ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE VIDEOJUEGO DE PLATAFORMAS 2D UTILIZANDO UNREAL ENGINE

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN DESARROLLO DE SOFTWARE

MATEO NICOLÁS BORJA RIVADENEIRA

DIRECTOR: ING. JUAN PABLO ZALDUMBIDE PROAÑO

DMQ, febrero 2022

CERTIFICACIONES

Yo, MATEO NICOLÁS BORJA RIVADENEIRA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

MATEO NICOLÁS BORJA RIVADENEIRA

mateo.borja@epn.edu.ec

mateo.nbr@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por MATEO NICOLÁS BORJA RIVADENEIRA, bajo mi supervisión.

JUAN PABLO ZALDUMBIDE PROAÑO

DIRECTOR

juan.zaldumbide@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

MATEO NICOLÁS BORJA RIVADENEIRA

DEDICATORIA

Este proyecto está dedicado a mí mismo, ya que logré probar que el sueño que tuve desde hace muchos años de realizar mi propio videojuego era posible. Lograr este desafío es un pequeño primer paso hacia un futuro brillante en el cual haré feliz con mis creaciones a muchos jugadores igual que yo he sido feliz jugando las grandes creaciones de otras personas.

MATEO NICOLÁS BORJA RIVADENEIRA

AGRADECIMIENTO

Quisiera agradecer a todas las personas que me apoyaron desde que esta idea surgió en mi cabeza hasta que escribí la última palabra en este documento.

Empezando por mis padres, quienes pese a ser ajenos al mundo de los videojuegos, entendieron mi pasión y constantemente se alegraban con mis avances y me apoyaban en lo que podían para que yo pueda dedicar más tiempo a este proyecto.

A mi hermano, a quien me siento orgulloso de haber introducido al mundo de los videojuegos pensando que un momento como este podría llegar algún día, un momento donde el sería el primer crítico de mi creación y usaría su propia experiencia para proporcionar las segundas opiniones que ayudaron a construir la experiencia que estaba buscando al hacer este videojuego.

A mi novia quien siempre estuvo ahí sin importar nada para levantar mis ánimos en los momentos en los que pensaba en darme por vencido o dudaba poder superar este desafío. Ella escuchó todos mis problemas pese a no entender muchos de ellos y me dio recompensas por mis esfuerzos que honestamente no merecía. Mi novia creyó en mi cuando yo no creía en mí mismo.

Finalmente, a mis amigos, quienes constantemente preguntaban cómo me estaba yendo en el proyecto y alababan mis avances cada vez que se los mostraba.

Saber que alguien aparte de mí mismo esperaba con ansias jugar mi juego me impulsó a terminarlo.

MATEO NICOLÁS BORJA RIVADENEIRA

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Este documento explica el proceso de desarrollo de un videojuego de plataformas 2D siguiendo la metodología SUM y utilizando Unreal Engine. El videojuego desarrollado tiene el título de "*Ricken the Chicken's Eggcellent Adventure"* y pone a los jugadores al control del gallo Ricken a través de 4 niveles llenos de obstáculos y enemigos.

Los videojuegos de plataformas 2D son simples y tienen un nivel de dificultad moderado o bajo para poder ser disfrutados por jugadores sin experiencia. Su objetivo es llegar al final de cada nivel lidiando con las amenazas presentes. El videojuego desarrollado cuenta con un menú de inicio, pantallas de información previas a cada nivel, pantalla final, función de pausar, guardar y cargar una partida y los 4 niveles únicos; todas estas funciones fueron codificadas usando *blueprints*¸ el sistema de codificación visual de Unreal Engine.

El documento empieza describiendo el proyecto de forma muy general y proporcionando sustento teórico sobre el tema, luego procede a elaborar sobre la metodología SUM utilizada, la descripción de las tareas realizadas en cada sprint y termina con conclusiones y recomendaciones.

PALABRAS CLAVE: Videojuego, Unreal Engine, SUM, Plataformas 2D, Blueprints.

ABSTRACT

This document explains the development process of a 2D Platformer videogame following the SUM methodology and using Unreal Engine. The title of the developed videogame is “Ricken the Chicken’s Eggcellent Adventure”, it puts the players in control of Ricken the chicken across 4 levels full of enemies and obstacles.

2D Platformer videogames are simple and have a medium or low level of difficulty so that they can be enjoyed by inexperienced gamers. Their objective is to get to the end of each level while dealing with the hazards that it has. The developed videogame has a start menu, introduction screens before every level, ending screen, the ability to pause, save and load the game and 4 unique levels; all these functionalities were coded using the engine’s visual scripting system called blueprints.

The document starts with a general overview of the project and a theoretical basis of the topic, then elaborates on the SUM methodology that was used and descriptions of the tasks encompassed by each sprint, finally it ends with conclusions and recommendations.

KEYWORDS: Videogame, Unreal Engine, SUM, 2D Platformer, Blueprints.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

Para la elaboración del componente de este trabajo de integración curricular, se desarrolló un videojuego de género plataformas 2D utilizando el motor gráfico Unreal Engine. El juego se puede ejecutar en una PC de 64 bits con sistema operativo Windows sin necesidad de conexión a internet, su dificultad es considerada media o alta dependiendo del nivel de experiencia del jugador y todo su contenido puede ser finalizado en menos de tres horas.

El juego fue diseñado con dificultad suficientemente moderada con el objetivo de que pueda ser jugado incluso por personas sin experiencia previa con videojuegos pero que, a la vez, constituya un desafío hasta no dominar todos los obstáculos presentes. Todas las acciones del juego son controladas con teclado para que sea jugable por personas que no poseen periféricos de juego, además, su interfaz se ha diseñado de forma intuitiva para que no haya posibilidad de confusión ante la forma de iniciar el juego o las acciones que se pueden realizar en él.

El juego desarrollado, cuyo título es *Ricken the Chicken's Eggcellent* Adventure, pone a los jugadores al control de *Ricken*, un joven gallo de segunda que no tiene la apariencia ni las habilidades de un héroe, pero que tendrá que asumir el rol de uno ante la ausencia del gallo principal líder del gallinero. A través de los 4 niveles del juego, *Ricken* intentará encontrar el paradero de los huevos que misteriosamente desaparecieron del gallinero en la granja del abuelo Joe; esta granja posee varios lugares donde buscar y enemigos que se interpondrán en dicha búsqueda, por lo que el jugador deberá dominar el salto, ataque y movimiento de *Ricken* para llegar al final de cada nivel. Al final de cada nivel, se observarán sencillas escenas para contribuir a la progresión del juego, el cual concluye con un final que dejará al jugador deseando saber que sucede luego.

Los juegos de plataformas 2D son de los primeros que se desarrollaron y, a menos que se quiera lo contrario, son más simples de desarrollar y más fáciles de jugar que los juegos en 3D; por esta razón se eligió este género para este proyecto.

1.1 Objetivo general

Desarrollar un videojuego de plataformas 2D utilizando Unreal Engine

1.2 Objetivos específicos

1. Crear una historia sencilla y personajes que brinden caracterización al videojuego
2. Diseñar el entorno del videojuego incluyendo escenarios, sonido, elementos (assets) utilizando Unreal Engine y otras herramientas
3. Construir la cantidad adecuada de contenido que permita que el videojuego sea una experiencia de usuario satisfactoria
4. Verificar el funcionamiento de todos los elementos que constituyen el videojuego para tener certeza de la calidad de este.

1.3 Alcance

El presente proyecto abarca la conceptualización, desarrollo y pruebas de un videojuego sencillo y corto que necesitará únicamente de una computadora de 64 bits con sistema operativo Windows para funcionar. El producto final es un videojuego que puede ser jugado de inicio a fin sin presentar errores, para lograr esto, se construyó una versión preliminar del juego, sobre la cual se realizaron pruebas para encontrar errores, estos errores se solucionaron en la versión final.

El proyecto fue desarrollado en base a la metodología SUM, la cual por sí misma se deriva de Scrum y, por lo tanto, tiene roles similares y un sistema de entrega continua a través de iteraciones. La fase de planeamiento es reemplazada por la conceptualización, pero esencialmente cumplen el mismo rol de definir qué es lo que se va a desarrollar. La etapa de implementación fue la más larga y consistió en un ciclo iterativo de desarrollo y evaluación de cada componente. La fase final fue la fase de pruebas, donde se examina la versión preliminar o Beta del video juego y se realizan las correcciones necesarias antes de entregar la versión final.

El producto final puede ser jugado por personas con o sin experiencia y se espera demostrar a través de él todo el potencial que tienen herramientas como Unreal Engine en el desarrollo de videojuegos de cualquier género y complejidad.

1.4 Marco Teórico

Antes de comenzar con el desarrollo de un videojuego conviene tener algo de contexto sobre el tema, el presente apartado expone temas de interés sobre los videojuegos en general y el género que se eligió para este proyecto con el objetivo de solidificar la idea del desarrollo de un videojuego como proyecto.

Videojuegos y sus ventajas

El auge de la tecnología en el siglo XXI ha provocado que cada vez más personas tengan acceso a un computador y un teléfono inteligente, dispositivos que han facilitado la vida del ser humano en todo aspecto, incluyendo el entretenimiento. Las computadoras han pasado de ser exclusivamente herramientas de trabajo a máquinas de ocio utilizadas para leer, socializar, consumir multimedia y jugar videojuegos; esto último se ha convertido en una forma de entretenimiento tan popular que, en el 2020, la industria de los videojuegos generó más ingresos que la industria cinematográfica y deportiva combinadas [1]. Los videojuegos siempre han tenido sus fanáticos dedicados, pero este aumento de popularidad ha hecho imposible que continúen siendo ignorados por las masas, quienes se han dividido entre los que aceptan a los videojuegos como un entretenimiento de la talla del cine y los que están en contra de ellos, viéndolos como un veneno para la mente de los jóvenes y una señal de inmadurez y vicio en los adultos. Pero estudios que se han realizado desde el 2008 señalan que los videojuegos no son culpables de lo que se les acusa, específicamente la violencia [2]. Otros estudios también prueban que jugar videojuegos trae muchos beneficios para todas las edades como aumento de motricidad fina, mejor capacidad para resolver problemas, predisposición a no darse por vencidos, etc. [3]

Breve historia de los videojuegos

Los videojuegos no siempre fueron las maravillas artísticas y audiovisuales que son hoy en día, al igual que toda tecnología, iniciaron de forma muy simple. El primer videojuego se puede rastrear a 1958 cuando William Higginbotham, un físico del laboratorio nacional de Brookhaven mostró al público *Tennis for Two*, el primer medio visual interactivo creado únicamente con propósitos de entretenimiento [4]; este se podía visualizar en una pantalla de un osciloscopio conectado a un computador analógico y se jugaba con controles especialmente diseñados para ello. Poco más de una década después, el concepto de *Tennis for Two* alcanzaría fama mundial con el lanzamiento de *Pong* en máquinas Arcade en 1972 y en consolas domésticas 1975, ambas producidas por Atari [5].

Con el avance de la tecnología en las siguientes décadas, la potencia de las consolas domésticas incrementaría drásticamente y, por ende, los videojuegos se harían gradualmente más complejos. En el periodo comprendido entre 1985 y 2005 surgirían grandes hitos de la historia de los videojuegos, muchos de los cuales iniciarían franquicias multimillonarias, en esta lista se puede encontrar a *Super Mario Bros. (1985), The Legend of Zelda (1986), Street Fighter (1987), Final Fantasy (1987), Sonic the Hedgehog (1991), Pokemon Red/Blue (1996), Resident Evil (1996), Grand Theft Auto (1997), Super Smash Bros. (1999), Halo: Combat Evolved (2001), Call of Duty (2003), Monster Hunter (2004),* etc. [6] Si se echa un vistazo a estos títulos, será evidente lo diferentes que son unos de otros; esto se debe a que el avance de la tecnología no solo permitió hacer mejores videojuegos que antes, sino hacer videojuegos que no eran posibles antes. En la década de los 80s era una gran hazaña mostrar un personaje y varios obstáculos en un nivel que se extendía más allá de una sola pantalla, a inicios de los 90s se desafiaron los límites de la tecnología con juegos que daban la apariencia de ser 3D [7] y desde mediados de los 2000 hasta la actualidad los videojuegos han demostrado año tras año que se pueden crear vastos mundos con gráficos extraordinariamente realistas en los cuales todo tipo de escenarios y jugabilidad pueden tomar lugar. La industria no muestra señales de detenerse, por lo que solo queda esperar los títulos y avances que vendrán en el futuro.

Videojuegos de Plataformas 2D

Debido a la diversidad de contenido que existe hoy en día en los videojuegos, estos se clasifican en varios géneros dependiendo de su estilo, sus gráficos y sus mecánicas; los géneros frecuentemente determinarán la dificultad del juego y su audiencia.

Entre los juegos que se mantienen populares actualmente están aquellos que utilizan la misma fórmula familiar para los jugadores de antaño: un personaje cuyo movimiento se limita a un plano y que debe atravesar varios enemigos y obstáculos para llegar al final de un nivel, este género se denomina videojuego de plataformas 2D [8] y es representado hasta la actualidad por franquicias como *Super Mario, Donkey Kong Country* y destacados modernos como *Rayman Legends* y *Ori and the Will of The Wisps;* esto deja claro que este género aún es disfrutado y demandado pese a que la tecnología desde hace mucho ha permitido hacer juegos mucho más complejos [9].

Los videojuegos de plataformas 2D suelen ser preferidos por jugadores casuales o primerizos, quienes suelen tener problemas controlando las muchas más acciones disponibles en un juego en 3D, de igual manera, los desarrolladores que inician en la industria suelen empezar desarrollando un juego en 2D ya que puede ser más fácil que un

juego en 3D, pero esto depende de la calidad y complejidad que se busque [10]. Exceptuando ciertos casos, un videojuego de plataformas 2D será fácil y por lo tanto podrá ser disfrutado por niños y adultos mayores, quienes, contrario a la creencia popular, también recurren a los video juegos como medio de entretenimiento y relajación [11].

2 METODOLOGÍA

2.1 Metodología de Desarrollo SUM

Como bien se sabe, las metodologías ágiles han ganado terreno de forma muy acelerada en el desarrollo de software; esto se debe a que, a través de ellas, se puede implementar un sistema de entrega continua con intervención del cliente que funciona muy bien con equipos de desarrollo pequeños y proyectos que pueden cambiar durante su desarrollo, aunque no sea totalmente escalable [12].

De las metodologías ágiles se puede destacar Scrum, esta es la metodología más utilizada; los resultados de una encuesta realizada por la *Scrum Alliance* indicaron que Scrum se usa en más de la mitad de los proyectos de una organización, la tasa de éxito de proyectos realizados con Scrum es del 62% y el 87% de los equipos piensan que Scrum ha mejorado su trabajo [13].

Al hablar del desarrollo de videojuegos, una rama muy particular del desarrollo de software, la historia es otra: Una pequeña encuesta del 2010, reveló que la mayoría de los desarrolladores de videojuegos pensaron que la implementación de Scrum no trajo ningún beneficio y, en algunos casos, incluso afecto negativamente su proceso de desarrollo [14]. Es posible que en la década que ha pasado desde entonces el panorama y las opiniones hayan cambiado, pero los resultados de la encuesta dejaron claro que el desarrollo de videojuegos tiene sus propios desafíos y necesidades que, en la mayoría de casos, no se logran satisfacer con una metodología ágil ya establecida. Para solucionar esto mientras se mantienen los principios ágiles en el desarrollo de videojuegos, se creó la metodología SUM.

SUM es una metodología de desarrollo basada en el *Software & Systems Process Engineering Meta-Model 2.0* (SPEM 2.0), el cual es un modelo para definir los procesos de desarrollo y los componentes de sistemas y software. SPEM utiliza diagramas UML para definir los componentes que un sistema debe tener para estar completo de acuerdo con este modelo, pero proporciona mucha flexibilidad para que cada persona pueda adaptarlo;

SPEM fue propuesto por el *Object Management Group* (OMG), un consorcio de estándares de la industria de computación [15].

Debido a la flexibilidad proporcionada por SPEM 2.0, se pudo diseñar la metodología SUM para mantener los principios y características ágiles como son mejora continua de procesos, obtención de resultados predecibles, administración eficiente de recursos y riesgos, aumento de la productividad del equipo de Desarrollo, etc. SUM adapta estos principios al desarrollo de videojuegos y funciona mejor para proyectos pequeños de duración inferior a un año. [16]

Al igual que Scrum, SUM tiene un ciclo de vida para sus proyectos que tambien es iterativo con excepción de las etapas de inicio y cierre, además, estas etapas son propias del desarrollo de videojuegos por lo que reemplazan o varían con respecto a proyectos de desarrollo comunes. A su vez, si alguien no está satisfecho con la metodología SUM, puede añadir elementos o modificarla según sea el caso; pero, en general, esta metodología consta de las siguientes etapas [17]:

Concepto: Esta es la etapa inicial de todo desarrollo de videojuegos ya que en ella se plantea el tipo de juego que se va a desarrollar. Es importante que se abarque la mayor cantidad de información posible sobre el juego, esto incluye historia, ambientación, personajes, estilo, género y jugabilidad; esto último consiste en las mecánicas que tendrá el juego, sus reglas y las acciones que el jugador podrá realizar, todo esto es particularmente necesario en la fase de concepto porque permite al equipo de desarrollo comenzar a pensar en formas para implementar cada uno de estos elementos. Puede que sean necesarias varias reuniones e intercambios de ideas entre todos los involucrados en el proyecto antes de llegar a un acuerdo sobre el concepto del videojuego.

Planificación: Esta es la fase administrativa del proyecto donde se plantean todas las tareas que se deben realizar para implementar toda la funcionalidad del videojuego y completar el resto de las etapas del desarrollo del proyecto; teniendo una lista de tareas por realizar, se las agrupa en iteraciones, se las asigna a miembros del equipo de desarrollo y se elabora un cronograma para poder mantener una entrega continua como cualquier metodología ágil describe. Las iteraciones y el cronograma son flexibles, es decir, se pueden modificar según lo requiera cada fase del desarrollo o para adaptarse a potenciales cambios en el proyecto.

Elaboración: En esta fase se tiene como objetivo final desarrollar una versión estable del videojuego, para ello se sigue cada iteración del cronograma y, en cada una, se plantean objetivos a cumplir, métricas a utilizar y tareas a implementar; el desarrollo de todas estas

tareas se realiza en el tiempo que dure la iteración, a la vez que, se mantiene un seguimiento continuo para asegurarse que lo que se va logrando coincida con los objetivos del proyecto. Al final de cada iteración, se evalúa lo desarrollado, se analiza si se cumplieron los objetivos y se examina la situación del proyecto; en base a esta situación, puede que se deban hacer cambios a próximas iteraciones o al proyecto como tal y se deban tomar decisiones para continuar con el desarrollo. Toda la fase de elaboración también contribuye al crecimiento del equipo de desarrollo y le permitirá ganar experiencia para incrementar su productividad y adaptación para proyectos futuros.

Beta: Esta fase corresponde al lanzamiento de versiones preliminares del videojuego. Una vez terminadas todas las iteraciones de la fase de elaboración. Se tendrá un producto jugable sobre el cual se podrán realizar todo tipo de pruebas para verificar su calidad y si concuerda con el objetivo final que se tenía para todo el proyecto. Dependiendo del alcance del proyecto, la versión beta puede ser evaluada por el equipo de desarrollo, el resto de los involucrados o un grupo de personas designado para esto (*testers)*; estas pruebas tienen como finalidad hallar errores en el juego y examinar factores como dificultad, curva de aprendizaje, funcionamiento de las mecánicas, etc. Si se tienen que arreglar errores o realizar cambios, se realizará un nuevo lanzamiento beta luego de haber hecho los arreglos necesarios y las pruebas se repetirán; proyectos con largo alcance incluso pueden elegir distribuir versiones beta con todos o algunos de sus usuarios finales. La fase beta terminará cuando el videojuego desarrollado cumpla con los objetivos y estándares de calidad planteados y se pueda jugar sin que presente algún error.

Cierre: Esta es la fase final del proyecto, donde se realiza el lanzamiento de la versión final del videojuego. Esta fase también sirve para evaluar todo el desarrollo del proyecto, los objetivos logrados, problemas encontrados, soluciones aplicadas y conclusiones obtenidas. Por último, se hace una retrospectiva que permitirá mejorar la aplicación de la metodología y el desempeño del equipo en proyectos futuros.

Gestión de Riesgos: Esta fase no tiene posicionamiento fijo en el proceso de desarrollo, sino que consiste en el siempre presente seguimiento y control de los potenciales riesgos que se pueden dar en cualquier parte del proyecto y de las etapas anteriores. El estar conscientes de que estos riesgos pueden ocurrir en cualquier momento implica el establecimiento de planes de contingencia, los cuales se deben realizar de acuerdo con la probabilidad y la gravedad de cada riesgo. La implementación adecuada de una gestión de riesgos permitirá disminuir las repercusiones que estos riesgos pueden tener en el proyecto.

La Fig. 1 muestra un esquema de todas las etapas del ciclo de vida anteriormente mencionadas.

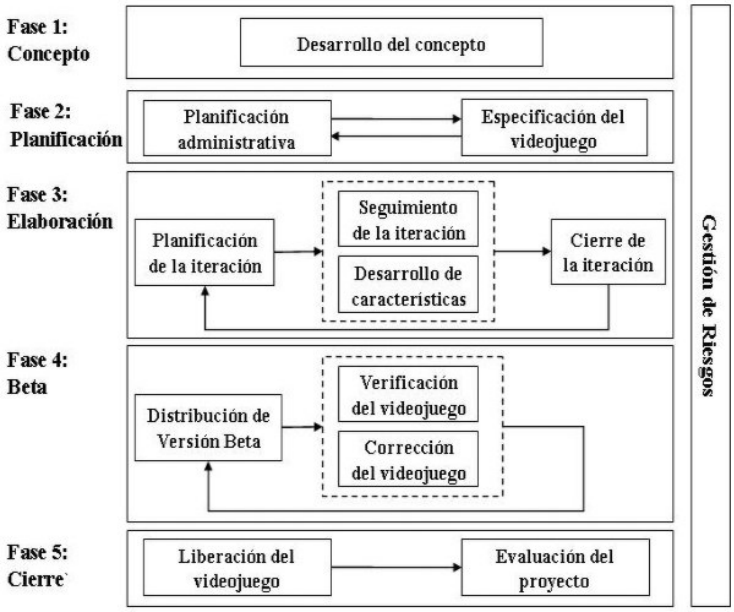

Fig. 1: Ciclo de vida de proyectos SUM

Roles

Como se encuentra descrito en su recurso oficial [18], La metodología SUM define cuatro roles para sus proyectos, tres de ellos son consistentes con los roles presentes en Scrum, pero tienen sus peculiaridades que les permiten funcionar específicamente para el desarrollo de un videojuego. Estos roles se explican a continuación:

Cliente: El rol de cliente en SUM es equivalente al rol de *product owner* en Scrum y, como tal, posee las responsabilidades de administrar el proyecto, monitorear su progreso y servir como vínculo con los *stakeholders* [19]. En proyectos de software comunes, la persona u organización que ordenó el proyecto y especificó los requerimientos de este, es considerado cliente; pero los videojuegos se desarrollan para un grupo de consumidores, de forma similar al software comercialmente disponible, ya que es imposible mantener una relación de monitoreo directa con millones de personas, el cliente de SUM debe cumplir ese rol de líder del proyecto y asegurarse que cada parte del desarrollo vaya de acuerdo con la visión establecida para obtener un resultado que pueda satisfacer a los usuarios y a los responsables financieros del videojuego.

Productor Interno: rol correspondiente con el *Scrum master*, se encarga de verificar el desarrollo del proyecto desde su interior, asegurándose de solucionar todos los problemas que se presenten y de mantener organizado al equipo de desarrollo para garantizar el

cumplimiento de los objetivos planteados. Además, es el intermediario entre el equipo de desarrollo y el cliente, por lo que debe comunicar a este último las dificultades o eventos inesperados que se están presentando en el proyecto para que se pueda tomar decisiones o realizar cambios de ser necesario; de igual forma, debe mantener al equipo de desarrollo al tanto de las decisiones administrativas tomadas y adaptar el desarrollo en torno a ellas.

Equipo de Desarrollo: En SUM, debido al carácter multidisciplinario del desarrollo de un videojuego, el equipo de desarrollo se subdivide en roles específicos que pueden consistir en una o dos personas dependiendo del tamaño del equipo, cada uno de estos roles abarca una parte del desarrollo del videojuego por separado, pero todos los miembros del equipo deben mantenerse en constante comunicación para asegurarse que todas las partes del producto desarrollado concuerden. Los roles definidos por SUM dentro del equipo de desarrollo son: el diseñador de juego, quien se encarga de diseñar todos los elementos incluyendo jugabilidad, personajes, historia y ambientación para que el videojuego se adapte a la visión que se tiene y pueda complacer a los usuarios; el programador es quien se encarga de codificar todas las características que permiten que el videojuego funcione como se espera, esto incluye verificar y corregir errores técnicos según se presentan; el artista sonoro, quien está a cargo de la música y los efectos de sonido que le dan vida al videojuego; el artista gráfico, finalmente, es quien se encarga de dibujar o modelar los personajes y escenarios conceptualizados, animarlos y asegurarse que el juego tenga el nivel de detalle artístico para hacerlo disfrutable.

Verificador Beta: Quienes cumplen este rol se les conoce también como *testers*, ya que se encargan de probar las versiones preliminares del videojuego para encontrar errores y verificar el flujo de juego, curva de aprendizaje, dificultad, entre otros aspectos. Los hallazgos de un verificador beta deben ser reportados oportunamente para que se puedan realizar las correcciones necesarias al videojuego. Usualmente los verificadores beta son ajenos al equipo de desarrollo y tener conocimientos en cualquier área del desarrollo no es un requerimiento para desempeñar este rol.

La TABLA I muestra las personas cumplen los respectivos roles en este proyecto.

TABLA I Roles en el Proyecto

Rol	Encargado
Cliente	Mateo Borja
Productor Interno	Juan Pablo Zaldumbide
Equipo de Desarrollo	Mateo Borja
Verificador Beta	Mateo Borja

Artefactos

La metodología SUM no es estricta con respecto a los artefactos que se elaboran al momento de planificar y desarrollar el proyecto, por lo tanto, cada equipo adapta la documentación según sus necesidades lo requieran, siendo común mezclarla con los artefactos de Scrum en menor o mayor medida [20]. Los artefactos producidos en este proyecto se mencionan a continuación.

Concepto del Juego: Este documento detalla en alto nivel todos los aspectos del videojuego: La historia, la jugabilidad, los personajes, la ambientación, etc. Dependiendo de quien vaya a ver este documento, se puede añadir también información administrativa como modelo de negocio, plan de marketing o prototipos de personajes y arte para orientar a los artistas gráficos [21]. El objetivo principal del concepto de juego es informar y orientar a todos los involucrados sobre el proyecto que se va a desarrollar.

Historias de Usuario: En el desarrollo de un videojuego, elaborar historias de usuario detalladas como se lo hace en proyectos de software no es conveniente, pues varias historias corresponderían a acciones ligadas entre sí o muy similares; por este motivo resulta mejor elaborar un número pequeño de historias que abarcan una característica principal sin subdividirla en características específicas, bajo esta definición, estas historias son épicas [22].

La TABLA II muestra una de las historias de usuario de este proyecto.

TABLA II Historia de Usuario No. 1

HISTORIA DE USUARIO	
Identificador (ID): HU001	**Usuario:** Jugador
Nombre Historia: Ejecutar Juego	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Bajo
Iteración Asignada: 4	
Responsable: Mateo Borja	
Descripción: El jugador podrá ejecutar el video juego en su PC de sistema Windows haciendo click en el archivo ejecutable, el cual será proporcionado como parte de este proyecto.	
Observaciones: El archivo ejecutable estará ubicado en la carpeta descomprimida en la cual se distribuye el juego. Se contempla que el juego pueda funcionar todos los computadores Windows de 64 bits.	

Plan de Proyecto: El formato utilizado para este artefacto puede variar, asemejándose al *product backlog* de Scrum. En este caso, se lo elaboró como una lista de tareas de alto

nivel que muestra las iteraciones, historias y nivel de prioridad que corresponden a cada una de ellas.

La TABLA III muestra parte del plan del proyecto elaborado.

TABLA III Formato del Plan de Proyecto Utilizado

Plan de Proyecto				
HU-ID	**Actividad**	**Iteración**	**Estado**	**Prioridad**
TODAS	Conceptualizar juego	1	Finalizada	Alta
N/A	Diseñar elementos	1	Finalizada	Media
HU004, HU007	Animar elementos	1	Finalizada	Media
HU004, HU007	Codificar elementos	2	Finalizada	Alta
N/A	Construir niveles	2	Finalizada	Alta

Sprint Backlog: Según Scrum, el *sprint backlog* es usado por el equipo de desarrollo para identificar las tareas del *product backlog* que deben ser realizadas en cada iteración [23]. En este proyecto, el *sprint backlog* contiene una lista específica de las tareas a realizar en cada iteración, junto con las historias de usuario asociadas con ellas y un estimado de tiempo de cuanto tardó cada iteración. Debido a la naturaleza no específica de las historias épicas, existen tareas en el *sprint backlog* que corresponden a varias épicas y otras tareas que no corresponden a ninguna.

La TABLA IV muestra el formato utilizado con el primer sprint del proyecto.

TABLA IV Sprint Backlog de la Iteración No. 1

Sprint Backlog					
SB-ID	**Nombre**	**HU-ID**	**Historia de Usuario**	**Tareas**	**Tiempo Estimado**
SB001	Conceptualización, Configuración y Diseño	N/A	N/A	• Configurar el ambiente de desarrollo • Instalar Extensiones y complementos • Autoeducación en el uso de Unreal Engine	
		Todas	Todas	• Conceptualizar Juego	
		N/A	N/A	• Diseñar personajes, enemigos, escenarios, obstáculos	

				• Obtener música y sonidos	
		HU004, HU007	Realizar acciones en el juego, recibir daño/ morir	• Animar acciones de personaje principal • Animar personajes adicionales • Animar enemigos	

2.2 Diseño de interfaces (*mockups*)

El diseño visual de una aplicación puede darse en diferentes niveles de detalle dependiendo del tipo de aplicación que se esté desarrollando. Si se diseña una página estática sin mucha funcionalidad, bastará un *Wireframe* simple, contrario a esto, una aplicación multimedia de múltiples páginas se beneficiará de un prototipo similar al producto final; en cualquier caso, tener un *mockup* sirve para que el equipo de desarrollo tenga una visión de lo que se va a desarrollar y pueda orientar el producto final en base a ella [24].

Un videojuego es una aplicación que depende mucho de su parte visual, por lo que su apariencia se discute al momento de conceptualizarlo como se mencionó en la sección anterior. Es importante que el o los diseñadores y artistas tengan una idea de cómo se verá el videojuego, por lo que es común producir diseños preliminares de personajes y escenarios, los cuales pueden ser en papel, digitales y con varios niveles de detalle. Estos diseños se denominan arte del concepto [25]

Para este proyecto se hicieron dibujos en papel del personaje principal, así como de la estructura de los niveles, los dibujos de personajes sirvieron como base para luego diseñar los *sprites* de los mismos tal y como se ven en el juego final. Los conceptos de niveles se digitalizaron y se utilizaron para construir la versión final de los niveles del juego.

Herramienta utilizada para el diseño

Herramientas conocidas de diseño como *Adobe Photoshop* e *Illustrator* se pueden utilizar tanto para el arte preliminar como para el diseño de componentes finales. Para este proyecto se optó por utilizar herramientas ligeras y de uso libre: Los *sprites* y las animaciones de los personajes fueron realizadas con la aplicación web *Piskel*, el concepto digital de niveles fue hecho en *Microsoft Paint* y el diseño de botones y pantallas fue realizado con el editor web de imágenes *Pixlr* en conjunto con herramientas para archivos png.

Diseño de Personajes

Los personajes en 2D se diseñan por medio de Sprites, elementos no estáticos independientes del fondo del juego [26], y se animan reproduciendo cuadros en sucesión, el elemento que posee todos los cuadros de una animación y que es necesario para implementarla en el juego se denomina *SpriteSheet.*

La Fig. 2 muestra el concepto del personaje principal junto con su *Sprite* final

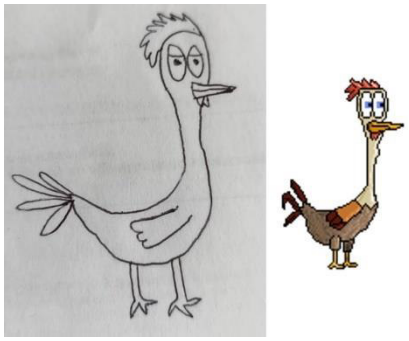

Fig. 2: *Ricken*: Concepto vs *Sprite*

Diseño de Niveles

En un juego de plataformas 2D, los niveles constituyen la totalidad del contenido jugable. En la mayoría de los casos, el juego consistirá en recorrer cada uno de los niveles de inicio a fin lidiando con obstáculos y enemigos; queda a criterio del desarrollador la complejidad de cada nivel y la adición o ausencia de rutas alternativas y condiciones adicionales para terminar un nivel, pero se debe hallar un balance que haga cada nivel disfrutable para la audiencia objetivo en términos de dificultad, longitud y características. En lo que respecta a apariencia, el estilo gráfico de los niveles suele concordar con el estilo de los personajes y el resto del juego, este estilo se discute en el concepto y tiene un gran impacto en la calidad del juego.

Tener un concepto de los niveles agilita significativamente la construcción de los mismos, este concepto puede ser un dibujo de la totalidad o partes de cada nivel con diferentes grados de fidelidad. Los niveles pueden ser una réplica de su concepto o pueden sufrir cambios con respecto a este.

La Fig. 3 muestra el concepto del primer nivel dibujado digitalmente con baja fidelidad.

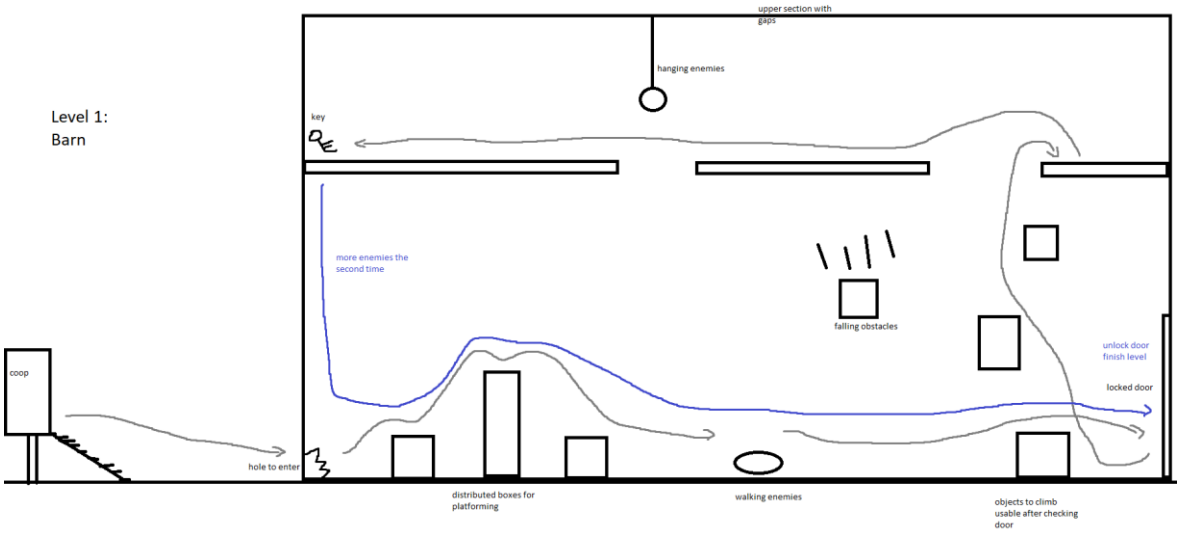

Fig. 3: Concepto del primer nivel

Diseño de Pantallas

La pantalla de título de un juego junto con su menú principal y cualquier otro menú o pantalla que exista también son parte de la interfaz gráfica, como tal, se deben diseñar de forma que cumplan su función y se vean llamativas en concordancia con el resto del juego. La complejidad artística de una pantalla de título queda a criterio del desarrollador, es común ver pantallas que contengan música exclusiva, fondos y personajes animados o incluso muestras de lo que será la jugabilidad del juego. Una pantalla de título bien hecha puede contribuir a dar una buena impresión en los jugadores y lograr que el juego se vuelva icónico [27].

La Fig. 4 muestra el diseño usado en la pantalla de título del presente proyecto.

Fig. 4: Pantalla de título sin sus botones

2.3 Diseño de la arquitectura

Este apartado está destinado a exponer la arquitectura con la que se trabajó en el proyecto, la cual en este caso corresponde a la arquitectura de Unreal Engine y los videojuegos desarrollados con él. [28]

Componentes de Unreal Engine

Unreal Engine es un motor gráfico de código abierto desarrollado por *Epic Games*, pero la palabra motor gráfico no hace justicia a todos los componentes que posee esta herramienta y que intervienen en todos los aspectos del funcionamiento de un videojuego. A continuación, se mencionan aquellos componentes relevantes para este proyecto

Motor de sonido: Componente que trabaja en segundo plano a las acciones, pero cumple la importante función de agregar música y sonido al juego. Las funciones proporcionadas por este motor permiten añadir varias capas de sonido que siempre se están reproduciendo o que dependen de la jugabilidad.

Motor de física: Unreal Engine utiliza el motor *PhysX* de NVIDIA para constantemente realizar los cálculos que permiten a los objetos del juego obedecer las leyes físicas del mundo real. Este motor ahorra mucho tiempo a los desarrolladores ya que implementa automáticamente en objetos comportamientos como gravedad, colisión y mecánica de fluidos.

Motor gráfico: Este término en español abarca todo lo que son herramientas como Unreal Engine, pero su definición exacta como componente se refiere al motor responsable de la salida de imágenes en la pantalla; la entrada para generar esta salida es información como colores, sombras, geometría, iluminación, texturas, etc. El motor realiza complejos cálculos constantemente para poder construir píxel por píxel la imagen mostrada.

Marco de jugabilidad: Todas las funciones del juego se construyen en este componente, entre estas se pueden mencionar el registro de HUD y procesamiento de *Input*, elementos que son indispensables ya que permiten jugar el juego como tal. Este componente tambien provee el sistema de clases usado para crear todos los objetos del juego.

De igual manera cabe mencionar que Unreal Engine cuenta con una inmensa cantidad de editores muy potentes para trabajar con las varias clases de objetos. Los que se usaron en este proyecto fueron: editor de *widgets,* editor de actores, editor de *tilesets,* editor de *tilemaps*, editor de *flipbooks*, editor de *soud cues* y extractor de *sprites.*

Sistema de clases de Unreal Engine

Unreal Engine implementa un extenso sistema jerárquico de clases, todos los objetos que se codifican en esta herramienta son instancias de clases para la funcionalidad específica de dicho tipo de objeto que heredan de clases más generales. Todos los objetos que forman parte de la jugabilidad se derivan de la clase base denominada “actor”; cada extensión de esta clase añade funcionalidades especiales que se siguen diversificando en subsecuentes herencias, obteniendo así la gran variedad de objetos disponibles para el juego.

Codificación gráfica y *Blueprints*

Una de las características más atractivas de Unreal Engine es su muy eficiente sistema de codificación visual llamado *blueprints*, estos son bloques de funcionalidad que se unen entre sí a modo de diagrama de flujo para codificar un objeto; las *blueprints* permiten a no programadores desarrollar un videojuego sin preocuparse por escribir código pero también se sienten familiares para programadores ya que con ellas se pueden manejar variables y elaborar funciones de similar forma a como se haría en código.

Estructura del proyecto

Unreal Engine posee plantillas que se pueden seleccionar al crear un nuevo proyecto, estás plantillas corresponden a géneros comunes de videojuegos y vienen con una gran cantidad de ajustes y funciones para comenzar a construir el juego; pese a la existencia de una plantilla de plataformas 2D, se decidió trabajar con un proyecto en blanco ya que de esta forma se tiene absoluto control sobre las clases que se crean, se sabe cuál es el origen de todo lo que pasa en el juego y se tiene la certeza de que nada está sucediendo que el desarrollador no haya codificado.

La Fig. 5 muestra la ventana del editor de Unreal Engine.

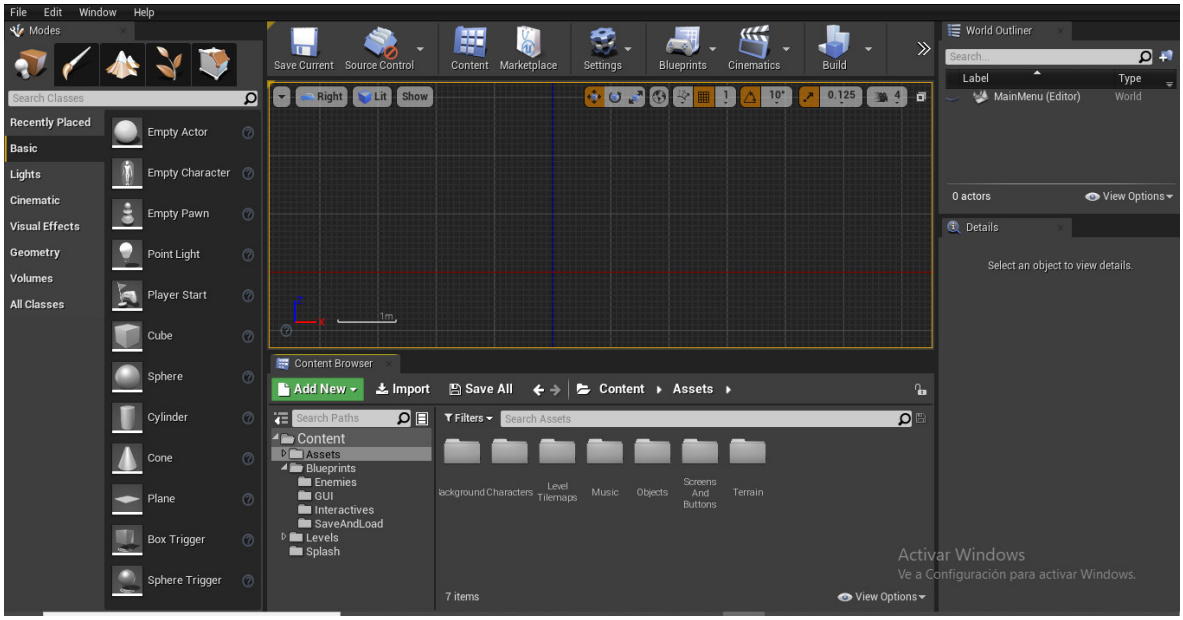

Fig. 5: Ventana de Unreal Engine

En esta ventana se pueden distinguir las siguientes partes:

Vista: Es el área central donde se visualiza el nivel en el que se está trabajando y permite arrastrar objetos y realizar diferentes acciones para modificar dicho nivel.

Explorador de contenido: Ubicado directamente bajo la vista, es la sección donde se puede acceder a todos los *assets* que se importaron desde el computador o se descargaron de la tienda de Unreal Engine. La estructura de este explorador depende únicamente del desarrollador, al cual le conviene ubicar sus *assets* en carpetas y categorías bien definidas para asegurarse de que siempre pueda encontrar un *asset* cuando lo necesita.

Barra de herramientas: Ubicada sobre la vista, es una lista horizontal de herramientas comunes como guardar, jugar, construir el proyecto, configuraciones, acceder a la tienda, abrir *blueprints*, entre otros. Esta barra se puede personalizar según las necesidades del usuario.

Descripción de la escena: Ubicada en el borde derecho de la ventana, muestra todos los objetos que se encuentran en el presente nivel y su tipo. Si se selecciona un objeto, se podrán ver sus propiedades en la ventana inmediatamente inferior.

Modos de trabajo: Ubicados en el borde izquierdo, presentan todas las opciones de trabajo como son creación de diversas clases y construcción de terreno. Esta es la única función del editor principal que no se utilizó en este proyecto.

2.4 Herramientas de desarrollo

La principal herramienta utilizada en el desarrollo de este proyecto fue Unreal Engine, al ser este un programa muy completo, no fue necesario recurrir a ninguna otra herramienta en lo que respecta a codificación, simulación y construcción. Cabe mencionar que se usaron herramientas externas para el diseño de los objetos o *assets* con los que trabaja Unreal Engine como son *sprites,* botones, *tilesets*, etc.

La TABLA V muestra cada herramienta utilizada en este proyecto con la justificación de porque se usó.

TABLA V Herramientas para el desarrollo del videojuego

Herramienta	Justificación
Unreal Engine	De entre los muchos motores gráficos existentes, Unreal Engine se ha posicionado entre los más populares debido a su potencia y amigables características. Los grandes desarrolladores lo eligen por los gráficos fotorrealistas que se pueden procesar, pero cualquier persona puede crear un juego simple para cualquier plataforma haciendo uso de su lenguaje visual (*Blueprints)*, su flexible licencia de uso y la cantidad de contenido listo para usar creado por la comunidad [29]. Este proyecto es la prueba de que se pueden aprovechar las características que este motor proporciona incluso en juegos pequeños.
Piskel	Esta es una aplicación de dibujo y diseño gratuita que se puede descargar o utilizar desde el navegador. Fue construida en JavaScript y permite crear todo tipo de arte con pixeles (*Pixelart)* incluyendo *Sprites* para videojuegos. [30] Pese a ser una herramienta portátil y ligera, es muy versátil; tiene muchas herramientas que permiten realizar diseños de alta calidad y un visor que permite visualizar la animación resultante de combinar todos los cuadros dibujados. En este proyecto se la utilizó para diseñar los *Sprites* de los personajes y algunos obstáculos.
Pixlr	Igual que Piskel, Pixlr se puede utilizar en línea o descargar, se trata de una herramienta de diseño y edición fotográfica que combina las funciones de *Photoshop* e *Illustrator*, permitiendo crear todo tipo de imágenes para cualquier fin. [31] Pixlr tiene un editor básico y uno avanzado disponibles de forma gratuita para todos. Los planes pagados brindan acceso a herramientas de edición con inteligencia artificial y bibliotecas de contenido que contienen millones de imágenes stock, stickers y objetos. En este proyecto se usó esta herramienta para diseñar el logo del juego, los botones de los menús y las pantallas de título, pausa y transición. De esta forma se demuestra que la versión gratuita es suficiente para las necesidades del usuario promedio.

3 RESULTADOS

El desarrollo de este proyecto se dividió en iteraciones o *Sprints* como está especificado en la metodología SUM, a continuación, se describen las actividades realizadas en cada uno de estos *Sprints* en concordancia con el *Sprint Backlog* y los resultados obtenidos luego de realizarlas.

Sprint 1. Conceptualización, configuración y diseño

Para este proyecto se decidió juntar las tareas de configuración usualmente realizadas en el llamado *Sprint 0* con el resto de las tareas que conforman el primer *sprint*, de esta forma, se realizaron las siguientes tareas en esta iteración:

- Configurar el ambiente de desarrollo.
- Autoeducación en el uso de Unreal Engine.
- Conceptualizar el juego.
- Diseñar personajes, enemigos y obstáculos.
- Obtener escenarios, fondos y elementos adicionales.
- Obtener música.
- Animar acciones del personaje principal.
- Animar enemigos.

Configuración del ambiente de desarrollo

Unreal Engine es un producto de la compañía *Epic Games*, para instalarlo se debe descargar de su página principal un instalador para el *Launcher* de *Epic Games*, el cual es un programa que cumple la función de *hub* para acceder a todos los productos que ofrece esta compañía. Una vez instalado el *Launcher*, se pedirá iniciar sesión con una cuenta de *Epic Games*, por lo que se debe crear una si no se tiene; esto permitirá descargar e instalar la última versión del motor gráfico. Una vez instalado Unreal Engine, se lo puede abrir desde su acceso directo, pero se necesitará dirigirse al *Launcher* siempre que se lo necesite actualizar.

En la Fig. 6 se puede observar la versión de Unreal Engine que se utilizó para el proyecto instalada desde el *Launcher*.

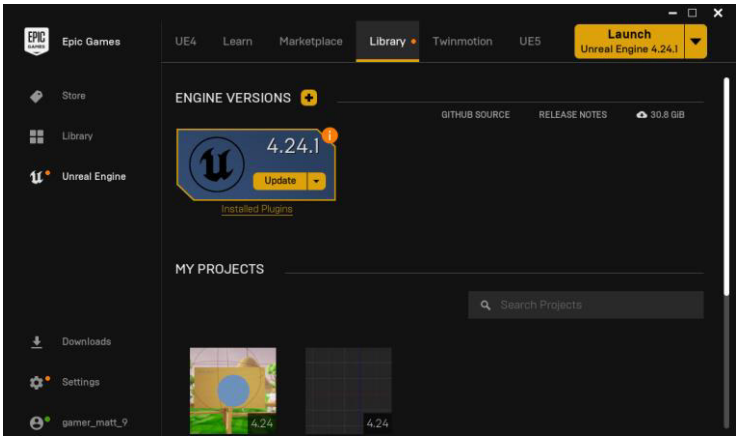

Fig. 6: *Launcher* de *Epic Games* con Unreal Engine instalado

El proyecto realizado no requirió instalar ninguna otra herramienta ni componentes adicionales para el motor gráfico ya que el resto de las herramientas utilizadas se encuentran disponibles como aplicaciones web.

Autoeducación en el uso de Unreal Engine

Al ser una herramienta tan popular, Unreal Engine posee abundantes recursos que permiten aprender a usarlo, desde documentación oficial hasta foros de la comunidad. También existen un sinnúmero de videos tutoriales realizados por terceros que abarcan todo aspecto de esta herramienta; esta última alternativa se eligió para la realización de este proyecto.

Los siguientes son los tutoriales que se siguieron de forma gradual para aprender a utilizar la herramienta:

- Unreal Engine 4 – Making a 2D Platformer in UE4 – Full Length [32]
- How To Setup Multiple Checkpoints And Respawn At Them – Unreal Engine 4 Tutorial [33]
- [UE4] 2D Platformer Tutorial! [34]
- UE4 – Level Music – Tutorial [35]
- Packaging And Exporting Your Game – Unreal Engine 4 Tutorial [36]

Conceptualización del juego

Conceptualizar el juego es lo primero que se debe hacer antes de empezar con su desarrollo, este proyecto no fue la excepción. El hecho de que el nivel de complejidad de un juego de plataformas es menor en comparación con otros géneros se traduce en un

concepto más simple, es decir, es aceptable tener historia, mecánicas y personajes sencillos. A continuación, se describen los principales elementos del concepto desarrollado.

Objetivo del juego: Recorrer cada uno de los niveles de inicio a fin lidiando con los enemigos y obstáculos que se presenten.

Historia del juego: En la granja del abuelo Joe, los huevos del gallinero desaparecieron sin previo aviso. En ausencia del gallo que lidera el gallinero, le corresponde a Ricken, un gallo joven sin características rescatables, buscar los huevos alrededor de la granja.

Personajes, enemigos y obstáculos: *Ricken the Chicken* es el protagonista del juego y el único personaje explícitamente mostrado. En su corto viaje encontrará criaturas comunes en una granja como son ratones, arañas y pájaros; estas sirven como enemigos que se deben derrotar en cada nivel. De igual manera, existen obstáculos como espinos y rocas rodantes que también constituyen un peligro para el protagonista y, por lo tanto, se deben evitar o destruir según sea el caso.

Niveles: El juego cuenta con 4 niveles en total, cada uno de ellos se ambienta en un escenario diferente e introduce nuevas amenazas y elementos que permiten que la experiencia del juego se sienta diversa y más desafiante conforme se avanza. Los cuatro niveles son: el granero, el establo, las praderas y la casa del abuelo Joe.

Mecánicas del juego: Un juego de plataformas 2D casi siempre tiene mecánicas sumamente sencillas y limita las acciones que el jugador puede realizar a las primordiales para terminar cada nivel. En el videojuego desarrollado, el jugador puede moverse lateralmente, saltar, acelerar tanto en tierra como durante un salto y atacar; también es posible pausar el juego y reaparecer en el último punto de control luego de recibir suficiente daño para agotar la salud del personaje (morir). Atacar y aterrizar sobre un enemigo son los únicos dos métodos de derrotarlo, mientras que el salto y la aceleración se deberán usar en conjunto para navegar de forma efectiva por el *layout* del nivel. En definitiva, depende del jugador como y cuando use las acciones disponibles, pero deberá manejarlas bien si desea terminar el juego.

Características adicionales: El juego progresará de forma lineal, cada nivel tendrá una pantalla que provee una breve descripción del mismo y se deberá completar para poder ir al siguiente nivel. El progreso del juego queda automáticamente guardado al finalizar cada nivel, de esta forma, el jugador puede volver al nivel en el que se quedó en caso de que no

termine el juego en una sola sesión, igualmente, puntos de control ubicados en los niveles aseguran que el jugador no tenga que repetir todo el nivel desde el inicio si llega a morir.

El concepto descrito en esta sección consiste en una síntesis de los elementos más importantes, referirse al documento de concepto ubicado en el Anexo II si se desea ver toda la información en su totalidad.

Diseño de personajes, enemigos y obstáculos

Como se mencionó en secciones anteriores, los personajes en el videojuego desarrollado fueron diseñados en forma de *sprites* utilizando la herramienta Piskel, esta herramienta tiene una interfaz sencilla diseñada para cualquier forma de *pixelart*, esta interfaz se muestra en la Fig. 7.

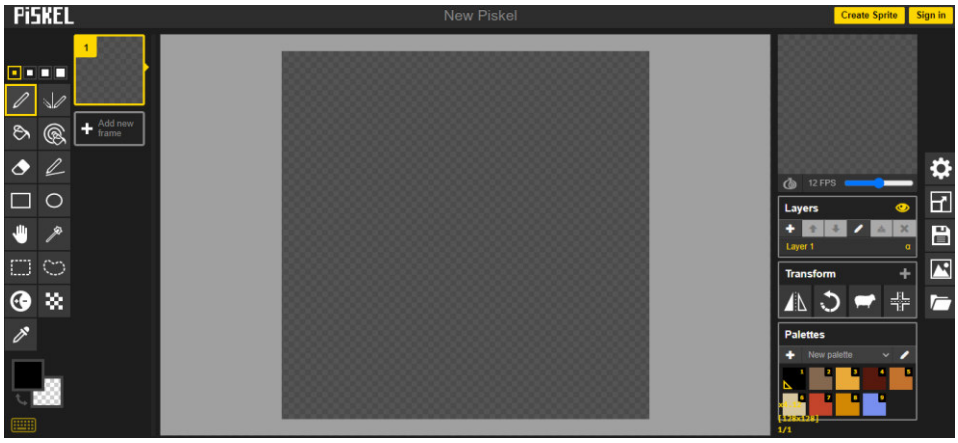

Fig. 7: Ventana de Piskel

Tomando como referencia la Fig. 7, se pueden distinguir los siguientes elementos de izquierda a derecha:

- Herramientas de dibujo comunes como lápiz, línea, selección, borrador, selección de colores, sombreado y tamaño de cursor.
- Lista de cuadros diseñados con opción para agregar, eliminar o duplicar.
- Área o lienzo de trabajo, se puede acercar o alejar según se necesite.
- Vista previa de la animación, lista de capas, herramientas de transformación y paleta de colores.
- Preferencias, cambio de tamaño del lienzo y opciones de exportación e importación.

Para este proyecto se decidió dibujar todos los personajes y obstáculos en un lienzo cuadrado de 128 pixeles. Fig. 8 muestra el Sprite finalizado del personaje principal en Piskel.

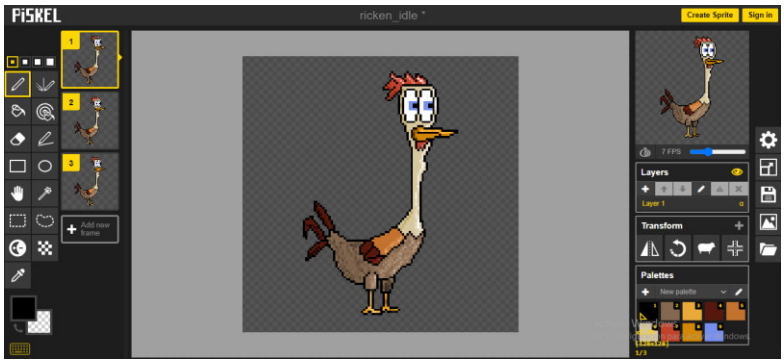

Fig. 8: *Ricken the Chicken* en Piskel

El proyecto final cuenta con 9 *Sprites* únicos que cumplen el rol de personajes, enemigos u obstáculos; referirse al Anexo II para visualizar cada uno de estos *sprites*.

Obtención de escenarios, fondos y elementos adicionales

El diseño de *sprites* y otros *assets* para un juego es una tarea que consume mucho tiempo y requiere cierto nivel de habilidades artísticas, debido a esto y a que el enfoque de este proyecto no era el arte de videojuegos, se decidió obtener de internet los *assets* que conforman el terreno, los fondos y los objetos de cada nivel. Cabe mencionar que Unreal Engine tiene su propia tienda de *assets* creados por la comunidad, pero lo necesario para este proyecto no se encontró aquí, por lo que se recurrió a otras tiendas de contenido donde se lograron encontrar suficientes *assets* gratuitos para crear el videojuego. La tienda de donde se obtuvieron la mayoría de *assets* es *Gamedev Market.* [37]

Todos los *assets* diseñados y obtenidos se deben importar a Unreal Engine desde su gestor de contenido para usarse en un proyecto, la Fig. 9 muestra los *assets* de este proyecto importados.

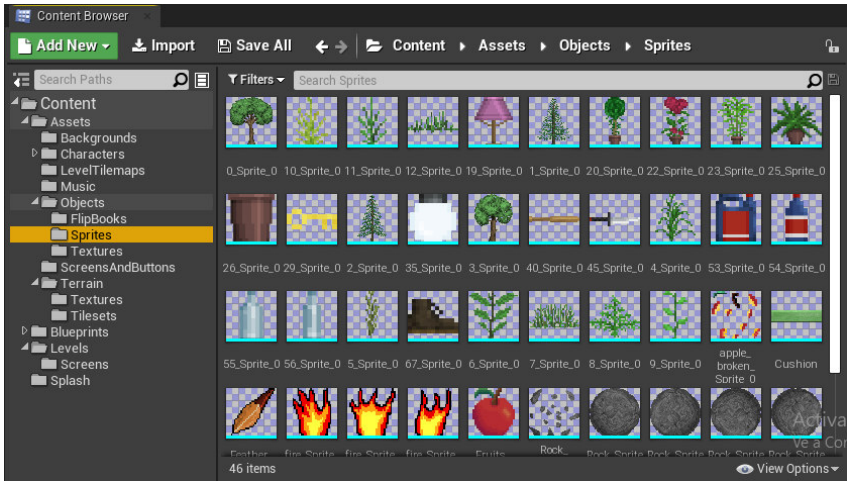

Fig. 9: Contenido importado en Unreal Engine

Referirse al Anexo II para visualizar imágenes de todos los *assets*.

Obtención de música

La música usada en todo el juego se descargó de internet por las mismas razones expuestas en el apartado anterior. En este caso, la búsqueda fue más tediosa debido a que convenía utilizar música sin derechos de autor para evitar potenciales problemas legales. Al final se decidió utilizar la música "*The old country farm"* compuesta por Darren Curtis, esta y el resto de sus composiciones se pueden encontrar en su portafolio en *SoundCloud.* [38]

Animación de acciones del personaje principal

La herramienta Piskel permite visualizar una sucesión de los *sprites* individuales realizados, de esta forma se pudieron crear *sprites* con variaciones en la "anatomía móvil" del personaje principal y verificar que la ilusión de movimiento que se genera al reproducir los cuadros en sucesión sea satisfactoria. El producto de realizar esto son las *spritesheets*, imágenes individuales que contienen todos los cuadros espaciados uniformemente.

Las animaciones de personajes 2D en Unreal Engine se realizan a través de *flipbooks*, estos son *assets* especiales que se generan luego de importar las *spritesheets* como texturas, extraer cada cuadro como *Sprite* y unir los cuadros de un mismo grupo para crear la animación. Este proceso también se puede hacer importando *sprites* por separado, pero el motor tiende a no reconocerlos como grupo y genera *flipbooks* estáticos con cada cuadro, dificultando el trabajo de animación.

En este proyecto se creó una animación para cada “estado” del personaje (saltando, corriendo, etc.), estas animaciones se ubicaron en *flipbooks* separados para que luego sean fáciles de integrar al código.

La Fig. 10 muestra las animaciones del personaje principal en Unreal Engine.

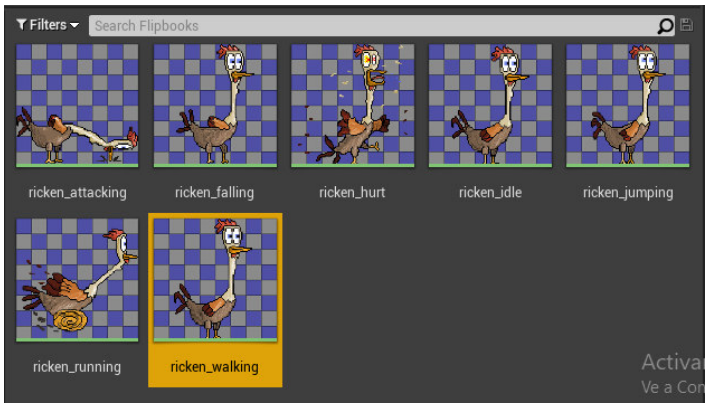

Fig. 10: *Flipbooks* del personaje principal

Animación de enemigos

El mismo principio y proceso descrito en el apartado anterior aplica para los enemigos, con la diferencia de que estos solo tienen una animación de movimiento al ser incapaces de realizar otras acciones y un cuadro estático utilizado cuando son derrotados.

Las animaciones de los enemigos en Unreal Engine se muestran en la Fig. 11.

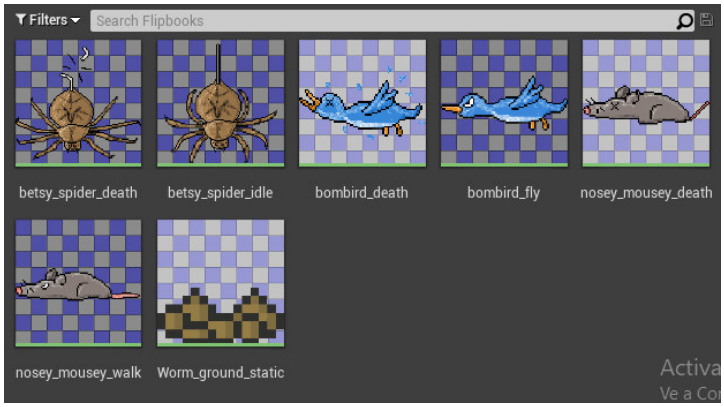

Fig. 11: *Flipbooks de enemigos*

Sprint 2. Codificación y construcción

Esta iteración incluye la implementación de todas las funciones esenciales para el videojuego, las cuales se codificaron en Blueprints o se construyeron utilizando los editores

específicos de *assets* que proporciona Unreal Engine. Las actividades específicas que se realizaron son:

- Codificar acciones de personaje principal
- Codificar enemigos
- Codificar interacciones entre personajes y enemigos / obstáculos
- Codificar daño y muerte de personaje principal
- Codificar muerte / desaparición de enemigos / obstáculos
- Construir *layout* y terreno de niveles
- Añadir enemigos, obstáculos y elementos a niveles
- Añadir fondos y música a niveles
- Diseñar e implementar HUD
- Calibrar interacciones en niveles
- Diseñar logo del juego
- Diseñar e implementar pantalla de título con menú de inicio
- Diseñar escenas y transiciones
- Añadir sonido a escenas y transiciones

Codificación de acciones de personaje principal

Las primeras dos clases que se crearon en este proyecto son la clase de modo de juego y la clase base del jugador, la primera no hace nada más que recibir la segunda como parámetro y ubicarse en las configuraciones del proyecto en lugar de la clase que viene por defecto, mientras que la segunda es la clase más grande del proyecto donde se codificó todo lo relacionado al jugador.

Por motivos de orden, se decidió especificar las teclas correspondientes a cada acción del personaje en las configuraciones generales del proyecto en lugar de en la clase del jugador, esto se traduce en eventos de cada tecla que se pueden llamar desde la clase del jugador para vincularlos con las acciones que se esperan de cada tecla. De esta forma, se construyeron los *Blueprints* para las acciones de movimiento lateral, saltar, correr y atacar; todos estos *Blueprints* se ubicaron aparte del resto de funciones del jugador para mantener el orden del proyecto y se muestran en la Fig. 12.

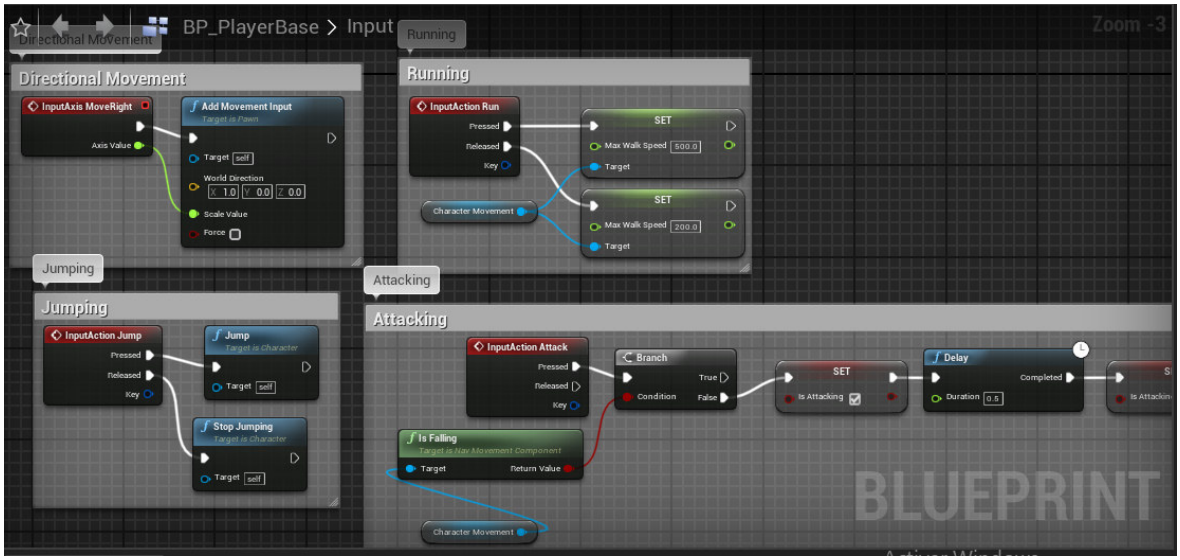

Fig. 12: *Blueprints* de acciones del jugador

Lograr que se reproduzcan las animaciones correspondientes a las acciones anteriores y otros estados del jugador requirió crear una enumeración con dichos estados y una función que cambia el *flipbook* del personaje si se cumplen las condiciones para cada uno de los estados. La Fig. 13 muestra las *blueprints* de esta función.

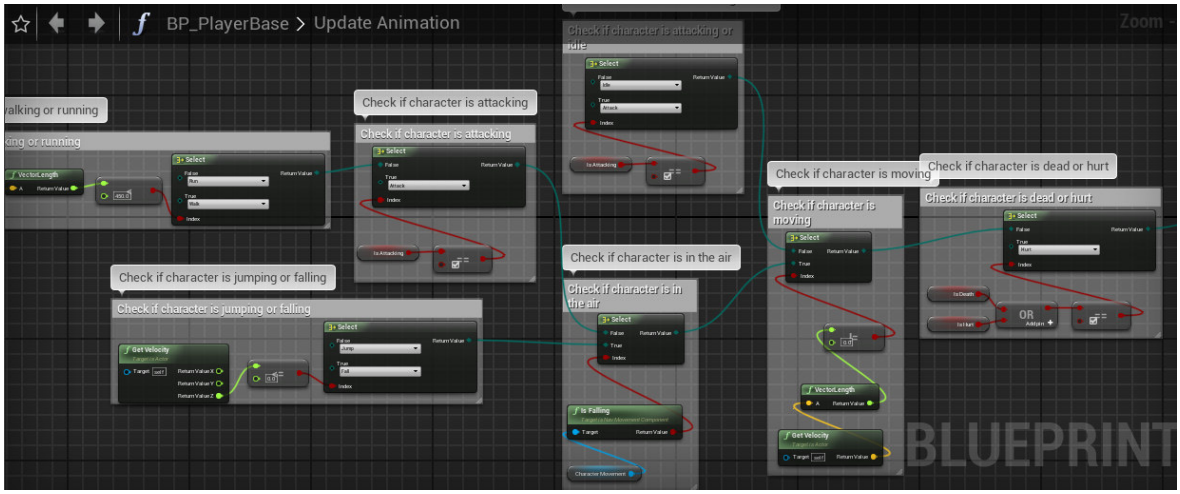

Fig. 13: *Blueprints* para actualizar la animación del personaje principal

Codificación de enemigos

Para los enemigos se comenzó creando una clase base con funcionalidad general de la cual se heredaron las clases para cada enemigo. Dentro de las clases de cada enemigo se comenzaron creado las *Blueprints para* su movimiento y cambio de dirección al impactar una pared u otro enemigo, esto es todo lo que realiza el enemigo más básico *nosey mousey*, para el resto de los enemigos se codificaron patrones de movimiento diferentes

como es el caso de *Betsy spider* (movimiento aéreo vertical) y acciones completamente únicas como arrojar manzanas en el caso de *bombird*.

Ciertas codificaciones como permitir a un enemigo volar o cambiar su velocidad fueron tan fáciles como alterar un valor en las configuraciones de clase mientras que otras fueron muy complejas y requirieron de interacción entre muchas clases y objetos para que funcionen correctamente, por este motivo, la codificación de los enemigos se realizó de forma gradual en el transcurso de todo el proyecto

La Fig. 14 muestra las *blueprints* básicas de movimiento de los enemigos.

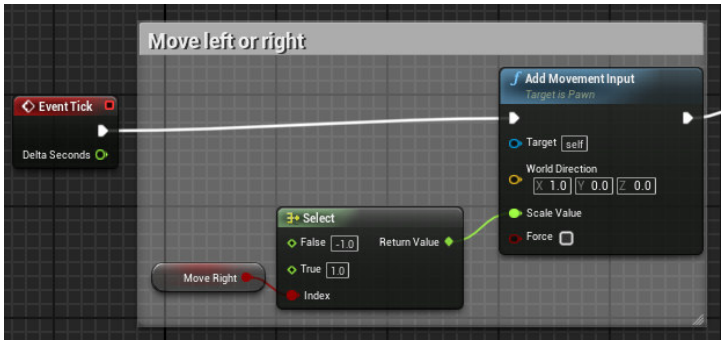

Fig. 14: Movimiento continuo para un enemigo

Codificación de interacciones entre personajes y enemigos / obstáculos

Debido a la naturaleza simple de un juego de plataformas 2D, las interacciones entre el jugador y los enemigos u obstáculos se suelen limitar a recibir o infligir daño, todos salvo uno de los enemigos de este videojuego siguen esta regla, por lo que la codificación se realiza mayormente en la clase del jugador haciendo referencia a la clase base de enemigos, el resultado final son *blueprints* en la clase del jugador que registran el ataque del jugador hacia el enemigo y *blueprints* en la clase base del enemigo que aplican daño cuando el jugador lo ataca, estos *blueprints* se muestran en la Fig. 15 y la Fig. 16.

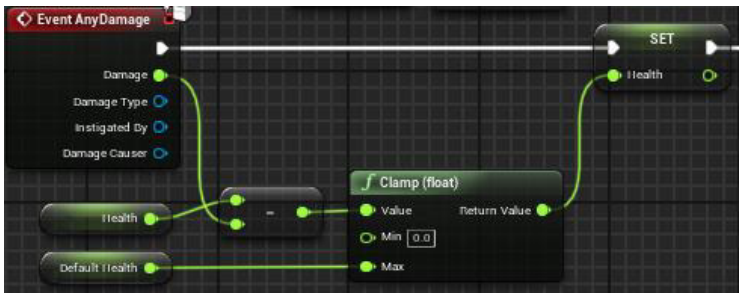

Fig. 15: *Blueprints* de daño hacia el enemigo en la clase base del enemigo

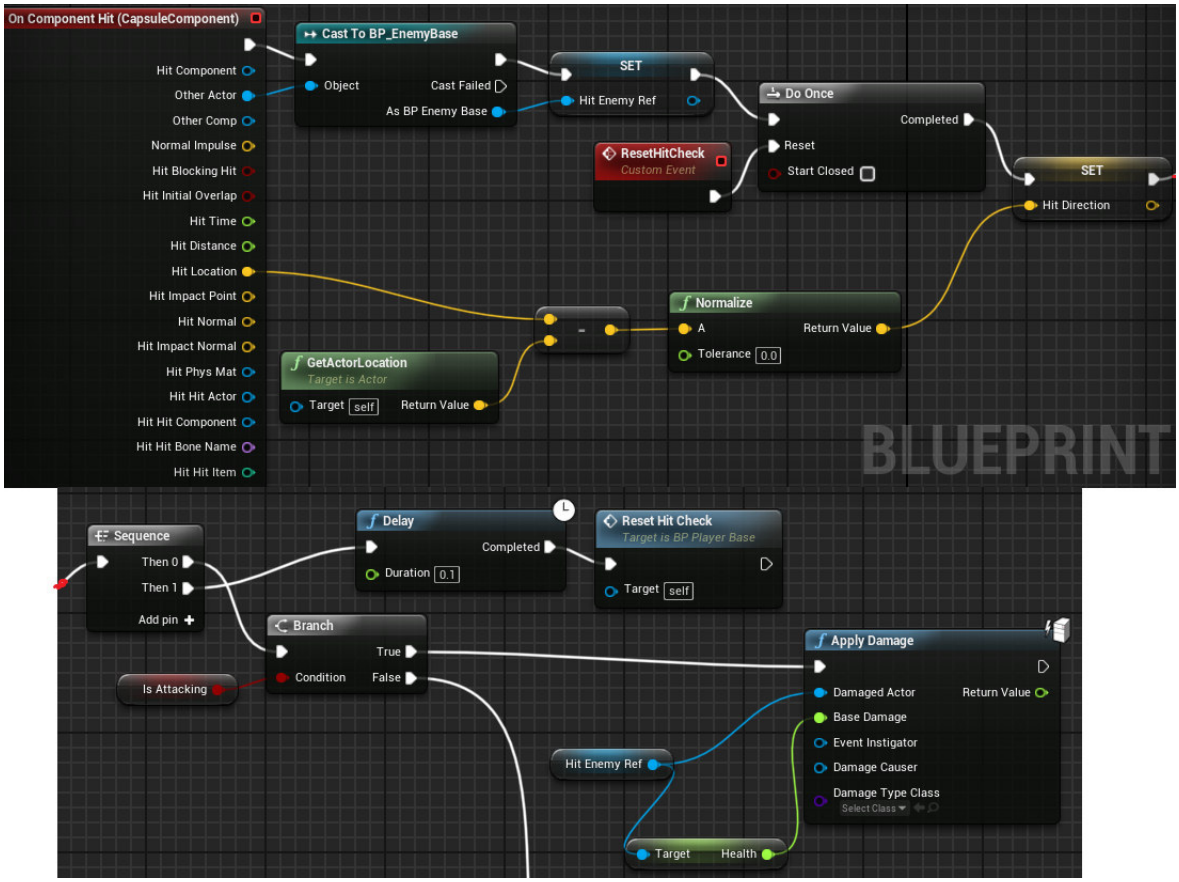

Fig. 16: *Blueprints* de daño hacia el enemigo en la clase del jugador

Los obstáculos heredan de su propia clase: la clase interactiva base, la cual contiene una función en blanco para interactuar con el jugador, esto permite que cada objeto heredado implemente su propia interacción. En el videojuego desarrollado existen los siguientes objetos que heredan de esta clase:

- Objetos afectados por la gravedad que caen sobre el jugador si este pasa por debajo.
- Espino de madera que lastima al jugador si lo toca y no puede ser destruido.
- Cojín que impulsa al jugador u otros enemigos.
- Pluma que recupera la salud del jugador
- Fuego que lastima al jugador y desaparece luego de poco tiempo.
- Llave que permite finalizar el primer nivel

No todos los objetos en la lista anterior son obstáculos peligrosos para el jugador, lo cual muestra la versatilidad que se consiguió al implementar esta clase padre.

La Fig. 17 muestra las *blueprints* de interacción del espino, el obstáculo más simple del juego.

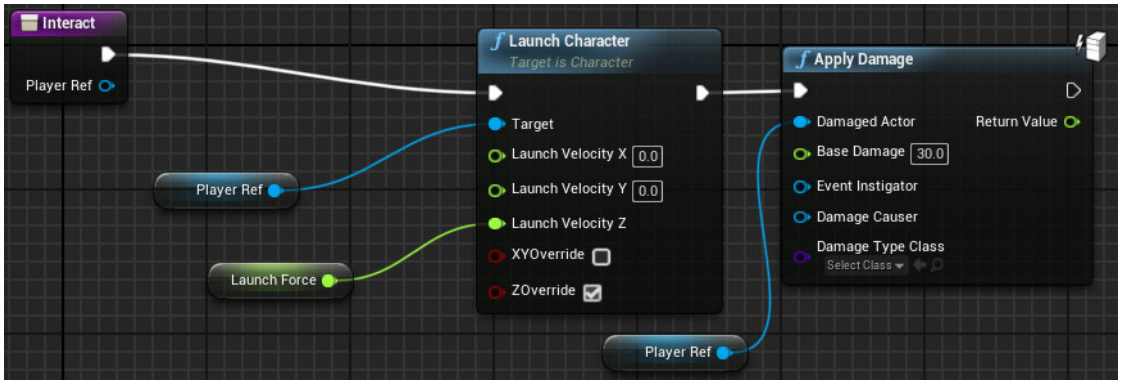

Fig. 17: *Blueprints* de interacción del espino de madera

Codificación del daño y muerte del personaje principal

El daño infligido al jugador o al enemigo son ambos resultados de la misma interacción bajo condiciones diferentes, por esta razón bastó con expandir las *blueprints* de daño hacia el enemigo con una condición para que el jugador también pueda recibir daño y agregar otras *blueprints* para registrar dicho daño en el jugador. La muerte, tanto de enemigos como el jugador, está ligada a la reducción total de su salud, por lo que se crearon funciones que efectúan la acción de muerte y solo se llaman si se verifica que la salud está en cero. La Fig. 18 muestra la función de muerte del jugador.

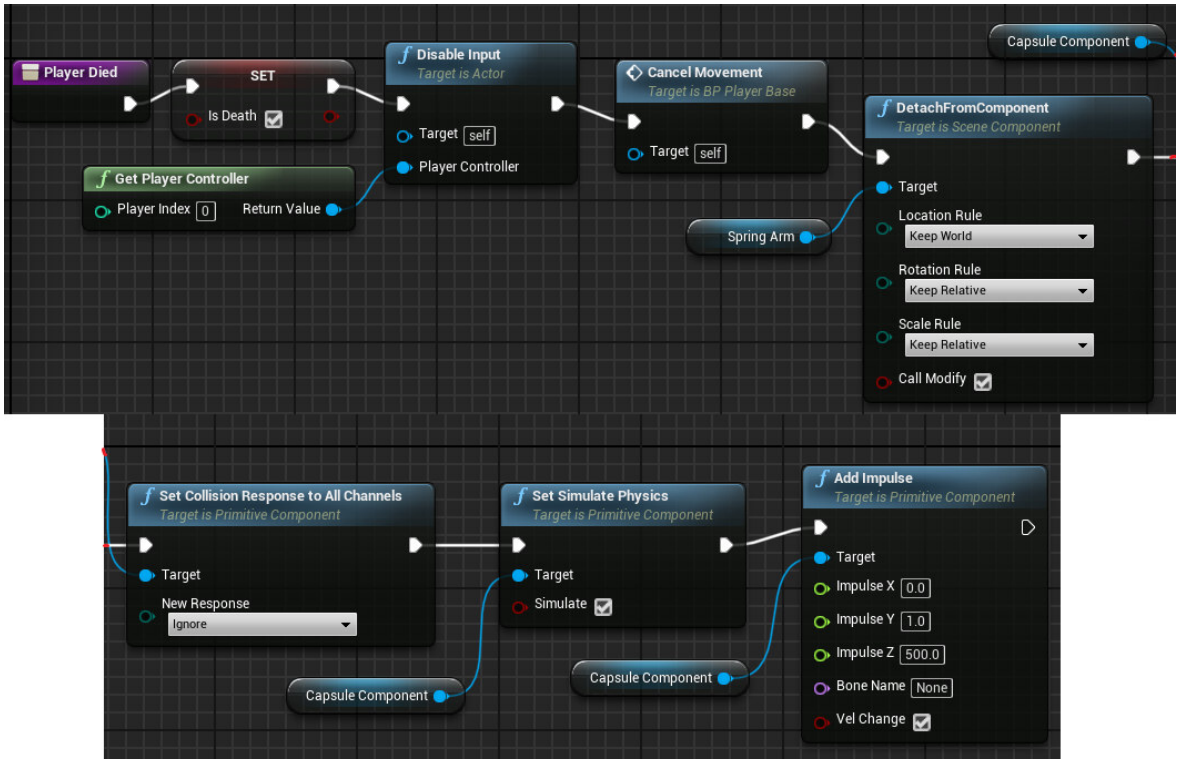

Fig. 18: *Blueprints* para la muerte del jugador

Si el jugador muere, debe poder reaparecer en el nivel para intentar de nuevo, esta funcionalidad se implementó como otra acción vinculada a una tecla que solo funciona si se verifica que el jugador ha muerto y le permite reaparecer al inicio del nivel; posteriormente en el desarrollo se implementaría la funcionalidad de puntos de control que requirió expandir la función de reaparición para verificar si el jugador ha pasado por un punto de control. La Fig. 19 muestra las *blueprints* completas de esta función.

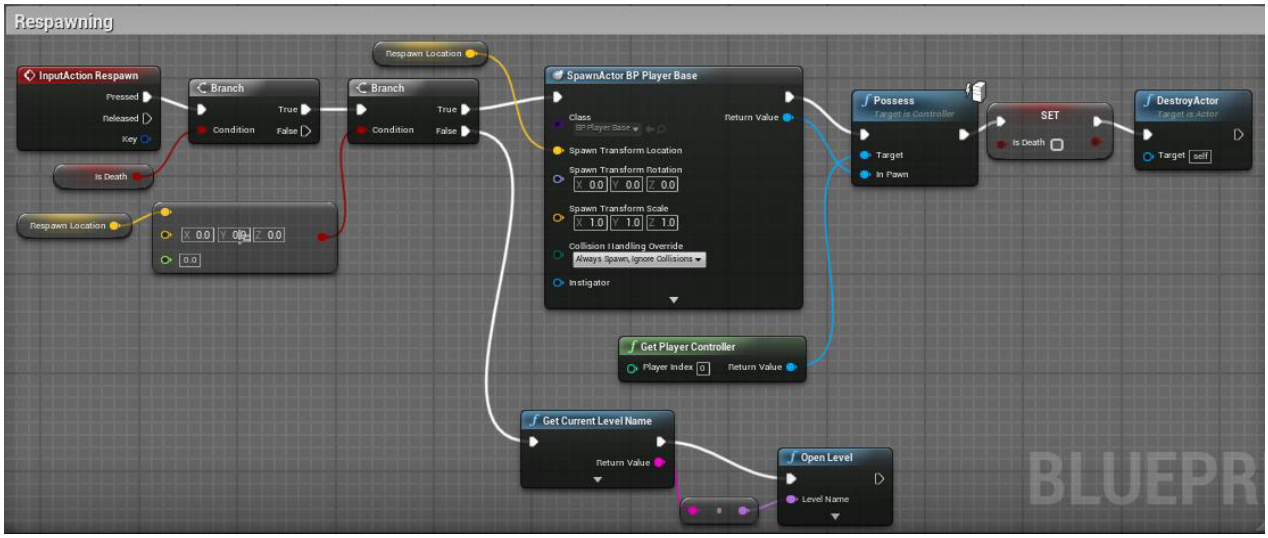

Fig. 19: *Blueprints* para revivir al jugador

Codificación de muerte / desaparición de enemigos / obstáculos

La lógica detrás de la muerte de los enemigos es similar a la del jugador y, al ser una función que afecta a todos los enemigos creados, se implementa en la clase base y se modifica en cada clase de enemigo para cambiar el *flipbook* correspondiente. Cada enemigo también fue codificado para reaparecer en su posición inicial luego de llevar un tiempo muerto con el objetivo de que el nivel nunca quede vacío si el jugador mata a todos los enemigos. La Fig. 20 muestra las *blueprints* que permiten esta reaparición.

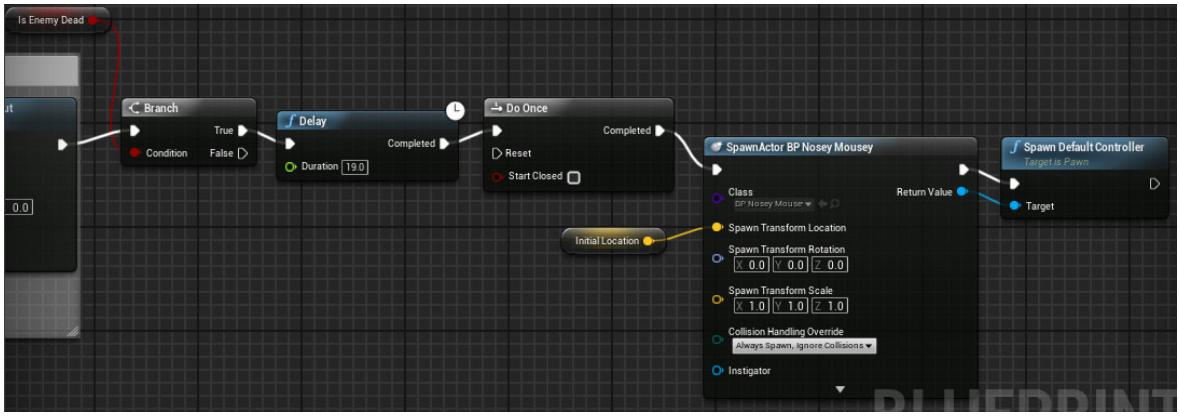

Fig. 20: *Blueprints* de reaparición de enemigos

En cuanto a los obstáculos y objetos, su desaparición está manejada dentro de sus individuales funciones de interacción, las cuales van desde no desaparecer nunca (espino de madera), hasta desaparecer luego de tocar al jugador o al terreno y no reaparecer. Luego está el caso especial de las rocas rodantes, obstáculos por definición, pero programados como enemigos que hacen daño al jugador, pueden recibir daño de este y morir únicamente si se las ataca; pero tambien desaparecen si chocan con una pared y no reaparecen por si solas, sino que actúan en conjunto con otro objeto que las genera ilimitadamente. Debido a todas estas características particulares las rocas son su propia clase y no heredan ni de la base de enemigos ni de la base interactiva.

Construir *layout* y terreno de niveles

Existen muchas maneras de generar terreno para los niveles en Unreal Engine, para este proyecto se utilizaron *tilesets* y *tilemaps*, opciones que se adaptan muy bien a juegos de plataformas 2D. Los *tilesets* son *assets* generados a partir de texturas igual que los *sprites*, por si solos no tienen utilidad, pero son los bloques base que sirven para construir *tilemaps*, los cuales son en sí el terreno de un nivel.

Unreal Engine abre un editor especializado para estos *assets* especiales, por lo que no es necesario crear *blueprints.* El editor de *tilesets* permite dividir la textura base en bloques de dimensión fija (en pixeles) a los cuales se les añade geometría de colisión, es decir formas poligonales que por defecto no permiten que los personajes atraviesen su perímetro; estas formas deben ser construidas según el uso que el *tileset* vaya a tener en el nivel, no tiene sentido añadir colisión a partes que los personajes nunca tocarán. La Fig. 21 muestra la edición de un *tileset*.

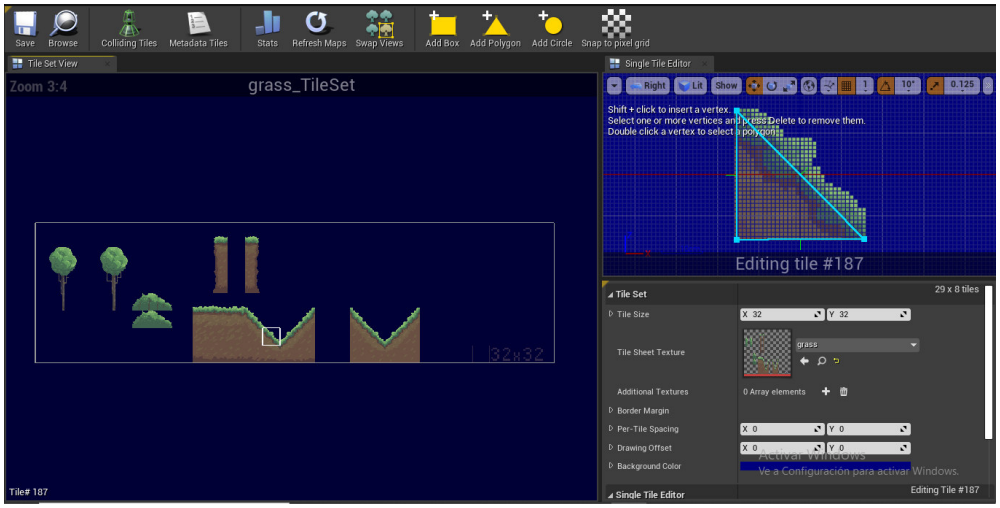

Fig. 21: *Tileset* con colisión en el editor de *tilesets*

El editor de *tilemaps*, por otro lado, es un lienzo de tamaño variable que se llena con los bloques formados en los *tilesets* para formar el *layout* de un nivel entero o trozos de terreno que se pueden agrupar para constituir dicho layout. Este editor permite seleccionar un *tileset* a la vez y llenar el *tilemap* bloque por bloque o con una selección de bloques; no cuenta con suficientes herramientas por lo que construir un *tilemap* es un trabajo muy tedioso y es necesario borrar cada bloque manualmente si existen fallas o se quiere modificar terreno ya existente. La Fig. 22 muestra un *tilemap* completo en el editor.

Fig. 22: *Tilemap* del primer nivel en el editor

Una vez creado un *tilemap,* se lo puede arrastrar al nivel creado, añadir un punto de inicio para el jugador y observar como el personaje puede recorrer todo el terreno diseñado. En este proyecto se realizaron *tilemaps* completos para cada nivel y se los ubicó en el mundo diseñado para cada uno de ellos.

Adición de enemigos, obstáculos y elementos a niveles

Luego de haber construido el terreno de un nivel se lo debe poblar con todos los elementos que permiten conseguir la experiencia de juego deseada, estos elementos son *sprites,* clases y *flipbooks*, los cuales se pueden arrastrar en la posición deseada para añadirlos al nivel; al ser arrastrados, los *flipbooks* aparecen como animaciones sin colisión o funcionalidad, los *sprites* generan por defecto colisión en un área cuadrangular que encierra todo su perímetro y las clases se convierten en objetos instanciados que tienen la funcionalidad que se programó en la clase. Cabe recalcar que Unreal Engine sigue tratando al nivel en 3D pese a que todos sus elementos son en 2D, por este motivo se debe prestar atención a la profundidad en la que se ubica un objeto al arrastrarlo, de lo contrario este objeto se verá, pero nunca será posible interactuar con él debido a que se encuentra en un plano diferente al resto de elementos.

En este proyecto se arrastraron una combinación de los 3 elementos antes mencionados para poblar cada nivel, la Fig. 23 muestra el primer nivel con todos sus elementos.

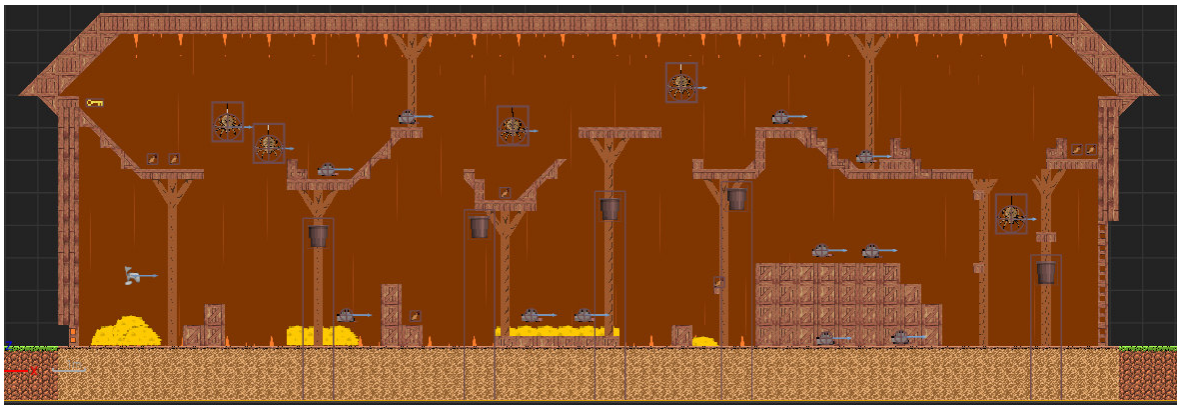

Fig. 23: Primer nivel completo

Adición de fondos y música a niveles

Los últimos elementos que se deben añadir a un nivel son un escenario de fondo y música que se reproduzca durante todo el nivel, estos son elementos intangibles que no alteran la jugabilidad en lo absoluto, pero le dan ambientación al nivel y hacen más estéticamente agradable la experiencia de juego. Los fondos se pueden arrastrar al nivel como imágenes de tamaño varias veces superior a todo el nivel que se convierten en *sprites* y se ubican mucho más profundo que el resto de los elementos, de esta forma se consigue que el fondo sea un elemento omnipresente pero que no pueda interferir en el juego; alternativamente se puede construir el fondo con bloques en el *tilemap* ubicados en una capa diferente al

resto de bloques para que no generen colisión o no agregar geometría a los bloques bases. La Fig. 24 muestra el primer nivel luego de añadirle un fondo.

Fig. 24: Nivel 1 con su fondo

La música y cualquier archivo de sonido se puede importar a Unreal Engine en varios formatos, este automáticamente los convierte en formato WAV y los ubica en el explorador de contenido como otro *asset*; dicho *asset* se debe vincular a una clase de sonido denominada *sound cue*, la cual se puede llamar dentro de *blueprints* para controlar cuando reproducir el sonido o música. Para este proyecto se decidió reproducir la única música obtenida en repetición ilimitada en cada nivel, la Fig. 25 muestra las *blueprints* usadas para esto.

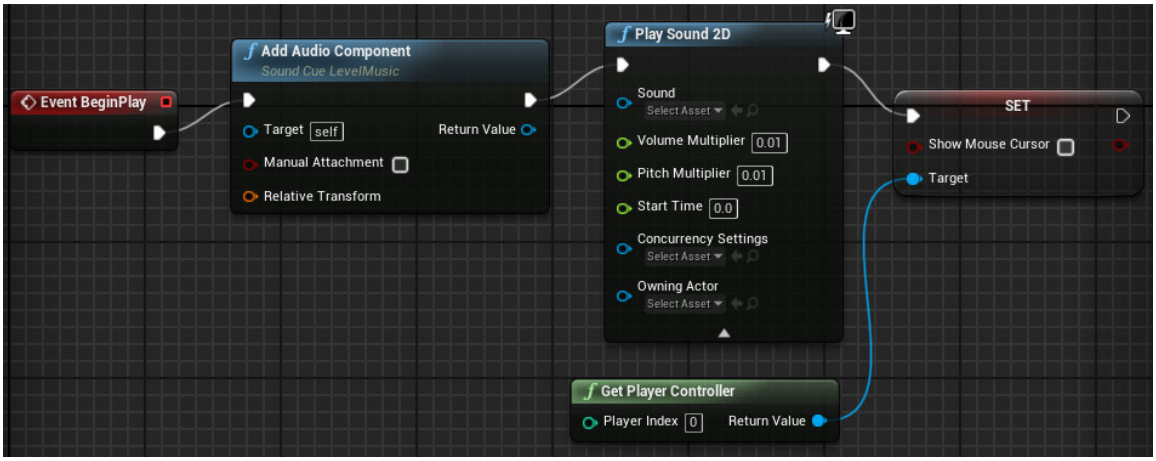

Fig. 25: *Blueprints* para reproducir música en el primer nivel

Diseño en implementación de HUD

El *heads up display* es la interfaz intangible que muestra al jugador información sobre el estado del juego; en los juegos de plataformas 2D, el HUD es mínimo o inexistente. En

este proyecto se limitó el HUD a una barra representada por una pluma que muestra la salud del jugador, esto se puede observar al iniciar cualquier simulación del juego como se muestra en la Fig. 26.

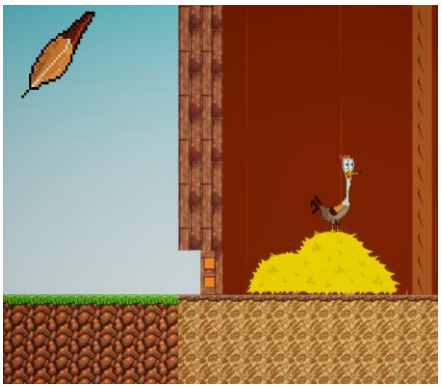

Fig. 26: Barra de salud del jugador en la esquina del juego

Para hacer esto posible, se utilizó la clase que Unreal Engine tiene para estos casos llamada *widget*, en ella se tiene un editor que permite diseñar el HUD o cualquier interfaz con varios elementos personalizables y codificar dichos elementos con *blueprints* para que tengan funcionalidad. Debido a que el HUD debe siempre estar presente para el jugador, se llama al *widget* desde la clase del jugador para instanciarlo siempre que se inicie el juego, esto se muestra en la Fig. 27.

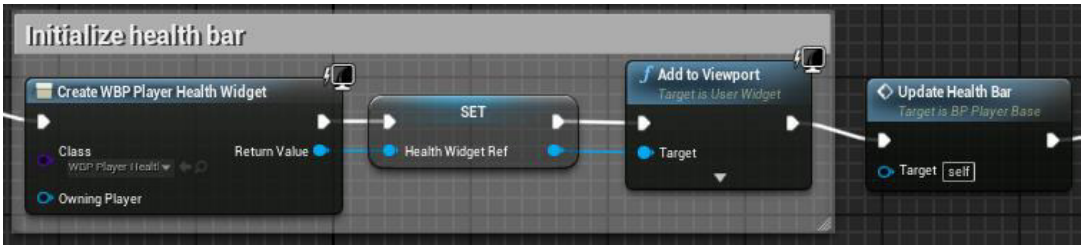

Fig. 27: *Blueprints* para iniciar el *widget* junto con el jugador

Calibración de interacciones en niveles

Un juego debe tener una experiencia balanceada para ser disfrutable, la única forma de verificar que esto se está logrando es realizar simulaciones y ejecuciones del juego cada vez que se añaden elementos a un nivel para verificar que cumplen el rol adecuado; un juego puede estar mal balanceado si los enemigos u objetos obstruyen o abruman al jugador, la navegación por el nivel es innecesariamente complicada o existen pocas posibilidades de sobrevivir para llegar al final del nivel. En este proyecto se realizaron pruebas en el transcurso de todo el desarrollo para mantener cada nivel balanceado.

Diseño del logo del juego

El logo del juego usualmente se muestra en la pantalla de título del juego, si este es llamativo, puede dejar una buena primera impresión en el jugador. En este proyecto se utilizó la herramienta Pixlr para diseñar el logo y el resto de las imágenes para las pantallas del juego.

La Fig. 28 muestra el logo diseñado en Pixlr, en esta imagen se pueden observar que este editor tiene todas las herramientas comunes para crear y editar fotos y multimedia.

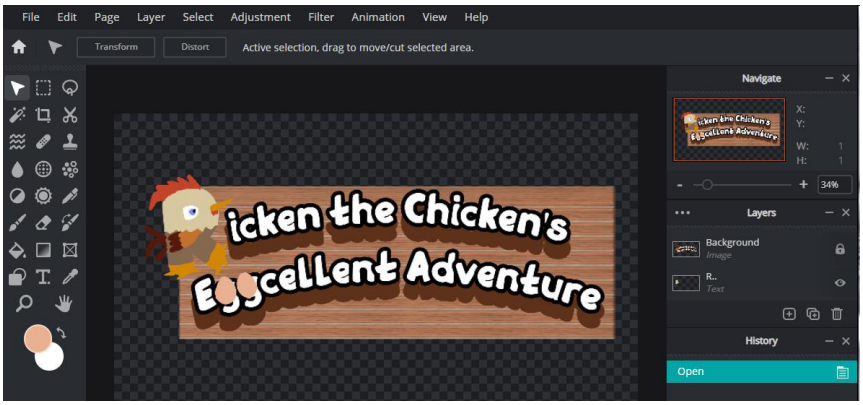

Fig. 28: Logo diseñado en Pixlr

Diseño e implementación de pantalla de título con menú de inicio

La pantalla de título, además de ser llamativa, debe tener toda la funcionalidad que permita al jugador iniciar el juego en diferentes ocasiones. En este proyecto, se creó una pantalla de título con botones para iniciar una nueva partida, continuar en una partida guardada, ver créditos y salir del juego; la pantalla y los botones se diseñaron en Pixlr y se implementaron usando la clase *widget* para dar la funcionalidad específica a cada botón. La Fig. 29 muestra la pantalla de título completa.

Fig. 29: Pantalla de título y menú principal del juego

Diseño de escenas y transiciones

El proyecto desarrollado posee pantallas de introducción antes de comenzar cada nivel, estas pantallas sirven de transición entre niveles, brindan al jugador contexto y pistas sobre el nivel y poseen botones para continuar o salir al menú principal. Cada una de estas pantallas fue diseñada en Pixlr e implementada en un *widget.* La Fig. 30 muestra una de estas pantallas.

Fig. 30: Pantalla introductoria del primer nivel

Adición de sonido a escenas y transiciones

Por simplicidad, se decidió reproducir la misma música que se usa en los niveles para el menú de inicio y cada una de las pantallas de transición, para esto se puede usar el mismo *sound cue* creado para los niveles, efectivamente logrando que esta tarea sea muy corta y sencilla.

Sprint 3. Agrupación y funciones adicionales

Esta iteración abarca todas las tareas que se realizan luego de haber finalizado todas las tareas anteriores debido a su menor riesgo de desarrollo o a que son una conclusión lógica de las mismas; la funcionalidad resultante de algunas de estas tareas puede incluso no ser crítica para el funcionamiento del juego, pero no se puede afirmar que son opcionales, pues podrían hacer al juego más estable y robusto a ojos del usuario. Las tareas son las siguientes:

- Diseñar e implementar menú de pausa
- Calibrar pausa dentro del juego y botón de pausa
- Incorporar escenas de inicio en sus niveles respectivos
- Implementar finalización de niveles
- Implementar función de guardado al final de los niveles

- Implementar función de carga en el menú de inicio

Diseñar e implementar menú de pausa

La mayoría de los videojuegos se pueden pausar, es una función simple pero que evita que los jugadores tengan algo que perder si deben abandonar su juego repentinamente. Igual que las otras pantallas, se diseñó en Pixlr con un fondo gris con transparencia para que se pueda ver el nivel de fondo a la vez que se sepa que el juego está pausado. Sus dos botones permiten al jugador volver al juego o salir al menú principal, pero la principal función del menú de pausa en este proyecto es mostrar al jugador las teclas correspondientes a cada acción del juego.

La Fig. 31 muestra el menú de pausa en el editor de *widgets* de Unreal Engine.

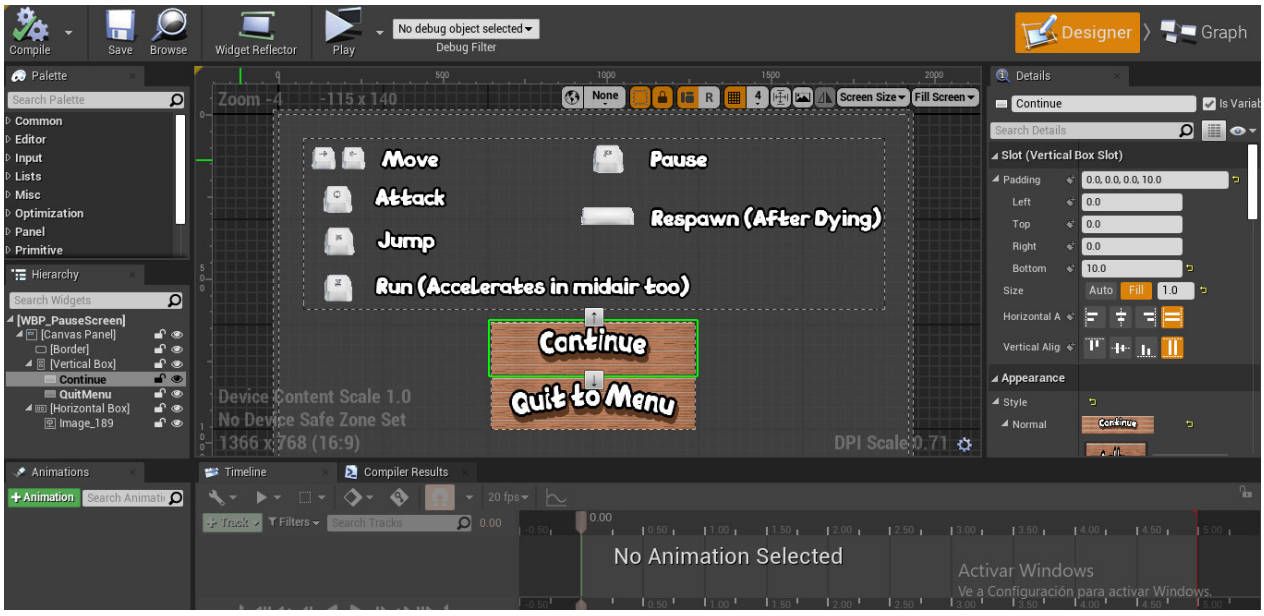

Fig. 31: Menú de pausa siendo implementado como *widget*

Calibración de pausa dentro del juego y botón de pausa

Lo que diferencia a la pantalla de pausa del resto de pantallas del juego es que esta no tiene un espacio exclusivo entre niveles, sino que debe aparecer y desaparecer en medio de cualquier acción en cualquier nivel; para lograr esto, se crea otro evento con una tecla en la clase del jugador que abre el menú cuando dicha tecla se presiona y lo cierra cuando se hace click en el botón de continuar, la pausa del juego en sí se logra llamando a una función prediseñada de Unreal Engine que hace exactamente lo que se requiere.

La Fig. 32 muestra las *blueprints* en la clase del jugador que se necesitaron para pausar el juego con una tecla.

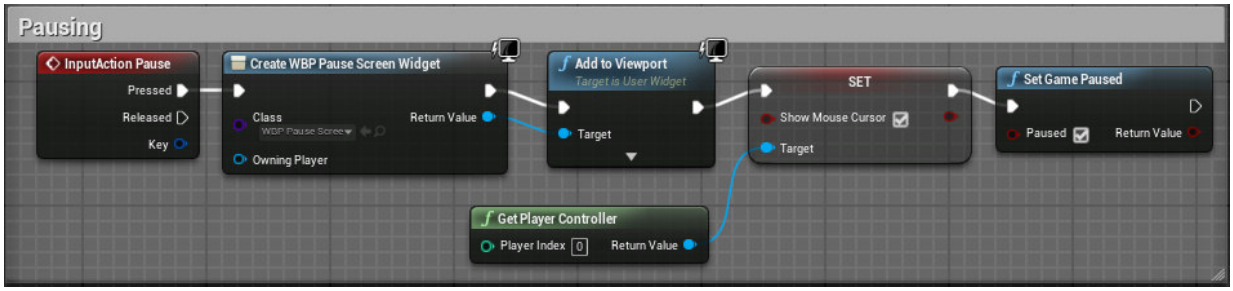

Fig. 32: *Blueprints* de pausa en la clase del jugador

Incorporación de escenas de inicio en sus niveles respectivos

Habiendo diseñado e implementado las pantallas previas de cada nivel, se debe juntar cada una con su nivel de modo que siempre se muestre la pantalla antes de continuar al nivel. Aparentemente la pantalla introductoria es parte del nivel, pero en la práctica se crea un nivel vacío que "hospeda" a la pantalla, de este modo, continuar hacia el respectivo nivel no es más que la transición del nivel vacío al nivel jugable. Este principio se aplica para tambien para otras pantallas como el menú de inicio, los créditos y la pantalla final; la Fig. 33 muestra las *blueprints* necesarias para realizar esta transición.

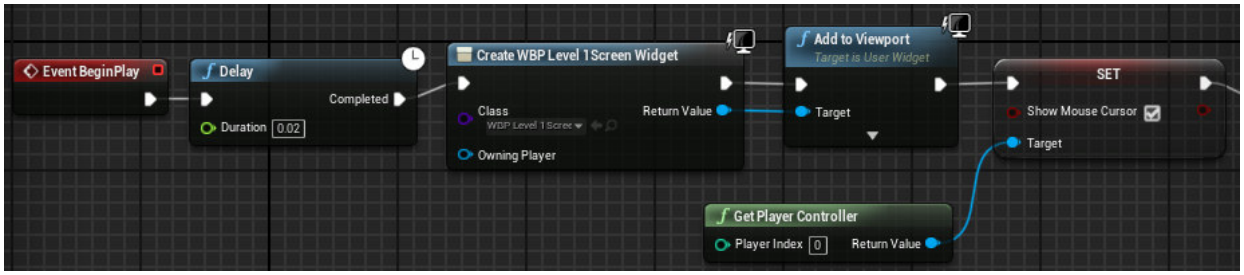

Fig. 33: *Blueprints* del nivel "hospedador" de las pantallas de introducción

Implementación de finalización de niveles

En todo videojuego, la progresión entre niveles debe ser lo más orgánica y automática posible, esto implica crear una transición hacia el siguiente nivel o pantalla tan pronto como el jugador llega al final de un nivel; para crear dicha transición en contacto con el jugador se creó un objeto interactivo invisible e intangible y se lo ubicó al final de cada nivel, para el jugador esto es transparente, pero este objeto abre el siguiente nivel tan pronto como el personaje principal colisiona con él. Cabe recalcar que, pese a que la funcionalidad de dicho objeto es la misma, el nivel al que lleva es diferente, por tal motivo, se requieren crear tantos objetos únicos como niveles se tenga en el videojuego.

Implementación de función de guardado al final de los niveles

El videojuego se diseñó para guardar la partida de forma automática cada vez que el jugador termina un nivel, para esto, Unreal Engine proporciona una clase específica para guardar datos del juego; en este caso, solo se tienen datos de finalización de cada nivel que se guardan en forma de variables booleanas, pero esto es suficiente para poder llamar a los métodos de guardado propios de esta clase que se encargan de instanciar un objeto de la clase de guardado, bautizarlo como ranura de guardado y utilizarlo para llevar registro de los cambios en las variables de finalización. Este funcionamiento está vinculado al objeto de finalización de cada nivel, por lo que son sus *blueprints* las que llaman al método de guardado y actualizan las variables al final de cada nivel, estas *blueprints* muestran en la Fig. 34.

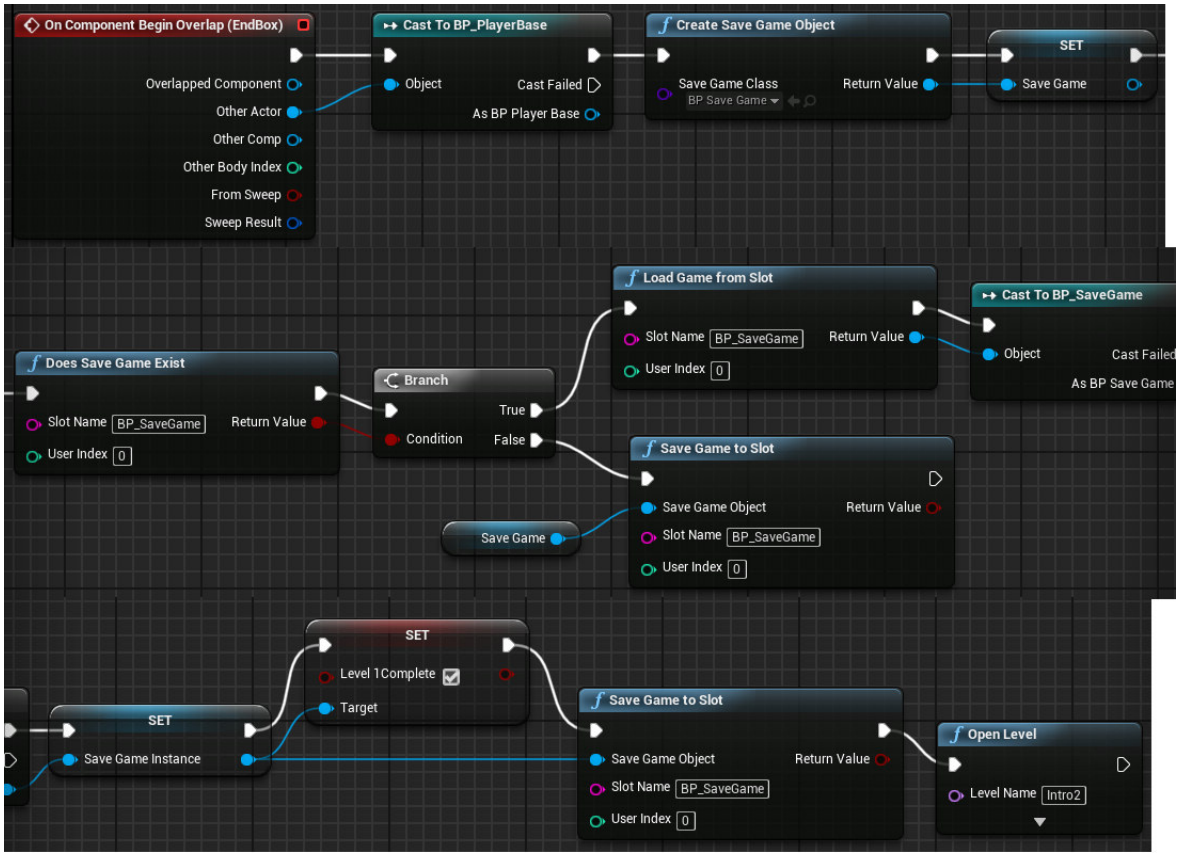

Fig. 34: *Blueprints* de guardado en la clase del objeto de finalización

Implementación de función de carga en el menú de inicio

El guardado y la carga de partidas son dos caras de la misma moneda, llevando un registro del estado de las variables de finalización, cargar una partida no es más que leer dichas variables y usarlas para ejecutar la transición al nivel que corresponde. Las *blueprints* mostradas en la Fig. 35 corresponden al botón de carga del menú de inicio y se encargan

de revisar si existe un archivo guardado, crearlo y empezar desde el inicio si no existe, y entrar a un grupo de condiciones anidadas para cargar el nivel correcto si el archivo existe. El botón de nueva partida automáticamente crea un archivo de guardado que reemplaza al anterior, permitiendo al jugador empezar desde el inicio.

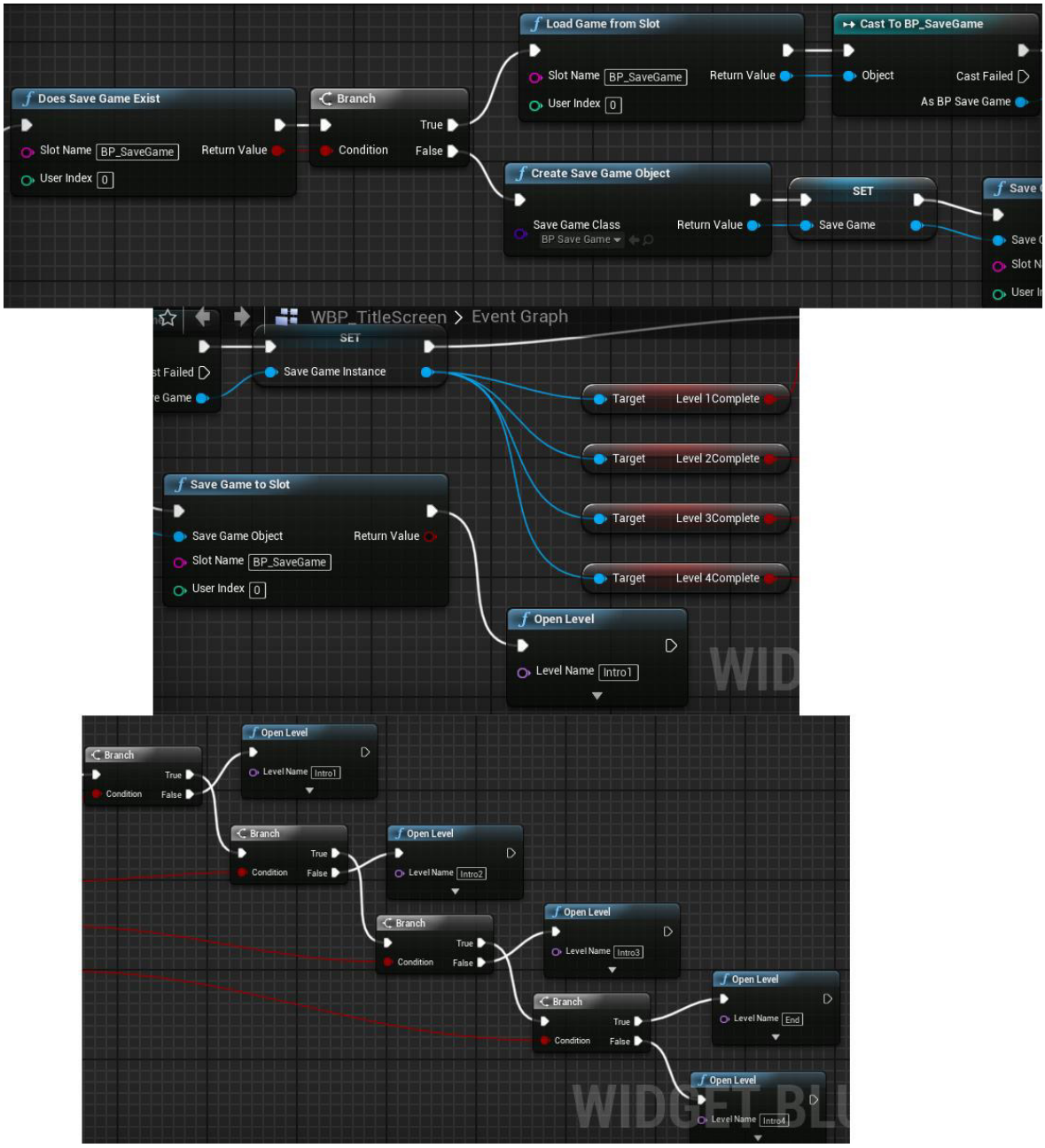

Fig. 35: *Blueprints* carga de partida

Sprint 4. Pruebas, depuración y lanzamiento

Esta es la iteración final que abarca la fase beta en concordancia con la metodología SUM, las tareas realizadas consisten en probar minuciosamente el juego desarrollado y realizar

las correcciones necesarias antes de lanzar la versión final. En esta sección se mencionan brevemente los resultados que se esperan de las pruebas, referirse al Anexo II para visualizar las pruebas con más detalle.

Las tareas realizadas son las siguientes:

- Lanzar versión beta
- Probar menú de inicio del juego
- Probar acciones del personaje
- Probar interacción con enemigos / obstáculos / entorno, muerte
- Verificar comportamiento, muerte de enemigos
- Verificar escenas y transiciones
- Probar función de guardado y carga
- Probar el juego completo
- Lanzar versión final
- Crear ejecutable de la versión final

Lanzar la versión beta

Debido al pequeño tamaño de este proyecto, no es necesario realizar el proceso de lanzamiento para la versión beta, resulta más conveniente jugar esta versión preliminar en el propio editor de Unreal Engine, de esta forma se pueden realizar cambios instantáneamente y acceder a opciones de desarrollador como impresión de *strings*, selección de niveles y salir del juego en cualquier momento. La Fig. 36 muestra el juego ejecutándose en Unreal Engine.

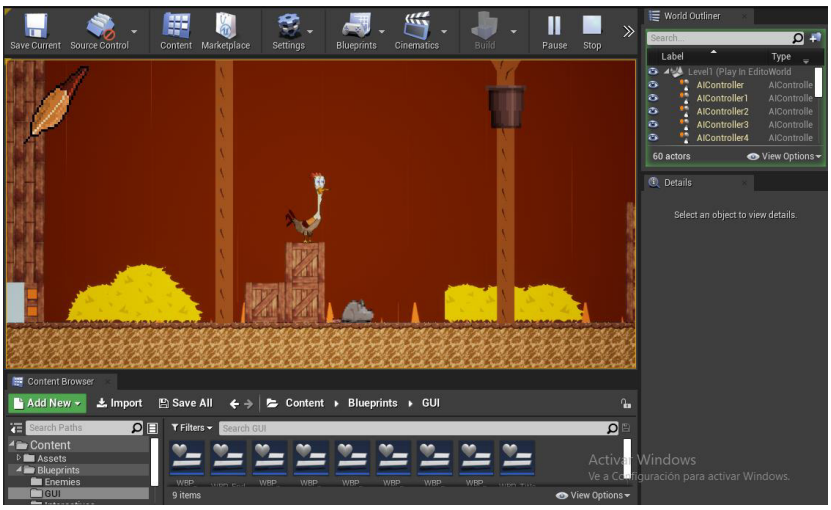

Fig. 36: Jugando versión beta en Unreal Engine

Prueba de menú de inicio del juego

El menú de inicio debe ser la primera pantalla en aparecer al iniciar el juego, debe tener música de fondo y sus botones se debe poder presionar para iniciar el juego, continuarlo, ir a créditos o salir del juego. Los botones deben cambiar de color si se pasa el cursor sobre ellos.

Prueba de acciones del personaje

las teclas correspondientes a las acciones del jugador son: flechas derecha e izquierda para moverse, Z para correr, X para saltar, C para atacar, P para pausar y espacio para revivir. Al presionar cada tecla se debe ver la animación respectiva de la acción, estas animaciones no se pueden reproducir simultáneamente, por lo que una acción debe cancelar a la anterior o impedir que se realice otra hasta que haya finalizado; la excepción a esto es la animación de daño, la cual dura durante el periodo de invencibilidad y reemplaza al resto de animaciones durante este periodo.

Prueba de interacción con enemigos / obstáculos / entorno, muerte

El jugador debe infligir daño a los enemigos si colisiona con estos durante la animación de ataque o aterriza sobre ellos y recibir daño en cualquier otra colisión. Los obstáculos deben interactuar con el jugador de acuerdo con su función de interacción. El jugador debe morir siempre que su salud llegue a cero, reflejado por la pluma del HUD quedando destruida luego de recibir daño 3 veces; una vez que el jugador muere, su personaje caerá a través del terreno y el jugador podrá presionar la tecla correspondiente para revivir al inicio del nivel o en un punto de control con su salud reestablecida.

Verificación de comportamiento, muerte de enemigos

Todos los personajes y objetos deben estar sujetos a colisiones con el terreno y deben poder navegar a través de este sin quedar atascados y sin presentar comportamientos erráticos; los enemigos deben colisionar entre ellos, pero no recibir daño de esta colisión. Los objetos interactivos deben reaccionar únicamente al jugador, no a enemigos u otros objetos, y tampoco deben colisionar entre ellos o con enemigos (existen excepciones). Cada enemigo morirá luego de recibir daño del jugador una vez y generará una copia de sí mismo en su posición inicial luego de un intervalo de tiempo desde su muerte.

Verificación de escenas y transiciones

Las pantallas de transición deben tener música y botones que permiten avanzar al nivel correspondiente o salir al menú principal, estas pantallas siempre se deben mostrar antes

de un nivel, sin importar de qué forma se acceda a dicho nivel. La pantalla final solo se podrá acceder al terminar el juego y tendrá conexiones con el menú principal y los créditos; los créditos se pueden acceder desde esta pantalla o el menú de inicio y solo tienen conexión con este último.

Prueba de función de guardado y carga

El progreso del juego se guarda luego de cada nivel sin intervención del jugador y de forma completamente transparente para él. La función de carga debe regresar al jugador a la pantalla de introducción del nivel próximo al último nivel terminado, o a la pantalla final si ha terminado el último nivel. En cualquier momento se puede empezar una nueva partida, lo cual sobrescribe el progreso guardado anterior.

Prueba del juego completo

Luego de realizar todas las pruebas anteriores y realizar todas las correcciones necesarias, se prueba la versión beta en su totalidad; al realizar esto en el proyecto, se pudo verificar que el videojuego cumple con las expectativas planteadas y proporciona la experiencia que se planeó desde el inicio.

Lanzamiento de la versión final

La versión final de un videojuego es aquella que se distribuye a los usuarios, por lo que debe estar terminada y no presentar errores. Unreal Engine posee las herramientas necesarias para convertir un proyecto en un juego listo para ser distribuido, el proceso involucra modificar configuraciones en las siguientes secciones de configuración del proyecto:

- En la sección de *packaging* se especifican los niveles (llamados aquí mapas) que deben incluirse, el tipo de distribución (en este caso *shipping*) y si se deben incluir con el paquete del juego los componentes necesarios para su ejecución.
- En *maps and modes* se selecciona el nivel que siempre se abrirá primero en el juego, en este caso este es el menú principal.
- En *supported platforms* se elige las plataformas sobre las que se podrá ejecutar el juego, en este caso solo Windows.
- En la categoría de plataformas se debe seleccionar la plataforma elegida para el juego y revisar todos los ajustes para configurar el juego para dicha plataforma; para Windows los ajustes por defecto fueron suficientes, solo se subió imágenes para que aparezcan como ícono y portada del juego al momento de iniciarlo en otros computadores.

La Fig. 37 muestra parte de los ajustes para la plataforma Windows.

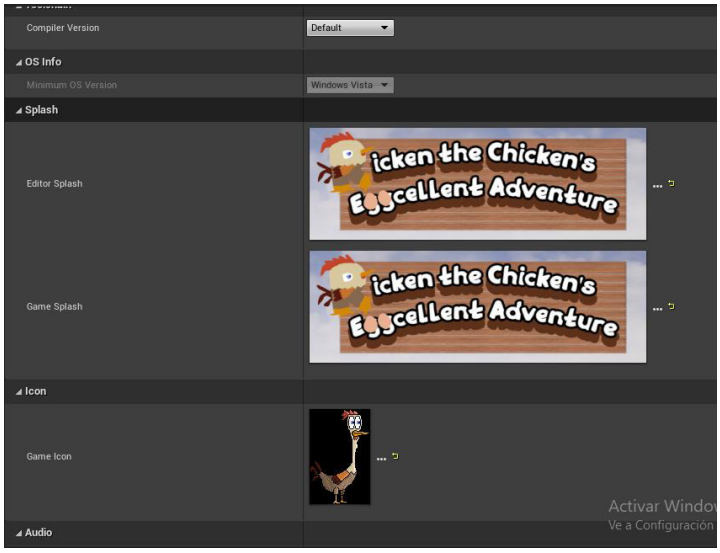

Fig. 37: Cambio de portada e ícono en ajustes de plataforma

Luego de ajustar los ajustes se debe construir el proyecto con la opción de la barra de herramientas y comenzar la creación del paquete con la opción *package Project* ubicada en el menú de archivo, ambas opciones se pueden visualizar en la Fig. 38.

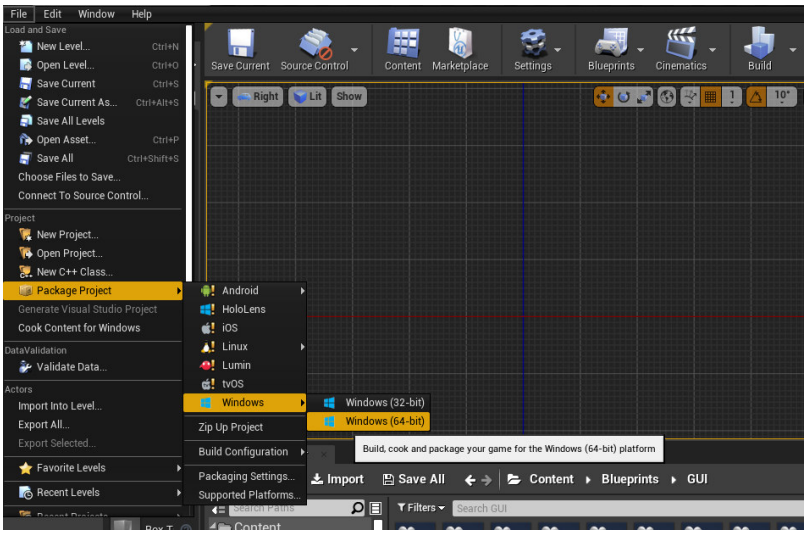

Fig. 38: Opciones de construcción y creación del paquete en Unreal Engine

Creación de ejecutable de la versión final

Al hacer click en crear el paquete del proyecto en la plataforma deseada, se selecciona una carpeta para guardar el paquete y Unreal Engine se encarga del resto en un proceso que puede tardar varios minutos dependiendo del tamaño del proyecto y de las especificaciones del computador que se usó para desarrollar el videojuego. Al terminar este

proceso, se obtiene una carpeta de archivo donde se encuentra el ejecutable, esta carpeta es la que se debe distribuir a los usuarios y se muestra en la Fig. 39.

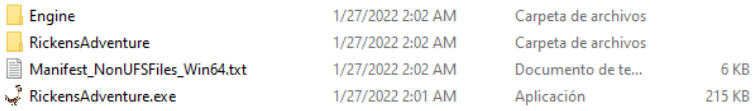

Fig. 39: Ejecutable del juego en la carpeta generada

4 CONCLUSIONES

- Como demostración del potencial que Unreal Engine tienen para crear todo tipo de video juegos, se ha desarrollado un juego de plataformas 2D. De esta manera se cumple el objetivo general de este proyecto.
- La historia de los videojuegos de este género suele ser muy sencilla, por lo que los personajes y una trama satisfactoria se pueden crear en poco tiempo.
- La tarea de diseño es más complicada si no se tienen habilidades artísticas y periféricos adecuados. En este proyecto se evidencia el cumplimiento casi total del objetivo de diseño a través de los personajes; para el resto de *assets* se recurre a internet ya que es imposible completar un proyecto de esta magnitud en el tiempo dado si se diseñan todos los elementos.
- Debido al corto tiempo disponible existe la necesidad de cortar contenido extra pensado para el videojuego, no obstante, este sigue teniendo la longitud adecuada para hacerlo una experiencia disfrutable.
- Durante todo el desarrollo y la fase beta, el juego debe pasar por constantes pruebas para asegurarse que la versión final no tenga fallas y funcione como se había concebido.
- El desarrollo de un videojuego es drásticamente diferente al desarrollo de cualquier aplicación web, móvil o de escritorio; empezando por el lenguaje visual (*blueprints)* que permite codificar sin escribir código como tal. Sin embargo, los conceptos de programación orientada a objetos facilitan entender la lógica detrás del funcionamiento de las *blueprints* y la estructura de clases, herencia y polimorfismo con la que trabaja Unreal Engine.
- Adaptar el proceso de desarrollo estándar al desarrollo de un videojuego es un desafío, sobre todo al momento de producir artefactos, pero se puede lograr debido a la existencia de la metodología dedicada SUM y al entendimiento de los conceptos de desarrollo.

5 RECOMENDACIONES

- La primera recomendación que surge luego de finalizar este proyecto es que un videojuego puede beneficiarse drásticamente de más tiempo de desarrollo, esto es algo que se evidencia incluso en títulos desarrollados por las más grandes compañías. Por lo que antes de comenzar con el desarrollo se recomienda asegurarse de que se cuenta con suficiente tiempo para hacer un producto de calidad sin complicaciones.
- El desarrollo de un videojuego tampoco debe ser el trabajo de una sola persona, ya que incluso con todas las herramientas y recursos disponibles, existen demasiadas disciplinas que se deben perfeccionar; no es razonable esperar ser bueno en todo, por lo que se recomienda tratar de aportar valor a un equipo en lugar de asumir toda la carga personalmente.
- Unreal Engine es con certeza la herramienta más versátil y potente para desarrollar videojuegos, dicha potencia sin embargo no se aprovecha si se lo utiliza para hacer juegos muy simples e incluso puede traer efectos negativos en el producto final como archivos demasiado pesados y requerimientos elevados para ejecutarlo; esto sin mencionar que no cualquier computador puede manejar Unreal Engine efectivamente. Se recomienda seguir usando Unreal Engine pero también explorar otras opciones más ligeras siempre que se tengan ideas de juegos muy simples.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] B. Bullard, "VIDEO GAMES ARE 2020'S BIGGEST ENTERTAINMENT MONEYMAKER, BLOWING PAST MOVIES & SPORTS COMBINED," SyFyWire, 23 Diciembre 2020. [Online]. Available: https://www.syfy.com/syfywire/video-games-2020-revenues-outpace-movies-sports. [Accessed 8 Noviembre 2021].

[2] A. Hern, "Playing video games doesn't lead to violent behaviour, study shows," The Guardian, 22 Julio 2020. [Online]. Available: https://www.theguardian.com/games/2020/jul/22/playing-video-games-doesnt-lead-to-violent-behaviour-study-shows. [Accessed 20 Enero 2022].

[3] Geico Living, "Benefits of Video Games For Kids & Adults," Geico, [Online]. Available: https://www.geico.com/living/home/technology/9-reasons-to-give-video-games-a-try/. [Accessed 20 Enero 2022].

[4] J. Rossen, "What Was The First Video Game?," Mental Floss, 14 Junio 2017. [Online]. Available: https://www.mentalfloss.com/article/501496/what-was-first-video-game. [Accessed 9 Noviembre 2021].

[5] Encyclopedia Britannica, "Pong," Britannica, 7 Mayo 2020. [Online]. Available: https://www.britannica.com/topic/Pong. [Accessed 9 Noviembre 2021].

[6] C. Hallman, "The Top 50 Highest-Grossing Video Game Franchises," TitleMax, [Online]. Available: https://www.titlemax.com/discovery-center/lifestyle/the-top-50-highest-grossing-video-game-franchises/. [Accessed 9 Noviembre 2021].

[7] L. Johnson, "1993's 'Doom' Wasn't the 3D Game We Think It Was," Vice, 15 Mayo 2016. [Online]. Available: https://www.vice.com/en/article/4xaagg/doom-wasnt-3d. [Accessed 10 Noviembre 2021].

[8] W. Hosch, "Electronic platform game," Encyclopædia Britannica, 30 Julio 2018. [Online]. Available: https://www.britannica.com/topic/electronic-platform-game. [Accessed 10 Noviembre 2021].

[9] S. Powell, "Why we still love platform games," bbc.com, 27 Agosto 2015. [Online]. Available: https://www.bbc.com/news/newsbeat-34070835. [Accessed 10 Noviembre 2021].

[10] R. Kay, "The future of 2D gaming," GamesIndustry.biz, 28 Noviembre 2018. [Online]. Available: https://www.gamesindustry.biz/articles/2018-11-27-the-future-of-2d-gaming. [Accessed 11 Noviembre 2021].

[11] K. Terrel, "Video Games Score Big With Older Adults," AARP, 16 Diciembre 2019. [Online]. Available: https://www.aarp.org/home-family/personal-technology/info-2019/report-video-games.html. [Accessed 11 Noviembre 2021].

[12] K. Petersen and C. Wohlin, "A Comparison of Issues and Advantages in Agile and Incremental Development between State of the Art and an Industrial Case," 28 Enero 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.717.6830&rep=rep1&type=pdf. [Accessed 21 Enero 2022].

[13] S. Denning, "Agile: The World's Most Popular Innovation Engine," Forbes, 23 Julio 2015. [Online]. Available: https://www.forbes.com/sites/stevedenning/2015/07/23/the-worlds-most-popular-innovation-engine/?sh=1f57614e7c76. [Accessed Noviembre 21 2021].

[14] C. Keith, "The State of Agile in the Game Industry," Game Developer, 4 Marzo 2010. [Online]. Available: https://www.gamedeveloper.com/business/the-state-of-agile-in-the-game-industry. [Accessed 22 Noviembre 2021].

[15] Object Management Group, "Software & Systems Process Engineering Meta-Model Specification," Abril 2008. [Online]. Available: https://www.omg.org/spec/SPEM/2.0/PDF. [Accessed 21 Noviembre 2021].

[16] N. Acerenza, A. Coppes, G. Mesa, A. Viera, E. Fernandez, T. Laurenzo and D. Vallespir, "Una Metodología para Desarrollo de," 2009. [Online]. Available: https://www.fing.edu.uy/sites/default/files/biblio/22811/asse_2009_16.pdf. [Accessed 23 Noviembre 2021].

[17] X. Murillo, A. Gutierrez, A. Ibañez, J. Quiroz, G. Sahonero and F. Díaz, "Implementación de la metodología SUM modificada para el desarrollo de videojuegos orientados al aprendizaje en Bolivia," 2018. [Online]. Available: https://www.imt.ucb.edu.bo/documents/publications/Implementacion-de-la-Metodologia-SUM-Modificada-para-el-Desarrollo-de-Videojuegos-Orientados-al-Aprendizaje-en-Bolivia.pdf. [Accessed 23 Noviembre 2021].

[18] Gemserk.com, "SUM para Desarrollo de Videojuegos," Gemserk.com, [Online]. Available: http://www.gemserk.com/sum/#:~:text=La%20metodolog%C3%ADa%20SUM%20para%20videojuegos,eficacia%20y%20eficiencia%20de%20esta. [Accessed 24 Noviembre 2021].

[19] Scrum.org, "What is a Product Owner?," Scrum.org, [Online]. Available: https://www.scrum.org/resources/what-is-a-product-owner. [Accessed 23 Noviembre 2021].

[20] A. Alvear and E. Vargas, "Desarrollo de un videojuego role playing game con unreal engine.," 1 Junio 2021. [Online]. Available: https://bibdigital.epn.edu.ec/handle/15000/21674. [Accessed 24 Noviembre 2021].

[21] Pluralsight, "Game Concept: How To Come up With a Game Idea," Pluralsight, 26 Junio 2014. [Online]. Available: https://www.pluralsight.com/blog/film-games/creating-game-concept-first-step-getting-game-ground. [Accessed 25 Noviembre 2021].

[22] Agile Alliance, "Epic," Agile Alliance, [Online]. Available: https://www.agilealliance.org/glossary/epic/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'epic))~searchTerm~'~sort~false~sortDirection~'asc~page~1). [Accessed 29 Noviembre 2021].

[23] Scrum.org, "What is a Sprint Backlog?," Scrum.org, [Online]. Available: https://www.scrum.org/resources/what-is-a-sprint-backlog. [Accessed 29 Noviembre 2021].

[24] O. Pleten, "What Is a Mockup and Why Do We Need It," keen ethics, 22 Octubre 2019. [Online]. Available: https://keenethics.com/blog/1521631041972-the-importance-of-mockups. [Accessed 29 Noviembre 2021].

[25] Game-Ace, "A beautiful preview: Why video game concept art matters," Game-Ace, 28 Enero 2020. [Online]. Available: https://game-ace.com/blog/video-game-concept-art/. [Accessed 5 Diciembre 2021].

[26] J. Sobolev, "What Are Sprites And How They Work In Games?," Gaming Shift, [Online]. Available: https://gamingshift.com/sprites-in-games/. [Accessed 6 Diciembre 2021].

[27] C. McCarthy, "20 Video Game Starting Screens That Left A Fantastic First Impression," US Gamer, 11 Junio 2020. [Online]. Available: https://www.usgamer.net/articles/the-20-best-video-game-start-title-screens. [Accessed 7 Diciembre 2021].

[28] Packt, "An overview of Unreal Engine," Packt, 9 Diciembre 2015. [Online]. Available: https://hub.packtpub.com/overview-unreal-engine/. [Accessed 7 Diciembre 2021].

[29] C. Jansen, "Benefits of using Unreal Engine for your next Game app development," WhaTech, 21 Octubre 2019. [Online]. Available: https://www.whatech.com/games/press-releases/622394-benefits-of-using-unreal-engine-for-your-next-game-app-development. [Accessed 7 Diciembre 2021].

[30] J. Descottes, "piskelapp / piskel," Github, 25 Noviembre 2018. [Online]. Available: https://github.com/piskelapp/piskel. [Accessed 13 Diciembre 2021].

[31] Pixlr, "Photo editor, animation and design," 123RF, [Online]. Available: https://pixlr.com/. [Accessed 13 Diciembre 2021].

[32] Dev Enabled, "Unreal Engine 4 - Making a 2D Platformer in UE4 - Full Length," YouTube, [Online]. Available: https://www.youtube.com/watch?v=Rr9gYK50IuY. [Accessed 5 Enero 2022].

[33] M. Aspland, "How To Setup Multiple Checkpoints And Respawn At Them - Unreal Engine 4 Tutorial," YouTube, [Online]. Available: https://www.youtube.com/watch?v=TaGE3e93hM8. [Accessed 5 Enero 2022].

[34] EvilDoUsHarm, "[UE4] 2D Platformer Tutorial!," YouTube, 22 Enero 2020. [Online]. Available: https://www.youtube.com/playlist?list=PLKMRiZuSgt-7wY7hfUzg8JcuTFXAf85jv. [Accessed 7 Enero 2022].

[35] The Specialist, "UE4 - Level Music - Tutorial," YouTube, 11 Junio 2019. [Online]. Available: https://www.youtube.com/watch?v=Ky_Hw8Tlmy8. [Accessed 7 Enero 2022].

[36] M. Aspland, "Packaging And Exporting Your Game - Unreal Engine 4 Tutorial," YouTube, [Online]. Available: https://www.youtube.com/watch?v=MatkFIvCKMY. [Accessed 25 Enero 2022].

[37] Gamedev Market, "Game Assets For Indie Game Developers," Gamedev Market, [Online]. Available: https://www.gamedevmarket.net/. [Accessed 1 Enero 2022].

[38] D. Curtis, "Spotlight," SoundCloud, [Online]. Available: https://soundcloud.com/desperate-measurez. [Accessed 20 Enero 2022].

7 ANEXOS

ANEXO I. Porcentaje de plagio

ANEXO II. Manual técnico

ANEXO III. Manual de usuario

ANEXO IV. Guía de inicio rápido

Anexo 1. Certificado de Originalidad

CERTIFICADO DEORIGINALIDAD

Quito, D.M. 18 de febrero de 2022

De mi consideración:

Yo, Juan Pablo Zaldumbide Proaño, en calidad de director del Trabajo de Integración Curricular titulado DESARROLLO DE VIDEOJUEGO DE PLATAFORMAS 2D UTILIZANDO UNREAL ENGINE asociado al proyecto DESARROLLO DE VIDEOJUEGO DE PLATAFORMAS 2D UTILIZANDO UNREAL ENGINE de la carrera en TECNOLOGÍA SUPERIOR EN DESARROLLO DE SOFTWARE, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 8%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presentedocumento para los trámites de titulación.

NOTA: Se adjunta el informe generado por la herramienta Turnitin en el Quipux correspondiente.

Atentamente,

Ing. Juan Pablo Zaldumbide Proaño, M.Sc.
Docente ESFOT

ANEXO II

1 DOCUMENTO DE CONCEPTO

A continuación, se presenta este artefacto que se generó en la fase de concepto y se modificó al iniciar la fase de desarrollo.

Concepto del Juego

Título: Ricken the Chicken's Eggcellent Adventure

Género: Juego de Plataformas 2D

Historia:

Era un día como cualquier otro en la granja del abuelo Joe, o tal vez no lo era, ya que los gritos de las gallinas habían inundado la granja: todos los huevos habían desaparecido. En esta terrible situación le correspondía al gallo, grande y fuerte, líder del gallinero, uno de los animales más confiables del abuelo Joe, averiguar qué había pasado…el problema es que el gallo no está disponible por el momento. La única esperanza, el flaco gallo de reserva, Ricken. Ante los ruegos de todas las aves de corral, buscará toda la granja para encontrar esos huevos y probará que las apariencias no siempre hacen al héroe.

Personajes:

- **Ricken the Chicken –** Protagonista, gallo de reserva de la granja. Flaco, débil, no muy listo, pero tiene una gran determinación para lograr sus objetivos.
- **Grandpa Joe (mencionado) –** Dueño de la granja, con muchos años encima.

Enemigos:

- **Nosey Mousey –** Ratón de campo, solo sabe moverse en una dirección y puede ser derrotado fácilmente.
- **Betsy Spider –** Araña colgante ubicada en lugares inoportunos, su presencia puede dificultar algunos saltos o contribuir a ellos si se aprovecha el impulso de su derrota como plataforma de único uso.
- **Bombird –** Pájaros gruñones que sobrevuelan dejando caer manzanas sobre la cabeza de quien se encuentre pasando por debajo. Se puede aterrizar sobre ellos si se llega a su altura.

Obstáculos:

- **Espino de madera –** Tronco afilado de orígenes inciertos que siempre lastimará a Ricken si lo toca, no puede ser movido ni destruido.
- **Objetos que caen –** Varios objetos que podrían ser inofensivos si se encontraran estáticos en el piso, pero causan daño por el hecho de caer sobre la cabeza de Ricken cuando este pasa por debajo de ellos. Dependiendo del nivel pueden ser manzanas, cuchillos, baldes, amasadores o varios de ellos.
- **Fuego y quemador –** El fuego siempre lastimará a Ricken si lo toca, pero no permanece encendido por mucho tiempo; para esto están los quemadores, que seguirán encendiéndose en periodos aleatorios.
- **Topo lanza-piedras –** Topo cobarde que se esconde en su hoyo y arroja piedras que ruedan hacia Ricken; estas piedras son el verdadero obstáculo pues varían en tamaño, velocidad y dureza haciendo que Ricken tenga problemas evitándolas o destruyéndolas. El topo como tal escapará bajo tierra tan pronto Ricken se acerque a su hoyo, en este momento la amenaza de rocas terminará.
- **Roca gigante –** Algunos topos antes de escapar para siempre pueden hacer aparecer una roca gigante que manda a volar a Ricken con su impacto en el suelo y rueda donde él de forma muy lenta pero segura sin que ningún obstáculo la detenga. No se puede evitar, la única alternativa es picotearla hasta romperla.
- **Cojín –** Obstáculo generalmente beneficioso que hace a Ricken saltar muy alto, usado para atravesar amenazas que no se podrían sobrepasar con un salto normal. Los enemigos tambien pueden usarlo no saber controlar este salto podría hacer caer a Ricken en las mismas amenazas que intentaba evadir.

Ítems:

- **Pluma –** Permite a Ricken recuperar un tercio de su salud si la toca, no tendrá ningún efecto si se tiene salud completa.
- **Llave –** Permite terminar el primer nivel abriendo la puerta de salida del granero.
- **Punto de control –** Invisible pero ubicado estratégicamente para lograr que el jugador no empiece desde el inicio del nivel siempre.

Escenarios:

- **Granja del Abuelo Joe –** Lugar donde se desarrolla todo el juego, todas las otras ubicaciones serán parte de ella. Se trata de una granja de gran tamaño, pero no tan

bien mantenida, razón por la que encontrar criaturas agresivas y obstáculos cortopunzantes es una ocurrencia común

- **Granero –** Construcción usada para almacenar paja y cajas de lo que sea que se necesite. Este es el escenario del primer nivel, donde Ricken aprenderá a lidiar con enemigos básicos y objetos que caen mientras practica sus saltos sin preocuparse por caer y lastimarse. La llave ubicada en el segundo piso se deberá obtener para finalizar este nivel.
- **Establo –** Amplio espacio cubierto de lodo y cercas de madera que están parcialmente destruidas. Este es el escenario del segundo nivel donde Ricken es introducido a los topos lanza-piedras y a los peligrosos espinos; Ricken deberá tener más control sobre sus saltos, de lo contrario caerá hasta una muerte segura.
- **Pradera –** Amplia extensión de terreno parte de la granja donde es común encontrar árboles de manzanas que no se han cosechado en año; es el escenario del tercer nivel donde Ricken tendrá que lidiar con una gran cantidad de manzanas que caen de los árboles y de los pájaros bombarderos que debutan en este nivel.
- **Casa del Abuelo Joe –** Casa ubicada junto al camino de entrada a la granja, inusualmente grande para ser ocupada solo por un anciano e infestada de enemigos debido al poco mantenimiento que se le da. Escenario del cuarto nivel donde Ricken tendrá que realizar saltos difíciles utilizando los cojines introducidos en este nivel mientras utiliza los muebles como plataformas para recorrer toda la casa hasta encontrar una salida en la cocina teniendo cuidado del fuego y cuchillos que caen; eso sin contar la alta concentración de enemigos presentes.

Gameplay:

El jugador controlará al protagonista Ricken the Chicken, quien buscará los huevos perdidos en las diferentes partes de la granja que servirán como niveles individuales y deberán ser atravesados lidiando con la variedad de enemigos y obstáculos presentes.

Ricken deberá evitar recibir daño 3 veces (representado por 3 plumas), de lo contrario se deberá empezar el nivel de nuevo o desde un punto de control. Para superar cada nivel, el jugador deberá usar al máximo las simples pero efectivas habilidades de Ricken que son:

- Saltar – Pese a sus flacas piernas, Ricken puede saltar muy alto, lo cual le permitirá alcanzar plataformas de una u otra forma y evadir la mayoría de los obstáculos.
- Correr – Cuando caminar no es suficiente, Ricken puede correr velozmente para recorrer más distancia en menos tiempo o aumentar la distancia recorrida en un salto.

- Picotear – El pico de Ricken es sorpresivamente fuerte, y será su modo de ataque contra enemigos a parte del aterrizaje. Su uso es obligatorio contra rocas, pues aterrizar sobre ellas causará daño.

El jugador deberá usar su intuición y aprender de sus errores para saber cómo lidiar con los enemigos y obstáculos. Los reinicios no serán ocurrencia frecuente si el jugador sabe cuidarse de las amenazas y derrotar a los enemigos eficientemente, pero se tendrá la libertad de intentar las veces que sean necesarias.

Un jugador promedio debería poder terminar el juego en una sola sesión, pero se podrá guardar al final de cada nivel si este no es el caso para poder continuar jugando en otra ocasión; el juego tambien se podrá pausar si se necesita un respiro.

Aspectos adicionales:

El juego tendrá un menú principal y pantallas previas a cada nivel, los botones en las diferentes pantallas permitirán al jugador continuar y salir del juego en cualquier momento.

Se tendrá música en todos los niveles y en las pantallas.

2 HISTORIAS DE USUARIO

Las historias de usuario generadas para este proyecto se muestran a continuación desde la Tabla I a la Tabla IX.

TABLA I Historia de usuario #1

HISTORIA DE USUARIO	
Identificador (ID): HU001	**Usuario:** Jugador
Nombre Historia: Ejecutar Juego	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Bajo
Iteración Asignada: 4	
Responsable: Mateo Borja	
Descripción: El jugador podrá ejecutar el video juego en su PC de sistema Windows haciendo click en el archivo ejecutable, el cual será proporcionado como parte de este proyecto.	
Observaciones:	

El archivo ejecutable estará ubicado en la carpeta descomprimida en la cual se distribuye el juego. Se contempla que el juego pueda funcionar todos los computadores Windows de 64 bits.

TABLA II Historia de usuario #2

HISTORIA DE USUARIO	
Identificador (ID): HU002	**Usuario:** Jugador
Nombre Historia: Iniciar Juego o Salir	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 2	
Responsable: Mateo Borja	
Descripción: El jugador podrá abrir el juego haciendo click en el ejecutable del mismo, una vez hecho esto, se mostrará la pantalla de título del juego, la cual incluirá opciones para iniciar una nueva partida, cargar una partida previa, mostrar créditos y salir del juego. Se deberá utilizar el cursor para seleccionar cualquiera de las opciones.	
Observaciones: Puede que sea necesario hacer click en la pantalla para que los botones se activen. Dependiendo del computador, puede haber pequeños tiempos de espera en la carga de niveles y menús.	

TABLA III Historia de usuario #3

HISTORIA DE USUARIO	
Identificador (ID): HU003	**Usuario:** Jugador
Nombre Historia: Mostrar Controles e Instrucciones	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Bajo
Iteración Asignada: 3	
Responsable: Mateo Borja	
Descripción: Como parte de la interfaz intuitiva, el usuario podrá ver las teclas que corresponden a cada acción del juego si pausa el mismo. De igual forma, pese a ser un juego sencillo, el jugador podrá ver un cuadro de texto con pistas en las escenas previas a cada nivel.	

Observaciones: El jugador deberá presionar la tecla P para pausar el juego en cualquier nivel, esta será la única forma de ver los controles del juego; aunque un jugador con experiencia debería poder descubrirlos por su cuenta.

TABLA IV Historia de usuario #4

HISTORIA DE USUARIO	
Identificador (ID): HU004	**Usuario:** Jugador
Nombre Historia: Realizar Acciones en el Juego	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Iteración Asignada: 2	
Responsable: Mateo Borja	
Descripción: El jugador podrá controlar al personaje protagonista del juego, el personaje podrá realizar las acciones necesarias que le permitirán al jugador terminar el juego sin problemas, estas acciones son: • Movimiento lateral • Saltar • Correr • Atacar	
Observaciones: La acción de atacar se usará para eliminar enemigos y romper ciertos obstáculos. Los enemigos también podrán ser eliminados si se aterriza sobre ellos luego de un salto. La acción de correr también funciona para acelerar al personaje en el aire, esto permite incrementar la distancia recorrida al saltar.	

TABLA V Historia de usuario #5

HISTORIA DE USUARIO	
Identificador (ID): HU005	**Usuario:** Jugador
Nombre Historia: Pausar Juego	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Media
Iteración Asignada: 3	
Responsable: Mateo Borja	

Descripción: El jugador podrá pausar el juego presionando la tecla “P”, al hacerlo se mostrarán las teclas de las diferentes acciones y una opción para salir a la pantalla de título.
Observaciones: Se deberá hacer click en el botón “Continuar” para volver al juego. Salir a la pantalla de título posiblemente también tendrá un pequeño tiempo de espera.

TABLA VI Historia de usuario #6

HISTORIA DE USUARIO	
Identificador (ID): HU006	**Usuario:** Jugador
Nombre Historia: Ver el Estado del Juego	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Baja
Iteración Asignada: 2	
Responsable: Mateo Borja	
Descripción: El jugador podrá ver en la esquina superior el estado del personaje, el cual en este juego se limitará a su salud. La cual se verá denotada por una pluma.	
Observaciones: La posición de la salud permanecerá fija en la pantalla sin importar el movimiento de esta, y será lo suficientemente grande para poder ser vista fácilmente sin obstruir el juego. Elementos con estas características que muestran el estado del juego se denominan “HUD” (Heads Up Display); los juegos de plataformas 2D suelen tener un HUD muy simple o incluso carecer de él.	

TABLA VII Historia de usuario #7

HISTORIA DE USUARIO	
Identificador (ID): HU007	**Usuario:** Jugador
Nombre Historia: Recibir Daño/ Morir	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	
Responsable: Mateo Borja	
Descripción:	

El personaje controlado por el jugador recibirá daño equivalente a un tercio de su salud total cada vez que haga contacto no favorable con algún enemigo u obstáculo de carácter peligroso. La salud del personaje se representará con una pluma, cada contacto no favorable cortará la pluma un tercio de su longitud inicial; en base a esto, el personaje contará con una pluma entera de salud al inicio de cada nivel. Si la pluma queda destruida luego de tres cortes, el jugador morirá y deberá empezar el nivel desde el inicio o desde un punto de control.
Observaciones: Se entiende como contacto no favorable a las ocasiones donde el personaje no está en posición de hacerle daño al enemigo; en este caso, cuando el contacto se produce fuera de la animación de ataque y dicho contacto no es un aterrizaje. Obstáculos de carácter peligroso deberán ser evitados, todo contacto con ellos (incluido ataque) resultará en daño al personaje. Cada vez que el personaje reciba daño, se mostrará una animación pertinente y se tendrán dos segundos de inmunidad, esto con el objetivo de que el jugador pueda reaccionar al peligro y no reciba daño varias veces seguidas; esta práctica está presente en todo juego de este género que cuenta con la posibilidad de recibir daño más de una vez antes de morir. En la mayoría de los casos el jugador debería ser capaz de intuir por su cuenta los obstáculos peligrosos, no se informará al jugador de esto por lo que tendrá que descubrirlo por prueba y error si su intuición no le alerta. Los niveles tendrán plumas que se pueden recolectar para recuperar el equivalente a un tercio de la salud total, recolectar plumas cuando la salud del personaje está completa no tendrá ningún efecto.

TABLA VIII Historia de usuario #8

HISTORIA DE USUARIO	
Identificador (ID): HU008	**Usuario:** Jugador
Nombre Historia: Progresar en el Juego	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Mateo Borja	
Descripción: El jugador podrá avanzar al siguiente nivel si llega al final del nivel actual, el jugador terminará el nivel automáticamente al llegar al final de este, se mostrarán pantallas	

para progresar con la historia, dar pistas y permitir que el jugador salga del nivel; si se decide continuar, el personaje aparecerá al inicio del siguiente nivel.
Observaciones: Puede que haya tiempos de espera entre niveles o para mostrar las pantallas de la historia. El juego está contemplado para que tenga una progresión sencilla, es decir, el jugador solo deberá terminar los niveles y hacer click en el botón para continuar. Cuando el jugador termine el último nivel y vea la última escena, el juego terminará y se podrán ver los créditos o regresar a la pantalla de título.

TABLA IX Historia de usuario #9

HISTORIA DE USUARIO	
Identificador (ID): HU009	**Usuario:** Jugador
Nombre Historia: Guardar la Partida	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 3	
Responsable: Mateo Borja	
Descripción: El jugador podrá guardar su partida para continuar jugando en otra ocasión sin necesidad de empezar el juego de nuevo. La función de guardar la partida será realizada de forma automática al finalizar un nivel, subsecuentes inicios del juego se podrán empezar desde el nivel siguiente al último nivel terminado.	
Observaciones: Este será un juego corto, los jugadores con media o abundante experiencia en los video juegos deberían poder completar el juego completo en una hora o menos, por lo que podrían no tener la necesidad de guardar la partida. Jugadores novatos o sin experiencia deberían terminar el juego en dos o tres horas, estos podrían o no necesitar guardar la partida, todo dependerá de su preferencia personal en cuanto al tiempo continuo que quieran permanecer jugando.	

3 PLAN DEL PROYECTO

La Tabla X muestra el plan del proyecto con todas las actividades generales a realizar.

TABLA X Plan del proyecto

Plan de Proyecto				
HU-ID	**Actividad**	**Iteración**	**Estado**	**Prioridad**
TODAS	Conceptualizar juego	1	Finalizado	Alta
N/A	Diseñar elementos	1	Finalizado	Media
HU004, HU007	Animar elementos	1	Finalizado	Media
HU004, HU007	Codificar elementos	2	Finalizado	Alta
N/A	Construir niveles	2	Finalizado	Alta
HU006, HU008	Codificar aspectos adicionales en niveles	2	Finalizado	Alta
HU002	Diseñar e implementar pantalla de título	2	Finalizado	Media
HU008	Diseñar e implementar escenas y transiciones	2	Finalizado	Media
HU003, HU005	Implementar función y menú de pausa	3	Finalizado	Baja
HU008	Juntar niveles, escenas y transiciones	3	Finalizado	Alta
HU009	Implementar función de guardado / carga de partidas	3	Finalizado	Media
TODAS	Realizar pruebas de jugabilidad	4	Finalizado	Alta
TODAS	Lanzar versión beta y realizar pruebas exhaustivas	4	Finalizado	Alta
TODAS	Corregir defectos	4	Finalizado	Alta
HU001	Crear archivo ejecutable del juego	4	Finalizado	Alta
TODAS	Lanzar versión final	4	Finalizado	Alta

4 SPRINT BACKLOG

La Tabla XI muestra el sprint backlog con las tareas específicas a realizar en cada Sprint.

TABLA XI Sprint Backlog

Sprint Backlog					
SB-ID	**Nombre**	**HU-ID**	**Historia de Usuario**	**Tareas**	**Tiempo Estimado**
SB001	Conceptualización, Configuración y Diseño	N/A	N/A	• Configurar el ambiente de desarrollo • Autoeducación en el uso de Unreal Engine	1 semana
		Todas	Todas	• Conceptualizar el juego	
		N/A	N/A	• Diseñar personajes, enemigos y obstáculos • Obtener escenarios, fondos y elementos adicionales • Obtener música	
		HU004, HU007	Realizar acciones en el juego, recibir daño/ morir	• Animar acciones de personaje principal • Animar enemigos	
SB002	Codificación y Construcción	HU004, HU007	Realizar acciones en el juego, recibir daño/ morir	• Codificar acciones de personaje principal • Codificar enemigos • Codificar interacciones entre personajes y enemigos / obstáculos • Codificar daño y muerte de personaje principal • Codificar muerte / desaparición de enemigos / obstáculos	8 semanas

		N/A	N/A	• Construir layout y terreno de niveles • Añadir enemigos, obstáculos y elementos a niveles • Añadir fondos y música a niveles	
		HU006, HU008	Ver estado del juego, Progresar en el juego	• Diseñar e implementar HUD • Calibrar interacciones en niveles	
		HU002	Iniciar o salir del juego	• Diseñar Logo del juego • Diseñar e implementar pantalla de título con menú de inicio	
		HU008	Progresar en el juego	• Diseñar escenas y transiciones • Añadir sonido a escenas y transiciones	
SB003	Agrupación y Funciones Adicionales	HU003, HU005	Mostrar controles e instrucciones, Pausar juego	• Diseñar e implementar menú de pausa • Calibrar pausa dentro del juego y botón de pausa	2 semanas
		HU008	Progresar en el juego	• Incorporar escenas de inicio en sus niveles respectivos • Implementar finalización de niveles	
		HU009	Guardar la partida	• Implementar función de guardado al final de los niveles • Implementar función de carga en el menú de inicio	
SB004	Pruebas, Depuración y Lanzamiento	Todas	Todas	• Lanzar versión beta • Probar menú de inicio del juego • Probar acciones de personaje	1 semana

				• Probar interacción con enemigos / obstáculos / entorno, muerte • Verificar comportamiento, muerte de enemigos • Verificar escenas y transiciones • Probar función de guardado y carga • Probar el juego completo • Lanzar versión final	
		HU001	Ejecutar juego	• Crear ejecutable de la versión final	

5 ASSETS

En esta sección se muestran los objetos usados en el videojuego. Solo se muestran aquellos que tiene sentido mostrar en forma de imagen, de igual manera, si existe el mismo *asset* en dos formatos distintos, se elige mostrar el que sea más conveniente.

5.1 *Sprites*

Acciones del personaje principal

De la Fig. 1 a la Fig. 6 se muestran las *spritesheets* del personaje principal

Fig. 1: Ricken parado

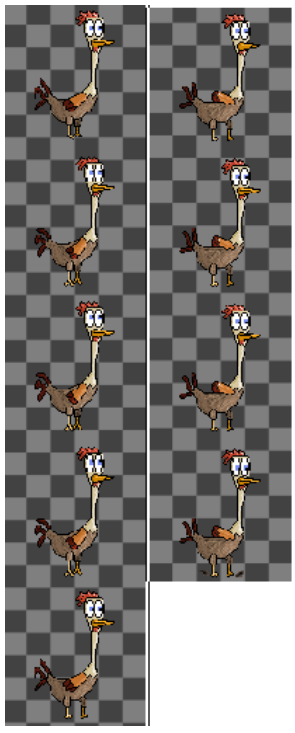

Fig. 2 Ricken saltando

Fig. 3: Ricken caminando

Fig. 4: Ricken corriendo

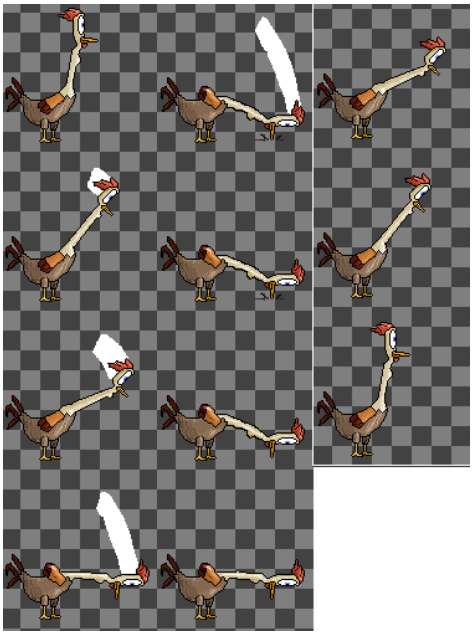

Fig. 5: Ricken atacando

Fig. 6: Ricken lastimado

Enemigos

De la Fig. 7 a la Fig. 9 se muestran las *spritesheets* de los enemigos

Fig. 7: Nosey Mousey

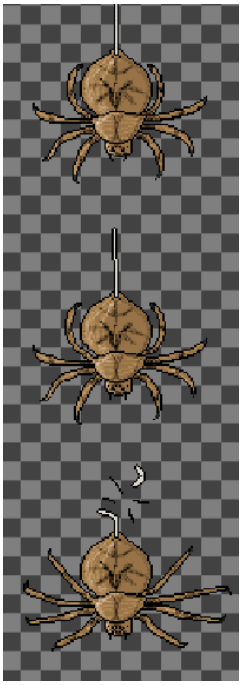

Fig. 8: Betsy Spider

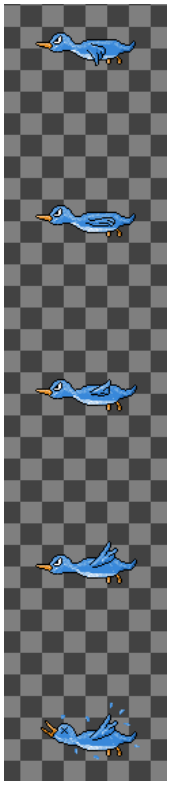

Fig. 9: Bombird

Obstáculos y objetos interactivos

De la Fig. 10 a la Fig. 19 se muestran las *spritesheets* y *sprites de* todos los objetos que pueden interactuar con el jugador.

Fig. 10: Espino

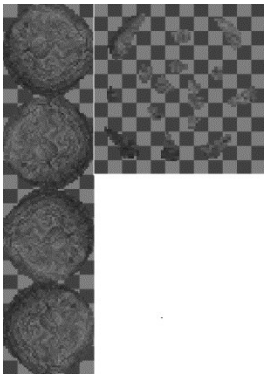

Fig. 11: Roca

Fig. 12: Fuego

Fig. 13: Pluma

Fig. 14: Llave

Fig. 15: Manzana

Fig. 16: Amasador

Fig. 17: Balde

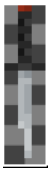

Fig. 18: Cuchillo

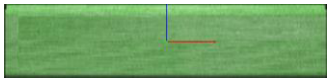

Fig. 19: Cojín

Objetos del nivel

De la Fig. 20 a la Fig. 26 se muestran algunos de los objetos decorativos de los niveles que se utilizaron en forma de *sprites.* Por simplicidad, si varios *assets* son similares, se muestra solo uno de ellos.

Fig. 20: Árbol

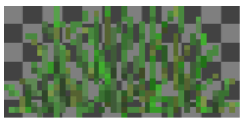

Fig. 21: Pasto

Fig. 22: Planta

Fig. 23: Planta en maceta

Fig. 24: Botella

Fig. 25: Lámpara

Fig. 26: Bota

Fondos

De la Fig. 27 a la Fig. 34 se muestran los *sprites* de gran tamaño usados como fondos de los niveles.

Fig. 27: Cielo despejado

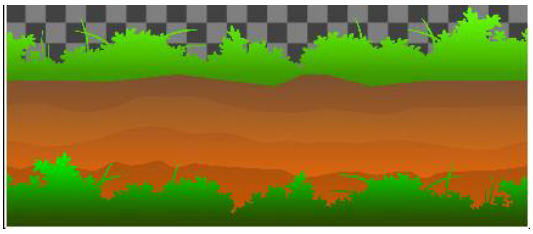

Fig. 28: Terreno con vegetación

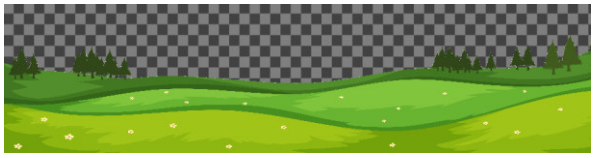

Fig. 29: Colinas y árboles

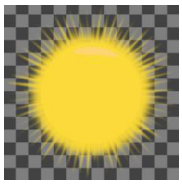

Fig. 30: Sol

Fig. 31: Cielo de atardecer

Fig. 32: Cielo nocturno

Fig. 33: Terreno seco

Fig. 34: Luna llena

5.2 Terreno

Tilesets

De la Fig. 35 a la Fig. 63 se muestran las texturas trabajadas como *tilesets* para construir los niveles.

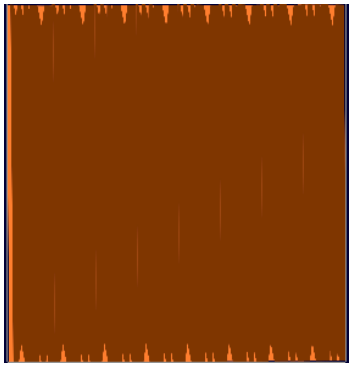

Fig. 35: Pared de granero

Fig. 36: Pared pintada

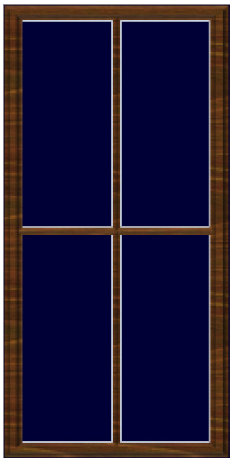

Fig. 37: Ventana

Fig. 38 Puerta

Fig. 39: Pared de madera

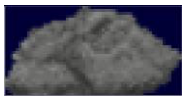

Fig. 40: Roca

Fig. 41: Piso de madera

Fig. 42: Piso de baldosa

Fig. 43: Tronco

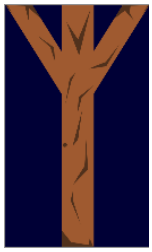

Fig. 44: Columna de madera

Fig. 45: Columna de cerca

Fig. 46: Viga de cerca

Fig. 47: Caja de madera

Fig. 48: Paja

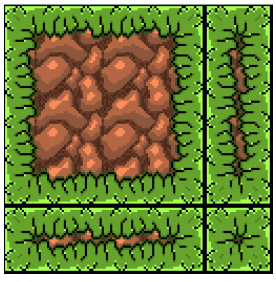

Fig. 49: Terreno con césped

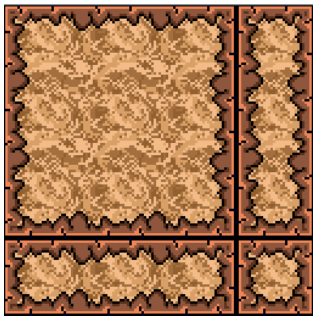

Fig. 50: Terreno árido

Fig. 51: Lodo

Fig. 52: Mesa de barril

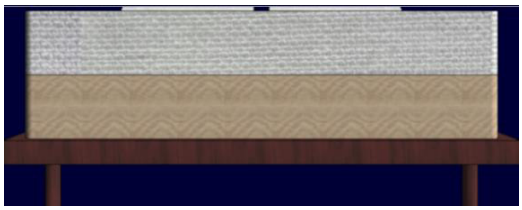

Fig. 53: Cama

Fig. 54: Silla

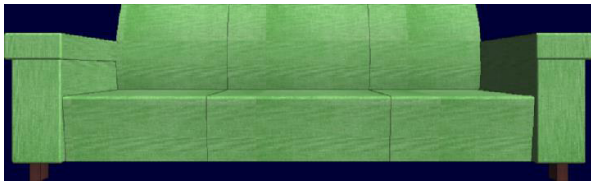

Fig. 55: Sillón

Fig. 56: Mesilla

Fig. 57: Escritorio

Fig. 58: Banco

Fig. 59: Reloj de péndulo

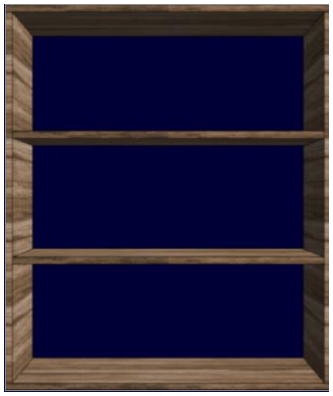

Fig. 60: Repisa

Fig. 61: Cocina

Fig. 62: Mesa

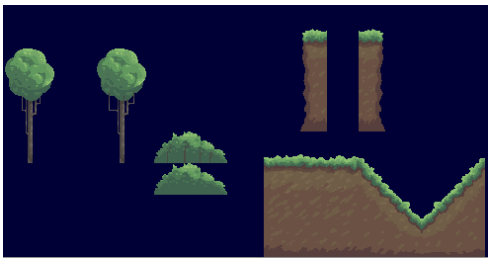

Fig. 63: Árboles y terreno

Tilemaps

De la Fig. 64 a la Fig. 67 se muestra el terreno de los niveles construido en forma de *Tilemaps.*

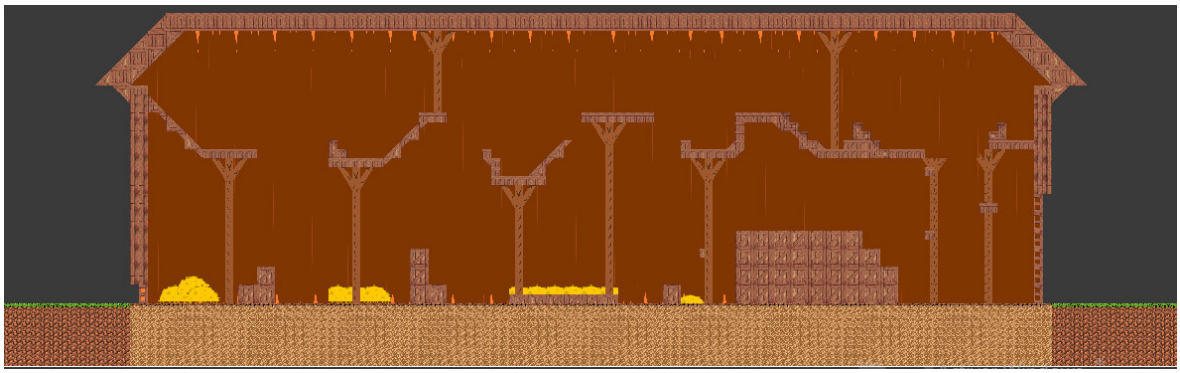

Fig. 64: Terreno del Nivel 1

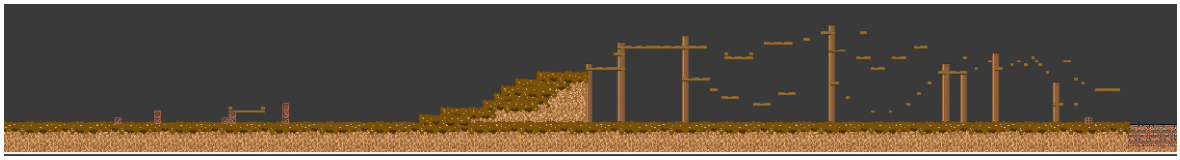

Fig. 65: Terreno del Nivel 2

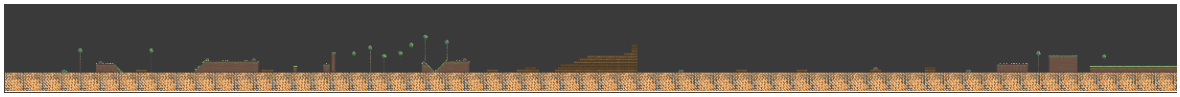

Fig. 66: Terreno del Nivel 3

Fig. 67: Terreno del Nivel 4

5.3 Pantallas y botones

De la Fig. 68 a la Fig. 76 se muestran las pantallas y el formato de los botones usados en el juego.

Fig. 68: Formato de botones por defecto

Fig. 69: Formato de botones al pasar el cursor por encima

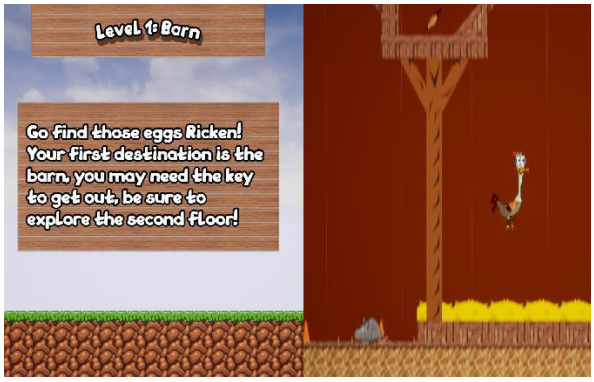

Fig. 70: Pantalla previa al primer nivel

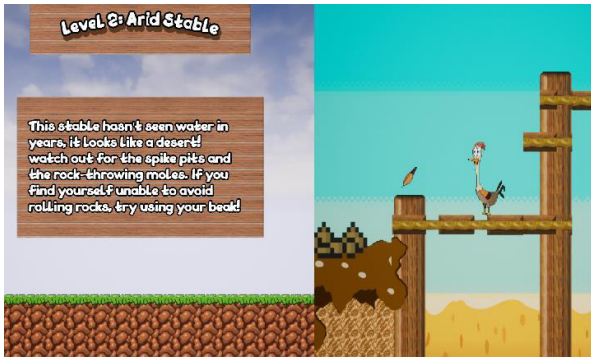

Fig. 71: Pantalla previa al segundo nivel

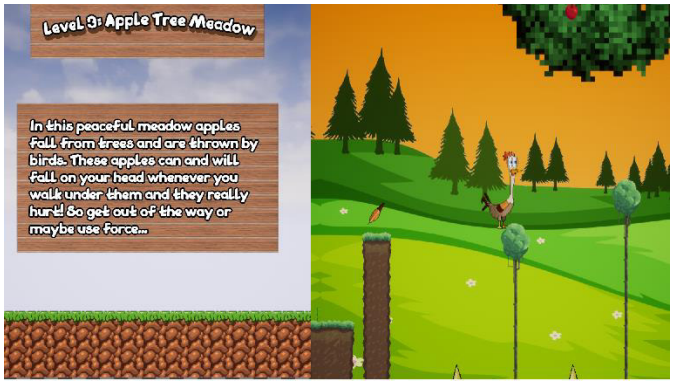

Fig. 72: Pantalla previa al tercer nivel

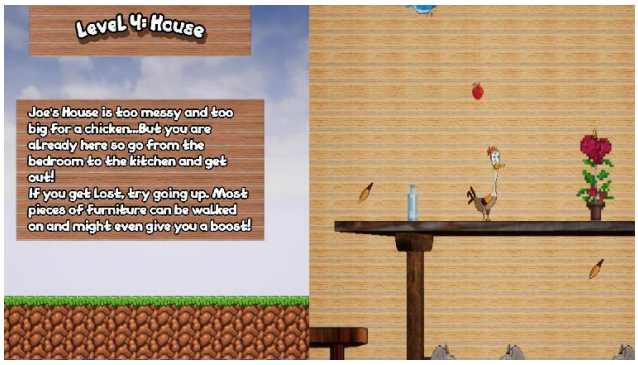

Fig. 73: Pantalla previa al cuarto nivel

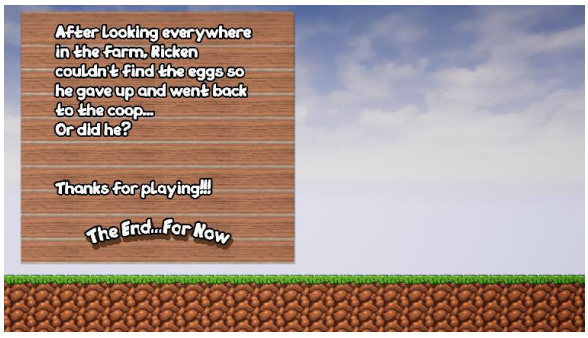

Fig. 74: Pantalla final

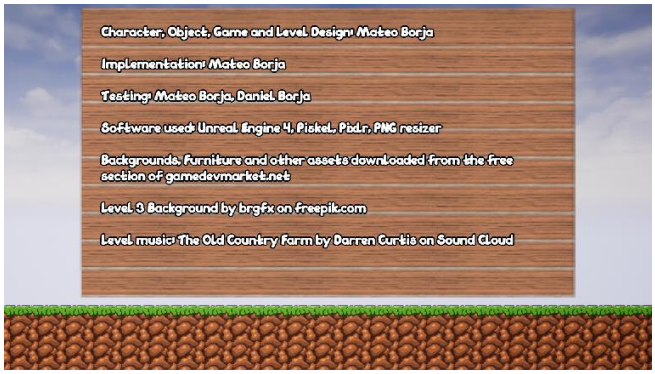

Fig. 75: Pantalla de créditos

Fig. 76: Teclas para menú de pausa

6 PRUEBAS

6.1 Aceptación

En esta sección se explican a detalle las pruebas realizadas como parte del *sprint* final, para esto se utiliza un formato estándar repetido en cada prueba. Las pruebas de aceptación de este proyecto se muestran desde la Tabla XII hasta la Tabla

TABLA XII Prueba de aceptación #1

PRUEBA DE ACEPTACIÓN	
Identificador de Prueba (ID): PA001	**Identificador de Historia de Usuario:** HU002
Nombre de la Prueba de Aceptación: Probar menú de inicio del juego	
Descripción: El jugador puede ver el menú de inicio al iniciar el juego y utilizar todos sus botones.	
Condiciones de la Prueba: • Ejecutar el juego • Hacer click en la pantalla del juego • Tener salida de audio	
Pasos de Ejecución: • Observar la pantalla que se muestra al abrir el juego • Hacer click en el botón para iniciar nueva partida • Hacer click en el botón para cargar partida • Hacer click en el botón de créditos • Hacer click en el botón para salir del juego	

Resultado Deseado: El menú de inicio debe ser la primera pantalla en aparecer al iniciar el juego, debe tener música de fondo y sus botones se debe poder presionar para iniciar el juego, continuarlo, ir a créditos o salir del juego. Los botones deben cambiar de color si se pasa el cursor sobre ellos.
Evaluación de la Prueba: El resultado obtenido es igual al deseado, aprobación: 100%

TABLA XIII Prueba de aceptación #2

PRUEBA DE ACEPTACIÓN	
Identificador de Prueba (ID): PA002	**Identificador de Historia de Usuario:** HU004
Nombre de la Prueba de Aceptación: Probar acciones del personaje	
Descripción: El jugador puede presionar las respectivas teclas para que el personaje realice las acciones que corresponden a cada una de ellas.	
Condiciones de la Prueba: • Ejecutar el juego • Hacer click en la pantalla del juego • Estar dentro de un nivel	
Pasos de Ejecución: • Presionar las flechas izquierda y derecha • Presionar la tecla C • Presionar la tecla Z • Presionar la tecla X • Presionar la tecla P • Segundos luego de morir, presionar la barra espaciadora • Presionar las teclas anteriores luego de recibir daño • Presionar dos o más de las teclas anteriores al mismo tiempo	
Resultado Deseado: las teclas correspondientes a las acciones del jugador son: flechas derecha e izquierda para moverse, Z para correr, X para saltar, C para atacar, P para pausar y espacio para revivir. Al presionar cada tecla se debe ver la animación respectiva de la acción, estas animaciones no se pueden reproducir simultáneamente, por lo que una acción debe cancelar a la anterior o impedir que se realice otra hasta que haya finalizado; la	

excepción a esto es la animación de daño, la cual dura durante el periodo de invencibilidad y reemplaza al resto de animaciones durante este periodo.
Evaluación de la Prueba: El resultado obtenido es igual al deseado, pero tal vez el jugador no debería poder atacar mientras se encuentra herido, aprobación: 90%

TABLA XIV Prueba de aceptación #3

PRUEBA DE ACEPTACIÓN	
Identificador de Prueba (ID): PA003	**Identificador de Historia de Usuario:** HU004, HU006, HU007
Nombre de la Prueba de Aceptación: Probar interacción con enemigos / obstáculos / entorno, muerte	
Descripción: El jugador puede tener interacciones favorables y no favorables con los enemigos y obstáculos, puede morir si su salud llega a 0 y ver esto reflejado en el HUD, puede revivir luego de morir.	
Condiciones de la Prueba: • Ejecutar el juego • Hacer click en la pantalla del juego • Pasar por todos los niveles	
Pasos de Ejecución: • Aterrizar sobre un enemigo de cada tipo • Atacar un enemigo de cada tipo • Permitir que un enemigo de cada tipo entre en contacto con el jugador • Tocar un obstáculo y objeto de cada tipo • Prestar atención al HUD al recibir daño • Recibir daño 3 veces seguidas • Presionar la barra espaciadora para revivir	
Resultado Deseado: El jugador debe infligir daño a los enemigos si colisiona con estos durante la animación de ataque o aterriza sobre ellos y recibir daño en cualquier otra colisión. Los obstáculos deben interactuar con el jugador de acuerdo con su función de interacción. El jugador debe morir siempre que su salud llegue a cero, reflejado por la pluma del HUD quedando destruida luego de recibir daño 3 veces; una vez que el jugador muere, su personaje	

caerá a través del terreno y el jugador podrá presionar la tecla correspondiente para revivir al inicio del nivel o en un punto de control con su salud reestablecida.
Evaluación de la Prueba: El resultado obtenido es igual al deseado, aprobación: 100%

TABLA XV Prueba de aceptación #4

PRUEBA DE ACEPTACIÓN	
Identificador de Prueba (ID): PA004	**Identificador de Historia de Usuario:** HU007
Nombre de la Prueba de Aceptación: Verificar comportamiento, muerte de enemigos	
Descripción: Los enemigos pueden moverse de manera adecuada y morir al ser atacados por el jugador.	
Condiciones de la Prueba: • Ejecutar el juego • Hacer click en la pantalla del juego • Pasar por todos los niveles	
Pasos de Ejecución: • Observar el movimiento de un enemigo de cada tipo • Infligir daño en un enemigo de cada tipo • Observar la interacción de dos enemigos del mismo tipo • Observar la interacción de dos enemigos de diferente tipo • Observar la interacción de enemigos y obstáculos / objetos • Permanecer cerca del lugar donde un enemigo murió por más de 20 segundos	
Resultado Deseado: Todos los personajes y objetos deben estar sujetos a colisiones con el terreno y deben poder navegar a través de este sin quedar atascados y sin presentar comportamientos erráticos; los enemigos deben colisionar entre ellos, pero no recibir daño de esta colisión. Los objetos interactivos deben reaccionar únicamente al jugador, no a enemigos u otros objetos, y tampoco deben colisionar entre ellos o con enemigos (existen excepciones). Cada enemigo morirá luego de recibir daño del jugador una vez y generará una copia de sí mismo en su posición inicial luego de un intervalo de tiempo desde su muerte.	
Evaluación de la Prueba: El resultado obtenido es igual al deseado con excepción de que algunos enemigos parecen no reaparecer el 100% de las veces que mueren, esto sin embargo no afecta la experiencia de juego. Aprobación: 100%	

TABLA XVI Prueba de aceptación #5

PRUEBA DE ACEPTACIÓN	
Identificador de Prueba (ID): PA005	**Identificador de Historia de Usuario:** HU005, HU008
Nombre de la Prueba de Aceptación: Verificar escenas y transiciones	
Descripción: El jugador puede ver pantallas previas a cada nivel, pantalla de créditos y menú de pausa si se cumplen las respectivas condiciones.	
Condiciones de la Prueba: • Ejecutar el juego • Hacer click en la pantalla del juego • Tener salida de audio • Pasar por todos los niveles	
Pasos de Ejecución: • Iniciar una nueva partida • Terminar el nivel 1 • Terminar el nivel 2 • Terminar el nivel 3 • Terminar el nivel 4 • Presionar la tecla P en cualquier nivel • Presionar el botón de créditos en la pantalla final • Acceder al menú principal desde cada una de las pantallas	
Resultado Deseado: Las pantallas de transición deben tener música y botones que permiten avanzar al nivel correspondiente o salir al menú principal, estas pantallas siempre se deben mostrar antes de un nivel, sin importar de qué forma se acceda a dicho nivel. La pantalla final solo se podrá acceder al terminar el juego y tendrá conexiones con el menú principal y los créditos; los créditos se pueden acceder desde esta pantalla o el menú de inicio y solo tienen conexión con este último.	
Evaluación de la Prueba: El resultado obtenido es igual al deseado, aprobación: 100%	

TABLA XVII Prueba de aceptación #5

PRUEBA DE ACEPTACIÓN	
Identificador de Prueba (ID): PA005	**Identificador de Historia de Usuario:** HU009
Nombre de la Prueba de Aceptación: Probar función de guardado y carga	
Descripción: El jugador puede iniciar desde el próximo nivel al último que termino, puede iniciar una nueva partida en cualquier momento.	
Condiciones de la Prueba: • Ejecutar el juego • Hacer click en la pantalla del juego • Pasar por todos los niveles	
Pasos de Ejecución: • Iniciar una nueva partida • Volver a la pantalla de inicio luego de terminar cada nivel • Hacer click en el botón cargar partida • Iniciar una nueva partida en cualquier momento del juego	
Resultado Deseado: El progreso del juego se guarda luego de cada nivel sin intervención del jugador y de forma completamente transparente para él. La función de carga debe regresar al jugador a la pantalla de introducción del nivel próximo al último nivel terminado, o a la pantalla final si ha terminado el último nivel. En cualquier momento se puede empezar una nueva partida, lo cual sobrescribe el progreso guardado anterior.	
Evaluación de la Prueba: El resultado obtenido es igual al deseado, aprobación: 100%	

6.2 Compatibilidad

En la Tabla XVIII se muestran los resultados de probar el videojuego en computadores de diferentes características.

TABLA XVIII Resultados de pruebas de compatibilidad

Computador	Prestaciones	Desempeño del juego	Observaciones
Laptop Toshiba	Medias, gráficos integrados	Jugable, pero con pequeñas fallas gráficas	Este equipo se usó para el desarrollo del proyecto

Laptop Sony Vaio	Altas, gráficos Nvidia	Perfecto, Animaciones y gráficos se aprecian de forma nítida	El juego consume más recursos de lo esperado.
PC MSI	Altas, gráficos AMD	Perfecto, Animaciones y gráficos se aprecian de forma nítida	Este equipo funciona con dos monitores, el juego se puede ejecutar en cualquiera de ellos.
PC Taurus	Bajas, gráficos Nvidia de muy baja gama	No jugable, varias fallas gráficas y número de cuadros por segundo por debajo del límite mínimo	Esta PC es muy antigua, tiende a sobrecalentarse o congelarse con tareas mucho menos intensas.

ANEXO III

Observar en el siguiente link el video realizado como manual de usuario donde se muestra como acceder al juego y las principales funciones del mismo.

https://youtu.be/IhOVor_GpsI

ANEXO IV

Pasos para iniciar el juego como jugador:

1. Descargar la carpeta del siguiente link:

   https://drive.google.com/drive/folders/1i0yPTFeUWXv_lDIfJ1QkUIxeguPq8xyM?usp=sharing

2. Abrir la carpeta y entrar a la subcarpeta *PackagedGame*

3. Abrir la subcarpeta *WindowsNoEditor*, en su interior se debería visualizar los archivos mostrados en la Fig. 1

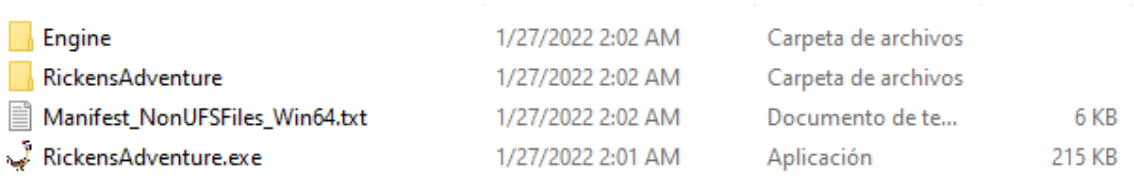

Fig. 1: Archivos del juego

4. Hacer doble click en el archivo *RickensAdventure.exe*, esto abrirá el juego

5. El juego iniciará en el menú principal, desde donde se podrá iniciar una nueva partida o continuar una partida anterior, en la Fig. 2 se muestra este menú principal.

Fig. 2: Menú principal del juego

6. En cualquier momento del juego se puede presionar la tecla P para acceder al menú de pausa y visualizar los controles del juego.

Pasos para iniciar el juego como desarrollador:

1. Descargar la misma carpeta del link anterior

2. Descargar e instalar Unreal Engine de su página oficial, se requiere una versión igual o superior a la que se usó para crear el proyecto (4.24).

3. En la carpeta descargada, identificar el archivo del proyecto. Si Unreal Engine está instalado, el archivo tendrá su logo y se abrirá automáticamente con el editor. Este archivo se muestra en la Fig. 3

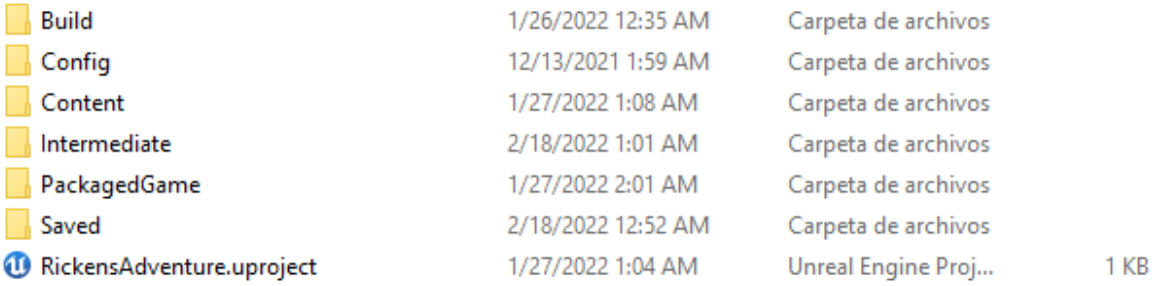

Fig. 3: Archivo del proyecto en la carpeta

4. Una vez que se abre este proyecto por primera vez, este aparecerá en la lista de proyectos recientes al abrir Unreal Engine y se puede acceder desde allí como se muestra en la Fig. 4

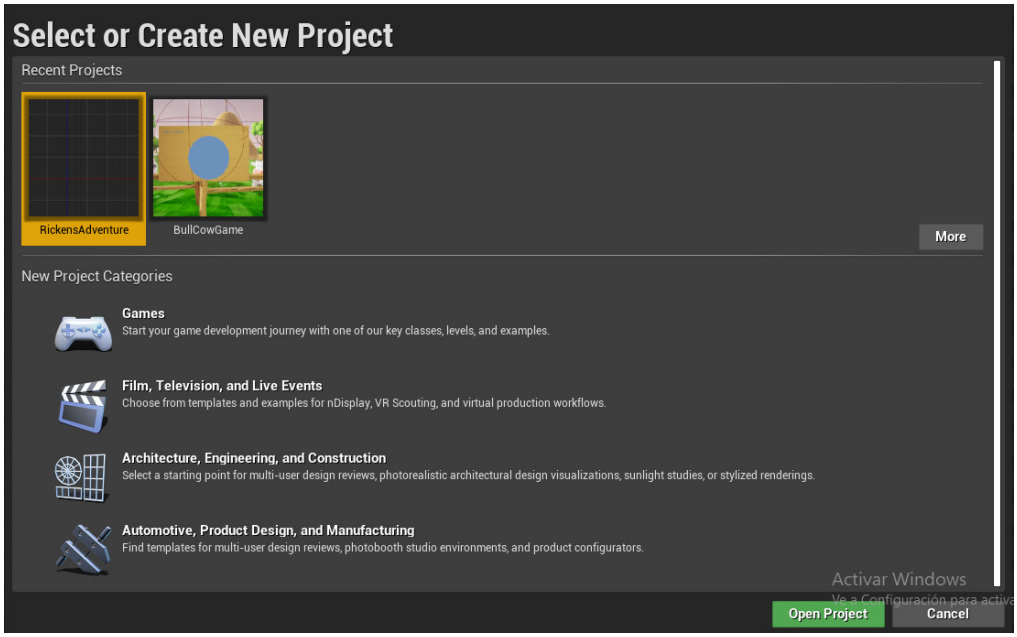

Fig. 4: Proyecto en Unreal Engine

5. En el editor se podrá revisar el juego, mantenerlo y modificarlo. Tomar en cuenta que el uso de Unreal Engine no es aparente, por lo que se recomienda seguir tutoriales antes de modificar cualquier proyecto.

Nota: Por simplicidad, la carpeta provista contiene tanto el ejecutable del juego como su proyecto, esto no debería hacerse si se desea publicar el juego, en cuyo caso se debería enviar únicamente la carpeta con el ejecutable. Los jugadores que accedan a la carpeta provista pueden y deberían ignorar todos los archivos fuera de la carpeta *PackagedGame.*