Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería Mecánica

Diseño y desarrollo de juegos educativos 3D en Unity basados en la Industria 4.0 y en herramientas Lean

Autor:

Díez García, Rodrigo

Tutores:

Sanz Angulo, Pedro

Galindo Melero, Jesús

Dpto. de Organización de Empresas y C. e I.M.

Valladolid, mayo de 2023.

DEDICATORIA

En este final de trayecto, me gustaría dedicar este Trabajo Fin de Grado a mi abuelo, quien siempre fue un apoyo incondicional en mi formación y desarrollo personal. Aunque hoy ya no está con nosotros, sé que estaría orgulloso de mi logro al alcanzar este hito académico.

Además, me gustaría concluir esta dedicatoria con una frase que me dijo una vez un buen profesor y que me ha inspirado durante todo el proceso de elaboración de este trabajo: "Nadie te va a preguntar cuánto tiempo has tardado, te preguntarán si lo has logrado". Esta cita me ha recordado la importancia de mantener el enfoque en los objetivos y trabajar con dedicación para alcanzarlos. Espero que este trabajo demuestre el esfuerzo y el compromiso que he puesto en él, y que pueda ser una contribución valiosa en el ámbito de la ingeniería.

AGRADECIMIENTOS

Me gustaría empezar agradeciéndome a mí mismo todo el sacrificio que he tenido que realizar para llegar a este punto. No fue sencillo, pero se logró. Puede sonar prepotente y egocéntrico, pero la realidad es que durante el trayecto ha habido de todo, desde el mayor lloro hasta la mayor ilusión. Con gran constancia y dedicación gracias tener claro mis objetivos.

Gracias al apoyo incondicional de mis padres y hermana durante todo el trayecto académico. Gracias a mi padre Gregorio por darme siempre mucho más de lo que necesitaba. A mi madre, Rosa, por esas charlas motivacionales tan esenciales para el aprendizaje. También a mi hermana Paula, que cuando he necesitado ayuda de verdad siempre ha estado ahí. Y a Sara, ese pilar fundamental que he necesitado siempre con un apoyo increible.

Gracias también a mi tutor del proyecto Pedro por hacer más ameno el estrés y el agobio de llevar un proyecto de este calibre y por ser tan comprensivo con mis inquietudes y necesidades.

Por último, me gustaría agradecer a FischerTechnik el que nos haya facilitado los modelos 3D de la fábrica, necesarios para la creación de una parte de este proyecto.

Gracias a todos por cada granito.

Resumen

En el presente documento se describe el diseño y programación de unos videojuegos diseñados en Unity que permiten a los usuarios recorrer un modelo de fábrica de Industria 4.0 y trabajar conceptos de la teoría de Lean Manufacturing. En concreto, se ha creado un juego basado en la maqueta de la empresa FischerTechnik en la que el jugador realiza un tour por la fábrica de industria 4.0 desarrollada por dicha empresa. Además, se han desarrollado unos minijuegos interconectados, que integran los conceptos del método Kanban, la filosofía de los cinco ceros y las 5S. El documento describe el desarrollo paso a paso del diseño y programación de estos videojuegos en el leguaje C# de Visual Studio.

Palabras clave

Videojuegos 3D; Educación; Unity; Lean Manufacturing; Industria 4.0; Programación C#; Método Kanban; Filosofía cinco ceros; 5S

Abstract

This document describes the design and programming of video games designed in Unity that allow users to go through an Industry 4.0 factory model and work on Lean Manufacturing theory concepts. Specifically, a game has been created based on the model of the FischerTechnik company in which the player takes a tour of the industry 4.0 factory developed by the company. In addition, some interconnected mini-games have been developed, which integrate the concepts of the Kanban method, the philosophy of the five zeros and the 5S. The document describes the step-by-step development of the design and programming of these video games in the C# language using Visual Studio IDE.

Keywords

3D video games; Education; Unity; Lean Manufacturing; Industry 4.0; C# programming; Kanban method; Philosophy five zeros; 5S

Índice de contenido

ÍNDICE DE CÓDIGOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Antecedentes

Hoy en día, los cambios tecnológicos suceden de forma continua y tienen la capacidad de transformar el mundo en el que vivimos. Estos cambios, asociados a la llamada cuarta revolución industrial o Industria 4.0 (I40), afectan a todos los ámbitos de nuestras vidas, incluyendo el sistema productivo y la forma de trabajo. En el sector industrial se puede apreciar una creciente importancia de la digitalización de los procesos, al igual que la expansión de conceptos como el internet de las cosas (IoT, *Internet of Things*), la realidad virtual y aumentada o la inteligencia artificial (AI, *Artificial Intelligence*) (Barleta, 2020).

Todas estas tecnologías deben verse como un medio para mejorar la competitividad y sostenibilidad de las operaciones logísticas necesarias para satisfacer las demandas actuales de desarrollo en una empresa. Los fabricantes deben incorporar nuevas tecnologías en sus instalaciones de producción como sensores avanzados, un software integrado y robótica (entre otros), que recopile y analice datos, con el fin de mejorar la toma de decisiones.

Es evidente que el verdadero reto está en cómo las personas y sus competencias liderarán y gestionarán este proceso de transformación digital en las empresas, así como el cambio que implicará adaptarse a trabajar en los nuevos entornos interconectados. Aquellos que puedan aprovechar las nuevas oportunidades que surjan a raíz de estas tecnologías emergentes, tendrán la posibilidad de diferenciarse respecto a aquellos que no logren adaptarse.

Las tecnologías habilitadoras que nos ofrece la industria 4.0 son una realidad y están disponibles; el reto está en hacer un uso efectivo de ellas en la empresa, además de

contar con el personal que tenga las competencias necesarias para poder explotarlas al máximo (Martínez, 2021).

Motivación

La aparición mundial del COVID–19 ha contribuido, sin ninguna duda, a acelerar la digitalización, el teletrabajo y la docencia telemática. Las tecnologías habilitadoras han ayudado al enfrentamiento de esta crisis global, dando la posibilidad de una comunicación más efectiva. Sin embargo, aún queda mucho por desarrollar y mejorar en todo este proceso. A medida que la tecnología ha ido avanzada, los videojuegos se han convertido en una herramienta educativa eficaz y atractiva para una amplia variedad de públicos.

Actualmente vivimos en una sociedad donde desde edades muy tempranas se tienen un control y acceso libre y directo con medio dinámicos y juegos en diferentes plataformas. Estos juegos pueden orientarse hacia todo tipo de aprendizajes y hacia la docencia en ámbitos de ingeniería que contribuyen a motivar e incentivar estos aprendizajes que en ocasiones pueden llegar a ser muy tediosos.

Por tanto, podemos decir que la docencia está en un proceso de cambio en el que las clases deben evolucionar junto a la tecnología para que las nuevas generaciones puedan aprender de una manera mucho más dinámica y efectiva. De esta manera, se conseguirá que los estudiantes tengan una perspectiva diferente y útil en su aprendizaje.

Recientemente, en el departamento de Organización de Empresas y C. e I.M. de la UVa se adquirió un modelo físico de la fábrica de industria 4.0 a la empresa FischerTechnik. Con ella "*se pueden simular, aprender e implementar estas actividades de digitalización a pequeña escala, antes de aplicarlas a una escala mayor en cualquier empresa*" (FischerTechnik, 2022).

A raíz de su adquisición, se plantearon diferentes alternativas para expandir sus posibilidades, tanto de las tecnologías habilitadoras de I40 que ya implementan (*IoT*, computación en la nube, sistemas ciber físicos, ...) y, en especial, de las que no, como es el caso de la realidad virtual, una tecnología que cada vez tiene más peso en el ámbito industrial. Como una primera aproximación, en este TFG nos planteamos crear diferentes videojuegos empleando las mismas herramientas que después puedan servir para introducirnos en el metaverso.

Objetivos

El objetivo de este trabajo fin de grado consiste en diseñar y desarrollar videojuegos basados en el modelo físico de industria 4.0 de FischerTechnik y en herramientas de Lean Manufacturing. Estos juegos educativos podrían ser un instrumento muy útil para enseñar a los estudiantes sobre los principios y las prácticas de la industria 4.0 y Lean Manufacturing, en un entorno divertido e interactivo.

Debe quedar claro el alcance que se persigue, que en ningún caso es la publicación a gran escala de ninguno de los juegos que se realicen, sino el diseño y la creación de estos juegos con el objetivo de aplicarlos a pequeña escala. Es decir, se pretende un uso localizado que permita a estudiantes de ingeniería apoyar la comprensión de estos conceptos de una manera practica y virtual. De esta manera conseguiremos atraer la atención de los alumnos hacia conceptos de la I40 y del Lean Manufacturing.

Además, y como se ha indicado, el desarrollo de juegos educativos es solo el primer paso hacia la creación de juegos de realidad virtual que simulan experiencias y entornos reales en las líneas futuras de investigación. Estos juegos de realidad virtual tienen el potencial de ser herramientas de aprendizaje aún más poderosas, al permitir a los usuarios sumergirse en entornos virtuales que simulan situaciones reales de trabajo.

Cabe destacar que este documento podrá servir como base de desarrollo útil e interesante para las líneas futuras donde se podrá trabajar con realidad virtual, aumentada o mixta. Además, en ningún momento se buscará que el potencial gráfico sea alto, sino que los juegos que se creen sean útiles y funcionales, de modo que los los alumnos puedan usarlos desde dispositivos de PC habituales; es decir, sin que sea preciso contar con ordenadores muy potentes o cualquier otro dispositivo fuera de alcance habitual.

Este trabajo tiene el potencial de ayudar a mejorar la educación en estos campos, y puede ser de gran utilidad para aquellos interesados en la creación de herramientas de aprendizaje innovadoras y atractivas.

Con el fin de alcanzar el objetivo propuesto, es necesario llevar a cabo un estudio de las herramientas disponibles para el diseño y desarrollo de videojuegos. Una vez concluido este análisis, se procederá a la selección de la herramienta más adecuada en función de los objetivos finales del proyecto. Adicionalmente, es preciso adquirir conocimientos avanzados en programación orientada a objetos ya que, aunque el grado en ingeniería proporciona una base sólida, es necesario ampliarla para adaptarla a los requerimientos específicos de los videojuegos.

Una vez se hayan establecido los puntos anteriores, se deberá estudiar el funcionamiento y posibilidades de la maqueta de la industria 4.0 de FischerTechnik. Este análisis permitirá definir con precisión los juegos que se pueden desarrollar y construir, teniendo en cuenta las limitaciones y posibilidades que se presentan. Cabe

destacar que en este proyecto no se cuenta con el apoyo de diseñadores gráficos o programadores profesionales, por lo que se busca una perspectiva más limitada pero funcional.

Por último, es fundamental tomar conciencia de las herramientas de Lean Manufacturing para poder diseñar juegos que estén relacionados con estos conceptos y que permitan afianzar o explicar de manera práctica y cercana dichos conceptos al alumno. Además, aprovechando los conocimientos de ingeniería mecánica adquiridos durante la carrera, se buscará en todo momento un contexto industrial en las escenas del juego que permita al estudiante comprender de manera más clara el mundo de la industria 4.0.

La integración de estos conceptos en los videojuegos permite que los usuarios se familiaricen con ellos de una manera interactiva y entretenida, lo que puede mejorar su capacidad para aplicarlos en situaciones del mundo real. En especial, las denominadas tecnologías habilitadoras y su recurrencia en la práctica del aprendizaje, con la finalidad de contribuir al desarrollo de competencias demandadas por la industria 4.0.

Estructura

En la INTRODUCCIÓN se contextualiza el trabajo, se plantea su motivación y se presenta el objetivo general del trabajo, explicando su importancia en el contexto actual de la Industria 4.0.

En el primer capítulo, titulado MARCO TEÓRICO, se presenta una revisión de la literatura sobre los temas relevantes al TFG. Se describen las principales características de la Industria 4.0, la relación entre los videojuegos y el aprendizaje, las bases para el desarrollo de un mundo virtual y las herramientas del Lean Manufacturing. Además, se analiza la aplicación de la Industria 4.0 en la empresa FischerTechnik.

En el segundo capítulo, titulado FISCHERTECHNIK VIRTUAL, se explica cómo se desarrolló el primer videojuego orientado al modelo físico de Industria 4.0 de FischerTechnik. No obstante, antes se presenta una descripción detallada de los conceptos básicos de Unity, el software utilizado para el desarrollo de los minijuegos.

A continuación, en el DESARROLLO DE MINIJUEGOS LEAN se describe el proceso de diseño y justificación de los minijuegos, la metodología de aplicación dentro del juego y la construcción y diseño de las diferentes escenas. Además, se detalla el proceso de generación del ejecutable del juego.

El capítulo de ESTUDIO ECONÓMICO se enfoca en el análisis económico del proyecto. Se definen las fases del proyecto, se analiza el coste de las horas efectivas y las tasas

horarias del personal, las amortizaciones del equipo informático, los costes de material consumible y los costes indirectos. También se asignan los costes a cada una de las fases del proyecto.

Finalmente, encontramos las CONCLUSIONES Y LÍNEAS FUTURAS donde se resumen las conclusiones alcanzadas en el desarrollo del TFG y se plantean posibles líneas de investigación futura.

En la BIBLIOGRAFÍA se incluyen las referencias bibliográficas consultadas durante la realización del TFG.

Por último, en los ANEXOS se incluyen todos los materiales adicionales que se consideran relevantes para el trabajo, como capturas de pantalla, códigos de programación, ..., entre otros.

1 MARCO TEÓRICO

A continuación, se presentan los conceptos fundamentales que conforman la Industria 4.0, que se refiere a la cuarta revolución industrial. Además, se examina el enfoque educativo que implica la utilización de videojuegos y la relevancia que esto conlleva. En este contexto, se propone un avance en los videojuegos al relacionarlos con la realidad virtual. Por último, se aborda la base de desarrollo requerida para la creación de este tipo de contenido, tanto en términos de hardware como de software.

1.1 Industria 4.0

El sector industrial es un organismo vivo en continuo cambio. Los fabricantes siempre buscan la mejora, el crecimiento, y la reducción tanto de los costes de producción como el tiempo de entrega. Hoy en día, las empresas deben competir entre sí y adaptarse tan rápido como las sea posible a las necesidades del mercado. Por eso surge la denominada cuarta revolución industrial, también conocida como Industria 4.0. Esta viene precedida de otras revoluciones industriales en las que se han realizado cambios importantes, tal y como se recoge en la Figura 1.

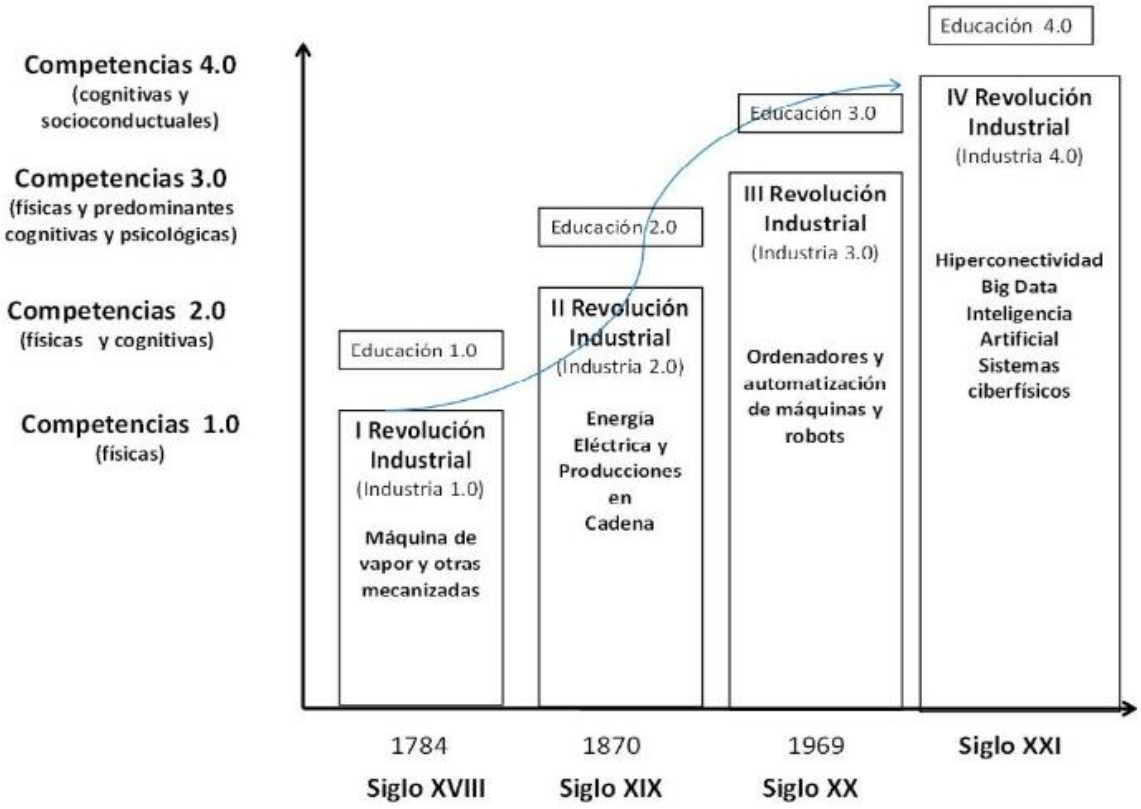

Figura 1: Evolución de las revoluciones industriales hasta la actualidad. (Martínez, 2021)

Esta cuarta revolución está basada en la digitalización, es decir, en la interconexión de los sistemas y el internet de las cosas (IoT, *Internet of Things*). A continuación, se exponen algunas de las características principales de la Industria 4.0 (RIPIPSA, 2019):

1. *Big Data*; guardado y análisis de grandes cantidades de información en tiempo real.
2. Robotización.
3. Simulación.
4. Sistemas de integración.
5. Internet de las cosas; conexión en red de varios dispositivos.
6. Ciberseguridad; protección de la información y procesos.
7. *Cloud computing*; nube digital que facilita el uso de aplicaciones y compartir información.
8. Manufactura aditiva; impresoras 3D para la creación de planos y dibujos complejos.
9. Realidad Virtual (VR, *Virtual Reality*) y Realidad Aumentada (AR, *Augmented Reality*).

Estas características son conocidas como las tecnologías habilitadoras de la industria 4.0. La Figura 2 representa de forma esquemática estas tecnologías.

Figura 2: Tecnologías habilitadoras de la Industria 4.0 (RIPIPSA, 2019).

En este trabajo se le da especial importancia a la última de las características expuestas anteriormente: la realidad virtual y realidad aumentada. Estas son "sistemas de visualización avanzada cuyas principales ventajas en la industria son la mejora de procesos, el aumento de la seguridad y la reducción de los costes." (Afonso, 2018).

1.2 Relación entre los videojuegos y el aprendizaje.

El juego siempre ha sido una herramienta de educación en todas las culturas, ya que es una transmisión de aprendizaje muy eficaz. Desde el principio, el juego era una herramienta de aprendizaje y supervivencia para la vida adulta, utilizándose en ocasiones como rito de iniciación o para el entrenamiento de los guerreros. Hoy en día, el juego se ha convertido en una forma de pasar el tiempo y ha evolucionado hacia formas más abstractas y multifuncionales. Depende en gran medida de los dispositivos electrónicos como las videoconsolas, los ordenadores, los teléfonos y las tablets.

La evolución de los videojuegos hace que sea más sencillo vivir una experiencia inversiva en un mundo virtual. Los videojuegos combinan tecnología y creatividad al mismo tiempo, lo que nos abre una gran cantidad de posibilidades en la creación de material didáctico. Existen certezas teóricas de que la tridimensionalidad y la realidad virtual favorecen notablemente "*la inmersión perspectiva y emocional del jugador*". (Perez García, 2015).

Los videojuegos pueden contribuir al desarrollo de las personas ya que proporcionan una gran motivación gracias a la estimulación visual y auditiva. Pueden ser un medio de aprendizaje atractivo y efectivo al permitir la repetición instantánea, lo que facilita el dominio de habilidades. También ayuda con la interacción con el entorno y la resolución de problemas (Perez García, 2015).

Según menciona Álvaro Pérez García en su informe "El aprendizaje como videojuego" (2015), el potencial de los videojuegos puede ser especialmente útil para estudiantes con necesidades educativas especiales. En este sentido, los videojuegos pueden abrir nuevas posibilidades a la creación de materiales didácticos más accesibles e inclusivos.

Los juegos son atractivos porque permiten crear y moldear al avatar o personaje; es precisamente esa libertad la que los hace tan interesantes. Además, como hemos visto, los juegos siempre han tenido un elemento educativo y en los últimos años se han empezado a utilizar como herramientas complementarias para la educación formal en el aula (Revuelta Dominguez & Guerra Antequera, 2012).

Cabe destacar que existen los llamados "*social games*" y el modelo de jugador casual, generados por empresas que buscan a los jugadores no "constantes". Aquellos jugadores que buscan algo más dentro del ocio, es decir, buscan un valor añadido dentro del propio juego además del entretenimiento. Por eso surgen videojuegos que sirven para hacer ejercicio, para aprender idiomas o incluso profesiones (Revuelta Dominguez & Guerra Antequera, 2012).

Podemos poner ejemplos muy claros de como todos los juegos tienen un aprendizaje intrínseco: el aprendizaje de las matemáticas dentro de los juegos de cartas, la obtención de cultura general en juegos como el Trivial, la cooperación en equipo en juegos como *Call of Dutty*, *League of Legends o Clash Royale entre otros.*

A continuación, se ha elaborado una lista en base a un trabajo social y estadístico titulado "¿Qué aprendo con videojuegos?: una perspectiva de meta-aprendizaje del video jugador" (Revuelta Dominguez & Guerra Antequera, 2012). En este nos encontramos con una muestra de 115 personas (47 hombres y 68 mujeres de edades entre 8 y 47 años). Se realizó un listado de los aprendizajes y procesos favorecedores del aprendizaje con videojuegos que desde el punto de vista de los jugadores son relevantes para la educación formal:

- Elemento motivador y favorecedor del rendimiento
- Aprovechamiento del binomio lúdico-educativo
- Adquisición de habilidades y/o destrezas para la resolución de problemas
- Socialización y cooperación
- Aumento de la concentración
- Capacidad de interacción
- Simulación de situaciones
- Mejora en la toma de decisiones
- *Feedback* inmediato
- Habilidades psicomotrices
- Gestión de recursos
- Motivación
- Aumento de la creatividad y la imaginación
- Trabajo en equipo

1.3 Propuesta de Realidad virtual

Aunque el presente documento no tiene como objetivo la implantación de tecnologías relacionadas con la realidad virtual, se considera importante mencionar que esta podría ser una de las líneas futuras de investigación y desarrollo de proyectos similares. Es plausible que la realidad virtual ofrezca numerosas oportunidades para mejorar la eficiencia de diversos campos de estudio, lo que justifica su potencial relevancia en futuras investigaciones.

Hoy en día no existe una definición oficial de lo que es la realidad virtual, por lo que podemos encontrar múltiples definiciones de este término. No obstante, a continuación se recoge la que creemos es la más completa de todas:

“Un sistema para poder ser considerado de realidad virtual debe ser capaz de generar digitalmente un entorno tridimensional en que el usuario se sienta presente y en el cual pueda interactuar intuitivamente y en “tiempo real” con los objetos que encuentre dentro de él.” (Levis & Levis, 2006).

Es importante destacar que la imaginación del creador juega un gran papel, ya que la interacción y la inmersión en el mundo virtual será más realista cuanto mayor sean los detalles.

No se puede hablar de realidad virtual sin mencionar la realidad aumentada. “La realidad aumentada (RA) es una tecnología que combina elementos del mundo real con elementos generados por computadora, creando una experiencia interactiva en tiempo real.” (Del Giorgio & Mon, 2018).

El funcionamiento de un ordenador para realidad virtual debe contar con mecanismos de entrada y salida para permitir la interacción del usuario con el mundo virtual. Los dispositivos de entrada deben ser precisos para leer las órdenes del usuario y las mediciones necesarias para actualizar la escena. Además, el ordenador debe constantemente localizar la posición del usuario y determinar las acciones de los objetos del mundo virtual de acuerdo con las instrucciones del usuario, las características de los objetos y el estado del sistema. Una vez que se tiene esta información, el ordenador actualiza la situación de la escena generando gráficos, sonidos y respuestas táctiles. (Levis & Levis, 2006).

Es importante que los dispositivos de entrada y salida transmitan la información de manera rápida e instantánea para no afectar la experiencia inversiva del usuario. Según Coiffet (1995), se establece un orden de prioridad para los estímulos sensoriales, siendo la vista el sentido principal, seguido del oído y, por último, las sensaciones táctiles y de esfuerzo. (Levis & Levis, 2006).

La realidad virtual y la realidad aumentada se están usando en diferentes sectores, como la educación, producción, logística y mantenimiento. En educación, se usan para prácticas seguras y visualización en 3D. En producción, para tener una vista global del proceso de fabricación y despiece de productos. En logística, para indicaciones visuales

y mejorar la manipulación de la mercancía. En mantenimiento y soporte, para detectar problemas en el lugar de trabajo y optimizar tareas con guiados en interiores y exteriores. (Del Giorgio & Mon, 2018).

1.4 Bases de desarrollo de un mundo virtual

Para la creación de un entorno virtual satisfactorio se precisan herramientas de software y hardware que logren la mejor inmersión posible. Estos dispositivos y programas contribuyen a estimular los sentidos, como la vista, el oído y el tacto, y son cruciales para lograr una conexión entre el mundo real y el virtual. Es importante que el hardware de interacción sea fácil de usar e intuitivo para que la experiencia sea exitosa (Lara, Santana, Lira, & Peña, 2019).

Para que el usuario tenga una experiencia con una inmersión total, dicha inmersión deberá cumplir con las siguientes cuatro condiciones asociadas al equipamiento (Escartín, 2000):

I. *Display* de campo visual total, que se puede conseguir por medio de una pantalla de ordenador, teléfono...
II. Seguimiento de las ubicaciones y las posiciones de los cuerpos de los participantes.
III. Seguimiento de los movimientos y las acciones de los participantes por parte de la computadora.
IV. Retardo de tiempo despreciable en la actualización del *display* con la retroalimentación de los movimientos de los cuerpos y las acciones de los participantes.

A continuación, se muestra una clasificación de las diferentes herramientas de hardware y software que existen y pueden utilizarse para nuestro proyecto.

1.4.1 Hardware

Hardware es una palabra de origen inglesa, que la Real Academia Española (R.A.E) define como “el conjunto de aparatos o dispositivos auxiliares e independientes de una computadora.” Es decir, son los dispositivos periféricos que nos envuelven en un mundo virtual e inmersivo.

A. Computadoras

Hoy en día, los ordenadores comunes que cualquier usuario tiene en casa podrían utilizarse para explorar mundos virtuales simples. La velocidad de procesamiento de la computadora marcará la complejidad que es capaz de asumir. Para determinar qué complejidad se puede incluir en un ambiente virtual, se necesita describir la velocidad

de la computadora. Con el objetivo de mantener una ilusión de realidad digna, la información espacial enviada a los dispositivos de visualización debe ser recalculada y actualizada más de veinte veces cada segundo.

Para el cálculo visual (gráficos 3D), una métrica conveniente es la cantidad de polígonos que una computadora puede dibujar en un segundo. Esto es así porque los gráficos se desarrollan a partir de polígonos simples. Con el objetivo de mostrar un ejemplo de la eficiencia necesaria, en los videojuegos más complejos como los de realidad virtual, debido a que la computadora debe dibujar dos vistas diferentes (una para cada ojo) por lo menos 20 veces cada segundo, es necesario dividir la cantidad de polígonos por segundo entre 40, para determinar la cantidad máxima de polígonos que pueden estar visibles en el mundo virtual. De manera que una computadora que puede dibujar 50000 polígonos por segundo será capaz de soportar un ambiente virtual que contenga un máximo de 1250 polígonos (Escartín, 2000).

En el caso que nos compete en este proyecto, los ordenadores más habituales con características básicas podrían servir para desarrollar juegos como los que veremos más adelante.

B. Dispositivos visuales

Los dispositivos visuales más comunes utilizados para jugar videojuegos son los monitores y los televisores. Estos dispositivos muestran los gráficos del juego y permiten al usuario interactuar con su interfaz. También se utilizan proyectores para mostrar imágenes de gran tamaño en una pantalla o pared. Los proyectores ofrecen una experiencia de juego más grande y envolvente

Existen otras formas alternativas para conseguir la sensación de inmersión virtual en los videojuegos, como son los sistemas de *display* binocular de RV montado en la cabeza (HMD), el uso de espejuelos para visión tridimensional (las gafas 3D empleadas ampliamente en el cine) y los sistemas de proyección 3D, llamados CAVE. Esta última se trata de una sala en forma de cubo en la que hay proyectores orientados hacia las diferentes paredes. Dependiendo del resultado que se quiera obtener se proyectará la imagen a todas o solo alguna de las paredes de la sala. Son más utilizados en museos o en instituciones didácticas.

C. Dispositivos auditivos

Del mismo modo que los dispositivos visuales se adaptaban a los ojos, los dispositivos auditivos, se adaptan a los dos oídos. Es por ello, que se hace uso de un sonido estereofónico. Se crean dos perspectivas de audio para crear un paisaje sonoro en las tres dimensiones (envolvente). Con ayuda de bocinas estereofónicas fijas, los sonidos derecho e izquierdo se mezclan, y ambos oídos reciben sonidos de ambas bocinas. Para conseguirlo, habitualmente se hace uso de altavoces (que ya vienen incorporados en el ordenador) o incluso en auriculares y cascos para una mayor inmersión auditiva.

1.4.2 Software

La R.A.E. define software como “el conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora”. Debemos diferenciar entre softwares destinados al montaje tridimensional de la escena y los destinados a la simulación:

A. Software para el montaje tridimensional

Dado que un ambiente virtual es un medio 3D, todos los objetos en un mundo virtual tienen que ser descritos de manera que puedan ser vistos desde cualquier ángulo. Una simple imagen del objeto no es suficiente. La verdadera geometría de los objetos tiene que especificarse usando un software de modelaje 3D, y luego exportarse hacia el ambiente virtual.

A continuación, se presentan algunos de estos softwares de montaje y creación 3D con sus respectivas características.

I. Blender.

Blender (Blender, 2023) (véase Figura 3) es una plataforma de creación 3D gratuita y de código abierto con la que se puede crear visualizaciones 3D, así como imágenes fijas, animaciones 3D, tomas VFX. Al ser una aplicación multiplataforma, Blender se ejecuta en sistemas Linux, macOS, así como en Windows. También tiene requisitos de memoria y unidad relativamente pequeños en comparación con otras plataformas de creación tridimensional.

Figura 3: Logo de Blender (Blender, 2023)

Posee una amplia variedad de herramientas que lo hacen adecuado para casi cualquier tipo de producción de medios. Profesionales, aficionados y los estudios de todo el mundo lo usan para crear animaciones, activos de juegos, gráficos en movimiento, programas de televisión, arte conceptual, guiones gráficos, comerciales y largometrajes.

Gracias a Blender se llega a conseguir resultados como el que se muestra en la Figura 4.

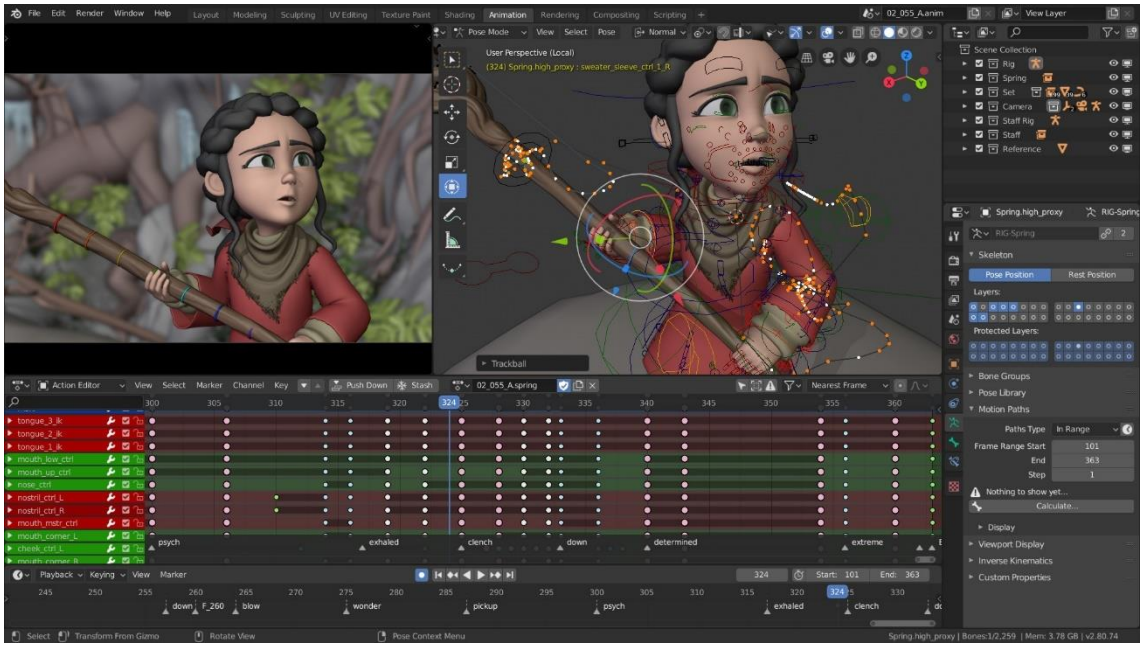

Figura 4: Ejemplo de la creación de una animación en Blender (Blender, 2023).

II. **Maya**.

Maya (Autodesk Maya, 2023) (véase Figura 5), es un conjunto de herramientas profesionales de animación, modelado, simulación y renderización en 3D, diseñado para crear personajes realistas y grandes efectos.

Figura 5: Logo de Autodesk Maya) (Autodesk Maya, 2023)

Es un programa de Autodesk, lo que significa que existen versiones de prueba gratuitas y versiones de pago que ofrecen mayores beneficios.

Con este software se puede desde dar crear criaturas fantásticas hasta paisajes y secuencias de batallas explosivas (véase Figura 6). Posee un conjunto de herramientas para dar vida a las películas, los programas de televisión y los videojuegos de animación con mejor acogida entre el público de hoy en día.

Figura 6: Ejemplo de las posibilidades que ofrece Maya (Autodesk Maya, 2023).

III. 3D Max

3D Max (AutoDesk 3D Max, 2023) (véase Figura 7) se utiliza para modelar, animar y renderizar personajes 3D detallados, diseños fotorrealistas y escenas complejas para cine y televisión, videojuegos y proyectos de visualización de diseños. Es un software profesional de modelado, renderización y animación 3D que permite crear mundos expansivos y diseños de máxima calidad.

Figura 7: Logo 3D Max (AutoDesk 3D Max, 2023).

Posee módulos de aprendizaje de todos los niveles que te enseña a usar para crear componentes y desarrollar mundos como el que se muestra a continuación (véase Figura 8).

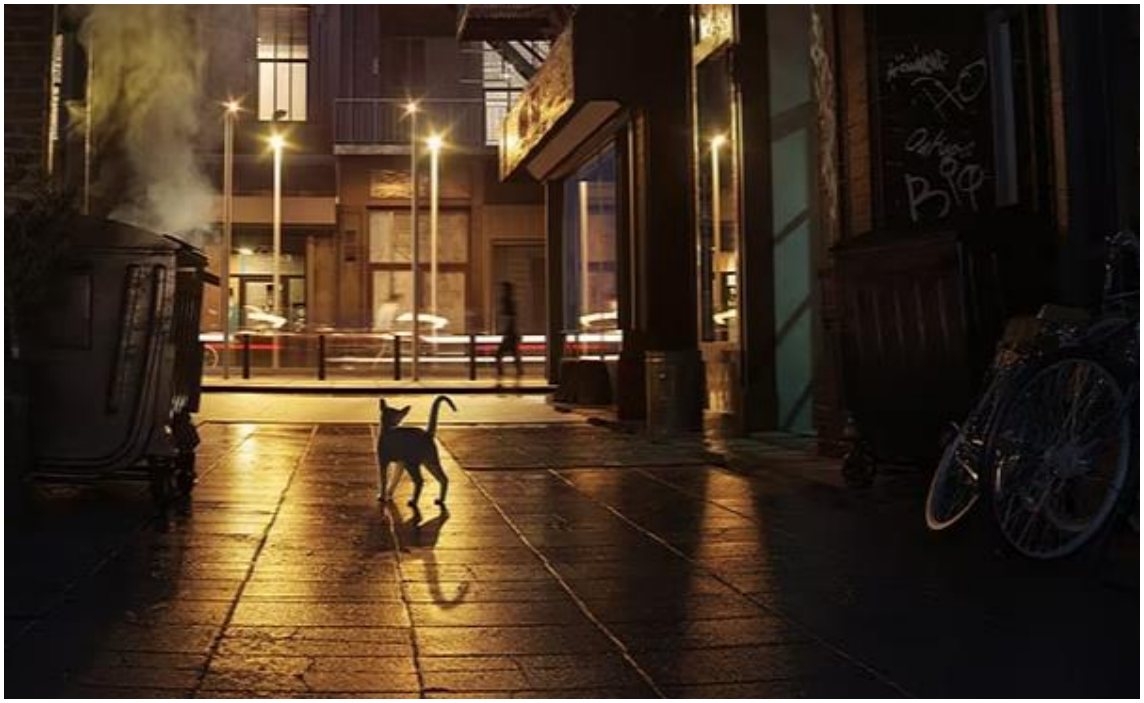

Figura 8: Captura de escena realizada con 3D Max. (AutoDesk 3D Max, 2023).

IV. Cinema 4D

Cinema 4D (Cinema 4D, 2023) (Figura 9) es una solución de software profesional de modelado, animación, simulación y renderizado 3D. Su conjunto de herramientas rápido, potente, flexible y estable hace que los flujos de trabajo 3D sean más accesibles y eficientes para el diseño, gráficos en movimiento, VFX, AR/MR/VR, desarrollo de juegos y todo tipo de profesionales de la visualización. Cinema 4D produce resultados asombrosos, ya sea trabajando solo o en equipo.

Figura 9: Logo Cinema 4D. (Cinema 4D, 2023)

Cinema 4D es ampliamente reconocido como uno de los paquetes 3D más fáciles y accesibles de aprender y usar. Para suavizar esa curva de aprendizaje, Maxon ofrece miles de tutoriales sobre Cineversity y cientos de consejos rápidos. El equipo de capacitación de Maxon también organiza seminarios web en vivo semanales y

mensuales para ayudar a los usuarios a tener éxito. En la Figura 10 podemos ver un ejemplo de las posibilidades de creación de escena que nos ofrece este software.

Figura 10: Captura de una escena creada a través de Cinema 4D. (Cinema 4D, 2023).

V. SolidWorks

SolidWorks (SolidWorks, 2023) es un software de diseño asistido por ordenador, que se utiliza para diseñar y modelar modelos mecánicos 2D y 3D. Se uso está restringido a usuarios de Microsoft Windows. El software posibilita la creación de modelos de piezas y conjuntos, y la obtención de planos técnicos y datos relevantes para la fabricación (véase Figura 11).

Figura 11: Logo de SolidWorks en la parte superior y ejemplos de diseño en la parte inferior

Su funcionamiento se basa en modernas técnicas de modelado mediante sistemas CAD. El proceso implica plasmar la concepción del diseñador en el programa, y simular digitalmente la construcción de la pieza o conjunto. De esta manera, se automatiza la extracción de información, incluyendo planos y archivos intercambiables.

Nuestra empresa colaboradora, FischerTechnik, nos ha facilitado los diseños de las máquinas que utilizaremos en el entorno virtual a través de este software. Es importante destacar que, aunque este programa no es específico de diseño de videojuegos y realidades virtuales, también se puede emplear en el desarrollo de las escenas siendo consciente de sus limitaciones.

SolidWorks ofrece una amplia variedad de herramientas y funciones para el diseño de piezas y ensamblajes, una interfaz fácil de usar, la posibilidad de colaboración en equipo, integración con otras herramientas y opciones de automatización que lo hacen una opción valiosa para diseñadores, ingenieros y fabricantes. Por ello este es el software elegido finalmente para parte del desarrollo de este proyecto

B. Software para simulación

Los videojuegos exigen un elaborado software de simulación que provoque en el usuario la sensación de estar inmerso en un mundo virtual, cercano o no a la realidad, donde percibe todo tipo de sensaciones y enseñanzas. Este software debe ser capaz de procesar las señales de los dispositivos de entrada para actualizar la escena de forma imperceptible para el usuario.

En muchos casos, esto es más complejo de lo que puede parecer a priori. En numerosas ocasiones no existe uno sino varios participantes a la vez que interactúan entre sí. Esto hace que el software modifique no solo la pantalla de un único usuario, sino que debe hacerlo para todos los participantes de forma coordinada.

Es decir, el software tiene que crear y mantener actualizada una base de datos (*Big Data*) que analiza y estudia de todos los objetos presentes en el mundo virtual, registra continuamente en la base de datos los cambios que se van produciendo y distribuye esta información a todas las computadoras participantes en el ambiente virtual.

Del mismo modo que se ha hecho antes, se presentan a continuación dos softwares de especial relevancia con los que realizar las simulaciones.

I. Unreal

Unreal Engine (Unreal Engine, 2023) (véase Figura 12) es un conjunto completo de herramientas de creación para el desarrollo de videojuegos, la visualización en la arquitectura y la automoción, la creación de contenido para películas y programas de televisión lineales, la producción de eventos retransmitidos y en directo, la formación y la simulación, y otras aplicaciones en tiempo real.

Figura 12: Logo de Unreal Engine (Unreal Engine, 2023).

Unreal Engine permite dar vida a increíbles experiencias en tiempo real utilizando la herramienta de creación 3D en tiempo real más avanzada del mundo. Su uso no se limita a los desafíos más exigentes, sino que también puede utilizarse para los primeros proyectos. Los recursos son gratuitos y accesibles.

Cuenta con cientos de horas de contenido educativo en línea gratuito, una amplia biblioteca de seminarios web y opciones de formación con instructores.

II. Unity

Unity (Unity, 2023) (véase Figura 13) es lo que se conoce como un motor de desarrollo o motor de juegos. El término motor de videojuego, *game engine*, hace referencia a un software que tiene una serie de rutinas de programación que permiten el diseño, la creación y el funcionamiento de un entorno interactivo; es decir, de un videojuego.

Figura 13: Logo de Unity. (Unity, 2023)

Unity, antes llamado Unity 3D, es una herramienta que permite crear videojuegos para diversas plataformas (PC, videoconsolas, móviles, etc.) mediante un editor visual y programación vía *scripting*, y pudiendo conseguir resultados totalmente profesionales. Puede usarse junto con Blender, 3D Max, Maya, Softimage, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance. Los cambios realizados a los objetos creados con estos otros softwares se actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volverlos a importar manualmente (Master.D, 2023).

1.4.3 Elección

Tras esta breve introducción se puede destacar que con la ayuda de programas como Blender o Maya se crea el cuerpo de las estructuras 3D y, finalmente, por medio de Unreal o Unity se le da vida a un mundo virtual.

En conclusión, ambas opciones tienen sus pros y contras, pero en términos de facilidad de aprendizaje y accesibilidad, Unity es una opción más viable para la mayoría de los desarrolladores y empresas. Sin embargo, si el proyecto requiere gráficos avanzados y efectos visuales, Unreal Engine puede ser la mejor opción. Al final, la elección del motor dependerá del proyecto específico y de las necesidades y habilidades del equipo de desarrollo.

Para tomar una mejor decisión, se consultó a la empresa Apolo Estudio Creativo, especializada en el mundo del metaverso. Tras una reunión presencial, se determinó que la herramienta más recomendable para realizar nuestro proyecto era Unity, con la que podríamos empezar desde cero e ir evolucionando hasta el punto de cumplir nuestros objetivos.

Por todo esto, en nuestro caso, se ha decidido que el motor de desarrollo sea Unity, ya que nuestro proyecto tiene objetivos relativamente sencillos y gráficos simples. Además, Unity ofrece una amplia gama de recursos para el aprendizaje eficiente, como una gran cantidad de foros accesibles y cursos gratuitos que se adaptan al nivel de conocimiento del usuario, desde lo más básico hasta módulos avanzados como los de realidad virtual.

Es importante destacar que este proyecto tiene como línea futura de desarrollo la implantación de juegos en realidad virtual y aumentada, por lo que comenzar el desarrollo en esta plataforma será beneficioso para investigaciones futuras. Por tanto, este proyecto será la base de desarrollo para futuros proyectos relacionados. Unity también cuenta con un gran contenido multimedia para aprender la interfaz de usuario y la programación de *scripts* de manera sencilla.

1.5 Lean Manufacturing y sus herramientas

Lean Manufacturing es un enfoque de producción que se centra en la eliminación de desperdicios y la mejora continua del proceso productivo para aumentar la eficiencia y la productividad.

La filosofía de Lean Manufacturing se basa en el principio de que todo lo que no agrega valor al producto final se considera un desperdicio. Por lo tanto, el objetivo es reducir y eliminar los desperdicios en el proceso productivo, que pueden incluir el tiempo de

espera, el transporte, el exceso de inventario, la sobreproducción, los defectos y los movimientos innecesarios.

1.5.1 Método Kanban

El método Kanban es un enfoque para la gestión de la producción y el flujo de trabajo que se originó en la industria manufacturera japonesa. El objetivo principal del método Kanban es maximizar la eficiencia, la calidad y la satisfacción del cliente al minimizar los desperdicios, la variabilidad y el tiempo de espera. La palabra Kanban está formada por las palabras "*kan*" que significa visual y "*ban*" que significa tarjeta o tablero. La idea surge a raíz de la metodología *Lean Manufacturing*, la cual fue desarrollada por Toyota, para mejorar la producción basándose en técnicas como el justo a tiempo (*Just in Time*, JIT). (Arango Serna, Campuzano Zapata, & Zapata Cortes, 2015).

Kanban promueve la calidad perfecta, la minimización del despilfarro y la mejora continua en el proceso de producción o desarrollo. La flexibilidad permite priorizar y condicionar tareas según necesidades puntuales. Se construye y mantiene una relación a largo plazo con proveedores. La metodología Kanban se basa en un conjunto de seis reglas (Figura 14) y utiliza tarjetas de tareas en un tablero visual para mejorar la organización del trabajo.

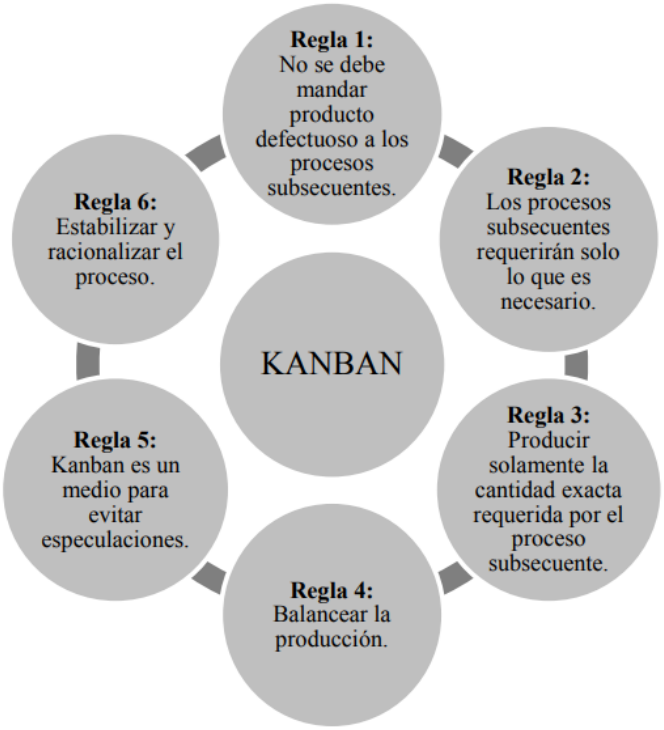

Figura 14: Reglas del método Kanban (Arango Serna, Campuzano Zapata, & Zapata Cortes, 2015).

Esta metodología se basa en el uso de las tarjetas o Kanban, que son una herramienta que permite llevar un registro visual del proceso. Normalmente, en dicha tarjeta se escribe la información más relevante que un trabajador puede necesitar para llevar a cabo la tarea: el nombre, el tiempo estimado, etc. Estas tarjetas están colocadas sobre

un estado que indican en qué etapa del proceso se encuentra dicha tarea (pendiente, en proceso o finalizada) (Arango Serna, Campuzano Zapata, & Zapata Cortes, 2015).

El método Kanban permite eliminar gran parte de la burocracia, ya que estas tarjetas de tareas son compartidas con todo el equipo. Esto permite mejorar la comunicación y la colaboración dentro de este organismo, además de aumentar la eficiencia de los procesos a los que se le aplica. El sistema Kanban busca garantizar una producción constante y sostenible para evitar la acumulación de productos terminados, así como cuellos de botella y demoras en la entrega de pedidos (Castellano Lendínez, 2019).

1.5.2 Cinco Ceros

La teoría de los cinco ceros podría considerarse como un resumen de los objetivos del Just in Time. Esta filosofía consiste principalmente en "optimizar un sistema de producción de manera que se logre producir los elementos necesarios en las cantidades necesarias y en el momento necesario". (Lean Manufacturing Web, 2023).

Es una teoría que defiende cero defectos, cero averías, cero burocracias, cero plazos y cero stocks. En definitiva, promueve la eliminación de toda actividad que no tenga valor añadido para el producto final. Aunque a menudo se agregan otros objetivos como cero accidentes, cero desprecios por las capacidades del personal y cero tiempos de espera para sacar un producto al mercado. (Yacuzzi, Fajntich, & Pía Romeo, 2013).

- *Cero defectos.* Se busca una mayor productividad, por eso al minimizar los defectos aumentamos la producción.
- *Cero averías.* Las averías provocan el incumplimiento de los objetivos, por lo que siempre será mejor buscar las alternativas necesarias para evitar el retraso de los productos o servicios.
- *Cero stock.* El almacenamiento de producto cuesta dinero, por lo que se busca que no exista este almacenamiento. Es decir, que la materia prima que llega del proveedor llegue en el momento justo y el producto acabado entregarlo al cliente nada más terminar su producción.
- *Cero plazos.* "Quien golpea primero, ..., golpea dos veces". Se debe reducir el número de ciclos de fabricación, reduciendo el nivel de stock y consiguiendo mayor flexibilidad para adaptarse a los cambios que el cliente demanda.
- *Cero burocracia.* Se busca la simplicidad, es decir, eliminar cualquier burocracia disminuyendo los tiempos de toma de decisión o reduciendo las actividades administrativas.

1.5.3 Las 5S

Las 5S son una práctica de mejora continua que busca involucrar a todos los miembros de una organización para mejorar la calidad del trabajo y el entorno laboral, y así aumentar la productividad y la satisfacción del cliente. Las 5S vienen de cinco palabras

japonesas que inician con la letra S (Figura 15): Seiri (selección), Seiton (orden), Seiso (limpieza), Seiketsu (normalización) y Shitsuke (disciplina). (Arrieta, 1999).

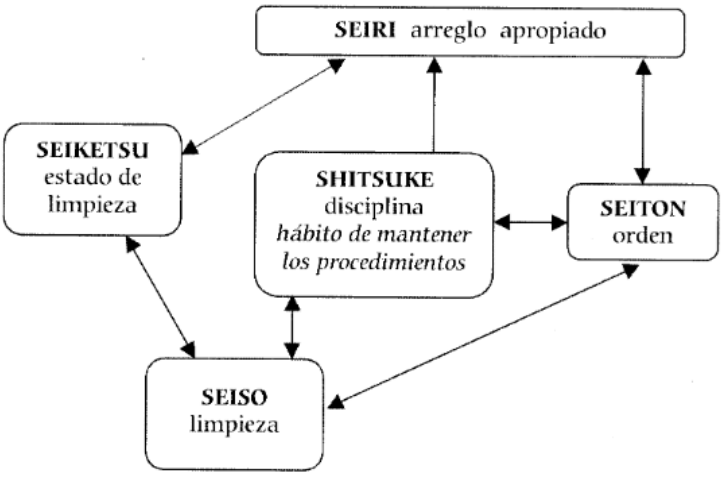

Figura 15: Filosofía de 5S. (Arrieta, 1999)

La implementación de las 5S en el lugar de trabajo implica la realización de tareas de manera segura para el trabajador, la maquinaria y el producto. Además, el mantenimiento continuo de las 5S contribuye a un mejor desempeño y a la adhesión de las normas y regulaciones de seguridad en el entorno laboral. La filosofía de las 5S se enfoca en mejorar la eficiencia y uniformidad del trabajo en los centros laborales, y esto se logra a través de una fábrica ordenada y limpia.

Cuando una fábrica mantiene un ambiente ordenado y limpio, se asegura una producción con menos defectos, un cumplimiento más efectivo de los plazos, un ambiente de trabajo más seguro y productivo, y una mayor facilidad para llevar a cabo tareas de mantenimiento. Además, se logra una mayor calidad del producto y un ambiente de trabajo más motivante para el personal. La organización permite una mayor diversificación de la producción y el crecimiento corporativo, y brinda una mayor fiabilidad y confianza a los clientes (Arrieta, 1999).

1.6 La fábrica de industria 4.0 de FischerTechnik

Como ya vimos en un apartado anterior, la cuarta revolución industrial (Industria 4.0) está basada en la digitalización. Mediante la fábrica de aprendizaje 4.0 de FischerTechnik se pueden simular, aprender e implementar estas actividades de digitalización a pequeña escala, antes de aplicarlas a una escala mayor en cualquier empresa.

Esta maqueta (Figura 16) es una herramienta útil para la investigación, la enseñanza y el desarrollo en universidades, empresas y departamentos de informática. La fábrica

consta de varios módulos, como una estación de entrada-salida, un robot manipulador, una estación de almacenamiento y traslado, una multiestación de procesamiento, una cinta de clasificación, un sensor medioambiental y una cámara giratoria.

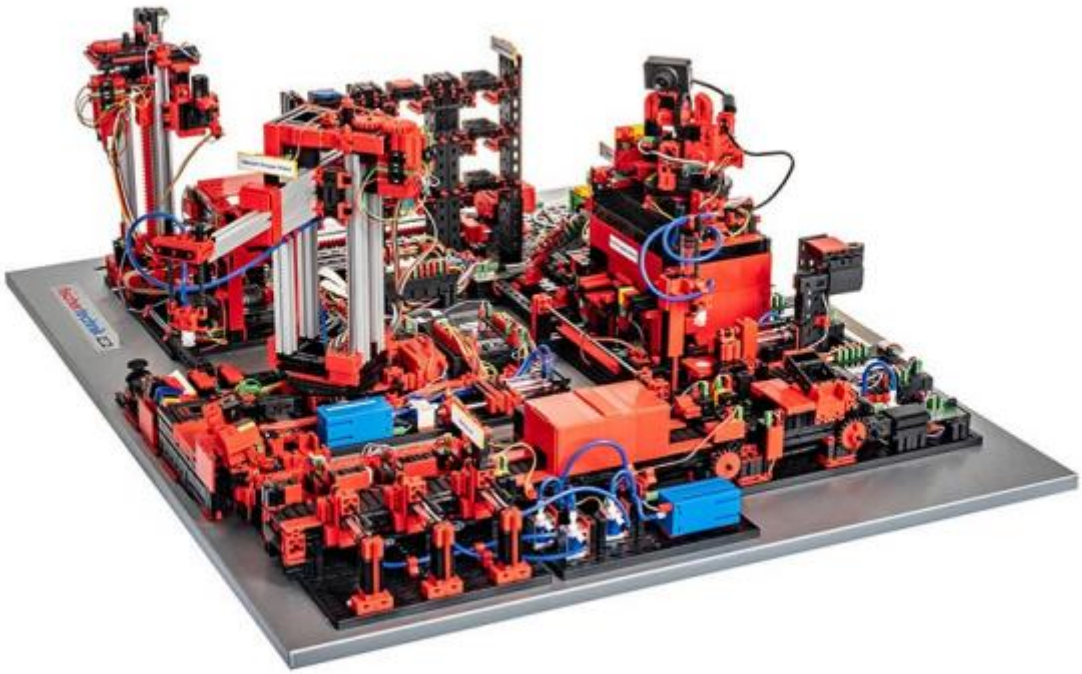

Figura 16: Vista global de la fábrica de aprendizaje 4.0 de FischerTechnik (FischerTechnik, 2022)

El proceso comienza cuando una pieza entra en la fábrica a través de la estación de entrada, y es recogida por el robot manipulador que la lleva al almacén. Si se realiza un pedido, la pieza es transportada a la estación de multiprocesamiento donde se somete a una operación de cocción y fresado. Luego, las piezas se ordenan por color en la cinta de clasificación y se transportan al punto de expedición.

La fábrica se controla mediante un panel de control, una cámara giratoria y una tecnología NFC (*Near Field Communication*) que rastrea las piezas de trabajo individuales. Estas piezas se localizan a través de un número de identificación (*ID*) lo que hace posible seguir la pieza y conocer su estado en todo momento. Los controladores *TXT Controller de FischerTechnik* se utilizan para comunicarse entre sí mediante el protocolo de mensajes MQTT (*Message Queuing Telemetry Transport*), que hace posible el intercambio de datos en forma de mensajes entre los dispositivos.

2 FISCHERTECHNIK VIRTUAL

Ante tanta cantidad de oportunidad de desarrollo que ofrece una plataforma como Unity junto a la programación de C# en Visual Studio, se ha realizado un primer juego basado en la maqueta de industria 4.0 desarrollada por FischerTechnik. En primer lugar, se indican unos conceptos básicos relacionados con el control y manejo de la interfaz de Unity. Posteriormente, se explica cómo se ha avanzado en la implantación y puesta en marcha de una simulación en tres dimensiones de dicha maqueta.

2.1 Conceptos básicos de Unity

Antes de la realización de nuestro proyecto inicial hemos debido familiarizarnos con el entorno y la programación del programa principal de uso: Unity. Con ayuda de manuales de usuario, cursos oficiales, foros, vídeo tutoriales y la página oficial de ayuda que ofrece dicho programa, se han realizado pequeños proyectos con los que aprender las herramientas necesarias. Hemos utilizado paquetes ya creados por otros usuarios, de uso público para poder dirigir el aprendizaje directamente hacia el programa de Unity, sin pasar por programas de diseño. Estos juegos de aprendizaje se pueden ver mucho más desarrollados en el anexo.

2.1.1 Unity Hub

Antes de empezar, es importante destacar cómo configura el entorno de nuestro videojuego. Este será común para cualquiera de nuestros proyectos. Se inicia la configuración a través no del programa Unity sino un módulo adyacente a este llamado Unity Hub. Para empezar, debemos crear un nuevo proyecto: *New Project*. En él le daremos el nombre y la ubicación donde guardaremos los datos relativos a este. Además, deberemos seleccionar el tipo de juego que vamos a crear. Existen multitud de opciones, entre las que destacan:

- 3D: método más general para la creación de renderizado en tres dimensiones
- 2D: método general para creación en dos dimensiones
- *Runner Game*: podemos encontrar tutoriales y contenido de muestra para crear juegos de tipo “corredor”.
- VR: utilizado para el inicio rápido de aplicaciones de realidad virtual donde encontraremos escenas de muestra y paquetes de configuración recomendables para estos casos.
- *First Person*: controlador de personajes con una perspectiva en primera persona.
- *Third Person*: controlador de personajes con una preconfiguración en tercera persona.
- *Karting Microgame*: utilizado para juegos de carreras de karts donde encontramos ayudas gracias a escenas y guiones precargados.

Para nuestro proyecto escogeremos la opción más general para el renderizado de juegos en tres dimensiones, donde podremos empezar desde cero (véase Figura 17). La velocidad en cargar el proyecto vacío seleccionado dependerá del tipo de computadora con la que se realice.

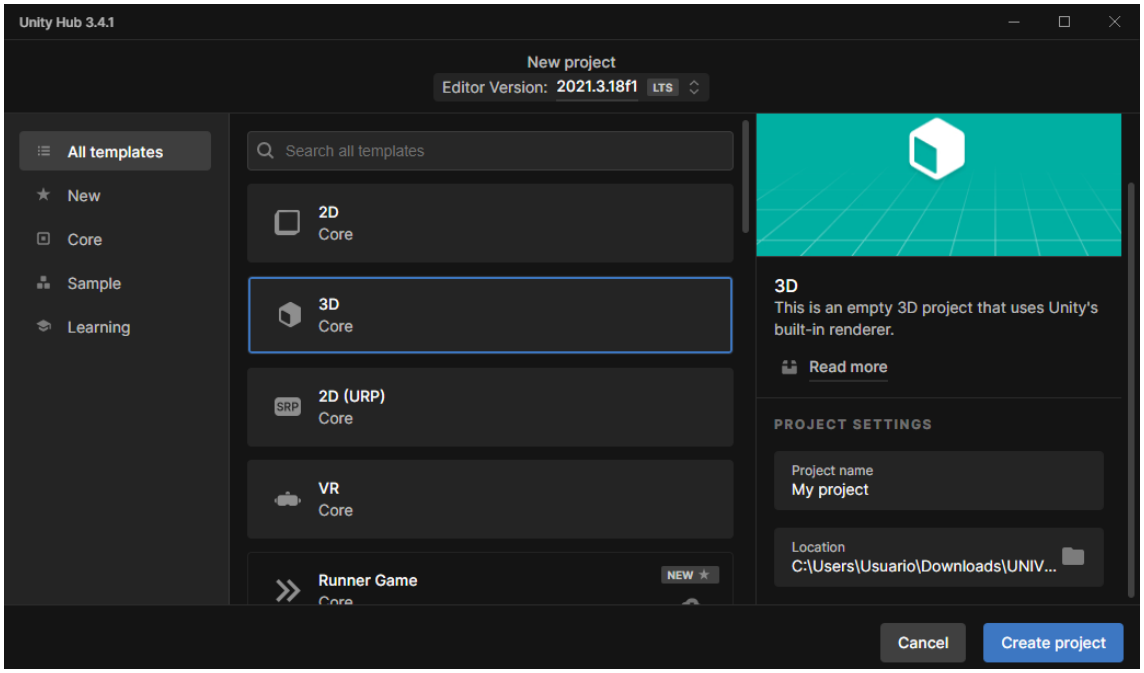

Figura 17: Creación de un nuevo proyecto a través de Unity Hub.

Cabe destacar que los juegos realizados para el aprendizaje (ver anexo) son cada vez de mayor complejidad no solo de programación sino de control del entorno desde Unity. Esto nos ayuda con la comprensión progresiva de los conceptos más generales a los más específicos.

2.1.2 Entorno de Unity

Cuando abrimos por primera vez el programa nos encontramos con un aspecto inicial de la interfaz gráfica, que no necesariamente debe ser con la que trabajaremos. El aspecto es muy subjetivo y el usuario puede colocar las pantallas a su gusto (véase la Figura 18). A continuación, se muestran las diferentes ventanas de las que se harán uso, así como una posible configuración que se ha utilizado a lo largo del proyecto:

- Panel de jerarquía (*Hierarchy*): en este podremos ver un árbol jerárquico en el que aparecerán todos los componentes que tiene la escena.
- Proyectos (*Project*): pestaña de organización relativa a todo el contenido del proyecto, aparezca en ese instante en la escena o no.
- Consola (*Console*): aquí podremos ver todas las interacciones que el programa comunica al usuario (errores de compilación, de ejecución o de diseño).
- Inspector (*Inspector*): son las propiedades que tiene cada componente, donde accederemos para modificar las características de estos.
- Pantalla 3D de movimiento (*Scene*): es el escenario en el que conformaremos nuestro juego. Ajustaremos los diseños, pantallas, prefabs, …

- Visualizador de cámara (*Game*): esta será la ventana que el usuario jugador verá durante el juego.

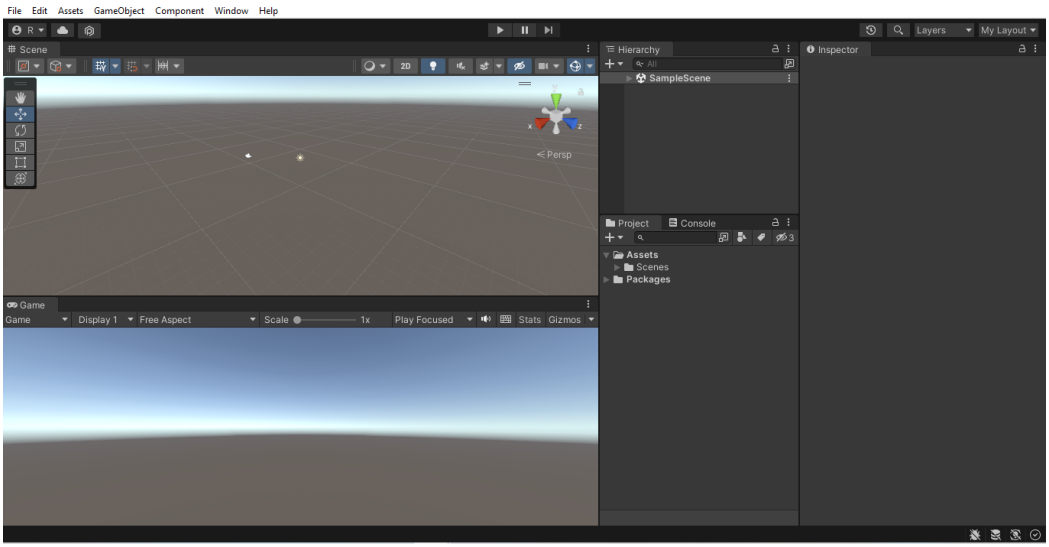

Figura 18: Proyecto vacío en Unity con las diferentes pantallas. Con una configuración personalizada (My Layout).

2.1.3 Programación de Unity

La programación en Unity se realiza principalmente en C#, aunque también es posible usar otros lenguajes como JavaScript o Boo. C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft, que se caracteriza por ser seguro, eficiente y fácil de aprender.

Unity proporciona su propio IDE (*Integrated Development Environment*), llamado "Unity Editor", que incluye un editor de código con características como el resaltado de sintaxis, autocompletado y depuración de código. Sin embargo, los desarrolladores pueden utilizar cualquier otro editor de código compatible con C# y cargar los *scripts* en Unity. Esta es la opción por la que se ha preferido optar, ya que se ha llevado a cabo a través de Visual Studio 2022. Esta decisión se sustenta en base al gran contenido online que existe relacionado con este último.

Unity también cuenta con una gran cantidad de recursos en línea para los desarrolladores que desean aprender a programar en C# para Unity, incluyendo tutoriales, documentación y una comunidad activa que comparte sus conocimientos y experiencias en foros y redes sociales[1].

[1] Se puede consultar el anexo de este documento relacionado con la programación para obtener los conocimientos más básicos de programación con C#.

2.2 Desarrollo del primer juego basado en la fábrica I40 de FischerTechnik

El propósito de este juego es brindar al usuario una experiencia virtual de recorrido por una fábrica de la industria 4.0, específicamente por la Fábrica de Aprendizaje 4.0 de FischerTechnik. Para lograr este objetivo, se inició una comunicación por correo electrónico con la empresa, solicitando permisos y planteando la idea de crear una fábrica virtual a escala real. La intención es proporcionar a estudiantes y trabajadores nuevos una perspectiva diferente de la empresa, en comparación con lo que se ofrece en la maqueta física.

Es importante destacar que este capítulo es una aproximación y no se busca un alto nivel de detalle o funcionalidad. El propósito es ofrecer un tour en 3D del entorno de la empresa a escala real. La empresa proporcionó una serie de archivos con los que se podía trabajar de manera limitada, los cuales se encontraban en tres formatos diferentes diseñados en SolidWorks: .easm, .sldasm y .step.

Inicialmente, se pensó que la empresa proporcionaría archivos individuales de cada máquina; sin embargo, se descubrió que no era así al abrirlos en Catia V5, el programa de diseño asistido por ordenador. Se recibieron un total de cinco archivos en los formatos mencionados, de los cuales uno contenía un único módulo y los otros archivos incluían conjuntos de máquinas. Todos los archivos compartían la característica de no poder ser modificados, lo que impedía el acceso libre a su contenido. Afortunadamente, uno de los archivos contenía la fábrica completa, por lo que se decidió trabajar con él.

No obstante, al intentar importar el archivo a Unity, nos encontramos con un problema de compatibilidad. Tras investigar los formatos válidos para un programa como Unity, se determinó que era necesario transformar los archivos de formato .step a formato .obj. Para llevar a cabo esta tarea, se utilizó la plataforma en línea llamada Imagetostl[2], que permite realizar múltiples cambios de formatos de archivos de manera gratuita y sin necesidad de registrarse (Google Analytics, 2023).

Una vez seleccionado el archivo y transformado a las condiciones adecuadas para su uso en Unity, se importó para instanciarlo en la escena. Además de la empresa, se crearon los límites de movilidad a través de planos que funcionaron como paredes y suelo. Por último, se añadió un personaje a la escena. Ver Figura 19.

[2] https://imagetostl.com/es/convertir/archivo/step/a/obj

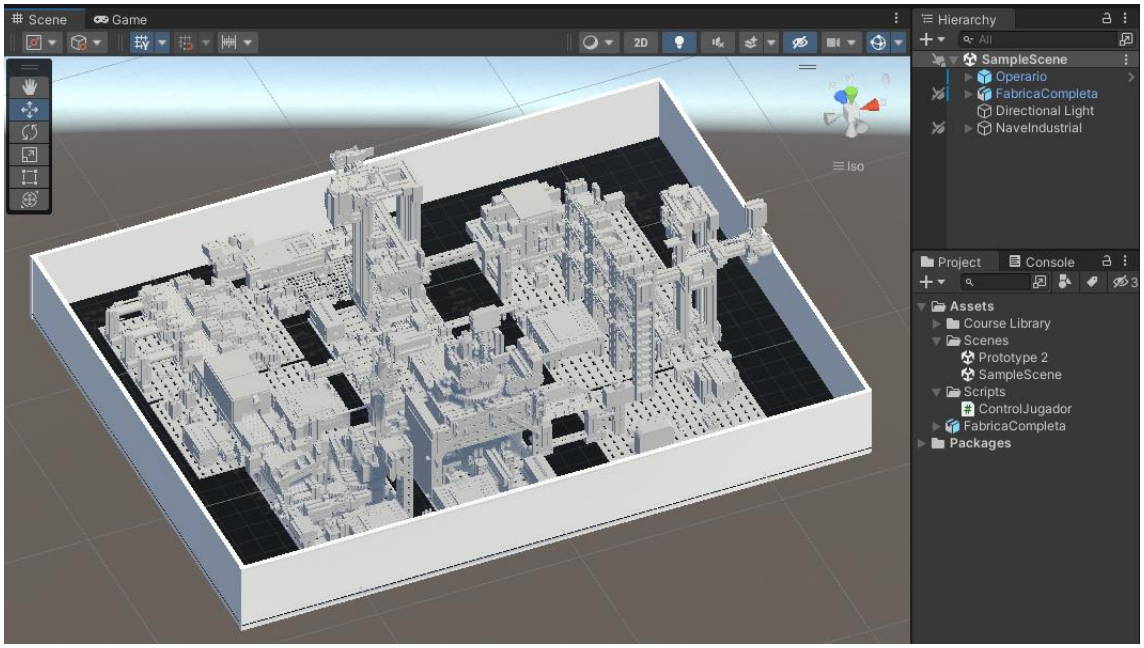

Figura 19: Escena de la fábrica de FischerTechnik virtual

Para dar vida al jugador, se debe manipular el Inspector. Se le añaden un total de tres nuevos componentes: Rigidbody (para concederle propiedades físicas), Box Collider (para poder programar las colisiones) y, por último, un *script* llamado ControlJugador. Este *script* ControlJugador (véase Código 1) tiene como objetivo controlar el movimiento del jugador en el juego a través del ratón y el teclado.

Código 1: Control del jugador para el juego de industria 4.0 de FischerTechnik.

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

public class ControlJugador : MonoBehaviour
{
    public float sensibilidadRaton = 50.0f;
    public float velocidad = 10.0f;
    Camera camara;
    public float rotacionCabeceo;
    Rigidbody rbJugador;

    public float rangoDerechaX = 148;
    public float rangoIzquierdaX = -153;
    public float rangoFrontalZ = -347;
    public float rangoAtrasZ = -577;

    public float amortiguacion= 0.5f;


    void Start()
    {
        //Inicializamos la variable y la asociamos al objeto hijo
        cámara = GetComponentInChildren<Camera>();
         // Iniciamos la variable a la posicion actual del eje x
```

```
            rotacionCabeceo = camara.transform.rotation.x;
        rbJugador = GetComponent<Rigidbody>();
            // Hacer que desaparezca el cursor. Para salir del modo juego esc o ctrol + P
        Cursor.lockState = CursorLockMode.Locked;
    }


    void Update()
    {
        //Llamada a las funciones
        MovimientoCabeza();
        MoverCuerpo();
    }
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.name == "Pared")
        {
            transform.position = transform.position - collision.contacts[0].normal *
                        collision.relativeVelocity.magnitude * Time.deltaTime;
            // Aplica una pequeña cantidad de amortiguación al movimiento del jugador que
                // evita el giro incontrolado.
            GetComponent<Rigidbody>().velocity *= (1 - amortiguacion);
        }
    }

    void MovimientoCabeza()
    {
        float xR = Input.GetAxis("Mouse X") * sensibilidadRaton * Time.deltaTime;
        float yR = Input.GetAxis("Mouse Y") * sensibilidadRaton * Time.deltaTime;

            // Colocamos un - a pesar de que lo estoy sumando (mov contrario)
        rotacionCabeceo = rotacionCabeceo - yR;
            // Propiedad que nos permite limitar el valor de la rotacionCabeceo entre esos
            // valores
        rotacionCabeceo = Mathf.Clamp(rotacionCabeceo, -90, 90);

        transform.Rotate(0, xR, 0);

        // Giro de la cabeza
            // Otra manera de realizar un giro a traves del un metodo de quaternios
        camara.transform.localRotation = Quaternion.Euler(rotacionCabeceo, 0, 0);
            // Destacable el método localRotation para usar los ejes locales
    }

    void MoverCuerpo()
    {
        float movimientoLateral = Input.GetAxis("Horizontal");
        float movimientoAvance = Input.GetAxis("Vertical");

        // Movimiento del cuerpo

        transform.Translate(Vector3.right * velocidad * movimientoLateral *
            Time.deltaTime);
        transform.Translate(Vector3.forward * velocidad * movimientoAvance *
            Time.deltaTime);

    }
}
```

Como puede verse, se han utilizado líneas de comentario dentro del propio código con el fin de facilitar su comprensión. Además, dado que este juego solo lleva un único *script* asociado al jugador, este se ha realizado de manera un poco más completa de lo

normal. Al comienzo, se declaran unas variables públicas, lo que le permite al usuario que está programado desde Unity manipularlo en cualquier momento durante la simulación.

Las variables más importantes son: "sensibilidadRaton", que define la velocidad a la que se mueve la cabeza del jugador al mover el ratón y la variable "velocidad", que define la velocidad de movimiento del jugador. Por último, me gustaría resaltar que hay cuatro rangos de movimiento que pretenden limitar el movimiento del jugador dentro de ciertos límites. Sin embargo, si analizamos en global el código podemos ver que no se han llegado a utilizar. He preferido dejarlos con el fin de mostrar que existen maneras (menos eficientes) de limitar un área de movimiento del jugador. En los puntos posteriores de este mismo documento se podrá ver que existen otras formas más eficientes de limitar una escena sin necesidad de colocar cifras numéricas.

En el método Start() se inicializan las variables "camara" y "rbJugador" con referencias a los componentes de la cámara y el Rigidbody del jugador. También se establece el modo de bloqueo del cursor para que desaparezca durante el juego. Esto último con el fin de que el usuario tenga una experiencia más inversiva.

Tenemos el metodo Update*()*, que llama a su vez a otras dos funciones que nos sirven para actualizar cada el movimiento de la cabeza del jugador y el movimiento del cuerpo.

El método OnCollisionEnter() se llama cuando el jugador choca con una pared. Si el objeto de colisión tiene el nombre "Pared", se aplica una pequeña cantidad de amortiguación al movimiento del jugador para evitar un giro incontrolado. Además, el jugador es empujado hacia atrás en la dirección opuesta a la normal de la colisión. Esta es la manera más eficaz de limitar una escena y no la que se presenta antes a través de cifras numéricas.

La función "MovimientoCabeza()" se encarga de controlar el movimiento de la cabeza del jugador en función de la entrada del ratón. Primero, utiliza la función "*Input.GetAxis()*" para obtener la posición del ratón en el eje X e Y. Luego, multiplica estas posiciones por la sensibilidad del ratón y el tiempo desde el último fotograma para obtener la cantidad de rotación que se aplicará.

A continuación, la variable "rotacionCabeceo" se actualiza con el movimiento en el eje Y (inversamente ya que se resta). Luego se utiliza la función "*Mathf.Clamp()*" para asegurarse de que la rotación de la cabeza esté dentro del rango permitido de -90 a 90 grados.

Finalmente, se utiliza la función "*transform.Rotate()*" para aplicar la rotación en el eje X al transform del objeto jugador. También se utiliza "*camara.transform. localRotation*" para aplicar la rotación de la cabeza en el eje Y a la cámara, con el método "*Quaternion.Euler()*" para convertir la rotación a un quaternion.

La función "MoverCuerpo()" está encargada de mover el objeto jugador en función de las entradas del usuario en los ejes horizontal y vertical. Primero utiliza "*Input.GetAxis()*" para obtener la cantidad de movimiento en cada eje.

Luego, se utiliza el método *transform.Translate()* para realizar la traducción del objeto en los ejes X y Z (en el plano horizontal). Para ello, se multiplica la velocidad de movimiento (velocidad) por la entrada de movimiento lateral o vertical (movimientoLateral o movimientoAvance, respectivamente) y se multiplica todo esto por *Time.deltaTime*, que es una medida del tiempo que ha pasado desde la última actualización del fotograma (*frame)*.

Finalmente se habrá obtenido el objetivo que se buscaba, es decir, se tendrá un juego funcional en el que el usuario podrá disfrutar de conocer de manera virtual, la empresa a tamaño real. Para ello podrá moverse libremente por toda el área acondicionada para ello. A continuación, se presenta una serie de figuras (Figura 20, Figura 21 y Figura 22) en las que se muestran diferentes escenas durante la simulación.

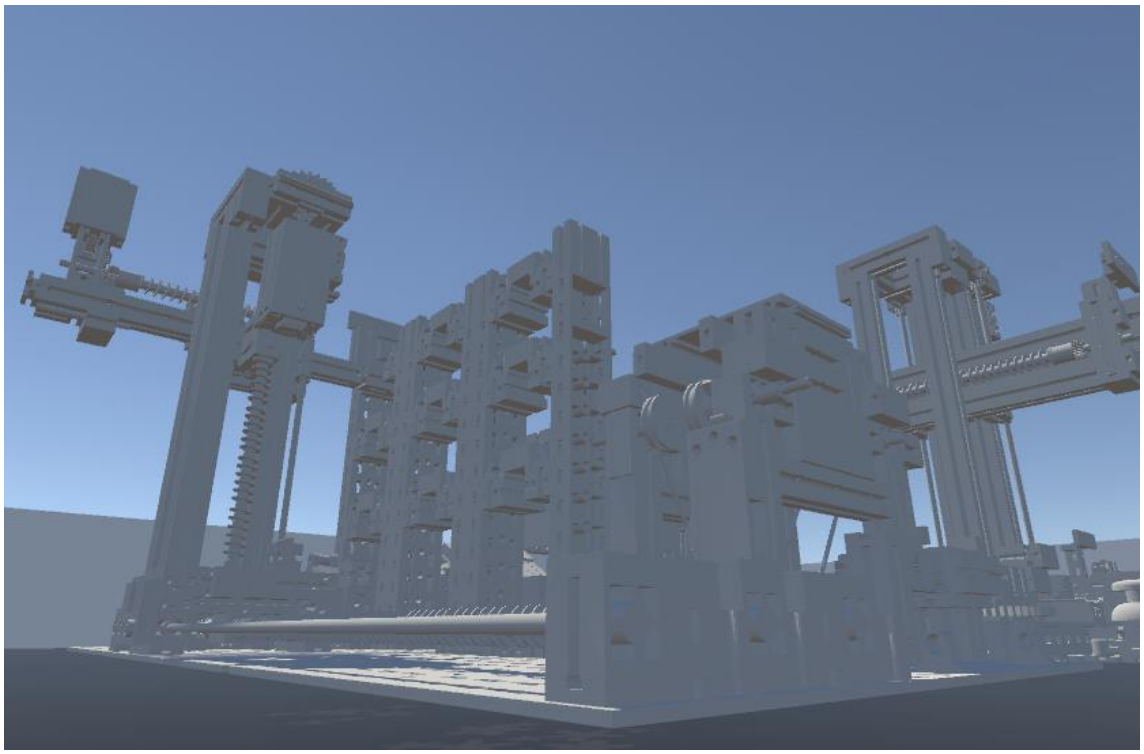

Figura 20: Visión 1 durante la simulación de la empresa FischerTechnik

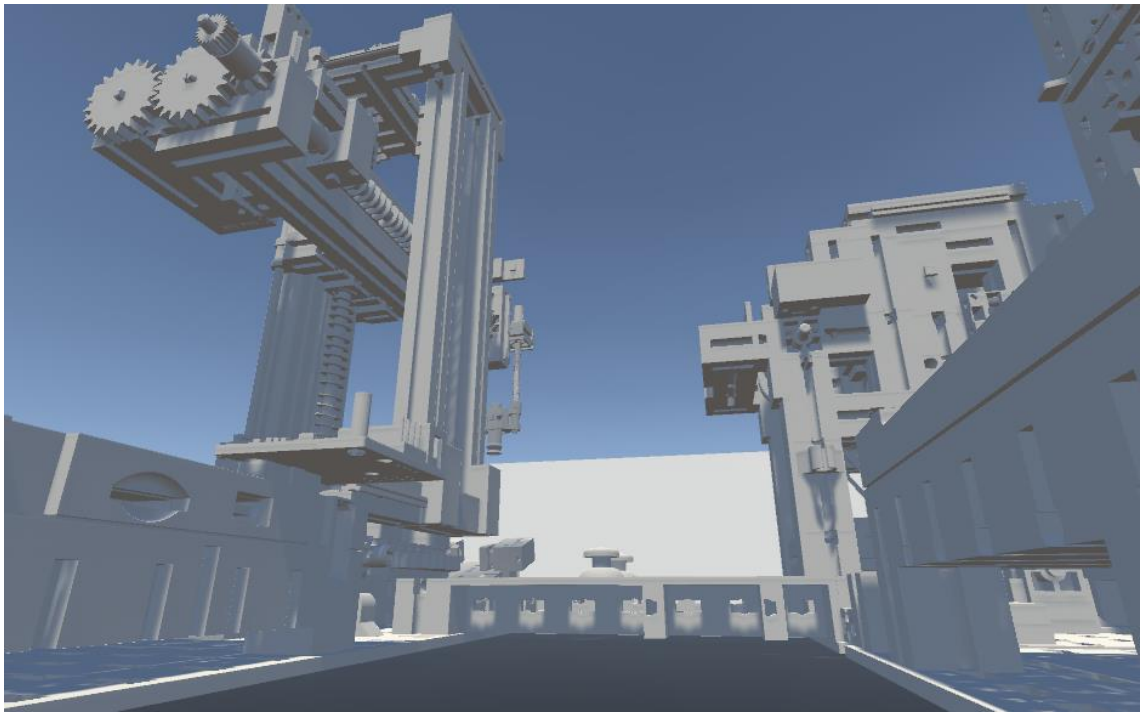

Figura 21: Visión 2 durante la simulación de la empresa FischerTechnik

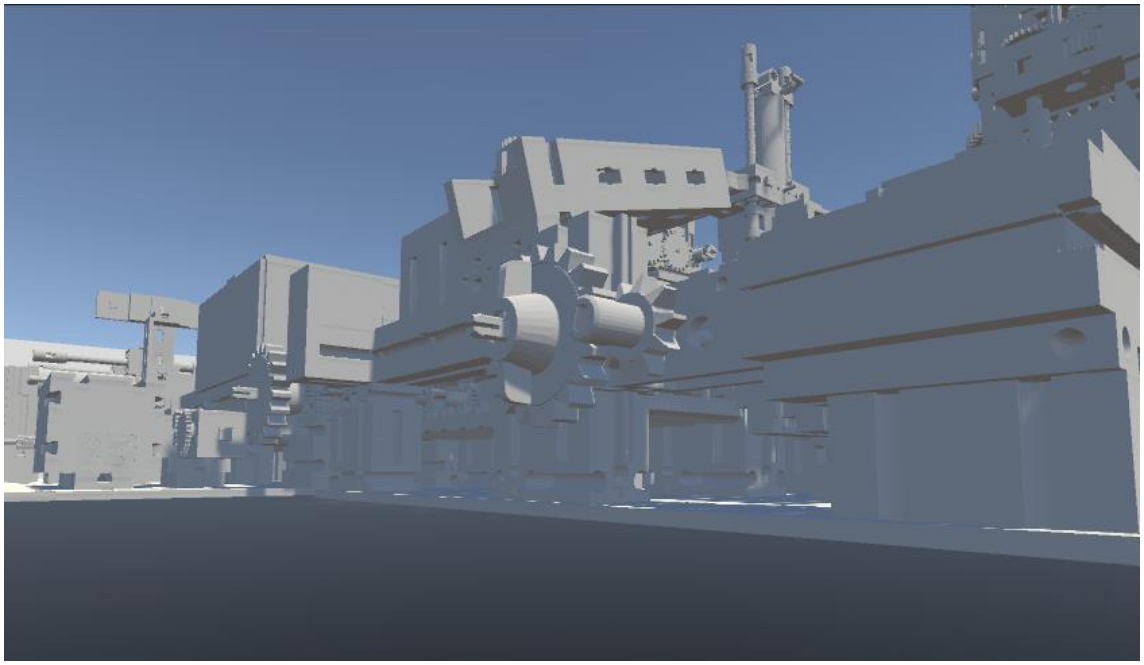

Figura 22: Visión 3 durante la simulación de la empresa FischerTechnik

3 DESARROLLO DE MINIJUEGOS LEAN

Los “minijuegos” que se han creado servirán como herramienta de apoyo al aprendizaje de algunos conceptos muy básicos de cómo debe una empresa organizar un buen trabajo. Es importante destacar que estos juegos no son complejos porque la intención es que sirvan única y exclusivamente para la ilustración gráfica y práctica de conceptos básicos. A lo largo de las diferentes pantallas que podemos encontrar en el juego, se podrán ver reflejadas ideas relacionadas principalmente con el método Kanban, la teoría de los cinco ceros y las 5S.

3.1 Diseño y justificación de los minijuegos

Antes de emprender cualquier proyecto, es fundamental elaborar una idea preliminar de lo que se desea llevar a cabo. En este sentido, el primer paso consistió en identificar los conceptos clave que deben ser reflejados en los minijuegos. Estas ideas deben ser claras y comprensibles para poder crear ejemplos sencillos de aplicación. En otras palabras, se pretende que los usuarios que jueguen a estos minijuegos sean capaces de comprender de manera práctica las ideas que se desean transmitir. De acuerdo con lo anterior, se han identificado tres conceptos fundamentales: el método Kanban, la filosofía de cinco ceros y la de las 5S.

El proceso de creación del videojuego se inició con la elaboración de la primera de las escenas, en la que el jugador podrá desplazarse con libertad. Esta escena será conocida como "Principal", ya que es en ella donde el usuario iniciará su aventura y a la que regresará cada vez que complete una tarea. El objetivo principal de esta escena es diseñar un escenario que represente el patio de una empresa, en el que el jugador pueda moverse con total libertad hasta encontrar una pizarra.

Ya que el objetivo no es crear exactamente los objetos 3D que aparecerán en las diferentes escenas, debemos encontrar estos objetos a través de la plataforma que facilita el programa de Unity. Para acceder a esta plataforma (Figura 23): *Window / Asset Store / Search online.*

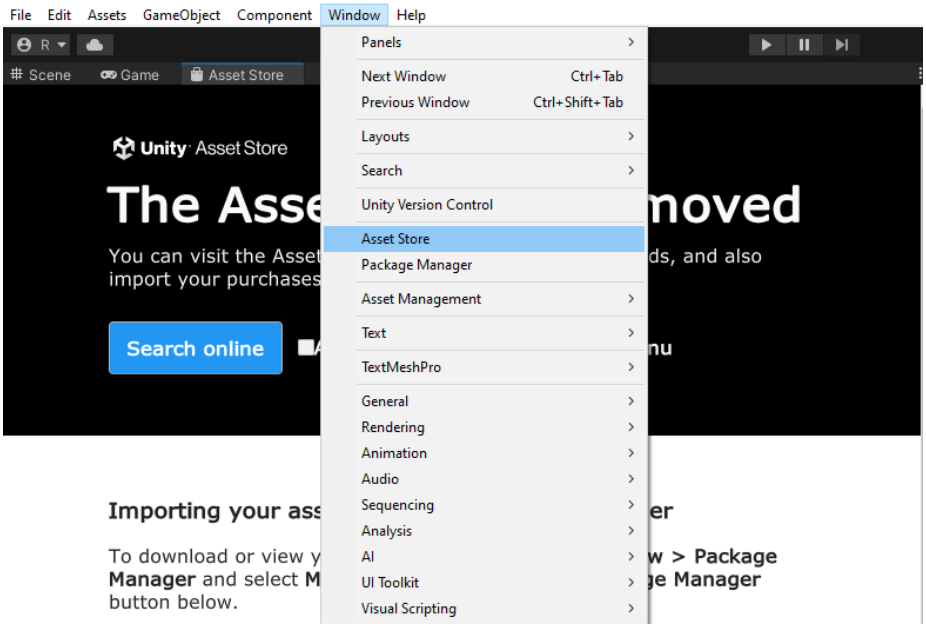

Figura 23: Acceso a la plataforma de Asset Store.

Al seleccionar la opción "Search online" se abrirá automáticamente una nueva pestaña en el navegador que nos proporcionará acceso a la plataforma correspondiente. Además, se observará que el usuario de Unity se ha vinculado automáticamente con la plataforma, lo que implica que el programa de Unity y la plataforma están conectados directamente. Este hecho es de gran importancia, ya que al descargar los paquetes de recursos que se deseemos, estos se agregarán directamente en nuestro programa gracias a la conexión establecida entre ambas herramientas.

A continuación, a través de palabras clave, en inglés, buscaremos todos aquellos recursos que podamos necesitar para nuestra escena. Es muy recomendable aplicar el filtro "*Free Assets*", ya que nos ayudará a no perder el tiempo con aquellos recursos que, aunque pueden ser muy útiles, sean de pago y, por tanto, no nos interesen para este proyecto. En la Figura 24 podemos ver un ejemplo en el que se ha buscado una puerta para colocar en nuestro minijuego.

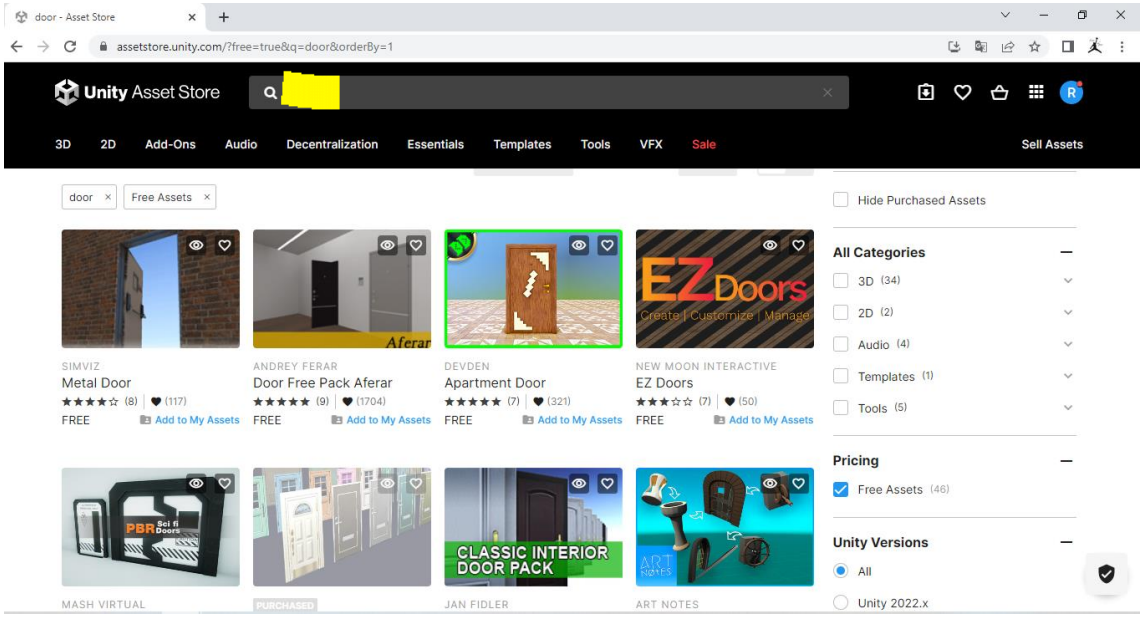

Figura 24: Ejemplo de búsqueda en Assets Store y aplicación del filtro "Free Assets".

Posteriormente, solo es necesario seleccionar el paquete que se prefiera y agregarlo al programa. Para hacerlo, se debe hacer clic en el botón "*Add to My Assets*", lo que permitirá descargar (*download*) e importar el paquete seleccionado a Unity. Se abrirá una nueva pestaña dentro del programa, en la que se deberá descargar el paquete de recursos y, posteriormente, importarlo, tal y como se ilustra en las Figura 25 y Figura 26.

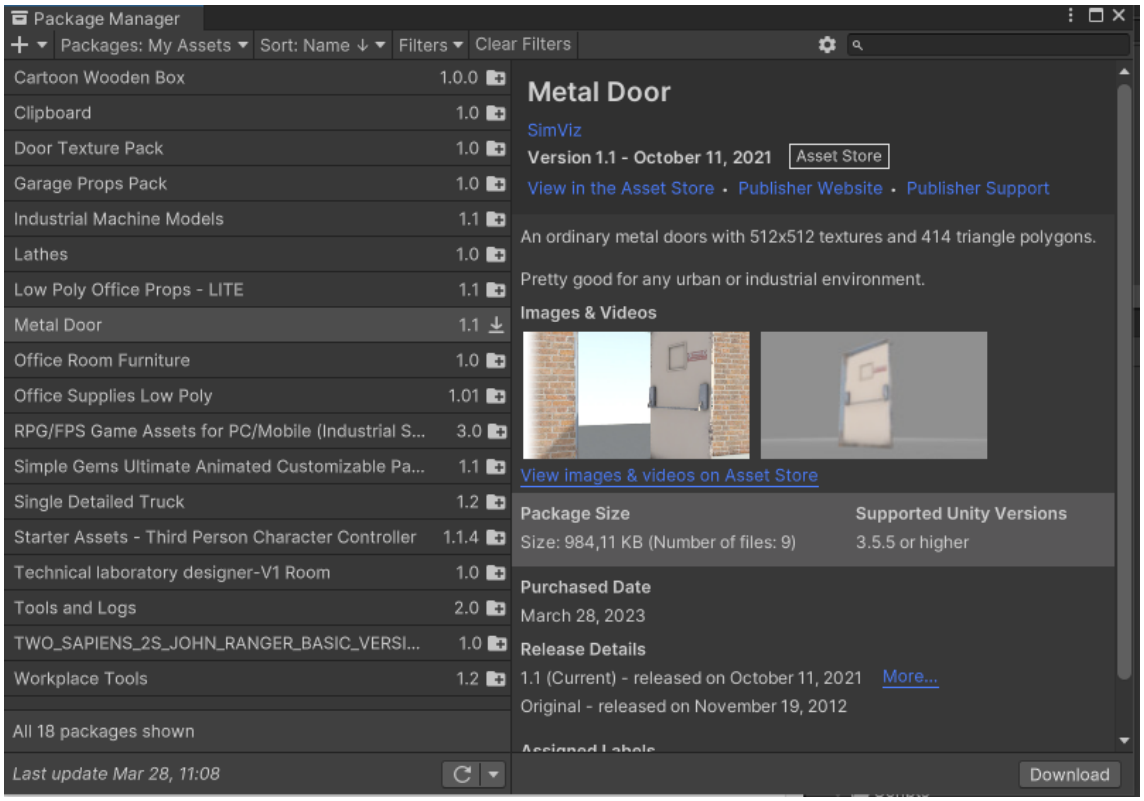

Figura 25: Descarga de un paquete de recursos del Assets Store.

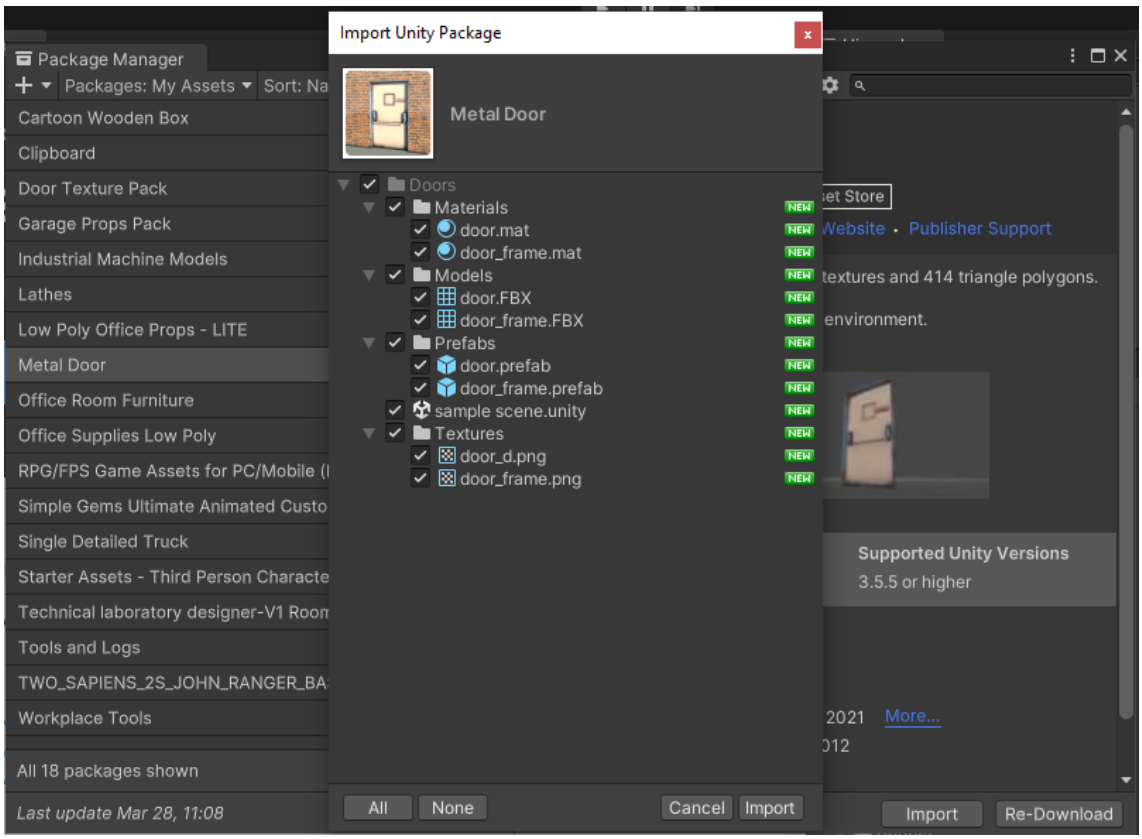

Figura 26: Importar paquete de recursos del Assets Store.

Una vez finalizado este proceso, nuestros nuevos recursos estarán listos y disponibles en todo momento desde la pestaña de "*Project*". Es importante destacar que, aunque se debe importar el paquete completo, no es necesario utilizar todos los recursos que se encuentren dentro de él, sino solo aquellos que sean útiles para nuestro proyecto en específico.

3.2 Metodología de aplicación dentro del juego

A continuación, se explicará en qué consiste el juego y cómo podemos movernos a través de las diferentes pantallas y menús que existen. Además, se explicará el porqué de cada escena en relación con los conceptos vistos hasta el momento. Es importante volver a remarcar que el objetivo de este juego es conseguir plasmar de forma muy básica conceptos como el método Kanban o las filosofías de las 5S y cinco ceros. Es un juego didáctico a través del cual el usuario podrá realizar ejercicios fáciles que le ayudarán a fijar conocimientos acerca de estas tareas.

Cuando abrimos por primera vez en juego, lo primero que nos encontramos es un mensaje en el podemos ver unas instrucciones que nos explican de forma breve que nos encontramos en una fábrica industrial al comienzo de nuestra jornada laboral y lo

primero que debemos hacer es buscar la pizarra de tareas. Esta pizarra se encuentra en el patio de esta empresa.

El usuario deberá pulsar, con ayuda del ratón, al único botón existente para dar comienzo al juego. El jugador podrá moverse libremente por la escena disfrutando de una visión 3D realista de una empresa de carácter industrial, tal y como se puede ver en la Figura 27. Para ello, de ahora en adelante el usuario podrá mover a su personaje a través de los siguientes controles:

- *Control del cuerpo*: flechas del teclado o teclas A (izquierda), S (atrás), D (derecha), W (adelante).
- *Control de la cámara* (cabeza): a través del ratón el usuario podrá mover la imagen de la pantalla.

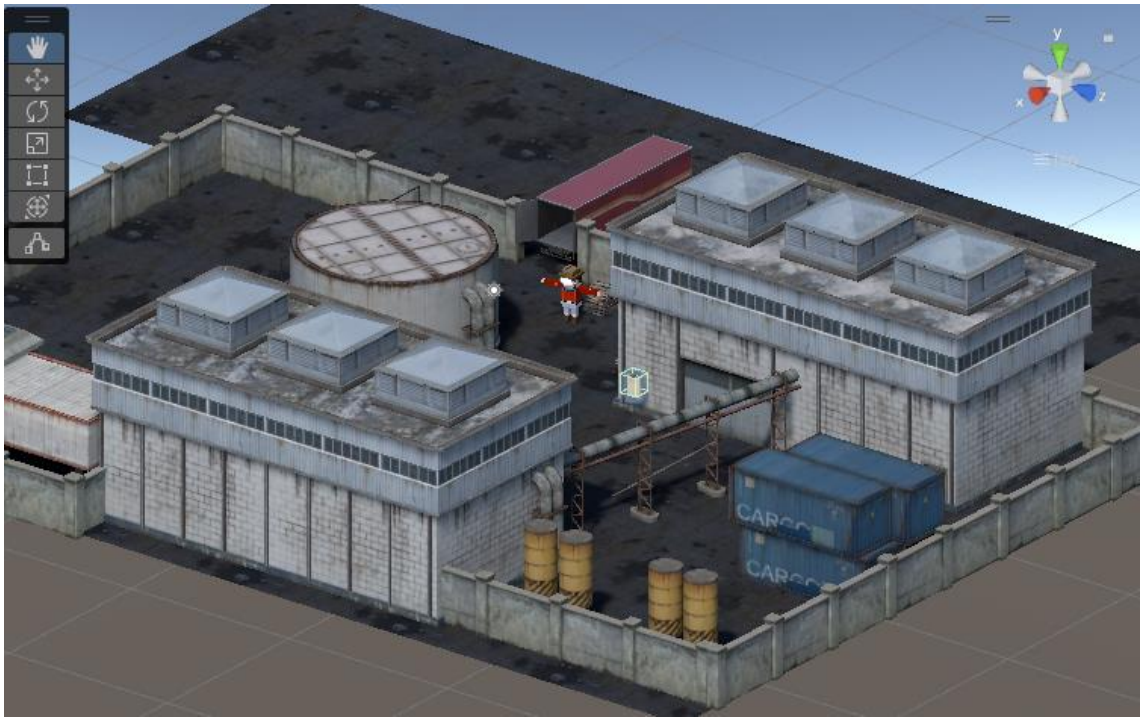

Figura 27: Imagen donde puede apreciarse la nave industrial y la ubicación de la pizarra.

La pizarra se encuentra fácilmente ya que se ha creado una animación que hace que esta suba y baje suspendida en el aire. Posteriormente, se explicará cómo se ha creado dicha animación. Finalmente, cuando el usuario consiga encontrar la pizarra y entre en contacto directo con ella (colisione), automáticamente el entorno cambiará. Tendremos por pantalla un menú que simula una pizarra de organización (Figura 28).

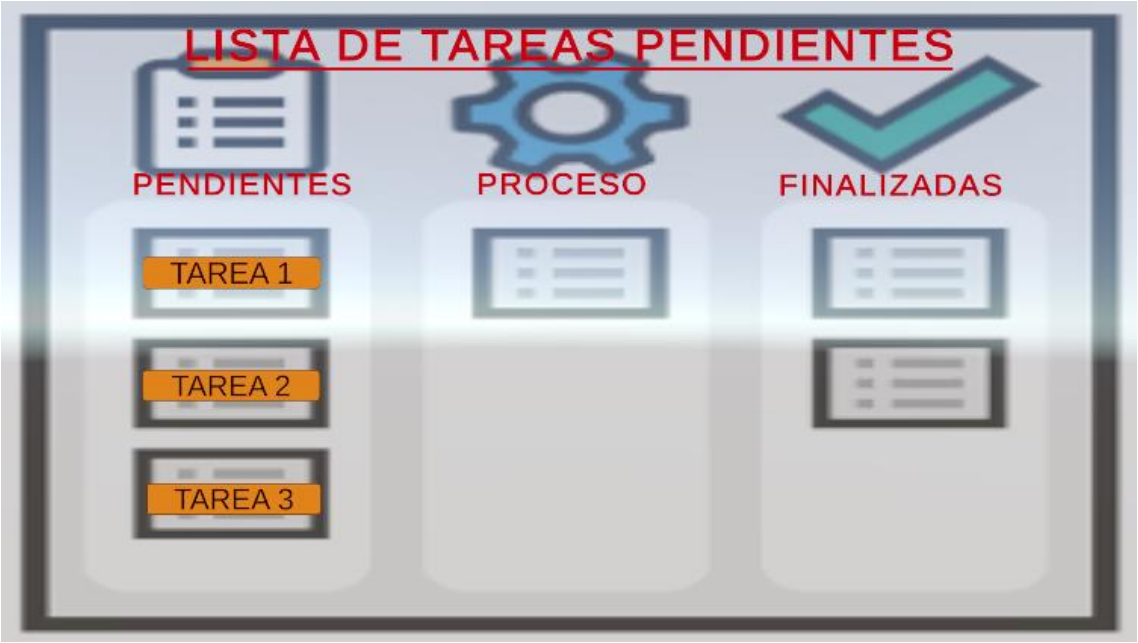

Figura 28: Imagen del menú de selección de las tareas.

Esta es la primera de las aplicaciones prácticas que nos encontramos. Este menú está dividido en tres columnas en las que dependiendo del lugar en el que nos encontremos, la tarea pertenece al campo de proceso pendiente, realizándose o terminado. Como es la primera vez que se ingresa en este menú, el jugador verá que todas las tareas se encuentran dentro de la columna de pendientes, pero esto cambiará según vaya terminando cada una de ellas. Posteriormente, tendremos la ocasión de volver a este menú dinámico y ver cómo tareas que ya se han realizado se encuentran en el apartado de "terminadas".

Este menú es un fiel reflejo de la metodología Kanban, donde los diferentes botones de las tareas simbolizan las tarjetas Kanban que se van desplazando por las columnas según el lugar que les corresponde en el proceso productivo. Cabe resaltar que, aunque esto es un juego y solo existe un usuario participando, en una empresa real, son muchas las personas que tienen acceso a esta "pizarra" y es una manera fácil y sencilla de conseguir más eficiencia en el proceso como ya vimos anteriormente. Incluso a través de este método, los trabajadores sabrán en todo momento cuáles son las tareas que sí pueden realizar porque se encuentran en la columna de pendientes o cuáles ya se han finalizado con éxito, además de las que se están realizando en ese preciso instante.

El jugador seleccionará una de las tareas propuestas y automáticamente cambiará de escenario. Cada una de estas tareas simula de forma básica nociones relativas con la filosofía de cinco ceros o las 5S como veremos a continuación. No se pretende en ningún momento que dentro de cada tarea se puedan ver todas las partes de dichas filosofías, sino que se puedan identificar dentro de cada una de ellas algunos rasgos característicos.

TAREA 1

Esta primera tarea se realiza en el ambiente de una pequeña oficina. El jugador debe moverse libremente por ella y conseguir encontrar los objetos que están fuera de lugar y eliminarlos de la escena.

Conforme va encontrando estos objetos que no deberían estar dentro de una oficina o que simplemente están fuera de su sitio, el jugador debe hacer clic con el ratón sobre estos haciendo que desaparezcan. Son un total de siete objetos los que debe identificar: un paraguas, un trozo de pizza, una sierra, un destornillador, una pala, una escoba que esta tirada por el suelo y unos plátanos. Tras encontrar los siete objetos, el jugador podrá ver el puesto de trabajo limpio y ordenado, por lo que habrá logrado el objetivo de esta tarea y podrá regresar a través del botón “Volver al menú” a la fábrica donde, de nuevo, debe acudir a la pizarra y escoger de nuevo otra tarea.

Este minijuego se basa en saber identificar defectos en un puesto de trabajo de oficina. El objetivo que se persigue con esta tarea es conseguir que el jugador comprenda que debe mantener limpio el puesto de trabajo y en buenas condiciones. Debe ser capaz de identificar al menos dos de las 5S, que son: seleccionar y limpiar.

TAREA 2

En esta segunda tarea podremos movernos a través de un pequeño taller. A través de una breve explicación inicial que sale por pantalla, podemos ver en qué consiste nuestra tarea.

A través de este escenario, el jugador podrá ver el interior de un taller de mecanizado y una disposición simple de su funcionamiento. Encontrará máquinas como tornos y fresadoras, las cuales crean las piezas para posteriormente ser ensambladas en una mesa acondicionada con herramientas. Después, mete las piezas ensambladas en cajas para enviarlas como producto acabado. Para ello, a través de un robot (protegido por una jaula) mueve las cajas pesadas hasta una cinta transportadora para aproximar dichas cajas a la zona de salida de productos, donde finalmente serán enviadas al cliente a través de camiones.

El objetivo de esta tarea será básico. Buscar el palé que está en un lugar no acondicionado para su almacenaje, y una vez identificado, cogerlo aproximándonos a él y pulsando la barra de espacio. A continuación, debemos movernos hasta el lugar donde se encuentran apilados el resto de los pallets y volver a pulsar la barra espaciadora para soltar el objeto.

Este minijuego pretende mostrar, de forma simplificada, que no existen puntos de almacenamiento grandes de piezas que supongan pérdidas, es decir, se puede ver que la intención de la empresa es no acumular stock. Además de la tarea que debemos realizar, en la que estamos identificando un defecto (un palé que no está en su sitio) y solucionándolo (moviéndolo hasta su lugar correcto). Con esto estamos ayudando a que no se produzca una avería dentro de la jaula del robot por alguna colisión del robot contra el palé. Por tanto, una vez mencionado todo esto se puede identificar el objetivo de esta tarea es mostrar parte de la filosofía de los cinco ceros. Aunque también podríamos reconocer una de las 5S, ordenar el puesto de trabajo.

TAREA 3

Esta es la tarea más simple y básica de entender. Consiste en que el jugador debe colocar todas las cajas en los diferentes huecos del almacén según su color, es decir, la caja roja debe ir a la estantería roja, la verde a la estantería verde y así sucesivamente. Aunque la tarea aparenta ser un juego para niños, la intención que se busca es que el trabajador encargado de esta tarea comprenda que cada cosa tiene su lugar adecuado.

La tarea consiste en ubicar cajas en diferentes estantes según su correspondiente color en un almacén, con el propósito de fomentar en el trabajador la comprensión de la importancia de asignar un lugar apropiado a cada objeto. A pesar de que esta tarea puede parecer sencilla, se puede interpretar como una metáfora del trabajo del carretillero en un almacén, quien debe posicionar piezas en sitios específicos, de acuerdo con la información que acompaña a cada caja. Esta tarea ilustra la disciplina y normalización de las 5S, dos aspectos fundamentales para garantizar una correcta selección, orden y limpieza en un entorno laboral. Por lo tanto, la formación y disciplina del trabajador son esenciales para lograr una adecuada implementación de las 5S.

3.3 Construcción de las escenas

3.3.1 Pantalla PRINCIPAL

Llegados a este punto, habremos invertido un tiempo en buscar aquello que nos será útil para la creación de nuestra escena. Aunque si más adelante nos hiciera falta algún recurso más, tan solo debemos regresar al navegador y repetir el mismo proceso que se acaba de explicar.

Dentro de estos paquetes de recursos, normalmente encontraremos materiales y prefabs. Los materiales son los “colores” con los que diseñaremos el objeto, es decir,

aunque el recurso viene totalmente diseñado, podremos modificarlo a nuestro gusto. Y los prefabs son los objetos como tal que debemos arrastrar a nuestra escena.

Durante la creación de cada escena debemos darnos cuenta de que algunos objetos deberán incluir un componente *Rigidbody*, lo que les dará facultades físicas como la gravedad. Por lo que en todas las escenas debemos incluir al menos un plano que actúe como suelo, para evitar que nuestros objetos caigan al hacer las simulaciones. Para introducir este objeto plano debemos ir a la pestaña de jerarquía: *Hacer clic sobre el botón derecho del ratón / 3D Objet / Plane.*

Este objeto tiene la capacidad de ser utilizado tanto en el suelo como en las paredes o el techo de los entornos. Cuando aparece en la escena, presenta una característica especial en la que habrá un lado visible y otro que no, a pesar de que siempre estará presente el plano. Al crearlo, se generará un plano que por defecto se mostrará de color blanco; no obstante, al girarlo 180 grados se volverá transparente y dará la sensación de que no hay nada colocado, aunque en realidad el objeto estará presente. En el caso específico que nos implica de este proyecto, las paredes (o muros como los llamaremos en los *scripts*), serán utilizadas en un futuro para establecer los límites del juego para el jugador. Cabe destacar que se proporcionarán detalles adicionales sobre su uso más adelante.

Tras colocar rigurosamente cada objeto en el entorno, el primer escenario que incluiría al jugador quedaría tal y como se muestra en la Figura 29.

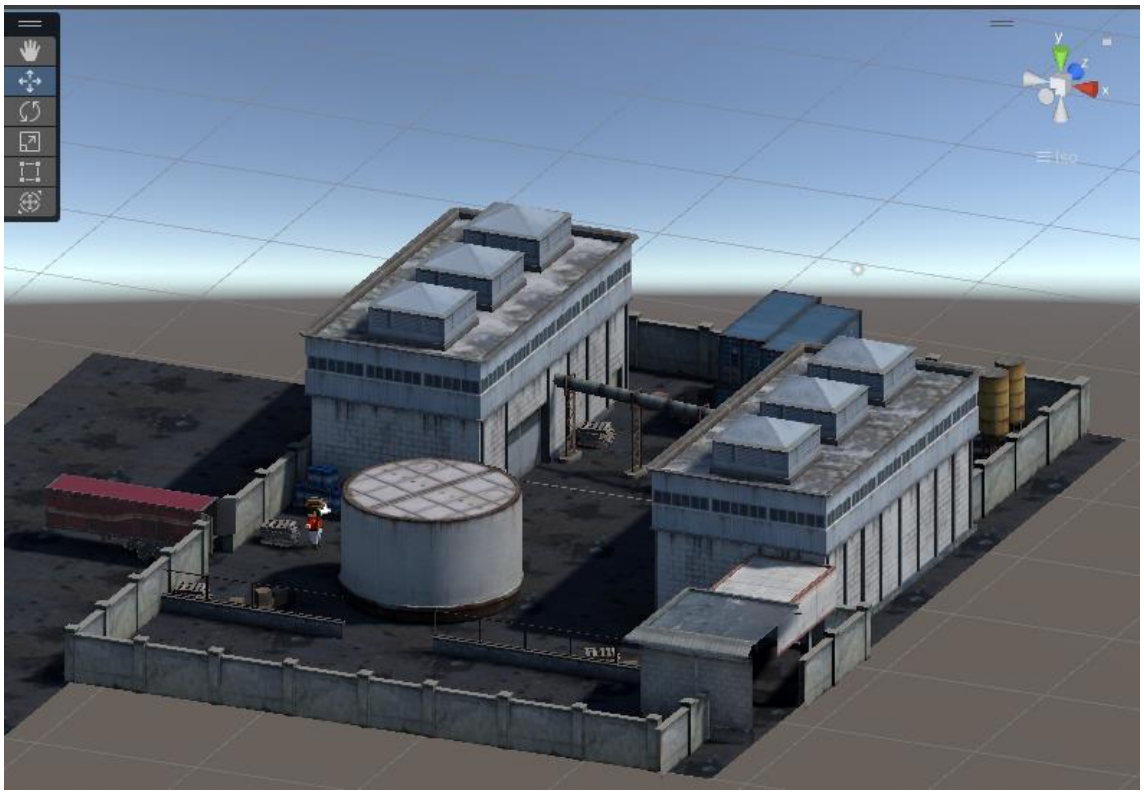

Figura 29: Escena principal

Si nos fijamos en la Figura 29, podemos ver cómo, además de insertar un personaje en la escena, se ha modificado de lugar la cámara para que se sitúe en la cabeza del jugador. Esto se hace porque nuestro objetivo será que podamos movernos por la escena en primera persona. Además, podemos ver en la Figura 30 que la componente de la cámara ("*MainCamara*") se encuentra dentro del panel de jerarquía como un objeto hijo del objeto "Jugador". Esto implica que la posición de la cámara siempre dependerá de la posición en la que se encuentre el jugador; así, si el jugador se mueve, la cámara conservará su posición relativa a este.

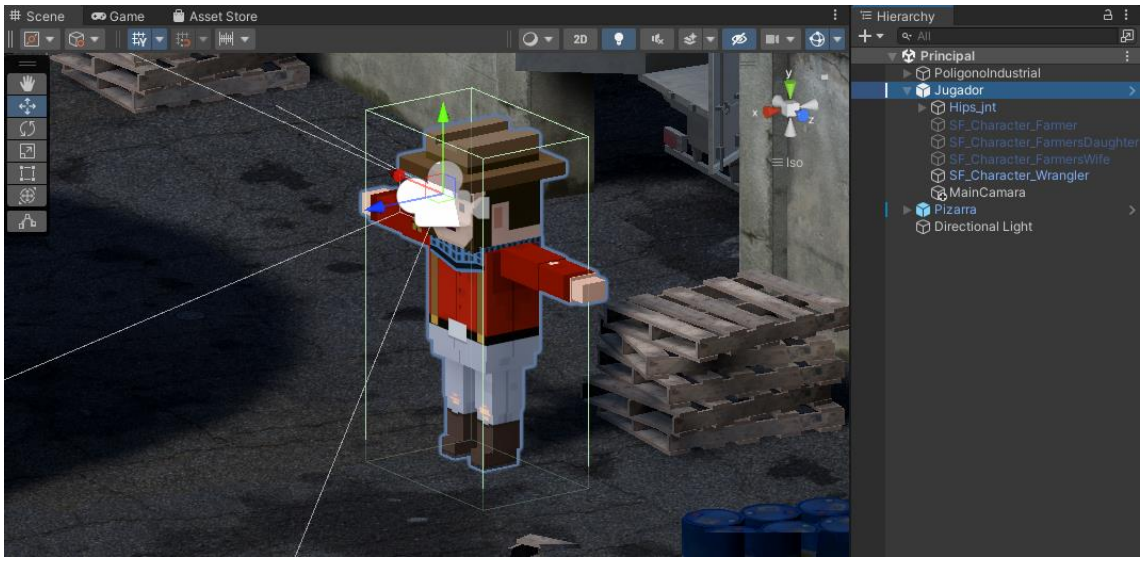

Figura 30: Jugador con la Cámara colocada para primera persona.

Con el fin de mantener una buena organización en el panel de jerarquía, se ha creado una carpeta denominada "Polígono Industrial" en la que se han incluido los objetos que forman parte de la empresa. Esta medida permite tener los elementos ordenados y facilita su manipulación. En el panel de jerarquía también se encuentra un objeto denominado "Pizarra", que aún no ha sido presentado. Este objeto será el punto de interacción para el usuario, ya que, al colisionar con él, accederá al menú de selección de tareas, del que se hablará más adelante.

Con el fin de que el usuario identifique más fácilmente este objeto, se ha creado una animación con la que la "Pizarra" suba y baje continuamente durante la simulación. Para ello seleccionaremos la pizarra e iremos a las siguientes pestañas: *Windows / Animation / Animation* (Figura 31).

Al hacer clic en el botón "*Create*" se abrirá una pestaña en la que se podrá poner un nombre y guardar la animación que se desea crear. Para crear la animación, simplemente se deben seleccionar dos puntos diferentes de la posición del objeto. En la pestaña de la animación creada, se debe presionar el botón de grabación (un botón con dos círculos concéntricos y el interior coloreado de rojo).

A continuación, se debe hacer clic sobre la posición inicial deseada de la pizarra, de manera que aparezcan dos puntos en la pestaña de animación que servirán para

verificar que la posición inicial se ha registrado correctamente. Posteriormente, se debe hacer clic sobre la línea de tiempo que se desea que tarde la animación (en este caso, tres segundos) y mover el objeto hasta la posición final de la animación. Finalmente, se debe volver a presionar el botón de grabación para registrar el movimiento completo. Véase la Figura 32 para una mejor comprensión.

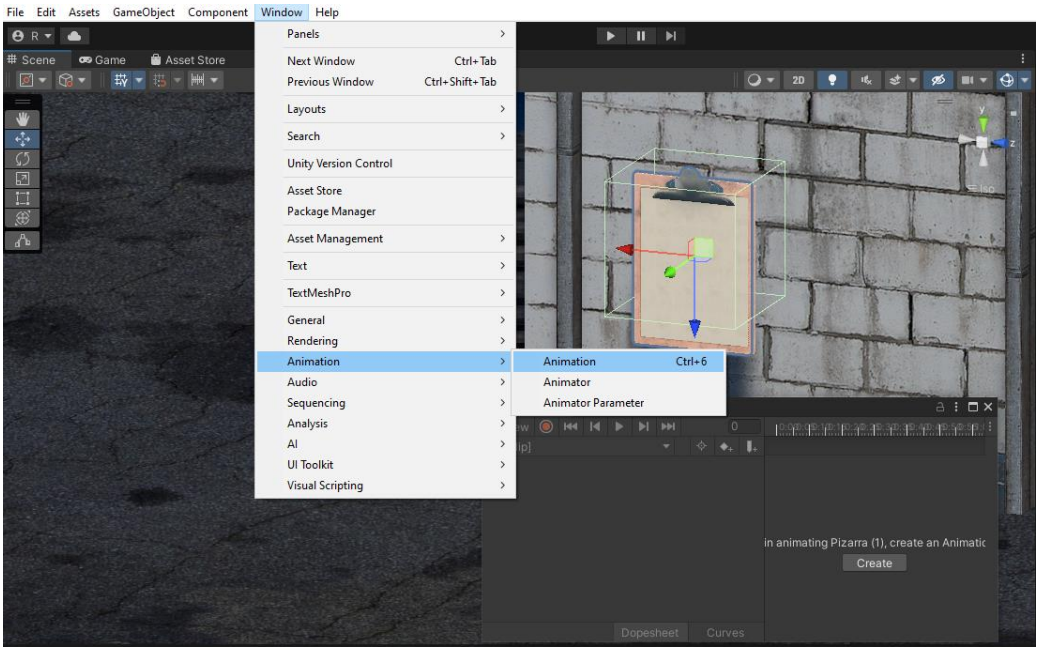

Figura 31: Acceso y creación de la animación del objeto "Pizarra".

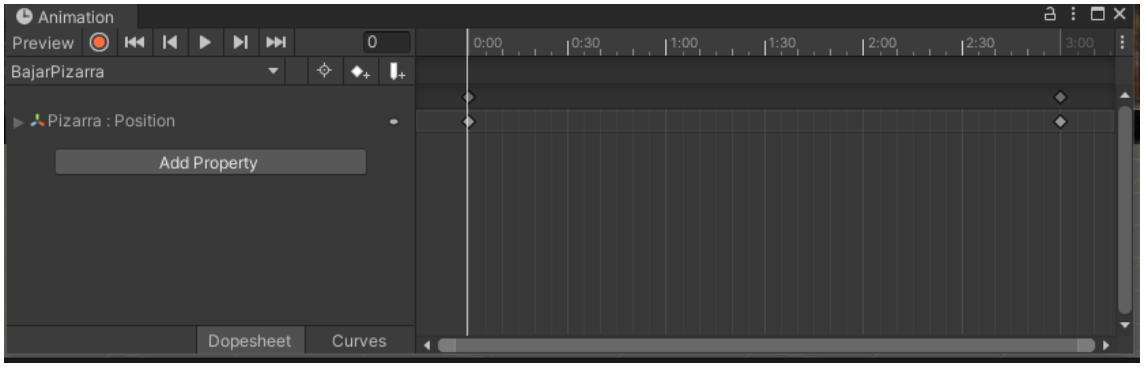

Figura 32: Registro de la animación.

Una vez creada la animación, es importante verificar que se ha asignado correctamente al objeto deseado. Para eso, se debe comprobar que en el Inspector del objeto "Pizarra" se ha creado una sección llamada "*Animator*" y que dentro de ella se encuentra la animación creada. En nuestro caso, se han creado dos animaciones antagónicas: una para el descenso de la pizarra y otra para su ascenso al lugar original. Además, se ha realizado una transición de una animación a otra (*Make Transition*), provocando que ambas animaciones se reproduzcan en bucle, tal y como se aprecia en la Figura 33.

Para finalizar, solo nos falta dar vida a nuestro personaje y a la escena, para lo que debemos crear nuestros scripts. Empezaremos con la explicación del Jugador, ya que la manera en la que configuremos a este personaje nos servirá para el resto de las ocasiones en las que tengamos que utilizarlo. En primer lugar, debemos añadir dos componentes a este objeto: *Rigidbody* y *Box Collider*. Este segundo, debemos ajustarlo de tal modo que tenga aproximadamente las dimensiones del objeto Jugador. Además, debemos asignarle un script al que hemos llamado **MoverJugador**, cuyo contenido se muestra en el Código 2.

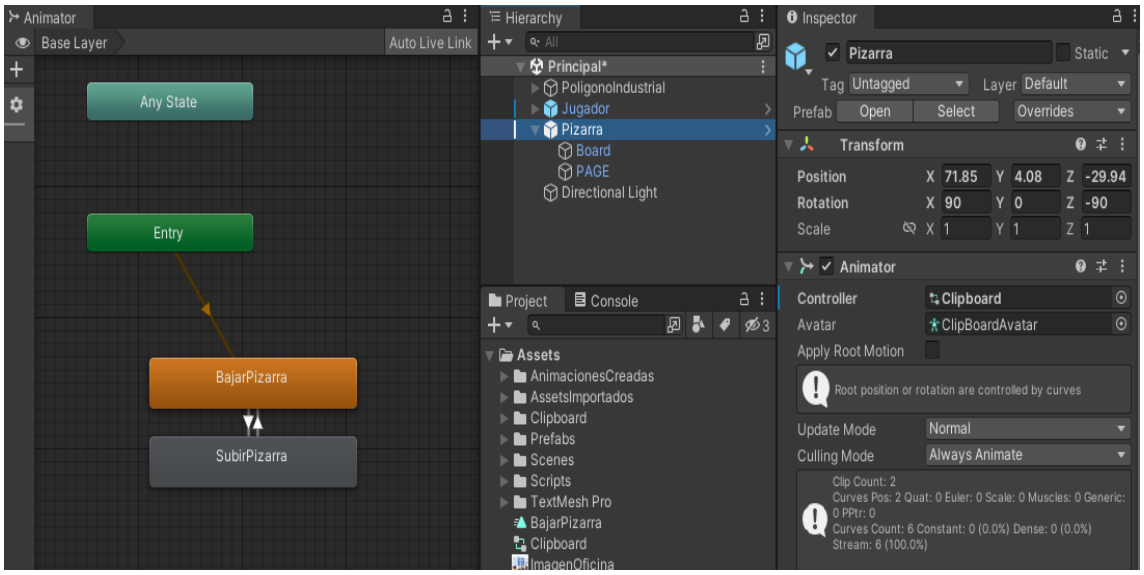

Figura 33: Animaciones de la Pizarra.

Código 2: Script para controlar el movimiento básico del jugador.

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

public class MoverJugador : MonoBehaviour
{
    // Velocidad de movimiento del jugador
    [SerializeField] private float velocidadMovimiento = 5f;
    // Velocidad de rotación de la cámara
    [SerializeField] private float velocidadRotacion = 3f;
    // La transformación de la cámara del jugador
    [SerializeField] private Transform cameraTransform;

    private float verticalRotation = 0f; // La rotación vertical actual de la cámara

    private void Update()
    {
        // Movimiento del jugador
        float horizontalMovement = Input.GetAxis("Horizontal");
        float verticalMovement = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(horizontalMovement, 0f, verticalMovement) *
                                    velocidadMovimiento * Time.deltaTime;
```

```
        RaycastHit hit;
            // Verificación de objeto en el camino
        if (Physics.Raycast(transform.position, movement.normalized, out hit,
                movement.magnitude + 0.1f))
        {
            if (hit.collider.CompareTag("Muros"))
                        //Si el obstáculo es un objeto con la etiqueta "Muros", entonces se
                          detendrá el movimiento del jugador
            {
                movement = Vector3.zero;
            }
        }

        transform.Translate(movement, Space.Self);

        // Rotación de la cámara
        float mouseX = Input.GetAxis("Mouse X");
        float mouseY = Input.GetAxis("Mouse Y");

        verticalRotation -= mouseY * velocidadRotacion;
        // Asegurar que la rotación vertical se mantenga en un rango de -90 a 90 grados
        verticalRotation = Mathf.Clamp(verticalRotation, -90f, 90f);

        cameraTransform.localEulerAngles = new Vector3(verticalRotation, 0f, 0f);
        transform.Rotate(Vector3.up, mouseX * velocidadRotacion);
    }
}
```

Como resumen del contenido de este *script* que acabamos de ver, podemos decir que es un *script* habitual en las escenas de Unity, una pieza esencial de la mecánica de juego, ya que permite al jugador controlar el movimiento y la orientación de la cámara de forma fluida y natural. Este script controla el movimiento del jugador y la rotación de la cámara en una escena 3D. El objetivo del *script* es permitir que el jugador se mueva hacia adelante y hacia atrás, y gire a la izquierda y a la derecha, utilizando los controles del teclado y el ratón.

En el código se definen tres variables privadas: la velocidad de movimiento del jugador, la velocidad de rotación de la cámara y la transformación de la cámara del jugador. Este último se asigna desde Unity, es decir, debemos indicar antes de realizar la simulación que esta variable es comandada por el objeto que hace las veces de cámara de juego. El movimiento del jugador se controla en el método "*Update()*" que se ejecuta en cada fotograma (*frame*), utilizando la entrada del teclado para determinar la dirección del movimiento. El *script* también utiliza un ***raycast*** (más adelante se hablará de esto) para detectar si el jugador choca con los "Muros" y detener el movimiento en ese caso.

La rotación de la cámara se controla en el mismo método "*Update()",* utilizando la entrada del ratón para determinar la rotación horizontal y vertical de la cámara. El *script* limita la rotación vertical a un rango de -90 a 90 grados para evitar que la cámara gire más allá de los límites de la pantalla.

Me gustaría destacar una sección específica del código que cumple una función fundamental en la garantía de una experiencia de juego coherente y realista. Se trata del método ***Raycast***, que traza una línea recta desde un punto en una dirección determinada y devuelve información sobre cualquier objeto que se interfiera en dicha línea. En este contexto, el ***Raycast*** se ejecuta desde la posición actual del jugador en la dirección de su movimiento, con el objetivo de detectar cualquier objeto que pudiera obstaculizar su recorrido.

Cabe destacar que la longitud del ***Raycast*** se establece en base a la magnitud del vector de movimiento del jugador, adicionado a un valor pequeño (0.1f en este caso), a fin de asegurar la detección de cualquier objeto cercano. La información recopilada por el *Raycast* se almacena en la variable "*hit*", y se utiliza para verificar si el objeto detectado tiene la etiqueta "Muros" mediante el método "*CompareTag()*". En caso de que se confirme la presencia de un obstáculo sólido en la trayectoria del jugador, el vector de movimiento se establece en cero, lo que implica una detención inmediata del personaje.

Por otro lado, tenemos uno de los *scripts* más curiosos a la par de simples de los que vamos a hacer uso, que nos permitirá cambiar de escena e ir a una escena donde tenemos preparado un menú de selección. Este *script*, llamado "CargarMenuCambioEscena", se asignará al objeto “Pizarra”. Pero antes, debemos añadir el componente *Box Collider* a este objeto y activar la opción “Is Trigger” (Véase Figura 34). Esto le permitirá a Unity saber que el objeto es un disparador, es decir, un objeto que al hacer contacto con él realizará alguna acción/evento que programaremos, que en nuestro caso será lanzar la escena del menú.

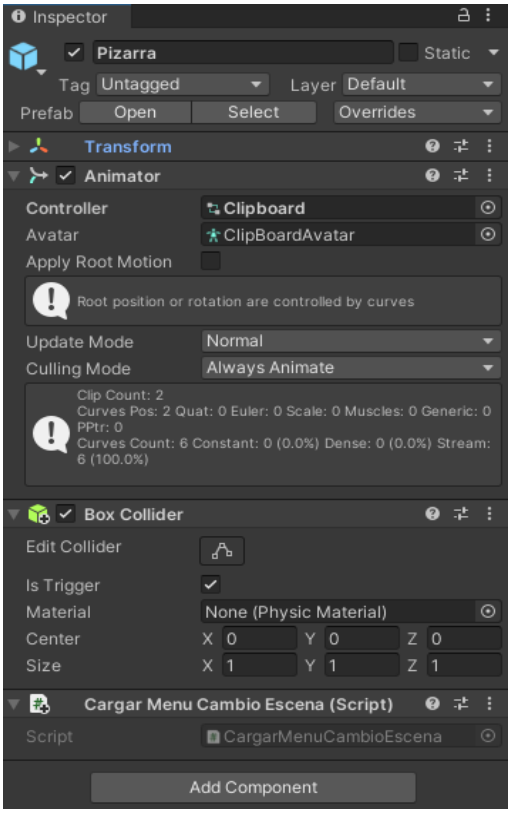

Figura 34: Inspector de la Pizarra con todos los componentes necesarios.

El *script* de Código 3 quedaría de la siguiente manera:

Código 3: Script de cambio de escena fijado al menú.

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

public class CargarMenuCambioEscena : MonoBehaviour
{
   private void OnTriggerEnter(Collider other) //Cuando entra en colisión...
   {
      if (other.gameObject.CompareTag("Player"))
            //Si el objeto que entró en la zona es el jugador, comprobando si tiene la etiqueta
            //"Player"
      {
         SceneManager.LoadScene(2); //Carga la escena con el índice 2
      }
   }
}
```

En resumen, este *script* permite cargar una escena específica al entrar en contacto con un objeto con la etiqueta "*Player*" (la cual asignaremos al Jugador) que a su vez tenga el componente *Collider*.

Es un *script* que carga una escena específica (en este caso, la escena número 2) al entrar en contacto con un objeto que tenga el componente *Collider* (Jugador). Este componente debe tener la etiqueta "*Player*" para que se cumpla la condición y se

active la carga de la escena. En concreto, el *script* utiliza el método *OnTriggerEnter()*, que se ejecuta cuando un objeto entra en contacto con otro teniendo ambos un componente *Collider*.

Dentro del método OnTriggerEnter() se verifica si el objeto que ha entrado en contacto tiene la etiqueta "*Player*", mediante el método *CompareTag()*. Si la condición se cumple, se carga la escena número dos utilizando el método *LoadScene()*, que permite cargar una escena por su índice o su nombre. Es fundamental tener en cuenta que al terminar el proyecto por completo es necesario cargar cada escena para continuar trabajando en su construcción. En particular, para la escena que contiene el menú, se le ha asignado el número dos. En el futuro se hará un *script* similar a este, donde se explorará una forma alternativa de programar la carga de escenas, utilizando el nombre en vez del número de índice.

3.3.2 Pantalla TAREA 1

Esta tarea se ha basado en una escena obtenida de la plataforma oficial de Unity (*Assets Store*). En esta ocasión, tras importar todo el paquete de recursos, existía una escena de prueba en la que hemos basado nuestra escena particular. Inicialmente teníamos la estructura edificada de la oficina por lo que solo faltaba llenar de contenido la escena, es decir, colocar nuestro personaje y añadir todos aquellos recursos que vayamos a utilizar. A continuación, en la Figura 35 puede verse la parte estructural de la que se ha partido.

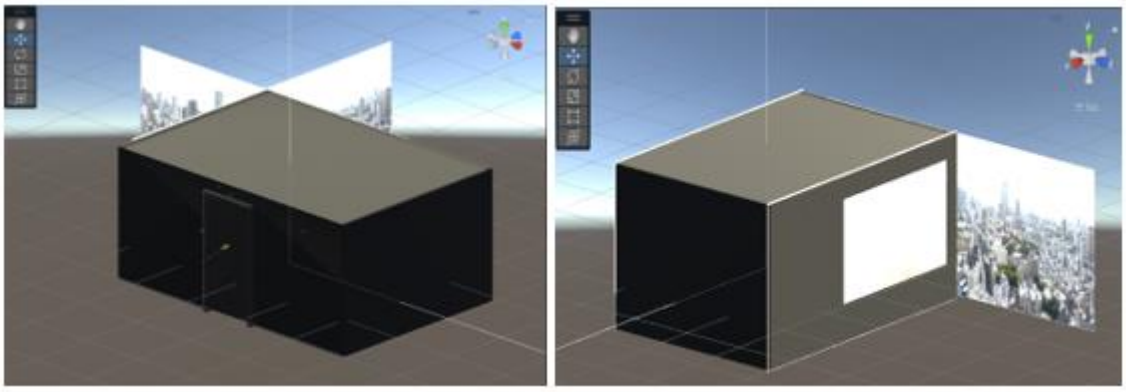

Figura 35: Estructura de partida de un paquete importado.

Como hemos visto anteriormente, el objetivo del usuario en esta primera tarea será encontrar ciertos objetos que no deberían encontrarse en la escena, como son: un paraguas, un trozo de pizza, unos plátanos, una sierra, una pala y un destornillador. Un total de seis objetos que el usuario haciendo clic sobre ellos podrá hacer desaparecer de la escena. Tras colocar todos los objetos en la oficina (tanto los que si desaparecen como los que no), la escena quedaría de la siguiente forma (Figura 36 y Figura 37).

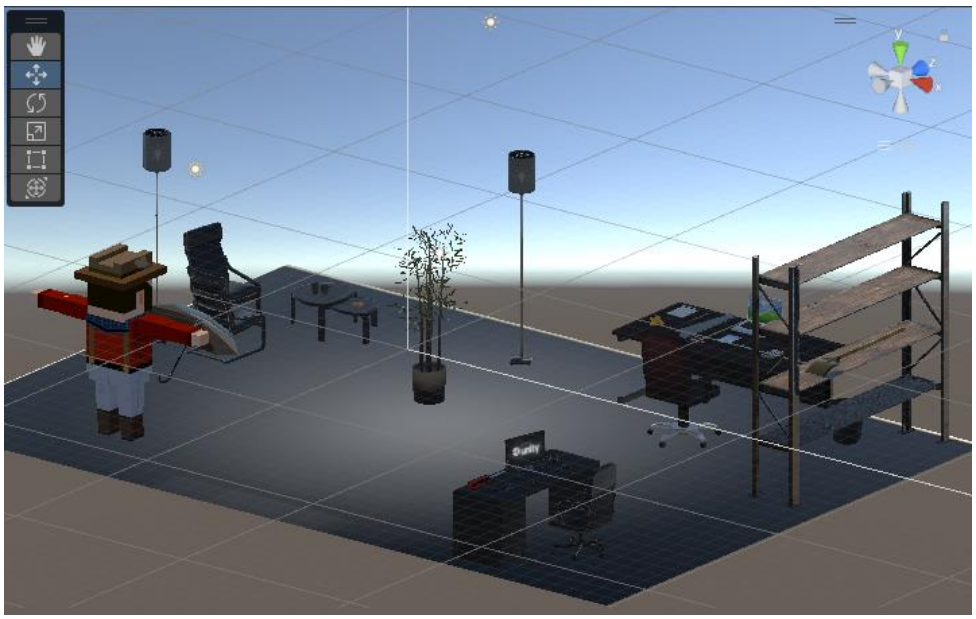

Figura 36: Escena de la Tarea 1 eliminando la parte estructural.

Figura 37: Imagen del interior de la oficina

Una vez colocados todos los recursos sobre nuestra escena, debemos darle vida. Esto se hará por medio de scripts. El primero y más sencillo de todos será el de dar jugabilidad a nuestro personaje, y para ello le asignaremos el *script* que vimos anteriormente llamado "MoverJugador". Además, debemos añadirle dos componentes que también hemos visto anteriormente: *Rigidbody* y *Box Collider*. Estos nos servirán

para dotar a nuestro personaje de propiedades físicas como el peso o la gravedad, además de un volumen de ocupación para identificar colisiones con otros objetos.

Seguimos con la programación del siguiente script llamado "**ControlJugador**", que asignaremos a todos los objetos que queremos que desaparezcan de la escena cuando el usuario lo ordene. El Código 4 quedaría de la siguiente manera:

Código 4: Control del jugador para eliminar objetos y actualizar marcador.

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

public class ControlJugador : MonoBehaviour
{
   // Variable controlJuego que es una referencia al script Marcador
   private Marcador controlJuego;

   void Start()
   {
      //Ubicación exacta donde encontrar la variable
            controlJuego = GameObject.Find ("Gestor de Juego").GetComponent <Marcador> ();
   }

   private void OnMouseDown()
   {
      if (gameObject.name == "Pala" || gameObject.name == "Pizza" || gameObject.name ==
            "Platano" || gameObject.name == "Sierra" || gameObject.name == "Paraguas" ||
            gameObject.name == "Destornillador")
      {
         Debug.Log("Has hecho clic sobre" + gameObject.name); //Escribe en la consola
         Destroy(gameObject); //Elimina de la escena el objeto
         controlJuego.ActualizarMarcador(1);
      }
   }
}
```

Este *script* tiene una característica especial, ya que está conectado con un segundo script llamado "**Marcador**", que será explicado más adelante. Dentro de la clase, se define una variable privada llamada "**controlJuego**" que actúa como una referencia al *script* "Marcador". Es importante destacar este hecho, ya que hasta este momento no había sido utilizado. La variable "controlJuego" se utiliza para acceder a los métodos y propiedades del *script* "Marcador" desde este mismo *script*.

Posteriormente, se inicializa la variable "controlJuego" con la referencia al objeto "Gestor de Juego". Este objeto es una carpeta en el panel de jerarquía de Unity que contiene todos los objetos que se desean hacer desaparecer (plátano, sierra, …). En otras palabras, esta línea de código se encarga de buscar el objeto "Gestor de Juego" en la escena y asignar su *script* "Marcador" a la variable "controlJuego".

Cuando el jugador hace clic en el objeto asociado a este *script*, se llama al método "*OnMouseDown()*". Dentro de este método, se utiliza una estructura condicional "*if*" para verificar si el objeto sobre el que se ha hecho clic es uno de los objetos que se desean hacer desaparecer. Si es así, se escribe un mensaje en la consola indicando que

se hizo clic en ese objeto, se destruye el objeto y se llama al método "ActualizarMarcador" del *script* "Marcador" para sumar un punto al marcador del jugador. Cabe destacar que la consola solo la podrá ver el diseñador del juego, ya que, durante la simulación real del juego, el usuario no tendrá acceso a esta consola.

Además de este *script*, cada uno de estos objetos tendrá asociado un componente "*Box Collider*" con el que Unity podrá identificar cuándo se hace clic sobre el objeto. Por último, solo queda hablar del "*Canvas*". Esta es una carpeta que Unity genera de manera automática al crear alguna de las múltiples opciones de interfaz de usuario (*UI, User Interface*). Más adelante se dará una explicación más precisa de esta interfaz de usuario (*UI*). En esta ocasión, haremos un texto interactivo y un botón para regresar a un menú siempre que lo deseemos. Como podemos ver en la Figura 38, el usuario podrá ver en todo momento durante la simulación un botón que le dará acceso a un menú, además de ver un contador de objetos localizados. Este último se irá incrementando conforme se vayan encontrando los objetos de búsqueda.

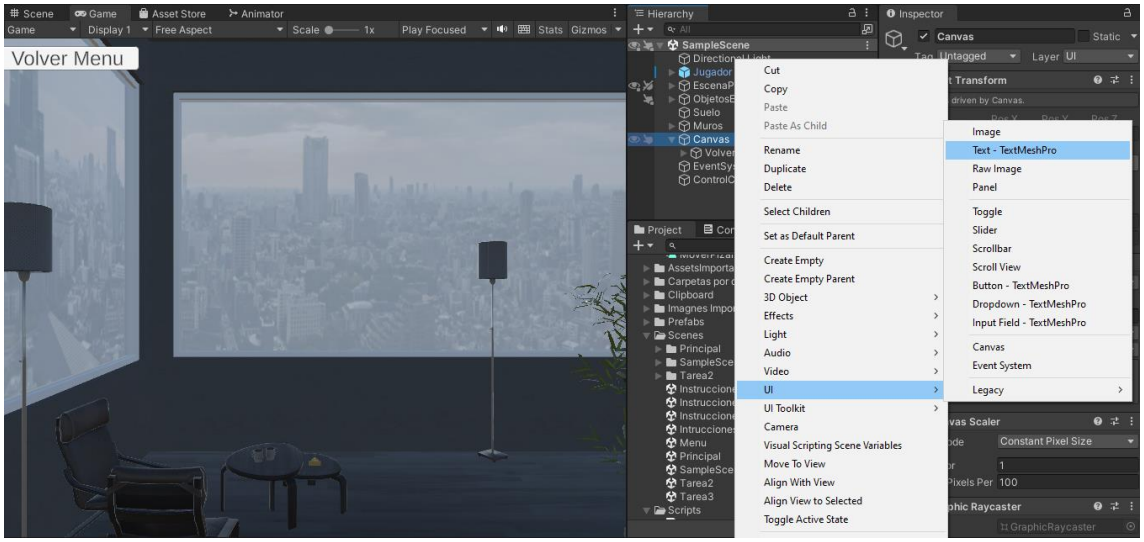

Figura 38: Acceso a la UI. Imagen del texto contador y del botón para regresar al menú.

Para poder incorporar un texto o un botón debemos situarnos en el panel de jerarquía y proceder de la siguiente forma: Botón derecho del ratón / *UI* / *Text -TextMeshPro* o *Button -TextMeshPro.*

Comenzaremos hablando de la incorporación del texto. Tras proceder como se ha dicho, se abrirá el *Inspector* donde configuraremos todo lo necesario. Para empezar, le pondremos un nombre, que en nuestro caso ha sido "Marcador", para poder identificarlo correctamente más adelante.

Dentro de la pestaña "*TextMeshPro -Text (UI*)", colocaremos aquel texto que queremos que el usuario pueda ver al comienzo de la tarea. Además de esto, existen

múltiples estilos con los que podremos configurar a nuestro gusto el texto que sale por pantalla (negrita, cursiva, mayúsculas, tamaño de letra, …). En una pestaña inferior llamada "*Face*" podremos cambiar el color y opacidad de nuestro texto. Finalmente, solo quedará situarlo en el lugar de la pantalla que queramos.

Queda pendiente de explicación la pestaña "*Rect Transform*" que, aunque tiene una gran importancia, se describirá más adelante para mayor facilidad de comprensión (Figura 39).

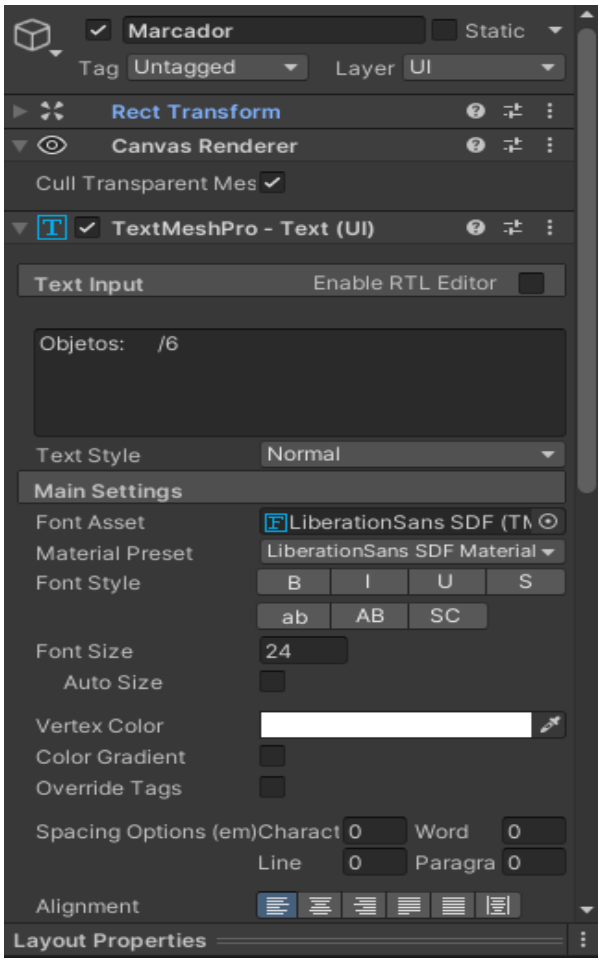

Figura 39: Inspector del marcador.

Para continuar con la manipulación del marcador de objetos, crearemos un objeto vacío al que hemos llamado "Gestor de Juego" y le asociaremos el nuevo *script* "Marcador". Este Código 5 queda de la siguiente forma:

Código 5: Contador de objetos.

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

public class Marcador : MonoBehaviour
{
   public TextMeshProUGUI textoMarcador;
```

```
    public int marcador = 0;

    void Start()
    {
        ActualizarMarcador(0); //Llamar a la funcion con una entrada inicial nula
    }

    public void ActualizarMarcador(int puntosASumar)
    {
        marcador += puntosASumar; //Sumamos uno al marcador
        Debug.Log("Llamo a la funcion actualizar marcador");
        Debug.Log("El marcador esta en " + marcador);
        textoMarcador.text = "Objetos:   " + marcador + " / 6"; //Escritura del marcador
    }
}
```

En la primera línea del código se están importando algunas librerías necesarias para el correcto funcionamiento del código, de donde se destaca la librería *TMPro*. Esta se utiliza para poder tener acceso a la manipulación de texto de la interfaz de usuario.

En el método "*Start()*" se llama al método "ActualizarMarcador()" para inicializar el marcador a cero. Esta función es la encargada de sumar puntos al marcador del jugador. Recibe como parámetro la cantidad de puntos que se desean sumar al marcador y los agrega a la variable "marcador". Como hemos visto, desde el *script* "ControlJugador" se llama a esta función indicándole que sume uno al hacer clic sobre un objeto.

Luego, se imprime un mensaje de depuración en la consola indicando que se ha llamado a la función "ActualizarMarcador" y el valor actual del marcador (para el control del programador). Finalmente, se actualiza el valor del objeto "textoMarcador" para mostrar la puntuación actual del jugador en la pantalla del juego. No debemos olvidar que la variable "textoMarcador" debe asignarse desde Unity.

Por otro lado, tenemos el botón "Volver Menú". Al hacer clic sobre "*Button - TextMeshPro*" se abre el *Inspector* para su configuración. Vemos en la Figura 40 que la parte más importante es la pestaña "*Button*", más concretamente el apartado "*On Click ()*". En él se configura el salto a otra escena señalando lo siguiente: "*Runtime Only*" y determinando el *script* configurado para el cambio de escena, además del nombre de la escena (Principal). Del mismo modo que el texto anterior, reservaremos la pestaña "*Rect Transform*" y el *script* de cambio de escena para más adelante. Ver Figura 40.

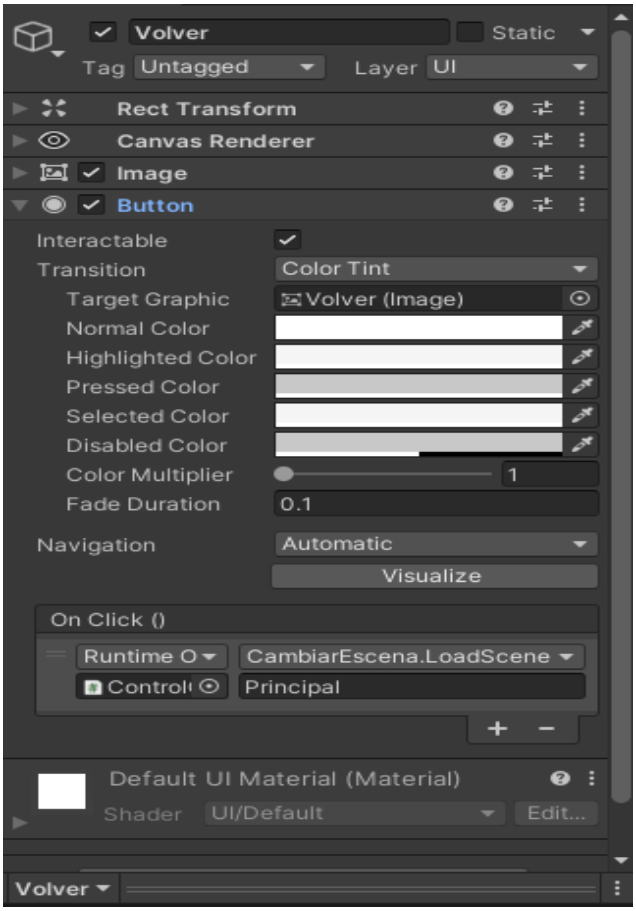

Figura 40: Inspector del botón "Volver Menú".

En este punto tendremos una pantalla de juego plenamente funcional en la que hemos cumplido con el objetivo de la tarea: el usuario podrá moverse por la estancia para encontrar los objetos fuera de lugar y hacer clic sobre ellos para hacerlos desaparecer. Además, lleva el control en todo momento de los objetos que ha eliminado y puede regresar al menú siempre que quiera. A continuación, se muestra un ejemplo de lo que podría ver el usuario durante la simulación. Ver Figura 41 y Figura 42.

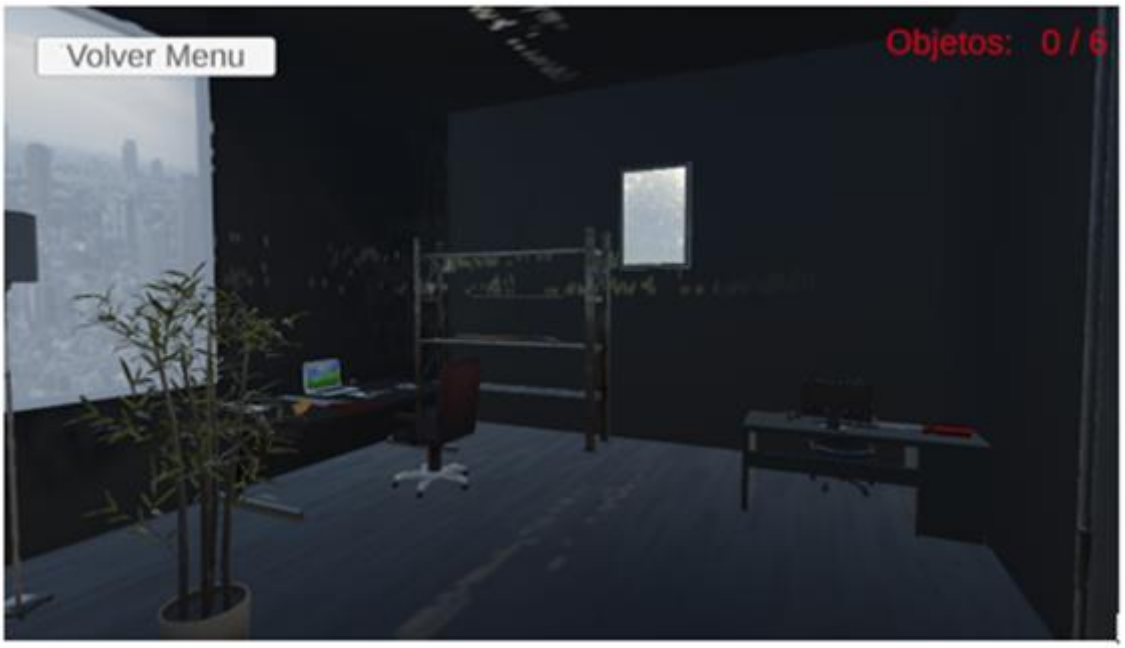

Figura 41: Ejemplo 1 de visualización de la Tarea 1.

Figura 42: Ejemplo 2 de visualización de la Tarea 1

3.3.3 Pantalla TAREA 2

Esta tarea se ha basado en un proceso de mecanizado y ensamblaje de piezas, tratando de simplificar el proceso de fabricación con el fin de ser más fácil su jugabilidad. Como venimos haciendo desde el principio, lo primero será la puesta en

escena de todos aquellos recursos que nos puedan ser de utilidad. Comenzaremos con la parte estructural donde incorporaremos una nave industrial y un patio (Figura 43).

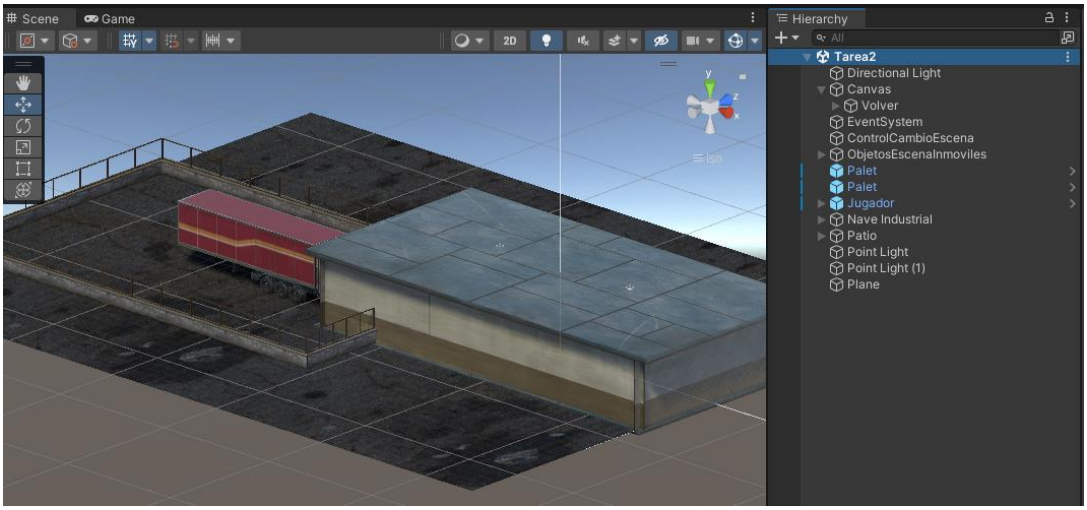

Figura 43: Nave industrial y patio de la tarea 2. Panel de jerarquía completo.

Entre los objetos podemos encontrar máquinas como fresadoras o tornos, unos palés, un robot o incluso una cinta transportadora (Figura 44).

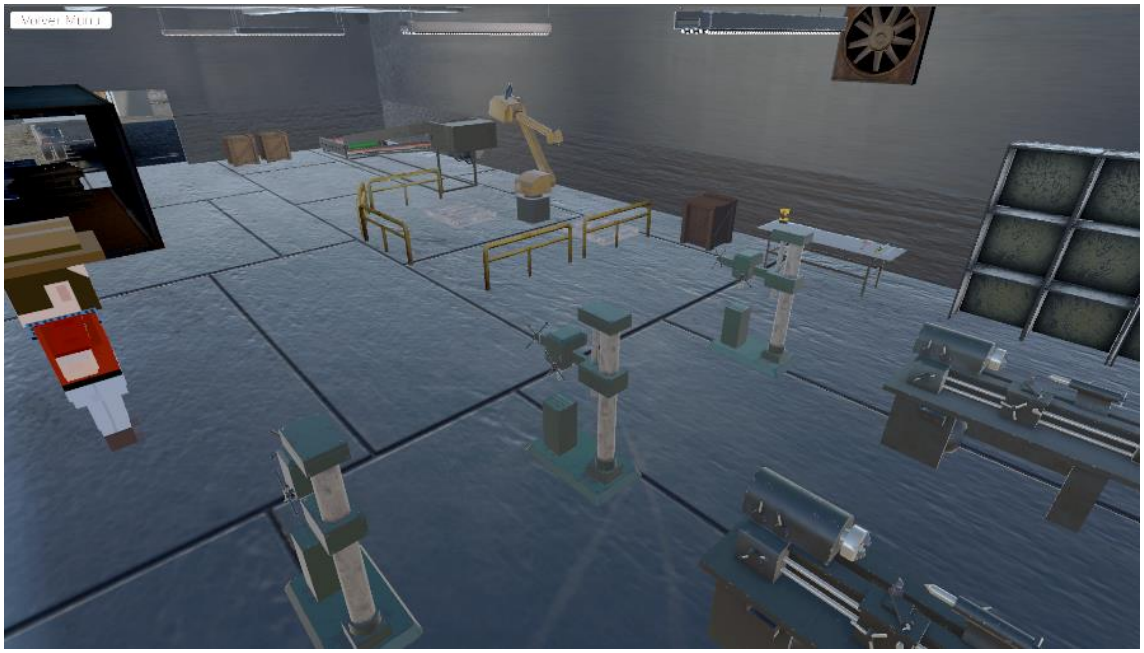

Figura 44: Disposición del taller de la Tarea 2.

El jugador conserva todos los componentes que se le han asignador hasta ahora, es decir, *Rigidbody*, *Box Collider* y el *script* llamado "MoverJugador". Para más información se aconseja regresar al Código 2 donde se ha explicado cuidadosamente este *script* y su funcionalidad. Además, en esta ocasión se le asigna la etiqueta "*Player*" con la que lograremos identificar nuestro personaje en la programación.

En cuanto a la disposición de las paredes del taller y el patio, solo cabe destacar que se han dispuesto con el nombre de "Muro", del mismo modo que en el resto de las escenas. Esto es así porque, como ya se mencionó, en el script asociado al jugador existe una condición para detener el movimiento en caso de que el jugador colisione contra un objeto llamado de esta forma.

Del mismo modo que se hizo para la primera tarea, también se hace uso de los botones de la UI, para colocar un botón que nos lleve de nuevo a la escena de menú. Vale la pena destacar que también se ha creado un objeto vacío al que hemos llamado "ControlCambioEscena" al que se le ha asignado el *script* llamado "**CambiarEscena**".

Por otro lado, a los objetos llamados "Palet" se les ha añadido tres componentes: *Rigidbody*, *Box Collider* y un nuevo *script*. Este se llama "**CogerObjetos**" y su objetivo es la manipulación de este por parte del jugador. El Código 6 queda de la siguiente manera:

Código 6: Coger y trasladar objetos de posición.

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

public class CogerObjetos : MonoBehaviour
{
    public float paletPeso = 2f;
    public float paletVelocidadMovimiento = 50f;
    public float maxDistancia = 2f;

    private bool cogerPalet = false;
    private Vector3 posicionOriginal;
    private Rigidbody paletRigidbody;
    private Rigidbody playerRigidbody;

    private void Start()
    {
        paletRigidbody = GetComponent<Rigidbody>();
        posicionOriginal = transform.position;
        playerRigidbody = GameObject.FindWithTag("Player").GetComponent<Rigidbody>();
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) //Si pulsas la barra de espacio
        {
            if (!cogerPalet) //Si no es es cogerPalet...
            {
                // Si el jugador está lo suficientemente cerca del palet
                if (Vector3.Distance(transform.position, playerRigidbody.position)
                            <= maxDistancia)
                {
                    //Cambiar variables booleanas
                            paletRigidbody.useGravity = false;
                    paletRigidbody.isKinematic = true;
                            //Transformar la posicion
                    transform.position = playerRigidbody.position +
                                        playerRigidbody.transform.forward * 2f;
                    transform.position = new Vector3(transform.position.x, paletPeso,
                                                    transform.position.z);
```

```
                cogerPalet = true;
            }
        }

        else //En caso contrario...
        {
            paletRigidbody.useGravity = true;
            paletRigidbody.isKinematic = false;
            cogerPalet = false;
        }
    }


    if (cogerPalet)
    {     //Mueve el objeto "paletRigidbody" hacia la nueva posición
        Vector3 newPosition = playerRigidbody.position +
                                    playerRigidbody.transform.forward * 2f + new Vector3(0f,
                                            paletPeso, 0f);
        paletRigidbody.MovePosition(Vector3.Lerp(paletRigidbody.position, newPosition,
                    paletVelocidadMovimiento * Time.deltaTime));
    }
  }
}
```

En las primeras líneas del código se definen varias variables públicas que permiten ajustar algunos parámetros del *script*, como el peso del objeto, su velocidad de movimiento y la distancia máxima desde la que se puede coger.

En el método "*Start()*" se inicializan algunas de las variables del *script*, como el componente "*Rigidbody*" del palé, su posición original y el componente "*Rigidbody*" del jugador. El palé se encuentra asociado al *script* a través de la variable "paletRigidbody".

Por su parte, en el método "*Update()*" se comprueba si se ha presionado la barra de espacio por medio de una condición. Si el objeto no se encuentra agarrado por el jugador, se verifica si el jugador está lo suficientemente cerca de él como objeto para cogerlo. Si es así, se desactiva la gravedad y la física del objeto y se mueve hacia la posición del jugador, ajustando su altura para que parezca que lo está sujetando. El objeto se vuelve "fijo" en la posición del jugador y la variable "cogerPalet" se marca como "*true*".

Si el objeto ya se encuentra en posesión del jugador y se presiona la barra de espacio nuevamente, el objeto se suelta y se reactiva la gravedad y la física del objeto para que se comporte de nuevo de manera normal.

Finalmente, si el objeto está en posesión del jugador, se utiliza la función "*MovePosition()*" del componente "*Rigidbody*" del objeto para moverlo hacia una nueva posición frente al jugador, de manera que el objeto siempre esté en frente de él y a una distancia constante. Esto permite que el jugador se mueva mientras sostiene el palé. La velocidad de movimiento del objeto se determina a partir de la variable "paletVelocidadMovimiento". Gracias a esta, podremos dar mayor o menor credibilidad de movimiento de nuestra acción de “coger un objeto”.

3.3.4 Pantalla TAREA 3

Esta es una escena relativamente sencilla en cuanto al diseño de opciones que se le presentan al usuario para elegir, pero resulta bastante complejo en términos de programación. El usuario debe ser capaz de seleccionar y recoger cada caja de manera individual, para luego colocarla en el interior de cada una de las estanterías, respetando en todo momento el color correspondiente a cada una de ellas.

En primer lugar, procederemos a instanciar y colocar los recursos necesarios para esta escena. Para ello, crearemos cinco planos que nos servirán para delimitar el suelo y las paredes del juego. Además, colocaremos una mesa sobre la que apoyaremos un total de doce cajas y doce estanterías colocadas boca arriba en el suelo.

Posteriormente, procederemos a cambiar el color de cada objeto, asegurándonos de que siempre haya una correspondencia entre el número de cajas y estanterías rojas, por ejemplo. Asimismo, cambiaremos el nombre de todos los objetos en función del color que les hayamos asignado, para poder identificarlos posteriormente en el *script*. Y para que quede mucho más limpio y ordenado el panel de jerarquía haremos carpetas donde meter cada objeto en función de la forma que tiene. Una vez colocados todos los objetos, la escena queda como se muestra en la Figura 45.

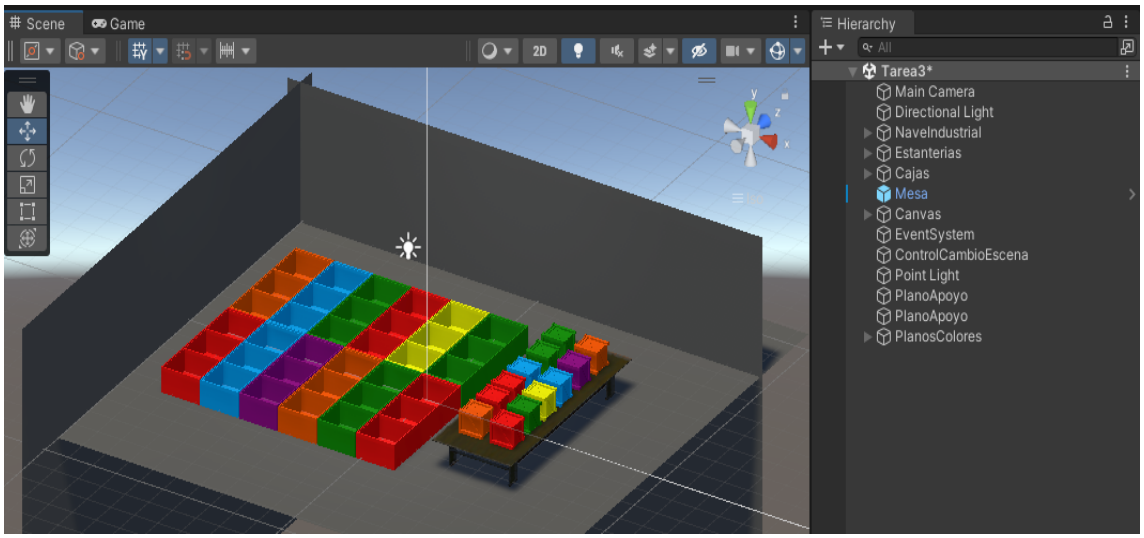

Figura 45: Apariencia de la Tarea 3.

Es importante destacar los componentes que integran cada objeto. En particular, las estanterías deben contar con el componente *Box Collider* asignado. A diferencia de otras situaciones, en este caso, las estanterías poseen varios de estos componentes para permitir la creación de su forma hueca, tal como se observa en la Figura 46.

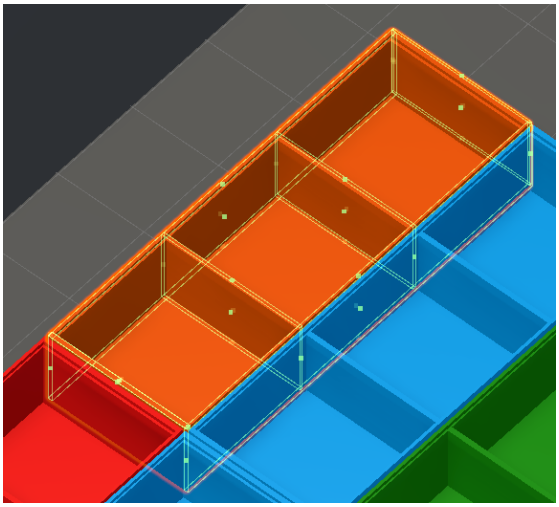

Figura 46: Ubicación del componente Box Collider de las estanterías.

En cuanto a las cajas, se requiere que dispongan de tres componentes: *Rigidbody, Box Collider* y un *script* denominado "**Cajas**". Es necesario marcar "*Is Trigger*" dentro del componente *Box Collider* para poder utilizarlo junto con el mencionado *script*. En Código 7 se recoge el código de la clase Cajas:

Código 7: Control del manejo de las cajas.

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

public class Cajas : MonoBehaviour
{
    private bool estaSiendoAgarrada = false;
    private Vector3 posicionInicial;
    private float distancia = 10f;

    private void Start()
    {
        posicionInicial = transform.position;
    }

    private void OnMouseDown()
    {
        estaSiendoAgarrada = true;
    }

    private void OnMouseUp()
    {
        estaSiendoAgarrada = false;
        GetComponent<Rigidbody>().isKinematic = false;
    }

    private void Update()
    {
        if (estaSiendoAgarrada) // Si se está agarrando la caja...
        {
            Vector3 posicionMouse = Input.mousePosition;
            posicionMouse.z = distancia; //Se guarda la posición del eje z en todo momento

                // Convierte las coordenadas de la posición del mouse en la pantalla en un
```

```
            // vector de posición en el mundo 3D
            Vector3 posicionObjeto = Camera.main.ScreenToWorldPoint(posicionMouse);

            posicionObjeto.y = transform.position.y;

            transform.position = posicionObjeto;
        }
    }

// Se presentan funciones con condicionales que verifican el color de la caja y si
// coincide con el color del plano o no.
// "gameObject.name" se refiere al nombre del objeto que tiene el script adjunto.
// "other.gameObject.name" se refiere al nombre del objeto que colisionó con el
// objeto actual.
    private void OnTriggerEnter(Collider other)
    {
        if (gameObject.name == "CajaVerde")
        {
            if (other.gameObject.name == "Plano Verde" && gameObject.name == "CajaVerde")
            {
                Debug.Log("OK");
            }
            if (other.gameObject.name == "Plano Naranja" || other.gameObject.name ==
                        "Plano Azul" || other.gameObject.name == "Plano Rojo" ||
                other.gameObject.name == "Plano Morado" || other.gameObject.name == "Plano
                        Amarillo" && gameObject.name == "CajaVerde")
            {
                Debug.Log("No es de color Verde");
                transform.position = posicionInicial; //Vuelve al origen
            }
        }
        if (gameObject.name == "CajaAmarilla")
        {
            if (other.gameObject.name == "Plano Amarilla" && gameObject.name ==
                        "CajaAmarilla")
            {
                Debug.Log("OK");
            }
            if (other.gameObject.name == "Plano Naranja" || other.gameObject.name ==
                        "Plano Azul" || other.gameObject.name == "Plano Rojo" ||
                other.gameObject.name == "Plano Morado" || other.gameObject.name == "Plano
                        Verde" && gameObject.name == "CajaAmarilla")
            {
                Debug.Log("No es de color Amarillo");
                transform.position = posicionInicial;
            }
        }
        if (gameObject.name == "CajaMorada")
        {
            if (other.gameObject.name == "Plano Morado" && gameObject.name ==
                        "CajaMorada")
            {
                Debug.Log("OK");
            }
            if (other.gameObject.name == "Plano Naranja" || other.gameObject.name ==
                        "Plano Azul" || other.gameObject.name == "Plano Rojo" ||
                other.gameObject.name == "Plano Verde" || other.gameObject.name == "Plano
                        Amarillo" && gameObject.name == "CajaMorada")
            {
                Debug.Log("No es de color Morado");
                transform.position = posicionInicial;
            }
        }
```

```
        if (gameObject.name == "CajaRoja")
        {
            if (other.gameObject.name == "Plano Roja" && gameObject.name == "CajaRoja")
            {
                Debug.Log("OK");
            }
            if (other.gameObject.name == "Plano Naranja" || other.gameObject.name ==
                        "Plano Azul" || other.gameObject.name == "Plano Verde" ||
                other.gameObject.name == "Plano Morado" || other.gameObject.name == "Plano
                        Amarillo" && gameObject.name == "CajaRoja")
            {
                Debug.Log("No es de color Roja");
                transform.position = posicionInicial;
            }
        }

        if (gameObject.name == "CajaAzul")
        {
            if (other.gameObject.name == "Plano Azul" && gameObject.name == "CajaAzul")
            {
                Debug.Log("OK");
            }
            if (other.gameObject.name == "Plano Naranja" || other.gameObject.name ==
                        "Plano Verde" || other.gameObject.name == "Plano Rojo" ||
                other.gameObject.name == "Plano Morado" || other.gameObject.name == "Plano
                        Amarillo" && gameObject.name == "CajaAzul")
            {
                Debug.Log("No es de color Azul");
                transform.position = posicionInicial;
            }
        }
        if (gameObject.name == "CajaNaranja")
        {
            if (other.gameObject.name == "Plano Naranja" && gameObject.name ==
                         "CajaNaranja")
            {
                Debug.Log("OK");
            }
            if (other.gameObject.name == "Plano Verde" || other.gameObject.name == "Plano
                        Azul" || other.gameObject.name == "Plano Rojo" ||
                other.gameObject.name == "Plano Morado" || other.gameObject.name == "Plano
                        Amarillo" && gameObject.name == "CajaNaranja")
            {
                Debug.Log("No es de color Naranja");
                transform.position = posicionInicial;
            }
        }
        if (other.gameObject.name == "PlanoApoyo" && gameObject.name == "CajaVerde" ||
                        gameObject.name == "CajaNaranja" || gameObject.name == "CajaAzul" ||
            gameObject.name == "CajaRoja" || gameObject.name == "CajaMorada" ||
                         gameObject.name == "CajaAmarilla")
        {
            GetComponent<Rigidbody>().isKinematic = true;
        }
    }
}
```

Como podemos ver, este es un *script* mucho más largo de lo habitual, pero si lo analizamos en profundidad veremos que hay elementos ya conocidos. En esencia el

código nos permite que la caja pueda ser arrastrada por el usuario mientras se hace clic en ella y se mueve el ratón. Si la caja se coloca en el plano correcto, se mostrará un mensaje "OK". Si no es el plano correcto, la caja volverá a su posición inicial y se mostrará un mensaje de error.

En primer lugar, nos encontramos la declaración de tres variables: "estaSiendoAgarrada", que es un indicador booleano que controla si la caja está siendo arrastrada; "distancia", que contiene la distancia entre la caja y la cámara del usuario; y "posicionInicial", que almacena la posición inicial de la caja en el mundo. Esta última se inicializa en el método "*Start()*".

Después nos encontramos los métodos "OnMouseDown()" y "OnMouseUp()", donde la variable "estaSiendoAgarrada" cambia a verdadero y falso respectivamente. También establece "*isKinematic*" en "*false*" para permitir que la caja responda a la física del juego.

Dentro del método "*Update()*" se evalúa una condición para verificar si se está agarrando la caja. De ser así, se obtiene la ubicación del ratón en el mundo del juego. Una vez calculada esta ubicación, el *script* actualiza la posición de la caja para que se desplace hacia dicha ubicación en el mundo. En conclusión, el *script* traduce la posición del ratón en una ubicación global y actualiza el lugar ocupado por la caja para que se desplace hacia esa ubicación en el mundo.

Por último, la función "*OnTriggerEnter*" se utiliza para detectar cuándo un objeto con un "*Collider*" entra en contacto con otro objeto con un "*Collider*". En este caso, la función se utiliza para detectar cuándo una de las seis cajas de diferentes colores (verde, amarilla, morada, roja, azul y naranja) entra en contacto con uno de los siete planos (uno de cada color más un plano de apoyo). De estos planos y su ubicación hablaremos más adelante.

La función comienza con una serie de condicionales que comprueban el nombre del objeto actual, es decir, la caja. Si la caja es de color verde y entra en contacto con el plano verde, se muestra un mensaje de "OK" en la consola. Si la caja es de cualquier otro color y entra en contacto con cualquier otro plano, se muestra un mensaje de "No es de color Verde" en la consola y se mueve la caja de vuelta a su posición inicial.

Por último, la función comprueba si el objeto con el que entra en contacto es el plano de apoyo y, si lo es, desactiva la física de la caja para que quede fija en el lugar de colisión.

En relación con el trabajo que estamos llevando a cabo, es importante destacar la reciente incorporación de unos planos en la escena. Estos planos se han ubicado sobre cada una de las estanterías y tienen asignado el nombre correspondiente al color de la estantería sobre la que se encuentran. Por ejemplo, si los planos se sitúan sobre una estantería de color rojo, su nombre será “Plano Rojo”.

Esta medida tiene como objetivo principal identificar si el usuario está llevando a cabo correctamente el trabajo. Además, como hemos jugado con la activación y desactivación de la física en la simulación, se ha considerado conveniente incorporar un "Plano Apoyo" que permita frenar las cajas en el momento en que entren en contacto con él.

En concreto, se han ubicado dos de estos planos en la estancia: uno en la parte inferior de las estanterías y otro sobre la mesa para poder sostener las cajas. Esta medida facilitará el trabajo y mejorará la eficiencia del proceso que estamos llevando a cabo (Figura 47).

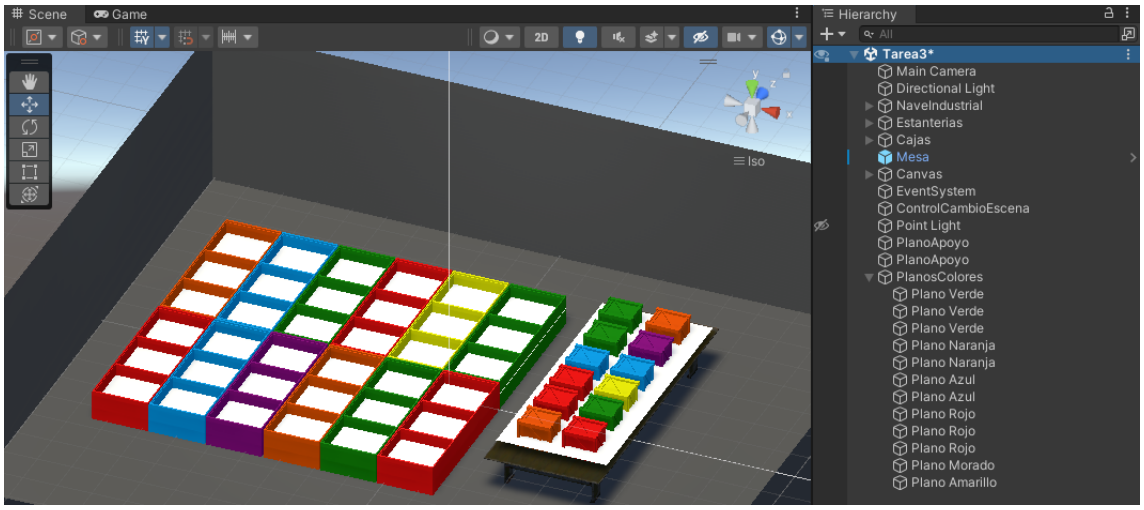

Figura 47: Ubicación de planos de apoyo y de colores.

Finalmente, lo único que quedará por hacer de esta tarea será añadir el botón "Volver Menú" con el método "*Canvas*" del mismo modo que se ha hecho en el resto de las escenas. En la Figura 48 podemos ver la escena de simulación de juego en la que el usuario jugará.

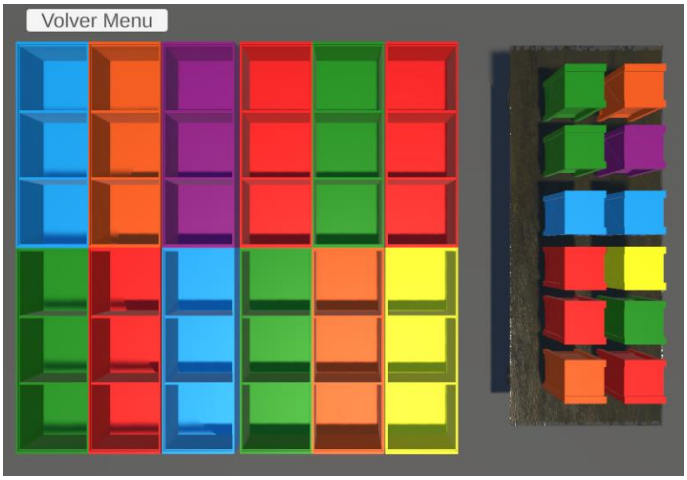

Figura 48 :Escena de simulación de la Tarea 3.

3.3.5 Escenas de menú e instrucciones

Las escenas creadas con el fin de ser solamente de transición, es decir, donde el usuario solo deberá leer el contenido y pulsar algún botón tienen todas ellas la misma estructura. Es por esto, que se explicará una de ellas asumiendo que el resto son una copia de esta original con ligeras modificaciones.

Para la mejor explicación se ha seleccionado la escena del menú donde encontramos la lista de tareas pendientes que puede realizar el usuario. En primer lugar, generaremos un objeto vacío llamado "ControlCambioEscena" que comandará la gestión de esta mediante un script llamado "**CambiarEscena**" (Código 8).

Código 8: Cambio de escena por medio de su nombre.

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

public class CambiarEscena : MonoBehaviour
{
   public void LoadScene (string NombreEscena)
   {
      //Cambio de escena en funcion del Nombre que se le ha puesto en Unity
            SceneManager.LoadScene (NombreEscena);
   }
}
```

Dentro de *script* se define una función pública llamada LoadScene que toma un parámetro de tipo *string* llamado NombreEscena. Esta función utiliza la librería *SceneManager* de Unity para cargar la escena especificada por el parámetro NombreEscena. La especificación exacta de la escena se detallará más adelante.

A continuación, procederemos con una explicación más precisa del uso de la *UI*. *UI* en Unity es un conjunto de herramientas y componentes que permiten a los diseñadores de videojuegos crear interfaces de usuario para sus juegos y aplicaciones. La carpeta *Canvas* es una parte importante de esta interfaz, ya que es el contenedor principal para todos los elementos de la interfaz de usuario.

Dentro de la carpeta *Canvas*, los desarrolladores pueden agregar varios elementos de interfaz de usuario, como texto, imágenes, botones, menús, barras de progreso, etc. En nuestro caso, hemos incorporado un panel, texto y botones. Como nuestro objetivo es que se utilice desde la pantalla de un ordenador, debemos configurarlo para ello. En el *Inspector* de *Canvas*, dentro de la pestaña "*Canvas Scaler*" debemos seleccionar "*Scale With Screen Size*" y colocar las dimensiones que se ven en la Figura 49.

En la configuración de cualquier interfaz de usuario nos encontremos con una pestaña llamada *Rect Transform* dentro del Inspector de Unity. Es una herramienta que se utiliza para ajustar y definir la posición, tamaño y escala de los elementos de la interfaz de usuario.

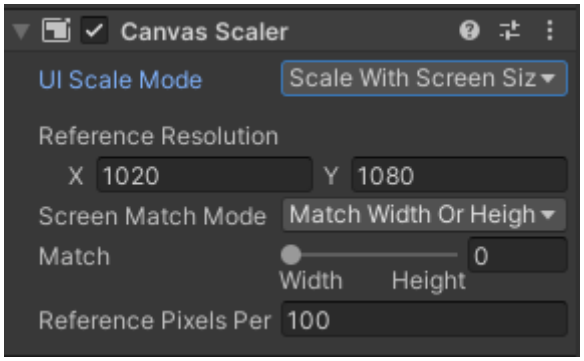

Figura 49: Configuración del tamaño de pantalla.

Por ejemplo, para ajustar la posición de un botón, pueden utilizarse las opciones de posición X e Y del componente *Rect Transform* para definir su ubicación exacta en la pantalla. También pueden utilizarse las opciones de ancho y altura para ajustar su tamaño.

Es importante hablar de los puntos de pivote que tiene cada elemento del *Canvas.* Estos ayudan a ubicar exactamente el lugar que ocupa el elemento dentro de la pantalla. Existe la posibilidad de que simplemente al hacer clic sobre la parte acondicionada para ello dentro del *Rect Transform* la ubiquemos en unas posiciones predefinidas para ello, como vemos en la Figura 50.

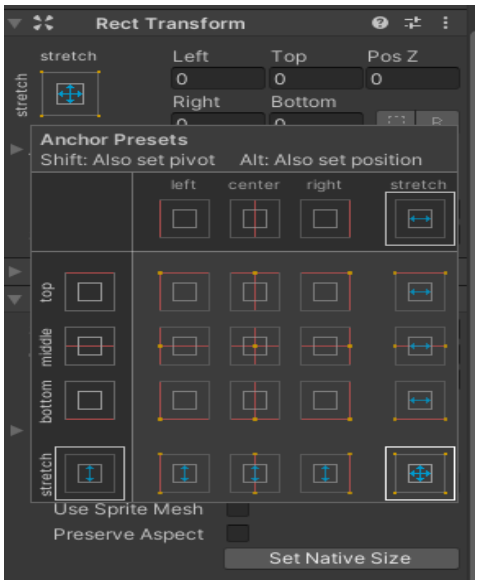

Figura 50 :Ubicación predefinida de un objeto.

Aunque también existe otra posibilidad más recomendada, que consiste en definir de forma manual la ubicación de las diferentes instancias. Para ello debemos coger los puntos de pivote que aparecen en la escena y arrastrarlos hasta el lugar que queremos que aparezcan, tal y como se muestra en Figura 51.

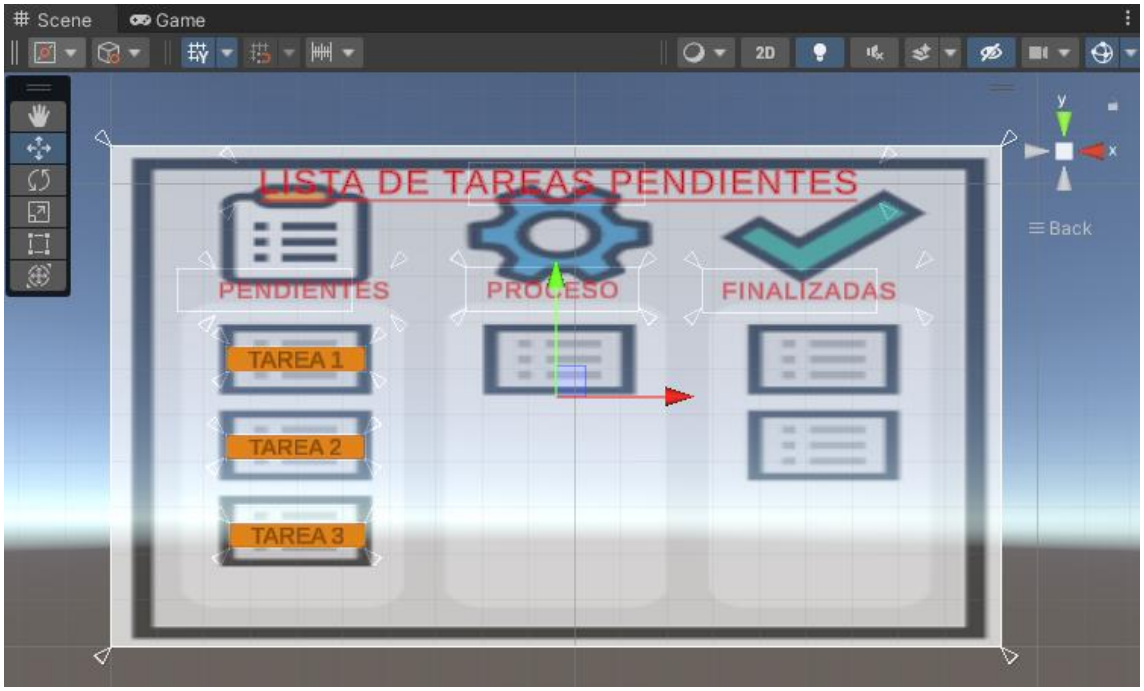

Figura 51: Menú donde vemos los puntos de pivote del Canvas.

Como podemos ver en la Figura anterior, se ha incorporado de fondo una imagen. Para ello debemos incorporar un panel dentro de nuestra carpeta Canvas y acceder al apartado *Image* dentro del Inspector. En él podremos incorporar la imagen haciendo clic sobre *Source Image* y buscando la imagen deseada (Figura 52).

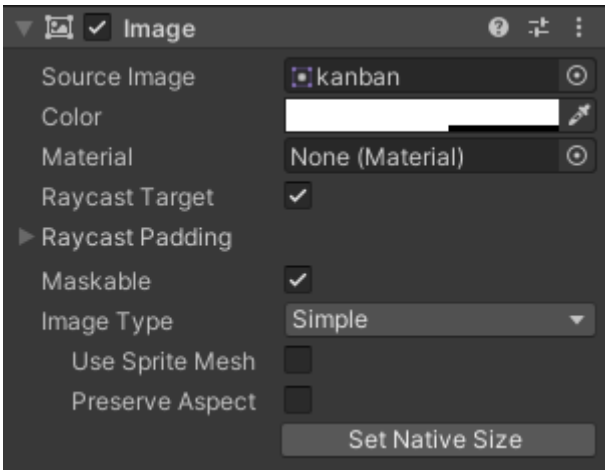

Figura 52: Ventana de configuración de un panel.

Cabe destacar que para poder incorporar imágenes externas a nuestro juego debemos importarlas previamente como un Asset. Esto lo haremos desde la ventana Project: *Botón derecho / Import New Asset… / Buscando la imagen en los archivos.*

En este caso específico, en el que la imagen será utilizada como fondo de un menú o panel de instrucciones en Unity, es importante realizar un proceso de configuración adicional después de importarla. Desde la ventana *Project*, es necesario seleccionarla y

abrir el Inspector de Unity. Luego, se debe cambiar el tipo de textura de la imagen a "*Sprite (2D and UI)*" en la sección "*Texture Type*".

Este cambio asegura que la imagen sea adecuada para su uso en la interfaz de usuario y permitirá que se pueda ajustar su tamaño y posición utilizando el componente *Rect Transform*, tal y como se explicó anteriormente. Es importante guardar los cambios después de realizar esta configuración.

Para terminar, hablaremos de la configuración de los botones. Esta, como viene siendo habitual, se hará desde la ventana del Inspector. En la pestaña *Button* (Figura 53) hay varias secciones que permiten configurar el aspecto y el comportamiento del botón.

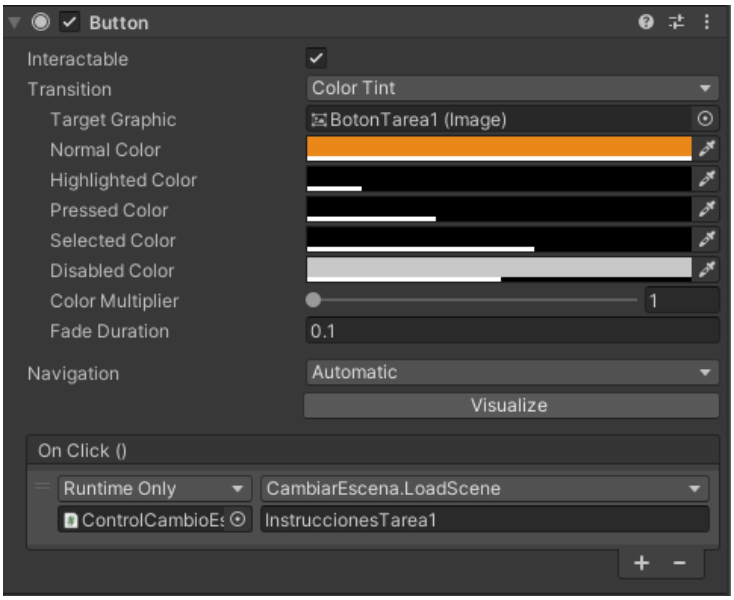

Figura 53: Configuración de los botones.

La sección *Transition* permite seleccionar el tipo de transición que se utilizará cuando el botón cambie de estado; por ejemplo, de normal a presionado. Hay varios tipos de transición disponibles, como *Color Tint*, *Sprite Swap*, *Animation*, etc. Para este proyecto siempre se ha utilizado *Color Tint*, que es la opción que viene por defecto. Además, dentro de *Transition* vamos a poder modificar el color del botón y su transparencia al pasar por encima el ratón. En todo momento se han configurado los botones para que, si el usuario pasa por encima el ratón, pueda llamar su atención cambiando ligeramente de color. Para ello, como vemos en Figura 53, la transparencia ha ido aumentando.

La parte "*On Click*" del botón *UI* de Unity permite asignar un *script* y una función que se ejecutará cuando el botón sea presionado. En el ejemplo de la Figura anterior se ha asignado un *script* llamado "ControlCambioEscena" y se ha configurado para cambiar la escena a "InstruccionesTarea1". Cuando se presiona el botón en la escena, se ejecuta la función "*LoadScene*" del *script* "ControlCambioEscena" (visto anteriormente), pasando como argumento el nombre de la escena a la que queremos cambiar (en este caso, "InstruccionesTarea 1").

Este script no es la única opción existente. En la escena principal, cuando el usuario entra en contacto con la pizarra, automáticamente se produce un salto a la escena donde se realiza la elección de la tarea. En este caso, el *script* asignado a la pizarra es el Código 9:

Código 9: Cambio de escena hacia la pantalla del menú directamente.

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

public class CargarMenuCambioEscena : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Player"))
        {
            SceneManager.LoadScene(2);
        }
    }
}
```

En el interior de este código, se comprueba si el objeto con el que se ha producido la colisión tiene la etiqueta (*tag*) "*Player*" utilizando el método "*CompareTag*". Si el objeto tiene dicha etiqueta, se carga la escena número dos utilizando la clase *SceneManager* de la librería de Unity. Esta cifra se asigna de forma automática cuando vamos a generar un ejecutable del juego.

A continuación, se muestran las diferentes escenas por las que el usuario irá pasando al jugar (véase Figura 54, Figura 55, Figura 56 y Figura 57).

Figura 54: Pantalla de instrucciones iniciales del juego.

Figura 55: Pantalla de instrucciones Tarea 1.

Figura 56: Pantalla de instrucciones de la Tarea 2.

Figura 57: Pantalla de instrucciones Tarea 3.

3.3.6 Generando el ejecutable del juego

Para generar un ejecutable del juego de Unity que permita un libre acceso desde cualquier ordenador sin necesidad de tener instalado ningún programa, es necesario seguir los siguientes pasos:

En primer lugar, se debe acceder a la opción "*Build Settings*" en el menú "*File*". En esta sección, es importante verificar que todas las escenas que se quieren incluir en el ejecutable estén en la sección "*Scenes In Build*" (Escenas en compilación). Para añadir una escena a la lista, es necesario hacer clic en "*Add Open Scenes*". De forma automática, se le asignará un número a cada escena, el cual será utilizado en el script "CargarMenuCambioEscena".

A continuación, se debe seleccionar la plataforma de destino para la cual se desea crear el ejecutable en la sección "*Platform*" y hacer clic en "*Switch Platform*" (Cambiar plataforma). Una vez seleccionada la plataforma, se deben ajustar las opciones de compilación según las necesidades en el botón "*Player Settings…*". En esta sección, se pueden seleccionar la arquitectura, la calidad de gráficos y otros ajustes.

Una vez ajustadas las opciones de compilación, se debe hacer clic en "*Build*" (Compilar) para generar el ejecutable. Al hacer esto, se solicitará que se seleccione una ubicación para guardar el archivo. Una vez seleccionada la ubicación, se debe hacer clic en "Guardar" y Unity comenzará a compilar el juego.

Finalmente, después de que Unity haya terminado de compilar el juego, se generará un archivo ejecutable (tipo .exe) que se encontrará en la ubicación seleccionada en el

paso anterior. Este archivo podrá ser utilizado en cualquier ordenador sin necesidad de instalar ningún programa adicional.

Es importante tener en cuenta que la generación de un ejecutable puede tardar varios minutos, dependiendo del tamaño del proyecto y de la plataforma de destino. Además, es recomendable probar el ejecutable en la plataforma de destino antes de distribuirlo para poder verificar su comportamiento.

4 ESTUDIO ECONÓMICO

4.1 Introducción

El presente estudio económico tiene como objetivo la evaluación aproximada de los costes necesarios para el desarrollo del Trabajo Fin de Grado (TFG). Para lograr dicho objetivo se ha procedido a desglosar la elaboración del TFG en cada una de las diferentes etapas que la componen, desde su planteamiento inicial hasta su terminación y presentación de los resultados.

En la valoración de costes se han considerado las horas necesarias dedicadas por profesionales en una empresa especializada en la elaboración de trabajos académicos, así como los recursos materiales empleados en el desarrollo de cada una de las etapas. De esta forma, se ha llevado a cabo una estimación realista de los costes que se deben asumir para la elaboración del TFG.

Cabe destacar que la presente evaluación económica es aproximada, ya que depende de diversos factores, como la complejidad del tema abordado, el nivel de detalle requerido por la institución académica y las especificaciones del tutor o director del trabajo. No obstante, esta evaluación sirve como una guía orientativa para el estudiante en cuanto a los costes que puede esperar en la elaboración de su TFG.

4.2 Profesionales partícipes del proyecto

En la elaboración de un proyecto de estas características, intervienen diversas personas con roles y cargos definidos, recogidos en la Figura 58.

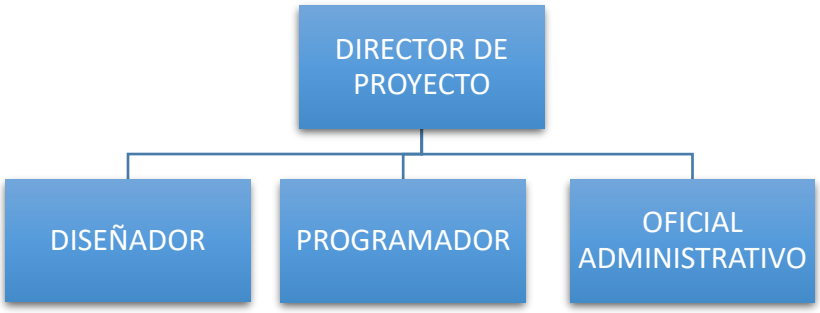

Figura 58: Organigrama de roles de la empresa

En primer lugar, se encuentra el **Director de Proyecto**, que es el responsable máximo del mismo. Este profesional se encarga de proporcionar las ideas generales, coordinar, aconsejar y guiar al resto del equipo en todo el desarrollo a partir de su amplia experiencia. Asimismo, será el encargado de aprobar y validar finalmente el proyecto.

El Director de proyecto es un profesional con habilidades de liderazgo, capacidad de toma de decisiones y amplios conocimientos en el campo en el que se desarrolla el proyecto. Además, debe tener una visión amplia del proyecto y de sus objetivos, así como la capacidad de comunicarla claramente al resto del equipo.

Es importante destacar que, si bien el Director de proyecto es el responsable máximo, su labor no debe ser considerada como una tarea individual, sino como una labor de equipo. Es fundamental que el resto del equipo de trabajo participe activamente en la toma de decisiones y en el desarrollo del proyecto, y que el director de Proyecto actúe como un facilitador y coordinador del equipo, fomentando la colaboración y el trabajo en equipo para lograr los objetivos del proyecto de manera eficiente y efectiva.

Después del Director de Proyecto, en este caso, se encuentran los dos **ingenieros**: el **diseñador de videojuegos** y el programador del videojuego. El primero es el responsable de diseñar la jugabilidad, el argumento y la mecánica de los juegos; en nuestro caso, además es responsable de área. Debe tener una comprensión profunda de cómo funcionan los juegos y cómo los jugadores interactúan con ellos. El **programador**, por su parte, es responsable de escribir el código que hace que los videojuegos funcionen. Debe tener experiencia en lenguajes de programación y en el uso de herramientas de desarrollo de juegos.

El responsable de área a partir de sus conocimientos en organización y logística lleva a cabo el trabajo más complejo y extenso de exploración y desarrollo para cumplir con las especificaciones marcadas y los objetivos finales. Asimismo, se encargan de realizar

la búsqueda de información, estructuración, redacción y corrección de los documentos pertinentes.

Es importante destacar que el trabajo de los ingenieros no es independiente, sino que está en estrecha colaboración con el resto del equipo de trabajo. La comunicación y la colaboración efectiva son fundamentales para lograr los objetivos del proyecto en tiempo y forma. Por tanto, los ingenieros trabajan en equipo, incluyendo al director de proyecto, para llevar a cabo un proyecto exitoso y cumplir con las especificaciones y objetivos establecidos.

En el equipo de trabajo también se precisa de una persona encargada de las funciones burocráticas: el **Oficial Administrativo**. Se encarga de las tareas administrativas y burocráticas que permiten al resto del equipo centrarse en el trabajo específico del proyecto. Su labor es fundamental para el buen funcionamiento de la empresa y para que el proyecto se desarrolle en tiempo y forma.

4.3 Definición de las fases

Este TFG se basa en la estructura de empresas como Ubisoft o 2K Games y se estructura en cuatro etapas (Figura 59). La primera es la planificación inicial, donde se definen los componentes, recursos, procesos y plazos del proyecto. La segunda es la recopilación de información, donde se busca material fiable para establecer el marco de referencia y enfoque del proyecto. La tercera etapa corresponde al desarrollo práctico, donde se diseña y construye el videojuego. La última etapa es la descripción y documentación de todo el proceso en una memoria teórica para su entrega al cliente.

Figura 59: Fases de elaboración del proyecto

4.4 Análisis económico

Este apartado está reservado para la valoración económica del presente TFG, teniendo en cuenta todos los recursos necesarios para su perfecto desarrollo. Esta evaluación tendrá en cuenta los costes directos e indirectos asociados a: el personal, los equipos informáticos, materiales … Además, se hará un estudio económico de cada una de las partes de manera individual con el fin de poder valorar la influencia de cada parte.

4.4.1 Horas efectivas y tasas horarias del personal

En esta sección se obtienen las tasas por hora y por semana de cada uno de los profesionales que participan en la realización de este proyecto, para la valoración de los costes asociados al personal.

Inicialmente se determinan las fechas inicio y finalización de proyecto para poder determinar la cantidad de días que existen entre dichas fechas. A continuación, se calcula la cantidad de sábados, domingos y días festivos nacionales. Finalmente, se hace un cálculo de los días hábiles que se han trabajado.

Es importante destacar que el cálculo de horas efectivas de ha hecho como si la jornada laboral hubiera sido de nueve horas, aunque realmente haya sido de ocho. Esto se hace así para compensar los días que se ha trabajado en fin de semana y no ha quedado reflejado en la Tabla 1 que se presenta a continuación.

Tabla 1: Cálculo de las horas/días/semanas hábiles en el periodo

Concepto	Valor
Día inicio periodo	01/02/2023
Día fin de periodo	30/04/2023
Periodo (días)	88
Sábados y Domingos	28
Festivos	3
Vacaciones	2
Días de baja medica	1
TOTAL DÍAS HÁBILES	54
TOTAL HORAS EFECTIVAS	486
TOTAL SEMANAS HÁBILES	10,8

Los trabajadores ya han sido definidos anteriormente por lo que en esta sección se procede a su presentación de costes del personal. Ver Tabla 2.

Tabla 2: Cálculo de coste de personal por hora/mes/año.

Concepto	%	Director Proyecto	Responsable de área	Programador	Oficial Administrativo
Salario bruto anual (€/año)		45000	27000	21000	18000
Salario bruto mensual (€/mes)		3750	2250	1750	1500
Seguridad Social (%)	23		517,5	402,5	345
	35	1312,5			
Coste bruto (€/hora)		23,4375	14,0625	10,9375	9,375
Coste bruto (€/semana)		937,5	562,5	437,5	375

4.4.2 Amortizaciones del equipo informático

En la Tabla 3 se pueden ver los costes asociados a los equipos informáticos, hardware y software, necesarios para el óptimo desarrollo de este proyecto. El periodo de amortizaciones considerado es de cinco años los, con una cuota lineal, por lo que los costes totales se reparten de forma equitativa entre esos cinco años. Sin embargo, en este proyecto la amortización de los equipos informáticos se hace de forma proporcional a su duración.

Además, el material de hardware escogidos no es estrictamente necesarios ese mismo modelo, pero sí recomendable dadas sus características. Por ello, los cálculos deben tenerse en cuenta como un cálculo económico para un proyecto óptimo (Tabla 3).

Tabla 3: Cálculo de amortizaciones de los equipos informáticos.

Amortizaciones de equipo	**Modelo**	**Coste (€/und)**	**Unidades**	**Coste total (€)**
Hardware				
Ordenador	DELL XPS 15 Intel Core i7	1798,98	4	7195,92
Ratón	Logitech G502 Hero	92,99	4	371,96
Impresora	DCP-L3550CDW	410,69	1	410,69
Software				
Licencia Windows		220,99	4	883,96
Licencia Office	€/mes	5,6	4	67,2
Licencia Unity		0	3	0
TOTAL DE AMORTIZACIONES				8929,73

Tipo	**Numero**	**Amortización**
Anual (años)	5	1785,95
Mensual (meses)	3	446,49

La cantidad total que amortizar, durante el período de 3 meses de elaboración del TFG, correspondiente a los equipos informáticos adquiridos es de 446,49 €.

4.4.3 Costes de material consumible

Los costes consumibles se resumen en la Tabla 4. En base a un consumo medio por persona, se ha determinado que el coste del material consumible es de 0,20€ por persona y una hora de trabajo.

Tabla 4: Cálculo de los costes material consumible por trabajador/hora.

Material	Concepto	Coste (€)	Unidades	Coste total (€)
Papel Impresora	€/500 folios- 1 paquete	3,95	1	3,95
Tóner	PACK 4 TÓNER BROTHER TN247	49,95	2	99,9
Papelería y Reprografía		290	1	290
TOTAL				393,85
Coste total (€/persona)				98,46
Coste (€/persona*hora)				0,2

4.4.4 Costes indirectos

Estos costes están basados en los servicios básicos, como la electricidad, agua, calefacción, alquiler del local, mobiliario de la oficina, el contrato de servicio telefónico e internet entre otros. En la Tabla 5 se determina que el coste total es de 3105€, que corresponde con la duración aproximada de tres meses de la elaboración del TFG.

Tabla 5: Costes indirectos de servicios.

Coste Indirectos		
Concepto	Coste (€/mes)	Coste periodo
	700	2100
Teléfono e internet	60	180
Electricidad, agua, calefacción	175	525
Otros (Seguros)	100	300
	TOTAL (€)	**3105**

4.4.5 Tiempo asociado a cada fase del proyecto

A continuación, se presentarán los tiempos que cada individuo ha dedicado al proyecto. Además, se ha elaborado en las cuatro fases del proyecto que se describieron anteriormente. Esta segregación de los tiempos dedicados a cada una de las fases permite determinar el coste derivado de las mismas sobre el total del proyecto. Por ello, se calcula la relación de horas que representa cada fase sobre el total del TFG (Tabla 6).

Tabla 6: Cálculo de las horas de trabajo en función de la fase del proyecto.

Personal	FASE 1 (h)	FASE 2 (h)	FASE 3 (h)	FASE 4 (h)
Director Proyecto	10,9	18,2	62,0	30,4
Responsable área	14,0	23,3	79,3	38,9
Programador	14,0	23,3	79,3	38,9
Oficial Administrativo	4,8	8,0	27,3	13,4
TOTAL (horas/fase)	44	73	248	122
TOTAL (horas)	486			
Relación (% fase/ total)	9	15	51	25

4.5 Costes asignados a cada fase del proyecto

Para calcular los costes de cada fase del proyecto se tiene en cuenta el tiempo dedicado por los empleados y los costes establecidos en el análisis económico. El coste estimado de cada etapa incluye el coste del personal, el material consumible, la amortización de los equipos informáticos y los costes indirectos de la organización.

Los costes de servicios indirectos y amortización son cantidades fijas mensuales que se asignan de forma proporcional al número de horas dedicadas a cada fase en relación con el total de horas del proyecto.

4.5.1 Planificación inicial

En esta primera fase del proyecto (véase la Tabla 7) la mayor intervención es por parte de los ingenieros, aunque también participa tanto el Director de Proyecto, como el Oficial Administrativo (este último en menor medida). Durante este periodo, se establece la metodología de trabajo a seguir, se definen los objetivos, organizan los recursos y se estructura el proyecto. Por la parte administrativa comienzan los trámites burocráticos del proyecto.

Tabla 7: Costes total de la fase 1.

Concepto		***Horas***	***Coste (€/h)***	***Coste (€)***
Personal	*Director Proyecto*	*10,9*	*23,4*	*256,3*
	Responsable	*14,0*	*14,1*	*196,8*
	Programador	*14,0*	*10,9*	*153,1*
	Administración	*4,8*	*9,4*	*45,1*
Material Consumible	*Varios*	*44*	*0,2*	*8,7*
		Cantidad	*Porcentaje (%)*	*Coste(€)*
Amortización	*Equipo informático*	*446,5*	*0,09*	*40,2*
Costes indirectos	*Servicios*	*3105,0*	*0,09*	*279,5*
			Coste Total FASE 1	*979,7*

4.5.2 Recopilación de información

En esta segunda fase, la principal tarea la realiza el programador y el diseñador de videojuegos, ya que estos se encargarán de todo el proceso de investigación y búsqueda de documentación de múltiples fuentes. Por otro lado, el director de proyecto y el personal administrativo participan a una menor escala de manera eventual, ayudando con algún asunto o resolviendo posibles inconvenientes. En la siguiente tabla se muestran los costes particulares de esta segunda etapa de proyecto (Tabla 8).

Tabla 8: Costes total de la fase 2.

Concepto		***Horas***	***Coste (€/h)***	***Coste (€)***
Personal	*Director Proyecto*	*18,2*	*23,4*	*427,1*
	Responsable	*23,3*	*14,1*	*328,1*
	Programador	*23,3*	*10,9*	*255,2*
	Administración	*8,0*	*9,4*	*75,2*
Material Consumible	*Varios*	*73*	*0,2*	*14,6*
		Cantidad	*Porcentaje (%)*	*Coste(€)*
Amortización	*Equipo informático*	*446,5*	*0,15*	*67,0*
Costes indirectos	*Servicios*	*3105,0*	*0,15*	*465,8*
			Coste Total FASE 2	*1632,8*

4.5.3 Desarrollo del proyecto

La fase principal del proyecto implica la contribución y participación de todos los empleados de la organización, y es la más compleja y prolongada. Los Ingenieros son los principales responsables de su desarrollo, aunque a menudo requieren la cooperación y coordinación del Oficial Administrativo y el Director (Tabla 9).

Tabla 9: Costes total de la fase 3.

Concepto		***Horas***	***Coste (€/h)***	***Coste (€)***
Personal	*Director Proyecto*	*62,0*	*23,4*	*1452,3*
	Responsable	*79,3*	*14,1*	*1115,4*
	Programador	*79,3*	*10,9*	*867,5*
	Administración	*27,3*	*9,4*	*255,6*
Material Consumible	*Varios*	*248*	*0,2*	*49,6*
	Cantidad		*Porcentaje (%)*	*Coste(€)*
Amortización	*Equipo informático*	*446,5*	*0,51*	*227,7*
Costes indirectos	*Servicios*	*3105,0*	*0,51*	*1583,6*
			Coste Total FASE 3	*5551,6*

4.5.4 Elaboración de la documentación

Esta es la última fase del proyecto. Requiere una alta colaboración por parte de todos los miembros del proyecto. En la Tabla 10 se exponen los cálculos derivados de la documentación y registro del TFG.

Tabla 10: Costes total de la fase 4.

Concepto		***Horas***	***Coste (€/h)***	***Coste (€)***
Personal	*Director Proyecto*	*30,4*	*23,4*	*711,9*
	Responsable	*38,9*	*14,1*	*546,8*
	Programador	*38,9*	*10,9*	*425,3*
	Administración	*13,4*	*9,4*	*125,3*
Material Consumible	*Varios*	*122*	*0,2*	*24,3*
		Cantidad	*Porcentaje (%)*	*Coste(€)*
Amortización	*Equipo informático*	*446,5*	*0,25*	*111,6*
Costes indirectos	*Servicios*	*3105,0*	*0,25*	*776,3*
			Coste Total FASE 4	*2721,4*

4.6 Resultados finales

Este punto tiene como objetivo mostrar al lector, con una visión resumida y global, todos los gastos económicos explicados en los puntos anteriores, basándose en el número de horas que lleva cada fase y el coste asociado a cada fase.

Como se puede ver en la Tabla 11, la elaboración del presente TFG se ha obtenido con la suma de todos los tiempos de todas las etapas. En esa misma tabla se realiza un desglose dedicado a las cuatro fases, dando como resultado final un valor de 486 horas y un valor económico de 10 885,5 € (véase Figura 60).

Tabla 11: Costes totales divididos por fases.

RESULTADOS FINALES

Fases	Tiempo (h)	Coste (€)
1 - Planificación inicial	44	979,7
2 - Recopilación de información	73	1632,8
3 - Desarrollo del proyecto	248	5551,6
4 - Documentación técnica	122	2721,4
TOTAL	486	10885,5

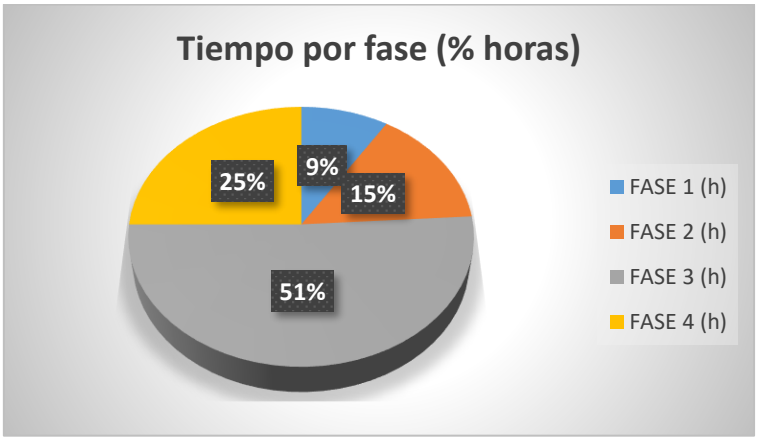

Figura 60: Representación de los tiempos de cada fase sobre el total del TFG.

Como era de esperar, la mitad del tiempo necesario ha sido empleado en la etapa principal de desarrollo del proyecto, y una cuarta parte en la elaboración de la memoria y el resto de documentación. No obstante, las otras dos etapas fundamentales para el desarrollo del TFG requieren la suficiente dedicación de tiempo (24%).

La Tabla 12 está organizado en función de las categorías que hemos estado utilizando durante todo el análisis económico del TFG. En este podemos ver que el mayor porcentaje de gasto pertenece a los costes directos debidos a los salarios del personal (66,48%) y a los costes indirectos (28,58 %) que se explicaron anteriormente. La Figura 61 muestra la contribución de cada uno de estos costes al gasto total.

Tabla 12: Costes divididos por categorías.

Categorías	Coste total (€)	Coste total (%)
Director Proyecto	2847,66	26,16
Responsable	2187,00	20,09
Programador	1701,00	15,63
Administración	501,19	4,60
Equipo informático	446,49	4,10
Costes indirectos	3105,00	28,52
TOTAL (€)	10885,5	

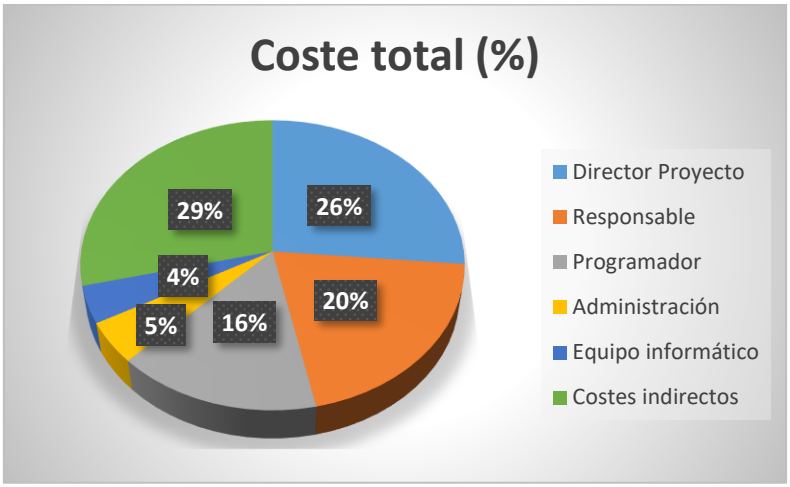

Figura 61: Análisis de los costes por categorías.

Por último, en la Tabla 13 se ha hecho una simulación de ventas del proyecto, en la que se obtiene un beneficio del 30% del total del proyecto y de un 21% de impuestos (IVA). Por lo que el coste final de este proyecto de fin de grado correspondería a 16 437.15 €.

Tabla 13: Coste total del proyecto.

TOTAL		**10885,53**
Beneficio	30%	3265,66
I.V.A.	21%	2285,96
COSTES TOTAL DEL PROYECTO		16437,15

CONCLUSIONES Y LÍNEAS FUTURAS

Conclusiones

Tras la lectura del presente documento se puede afirmar que se ha cumplido el objetivo planteado al comienzo. Se han diseñado y desarrollado videojuegos basados en el modelo físico de industria 4.0 de FischerTechnik y en algunas de las herramientas Lean. Estos juegos educativos pueden ser un instrumento muy útil para apoyar el aprendizaje de los estudiantes en torno a estas herramientas de un modo divertido e interactivo.

Para ello, y como se ha podido observar, se ha realizado con éxito un análisis inicial de las posibilidades del software y hardware que podrían ser útiles. A continuación, y de una manera crítica, se ha seleccionado Unity como principal herramienta de trabajo. Además, tras unas jornadas de autoaprendizaje a través de foros, vídeos, manuales y cursos formativos relacionados con este material, se establecieron los conocimientos básicos de diseño y programación orientada a objetos con los que se pudo comenzar a trabajar.

Es importante destacar, una vez más, que la autoformación obtenida ha sido vital para poder comenzar a trabajar, ya que, hasta este momento, no se había trabajado nunca con herramientas de este tipo. Es decir, a lo largo de la carrera de ingeniería mecánica no se trabaja con programas que tengan un diseño tridimensional a la par de una programación de código que simule el movimiento. Lo más cercano fueron las asignaturas de diseño asistido por ordenador y la asignatura de programación de C++; en particular, ha sido útil para trabajar con el lenguaje empleado en este proyecto: C#.

Mediante la utilización de la maqueta de industria 4.0 de FischerTechnik, empresa que nos ha proporcionado los diseños tridimensionales creados mediante SolidWorks, se ha logrado la creación de un juego basado en la industria 4.0. Este juego presenta un mundo virtual industrial a escala real que permite al usuario visualizar con precisión toda la fábrica y desplazarse libremente por ella.

Adicionalmente, se han desarrollado varias escenas interconectadas que constituyen minijuegos educativos en los que se trabajan algunas herramientas del Lean Manufacturing, como el método Kanban, las 5S y la teoría de los cinco ceros. Con el objetivo de aumentar su atractivo visual, se han incorporado entornos industriales, ya que los videojuegos deben ser entretenidos, visualmente atractivos y no excesivamente complejos en función de los objetivos establecidos.

Es importante destacar que, a pesar de ser un proyecto básico de videojuegos y programación, el trabajo realizado es completamente funcional y puede ser utilizado con fines didácticos desde el momento de su publicación. No obstante, se debe tener en cuenta las limitaciones que presentan, dado que no se buscó la generación de gráficos complejos ni una jugabilidad superior a la de un ordenador común.

Este proyecto sienta las bases para futuras líneas de trabajo, en las que se puede aplicar este mismo material para la creación de nuevos diseños y ampliaciones de código para generar juegos de realidad virtual o aumentada. Como se mencionó anteriormente, la elección de Unity ofrece la ventaja de poder trabajar con gafas de realidad virtual en futuras generaciones del proyecto.

Bajo mi punto de vista, Unity es un programa de desarrollo que ayuda a dar vida a materiales diseñados en otros programas. A pesar de venderse como un programa de diseño, realmente no es así, ya que necesitas tener unos diseños 3D de partida con los que poder trabajar o, en su defecto, tener conocimientos de otros programas, como por ejemplo Blender, con los que comenzar haciendo un diseño tridimensional de los personajes y las escenas. En este TFG los diseños se han obtenido de la tienda que ofrece Unity (*Assets Store*) a todos sus usuarios de manera gratuita, lo que constituye una clara ventaja frente a otras plataformas de desarrollo.

La realización de un TFG de estas características aporta al creador unas habilidades que antes no poseía ya que desde un principio se han analizado y utilizado programas que, como se ha comentado, no se enseñan en la carrera. Esto permite desarrollar habilidades como la disciplina, la capacidad de análisis y capacidad crítica, síntesis, esfuerzo continuo, motivación. Cabe destacar los conocimientos nuevos aprendidos acerca de la cuarta revolución industrial (industria 4.0), así como de las herramientas de Lean Manufacturing. Entre ellas a destacar en este documento el método Kanban, y las filosofías de 5S y cinco ceros. Sin olvidar que la creación de la memoria ayuda a mejorar la capacidad escrita, e incrementa la experiencia en el uso de herramientas ofimáticas como MS Office (Word y Excel). Además, mejora las habilidades de

búsqueda de información, permite ver la importancia de la citación y nos inicia en estudios económicos que ayudan a conocer el valor y el posible alcance del trabajo realizado.

Líneas futuras

Una vez analizado y desarrollado el proyecto, es importante hacer una orientación sobre los factores que podrían ser relevantes a la hora de profundizar en temas relacionados. A continuación, se describen seis aspectos a considerar para los futuros proyectos relacionados con este trabajo.

En primer lugar, se podría profundizar en la explicación y uso de diferentes recursos y elementos que Unity ofrece y que no fueron utilizados en este proyecto. Algunos ejemplos podrían ser la creación de animaciones o la realización de trailers de animación, entre otros.

También, se podría explorar el uso de software de diseño de videojuegos como Blender o Cinema 4D para la realización de mundos virtuales más complejos y detallados.

En tercer lugar, se podría considerar la implantación de tecnologías como la Realidad Virtual (RV) o la Realidad Aumentada (RA) en los juegos propuestos para lograr una experiencia inversiva más completa para el usuario.

Además, se podría explorar la implantación de los juegos para que puedan ser reproducidos desde otros dispositivos, como teléfonos móviles o *tablets*, incluida la opción de RV o RA.

En quinto lugar, se podría trabajar en la mejora de la interfaz de los juegos, así como de los códigos proporcionados a lo largo del documento para lograr un mejor rendimiento.

Por último, se podría recomendar la práctica con los juegos de aprendizaje proporcionados en el Anexo de este documento para profundizar en el conocimiento y manejo de Unity, así como para adquirir nuevas habilidades y destrezas en el diseño y desarrollo de videojuegos.

BIBLIOGRAFÍA

Afonso, E. C. (2018). Sistema de adiestramiento en tareas de ensamblado mediante realidad virtual y mineria de procesos. Recuperado el 6 de Marzo de 2023

Arango Serna, M. D., Campuzano Zapata, L. F., & Zapata Cortes, J. A. (11 de Mayo de 2015). Mejoramiento de procesos de manufactura utilizando Kanban. *Revista Ingenierías Universidad de Medellín*, págs. 222-233. Recuperado el 5 de Marzo de 2023

Arrieta, J. (1999). Las 5S pilares de la fábrica visual. *Revista Universidad EAFIT, 35*(114), págs. 35- 48. Recuperado el 5 de Marzo de 2023

AutoDesk 3D Max. (Febrero de 2023). *Web oficial de AutoDesk.* Recuperado el 17 de Febrero de 2023, de https://acortar.link/kqwF4Z

Autodesk Maya. (Febrero de 2023). *Web oficial de Autodesk.* Recuperado el 16 de Febrero de 2023, de https://acortar.link/oxGkWA

Barleta, E. P. (21 de Abril de 2020). La revolución industrial 4.0 y el advenimiento de una logística 4.0. Recuperado el 4 de Marzo de 2023, de https://hdl.handle.net/11362/45454

Blender. (Febrero de 2023). *Manual de referencia Blender 3.6.* Recuperado el 26 de Febrero de 2023, de Web oficial de Blender: https://acortar.link/ozSOOU

Castellano Lendínez, L. (2019). *Metodología para aumentar la eficiencia de los procesos Kanban.* Valencia: Universidad politecnica de Valencia. doi:http://dx.doi.org/10.17993/3ctecno/2019.

Cinema 4D. (Febrero de 2023). *Web oficial de Maxon.* Recuperado el 26 de Febrero de 2023, de https://www.maxon.net/es/cinema-4d

Del Giorgio, H. R., & Mon, A. (2018). Niveles de productos software en la industria 4.0. *International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)*, 53-62. Recuperado el 10 de Marzo de 2023, de www.ijisebc.com

Escartín, E. R. (2000). La realidad virtual, una tecnología educativa a nuestro alcance. *Revista de Medios y Educación, 15*, 5-21. Recuperado el 7 de Marzo de 2023

FischerTechnik. (2022). *Web oficial de FischerTechnik*. Recuperado el 28 de Febrero de 2023, de https://www.fischertechnik.de/es-es/servicio/empresa/sobre-fischertechnik

Google Analytics. (2023). *Pagina ofical de ImageToStl*. Recuperado el 10 de Abril de 2023, de https://imagetostl.com/es/convertir/archivo/step/a/obj

Lara, G., Santana, A., Lira, A., & Peña, A. (24 de 02 de 2019). El Desarrollo del Hardware para la Realidad Virtual. (U. d. Guadalajara, Ed.) *Revista lbérica de Sistemas y Tecnologías de Información*. doi:10.17013/risti.31.106–117

Lean Manufacturing Web. (2023). *LeanManufacturing*. Recuperado el 28 de Marzo de 2023, de https://www.leanmanufacturingweb.com/la-teoria-de-los-cinco-ceros/

Levis, D., & Levis, A. (2006). *¿ Qué es la realidad virtual?* Scielo. Recuperado el 27 de Febrero de 2023

Martínez, I. L. (Abril - Junio de 2021). La transformación del talento humano en el marco de la Industria 4.0. *Revista Cubana de Transformación Digital Vol 2 Num 2*, págs. 118-133. Recuperado el 27 de Febrero de 2023, de https://rctd.uic.cu/rctd/article/view/122

Master.D. (Febrero de 2023). *Web oficial de Masterd blog.* Recuperado el 26 de Febrero de 2023, de https://acortar.link/14hX0D

Perez García, A. (2015). El aprendizaje con videojuegos: experiencias y buenas prácticas realizadas en las aulas españolas. 135-156. Recuperado el 14 de Marzo de 2023

Revuelta Dominguez, F. I., & Guerra Antequera, J. (2012). ¿ Qué aprendo con videojuegos?: una perspectiva de meta-aprendizaje del videojugador. *RED. Revista de educación a distancia.* Recuperado el 16 de Marzo de 2023

RIPIPSA. (Noviembre de 2019). *Tecnologia al servicio de la industria*. Recuperado el 26 de 02 de 2023, de https://acortar.link/rLC7on

SolidWorks. (2023). *Web oficial de Dassault Systemes SolidWorks Corporation*. Recuperado el 06 de Marzo de 2023, de https://www.solidworks.com/es

Unity. (Febrero de 2023). *Web oficial de Unity.* Recuperado el 27 de Febrero de 2023, de https://unity.com/es

Unreal Engine. (Febrero de 2023). *Web oficial de Epic Game.* Recuperado el 16 de Febrero de 2023, de https://www.unrealengine.com/es-ES

Yacuzzi, E., Fajntich, C., & Pía Romeo, M. (2013). *Aplicaciones del Just-In-Time en Argentina.* Buenos Aires: Universidad del CEMA. Recuperado el 10 de Marzo de 2023

ANEXOS

Programación C#

La herramienta de creación de código utilizado en este documento es Visual Basic Studio (Visual Basic). Inicialmente cuando abrimos por primera vez un script nos encontramos con un código preestablecido como el que se muestra en la Figura 62:

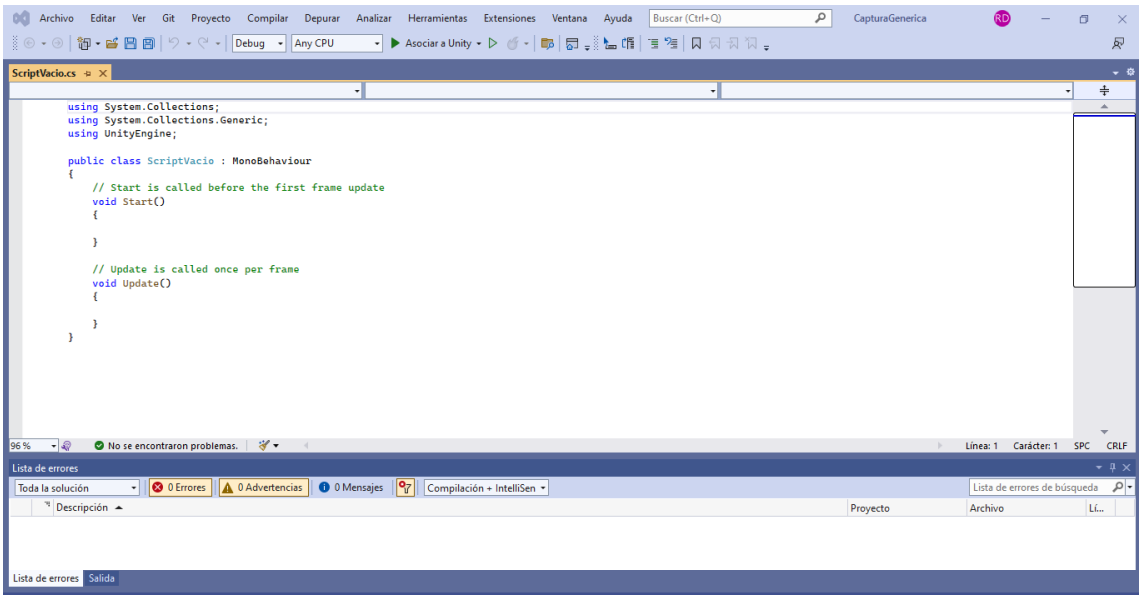

Figura 62: Script vacío.

En el podremos ver que en la parte superior existen unas líneas con la primera palabra "*using*", estas serán las bibliotecas de datos que por defecto Visual Basic y Unity dan acceso. La estructura básica de cualquier código es la siguiente:

- Declaración de variables. Aquí se determinará si la variable es pública (*public*) o si es privada (*private*) inicialmente, después se pondrá el tipo de variable: entera (*int*), decimal (*float*), un objeto del juego (*GameObject*) entre otras opciones.
- *Star* (): función que el programa leerá en primer lugar al comenzar el script y una única vez. En esta, se iniciarán variables o se generarán los primeros objetos que queremos en escena (entre otras alternativas que dependerán del objetivo del script).
- *Update*(): función que el programa recorrerá una vez por cada fotograma (*frame*) actualizando la imagen en función de su contenido.

La programación orientada a objetos puede ser muy variada, es decir, para un mismo objetivo de programación existen múltiples formas de hacer lo mismo. Es por ello, que a continuación de mencionará la parte de programación orientada a objetos más básica y utilizada para este proyecto en particular, aunque también se hablará ligeramente de otras opciones que podrían ser interesantes en líneas futuras de investigación. Para ello se pondrán líneas de código concretas que corresponderían con los juegos de aprendizaje de Unity.

❖ Control de gravedad. (Código 10).

Código 10: Control de gravedad.

```
rbJugador = GetComponent<Rigidbody>();
```

Esta suele ser una línea muy utilizada en la primera función Start (), en la que se inicializa una variable de tipo *Rigidbody* del Inspector de Unity. Es decir, estamos otorgando el poder de modificar efectos físicos relacionados con la gravedad a nuestro objeto.

❖ Movimiento básico de un jugador. (Código 11)

Código 11: Movimiento básico de un jugador.

```
controlHorizontal = Input.GetAxis("Horizontal");
controlAvance = Input.GetAxis("Vertical");
```

Declaración de dos variables (controlHorizontal y controlAvance). Estas solo podrán tomar valores de desde -1 hasta 1 dependiendo del tiempo pulsado. La entrada se producirá por medio del teclado. Las teclas exactas se han concedido según una configuración por defecto que tiene Unity en la que:

- Movimiento izquierdo "←" o "a"
- Movimiento derecho "→" o "d"

❖ Límite de movimiento de un jugador según un rango. (Código 12).

Código 12: Restricción de movimiento.

```
if (transform.position.x < -rangoX ) // Si la posición de x<-20
{
transform.position = new Vector3(-rangoX, transform.position.y, transform.position.z);
}

if (transform.position.x > rangoX) // Si la posición de x<-20
{
transform.position = new Vector3(rangoX, transform.position.y, transform.position.z);
}
```

Uso de la función de programación condicional (*if*), en la que el programa solo entrara a su contenido si las condiciones impuestas entre paréntesis se cumplen. Se imponen condiciones de posición en el eje X del Jugador según el límite de la escena especificado en la variable rangoX. Una vez entra en cada condición se hace modificar la posición del jugador de tal manera que la coordenada X de este corresponda con la del limite. Para este caso particular, se leería de la siguiente manera:

Si la posición del jugador es menos que -20 en el eje X, entonces la posición vectorial de este jugador pasará a tomar el valor de (-20, la misma coordenada que tuviera en Y, la misma coordenada que tuviera en Z).

❖ Instanciar objetos en la escena. (Código 13).

Código 13: Instanciar un objeto.

```
public GameObject proyectilPrefab;
void Update()
{

        if (Input.GetKeyDown (KeyCode.Space)) //Si detecta la pulsación del espacio
            {
               Debug.Log("Lanzamiento");
               Instantiate(proyectilPrefab , transform.position ,
                                        proyectilPrefab.transform.rotation );
            }
}
```

Se declara unas variables de tipo *GameObject* que deberá colocarse desde Unity para determinar sus pertenencias, es decir, a cada variable se le asigna un objeto de la escena. Es importante destacar el termino Prefab. Esta será una carpeta que crearemos en nuestro Projects donde introduciremos todos los objetos que vayan a aparecer y desaparecer a lo largo de la simulación, es decir, no incluiremos aquellos que permanecen continuamente en la escena. Antes de introducir todos estos objetos, debemos haberlos situado y configura do correctamente habiéndoles asignado sus respectivos scripts. Aunque esto se puede hacer más adelante. Los Prefabs no serán más que objetos prefabricados de los que se irán haciendo copias a lo largo de la simulación, pudiendo incluso aparecer múltiples copias a la vez dentro de una misma

escena. Serán estos objetos situados en la carpeta Prefab los que asignaremos a las variables.

Dentro de la función *Update (),* se hace uso de la función de programación condicional (*if*), en la que el programa solo entrara a su contenido si las condiciones impuestas entre paréntesis se cumplen. "Input.GetKeyDown (KeyCode.Space)" se utiliza para determinar que la pulsación de la tecla cuyo código en Unity es "*Space*" (barra de espacio). En este caso se ha utilizado GetKeyDown sin embargo no es el único existente:

- *GetKeyDown*: Cada vez que pulse la tecla
- *GetKeyUp*: Cada vez que suelto la tecla
- *GetKey*: Cuando dejo pulsada la tecla
- *KeyCode*: detecta la tecla correspondiente con el código impuesto

En el interior del primer if nos encontramos con el código "Instantiate" que se utilizara a menudo a lo largo de los diferentes juegos. Se utiliza para realizar copias del objeto que le indiquemos como parámetro, en la posición desde la que inicie el proyectil y la rotación (que, en este caso, no queremos que tenga ninguna, por eso ponemos la que tenga en ese instante).

La parte de código definida como "Debug.Log" es otro de los códigos que se utilizan con frecuencia para hacer que aparezca un mensaje que se escribe entre paréntesis y comillas para que salga por la consola de Unity. Normalmente lo utiliza el programador para verificar líneas de código.

❖ Destrucción de un objeto de la escena. (Código 14).

Código 14: Destrucción de un objeto.

```
if (transform.position.z > limiteSuperior)
  {
    Destroy (gameObject);
  }
```

"Destroy (gameObject)" se utilizará para hacer desaparecer de la escena y borrar del panel de jerarquía el objeto al que esté asociado. En concreto estas líneas de código se leerían de la siguiente forma:

Si la posición del jugador en el eje z supera el valor impuesto en la variable limiteSuperior, el objeto será eliminarlo de la escena.

❖ Script de especial relevancia. (Código 15).

Código 15: Script de generación de animales.

```
public class Generador : MonoBehaviour
{
   public GameObject[] animalesPrefabs; // Lista array

   private float rangoXGenerador = 20f;
   private float posZGenerador = 20f;
   private float retardoInicial = 2.0f;
```

```
    private float intervaloGeneracion = 1.5f;

    void Start()// Generacion de animales a intervalos regulares de tiempo
    {
        InvokeRepeating ("GeneradorDeAnimales", retardoInicial , intervaloGeneracion );
    }

// Creacion de una función a la que podemos llamar las veces que haga falta en múltiples escenarios.

    void GeneradorDeAnimales ()
    {
        int indexAnimal = Random.Range(0, animalesPrefabs.Length);
        Vector3 posGenerador = new Vector3(Random.Range(-rangoXGenerador,
                                        rangoXGenerador), 0, posZGenerador

        // Generar un animal
        Instantiate(animalesPrefabs[indexAnimal], posGenerador,
                                        animalesPrefabs[indexAnimal].transform.rotation);
    }
}
```

Este código, forma parte de un juego generador de animales. Y podemos encontrar múltiples líneas código interesantes. Para empezar, se han declarado múltiples variables de tipo *float*, cuya funcionalidad es diferente según donde se han utilizado en el código. Por otro lado, tenemos un tipo nuevo de variable que es "GameObject[]". Este le utilizaremos para crea una pestaña en forma de lista (*array*) dentro del Inspector de Unity en el que introduciremos los prefabs de los animales. Además, es importante destacar que a cada uno de estos animales se le otorga una numeración que se podría utilizar posteriormente en el código si quisiéramos llamar a uno de estos elementos en concreto. Ver Figura 63.

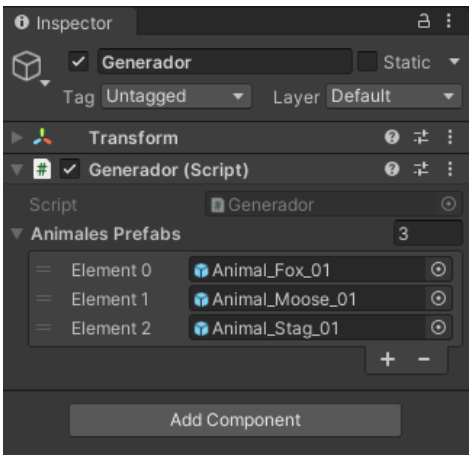

Figura 63: Variable de tipo array vista desde el Inspector de Unity.

A continuación, nos encontramos con la función *Start ()* que se utilizará para generar repetidas copias de un objeto. Para ello, colocamos entre paréntesis, el nombre del objeto (que en este caso es una función llamada "*GeneradorDeAnimales*") aunque lo más habitual es colocar directamente el objeto, el tiempo que tarda en comenzar a generar la primera copia y por último el intervalo de aparición entre las copias.

Y finalmente nos encontramos con la creación de una función nueva, a la que podríamos llamar las veces que haga falta a lo largo del código. Dentro de esta hemos declarado dos variables nuevas. La primera de tipo entero (*int*), donde con el código "Random.Range(0,animalesPrefabs.Length)" creamos una variable aleatoria desde el valor cero hasta un valor final, que hemos puesto que sea la longitud de nuestra lista. Es decir, si nuestra lista contiene 3 elementos, la variable tomará valores aleatorios desde 0 hasta 2 (uno menos dado que los elementos empiezan a numerarse desde cero).

Y la segunda variable vectorial, que nos ayudará para generar la posición aleatoria en el eje X de los animales. Si nos fijamos en el código, este se ha puesto de tal forma que siempre se generen en cualquier punto de la parte superior de la pantalla de simulación.

Por último se ha utilizado un lenguaje del código que ya conocemos para hacer aparecer animales aleatorios según el *array* creado, en la posición asignada por la variable *posGenerador*, con la orientación original en la que se crearon los animales.

Es importante destacar que, para hacer uso de este script creado, debemos crear un objeto vacío (al que hemos llamado Generador) en el panel jerárquico de Unity y asociarlo con él. Esta suele ser una práctica muy habitual para objetos organizadores o controladores de diferentes situaciones.

❖ Asociación entre scripts. (Código 16).

Código 16: Asociación de scripts.

```
detenerJuego = GameObject.Find ("Jugador").GetComponent <ControlJugador> ();
```

Siempre que se utiliza una variable de este estilo, es necesario inicializarla dentro de la función *Start().* Es decir, daremos a la variable "detenerJuego" la ubicación exacta de donde debe encontrar su valor. En este caso se le está imponiendo que busque un objeto en la escena cuyo nombre exacto sea "Jugador" y que entre dentro del *script* ControlJugador que tiene asociado ese objeto. Es importante destacar que un objeto puede tener asociados varios *scripts* a la vez, por ello la importancia de especificar exactamente la dirección.

❖ Comparación de objetos y múltiples condiciones. (Código 17).

Código 17: Comparación de objetos.

```
if (transform.position.x < limiteIzquierdo && gameObject.CompareTag ("Obstáculo"))
{
   Destroy(gameObject); // Destrucción del objeto
}
```

Hay que tener en cuenta que, en este caso, dentro del comando *if*, hemos impuesto dos condiciones diferentes, pero que deben cumplirse al mismo tiempo para poder entrar dentro de la misma. La segunda condición escrita en el código es:

gameObject.CompareTag ("Obstáculo"). En este caso estamos diciendo en el código que el programa compara y verifica si el objeto al que está asociado este *script* tiene una etiqueta con el nombre de “Obstáculo”.

Hasta ahora no habíamos mencionado lo que son las etiquetas (*Tag*). Estas se configuran en el Inspector de Unity dentro de cada objeto añadiendo una nueva etiqueta (*Add Tag*). Para ello pulsamos el botón donde aparece el símbolo de la suma (+) y colocamos el nombre que deseemos. Esto como hemos visto, se utiliza para poder encontrar y manipular dentro del código de C# un objeto concreto.

Como se ha indicado al comienzo de este anexo de programación, esto no es más que una iniciación en la programación de código de C#, por lo que este conocimiento expuesto está limitado a este documento. Es por ello por lo que, para terminar con esta área, se presentan unas nuevas bibliotecas que podrían ser útiles para líneas futuras de investigación en este campo. Cada una nos da acceso a aspectos diferentes:

- "*UnityEngine*" es la biblioteca principal de Unity que proporciona acceso a la mayoría de las clases y funciones de Unity.
- "*TMPro*" es una biblioteca de texto avanzada que se utiliza para crear y gestionar texto, como fuentes personalizadas, estilos de texto y efectos de animación.
- "*UnityEngine.UI*" es una biblioteca que proporciona una serie de componentes de interfaz de usuario (*UI*) de Unity, como botones, deslizadores y cuadros de texto, que se pueden usar para crear interfaces de usuario interactivas en los juegos.
- "*Unity.VisualScripting*", es un paquete separado que se puede instalar en Unity para agregar funcionalidades de programación visual a los proyectos. Esta biblioteca no es una parte estándar de Unity y requiere una instalación adicional.

Este anexo tiene como propósito presentar al lector una selección de juegos de aprendizaje básico que permiten conocer algunos aspectos del entorno de desarrollo Unity. Es importante destacar que en ningún momento se mostrará la programación en C#, aunque se considera que las nociones presentadas permitirían abordar esta tarea sin grandes dificultades. Los juegos aquí expuestos se basan en los cursos gratuitos que ofrece Unity a sus usuarios, así como en los diversos paquetes de recursos disponibles a través de esta empresa. Por tanto, si el lector tiene alguna duda, se recomienda que consulte la página oficial de Unity y realice los cursos de aprendizaje que allí se ofrecen.

Es importante señalar que los juegos presentados en este anexo son de creación propia y no se encuentran registrados en ningún lugar. No obstante, se basan en juegos que se deben realizar en los cursos de aprendizaje de Unity.

Juego 1:

El objetivo de este primer proyecto será colocar un vehículo sobre una carretera en la que pondremos diferentes obstáculos y el usuario por medio del teclado pueda desplazarse a lo largo del camino eludiendo cualquier obstáculo a su paso. Ver Figura 64.

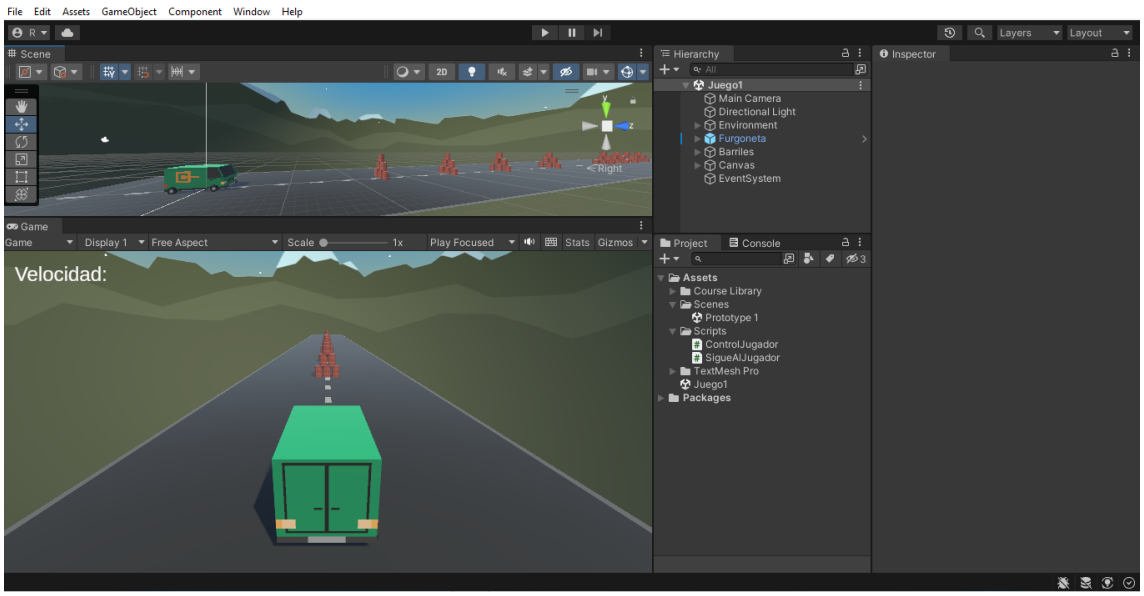

Figura 64: Juego1 completado.

Para comenzar debemos importar nuestro paquete de Assets donde encontraremos todo lo imprescindible y básico: *Assets / Import Package / Custom Package.*

Con ello cargaremos una escena básica de juego con un fondo con montañas y una carretera finita. A continuación, añadiremos objetos a la escena y los colocaremos donde queramos, para ello en la pestaña de project podremos abrir una carpeta llamada "Couse Library" donde encontraremos a su vez mas carpetas organizadas según su contenido: obstáculos, vehículos, entorno… Lo introducimos haciendo doble clic sobre el objeto deseado. Es importante hacer mención de que, al introducirlo de esta manera, el objeto se situará en el origen de coordenadas de la escena (0,0,0), pero luego a través de la pantalla *Scene* o en el *Inspector* podremos modificar su posición y orientación. Figura 65.

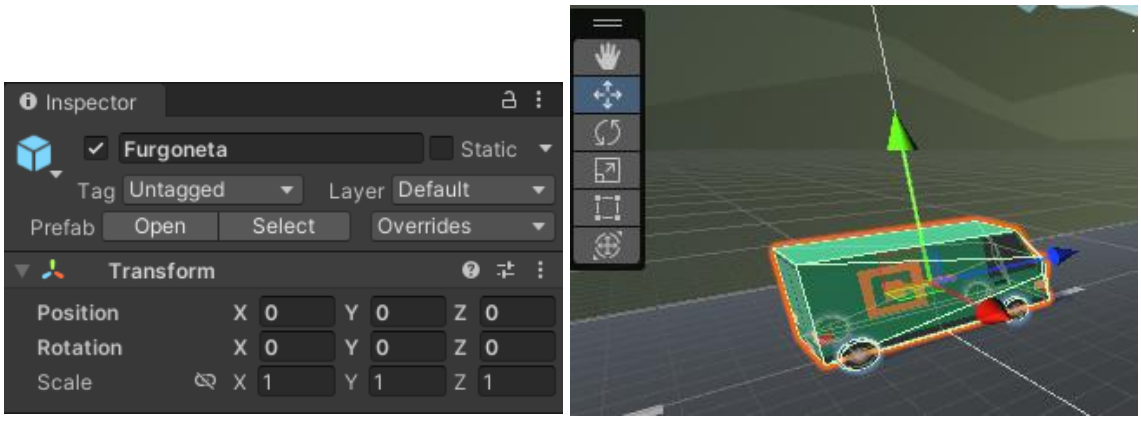

Figura 65: Modificación de posición. La izquierda refleja la modificación a través del Inspector, mientras que la derecha a través de los accesos directos de la escena.

Metemos una furgoneta y multitud de obstáculos (barriles) que hemos amontonado y colocado a lo largo de la carretera como se muestra a continuación: Ver Figura 66.

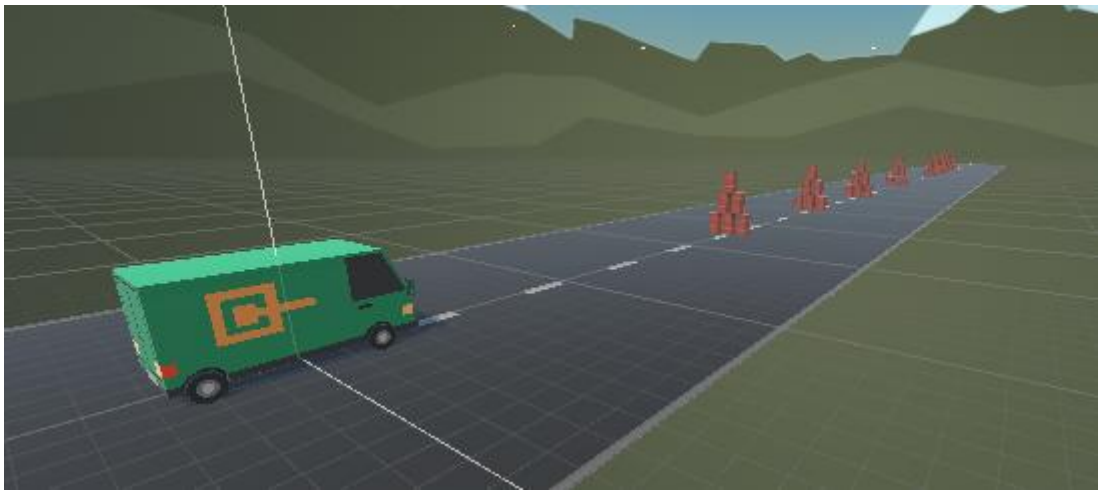

Figura 66: Escena que incluye la colocación de una furgoneta y unos barriles amontonados en la carretera.

Se debe añadir a través del inspector para cada uno de estos objetos una nueva componente llamada "*Rigidbody*", que otorga a cada objeto facultades físicas. Es decir, estamos configurándolo para que cada objeto tenga masa y pueda ser afectado por parámetros físicos como la gravedad entre otros. Este componente posteriormente se utilizará para hacer el se mueva el vehículo.

Después hemos movido la cámara para situarla en una posición donde el usuario que juegue pueda ver la furgoneta en tercera persona como si la estuviera conduciendo. Para ello en el panel de jerarquía hemos seleccionado "*Main Camera*" y aparece una pequeña pantalla en la escena donde podremos ver qué es exactamente lo que ve el usuario. Con ayuda de las herramientas de configuración de la escena, podremos mover y rotar la imagen, según unos ejes que aparecen en la escena. Ver Figura 67. Cabe destacar que el programa nos ayuda de forma intuitiva haciendo aparecer un icono de una cámara de vídeo en la escena que simboliza la ubicación de esta.

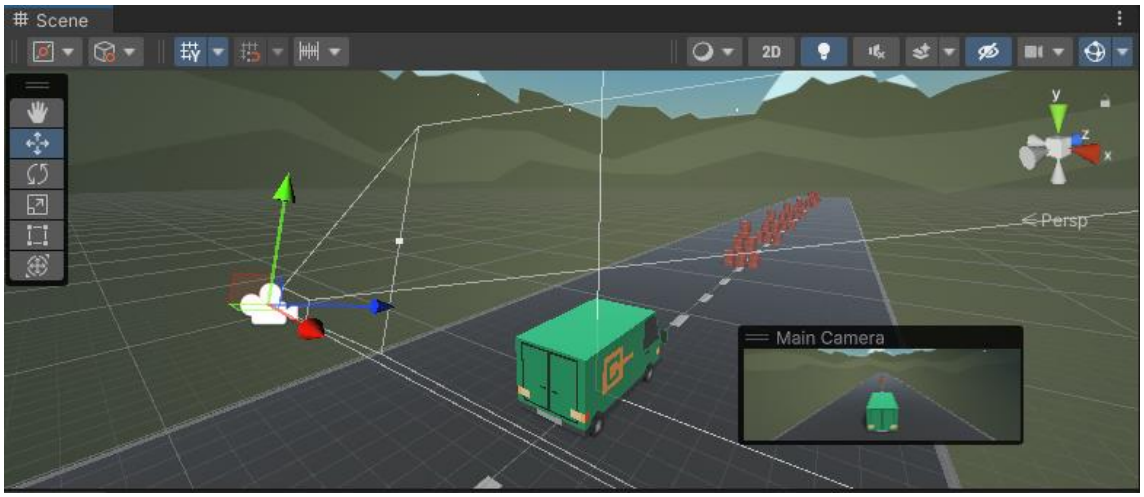

Figura 67: Herramientas de configuración de movimiento de la escena, la pantalla de la cámara y el icono de la cámara.

A continuación, empezaremos a darle vida a nuestro vehículo. Para ello, haremos uso de la programación que nos ofrece Unity con C#. Con el objetivo de tener bien organizado todo el contenido por carpetas, en la pestaña *Projects* crearemos una carpeta vacía: *Botón derecho del ratón* / *Create* / *Folder*

De ahora en adelante para los demás proyectos incluidos, siempre crearemos esta carpeta a la que llamaremos "*Scripts*" y en ella introduciremos los scripts necesarios para el funcionamiento del juego: *Botón derecho del ratón / Create / C# Script*

A cada *script* le daremos un nombre (sin espacios) con el que identificaremos rápidamente su contenido. Por último, solo debemos realizar la programación de estos haciendo uso de los comandos vistos en el anexo anterior para acabar consiguiendo el movimiento de la furgoneta. Se recomienda al lector, hacer uso de dos *scripts*, uno para controlar la furgoneta y otro para conseguir que la cámara siga al jugador en todo momento.

Juego 2

A este nuevo juego lo hemos llamado “Ninja Clic”. A diferencia del juego anterior, este dispondrá de un menú inicial en el que podremos determinar la dificultad, dejando que el usuario realice una elección entre fácil, medio o difícil. Además de ello, también dispone de un botón para jugar una partida nueva en caso de finalizar y llegar al *game over*. El juego consistirá en hacer clic con el botón izquierdo del ratón sobre las piezas de comida para poder ir sumando puntos, en caso de hacer clic sobre la calavera, los puntos se restarán. El juego termina en el momento en que el usuario deje escapar una de estas piezas de comida. Será en ese momento cuando salga el mensaje de *Game Over* y el botón de “Juego Nuevo”. En las Figura s que a continuación se muestran, se puede ver los objetivos que este juego persigue, ver Figura 68, Figura 69 y Figura 70.

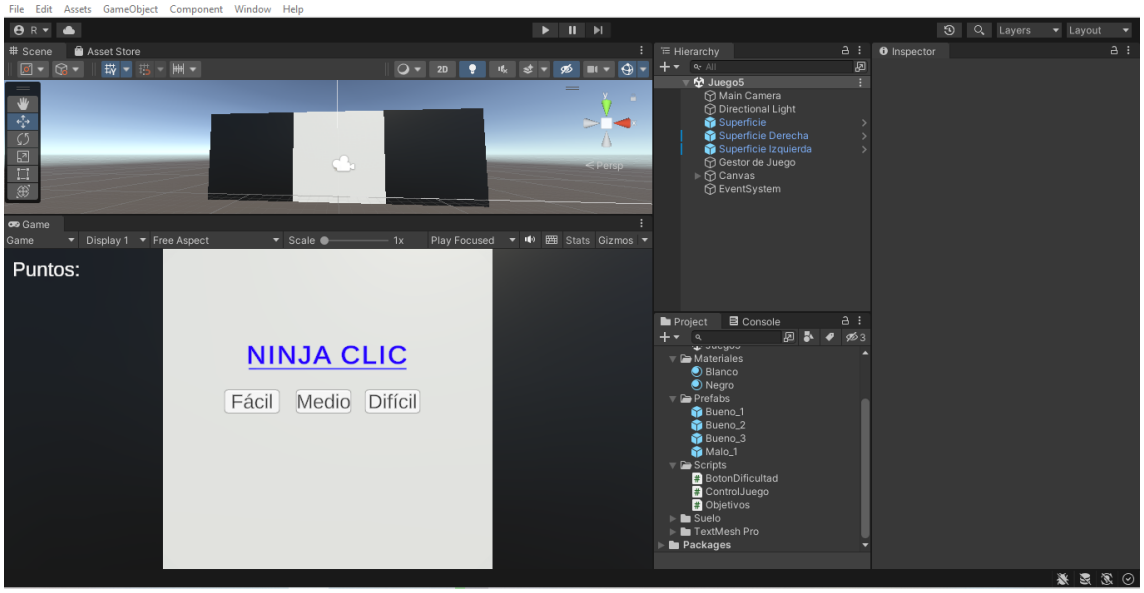

Figura 68: Pantalla de inicio del Juego2.

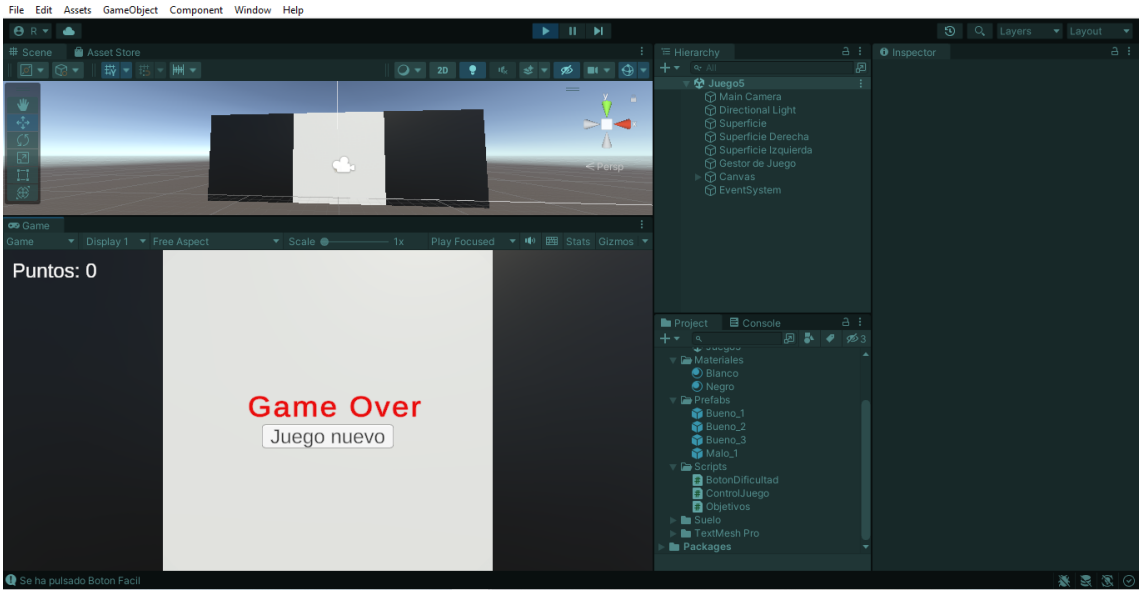

Figura 69: Pantalla de Game Over junto al botón para una partida nueva

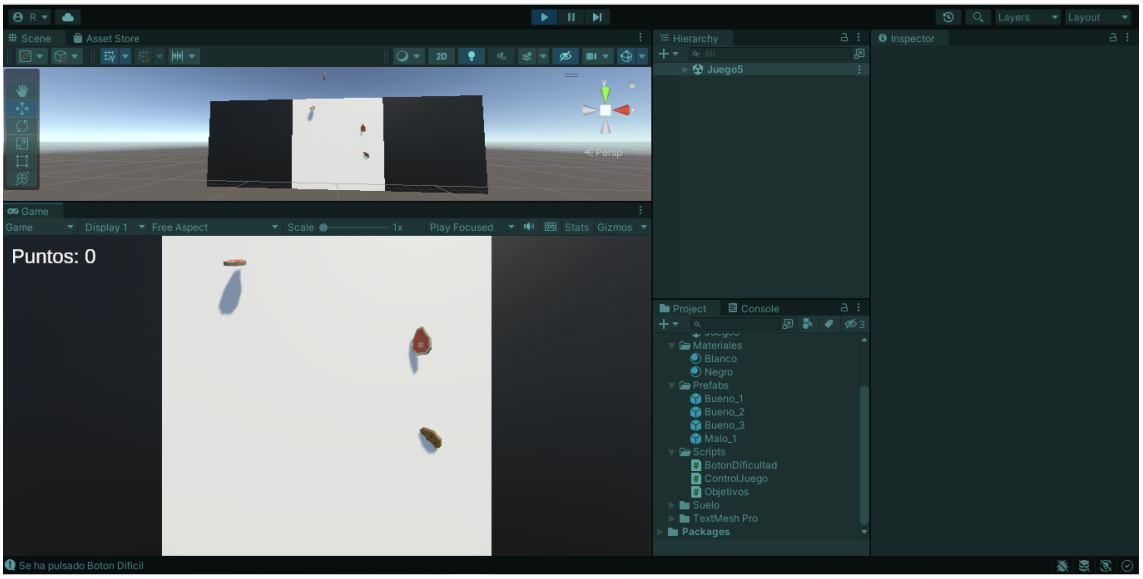

Figura 70: Captura durante la simulación del Juego2.

Comenzaremos creando el entorno de juego, para eso (como hemos dicho anteriormente), haremos uso de los paquetes que ofrecen los cursos de Unity, para instanciar todos los recursos necesarios: los diferentes trozos de comida y la calavera de fin de juego.

Añadiendo a cada uno de estos objetos, los componentes de Rigidbody y BoxCollider. Hemos cambiado el nombre de los diferentes tipos de comida en función si para nuestro juego suma puntos, será “Bueno_X”, y si de lo contrario, nos resta, será “Malo_X”. Además, estos se copiarán en una carpeta que crearemos llamada Prefabs y los borraremos del panel de jerarquía. De esta manera, logramos que, al comenzar la simulación, no haya ninguno de estos en la escena, y solo aparezcan en el caso de ser arrastrados manualmente durante la simulación al panel de jerarquía.

En esta ocasión no solo hemos hecho uso de las funciones habituales de *Start*() y *Update*(), sino que utilizamos la declaración de nuevas funciones que nos ayudan a cumplir con el objetivo de cada *script*.

En la escena vamos a hacer uso de texto y de botones, por ello debemos acceder a la interfaz de usuario: *Panel de jerarquía / Botón derecho / UI / Text – TextMeshPro; Panel de jerarquía / Botón derecho / UI / Button – TextMeshPro.*

Al seleccionarlo, si es la primera vez, nos saldrá una pestaña donde debemos hacer clic sobre el botón "*Import TMP Essentials*". Esto nos creará una nueva sección en la jerarquía llamada "*Canvas*". A partir de este momento esta carpeta la asociaremos siempre con la escritura de textos y botones que puedan aparecer en la escena. De esta carpeta, cuelga lo que por defecto se llama *"Text (TMP)"*, que no es mas que un editor de texto donde a través del Inspector podremos modificar sus propiedades.

Cuando queremos editar este texto o los botones, debemos fijarnos que en la pantalla 3D de la escena, deberemos alejar la cámara para ver correctamente su distribución. Es decir, desde la pantalla de *Scene* modificaremos la localización del texto o de los botones y desde la pestaña de *Game* podemos ver la localización real en mi escena de simulación.

La programación de este videojuego podríamos organizarla con estos tres *script*:

1. *Objetivos*: su función es controlar el comportamiento de los objetivos en el juego, asignando un valor de puntos y generando una posición y fuerza aleatorias para cada objetivo. El objetivo se destruye si cae por debajo de una cierta altura, y si se hace clic sobre él, se destruye y se muestra una explosión, se actualiza el marcador de puntos y se comprueba si el juego sigue activo. También se asigna un componente "ControlJuego" para poder interactuar con el gestor de juego.
2. *ControlJuego*: su función es controlar el juego en sí mismo, contando los puntos obtenidos y generando objetivos aleatorios. El *script* cuenta con una lista de objetivos y variables de texto para mostrar el marcador y el fin del juego. También incluye funciones para actualizar el marcador, reiniciar el juego y comenzar el juego con diferentes niveles de dificultad. El juego genera objetivos aleatorios a través de una corrutina y el juego se detiene si se alcanza la condición de fin del juego.
3. *BotonDificultad*. Este *script* de Unity crea un botón que permite establecer la dificultad del juego y, al ser pulsado, llama a una función que inicia el juego con la dificultad elegida. El *script* utiliza las bibliotecas de Unity y de interfaz de usuario y contiene tres variables: una para la dificultad elegida, otra para el botón y otra para el control del juego. La función "*Start*()" inicializa las variables y la función "EstablecerDificultad()" establece la dificultad elegida y llama a la función "IniciarJuego()" del controlador del juego para iniciar el juego.