cuando la sueltes caerá sobre ti. Pero no te preocupes, no te hará daño alguno.

Siempre que aciertes, se activará algo. ¡Sigue el rastro de la estela verde para averiguar qué ha activado!


En las Salas del saber encontrarás toda la información que necesitarás. ¿Qué misterios guardarán?

Figura B.3: Captura de pantalla de una de las Salas del saber

## B.6 Portales

Los portales te presentan una serie de preguntas, en donde cada portal tiene una respuesta asociada a ella. Tu objetivo será atravesar varios portales seguidos para poder continuar con tu aventura. Cada portal acertado avanza una pregunta, pero cada fallo te hace retroceder en una pregunta. Para entrar por un portal, pulsa la tecla E cuando estés cerca de uno.

## B.7 Cañón

La prueba del cañón representa un desafío para el jugador, en el que tiene que conseguir disparar una bola hasta una canasta que se va moviendo. Este minijuego consta de tres fases en las que la gravedad actuará de formas distintas. En la primera fase la gravedad es normal, y la bola seguirá la trayectoria predicha. En la segunda fase, la gravedad es menor, y por lo tanto la bola pesará menos e irá por encima de lo predicho. Y en la tercera y última fase, la gravedad es mayor, por lo que la bola pesará más e irá por debajo de lo predicho.

Para controlar el cañón tendrás que interactuar con él pulsando la tecla E una vez estés cerca. Para ajustar el ángulo de disparo, usa las teclas W y S, o bien las flechas arriba y abajo para subirlo o bajarlo. Para darle o quitarle potencia al disparo, utiliza las teclas A y D, o bien las flechas izquierda y derecha para aumentar o disminuirla. También se pueden utilizar las flechas para hacer lo mismo. Para dejar de utilizar el cañón pulsa la tecla Q. Y lo más importante, para disparar, pulsa la Barra Espaciadora.


Puedes disparar todas las veces que quieras, ¡así que apunta y dispara!

Figura B.4: Captura de pantalla del cañón en funcionamiento

## B.8 Laberinto

¡En esta prueba se encuentra la pieza que necesita tu nave para salir del planeta! Tendrás que sacarla de este laberinto inclinándolo hacia la izquierda o la derecha. Para ello, una vez te sitúes en la plataforma, podrás interactuar pulsando E. Para inclinar el laberinto, usa las teclas A y D, o bien las flechas izquierda y derecha para girarlo hacia la derecha o izquierda respectivamente. Para dejar de utilizarlo, pulsa la tecla Q.


Acabas de llegar al planeta Kara-B, pero aquí hay algo que no cuadra...

Figura B.5: Captura de pantalla del segundo nivel

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga