UNIVERSIDAD ANDINA DEL CUSCO

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

INVESTIGACIÓN DE TIPO APLICADA

PLUGIN PARA EL GUARDADO DE PARTIDAS DE VIDEOJUEGOS USANDO EL SISTEMA DE PROGRAMACIÓN VISUAL DE UNREAL ENGINE 4

Presentado por:

Bach. Kevin Yabar Garces.

Bach. Ronald Jesús Ochoa Cuba.

Para optar por el título de:

Ingeniero de Sistemas.

Asesor:

Mg. Iván Molero Delgado.

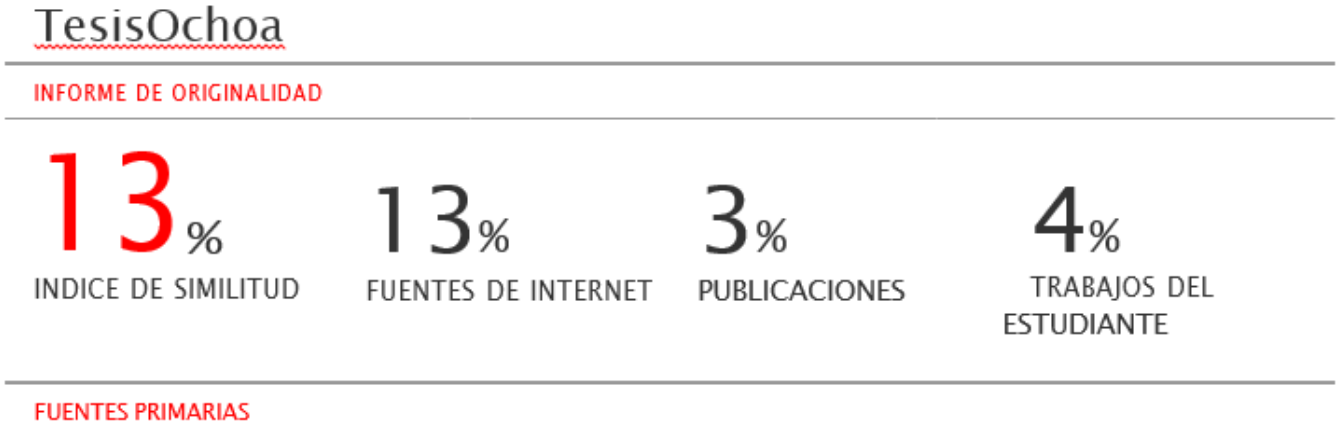
CUSCO-PERÚ

2021

PLUGIN PARA EL GUARDADO DE PARTIDAS DE VIDEOJUEGOS USANDO EL SISTEMA DE PROGRAMACION VISUAL DE UNREAL ENGIE 4, TIENE UN 13% DE SIMILIUD SEGÚN EL INFORME DE ORIGINALIDAD CON RESPECTO A OTROS TRABAJOS.

Ochoa y Yabar

Fecha de entrega: 29-jun-2023 10:11p.m. (UTC-0500)
Identificador de la entrega: 2124588105
Nombre del archivo: TESISOCHOA.pdf (4.99M)
Total de palabras: 21219
Total de caracteres: 121605



TesisOchoa

INFORME DE ORIGINALIDAD

13%	13%	3%	4%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	repositorio.uandina.edu.pe Fuente de Internet	2%
2	hdl.handle.net Fuente de Internet	2%
3	upc.aws.openrepository.com Fuente de Internet	1%
4	Submitted to tec Trabajo del estudiante	1%
5	www.coursehero.com Fuente de Internet	<1%
6	repository.eafit.edu.co Fuente de Internet	<1%
7	repositorio.uss.edu.pe Fuente de Internet	<1%
8	qdoc.tips Fuente de Internet	<1%
9	uvadoc.uva.es Fuente de Internet	<1%

Dedicatoria

Dedicamos esta tesis a nuestra familia, amigos y personas especiales en nuestra vida que nos apoyaron durante tanto tiempo para conseguir el logro de terminar nuestros estudios y obtener nuestro título profesional. No podríamos sentirnos más felices sobre la confianza puestas en nosotros, por lo que les dedicamos este trabajo con mucho cariño.

Agradecimientos

Agradecemos a nuestros docentes y a la Universidad Andina del Cusco por brindarnos un entorno de educación de calidad con el cual conseguimos llegar tan lejos como personas y profesionales. Asimismo, agradecemos a nuestra familia por confiar y apoyarnos en todo momento. Llegar tan lejos no habría sido posible sin ninguno de ustedes.

Resumen

La presente investigación aplicada consiste en el desarrollo de un plugin para el motor de videojuegos Unreal Engine 4. Este motor de videojuegos es un software enfocado en el desarrollo de videojuegos y que cuenta con un diseño modular que permite la integración de plugins (extensiones) desarrollados por terceros.

En esta investigación, se busca crear un plugin para Unreal Engine 4 que tiene como fin mejorar el proceso de implementación de un sistema de guardado y cargado de partidas para videojuegos. De esta forma, se busca que el plugin se integre al motor de videojuegos y permita a desarrolladores de videojuegos implementar sus propios sistemas de guardado y cargado de partidas de acuerdo a los requerimientos específicos que pueda tener cada proyecto.

Es importante, sin embargo, tener en cuenta que Unreal Engine 4 ya incluye por defecto herramientas de guardado y cargado de partidas para videojuegos. Por ello, este trabajo pretende desarrollar una alternativa a las herramientas por defecto mencionadas; una alternativa que se distribuya en forma de plugin y que esencialmente cuente con una calidad superior.

Por tanto, resulta importante determinar cómo el plugin influye en la implementación de un sistema de guardado y cargado partidas con respecto a las herramientas por defecto de Unreal Engine 4 y, específicamente, el determinar si el plugin desarrollado cuenta con una mejor calidad. Por ello, en esta tesis se hace uso de la norma ISO/IEC 9126 para la evaluación calidad de software, ya que esta norma permite tanto evaluar como comparar la calidad del plugin desarrollado con respecto a las herramientas por defecto de Unreal Engine 4.

Abstract

The following research consists on the development of a plugin for the game engine Unreal Engine 4. This game engine is a software oriented in creating videogames that supports a modular design that is able to integrate plugins created by a third party.

During this research, the goal is to create a plugin for Unreal Engine 4. This plugin has to enhance the process of implementation of a save and load system for videogames. This way, the plugin is able to integrate to the game engine and this, in turn, allows game developers to implement their own save and load systems according to their project requirements.

It's important, however, to consider that Unreal Engine 4 already includes save and load tools by default. Furthermore, this work tries to develop an alternative to the default tools; an alternative that is distributed in the shape of a plugin and that essentially has a superior quality.

It's important to determine how the plugin affects the implementation of a save and load system with respect to the default tools included in Unreal Engine 4 and, specifically, to determine if the developed plugin has a superior quality. For this reason, the ISO/IEC 9126 standard for the evaluation of software quality is used during this research. This standard allows to evaluate and compare the quality of the plugin with respect to the default tools included in Unreal Engine 4.

Introducción

El presente documento se enfoca en el proceso de desarrollo de un plugin para el guardado de partidas de juego en el motor de videojuegos Unreal Engine 4. Asimismo, se analiza las herramientas ya existentes en Unreal Engine 4 con respecto al guardado y cargado de partidas de juegos. Todo esto con el fin de mejorar la calidad de las herramientas orientadas al guardado y cargado de partidas de juego en Unreal Engine 4, calidad que se determina mediante el uso de la norma ISO/IEC 9126 para la evaluación de la calidad de un software.

Además, se hace uso de la metodología ágil AUP (Proceso ágil unificado) ya que se considera como una metodología flexible y aplicable a la presente investigación, tal como se describe en el marco teórico de la presente investigación.

Índice General

Índice de Tablas

Índice de Figuras

Capítulo Primero. Problema de la Investigación

1.1. Ámbito de Influencia

1.1.1. Ámbito de Influencia Teórica.

De acuerdo a Hilbert & López, la Tecnología de la Información (TI) se define como el uso o aplicación de los ordenadores para el almacenamiento, recuperación, transmisión y/o manipulación de datos (Hilbert & López, 2011). Se considera que esta tesis se ve influenciada por esta disciplina ya que el plugin a desarrollarse manipula los datos de partida de un videojuego.

De igual forma, las métricas internas y externas de calidad de software propuestas por la norma ISO/IEC 9126 resultan relevantes en esta investigación ya que establecen parámetros para medir y evaluar la calidad de un software (ISO/IEC, 2001).

Asimismo, la ingeniería de Software se define como la aplicación de un modelo o método durante los procesos de desarrollo de software (Somerville, 2015). Ya que se busca aplicar un modelo estándar (ISO 9126) para medir la calidad del software, se observa que esta tesis también se encuentra influenciada por esta disciplina.

1.1.2. Área de Dominio.

El dominio de esta investigación es el área de Tecnologías de la Información (TI).

1.1.3. Línea de Investigación.

La línea de investigación es el de Herramientas y Técnicas para el desarrollo de Sistemas.

1.2. Planteamiento del problema

1.2.1. Descripción de la Situación Actual.

Se observa que en la actualidad los videojuegos implementan diversas formas de sistema de guardado y cargado de partidas para satisfacer los estándares que los jugadores esperan de estos

productos. Estos sistemas permiten a los jugadores guardar el estado de la partida de juego en un momento determinado, así como restaurar el estado de la partida haciendo uso de uno de los datos guardados existentes.

Estos sistemas de guardado y cargado de partidas para videojuegos, aunque difieran de un videojuego a otro, pueden agruparse en dos categorías de acuerdo a su implementación: La primera consiste en una implementación de guardado manual, el cual requiere de acción por parte del jugador y, por tanto, suele detener la capacidad para controlar el juego mientras el guardado este en proceso. La segunda categoría consiste en una implementación de guardado de partidas automático, el cual no requiere de acción por parte del jugador y suele procesarse mientras el juego este siendo controlado por este. Ambas implementaciones usan un mecanismo de cargado de partidas similar ya que este se da en situaciones puntuales durante la partida de juego (Moran, 2010).

Tanto las implementaciones de guardado de partidas manuales como los automáticos suelen almacenar los datos de guardado en un archivo (en disco) dentro de un directorio en la plataforma donde el juego este siendo ejecutado. (Adams, 2014).

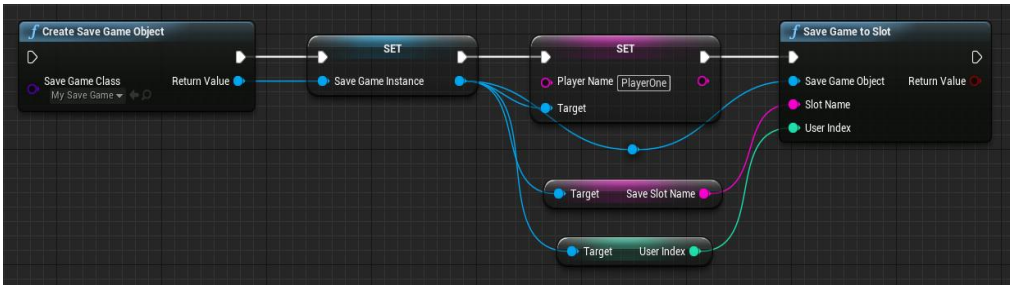


Figura: 1 Sistema de Guardado en Blueprints

Ejemplo simple de una implementación de guardado de juego en Unreal Engine 4. Este ejemplo muestra el guardado de la variable “Player Name”. Extraído de la documentación oficial de Unreal Engine 4 (Epic Games, 2018).

1.2.2. Descripción del Problema.

A partir de lo expuesto en la situación actual y de las métricas propuestas por la norma ISO/IEC 9126 de calidad de software, la cual esta descrita con mayor detalle en el marco teórico

de la investigación, se puede determinar que las herramientas de guardado y cargado de partidas de juego incluidas en el sistema de programación visual Blueprints de Unreal Engine 4 no cuentan con un nivel suficiente de calidad para satisfacer parte de los requerimientos introducidos en la sección anterior. De esta forma, se busca desarrollar un plugin compatible con Unreal Engine 4 de manera que se consiga mejorar la calidad de estas herramientas, usando la norma ISO/IEC 9126 como guía de desarrollo.

Por ejemplo, si se tiene un videojuego que requiera de un sistema de guardado y cargado de partidas de juego, en donde los archivos de guardado son almacenados en la nube y por tanto el sistema depende de una conexión a internet. Si se mide las herramientas de guardado incluidas en por defecto en Unreal Engine 4 con respecto a las métricas de eficiencia propuestas en el modelo ISO/IEC 9126, el resultado más probable será que las herramientas de guardado por defecto tendrán un desempeño deficiente en cuanto a su comportamiento en el tiempo, ya que estas no ofrecen una funcionalidad de compresión de datos (Epic Games, 2018). En cambio, si se desarrollara un plugin que busque hacer frente a este problema y, por ejemplo, se integrasen funcionalidades de compresión de datos en las herramientas de guardado, entonces podría concluirse que se ha mejorado la calidad del software en este aspecto.

1.2.3. Formulación del Problema.

¿Cómo influye el plugin en la calidad de las herramientas de guardado y cargado de partidas para videojuegos en Unreal Engine 4?

1.2.4. Objetivos.

1.2.4.1. Objetivo General.

Determinar la influencia del plugin en la calidad de las herramientas de guardado y cargado de partidas para videojuegos en Unreal Engine 4.

1.2.4.1. Objetivos Específicos.

a. Definir los requerimientos de un sistema de guardado y cargado de partidas para videojuegos.
b. Diagnosticar la calidad de las herramientas de guardado y cargado de partidas para videojuego incluidas en Unreal Engine 4.
c. Desarrollar un plugin de guardado y cargado de partidas para videojuegos en Unreal Engine 4.
d. Implementar pruebas funcionales para medir la calidad del plugin durante su desarrollo.
e. Comparar la calidad del plugin con respecto a las herramientas incluidas en Unreal Engine 4.

1.2.5. Justificación de la Investigación.

En cuanto a su aporte a la investigación, este trabajo busca generar nuevas tecnologías y conocimiento puesto que se busca mejorar la calidad de las herramientas de guardado y cargado de partidas de juego incluidas en Unreal Engine 4. Esto se logra mediante el desarrollo de un plugin compatible con Unreal Engine 4, de forma que se integren nuevas funcionalidades enfocadas al guardado y cargado de partidas de juego. Este plugin además cuenta con el código fuente disponible para cualquiera que desee adquirirlo por lo que el software desarrollado puede ser utilizadas como fuente de aprendizaje.

En cuanto a su aporte práctico, el plugin desarrollado en esta investigación permite a otros desarrolladores de videojuegos implementar sistemas de guardado y cargado de juego en sus proyectos con herramientas de una mayor calidad con respecto a las herramientas de guardado incluidas por defecto en Blueprints.

En el área social, en nuestra región existe cada vez un mayor interés por promover emprendimientos tecnológicos o de mercados vírgenes por lo que esta investigación resulta relevante ya que incursiona en un sector poco explorado en el país: el desarrollo de videojuegos (Acerca de Start up Perú, 2019).

En cuanto al aspecto personal, consideramos que este proyecto asienta aún más las bases de conocimiento con respecto al motor Unreal Engine 4. Esto es importante puesto que el desarrollo de videojuegos es un área de interés personal.

Asimismo, se pretende lanzar el plugin en la plataforma de ventas digital de Unreal Engine 4 por lo que se busca que el presente proyecto sea usado por desarrolladores de videojuegos con el fin de que estos ahorren tiempo implementando un sistema de guardado y cargado de partidas de juego.

1.2.6. Alcances y Limitaciones.

En cuanto al alcance de esta investigación, se espera desarrollar un plugin para el motor de videojuegos Unreal Engine 4. Se busca que el plugin otorgue a desarrolladores de juegos nuevas herramientas para el guardado y cargado de partidas de juego de forma que estos puedan hacer uso de estas herramientas para implementar sus propios sistemas de guardado en sus videojuegos. Además, se espera que el plugin mejore la calidad de las herramientas de guardado incluidas por defecto en el sistema de programación visual Blueprints de Unreal Engine 4.

En cuanto a las limitaciones, esta investigación no evalúa las herramientas de guardado que hacen uso del lenguaje de programación C++. Esto debido a que se considera que evaluar las herramientas de guardado haciendo uso exclusivamente del lenguaje de programación visual Blueprints es suficiente para evaluar la calidad del software.

Capítulo Segundo. Marco Teórico

2.1. Antecedentes de la Tesis

2.1.1. Antecedentes a Nivel Nacional.

Tabla 1
Antecedente Nacional 1

Tema	Contenido
Referencia APA	Domínguez Zarate, Romell Freddy. (2016). Aplicación de métricas de calidad en uso utilizando la ISO 9126 para determinar el grado de satisfacción del Sistema Único de Matrícula. Tesis de grado en ingeniería de sistemas, Universidad nacional mayor de San Marcos, Lima-Perú (Domínguez Zarate, 2016).
Resumen	La necesidad de evaluar un producto software a fin de corroborar la calidad que posee, nos lleva a plantear si luego del despliegue en producción aún la calidad del software se sigue manteniendo. En esta etapa el Sistema Único de Matrícula requiere una evaluación a nivel del usuario, para ello se ha utilizado el estándar internacional ISO 9126, la cual permitió establecer formalmente métricas a partir de las sub-características; se utilizó conjuntamente con buenas
Interés en la investigación	Se observa que se aplica las métricas propuestas en la norma ISO 9126 para determinar el grado de satisfacción del cliente. Aunque en la presente investigación se utilicen las métricas internas y externa.

Fuente: (Domínguez Zarate, 2016).

Tabla 2
Antecedente Nacional 2

Tema	Contenido
Referencia APA	Grupo Avatar (2014). Videojuego de la Pontificia Universidad Católica del Perú "1814: La rebelión del Cuzco". Pontificia Universidad Católica del Perú, Lima-Perú (Avatar, 2014).
Resumen	Este es un proyecto desarrollado por el Grupo AVATAR de la Pontificia Universidad Católica del Perú. Este videojuego combina la complejidad y la acción de dos tipos de géneros: juegos de rol y juegos de estrategia en tiempo real.
Interés en la investigación	Se observan las técnicas utilizadas para su desarrollo. De esta forma se pueden analizar sus requerimientos con respecto a las herramientas de guardado y cargado de partidas de juego.

Fuente: (Avatar, 2014).

Tabla 3
Antecedente Nacional 3

Tema	Contenido
Referencia APA	Medina Sanes, Gustavo Martin. (2014). Definición y evaluación de un modelo de calidad de uso para un portal de bolsa de trabajo utilizando la norma ISO/IEC 25000. Tesis de grado en ingeniería informática. Pontificia Universidad Católica del Perú, Lima-Perú (Medina Sanes, 2014).
Resumen	Este es un proyecto desarrollado por el Grupo AVATAR de la Pontificia Universidad Católica del Perú. Este videojuego combina la complejidad y la acción de dos tipos de géneros: juegos de rol y juegos de estrategia en tiempo real.
Interés en la investigación	Este antecedente analiza y evalúa modelos relacionados a norma ISO/IEC 25000. Esta normativa es la sucesora del modelo ISO/IEC 9126 explorada en esta investigación por lo que su relevancia es alta. Los modelos analizados en este antecedente permitirán tener una mayor idea de las métricas que determinan la calidad de un producto de software.

Fuente*:* (Medina Sanes, 2014).

Tabla 4
Antecedente Nacional 4

Tema	Contenido
Referencia APA	Canaval Sánchez, Luis Martin. (2017). Desarrollo de un videojuego con Unreal Engine 4. Tesis de grado en ingeniería informática. Pontificia Universidad Católica del Perú, Lima-Perú (Canaval Sánchez, 2017).
Resumen	El objetivo de este proyecto es resolver el problema de la ausencia de una metodología de desarrollo enfocada en videojuegos en el Perú. Para lograrlo, se diseña un videojuego de género plataformas y acción utilizando la tecnología de Unreal Engine 4, motor de videojuegos desarrollado por Epic Games. Los resultados obtenidos al finalizar este proyecto fueron beneficiosos, se pudo crear un videojuego de alta calidad en un corto periodo de tiempo y con recursos limitados. Así mismo, se encontró que la metodología puede ser adaptada dependiendo del alcance del proyecto, presupuesto y equipo.
Interés en la investigación	Este antecedente analiza hace uso del motor de videojuegos Unreal Engine 4 para desarrollar su investigación. Esta información resulta útil puesto que podemos determinar los momentos en los que suele implementarse un sistema de guardado y cargado de partidas de juego.

Fuente: (Canaval Sánchez, 2017).

2.1.2. Antecedentes a Nivel Internacional.

Tabla 5
Antecedente Internacional 1

Tema	Contenido
Referencia APA	Neupane, Pradip (2015). ESSENTIAL ELEMENTS OF GAME DEVELOPMENT: A CASE STUDY. Tesis de grado en "Information Technology", Turku University of applied sciences, Finlandia (Neupane, 2015).
Resumen	Esta investigación busca definir cuáles son los elementos esenciales para el diseño de un videojuego. No existe un estándar para la definición de los elementos esenciales que el desarrollo de un videojuego debe contener.
Interés en la investigación	Esta investigación determina los elementos esenciales en el desarrollo de un videojuego. Estos elementos sirven como guía para no obviar aspectos importantes a considerar durante el desarrollo de las herramientas de guardado y cargado de partidas de juego.

Fuente: (Neupane, 2015).

Tabla 6
Antecedente Internacional 2

Tema	Contenido
Referencia APA	Ivan Carmosino, Francesco Bellotti, Riccardo Berta, Alessandro De Gloria (2017). A game engine plugin for efficient development of investigation mechanics in serious games. Dept. of Electrical, Electronics and Telecommunication Engineering and Naval Architecture DITEN, University of Genoa, Via Opera Pia 11a, 16145 Genova, Italy (Carmosino, Bellotti, Berta, & De Gloria, 2017).
Resumen	A pesar de su potencial, el mercado para juegos educacionales (Serious Games) se encuentra limitado, en parte por los altos costes en diseño y producción. En este artículo, se propone un plugin para Unreal Engine 4 para el desarrollo eficiente de investigación de mecánicas para juegos serios.
Interés en la investigación	En este artículo, al igual que en la presente investigación, se busca desarrollar un plugin para el motor de videojuegos Unreal Engine 4. En el caso del artículo, el objetivo es mejorar la eficiencia de desarrollo al momento de investigar mecánicas para aplicarlas a videojuegos serios o educativos.

Fuente: (Carmosino, Bellotti, Berta, & De Gloria, 2017).

Tabla 7
Antecedente Internacional 3

Tema	Contenido
Referencia APA	Norita B. Ahmad, Salahudin Rahman Barakji, Tarak Mohamed Abou Shahada, Zeid Ayman Anabtawi (2017). How to launch a 12uccessful video game: A framework. Herramienta de software, Estados Unidos. Department of Marketing & Information Systems, School of Business Administration, American University of Sharjah, PO Box 26666, Sharjah, United Arab Emirates (Ahmad, Barakji, Shahada, & Anabtawi, 2017).
Resumen	Lanzar un videojuego exitoso al mercado es crucial para cualquier desarrolladora de videojuegos dada la naturaleza competitiva de la industria. Un videojuego exitoso puede garantizar la continuidad de algunas compañías, así como sus planes en la industria. Por el contrario, un videojuego que no genera suficientes ventas puede hacer caer el negocio. En este trabajo se hacen extensivos análisis de los actuales frameworks para el desarrollo de videojuegos como MDA.
Interés en la investigación	Aunque el Framework propuesto por este articulo este orientado al desarrollo de videojuegos y no al desarrollo de herramientas de juegos, el análisis que se realiza en cuanto al mercado actual, los motores de videojuegos y factores de éxito de un producto en la industria resultan muy útiles para la actual investigación.

Fuente: (Ahmad, Barakji, Shahada, & Anabtawi, 2017).

Tabla 8
Antecedente Internacional 4

Tema	Contenido
Referencia APA	Abran, Alain. Al-Qutaish, Rafa E. Cuadrado-Gallego, Juan. (2006). Analysis of the ISO 9126 on Software Product Quality Evaluation from the Metrology and ISO 15939 perspectives. Department of Computer Science. University of Alcalá, Spain (Abran, Al-Qutaish, & J Cuadrado-Gallego, 2006).
Resumen	Aunque la metrología tiene una larga tradición de ser usada en física y química, es raramente referida en medidas de ingeniería de software, y en particular, en el diseño y documentación de medidas de software. Usando la ISO 9126 para la calidad del software en un caso de estudio, este artículo describe la extensión en la cual esta norma ISO abarca los criterios típicos de metrología.
Interés en la investigación	Este artículo analiza la eficacia de la ISO 9126 en cuanto a su aporte a la medición de calidad del software en relación a la ISO 15939. Este análisis resulta importante puesto que nos asegura que el modelo ISO 9126 propone unas métricas útiles de forma que se pueda determinar la calidad de un producto de software.

Fuente: (Abran, Al-Qutaish, & J Cuadrado-Gallego, 2006).

2.2. Bases Teórico-Científicos

2.2.1. Videojuegos.

Los videojuegos son productos de software diseñados específicamente para interactuar con uno o más jugadores de forma que estos se involucren en mundos virtuales mediante dispositivos de entrada y salida. Estos softwares se desarrollan principalmente con el fin de entretener al usuario, aunque también existen videojuegos diseñados con otros propósitos, como la educación, entrenamiento militar, entre otros (Welcome to Pong-Story, 2013).

Los sistemas electrónicos usados para jugar a estos videojuegos son conocidos como plataformas de juego. Las plataformas de juego actuales más usadas son: PC, PlayStation 4, Xbox One, Nintendo Switch y dispositivos Android/IOS (New Media Society, 2008).

Una de las características que diferencian a los videojuegos de cualquier otro tipo de software es que los primeros se enfocan en otorgar al usuario un medio de interacción con un entorno virtual. Estos medios de interacción son generalmente dispositivos de entrada entre los cuales destacan los mandos de juego, pantallas táctiles, sensores de movimiento, joysticks, teclados, etc. Además de estos medios de entrada, los videojuegos otorgan una respuesta al jugador haciendo uso de dispositivos como pantallas, cascos de realidad virtual, efectos de sonido, entre otros (Pursell, 2015).

En la actualidad, los videojuegos son desarrollados y lanzados en plataformas de juego específicas. Los estudios o individuos que desarrollan estos juegos deben estar al tanto de las tecnologías y tendencias del mercado, en muchos casos haciendo uso de tecnologías y/o herramientas de terceros para acelerar el proceso de desarrollo (Rabin, 2006).

2.2.2. Sistemas de Guardado de Partidas en Videojuegos

Los sistemas de guardado y cargado en videojuegos nacen como una necesidad debido a la creciente complejidad y duración de videojuegos durante sus primeros años de existencia. Estos sistemas permiten guardar los datos del jugador y del estado del juego en un tiempo determinado con el fin de que el jugador pueda retomar el juego y reanudar su partida en el futuro. Los sistemas de guardado y cargado suelen serializar los datos de juego en un medio de almacenamiento no volátil. En la actualidad, estos datos suelen ser almacenados en discos duros; aunque también pueden ser almacenados en la nube, como es el caso de los juegos multijugador masivos (Sirlin, 2008).

Figura: 2 Menú de Guardado en Resident Evil 4

El menú de guardado del juego Resident Evil 4 muestra información relevante al estado del juego en cada fila (slot). Extraído del portal RE4HD (Capcom, 2010).

Estos sistemas de guardado y cargado de partidas de juego se pueden agrupar en las siguientes categorías de acuerdo a sus requerimientos:

a. **Sistema manual por "checkpoints"**: Las cuales consisten en aquellos sistemas que permiten guardar y restaurar el estado de juego en puntos específicos del juego. Estos sistemas de guardado suelen detener el juego por un momento mientras el jugador

navega por una interfaz gráfica la cual permite al usuario guardar y/o cargar el juego. El cargado del estado del juego se suele realizar de forma automática en caso de que el jugador haya perdido.

b. **Sistema automático por "checkpoints"**: Similar a la anterior categoría, con la diferencia de que el jugador solo necesita ubicarse en un punto determinado del juego para que el estado de juego se guarde de forma automática. Para restaurar el estado de juego, se suele usar una interfaz gráfica o de forma automática en caso de que el jugador pierda.

c. **Sistema automático por intervalos de tiempo**: Estos sistemas de guardado y cargado almacenan los datos del jugador de forma automática en intervalos cortos de tiempo por lo que suelen usar los recursos del sistema de forma constante. El cargado de partidas se realiza de forma similar a las anteriores categorías, ya sea de forma automática cuando el jugador pierda o de forma manual en caso de que el jugador reanude la partida luego de haber cerrado el videojuego.

2.2.3. Motor de videojuegos.

Un motor de videojuegos es un software orientado a la creación de videojuegos que incluye diversas tecnologías enfocadas en este propósito. Estas pueden ser tan primordiales como un motor de renderizado de gráficos 2D/3D, motor de audio, físicas o un entorno de programación. Así como tecnologías que ayuden a mejorar la productividad del desarrollo, como herramientas de colaboración, editores de niveles, editores de programación visual, entre otros.

En la actualidad, existen motores de videojuegos que son creados con el propósito de ser flexibles a diversos requerimientos por lo que pueden ser utilizados para la creación de diversos tipos de videojuegos. Motores de juego como Unreal Engine 4, Unity 3D y Godot son unos de

los motores de juego más populares y flexibles del mercado y pueden ser usados por terceros para el desarrollo de videojuegos (Ward, 2008).

2.2.4. Unreal Engine 4.

Unreal Engine 4 es un motor de videojuegos profesional desarrollado por Epic Games. Este motor incluye una suite de desarrollo que facilita la creación de videojuegos para un número importante de plataformas de juego y ofrece diversas tecnologías tales como un motor de renderizado 2D/3D, motor de audio, motor de físicas, un entorno de programación visual llamado Blueprints, entre otros. Uno de los aspectos más destacables de este motor es la posibilidad acceder a su código fuente de forma gratuita a pesar de ser uno de los motores, de acceso libre, más importantes de la industria. Unreal Engine 4, previamente conocido como "Unreal Development Kit", cuenta con más de 20 años en desarrollo y es en la actualidad uno de los motores de videojuegos más usados en la industria de videojuegos, siendo usado además en gran medida por la industria del cine, tv, arquitectura digital y realidad mixta (Epic Games, 2017).

Unreal Engine 4 soporta, de forma nativa, 2 lenguajes de programación: C++ y Blueprints. Ambos lenguajes cumplen diferentes funciones y se complementan entre sí. Mientras que el uso de C++ se enfoca en la programación del motor y de sistemas de bajo nivel, tales como módulos del motor y plugins; Blueprints está diseñado para la programación de la lógica de juego de alto nivel y para hacer uso de los sistemas programados en C++. (Epic Games, 2017e).

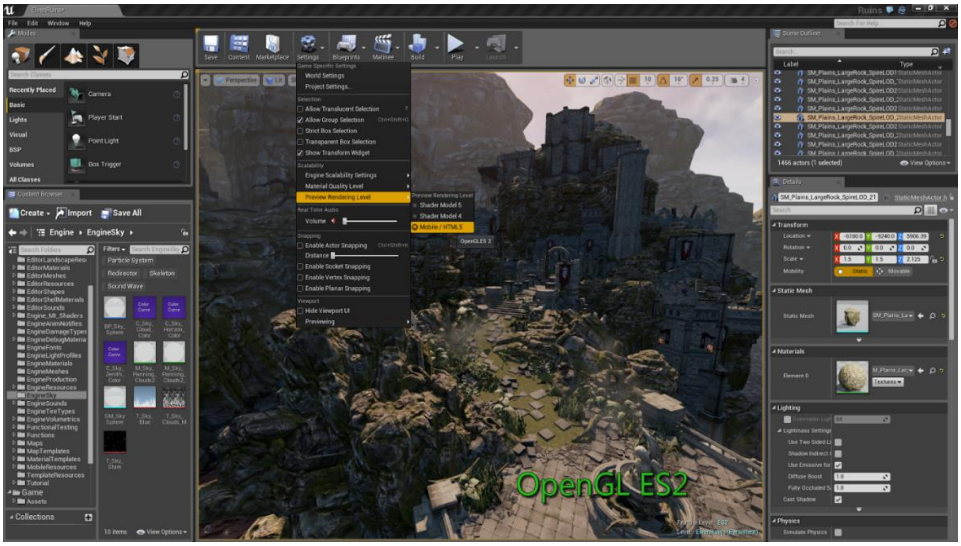


Figura: 3 Unreal Engine 4

Un vistazo a la ventana principal del editor UE4. Extraído del portal de Unreal Engine 4 (Epic Games, 2019).

2.2.4.1. Blueprints Visual Scripting.

Blueprints es el lenguaje de programación visual integrado en Unreal Engine 4. Este lenguaje está orientado a la programación de la lógica de juego y hace uso de nodos visuales que representan funciones creadas en C++ o en el mismo lenguaje visual. Este sistema es extremadamente flexible y práctico ya que permite que personas que no son programadores de profesión puedan crear lógica simple de juego haciendo uso de funciones escritas en C++, algo que es útil en un entorno de desarrollo de juego ya que permite que a los diseñadores probar el juego y realizar cambios cuando sea necesario. (Epic Games, 2017). Debido a su capacidad para representar funciones creadas en C++ a través de nodos visuales, este lenguaje es utilizado en esta investigación para otorgar a desarrolladores de juegos una mayor accesibilidad al momento de implementar sistemas de guardado de partidas en videojuegos. Además, este lenguaje visual permite modificar y/o extender ciertas características del plugin.

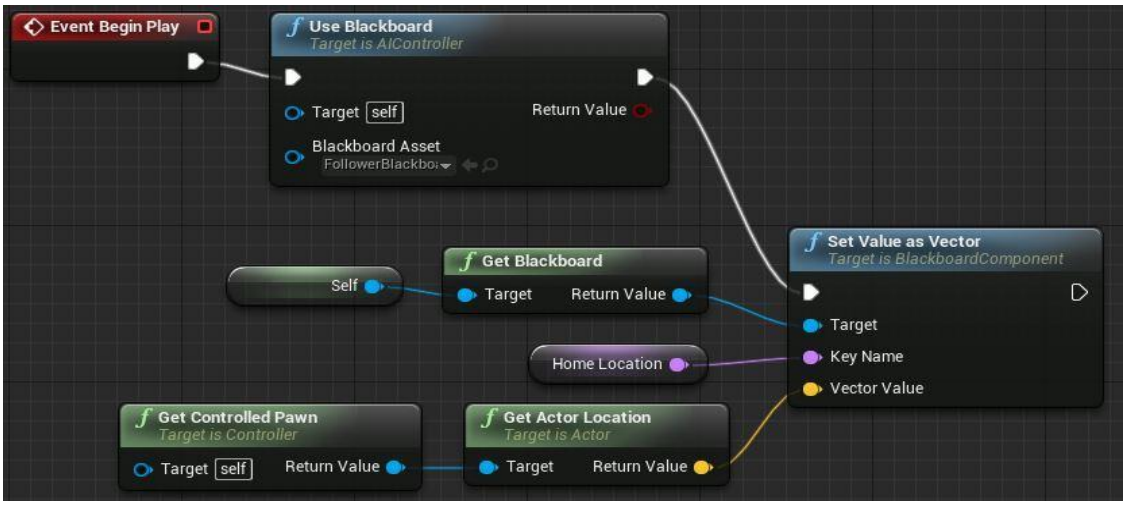


Figura: 4 Blueprints Visual Scripting

Un vistazo a Blueprints. Los nodos rojos son eventos que inician una acción; los azules, funciones de retorno; los verdes, funciones de ejecución. Extraído de la documentación de Unreal Engine 4 (Epic Games, 2017).

2.2.4.2. Unreal Engine C++.

Unreal Engine 4 permite la programación de la lógica de juego tanto en C++. Considerando tan solo lo concerniente a esta investigación, C++ en Unreal Engine 4 permite la creación de plugins para este motor. Además, resulta importante destacar que Unreal Engine 4 ofrece un marco de trabajo en C++; lo cual ayuda a no tener que preocuparse demasiado por ciertos aspectos técnicos del lenguaje, tales como la gestión de memoria y de punteros (Epic Games, 2017).

Figura: 5 C++ en Unreal Engine 4

Plantilla de una Clase “MyActor” en C++. Extraído de la documentación de Unreal Engine (Epic Games, 2017).

2.2.4.3. Gameplay Framework.

El "Gameplay Framework" es un framework que contiene herramientas y clases incluidas en Unreal Engine 4 que tienen el fin de otorgar un modelo de desarrollo a seguir a la hora de crear videojuegos en el motor. Este framework está diseñado para ser flexible y adaptable a diversos requerimientos de acuerdo a los distintos tipos de juego (Looman, 2018). Este framework cuenta con un gran número de clases, siendo las más relevantes:

a. **UObject**: La clase base de la cual heredan todas las clases del "Gameplay Framework". Esta clase no cuenta con la capacidad de ser renderizado en el mundo por lo que suele ser utilizado exclusivamente para implementar lógica de juego y/o para almacenar información en memoria (Looman, 2018).
b. **World**: Esta clase hereda de "UObject" y se encarga exclusivamente de representar un mapa o nivel en el cual los "Actor" y "Actor Components" existen y son visualizados (Looman, 2018).
c. **Game Mode Base**: Esta clase hereda de "UObject" y permite especificar las reglas de juego en un nivel determinado. Esta clase permite además asignar clases relevantes a un nivel, como el "Player Controller", "Pawn", entre otros (Looman, 2018).
d. **Save Game Object**: Clase heredada de "UObject" cuya única función es la de almacenar variables para luego ser serializado en disco (Looman, 2018).
e. **Actor**: Clase que hereda de "UObject" y que cuenta con la capacidad de ser instanciado y renderizado en un nivel ("World"). Actor es la clase más importante del "Gameplay Framework" ya que todos los objetos que van a ser visualizados en un videojuego en Unreal Engine 4 requieren heredar de esta clase. Un "Actor" puede contener uno o más "Actor Components" (Looman, 2018).

f. **Player Controller**: Clase que hereda de "Actor" y se encarga de gestionar los eventos de los controladores de entradas de un juego, tales como el mouse y teclado. Usado principalmente para controlar al personaje en un videojuego (Looman, 2018).

g. **Actor Component**: Clase que hereda de "UObject" y es usado para añadir funciones y/o elementos visuales a los "Actors". Estos componentes pueden ser acoplados y/o desacoplados en los "Actors" y son extensivamente utilizados en Unreal Engine 4 (Looman, 2018).

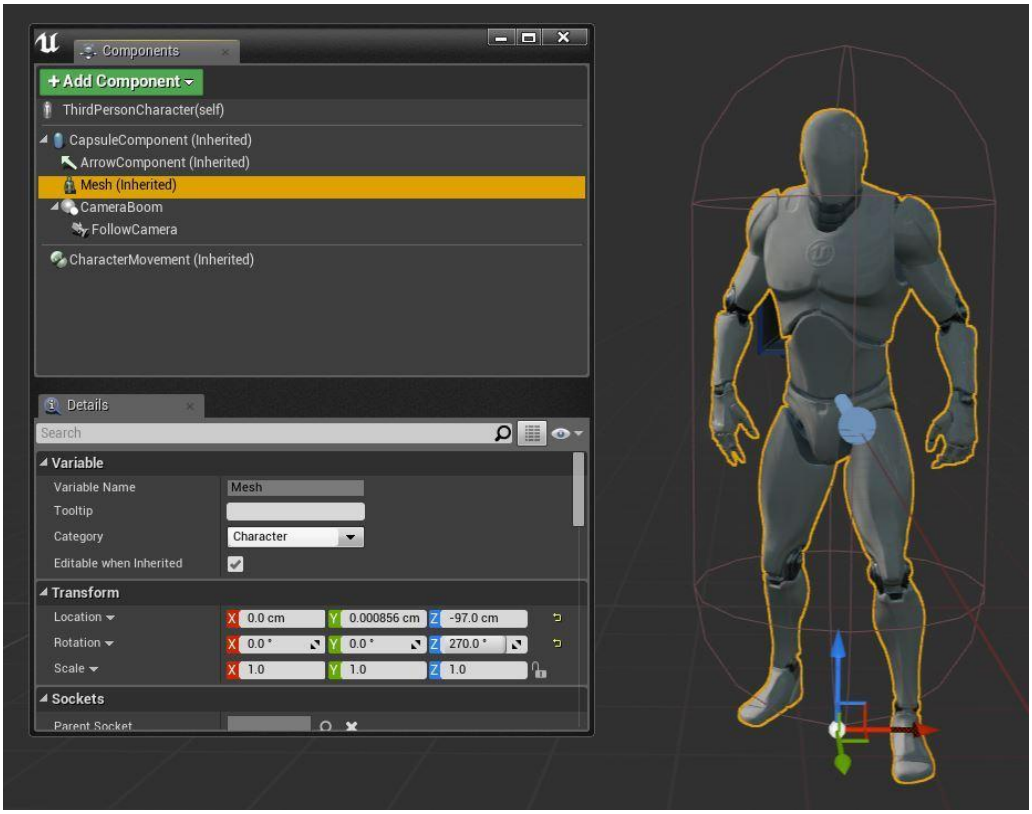


Figura: 6 Actor Component

Un vistazo a la ventana de componentes en el editor de Blueprints. Se observa una jerarquía de componentes ("CapsuleComponent", "Arrow Component", "Mesh", "Camera Boom", "FollowCamera" y "Character Movement") acoplados al Actor "ThirdPerson Character". Extraído de la documentación de Unreal Engine 4 (Epic Games, 2017).

2.2.4.4. Herramientas de Guardado en Unreal Engine 4.

Unreal Engine 4 cuenta con ciertas herramientas para el guardado y cargado de partidas expuestas a su lenguaje de programación visual Blueprints. Estas herramientas permiten serializar y deserializar el estado de un juego a un medio no volátil como el disco duro de una computadora. “Save Game Object” es la clase en la cual las variables son almacenadas para luego ser serializadas a disco. Además, Unreal Engine 4 cuenta con 2 funciones estáticas: “Save Game ToSlot” y “Load Game From Slot”, las cuales son utilizadas para guardar y restaurar los datos de juego de un “Save Game Object” respectivamente (Epic Games, 2018).

2.2.4.5. Plugins en Unreal Engine 4.

Un plugin o complemento no es más que una extensión de software que añade una o más prestaciones a un programa diseñado de tal forma que soporte el acoplamiento de estos. Un plugin se caracteriza por estar completamente desacoplado del programa al que está asociado, por lo que su ausencia no implica que el programa deje de funcionar. Debido a esta característica, un plugin no es capaz de realizar funciones por sí mismo, sino que requiere del software principal para ser utilizado (Lee, Chun, & Kang, 2014).

Unreal Engine 4 cuenta con una arquitectura modular que permite el acoplamiento y/o desacoplamiento de módulos. Estos módulos pueden extender o reemplazar las funcionalidades del motor. Un Plugin en Unreal Engine 4 no es más que una colección de módulos con esta capacidad (Epic Games, 2017).

2.2.4.6. El Marketplace de Unreal Engine 4.

El Marketplace de Unreal Engine 4 es una plataforma de venta digital de contenido para el desarrollo de juego. Esta plataforma de ventas oficial permite, entre otras cosas, la distribución de plugins compatibles con Unreal Engine 4 (Epic Games, 2018).

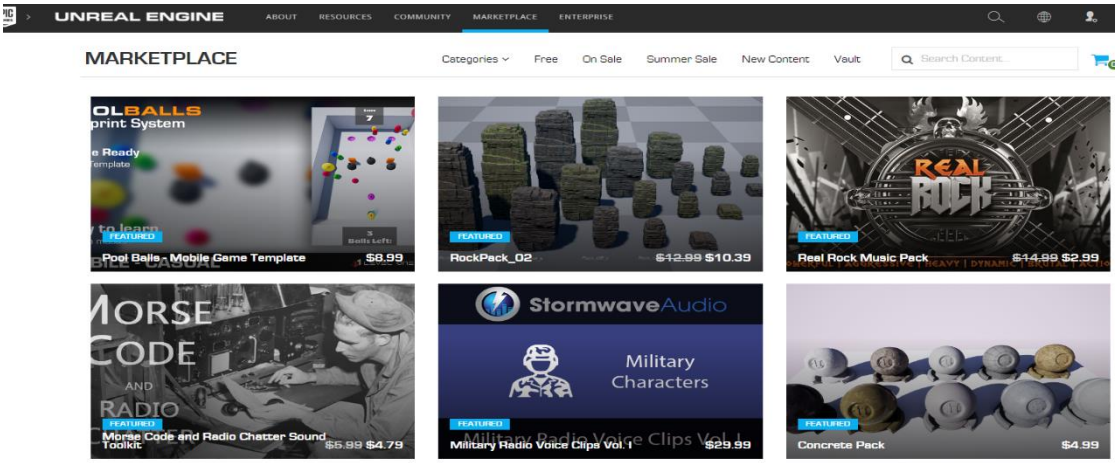


Figura: 7 El Marketplace de Unreal Engine 4

Un vistazo a la plataforma de ventas de Unreal Engine 4. Estos contenidos pueden adquirirse para ser usados en el desarrollo de videojuegos. Extraído del portal de Marketplace de Unreal Engine 4 (Epic Games, 2018).

2.2.5. Norma ISO/IEC 9126 para la Calidad de Software.

La calidad de software se define como el conjunto de características de un producto de software que tienen como fin el satisfacer necesidades explícitas o implícitas del usuario, las cuales son definidas en forma de requerimientos del producto. Por tanto, se puede definir a la calidad de un software como el nivel en que un sistema, componente o proceso cumple los requisitos especificados, así como las expectativas y/o necesidades del usuario. Este concepto de calidad de software resulta importante para la investigación, ya que medir y/o comparar la calidad de un software permitirá realizar procesos de mejora continua sobre el mismo (ISO/IEC, 2001).

La norma ISO/IEC 9126 es un modelo que define métricas que pueden ser utilizadas para medir la calidad de un producto de software. Esta norma tiene en cuenta las necesidades implícitas y explicitas del usuario, de forma que se cumplan ciertos niveles de calidad de acuerdo a los requerimientos del producto. Es importante tener en cuenta, además, que esta norma se desarrolla teniendo en cuenta la subjetividad, tanto de los usuarios como de los desarrolladores, a la hora de definir la calidad de un software. Por lo que la norma tiene como fin establecer un

marco objetivo que permita medir la calidad de un software de la forma más objetiva posible (ISO.ORG, 2001).

Las métricas que la norma establece pueden ser aplicadas en procesos de mejora continua durante el desarrollo de un software. Todo esto con el fin de planificar, seguir, controlar, replanificar y evaluar la calidad del software (ISO/IEC, 2001).

La norma ISO/9126 contiene métricas internas y externas que permiten a las personas encargadas del desarrollo de software evaluar la calidad durante el proceso de desarrollo y durante las pruebas del software. Las métricas internas son aquellas que no requieren de un software en ejecución para medirse, mientras que las métricas externas son aplicables a softwares en ejecución.

Es importante mencionar además que esta norma permite utilizar métricas e indicadores de forma flexible de acuerdo al producto de software a evaluar. Es decir que las métricas e indicadores a usarse para medir la calidad durante el desarrollo de un software pueden variar de acuerdo a los requerimientos del software (ISO/IEC, 2001).

A continuación, se listan las métricas internas y externas de calidad de software propuesta por la norma ISO/IEC 9126 y que son más relevantes para la investigación:

2.2.5.1. Métrica de Funcionalidad.

Esta métrica mide el nivel en el cual el software cumple y provee funcionalidades que satisfagan los requerimientos. Esta métrica cuenta con los siguientes indicadores:

a. **Adecuación**: Nivel en el cual el software cumple los objetivos y requerimientos.

2.2.5.2. Métrica de Usabilidad.

Esta métrica mide el nivel en el cual el software puede ser entendido, aprendido y usado por los usuarios. Esta métrica cuenta con los siguientes indicadores:

a. **Entendimiento**: Nivel en el cual el software puede ser entendido y aplicado para solucionar problemas en entornos de producción.
b. **Aprendizaje**: Nivel de esfuerzo necesario para aprender a usar el software.
c. **Operatividad**: Nivel de facilidad de uso del software.

2.2.5.3. Métrica de Eficiencia.

Esta métrica mide el nivel en el cual el software utiliza los recursos del ordenador y cómo este se comporta en términos de rendimiento. Esta métrica cuenta con los siguientes indicadores:

a. **Comportamiento de tiempos**: Nivel en el cual los tiempos de respuesta y procesamiento son adecuados de acuerdo a los requerimientos.
b. **Utilización de recursos**: Nivel en el cual el software usa los recursos del ordenador (procesador, memoria, disco, red, entre otros).

Además, la norma cuenta con las siguientes métricas, las cuales no son utilizadas en esta investigación debido a que abarcan aspectos que no aplican al desarrollo de un plugin para Unreal Engine 4 o al enfoque de la presente investigación:

a. **Métrica de Confiabilidad**: El cual mide los atributos que hacen que un software mantenga un nivel de rendimiento bajo condiciones y periodos de tiempo arbitrarios.
b. **Métrica de Mantenibilidad**: El cual mide el nivel de esfuerzo que se requiere para realizar modificaciones a un software luego de que este haya sido lanzado en un entorno de producción.
c. **Métrica de Portabilidad**: El cual mide la capacidad de un software de ser ejecutado en distintos entornos de producción (Sistemas Operativos, Plataformas, etc).

2.2.6. Proceso Ágil Unificado de Desarrollo (AUP).

El AUP (Agile Unified Process) es una metodología que describe de manera simple y entendible el cómo abarcar el desarrollo de software mediante el uso de técnicas y conceptos ágiles de forma que se consiga una mayor fidelidad del software con respecto a sus requerimientos.

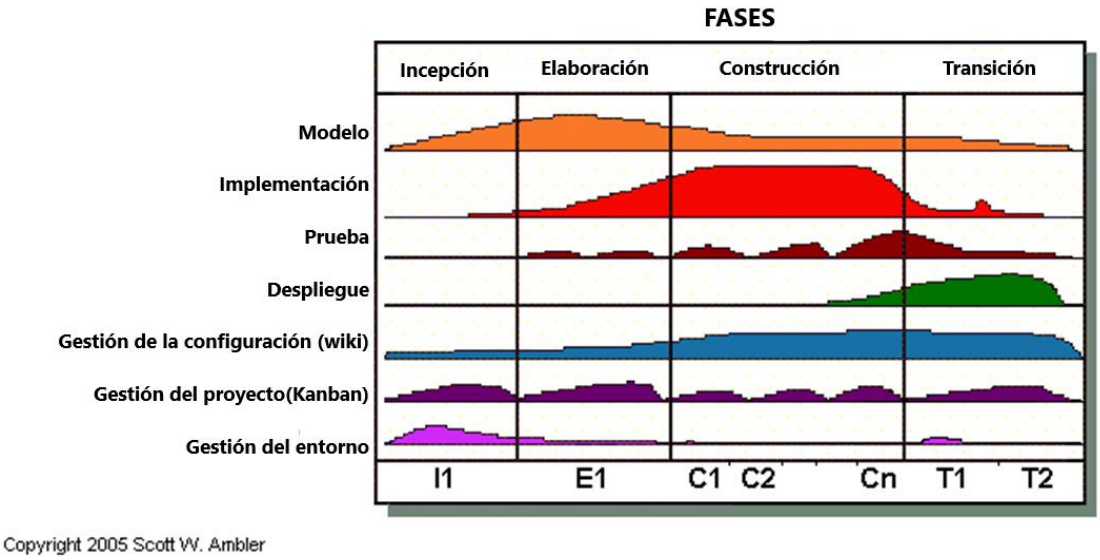


Figura: 8 Ciclo de Vida de AUP

El ciclo de vida de AUP. Se observan las fases y disciplinas (y su importancia) de esta metodología. Extraído de ambysoft (ambysoft, 2006).

Esta metodología describe cuatro fases generales, las cuales son:

a. **Fase de Incepción**: Esta fase busca identificar el alcance inicial del proyecto, definir una potencial arquitectura del sistema y obtener los fondos iniciales y/o el visto bueno de inversionistas.

b. **Fase de Elaboración**: Esta fase busca poner a prueba la arquitectura inicial del sistema. Esto puede lograrse mediante la creación de un prototipo inicial del software.

c. **Fase de Construcción**: En esta fase se construye el software de forma regular, incremental e iterativa teniendo siempre en cuenta la prioridad de las tareas a realizarse durante cada iteración.
d. **Fase de Transición**: Esta fase busca validar y lanzar el software en un entorno de producción.

Además, AUP describe disciplinas que forman parte de cada una de las fases generales, las cuales son aspectos del desarrollo que deben implementarse. Estas disciplinas varían su importancia de acuerdo a la fase en la que se encuentre el proyecto y son flexibles a cambios conforme a una metodología ágil. Estas disciplinas consisten en:

a. **Modelo**: Esta disciplina busca entender el negocio de la organización mediante el modelo del dominio del problema, así como la identificación de soluciones viables para abarcar el mismo. Esto puede conseguirse mediante la modelación de los requerimientos del software haciendo uso de las diversas herramientas y/o técnicas de modelado, tales como BPMN, UML, entre otros. Esta disciplina tiene mayor importancia durante la fase de incepción y elaboración del proyecto.
b. **Implementación**: Esta disciplina busca transformar el modelo en código ejecutable. Su importancia es mayor durante la fase de construcción.
c. **Prueba**: Esta disciplina busca asegurar la calidad del software mediante la realización de evaluaciones diseñadas para encontrar defectos y verificar que los requerimientos del proyecto sean abarcados en la implementación. Esto se consigue mediante la aplicación de técnicas y/o herramientas diseñadas para poner a prueba un software tales como las pruebas unitarias, pruebas funcionales, casos de prueba, entre otros. La

importancia de esta disciplina es mayor durante la fase de construcción y transición del proyecto.

d. **Despliegue:** Esta fase busca planificar y ejecutar el plan de despliegue del software. Esto puede lograrse planificando y ejecutando manualmente los de hitos de despliegue, así como haciendo uso de técnicas más complejas como la implementación de un sistema de integración y entrega continua (CI/CD), el cual consiste en la automatización del despliegue y, en algunos casos, automatización de las pruebas. Esta disciplina cuenta con mayor importancia durante la fase de transición del proyecto.

e. **Gestión de la configuración:** Esta disciplina busca gestionar el acceso a los artefactos generados durante el proyecto, sus versiones y posibilitar los cambios de estos mismos. Estos artefactos son subproductos tangibles creados durante el desarrollo del software y pueden consistir en documentos de diseño, planes, casos de uso, etc. Esto puede lograrse mediante el uso de herramientas de gestión de proyectos tales como Azure DevOps el cual incluye un gestor de documentación de proyectos. Esta disciplina tiene importancia durante todas las fases del proyecto, en especial durante las últimas etapas.

f. **Gestión del proyecto:** Esta disciplina busca gestionar y dirigir las actividades que toman lugar durante el proyecto, esto con fin de manejar y reducir los riesgos de desarrollo (tiempo, costo, calidad, etc.) así como coordinar a las personas encargadas del desarrollo para asegurar que el proyecto sea terminado a tiempo y dentro del presupuesto. Esto puede conseguirse mediante el uso de metodologías de gestión de proyectos tales como Kanban, el cual está incluido en la herramienta de gestión Azure

DevOps. Esta disciplina tiene importancia durante todas las fases del proyecto y se comporta de forma iterativa durante la fase de construcción.

g. **Gestión del entorno de desarrollo:** Esta disciplina busca asegurar que los procesos correctos, guías (estándares) y herramientas (tanto de software como de hardware) estén disponibles para el equipo de desarrollo en cuanto sea necesario. Esto se logra mediante el establecimiento de un entorno de desarrollo y el uso de herramientas durante la etapa inicial del proyecto. En el contexto de la investigación, se hace uso del estándar ISO/IEC 9126 como guía de calidad de software, así como de diversas herramientas de desarrollo y gestión tales como Unreal Engine 4, Visual Studio, Azure DevOps, etc.

En resumen, AUP es una metodología ágil de desarrollo de software que permite gestionar el desarrollo de software otorgando un marco flexible y entendible que permite desarrollar software de forma iterativa (ambysoft, 2006).

2.2.7. Kanban.

De acuerdo al artículo "A statistical analysis of the effects of Scrum and Kanban on software development process" (2015): "*Kanban es otra metodología de gestión de proyectos para el desarrollo de software que se enfatiza en la estrategia de justo a tiempo. El principal enfoque de Kanban es el de determinar de forma precisa que trabajo necesita ser hecho, y cuando necesita ser realizado. Esto se realiza priorizando tareas, así como definiendo flujos de trabajo y tiempos para realizar las tareas. El proceso de Kanban presenta explícitamente las tareas más importantes que necesitan mayor atención de forma que el riesgo de no cumplirlas se reduzca, además de incrementar la flexibilidad entre otras tareas en el proyecto. La metodología Kanban se centra así en tener el trabajo correcto en el tiempo justo, dadas las habilidades de los*

desarrolladores. Los desarrolladores pueden tener diferentes habilidades y rapidez de trabajo. Los desarrolladores del proyecto empiezan implementando componentes que añadan valor al proyecto. En adición, los desarrolladores del proyecto no implementan características innecesarias, ni escriben más especificaciones de las que pueden programar, ni escriben más código de lo que pueden probar, ni tampoco prueban más código del que pueden desplegar. Así, la metodología Kanban elimina tiempos perdidos en cada paso, y se ajusta a los proyectos de ingeniería de software" (Howard, Farnaz, Pradeep Kumar, & Ozcan, 2015).

En resumen, se considera a Kanban como una metodología de gestión de proyectos ágil, simple, práctica y flexible, por lo que es aplicable al desarrollo de software. Kanban es capaz de revelar cuellos de botella en el proceso de producción, permitiendo fragmentar actividades complejas en tareas más sencillas (Anderson, 2010).

En su forma estándar, Kanban consiste de un tablero dividido en secciones verticales tales como "Pendiente", "En proceso", "Finalizado". Cada una de estas secciones contiene una o más tarjetas enumeradas que representan las tareas a realizar. Esta metodología es lo suficientemente flexible como para adaptarse a cualquier proyecto por lo que la tabla de Kanban puede sufrir variaciones de acuerdo al proyecto (Peterson, 2015).

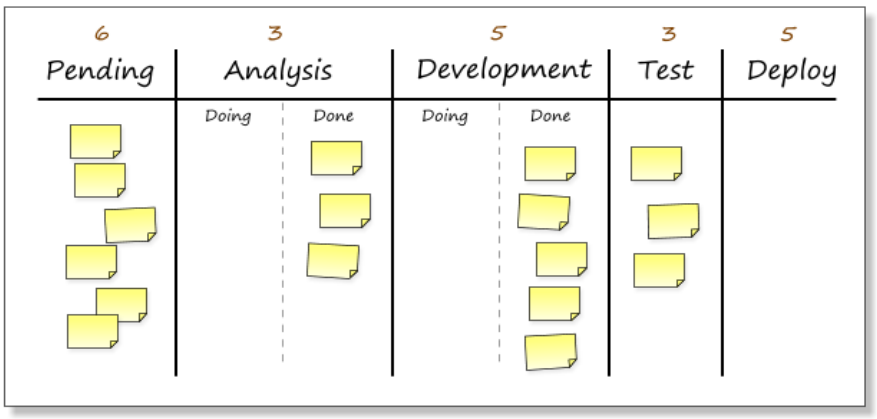


Figura: 9 Tablero Kanban

Un tablero Kanban con 5 columnas. Su aspecto puede sufrir variaciones de acuerdo al tipo de proyecto. Extraído de "What is Kanban?" (Peterson, 2015).

2.2.8. Azure DevOps.

Azure DevOps es un conjunto de herramientas orientadas a la colaboración en los procesos de desarrollo de software. Desarrollado por Microsoft y basado en la nube, Azure DevOps contiene herramientas que ayudan a planificar, desarrollar y administrar proyectos de software (Microsoft, 2018).

Azure DevOps usa tableros Kanban como herramienta para planificar el desarrollo de software además de ofrecer otros servicios como repositorios de control de versiones por lo que resulta una herramienta útil para esta investigación.

Capítulo Tercero. Desarrollo del Proyecto

Para el desarrollo del plugin se hace uso de la metodología de desarrollo de software AUP (Proceso ágil unificado de desarrollo). Por ello, el presente capítulo se estructura de acuerdo a las fases propuestas por AUP, las cuales están descritas en el marco teórico de esta investigación y consisten en las fases de: incepción, elaboración, construcción y transición. Además, cada una de estas fases cuenta con disciplinas de desarrollo que varían su importancia de acuerdo a la fase en la que se encuentre el desarrollo.

Además, se hace uso de Kanban para la implementación la disciplina de "gestión del proyecto" propuesta por AUP, la cual es relevante durante todas las fases de desarrollo.

Finalmente, ya que el objetivo de esta tesis consiste en determinar la influencia del plugin en la calidad de las herramientas de guardado y cargado de partidas para videojuegos en Unreal Engine 4, resulta importante destacar la importancia de la norma ISO/IEC 9126; la cual establece métricas de calidad que servirán como guía de desarrollo de software con el fin de garantizar una mejora en la calidad con respecto a las soluciones de guardado con las que actualmente cuenta Unreal Engine 4. Por ello, se considera que resulta importante no solo establecer un punto de partida mediante el análisis de la calidad de las herramientas de guardado actuales con las que cuenta el motor, sino también realizar evaluaciones iterativas de la calidad del plugin durante la construcción del mismo.

3.1. Fase de Incepción

Durante esta fase, se busca identificar el alcance inicial del plugin, definir una arquitectura de software inicial y determinar la viabilidad del desarrollo. Las disciplinas más relevantes durante esta fase son: La disciplina de modelo, gestión de la configuración y gestión del entorno de

desarrollo; así como la disciplina de gestión del proyecto, la cual es relevante durante todas las fases el proyecto.

3.1.1. Cuadro General del Proyecto.

El siguiente cuadro describe el entendimiento inicial del plugin, así como los problemas y posibles soluciones a abarcar:

Tabla 9
Cuadro General del Proyecto

Problema	Solución	Valor único	Posibles desventajas
La calidad de las herramientas de guardado incluidas en Unreal Engine 4.	Desarrollar un plugin en Unreal Engine 4 que mejore la calidad de las herramientas.	Mejor calidad de software de acuerdo a las métricas propuestas por la norma ISO/IEC 9126.	El plugin requiere de un canal de distribución oficial debido a su necesidad de mantenimiento.
Clientes objetivo	**Alternativas**	**Métricas clave**	**Distribución**
Desarrolladores de videojuegos que hagan uso del motor de videojuegos Unreal Engine 4.	Las herramientas incluidas en Unreal Engine 4.	Métricas de funcionalidad, usabilidad y eficiencia. De acuerdo a la norma ISO/IEC 9126.	El Marketplace de Unreal Engine 4.

Fuente: Elaboración propia.

3.1.2. Configuración del Entorno de Desarrollo.

Esta investigación hace uso de Azure DevOps como herramienta de gestión del proyecto, gestión de la configuración y como herramienta de colaboración. Para ello, se configura un proyecto en Azure, tal como se muestra en la siguiente figura.

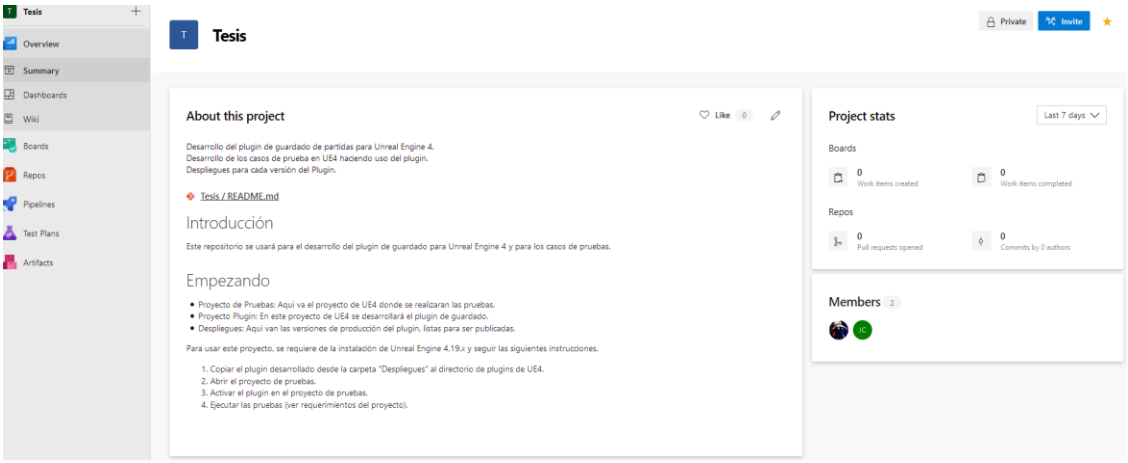


Figura: 10 Configuración del Entorno de Desarrollo

Proyecto “Tesis”, cuenta con instrucciones generales y con acceso para ambos miembros de esta investigación.
Fuente: Elaboración propia.

Se configura además un repositorio de documentos de forma que sea posible acceder y editar documentos generados durante el ciclo de desarrollo del plugin.

Figura: 11 Wiki del Proyecto

Fuente: Elaboración propia.

Asimismo, se configura un repositorio de control de versiones para facilitar la colaboración y trabajo en equipo. Este repositorio contiene 2 proyectos de Unreal Engine 4 (Uno de desarrollo y uno de pruebas) así como una carpeta para los despliegues o versiones de producción del plugin.

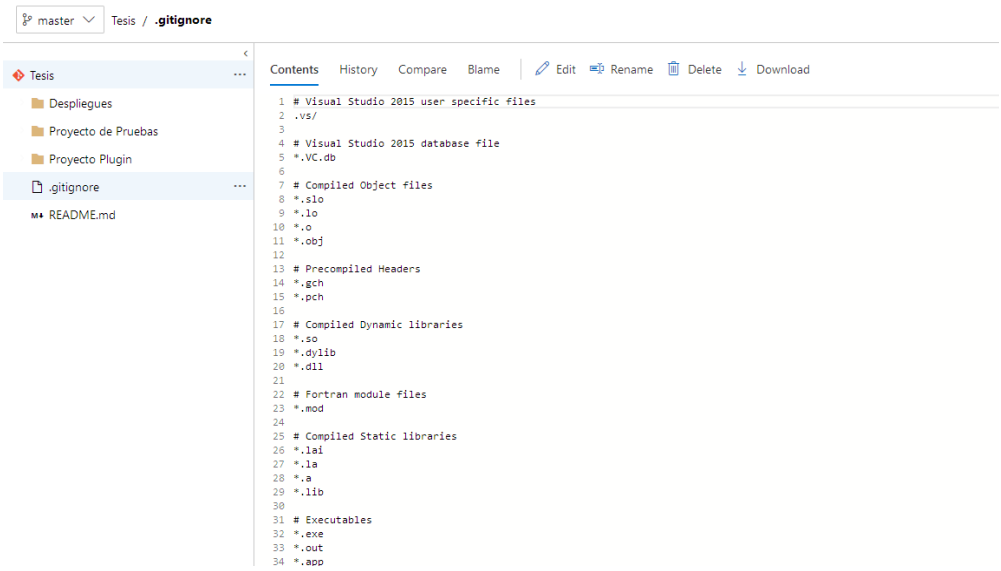


Figura: 12 Control de Versionado del Proyecto (Git)

Fuente: Elaboración propia.

Finalmente, se configura el tablero Kanban el cual será usado para gestionar y dirigir las tareas a realizarse durante las iteraciones del proyecto. Para el presente proyecto, el tablero está dividido en 4 columnas las cuales consisten en:

a. **Backlog**: Esta columna lista las tareas del proyecto de acuerdo a su prioridad. Estas actividades pueden variar su número o ser eliminadas de acuerdo a como avance el proyecto.
b. **Pendiente**: Esta columna lista las tareas que cuentan con prioridad mayor a las actividades en el "Backlog" y, por tanto, son las siguientes a realizarse.
c. **En Proceso**: Esta columna lista las tareas que están siendo realizadas en un momento determinado.
d. **Terminado**: Columna donde se listan las tareas finalizadas.

En cuanto a la configuración del entorno local, se tienen 2 proyectos de Unreal Engine 4. El primero es un proyecto de Unreal Engine C++ que será usado para el desarrollo del plugin mientras que el segundo, un proyecto de Unreal Engine 4 Blueprints que será usado para las pruebas.

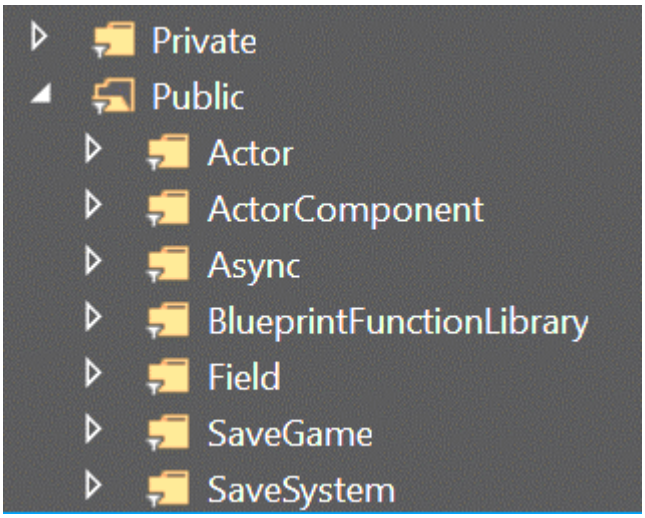


Figura: 13 Configuración del Entorno Local

Fuente: Elaboración propia.

Para el proyecto de desarrollo del plugin se crea una jerarquía de carpetas de forma que contengan clases de acuerdo a su función, tal como se describe a continuación:

a. **Actor**: Contiene clases con la capacidad de ser instanciados de forma visible en un nivel de juego.
b. **Actor Component**: Contiene componentes que pueden ser acoplados a diversos Actores
c. **Async**: Contiene clases que implementen funciones asíncronas.
d. **Blueprint Function Library**: Contiene clases que implementen librerías estáticas y accesibles desde Blueprints.
e. **Field**: Contiene estructuras de datos, enumeradores e interfaces.

f. **Save Game**: Contiene clases con la capacidad de almacenar variables y ser serializados en disco.

g. **Save System**: Contiene clases que implementen funciones que hagan uso de funciones genéricas compatibles con diversas plataformas de juego (PC, Android, Consolas, etc.).

En cuanto al proyecto de pruebas, este se organiza de forma que cada carpeta contenga un nivel de juego con el fin de poner a prueba los requerimientos del proyecto. La descripción de la función de cada carpeta es la siguiente:

a. **Automático**: Esta carpeta contiene un ejemplo de un sistema de guardado automático por intervalos de tiempo.

b. **Checkpoints**: Esta carpeta contiene un ejemplo de un sistema de guardado por condición.

c. **Menú**: Esta carpeta contiene un ejemplo de un sistema de guardado con menú de opciones.

3.1.3. Modelación.

Parte de la disciplina del modelo, en esta sección se busca realizar una modelación del dominio del proyecto. Esto se logra mediante el análisis de los requerimientos del mismo junto a una posterior modelación de la arquitectura del software. De esta forma no solo se tendrá un mejor entendimiento de lo que se espera conseguir con el desarrollo del plugin, sino que además será posible identificar posibles soluciones a los problemas que serán embarcados durante las fases posteriores del desarrollo.

3.1.3.1. Definición de los Requerimientos de Alto Nivel.

De acuerdo a la definición de un sistema de guardado y cargado de partidas, y de las métricas relevantes de calidad de software descritas en el marco teórico de esta investigación, se obtienen los siguientes requerimientos iniciales:

3.1.3.1.1. De acuerdo a su funcionalidad.

a. **Se debe permitir guardar el juego de forma tradicional mediante un menú**: Ya que se considera como requerimiento funcional de un sistema de guardado y cargado de partidas para videojuegos de acuerdo a lo descrito en el marco teórico de la presente investigación.
b. **Se debe permitir guardar el juego de forma automática en "checkpoints"**: Ya que se considera como requerimiento funcional de un sistema de guardado y cargado de partidas para videojuegos de acuerdo a lo descrito en el marco teórico de la presente investigación.
c. **Se debe permitir guardar el juego de forma automática en intervalos de tiempo**: Ya que se considera como requerimiento funcional de un sistema de guardado y cargado de partidas para videojuegos de acuerdo a lo descrito en el marco teórico de la presente investigación.

3.1.3.1.2. De acuerdo a su usabilidad.

a. **Debe ser fácil de entender:** Requerimiento obtenido de la norma ISO/IEC 9126 para la calidad de software. Este requerimiento ayuda a medir el nivel en el que el usuario entiende las herramientas de guardado y sus posibles aplicaciones aun sin haberlas usado previamente.

b. **Debe ser fácil de aprender a usar**: Requerimiento obtenido de la norma ISO/IEC 9126 para la calidad de software. Este requerimiento ayuda a medir el nivel de dificultad para aprender a usar las herramientas y aplicarlas para implementar sistemas de guardado y cargado de partidas de juego.

c. **Debe ser sencillo de operar**: Requerimiento obtenido de la norma ISO/IEC 9126 para la calidad de software. Este requerimiento ayuda a medir el nivel de facilidad de uso y aplicación de las herramientas para la implementar sistemas de guardado y cargado de partidas de juego.

3.1.3.1.3. De acuerdo a su eficiencia.

a. **Debe tener un tiempo de procesamiento aceptable**: Requerimiento obtenido de la norma ISO/IEC 9126 para la calidad de software. Este requerimiento ayuda a medir el tiempo de procesamiento que las herramientas de guardado toman al momento de guardar y cargar los datos de la partida de juego.

b. **No debe afectar el rendimiento del juego**: Requerimiento obtenido de la norma ISO/IEC 9126 para la calidad de software. Este requerimiento ayuda a medir el nivel en el que las herramientas de guardado hacen uso de los recursos del ordenador al momento de guardar y cargar los datos de partida de juego.

3.1.3.2. Análisis de la Calidad de las Herramientas de Unreal Engine 4.

Debido a que se pretende mejorar la calidad de las herramientas de guardado y cargado de partidas incluidas en el motor, resulta importante realizar un análisis preliminar de la calidad de estas herramientas haciendo uso de las métricas propuestas por la ISO/IEC 9126. Esta evaluación servirá como punto de partida para garantizar que el plugin a desarrollarse influencie de manera positiva en la calidad del mismo.

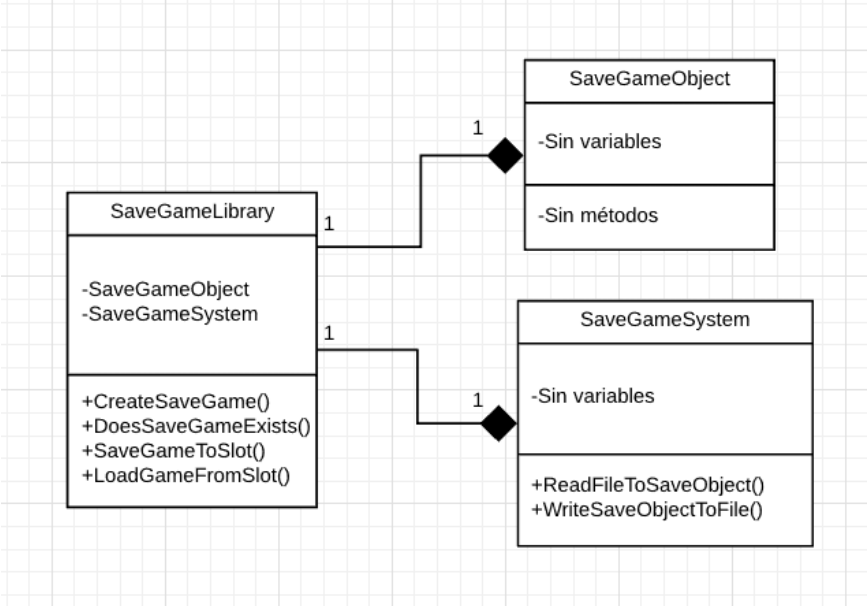


Figura: 14 Diagrama de Clases de las Herramientas de Unreal Engine 4

Fuente: Elaboración propia.

Como se observa en la figura anterior, las herramientas de guardado y cargado de partidas de videojuego del "Gameplay Framework" de Unreal Engine 4 consisten de 3 clases que, usadas en conjunto, permiten la implementación de un sencillo sistema de guardado y cargado de partidas de juego. Estas clases consisten en:

a. **Save Game Object**: Esta clase es usada como objeto de almacenamiento de variables. Para hacer uso de esta clase es necesario crear variables de forma manual (dentro de una instancia de esta clase) de acuerdo a los requerimientos del videojuego.
b. **Save Game System**: Esta clase cuenta con métodos para la serialización y deserialización de un "Save Game Object". Estos métodos permiten escribir el estado de un "Save Game Object" a un archivo en disco y viceversa.
c. **Save Game Library**: Esta clase es una librería de funciones estáticas que hacen uso de "Save Game Object" y de "Save Game Library" para soportar la implementación de un sistema de guardado y cargado de partidas. Estas funciones son accesibles tanto en C++ como en Blueprints.

Figura: 15 Variables del Proyecto de Ejemplo

Se observa a la clase "Mi Objeto De Guardado" heredada de "Save Game Object". En este ejemplo, esta clase cuenta con 2 variables que definen el estado del juego. Fuente: Elaboración propia.

Si se toma como ejemplo un juego pequeño en donde exista un jugador y un enemigo; y donde se requiere guardar el estado de ambos (La variable "Salud" en el caso de ambos objetos), entonces se requerirá de una clase heredada de "Save Game Object" a la cual se denominará "Mi Objeto De Guardado", tal como se muestra en la figura anterior. Esta clase no contiene métodos, pero si las variables necesarias para guardar y cargar el estado del juego.

En este ejemplo se tienen entonces 3 clases: La clase "Jugador", que contiene la información de la salud del jugador y donde se implementa el sistema de guardado y cargado de partidas; la clase "Enemigo", que contiene la información de la salud del enemigo; y la clase "Mi Objeto De Guardado", que contiene 2 variables usadas para guardar y restaurar la salud tanto de "Jugador" como de "Enemigo".

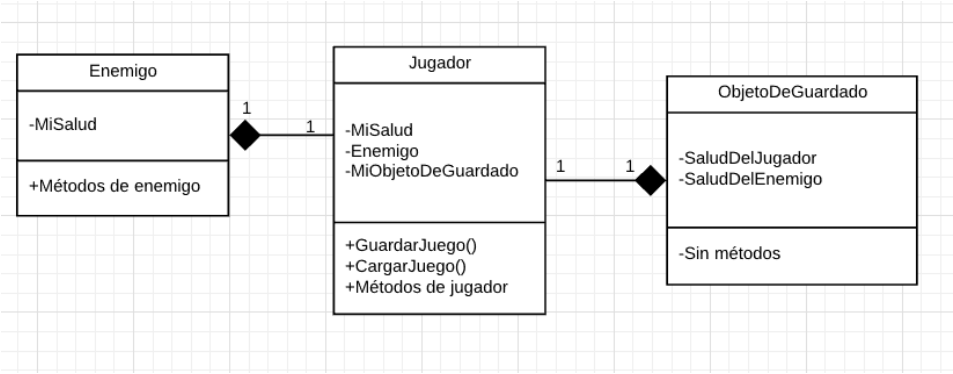


Figura: 16 Diagrama de Clases del Proyecto de Ejemplo

Fuente: Elaboración propia.

Luego, se implementa una lógica de guardado simple dentro de la clase “Jugador”, tal como se muestra en la figura posterior.

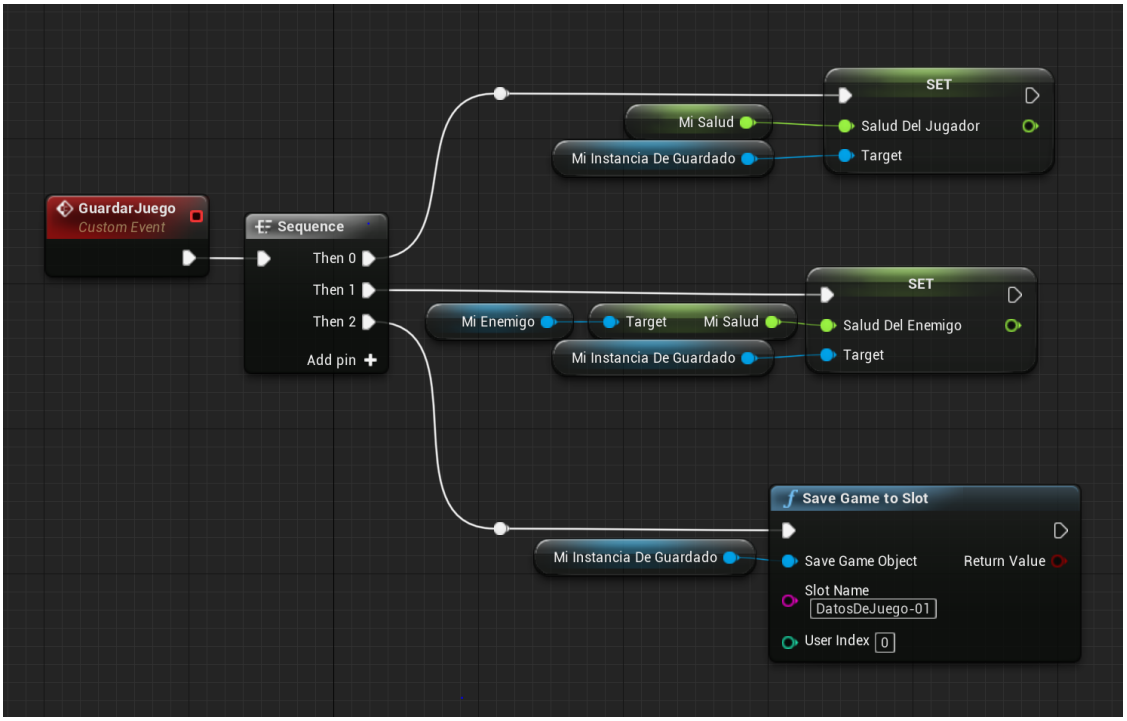


Figura: 17 Implementación de un Sistema de Guardado de Ejemplo

Fuente: Elaboración propia.

En el primer paso, se copia el valor de la salud del jugador dentro de una instancia de “Mi Objeto De Guardado”. En el segundo paso, se copia el valor de la salud del enemigo dentro de la misma instancia. Y en el último paso, se serializa la instancia de “Mi Objeto De Guardado” en disco (En un archivo “DatosDeJuego-01.sav”) que ahora contiene los datos de salud del jugador y el enemigo. Como se observa, se hace uso del método “Save Game To Slot” perteneciente a la clase “Save Game Library” mencionada anteriormente.

Para el caso de cargado de juego, el algoritmo resulta similar, tal como se observa en la siguiente figura.

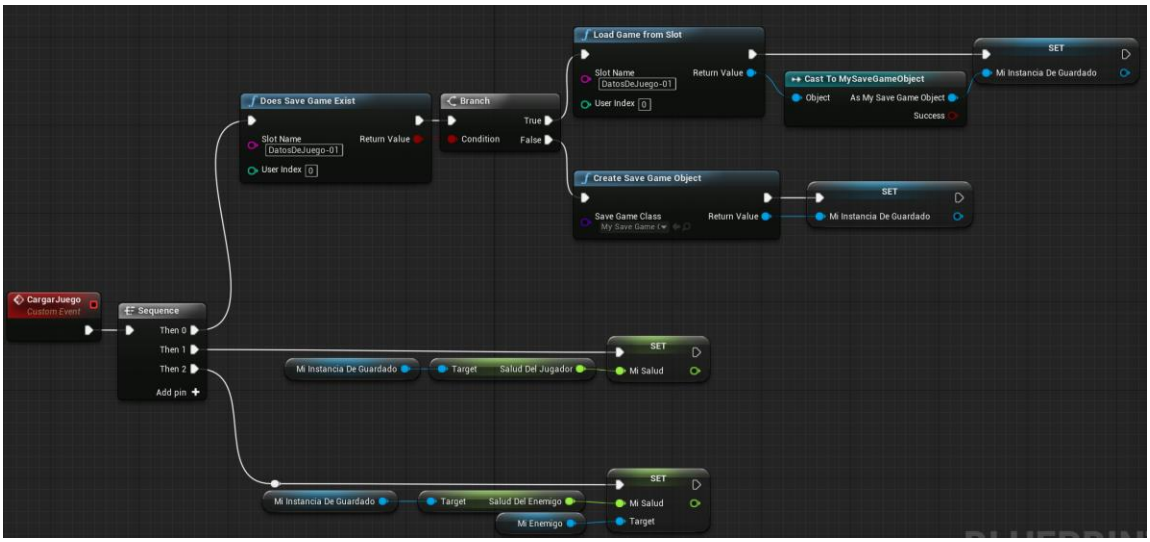


Figura: 18 Implementación de un Sistema de Cargado de Ejemplo

Fuente: Elaboración propia.

En el primer paso se carga el estado de juego (Desde un archivo “DatosDeJuego-01.sav”) asegurándose primero que el archivo exista en primer lugar mediante el uso del método “Does Save Game Exist” y en caso de que el archivo no exista se crea una instancia de “Save Game Object” por defecto mediante el método “Create Save Game Object”. En el segundo paso se carga la salud del jugador y en el último, la del enemigo. Como se observa, en este caso, se hace uso de otros métodos estáticos perteneciente a la clase “Save Game Library”.

Tomando en cuenta el ejemplo anterior, se tiene entonces un pequeño juego que implementa un sencillo sistema de guardado y cargado de partidas haciendo uso de las herramientas del motor.

Y como se observa en el diagrama de clases mostrado anteriormente, la clase “Jugador” tiene una dependencia tanto de “Enemigo” como de “Mi Objeto De Guardado”. La dependencia con “Mi Objeto De Guardado” es esperable (Ya que, en el ejemplo, la implementación del sistema de guardado se realiza en la clase “Jugador”). Sin embargo, la dependencia con la clase “Enemigo” es innecesaria y puede convertirse en un problema a medida que los requerimientos del videojuego crezcan y aumenten el número de dependencias. Esta dependencia ocurre debido a

que la implementación del sistema de guardado requiere de una referencia directa a la clase “Enemigo” para acceder a sus variables.

Si al ejemplo mostrado se añadieran además otros 2 tipos de enemigos (dos clases distintas), entonces se puede esperar que la implementación crezca en complejidad y que la clase “Jugador” contenga un mayor número de dependencias. Si el videojuego continúa creciendo y se empiezan a requerir almacenar aún más datos de los enemigos y otros objetos del juego (Como armas, accesorios, etc.) entonces la complejidad de un sistema de guardado y cargado puede crecer considerablemente hasta el punto de ser potencialmente insostenible en cuanto a su desarrollo y mantenibilidad. Aparte del incremento en complejidad, se requieren además cambios en la lógica de “Guardar Juego” y “Cargar Juego” para cada variable nueva que se requiera guardar y/o cargar (Además de la necesidad de recrear manualmente la nueva variable dentro de la clase “Mi Objeto De Guardado”). Y esto puede conllevar a situaciones donde se introduzcan errores en un algoritmo que funcionaba correctamente con anterioridad.

Además de los problemas mencionados, está el caso donde que se requieran guardar datos de una lista de instancias de objetos de una misma clase, ya que surge el reto de mantener el orden de guardado y cargado de los datos de forma que se mantenga la consistencia de estos mismos. En este caso, es necesario encontrar una manera de identificar a cada objeto, lo que resulta en un mayor incremento en la complejidad de implementación de un sistema de guardado. Además, la dificultad de esta identificación de objetos puede aumentar si el videojuego hace uso de más de un nivel de juego, ya que resultará importante además identificar a que nivel pertenece cada objeto en la lista.

Asimismo, la librería estática “Save Game Library” incluida por defecto en Unreal Engine 4 puede resultar muy poco flexible a los requerimientos de ciertos tipos de videojuego debido a

que no soporta la opción de guardar y cargar datos haciendo uso de directorios personalizados Esta librería tampoco cuenta con funciones asíncronas ni con funciones de compresión y descompresión de datos, por lo que su flexibilidad se ve aún más reducida de acuerdo a los posibles requerimientos que un videojuego pueda tener.

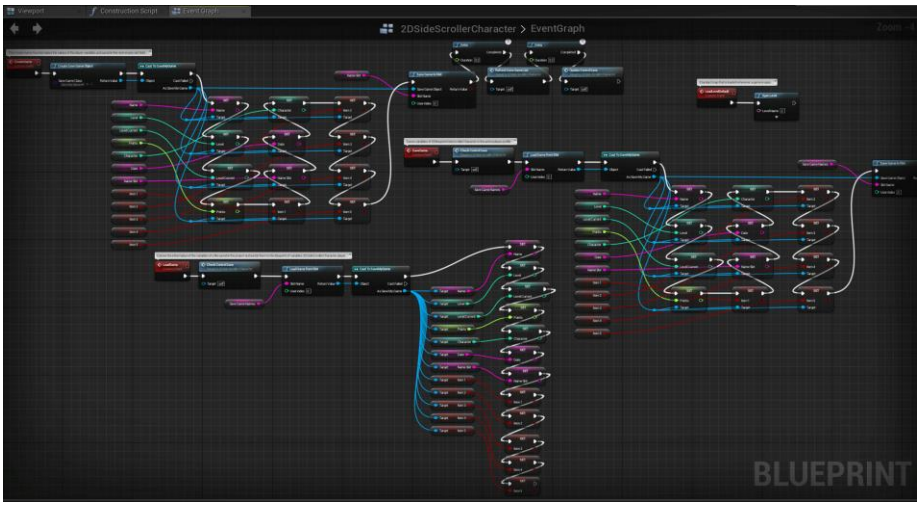


Figura: 19 Complejidad de un Sistema de Guardado en Blueprints

La implementación de un sistema de guardado incrementa su complejidad de acuerdo al número de variables que se requieran guardar y/o restaurar. Fuente: Elaboración propia.

Tomando entonces en cuenta el ejemplo realizado y haciendo uso de los requerimientos de software establecidos en esta investigación, así como las métricas de calidad de software definidas por la norma ISO/IEC 9126, se determina entonces la siguiente evaluación de la calidad de las herramientas de guardado y cargado de Unreal Engine 4.

3.1.3.2.1. Diagnóstico de acuerdo a su funcionalidad.

a. **Se debe permitir guardar el juego de forma tradicional mediante un menú**: Las herramientas incluidas en Unreal Engine 4 son capaces de permitir la implementación de un sistema de guardado tradicional por menú. Sin embargo, los problemas mencionados en el ejemplo se acarrean y su implementación puede resultar inviable de acuerdo a la complejidad de los requerimientos de guardado del videojuego.

b. **Se debe permitir guardar el juego de forma automática en "checkpoints"**: De igual forma, es posible implementar este tipo de sistema de guardado y cargado de partidas. Sin embargo, los problemas mencionados en el ejemplo se acarrean y su implementación puede resultar inviable de acuerdo a la complejidad de los requerimientos de guardado del videojuego.

c. **Se debe permitir guardar el juego de forma automática en intervalos de tiempo:** Igualmente, la viabilidad de implementación de este sistema de guardado depende de la complejidad del videojuego. Además, la implementación de un sistema de guardado por intervalos puede afectar el rendimiento del juego por lo que este aspecto será descrito en la comparación de resultados.

3.1.3.2.2. Diagnóstico de acuerdo a su usabilidad.

a. **Debe ser fácil de entender**: No es posible diagnosticar subjetivamente este aspecto de forma interna, pero se considera que las herramientas de guardado incluidas en Unreal Engine 4 son fáciles de entender, ya que por su nombre y por el pequeño número de clases expuestas (3 clases) resulta fácil deducir cual es su función y sus posibles aplicaciones.

b. **Debe ser fácil de aprender a usar**: No es posible diagnosticar subjetivamente este aspecto de forma interna. Pero como se observa en el ejemplo, se considera que las herramientas de guardado y cargado de Unreal Engine 4 son fáciles de aprender ya que cuenta con solamente 3 clases expuestas.

c. **Debe ser sencillo de operar**: No es posible diagnosticar subjetivamente este aspecto de forma interna. Pero como se observa en el ejemplo, se considera que la complejidad

de su uso es directamente proporcional al número de variables que se requieran guardar en un videojuego.

3.1.3.2.3. Diagnóstico de acuerdo a su eficiencia.

a. **Debe tener un tiempo de procesamiento aceptable**: No se diagnosticó este requerimiento en el ejemplo, pero se hará uso del mismo en la comparación de resultados. Sin embargo, se considera que la implementación de compresión y descompresión de datos en el plugin ayudará a obtener mejores resultados en términos de calidad de software.
b. **No tiene que afectar el rendimiento del juego:** No se diagnosticó este requerimiento en el ejemplo, pero se hará uso del mismo en la comparación de resultados. Sin embargo, se considera que la implementación de funciones asíncronas en el plugin ayudará a obtener mejores resultados en términos de calidad de software.

A continuación, se tiene las tablas de resumen para la evaluación inicial de la calidad de las herramientas de guardado y cargado de partidas incluidas en Unreal Engine 4.

3.1.3.2.4. Resultados del análisis de calidad.

Tabla 10
Evaluación de las Herramientas de Unreal Engine 4 (Por su Funcionalidad)

Indicador	Observaciones
Adecuación	Las herramientas incluidas en Unreal Engine 4 cumplen los requerimientos definidos a partir de las métricas de funcionalidad. Pero se deduce, a partir del ejemplo descrito en este análisis, que su operatividad puede ser inviable en caso de haber demasiadas variables.

Fuente: Elaboración propia.

Tabla 11
Evaluación de las Herramientas de Unreal Engine 4 (Por su Usabilidad)

Indicador	Observaciones
Entendimiento	No es posible diagnosticar este indicador de forma interna. Pero se deduce, a partir del ejemplo descrito en este análisis, que estas herramientas pueden ser fácilmente entendidas; ya que es posible discernir su función y sus posibles aplicaciones a partir de su nombre y sus clases expuestas.
Aprendizaje	No es posible diagnosticar este indicador de forma interna. Pero se deduce, a partir del ejemplo descrito en este análisis y de que UE4 expone solamente 3 clases relacionadas al guardado de juego, que estas herramientas son fáciles de aprender.
Operatividad	No es posible diagnosticar este indicador de forma interna. Pero como se observa en el ejemplo descrito en este análisis, se deduce que la complejidad del uso de las herramientas incrementa de forma proporcional al número de variables a guardar en un videojuego.

Fuente: Elaboración propia.

Tabla 12
Evaluación de las Herramientas de Unreal Engine 4 (Por su Eficiencia)

Indicador	Observaciones
Comportamiento de tiempos	No se realizó un diagnostico de esta métrica. Pero se deduce que la implementación de compresión de datos en el plugin a desarrollar ayudará a obtener mejores resultados en cuanto a esta métrica.
Utilización de recursos	No se realizó un diagnóstico de esta métrica. Pero se deduce que la implementación de funciones asíncronas en el plugin a desarrollar ayudará a obtener mejores resultados en cuanto a esta métrica.

Fuente: Elaboración propia.

Se observa, por tanto, que mediante un análisis inicial de las herramientas de guardado y cargado de partidas incluidas en Unreal Engine 4 se determinan los aspectos mejorables de los mismos. De esta forma, la información obtenida puede ser usada como guía para garantizar que el plugin a desarrollarse influencie de forma positiva en la calidad de las herramientas de guardado y cargado de partidas para videojuegos en Unreal Engine 4.

3.1.3.3. Dominio del Problema.

Habiéndose definido los requerimientos de alto nivel, así como establecido un punto de partida de calidad esperada, en esta sección se define el modelo del dominio del plugin a desarrollarse. Con esto se busca modelar un diseño inicial que defina posibles soluciones con el fin de garantizar una mejor calidad de software con respecto a las herramientas de guardado incluidas por defecto en el motor.

3.1.3.3.1. Diagrama de clases.

Se define el siguiente diagrama de clases de acuerdo a los requerimientos descritos. Este diagrama de clases describe el diseño del plugin que tiene como fin el solucionar los problemas

mencionados en el análisis de calidad de las herramientas de guardado de Unreal Engine 4. Para ello, se tiene como intención extender la arquitectura de las herramientas por defecto de Unreal Engine 4 con el fin de asegurar la compatibilidad con proyectos existentes.

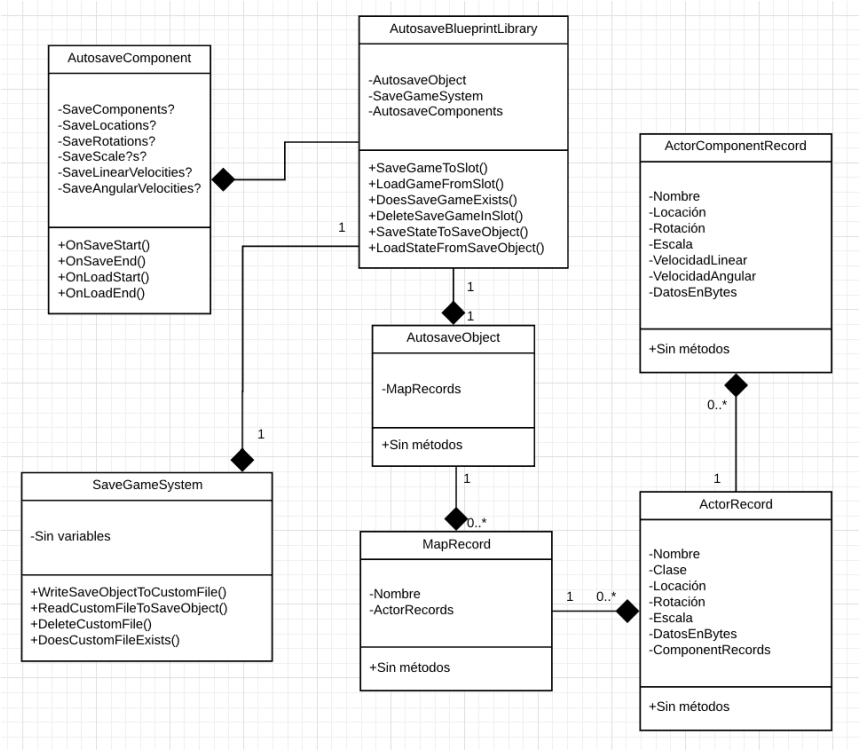


Figura: 20 Diagrama de Clases del Plugin

Fuente: Elaboración propia.

Se observan las clases “Autosave Library”, “Autosave Object” y “Save Game System”, las cuales cumplen propósitos similares a sus clases respectivas incluidas en el motor Unreal Engine 4. El resto de clases son extensiones que tienen el fin de automatizar tareas y reducir la complejidad de implementación de un sistema de guardado. Esto se consigue mediante la auto identificación de “Actors” (y sus componentes) en un nivel, los cuales son agrupados en un “Map Record”.

Asimismo, gran parte de la automatización se consigue mediante el auto reconocimiento de variables que requieren ser guardadas. Esto puede conseguirse activando la propiedad de variable “Save Game”, la cual permite identificar a aquellas variables que requieran ser serializadas. La

propiedad de variable “Save Game” es una propiedad que puede ser activada en cualquier tipo de variable (cadena, entero, etc.) y que viene implementada por defecto en Unreal Engine 4. Es importante tener en cuenta que esta propiedad o atributo de variable “Save Game” es tan solo un identificador y no es usado en ningún momento por las herramientas de guardado por defecto de Unreal Engine 4.

En esta investigación, se aprovecha la existencia de esta propiedad con el fin de simplificar la implementación de un sistema de guardado de manera que tan solo sea necesario activar la propiedad “Save Game” en una variable. De esta forma el plugin realiza el trabajo de guardar y restaurar las variables de un objeto de forma automática. Esta propiedad permite además acceder a las variables de una clase de forma agnóstica (sin requerir el tipo de clase), eliminando por tanto cualquier tipo de dependencia innecesaria. El acceso a variables de forma agnóstica es realizado mediante el uso del sistema de reflexión de Unreal Engine 4.

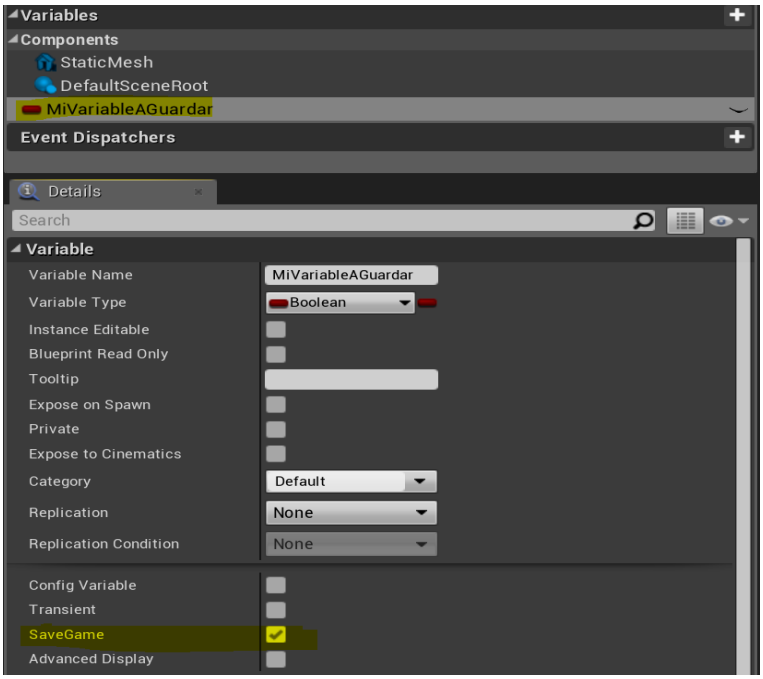


Figura: 21 Propiedad "Save Game" de las variables en Unreal Engine 4

Se observa que la variable “Mi Variable A Guardar” cuenta con la propiedad “Save Game”, la cual puede ser activada de forma que el plugin reconozca esta variable y la procese automáticamente. Fuente: Elaboración propia.

Habiéndose explicado de forma general el propósito y las soluciones que se intentan abarcar mediante la implementación del diagrama de clases propuesto; a continuación, se detalla el uso de cada clase:

a. **Map Record**: Es una estructura de datos que representa el estado de un mapa o nivel. En este caso, el nombre del nivel (identificador) así como un arreglo de "Actor Record" que representan los estados de cada "Actor" que pertenecen al nivel. Con esta información es posible guardar y/o restaurar el estado de todo un nivel y los actores que pertenecen al mismo.
b. **Actor Record:** Es una estructura de datos que representa el estado de un "Actor". Esta estructura almacena el nombre, clase, ubicación, rotación y tamaño de un Actor; así como sus datos (Variables con la propiedad "Save Game" activada) y un arreglo que representan los estados de cada "Actor Component" que pertenece al "Actor". Con esta información es posible guardar y/o restaurar el estado de un "Actor" y sus componentes.
c. **Actor Component Record:** Es una estructura de datos que representa el estado de un "Actor Component". Esta estructura almacena el nombre, ubicación relativa, rotación relativa, y tamaño relativo de un componente; así como sus datos (Variables con la propiedad "Save Game" activada) y el estado de su simulación de físicas (en el caso de que el Componente simule físicas). Con esta información es posible guardar y/o restaurar el estado de un "Actor Component".
d. **Autosave Object:** Es una clase con la habilidad de ser serializada y deserializada (Mediante el uso de los métodos de la clase "Autosave Library"). Esta clase contiene como variable un arreglo de "Map Record" que puede ser usado para guardar y/o

restaurar la información de varios niveles en un videojuego. Se planea que este arreglo sea manejado de forma automática por el plugin de manera que el usuario final no tenga que preocuparse por su implementación.

e. **Save Game System:** Esta clase implementa funciones que interactúan directamente con la plataforma de juego (PC, Android, etc.). Sus funciones son principalmente la de escribir datos de memoria a disco y leer datos de disco a memoria. Es usado por "Autosave Blueprint Library" principalmente para guardar el estado de un "Autosave Object" en un archivo en disco y para restaurar un "Autosave Object" a partir de un archivo de guardado.

f. **Autosave Component:** Este "Actor Component" permite no solo identificar a los "Actor" que un videojuego requiera guardar y/o restaurar, sino que también permite implementar un comportamiento por defecto para estos mismos. En otras palabras, un "Actor" que acople este componente será detectado por las funciones de guardado y cargado encontradas en el "Autosave Library". De esta forma, este "Actor" será guardado y/o restaurado automáticamente. Además, el uso de un "Actor Component" permite que el plugin pueda realizar operaciones por defecto en los "Actor". En este caso, permitir guardar la ubicación, rotación, tamaño y el estado de la simulación de físicas de los "Actor" de forma automática. Esta funcionalidad debería poder ser desactivada en caso de que no se requiera.

g. **Autosave Blueprints Library:** Esta clase contiene funciones estáticas accesibles desde Blueprints por lo que se considera la capa más externa (accesible por el usuario final). Estas funciones están diseñadas para que el usuario final pueda implementar un sistema de guardado y cargado de partidas en un videojuego de forma sencilla.

Además, funciones como “Save State To Save Object” y “Load State From Save Object” permiten guardar y/o restaurar el estado de un videojuego de forma automática por lo que reducen considerablemente la complejidad de implementación de un sistema de guardado.

3.1.3.3.2. Diagramas de flujos.

A continuación, se describen mediante diagramas de flujos las funciones más importantes mencionadas anteriormente. La modelación de diagramas de flujo para estas funciones tiene como propósito el detallar de mejor manera la comunicación que se realiza entre las clases involucradas, así como el de mejorar el entendimiento de estos mismos.

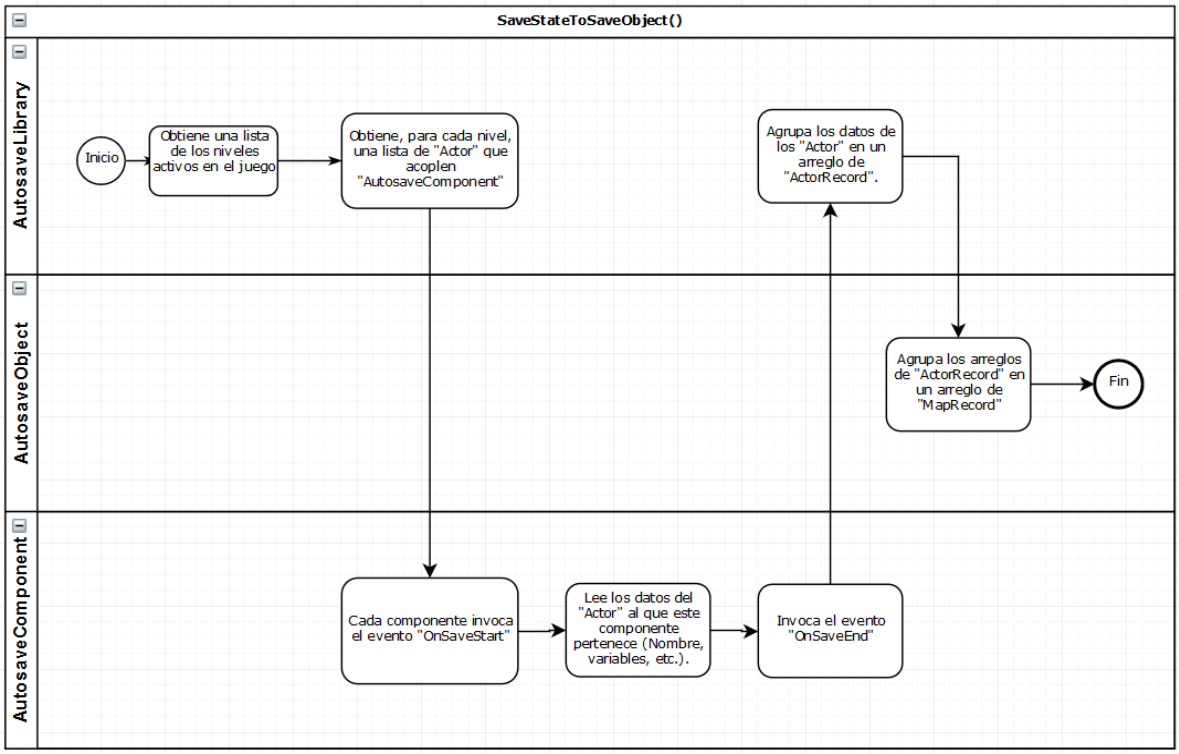


Figura: 22 Diagrama de Flujos para Guardar el Juego en Memoria

El diagrama de flujo de la función “Save State To Save Object”, el cual describe el proceso por el cual los datos de los niveles activos (y sus “Actor”) son almacenados en un arreglo de “Map Record” dentro de un objeto “Autosave Object”.

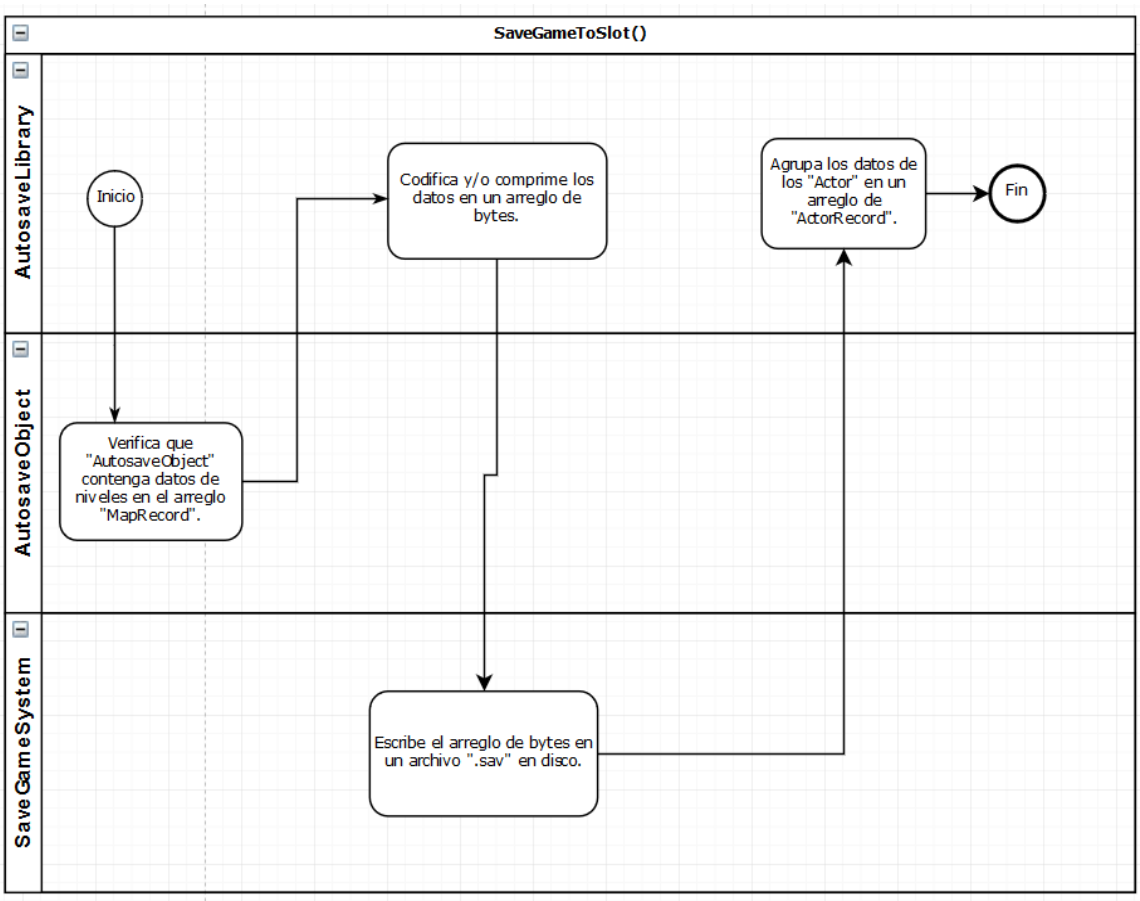


Figura: 23 Diagrama de flujo para Escribir los Datos en Disco

Fuente: Elaboración propia.

El diagrama de flujo de la función “Save Game To Slot”, el cual describe el proceso por el cual los datos de un “Save Object” son serializado en un archivo en disco.

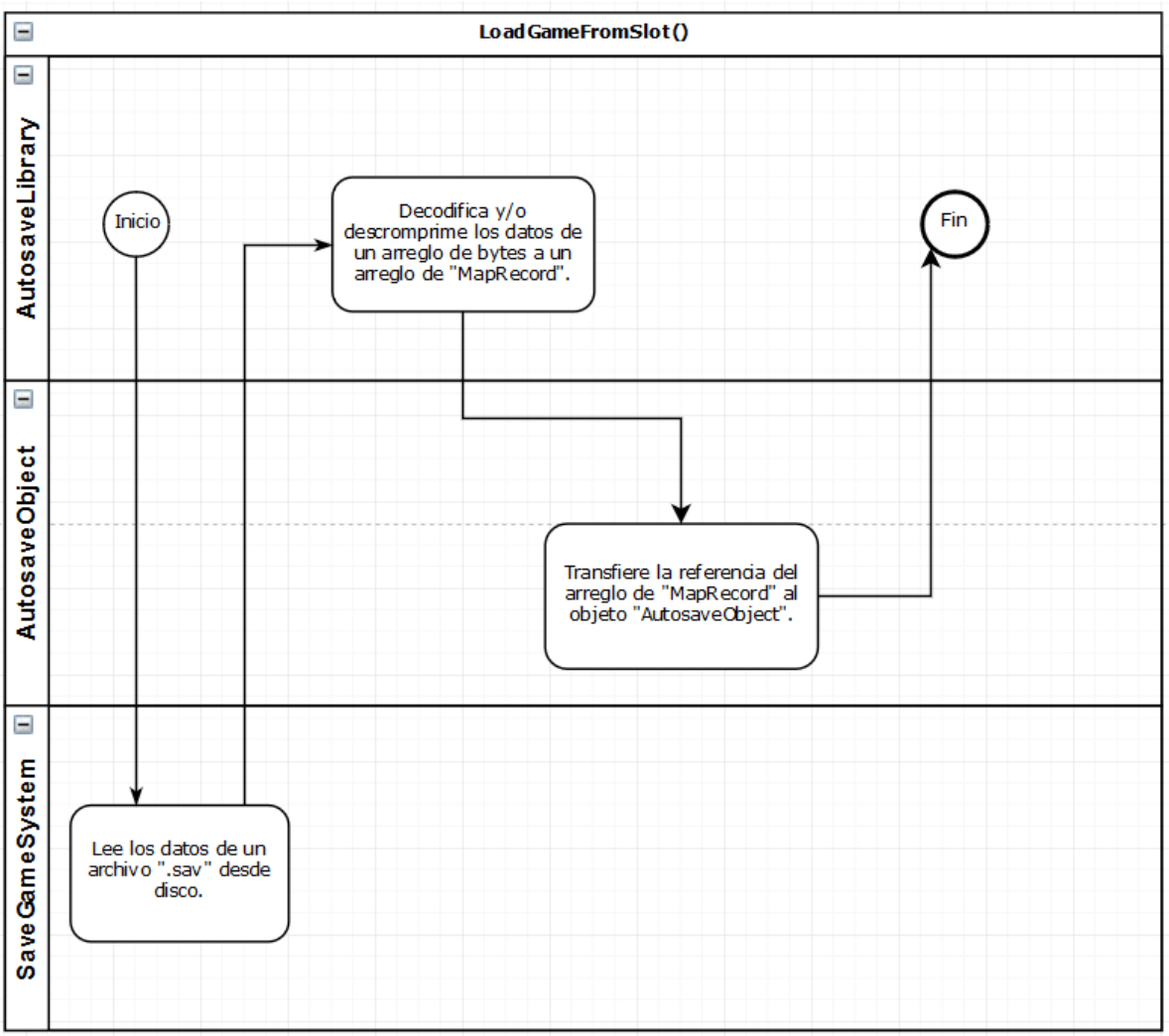


Figura: 24 Diagrama de Flujo para Leer los Datos de Disco

Fuente: Elaboración propia.

El diagrama de flujo de la función “Load Game From Slot”, el cual describe el proceso por el cual los datos de un “Autosave Object” son restaurados mediante el cargado de archivo de guardado existente.

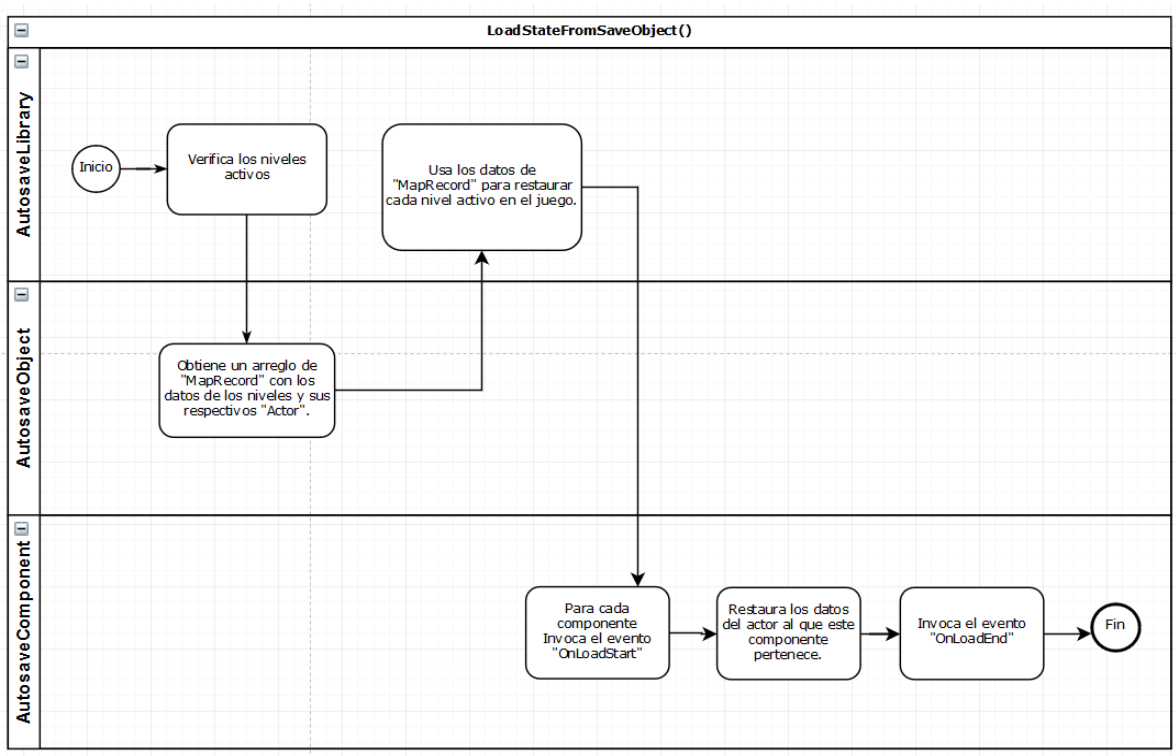


Figura: 25 Diagrama de Flujo para Restaurar el Estado de Juego

Fuente: Elaboración propia.

El diagrama de flujo de la función “Load State From Save Object”, el cual describe el proceso por el cual los estados de todos los niveles activos en un videojuego son restaurados haciendo uso de los datos encontrados en un “Autosave Object”.

3.1.4. Estimación General de Costos y Tiempos.

Como parte de la disciplina de gestión de proyectos, en esta sección se realiza una estimación inicial de costos y tiempos del proyecto.

3.1.4.1. Estimación de Costos del Proyecto.

La siguiente tabla describe la estimación de costos a requerirse para el desarrollo del plugin.

Tabla 13
Estimación de Costos del Proyecto

Descripción	Costo mensual (S/.)	Tiempo (Meses)	Total (S/.)
Licencia de Unreal Engine 4	Gratuito	2	-
Licencia de Azure DevOps	Gratuito para proyectos pequeños	2	-
Licencia de Visual Studio	Gratuito	2	-
Honorarios/Costo de oportunidad	930 (2 personas)	2	3720.00
		TOTAL	**3720.00**

Fuente: Elaboración Propia.

3.1.4.2. Estimación de Tiempos.

A continuación, se describe la estimación inicial de actividades de forma general, así como sus respectivos tiempos estimados.

		Name	Duration
1		**TESIS COMPLETA**	62days
2		**ESTIMACIÓN DE LA CALIDAD DE LAS HERRAMIENTAS DE UE4 (ISO 9126)**	3days
3		Descripción de los requerimientos de alto nivel	1day
4		Descripción de las historias de usuario	1day
5		Estimación de calidad bajo las métricas relevantes (ISO 9126)	2days
6		**DESARROLLO DEL PLUGIN EN UNREAL ENGINE 4**	59days
7		**FASE DE CONCEPCIÓN (1-3 días aprox.)**	1day
8		Definición del problema y objetivos del proyecto	1day
9		Definición inicial del entorno de trabajo (VSTS)	1day
10		Definición de los requerimientos de alto nivel	1day
11		Definición de las historias de usuario	1day
12		Estimación inicial de costos y tiempos	1day
13		Determinación de la factibilidad del proyecto	1day
14		**FASE DE ELABORACIÓN (1-7 días aprox.)**	7days
15		Estimación de tareas a realizar (Brainstorming)	1day
16		Re-evaluación de los requerimientos	1day
17		Modelación de la arquitectura (BPMN)	3days
18		Definición de los casos de prueba de alto nivel	1day
19		Creación inicial de tareas en Kanban (Backlog inicial)	1day
20		Ajuste de la estimación de costos y tiempos	1day
21		**FASE DE CONSTRUCCIÓN**	40days
22		Preparación del entorno de trabajo	3days
23		Preparación del entorno de pruebas	1day
24		Desarrollo del prototipo inicial	7days
25		Re-modelación de la arquitectura (BPMN)	1day
26		**ITERACIÓN**	31days
27		Creacion de tareas en Kanban	31days
28		Desarrollo del plugin	31days
29		Desarrollo de los casos de prueba	31days
30		Evaluación de satisfacción de requerimientos	31days
31		Re-modelación de la arquitectura (BPMN)	31days
32		**FASE DE TRANSICIÓN**	11days
33		Documentación relevante del proyecto	11days
34		Pruebas finales	2days
35		Lanzamiento del plugin (UE4 Marketplace)	1day
36		Mantenimiento	0day
37		**EVALUACIÓN DE LA CALIDAD DEL SOFTWARE (ISO 9126)**	3days
38		Evaluación de la calidad de las herramientas de UE4	1day
39		Evaluación de la calidad de las herramientas desarrolladas	1day
40		Comparación, resultados y comentarios	1day

Figura: 26 Estimación de Tiempos del Proyecto

Fuente: Elaboración propia.

3.2. Fase de Elaboración

En esta fase se busca poner a prueba el diseño y entendimiento general del software mediante el desarrollo de un prototipo inicial del mismo.

3.2.1. Plan de Actividades.

El siguiente tablero Kanban describe las actividades a realizarse durante la presente fase, se observan actividades en la columna de “Pendiente” y “En Proceso”. Ya que durante esta fase se busca desarrollar un prototipo inicial del software, no se pretende desarrollar una versión usable del plugin (por el usuario final) sino una versión que permita determinar la capacidad del diseño del software para satisfacer los requerimientos del proyecto y, en caso de que se encuentren problemas durante el desarrollo, realizar ajustes en el diseño.

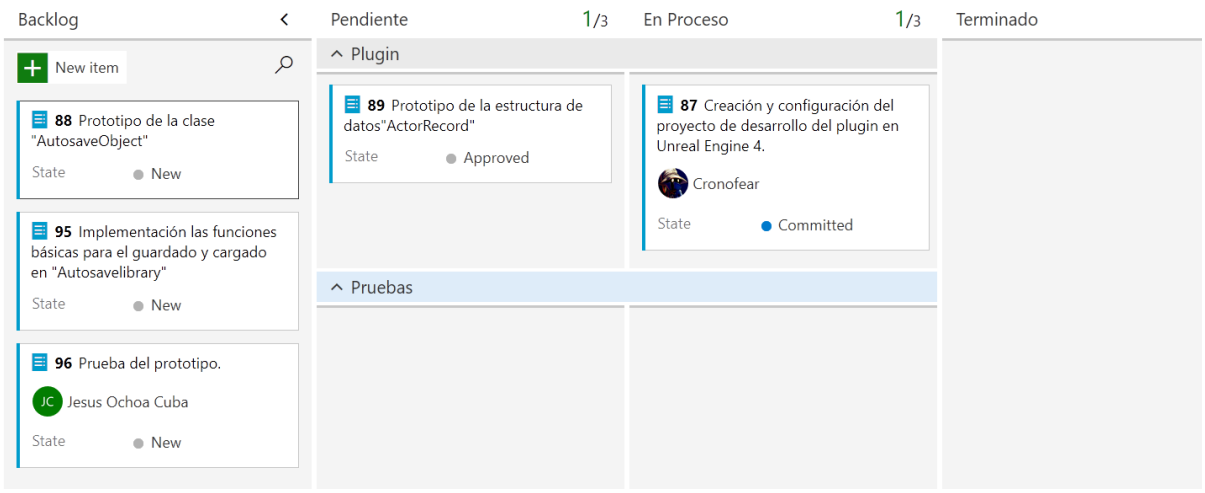


Figura: 27 Plan de Actividades del Proyecto (Prototipo)

Fuente: Elaboración propia.

3.2.2. Construcción del Prototipo.

Durante la presente sección se describen de forma general los procesos realizados para la construcción del prototipo inicial del plugin.

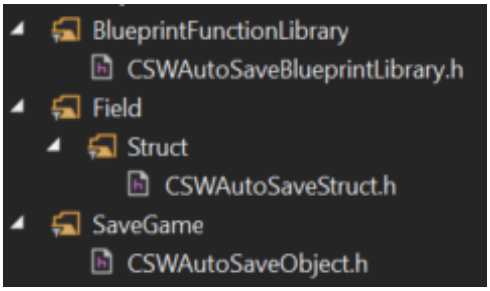


Figura: 28 Clases del Proyecto (Prototipo)

Fuente: Elaboración propia.

Para empezar, se crea la estructura de datos “Actor Record” dentro del archivo “Autosave Struct”, el cual será usado para agrupar las estructuras de datos requeridas para el proyecto. Esta versión inicial de “Actor Record” no contará con un arreglo de “Actor Component Record” ni pertenecerá a un “Map Record”, ya que no se busca poner a prueba la forma en la que el plugin mantiene la consistencia de datos durante esta fase.

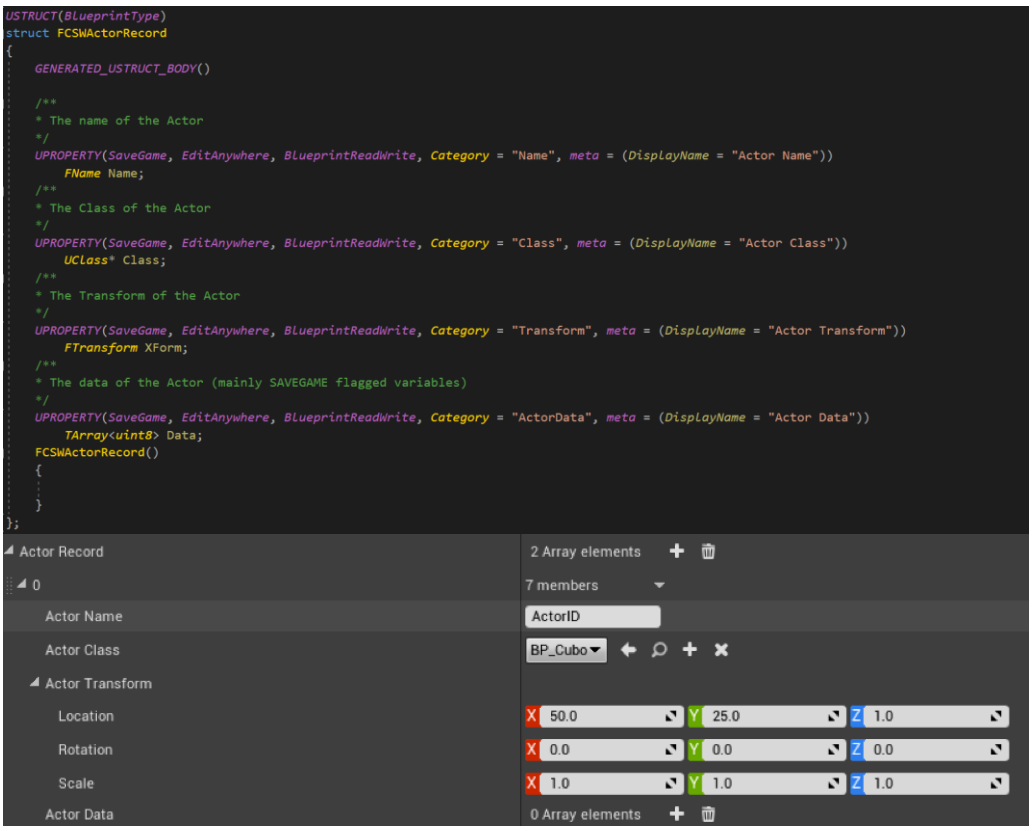


Figura: 29 Implementación de la Estructura de Datos (Prototipo)

Implementación de la estructura de datos “Actor Record” (en C++ y su representación en Blueprints) haciendo uso de las Macros de UE4 que facilitan la accesibilidad de esta estructura en el motor. Fuente: Elaboración propia.

Asimismo, se observa en la figura anterior el uso de Macros del “framework” de Unreal Engine los cuales son usados para facilitar la accesibilidad de las clases dentro del motor. Debido la alta verbosidad de estos Macros, se tratará de evitar los detalles de su uso con el fin de facilitar la lectura de esta investigación. Sin embargo, es importante mencionar el uso de la propiedad-Macro “SaveGame” en las variables, el cual define que dichas variables van a ser procesadas guardadas y/o cargadas automáticamente por el plugin.

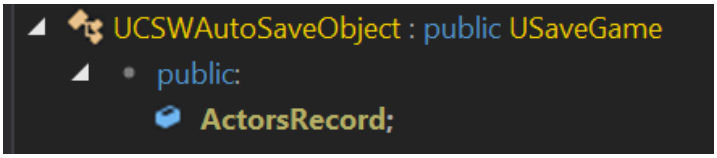


Figura: 30 Implementación de la Clase "Autosave Object"

La clase “Autosave Object” contiene una variable “Actors Record”, la cual será usada para almacenar el estado de varios “Actor” de forma automática. Fuente: Elaboración propia.

Luego se procede a implementar la versión inicial de la clase “Autosave Object”, la cual contendrá un arreglo de “Actor Record”. Esta clase hereda de la clase interna de UE4 “Save Object”. De esta forma se garantiza la compatibilidad con las herramientas de guardado existente en el motor.

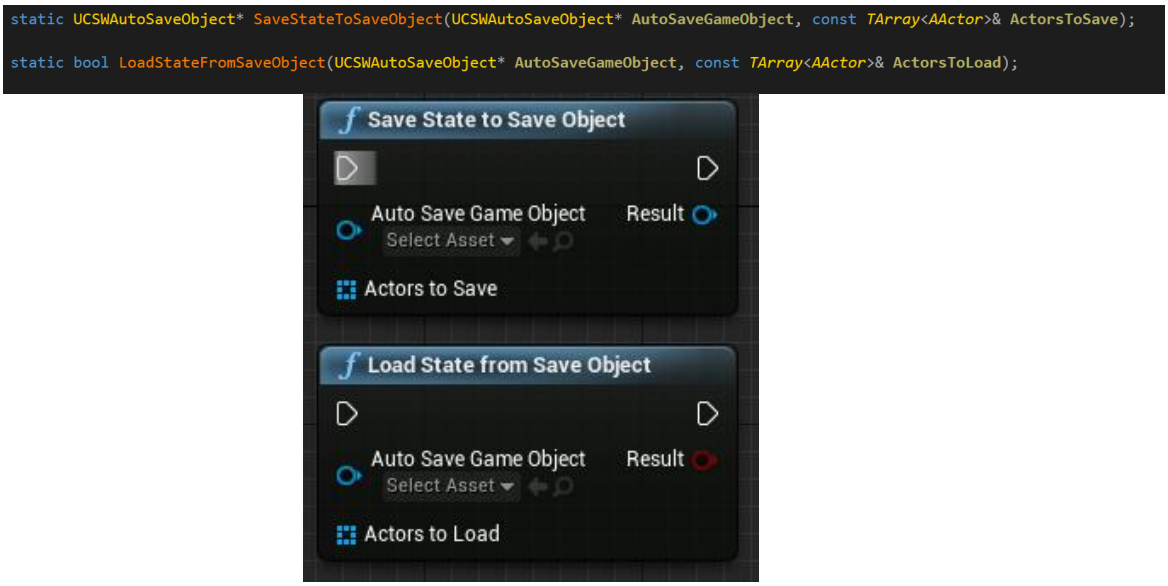


Figura: 31 Funciones para Guardado y Cargado de Partidas en Memoria

Fuente: Elaboración propia.

Finalmente, en la librería “Autosave Library” se implementan las versiones iniciales de las funciones “Save State To Save Object” y “Load State From Save Object”. La función “Save State To Save Object” permite guardar una lista de “Actors” y sus variables marcadas con la propiedad “SaveGame” en un objeto “Autosave Object”. Mientras que “Load State From Game Object” hace uso de una instancia de “Autosave Object” para restaurar el estado de los “Actors” de la lista. Estas funciones procesan estos “Actors” de forma que se accedan a sus nombres únicos, tipos de clase y sus “Transform” (ubicación, rotación y tamaño); además de las variables de estos “Actors” marcadas con la propiedad “SaveGame”.

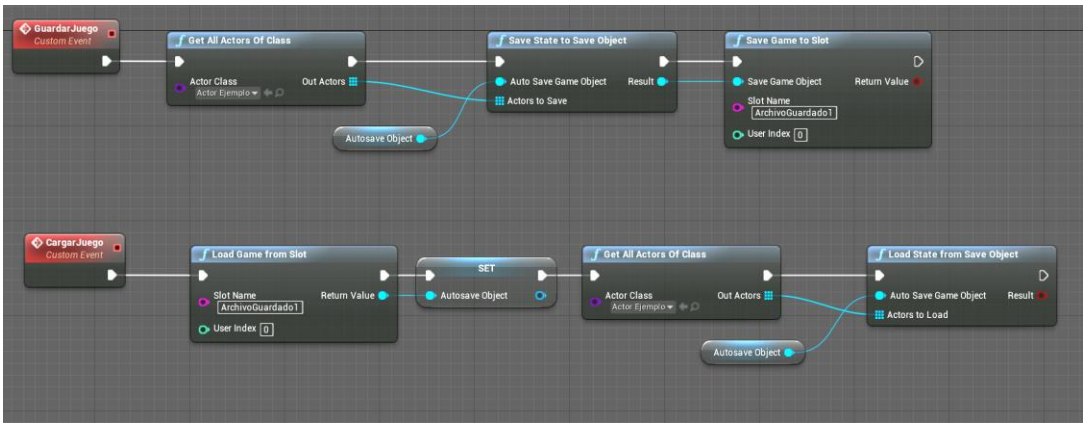


Figura: 32 Implementación de un Sistema de Guardado Usando el Plugin

Sistema de guardado para las instancias de “Actor Ejemplo” donde se guarda y restaura la variable “Ubicación” para cada instancia. Fuente: Elaboración propia.

A continuación, se pone a prueba el prototipo del plugin desarrollado mediante la implementación de un sistema de guardado. Para esta prueba, se crea una clase “Actor Ejemplo” la cual contiene una variable “Ubicación” que tiene activada la propiedad “Save Game”. Se crean 1000 instancias de esta clase en un nivel de prueba.

Como se observa en la figura anterior, la implementación del algoritmo de guardado y cargado no requiere de un acceso directo a la variable “Ubicación”. El arreglo de 1000 instancias de “Actor Ejemplo” es procesado internamente dentro de las funciones “Save State To Save

Object" y "Load State From Save Object". De esta forma, los estados de estas instancias son guardadas y restauradas automáticamente haciendo uso de una instancia de "Autosave Object".

Finalmente, se confirma que el prototipo funciona correctamente y que el diseño del software es viable.

3.3. Fase de Construcción

En este capítulo se describen las iteraciones realizadas durante el desarrollo del plugin.

3.3.1. Primera Iteración.

Durante esta iteración se busca conseguir una versión beta del software, la cual pueda ser desplegada como plugin. De esta manera, se busca poner a prueba los requerimientos definidos en el proyecto.

3.3.1.1. Plan de Actividades.

El siguiente tablero Kanban describe las actividades a realizarse durante la primera iteración del desarrollo, las cuales se centrarán en la implementación del resto de clases y funcionalidades esenciales del plugin, de forma que sea posible poner a prueba los requerimientos del software

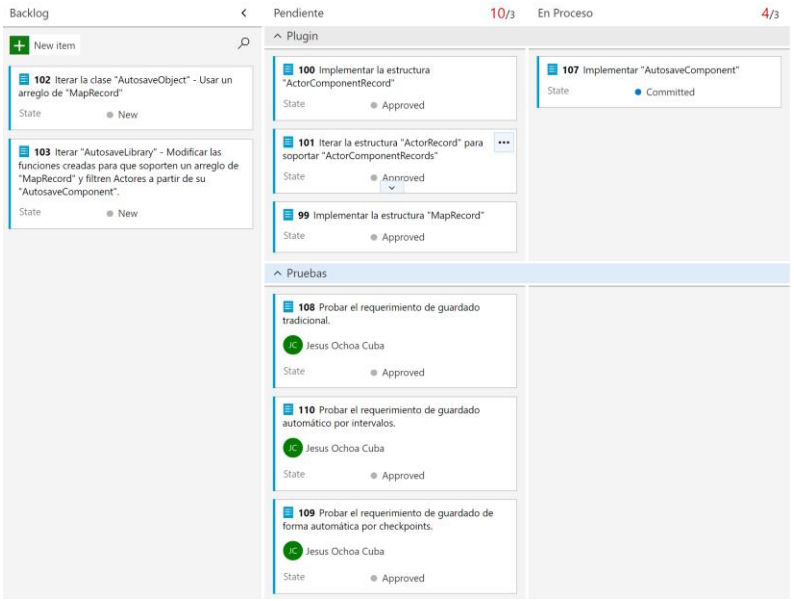


Figura: 33 Plan de Actividades del Proyecto (Primera Iteración)

Fuente: Elaboración propia.

3.3.1.2. Implementación.

En esta fase se busca conseguir una versión del plugin que pueda ser probada mediante casos de prueba de funcionalidad. Debido a ello, esta fase no busca aún implementar características que busquen mejorar aspectos relacionados a la usabilidad y eficiencia del plugin.

```
#pragma region CustomFunction
    public:
        OnSaveStart(const UCSWAutoSaveObject* CSWAutoSaveObject);
        OnSaveEnd(const UCSWAutoSaveObject* CSWAutoSaveObject);
        OnLoadStart(const UCSWAutoSaveObject* CSWAutoSaveObject);
        OnLoadEnd(const UCSWAutoSaveObject* CSWAutoSaveObject);
        OnBeginDestroyUnsavedActor(const UCSWAutoSaveObject* CSWAutoSaveObject);
        OnUnchangedActor(const UCSWAutoSaveObject* CSWAutoSaveObject);
    #pragma endregion
#pragma region Variables
    private:
        bEnable = true;
        bSaved = false;
        bDestroyOnLoad = false;
        bRandomID = false;
        bSaveComps = false;
        bSaveLocs = true;
        bSaveRots = true;
        bSaveScals = true;
        bSaveLVel = true;
        bSaveAVel = true;
        Optns;
```

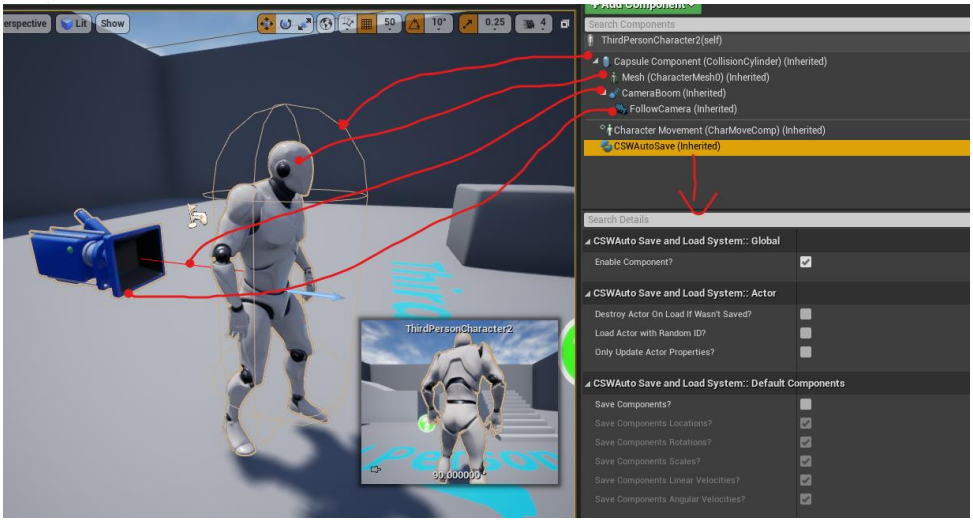


Figura: 34 Componente "Autosave Component"

Eventos y variables de la clase “Autosave Component”, así como su representación en Blueprints. Estos parámetros son usados por las funciones de la librería “Autosave Blueprint Library” para personalizar como un Actor va a ser guardado y/o restaurado. Fuente: Elaboración propia.

Se implementa la clase “Autosave Component” que permitirá acoplarse a los “Actors” que requieran ser guardados y/o restaurados en el juego. Este componente permitirá identificar a los “Actors” relevantes para un sistema de guardado, así como exponer parámetros y eventos de forma que el usuario final (desarrollador de juego) pueda tener un mejor control con respecto a cómo implementar el sistema de guardado.

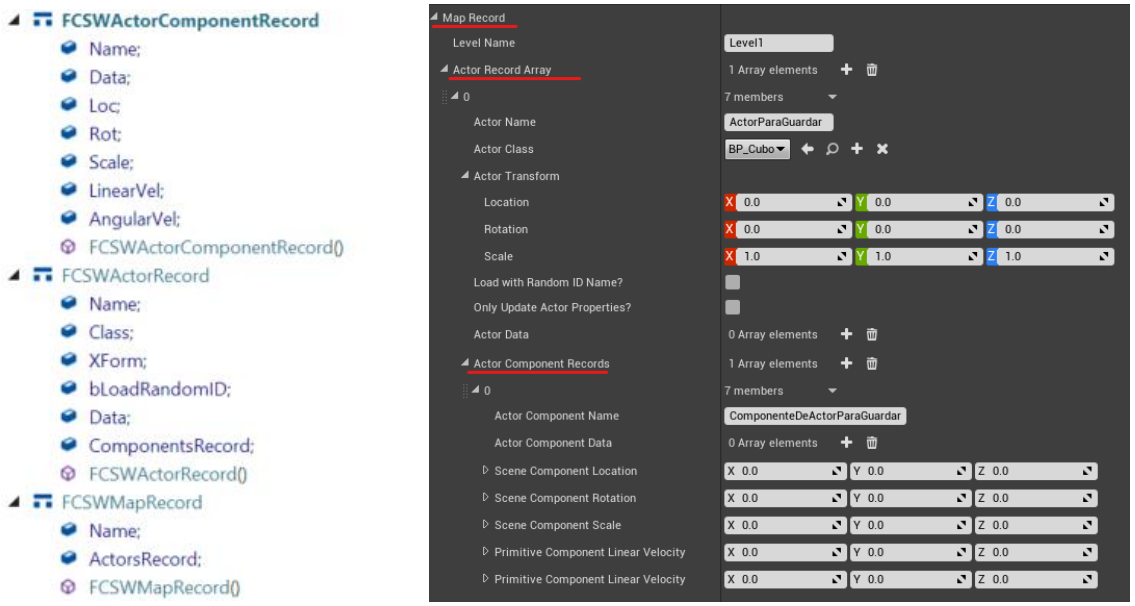


Figura: 35 Estructura de Datos (Primera Iteración)

Definición de las estructuras de datos que son usadas para almacenar los datos de varios niveles de juego, así como su representación en Blueprints. Fuente: Elaboración propia.

Luego se implementan las estructuras de datos que serán usadas para almacenar datos de niveles (“Map Record”). Estos a su vez contienen un arreglo de “Actor Record”, los que a su vez contienen un arreglo de “Actor Component Record”.

```
UCSWAutoSaveObject : public USaveGame
  public:
    LevelsRecord;
    GetContainsSaveData()
    GetNumberOfLevelsStored()
    GetNumberOfActorsStoredInLevel(int32 index)
    GetTotalNumberOfActorsStored()
```

Figura: 36 Clase "Autosave Object"

Definición de la clase “Autosave Object”. Se observa el arreglo “Map Record” así como funciones públicas que exponen información sobre los datos almacenados. Fuente: Elaboración propia.

Habiéndose implementado las estructuras de datos, se itera la clase “Autosave Object” de forma que contenga un arreglo de “Map Record”, en vez de un arreglo de “Actor Record” como lo hacía en su versión de prototipo. Asimismo, se crean funciones públicas de forma que se exponga información sobre los datos contenidos en el arreglo de “Map Record”, tales como el número de niveles contenidos, número de “Actors” contenidos en un nivel, etc.

Figura: 37 Funciones Estáticas del Plugin

Funciones de la clase “Autosave Blueprint Library” y su representación en Blueprints. Fuente: Elaboración propia.

Finalmente, se iteran las funciones de la librería “Autosave Blueprint Library” de forma que hagan uso de los cambios realizados. De esta forma, esta librería permitirá la implementación de sistemas de guardado mediante la detección automática de los “Actors” relevantes siempre y cuando estos acoplen el componente “Autosave Component”.

3.3.1.3. Pruebas de Requerimientos.

En esta sección se ponen a prueba los requerimientos mediante casos de prueba funcionales. Para ello se hace uso del proyecto de pruebas creado con anterioridad, en el que se pondrán a prueba los requerimientos de funcionalidad del proyecto.

Figura: 38 Ejemplo de Menú de Guardado de Partidas

Un sencillo menú para el guardado de partidas. Cada slot almacena información de la posición del jugador y el tiempo de juego. Fuente: Elaboración propia.

Se pone a prueba el plugin mediante la implementación de un sistema de guardado tradicional por menú. Para ello se crea una interfaz de guardado, así como sus consiguientes botones de guardado y cargado. Estos botones serán usados para llamar a las funciones expuestas en la librería “Autosave Blueprint Library”.

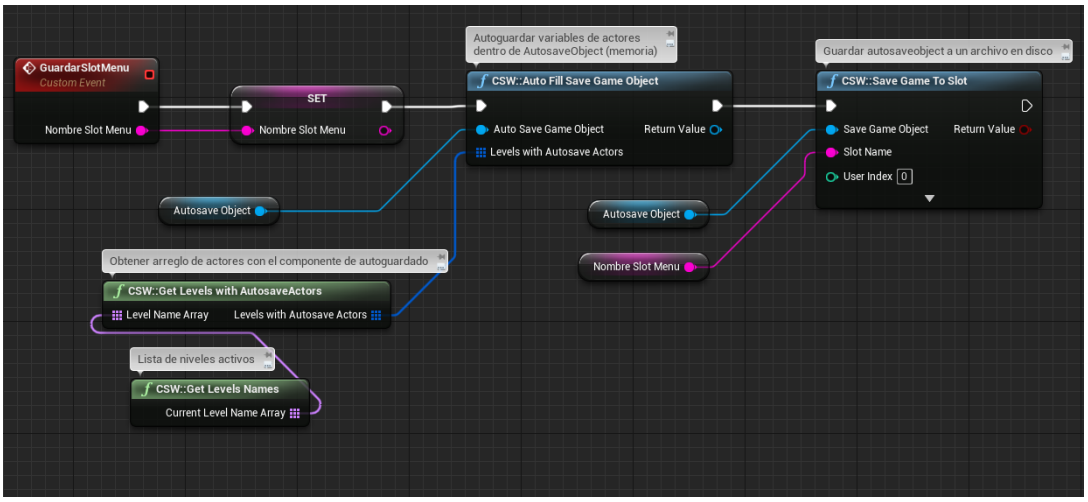


Figura: 39 Implementación de un Sistema de Guardado por Menú

La implementación del sistema de guardado que es ejecutado al presionarse los botones de la interfaz de usuario. Fuente: Elaboración propia.

Al presionarse el botón se llama entonces a la función de guardado (función de cargado en caso del menú de cargado de partidas) la cual llama a las funciones expuestas por el plugin. Siempre y cuando las variables a guardarse estén contenidas en “Actors” que acoplen el componente “Autosave Component”, estos datos se guardarán y restaurarán automáticamente con una sencilla implementación del sistema de guardado, tal y como se muestra en la figura anterior.

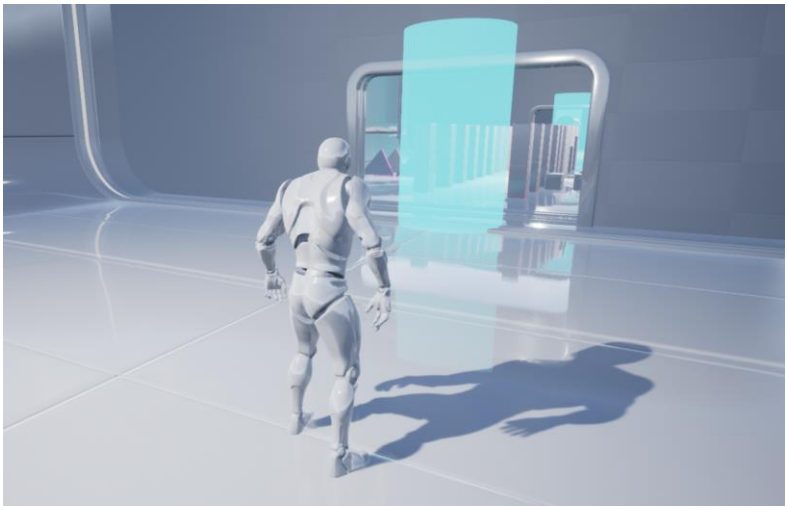


Figura: 40 Ejemplo de un Sistema de Guardado por "Checkpoints"

Las instancias del Actor “Save Spot” invocarán al evento de guardado en cuanto detecten colisión con el jugador. Fuente: Elaboración propia.

En cuanto a la implementación de un sistema de guardado por “checkpoints”, se crea los Actores “Save Spot” (volúmenes transparentes) los cuales llamarán al evento de guardado cada vez que el jugador colisione con el volumen; mientras que el evento de cargado será llamado en caso el jugador pierda o decida reiniciar la partida.

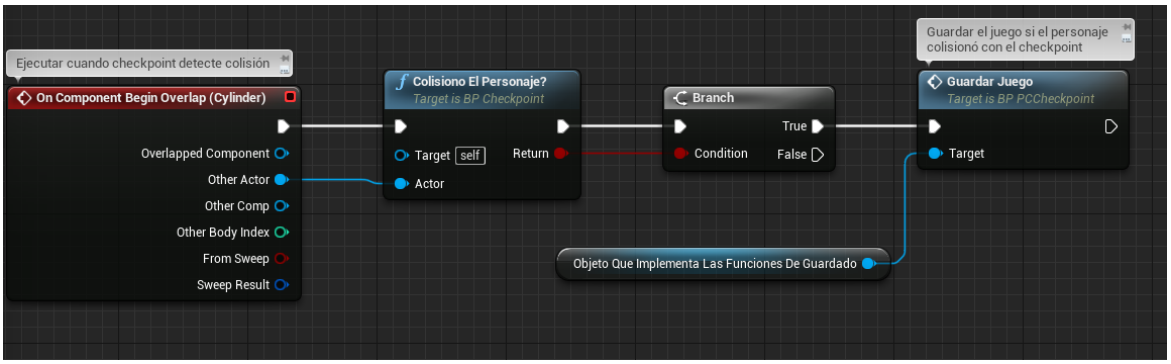


Figura: 41 Implementación de un Sistema de Guardado por "Checkpoints"

Implementación de “Save Spot” el cual verifica si el jugador colisiona con su volumen y en caso de que así sea llama al evento de guardado de juego. Fuente: Elaboración propia.

Bajo esta premisa, queda claro que la implementación del sistema de guardado mostrado en la sección anterior no sufrirá demasiados cambios en implementaciones posteriores, sino que serán los invocadores a estos eventos los que sufran variaciones. En el caso del sistema de guardado por menú, es el jugador el que llama a estos eventos cada vez que los botones de guardado/cargado son presionados. En cambio, en el sistema de guardado por “checkpoints”, será el Actor “Save Spot” el que invoque el evento de guardado, mientras que el evento de cargado será invocado cuando el jugador pierda la partida.

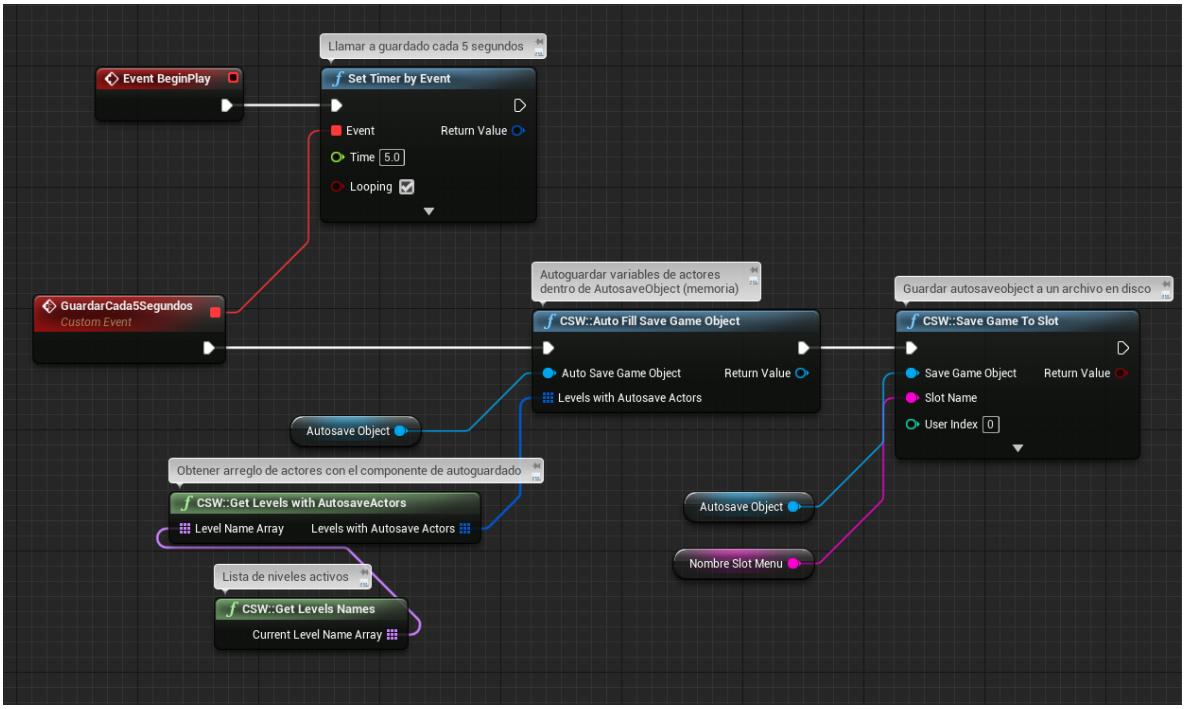


Figura: 42 Implementación de un Sistema de Guardado por Intervalo de Tiempo

El "Timer" inicializado al empezar el juego invoca al evento de guardado cada 5 segundos. Fuente: Elaboración propia.

Para la prueba de guardado automático se crea un "Actor" llamado "Automatic Save" que guarda el juego en intervalos de tiempo, en este caso la lógica de guardado es invocada por un "Timer" que se ejecuta cada 5 segundos. Para poner a prueba los casos previamente descritos, se instancian 1000 actores en el mundo que requieren guardar su estado en el mundo (posición, rotación, velocidad linear y velocidad angular). Estos Actores tan solo requieren acoplar el componente "Autosave Component" y modificar los parámetros expuestos de acuerdo lo requerido.

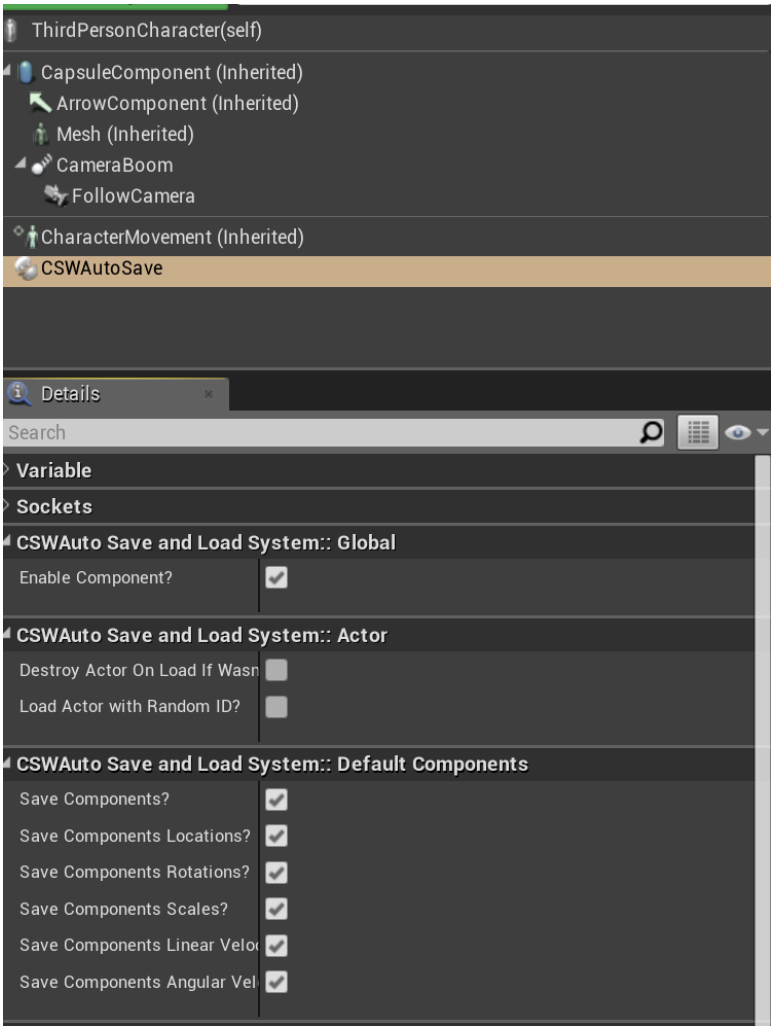


Figura: 43 Ejemplo de Uso del Componente "Autosave Component"

Los “Actors” que requieran ser guardados y restaurados pueden acoplar el componente “Autosave Component” y configurar sus parámetros para definir cómo van a ser gestionados por el sistema de guardado. Fuente: Elaboración propia.

Finalmente, se observan los siguientes resultados. Los cuales permiten observar cómo las funciones de guardado y cargado en un gran número de Actores pueden afectar la usabilidad y eficiencia del plugin. Esto puede no resultar muy relevante en el caso de un sistema de guardado manual o por “checkpoints”, pero puede ser muy importante para el caso de un sistema de guardado automático ya que, si el intervalo de guardado es muy corto, la experiencia de juego podría verse considerablemente afectada.

Tabla 14
Resultados de las Pruebas (Primera Iteración)

Acción	Tiempo promedio de renderizado	Ejecución Síncrona (Plugin y UE4)
Guardado 1000 Actores	~8ms (120fps)	~106ms (9fps)
Cargado 1000 Actores	~8ms (120fps)	~90ms (11fps)

Fuente: Elaboración Propia.

Como se observa en la tabla anterior, tanto el plugin como las herramientas incluidas en UE4 reducen considerablemente la velocidad de juego mientras el guardado y/o cargado de juego se procesa. Esto debido a que, en este punto, ambas herramientas funcionan de forma prácticamente igual en las funciones más importantes de guardado y cargado de partidas de juego. Asimismo, se considera que comparar las métricas de funcionabilidad y usabilidad en esta etapa de desarrollo no es muy útil, y dicha comparación será realizada al finalizar el desarrollo del plugin.

Se concluye entonces que el plugin puede (y debe) pasar a una siguiente iteración del desarrollo donde se buscará implementar funcionalidades enfocadas a su usabilidad y eficiencia.

3.3.1.4. Ajustes.

En cuanto a su funcionalidad, se observa que el plugin es capaz de cumplir con los requerimientos descritos.

En cuanto a su usabilidad, se observa que el plugin es fácil de operar y es aplicable a un entorno de producción, aunque se requieren mejoras en cuanto a su aplicación en juegos que puedan demandar una mayor cantidad de recursos en cortos periodos de tiempo.

En cuanto a su eficiencia, se observa que el plugin es aplicable en muchas configuraciones de juego, pero un sistema de guardado automático podría demandar demasiados recursos.

3.3.2. Segunda Iteración.

Esta iteración busca desarrollar una versión desplegable del plugin, de forma que pueda ser usada por desarrolladores de videojuegos para que estos implementen sus propios sistemas de guardado y cargado de partidas de juego.

3.3.2.1. Plan de Actividades.

El plan de actividades de la segunda iteración, la cual busca principalmente mejorar la usabilidad y eficiencia del plugin.

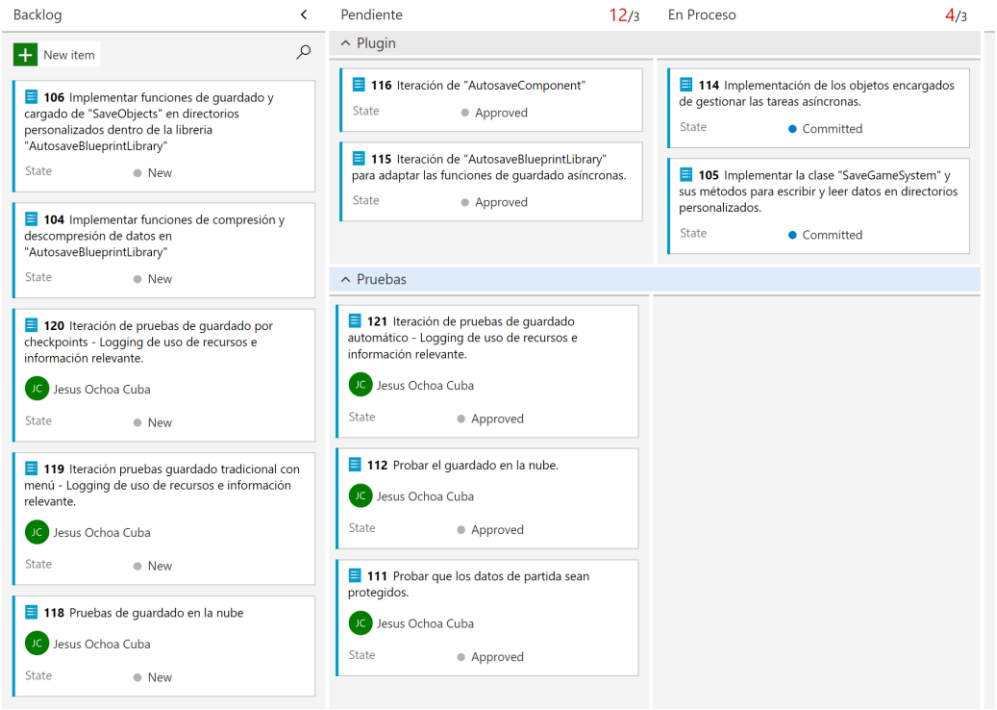


Figura: 44 Plan de Actividades del Proyecto (Segunda Iteración)

Fuente: Elaboración propia.

3.3.2.2. Implementación.

En esta fase se busca mejorar la eficiencia del plugin mediante la implementación de funciones asíncronas, las cuales procesarán el guardado y/o cargado de juego en intervalos largos de tiempo.

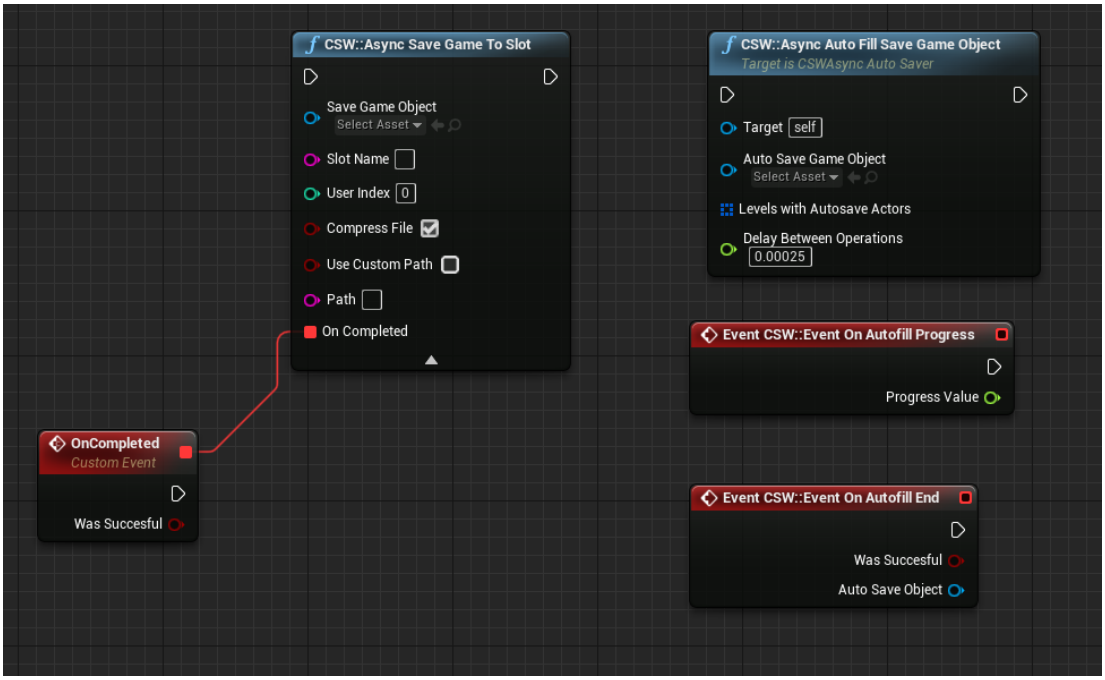


Figura: 45 Funciones de Guardado Asíncrono

A diferencia de las funciones síncronas, las funciones asíncronas invocan a su evento vinculado cuando finalizan, lo cual puede tomar más de un cuadro de juego. Fuente: Elaboración propia.

Las funciones implementadas de forma asíncrona son las que se observan que utilizan una mayor cantidad de recursos durante las pruebas realizadas en la primera iteración. Estas funciones son “Save Game To Slot”, “Load Game From Slot”, “Save State To Save Object” (“Autofill Save”) y “Load State From Save Object” (“Autoload Save”).

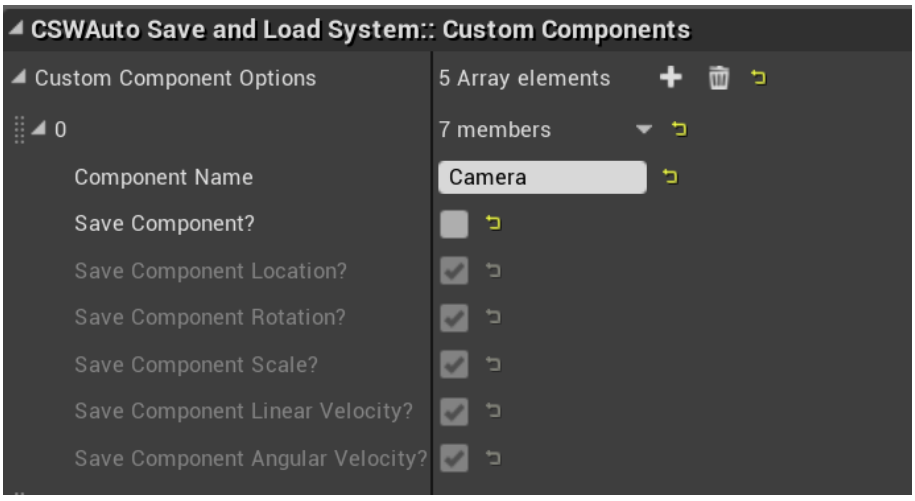


Figura: 46 Opciones Expuestas del Componente "Autosave Component"

Fuente: Elaboración propia.

Asimismo, se realizan modificaciones en la clase “Autosave Component” de forma que el usuario final pueda tener un mayor control sobre que componentes guardar para cada instancia de Actor.

Figura: 47 Definición de la clase "Save Game System"

La clase “Save Game System” implementa funciones para la gestión de archivos. Fuente: Elaboración propia.

En cuanto a la implementación de la clase “Save Game System”, se crean funciones que permitan escribir y leer datos en directorios personalizados, así como la posibilidad de comprimir y/o descomprimir estos mismos.

3.3.2.3. Pruebas de Requerimientos.

Las pruebas realizadas durante esta iteración hacen uso de las funciones asíncronas implementadas, así como la habilidad de comprimir y/o descomprimir datos.

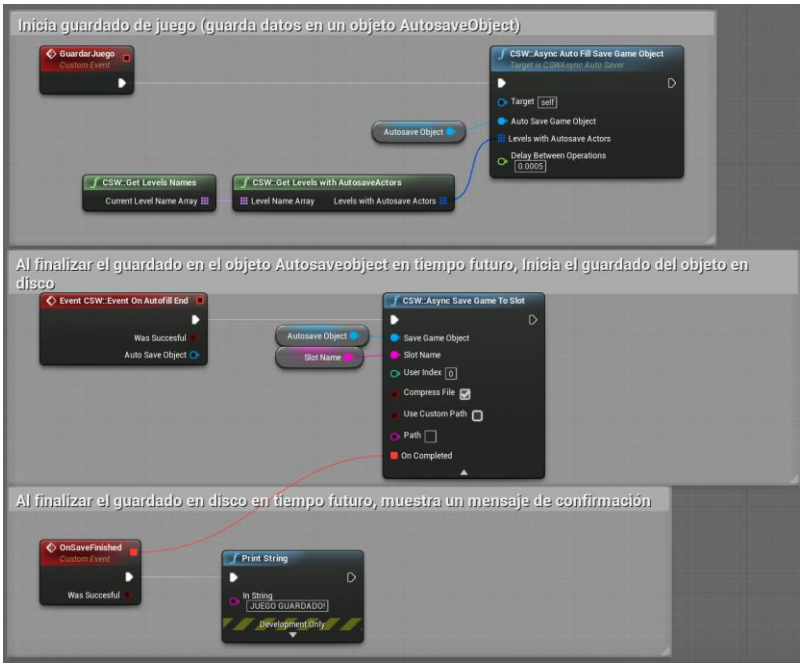


Figura: 48 Implementación de un Sistema de Guardado Asíncrono

Fuente: Elaboración propia.

Por tanto, se obtienen los siguientes resultados, los cuáles muestran una mejora considerable en cuanto a la usabilidad y eficiencia del plugin, siempre y cuando se hagan uso de las funciones asíncronas implementadas.

Tabla 15
Resultado de las Pruebas (Segunda Iteración)

Acción	Tiempo promedio de renderizado	Ejecución Síncrona (Plugin y UE4)	Ejecución con Compresión (Plugin)	Ejecución Asíncrona (Plugin)	Ejecución Asíncrona + Compresión (Plugin)
Guardado de 1000 Actores	~8ms (120fps)	~106ms (9fps)	~155ms (7fps)	~8ms (120fps) durante ~520ms ~10ms	~8ms (120fps) durante ~520ms ~10ms
Cargado de 1000 Actores	~8ms (120fps)	~90ms (11fps)	~120ms (8fps)	(100fps) durante ~570ms	(100fps) durante ~ 570ms

Fuente: Elaboración Propia.

Como se observa en la tabla anterior, se observa que un sistema de guardado y cargado de partidas síncrono reduce considerablemente la tasa de cuadros por segundo de un videojuego. Se observa además que esto puede solucionarse mediante la implementación de un sistema de guardado y cargado de partidas asíncrono, el cual distribuye el proceso de guardado y cargado en un intervalo de tiempo. Asimismo, se observa que el guardado y cargado con compresión de datos reduce aún más la tasa de cuadros por segundo de un videojuego, por lo que su ejecución

se recomienda tan solo en casos donde tener un archivo de guardado menos pesado sea necesario (por ejemplo, si se requiere almacenar dicho archivo en la nube). Finalmente, ya que las herramientas de guardado de UE4 no soportan compresión de datos ni funciones asíncronas; se puede concluir en este punto que, en términos de métricas de eficiencia, el plugin desarrollado cuenta con una mejor calidad de software.

Figura: 49 Resultados de la compresión de archivos de guardado

Fuente: Elaboración propia.

En la anterior figura se observan los resultados de la compresión de los datos de 1000 actores. Ya que las herramientas de UE4 no soportan la compresión de datos, se puede deducir que el plugin desarrollado cuenta con una mayor calidad en términos de funcionalidad (ya que el plugin puede ser aplicable a un mayor número de requerimientos) y eficiencia (ya que los archivos pueden ser transferidos en la red haciendo uso de un menor ancho de banda).

3.3.2.4. Ajustes.

Se considera que el plugin está listo para el despliegue ya que se cumplen en mayor medida con los requerimientos de funcionabilidad, usabilidad y eficiencia.

3.4. Fase de Transición

3.4.1. Documentación de la Guía de Uso del Plugin.

La documentación de guía de uso del plugin se realizó mediante video tutoriales, las cuales están referenciadas en el anexo de este documento.

3.4.2. Despliegue de la Versión Final del Plugin.

Se despliega el plugin siguiendo la guía oficial de publicación de contenido para el Marketplace de Unreal Engine 4. Esta guía describe pautas y reglas que deben de seguirse al

momento de publicar un plugin a la tienda de contenido digital de Unreal Engine 4. En la siguiente figura, se observa el plugin ya publicado en el Marketplace de Unreal Engine 4, luego de haber pasado las evaluaciones correspondientes.

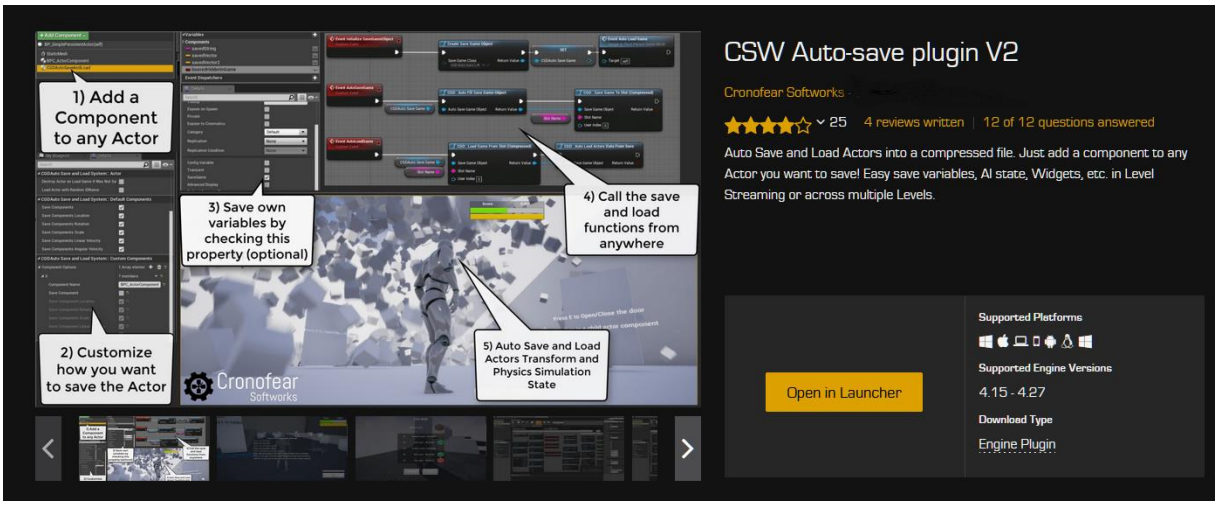


Figura: 50 Plugin publicado en el Marketplace de Unreal Engine 4

Fuente: Elaboración propia.

3.4.3. Plan de Mantenimiento.

Haciendo uso de la plataforma de ventas oficial de Unreal Engine 4, se considera que el plan de mantenimiento del plugin consiste en implementar nuevas funcionalidades del plugin de acuerdo a las peticiones observadas por parte de los potenciales clientes, así como de lanzar actualizaciones del plugin para cada versión nueva de Unreal Engine 4.

La implementación de nuevas funcionalidades del plugin será solamente incluida en la última versión de Unreal Engine 4 en el momento de dicha implementación, esto con el fin de evitar problemas de compatibilidad con usuarios existentes que ya hagan uso de una versión anterior del plugin. En cuanto al lanzamiento de actualizaciones obligatorias, estas se realizarán de acuerdo al calendario de lanzamientos de nuevas versiones de Unreal Engine 4.

Capítulo Cuarto. Resultados

4.1. Resultados de la calidad del plugin

Durante el desarrollo del proyecto se realizaron mediciones y comparaciones de las métricas de eficiencia, ya que estas métricas son fácilmente medibles durante las etapas de desarrollo de software. En esta sección, ya habiendo finalizado el desarrollo del plugin, se procede a describir los resultados para todas las métricas de calidad de software relevantes para esta investigación.

4.1.1. De acuerdo a su funcionalidad.

Durante esta investigación se obtuvieron los siguientes requerimientos a partir de las métricas de funcionalidad:

a. Se debe permitir guardar el juego de forma tradicional mediante un menú.
b. Se debe permitir guardar el juego de forma automática en “checkpoints”.
c. Se debe permitir guardar el juego de forma automática en intervalos de tiempo.

Y se realizaron pruebas de dichas funcionalidades durante las fases de desarrollo, por lo que se determina que tanto las herramientas de guardado incluidas en Unreal Engine 4 como el plugin desarrollado son capaces, en mayor o menor medida, de satisfacer dichos requerimientos.

Sin embargo, es importante recalcar que como el plugin desarrollado implementa funciones asíncronas y compresión de datos, se deduce que dicho plugin puede aplicarse a un mayor número de requerimientos. Por ejemplo, la compresión de datos puede permitir que un juego que requiera distribuir los archivos de guardado en la nube sea viable. O que la implementación de funciones asíncronas permita que un juego que requiera de un guardado de una gran cantidad de datos de forma continua sea posible sin afectar el rendimiento de dicho juego. Por lo que se considera que el plugin desarrollado cuenta con una mayor calidad con respecto a las herramientas incluidas en Unreal Engine 4.

4.1.2. De acuerdo a su usabilidad.

Aunque no es posible determinar de forma objetiva los resultados de esta métrica de forma interna, se considera que es posible deducir algunos resultados de los indicadores descritos por dicha métrica.

En cuanto al indicador de entendimiento de software, es posible deducir que tanto el plugin desarrollado como las herramientas incluidas en Unreal Engine 4 pueden ser fácilmente discernidas por su contexto, nombre y descripción. Por lo que sus posibles aplicaciones pueden ser fácilmente comprendidas, especialmente si consideramos que un desarrollador de videojuegos va a descubrir estas herramientas a partir de la necesidad de guardar los datos en su proyecto. Por tanto, se considera que tanto el plugin desarrollado como las herramientas incluidas en Unreal Engine 4 cuentan con la misma calidad con respecto a este indicador.

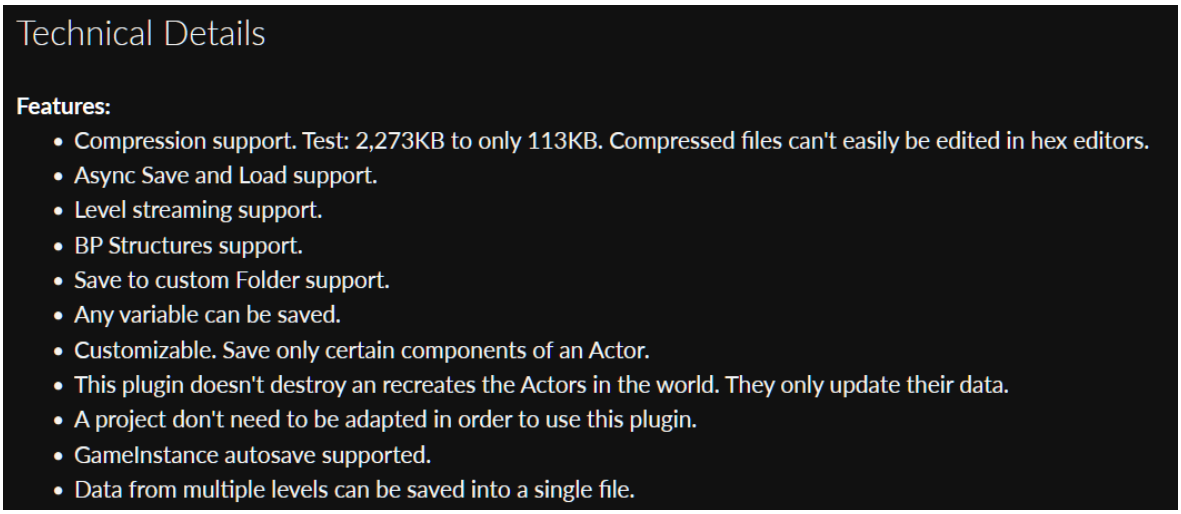


Figura: 51 Ficha técnica del plugin desarrollado.

Un cliente podría discernir fácilmente las posibles aplicaciones del plugin a partir de sus características. Fuente: Propia

En cuanto al indicador de facilidad de aprendizaje, es posible deducir que el plugin desarrollado es más difícil de aprender que las herramientas incluidas en Unreal Engine 4, esto debido a que el plugin cuenta con un mayor número de funciones y clases expuestas. Por lo que

se considera que el plugin desarrollado cuenta con una menor calidad con respecto a las herramientas incluidas en Unreal Engine 4 en términos de dicho indicador.

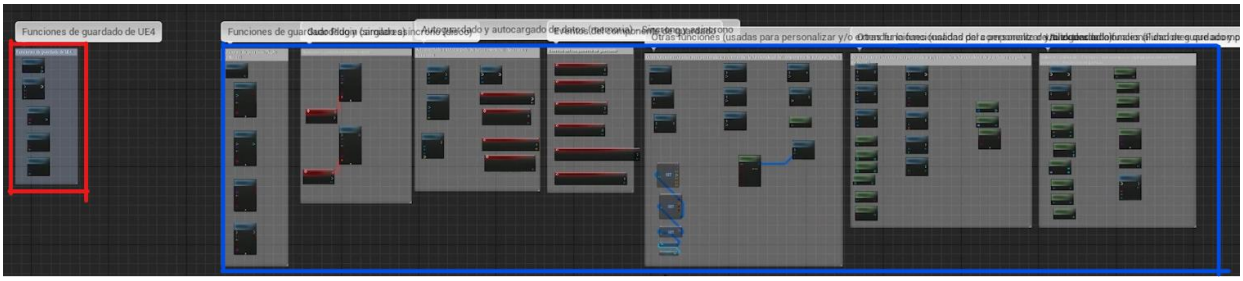


Figura: 52 Comparación de clases y funciones expuestas.

Las clases y funciones expuestas por UE4 (cuadro rojo) y clases expuestas por el plugin desarrollado (cuadro azul). Fuente: Propia

En cuanto al indicador de operatividad; es posible deducir que, si bien el plugin desarrollado es más difícil de aprender, resulta más fácil de usar una vez aprendido. Esto se obtiene a partir de lo descrito durante el análisis de la calidad de las herramientas de Unreal Engine 4, donde se observa que la dificultad de implementación de un sistema de guardado con dichas herramientas aumenta de forma directamente proporcional al número de variables y objetos a guardar en un videojuego. Por otro lado, el plugin desarrollado no cuenta con dicho problema, ya que automatiza gran parte del proceso de guardado y cargado de datos, y otorga la responsabilidad del guardado de variables a su clase correspondiente en vez de tener una clase monolítica a cargo de dicha responsabilidad. Por lo que se considera que el plugin desarrollado cuenta con una mejor calidad con respecto a las herramientas incluidas en Unreal Engine 4 en términos de dicho indicador.

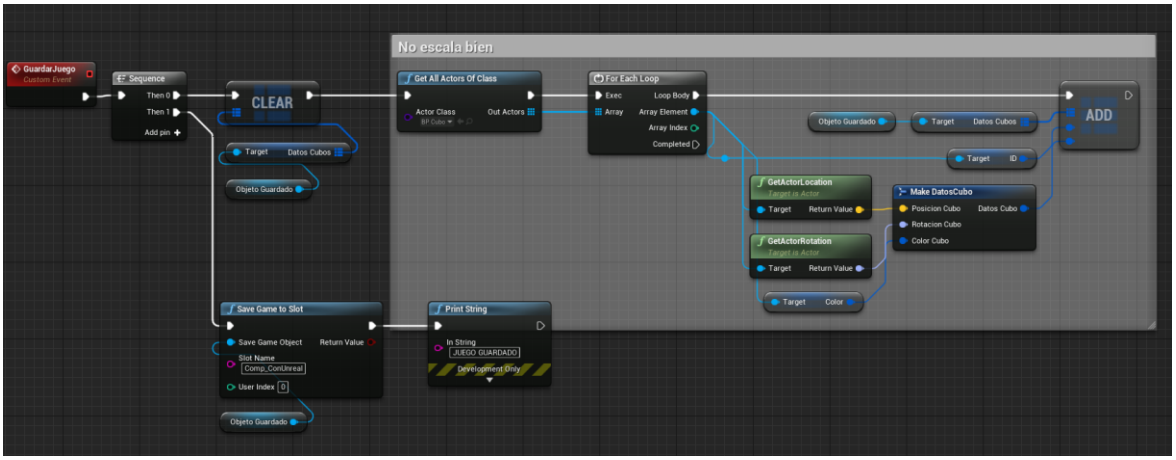


Figura: 53 Implementación de un sistema de guardado simple con UE4.

Las herramientas de guardado incluidas en Unreal Engine 4 no escalan bien cuando hay un gran número de clases y variables a guardar. Los nodos contenidos en el cuadro de comentario deben ser repetidos para cada clase y variables a guardar, lo que incrementa su dificultad de uso. Fuente: Propia

4.1.3. De acuerdo a su eficiencia.

Descrito con mayor detalle en las pruebas de requerimientos. Se considera que el plugin desarrollado cuenta con una mayor calidad con respecto a las herramientas incluidas en Unreal Engine 4. Esto debido a que la implementación de funciones asíncronas reduce el comportamiento de tiempos de proceso durante el guardado y cargado de datos; mientras que la compresión de datos ayuda a reducir la utilización de recursos en términos de uso de espacio en disco, y de ancho de banda en caso de que los archivos vayan a ser transferidos por la red.

4.2. Comprobación de la prospectiva

Habiéndose descrito los resultados obtenidos mediante el uso de las métricas descritas por la norma ISO/IEC 9126 de calidad de software; a continuación, se describe la comprobación de la prospectiva en términos de los requerimientos definidos para esta investigación.

4.2.1. De acuerdo a su funcionalidad.

Tabla 16
Comprobación de Resultados (Métrica de Funcionalidad)

Requerimiento	Observaciones para Unreal Engine 4	Observaciones para el Plugin
Guardar un juego de forma tradicional mediante un menú.	Se cumple el requerimiento. Pero la ausencia de funciones asíncronas y compresión de datos puede hacer inviable su implementación en algunos juegos.	Se cumple el requerimiento.
Guardar un juego de forma automática en "checkpoints".	Se cumple el requerimiento. Pero la ausencia de funciones asíncronas y compresión de datos puede hacer inviable su implementación en algunos juegos.	Se cumple el requerimiento.
Guardar un juego de forma automática en intervalos de tiempo.	Se cumple el requerimiento. Pero la ausencia de funciones asíncronas y compresión de datos puede hacer inviable su implementación en algunos juegos.	Se cumple el requerimiento.

Fuente: Elaboración Propia.

Se observa que tanto las herramientas de guardado de Unreal Engine 4 como las herramientas del plugin desarrollado cumplen los requerimientos de funcionalidad. Sin embargo, se considera que el plugin desarrollado es capaz de abarcar un mayor número de requerimientos en términos prácticos, esto debido a la implementación de funciones asíncronas y compresión de datos, lo cual le otorga una mayor flexibilidad en cuanto a su aplicabilidad en distintos proyectos.

4.2.2. De acuerdo a su usabilidad.

Tabla 17
Comprobación de Resultados (Métrica de Usabilidad)

Requerimiento	Observaciones para Unreal Engine 4	Observaciones para el Plugin
Debe ser fácil de entender	Se cumple el requerimiento.	Se cumple el requerimiento.
Debe ser fácil de aprender a usar.	Se cumple el requerimiento.	Se cumple el requerimiento de forma parcial. Ya que se solventa la dificultad añadida de aprendizaje mediante la creación de video guías de uso del plugin.
Debe ser sencillo de operar.	Se cumple el requerimiento solamente si un juego no requiere del guardado de un gran número de variables.	Se cumple el requerimiento.

Fuente: Elaboración Propia.

Aunque el plugin cuente con un resultado negativo en cuanto a la métrica de facilidad de aprendizaje, se considera que su mayor sencillez de operación, así como su mayor flexibilidad para ser aplicable en entornos de producción tienen un mayor peso debido a que la dificultad de aprendizaje se amortigua con la creación de video guías de uso para el plugin.

4.2.3. De acuerdo a su eficiencia.

Tabla 18
Comprobación de Resultados (Métrica de Eficiencia)

Requerimiento	Observaciones para Unreal Engine 4	Observaciones para el Plugin
Debe tener un tiempo de procesamiento aceptable.	Se cumple el requerimiento solamente si un juego no requiere del guardado de un gran número de variables.	Se cumple el requerimiento.
No se tiene que afectar el rendimiento del juego.	Se cumple el requerimiento solamente si un juego no requiere del guardado de un gran número de variables. O si dicho juego no requiere de un guardado continuo y frecuente de sus datos.	Se cumple el requerimiento.

Fuente: Elaboración Propia.

Aunque el guardado y cargado asíncrono pueda requerir un mayor tiempo de procesamiento que su versión síncrona, se considera que el plugin mejora la calidad de las herramientas de guardado en ambas métricas ya que, aunque el tiempo de proceso de guardado asíncrono pueda ser de hasta 10 veces mayor, el rendimiento del juego no se ve afectado durante todo este intervalo de tiempo.

Por lo tanto, se concluye, en términos generales, que el plugin desarrollado cuenta con una mayor calidad de software con respecto a las herramientas de guardado y cargado de partidas incluidas por defecto en Unreal Engine 4. Asimismo, cabe destacar la respuesta generalmente

positiva por parte de la comunidad de desarrolladores de juegos que adquirieron el plugin, lo cual evidencia una calidad positiva del software desarrollado.

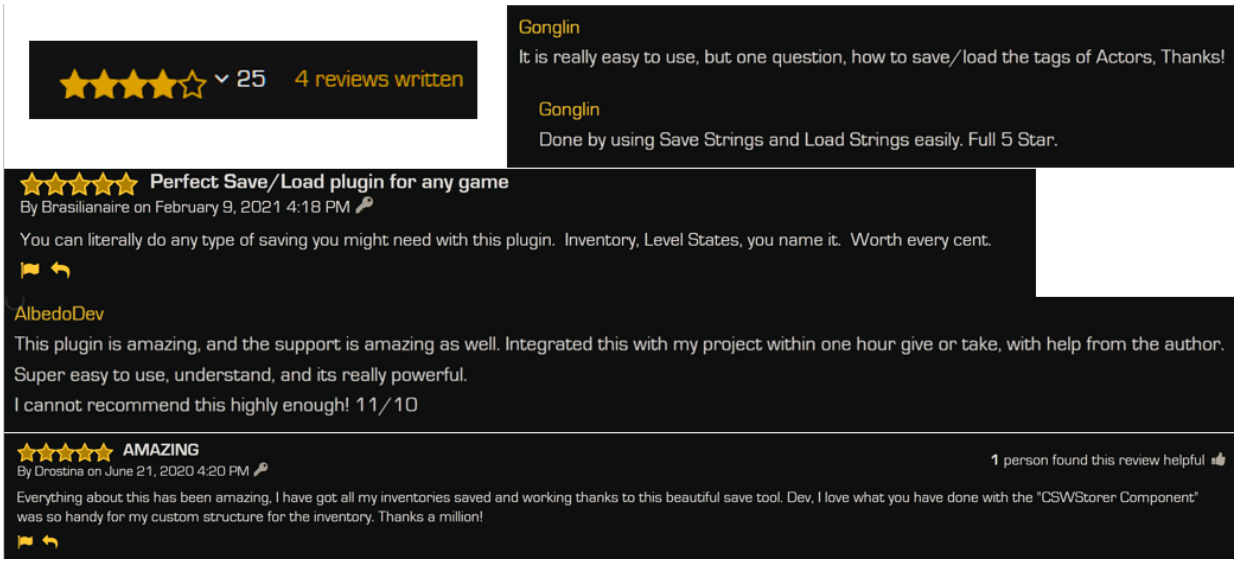


Figura: 54 Críticas recibidas.

Críticas por parte de los clientes que adquirieron el plugin desarrollado en esta investigación. Fuente: Propia

4.3. Cumplimiento de Objetivos

Con respecto al primer objetivo específico: *"Definir los requerimientos de un sistema de guardado y cargado de partidas para videojuegos."*. Se determina que se cumple este objetivo al haberse documentado los requerimientos del proyecto, los cuales se encuentran documentados en el ítem "*3.1.3.1. Definición de los requerimientos de alto nivel"* del Capítulo Tercero de la presente investigación.

En cuanto al segundo objetivo específico: *"Diagnosticar la calidad de las herramientas de guardado y cargado de partidas para videojuego incluidas en Unreal Engine 4."*. Se determina se cumple este objetivo al haberse realizado un diagnóstico de la calidad de las herramientas de guardado de partidas de Unreal Engine 4, el cual se encuentra documentado en el ítem "*3.1.3.2. Análisis de la calidad de las herramientas de Unreal Engine 4.*" del Capítulo Tercero de la presente investigación.

En cuanto al tercer objetivo específico: "*Desarrollar un plugin de guardado y cargado de partidas para videojuegos en Unreal Engine 4.*", Se determina que se cumple este objetivo, el cual se encuentra documentado en el Capítulo Tercero de la presente investigación.

En cuanto al cuarto objetivo específico: "*Implementar pruebas funcionales para medir la calidad del plugin durante su desarrollo.*". Se determina que se cumple este objetivo, el cual se encuentra documentado en las secciones de "*Pruebas de requerimientos*" del ítem "*3.3. Fase de Construcción*" perteneciente al Capítulo Tercero de la presente investigación.

En cuanto al quinto objetivo específico: "*Comparar la calidad del plugin con respecto a las herramientas incluidas en Unreal Engine 4"*. Se determina que se cumple este objetivo, el cual se encuentra documentado en el ítem "*4.1 Resultados de la calidad del plugin*" perteneciente al Capítulo Cuarto de la presente investigación.

Finalmente, mediante la comprobación resultados documentada el Capítulo Cuarto de la presente investigación, se determina que el plugin desarrollado influencia de manera positiva en la calidad de las herramientas de guardado y cargado de partidas de juego de Unreal Engine 4; por lo que se cumple el objetivo general de la investigación.

4.4. Contribuciones

a. Se contribuye a la comunidad de desarrolladores de videojuegos de Unreal Engine 4 al haber construido una herramienta que apoya en el desarrollo de un proyecto de Unreal Engine 4 que pueda requerir de la implementación de un sistema de guardado y cargado de partidas.
b. Se contribuye con la presente investigación al documentar parte del proceso de desarrollo de un plugin en Unreal Engine 4.

Glosario

Actor: Perteneciente al “Gameplay Framework”. Un Actor es un objeto que puede ser instanciado en un nivel de Unreal Engine. Por tanto, este actor depende la existencia del “framework” de Unreal Engine.

Actor Component: Perteneciente al “Gameplay Framework”. Es un componente que puede ser adjuntado a un Actor y que define el comportamiento o el aspecto de este.

Assets: Activos para videojuegos, el cual se refiere a elementos que pueden formar parte de un videojuego. Estos activos incluyen plugins en C++, modelos 3D, texturas, animaciones, sonidos, etc.

AUP: Proceso ágil unificado de desarrollo de software. Metodología ágil de desarrollo de software que define 4 fases: Incepción, Elaboración, Construcción y Transición.

Azure DevOps: El servicio en la nube de Microsoft para la gestión de proyectos de software. Incluye un sistema de control de versiones (Git), tablero Kanban, herramientas para generación de documentos (wiki), entre otros.

Blueprints: El lenguaje de programación visual que incluye Unreal Engine 4, es considerado un lenguaje flexible y amigable tanto para diseñadores como para programadores.

Checkpoint: Punto de guardado. Área o zona dentro de un videojuego donde es posible guardar el estado de partida de un videojuego.

Epic Games: Compañía de videojuegos ubicada en Carolina del Norte, USA y fundada en 1991. Es la compañía que creó Unreal Engine 4 y que se encarga de mantener el motor en la actualidad.

Gameplay Framework: Se refiere a las herramientas y clases proporcionadas por Unreal Engine con el fin de ofrecer un marco de desarrollo a seguir a la hora de crear un proyecto en el motor.

Git: Es un sistema de control de versiones distribuido, diseñado para la colaboración de programadores en un mismo proyecto de software. Además, permite mantener el ciclo de vida de desarrollo del software de una forma más óptima. Al ser distribuido, cada usuario cuenta con una versión distinta del software, el cual suele ser fusionado en una versión principal (en un servidor), al cual todos los usuarios tienen acceso.

Jenkins: Es un sistema que permite gestionar la automatización de tareas repetitivas. Hace uso de scripts (En Bach, Python, JavaScript, etc.) y un gestor de tiempos o eventos para controlar cuando y como estas tareas repetitivas van a ser realizadas. Por ejemplo, es posible crear una actividad que se realice cada domingo y que consista en descargar la última versión del software desde un servidor en Git, compilar el software, realizar ciertas pruebas, empaquetar el software y subirlo a un repositorio de ejecutables. Todo de forma automática.

Kanban: Es considerado una metodología ágil y flexible para la gestión de proyectos (independiente si este sea un proyecto de software). En su forma estándar, consiste en un tablero visual con columnas “Pendiente”, “En Proceso” y “Finalizado” el cual contiene tarjetas (con tareas a realizar) en cada columna de acuerdo al estado de cada tarea. Asimismo, se define algunas reglas de uso, tal como limitar el número de tarjetas en cada columna, de acuerdo a los requerimientos del proyecto.

Level: Perteneciente al “Gameplay Framework”. Mapa o nivel de juego. Es una instancia que define un área de juego. Esta instancia es capaz de contener diversos Actor.

Marketplace (Unreal Engine 4): Es una tienda digital donde desarrolladores pueden publicar assets para el desarrollo de videojuegos en Unreal Engine 4, así como adquirir estos "assets".

Modelos 3D: Se refiere a un modelo tridimensional que representa (de forma visual) a un objeto en el mundo real.

Motor de juegos: Software orientado a la creación de videojuegos que incluye tecnologías que facilitan el desarrollo de los mismos, tales como motor de audio, de físicas, soporte para un lenguaje de scripting, etc.

Norma ISO/IEC 9126: Esta norma define métricas que pueden ser utilizadas para medir la calidad de un software de forma objetiva.

Perforce: Es un sistema de control de versiones centralizado, diseñado para la colaboración de programadores en un mismo proyecto de software. Además, permite mantener el ciclo de vida de desarrollo del software de una forma más óptima. Al ser centralizado, todos los usuarios cuentan con la misma versión del software, el cual está acorde a la versión principal del servidor.

Plugin (Unreal Engine 4): Es un software desarrollado en C++ que se acopla al motor de videojuegos con el fin de extender y/o reemplazar funcionalidades del mismo.

RGBA: Se refiere a los 4 canales de información el cual usan las imágenes digitales. "RGB" hace referencia a los colores rojo, verde y azul respectivamente. Mientras que "A" hace referencia al valor "alfa", el cual define la transparencia de una imagen.

Serialización: En este contexto, se refiere al proceso de guardar el estado de un objeto que se encuentra en memoria. En el contexto de la investigación, el estado de juego es guardado en un archivo en formato ".sav" o ".csav" en disco.

Sistema de guardado y cargado de partidas de videojuegos: Mecanismo implementado en un videojuego el cual permite tanto guardar como restaurar los datos del jugador y el estado el

que se encuentra de un videojuego. Por ejemplo: el número de vidas del jugador, niveles completados, entre otros.

Texturas: Imágenes con información de color (RGBA) los cuales son usados para “pintar” o dar diversos acabados (metálicos, plásticos, etc.) a los modelos 3D. También son usados para la creación de efectos especiales como fuego, humo, entre otros.

Unreal Engine 4: Es un motor de videojuegos desarrollado por Epic Games, soporta la integración de plugins para extender o reemplazar sus funcionalidades.

Videojuegos: Productos de software diseñados para entretener a usuarios, permiten la interacción de uno o más jugadores de forma que estos interactúen con mundos virtuales mediante diversos dispositivos de entrada y/o salida.

Conclusiones

a. En esta tesis se determinó la influencia del plugin en la calidad de las herramientas de guardado y cargado de partidas para videojuegos en Unreal Engine 4, la cual resulta en una influencia positiva; ya que el plugin desarrollado mejora la calidad de dichas herramientas con respecto a las incluidas en Unreal Engine 4.

b. En esta tesis se definieron los requerimientos de un sistema de guardado y cargado de partidas para videojuegos, los cuales fueron obtenidos considerando los requerimientos de un videojuego promedio y las métricas de calidad de software descritas por la norma ISO/IEC 9126.

c. En esta tesis se diagnosticó la calidad de las herramientas de guardado y cargado de partidas para videojuegos incluidas en Unreal Engine 4, calidad que se consideró mejorable debido a que falla en cumplir todos los requerimientos de un sistema de guardado y cargado de partidas para videojuegos definidos en esta investigación.

d. En esta tesis se desarrolló un plugin de guardado y cargado de partidas para videojuegos en Unreal Engine 4, desarrollo que fue apoyado con la metodología AUP (Proceso ágil unificado de desarrollo) y que finalizó con el despliegue del plugin en un entorno de producción.

e. En esta tesis se implementaron pruebas funcionales de forma que se midió la calidad del plugin durante su desarrollo, las cuales ayudaron a garantizar que el plugin creado en esta investigación cuente con una calidad de software deseada.

f. En esta tesis se comparó la calidad del plugin con respecto a las herramientas incluidas en Unreal Engine 4, en la cual se determina que el plugin desarrollado en esta investigación cuenta con una mejor calidad de software.

Recomendaciones

a. Se recomienda hacer uso de las métricas de calidad de software descritas por la norma ISO/IEC 9126 para ayudar a definir los requerimientos de un proyecto, ya que al definirlos de esta forma permitiremos que el desarrollo de dicho software sea guiado por métricas que garanticen una calidad óptima.
b. Se recomienda diagnosticar la calidad de uno o más softwares similares al desarrollado en una investigación, esto con el fin de tener un punto de apoyo o guía para asegurar que el software creado cuente con una calidad óptima con respecto a otros softwares similares y sus requerimientos.
c. Se recomienda el uso de la metodología AUP (Proceso ágil unificado de desarrollo), ya que es una metodología flexible y adaptable a requerimientos específicos; como es el caso del desarrollo de un plugin para Unreal Engine 4.
d. Se recomienda la implementación de pruebas durante el desarrollo de un software, ya que dichas pruebas ayudan a garantizar que el software creado cuente con una calidad óptima antes de ser desplegado en un entorno de producción.
e. Se recomienda la implementación de funciones asíncronas para mejorar la usabilidad y experiencia de usuario de un software, ya que estas funciones permiten distribuir la carga de una función pesada en un periodo de tiempo más largo; lo cual ayuda a que el software se ejecute de una forma más fluida y sin interrupciones.
f. Se recomienda el uso de diccionarios para definir la estructura de datos necesaria para la implementación de un sistema de guardado para videojuegos, esto porque los arreglos no son suficientemente rápidos para manipular o leer dichos datos en memoria.

g. Se recomienda el uso del patrón de diseño “observador” para la implementación de un sistema de guardado para videojuegos, ya que este patrón de diseño permite que los objetos de juego que requieran el guardado y cargado de sus datos puedan subscribirse a un objeto global y dicho objeto global puede gestionar el estado de todos estos objetos en tiempo de ejecución.

Referencias

Acerca de Start up Perú. (2019). Obtenido de Start up Perú: https://www.start-up.pe/faq/

Adams, E. (2014). *Fundamentals of Game Design.*

ambysoft. (2006). *The Agile Unified Process (AUP)*. Obtenido de ambysoft: http://www.ambysoft.com/unifiedprocess/agileUP.html

Anderson, D. J. (2010). *Kanban: Succesful Evolutionary Change of Your Technology Business.* Amazon.

Capcom. (2010). *RE4HD*. Obtenido de Resident Evil 4 HD: https://www.re4hd.com/

Epic Games. (2017). *About Components in Unreal Engine*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors/Components/

Epic Games. (2017). *About plugins in Unreal Engine 4*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/latest/INT/Programming/Plugins/

Epic Games. (2017). *Blueprints Visual Scripting*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/

Epic Games. (2017). *Introduction to C++ programming in UE4*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/latest/INT/Programming/Introduction/

Epic Games. (2017). *What is Unreal Engine 4*. Obtenido de Unreal Engine Documentation.

Epic Games. (2017e). *Programming*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/en-us/Programming

Epic Games. (2018). *Saving Your Game with Blueprints*. Obtenido de Unreal Engine: https://docs.unrealengine.com/en-us/Gameplay/SaveGame/Blueprints

Epic Games. (2018). *Saving your game*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/en-us/Gameplay/SaveGame

Epic Games. (2018). *Unreal Engine 4 Marketplace FAQ*. Obtenido de Unreal Engine Documentation: https://www.unrealengine.com/en-US/marketplace-faq

Epic Games. (2018). *Unreal Engine Marketplace*. Obtenido de Unreal Engine Marketplace: https://www.unrealengine.com/marketplace/store

Epic Games. (2019). *Getting Started*. Obtenido de Unreal Engine Documentation: https://docs.unrealengine.com/en-US/GettingStarted/index.html

Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332.

Howard, L., Farnaz, G., Pradeep Kumar, J., & Ozcan, P. (2015). A statistical analysis of the effects of Scrum and Kanban on software development projects. 15.

ISO.ORG. (2001). *ISO.ORG*. Obtenido de Software engineering — Product quality — Part 1: Quality model: https://www.iso.org/standard/22749.html

ISO/IEC. (2001). ISO 9126 Software engineering - Product quality.

Lee, W.-S., Chun, J. A., & Kang, K. (2014). Development and application of GIS-based PRISM integration through a plugin approach. *Journal of Hydrology*, 58-67.

Looman, T. (2018). *Unreal Engine Gameplay Framework Primer for C++*. Obtenido de tomlooman: https://www.tomlooman.com/ue4-gameplay-framework/

Microsoft. (2018). *Azure Devops*. Obtenido de https://docs.microsoft.com/en-us/azure/devops/project/search/faq-search?view=azure-devops-2020

Moran, C. (2010). *Playing with Game Time*. Obtenido de The Firbreculture Journal: http://sixteen.fibreculturejournal.org/playing-with-game-time-auto-saves-and-undoing-despite-the-magic-circle/

New Media Society. (2008). Console video games and global corporations: Creating a hybrid culture.

Peterson, D. (2015). *What is Kanban?* Amazon.

Pursell, C. (2015). *The Interaction of Technology and Play.* John Hopkins University Press.

Rabin, S. (2006). *Introduction to Game Development.* Massachusetts: Charles River Media.

Sirlin, D. (2008). Saving the Day: Save Systems in Game.

Somerville, I. (2015). *Software Engineering.*

Ward, J. (2008). What is a Game Engine?

Welcome to Pong-Story. (2013). Obtenido de Pong-Story: http://www.pong-story.com/intro.htm