



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**título del TFG
Documentación Técnica**



Presentado por nombre alumno
en Universidad de Burgos — 9 de junio de 2021
Tutor: nombre tutor

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	vi
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	2
A.3. Estudio de viabilidad	4
Apéndice B Especificación de Requisitos	9
B.1. Introducción	9
B.2. Objetivos generales	9
B.3. Catalogo de requisitos	9
B.4. Especificación de requisitos	12
Apéndice C Especificación de diseño	25
C.1. Introducción	25
C.2. Trabajo propio vs. trabajo ajeno	25
C.3. Diseño de datos	27
C.4. Diseño procedimental	28
C.5. Diseño arquitectónico	35
Apéndice D Documentación técnica de programación	47
D.1. Introducción	47
D.2. Estructura de directorios	47

D.3. Manual del programador	49
D.4. Compilación, instalación y ejecución del proyecto	58
D.5. Pruebas del sistema	62
Apéndice E Documentación de usuario	71
E.1. Introducción	71
E.2. Requisitos de usuarios	71
E.3. Instalación	73
E.4. Manual del usuario	76
Apéndice F Documento de diseño del juego	81
F.1. Introducción	81
F.2. Mecánicas del juego	83
F.3. Niveles	86
F.4. Interfaces	90
F.5. Entidades	91
F.6. Controles	100
Bibliografía	103

Índice de figuras

A.1. Captura de pantalla del tablero Kanban utilizado	2
A.2. Tabla de Excel con el sueldo calculado del trabajador	5
A.3. Tabla de Excel con los activos del proyecto	5
A.4. Tabla de Excel con el balance del proyecto	6
A.5. Tabla de Excel con el balance del proyecto con 2 meses más de desarrollo	6
A.6. Tabla de Excel con el balance del proyecto con 4 meses más de desarrollo (precio del videojuego 10 €)	7
A.7. Tabla de Excel con el balance del proyecto con 4 meses más de desarrollo (precio del videojuego 15 €)	7
 B.1. Diagrama de casos de uso	12
C.1. Diagrama UML de las clases que hacen uso del AudioSource . .	28
C.2. Diagrama UML de las clases SpriteAnimator y OneTimeAnimator	29
C.3. Pseudocódigo que resume el sistema de colisiones	30
C.4. Simulación del proceso de detección de colisiones	31
C.5. Vector normal del muro en función de la posición del KinematicObject	32
C.6. Diagrama UML del sistema de colisiones	32
C.7. Diagrama UML de la clase encargada de la gestión de la gravedad del KinematicObject	33
C.8. Pseudocódigo del proceso de modificación de la gravedad del KinematicObject	33
C.9. Clases que heredan de TimeAffectedObject	34
C.10. Pseudocódigo del método llamado para resetear los TimeAffectedObject	35
C.11. Diseño de la clase GameController	36

C.12.Pseudocódigo que refleja todas las operaciones que hay que realizar para establecer un estado de juego inicial estable	37
C.13.Diagrama que representa un ejemplo de las operaciones llevadas a cabo en la llamada al método Tick	38
C.14.Diagrama de transición entre estados de PlayerController	40
C.15.Diagrama de transición entre estados de PlayerController	42
C.16.Diagrama de transición entre estados de PlayerController	43
C.17.Diagrama UML de la clase PatrolObstacle	43
C.18.Pseudocódigo de cálculo del tiempo que lleva recorrer cada sección	44
C.19.Cálculo del tiempo que se tarda en atravesar cada sección	45
C.20.Calcular la siguiente posición a la que debe moverse el obstáculo que sigue una rutina	46
D.1. Consola de Unity	50
D.2. Ventana "Project"de Unity	50
D.3. Ventana "Hierarchy"de Unity	51
D.4. Información asociada al GameObject Main Camera	52
D.5. Ventana "Game"de Unity	53
D.6. Ventana "Scene"de Unity	54
D.7. Importación del proyecto de Unity desde Unity Hub	54
D.8. Selección ruta del proyecto de Unity desde Unity Hub	55
D.9. Proyecto importado correctamente	55
D.10.Ventana de construcción del juego	56
D.11.Selección de plataforma y Sistema operativo	56
D.12.Escenas seleccionadas para exportar	57
D.13.Ficheros generados al exportar el juego	57
D.14.Página de descarga de Unity	58
D.15.Unity Hub	59
D.16.Descarga de una versión de Unity desde Unity Hub	59
D.17.Selección de una versión de Unity desde Unity Hub	60
D.18.Selección de módulos de Unity desde Unity Hub	60
D.19.Consola de Unity	61
D.20.Botones de "Playz "Pause"del editor de Unity	62
E.1. Cubo que rota si el navegador soporta WebGL	75
E.2. Error de la versión del ejecutable de WebGL	75
E.3. Controles de mando del menú	78
E.4. Controles de teclado del menú	78
E.5. Controles de mando de los niveles jugables	80
E.6. Controles de teclado de los niveles jugables	80

F.1. Nivel tutorial de los portales	87
F.2. Pantalla de prueba de mecánicas básicas	88
F.3. Pantalla de prueba de los portales	88
F.4. Pantalla de prueba de los obstáculos móviles	88
F.5. Pantalla de prueba de los creadores de impulso	89
F.6. Pantalla de prueba de los modificadores de gravedad	89
F.7. Pantalla de prueba de los modificadores temporales	90
F.8. Idea inicial de la pantalla de selección de nivel	90
F.9. Idea inicial de la pantalla de opciones	91
F.10. Sprite utilizado para el Player	92
F.11. Sprite utilizado para el obstáculo estático	93
F.12. Sprite utilizado para el obstáculo que sigue una rutina	93
F.13. Obstáculo móvil rápido	94
F.14. Obstáculo móvil (velocidad intermedia)	94
F.15. Obstáculo móvil lento	94
F.16. Sprite utilizado para los portales	95
F.17. Sprite utilizado para la partícula de impulso	96
F.18. Sprite utilizado para el amplificador de impulso	97
F.19. Sprite utilizado para la plataforma de salto	97
F.20. Sprite utilizado para la zona de tiempo escalado	98
F.21. Sprite utilizado para el inversor de gravedad	99
F.22. Sprite utilizado para el obstáculo superdenso	99
F.23. Nomenclatura de los mandos de PlayStation	100

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La metodología de trabajo seguida ha sido la metodología Kanban [10]. El método Kanban toma la filosofía de las metodologías evolutivas e incrementales haciendo énfasis en la intención de la entrega del producto justo a tiempo.

Uno de los elementos clave de esta metodología es el tablero Kanban [5]. Este tablero está dividido en una serie de apartados definidos que representarán el flujo de trabajo de las tareas a realizar. A este tablero se van añadiendo las tareas que se van a realizar, y estas van viajando de una fase de desarrollo de la tarea a otra. Habrá una fase de desarrollo final que represente que esa tarea está completada que será la última fase a la que accedan todas las tareas.

Las tareas del tablero Kanban pueden ser clasificadas en distintas etiquetas según el ámbito de la tarea.

Un videojuego es un producto software cuyo grado de completitud es difícil de estimar, un bug puede retrasar mucho la ejecución. Es además frecuente que no se cumplan los plazos de desarrollo de los videojuegos. Los tableros Kanban puede resultar útil para enfrentar estos problemas ya que son útiles para:

- Ayudar a visualizar el flujo de trabajo, facilitando identificar adelantos y retrasos y conocer el estado del proyecto.
- Identificar y reducir cuellos de botella.

- Es muy compatible con la integración continua.

A.2. Planificación temporal

Tablero Kanban

El tablero Kanban diseñado para este proyecto es uno en el que las tareas pueden estar en cuatro posibles fases de desarrollo:

- **To do:** En esta fase se almacenan las tareas que han sido identificadas y se tiene planeado llevar a cabo pero cuyo desarrollo no ha comenzado todavía.
- **In progress:** La tarea está siendo llevada a cabo.
- **QA/Testing:** Fase de revisión de la tarea. Se comprueba que se ha realizado correctamente.
- **Done** Esta es la fase final del tablero. Todas las tareas que llegan a esta fase se dan por terminadas.

El tablero Kanban se puede encontrar en el repositorio de Github del proyecto en el apartado de "Projects" (<https://github.com/Kencho/mri1001-tfg/projects/1>)

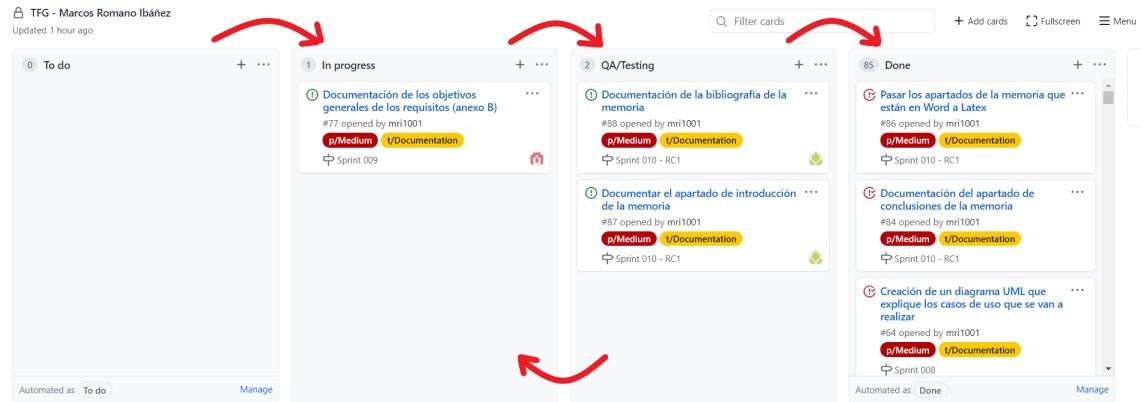


Figura A.1: Captura de pantalla del tablero Kanban utilizado

Fases del desarrollo

Proyecto consta de tres fases separadas del desarrollo:

- La fase de desarrollo del proyecto en la que se implementarán las mecánicas y los elementos del juego. Esta fase es la que más tiempo llevará, durando desde el 1 de febrero de 2021 hasta el 16 de mayo de 2021. Salvo por algunos bugs que quedaron pendientes de solucionar, los objetivos planteados para esta fase se cumplió satisfactoriamente.
- La fase de beta en la que se considera que el juego ya ha tomado forma como producto software y se entra a una fase centrada en la solución de bugs y documentación de la memoria. Esta fase duró desde el 17 de mayo de 2021 hasta el 6 de junio de 2021.
Durante esta fase se realizaron refactorizaciones (facilitan encontrar cambios), corrigieron bugs, se documentó la memoria y se añadieron recursos estéticos para mejorar la sensación de juego (principalmente sonidos).
Se invirtió demasiado tiempo en la implementación de los recursos que mejorasen la sensación de fuego (sobre todo la opción de modificación del volumen de la música) dedicándose poco tiempo a la memoria y ralentizando el cierre del proyecto.
- La fase de "Release candidate" que durará una semana donde comprobará que el desarrollo está listo, el código es accesible y el proyecto se puede descargar y ejecutar satisfactoriamente. Esta fase requerirá más trabajo del esperado ya que el retraso generado en la fase de beta ha obligado a arrastrar algunas de las tareas de esta a la "Release candidate". Sin embargo este retraso es más asumible debido a que se ha hecho la planificación temporal contando con un colchón de tiempo por si algún contratiempo retrasaba el proyecto.

Cuellos de botellas

Durante el desarrollo del proyecto se han encontrado tres cuellos de botella en los que la realización de ciertas tareas ha llevado más tiempo del esperado:

- La implementación del Player: La implementación del Player y todas las mecánicas relativas a este se tenía la percepción de que en 25-30 horas estarían implementadas. Sin embargo este proceso se alargó hasta las 48 horas.

- La implementación del sistema de colisiones estaba planeada para durar una semana de trabajo. Sin embargo se alargó hasta las dos.
- No estaba planeado añadir la opción de modificación del volumen de la música. Añadir esta tarea improvisada sobre la marcha, requirió de dos días que estaba planeado se dedicasen a tareas no improvisadas.

A.3. Estudio de viabilidad

A continuación se va a proceder a realizar el estudio de viabilidad del proyecto. Dejar claro que este estudio de viabilidad no va a ser representativo de un estudio de viabilidad de un videojuego, ya que este TFG se ha centrado sobre todo en el punto de vista del programador obviando el resto de vertientes del proyecto como el apartado artístico o sonoro en los cuales se ha hecho uso de recursos de la comunidad gratuitos, de manera que los sueldos y otros gastos asociados a estos trabajadores no van a constar en el estudio de viabilidad económica.

Otro tema que diferirá de otros videojuegos desarrollados con Unity será que si en el último ejercicio fiscal los ingresos fueron superiores a 100.000 dolares americanos, se está obligado a comprar la edición profesional de Unity [9]. Al no comercializar el juego no es necesario estar pendiente de estos gastos. Presuponiendo también que es el primer producto desarrollado con Unity no habrá precedentes que obliguen a obtener la licencia profesional de Unity.

Viabilidad económica

Se va a valorar la viabilidad económica del proyecto desde el punto de vista económico. Para ello se evaluarán dos aspectos: los costes y los beneficios.

Los cálculos se harán asumiendo que el proyecto lo lleve a cabo una empresa y que se tenga liquidez suficiente como para no requerir financiamiento.

Costes

Costes de recursos humanos: El salario medio de un programador de videojuegos en España es de 32.100 € [4]. Sin embargo el sueldo varía mucho de un programador senior a uno junior. Se va a asumir que en este caso el programador es junior (es un alumno de 4º). El sueldo neto medio mensual de un programador junior es de 1.160 €.

Gatetoria profesional	Salario base mensual	SS a cargo de la empresa	SS a cargo del trabajador	SS total	IRPF	Coste total mensual	Cote total 5 meses
Programador Junior	1160	29,90%	6,35%	36,25%	30%	1856	9280

Figura A.2: Tabla de Excel con el sueldo calculado del trabajador

Activos: El único activo que ha sido necesario para el desarrollo del proyecto ha sido el ordenador portátil en el que se ha creado el videojuego. Se estima que la vida útil de este activo será de 5 años, con lo que se amortizará en ese tiempo. En los 5 meses que ha durado el proyecto el coste de amortización del portátil es de 100 €

Las ventas no superarán los 100.000 €, así no hará falta pagar la versión profesional de Unity, resultando el software utilizado gratuito.

ACTIVOS	Año 1
Ordenador	1.200
TOTAL	1200

Figura A.3: Tabla de Excel con los activos del proyecto

Beneficios

La media de unidades vendidas de por los videojuegos indies (desarrollados por muy pocas personas) está en torno a las 1.500 € unidades [3]. Si se desea vender el juego, el precio máximo que podría alcanzar es de 2€ por copia vendida, generando 3.000 € de ingresos.

Balance

El proyecto actualmente no es viable económico. Sin embargo cabe destacar que el principal motivo de esto es el precio del juego (2 euros). Es

un precio bajo, pero actualmente tiene muy poco contenido el juego y no tiene sentido pedir más dinero por él.

Dejar claro también que con un 2 meses de producción (centrados en generar contenido jugable) más el juego se podría cobrar dignamente a 5 € y con otros 4 meses a 10 € o incluso 15 €.

Balance actual:

ACTIVOS		Año 1		IVA
Ordenador		1.200		
TOTAL		1200		

CUENTA DE PERDIDAS Y GANANCIAS PREVISIONAL				
INGRESOS		Año 1		
Ventas		2.370,00 €		
GASTOS		Año 1		
Amortización		-100,00 €		
Personal		-9.280,00 €		
TOTAL GASTOS		9.040,00 €		
TOTAL		-7.010,00 €		

Tesorería				
Cobros por ventas		2.370,00 €		
Ordenador		-1.200,00 €		
Sueldos		-9.280,00 €		
Total		-8.110,00 €		

Figura A.4: Tabla de Excel con el balance del proyecto

Balance a los 2 meses:

Figura A.5: Tabla de Excel con el balance del proyecto con 2 meses más de desarrollo

Balance a los 4 meses A los 4 meses de trabajo hay dos opciones:

Que el producto terminado posea el contenido suficiente pero su calidad deje algo que desear, en cuyo caso el juego se cobrará a 10 €.

A.3. Estudio de viabilidad

7

Figura A.6: Tabla de Excel con el balance del proyecto con 4 meses más de desarrollo (precio del videojuego 10 €)

Que el producto terminado posea el contenido suficiente y haya sobrado tiempo para mejorar el apartado artístico. La calidad del juego es buena y se puede pedir 15 € por él.

ACTIVOS	Año 1			IVA
Ordenador	1.200			
		CUENTA DE PERDIDAS Y GANANCIAS PREVISIONAL		
		INGRESOS	Año 1	
		Ventas	17.775,00 €	
		GASTOS	Año 1	
		Amortización	-180,00 €	
		Personal	-16.704,00 €	
		TOTAL GASTOS	9.040,00 €	
		TOTAL	891,00 €	
TOTAL	1200	Tesorería		
		Cobros por ventas	17.775,00 €	
		Ordenador	-1.200,00 €	
		Sueldos	-16.704,00 €	
		Total	-129,00 €	

Figura A.7: Tabla de Excel con el balance del proyecto con 4 meses más de desarrollo (precio del videojuego 15 €)

Viabilidad legal

En cuanto a la viabilidad legal, afortunadamente para este proyecto es sencillo el marco legal.

Los derechos de autoría corresponderán al desarrollador del videojuego y los de explotación a también a él. En caso de trabajar para una empresa los derechos de explotación se transmitirán a esta [1].

Los elementos artísticos utilizados en el videojuego han sido obtenidos de personas que los han cedido para su libre uso. Todo este contenido se han regido por distintas leyes de CopyRigth, pero todas las estas leyes permiten su libre uso pero requiriendo mencionar al autor.

Esas leyes son:

- Attribution 4.0 International (<https://creativecommons.org/licenses/by/4.0/>)
- Attribution 3.0 Unported (<https://creativecommons.org/licenses/by/3.0/>)
- Attribution 2.0 Generic (<https://creativecommons.org/licenses/by/2.0/>)
- Attribution 1.0 Generic (<https://creativecommons.org/licenses/by/1.0/>)

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

El desarrollo del proyecto busca lograr los siguientes objetivos:

- Desarrollar un fichero de diseño del juego que resuma en qué va a consistir el juego y que mecánicas implementará.
- Desarrollar un videojuego que implemente las mecánicas definidas en el fichero de diseño del juego.
- Ofrecer versiones del producto final (el videojuego) para Windows, Linux y WebGL.
- Ofrecer un juego controlable tanto con teclado y ratón como con mando.

B.3. Catalogo de requisitos

Requisitos funcionales

Los requisitos funcionales especificarán cual es el funcionamiento que se espera del producto. Se concretarán a continuación.

- **RF-1 Gestión de menús:** El jugador deber poder navegar por los menús.

- **RF-1.1 Navegación entre pantallas:** Las pantallas deben de permitir navegar a otras pantallas (de manera directa o indirecta).
- **RF-2 Gestión del menú principal:** Deberá haber un menú principal que sea la primera escena que se muestre en el videojuego.
 - **RF-2.1 Selección de nivel:** Debe de poderse acceder a cualquier nivel desde el menú principal.
 - **RF-2.2 Cierre de la aplicación:** Se debe de poder cerrar la aplicación desde el menú principal.
 - **RF-2.3 Viaje al menú de opciones:** Se puede viajar desde el menú principal al menú de opciones.
- **RF-3 Gestión del menú de opciones:** Deberá haber un menú de opciones que permita modificar aspectos generales del juego. Desde el menú de opciones se debe poder volver al menú principal.
- **RF-4 Gestión de niveles:** Los niveles deben de contener todos los elementos necesarios y obligatorios de un nivel.
 - **RF-4.1 Controlar un avatar:** Debe de haber un avatar controlable por el jugador que realice las operaciones que le indique el jugador.
 - **RF-4.2 Zonas de muerte:** Debe haber zonas que maten al avatar del jugador cuando entre en contacto con ellas y devuelvan el nivel a un estado inicial.
 - **RF-4.3 Zonas de victoria (meta):** Debe haber zonas en las que, al entrar, se considere el nivel terminado y se vuelva al menú principal.
 - **RF-4.4 Gestión del menú de pausa:** Se debe poder acceder al menú de pausa, que parará la ejecución del nivel.
 - **RF-4.4.1 Abrir el menú de pausa:** Se deberá de poder abrir el menú de pausa, lo que pausara la ejecución del nivel.
 - **RF-4.4.2 Operaciones del menú de pausa:** Se deben de poder realizar las mismas operaciones que en el menú de opciones.

- **RF-4.4.3 Cerrar el menú de pausa:** Se deberá poder cerrar el menú de pausa, lo que reanudará la ejecución del nivel.
 - **RF-4.5 Manipulación de la gravedad:** Debe de ser posible manipular la gravedad del nivel.
 - **RF-4.6 Manipulación del tiempo:** Debe de ser posible modificar la escala de tiempo que afecta al nivel.
 - **RF-4.7 Aplicación de impulsos:** Se debe poder aplicar impulsos que varíen la trayectoria que lleva un objeto.
 - **RF-4.8 Obstáculos:** Puede haber obstáculos que maten al avatar jugable cuando entre en contacto con ellos.
 - **RF-4.9 Portales:** Puede haber portales que teletransporten al avatar jugable desde el punto en el que se encuentra a otro que corresponderá con otro portal.
- **RF-5 Gestión del avatar del jugador:** El jugador debe contar con un avatar controlable por el este. El avatar debe ser capaz de: saltar, moverse, realizar un acelerón y activar el tiempo bala.

Requisitos no funcionales

Al ser el producto a entregar un videojuego (un software de ocio), resulta clave especificar que requisitos no funcionales se van a tener en cuenta.

- **Facilidad de uso:** Un videojuego cuyo uso no sea sencillo y cómodo puede perder muchos jugadores por esa única razón. Es clave que los jugadores tengan una experiencia agradable cuando interactúen con el juego.
- **Soporte:** Los cambios y actualizaciones del videojuego tienen que ser transparentes al usuario, pues no tiene necesidad de conocer como funciona internamente el juego, sino solo abrir el ejecutable y disfrutar del juego.
- **Apariencia o interfaz externa (*look and feel*):** No se ha especificado un diseño de interfaz, sin embargo en el mundo de los videojuegos hay un modelo general muy establecido. Es lógico adoptarlo para ofrecerle al jugador una experiencia que le resulte familiar.

- **Escalabilidad:** En un videojuego se van a ir añadiendo continuamente funcionalidades sobre el código para poder implementar todas las mecánicas, tanto definidas como que puedan surgir en el futuro. Por ello el código debe ser fácil de mantener y extender.

B.4. Especificación de requisitos

Diagramas de casos de uso

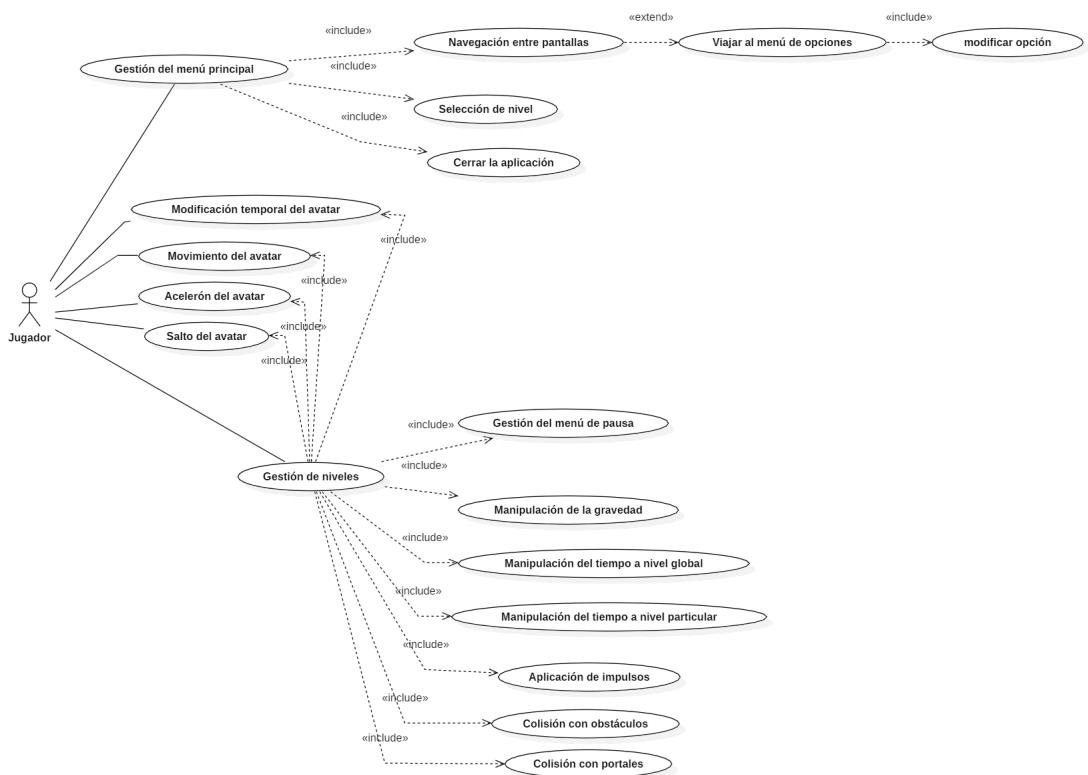


Figura B.1: Diagrama de casos de uso

Actores

Solo habrá un actor: el jugador del videojuego.

Casos de uso

Caso de uso 01

CU-01	Gestión del menú principal
Descripción	Permite al usuario seleccionar escena, cerrar el juego y acceder al menú de opciones
Requisitos funcionales	RF-1, RF-1.1, RF-2, RF-2.1, RF-2.2, RF-2.3
Precondición	Haber ejecutado el juego
Secuencia de pasos	<ol style="list-style-type: none"> 1. Mostrar todos los niveles que se pueden jugar 2. Mostrar el botón de transición al menú de opciones 3. Mostrar el botón cierre del programa
Postcondiciones	Haber transicionado a otra escena
Excepciones	Si se pulsa el botón de cierre del programa parar la ejecución del programa en vez de transicionar a otra escena
Frecuencia	Muy Alta
Importancia	Alta

Caso de uso 02

CU-02	Navegación entre pantallas
Descripción	Permite al usuario cambiar de la escena actual a la escena escogida
Requisitos funcionales	RF-1, RF-1.1, RF-2.1, RF-2.3, RF-3, RF-4.3, RF-4.4.3
Precondición	Tener una escena a la que se desea cambiar
Secuencia de pasos	<ol style="list-style-type: none"> 1. Cargar la escena a la que se va a transicionar 2. Inicializar la escena 3. Cambiar a la escena a la que se ha transicionado
Postcondiciones	Haber transicionado a otra escena
Excepciones	No las hay
Frecuencia	Muy Alta
Importancia	Alta

Caso de uso 03

CU-03	Selección de nivel
Descripción	Permite al usuario elegir el nivel que desea jugar
Requisitos funcionales	RF-1, RF-2, RF-2.1
Precondición	Estar en el menú principal
Secuencia de pasos	<ol style="list-style-type: none"> 1. Se muestra al jugador todos los niveles que se puede jugar 2. El usuario puede desplazarse entre los botones de selección de nivel 3. El jugador se sitúa sobre el botón de selección del nivel del nivel que desea jugar 4. El jugador pulsa el botón de acceso al nivel que desea jugar 5. Se inicializa el nivel seleccionado 6. El jugador entra al nivel seleccionado
Postcondiciones	Haber inicializado el nivel correctamente
Excepciones	<p>Si el nivel no se ha inicializado correctamente.</p> <p>Si no se ha asignado ninguna escena a la que transicionar al pulsar el botón de selección de niveles</p>
Frecuencia	Muy Alta
Importancia	Alta

Caso de uso 04

CU-04	Cerrar la aplicación
Descripción	Permitir al usuario cerrar el juego desde el menú principal
Requisitos funcionales	RF-1, RF-2, RF-2.2
Precondición	Estar en el menú principal

Secuencia de pasos	<ol style="list-style-type: none"> 1. Se muestra al jugador todos los botones con los que puede interactuar 2. El usuario se desplaza al botón de cierre del juego 3. El jugador pulsa el botón de cierre del juego 4. El juego se cierra
Postcondiciones	Haber parado la ejecución del programa
Excepciones	No las hay
Frecuencia	Moderada
Importancia	Alta

Caso de uso 05

CU-05	Viajar al menú de opciones
Descripción	Permitir al usuario viajar al menú de opciones desde el menú principal
Requisitos funcionales	RF-1, RF-2, RF-2.3
Precondición	Estar en el menú principal
Secuencia de pasos	<ol style="list-style-type: none"> 1. Se muestra al jugador todos los botones con los que puede interactuar 2. El usuario se desplaza al botón del menú de opciones 3. El jugador pulsa el botón del menú de opciones 4. Inicializar la escena del menú de opciones 5. Cambiar a la escena del menú de opciones
Postcondiciones	Estar en la escena del menú de opciones
Excepciones	No las hay
Frecuencia	Moderada-Baja
Importancia	Alta

Caso de uso 06

CU-06	Modificar opción
Descripción	Permitir al usuario modificar características generales del juego
Requisitos funcionales	RF-3
Precondición	Estar en el menú de opciones o el menú de pausa
Secuencia de pasos	<ol style="list-style-type: none"> 1. Se muestra al jugador todas las opciones que modificar 2. Seleccionar la opción que se desea modificar 3. Modificar el valor asociado a esa opción 4. Hacer el cambio persistente
Postcondiciones	Haber modificado esa característica general y como afecta al juego
Excepciones	A la hora de mostrar las opciones, si el jugador no le ha dado un valor concreto previamente, se le asignará a la característica general un valor por defecto
Frecuencia	Moderada-Baja
Importancia	Alta

Caso de uso 07

CU-07	Gestión de niveles
Descripción	El usuario manejará con un avatar en un nivel seleccionado hasta salir de él interactuando con los elementos que contiene
Requisitos funcionales	RF-4, RF-4.1, RF-4.2, RF-4.3, RF-4.4, RF-4.5, RF-4.6, RF-4.7, RF-4.8, RF-4.9, RF-5
Precondición	Haber inicializado correctamente el nivel

Secuencia de pasos	<ol style="list-style-type: none"> 1. Se inicializan los objetos de la escena 2. Se muestra el avatar jugable y su posición en el nivel 3. Se le da el control del avatar jugable al jugador 4. Se le da la opción al jugador de interactuar con distintos elementos 5. El avatar jugable podrá morir 6. Se puede abrir el menú de pausa 7. Llegar a la zona de victoria 8. Volver al menú principal
Postcondiciones	Haber vuelto al menú principal
Excepciones	Se puede volver al menú de pausa sin haber pasado por las operaciones 4, 5, 6, 7 y 8
Frecuencia	Muy Alta
Importancia	Muy Alta

Caso de uso 08

CU-08	Gestión del menú de pausa
Descripción	El usuario podrá abrir un menú de pausa similar al de opciones desde el nivel
Requisitos funcionales	RF-4.4, RF-4.4.1, RF-4.4.2, RF-4.4.3
Precondición	Estar en un nivel

Secuencia de pasos	<ol style="list-style-type: none"> 1. Pulsar el botón de apertura del menú de pausa 2. Pausar la ejecución del juego 3. Abrir el menú del nivel 4. El usuario podrá interactuar con menú de opciones 5. El usuario podrá volver al menú principal desde el menú de opciones 6. Pulsar el botón de cierre del menú de pausa 7. Cerrar el menú de pausa 8. Reanudar la ejecución del nivel
Postcondiciones	Haber vuelto al nivel, que se estará ejecutando con normalidad
Excepciones	En caso de pulsar el botón de volver al menú principal se volverá a este y no al nivel
Frecuencia	Moderada-Alta
Importancia	Alta

Caso de uso 09

CU-09	Manipulación de la gravedad
Descripción	Los objetos manipuladores de gravedad interactuarán con el avatar jugable
Requisitos funcionales	RF-4.5
Precondición	Estar en un nivel con objetos manipuladores de gravedad
Secuencia de pasos	<ol style="list-style-type: none"> 1. El avatar jugable entra en contacto con un objeto manipulador de gravedad 2. El objeto manipulador de gravedad modifica como afecta la gravedad al avatar jugable 3. El avatar jugable deja de estar en contacto con el objeto manipulador de gravedad
Postcondiciones	No las hay

Excepciones	Es posible que el objeto manipulador de gravedad deje de modificar la gravedad cuando el avatar
Frecuencia	Moderada-Baja
Importancia	Alta

Caso de uso 10

CU-10	Modificación del tiempo a nivel global
Descripción	Los objetos modificadores del tiempo podrán modificar la escala de tiempo que afecta a los objetos a nivel global jugable
Requisitos funcionales	RF-4.6
Precondición	Estar en un nivel con un avatar jugable
Secuencia de pasos	<ol style="list-style-type: none"> 1. El usuario pulsará el botón de escalado de tiempo 2. La escala de tiempo global será modificada mientras dure el efecto 3. El efecto de dejará de aplicarse 4. La escala de tiempo global volverá al estado en el que se encontraba antes de aplicar el efecto 5. El usuario deberá esperar un tiempo antes de volver a activar este efecto
Postcondiciones	El escala de tiempo tiene que ser la misma que antes de aplicar el efecto
Excepciones	<p>Sí el usuario intenta activar este efecto durante el tiempo que esta deshabilitado el efecto no se aplicará</p> <p>Sí el usuario intenta activar este efecto mientras ya se está llevando a cabo el efecto no se aplicará</p>
Frecuencia	Moderada-Baja
Importancia	Alta

Caso de uso 11

CU-11	Modificación del tiempo a nivel particular
--------------	--

Descripción	Los objetos modificadores del tiempo podrán modificar la escala de tiempo que afecta a un objeto en particular
Requisitos funcionales	RF-4.6
Precondición	Estar en un nivel con modificadores de tiempo a nivel particular Estar en un nivel con objetos a los que les afecte el tiempo a nivel particular
Secuencia de pasos	<ol style="list-style-type: none"> 1. Un objeto al que le afecte el tiempo entra en contacto con el objeto modificador de tiempo a nivel particular 2. Se modifica la escala de tiempo del objeto al que le afecta el tiempo 3. Se cancela la modificación sobre escala de tiempo que efectúa el modificador de tiempo a nivel particular 4. La escala de tiempo provocada por el modificador de tiempo a nivel particular se tiene que haber cancelado
Postcondiciones	La escala de tiempo provocada por el modificador de tiempo a nivel particular se tiene que haber cancelado
Excepciones	Si un objeto al que le afecte el tiempo esta en contacto con varios modificadores de tiempo particulares se aplicarán todas las modificaciones superponiéndose entre ellas
Frecuencia	Moderada-Baja
Importancia	Alta

Caso de uso 12

CU-12	Aplicación de impulsos
Descripción	Los objetos aplicadores de impulsos aplicarán impulsos sobre el avatar jugable
Requisitos funcionales	RF-4.7
Precondición	Estar en un nivel con objetos aplicadores de impulso Estar en un nivel con un avatar jugable

Secuencia de pasos	1. El avatar jugable entra en contacto con el objeto aplicador de impulso 2. El objeto aplicador de impulso aplica un impulso sobre el avatar jugable
Postcondiciones	La velocidad que lleva el avatar debe de haber variado
Excepciones	Un modificador de impulso que aplique un impulso que sea un multiplicador de impulso que lleva un avatar jugable quieto no variará la velocidad de este
Frecuencia	Moderada-Baja
Importancia	Alta

Caso de uso 13

CU-13	Colisión con obstáculos
Descripción	El avatar podrá colisionar con obstáculos
Requisitos funcionales	RF-4.8
Precondición	Estar en un nivel con obstáculos Estar en un nivel con un avatar jugable
Secuencia de pasos	1. El avatar jugable entra en contacto con el obstáculo 2. El avatar jugable muere 3. El avatar jugable reaparece
Postcondiciones	El avatar se encuentra en la posición de inicio del nivel
Excepciones	No las hay
Frecuencia	Moderada-Baja
Importancia	Moderada-Alta

Caso de uso 14

CU-14	Colisión con portales
Descripción	El avatar jugable se teletransportará al colisionar con un portal
Requisitos funcionales	RF-4.9
Precondición	Estar en un nivel con al menos 2 portales Estar en un nivel con un avatar jugable

Secuencia de pasos	<ol style="list-style-type: none"> 1. El avatar jugable entra en contacto con un portal 2. El avatar jugable cambiará su posición con la del portal pareja de aquel con el que ha colisionado
Postcondiciones	La posición del avatar será la misma que la del portal pareja de aquel con el que se ha colisionado
Excepciones	Si el portal no tiene un portal parejo el avatar jugable no cambiará su posición cuando colisione con él
Frecuencia	Moderada-Baja
Importancia	Moderada-Alta

Caso de uso 15

CU-15	Movimiento del avatar
Descripción	Permite usuario ordenarle al avatar jugable realizar un movimiento horizontal
Requisitos funcionales	RF-5
Precondición	Estar en un nivel con un avatar jugable
Secuencia de pasos	<ol style="list-style-type: none"> 1. El usuario pulsa el botón de desplazamiento del avatar jugable hacia la derecha 2. El avatar jugable se desplaza hacia la derecha 3. El usuario pulsa el botón de desplazamiento del avatar jugable hacia la izquierda 4. El avatar jugable se desplaza hacia la izquierda
Postcondiciones	La velocidad del avatar jugable ha variado
Excepciones	Si el avatar jugable esta en contacto con una pared no podrá moverse en esa dirección
Frecuencia	Muy Alta
Importancia	Muy Alta

Caso de uso 16

CU-16	Salto del avatar
Descripción	Permite usuario ordenarle al avatar jugable realizar un salto
Requisitos funcionales	RF-5
Precondición	Estar en un nivel con un avatar jugable
Secuencia de pasos	<ol style="list-style-type: none"> 1. El usuario pulsa el botón de salto del avatar jugable 2. El avatar jugable realiza el salto
Postcondiciones	El avatar se encontrará en el aire
Excepciones	Si el avatar jugable se encuentra en el aire no realizará el salto
Frecuencia	Muy Alta
Importancia	Muy Alta

Caso de uso 17

CU-17	Acelerón del avatar
Descripción	Permite usuario ordenarle al avatar jugable realizar un acelerón
Requisitos funcionales	RF-5
Precondición	Estar en un nivel con un avatar jugable
Secuencia de pasos	<ol style="list-style-type: none"> 1. El usuario pulsa el botón de acelerón del avatar jugable 2. El avatar jugable empieza a realizar el acelerón 3. El avatar jugable desplaza en una dirección durante un tiempo 4. Al acabar el tiempo el jugador deja de realizar el acelerón 5. Desactivar temporalmente el acelerón
Postcondiciones	El avatar ha variado su posición
Excepciones	Si el usuario intenta utilizar el acelerón mientras está desactivado no podrá utilizarlo
Frecuencia	Moderada-Alta
Importancia	Alta

Caso de uso 18

CU-18	Modificación temporal del avatar
Descripción	Permite al usuario ordenarle al avatar jugable realizar modificar el tiempo global del nivel
Requisitos funcionales	RF-5
Precondición	Estar en un nivel con un avatar jugable
Secuencia de pasos	<ol style="list-style-type: none"> 1. El usuario pulsa el botón de modificación temporal del avatar jugable 2. La escala de tiempo global cambia 3. Pasado un determinado tiempo la escala de tiempo global vuelve a su valor inicial 4. Se desactiva la modificación temporal del avatar jugable
Postcondiciones	La escala temporal global vuelve a ser la misma que antes de aplicar la modificación temporal global
Excepciones	Si el usuario intenta utilizar la modificación temporal mientras esta desactivado no podrá utilizarlo
Frecuencia	Moderada
Importancia	Alta

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado se van a explicar elementos del diseño del proyecto que explican su funcionamiento y estructura.

C.2. Trabajo propio vs. trabajo ajeno

Antes de nada es necesario mencionar que este trabajo ha partido de una plantilla que ofrece Unity como punto de partida. Esa plantilla se llama Platformer Microgame y se puede añadir a tus assets en el siguiente enlace: <https://bit.ly/3wZEU8W>.

De esta plantilla se ha conservado sobre todo los elementos estéticos, pero también se ha conservado la clase Simulation y los eventos que esta lanza.

Clase Simulation

La clase Simulation es una clase encargada de manejar los eventos del juego. El objeto GameController hace uso de esta clase para ir ejecutando los eventos a medida que entran en cola. Esta clase tiene una particularidad de C#. Simulaion es una “partial class”. Esto permite que la clase Simulation se construya en varios ficheros distintos. Para el funcionamiento de la clase Simulation, esta hace uso de otras dos subclases: Simulation.Event y Simulation.InstanceRegister. Se va a explicar a continuación por qué son clases que se consideran importantes y claves para entender el funcionamiento de la arquitectura del videojuego.

Simulation: Este fichero contiene la estructura principal del funcionamiento de Simulation. Simulation es una clase estática con una cola, también estática, que guarda eventos (clase Event) y los libera cuando GameController llama al método tick(). Este fichero tiene el método tick() y los métodos necesarios para añadir y remover elementos de la cola.

Simulation.Event: Contiene la clase interna Event que se encarga de ejecutar el comando asociado a ese evento. De esta clase de la que heredan todos los eventos que saltan durante la ejecución del juego (como por ejemplo EnemyDeath, el evento que salta cuando el jugador muere). Los eventos se guardan en su mayoría en la carpeta Assets/Scripts/Gameplay.

Simulation.InstanceRegister: Contiene la clase InstanceRegister. Esta clase simplemente devuelve una instancia nueva de un objeto cualquiera. Esta clase está creada para que Simulation pueda crear singlettons (patrón de diseño) de clases. Es utilizado para que todas las clases trabajen sobre el mismo modelo. Ese modelo es un script denominado PlatformerModel con una clase que exclusivamente tiene una serie de atributos (como el Player, las cámaras o el punto de aparición del jugador) que serán utilizados por varias clases.

Eventos

Los eventos son toda las clases que heredan de Simulation.Event. Todos los eventos se encuentran en la carpeta Assets/Scripts/Gameplay. Casi todas las clases evento del proyecto se han mantenido intactas de Platformer Microgame, sin embargo algunas eventos han sido modificados y otros añadidos. Eventos modificados:

- DisablePlayerInput.
- EnablePlayerInput.
- PlayerDeath.
- PlayerEnteredVictoryZone.

Eventos añadidos:

- SetGameInitialState: Evento que se lanza para forzar volver la escena al estado inicial.

- PlayerObstacleCollision: Evento que salta cuando el Player colisiona con un obstáculo. Este evento mata al jugador.
- LoadGameMenu: Evento que carga la escena del menú principal y cambia a ella.

C.3. Diseño de datos

Los datos necesarios para el desarrollo de un videojuego son los Sprites y los audios utilizados durante su ejecución.

Para utilizar estos recursos en Unity se hace uso de los AudioSource [7] y los SpriteRenderers [8].

AudioSource

La clase AudioSource es la clase de la que hace uso Unity para reproducir audios. Esta clase tiene un atributo público AudioClip que contendrá el audio que se va a reproducir. Se puede saber si el audio se está reproduciendo con el atributo booleano.isPlaying.

Se puede modificar el volumen de reproducción de todo audio reproducido con ese AudioSource.

Los métodos de reproducción de audios son los siguientes:

- Play: reproduce el audio asociado al AudioSource (- PlayDelayed: reproduce el audio asociado al AudioSource despues del tiempo pasado por parámetros.
- PlayOneShot: permite reproducir cualquier audio pasado como parámetro. El audio que se desea reproducir deberá estar encapsulado en una instancia de la clase AudioClip [6].

En el proyecto desarrollado AudioSource se ha usado para controlar el volumen del juego. Se ha creado una clase VolumeManager encargada de guardar y modificar el volumen general del juego y el volumen de la música. Sin embargo, los AudioSource del juego están protegidos por clases envoltorio encargadas de modificar el volumen del juego de acuerdo a las opciones de juego.

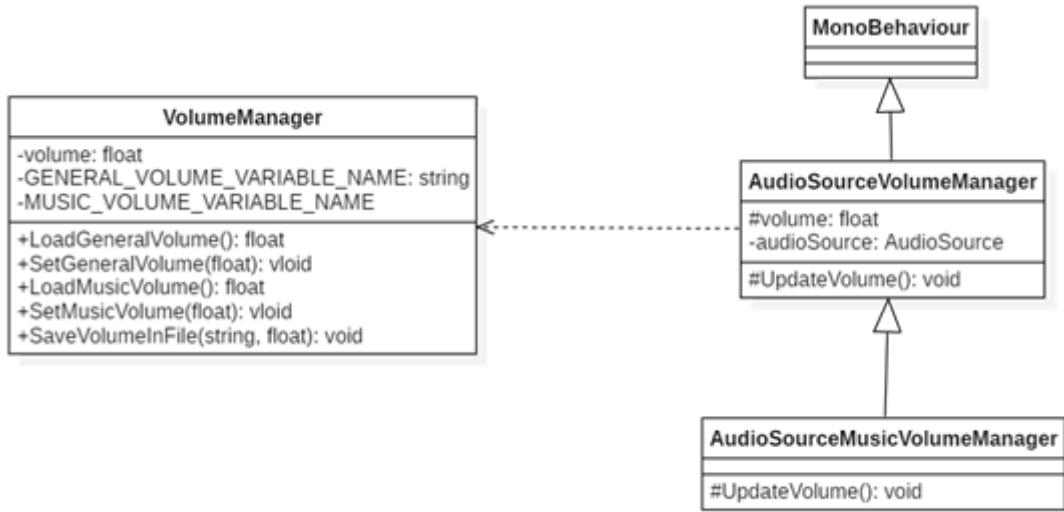


Figura C.1: Diagrama UML de las clases que hacen uso del AudioSource

SpriteRenderer

La clase **SpriteRenderer** es la encargada de hacer que se muestre una imagen en una zona del videojuego. Para poder mostrar las imágenes en Unity hay que utilizar la clase envoltorio **Sprite**. **SpriteRenderer** tiene un atributo público **Sprite** que contiene la imagen que mostrará ese **SpriteRenderer**.

En el proyecto se ha hecho uso de esta clase para mostrar imágenes estáticas y para mostrar animaciones.

Para crear las animaciones se ha creado dos clases **SpriteAnimator** y **OneTimeAnimator**. Estas clases simulan la animación intercalando a grandes velocidades un conjunto de imágenes. La diferencia entre estas dos clases es que **SpriteAnimator** reproduce una animación indefinidamente en bucle y **OneTimeAnimator** reproduce la animación una sola vez.

C.4. Diseño procedimental

En este apartado se explicarán los algoritmos que se han desarrollado para el correcto funcionamiento del videojuego.

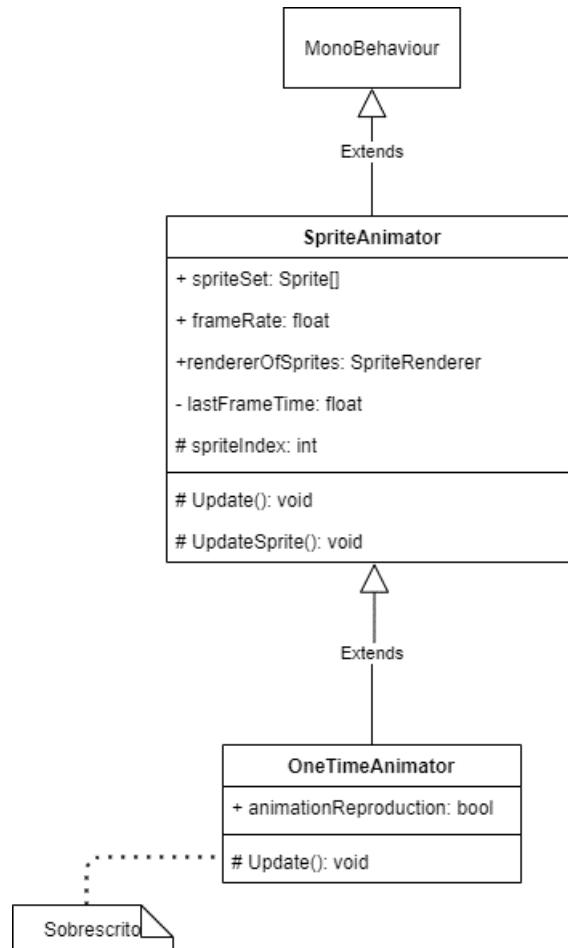


Figura C.2: Diagrama UML de las clases **SpriteAnimator** y **OneTimeAnimator**

Sistema de colisiones

Para simular el sistema de colisiones se optó por utilizar la velocidad como elemento principal.

Sistema de colisiones:**Calcular la posición futura del Player****Comprobar si el Player Colisiona con un "Wall"****Si es el caso:****Reducir la velocidad del Player en la dirección del "Wall"**

Figura C.3: Pseudocódigo que resume el sistema de colisiones

Para saber si un KinematicObject va a colisionar con el suelo o un muro (se identifican los objetos con los que se desea chocar porque tienen asignada la layer “Wall”) se coge la velocidad del KinematicObject (se obtiene del atributo velocity del Rigidbody2D), que está representada en unidades/segundo. Con la velocidad que lleva el KinematicObject y su posición se puede deducir la siguiente posición en la que se encontrará.

Hay un método que se llama Physics2D.BoxCast con el que puedes crear un rectángulo en una región del espacio y comprobar si se colisiona con algún objeto. En caso de colisionar con un objeto se devuelve un objeto de la clase RayCastHit2D con toda la información relativa a la colisión. Hay un método similar a Physics2D.BoxCast que es Physics2D.BoxCastAll que hace lo mismo, pero devolviendo un vector de RayCastHit2D con un elemento por cada objeto con el que has colisionado.

Se puede filtrar las colisiones por Layer, pudiendo solo tener en cuenta las colisiones con objetos que tengan asociada una Layer con el mismo nombre que el pasado por parámetro en el método. Para el sistema de colisiones solo se han tenido en cuenta los objetos con Layer igual a “Wall”.

Con el método Physics2D.BoxCastAll se va a crear un rectángulo del tamaño del Collider2D del KinematicObject en la siguiente posición en la que se encontrará el objeto y comprobarán cuántos objetos con Layer igual a “Wall” colisionarán en esa posición con el KinematicObject.

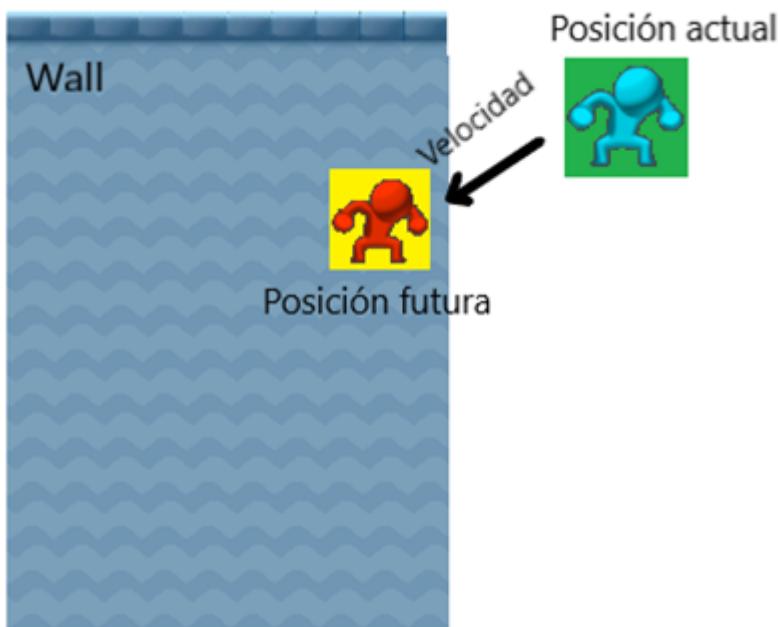


Figura C.4: Simulación del proceso de detección de colisiones

Una vez detectados con que muros se han colisionado (objetos con Layer igual a “Wall”) se va a simular el choque modificando la velocidad del KinematicObject poniendo a cero la velocidad en la dirección de la colisión del muro. Un ejemplo de aplicación sería un KinematicObject yendo a una velocidad marcada por el vector $(1, -2)$, es decir 1 unidad hacia la derecha (eje x) y dos unidades hacia abajo (eje y). Si se detecta que se va a colisionar contra el suelo (un muro que está a los pies del KinematicObject) la velocidad debería establecerse al vector $(1, 0)$, es decir continuar el desplazamiento a la derecha pero cesar el movimiento hacia abajo.

Para calcular en qué dirección hay que limitar la velocidad se utiliza el vector normal de la recta creada por la pared más cercana del muro. Ese vector normal lo ofrece el objeto RayCastHit2D en su atributo “normal”. Se va a añadir una figura explicativa:

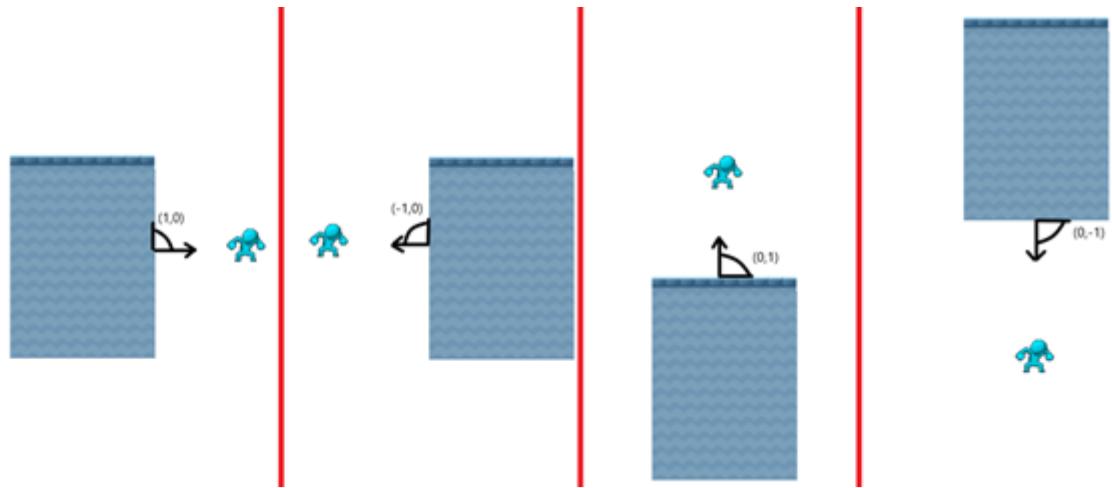


Figura C.5: Vector normal del muro en función de la posición del KinematicObject

De la colisión del KinematicObject se encarga el objeto KinematicObjectCollisionManager, que tiene una referencia a KinematicObject y se encarga de llamar al método Physics2D.BoxCastAll y limitar la velocidad del KinematicObject en caso de colisión.



Figura C.6: Diagrama UML del sistema de colisiones

Modificaciones gravitatorias

Durante la ejecución del juego puede ser que los objetos kinemáticos sufran modificaciones que varíen las fuerzas gravitatorias que los afectan. Estas modificaciones las provocan los modificadores de gravedad y no afectan a todos los elementos del juego sino a un objeto kinemático en concreto.

Dada esta premisa, se ha creado una clase KinematicObjectGravityManager encargada de la gestión de las modificaciones de gravedad que afectan al KinematicObject.

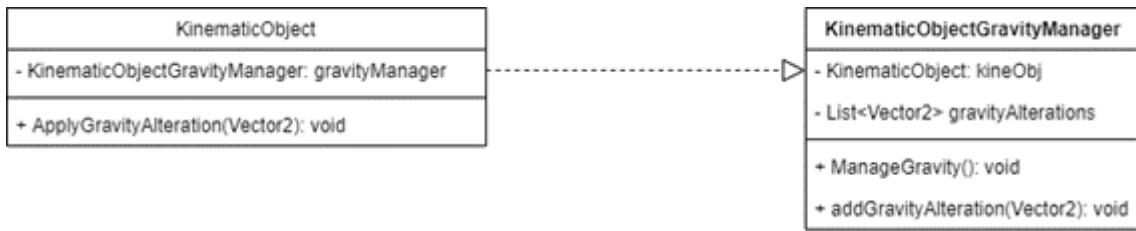


Figura C.7: Diagrama UML de la clase encargada de la gestión de la gravedad del KinematicObject

KinematicObjectGravityManager va a funcionar mediante una lista a la que se van a ir añadiendo objetos de clase Vector2 que representarán las alteraciones de la gravedad. En cada llamada al método FixedUpdate de KinematicObject se recorre la toda la lista acumulando las alteraciones gravitatorias y se aplican esas alteraciones al efecto gravitatorio por defecto (Physics2D.gravity) y se simula la gravedad. Después de este proceso se vacía la lista.

Paso 1:

Modificador_gravedad = (0,0)

Por cada alteración_gravitatoria **en** alteraciones_gravitatorias
Modificador_gravedad += alteración_gravitatoria

Paso 2:

gravedad = **Physic2D.gravity** + Modificador_gravedad

Aplicar_gravedad(gravedad)

Paso 3:

alteraciones_gravitatorias = **new List<Vector2>()**

Figura C.8: Pseudocódigo del proceso de modificación de la gravedad del KinematicObject

Hay dos tipos de modificadores de gravedad: los obstáculos superdensos y los inversores de gravedad. La diferencia fundamental entre estos modificadores

res de gravedad es que los obstáculos superdensos aplican una modificación temporal mientras que los inversores de gravedad modifican la gravedad de forma permanente (o hasta que se cancele el efecto).

TimeAffectedObjects

Hay una serie de objetos que se ven afectados por las modificaciones temporales particulares. Estos objetos heredarán de la clase TimeAffectedObject.

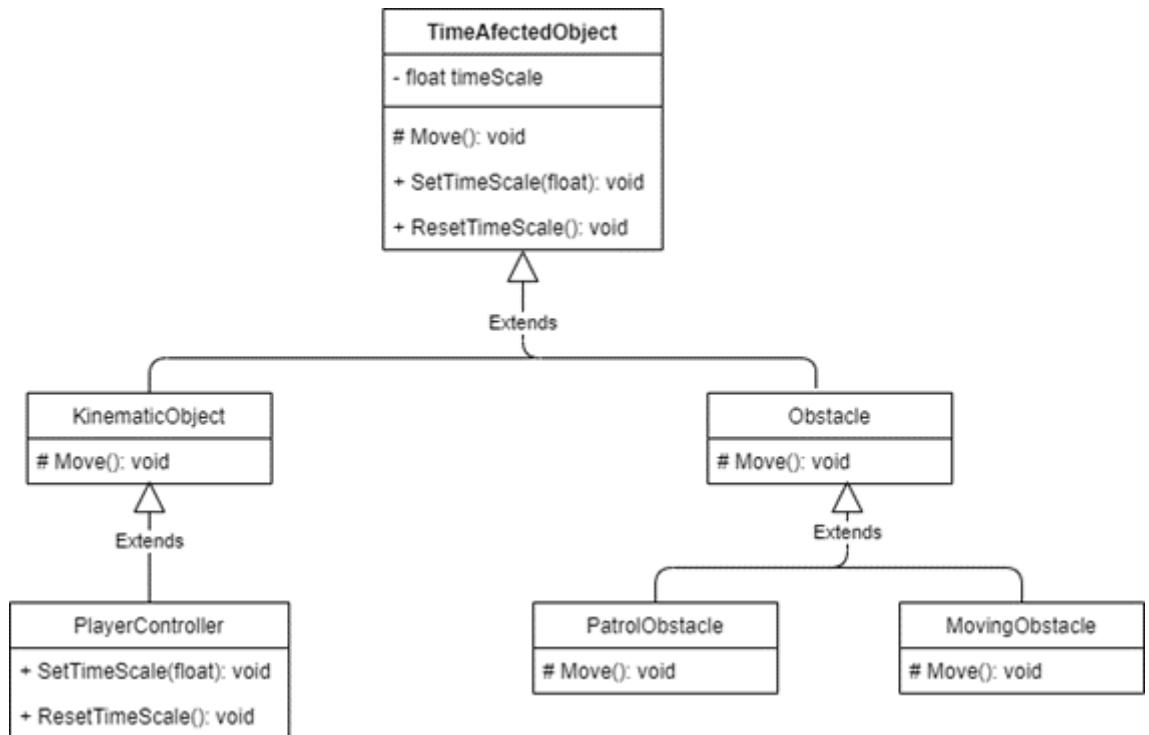


Figura C.9: Clases que heredan de TimeAffectedObject

El movimiento de las clases que hereden de esta deberá implementarse en el método `Move`, pues será en ese método en el que se aplicará la modificación temporal al movimiento acelerándolo o ralentizándolo según toque.

Cuando el juego vuelva al estado inicial será necesario restablecer la escala temporal de todos los `TimeAffectedObjects` a la escala por defecto.

ResetTimeAffectedObjects():

Para cada TimeAffectedObject en TimeAffectedObjects

Desescalar el tiempo de TimeAffectedObject

TimeAffectedObjects = new List<TimeAffectedObjects>()

Figura C.10: Pseudocódigo del método llamado para resetear los TimeAffectedObject

C.5. Diseño arquitectónico

GameController

Durante la ejecución de las escenas hay elementos que se requiere que estén en una posición inicial y que se pueda volver a ella cuando se requiera (concretamente cuando el Player muera y se tenga que volver al estado inicial de la escena).

Los objetos que gestiona el GameController puede ser cualquier GameObject, pero no todos necesitarán ser reiniciados, pues tendrán un estado estable durante toda la ejecución de la escena. Por ejemplo, los obstáculos móviles se desplazan horizontal y pueden incluso ser destruidos, pero cuando el Player muera el obstáculo móvil tiene que volver a la posición inicial en la que se encontraba al cargar la escena. De esta labor se encargará el GameController. Sin embargo, el obstáculo inmóvil se mantiene siempre estático en la misma posición y no puede ser destruido, con lo que el GameController no tiene que preocuparse por devolverlo a un estado estable, pues es el único estado en el que puede estar.

GameController está implementado de manera que simule el patrón de diseño Singleton, en la medida en la que Unity lo permite, llamándose en el método Awake al método GetInstance para asegurarse de que siempre se trabaja siempre sobre el mismo GameController, a pesar de que haya varias instancias de él.

Funcionamiento del estado estable

GameObject tiene tres atributos lista: initialObjects, initialStartingObjects e instancedObjects. De estas tres listas la única pública es initialObjects. La intención de esta lista es que se añada en el editor de Unity todos los



Figura C.11: Diseño de la clase GameController

elementos cuyo estado se desea que sean devueltos a un estado estable. Cuando se llama al método Awake el contenido de initialObjects se vuelca en initialStartingObjects. InitialStartingObjects no es una lista de GameObject sino una lista de StartingObject. La clase StartingObject es una clase que contiene dos atributos: el GameObject que se desea poder devolver a un estado estable y un atributo Transform con la información necesaria para devolver el GameObject a la posición inicial. Por último la lista instancedObject contiene los objetos que hay que se han devuelto al estado estable.

Estas tres listas pueden parecer redundantes, pero no lo son. Es cierto que entre intialObjects e initialStartingObjects no hay mucha diferencia, pero al ser pública la lista initialObjects se ha preferido crear una lista con un nivel de encapsulamiento privado y que haya tratado los elementos de initialObjects para adaptarse a la aplicación del estado estable. Los elementos de StartingObjects en realidad van a hacer la labor de prototipo. Esto se refiere a que los elementos de initialStartingObjects no van a aparecer en la escena, sino que van a ser utilizados para clonar GameObjects en el estado inicial deseado para que parezca que todos los elementos vuelven a su estado inicial, cuando en realidad están siendo destruidos e instanciados objetos iguales a los eliminados (esto es lo que hace el método SetStartingState).

Los elementos que tiene la lista instancedObjects son todos los GameObject que se van a destruir cuando se llame al método SetStartingState. InstancedObjects por supuesto tendrá todos los objetos clonados de initialStartingObjects, pero también contiene GameObjects que pueden ser creados en tiempo de ejecución, pero que al volver el juego a un estado inicial tienen que ser destruidos (como por ejemplo los obstáculos móviles que instancian las fábricas de obstáculos).

SetStartingState(float instantiateDelay):

Destruir todos los elementos de instancedObjects

Cancelar la gravedad invertida de todos los KinematicObjects

Cancelar el escalado de los TimeAffectedObjects

EscaladoGlobal = EscaladoGlobalPorDefecto

Esperar instantiateDelay segundos

Posición del Player = Posición del SpawnPoint

Crear clones de los elementos de initialStartingObjects

Figura C.12: Pseudocódigo que refleja todas las operaciones que hay que realizar para establecer un estado de juego inicial estable

Clase PlatformerModel

La clase PlatformerModel es una clase estática con atributos estáticos y sin métodos. La intención de esta clase es ofrecer acceso a una serie de objetos que pueden ser necesitados por cualquier clase en la escena. La única clase que debería asignar valores a PlatformerModel es GameObject, que lo hará exclusivamente en la llamada al método Awake y realizará la asignación una sola vez y no modificará en ningún momento más durante la ejecución los valores de los atributos de PlatformerModel. Los objetos que se podrán consultar mediante PlatformerModel son:

- La CinemachineVirtualCamera que utiliza la escena (PlatformerModel.virtualCamera).
- El PlayerController asignado al avatar jugable (PlatformerModel.player).
- El objeto de tipo Transform asociado al punto de reaparición e inicio de escena del Player (PlatformerModel.spawnPoint).

- El GameController de la escena (PlatformerModel.gameController).
- El GravityInverterManager asociado a la escena (PlatformerModel.gravityInverterManager).

Esta clase es muy útil para asegurar que todas las clases trabajan sobre los mismos objetos, sobre todo los eventos, pues puede resultar inadecuado hacer una cadena de mensajes para que los eventos tengan referencias a todos los objetos que puedan necesitar para realizar las operaciones que tienen que llevar a cabo.

Clase Simulation

La clase Simulation está explicada más detalladamente en la sección de Trabajos relacionados. Pero a grandes rasgos la clase Simulation es una clase estática con una estructura de datos similar a una cola que alberga eventos y un método Tick. Ese método lo que hace es ir avanzando eventos en la cola y ejecutándolos hasta que la cola esté vacía o hasta que se encuentre un evento que debe ser ejecutado en un momento de la ejecución posterior al momento en el que se encuentra el programa.

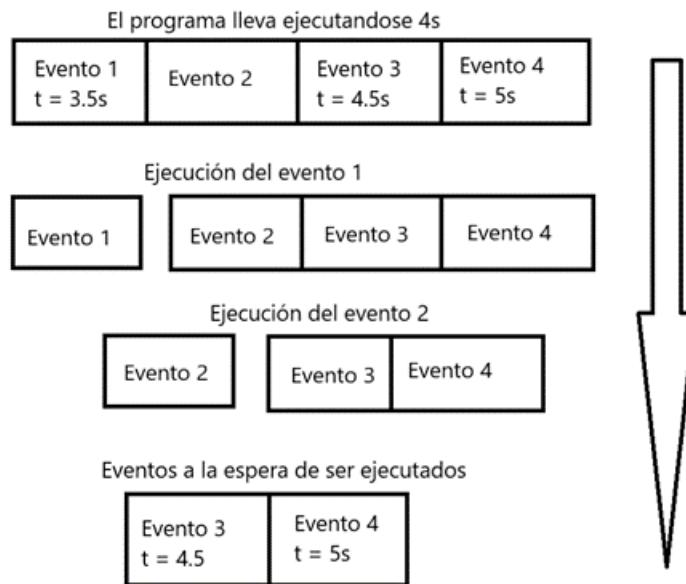


Figura C.13: Diagrama que representa un ejemplo de las operaciones llevadas a cabo en la llamada al método Tick

La clase GameController llama al método Tick de Simulation en cada llamada al Update.

Estados del Player

Los estados que utiliza PlayerController son una aplicación del patrón de diseño Estado que modificará el comportamiento del Player. Se ha tomado esta decisión porque las operaciones a realizar por PlayerController variarán en función de en qué situación se encuentre. Las situaciones en las que se encontrará el Player pueden cambiar en tiempo de ejecución y obligarán a realizar operaciones distintas.

A continuación se mostrará un diagrama con los posibles estados y las condiciones permiten pasar de uno a otro:

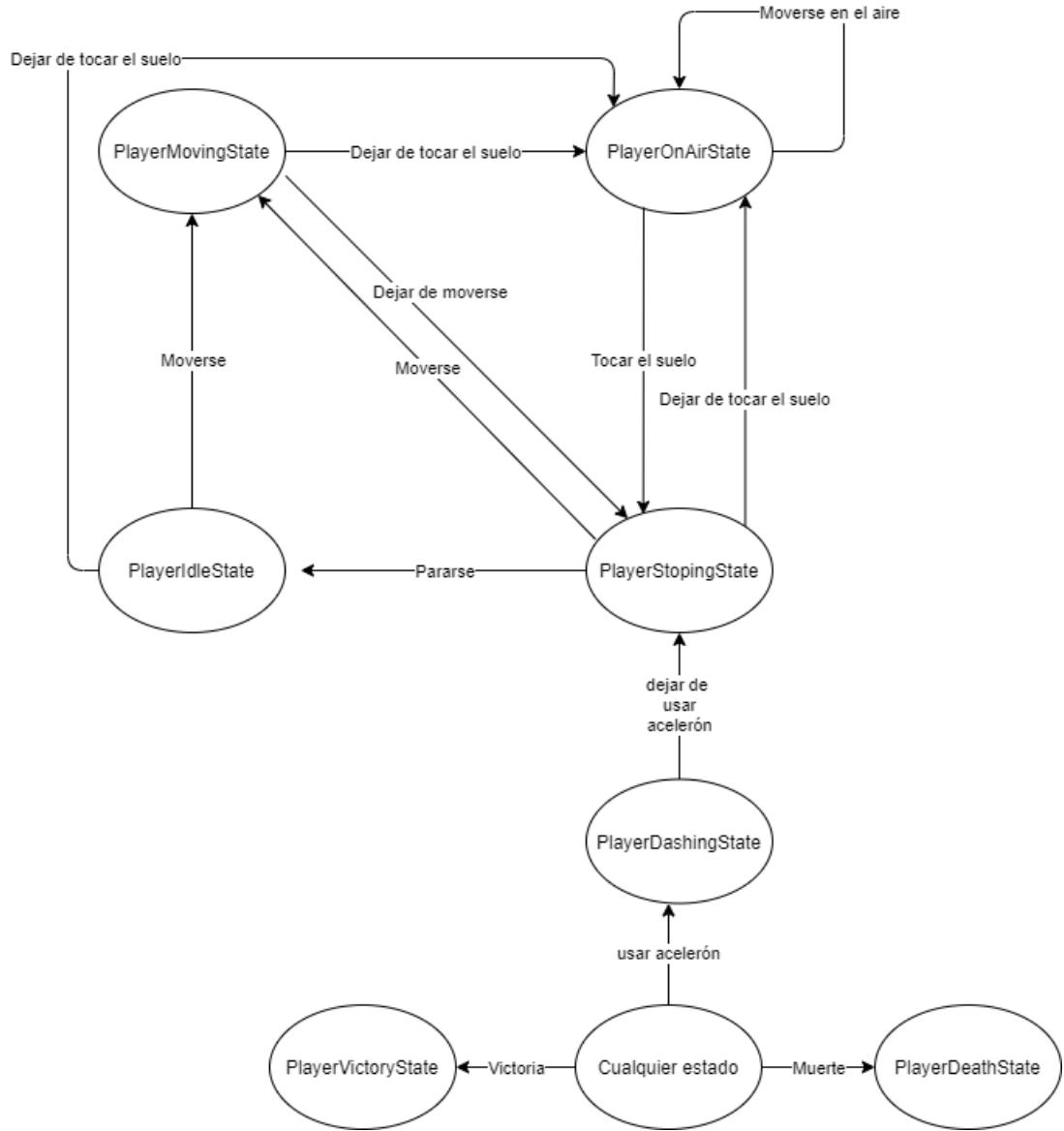


Figura C.14: Diagrama de transición entre estados de PlayerController

Todos los estados heredan de la interfaz PlayerState, que implementa los métodos UpdateState y FixedUpdateState. UpdateState será llamado en el método Update de PlayerController y FixedUpdateState será llamado por el método FixedUpdate de PlayerController. Los estados realizan las siguientes operaciones:

- PlayerIdleState en realidad no realiza ninguna acción pues cuando el avatar está quieto no realiza ninguna acción, pero se encarga de pasar a otros estados cuando toque. De PlayerIdleState se puede pasar a PlayerMovingState si se pulsan los botones de movimiento. También se puede pasar a PlayerOnAirState si el Player no está en contacto con el suelo. De primeras puede parecer una transición innecesaria, pues estando quieto es improbable que se pase de estar tocando el suelo a no estar tocándolo. Esta transición principalmente se ha añadido porque PlayerIdleState es el estado por defecto. Se va a instanciar PlayerController con el estado PlayerIdleState y cada vez que se modifique el estado de PlayerController de manera externa a esta clase (por ejemplo cuando se reaparece después de morir) se asignará el estado PlayerIdleState a PlayerController. Añadiendo esa transición se asegura mantener un estado acorde con la situación para todos los casos, se asigne en el momento en el que se asigne el estado PlayerIdleState.
- PlayerOnAirState es el estado que se adoptará siempre que no se esté en contacto con el suelo (excepto cuando se esté realizando el acelerón). Este estado realiza una acción que es controlar el movimiento del Player en el aire, pues es ligeramente distinto al movimiento en el suelo. De este estado solo se puede pasar al estado PlayerStopingState cuando se toque el suelo.
- PlayerMovingState es el estado en el que se está cuando el Player se está moviendo por el suelo. Este estado se encarga de variar la velocidad del Player cuando se está moviendo en función de que botones de movimiento se están pulsando. Se puede pasar a los estados PlayerOnAirState, si al moverse el Player deja de estar en contacto con el suelo (caerse por un precipicio, por ejemplo) o al estado PlayerStopingState si el jugador deja de pulsar los botones de movimiento.
- PlayerStopingState se encarga de, cuando se cesa el movimiento, se realice una deceleración sobre el Player generando un efecto de parada orgánico. De este estado se puede volver al estado de PlayerMovingState si se vuelve a pulsar los botones de movimiento. A PlayerOnAirState se puede pasar si durante la deceleración se deja de tocar el suelo. Cuando se termine de decelerar y el Player se quede quieto se pasa al estado PlayerIdleState.
- PlayerDashingState es un estado un poco particular que no puede ser pisado por ningún otro. PlayerDashingState hace que el jugador valla

a una velocidad constante durante un periodo de tiempo determinado sin verse afectado por las físicas como la gravedad. Aunque el Player no se vea afectado por las físicas durante el acelerón, las colisiones sí que se aplicarán sobre él. Cuando se termina de hacer el acelerón se pasa al estado PlayerStopingState.

- PlayerDeadState y PlayerVictoryState son dos estados cuya principal función es que el Player no realice ninguna función mientras se encuentren en ese estado. PlayerDeadState es un estado que se activa cuando el Player muere y se sale de él al hacer reaparecer al Player en la zona de reaparición, donde el Player pasará al estado PlayerIdleState. PlayerVictoryState es el estado final que alcanza el Player. Cuando se llega a la zona de victoria se pasa al estado PlayerVictoryState y no se sale de él, pues no hace falta. Cuando se llegue a la zona de victoria y se pase al PlayerVictoryState se pasará al siguiente nivel y por tanto no hará falta gestionar más los estados (los Player son independientes en cada nivel).

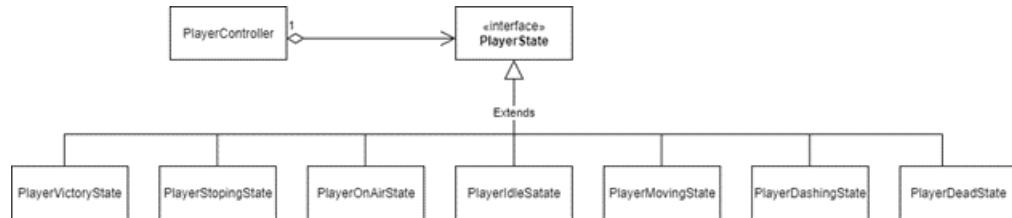


Figura C.15: Diagrama de transición entre estados de PlayerController

Adicionalmente la interfaz PlayerState añade los métodos EnterPlayerState y ExitPlayerState donde los hijos implementan las instrucciones que es necesario hacer para que al entrar y salir del estado el Player se mantenga consistente.

Las instrucciones que se llevan a cabo cada vez que se cambia de estado son las siguientes:

```
public void ChangeState(PlayerState playerState)
{
    if(this.playerState != null)
    {
        this.playerState.ExitPlayerState();
    }
    this.playerState = playerState;
    this.playerState.EnterPlayerState();
}
```

Figura C.16: Diagrama de transición entre estados de PlayerController

Obstáculos que siguen una rutina y la clase PatrolPath

Los obstáculos que siguen una rutina son obstáculos que viajan a través de una serie de puntos a una velocidad constante en un intervalo de tiempo marcado, de manera repetitiva hasta el final de la ejecución de la escena o el reinicio de esta. El punto de inicio y fin de la rutina coinciden.

Realmente el obstáculo que sigue una rutina (PatrolObstacle) no calcula a donde tiene que moverse, sino que delega esta labor a la clase PatrolPath y mueve su posición a donde le indica el PatrolPath.

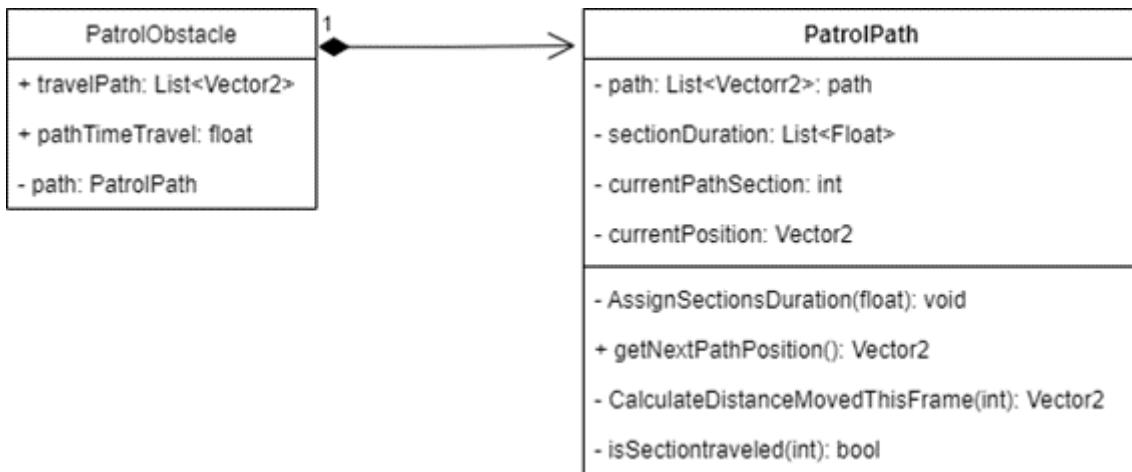


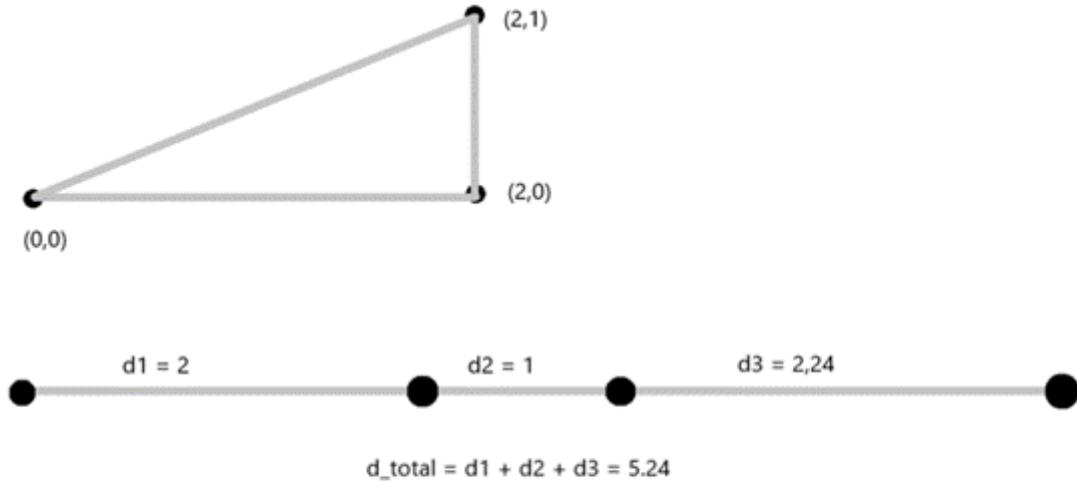
Figura C.17: Diagrama UML de la clase PatrolObstacle

PatrolPath recibe en el constructor la rutina que va a seguir el PatrolObstacle y el tiempo que tardará en recorrerla. PatrolPath calcula el tiempo que le llevará recorrer cada sección. Siendo que se tarda en hacer toda la rutina t segundos, cada sección se tardará en recorrer $\frac{dy * d}{t}$

Siendo dy la distancia euclíadiana de la sección y d la distancia total.

```
AssignSectionsDuration(float pathDuration):
    Para cada sección en path
        distancia = CalcularDistanciaDeLaSección
        distancias[sección] = distancia
    distanciaTotal = CalcularDistanciaTotalDelPath
    Para cada sección en sectionDurations
        duración = (distancias[sección]/distanciaTotal)*pathDuration
        sectionsDurations.Add(duración)
```

Figura C.18: Pseudocódigo de cálculo del tiempo que lleva recorrer cada sección



$$t_1 = (d_1/d_{\text{total}}) * \text{pathDuration}$$

$$t_2 = (d_2/d_{\text{total}}) * \text{pathDuration}$$

$$t_3 = (d_3/d_{\text{total}}) * \text{pathDuration}$$

Figura C.19: Cálculo del tiempo que se tarda en atravesar cada sección

PatrolPath sigue un diseño muy parecido al de un Iterador en el sentido de que el de la posición actual solo se puede obtener la siguiente y nada más. La próxima posición del obstáculo que sigue una rutina (calculada en cada Update) se calcula sumando a la posición actual la distancia que se recorrerá cada frame (cada Time.deltaTime segundos). La distancia que se recorre cada frame se puede obtener mediante la fórmula $\frac{(v_1 - v_2)t}{ty}$

Siendo v_1 el punto del que se parte, v_2 el punto al que se quiere llegar, t la fracción de espacio que se quiere recorrer (Time.deltaTime) en unidades de tiempo normalizado $conty$ (el tiempo que se tarda en recorrer la sección) (en un frame se recorre un t/ty sobre 1 de la distancia a recorrer).

GetNextPathPosition():

```
distanziaRecorrida = CalculateDistanceMovedThisFrame()  
posiciónActual = posiciónActual + distanzaRecorrida  
Si se ha terminado de recorrer la sección actual entonces  
    secciónActual = siguientePosición  
return posiciónActual
```

Figura C.20: Calcular la siguiente posición a la que debe moverse el obstáculo que sigue una rutina

Apéndice D

Documentación técnica de programación

D.1. Introducción

Hay una serie de elementos relativos al proyecto que no tiene que ver estrictamente con el desarrollo, pero que pueden ser útiles conocer (tales como la estructura de ficheros, la importación y exportación del proyecto o la pruebas de sistema). Se explicarán a continuación.

D.2. Estructura de directorios

Se va a comentar la estructura de ficheros del repositorio de GitHub que contiene todos los materiales relativos al proyecto.

- /: Directorio raíz del repositorio.
- /docs: Directorio que contiene todos los ficheros relativos a la documentación.
- /docs/Latex: Carpeta con todos los ficheros necesarios para la generación de la memoria y los anexos con Latex.
- /docs/Latex/img: Contiene todas las imágenes que aparecerán en los ficheros generados en Latex.
- /docs/Latex/tex: Contiene los ficheros de Latex secundarios que utilizarán los principales para generar la memoria y los anexos.

- /game: Contiene la solución de Visual Studio con el código relativo al videojuego desarrollado.
- /game/Assets: Contiene todos los ficheros de código en C# y en formatos especializados de Unity necesarios para que Unity pueda exportar el juego correctamente.
- /game/Assets/Audio: Ficheros de audio del proyecto.
- /game/Assets/Character: Elementos estéticos exclusivos del Player.
- /game/Assets/Character/Animations: Animaciones que provee Platformer Microgame utilizadas en el Player.
- /game/Assets/Character/Sprites: Sprites del Player.
- /game/Assets/Environment: Sprites utilizados para crear los mapas de los niveles jugables.
- /game/Assets/ModAssets: Elementos estéticos extra que provee Platformer Microgame para ofrecer más variedad estética.
- /game/Assets/Prefabs: Contiene todos los prefabs del proyecto.
- /game/Assets/Scenes: Contiene todas las escenas que conforman el proyecto.
- /game/Assets/Scripts: Ficheros de C# utilizados en el proyecto.
- /game/Assets/Scripts/Animation: Ficheros de C# utilizados en el proyecto relativos a la animación de sprites.
- /game/Assets/Scripts/Core: Ficheros de C# utilizados en el proyecto. Contiene las clases encargadas del funcionamiento básico de los niveles.
- /game/Assets/Scripts/Gameplay: Ficheros de C# utilizados en el proyecto. Contiene las clases que son eventos de Simulation.
- /game/Assets/Scripts/GizmosUI: Ficheros de C# utilizados en el proyecto. Contiene las clases utilizadas para mostrar los Gizmos en el editor.
- /game/Assets/Scripts/Mechanics: Ficheros de C# utilizados en el proyecto. Contiene las clases que implementa las mecánicas del juego.

- /game/Assets/Scripts/Model: Ficheros de C# utilizados en el proyecto. Contiene una clase con las variables que se consultarán durante los niveles.
- /game/Assets/Scripts/Sound: Ficheros de C# utilizados en el proyecto. Contiene las clases encargadas de gestionar el audio.
- /game/Assets/Scripts/UI: Ficheros de C# utilizados en el proyecto. Contiene todas las clases relativas a la interfaz de usuario.
- /game/Assets/Scripts/View: Ficheros de C# utilizados en el proyecto. Contiene clases que ofrecen comportamientos que afectan de forma especial a como debería verse un objeto.
- /game/Assets/Sprites: Sprites del proyecto que no pertenecen a Platformer Microgame.
- /game/Assets/Scripts: Elementos usados para generar mapas mediante la herramienta Tilemap de Unity.

D.3. Manual del programador

Ventanas de Unity

Se van a listar a continuación una serie de ventanas del editor de Unity y se explicará su propósito.

Console

Consola interna de Unity que mostrará los print realizados por las clases Monobehaviour y errores y warnings de compilación y ejecución.

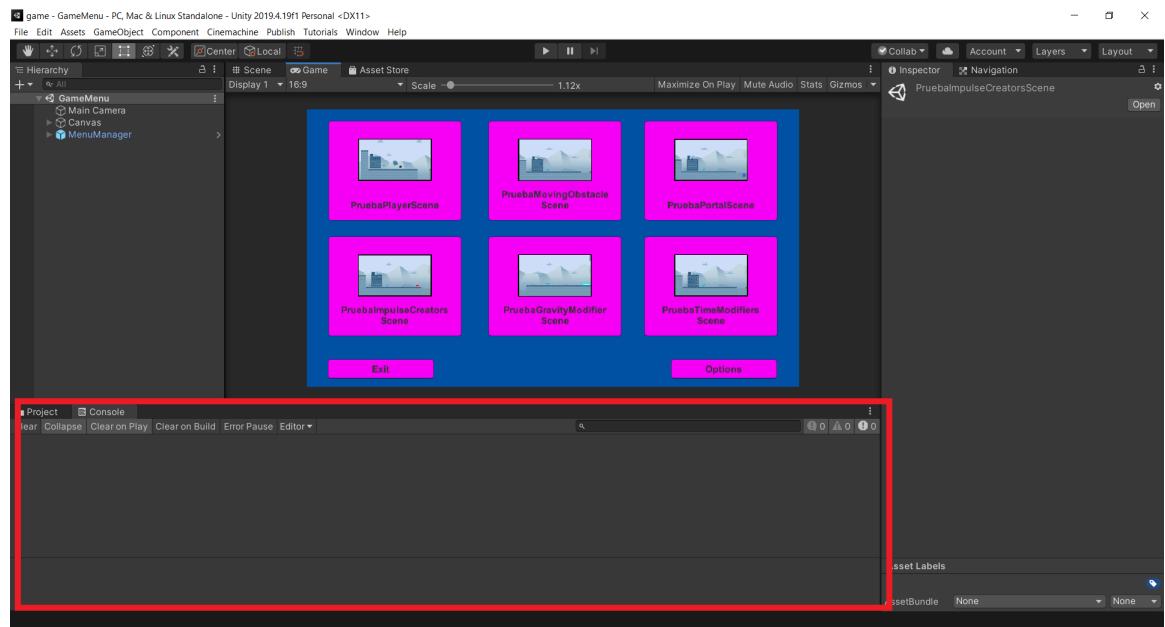


Figura D.1: Consola de Unity

Project

Ventana que muestra todo el sistema de ficheros del proyecto que se está desarrollando.

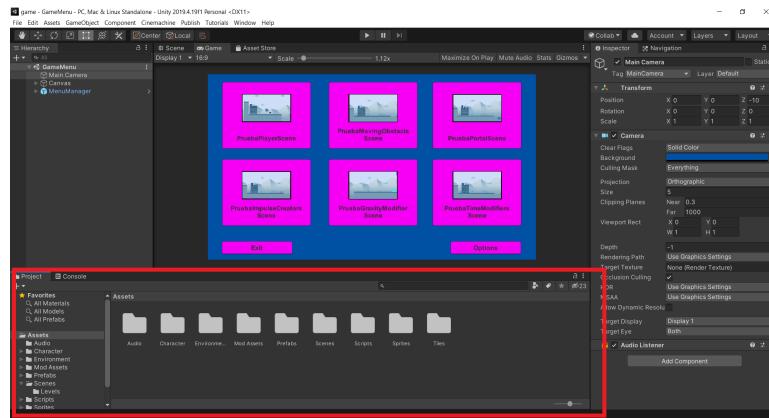


Figura D.2: Ventana "Project" de Unity

Hierarchy

Ventana que muestra los elementos que componen la escena y las dependencias padre-hijo entre estos.

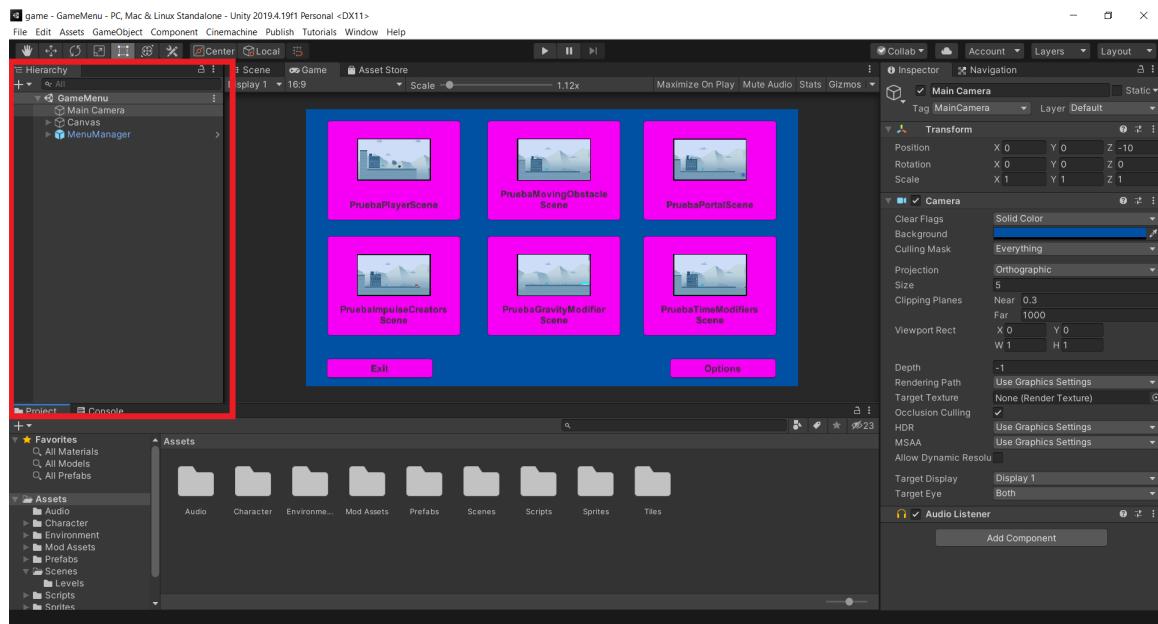


Figura D.3: Ventana "Hierarchy" de Unity

Estos elementos serán distintos para cada escena. Sin embargo en los niveles jugables tendrán una serie de elementos comunes:

- MainCamera: Cámara encargada de mostrar por pantalla lo que el jugador debe ver.
- OptionsMenuCanvas: Estructura de objetos que conformarán el menú de pausa.
- GameController: Objeto encargado de que el juego mantenga un estado estable. Tiene un objeto hijo que es un reproductor de música.
- SpawnPoint: Punto de aparición del Player. También reaparecerá en este punto tras morir.
- Player: Avatar que controlará el jugador.

- CineachineConfiner: La cámara no se podrá desplazar más allá de los límites marcados por este objeto.
- CM vcam1: Objeto encargado del control del movimiento de la cámara.
- Grid: Contiene los Tilemaps que construyen el nivel. Tiene 4 objetos hijos: FarBackgroundTilemap para elementos muy alejados del nivel (montañas), BackgroundTilemap para elementos alejados del nivel (nubes), LevelTilemap para elementos que forman parte del nivel y se espera poder interactuar con ellos (suelos y paredes) y ElementsTilemap para elementos que forman parte del nivel, pero son puramente estéticos (árboles y edificios).
- Zones: zonas relevantes del nivel. Podrán ser de tres tipos: muros o suelos, zonas de muerte y zonas de victoria.

Información del Objeto

Ventana que muestra la información y ajustes de los componentes que conforman un GameObject seleccionado.

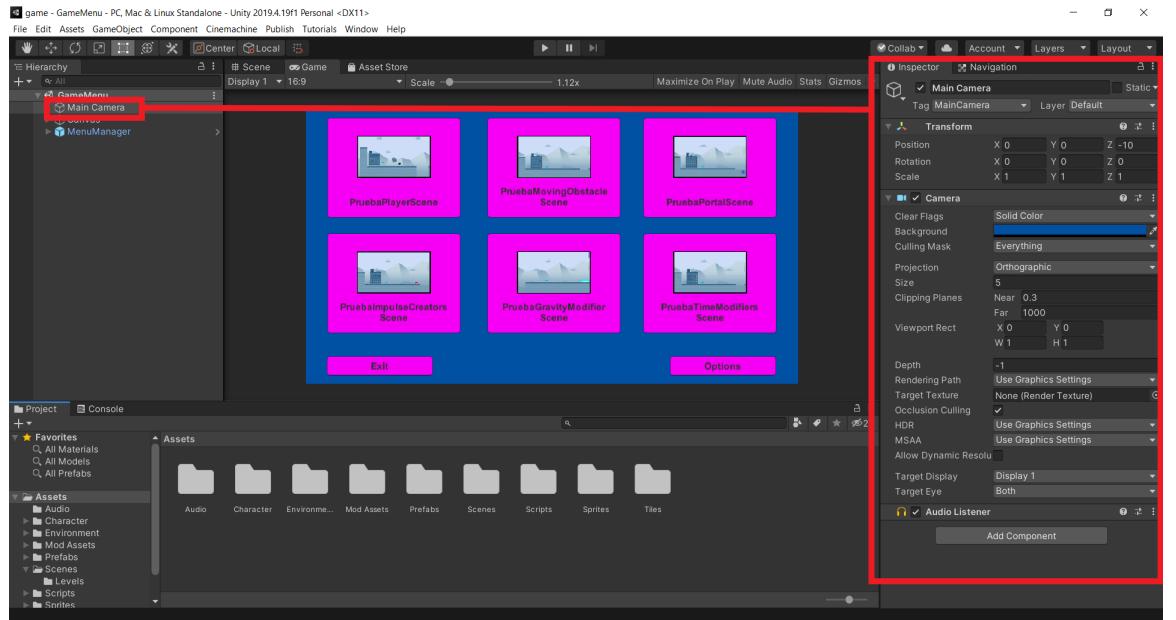


Figura D.4: Información asociada al GameObject Main Camera

Game

Ventana que muestra como se verá el juego en ejecución. Si se ejecuta el proyecto mostrará como se verá el juego en tiempo de ejecución actualizándose en cada frame.

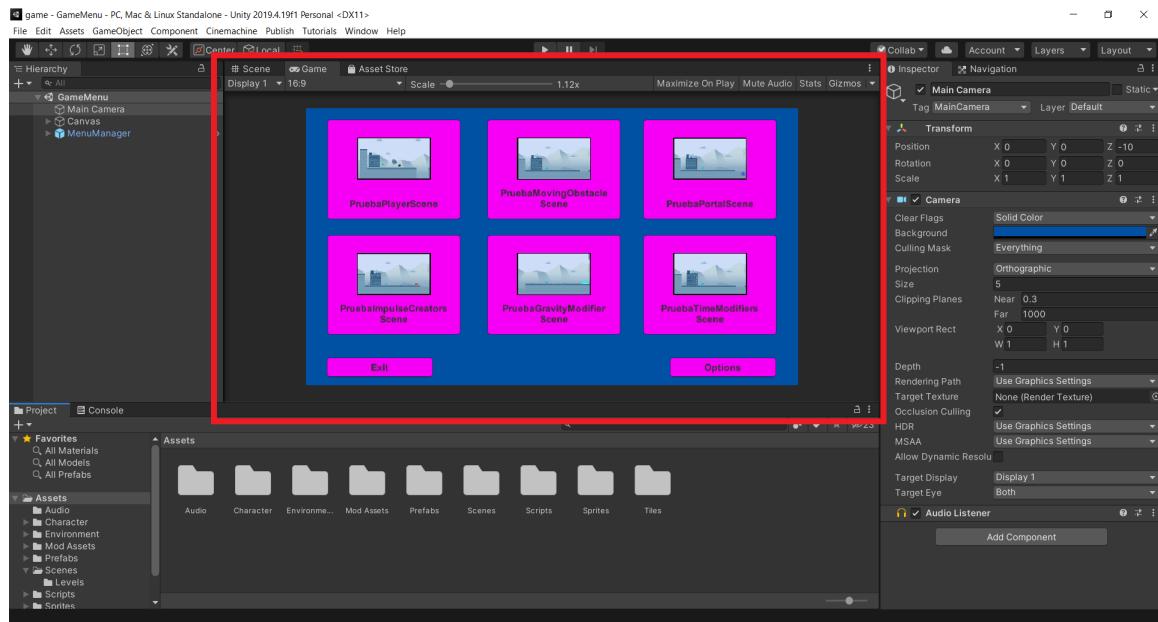


Figura D.5: Ventana "Game" de Unity

Scene

Ventana que muestra como visualmente donde estaría cada objeto de "Hierarchy".^{en} un entorno físico. Como el desarrollo es de un Plataformas 2D se mostrará el entorno físico en dos dimensiones, pero también es posible generar el entorno en 3D (en realidad el entorno generado es 3D, pero con una colocación de cámara que simula ser 2D).

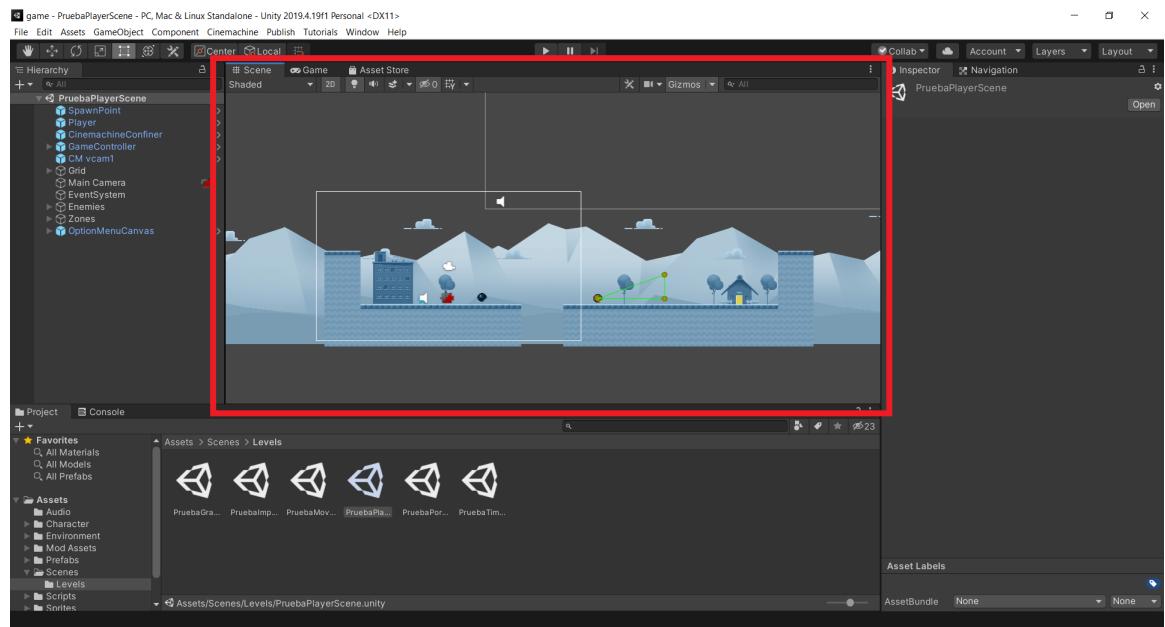


Figura D.6: Ventana "Scene" de Unity

Importación del proyecto

El proyecto de Unity está en la carpeta game del repositorio de Github y será necesario guardado en el ordenador en el que se desee importar para poder importarlo. Para importar el proyecto lo único que hay que hacer es, en el apartado "Projects" del Unity Hub pulsar el botón de ".ADD".

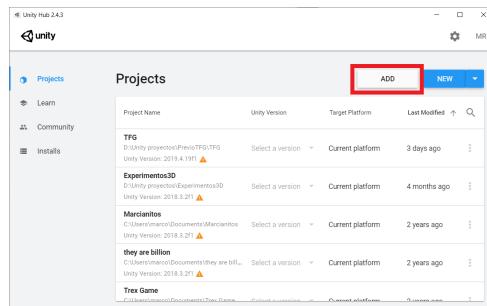


Figura D.7: Importación del proyecto de Unity desde Unity Hub

Después de pulsar el botón se pedirá una ruta. Introducir la ruta donde esté guardada la carpeta game del repositorio de GitHub.

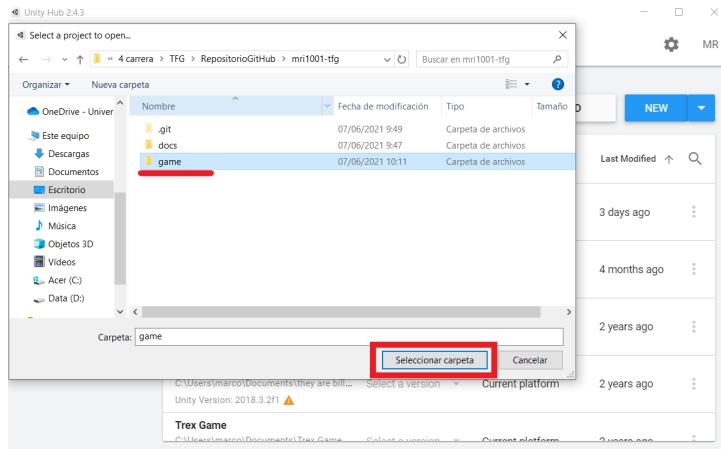


Figura D.8: Selección ruta del proyecto de Unity desde Unity Hub

Se añadirá a tu lista de proyectos de Unity el proyecto importado y ya se podrá acceder a él.

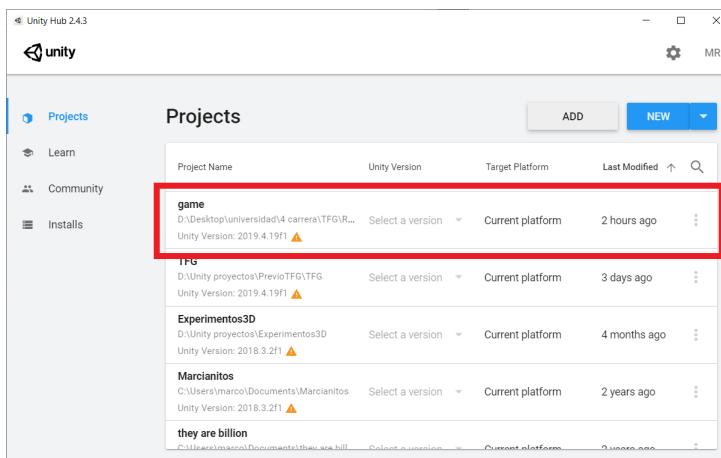


Figura D.9: Proyecto importado correctamente

Al abrir el proyecto por primera vez saldrá una escena por defecto vacía. Con seleccionar cualquier otra escena será suficiente para que desaparezca esa escena y el funcionamiento sea el normal.

Exportación del proyecto

Para exportar el proyecto y generar el videojuego con su correspondiente ejecutable solo es necesario ir a la pestaña de File → Build Settings de la barra de herramientas superior del editor de Unity.

Al hacerlo aparecerá la siguiente ventana:

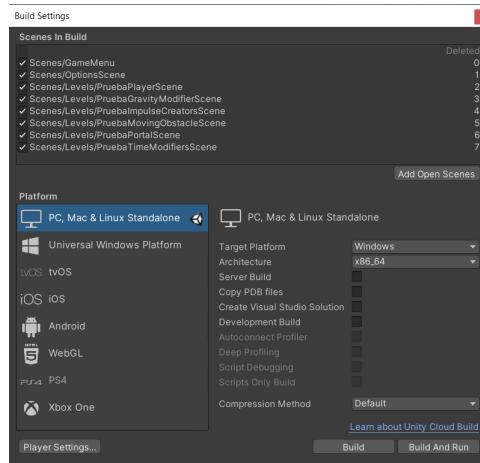


Figura D.10: Ventana de construcción del juego

Esta ventana da la opción de elegir a que sistema operativo y plataforma exportarlo entre otras opciones.

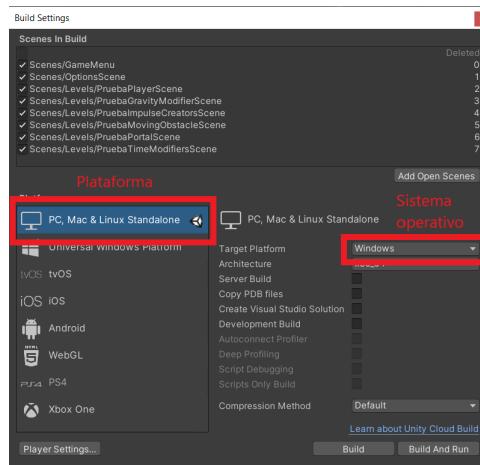


Figura D.11: Selección de plataforma y Sistema operativo

Es muy importante que el tick esté en todas las escenas a las que se desee transicionar para poder viajar a ellas.

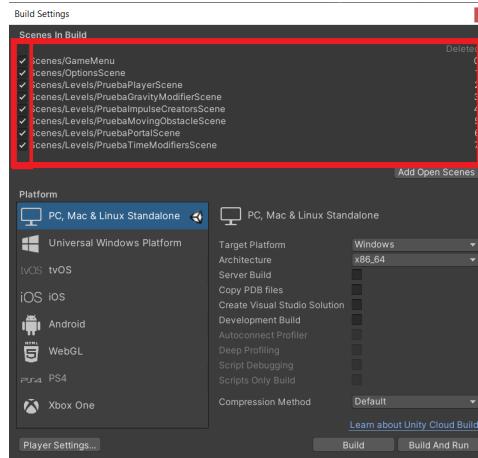


Figura D.12: Escenas seleccionadas para exportar

Una vez hecha esta comprobación se puede dar al botón "Build" para comenzar la exportación del juego. Pedirá una ruta para exportar el juego construido.

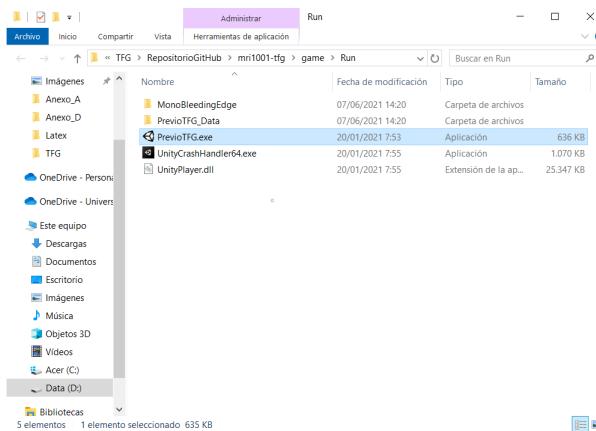


Figura D.13: Ficheros generados al exportar el juego

Entre los ficheros generados un .exe. Ese es el que iniciará la ejecución del juego.

D.4. Compilación, instalación y ejecución del proyecto

Instalación

Para este proyecto solo es necesario tener instalado Unity. Para descargarlo hay que acceder a la siguiente página web: <https://unity3d.com/get-unity/download>

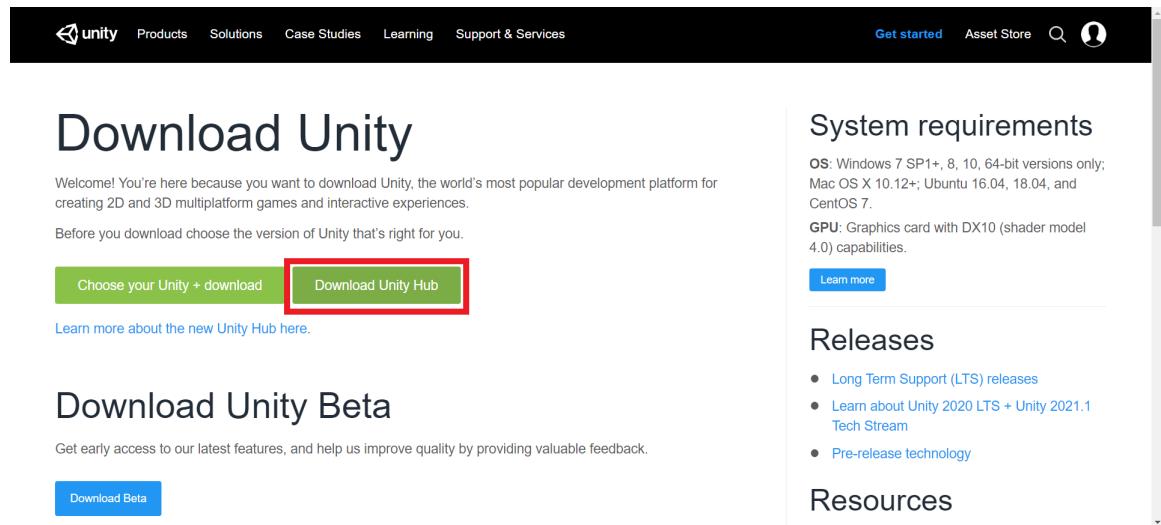


Figura D.14: Página de descarga de Unity

Pulsando el botón de "Download Unity Hub" se te descargará un ejecutable. Al ejecutarlo comenzará la instalación del Unity Hub. Es una instalación normal, con seguir las instrucciones y aceptar los términos de uso el Unity Hub se instalará correctamente.

Ahora solo falta descargar la versión de Unity que se desea descargar. El proyecto se ha desarrollado en la versión de Unity 2019.4.19f1. Es importante instalar esta versión, pues es la única que te dejará abrir el proyecto.

Para instalar La versión de Unity que queramos hay que abrir el Unity Hub y acceder a la pestaña "Install".

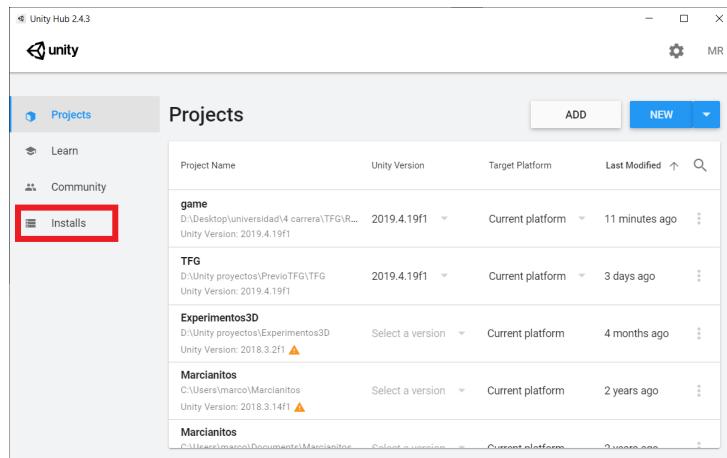


Figura D.15: Unity Hub

Una vez en esa pestaña pulsar el botón de add y elegir la versión de Unity que se desea descargar.

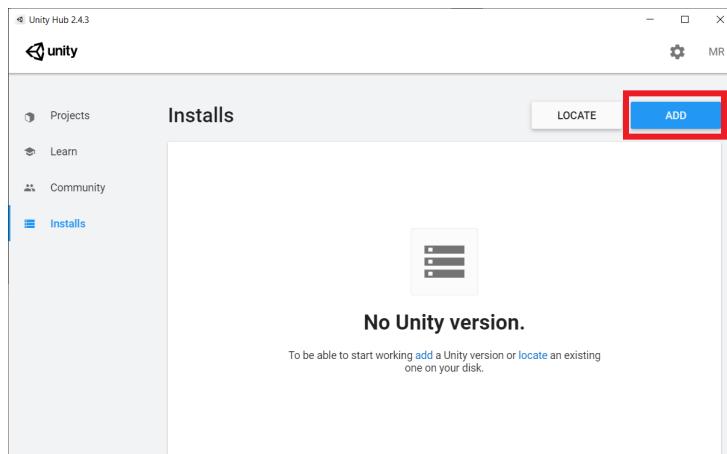


Figura D.16: Descarga de una versión de Unity desde Unity Hub

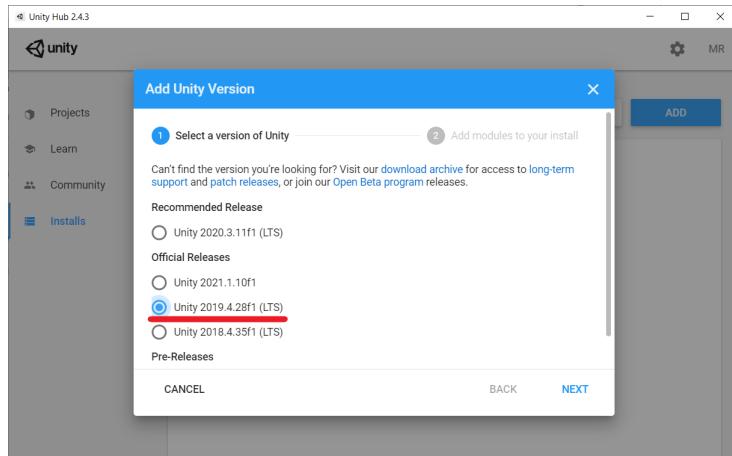


Figura D.17: Selección de una versión de Unity desde Unity Hub

Clicando en "Next." aparecerán los módulos de Unity que se desean añadir a la instalación. Añadir a los que se ofrecen por defecto los módulos "Linux Build Support (IL2CPP)" y "Linux Build Support (Mono)" para que al exportar el juego también sea compatible con Linux.

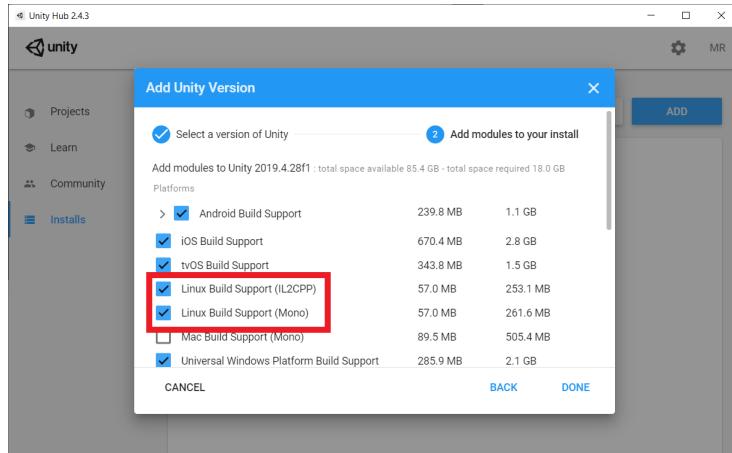


Figura D.18: Selección de módulos de Unity desde Unity Hub

Pulsar en "Next." esperar a que termine la instalación.

La instalación también se puede llevar a cabo después de importar el proyecto, pues no te dejará abrir el proyecto y te obligará a realizar la instalación. El proceso es similar al mostrado anteriormente. Si no se sabe la versión de Unity del proyecto que se desea importar es recomendable utilizar

esta opción y descargar la versión de Unity necesaria después de importar el proyecto.

Compilación

Unity se compila automáticamente. Cada vez que se modifiquen los scripts utilizados en el proyecto y se vuelva a la ventana de la aplicación de Unity se guardarán los cambios y compilará el proyecto. Si surgen errores se mostrarán en la consola que ofrece Unity. Adicionalmente la consola de Unity añade las opciones "Clear" para limpiar el contenido de la consola, "Clear on Play" para limpiar el contenido de la consola cada vez que se ejecute el juego en el editor, "Collapse" para juntar los mensajes de la consola que sean iguales en solo uno, "Clear on Build", para limpiar el contenido de la consola cada vez que se exporte el proyecto, y ".Error pause" para pausar la ejecución del programa cuando salte un error de ejecución.

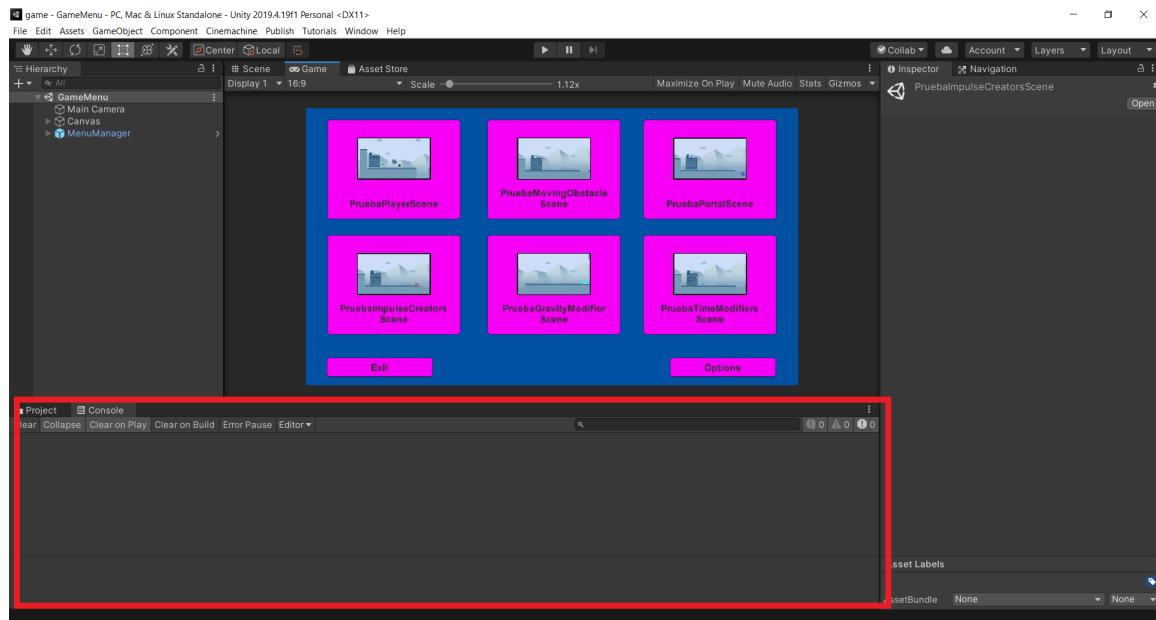


Figura D.19: Consola de Unity

Es posible que la consola no se encuentre en esa posición exacta de la ventana de Unity, pero siempre se podrá acceder a ella a través de la pestaña de Window → Windows → Console de la barra de herramientas superior de Unity.

Ejecución del proyecto

El proyecto se puede ejecutar antes de exportarlo para tener una aproximación a los que será el producto exportado. Para ello se ha de pulsar el botón de "Play" del editor de Unity. La ejecución se parará cuando se vuelva a pulsar el botón de "Play". Tener en cuenta que las instrucciones de cierre del programa no funcionarán en el editor pero si en la exportación.

La ejecución del programa se puede pausar desde el botón de "Pause". Cuando se vuelva a pulsar este botón se reanudará la ejecución.

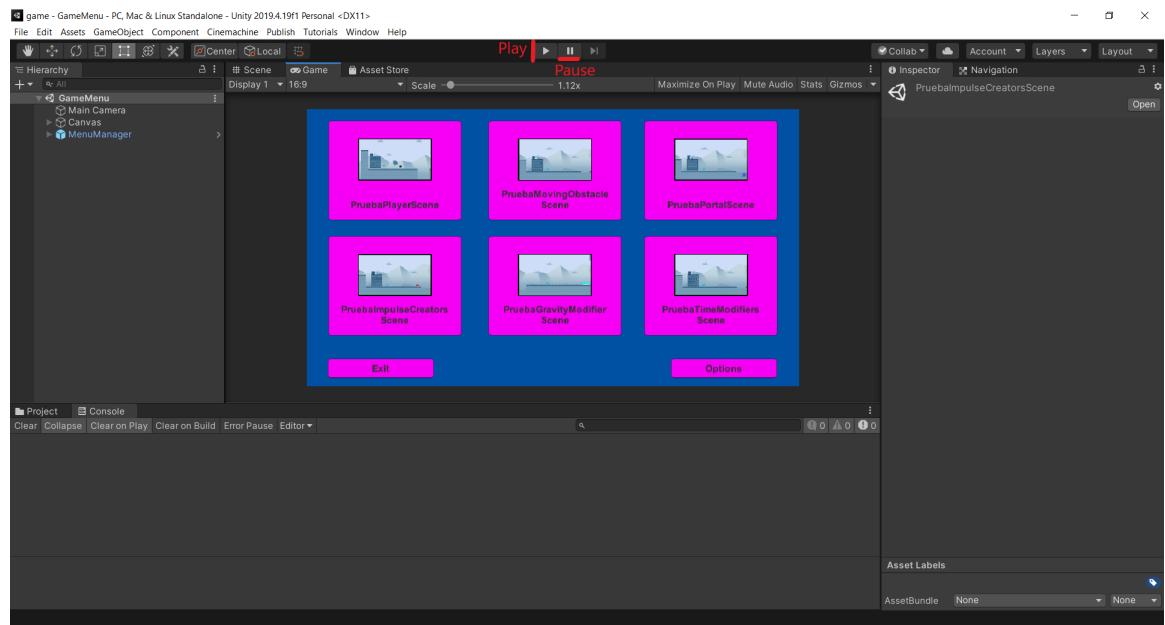


Figura D.20: Botones de "Play" y "Pause" del editor de Unity

D.5. Pruebas del sistema

Como se ha mencionado en la memoria la realización de pruebas durante el desarrollo de un videojuego es un tema complejo. Debido a ello no ha sido posible realizar pruebas automáticas. Sin embargo, se ha querido representar el funcionamiento básico de videojuego (deducido a partir de los casos de uso y el documento de diseño de juego), que si bien no sirven como sistema de pruebas, formalizan el funcionamiento esperado del programa, permitiendo comprobar si se está realizando una correcta implementación del programa o

allanando en el terreno por si en el futuro fuese posible implementar pruebas automáticas.

Para este propósito se ha seguido el sistema de representación de test Given-When-Then [2]. Este sistema de representación se sirve de tres premisas para estructurar los test:

- **Given:** Para describir el estado del programa antes de realizar la acción que se desea probar.
- **When:** Representará la acción que se desea probar.
- **Then:** Para describir los cambios esperados tras la realización de la prueba a realizada.

Se mostrarán a continuación las representaciones de los test desarrolladas:

Selección de niveles

Given: Se está en el menú principal.

When: Se selecciona el nivel al que se desea viajar (PruebaPlayerScene) y se pulsa el botón de acceso.

Then: Se ha viajado a la escena del nivel seleccionado (PruebaPlayerScene)

Viaje al menú de opciones

Given: Se está en el menú principal.

When: Se selecciona el botón de Options y se pulsa.

Then: Se ha viajado al menú de opciones (OptionsScene)

Modificar una opción

Given: Se está en el menú principal sobre la opción de Volumen general, que tiene el valor de 100.

When: Mover la barra de desplazamiento lo máximo hacia la izquierda.

Then: El texto que marca el valor del volumen general habrá cambiado a 0. El volumen del juego será 0. Si se para la aplicación y se vuelve a correr el volumen general seguirá siendo 0.

Abrir el menú de pausa

Given: Estar en un nivel jugable (PruebaPlayerScene).

When: Pulsar el botón asociado a la apertura del menú de pausa.

Then: Se ha abierto el menú de pausa y la ejecución del nivel se habrá detenido temporalmente (Time.timeScale será 0).

Cerrar el menú de pausa

Given: Estar en un nivel jugable (PruebaPlayerScene) con el menú de opciones abierto.

When: Se pulsará el botón del controlador que se esté usando asociado al cierre del menú de pausa o se pulsará el botón del menú de pausa asociado con el cierre del menú de pausa.

Then: Se ha cerrado el menú de pausa y la ejecución del nivel se habrá retomado (Time.timeScale será 1).

Vuelta al menú principal

Given: Estar en un nivel jugable (PruebaPlayerScene) con el menú de opciones abierto.

When: Pulsar el botón del menú de opciones "Back to Main Menú".

Then: Se habrá pasado al menú principal.

Invertir gravedad del Player

Given: Estar en un nivel jugable con inversores de gravedad (Prueba-GravityModifiersScene). El Player tiene una gravedad de Phisics2d.gravity (sin modificaciones).

When: El Player entra en contacto con un inversor de gravedad.

Then: La gravedad del Player se invierte siendo ahora -Phisics2d.gravity.

Entrar a la zona de influencia de un obstáculo superdenso

Given: Estar en un nivel jugable con un obstáculo superdenso (Prueba-GravityModifiersScene). El Player tiene una gravedad de Phisics2d.gravity (sin modificaciones).

When: El Player entra en la zona de influencia del obstáculo superdenso.

Then: La gravedad del Player se modifica al entrar en la zona de influencia, aumentando en la dirección del obstáculo superdenso hasta que colisiona con este y muere.

Entrar en una zona de modificación temporal

Given: Estar en un nivel jugable con una zona de modificación temporal (PruebaTimeModifiersScene). La escala de tiempo del Player es la por defecto.

When: El Player se desplaza hasta entrar en la zona de modificación temporal.

Then: La escala de tiempo del Player pasa a ser la escala de tiempo por defecto * la modificación a la escala temporal de la zona de modificación temporal.

Salir de una zona de modificación temporal

Given: Estar en un nivel jugable con una zona de modificación temporal (PruebaTimeModifiersScene). El Player se encuentra en una zona de modificación temporal. La escala de tiempo del Player es la por defecto * la modificación a la escala temporal de la zona de modificación temporal.

When: El Player se desplaza hasta salir de la zona de tiempo invertido.

Then: La escala de tiempo del Player pasa a ser la escala que tenía / la modificación a la escala temporal de la zona de modificación temporal (la escala de tiempo por defecto).

Modificar el tiempo a nivel global

Given: Estar en un nivel jugable (PruebaPlayerScene).

When: Pulsar el botón asociado a la mecánica de tiempo bala del Player.

Then: La ejecución del nivel se reducirá (Time.timescale se reducirá) y pasado un tiempo volverá a su estado natural (Time.timescale = 1)

Utilizar una plataforma de salto

Given: Estar en un nivel jugable con una plataforma de salto (PruebaImpulseCreatorsScene).

When: El Player entra en contacto con la plataforma de salto.

Then: El Player sale impulsado hacia arriba. Su velocidad (la RigidBody2D.Velocity del Player) habrá variado en el eje de las Y, pero no en el de las X. RigidBody2D.Velocity.Y del Player será ahora RigidBody2D.Velocity.Y + impulso de la plataforma de salto. RigidBody2D.Velocity.X deberá seguir siendo la misma.

Utilizar una partícula de impulso

Given: Estar en un nivel jugable con una partícula de impulso (PruebaImpulseCreatorsScene).

When: El Player entra en contacto con la partícula de impulso.

Then: El Player sale impulsado en la dirección designada por la partícula de impulso. RigidBody2D.Velocity.Y del Player será ahora RigidBody2D.Velocity.Y + impulso de la partícula de impulso en el eje de las Y. RigidBody2D.Velocity.X deberá ser RigidBody2D.Velocity.X + impulso de la partícula de impulso en el eje de las X.

Utilizar un amplificador de impulso

Given: Estar en un nivel jugable con un amplificador de impulso (PruebaImpulseCreatorsScene).

When: El Player entra en contacto con el amplificador de impulso.

Then: La velocidad del Player se escalará en la cantidad designada por el amplificador de impulso. RigidBody2D.Velocity del Player será ahora RigidBody2D.Velocity * multiplicador del amplificador de impulso.

Colisionar con una pared

Given: Estar en un nivel jugable con paredes (PruebaPlayerScene). El Player lleva una velocidad de 6 en el eje de las X y -3 en el de las Y.

When: El Player colisiona con una pared a su derecha (la dirección en el eje de las X en el que se está desplazando).

Then: El Player debe estar al lado de la pared. La velocidad en la dirección del muro debe cesar pero el resto continuar. La velocidad del Player será ahora 0 en el eje de las X y -3 en el eje de las Y.

Colisionar con el suelo

Given: Estar en un nivel jugable con suelos (PruebaPlayerScene). El Player lleva una velocidad de 6 en el eje de las X y -3 en el de las Y.

When: El Player colisiona con el suelo que hay a sus pies (la dirección en el eje de las Y en el que se está desplazando).

Then: El Player debe estar encima del suelo. La velocidad en la dirección del suelo debe cesar pero el resto continuar. La velocidad del Player será ahora 6 en el eje de las X y 0 en el eje de las Y.

Entrar por un portal

Given: Estar en un nivel jugable con un par de portales enlazados (portal1 y portal2) (PruebaPortalesScene).

When: El Player entra por el portal1.

Then: El Player se habrá teletransportando pasando a ser su nueva posición la del portal2.

Mover al Player a la derecha

Given: Estar en un nivel jugable con un Player (PruebaPlayerScene).

When: El Player pulsa el botón de desplazamiento hacia la derecha.

Then: La velocidad del Player habrá aumentado en el eje de las X positivamente. El Player se habrá desplazado hacia la derecha.

Mover al Player a la izquierda

Given: Estar en un nivel jugable con un Player (PruebaPlayerScene).

When: El Player pulsa el botón de desplazamiento hacia la izquierda.

Then: La velocidad del Player habrá disminuido en el eje de las X. El Player se habrá desplazado hacia la izquierda.

Saltar con el Player

Given: Estar en un nivel jugable con un Player (PruebaPlayerScene).

When: El Player pulsa el botón de salto.

Then: La velocidad del Player habrá aumentado en el eje de las Y. El Player realizará el salto.

Realizar el acelerón con el Player

Given: Estar en un nivel jugable con un Player (PruebaPlayerScene).

When: El Player pulsa el botón del acelerón.

Then: La velocidad del Player pasará a ser constante siendo 0 en el eje de las Y, y +(velocidad del acelerón) en el eje de las X en caso de haber estado yendo hacia la derecha o -(velocidad del acelerón) en el eje de las X en caso de haber estado yendo hacia la izquierda. El Player realizará el salto.

Muerte contra un obstáculo

Given: Estar en un nivel jugable con obstáculos (PruebaPlayerScene).

When: El Player colisiona con el obstáculo.

Then: Se debe iniciar la cadena de eventos de muerte del Player.

Cadena de eventos de la muerte del Player

Given: Estar en un nivel jugable (PruebaPlayerScene).

When: El Player muere.

Then: Se destruyen los objetos instanciados durante la ejecución el nivel y se reinicia el nivel devolviéndolo al estado inicial.

El Player cae en una zona de muerte

Given: Estar en un nivel jugable con zonas de muerte (PruebaPlayerScene).

When: El Player colisiona con la zona de muerte.

Then: Se debe iniciar la cadena de eventos de muerte del Player.

El Player cae en una zona de victoria

Given: Estar en un nivel jugable con zonas de victoria (PruebaPlayerScene).

When: El Player colisiona con la zona de victoria.

Then: El Player realiza la animación de llegada a la zona de victoria y se transiciona al menú principal.

El obstáculo que sigue una rutina se mueve

Given: Estar en un nivel jugable con un obstáculo que sigue una rutina (PruebaPlayerScene).

When: Pasar loops de ejecución del programa.

Then: El obstáculo debe haber pasado por todos los puntos que componen su rutina de viaje y haber vuelto a la posición inicial. Al haber vuelto a la posición inicial se debe continuar el movimiento del obstáculo repitiendo la rutina del viaje hasta que se pare la ejecución.

El obstáculo móvil se mueve

Given: Estar en un nivel jugable con un obstáculo que sigue una rutina (PruebaMovingObstacleScene).

When: Pasar loops de ejecución del programa.

Then: El obstáculo móvil tiene que avanzar de derecha a izquierda indefinidamente. Su posición en el eje de las X tiene que ser siempre menor que la que llevaba anteriormente.

Apéndice E

Documentación de usuario

E.1. Introducción

Puede ser que el usuario tenga dudas sobre el producto entregado o el manejo de este. En este anexo se ofrecerá toda la información que el usuario puede requerir para hacer un uso adecuado del producto entregado.

E.2. Requisitos de usuarios

El único requisito para que el usuario pueda ejecutar el videojuego es tener un ordenador con un sistema operativo Windows o Linux.

La otra preocupación del usuario puede ser que su ordenador no sea capaz de correr el juego debido a que es demasiado exigente. Es cierto que esto puede ocurrir con algunos juegos, pero suelen ser juegos de proporciones titánicas, con muchos eventos ocurriendo simultáneamente y con gráficos mucho más sofisticados que los de este juego.

A pesar de que se tiene la firme creencia firme de que puede ejecutar en prácticamente cualquier ordenador se van a listar las especificaciones de los dos ordenadores sobre los que se ha ejecutado el juego y comprobado su correcto funcionamiento.

Predator PH315-51

- Procesador Intel Core i7-8750H (6 núcleos, 9 MB Caché, 2.2 GHz hasta 4.1 GHz)

- Memoria RAM de 16 GB DDR4 (alcanzando un pico de 141 MB cuando se ejecuta)
- Disco HDD de 1 TB
- Disco SSD de 128 GB
- Tarjeta gráfica Nvidia GeForce GTX 1060
- Sistema operativo Windows 10 Home

Ordenador de ofimática

- Procesador Intel Core i3-6006U (2 núcleos, 3 MB Caché, 2 GHz)
- Memoria RAM de 4 GB DDR4
- Disco SSD de 200 GB
- Tarjeta gráfica Intel HD Graphics 520 (integrada)
- Sistema operativo Windows 10 Home

En ambos ordenadores el videojuego funciona sin ralentizaciones ni caídas de frames. Es cierto, aun así, que el ordenador de ofimática tarda un par de segundos más en cambiar entre escenas más que el otro ordenador (mucho más potente).

Linux

Para sistemas operativos Linux se ha ejecutado el juego en un ordenador con las siguientes especificaciones:

- Procesador Intel(R) Core(TM) i3-5005U CPU (4 núcleos, 3 MB Caché, 2 GHz)
- Memoria 8 GB (usando algo menos de 116 MB cuando se ejecuta)
- Tarjeta gráfica Intel HD Graphics 5500 (integrada)
- Sistema operativo Ubuntu 20.04.1

Requisitos mínimos

Partiendo de las pruebas hechas se puede generar un listado de recursos mínimos que consumirá la ejecución del videojuego:

- 130 MB de disco duro los ficheros que contienen el juego (en la versión de Linux que es la que más ocupa)
- 150 MB de RAM
- El procesador de gama más baja con el que se ha confirmado el correcto funcionamiento ha sido el procesador Intel Core i3-6006U (2 núcleos, 3 MB Caché, 2 GHz)
- La tarjeta gráfica de gama más baja con la que se ha confirmado el correcto funcionamiento ha sido la tarjeta gráfica Intel HD Graphics 5500

E.3. Instalación

El usuario no tendrá que instalar nada para poder iniciar el videojuego. Bastará con que se descargue el contenido del repositorio de GitHub asociado al proyecto (<https://github.com/Kencho/mri1001-tfg>).

Una vez descargado el repositorio, dentro de la carpeta ejecutables, habrá tres ficheros comprimidos (game_Windows.zip, game_Linux.tar.gz y game_WebGL.zip) que contendrán la estructura de ficheros del videojuego y el ejecutable que permitirá iniciarla.

El usuario solo tendrá que descomprimir el fichero asociado a su sistema operativo o el fichero de WebGL si no se quiere estar atado a ningún sistema operativo. Después tendrá que entrar en la carpeta generada y ejecutar el fichero .exe. Esto iniciará la ejecución del videojuego.

Instalación WebGL

Sin embargo, es importante mencionar que para la versión de WebGL es necesario que el buscador con el que lo abres soporte WebGL. Esto se puede comprobar accediendo al siguiente enlace: <https://get.webgl.org/>. Al entrar en esta página hay dos opciones: visualizar un cubo rotando o no visualizarlo. Si se visualiza significa que el navegador soporta WebGL.

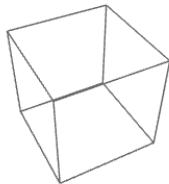
Lo más probable es que cualquier navegador moderno popular que utilicen soporten WebGL. Aun así, se ha comprobado empíricamente que los siguientes navegadores soportan WebGL:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Internet Explore
- Opera

En caso de que no se vea el cubo rotando, lo más probable es que con actualizar los Drivers de la tarjeta gráfica se solucione el problema.

Your browser supports WebGL

You should see a spinning cube. If you do not, please
[visit the support site for your browser.](#)



Check out some of the following links to learn more about WebGL and to find more web applications using WebGL.

[WebGL Wiki](#)

Want more information about WebGL?

[kronos.org/webgl](#)

Figura E.1: Cubo que rota si el navegador soporta WebGL

Aunque el navegador soporte WebGL, es muy probable que al ejecutar el fichero .html que inicia la ejecución en WebGL salte un error.

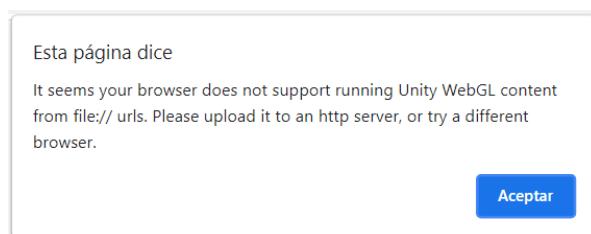


Figura E.2: Error de la versión del ejecutable de WebGL

Este error no es realmente un error, sino una restricción de permisos. Los navegadores suelen tener una opción de seguridad que impide a las páginas acceder a recursos file://. Sin esta restricción se podría acceder a los ficheros del equipo, lo cual es una gran riesgo.

Para poder ejecutar el juego en WebGL sin hacer uso de un servidor web es necesario desactivar esta opción. Esta opción se puede desactivar siguiendo

los pasos de explicados en el siguiente enlace: <https://bit.ly/3iEMWAj>. Esta versión del ejecutable no es recomendable que un usuario que solo quiera disfrutar del videojuego la use. Ha sido añadida como opción para el desarrollo, esperando que se aloje en un servidor web.

E.4. Manual del usuario

Bienvenido, **JUGADOR**, a nuestro nuevo videojuego. Ha habido un cataclismo intergaláctico y los **ciber trabajadores espaciales** se han extraviado de camino a sus oficinas.

¿Serán capaces de volver los **ciber trabajadores** a la oficina para que puedan generar beneficios a la empresa antes de que se vuelvan un lastre y los despidan?

Se trata de un Plataformas 2D dónde podrá controlar a los **ciber trabajadores** y hacer uso de todas las herramientas de modificación del espacio y el tiempo que les provee su generoso jefe para recorrer el **espacio multidi-mensional**.

Pero cuidado, muchos son los peligros que les aguardan y extraño el camino que ahora lleva hasta las oficinas.

Controles

Al iniciar el juego se encontrará con el menú principal. Desde él podrá seleccionar que **trabajador** desea ayudar a llegar a la oficina (como el 30 veces consecutivas empleado del mes, PruebaPlayerScene, o el irreemplazable conserje PruebaMovingObstacleScene).

¡Seleccione a que **trabajador** desea ayudar y tome posesión de su cuerpo para asegurarse de que no le despidan!

Pero igual no esta muy ducho en el arte de poseer **trabajadores**, o incluso de navegar por los abstractos "*Menús del juego*", que por supuesto no existen en otro **plano astral** completamente distinto al de los **trabajadores**.

No pasa nada, **JUGADOR**, vallamos paso a paso.

Controles del menú principal

Para navegar por los menús del juego solo tendrá que tener claro que botón está seleccionado ahora mismo. Eso lo sabrá porque, a diferencia del

resto de botones, de color **rosa fucsia**, el botón seleccionado será de color **rosa flirt**.

Ahora que usted, **JUGADOR**, sabe en qué botón se encuentra solo necesita saber que acciones puede realizar desde él.

- Podrá ir al rescate accediendo al nivel jugable pulsando el botón "X" de su mando (entendemos está usando una mando de Play Station) o la tecla ".ENTER" de su teclado. En cuanto lo haga viajará al **esperpéntico mundo** en el que el **trabajador** espera a ser devuelto a su oficina.
- Si le cae mal ese **trabajador** en particular y desea salvar a otro, podrá desplazarse al botón en el que se encuentra su **trabajador favorito** y acudir en su rescate.

De un botón se podrá viajar a otro de la siguiente manera:

- Si desea desplazarse al botón que se encuentra sobre el botón seleccionado empuje el Stick izquierdo del mando hacia arriba o pulse la flecha que apunta hacia arriba del teclado.
 - Si, en cambio, desea desplazarse al botón que se encuentra bajo el botón seleccionado empuje el Stick izquierdo del mando hacia abajo o pulse la flecha que apunta hacia abajo del teclado.
 - Si el desplazamiento vertical entre botones lo tiene dominado pruebe a desplazarse al botón que se encuentra a la derecha del botón seleccionado. Empuje el Stick izquierdo del mando hacia la derecha o pulse la flecha que apunta hacia la derecha del teclado.
 - Quiza le parezca, **JUGADOR**, una decisión de diseño arriesgada, pero si desea desplazarse al botón que se encuentra a la izquierda del botón seleccionado empuje el Stick izquierdo del mando hacia la izquierda o pulse la flecha que apunta hacia la izquierda del teclado.
- Es posible que la misteriosa música que ha invadido el universo (o todo sonido en general) le resulte molesta. Viaje al menú de opciones y cambie asígnele el valor que deseé al volumen. Puede acceder al menú de opciones seleccionando el botón ".options".
 - Si está cansado de los **trabajadores** se aprovechen de tu buena fe para llegar al trabajo sin dar palo al agua y desea abandonarles a su suerte, no pasa nada. Cierre el juego pulsando el botón de ".exit" tómese unas merecidas vacaciones que se descontarán de su sueldo.

Pero ¡cuidado!, es posible que los **trabajadores** sigan en su sitio a la espera de ser rescatados cuando vuelva a abrir el juego.



Figura E.3: Controles de mando del menú



Figura E.4: Controles de teclado del menú

Menú de pausa

Se encuentra, **JUGADOR**, en el menú de pausa. Desde este menú podrá modificar el volumen general situándose sobre la barra de desplazamiento de "General Volume"(de la misma forma que se hace con los botones del menú principal).

Empujando el Stick izquierdo del mando hacia la derecha o pulsando la flecha que apunta hacia la derecha del teclado se aumentará el volumen general.

Empujando el Stick izquierdo del mando hacia la izquierda o pulsando la flecha que apunta hacia la izquierda del teclado se reducirá el volumen general.

Si solo se desea cambiar el volumen de la música deberá hacer lo mismo que para el volumen general pero seleccionando la barra de desplazamiento de "Music Volume." en vez de la de "General Volume".

Controles de nivel jugable

Ahora que domina los menús seleccione el **trabajador** al que desea salvar. Se encuentra ahora, **JUGADOR**, en un **mundo desconocido**, para colmo, manejando un cuerpo que no es el suyo. Ya que puede ser la primera vez que se encuentra en una situación tan comprometida como esta. No se preocupe, se le dará una ligera explicación de como maniobrar un cuerpo ajeno.

- Para moverse a la izquierda empuje el Stick izquierdo del mando hacia la izquierda o pulse la flecha que apunta hacia la izquierda del teclado.
- Para moverse a la derecha empuje el Stick izquierdo del mando hacia la derecha o pulse la flecha que apunta hacia la derecha del teclado.
- El **trabajador** tiene la capacidad de saltar. Ordénele saltar pulsando el botón "X" del mando o la tecla ".Espacio" del teclado.

Por sobrecogedora que resulte la capacidad de saltar del **trabajador** no es su as bajo la manga. Cuenta con dos mecánicas más:

- La capacidad de realizar un acelerón en la dirección en la que está mirando. Podrá realizar esta acción pulsando el botón ".O" del mando o la tecla "S" del teclado.
- Podrás forzar a tu **trabajador**, en contra de su voluntad, a reducir la velocidad a la que pasa el tiempo pulsando el botón R1 del mando o la tecla "D" del teclado. Es recomendable que recuerde que el jugar con el tiempo tiene sus límites y cabo de unos pocos segundos la velocidad a la que pasa el tiempo volverá a su curso natural.

Es entendible que poseer un cuerpo que no le pertenece puede resultar agotador. Pulse el botón R2 del mando o la tecla ".ESC" del teclado para abrir el menú de pausa. Mientras el menú de pausa este activo, el tiempo se detendrá ofreciéndole un descanso hasta que cierre el menú de pausa (con los mismos controles). Puede ir a prepararse una leche con Nesquik con la tranquilidad de que nada atentará contra la vida del **trabajador**.

Desde el menú de pausa podrá modificar el volumen de las misma forma que desde el menú de opciones.



Figura E.5: Controles de mando de los niveles jugables



Figura E.6: Controles de teclado de los niveles jugables

Apéndice F

Documento de diseño del juego

F.1. Introducción

Plataforma:

Ordenador

Versión:

1.0

Jugabilidad y contenido:

El juego consistirá en una serie de niveles independientes que el jugador tendrá que atravesar hasta llegar a la zona de victoria del nivel. Los niveles tendrán una serie de elementos y mecánicas con los que el jugador habrá de interactuar para llegar a la zona de victoria, evitando a distintos obstáculos que pueden truncar la labor del jugador de llegar a la meta.

En el juego habrá un menú principal que permitirá acceder a todos los niveles jugables. Al alcanzar la meta del nivel se volverá al menú principal.

Debido a que la historia no es competencia del TFG, el juego no tendrá historia.

Categoría:

Plataformas 2D

Mecánica:

El jugador manejará un avatar virtual en una serie de niveles de plataformas en los que tratará de llegar a la zona de victoria del nivel utilizando distintas mecánicas de “viajes en el tiempo” y manipulación gravitatoria.

El jugador podrá realizar las siguientes acciones:

- Moverse.
- Saltar.
- Realizar un acelerón en una dirección.

Las distintas mecánicas con las que podrá interactuar el jugador son:

- Un portal que teletransportará al jugador de un punto a otro del nivel.
- Un creador de impulso que empujará al jugador en una dirección determinada. Esta mecánica se puede manifestar de distintas formas, como una plataforma de salto o una elemento del nivel que escala la velocidad que lleva el jugador.
- Tiempo bala, que ralentizará o acelerará el tiempo de uno o más elementos del nivel, haciendo que se muevan más rápido o más lento.
- Modificadores de la gravedad que influirán en como la gravedad afectará al elemento del nivel al que afectará.

Tecnología:

El juego se desarrollará en Unity, utilizando el lenguaje de programación C#.

Visión general del juego:

El juego consistirá en un plataformas 2D de viajes en el tiempo en el que el jugador atravesará una serie de niveles en los que el jugador hará uso de varias mecánicas que influirán en el espacio, el tiempo y la gravedad para esquivar los obstáculos que se interpongan en el camino del jugador hasta llegar la zona de victoria.

F.2. Mecánicas del juego

Jugador

El jugador controlará un avatar que es capaz de moverse hacia la derecha e izquierda. También puede saltar. El salto es uniforme, se saltará cada vez que se pulsa el botón de saltar (espacio en teclado y botón X en el mando) y saltará la misma distancia siempre (la fuerza del salto no variará en función de cuánto tiempo se mantenga pulsado el botón de salto).

Si el jugador colisionase con un obstáculo que le haga daño, el jugador morirá y reaparecerá en la zona inicial del nivel.

El jugador además tendrá la habilidad de dar un acelerón hacia la izquierda o la derecha. Este acelerón se realizará en una dirección horizontal, sin variar la posición vertical. Si el acelerón te desplazase X unidades hacia la derecha y se estuviese en el punto (1,1), el jugador acabará el acelerón en la posición (1, 1+X). El acelerón lo podrás hacer una sola vez mientras estés en el aire, y en cuanto toques el suelo podrás volver a utilizarlo. En el suelo podrás hacerlo ilimitadamente. Parte de la gracia de este acelerón, es que después de terminarlo se mantendrá la velocidad que se llevaba durante el acelerón, ahora sí, variando esta según las leyes que rigen el videojuego.

La última herramienta con la que contará el jugador es el tiempo bala. Esta mecánica permitirá reducir la velocidad a la que pasa el tiempo en el nivel permitiendo al jugador actuar con la precisión que ofrece que todo el nivel se reproduzca menor velocidad.

Enemigos

En principio no habrá enemigos vivos como tales. Sin embargo, sí que habrá obstáculos en el juego que, al tocarlos, mataran al jugador. Estos obstáculos podrán ser estáticos o móviles. Los enemigos estáticos no se mueven y la única dificultad radica en evitar colisionar con él. Los obstáculos móviles, sin embargo, podrán ser de dos tipos.

El primer tipo será un obstáculo que aparecerá en un extremo de la pantalla y se dirigirá al otro en línea recta. Son obstáculos fáciles de esquivar y predecibles. La dificultad de estos obstáculos radica en lo complejo que puede resultar enfrentarse a varios de ellos y la complejidad improvisada que puede surgir al añadir el obstáculo móvil a un escenario complejo ya de por sí.

El segundo tipo de obstáculo móvil será un obstáculo que se limite a una zona del nivel, pero que siga una rutina de movimiento en esa zona. El riesgo de este obstáculo se limita a una zona reducida del nivel, pero dentro de esta zona el jugador correrá serio riesgo de ser asesinado por el obstáculo.

Portal

Los portales son dos elementos enlazados que parten del hecho de que si se entra por un portal la posición del jugador (en ese momento la del portal que se ha atravesado) se convertirá en la posición del portal pareja del portal por el que se ha entrado. Esta mecánica es sencilla y para nada innovadora, pero que funciona muy bien. Un portal teletransportador es una idea sencilla de utilizar y entender por el jugador.

La gracia de los portales está en salir del segundo portal. Cuando se salga del segundo portal, el jugador mantendrá la velocidad con la que entró por el primer portal. Esto convierte al hecho de salir por un portal en una mecánica muy variada (y con posibilidades potencialmente infinitas) y que puede dar lugar a puzzles interesantes.

Creador de impulso

Esta mecánica consistirá en influir sobre la dirección de la velocidad que lleva un elemento afectado por físicas. Esta mecánica se puede manifestar de distintas formas:

- Partícula de impulso: Consistirá en un elemento estático en el mapa. Cuando el elemento afectado por las físicas entra en contacto con esta partícula de impulso, este saldrá disparado en una dirección predefinida. La dirección en la que saldrá disparado el elemento afectado por las físicas será siempre la misma. Eso sí, la dirección es individual para cada partícula, siendo que la partícula siempre disparará el elemento afectado por físicas en una dirección concreta, pero que no tendrá por qué ser la misma para dos partículas distintas.
- Amplificador de impulso: Consistirá en un elemento estático, al igual que la partícula, pero con un funcionamiento ligeramente distinto. Cuando el elemento afectado por físicas entra en contacto con el amplificador, la velocidad que lleva el elemento afectado por físicas se verá escalada. Dos ejemplos de amplificador serían: amplificador positivo de impulso (velocidad = velocidad * X) y amplificador negativo de impulso (velocidad = velocidad/X). Los amplificadores tendrán

cada uno un valor particular de escalado del impulso que no tiene por qué coincidir con el resto de amplificadores de impulso.

- Plataforma de salto: La típica plataforma de salto que propulsará al jugador cuando entra en contacto con esta. Puede no parecer un elemento creador de impulso, y puede que no lo sea, pero como su funcionamiento será igual al de un creador de impulso, se va a considerar un creador de impulso.

Modificadores de gravedad

Elementos que van a afectar en cómo influye la gravedad sobre uno o varios elementos afectados por físicas. Hay dos tipos de modificadores de gravedad en el juego:

Inversor de gravedad: Elemento estático que provocará que la gravedad se invierta. Esta inversión es “absoluta” en el sentido de que invertir una gravedad de -9,81 hace que esta se convierta en 9,81. Pero una gravedad de -8 invertida la convierte en 8 y no en 9,81.

Obstáculos superdensos: Estos son un tipo especial de enemigo, que, a su alrededor, generarán un campo gravitatorio que empujará a los elementos afectados por físicas hacia ellos. Estos obstáculos tendrán que luchar por la gravedad que afecta a los elementos afectados por físicas. Como ejemplo, si un elemento afectado por físicas se viese afectado por una gravedad de (-9,81, 0) y un obstáculo con una gravedad (5, 0) el elemento afectado por físicas, en caso de estar debajo del obstáculo verá su gravedad convertida a (-4,81, 0), pero si el elemento afectado por físicas está encima del obstáculo su gravedad será convertida a (-13,81, 0).

Esta es una explicación rápida para que se entienda, pero también afectará al elemento afectado por físicas la distancia a la que se encuentre del obstáculo.

En caso de alcanzar el centro de este obstáculo el elemento afectado por físicas morirá si tiene vida, si no se quedará atrapado en el centro.

Tiempo bala

Esta mecánica consistirá en manipular como el tiempo afecta a uno o varios elementos afectados por físicas. Esta mecánica es la más costosa de implementar tanto en tiempo como en esfuerzo. Es por ello que no se va a ser demasiado preciso al respecto, pero en principio, esta mecánica se manifestará de dos formas distintas.

Escalar el tiempo: El jugador podrá pulsar el botón de “Tiempo bala” y reducir el tiempo y como este afectará al entorno. Esta reducción de tiempo será una escala de $1/X$ veces la influencia que tiene el tiempo sobre los elementos afectados por físicas. Esta mecánica afectará a todos los elementos por igual y la escala será siempre la misma para todo el juego y todos los niveles.

Zonas de tiempo escalado: Son zonas en el nivel que escalarán el tiempo de todos los elementos que entren dentro de su área de influencia. La forma en la que escalarán el tiempo puede ser positiva ($\text{tiempo} = \text{tiempo} * X$) o negativa ($\text{tiempo} = \text{tiempo}/X$).

Las zonas de tiempo escalado serán estáticas en el mapa y la magnitud en la que escalan el tiempo es particular para cada zona, pudiendo ser distinta del resto de zonas de tiempo escalado.

F.3. Niveles

Habrá tres tipos distintos de niveles:

Niveles de prueba

Estos niveles serán inaccesibles para el jugador. Son niveles utilizados por el desarrollador para comprobar el correcto funcionamiento del juego e incluso forzar algunos errores intencionalmente. El uso de estos niveles será exclusivo para el desarrollo del proyecto.

Niveles tutorial

Niveles básicos utilizados para introducir mecánicas al jugador y ayudarles a comprender los conceptos que se le explicarán en un entorno controlado. Estos niveles podrán coincidir con algún nivel de prueba que resulte conveniente tanto para comprobar el funcionamiento básico de una mecánica como para explicar el funcionamiento de la mecánica.

Como ejemplo se va a poner el nivel de prueba de la mecánica de los portales:

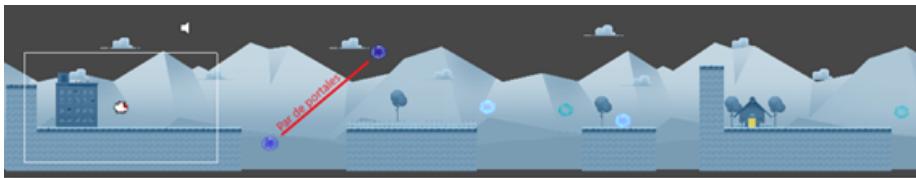


Figura F.1: Nivel tutorial de los portales

Este nivel se usará para comprobar el correcto funcionamiento de los portales. Pero adicionalmente introducirá una serie de conceptos interesantes de manera sencilla y “natural” que pueden servir como tutorial de esta mecánica al jugador.

Los conceptos que introducirá este nivel y lo hace un buen tutorial es que el jugador no tendrá mecánicas ajenas interponiéndose en el tutorial (y las que lo harán, como el salto, son mecánicas que el jugador ya tendrá interiorizadas). Otro punto fuerte del nivel reside en que solucionará dudas lógicas al jugador del tipo ¿Cómo puedo saber dónde voy a salir si entro por el portal y cómo diferencio pares de portales? Por último, este nivel no podrá ser superado sin hacer uso de la mecánica que se desea explicar.

Esta exemplificación es importante porque todos los niveles tutoriales seguirán (en mayor o menor medida) esta estructura.

Niveles desafiantes

El tercer tipo de nivel no dice mucho por su nombre, siendo este demasiado general. Esto no es algo malo, ya que este tercer apartado abarcará todos los niveles que no tienen un propósito explícito. El objetivo de este nivel será exclusivamente alcanzar la zona de victoria. La dificultad de estos niveles podrá ser variable y las mecánicas ser combinadas sin compromiso. Evidentemente estos niveles estarán regidos por algunas limitaciones como no utilizar mecánicas que no hayan sido presentadas en un tutorial todavía o que (preferentemente) la dificultad de los niveles sea mayor en los últimos niveles antes que en los primeros.

Pantallas implementadas

PruebaPlayerScene

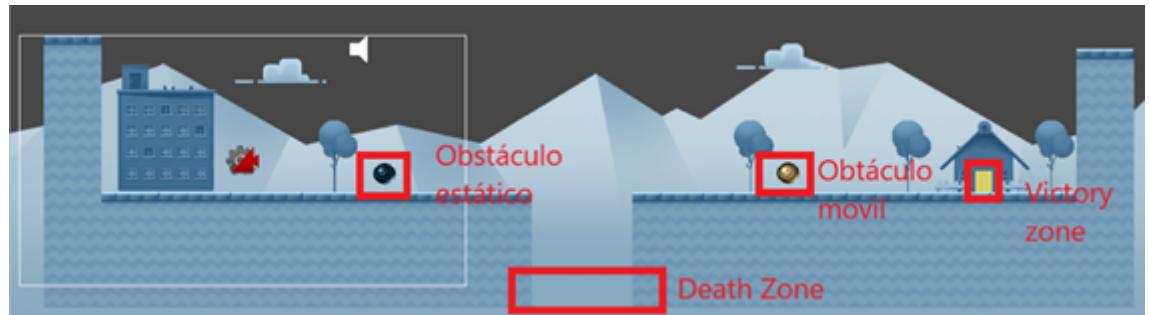


Figura F.2: Pantalla de prueba de mecánicas básicas

PruebaPortalScene



Figura F.3: Pantalla de prueba de los portales

PruebaMovingObstacleScene



Figura F.4: Pantalla de prueba de los obstáculos móviles

PruebaImpulseCreatorsScene

Figura F.5: Pantalla de prueba de los creadores de impulso

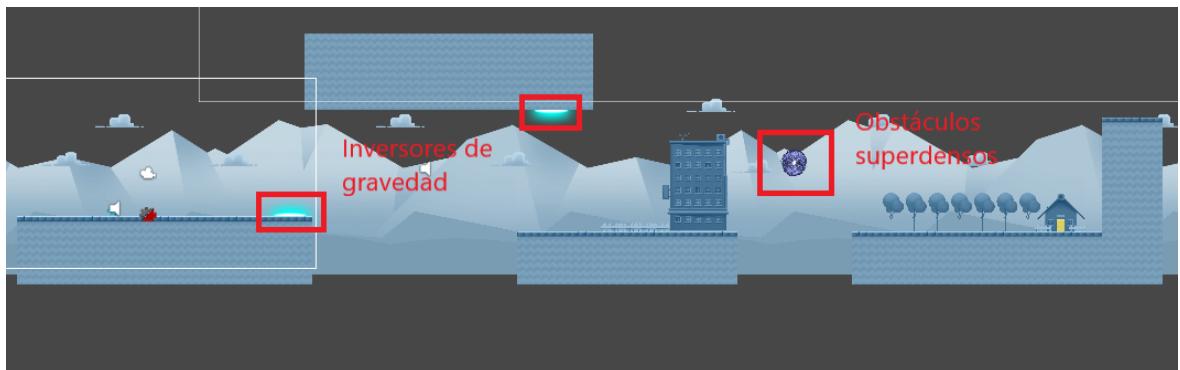
PruebaGravityModifierScene

Figura F.6: Pantalla de prueba de los modificadores de gravedad

PruebaTimeModifierScene

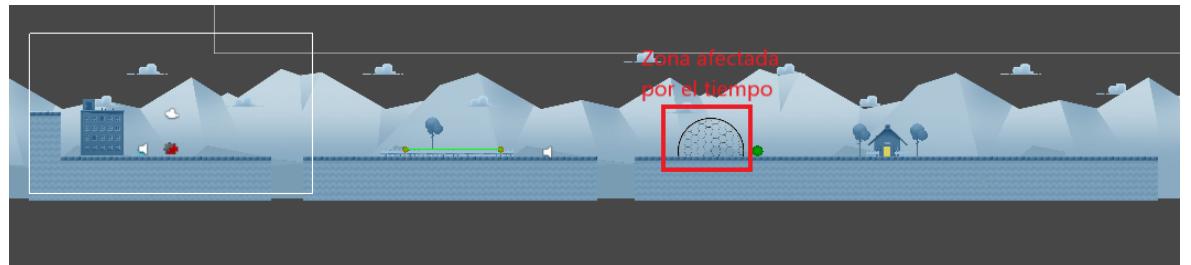


Figura F.7: Pantalla de prueba de los modificadores temporales

F.4. Interfaces

El patrón de pantalla que más se va a repetir será el nivel sin ningún elemento extradiegético que afecte a la pantalla del nivel. No se incluirán elementos HUD (Head-Up Display) que monitoricen la vida ni otros elementos, prácticamente por su ausencia (siendo el estado de los elementos variables, si se podrá saltar o si se estará el acelerón disponible para su uso, muy sencillos de mantener en mente y notificar con sonidos o animaciones).

Habrá, aun así dos interfaces extra disponibles para el jugador. Al no tener las interfaces todavía desarrolladas se va a añadir a continuación un esquema de cómo serán estas interfaces.

Pantalla de elección de nivel



Figura F.8: Idea inicial de la pantalla de selección de nivel

Pantalla de opciones

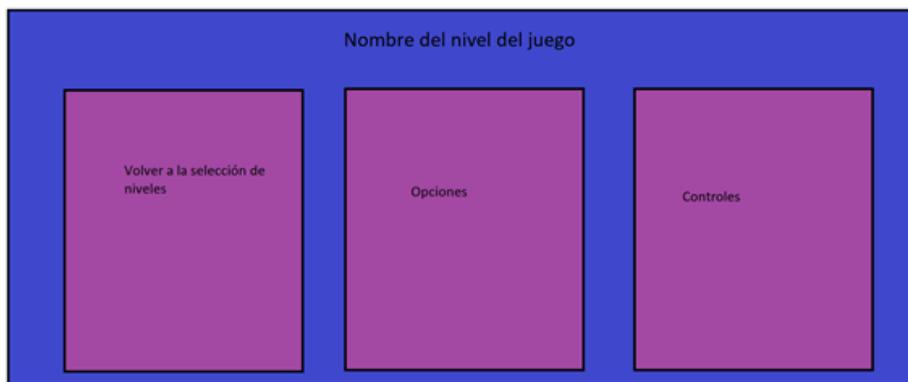


Figura F.9: Idea inicial de la pantalla de opciones

Estas pantallas ^{1,2}son orientativas y susceptibles de cambios. De la misma forma no se tiene que seguir ciegamente el patrón de colores, pero se ha de entender la existencia de un patrón de colores y como este se va a aplicar.

F.5. Entidades

A continuación se incluirá información más concreta al respecto del diseño de juego como sprites e imágenes que identificarán a una mecánica o elemento del juego en los niveles, configuración de los controles y descripción una por una de cada nivel del juego.

²El sonido de selección de los botones ha sido obtenido del enlace: <https://opengameart.org/content/button-sound-pack>.

Puedes encontrar al creador en twitter como @listener4me. https://twitter.com/search?q=%40listener4me&src=typed_query.

El sonido de desplazamiento entre botones ha sido obtenido en el siguiente enlace: <https://opengameart.org/content/menu-selection-click>.

²Los sonidos de cierre del menú de pausa y de la barra de los menús de opciones pertenecen a Jesús Lastra han sido obtenidos en el siguiente enlace: <https://opengameart.org/content/button-clicks-beeps-99-sounds>.

La música que suena mientras se está en estas escenas es 01 B-arb del album B-arb, pertenece a Axel Bermudez y se puede encontrar en el siguiente enlace (junto con el resto de canciones del album del que se ha sacado la canción): <https://axelbermudez.bandcamp.com/releases>.

Elementos

Player

Objeto que controlará el jugador. El jugador tendrá la capacidad de hacer que el Player se mueva, salte y pegue acelerones.

El Player ³se podrá morir en caso de que colisione con un objeto que reduzca su vida a 0. El Player tendrá solo 1 punto de vida y cuando colisione con un objeto que lo “dañe”, su vida se reducirá a 0 y morirá. Cuando el Player muera reaparecerá en un punto de reaparición establecido en la escena.

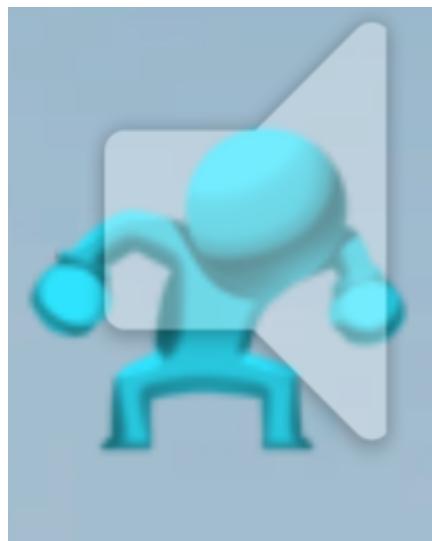


Figura F.10: Sprite utilizado para el Player

Los sprites, las animaciones y los sonidos del Player los ofrecía Platformer Microgame para uso libre.

³El audio del acelerón ha sido obtenido en el siguiente enlace: <https://opengameart.org/content/sci-fi-shwop-1>

Enemigos

- Obstáculos ⁴: Enemigos que no se moverán. Se encontrarán estáticos en un lugar y si el Player los tocase este se muere.

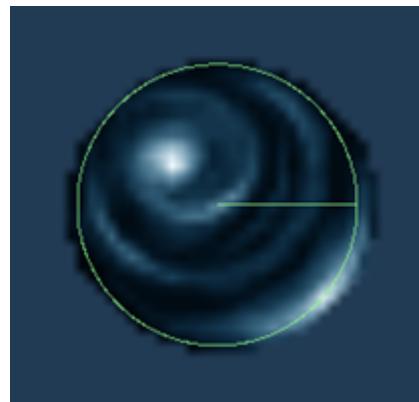


Figura F.11: Sprite utilizado para el obstáculo estático

- Obstáculos que seguirán una rutina ⁵: Enemigos que se encontrarán en una zona y recorrerán un camino cíclico continuamente.

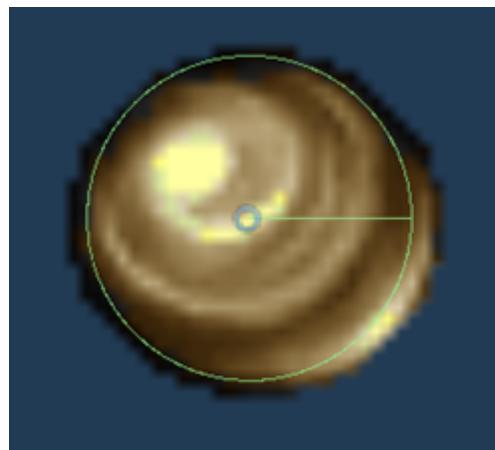


Figura F.12: Sprite utilizado para el obstáculo que sigue una rutina

- Obstáculos móviles ⁶: Enemigos que aparecerán en un extremo de la pantalla y la atravesará hasta el otro extremo. Si en algún momento el obstáculo móvil tocase al Player, este morirá.

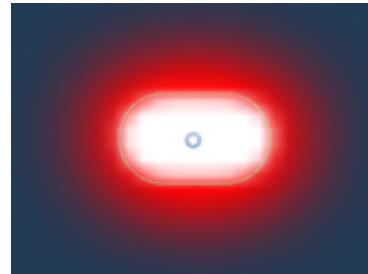


Figura F.13: Obstáculo móvil rápido

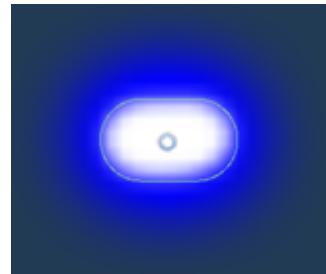


Figura F.14: Obstáculo móvil (velocidad intermedia)



Figura F.15: Obstáculo móvil lento

Escenarios

- Suelo
- Pared
- Zonas de muerte
- Zonas de victoria

Mecánicas

Portal

El portal⁷ que teletransportará a los objetos que entren manteniendo la dirección del movimiento del objeto.

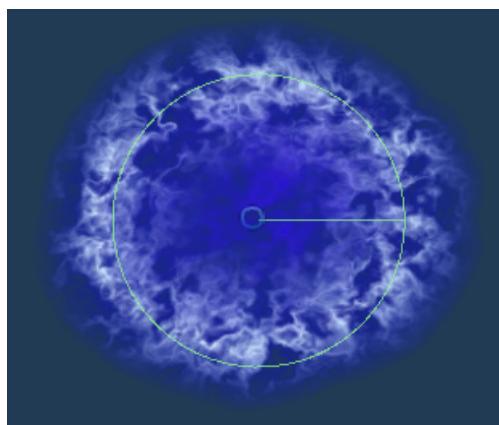


Figura F.16: Sprite utilizado para los portales

Creador de impulso

El objeto afectado recibirá un impulso en una dirección.

⁷El sprite utilizado para los portales pertenece a Hansjörg Malthaner y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/animated-portal-or-wormhole-several-variants>.

Enlace al trabajo de Hansjörg Malthaner: <http://opengameart.org/users/varkalandar>.

El audio utilizado para los portales pertenece a dklon y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/quick-zap>.

Enlace al trabajo de dklon: <https://opengameart.org/users/dklon>

- Partícula de impulso ⁸: objeto que se encontrará en el escenario. Cuando un objeto toque esa partícula recibirá un impulso en la dirección asignada a la partícula.

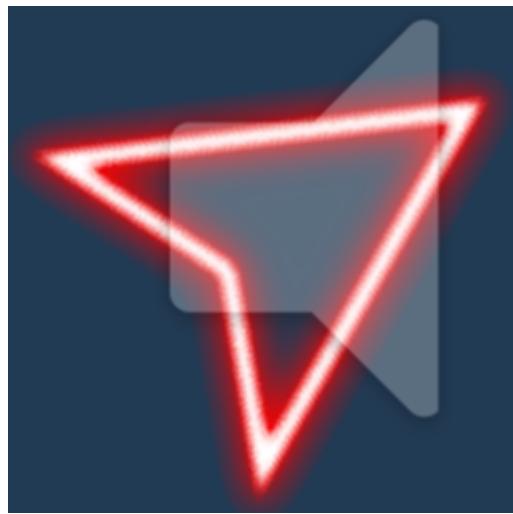


Figura F.17: Sprite utilizado para la partícula de impulso

- Amplificador de impulso ⁹: parecido a la partícula de impulso, pero escalando el impulso que lleva el objeto.

⁸Los sprites utilizados para la partícula de impulso pertenecen a oglSDL y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/glow-arrow>. El audio utilizado para la partícula de impulso de salto pertenece a Bart Kelsey y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/spell-2>.

⁹Los sprites utilizados para el amplificador de impulso son una modificación de un Sprite que pertenece a oglSDL que ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/glow-arrow>. El audio utilizado para la partícula de impulso de salto pertenece a Bart Kelsey y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/spell-2>



Figura F.18: Sprite utilizado para el amplificador de impulso

- Plataforma de salto ¹⁰: Se define por si sola.



Figura F.19: Sprite utilizado para la plataforma de salto

¹⁰Los sprites utilizados para la plataforma de salto pertenecen a diana23570 y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/spring>. El audio utilizado para la plataforma de salto pertenece a Blender Foundation y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/sprint-jumpinteraction-sound-yo-frankie>

Tiempo bala

Capacidad de “ralentizar o acelerar el tiempo” influyendo en la velocidad a la que afectará al uno o varios objetos que se vean afectados.

- Escalar el tiempo: Capacidad del Player para escalar el tiempo que afectará a todos los objetos.
- Zonas de tiempo escalado ¹¹: Zonas que escalarán el tiempo de los objetos que entren en la zona de influencia de la zona.

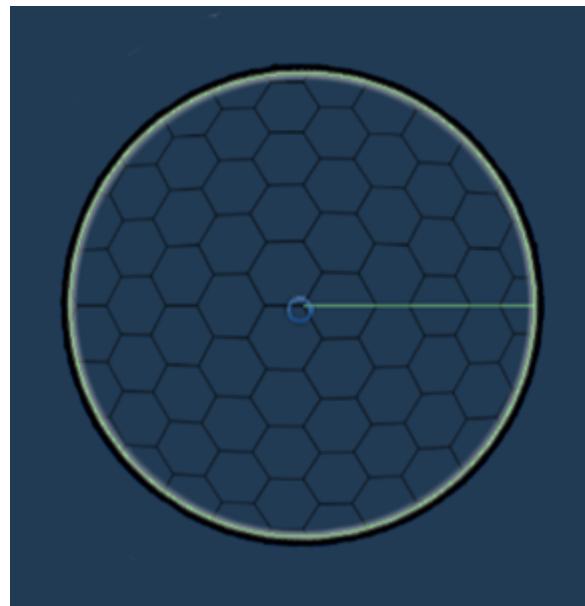


Figura F.20: Sprite utilizado para la zona de tiempo escalado

Acelerón

El jugador realizará un acelerón en la dirección en la que esté mirando.

Modificadores de gravedad

Objetos que afectarán a como la gravedad influye sobre ellos u otros objetos.

¹¹El sprite utilizado para las zonas de tiempo escalado es una modificación de un sprite que pertenece a scenna y ha sido obtenida en el siguiente enlace: <https://opengameart.org/content/circle>.

- Inversor de gravedad ¹²: volverá la gravedad negativa en positiva y viceversa.

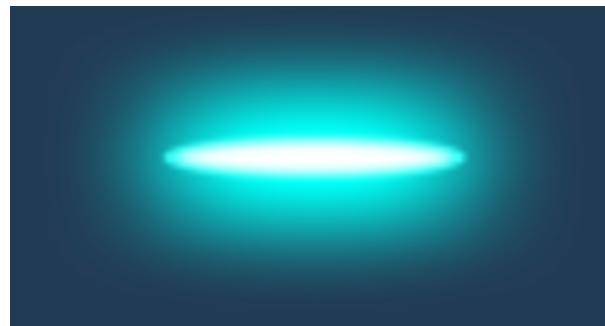


Figura F.21: Sprite utilizado para el inversor de gravedad

- Obstáculos superdensos ¹³: Obstáculos hacia los que atraerán a los objetos debido a su alta gravedad.

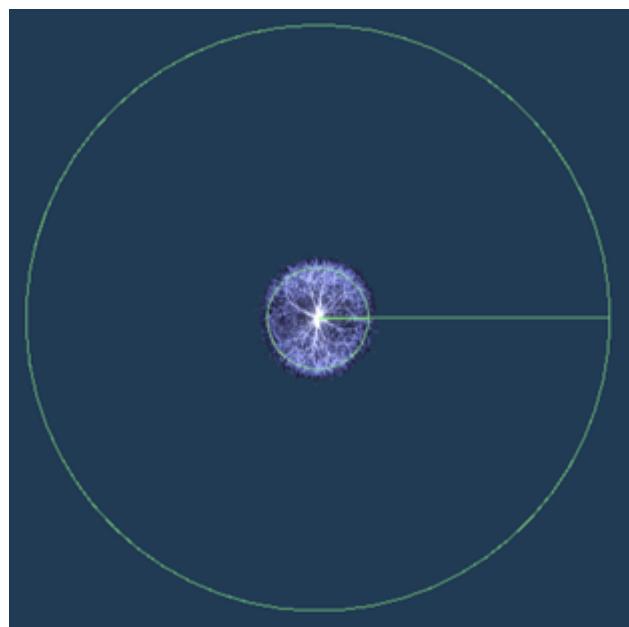


Figura F.22: Sprite utilizado para el obstáculo superdenso

F.6. Controles

Mando

Los controles del mando se regirán por la nomenclatura de los controles de PlayStation.



Figura F.23: Nomenclatura de los mandos de PlayStation

- **Movimiento:** Stick izquierdo.
- **Salto:** botón X (Joystick button 1 en Unity)
- **Acelerón:** botón O (Joystick button 2 en Unity)
- **Tiempo bala:** botón R1 (Joystick button 5 en Unity)
- **Abrir el menú de pausa:** botón R2(Joystick button 7 en Unity)

Teclado

- **Movimiento:** flecha izquierda para ir a la izquierda y flecha derecha para ir a la derecha.
- **Salto:** espacio
- **Acelerón:** tecla S

- **Tiempo bala:** tecla D
- **Abrir el menú de pausa:** tecla ESC

Bibliografía

- [1] Ministerio de cultura de España. Ley de propiedad intelectual. <https://boe.es/buscar/pdf/1996/BOE-A-1996-8930-consolidado.pdf>, 1996. [Internet; Ultimo acceso: 6-junio-2021].
- [2] Martin Fowler. Give-when-then. <https://martinfowler.com/bliki/GivenWhenThen.html>, 2021. [Internet; Ultimo acceso: 7-junio-2021].
- [3] Sergio C. González. Unidades vendidas por los juegos indies. https://as.com/meristation/2019/09/11/noticias/1568178068_654037.html, 2019. [Internet; Ultimo acceso: 6-junio-2021].
- [4] Jobted. Sueldo de un programador de videojuegos. <https://www.jobted.es/salario/programador-videojuegos>, 2021. [Internet; Ultimo acceso: 6-junio-2021].
- [5] Kanban tool. Tablero kanban. <https://kanbantool.com/es/tablero-kanban>, 2021. [Internet; Ultimo acceso: 6-junio-2021].
- [6] Unity. Documentación de la clase audioclip. <https://docs.unity3d.com/ScriptReference/ AudioClip.html>, 2021. [Internet; Ultimo acceso: 8-junio-2021].
- [7] Unity. Documentación de la clase audiosource. <https://docs.unity3d.com/ScriptReference/ AudioSource.html>, 2021. [Internet; Ultimo acceso: 8-junio-2021].
- [8] Unity. Documentación de la clase spriterenderer. <https://docs.unity3d.com/ScriptReference/ SpriteRenderer.html>, 2021. [Internet; Ultimo acceso: 8-junio-2021].

- [9] Unity. Faq unity. <https://unity3d.com/es/unity/faq/2491>, 2021. [Internet; Ultimo acceso: 6-junio-2021].
- [10] Wikipedia. Metodología kanban. [https://es.wikipedia.org/wiki/Kanban_\(desarrollo\)](https://es.wikipedia.org/wiki/Kanban_(desarrollo)), 2020. [Internet; Ultimo acceso: 6-junio-2021].