

# **Investigación de motores para desarrollar videojuegos 2D en entornos web**

Félix Hernández Muñoz-Yusta

6 de septiembre de 2024

ESNE: Escuela Universitaria de Diseño, Innovación y Tecnología  
Grado en Diseño y Desarrollo de Videojuegos  
Especialidad Programación  
Curso 2023/2024



## Índice

<b>Resumen.....</b>	<b>3</b>
<b>Abstract .....</b>	<b>3</b>
<b>Introducción.....</b>	<b>4</b>
<b>Objetivos generales.....</b>	<b>5</b>
<b>Marco conceptual y contextual: el estado de la cuestión .....</b>	<b>6</b>
<i>Tecnologías web .....</i>	<i>8</i>
<i>Motores de desarrollo .....</i>	<i>9</i>
<b>Metodología de la investigación.....</b>	<b>12</b>
<b>Desarrollo del proyecto .....</b>	<b>14</b>
<i>Unity.....</i>	<i>15</i>
<i>Godot.....</i>	<i>19</i>
<i>Construct 3 .....</i>	<i>23</i>
<i>Phaser Editor (Phaser.js) .....</i>	<i>26</i>
<i>Defold.....</i>	<i>29</i>
<i>Roles de programación, Análisis y Relación de tareas .....</i>	<i>32</i>
<i>Roles de programación .....</i>	<i>32</i>
<i>Análisis .....</i>	<i>32</i>
<i>Relación de tareas .....</i>	<i>32</i>
<b>Resultados y conclusiones .....</b>	<b>34</b>
<i>Resultados.....</i>	<i>34</i>
<i>Conclusiones .....</i>	<i>35</i>
<b>Post mortem del desarrollo y líneas de futuro .....</b>	<b>35</b>
<b>Anexos.....</b>	<b>36</b>
<b>Bibliografía .....</b>	<b>36</b>

## Índice de figuras

Figura 1 - Esquema del nivel a desarrollar.....	14
Figura 2 - Captura del editor de Unity .....	16
Figura 3 - Captura del editor de Visual Studio .....	16
Figura 4 - Captura del editor de Godot .....	19
Figura 5 - Captura del editor de Construct 3 .....	23
Figura 6 - Captura Hoja de Eventos de Construct 3 .....	24
Figura 7 - Captura del editor de Phaser Editor .....	27
Figura 8 - Captura del editor de Defold .....	29

## Resumen

Hoy en día existen múltiples herramientas capaces de desarrollar videojuegos que están al alcance de cualquier persona. Además, es cierto que actualmente hay una tendencia al desarrollo de juegos que se ejecutan directamente desde el navegador, por lo que no necesita instalaciones y permite un acceso más rápido a los usuarios a jugar estos contenidos.

El objetivo de esta investigación es recoger y comparar los motores de videojuegos que se encuentran actualmente y seleccionar los más idóneos para este estudio. Tras la selección, se han comparado las fortalezas y flaquezas de cada motor, además de crear un pequeño juego a modo de prueba para la investigación. Con esta investigación se pretende hacer un análisis de cómo están adaptados los motores para el desarrollo de juegos web. Los motores que han sido estudiados son Unity, Godot, Construct 3, Phaser.js y Defold.

videojuegos web, unity, godot, construct3, phaser.js, defold, análisis.

## Abstract

Nowadays, there are multiple tools capable of developing video games that are accessible to anyone. Additionally, it is true that there is currently a trend towards the development of games that run directly from the browser, meaning no installations are needed and allowing users quicker access to play these games.

The objective of this research is to gather and compare the video game engines that are currently available and select the most suitable ones for this study. After the selection, the strengths and weaknesses of each engine were compared, and a small game was created as a test for the research. This research aims to analyze how well the engines are adapted for web game development. The engines studied are Unity, Godot, Construct 3, Phaser.js, and Defold.

web video games, Unity, Godot, Construct 3, Phaser.js, Defold, analysis.

## Introducción

Hoy en día existen una inmensa cantidad de motores con los que se pueden desarrollar videojuegos. Estos motores son usados tanto en grandes empresas como en pequeños proyectos. Para lograr ser los más utilizados, tratan de ser lo más versátiles posibles, ya que así aumentan las posibilidades de uso en muchos más ámbitos. El juego online es tendencia actualmente, ya que los juegos más populares son de género multijugador, por lo que el desarrollo de videojuegos en línea es un campo en el que merece la pena invertir.

La idea de esta investigación surgió a partir de la noticia que salió en septiembre, en la que uno de los motores de videojuegos más utilizados, Unity, cambió su política de costes para los desarrolladores. Antes, se utilizaba un modelo de suscripciones con diferentes planes: Unity Personal (gratuito para ingresos inferiores a 100.000 \$/año), Unity Plus (399 \$/año para ingresos inferiores a 200.000 \$/año), Unity Pro (2.040 \$/año sin límite de ingresos), y Unity Enterprise (para grandes empresas con precios negociables). Sin embargo, Unity anunció un nuevo plan que generó polémica; pretendían cobrar a los desarrolladores una tarifa por cada instalación de un juego que usara su motor, siempre y cuando se superaran ciertos ingresos y número de instalaciones (200.000 \$ en ingresos y 200.000 instalaciones para Unity Personal y Plus, y 1 millón de dólares en ingresos y 1 millón de instalaciones para Unity Pro y Enterprise). Las tarifas comenzarían en 0,20 \$/instalación y podrían disminuir a 0,01 \$ dependiendo del número de descargas. Este cambio fue mal recibido por la comunidad de desarrolladores, ya que podría generar costes impredecibles y desproporcionados, especialmente para juegos gratuitos o indies. Como resultado de todo el malestar que se generó entre los desarrolladores, la empresa detrás de Unity, Unity Technologies, decidió dar marcha atrás a estos cambios y dejar todo como estaba anteriormente.

Tras esta noticia, muchos desarrolladores comenzaron a buscar alternativas para poder desarrollar sus juegos en un motor que les diera más libertad. Sin embargo, el problema con el que se encontraron es que actualmente existe una inmensa cantidad de motores, cada uno centrado en un campo específico, un lenguaje de programación distinto, una interfaz diferente, etc.

El problema que se pretende resolver es la combinación de dos temas. El primero es el auge de los juegos online o, como mínimo, que se encuentren en formato web. El segundo es la búsqueda de un motor gráfico para conocer las variedades que existen, además de los ya establecidos como los más utilizados en el mercado.

Para resolver este problema se van a estudiar y comparar una serie de motores seleccionados que cumplan el requisito de ser compatibles con la creación de juegos web. Los motores seleccionados finalmente son Unity (para comparar lo que muchos consideran el “estándar” de un motor de desarrollo), Godot (el motor que se considera que ha ganado más público tras lo ocurrido con las tarifas de Unity), Construct 3 (un motor completamente web), Phaser.js (una librería de JavaScript que cuenta con una versión de editor gráfico) y Defold (conocido por ser compatible con cualquier plataforma).

Los resultados obtenidos con esta investigación son una visión general de las ventajas y desventajas de cada motor y una explicación de que motor se considera más idóneo para el desarrollo de un juego para el formato web.

## Objetivos generales

Esta investigación está organizada siguiendo los siguientes objetivos:

1. Investigar si existen otras investigaciones similares a la que se está planteando.
2. Buscar cuáles son los motores de desarrollo más utilizados.
3. Investigar las herramientas y usos de los motores seleccionados.
4. Comenzar el desarrollo simultáneo en los motores.
5. Tomar notas de las diferencias entre los motores (durante el desarrollo).
6. Comparar los resultados obtenidos.

## Marco conceptual y contextual: el estado de la cuestión

Esta investigación es un estudio comparativo en el que se van a filtrar, de entre todos los motores para el desarrollo de videojuegos que existen actualmente, aquellos considerados más idóneos para el estudio, además de ser compatibles con la creación de juegos web, y se procederá a su análisis de forma individual.

Es importante recalcar que, para esta investigación, cuando se hace referencia a un formato web, se alude a la capacidad de que el juego se ejecute directamente desde una página web, es decir, sin necesidad de ninguna instalación para poder jugar. Esto no se refiere a que el motor tenga la capacidad de ofrecer un servicio multijugador, ya que eso implicaría tener que buscar servicios externos, como uno o varios servidores que soporten el juego.

Para poder empezar, es necesario conocer las definiciones de los temas a tratar en esta investigación. Se comprende como definición de un motor de videojuegos:

*Un motor de videojuegos es un software diseñado para facilitar el desarrollo de videojuegos, proporcionando una serie de componentes reutilizables y extensibles. Estos motores permiten a los desarrolladores centrarse en la creación de arte, la disposición de los mundos de juego y la definición de las reglas del juego, mientras que el motor gestiona aspectos esenciales como la renderización de gráficos en tres dimensiones, la detección de colisiones, la física de los objetos, la animación de personajes y el sistema de audio. La aparición del concepto de "motor de videojuegos" se remonta a mediados de la década de 1990, especialmente con juegos de disparos en primera persona como "Doom", donde se implementó una clara separación entre los componentes del motor y los contenidos específicos del juego. (Gregory, 2014)*

También es importante conocer la definición de lo que se conoce como un juego web, es decir, un juego cuyo soporte sea para páginas web:

*Los juegos en formato web son aquellos diseñados para ser jugados exclusivamente en navegadores web, utilizando tecnologías como HTML5, CSS3 y JavaScript, sin la necesidad de instalar plugins o aplicaciones adicionales en el dispositivo del usuario. (Makzan, 2011)*

Cabe señalar que esta definición se ajusta a un concepto básico de los juegos para formato web, ya que actualmente hay motores que permiten tener pequeños ejecutables dentro de la propia web.

Sin embargo, es importante destacar que la evolución tecnológica ha permitido que los juegos para formato web no se limiten a simples aplicaciones basadas en scripts o contenido multimedia.

No fue hasta la creación del lenguaje Java en los años 90 y el comienzo del uso de *applets* (pequeñas aplicaciones que se ejecutaban dentro de un programa) cuando se pudo empezar a incluir pequeños juegos dentro de los navegadores web.

Más adelante, ya en la década de los 2000, aparecería la tecnología Flash (Adobe Flash, anteriormente conocido como Macromedia Flash), que permitió a los desarrolladores crear animaciones y aplicaciones interactivas. Esta tecnología tuvo un gran auge, ya que no necesitaba ni descargas ni instalaciones para funcionar.

A pesar de su gran éxito, la tecnología Flash tenía sus limitaciones, y no fue hasta 2010 que llegó HTML5, introduciendo nuevas capacidades gráficas, multimedia y la manipulación de elementos en pantalla.

Posteriormente, llegó WebGL, una API de JavaScript para el renderizado gráfico 3D, que permitió crear juegos sin plugins adicionales. Además, al estar desarrollada con base en OpenGL, es compatible con la mayoría de los navegadores.

Conforme fue ganando importancia el desarrollo de juegos web, los motores de desarrollo comenzaron a intentar que sus motores fueran capaces de ejecutar juegos en páginas web. Anteriormente, estos motores se enfocaban en su mayoría en tratar de sacar el máximo potencial y rendimiento del ordenador, permitiendo crear juegos con mejores gráficos. Sin embargo, en el formato web no era el caso, ya que, debido a su limitada capacidad, se debía buscar la forma de comprimir los ejecutables del motor para no superar dichos límites. Por otro lado, comenzaron a surgir motores enfocados exclusivamente en el formato web, como es el caso de Phaser.js, que está basado en HTML5.

A día de hoy, se han desarrollado nuevas tecnologías como los WebSockets y WebRTC, que han permitido la creación de juegos web multijugador en tiempo real, ya que anteriormente estos requerían de complejas distribuciones de servidores.

La llegada de nuevos entornos, como los dispositivos móviles, ha ampliado aún más el público para los juegos web, ya que estos dispositivos permiten conectarse a internet y ejecutar dichos juegos desde cualquier lugar. Esto ha llevado a que muchos juegos adapten su entorno a los móviles, dado que el manejo del juego pasa de ser por teclado y ratón a controles táctiles en un dispositivo móvil.

## Tecnologías web

Dentro de las tecnologías que se van a tratar se encuentra HTML5, el cual se define como:

*HTML (Lenguaje de Marcas de Hipertexto, del inglés HyperText Markup Language) es el componente más básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento (JavaScript). (MDN contributors, 2024)*

En concreto, HTML5 ha sido clave para permitir el desarrollo de los juegos web, ya que ahora es capaz de añadir APIs (mecanismos que permiten a dos componentes de software comunicarse entre sí) que manejan gráficos, audio y video de forma muy eficiente y sin necesidad de plugins, algo que era obligatorio con Adobe Flash. Otro aspecto muy relevante es la compatibilidad con los navegadores, incluyendo los de los dispositivos móviles.

Por otro lado, se encuentra CSS3, el cual se puede definir como:

*Hojas de Estilo en Cascada (del inglés Cascading Style Sheets) o CSS es el lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML (incluyendo varios lenguajes basados en XML como SVG, MathML o XHTML). CSS describe cómo debe ser renderizado el elemento estructurado en la pantalla, en papel, en el habla o en otros medios. (MDN contributors, 2024)*

Esta hoja de estilos, dentro de este contexto, se usa para dar estilo a los elementos de la interfaz de usuario (UI) como botones, menús, texto, y la información en pantalla. También es capaz de crear animaciones o transiciones.

También cabe recalcar el uso de JavaScript, cuya definición es importante conocer:

*JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo, programación funcional). (MDN contributors, 2023)*

Este lenguaje se utiliza en múltiples motores de juegos web y es esencial para el desarrollo de juegos web, ya que es capaz de modificar el DOM (Document Object Model), gestionar eventos de usuario (detección de inputs) y ejecutar lógica del juego en tiempo real. Asimismo, es capaz de manejar APIs que permiten dibujar dentro del canvas, modificar gráficos en 2D y 3D (con la ayuda de WebGL), reproducir audio, y es compatible con el uso de WebSockets, haciendo posible la creación de juegos multijugador, entre muchas otras funciones.



Finalmente, es importante conocer la tecnología de WebGL ya que, a pesar de haber sido mencionada anteriormente en el contexto de JavaScript, es relevante en esta investigación:

*WebGL permite que el contenido web use una API basada en OpenGL ES 2.0 para realizar la representación 3D en un HTML <canvas> en navegadores que lo admitan sin el uso de complementos. Los programas WebGL consisten en un código de control escrito en JavaScript y un código de efectos especiales (shaders) que se ejecuta en la Unidad de procesamiento de gráficos (GPU) de una computadora. Los elementos WebGL se pueden mezclar con otros elementos HTML y combinar con otras partes de la página o el fondo de la página. (MDN contributors, 2023)*

La incorporación de WebGL a los entornos de desarrollo ha permitido que los juegos que se ejecutan desde los navegadores tengan gráficos comparables a los de una consola.

## Motores de desarrollo

Otro aspecto de vital importancia son los motores de videojuegos existentes. En esta investigación se van a centrar en los siguientes motores:

En primera instancia, tenemos uno de los motores más famosos, Unity, utilizado para el desarrollo tanto en 2D como en 3D desde su lanzamiento en 2005 por Unity Technologies. Entre sus principales características, podemos observar que utiliza el lenguaje de programación C# para su desarrollo y posee un editor gráfico con múltiples herramientas incorporadas que ayudan a los desarrolladores en su trabajo.

Es uno de los motores más versátiles que existen hoy en día, ya que es capaz de exportar los juegos a consolas, ordenadores, dispositivos móviles y navegadores web a través de WebGL. Cuenta con una tienda dentro del propio motor llamada “Asset Store”, en la que los desarrolladores pueden subir sus propios objetos, modelos, scripts u otros recursos compatibles con Unity para facilitar el desarrollo a otros usuarios.

Otro aspecto fundamental de este motor es su amplia comunidad y la extensa documentación existente, ya que, al ser un motor tan utilizado, sus foros y documentación están llenos de información útil para solucionar los problemas que puedan surgir durante el desarrollo.

En segundo lugar, tenemos a Godot. Este motor ha ganado mucha popularidad entre los pequeños estudios de videojuegos, ya que es muy accesible para dichos estudios, además de tener una gran flexibilidad. Fue publicado en 2007, y su desarrollo ha estado liderado por Juan Linietsky y Ariel Manzur, pero al ser open-source (código abierto), ha recibido la ayuda de miles de contribuyentes que han permitido crear un motor perfectamente capaz de desarrollar juegos de lo más exigentes. Es capaz de soportar proyectos tanto en 2D como en 3D y utiliza un lenguaje de programación propio, GDScript, similar a Python. Su uso es completamente gratuito.

La arquitectura de este motor está basada en nodos, es decir, toda la estructura del proyecto se puede organizar en base a nodos, lo que simplifica su estructuración. Por otro lado, cuenta con un editor visual que permite modificar los nodos y los scripts, todo dentro del editor. Es multiplataforma, lo que quiere decir que es capaz de exportar para ordenadores, dispositivos móviles y, por supuesto, a formato web a través de OpenGL.

Es importante recalcar que Godot ganó un impulso considerable tras la polémica noticia del cambio de costes de Unity, y a pesar de que posteriormente se retractaron, Godot se dio a conocer a muchos desarrolladores que prefirieron dar una oportunidad a este motor.

Por otro lado, tenemos el motor de Construct 3. Este es el único motor de esta investigación que se ejecuta directamente desde su página web, es decir, no es necesaria instalación para poder usarlo. También es un motor que no requiere conocimientos de programación, ya que utiliza un sistema de “cajas” o eventos para gestionar toda la lógica del juego. Fue desarrollado por Scirra y, aunque la versión inicial de este motor salió en 2007, no fue hasta 2017 que se lanzó la tercera versión, la cual trabaja con HTML5 y JavaScript.

Si bien es cierto que no es capaz de desarrollar juegos tan exigentes como otros motores de desarrollo, su facilidad de uso y su fácil acceso le permiten posicionarse como un motor competente en el desarrollo de juegos en 2D. El motor cuenta con una versión de pago, pero tiene una versión gratuita que es suficiente para hacer un juego de carácter personal.

El cuarto motor que se va a tratar en esta investigación es Phaser.js, que es concretamente una librería que utiliza el lenguaje JavaScript. A pesar de poder utilizar su librería de forma gratuita a través de un editor de código, existe un editor gráfico a modo de motor que puede utilizarse a cambio de un pago mensual. Desarrollado por Photon Storm en 2013, este editor gráfico se llama Phaser Editor y actualmente se encuentra en su cuarta versión. Esta librería destaca por su simplicidad y eficacia a la hora de crear juegos en 2D.

Al estar basado directamente en HTML5 y JavaScript, está diseñado para crear juegos web, por lo que su implementación es sencilla, además de que soporta tecnologías como WebGL. A lo largo de su historia, se ha creado una gran comunidad para esta librería, lo que ha permitido que existan foros activos con usuarios avanzados en el uso de este motor.

Por ultimo, nos encontramos con Defold, un motor desarrollado por unos empleados de la empresa King llamados Christian Murray y Ragnar Svensson. Trabajaron desde 2008 en este motor como un proyecto paralelo para el uso en la empresa, pero en 2016 se decidió hacerlo de uso libre y, finalmente, en 2020, la dirección de este motor se independizó por completo de la empresa, pasando a llamarse Defold Foundation.

Este motor destaca por ser completamente gratuito y su principal característica es su capacidad de exportar juegos a cualquier consola, ordenador, móvil e incluso navegador web. Como lenguaje de programación utiliza Lua y, si bien es compatible con el 3D, está principalmente diseñado para juegos en 2D.

Todos estos motores cumplen con los requisitos que se plantean en esta investigación, es decir:

- Tener un IDE, es decir, un entorno de desarrollo integrado dentro del propio motor.
- Ser compatibles con la creación de juegos 2D.
- Que sean compatibles con la posibilidad de crear juegos para la web.

Finalmente, se puede añadir que, tras realizar una investigación sobre otros estudios que hacen alusión a comparaciones entre diferentes motores de desarrollo, se ha observado que existen múltiples y variadas comparaciones de herramientas, evaluando aspectos como el costo, las funcionalidades o la facilidad de uso, según el criterio del autor.

No obstante, es destacable que no se ha encontrado ningún trabajo de investigación que, además de comparar dichos motores, se enfoque simultáneamente en sus posibilidades de exportación al formato web.

Si bien esta investigación no pretende ser innovadora o disruptiva, es cierto que, tras un análisis de los trabajos publicados en diversas plataformas de investigación relacionadas con este tema, no se ha identificado ningún estudio que sea similar a la presente investigación.

## Metodología de la investigación

El objetivo de esta investigación es comparar los motores seleccionados para su estudio. Para poder ser comparados, se utilizarán los propios motores, evaluando desde el apartado visual, como puede ser el IDE (entorno de desarrollo integrado) que ofrece cada motor, las herramientas que facilitan el desarrollo a los usuarios, la dificultad de la curva de aprendizaje, la documentación disponible para comprender su uso, hasta los lenguajes de programación que emplea cada uno.

Los motores que se van a estudiar son Unity, Godot, la versión de pago de Construct 3, la versión de pago de la librería Phaser.js (Phaser Editor) y Defold. Como repositorio para la creación y mantenimiento de los proyectos se ha utilizado GitHub.

La investigación se desarrollará conforme avance el desarrollo de los proyectos en cada motor. El primer paso es crear el proyecto dentro de cada motor y crear un repositorio en GitHub para permitir hacer copias de seguridad de los proyectos. Posteriormente, se adaptará el proyecto al formato web en caso de ser necesario.

Como primer paso dentro del proyecto, se creará el personaje y se ajustará al tipo de objeto específico de cada motor (nodo, objeto, escena, etc.), analizando qué tipo de componente utiliza cada uno.

Después, se añadirán las animaciones de movimiento al personaje, ya sea a través de sprite sheets (hojas de animación) si el motor lo permite o mediante imágenes separadas si no es capaz de comprender dichas hojas. Es esencial añadir las animaciones al personaje para que, en función de la acción realizada, se muestre la animación correcta en cada momento.

El siguiente paso será añadir el suelo y las plataformas que conformarán el nivel. Se investigarán las herramientas que puedan facilitar, en caso de ser posible, la creación de escenarios, ya que muchos motores incluyen actualmente funcionalidades para este propósito.

Posteriormente, se agregarán colisiones entre los elementos del nivel. Después de esto, se configurará la gravedad en los objetos, de manera que se aplique a los elementos necesarios.

Lo siguiente es crear los movimientos (*inputs*) del personaje para que pueda desplazarse por el nivel al pulsar las teclas asignadas. Del mismo modo, se integrarán las animaciones para que, según la acción que realice el personaje (estar estático, moverse, saltar o caer), se muestre la animación correspondiente.

Un aspecto importante es que la cámara siga al personaje a lo largo del nivel, ya que esta es una funcionalidad necesaria para el correcto desarrollo de un juego.

Aunque puede considerarse un simple elemento decorativo, añadir un fondo al nivel mejora la calidad del producto y permite conocer si hay herramientas dentro del motor que faciliten su uso.

Finalmente, se incluirá un pequeño texto dentro del canvas o UI (interfaz de usuario) para mostrar información relevante o instrucciones al jugador.

Tras completar estos objetivos, se pretende recopilar toda la información obtenida, así como los apuntes y resultados durante la investigación, en este documento. De esta manera, se puede obtener un resumen de las principales ventajas y desventajas de cada motor.

Además del desarrollo del nivel para probar las características de cada motor, se pretende analizar los elementos externos que los rodean, es decir, su documentación oficial, su comunidad y soporte activo, la compatibilidad con otras librerías y la arquitectura de software con la que es compatible.

No se busca encontrar una solución definitiva ni un ganador entre todos los motores seleccionados. Cada motor ha sido elegido por sus características particulares, y cada uno presenta fortalezas específicas que pueden hacerlo más idóneo según el tipo de juego o proyecto que se pretenda realizar. En este caso, se ha optado por un juego 2D de desplazamiento lateral, ya que es un género ampliamente conocido y que puede ejecutarse fácilmente desde un navegador web.

Todo el diseño artístico del personaje, escenarios y texturas no posee ningún tipo de licencia de uso ni comercial, es decir, es de uso completamente gratuito. Asimismo, es importante mencionar que la Universidad ESNE me ha proporcionado un código de acceso para poder usar la versión completa del motor Construct 3.

## Desarrollo del proyecto

A continuación, se relatará el desarrollo de los proyectos en los motores que se emplean en esta investigación.

Es importante destacar que el motor Unreal Engine no forma parte de esta investigación, a pesar de ser otro motor de gran fama y alcance, debido a que no es capaz de exportar a un formato web de forma nativa. Es cierto que existen plugins creados por usuarios externos a la empresa Epic Games (creadores de Unreal Engine), pero estos no son muy utilizados ni han demostrado tener una efectividad destacable.

Por otro lado, también es importante recalcar que todos los juegos se han planteado dentro del ámbito 2D y si bien es cierto que varios de los motores a investigar son compatibles con el formato 3D, en esta investigación se ha decidido limitar únicamente al campo de visión en dos dimensiones (2D).

El resultado final que se espera obtener es un nivel con un personaje al que se le aplique la gravedad, con colisiones con el suelo y las plataformas y que sea capaz de moverse a lo largo del escenario.

A continuación, en la Figura 1 se muestra el resultado esperado durante el desarrollo en los motores:



Figura 1 - Esquema del nivel a desarrollar

## Unity

La investigación comenzó con la creación del proyecto en Unity, uno de los motores más conocidos y utilizados en la actualidad. Unity requiere la instalación de un gestor llamado Unity Hub, el cual permite crear nuevos proyectos, seleccionar la versión del motor a utilizar y gestionar el tipo de proyecto que se desea desarrollar.

La versión utilizada para este proyecto es la 2022.3.22f1, una versión LTS (*Long Term Support*), que se destaca por ofrecer soporte extendido y es recomendada para proyectos a largo plazo debido a su estabilidad y menor riesgo de fallos. Unity Hub permite cambiar la versión de Unity utilizada por un proyecto específico, lo que puede ser útil para acceder a nuevas características o mejoras de rendimiento. Sin embargo, al actualizar a una versión más reciente, pueden surgir conflictos de compatibilidad que requieran ajustes en el código o en los recursos del proyecto. Es importante mencionar que es posible actualizar un proyecto a una versión más nueva de Unity, pero no es posible regresar a una versión anterior sin riesgo de pérdida de datos o funcionalidades.

Dentro de la interfaz de Unity, se puede observar que es altamente personalizable, permitiendo ajustar no solo las pestañas de los elementos a mostrar, sino que a través de la pestaña *Layout* es posible cambiar completamente la composición de los grupos de pestañas. La disposición que viene por defecto permite acceder a todos los elementos del motor sin problemas.

Unity permite agrupar los elementos en carpetas según las preferencias del desarrollador, lo cual facilita la organización del proyecto durante su desarrollo. Dentro del motor, los escenarios se agrupan en archivos llamados *Scenes* (escenas). Estos archivos almacenan todos los objetos que se van a mostrar en esa escena cuando se está ejecutando; a estos se les denomina *GameObjects* y existen de múltiples tipos, como personajes, cámaras, entre otros.

A continuación, se mostrará en la Figura 2, una imagen del editor de Unity con el proyecto del juego que se ha desarrollado para esta investigación.

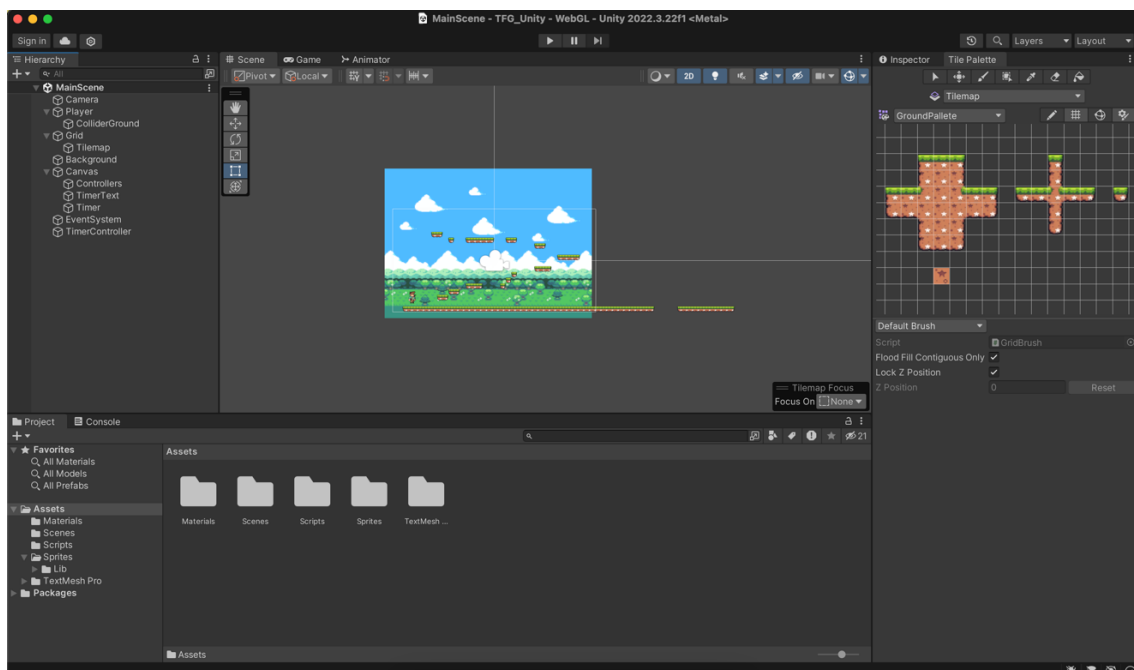


Figura 2 - Captura del editor de Unity

Dentro de esta imagen se pueden observar múltiples elementos. Si bien la organización de carpetas es libre para cada desarrollador, generalmente se tiende a mantener ciertas estructuras estándar, como la mostrada en la imagen, donde las escenas se almacenan en una carpeta, los scripts (hojas de código) en otra, los sprites (objetos) en otra, y así sucesivamente. En la ventana superior izquierda, se puede observar que la *MainScene* (escena principal) contiene todos los elementos que la componen: el jugador con sus colisiones, el escenario, el fondo de pantalla, la cámara y los elementos del canvas.

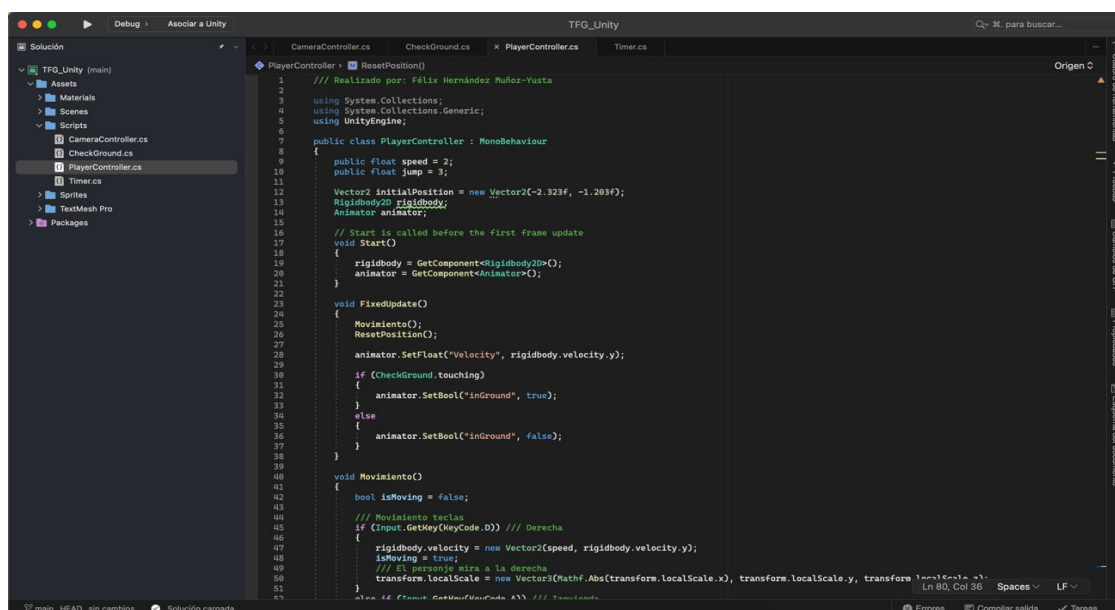
Normalmente, al crear una escena, Unity incluye por defecto una cámara, que es a través de la cual el jugador verá lo que se muestra en la escena, y una luz para iluminar los objetos en la escena.

Para las animaciones, Unity posee una ventana que permite crear una animación en base a un sprite sheet o un conjunto de imágenes separadas y puede asignar a un objeto de la escena. Además, cuando estas animaciones se asignan a un *GameObject*, como el personaje, Unity ofrece la posibilidad de cambiar entre las animaciones en función de unas condiciones indicadas, que el desarrollador deberá ajustar en función de su proyecto.

Por otro lado, Unity incluye una herramienta para crear escenarios basada en recortes de una imagen, llamada *Tilemap*. Como se puede ver en la Figura 2, en la ventana superior derecha aparece un terreno cortado en múltiples cubos de una imagen. Estos cubos se pueden colocar a lo largo de la escena con el objetivo de facilitar la creación de niveles. Las imágenes utilizadas para este fin se almacenan junto con los datos de sus recortes en archivos llamados *Tiles*, y cada cubo añadido en la escena ya posee colisiones y todos los elementos necesarios, como cualquier otro objeto.

Para escribir el código del proyecto, por defecto, Unity instala el entorno de desarrollo de Visual Studio, aunque es posible utilizar otros editores de texto siempre que sean compatibles con C#.

A continuación, se muestra en la Figura 3 el editor de Visual Studio con un fragmento del script del controlador del personaje.



```
1  // Realizado por: Félix Hernández Muñoz-Yusta
2
3  using System.Collections;
4  using System.Collections.Generic;
5  using UnityEngine;
6
7  public class PlayerController : MonoBehaviour
8  {
9
10     public float speed = 2;
11     public float jump = 3;
12
13     Vector2 initialPosition = new Vector2(-2.323f, -1.203f);
14     Rigidbody2D rigidbody;
15     Animator animator;
16
17     // Start is called before the first frame update
18     void Start()
19     {
20         rigidbody = GetComponent<Rigidbody2D>();
21         animator = GetComponent<Animator>();
22     }
23
24     void FixedUpdate()
25     {
26         Movimiento();
27         ResetPosition();
28
29         animator.SetFloat("Velocity", rigidbody.velocity.y);
30
31         if (CheckGround.touching)
32         {
33             animator.SetBool("inGround", true);
34         }
35         else
36         {
37             animator.SetBool("inGround", false);
38         }
39     }
40
41     void Movimiento()
42     {
43         bool isMoving = false;
44
45         // Movimiento hacia la derecha
46         if (Input.GetKey(KeyCode.D)) // Derecha
47         {
48             rigidbody.velocity = new Vector2(speed, rigidbody.velocity.y);
49             isMoving = true;
50             // El personaje mira a la derecha
51             transform.localScale = new Vector3(Mathf.Abs(transform.localScale.x), transform.localScale.y, transform.localScale.z);
52         }
53     }
54 }
```

Figura 3 - Captura del editor de Visual Studio



Con el proyecto ya creado, el primer paso es elegir a qué formato se va a exportar. Por defecto, está activa la opción para ordenadores (Windows, macOS y Linux), pero ya que el proyecto se enfocará en la web, la opción idónea es WebGL.

Se importan todos los elementos visuales y texturas que se van a utilizar. Unity es capaz de detectar *sprite sheets* y dividirlos automáticamente en los fotogramas necesarios, pero si no se realiza correctamente, se puede editar manualmente. Por otro lado, Unity tiene incorporado una tienda llamada *Asset Store* en la que se pueden obtener *scripts*, elementos visuales (*sprites*) u objetos (*GameObjects*) creados y desarrollados por otros usuarios que han subido su trabajo a esta tienda y se permite su uso a cambio de un importe económico.

Lo siguiente será crear, aunque sea de forma básica, el personaje utilizando cubos para definir su movimiento y colisiones. Para añadir estos componentes, es necesario incluir dentro de las características del *GameObject* los elementos *Rigidbody* (que gestiona la gravedad y las físicas del objeto) y *Box Collider* (que define la forma y los límites de colisión del objeto, generalmente de forma cuadrada pero ajustable en tamaño).

Una vez completado esto, el próximo paso es crear las animaciones usando las imágenes ya importadas y separarlas en animaciones distintas a través de la ventana *Animation*. Una vez que las animaciones necesarias estén guardadas, en la ventana *Animator* se organizan todas las animaciones creadas anteriormente en un controlador que gestionará la lógica para alternar las animaciones según variables ajustables por el desarrollador para adaptarse a cualquier escenario posible. Estas variables pueden modificarse desde el editor de Unity o mediante un script de código.

Para la creación del escenario en este proyecto de Unity, se utilizó una herramienta compuesta por dos componentes principales: *Tiles* y *Tilemap*. Los *Tiles* se generan a partir de imágenes, que se dividen en cubos simétricos. Estos *Tiles* representan pequeñas porciones del entorno del juego, como partes del suelo o paredes. Por otro lado, el *Tilemap* es la estructura organizativa que permite colocar y gestionar estos *Tiles* en una cuadrícula, formando así un mapa o nivel completo. Dado que los *Tilemaps* se utilizan comúnmente para suelos y otros elementos del escenario, ya cuentan con colisiones y físicas integradas, lo que facilita la recreación de un mundo coherente y funcional.

Con el personaje y el escenario ya creados, se desarrolla un *script* que se encarga de capturar las entradas del usuario desde el teclado (*inputs*) para aplicar fuerzas de movimiento y salto al personaje. Adicionalmente, si el desarrollador lo considera conveniente, puede hacer que las animaciones del personaje se modifiquen en respuesta a las pulsaciones del usuario, proporcionando una experiencia visual más dinámica y reactiva.

Un elemento fundamental para el correcto desarrollo de un juego es la capacidad de que la cámara siga al personaje a lo largo del nivel, asegurando que el jugador siempre permanezca centrado en el plano mostrado. Aunque existen muchos juegos que no utilizan esta lógica o que emplean variantes de la misma, en este proyecto se ha decidido implementarla para un análisis más completo. El seguimiento de la cámara se logra mediante un script que permite centrar la cámara en cualquier objeto seleccionado, en este caso, el personaje.

Otro aspecto importante, aunque de naturaleza más estética, es la inclusión de una imagen de fondo en el escenario. Para ello, se ha decidido que, al igual que la cámara, el fondo siga al personaje mediante el mismo *script* de la cámara. Esto asegura que la vista del usuario siempre tenga dicha imagen presente, mejorando la inmersión visual y la continuidad del escenario durante el juego.

Finalmente, para completar el nivel, se ha añadido información dentro de la capa de interfaz de usuario. Aunque se limita a instrucciones básicas sobre el funcionamiento del juego, este último añadido permite dar por terminado el desarrollo de un nivel completamente funcional.

Tras finalizar el desarrollo del primer juego, se puede concluir que Unity cuenta con múltiples herramientas incorporadas que facilitan un desarrollo fluido. Si bien es cierto que se requiere investigar dichas herramientas para comprender su funcionamiento y compatibilidades, una vez superada esta curva de aprendizaje, el desarrollo de juegos en Unity se vuelve bastante sencillo.

Es importante destacar la amplia información y documentación disponible, un ejemplo es la página oficial de Unity como en la *Unity Documentation*. Esta documentación se presenta en múltiples idiomas y se actualiza según la versión del motor utilizada. Además, existe un extenso catálogo de foros activos, donde los usuarios publican problemas y dudas, y buscan la ayuda de desarrolladores más experimentados.

En cuanto a compatibilidades, Unity puede instalarse en cualquier sistema operativo, ya sea macOS, Windows o Linux (aunque con menor soporte en este último). A la hora de exportar los proyectos, el motor es compatible con una amplia gama de plataformas, incluyendo consolas, ordenadores, dispositivos móviles, servidores y, como en el caso de esta investigación, formato web.

Durante el desarrollo, cabe destacar que, si no se tienen conocimientos previos de Unity, es necesario aprender a utilizar el motor y familiarizarse con C# para poder trabajar de manera efectiva. Hasta la fecha, la empresa detrás de Unity ha mantenido los costos originales sin modificaciones, por lo que, mientras se mantengan estas licencias, Unity seguirá siendo un entorno de desarrollo asequible para pequeñas empresas.

Finalmente, en cuanto al proyecto, una vez exportado, se genera una carpeta con los elementos básicos para la web, es decir, un archivo HTML, un archivo CSS, y varios archivos JavaScript, entre los que se encuentra la lógica del juego traducida para ser compatible con WebGL. También se incluyen archivos comprimidos con todo el arte proporcionado por el desarrollador, que posteriormente se descomprimen durante la ejecución al cargar el proyecto desde una página web.

## Godot

Este motor ha ganado mucha popularidad en los últimos años, ya que se presenta como un motor de uso libre y *open-source*. Esto significa que cualquier desarrollador puede crear versiones modificadas del motor sin costo alguno, aunque es necesario dar el crédito correspondiente a Godot en caso de que se realicen modificaciones.

El motor cuenta con un menú inicial que permite seleccionar entre distintos proyectos e incluso ofrece plantillas predefinidas como base. Existen diferentes versiones del motor, destacándose dos tipos principales: la versión LTS (*Long Term Support*), que se considera la más estable y se recomienda para proyectos de larga duración, y la última versión publicada, que incorpora más novedades y se actualiza constantemente, lo cual podría generar posibles fallos o inestabilidades.

La interfaz de este motor es altamente personalizable y cuenta con un sistema de pestañas preconfiguradas, como se puede observar en la Figura 4. Estas pestañas se pueden modificar según las necesidades del desarrollador.

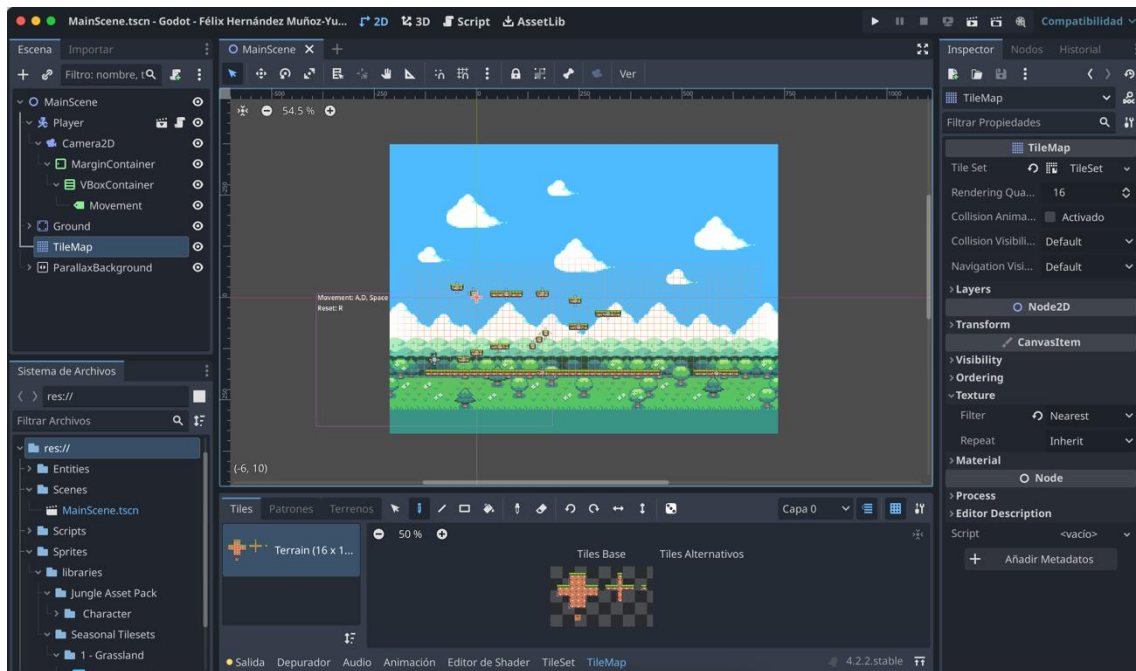


Figura 4 - Captura del editor de Godot

Este motor organiza sus componentes en nodos, los cuales son muy versátiles. Los nodos pueden servir como la escena principal que agrupa otros nodos, o pueden representar entidades específicas como personajes, imágenes, entre otros. Godot permite añadir nodos a través de un menú que muestra todos los nodos disponibles y pre-creados dentro del motor.

En la Figura 4, se puede observar, en el panel derecho, todos los elementos configurables para cada nodo. Estos elementos suelen ser comunes entre los nodos, pero también existen configuraciones específicas en función del tipo de nodo y su función particular.

En la parte izquierda de la interfaz, se encuentra un árbol jerárquico que muestra la organización de los nodos en grupos. Debajo de esta área, está el explorador de archivos, donde se organiza y gestiona el proyecto.

En cuanto al desarrollo en código, este motor es compatible con varios lenguajes de programación, incluyendo C#, C++ (el lenguaje en el que el motor mismo está desarrollado), y su propio lenguaje de programación, llamado GDScript, que tiene una gran similitud con Python. El editor de código está integrado en el propio motor, lo que elimina la necesidad de usar programas externos para la edición y desarrollo de scripts.

Al iniciar el desarrollo del proyecto en Godot, es fundamental determinar el formato al que se exportará el proyecto. Godot ofrece tres opciones principales:

1. Forward+: Esta opción está diseñada para juegos que requieren gráficos de alta calidad y una mayor escala, orientándose principalmente a ordenadores y consolas. Es ideal para proyectos que demanden un alto rendimiento gráfico.
2. Móvil: Esta opción está pensada para juegos menos exigentes en términos de gráficos, lo que facilita su exportación a dispositivos móviles, además de ordenadores y consolas. Es una buena elección para desarrolladores que desean alcanzar una audiencia más amplia en plataformas móviles sin comprometer la calidad.
3. Compatibilidad: Esta opción se limita a gráficos menos exigentes, pero es compatible con un mayor número de dispositivos, incluyendo ordenadores, consolas, móviles y el formato web. Ofrece una solución más versátil para proyectos que buscan ser accesibles desde múltiples plataformas.

Dado el enfoque de esta investigación, se ha optado por utilizar el formato de Compatibilidad. Esto permitirá que el proyecto sea accesible en una amplia gama de dispositivos, incluyendo la posibilidad de ejecución en formato web, alineándose con los objetivos establecidos para el desarrollo del juego.

Una vez que el proyecto ha sido creado e inicializado en Godot, se genera una escena con un nodo "raíz". Este nodo actúa como el punto de partida desde el cual se pueden crear nuevos nodos "hijo". Como se muestra en la Figura 4, en la ventana superior izquierda, se puede observar la organización de los nodos que componen el nivel.

Al crear el personaje, es recomendable iniciar con una nueva escena específica para gestionar todos los elementos y cambios asociados al personaje. Esto permite una organización más limpia y facilita la reutilización del personaje en diferentes partes del juego. Para lograrlo, se crea una nueva escena donde el nodo raíz es de tipo *CharacterBody2D*. Este nodo es ideal para personajes jugables, ya que permite ajustar propiedades esenciales como la gravedad y las colisiones que afectarán al personaje.

Para gestionar las animaciones del personaje, se agrega un nodo *AnimatedSprite2D* como hijo del nodo raíz del personaje. Este nodo es responsable de manejar todas las animaciones del personaje. Permite cargar diferentes secuencias de imágenes, dividiéndolas en el número de fotogramas correspondientes a cada animación. Las animaciones se ejecutarán basándose en llamadas específicas dentro del *script* que controla el comportamiento del personaje.

Dentro de la escena principal, Godot proporciona un nodo que encapsula la herramienta *Tilemap*, lo cual permite usar esta herramienta para construir el entorno del juego de manera eficiente. Para utilizar el *Tilemap*, es necesario crear un recurso *Tile* y dividir el *sprite* del terreno que se va a utilizar, especificando las dimensiones de los cubos en los que será dividido. Esto facilita la construcción del escenario, ya que permite colocar múltiples cubos para formar el nivel.

En este caso particular, las colisiones no se añaden automáticamente a los cubos en el *Tilemap*. Por lo tanto, es necesario incorporar un nodo *CollisionShape2D*. Este nodo permite crear colisiones con forma de cuadrado, ajustando su tamaño según sea necesario. Para gestionar mejor las colisiones, se puede crear un grupo de nodos. Esto implica usar un nodo vacío que actúa como contenedor, agrupando múltiples nodos de colisión para todas las plataformas presentes en el nivel. Asimismo, dentro de la escena del personaje, se añade un nodo *CollisionShape2D* para definir el área de colisión específica del personaje.

En cuanto a las interacciones por parte del usuario en Godot, es necesario configurar las teclas que se van a utilizar en el proyecto. Esto se realiza a través de las opciones del proyecto, específicamente en una pestaña llamada "Mapa de Entrada". En esta sección, se debe asignar un nombre a cada acción o efecto que se pretende implementar y, posteriormente, asociar las teclas que activarán dicha acción. Por ejemplo, para permitir el movimiento hacia la derecha, se podría crear una acción llamada "Derecha" y asignar las teclas 'D' y la flecha derecha para ejecutarla.

Una vez que todas las acciones han sido asignadas en el Mapa de Entrada, se procede a crear un *script* asociado al nodo del personaje. Este *script* contendrá todas las funciones necesarias para manejar el movimiento basado en las entradas del teclado y también controlará las animaciones que se ejecutarán en función del movimiento del personaje. Este enfoque permite que las animaciones cambien automáticamente según las acciones del jugador, ofreciendo una respuesta visual coherente con los movimientos.

Después de configurar y para probar el *script*, se puede instanciar el personaje en la escena principal, asegurando que todos los elementos estén correctamente vinculados y funcionando en conjunto.

Otro aspecto fundamental en el diseño de niveles es hacer que la cámara siga al personaje. Gracias a la estructura jerárquica de nodos en Godot, esto se puede lograr de manera sencilla arrastrando el nodo de la cámara dentro del nodo de la instancia del personaje. Al hacerlo, la cámara automáticamente seguirá los movimientos del personaje, manteniéndolo centrado en la pantalla.

Para añadir un fondo de pantalla que se mantenga alineado con la vista del usuario, Godot proporciona nodos específicos como *ParallaxBackground*. Este nodo permite configurar un fondo que se mueve a una velocidad diferente a la del primer plano, creando un efecto de paralaje. Al utilizar *ParallaxBackground*, se asegura que el fondo siempre esté presente y visible para el usuario, enriqueciendo la experiencia visual del juego.

En Godot, la interfaz de usuario (UI) se organiza mediante nodos específicos. Para gestionar la disposición de elementos en la UI, se utilizan nodos como *MarginContainer*, que determinan la posición relativa de los elementos respecto a los márgenes de la pantalla. Dentro de este nodo, se pueden añadir otros nodos como *VBoxContainer*, que permiten organizar elementos de UI en formato de lista vertical. Finalmente, para mostrar texto, se utiliza el nodo *Label*, que se inserta dentro de los contenedores y permite agregar el texto deseado que se mostrará en pantalla.

Con todos estos elementos implementados, se da por concluido el desarrollo del nivel. Se puede afirmar que Godot es una alternativa que posee múltiples herramientas facilitando la experiencia del desarrollador y permitiendo un trabajo mucho más ágil.

Es importante destacar que Godot cuenta con una documentación oficial llamada *Godot Docs*, la cual es un recurso valioso para los desarrolladores. Además, la comunidad de usuarios de Godot contribuye activamente al desarrollo del motor, ya que permite que estos modifiquen y añadan sus propias ideas. Aunque la tienda de recursos del motor, conocida como *AssetLib*, no es tan popular como las de otros motores, ofrece una variedad de recursos que pueden ser descargados gratuitamente o comprados, todos ellos compartidos por la comunidad.

Godot es compatible con múltiples sistemas operativos, ya que puede instalarse en ordenadores con Windows, macOS y Linux. Además, cuenta con una aplicación para dispositivos móviles Android, permitiendo a los desarrolladores trabajar directamente desde sus dispositivos. También es posible ejecutar una versión del motor directamente desde un navegador web, lo que amplía aún más sus posibilidades de uso.

Si bien la estructura de nodos planteada por Godot puede ser vista como una limitación al momento de crear juegos que requieren elementos específicos no disponibles, esta misma estructura facilita el acceso y aprendizaje del motor, haciéndolo más intuitivo para nuevos usuarios.

Por otro lado, el lenguaje de programación propio del motor, GDScript, es muy similar a Python. Aunque esta similitud puede ser una ventaja, es necesario aprender los detalles y particularidades del lenguaje para utilizarlo correctamente dentro del motor.

Respecto a la exportación del proyecto, al estar diseñado para formato web, Godot genera una serie de archivos que incluyen una cabecera HTML, archivos de JavaScript, y varios paquetes comprimidos con los recursos del motor. Una vez compilados, estos permiten que el juego se ejecute directamente desde la web.

En conclusión, a pesar de sus limitaciones, Godot es un motor muy capacitado para competir con otros motores del sector. Su modelo de licencia gratuita, que no impone costes para la publicación de juegos desarrollados con este motor, lo convierte en una opción muy atractiva para desarrolladores independientes y pequeñas empresas.

## Construct 3

Este motor se caracteriza por su accesibilidad, ya que no requiere descargas ni programación para su uso. Se ejecuta directamente desde su sitio web, permitiendo su utilización desde cualquier lugar o dispositivo que tenga conexión a internet.

El motor comienza con un selector de proyectos, que permite crear nuevos proyectos con las especificaciones necesarias. Una característica destacada es la capacidad de guardar los proyectos en servicios de almacenamiento en la nube como Google Drive, OneDrive o Dropbox. Esto facilita un guardado automático y garantiza que el proyecto esté disponible en cualquier momento y desde cualquier lugar. Además, Construct guarda todo el proyecto en un archivo comprimido con una extensión propia (.c3p).

La interfaz del motor no es muy configurable, más allá de la posibilidad de mover las ventanas del editor. Sin embargo, cuenta con varias pestañas que proporcionan acceso a todos los elementos necesarios para el desarrollo. En la Figura 5 se puede observar el editor del motor.

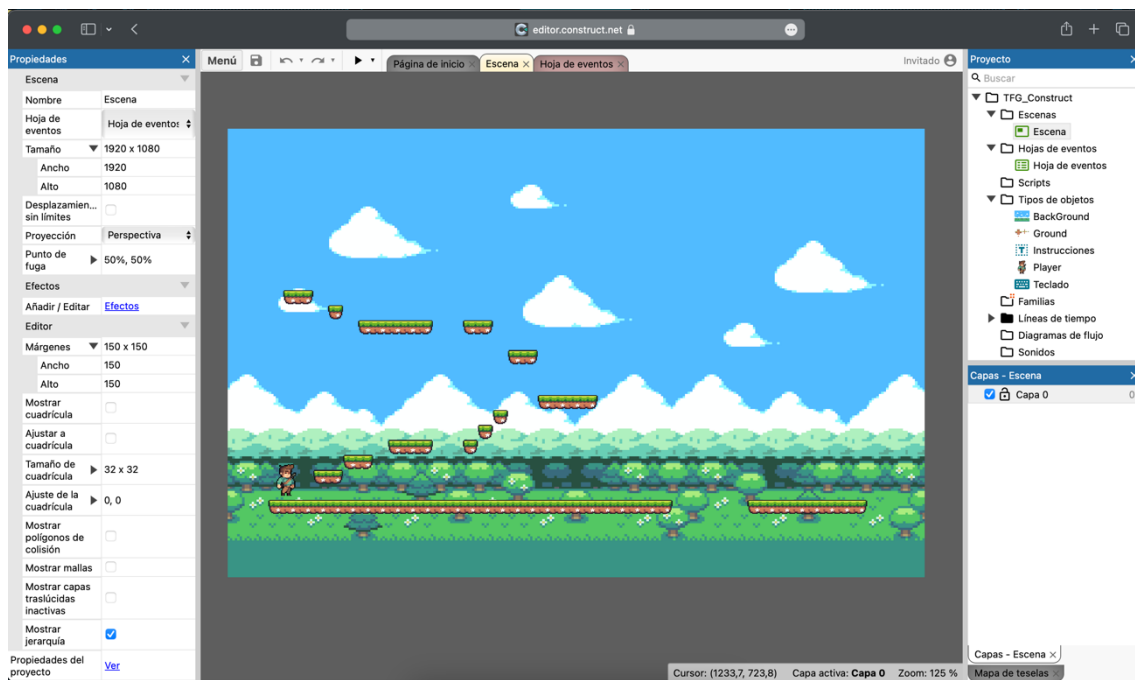


Figura 5 - Captura del editor de Construct 3

El motor está organizado con múltiples opciones para desarrollar una variedad de juegos. Sin embargo, es importante mencionar que solo es posible utilizar las herramientas y componentes que el propio motor proporciona. Aunque ofrece un catálogo bastante extenso de opciones, no permite la integración de componentes externos o personalizados.

Como se muestra en la Figura 5, el panel izquierdo del editor presenta todas las opciones y modificaciones disponibles para cada objeto seleccionado. En el panel derecho, se pueden ver todos los objetos que han sido añadidos al proyecto. En la parte superior, se encuentran una serie de pestañas que representan la escena actual y proporcionan acceso a otras funcionalidades, como regresar a la pantalla de inicio o acceder a la Hoja de eventos, donde se almacenan todas las instrucciones programadas por el desarrollador.

Una vez que el proyecto ha sido creado y se ha configurado la relación de tamaño de la pantalla, el siguiente paso en este motor es la creación del personaje. Para ello, es necesario añadir un objeto denominado *Sprite*. Al instanciar este objeto, es fundamental asignarle un comportamiento específico, dependiendo de la función que desempeñará en el juego. En este caso, dado que se trata de un personaje en un mapa 2D con movimiento lateral, es necesario configurar el comportamiento de Plataforma dentro de las opciones del sprite. Al hacerlo, el objeto adquiere una serie de propiedades específicas para este tipo de juego, como la velocidad máxima, la aceleración, la capacidad de realizar un doble salto, entre otras. Todas estas opciones vienen preconfiguradas en el motor, y el usuario solo necesita ajustarlas según las necesidades de su juego.

Además, al instanciar el objeto del personaje (*sprite*), se abre un pequeño editor que permite añadir *sprite sheets* para las animaciones. Este editor facilita la gestión de las animaciones del personaje, ya que el mismo objeto del personaje es capaz de dividir y almacenar sus distintas animaciones.

Para la creación del suelo y las plataformas, se utiliza otra herramienta llamada *Tilemap*. Este editor permite diseñar el terreno sobre el cual se moverá el personaje. Al igual que con el personaje, es necesario seleccionar dentro de las opciones del motor la opción de Mapa de Teselas. Una vez añadida la imagen que se usará como textura, se diseña el nivel dentro del escenario añadiendo los bloques necesarios mediante la herramienta de "pintura" que proporciona el editor. Al igual que con el personaje, se puede asignar un comportamiento específico llamado Sólido, que otorga a todos los bloques añadidos colisiones y evita que sean afectados por la gravedad.

Gracias a los comportamientos predefinidos del motor, tanto las colisiones como la gravedad del personaje y del escenario quedan configuradas automáticamente.

Para gestionar las entradas del usuario, el movimiento del personaje y el cambio de animaciones en función de su movimiento, Construct ofrece una herramienta denominada Hoja de Eventos. Esta hoja se basa en una serie de condicionales que permiten la gestión de los elementos añadidos al proyecto. En la Figura 6 se pueden observar los eventos registrados para este nivel.

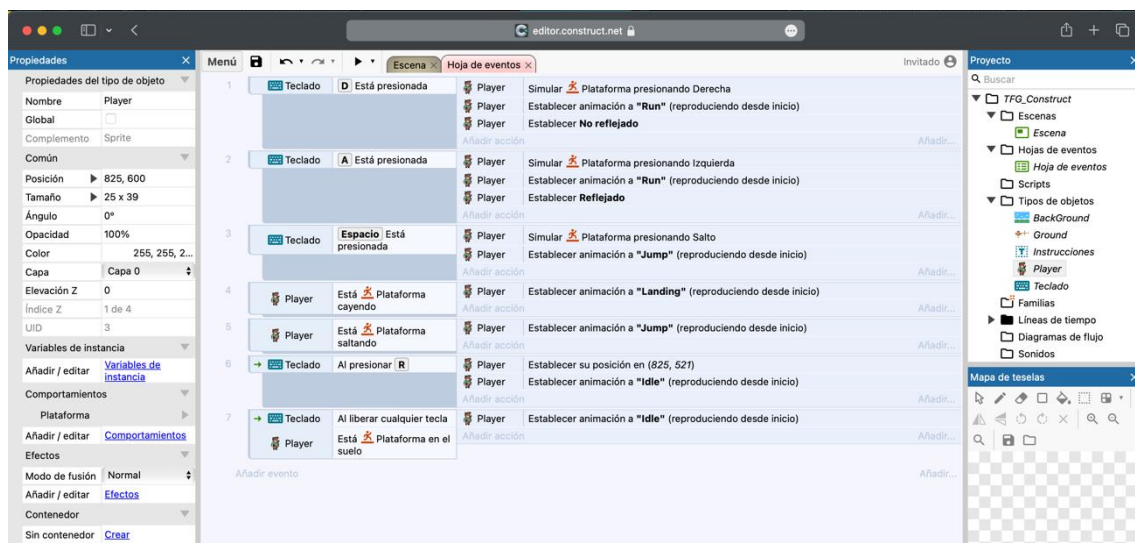


Figura 6 - Captura Hoja de Eventos de Construct 3



En cuanto al seguimiento de la cámara, Construct ofrece una funcionalidad básica llamada *DesplazarHasta*. Este comportamiento permite que la cámara se centre automáticamente en un objeto específico, como el personaje, lo que asegura que el jugador siempre lo tenga en el centro de la pantalla.

Por otro lado, para la gestión del fondo de pantalla, Construct permite la generación de fondos mediante mosaicos, creando un efecto de repetición infinita que es útil para niveles largos o sin fin. Sin embargo, debido a que estos fondos son estáticos, no es posible animarlos o interactuar con ellos de manera avanzada. La imagen del fondo debe colocarse al final del nivel, limitando su dinamismo.

Para la inserción de información en la interfaz del usuario, Construct proporciona un objeto llamado *Texto*. Este objeto permite añadir y mostrar texto directamente en la interfaz, sin necesidad de configuraciones adicionales, lo que facilita la creación de una *UI* o menús informativos de manera rápida y sencilla.

Una vez completado el desarrollo del nivel, se puede concluir que Construct, aunque limitado en términos de libertad creativa, es una excelente herramienta para el desarrollo de pequeños juegos, especialmente para aquellos sin experiencia en programación. La falta de necesidad de escribir código es una ventaja significativa para principiantes, aunque el motor también permite el uso de scripts en JavaScript para aquellos que prefieran personalizar más su proyecto. En este nivel, no fue necesario utilizar scripts personalizados.

Construct cuenta con una documentación oficial, denominada *Manual & Documentation*, disponible en su sitio web oficial. Aunque su comunidad de usuarios es relativamente pequeña, es activa, lo que facilita obtener respuestas a dudas que la documentación no pueda resolver.

En términos de compatibilidad, Construct se ejecuta directamente en el navegador, lo que significa que puede utilizarse en cualquier dispositivo con acceso a internet. Sin embargo, Construct está principalmente diseñado para crear juegos en formato web, lo que lo hace ideal para dispositivos con navegadores, como ordenadores y móviles. También permite la exportación de juegos como minijuegos en forma de anuncios para internet.

Aunque el motor es relativamente fácil de usar una vez que se dominan las herramientas disponibles, su naturaleza cerrada puede requerir un período de aprendizaje inicial. Una vez comprendidas sus funcionalidades, Construct permite un desarrollo rápido y ágil.

En cuanto a la exportación del proyecto, al seleccionar el formato HTML5, se genera un proyecto comprimido que incluye archivos HTML y JavaScript. Estos archivos contienen el motor de ejecución de Construct junto con las imágenes y otros recursos subidos por el usuario.

En resumen, Construct es una excelente opción para desarrolladores sin mucha experiencia que buscan una herramienta accesible y fácil de usar que se puede ejecutar desde cualquier lugar. Sin embargo, el hecho de que se requiera una licencia de pago para eliminar la marca de agua del motor o para desbloquear más funcionalidades puede limitar su atractivo para aquellos que buscan una solución completamente gratuita.

## Phaser Editor (Phaser.js)

Phaser es un popular motor de desarrollo de videojuegos en 2D, que ofrece dos formas principales de desarrollar un proyecto. La primera es mediante el uso directo de la librería Phaser.js, que se puede descargar desde el sitio web oficial de Phaser. Esta librería puede ser utilizada en cualquier editor de código, como Visual Studio Code, y permite a los desarrolladores escribir el código del juego de manera manual.

Por otro lado, existe una herramienta más visual llamada Phaser Editor. Este entorno de desarrollo ha sido creado para facilitar el uso de Phaser, permitiendo a los desarrolladores construir juegos en un entorno más gráfico e intuitivo. Phaser Editor se encuentra actualmente en su cuarta versión, y está diseñado para ofrecer una experiencia de desarrollo más accesible, especialmente para quienes prefieren una interfaz visual en lugar de trabajar directamente con código.

Al abrir Phaser Editor, lo primero que se muestra es una lista con todos los proyectos creados previamente, junto con la opción de crear nuevos proyectos. Al crear un nuevo proyecto, el desarrollador tiene varias opciones:

- Proyecto completamente vacío: Permite comenzar desde cero sin ninguna configuración predeterminada.
- Proyecto con bases de inicio: Ofrece plantillas básicas que proporcionan una estructura inicial.
- Ejemplos completos: Proyectos predesarrollados por el equipo de Phaser que sirven como referencia o punto de partida.
- Proyectos de Phaser vacíos con estructura para *Frameworks*: Proyectos vacíos pero preconfigurados para ser compatibles con *frameworks* externos como Angular, React, entre otros.

Una vez que se selecciona el tipo de proyecto, el editor muestra su interfaz. Este editor no es altamente personalizable, aunque permite mover algunas ventanas. La interfaz se organiza de la siguiente manera:

- Parte izquierda: Se muestran los elementos que componen la escena actualmente abierta.
- Parte inferior: Se divide en dos secciones; a la izquierda, una vista completa de la estructura de carpetas del proyecto, y a la derecha, los elementos importados y creados dentro del proyecto.
- Parte derecha: Aquí se presentan las propiedades y características de los elementos seleccionados dentro del editor.
- Parte superior: Se encuentran las pestañas que representan diferentes grupos de objetos o archivos abiertos, como la escena del nivel y el *script* de JavaScript correspondiente. Es importante destacar que Phaser Editor cuenta con un editor de texto integrado, permitiendo a los desarrolladores escribir y editar scripts directamente desde la misma interfaz.

Todos los elementos detallados pueden apreciarse en la Figura 7:

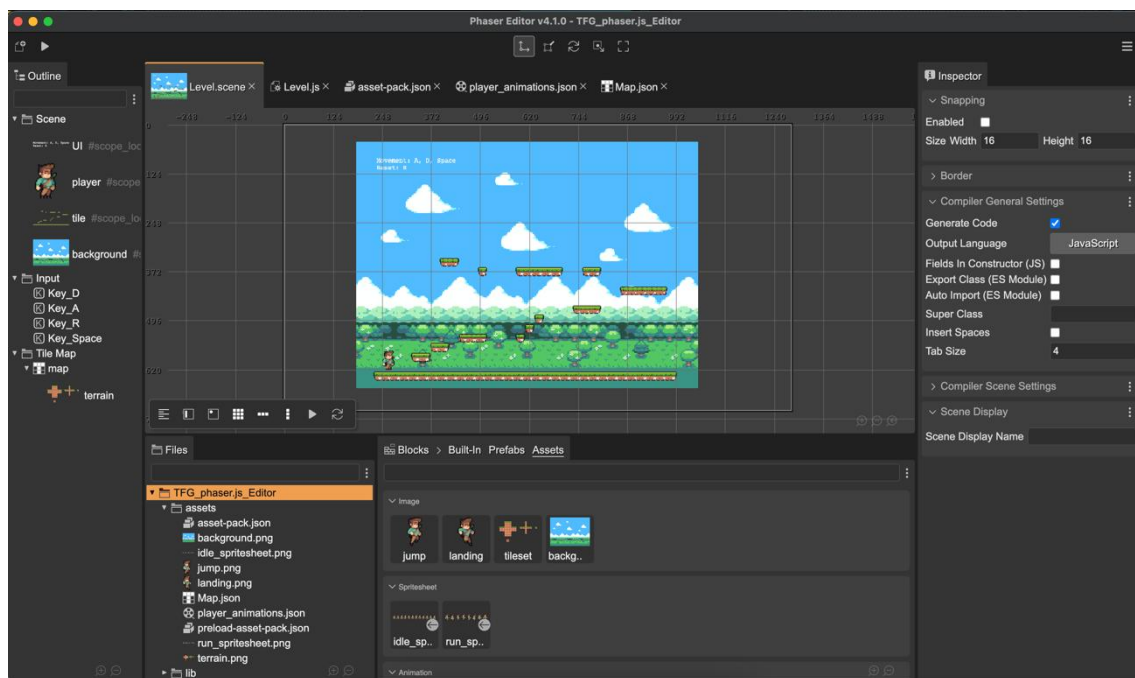


Figura 7 - Captura del editor de Phaser Editor

Una vez creado el proyecto, el primer paso es desarrollar el personaje que se utilizará. En este caso, se puede utilizar un *sprite* o, para este nivel en específico, un tipo específico llamado *ArcadeSprite*, que ya posee características integradas como la gravedad y las colisiones.

Al importar recursos de arte al proyecto, es necesario añadirlos a un archivo específico para que puedan ser utilizados. Estos archivos se llaman *asset-pack.json* y se pueden importar como imágenes o como *sprite sheets*. Luego, es necesario crear un objeto que encapsule todas las animaciones llamado *Animation File*, que permite combinar los distintos fotogramas y crear animaciones a partir de los *sprite sheets* ya importados. Este archivo *Animation File* también se guarda dentro del paquete de *assets* mencionado anteriormente.

Una vez terminadas las animaciones, se pueden invocar para ser reproducidas en el *sprite* del personaje a través de un *script*.

Para la creación del suelo y las plataformas, este editor no cuenta con una herramienta integrada capaz de hacerlo, pero permite añadir mapas creados con una estructura de *Tilemap*. La herramienta recomendada en la documentación para esto es Tiled. Tiled permite crear mapas con la estructura de distribución de los *Tilemaps*; es decir, a través de una imagen se divide en cubos, y con estos se puede "pintar" el nivel basándose en dichas imágenes recortadas. Una vez terminado el mapa, el editor de Phaser solo es compatible con una versión exportada en JSON del nivel completo.

Una vez terminado el mapa en Tiled, se añade al proyecto, se guarda dentro del *asset-pack.json* y se instancia un objeto llamado *Tile Map* que permite mostrar dicho mapa.

En este proyecto, al contar con la gravedad ya presente en el *sprite* del personaje, solo es necesario añadir las colisiones con el nivel. Con unas pocas líneas de código en el *script* del nivel se puede gestionar la colisión entre el *sprite* del personaje y las plataformas del *Tilemap*.

Dentro del editor de Phaser, se pueden configurar las teclas a las que el *script* debe reaccionar. Una vez configuradas en el editor, se instancian dentro del *script* del nivel y se desarrolla el código que permite tanto el movimiento como el cambio de animaciones del personaje.

En este editor, no existe una opción para modificar la cámara o el fondo del nivel de otra forma que no sea a través del código del nivel.

Finalmente, para añadir texto en la interfaz de usuario (UI), el editor ofrece un objeto muy sencillo llamado *Text*, que permite escribir el texto directamente desde el editor.

Con esto se puede concluir el desarrollo del nivel en este editor. Podemos destacar que la idea de este motor basado en JavaScript es innovadora, pero aún parece poco desarrollada y necesita más funcionalidades para ser realmente útil.

La documentación oficial de Phaser.js es bastante extensa y la web de los desarrolladores incluye múltiples ejemplos en formato de código, con la posibilidad de ver la ejecución de dicho código en la misma web. Sin embargo, la documentación de Phaser Editor es limitada y caótica, ya que se intentan explicar todas las funcionalidades del editor, pero sin detallar sus usos de manera clara. La comunidad también refleja esta disparidad; hay mucha más actividad en torno a la librería que al editor.

La librería de Phaser es compatible con cualquier navegador capaz de ejecutarlo, mientras que el editor solo es compatible con Windows, macOS y Linux. En cuanto a la exportación, Phaser está completamente orientado al formato web.

Es importante señalar que la curva de aprendizaje de este editor es más pronunciada que la de otros motores. Es necesario conocer JavaScript, comprender las aplicaciones que ofrece Phaser.js y entender las herramientas del editor para poder utilizarlo de manera efectiva.

Además, este editor no es gratuito; se requiere una suscripción mensual para poder descargarlo y utilizarlo.

Finalmente, en cuanto al proyecto exportado, el editor mantiene la estructura de un proyecto de Phaser.js, con archivos HTML y de JavaScript que el editor va modificando en función de los cambios realizados.

En conclusión, este editor no es de los más recomendados para desarrollar un proyecto, a menos que sea necesario utilizar específicamente la librería de Phaser.js. El coste añadido, junto a la falta de herramientas y documentación, puede dificultar significativamente el desarrollo del proyecto.

## Defold

Este motor ha ganado popularidad en los últimos años gracias a su liberación total y al cambio de licencias que lo han hecho completamente gratuito. Está diseñado principalmente para desarrollar juegos que requieren alta calidad gráfica, aunque también es compatible con el formato web, lo que permite adaptarse a diferentes escenarios.

Al iniciar el motor, se presenta un menú con todos los proyectos creados por el desarrollador y la opción de crear nuevos proyectos. Para los proyectos nuevos, se puede optar por crear un proyecto vacío, pero ya preparado para la plataforma a la que se planea exportar el juego. También se incluyen ejemplos predefinidos y plantillas más avanzadas que destacan herramientas específicas del motor.

A continuación, se muestra en la Figura 8 el motor de Defold:

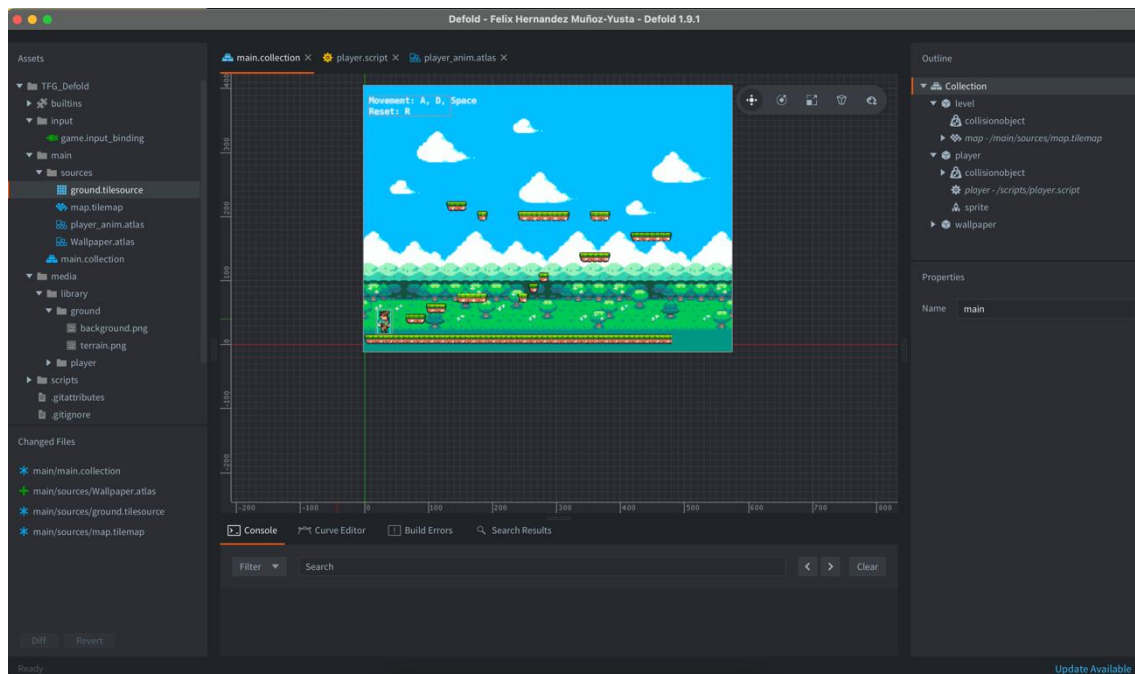


Figura 8 - Captura del editor de Defold

Aunque la interfaz no es configurable, es intuitiva y fácil de entender. La interfaz se organiza de la siguiente manera: en el lado izquierdo, se muestra la estructura de carpetas del proyecto, y debajo, los cambios realizados desde el último guardado en Git. En la parte inferior, se encuentran diversas herramientas de apoyo dentro del motor. A la derecha, se visualiza la estructura del nivel o, si se selecciona un objeto distinto, se muestran sus características y opciones. En la parte superior, se listan los distintos objetos del motor que pueden ser editados.

La estructura de este motor se organiza en torno a sus diversas herramientas, encapsuladas como si fueran objetos dentro del motor. Por ejemplo, una pestaña puede representar la escena, otra la configuración del proyecto, un conjunto de imágenes (*atlas*), la herramienta para crear terreno con *Tilemap*, entre otras.

Una vez creado el proyecto para la plataforma a la que se desea exportar, se muestra una interfaz con múltiples elementos ya configurados. Las escenas dentro de este motor se denominan *collections*, y se encargan de agrupar todos los elementos necesarios.

El primer objeto que se añade es el personaje. Para incorporarlo a la escena, llamada *collection* en el motor, se crea un *GameObject* vacío, y dentro de este se agrega el componente correspondiente al personaje, que, en un principio, no es más que un componente vacío.

En este motor, los *sprite sheets* de las animaciones pueden incluirse dentro de un objeto llamado *atlas*, que permite almacenar todos los fotogramas de las distintas animaciones y agruparlos según su animación correspondiente. Una vez creadas y organizadas las animaciones, estas se pueden vincular al componente del personaje. Para alternar entre las animaciones, se puede hacer uso de un *script*.

A la hora de crear las plataformas del nivel, se puede utilizar la herramienta de *Tilemaps* que incluye el motor, ya que permite crear plataformas de manera eficiente. A través de un archivo *tilesource*, se añade la imagen a partir de la cual se diseñará el nivel, que luego se guarda en un archivo llamado *tilemap* dentro del motor.

Con las plataformas y el personaje ya instanciados en la escena (*collection*), es necesario añadir las colisiones y la gravedad. Para ello, se debe agregar al objeto del personaje un componente llamado *sprite*, que permite aplicar propiedades físicas al cuerpo del personaje. En cuanto a las colisiones, es necesario incluir un componente llamado *CollisionObject*, el cual permite añadir colisiones utilizando formas geométricas básicas como cubos, esferas o cilindros (para entornos en 3D).

En lo que respecta al movimiento del personaje, primero se deben configurar las teclas que se utilizarán en el nivel. Esto se realiza en un archivo predeterminado del proyecto, llamado *game.input\_binding*. En este archivo, es necesario especificar las teclas que el motor debe detectar y asignarles un nombre para la acción correspondiente. Una vez configurado, se debe crear un script (escrito en Lua) donde se gestionará todo lo relacionado con el movimiento del personaje y el intercambio de animaciones según el movimiento realizado.

Para hacer que la cámara siga al personaje, es necesario, mediante un *script*, hacer coincidir las coordenadas del personaje y de la cámara en los ejes x e y, con una diferencia que puede ser ajustada por el usuario. De manera similar, si se desea que el fondo del nivel mantenga una alineación constante con el plano de la cámara, la imagen de fondo debe seguir las coordenadas de esta.

Finalmente, para añadir texto en el canvas del usuario (*UI*), se debe agregar un componente llamado *Label* dentro de un *GameObject* vacío o preexistente, preferiblemente el mismo donde se encuentra la imagen del fondo del nivel. Una vez implementado este componente, se puede añadir el texto deseado en la interfaz.

En conclusión, el desarrollo de un nivel en Defold es un proceso asequible. Si bien es cierto que el motor no permite una gran libertad en la modificación de componentes, ofrece suficientes herramientas para proporcionar un control adecuado y flexible, incluso con las limitaciones que presenta.

Defold cuenta con una documentación oficial accesible en su página web, la cual facilita la comprensión de las distintas herramientas y de la estructura del motor. Sin embargo, debido a que es un motor relativamente reciente, su comunidad está aún en proceso de desarrollo.

En cuanto a su compatibilidad, Defold es capaz de ejecutarse en sistemas operativos como Windows, macOS y Linux. Una de sus principales fortalezas es la capacidad de exportar los proyectos a diversas plataformas, desde consolas de última generación hasta juegos en línea en plataformas como Facebook.

Si bien es necesario aprender tanto el uso del motor como el lenguaje de programación Lua, la documentación y la facilidad de las herramientas permiten que su manejo sea rápido y accesible.

En el contexto de esta investigación, una vez que el proyecto se exporta en formato web, se configura como una carpeta que contiene un archivo HTML y las bibliotecas necesarias en formato JavaScript. Además, los recursos requeridos se presentan en archivos comprimidos, listos para ser utilizados por el navegador cuando se ejecute el juego.

En conclusión, Defold es una opción viable para el desarrollo de videojuegos, destacándose especialmente por su capacidad multiplataforma, lo cual resulta muy atractivo para desarrolladores que deseen abarcar varias plataformas de videojuegos. Sin embargo, la curva de aprendizaje, aunque presente, puede verse afectada por la limitada comunidad, lo que podría dificultar la resolución de dudas que no estén cubiertas en la documentación.

## Roles de programación, Análisis y Relación de tareas

### Roles de programación

En cuanto a los roles de programación, al tratarse de una investigación individual, todos los aspectos relacionados con la investigación, el desarrollo y la comparación han sido realizados íntegramente por el autor de este Trabajo de Fin de Grado.

### Análisis

Dentro del análisis, es importante destacar que para visualizar los juegos desarrollados es necesario ejecutar los proyectos ya compilados en un navegador compatible, como Microsoft Edge, Google Chrome, Safari o Firefox. Es decir, cualquier navegador que soporte HTML5, ya que este es el estándar requerido para la ejecución de los proyectos. Para ello, lo más recomendable es utilizar un ordenador.

Si se desea abrir los proyectos en los motores de desarrollo, es necesario instalar estos en entornos compatibles. Lo más adecuado es utilizar sistemas operativos de escritorio como Windows, macOS o Linux. Esto aplica para motores como Unity, Godot, Phaser y Defold. En el caso de Construct, basta con acceder al editor directamente desde un navegador web compatible, como Safari, Chrome o Firefox.

En cuanto a los periféricos necesarios, un teclado de ordenador es suficiente para ejecutar los niveles desarrollados, y para trabajar en los motores, el uso de un teclado y un ratón es suficiente.

### Relación de tareas

En cuanto a las tareas llevadas a cabo por el usuario, se puede concluir que han sido organizadas y gestionadas de manera efectiva.

Respecto al desarrollo del videojuego, las tareas se organizaron para permitir un desarrollo simultáneo en todos los motores de manera paralela. Para lograr esto, el trabajo se dividió en bloques numerados de la siguiente forma:

1. Crear un proyecto en el motor.
2. Crear un personaje.
3. Añadir animaciones al personaje.
4. Crear el suelo y las plataformas.
5. Aplicar gravedad.
6. Añadir colisiones entre el suelo y el personaje.
7. Añadir inputs de movimiento al personaje.
8. Hacer que la cámara siga al personaje.
9. Añadir un fondo al nivel.
10. Añadir un texto en el canvas / UI (interfaz de usuario).

Dado que este proyecto ha sido concebido como una investigación, era necesario documentar todas las etapas del desarrollo. Por ello, todo el proceso de desarrollo en los diferentes motores se encuentra detallado en el apartado [Desarrollo del Proyecto](#) y especificado posteriormente en cada motor analizado.



En cuanto a las tareas de investigación realizadas, ha sido fundamental el estudio de los motores y sus componentes, es decir, las herramientas empleadas en cada motor, el lenguaje de *scripting* específico de cada uno, así como los requisitos necesarios para exportar el proyecto.

Es importante destacar que en todos los motores se ha requerido el uso de herramientas como los *Tilemaps*, los cuales se han utilizado para crear los mapas en 2D mediante el uso de una imagen de recurso (*Tile*), permitiendo diseñar el nivel y posteriormente implementarlo en la escena.

Además, ha sido necesario investigar los distintos lenguajes de *scripting* utilizados en cada motor. En el caso de Unity, se ha profundizado en el lenguaje C#; para Godot, se ha estudiado su lenguaje nativo, GDScript; en Construct no ha sido necesario aprender un lenguaje de programación como tal, pero sí comprender la lógica de las "Hojas de Eventos"; para Phaser, se ha trabajado con JavaScript; y finalmente, en Defold, ha sido preciso aprender Lua.

En el manejo de los motores, ha sido esencial aprender sus diferentes configuraciones y estructuras, ya que, en motores como Unity, Defold o Phaser, los objetos pueden parecer similares bajo el nombre de *GameObjects*. Sin embargo, en Godot esta estructura cambia, representando los objetos como nodos.

Otro aspecto relevante es que cada motor ofrece métodos propios para importar, modificar y visualizar imágenes. Cada uno tiene su propia manera de gestionar los elementos gráficos. Por un lado, algunos motores, como Unity, Godot y Construct, permiten dividir automáticamente los *sprite sheets* dentro del mismo motor. Por otro lado, en motores como Phaser y Defold, es necesario separar y modificar manualmente los *sprite sheets* en imágenes individuales utilizando editores externos.

## Resultados y conclusiones

### Resultados

La investigación se ha centrado en el análisis y comparación de cinco motores de videojuegos 2D orientados al desarrollo para plataformas web: Unity, Godot, Construct 3, Phaser.js y Defold. Los resultados obtenidos se dividen en dos apartados:

#### Resultados técnicos:

- **Compatibilidad web:** Todos los motores seleccionados cumplen con los requisitos de exportación para juegos web. Cada motor es capaz de generar archivos HTML y JavaScript compatibles con los principales navegadores (Chrome, Firefox, Safari, etc.).
- **Plataformas de soporte:** Unity y Defold destacan por ser motores multiplataforma, permitiendo la exportación no solo a formato web, sino también a consolas y dispositivos móviles. Construct 3 es el más accesible en términos de uso, ya que no requiere instalación.
- **Lenguaje de programación:** La curva de aprendizaje varía considerablemente entre los motores. Unity utiliza C#, Godot emplea GDScript (un lenguaje propio), mientras que Construct 3 simplifica la programación mediante hojas de eventos. Phaser.js se basa en JavaScript, y Defold utiliza Lua.
- **Documentación y soporte:** Unity y Godot destacan por su extensa documentación y sus grandes comunidades de apoyo, lo que facilita la resolución de problemas. Phaser.js también cuenta con buena documentación, aunque su editor gráfico (Phaser Editor) presenta ciertas limitaciones. Construct 3 y Defold, aunque cuentan con comunidades más pequeñas, tienen usuarios activos.

#### Desarrollo del videojuego:

- **Facilidad de uso:** Construct 3 es el motor más accesible para desarrolladores novatos, ya que no requiere conocimientos de programación. Unity y Godot, aunque demandan mayor preparación, ofrecen una mayor flexibilidad y potencia. Defold y Phaser.js, por su parte, brindan un control más detallado, pero exigen mayor familiarización con sus lenguajes respectivos.
- **Herramientas gráficas:** Unity ofrece mayor automatización, como la detección automática de *sprite sheets*, mientras que Phaser.js y Defold requieren más intervención manual. La inclusión de un sistema de *tilemaps* en varios motores (Unity, Godot, Construct y Defold) facilitó la creación de escenarios 2D.

## Conclusiones

El objetivo principal de este proyecto fue investigar, comparar y evaluar motores de desarrollo de videojuegos 2D compatibles con entornos web, lo cual se cumplió de manera satisfactoria. A través del desarrollo de un pequeño juego en cinco motores (Unity, Godot, Construct 3, Phaser Editor y Defold), se identificaron sus fortalezas y debilidades, lo que permitió determinar cuál es más adecuado según las necesidades del proyecto y del desarrollador.

Se evaluaron aspectos como facilidad de uso, curva de aprendizaje y capacidad de exportar a la web. Los resultados indicaron que Godot y Defold ofrecen flexibilidad sin costos, mientras que Construct 3 es simple pero limitado en personalización. Unity y Phaser Editor mostraron ser potentes, aunque con mayor complejidad o coste.

Además, se adquirieron conocimientos técnicos sobre motores y tecnologías web, y el proyecto aportó experiencia en gestión de proyectos y análisis comparativo. Los objetivos iniciales se lograron plenamente, y para el futuro se plantea investigar nuevos motores o explorar funciones multijugador.

En conclusión, aunque todos los motores evaluados son viables para el desarrollo de videojuegos, Unity sigue siendo la opción más recomendada, incluso para el formato web. Esto se debe a que ofrece el mayor número de beneficios para el desarrollador, tanto durante el proceso de desarrollo como en las etapas posteriores.

## Post mortem del desarrollo y líneas de futuro

En esta investigación individual, habría sido beneficioso desarrollar niveles más complejos, con mayores recursos, para llevar los motores de desarrollo al límite de sus capacidades. Esto habría permitido evaluar cómo gestionan la compresión de texturas y el manejo de múltiples eventos al compilar proyectos para el formato web.

Además, no fue posible realizar una comparación del consumo de recursos y rendimiento de los proyectos compilados al publicarlos en una página web de libre acceso.

En futuras investigaciones, sería interesante incluir un análisis de los costes de desarrollo, tanto para empresas como para desarrolladores independientes.

## Anexos

En la carpeta donde se encuentran esta memoria se encuentra una carpeta llamada “Anexos de Programación” donde se pueden encontrar los proyectos desarrollados de los diversos motores, por ejemplo “Proyecto\_Unity” y luego el juego final ya compilado llamado por ejemplo “Juego\_Unity”.

Es importante destacar el hecho de que los juegos desarrollados están enfocados para el formato web, es decir, que no son un ejecutable, es necesario abrir el archivo con extensión HTML en el navegador dentro de la carpeta sin modificar nada dentro de la carpeta una vez descomprimida.

## Bibliografía

- Gregory, J. (2014). *Game Engine Architecture* (Vol. 2nd Edition). Florida, Estados Unidos: CRC Press.
- Makzan. (2011). *HTML5 Game Development by Example: Beginner's Guide*. Birmingham, Reino Unido: Packt Publishing.
- MDN contributors. (24 de Julio de 2023). *JavaScript*. Obtenido de MDN web docs: <https://developer.mozilla.org/es/docs/Web/JavaScript>
- MDN contributors. (18 de Agosto de 2023). *WebGL*. Obtenido de MDN web docs: [https://developer.mozilla.org/es/docs/Web/API/WebGL\\_API/Tutorial](https://developer.mozilla.org/es/docs/Web/API/WebGL_API/Tutorial)
- MDN contributors. (2024 de Junio de 2024). *CSS*. Obtenido de MDN web docs: <https://developer.mozilla.org/es/docs/Web/CSS>
- MDN contributors. (28 de Julio de 2024). *HTML*. Obtenido de MDN web docs: <https://developer.mozilla.org/es/docs/Web/HTML>