PAQUETE DE MECÁNICAS PARA VIDEOJUEGOS TIPO SURVIVAL HORROR

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Jorge Miguel Trujillo Liz

Tutores: Ing. Jorge Evelio Valdivia Hernández

Ing. José Emilio Badia Valdés

Ing. Luis Ángel Llull Céspedes

La Habana, 2022

FRASE

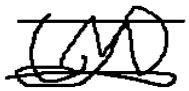
Internet es el desarrollo individual más importante en la historia de la comunicación humana desde la invención de la llamada en espera.

Dave Barry

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis que tiene por título: y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los _8__ días del mes de __julio__ del año _2022_.

Jorge Miguel Trujillo Liz

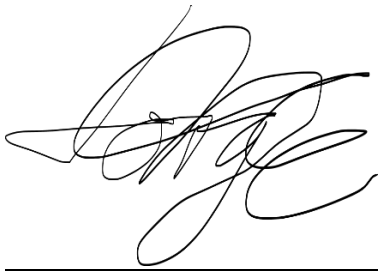


Firma del autor

Ing. José Emilio Badia Valdés

Firma del Tutor

Ing. Jorge Evelio Valdivia Hernández



Firma del Tutor

AGRADECIMIENTOS

A Dios por guiarme todo el camino a pesar de las dificultades.

A toda mi familia, mis abuelos, mis padres, por siempre creer en mí y apoyarme todo este tiempo.

A mis amigos de la UCI, Kevin, Frank, Leduan, Luis Miguel, Delvis, Mari, Tornillo, Dionis y Hector por estar ahí y ayudarme.

A Laura Loinaz Marrero y Wendy por darme palabras de aliento y escucharme cuando lo necesitaba.

A Samira por servirnos de guía en los estudios y los trabajos.

A todos mis amigos por los consejos que me dieron, a Diana y especialmente a mis mejores amigos Gretel e Isaac que siempre han estado ahí cuidando mis espaldas.

A mis tutores, a la profesora Enelis y a todos aquellos que ayudaron a mi formación profesional.

DEDICATORIA

A mis padres y a Dios que han hecho de mi la persona que soy ahora.

RESUMEN

Los videojuegos son una rama del entretenimiento enfocado al público de todas las edades que emplean como principal herramienta las mecánicas interactivas y los desafíos. El tiempo que toma su creación, pueden ser desde unos pocos meses a varios años; a pesar de los largos períodos de tiempo de desarrollo, estos no son aún mayores gracias a los motores de videojuegos que ofrecen a los desarrolladores un conjunto de componentes y funcionalidades reutilizables. La finalidad de la presente investigación se centra en desarrollar un paquete funcional editable con temática survival horror, que será realizado en base al análisis de las principales características y jugabilidad de tres ejemplos del género. Las herramientas de desarrollo son Unity como motor de videojuego en el que se implementará finalmente todo el proyecto para su respectivo funcionamiento, en Visual Studio Code se programará el código necesario y como lenguaje de programación utilizado c-sharp; todo esto brindará a los desarrolladores una base reutilizable en varios proyectos y acortará los tiempos de desarrollo. La propuesta de solución será validada con las pruebas de aceptación y unitarias en aras de un correcto funcionamiento del código. De igual forma se implementó un demo para validar la reusabilidad y extensibilidad de la solución, además de que serán aplicadas pruebas de rendimiento, demostrándose un resultado satisfactorio.

Palabras claves: Survival horror, mecánicas, paquete, Unity.

SUMMARY

Video games are a branch of entertainment focused on audiences of all ages that use interactive mechanics and challenges as their main tool. The time it takes to create them, can be from a few months to several years, despite the long periods of development time, these are not even longer thanks to the video game engines that offer developers sets of components and reusable functionality. The purpose of the present research is focused on developing an editable functional package with survival horror theme, which will be realized based on the analysis of the main features and gameplay of three examples of the genre. The development tools are Unity as video game engine in which the whole project will be finally implemented for its respective operation, in Visual Studio Code the necessary code will be programmed and as programming language used c-sharp, all this will provide developers with a reusable base in several projects and will shorten development times. The proposed solution will be validated with acceptance and unit tests in order to ensure the correct functioning of the code. A demo was also implemented to validate the reusability and extensibility of the solution, and performance tests were applied, showing a satisfactory result.

Keywords: Survival horror, mechanics, package, Unity.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

El juego nació como una herramienta de entretenimiento, usada principalmente por niños para desarrollar sus habilidades, es su forma de poner a trabajar su mente llenándola de fantasía, para vivir una aventura inspirada en sus propios deseos. De igual forma, los adultos hacen uso de esta herramienta en los videojuegos ya sea con fines competitivos o el mismo disfrute personal de realizar dicha actividad. Existen muchos tipos de juegos, por ejemplo, los que necesitan esfuerzo físico permitiendo un desarrollo óptimo de las capacidades de los que participan y los que no necesitan tanto movimiento, pero exigen concentración. De igual manera, los géneros existentes son muy variados, existiendo numerosos subgéneros dentro de los mismo, como por ejemplo el *survival horror*.

El videojuego desde sus albores ha tenido un gran desarrollo fomentado por el creciente nivel tecnológico alcanzado, en los años 40 y 50 la curiosidad impulsaba el desarrollo, pero no fue hasta los años 60 que el videojuego nacería por la voluntad de sus creadores, con lo cual, en 1961 nació "*Space War*" [31]. Estos primeros pasos son los que permitieron que hoy en día existan gráficos tan avanzados, controles especializados, realidad virtual, reconocimiento de movimiento mediante cámaras, etc. Existe una diversa gama de géneros y mecánicas las cuales permiten la interacción del usuario con el entorno y otros personajes existentes.

Los videojuegos se convirtieron en una fuente muy buscada de diversión, un pasatiempo entretenido y desafiante, y gracias a esto es una fuente importante de ingresos. Inicialmente en Cuba no se contaba con un medioambiente óptimo para desarrollar esta industria; no obstante, con el avance del tiempo se logró crear una infraestructura que permitiera la implementación de servicios orientados al entretenimiento. Actualmente en el país se encuentra naciendo la industria del videojuego, ya hace 10 años que se desarrollan, pero ahora es que se está impulsando. La plataforma Cosmox busca integrar los videojuegos a la sociedad cubana y aunque esta no cuenta con un catálogo tan amplio, se trabaja en su aumento.

Por tanto, en la Universidad de las Ciencias Informáticas, el Centro de Tecnologías Interactivas de la Facultad 4 que tiene como una de sus metas el desarrollo de videojuegos, usando principalmente el motor de videojuegos Unity, se acomete la labor de aumentar los títulos para Cosmox. Una plataforma en la cual se pretende incluir todos los videojuegos posibles; pero cuenta con una gama muy ínfima en variedad y género, conteniendo solo los

desarrollados en el Centro. Teniendo en consideración la falta de diversidad en los géneros de la plataforma, la cantidad tan pequeña de los mismos y el tiempo que toma su creación, que pueden tomar desde unos pocos meses a varios años. Se optó por la creación de uno de los géneros que tienen gran aceptación entre el público mundial en aras de enriquecer y optimizar los tiempos de desarrollo.

De esta manera se tiene como **problema de investigación:** ¿Cómo contribuir al desarrollo de videojuegos basado en la reutilización de componentes en Unity? Y se define como **objeto de estudio** el desarrollo de mecánicas para videojuegos.

Para la solución a la problemática se define como **objetivo de esta investigación:** desarrollar un paquete de mecánicas reutilizables para un videojuego tipo *Survival Horror*.

Por lo cual se establece como **campo de acción:** las mecánicas de videojuegos tipo Survival Horror.

Para dar cumplimiento al objetivo general del trabajo se definieron las **tareas de investigación** siguientes:

1- Elaboración del marco teórico de la investigación a partir del estado del arte de las mecánicas de videojuegos tipo Survival Horror.
2- Caracterización de paquetes para Unity destinados al desarrollo de videojuegos tipo Survival Horror, con el objetivo de determinar los elementos principales que componen a los mismos y escoger las funcionalidades que puedan incluirse en la solución propuesta.
3- Identificación de los requisitos funcionales y no funcionales de la solución.
4- Realización de los artefactos ingenieriles necesarios para los flujos de trabajo de Análisis y Diseño.
5- Implementación de una solución.
6- Validación, a través de pruebas, del cumplimiento de los requerimientos definidos.

Par dar solución al objetivo planteado se emplearon los siguientes **métodos científicos:**

Histórico – Lógico: Se utilizó para la fundamentación y sistematización de los aspectos teóricos en el desarrollo de la investigación referente a la trayectoria y acontecimientos actuales del desarrollo de mecánicas de videojuegos en consonancia con el trabajo.

Analítico – Sintético: La utilización de este método permitió el análisis de videojuegos tipo *Survival Horror* existentes y así extraer los elementos más importantes de los mismos relacionados con el presente trabajo.

Métodos empíricos:

Observación: se emplea como método referencial al observar distintos videojuegos tipo Survival Horror que sirvieron como objeto de análisis y comparación para establecer las características y elementos fundamentales que debía cumplir la propuesta del autor.

El presente trabajo cuenta con la siguiente estructura de capítulos relacionados con la actual investigación:

- **Capítulo 1. Fundamentos teóricos.**

En este capítulo se realiza un estudio sobre las mecánicas, algunas definiciones importantes para la presente investigación y el análisis de tres videojuegos como base para la obtención de los mecanismos importantes en la solución. Además, se explican la metodología y las herramientas para su desarrollo.

- **Capítulo 2. Propuesta solución.**

En este capítulo se realiza la definición de los requisitos funcionales, mediante las historias de usuarios definidas por la metodología utilizada, y los no funcionales. También se detallan la propuesta solución, el plan de iteraciones, la arquitectura a usar, los patrones de diseño, el diagrama de clases y los estándares a usar en el desarrollo.

- **Capítulo 3. Implementación y prueba.**

En este capítulo se implementa el sistema mediante las tareas en iteraciones. Fueron aplicadas las pruebas de unitarias y de aceptación para obtener un correcto funcionamiento del paquete y el código. Por último, se sometió a pruebas de rendimientos y demo desarrollado para probar la reusabilidad del paquete.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En este capítulo se presenta el estado del arte donde se definen conceptos relacionados a la problemática, tales como videojuegos, sus géneros “específicamente el *survival horror”*, junto a sus mecánicas y proceso de desarrollo. Se estudian tres videojuegos survival horror para distinguir los comportamientos fundamentales en sus mecánicas. Además, se describen tecnologías, metodologías y herramientas a utilizar en el progreso de la solución.

1.1 Videojuegos

Antes de llegar al concepto de videojuego se debe conocer el de juego. El juego es una acción o actividad voluntaria, cumplida dentro de ciertos límites de tiempo y lugar de acuerdo con una regla libremente consentida pero absolutamente imperiosa, provista de un fin en sí misma, acompañada por un sentimiento de tensión y de júbilo [1].

El concepto anterior engloba una buena idea general, aunque tenemos a Lalande, quien opinaba que juego es toda prodigación de actividad física o mental que no tiene un fin inmediatamente útil, ni tampoco un objetivo definido; y cuya razón de ser para la conciencia de quien la emprende es el puro placer que encuentra en la misma [2]. Aunque ambos autores difieren, sí que presentan un punto crucial en común, la diversión, o dicho de una forma más exacta, entretenimiento, con esto se dará paso a varias definiciones de videojuego.

Un videojuego es una aplicación interactiva orientada al entretenimiento que, a través de ciertos mandos o controles, permite simular experiencias en la pantalla de un televisor, una computadora u otro dispositivo electrónico. Los videojuegos se diferencian de otras formas de entretenimiento, como son las películas, en que deben ser interactivos; en otras palabras, los usuarios deben involucrarse activamente con el contenido. Para ello, es necesario utilizar un mando (también conocido como *gamepad* o *joystick*), mediante el cual se envían órdenes al dispositivo principal (un ordenador o una consola especializada) y estas se ven reflejadas en una pantalla con el movimiento y las acciones de los personajes [3].

Un video juego es un entretenimiento o actividad lúdica que se ejecuta por medio de un programa de software que es procesado por una máquina que posee dispositivos de entrada y de salida [4]. Los videojuegos recrean entornos y situaciones virtuales en los que el video jugador puede controlar a uno o varios personajes (o cualquier otro elemento de dicho entorno), para conseguir uno o varios objetivos por medio de reglas determinadas [5].

Según todo lo anteriormente planteado, se puede definir para el presente trabajo que un videojuego es: aquel programa informático que funciona sobre diversas plataformas o dispositivos, tales como computadoras, videoconsolas, o cualquier otro, que tenga periféricos. Además de permitir la interacción del jugador con el mundo o instancias de este previamente creados ficticiamente o basados en la realidad con un conjunto de reglas, limitaciones u obstáculos y la presencia o no de objetivos específicos, brindando diversas emociones al jugador.

1.2 Géneros de videojuegos

Existe una gran cantidad de géneros de videojuegos actualmente, conocerlos y saber de ellos ayudará a saber cuál es el juego que se adapta más al estilo de cada persona. Con el pasar del tiempo, algunos géneros que anteriormente eran los más populares, ahora se sitúan entre los menos usados, a su vez, con la evolución de la industria se ha podido presenciar todo un avance en géneros que antes no eran muy tenidos en cuenta [6].

Existe una gran diversidad, entre los cuales se encuentran: acción, aventura, arcade, deportivo, estrategia, simulación, musicales, entre otros. La gama es bastante amplia, ahora, teniendo en cuenta las características de los mismos, se descubrió que el subgénero survival horror, deriva de los géneros aventura y acción.

1.2.1 Género Survival

Existe mucha confusión sobre la definición de juegos de supervivencia, que a menudo se relaciona con la subcategoría más famosa de los juegos de terror de supervivencia. El término “juego de supervivencia” se utiliza para referirse a una gama bastante amplia de videojuegos para un jugador desarrollados desde la década de 1990. De hecho, los juegos de supervivencia también son un subgénero de otra categoría más amplia, a saber, los videojuegos de acción. Y la única diferencia entre los juegos de supervivencia y los de acción radica principalmente en la configuración, que es un entorno hostil de mundo abierto en el que los jugadores se ven obligados a iniciar aventuras con un equipamiento muy limitado. El objetivo principal del jugador es recolectar tantos recursos y elementos útiles como sea posible para sobrevivir y llegar al final con vida [7]. Ejemplos de este tipo de videojuegos serían, *minecraft, Conan Exiles, No Man´s Sky, Subnautica*, entre otros.

1.2.2 Subgénero Survival Horror

Survival horror (en español "terror de supervivencia") es un subgénero perteneciente, por lo general, al género de videojuegos conocido como aventuras de acción; caracterizado por combinar elementos de aventura gráfica con elementos de suspenso y acción, dentro de una atmósfera o ambientación de terror psicológico donde lo que prima es la supervivencia. El término se generalizó con *Resident Evil* que se convirtió en referente para otros *Survivals*. Generalmente, los *survival horror* utilizan distintos elementos para crear una atmósfera de terror psicológico en el jugador, intentando infundirle horror, el protagonista cuenta con poca libertad de movimientos (caminar, correr, apuntar y atacar) y es común contar con pocos recursos, es decir, pocas municiones, pocos elementos curativos, de ahí el sustantivo survival (supervivencia). Otra característica bastante común es la presencia de puzles o acertijos para resolver a lo largo de la historia, que generalmente requieren una capacidad de investigación y observación detallada por parte del jugador. También suelen aparecer enemigos de repente asustando [8].

1.3 Mecánicas de videojuegos

En los juegos de mesa y los videojuegos, la mecánica del juego son las reglas que gobiernan y guían las acciones del jugador, así como la respuesta del juego a ellas. La mecánica de un juego, por lo tanto, especifica efectivamente cómo funcionará el juego para las personas que lo juegan. No existen definiciones aceptadas concretas de la mecánica del juego. Algunas definiciones en competencia incluyen la opinión de que las mecánicas del juego son "sistemas de interacciones entre el jugador y el juego", que "son más de lo que el jugador puede reconocer, son solo aquellas cosas que impactan la experiencia de juego". [9].

El término mecánica ha llegado a indicar muchos tipos diferentes de relaciones subyacentes entre entidades en los juegos. A continuación, se presentan cinco tipos diferentes de mecánicas que se pueden encontrar en un juego [10]:

Física: La mecánica de los juegos a veces define la física -la ciencia del movimiento y la fuerza- en el mundo del juego (que puede ser diferente de la física del mundo real). En los juegos, los personajes suelen moverse de un lugar a otro, saltar o conducir vehículos. Calcular la posición de un elemento del juego, la dirección en la que se mueve y si se cruza o colisiona con otros elementos constituye la mayor parte de los cálculos en muchos juegos.

Economía interna: La mecánica de las transacciones con elementos del juego que se recogen, consume y comercian constituye la economía interna de un juego. La economía interna de un juego suele abarcar elementos fácilmente identificables como recursos: dinero, energía, munición, etc. Sin embargo, la economía de un juego no se limita a elementos concretos y tangibles; también puede incluir abstracciones como la salud, la popularidad y el poder mágico.

Mecanismos de progresión: En muchos juegos, el diseño de los niveles dicta cómo puede moverse el jugador por el mundo del videojuego. Tradicionalmente, el avatar del jugador tiene que llegar a un lugar determinado para rescatar a alguien o derrotar al malhechor principal y completar el nivel. En este tipo de juegos, el progreso del jugador está estrechamente controlado por una serie de mecanismos que bloquean o desbloquean el acceso a determinadas zonas. Palancas, interruptores y espadas mágicas que permiten destruir ciertas puertas son ejemplos típicos de estos mecanismos de progresión.

Maniobra táctica: Los juegos pueden tener una mecánica que se ocupa de la colocación de las unidades del juego en un mapa para obtener ventajas ofensivas o defensivas. Las maniobras tácticas son fundamentales en la mayoría de los juegos de estrategia, pero también aparecen en algunos juegos de rol y de simulación. La mecánica que gobierna las maniobras tácticas suele especificar qué ventajas estratégicas puede obtener cada tipo de unidad al estar en cada ubicación posible. Muchos juegos restringen la ubicación de las unidades a fichas discretas, como es el caso de un juego de mesa clásico como el ajedrez. Incluso los juegos de estrategia modernos que se juegan en el ordenador suelen implementar fichas, aunque hacen un buen trabajo al ocultarlas tras una capa visual detallada. Las maniobras tácticas aparecen en muchos juegos de mesa como el ajedrez y el Go, pero también en juegos de estrategia para ordenador como StarCraft o Command & Conquer: Red Alert.

Interacción social: Hasta hace poco, la mayoría de los videojuegos no regulaban la interacción social entre los jugadores, aparte de prohibir la colusión o exigir que mantuvieran ciertos conocimientos en secreto. Ahora, sin embargo, muchos juegos online incluyen mecánicas que recompensan el hacer regalos, invitar a nuevos amigos a unirse y participar en otras interacciones sociales. Además, los juegos de rol pueden tener reglas que regulen la actuación de un personaje, y un juego de estrategia puede incluir reglas que regulen la formación y ruptura de alianzas entre los jugadores. Los juegos de mesa y los

juegos populares jugados por los niños tienen una historia más larga de mecanismos de juego que guían las interacciones entre los jugadores.

Por lo tanto, de forma general, las mecánicas describen lo que el jugador puede hacer, cómo lo hace y las reglas que gobiernan estas acciones. A continuación, se efectúa el análisis de los videojuegos: *Alone in the Dark* (2008), *Resident Evil* (1996) y *Silent Hill* (1999), uno de los precursores del género survival horror y de los más aclamados. La finalidad es detectar los patrones principales y reunirlos en mecánicas bien definidas, tomando en consideración los elementos conceptuales de las categorías mencionadas de esta sección. Todas las mecánicas detectadas serán tomadas como las mecánicas esenciales de los videojuegos survival horror que no puede carecer la solución.

1.4 Mecánicas elementales de videojuegos survival horror

La industria de los videojuegos se encuentra en un constante auge, en el que se puede encontrar una gran cantidad de videojuegos del subgénero *survival horror.* Muchos de estos cuentan con estéticas y mecánicas diferentes por lo cual difieren bastante de los primeros en surgir. Cuando se piensa en este subgénero se pueden encontrar numerosos juegos, como, por ejemplo: *Silent Hill, Retro-Spective: Haunted House, Dead Space, Resident Evil, Outlast, Silent Hill, F.E.A.R, Alone in the Dark, The Forest, Soma, Scorn. Alone in the dark* 1992 fue el precursor del survival horror, pero se hablará de la versión del juego creada en 2008 para comparar las características más actuales con algunas más antiguas y concluir cuales se han mantenido o evolucionado.

Alone in the Dark tuvo varias fechas de lanzamiento para varias plataformas como PC, Xbox 360, Wii, Playstation 3 y Playstation 2, aunque todas estas en el año 2008. El objetivo de este juego es controlar al personaje principal y cumplir los objetivos para completar la trama. A pesar de entrar dentro del survival horror, este videojuego presenta características de otros por lo cual fue un tanto novedoso en la industria, combinando elementos de sandbox, aventura gráfica, conducción. A continuación, se estudiarán sus comportamientos y elementos esenciales que serán tomados como indicativo para definir las principales mecánicas de este subgénero. Entonces se mostrará una captura de pantalla del videojuego.

Figura 1. Captura de pantalla del videojuego Alone in the Dark.

Como se puede apreciar, la cámara se encuentra a la espalda del jugador o sea, en tercera persona, no obstante dicha cámara puede ser cambiada a la primera persona (vista desde la perspectiva del personaje); esta no tiene un comportamiento extra, solo el de seguimiento del personaje, no cuenta con acercamiento o alejamiento ni otras funciones, teniendo en cuenta el comportamiento de la misma y su movimiento, se puede relacionar con las mecánicas físicas, de ahora en adelante será nombrada la mecánica **Cámara.**

Además, se puede apreciar que la interfaz presente es bastante, por no decir en su totalidad limpia, mostrando únicamente el entorno y al PJ (personaje jugable); no existe una barra de salud, pero esta es representada mediante el enrojecimiento de la pantalla cada vez que es dañando el PJ. En ocasiones se pueden visualizar pequeñas imágenes las cuales corresponden a acciones posibles por parte del PJ al acercarse a ciertos objetos o puertas. Por lo tanto, como la interfaz controla la interacción con los elementos de la escena y la salud (elemento de economía interna) del PJ, estos mecanismos serán llamados **Interfaz.**

El usuario puede acceder al inventario presionando un botón donde puede interactuar de determinadas formas con los objetos almacenados; los cuales presionando el botón de acción al tener seleccionado el objeto, se pueden usar para diferentes finalidades, como pueden ser, aumentar la salud, desbloquear puertas o accionar otros objetos. En los mecanismos de inventario se puede apreciar la presencia de la economía interna y mecanismos de progresión al ser algunos de los objetos guardados necesarios para el progreso del PJ, por lo que se decide nombrar **Inventario** a esta mecánica que gestiona todo lo anteriormente planteado.

El PJ es capaz de desplazarse ya sea caminando o corriendo, en este caso esto es posible mediante teclas en específico que permiten dichas acciones, por ejemplo, si se presiona la tecla “w” se moverá hacia delante mientras camina, y si a esta acción se le suma la tecla “shift” procederá a correr. La orientación del movimiento se controla mediante el ratón el cual se encarga de denotar que es “adelante” y “atrás”. Por otra parte, existen personajes no jugables (PNJ) que son capaces igualmente de desplazarse, aunque estos usan un sistema diferente al usado por el PJ. Estas acciones se incluyen dentro de las mecánicas físicas, pero serán llamadas por el autor como **Movimiento** a la mecánica que recoge todos los mecanismos encargados de la locomoción de las unidades.

El PJ puede interactuar con los objetos y puertas, en ocasiones para estos primeros agarrándolos o añadiéndolos al inventario, y los últimos para abrir y cerrar según sea posible y si se cuenta con los objetos necesarios para desbloquearlas. Esto se encuentra relacionado con la economía interna al controlar recursos, al igual que el de progresión. Para el presente trabajo se nombrará **Interacción** a los mecanismos encargados de activar otros mecanismos y a la ejecución de acciones por parte del PJ o de los PNJ.

En estos videojuegos siempre hay presentes uno o varios PNJ encargados de frenar, entorpecer el avance o eliminar al jugador; que utilizando mecánicas de movimiento o interacción pueden tener comportamientos como la persecución, la toma de decisiones, la detección del PJ en un rango determinado, vagar o deambular en espera de que el PJ se encuentre en su rango de visión y proceder a atacarlo. Dicho lo anterior y tomando en consideración el tiempo que se tiene para la investigación, además de la complejidad que pueden presentar estos PNJ se implementarán solamente unas IAs (inteligencia artificial) con comportamientos simplificados, pero con capacidad de detectar al PJ, perseguirlo y deambular en ciertos puntos. Por lo tanto, los mecanismos que englobarán estás mecánicas serán llamadas **IA** en el presente trabajo.

En el videojuego existe la posibilidad de conducir autos, para desplazarse de un punto a otro. También se puede interactuar con partes del coche como la guantera, o tocar el claxon e incluso poner la radio. Dicha mecánica se nombrará **Conducción**.

El PJ puede progresar en la trama no mediante acertijos, puzles complicados o pulsar combinaciones específicas de teclas en un panel. Los puzles son del estilo de las aventuras gráficas en las cuales se interactúa con el escenario para buscar soluciones. Por ende, la mecánica será nombrada **Aventura gráfica**.

Habiendo definido las mecánicas principales del subgénero mediante el estudio de Alone in the Dark se procederá a compararlo con los otros dos ejemplos tomados y corroborar cuantas de estas mecánicas se mantiene o son similares.

1.4.1 Análisis de los videojuegos Resident Evil y Silent Hill

A continuación, se procede a analizar las mecánicas y los mecanismos presentes en *Residente vil* y *Silent Hill,* herederos del primer *Alone in the Dark*. Videojuegos que son referentes en la industria cuando de *survival horror* se trata.

Resident Evil: En 1996 Capcom desarrolló y distribuyó *Resident Evil*, un videojuego ambientado en un pueblo donde suceden cosas extrañas por lo que son enviados Chris Redfield y Jill Valentine para investigar la desaparición de su equipo enviado anteriormente con órdenes de indagar que sucedía [8]. En la siguiente imagen hay una captura de pantalla del juego.

Figura 2. Captura de pantalla del videojuego Resident Evil.

Silent Hill: fue desarrollado por *Team Silent* y publicado por Konami el 31 de enero de 1999, es un juego de acción y *Survival Horror* que cuenta con una cámara en tercera persona y entornos tridimensionales en tiempo real, la obra presenta a un protagonista sin cualidades especiales o destreza física por sobre lo normal [11]. A continuación, se muestra una captura del videojuego.

Figura 3. Captura de pantalla del videojuego Silent Hill.

Mediante una tabla se hará comparativa entre las mecánicas de estos títulos con respecto al videojuego *Alone in the Dark.* El contenido a tomar en cuenta serán los mecanismos definidos en las secciones anteriores, y se pondrá “si” en caso que sean similares y “no” en caso contrario.

Tabla 1. Descripción de las mecánicas de los videojuegos Resident Evil y Silent Hill.

Mecánica	Resident Evil	Silent Hill
Cámara	Sí Aunque no presenta una cámara en primera persona, se sigue manteniendo una en tercera.	Sí Tiene la misma característica que Resident Evil.
Interfaz	Sí Excepto que no se muestran imágenes cuando se es posible interactuar con objetos	Sí Tiene la misma característica que Resident Evil.
Inventario	Sí	Sí
Movimiento	Sí	Sí
Interacción	Sí	Sí
IA	Sí	Sí

Conducción	No No existe ningún tipo de conducción o interacciones con vehículos dentro del videojuego	No Al igual que Resident Evil, no presenta ningún tipo de interacción con vehículos.
Aventura gráfica	No La progresión en este videojuego es mediante puzles y búsqueda de llaves que no coinciden con el estilo de aventura gráfica.	No Al igual que Resident Evil, no presenta ningún tipo de puzles o progresión, que concuerde con el estilo de aventura gráfica.

Teniendo conocimiento de la tabla anterior, muchos de sus mecanismos son grandemente similares en estos videojuegos a pesar de la diferencia de años entre el primer juego con el que son comparados; por lo tanto, gran parte de sus características se mantienen en la medida de lo posible, exceptuando la conducción, la aventura gráfica y aquellas mejoras para la jugabilidad que van en consonancias con el avance tecnológico de las épocas. Aunque no se tenga una cámara en primera persona en Resident Evil ni Silent Hill, sí que presentan los tres una cámara en tercera persona, un poco diferentes, aunque con una interfaz limpia.

Las mecánicas seleccionadas para la solución son aquellas marcadas con Sí en la tabla número 1, al estas ser coincidentes en los tres videojuegos, y las que fueron descartadas, se los marcó con un No, al ser mecánicas que aportan al videojuego al que pertenecen, pero no son necesarias para la presente investigación. A continuación, se detallan los mecanismos identificados que no pueden faltar en la solución y estos serán detallados:

Cámara: Debe ser en tercera persona o primera persona, ambas formas tienen sus ventajas; mientras que la tercera persona proporciona una vista amplia de la zona donde se encuentra el PJ, la otra representa más la cercanía de vivir la experiencia desde los ojos del PJ propiciando una experiencia más inmersiva. Por lo tanto, se crearán dos mecánicas que contengan cada uno de los tipos dichos, y el movimiento de ambas serán guiadas por el desplazamiento del jugador y la orientación por el ratón (en el caso de la primera

persona), además de que esta se limitará en el eje vertical simulando el límite que puede un ser humano mirar arriba o abajo.

Interfaz: La interfaz será lo más limpia posible, sin mostrar ningún tipo de imágenes, esto permite una mayor inmersión. El único efecto mostrado será la representación de la salud cuando el PJ sea dañado.

Inventario: El inventario será accesible mediante un botón, este contará con varios espacios para almacenar objetos utilizables. El inventario será programado de forma que no se pausará la acción por lo que supondrá un desafío para los jugadores. Los artículos almacenados pueden ser usados para restaurar la salud o desbloquear zonas, puertas o lugares previamente inaccesibles en la escena.

Movimiento: Existen tres variaciones en el movimiento: caminar, correr y saltar; el PJ realiza estos mediante teclas; al presionar la tecla definida caminará, al igual que para saltar y la acción de correr por una combinación de dos teclas, en ambos casos la velocidad de desplazamiento no será muy grande para reforzar el desafío del jugador al tener un PJ vulnerable y lento. Junto a lo dicho anteriormente, en el movimiento también participa el ratón, este controla el cambio de dirección.

Interacción: La interacción se presenta de dos formas, la primera es la toma de objetos en la escena, el guardado en el inventario y su posterior toma para accionar otros mecanismos; y la segunda es la interacción con GameObjects (básicamente cualquier objeto que se use en el Unity) en la escena; ya sea el abrir o cerrar puertas, o activar objetos que tengan alguna funcionalidad.

IA: Los PNJ contarán con un sistema que le permita realizar algunas tareas: Estos contarán con un mecanismo de movimiento (apoyándose de una malla de navegación) e interacción. Sus principales funciones serán las de detección del PJ y la consiguiente persecución de este hasta que sea atrapado o la pérdida de vista, con lo que regresaría a deambular hasta su próxima interacción. El mecanismo de detección contará con dos modalidades; una en área, donde el PNJ no necesitará tener contacto visual con el PJ para perseguirlo, ya que con solo estar dentro de su radio de acción será detectado, esto puede ser usado en enemigos que representen animales con características especiales ya sea un olfato avanzado u otros, y la segunda modalidad, con un rango de visión limitado para el rastreo.

Teniendo explicado todas las mecánicas que debe tener la solución, se investigará sobre el proceso de desarrollo. Así como sobre las herramientas que se usarán en la presente investigación.

1.5 Unity

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

Dentro de las funcionalidades típicas que tiene un motor de videojuegos, se encuentran las siguientes:

- Motor gráfico para renderizar gráficos 2D y 3D

- Motor físico que simule las leyes de la física

- Animaciones

- Sonidos

- Inteligencia Artificial

- Programación o scripting

La solución se desarrollará en el motor de videojuegos Unity en su versión 2019.3.1f1, antes llamado Unity 3D; es un software que centraliza todo lo necesario para poder desarrollar videojuegos. Es decir, Unity es una herramienta que permite crear videojuegos para diversas plataformas (PC, videoconsolas, móviles, etc.) mediante un editor visual y programación vía scripting, llegando a la posibilidad de conseguir resultados totalmente profesionales. Prueba de ello son juegos muy famosos creados con Unity; tales como "*Monument Valley", "Gris" o "Cuphead".* Además, es muy utilizado en la mayoría de desarrollos de videojuegos para móvil [13]. Y es el motor gráfico utilizado por el Centro.

Uno de los grandes puntos fuertes que tiene Unity es la gran comunidad de usuarios que tiene. Esto permite tener acceso a multitud de documentación, foros y comunidades donde se preguntan y resuelven dudas, donde se explican diferentes métodos y técnicas nuevas, etc.

En Unity toda la creación gira en torno a su editor visual para crear juegos. El contenido del juego se crea desde dicho editor; y no es otra cosa que su interfaz principal. A continuación, se describe cada parte:

1.5.1 Interfaz

Una vez abierto Unity, esto es lo que se verá, es la interfaz habitual de este motor de videojuegos. Se tienen imágenes de un proyecto tomadas como ejemplo [13].

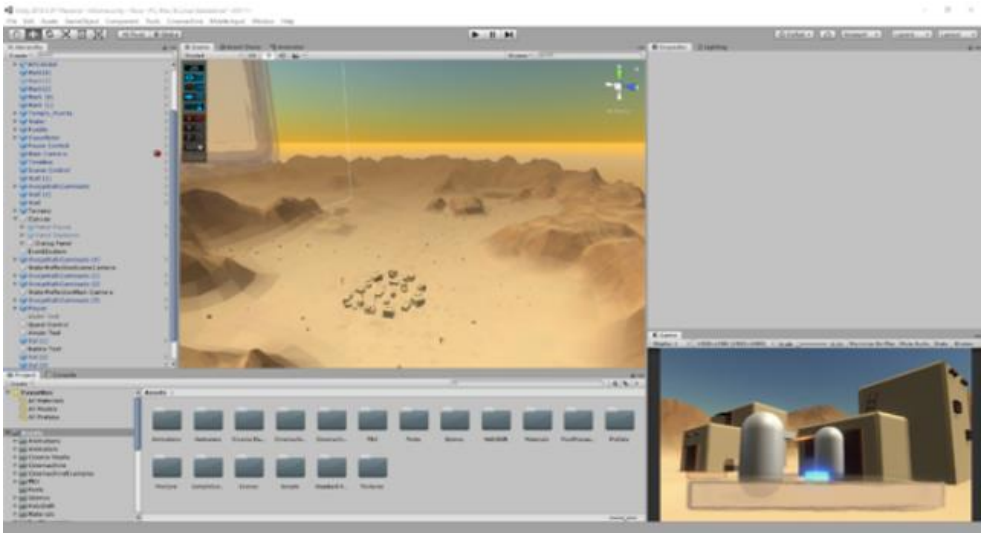


Figura 4. Interfaz de Unity [13].

Inspector View o Inspector:

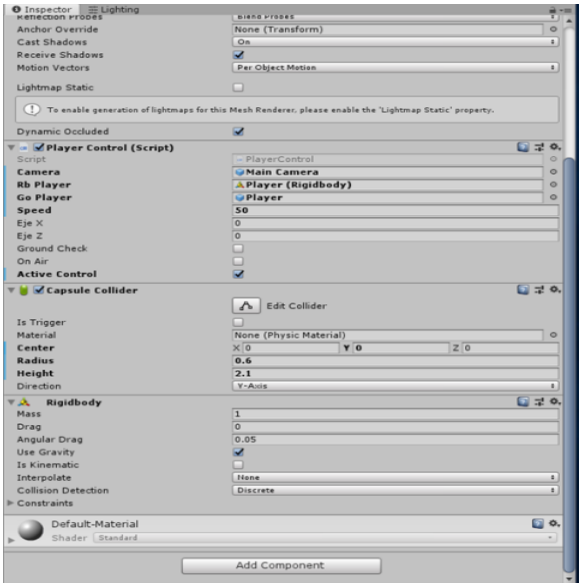


Figura 5. Inspector [13].

En esta parte se mostrará la información del objeto que se tiene seleccionado en ese momento. Se mostrarán los componentes, el material que tiene en caso de que tenga alguno, así como los scripts asociados y posición en la escena. Desde aquí se editan las propiedades más inmediatas de los objetos. Se puede bloquear las pestañas desde el candadito de arriba a la derecha para que estas no cambien a la hora de clicar otro objeto [13].

1.5.2 Malla de navegación

El sistema de Navegación le permite a usted crear personajes que pueden navegar el mundo del juego. Les da a sus personajes la habilidad de entender que necesitan tomar las escalares para alcanzar un segundo piso, o saltar sobre una zanja. El sistema de *NavMesh* de Unity consiste de las siguientes piezas [14]:

7- *NavMesh* (abreviación para *Navegation Mesh*) es una estructura de datos que describe las superficies caminables del mundo del juego y permite encontrar el camino de una ubicación caminable a otra. La estructura de dato es construida, o *baked*, de manera automática de la geometría de su nivel.
8- El componente NavMesh Agent ayuda a crear personajes que se evitan entre sí mientras se mueven hacia su objetivo. Los Agents razonan acerca del mundo de juego utilizando el NavMesh y saben cómo evitarse a cada uno al igual que obstáculos que se muevan.
9- El componente Off-Mesh Link permite incorporar atajos de navegación los cuales no pueden ser representados utilizando una superficie caminable. Por ejemplo, saltar sobre una zanja o valla, o abriendo una puerta antes de que se pueda caminar a través de ella, puede ser todo descrito como enlaces Off-mesh.
10- El componente NavMesh Obstacle permite describir obstáculos que se mueven, los cuales los agentes deberían evitar mientras navegan el mundo. Un barril o una caja controlada por el sistema de física es un buen ejemplo de un obstáculo. Mientras que el obstáculo se esté moviendo los agentes harán lo mejor para evitarlo, pero una vez que el obstáculo se vuelve estacionario, se va abrir un hueco en el navmesh para que los agentes cambien su camino y lo rodeen, o si el obstáculo estacionario está bloqueando el paso, los agentes encuentren otra ruta.

1.5.3 Conceptos relacionados

Escena: Una escena es la que contiene todos los objetos del juego [14]. Para una mejor compresión se debe ver la figura 4, todo los GameObjects presentes en la vista de jerarquía y todo lo visible desde la interfaz forma parte de la escena (con excepción de la parte donde pueden ser visualizadas algunas carpetas), por lo que, al crearse una nueva escena estos no estarán presentes y deben ser creados nuevos GameObjects

GameObject: A grandes rasgos todo lo presente en la escena es un GameObjects, por tanto, esto se refiere de forma global a todo aquello que represente un personaje, un sonido, luces, objetos, el mismo escenario e incluso aquellos que no son visibles. Todos los GameObjects pueden ser visualizados en la vista de jerarquía, ver la figura 6.

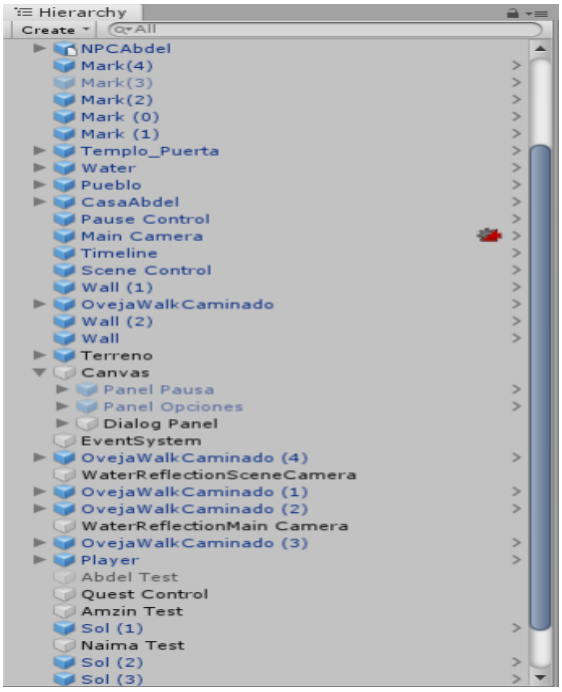


Figura 6. Vista de Jerarquía.

Script: Un *script* son instrucciones escritas en lenguaje de programación, ya sea en C-Sharp, *javascript*, etc, que hacen de puente entre los *GameObjects* y el funcionamiento interno de Unity. Implementando las funcionalidades de la clase MonoBehaviour; los scripts son añadidos a los objetos como componentes.

MonoBehaviour: Es una clase de programación que contiene variables y funciones previamente implementadas. Todos los scripts heredaran de esta clase y tendrán por defecto algunas funciones, como start, por ejemplo, que se ejecutara al iniciar el juego.

Componentes: Los componentes son una parte muy importante de Unity ya que estos son los que permiten los movimientos, las físicas, darle comportamientos y propiedades a los GameObjects, entre otras cosas. Ver la figura 5. Ejemplo de estos componentes se tienen:

11- Colliders: Es un componente que su función principal es la de definir las dimensiones del GameObject y permitir las colisiones físicas.
12- Rigidbody: El RigidBody, es el componente encargado de ponerle físicas al GameObject, como la como gravedad, para que estos se muevan de forma realista.

1.6 Definición del marco de trabajo

Se decidió que para la presente solución se utilizará la metodología de desarrollo de software XP (Programación extrema, por sus siglas en inglés, Extreme Programming) debido a su alto índice de eficiencia para las pruebas y planificación; al igual que su tasa de errores es bastante pequeña y los cambios son fáciles de efectuar, todo esto es útil para el desarrollo del paquete que se desea implementar. Se utilizará Visual Studio Code en su versión 1.67.2 que a pesar de no ser un IDE (Entorno de desarrollo integrado por sus siglas en inglés, *Integrated Development Environment*) por completo, sí que proporciona las herramientas que un desarrollador necesita para un ciclo rápido de código-construcción-depuración, se trabajará con el lenguaje de programación C-Sharp, el cual cuenta con una amplia documentación al respecto, además de su extensivo uso en el centro. Por último, como herramienta CASE (Ingeniería de Software Asistida por Computadora por sus siglas en inglés, *Computer Aided Software Engineering*) se utilizará Visual Paradigm 5.0 gracias a la gran experiencia que se tiene en el centro con dicha herramienta. Ahora, se procederá a caracterizar los mismos.

Metodología XP

La metodología XP es un conjunto de técnicas que dan agilidad y flexibilidad en la gestión de proyectos. También es conocida como Programación Extrema (Extreme Programming) y se centra en crear un producto según los requisitos exactos del cliente. De ahí, que le involucre al máximo durante el método de gestión del desarrollo del producto [15].

El uso de esta metodología supone, para muchos teóricos, una aproximación a la calidad óptima del producto. Pues durante el ciclo de vida del software, ocurren cambios naturales. Es más, cuantos más cambios, puede que más cerca se esté del mejor resultado que espera el cliente. Por eso, este cambio constante en el proyecto se llega a considerar como

favorable. Y si es posible aplicar una manera dinámica de gestionarlos, mejor. Esta forma es conocida como metodología XP [15].

El diseño XP sigue rigurosamente el principio MS (mantenlo sencillo). Un diseño sencillo siempre se prefiere sobre una representación más compleja. Además, el diseño guía la implementación de una historia conforme se escribe: nada más y nada menos. Se desalienta el diseño de funcionalidad adicional porque el desarrollador supone que se requerirá después [16].

Sus características principales son:

1- Participación del cliente.
2- La planificación es flexible y, el flujo de trabajo, sin presiones.
3- Hay una lista predeterminada de funciones para cada miembro del equipo.
4- Respuesta rápida a los cambios constantes.
5- El software que funciona está por encima de cualquier otra documentación.
6- Es importante la simplicidad.
7- Uso de pocos artefactos.
8- El cambio se acepta mediante liberaciones regulares del sistema a los clientes.

Visual Paradigm

Visual Paradigm es una herramienta CASE: Ingeniería de Software Asistida por Computación. La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.

Ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Constituye una herramienta privada disponible en varias ediciones, cada una destinada a satisfacer diferentes necesidades [21].

Visual Paradigm es una aplicación de software diseñada para que los equipos de desarrollo de software modelen los sistemas de información empresarial y gestionen los procesos de desarrollo. Además del soporte de modelado, esta tecnología proporciona capacidades de generación de informes y de ingeniería de código, incluyendo la generación de código [22].

Visual Studio Code

Es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para depuración, control de Git integrado, resaltado de sintaxis, finalización de código inteligente, fragmentos de código y refactorización de código. También es personalizable, de modo que los usuarios pueden cambiar el tema del editor, los métodos abreviados de teclado y las preferencias. Es gratuito y de código abierto [19].

El código combina la interfaz de usuario optimizada de un editor moderno con asistencia y navegación de código enriquecido y una experiencia de depuración integrada, sin la necesidad de un IDE completo. Visual Studio Code, cuenta con herramientas de Debug hasta opciones para actualización en tiempo real el código en la vista del navegador y compilación en vivo de los lenguajes que lo requieran (por ejemplo, en el caso de SASS a CSS). Además de las extensiones, se tendrá la posibilidad de optar por otros temas o bien configurarlo al gusto del usuario para modificar el esquema de colores y los iconos [19].

C-Sharp

C-Sharp es un lenguaje de programación moderno, orientado a objetos y de tipo seguro; permite a los desarrolladores crear muchos tipos de aplicaciones seguras y robustas que se ejecutan en .NET. C# tiene sus raíces en la familia de lenguajes C y resultará inmediatamente familiar a los programadores de C, C++, Java y JavaScript. C# es un lenguaje de programación orientado a objetos y a componentes que proporciona construcciones de lenguaje que soportan directamente estos conceptos, que lo convierte en un lenguaje natural para crear y utilizar componentes de software. Desde su origen, ha añadido características para soportar nuevas cargas de trabajo y prácticas de diseño de software emergentes. En su esencia, C# es un lenguaje orientado a objetos donde se definen los tipos y su comportamiento [20].

1.7 Consideraciones parciales

Durante todo el proceso de desarrollo de este capítulo se alcanzó un mayor entendimiento de todo lo relacionado a la solución y sus principales conceptos, por lo que fue posible arribar a las siguientes conclusiones:

1- La definición de los principales conceptos asociados al dominio de la presente investigación y las relaciones entre estos, permitió alcanzar una mayor comprensión de la propuesta de solución.

2- El estudio de los tipos de mecánicas y la comparativa de tres videojuegos base, permitió determinar el desarrollo de seis mecánicas imprescindibles a utilizar en la propuesta de solución: cámara, interfaz, inventario, movimiento, interacción e IA.
3- El análisis de la metodología de desarrollo, así como las herramientas, tecnologías y lenguajes de programación, permitió especificar el ambiente de desarrollo para la propuesta de solución.

CAPÍTULO 2. PROPUESTA SOLUCIÓN

Este capítulo contendrá la explicación de las características esenciales de la solución y se obtendrá una estimación de tiempo. Se presentan los resultados que se obtuvieron una vez cumplidas las fases de Exploración, Planificación y Diseño que propone la metodología XP. Se detallan las características de la solución, se especifican los requisitos no funcionales, las historias de usuario (como requisitos funcionales) y se presentan los artefactos generados para dar solución al problema planteado en la investigación.

2.1 Propuesta solución

Después de analizar las necesidades del sistema para dar solución a la problemática, se desarrollará un paquete para videojuegos Survival Horror (SH) para Unity. Dicha solución estará compuesta por los mecanismos definidos en el capítulo anterior (Cámara, Interfaz, Inventario, Movimiento, Interacción e IA), los cuales fueron seleccionados como mecánicas recurrentes en este tipo de videojuegos. Dicho paquete debe permitir su reutilización en cualquier nuevo proyecto.

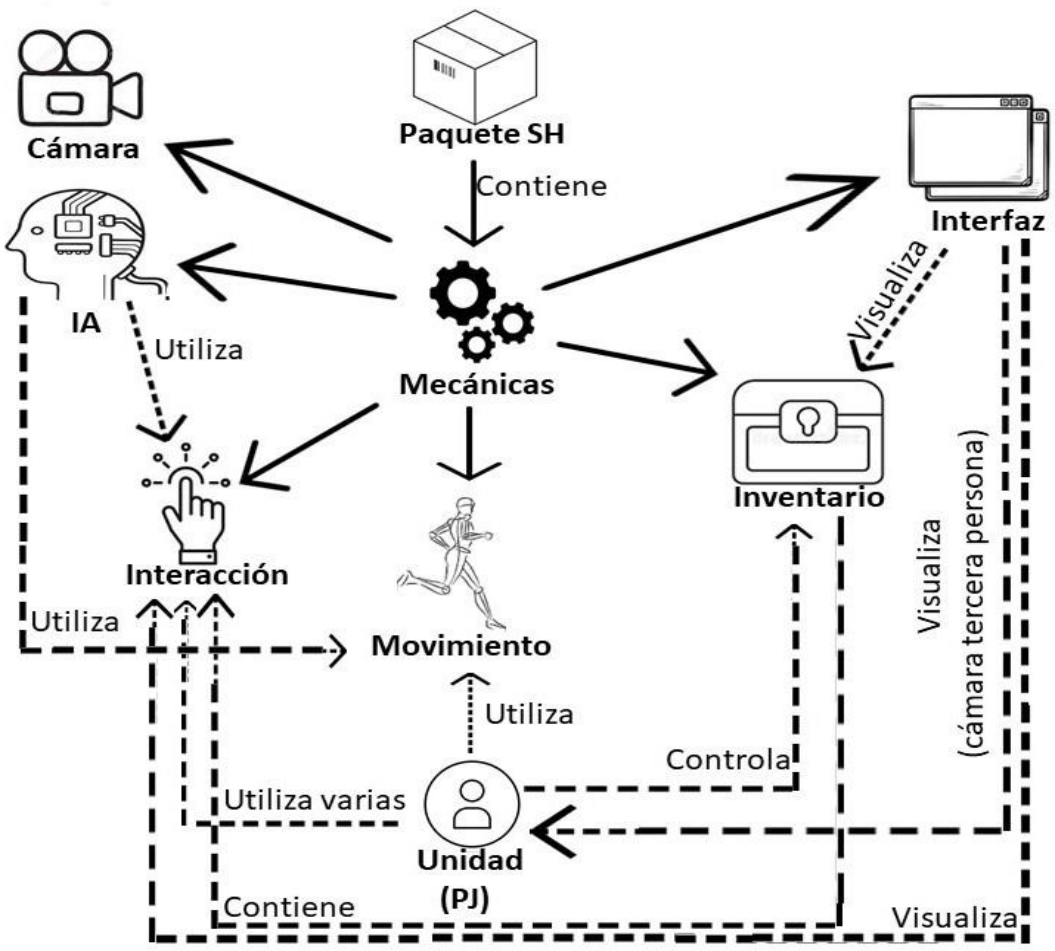


Figura 7. Estructura del paquete propuesto.

Cámara: Como su nombre indica, es el mecanismo encargado de proporcionar al usuario una cámara funcional en primera persona o en tercera persona, estás permitirán la visualización de todos los elementos presentes en el escenario.

IA: Este mecanismo es el encargado de proporcionar un PNJ capaz de antagonizar al PJ, y que es una de las partes primordiales de este tipo de videojuegos en los cuales usualmente el objetivo es huir. La presente mecánica funciona mediante la utilización de movimiento apoyado por el NavMesh para su correcto desplazamiento, al igual que algunas interacciones como la apertura automática de puertas o la eliminación del PJ si es atrapado. La IA tendrá dos tipos diferentes y que serán utilizados según las necesidades del videojuego.

Interacción: Esta mecánica garantiza que se pueda interactuar con los GameObjects, mediante botones previamente definidos y será visualizado la posible ejecución gracias a

un resaltado de los objetos, como, por ejemplo, la apertura de puertas, además de relacionarse con el inventario.

Movimiento: Permite al usuario la traslación del PJ mediante dos formas, correr o caminar, mediante el pulsado de teclas. Los parámetros, tales como velocidad o salto pueden ser configurables.

Inventario: Proporciona al usuario un espacio donde guardar objetos (ya sean curativos o aquellos necesarios para avanzar en el videojuego); esta mecánica es controlada directamente por el PJ mediante la interfaz que permite su visualización, además de contener interacciones. El inventario puede ser accionado en tiempo real en el que la acción del escenario no se detiene.

Interfaz: Al ser un paquete para Survival Horror, la interfaz muestra pocas cosas, siendo esta limpia casi en su totalidad, permitiendo solo la visualización del escenario, la vida cuando el PJ es dañado, el resaltado de los objetos con los que es posible interactuar o el inventario una vez se abre.

2.2 Requisitos no funcionales

Son limitaciones sobre servicios o funciones que ofrece el sistema. Incluyen restricciones tanto de temporización y del proceso de desarrollo, como impuestas por los estándares [17].

Software

RNF1. El sistema debe tener compatibilidad con Unity en sus versiones de 2019.3.1f1 o superiores.

Usabilidad

RNF2. El sistema solo debe ser utilizado con personal con experiencia en el uso del motor de videojuegos Unity.

Restricciones de diseño e implementación

RNF3. El sistema solo puede ser usado en el motor de videojuegos Unity.

RNF4. El sistema debe desarrollarse en el lenguaje de programación C#.

Extensibilidad

RNF5. El sistema debe permitir la incorporación de nuevas características para facilitar su crecimiento.

2.3 Fase de exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología [23]. En la presente fase se estudian las funcionalidades necesarias a desarrollar para lograr el éxito del proyecto.

2.3.1 Historia de usuario

Una historia de usuario es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final o cliente. El propósito de una historia de usuario es articular cómo un elemento de trabajo entregará un valor particular al cliente [24]. A continuación, se muestran algunas Historias de usuario con respecto al funcionamiento del paquete en Unity. Las demás pueden ser encontradas en el anexo 1.

Tabla 2. HU Insertar y configurar cámara.

Historia de usuario	
No 1: Insertar y configurar cámara	
Prioridad: alta	**Nivel de complejidad: alto**
Estimación: 5 días	**Iteración asignada:** 1
Descripción: El usuario puede añadir a la escena cualquiera de los dos tipos de cámara existentes, modificar los parámetros de estas en el inspector.	
Observaciones:	

Tabla 3. HU Definir y configurar los GameObjects interactuables.

Historia de usuario	
No 5: Definir y configurar los GameObjects interactuables	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 5 días	**Iteración asignada:** 1
Descripción: El usuario mediante la adición de script a los GameObjects puede hacerlos interactuables y añadir nuevas formas mediante la edición del script en cuestión.	
Observaciones: El prefabricado del HU 4 debe estar previamente creado.	

Tabla 4. HU Insertar y configurar mecánica de movimiento.

Historia de usuario	
No 7: Insertar y configurar mecánica de movimiento	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 3 días	**Iteración asignada:** 2
Descripción: El usuario podrá añadir el prefabricado que posibilita la ejecución de los movimientos de caminar, correr o saltar y configurar sus parámetros.	
Observaciones:	

2.4 Fase de planificación

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se

determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en unas pocas semanas [16].

2.4.1 Plan de iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. A continuación, se muestran las iteraciones necesarias para el desarrollo de la solución.

Tabla 5. Distribución de iteraciones por Historia de Usuarios.

Iteraciones	Historia de usuario a implementar	Tiempo de trabajo (días)
Iteración 1	Insertar y configurar cámara. Insertar y configurar mecánica de IA. Definir objetivo, movimiento y objetos con lo que interactúa la IA. Insertar y configurar mecánicas de interacción. Definir y configurar los GameObjects interactuables. Visualizar indicador de objeto interactuable.	25
Iteración 2	Insertar y configurar mecánica de movimiento Ejecutar el movimiento del personaje Insertar y configurar script de salud Insertar y configurar mecánica de inventario	23
Iteración 3	Acceder al inventario Interactuar con el inventario	12
Iteración 4	Configurar bloqueo y desbloqueo de puertas Brindar al usuario una base para las interacciones Configurar Interfaz	13

Iteración 5	Definir las regiones de movimiento del agente	7

2.4.2 Plan de entrega

Esta clase de documento es un contrato marco con el cliente que contiene las cantidades y fechas de entrega. El plan de entregas se cumple por medio de la creación de entregas en el plan tal y como se van venciendo los requisitos [25]. Por tanto, es el compromiso final con el cliente y su demora provocaría insatisfacción por parte del cliente. Ahora se definirán las versiones a entregar en cada iteración al finalizar estas, las cuales serán entregadas en la última semana de los cinco meses en que se desarrolla el producto.

Tabla 6. Plan de entregas del producto.

Producto	1ra Iteración	2da Iteración	3ra Iteración	4ta Iteración	5ta Iteración
Paquete de mecánicas para videojuegos tipo survival horror	Versión 0.1 del producto	Versión 0.2 del producto	Versión 0.3 del producto	Versión 0.4 del producto	Versión 1 del producto
Fecha de inicio	20-04-2022	15-05-2022	7-06-2022	19-06-2022	2-07-2022
Fecha de entrega	14-05-2022	6-06-2022	18-06-2022	1-07-2022	8-07-2022

2.5 Fase de diseño

En este paso se intentará trabajar con un código sencillo, haciendo lo mínimo imprescindible para que funcione. Se obtendrá el prototipo. Además, para el diseño del software orientado a objetos, se crearán tarjetas CRC (Clase-Responsabilidad-Colaboración) [15]. Ahora se describen lo trabajado en la fase.

2.5.1 Descripción de la arquitectura

La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de

responsabilidades [26]. Por lo tanto, se procederá a ajustar dicha arquitectura a la solución, en la próxima figura se muestra.

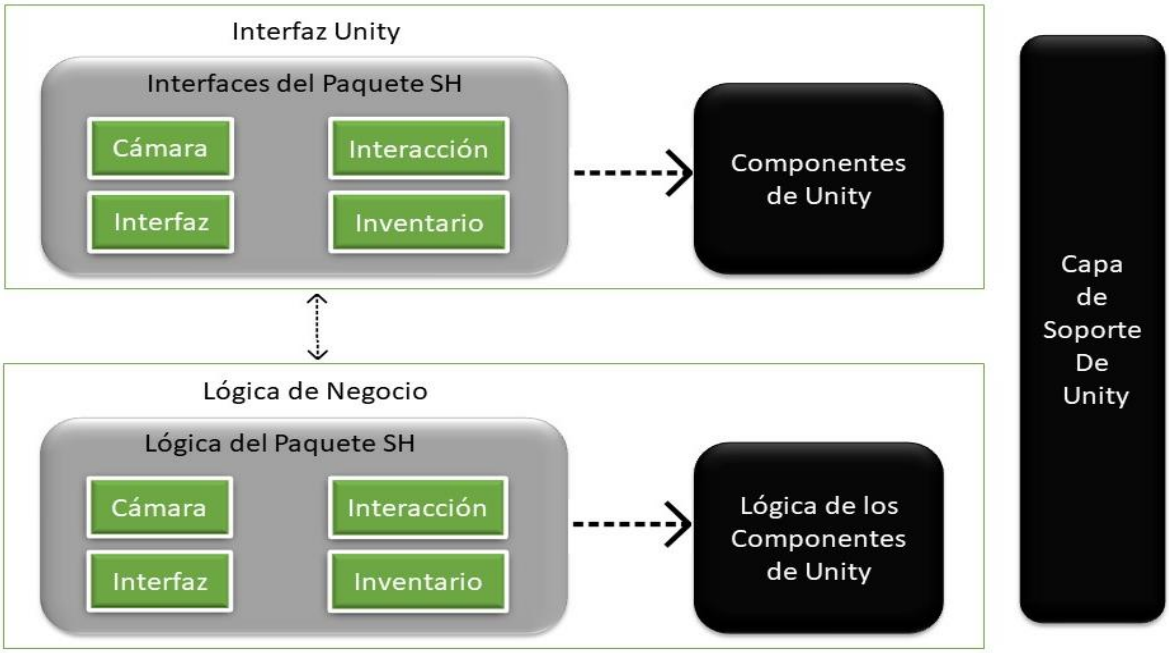


Figura 8. Diseño de la arquitectura.

Interfaz Unity: En esta capa se visualizan las interfaces con las que el usuario puede interactuar, aquí se encuentran el paquete “Interfaces del paquete SH” que contiene las interfaces de la solución. Estas tienen dependencia del paquete de “Componentes de Unity”.

Lógica de Negocio: En esta capa se encuentra la lógica de cada componente de la solución en el paquete “Lógica del Paquete SH”, y posee una dependencia del paquete de “Lógica de los Componentes de Unity”.

Capa de soporte de Unity: Tiene las librerías necesarias para las físicas, datos, sonidos y gráficos que aporta el motor de videojuegos. Esta información puede ser usada por las demás capas sin problemas.

2.5.2 Patrones de diseño

Los patrones de diseño se derivaron de ideas planteadas por Christopher Alexander (Alexander et al., 1977), quien sugirió que había ciertos patrones comunes de diseño de construcción que eran relativamente agradables y efectivos. El patrón es una descripción

del problema y la esencia de su solución, de modo que la solución puede reutilizarse en diferentes configuraciones. El patrón no es una especificación detallada. Más bien, puede considerarla como una descripción de sabiduría y experiencia acumuladas, una solución bien probada a un problema común. Los patrones y los lenguajes de patrón son formas de describir mejores prácticas, buenos diseños, y captan la experiencia de tal manera que es posible que otros reutilicen esta experiencia [17].

A continuación, se muestran los patrones GRASP (*General Responsibility Asignment Software Patterns*, por sus siglas en inglés) y GoF (*Gang of Four*, por sus siglas en inglés) que se utilizarán en la solución.

Creador: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento [27]. Este se pone de manifiesto en la clase inventario, ya que este se encarga de instanciar objetos y almacenarlos.

```
void Start()
{
    allSlots = slotHolder.transform.childCount;

    slot = new GameObject[allSlots];
```

Figura 9. Fragmento de código del script Inventario.

Experto: Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Da origen a diseños donde el objeto de software realiza las operaciones que normalmente se aplican al objeto real que representa, por lo que ofrece una analogía con el mundo real. Con la utilización de este patrón se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y mantener [27]. Este se constata en la clase Selected, al ser esta quien asigne responsabilidades, como se muestra en la figura 10, en el siguiente fragmento de código hit.collider.transform.GetComponent<**ObjetoInteractivo**>().ActivarObjeto(); se aprecia

cómo se delegan responsabilidades al script Objeto Interactivo para que pueda realizar las labores necesarias.

```
Deselect();
SelectedObject(hit.transform);
if (hit.collider.tag == "ObjetoInteractivo")
{
    if (Input.GetKeyDown(KeyCode.E))
    {
        hit.collider.transform.GetComponent<ObjetoInteractivo>().ActivarObjeto();
    }

}
```

Figura 10. Fragmento de código del script Selected.

De los patrones GoF se utilizará para la solución el siguiente:

Mediador: Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto [28]. Este patrón se emplea principalmente en la clase *MonoBehaviour,* contenedora de todos los métodos y funcionalidades que permiten la relación entre todos los objetos o clases.

2.5.3 Diagrama de clases de la solución

En aras de mejorar el entendimiento de la estructura de los scripts en la solución se tiene el siguiente diagrama de clases.

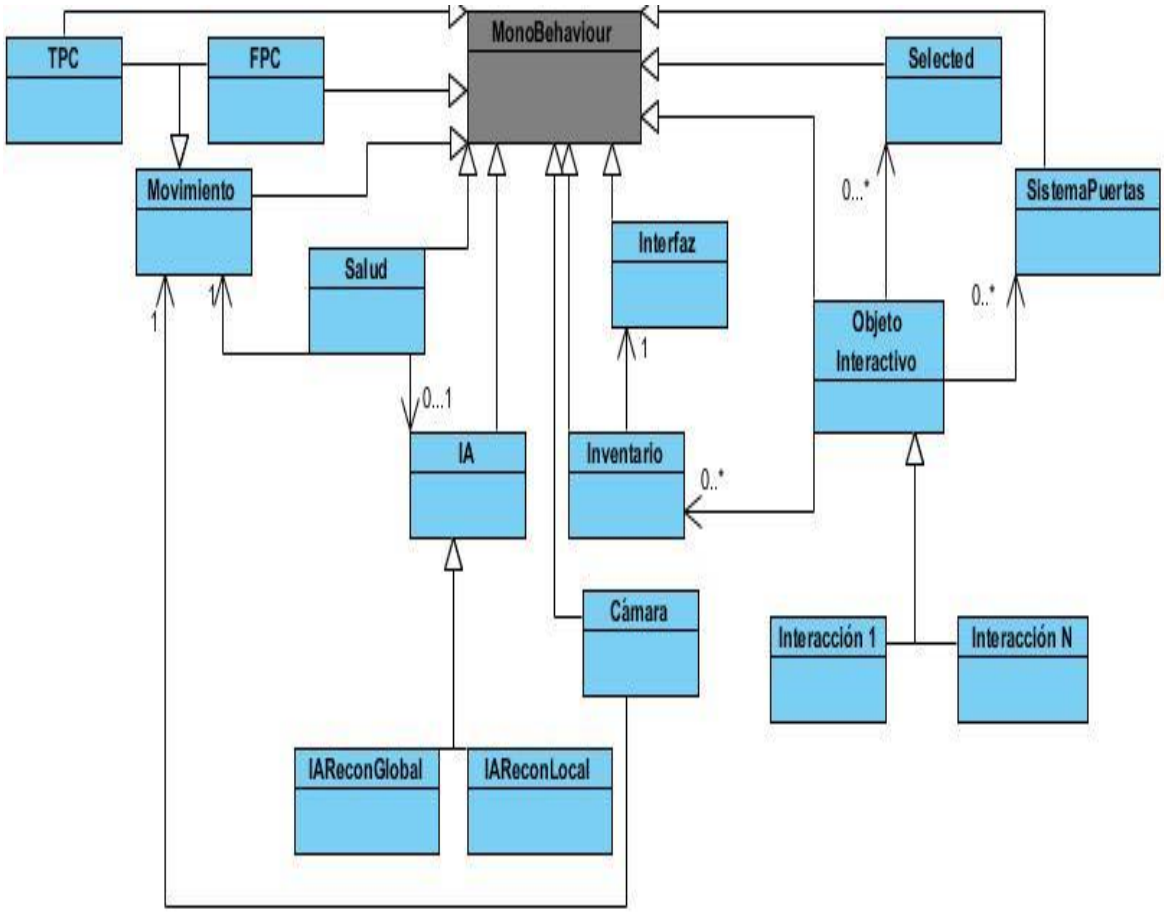


Figura 11. Diagrama de clases de la solución.

En el anterior diagrama las clases de color azul pertenecen a la solución y la clase gris "*MonoBehaviour*" pertenece a Unity de la cual heredan las otras clases, y todos sus objetos o elementos. Ahora, el patrón Mediador se evidencia gracias a la clase de Unity ya que esta permite la comunicación entre los diferentes objetos gracias a varias de sus funciones.

2.5.4 Descripción de las tarjetas CRC

Las tarjetas CRC (Clase, Responsabilidad y Colaboración) son una metodología para el diseño de software orientado por objetos creada por Kent Beck y Ward Cunningham. Es una técnica para la representación de sistemas OO (Orientado a Objeto), para pensar en objetos. Permite ver las clases como algo más que depositorios de datos, sino conocer el comportamiento de cada una en un alto nivel [29]. A continuación, se muestran las tarjetas necesarias en la solución.

Tabla 7. Tarjeta CRC Movimiento.

Tarjeta CRC	
Clase: Movimiento	
Responsabilidad	**Clases relacionada**
9- Permitir el desplazamiento del personaje. **10-** Permitir el salto. **11-** Detener el desplazamiento. **12-** Seleccionar uno de los tipos de Controlador de personaje (TPC o FPC)	13- MonoBehaviour 14- TPC 15- FPC 16- Cámara 17- Salud

Tabla 8. Tarjeta CRC TCP.

Tarjeta CRC	
Clase: TPC	
Responsabilidad	**Clases relacionada**
18- Permite el control del personaje haciendo uso de una cámara en tercera persona.	19- MonoBehaviour 20- Cámara 21- Movimiento

Tabla 9. Tarjeta CRC FPC.

Tarjeta CRC	
Clase: FPC	
Responsabilidad	**Clases relacionada**

22- Permite el control del personaje haciendo uso de una cámara en primera persona.	23- MonoBehaviour 24- Cámara 25- Movimiento

Tabla 10. Tarjeta CRC Salud.

Tarjeta CRC	
Clase: Salud	
Responsabilidad	**Clases relacionada**
26- Maneja la vida del PJ o PNJ **27-** Controla si el PJ o PNJ está muerto	28- MonoBehaviour 29- Movimiento 30- IA 31- Interfaz

Tabla 11. Tarjeta CRC IA.

Tarjeta CRC	
Clase: IA	
Responsabilidad	**Clases relacionada**
32- Controla el desplazamiento de la inteligencia artificial **33-** Maneja el sistema de NavMesh **34-** Verifica las distancias de los objetivos	35- MonoBehaviour 36- Salud 37- IAReconGlobal 38- IAReconLocal 39- SistemaPuertas

Tabla 12. Tarjeta CRC IAReconGlobal.

Tarjeta CRC	
Clase: IAReconGlobal	
Responsabilidad	**Clases relacionada**
40- Define el tipo de reconocimiento de la IA a un espacio extenso a su alrededor ignorando los obstáculos	41- MonoBehaviour 42- IA

Tabla 13. Tarjeta CRC IAReconLocal.

Tarjeta CRC	
Clase: IAReconLocal	
Responsabilidad	**Clases relacionada**
43- Define el tipo de reconocimiento de la IA a un espacio predefinido bastante limitado sin ignorar los obstáculos	44- MonoBehaviour 45- IA

Tabla 14. Tarjeta CRC Cámara.

Tarjeta CRC	
Clase: Cámara	
Responsabilidad	**Clases relacionada**
46- Controla el movimiento de la cámara **47-** Sigue al PJ si esta es en tercera persona	48- MonoBehaviour 49- Movimiento

Tabla 15. Tarjeta CRC Inventario.

Tarjeta CRC	
Clase: Inventario	
Responsabilidad	**Clases relacionada**
50- Controla los objetos interactuables en su interior	51- MonoBehaviour 52- Interfaz 53- Objeto Interactivo 54- Movimiento

Tabla 16. Tarjeta CRC Interfaz.

Tarjeta CRC	
Clase: Interfaz	
Responsabilidad	**Clases relacionada**
55- Se encarga de mostrar de forma visual con cuales objetos se puede interactuar **56-** Muestra la vida del PJ	57- MonoBehaviour 58- Inventario 59- Salud 60- Objeto Interactivo

Tabla 17. Tarjeta CRC Objeto Interactivo.

Tarjeta CRC	
Clase: Objeto Interactivo	
Responsabilidad	**Clases relacionada**

61- Se encarga de verificar y ejecutar las interacciones de los PJ y PNJ con los objetos del escenario **62-** Activa o desactiva las interacciones	63- MonoBehaviour 64- Selected 65- SistemaPuertas 66- Inventario 67- IA

Tabla 18. Tarjeta CRC Objeto Interactivo.

Tarjeta CRC	
Clase: SistemaPuertas	
Responsabilidad	**Clases relacionada**
68- Permite la interacción con las puertas **69-** Se encarga de verificar que objetos permiten la apertura de las puertas.	70- MonoBehaviour 71- Objeto Interactivo 72- Selected

Tabla 19. Tarjeta CRC Selected.

Tarjeta CRC	
Clase: Selected	
Responsabilidad	**Clases relacionada**
73- Se encarga de que el PJ pueda realizar cualquier interacción con todos los objetos interactuables del escenario **74-** Permite la modificación del color de los objetos interactuables para su identificación	75- MonoBehaviour 76- Objeto Interactivo 77- SistemaPuertas

2.5.5 Estándares de Codificación

En la implementación de la presente solución se definirán ciertos estándares en el desarrollo, en la creación de métodos, variables etc. Esto ayudara a la mejor comprensión del código fuente por parte de los programadores.

Definiciones generales

Las definiciones serán escritas en español para cerciorarse del correcto entendimiento de las mismas.

Clases:

Las clases comenzarán con mayúscula y en caso que sean más de una palabra serán escritas continuas y cada letra inicial en mayúscula.

Ejemplo:

public class PascalCase;

Declaración de variables:

Las variables se escribirán siempre con la primera palabra en minúscula y en caso de que sean necesarias dos o más palabras serán escritas continuas e iniciarán con mayúscula.

Ejemplo:

bool camelCase;

Métodos:

Los métodos al igual que las clases siempre se escribirán con mayúscula y en caso de usar más de una palabra, estas deben estar unidas y cada inicio de palabra con mayúscula.

Ejemplo:

void PascalCase;

2.6 Consideraciones parciales

Después de realizadas las fases de exploración, planificación y diseño de la propuesta de solución y haber generado los diferentes artefactos que dispone la metodología XP, se puede concluir lo siguiente:

1- La definición de las mecánicas abordadas en el capítulo anterior, definieron la estructura final del paquete a desarrollar.
2- El análisis de las características de la propuesta de solución permitió establecer las historias de usuario a desarrollar y los requisitos no funcionales.
3- Con la planificación realizada se estableció el plan de iteraciones con los tiempos de entrega estimados para cada versión del producto y las tarjetas CRC, teniéndose un tiempo total de 80 días, con fecha de inicio en el 20 de abril y conclusión el 8 de julio.
4- La identificación de los patrones de diseño, la arquitectura y el diseño del diagrama de clase, facilitó la visión en cuanto a composición física y lógica del sistema.

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBA

En este capítulo, después de realizada toda la investigación necesaria y el diseño de la propuesta solución, se procederá a realizar la implementación de la solución en iteraciones. Y para culminar se realizan las pruebas que definen si la solución cumple con los objetivos definidos.

3.1 Fase de implementación

La fase de implementación o desarrollo es una parte importante, ya que permite la obtención de resultados por iteración. Nuevas versiones del software se construyen varias veces al día y las versiones se entregan a los clientes [17], para que estos las revisen y retroalimenten el desarrollo hasta la obtención de la versión final del producto. A continuación, se muestran las cinco iteraciones creadas en la planificación del capítulo anterior, junto a las tareas ingenieriles concretas en la realización de cada historia de usuario.

Las tareas de ingeniería ayudan al programador a concluir las tareas de implementación de las historias de usuario a tiempo. Al igual que ofrecen el conocimiento detallado necesario para la implementación, y por esto, casi siempre son divididas en varias tareas.

3.1.1 Primera iteración

En la primera iteración se implementan algunas de las principales HU, como por ejemplo la cámara, los mecanismos que permiten la interacción con los objetos y la IA, y así tener una primera versión funcional de estos. Por lo tanto, se fijaron 7 tareas, dos de ellas se muestran a continuación y las demás pueden ser vistas en el anexo 2.

Tabla 20. Tarea 1 Iteración 1.

Tarea	
Número de tarea: 1.1	**Número de HU:** 1
Nombre de la tarea: Implementación de los atributos configurables y lógica de la clase Cámara	
Tipo de tarea: Implementación	**Estimación:** 5
Descripción: Se declaran los atributos configurables para suavidad de movimiento y distancia que pueden ser modificados en el inspector. Además de que son implementados los mecanismos encargados del funcionamiento de los dos tipos de cámara, como, el acercamiento o alejamiento de la misma.	

Tabla 21. Tarea 7 Iteración 1.

Tarea	
Número de tarea: 1.7	**Número de HU:** 6
Nombre de la tarea: Implementación de la visualización de un objeto interactuable	
Tipo de tarea: Implementación	**Estimación:** 1
Descripción: Se implementa la lógica de la visualización de un objeto en el script Selected, que permita el sombreado de aquellos objetos con los cuales es posible la realización de interacciones.	

Al finalizar la iteración se realizaron las pruebas de aceptación, 1, 2, 3, 4, 5 y 6, las cuales se encuentran en el anexo 4.

3.1.2 Segunda Iteración

En la presente iteración, se incorporan a la versión anterior los mecanismos necesarios para que el personaje ejecute el movimiento, el mecanismo de salud y una primera parte de los mecanismos de inventario. Dos de ellas se muestran a continuación y las demás pueden ser vistas en el anexo 2.

Tabla 22. Tarea 1 Iteración 2.

Tarea	
Número de tarea: 2.1	**Número de HU:** 7
Nombre de la tarea: Implementación de la clase FPC	
Tipo de tarea: Implementación	**Estimación:** 1
Descripción: Se declaran los atributos configurables necesarios para el correcto funcionamiento (ejemplo de esto sería el cursor) y modificación del controlador de tercera persona (TPC)**.**	

Tabla 23. Tarea 3 Iteración 2.

Tarea	
Número de tarea: 2.3	**Número de HU:** 8
Nombre de la tarea: Implementación del movimiento del personaje	
Tipo de tarea: Implementación	**Estimación:** 7

Descripción: implementa la lógica de los controladores de primera persona (FPC) y tercera persona (TPC).

En el acápite 3.2.2 se muestra una de las pruebas de aceptación y en el anexo 4 se encuentran las otras. Dichas pruebas son las No. 7, 8, 9 y 10.

3.1.3 Tercera iteración

Esta iteración incorpora a la solución lo restante del mecanismo de inventario. A continuación, se muestran dos de estas tareas y las restantes se encuentran en el anexo 2.

Tabla 24. Tarea 1 Iteración 3.

Tarea	
Número de tarea: 3.1	**Número de HU:** 11
Nombre de la tarea: Implementación del acceso al inventario	
Tipo de tarea: Implementación	**Estimación:** 5
Descripción: Se adiciona en las funciones implementadas anteriormente en la clase “Inventario”, el código necesario para acceder al inventario de una forma correcta y sin errores.	

Tabla 25. Tarea 3 Iteración 3.

Tarea	
Número de tarea: 3.3	**Número de HU:** 12
Nombre de la tarea: Implementación de la lógica de la clase item	
Tipo de tarea: Implementación	**Estimación:** 4
Descripción: Se implementan los mecanismos principales de la clase “Item”	

En esta iteración se realizaron las pruebas de aceptación, No. 11 y 12, en el acápite 3.2.2 se muestra la prueba número 12 y en el anexo 4 la número 11.

3.1.4 Cuarta iteración

Esta iteración tiene como objetivo incorporar al mecanismo de interacción la lógica para el bloqueo de puertas, así como una base para que el usuario pueda implementar nuevas interacciones, y el prefabricado de la interfaz. A continuación, se muestra una de estas y las demás pueden ser encontradas en el anexo 2.

Tabla 26. Tarea 2 Iteración 4.

Tarea	
Número de tarea: 4.2	**Número de HU:** 14
Nombre de la tarea: Implementación de un script base para crear nuevas interacciones	
Tipo de tarea: Implementación	**Estimación:** 2
Descripción: Se implementa un script llamado “ObjetoInteractivoBase” el cual el usuario podrá usar como base para implementar la lógica de nuevas interacciones, este contiene dentro comentarios para el entendimiento de la lógica.	

En esta iteración se realizaron las pruebas de aceptación, No. 13, 14 y 15, en el acápite 3.2.2 se muestra la prueba número 14 y las demás en el anexo 4.

3.1.5 Quinta iteración

Con la culminación de las tareas programadas en esta iteración se consigue la versión final del paquete con todas sus mecánicas funcionales. A continuación, se muestra una de ellas y la otra puede ser encontrada en el anexo 2:

Tabla 27. Tarea 1 Iteración 5.

Tarea	
Número de tarea: 5.1	**Número de HU:** 16
Nombre de la tarea: Definición de los objetos estáticos y obstáculos	
Tipo de tarea: Configuración	**Estimación:** 3
Descripción: Se definen los GameObjects que funcionarán como el suelo, paredes, y otros, además de aquellos, que tienen función de obstáculos y que por ende deben ser evitados, así como determinar los que serán estáticos.	

En esta iteración se realizó la prueba de aceptación, No. 17, en el anexo 4 se muestra la prueba.

3.2 Fase de prueba

Se deben realizar pruebas automáticas continuamente. Al tratarse normalmente de proyectos a corto plazo, este testeo automatizado y constante es clave. Además, el propio cliente puede hacer pruebas, proponer nuevas pruebas e ir validando las versiones. A

continuación, se describen las pruebas unitarias y las pruebas de aceptación propuestas en la metodología XP.

3.2.1 Pruebas unitarias

Las pruebas unitarias que se crean deben implementarse con el uso de una estructura que permita automatizarlas (de modo que puedan ejecutarse en repetidas veces y con facilidad) o de forma manual. Esto estimula una estrategia de pruebas de regresión siempre que se modifique el código (lo que ocurre con frecuencia, dada la filosofía del rediseño en XP) [16].

Tomando en consideración que las funciones presentes de la solución son pocas y tienen una complejidad igual de baja, se decide realizar las pruebas unitarias de forma manual, para verificar que el contenido visual se muestre correctamente. Dichas pruebas se realizaron en cada iteración.

A continuación, en la figura se muestra un ejemplo de un caso de prueba unitario realizado a los procedimientos “SelectedObject” y “Deselected”, de este caso se obtuvo un error; el objeto se visualizaba de color verde cuando era observado, pero al dejar de observarlo no retomaba su color normal. Este error fue solucionado previamente de ser detectado. En el anexo 3, se pueden observar otros ejemplos de pruebas realizadas a algunos de los procedimientos.

Figura 12. Caso de prueba unitaria SelectedObject/Deselect.

La eliminación de los errores del código no garantiza que la solución cumple con las especificaciones del cliente, por ello, se realizaron pruebas de aceptación, descritas a continuación.

3.2.2 Pruebas de aceptación

Las pruebas de aceptación, también llamadas pruebas del cliente, son especificadas por el cliente y se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. Las pruebas de aceptación se derivan de las historias de los usuarios que se han implementado como parte de la liberación del software [16].

Tabla 28. Caso de prueba de aceptación P7HU7.

Caso de Prueba de Aceptación	
Código: P7HU7	**Número de HU:** 7
Nombre: Insertar y configurar mecánica de movimiento	
Descripción: Se debe probar que los scripts de movimiento al insertarse al personaje y configurarse, funcione correctamente.	
Condiciones de ejecución: Debe de existir un PJ en la escena	
Entrada/Pasos de ejecución: **1-** El usuario selecciona el PJ en la ventana de escena o de jerarquía. **2-** En la ventana de inspector le adiciona el script “FPC” o “TPC” y modifica los parámetros configurables o deja los que posee por defecto.	
Resultado esperado: Se obtiene un PJ controlable por el usuario.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 29. Caso de prueba de aceptación P11HU11.

Caso de Prueba de Aceptación	
Código: P11HU11	**Número de HU:** 11
Nombre: Acceder al inventario	
Descripción: Se debe probar que el PJ pueda acceder sin ningún problema al inventario y la correcta visualización de este.	
Condiciones de ejecución: El PJ debe contener el script “Inventario”	
Entrada/Pasos de ejecución: 1- Cumplida la condición anterior, el usuario presiona el botón “play” en la ventana de juego para ejecutarlo.	

Resultado esperado: Al presionar la tecla correspondiente, se visualiza el inventario de forma correcta y este presenta las dimensiones exactas y todas sus partes son observables.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 30. Caso de prueba de aceptación P14HU14.

Caso de Prueba de Aceptación	
Código: P14HU14	**Número de HU:** 14
Nombre: Brindar al usuario una base para las interacciones	
Descripción: Se debe probar que los scripts responsables de las interacciones funcionen correctamente.	
Condiciones de ejecución: **1-** Los GameObjects correspondientes, en este caso un cubo y el PJ deben de tener los scripts correspondientes asignados y configurados de forma correcta	
Entrada/Pasos de ejecución: **1-** El usuario configura los métodos necesarios para la interacción, dichos métodos permiten la eliminación del GameObject de la escena. **2-** Presiona el botón “Play” en la ventana del juego para ejecutarlo.	
Resultado esperado: El usuario acerca el PJ al objeto con el cual es posible interactuar, luego de visualizar que este se resalta de forma correcta, procede a presionar el botón de interacción y el objeto es eliminado de la escena.	
Evaluación de la prueba: Resultado satisfactorio.	

Al finalizar cada iteración se realizaron las respectivas pruebas a las funcionalidades, solucionándose las no conformidades detectadas. Algunos ejemplos de las no conformidades más importantes detectadas tienen relación con los siguientes errores: los ítems no se añadían al inventario, la cámara sobrepasaba el límite de rotación en el eje Y, no se podían abrir las puertas o estas no se resaltaban para poder interactuar con ellas. De otra manera, los no significativos se centraban en errores ortográficos al nombrar métodos, atributos o los scripts. A continuación, se muestra el acta de aceptación por parte del cliente.

3.2.3 Acta de aceptación

En cumplimiento de los objetivos del trabajo de diploma “Paquetes de mecánicas para videojuegos de tipos survival horror” y en función de la ejecución del demo que engloba todas las mecánicas se hace entrega de los productos que se relacionan a continuación:

- Paquete de mecánicas de videojuegos de tipo survival horror.
- Trabajo de diploma Paquetes de mecánicas para videojuegos de genero survival horror”.
- Demo con integración de todas las mecánicas (Cámara, IA, Interacción, Inventario, Movimiento, Interfaz).

La Parte Cliente, luego de haber revisado los productos de trabajo determina que se aceptan.

El resultado obtenido, les brindará a los desarrolladores de la línea de videojuegos, una herramienta que agilizará sus tiempos de desarrollo, pues representan una base fundamental en el desarrollo de videojuegos de tipo survival horror.

Entrega	**Recibe**
Nombre y apellidos:	**Nombre y apellidos:**
Jorge Miguel Trujillo Liz 	Ing. Enelis Blanca Cuba Rondón 
Cargo: diplomante	**Cargo: Jefa de línea de desarrollo de videojuegos. Centro de Tecnologías Interactivas**

Observador independiente

Nombre y Apellidos: Ing. Yoandy Paz Perdigón

Cargo: Arquitecto de la línea de desarrollo de videojuegos

Firma: Yoandy Paz Perdigón CN=Yoandy Paz Perdigón, SERIALNUMBER=E14803

Fecha: 05/07/2022

A continuación, la figura muestra por iteración la cantidad de no conformidades significativas y no significativas.

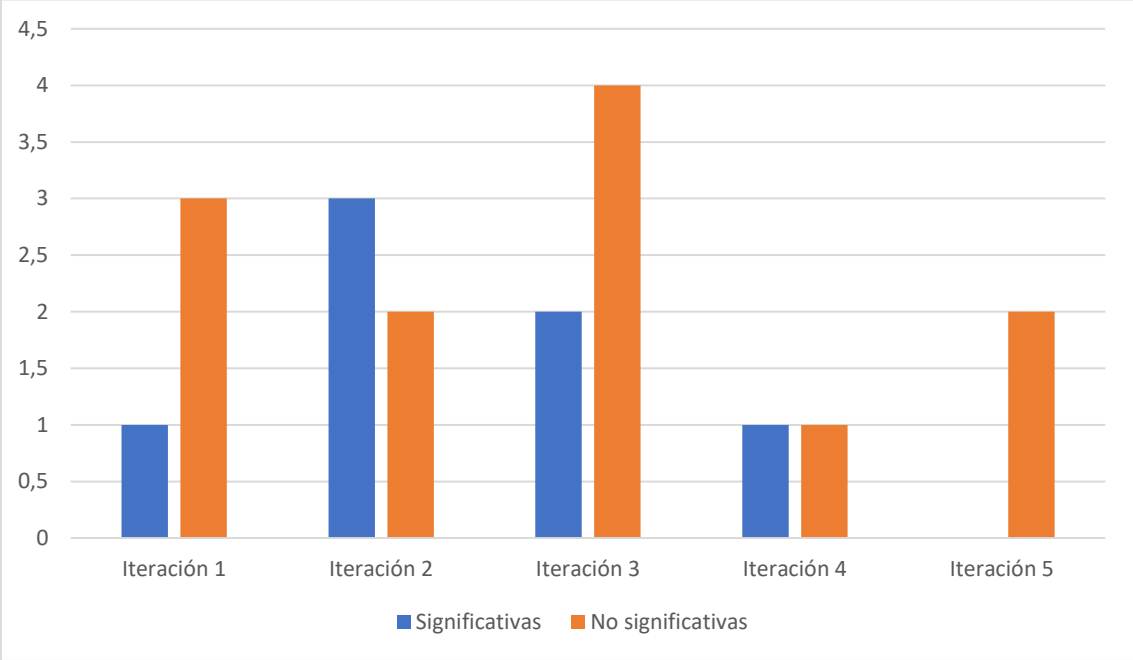


Figura 13. Resultados de las pruebas por iteraciones.

Al culminar la fase de prueba, se desarrolló un demo en Unity empleando el paquete creado, en aras de demostrar su flexibilidad, extensibilidad y que es reutilizable.

3.3 Análisis de resultado

El escenario representado en el demo es ambientado en una casa con varias habitaciones, donde se interactúa con los objetos que así lo permitan, el usuario cuenta con el PJ y este una linterna para los sitios oscuros. El PJ no tiene forma de atacar ya que este no está hecho para ello sino para huir. El objetivo del demo, es que el PJ se encuentre una forma de salir de la casa. Durante la travesía existe la posibilidad de encontrarse con una IA que al detectar al PJ lo perseguirá hasta eliminarlo. El escenario fue de creación propia utilizando la herramienta Blender y los assets utilizados pertenecen a varias páginas de contenido gratuito, por ejemplo, polihaven. En la siguiente imagen se muestra una captura de pantalla del demo.

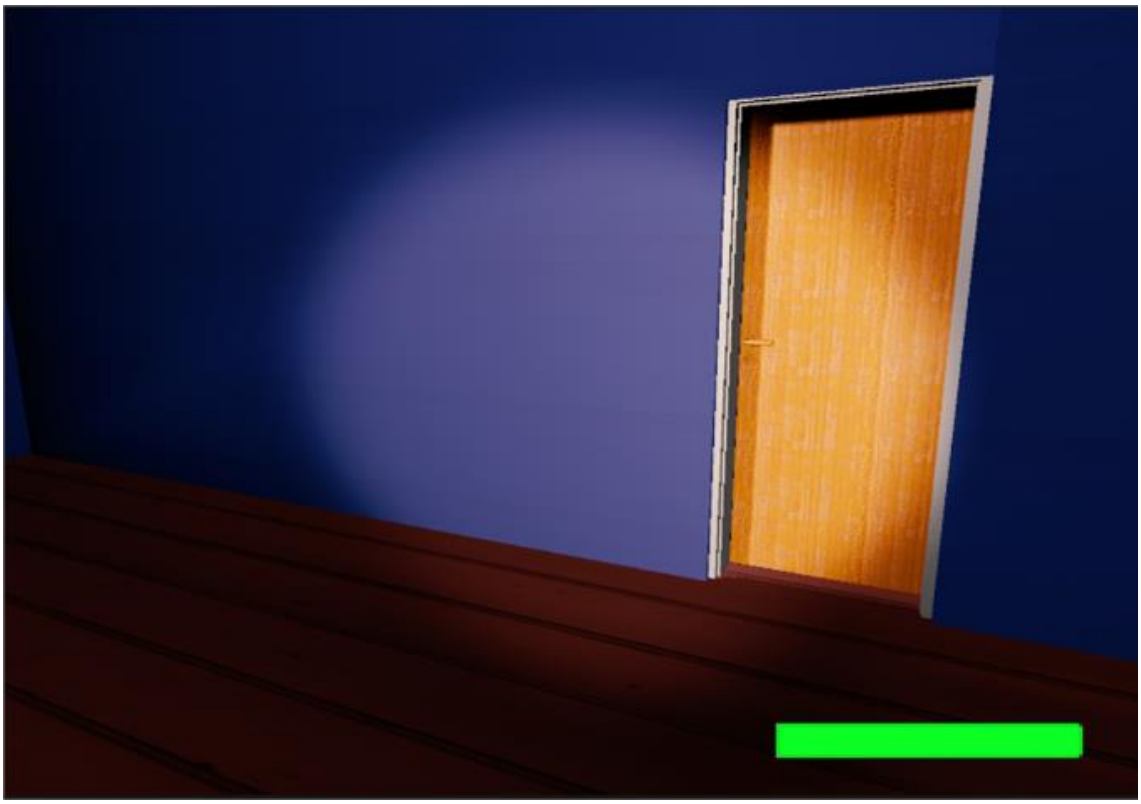


Figura 14. Captura de pantalla del demo realizado.

En la figura 14, se puede apreciar la vista desde la cámara en primera persona y la linterna que se le añadió al PJ, y en la parte inferior se aprecia la barra de vida. Para demostrar la flexibilidad, extensibilidad y reusabilidad. Se utilizan los conceptos aportados por Pressman [16]:

- Flexibilidad: Esfuerzo necesario para modificar un programa que ya opera.

- Extensibilidad: Un sistema es extensible cuando pueden incorporarse nuevas características al mismo sin mayor impacto sobre las características actuales.

- Reusabilidad: Grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones (se relaciona con el empaque y el alcance de las funciones que lleva a cabo el programa).

A continuación, se explica cómo cada uno de estos conceptos fue tratado en el demo. El primer paso fue importar el paquete en un nuevo proyecto, una vez importado, se llevó acabo la inserción de las mecánicas en la escena. Esta puede ser de varias maneras, por ejemplo, las mecánicas de la cámara, IA, movimiento e interfaz cuentan con prefabs que pueden ser añadidos directamente y modificados sus atributos en el inspector o

directamente en el script si así lo desea el usuario, en el demo fueron utilizados con sus configuraciones por defecto. Por otra parte, las mecánicas de interacción e inventario son *scripts*, y estos tuvieron que ser asignados a los GameObjects correspondientes. Con todo lo anterior se denota la reusabilidad.

Una vez el demo se encuentra funcional se ajustaron algunos valores, para obtener el resultado deseado. Se modificó el modelo de la IA, junto con la lógica de las interacciones, esto último usando como plantilla el script “ObjetoInteractivoBase” y directamente en el código. Este proceso anterior evidencia la flexibilidad frente a modificaciones.

La extensibilidad muestra en varios aspectos, por ejemplo, en el PJ, al cual se le adicionó una linterna la cual lleva un script nuevo, y en el escenario que fue añadido para crear un medio para interactuar; al igual se les adicionó a las puertas los scripts correspondientes para tener sonido. Todos estos añadidos, no modifican las funcionalidades ya existentes de las mecánicas, sino que añaden nuevo contenido. Al concluir el demo, se procedió a la realización de pruebas para analizar el rendimiento del paquete.

3.4 Pruebas de rendimiento

Debido a la naturaleza intrínseca de un videojuego, vinculada a las aplicaciones gráficas en tiempo real, resulta esencial contar con buenas herramientas que permitan depurar y optimizar el propio motor de juegos para obtener el mejor rendimiento posible. En este contexto, existe un gran número de herramientas de este tipo. Algunas de ellas son herramientas para propósito general que se pueden utilizar de manera externa al motor de juegos. Sin embargo, la práctica más habitual consiste en construir una herramienta de *profiling*, vinculada al análisis del rendimiento, o depuración que estén asociados al propio motor [30]. En el caso de Unity, no es necesario el uso de ninguna herramienta externa, por lo tanto, para las pruebas de rendimiento serán utilizadas las que presenta el motor de videojuegos Unity. A continuación, se nombran algunos de los principales procesos en el ciclo de actualización de un videojuego:

Renderizado: se encarga del dibujado de todos los objetos que se encuentren en el campo de visión de la cámara.

Scripts: manipular toda la lógica presente en el videojuego.

Físicas: Controla todas las físicas que presenta el videojuego.

Garbage Collector: Recolector de basura en español, se encarga de recuperar los espacios en memoria ocupados por objetos que no están en uso.

En el demo creado, se realizó un análisis del proceso Scripts, mediante la herramienta *profiler* de Unity, para valorar el rendimiento del mismo respecto al uso del CPU en milisegundos (ms). Dicho análisis fue realizado de cinco formas.

1- Prueba de rendimiento sin interacción del usuario y sin elementos gráficos.
2- Prueba de rendimiento sin interacción del usuario, con elementos gráficos y sin IA.
3- Prueba de rendimiento sin interacción del usuario, con elementos gráficos y con IA.
4- Prueba de rendimiento con interacción del usuario y sin elementos gráficos.
5- Prueba de rendimiento con interacción del usuario, con elementos gráficos y sin IA.
6- Prueba de rendimiento con interacción del usuario, con elementos gráficos y con IA.

Las pruebas anteriores fueron realizadas en una computadora con las siguientes prestaciones: procesador Intel Core i5-4210U a 1.70GHz, con 4 GB de ram y sistema operativo de 64 bits Windows 10. En el anexo 5 pueden observarse las figuras de las pruebas y a continuación, se muestran dos figuras con estos resultados para una mejor comprensión.

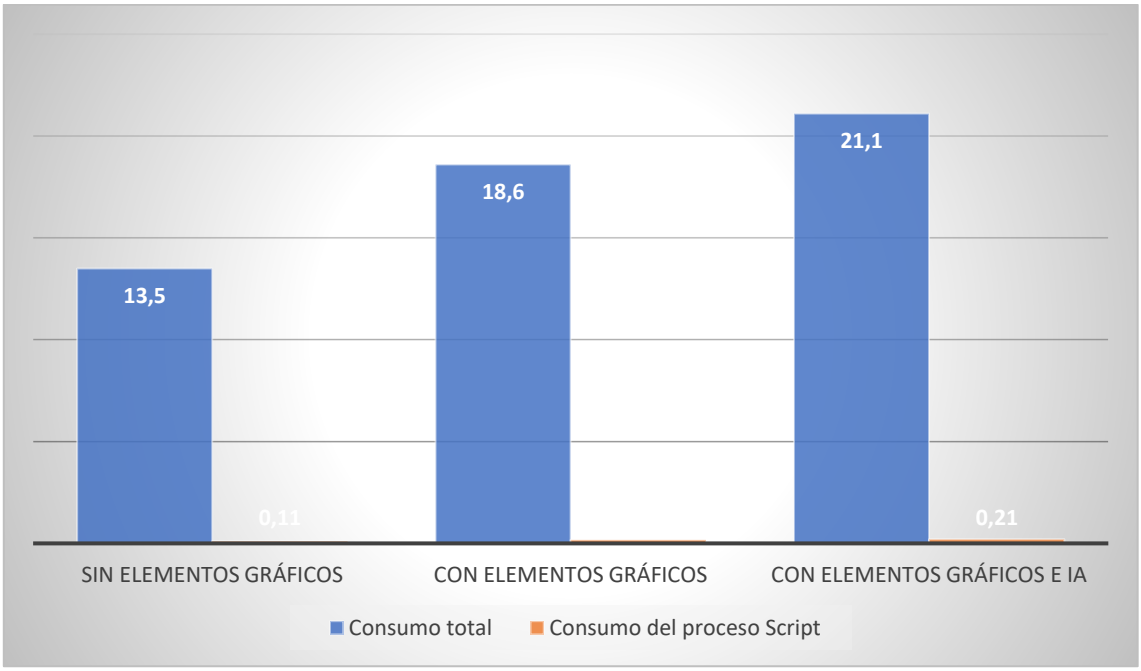


Figura 15. Resultados de las pruebas de rendimiento realizadas al demo sin la interacción del usuario.

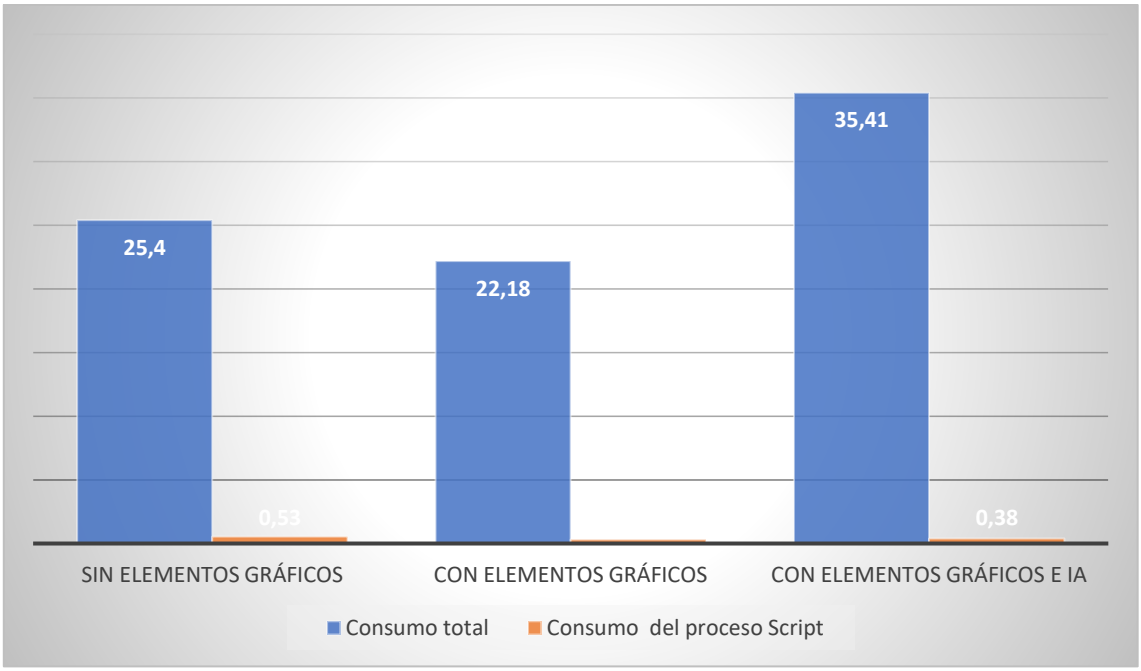


Figura 16. Resultados de las pruebas de rendimiento realizadas al demo con la interacción del usuario.

Teniendo en cuenta los resultados de las pruebas, el tiempo consumido por el proceso Script representa un porciento muy pequeño del tiempo que toma procesar un cuadro (fotograma), los tiempos de respuesta están en el intervalo de 13 (60 fps) ms a 35 ms (30 fps). De cualquier forma, es importante considerar el trabajo en la mayor optimización posible del paquete.

3.5 Conclusiones parciales

Al concluir con el presente capítulo, se puede arribar a las siguientes conclusiones:

1- La realización del plan de iteración y las tareas correspondientes, permitió la obtención del paquete de mecánicas para videojuegos tipo survival horror.
2- La definición y aplicación de los diferentes tipos de pruebas, permitió verificar el correcto funcionamiento del código.
3- La creación del demo, corroboró el cumplimiento de los requisitos determinados.

CONCLUSIONES

Luego de concluido el presente trabajo se llegó a las siguientes conclusiones:

- Existe una necesidad de crear componentes reutilizables que permitan la reducción de los tiempos de desarrollo y que estos sean flexibles.
- Mediante la investigación realizada y el estudio de tres videojuegos, las mecánicas identificadas para la solución que no pueden faltar en el desarrollo de los videojuegos survival horror son: cámara, interfaz, inventario, movimiento, interacción e IA.
- Se crearon los artefactos y se desarrollaron las tareas ingenieriles para darle cumplimiento a las fases de diseño e implementación.
- El presente paquete obtenido como solución, sirve como base para los desarrolladores en Unity a la hora de crear videojuegos.
- Las pruebas aplicadas para evaluar el producto, arrojaron que este tiene un correcto funcionamiento y ausencia de errores. Además de que no supone una carga para el rendimiento de los videojuegos donde sea utilizado.

RECOMENDACIONES

Para dar continuidad a la presente investigación se recomienda:

- Extender las mecánicas del paquete, añadiendo nuevas funcionalidades al sistema de inventario e interacciones.
- Continuar trabajando la optimización del paquete en base a la reducción del tiempo de respuesta en la ejecución de los scripts.
- Añadir un modo multiplayer.

REFERENCIAS BIBLIOGRÁFICAS

[1] HUIZINGA, J.: Homo Ludens. Madrid, Alianza Editorial, 1972.

[2] LALANDE, A.: Vocabulario técnico y crítico de la filosofía. París, Presses Universitares de France, 1951.

[3] Definición.De. [En línea] https://definicion.de/videojuego/

[4] Concepto de videojuego. [En línea] https://deconceptos.com/tecnologia/videojuego

[5] Videojuego. [En línea] https://www.ecured.cu/Videojuego

[6] Géneros que todos debemos conocer. [En línea] https://www.claro.com.co/institucional/tipos-de-videojuegos/

[7] ¿Qué es un juego de supervivencia? [En línea]https://nacionanime.com/que-es-un-juego-de-supervivencia-gaming-definition-meaning/

[8] Survival horror. [En línea] https://videojuegos.fandom.com/es/wiki/Survival_horror

[9] Mecánica de juego [En Línea] https://hmong.es/wiki/Game_mechanics

[10] Game mechanics : advanced game design [En línea] https://ebin.pub/game-mechanics-advanced-game-design-9780321820273-0321820274.html

[11] Silent Hill (videojuego) [En línea] https://silenthill.fandom.com/es/wiki/Silent_Hill_(videojuego)

[12] Las Etapas del Desarrollo de Juegos [En línea] https://starloopstudios.com/es/etapas-de-desarrollo-de-juegos-como-se-crean-los-videojuegos/

[13] Qué es Unity y para qué sirve [En línea] https://www.masterd.es/blog/que-es-unity-3d-tutorial

[14] Sistema de Navegación de Unity [En línea] https://docs.unity3d.com/es/2019.4/Manual/nav-NavigationSystem.html

[15] Metodología XP o Programación Extrema [En Línea] https://www.sinnaps.com/blog-gestion-proyectos/metodologia-xp

[16] PRESSMAN, R.S. 2010. Ingeniería del Software un Enfoque Prático. México. ISBN 978-607-15-0314-5.

[17] SOMMERVILLE, IAN. Ingeniería del software. Ed. por Miguel Martín-Romo. Madrid: PEARSON EDUCACIÓN, S.A, 2005.

[18] SOBRE EL CONCEPTO DE JUEGO [EN línea] https://gredos.usal.es/bitstream/handle/10366/69213/Sobre_el_concepto_de_juego.pdf?sequence=1&isAllowed=y

[19] Visual Studio Code [En línea] https://www.ecured.cu/Visual_Studio_Code

[20] A tour of the C# language [En línea] https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[21] Visual Paradigm [En línea] https://www.ecured.cu/Visual_Paradigm

[22] Visual Paradigm [En línea] https://www.oit.va.gov/Services/TRM/ToolPage.aspx?tid=10208

[23] Ciclo de vida de un proyecto XP [En línea] http://oness.sourceforge.net/proyecto/html/ch05s02.html

[24] Historias de usuario con ejemplos y plantilla [En línea] https://www.atlassian.com/es/agile/project-management/user-stories#:~:text=usuario%20del%20software.-,Una%20historia%20de%20usuario%20es%20una%20explicación%20general%20e%20informal,un%20valor%20particular%20al%20cliente.

[25] Definición de Plan de Entregas [En línea] https://www.cvosoft.com/glosario-sap/sap-sd/plan-de-entregas-2333.html#:~:text=Definición%20de%20Plan%20de%20Entregas,cantidades%20y%20fechas%20de%20entrega.

[26] Arquitectura basada en capas [En línea] https://geeks.ms/jkpelaez/2009/05/30/arquitectura-basada-en-capas/#:~:text=La%20arquitectura%20basada%20en%20capas,funcionalidad%20que%20está%20siendo%20desarrollada.

[27] Patrones de Asignación de Responsabilidades [En línea] https://www.ecured.cu/Patrones_de_Asignación_de_Responsabilidades

[28] Patrones GoF [En línea] https://www.ecured.cu/Patrones_Gof

[29] Tarjetas CRC [En línea] https://es-academic.com/dic.nsf/eswiki/1127486

[30] Creación de Videojuegos en Español: Arquitectura del Motor de Videojuegos [En línea] https://books.google.com.cu/books?id=ylnABAAAQBAJ&pg=PA23&dq=rendimiento+de+un+videojuego&hl=es-419&sa=X&ved=2ahUKEwi2kuba0sT4AhXwRDABHcTWDPQQuwV6BAgDEAk#v=onepage&q=rendimiento%20de%20un%20videojuego&f=false

[31] Historia de la tecnología: Spacewar!, el videojuego que nació en el MIT [En línea] https://hipertextual.com/2011/07/spacewar-el-videojuego-que-nacio-en-el-mit

GLOSARIO DE TÉRMINOS

NavMesh: Una malla de navegación, o navmesh, es una estructura de datos abstractos utilizada en aplicaciones de inteligencia artificial para ayudar a los agentes a encontrar caminos a través de espacios complicados.

Unity Profiling: El perfilador de unity es una herramienta que puede utilizar para obtener información sobre el rendimiento de su aplicación.

Rendering: El término renderización es un anglicismo para representación gráfica, usado en la jerga informática para referirse al proceso de generar imagen fotorrealista, o no, a partir de un modelo 2D o 3D por medio de programas informáticos.

Physics: Cuando se habla de física del videojuego estamos haciendo referencia al conjunto de reglas establecidas que determinan el comportamiento de los cuerpos dentro del videojuego.

CPU: El CPU o (Unidad central de procesamiento) es el cerebro de todo el funcionamiento del sistema, el encargado de dirigir todas las tareas que lleva a cabo el equipo y de ejecutar el código de los diferentes programas.

FPS: La tasa de fotogramas, expresada como fotogramas por segundo, es la frecuencia a la cual un dispositivo muestra imágenes llamadas fotogramas o cuadros. El término se aplica por igual a películas y cámaras de vídeo, gráficos por computadora y sistemas de captura de movimiento.

Prefab: Los prefabs en Unity son GameObjects preconfigurados y reutilizables que se crean en la escena y se almacenan en el proyecto.

ANEXO 1. HISTORIAS DE USUARIO

Tabla 31. HU Insertar y configurar mecánica de IA.

Historia de usuario	
No 2: Insertar y configurar mecánica de IA	
Prioridad: alta	**Nivel de complejidad:** bajo
Estimación: 5 días	**Iteración asignada:** 1
Descripción: El usuario tomará uno de los dos prefabricados de la mecánica IA y lo arrastrará a la escena, luego podrá configurar los parámetros en el inspector para tener los resultados esperados.	
Observaciones:	

Tabla 32. HU Definir objetivo, movimiento y objetos con los que interactúa la IA.

Historia de usuario	
No 3: Definir objetivo, movimiento y objetos con lo que interactúa la IA	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 5 días	**Iteración asignada:** 1
Descripción: El usuario podrá definir que PJ o PNJ debe atacar la IA, los lugares a los que puede desplazarse, saltar y evitar, además de definirle mediante un tag que GameObjects pueden ser abiertos y cuales no mediante el tag “Door”. Todo esto mediante scripts que pueden ser modificados en el inspector.	
Observaciones:	

Tabla 33. HU Insertar y configurar mecánicas de interacción.

Historia de usuario	
No 4: Insertar y configurar mecánicas de interacción	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 4 días	**Iteración asignada:** 1
Descripción: El usuario puede añadirles a los personajes un script que permita a las unidades interactuar con los GameObjects.	
Observaciones: Los GameObjects son todo aquello que se encuentre en la escena.	

Tabla 34. HU Visualizar indicador de objeto interactuable.

Historia de usuario	
No 6: Visualizar indicador de objeto interactuable	
Prioridad: media	**Nivel de complejidad:** medio
Estimación: 1 días	**Iteración asignada:** 1
Descripción: El PJ al acercarse a un objeto con el que es posible interactuar, dicho objeto a una distancia previamente definida debe cambiar de color y resaltarse en verde.	
Observaciones: Los objetos que pueden presentar este comportamiento deben ser definidos en HU 5.	

Tabla 35. HU Ejecutar el movimiento del personaje.

Historia de usuario	
No 8: Ejecutar el movimiento del personaje	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 7 días	**Iteración asignada:** 2
Descripción: El sistema debe permitir que el usuario mediante un botón haga que el PJ ejecute una forma de movimiento (caminar, correr, saltar).	
Observaciones: El movimiento se efectúa mediante teclas y que deben estar ya definidas en la HU 7.	

Tabla 36. HU Insertar y configurar script de salud.

Historia de usuario	
No 9: Insertar y configurar script de salud	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 3 días	**Iteración asignada:** 2
Descripción: El usuario tiene la posibilidad de añadir scripts a los PJ o PNJ que añadan salud y que por tanto posibiliten la muerte de estos.	
Observaciones:	

Tabla 37. HU Insertar y configurar mecánica de inventario.

Historia de usuario	
No 10: Insertar y configurar mecánica de inventario	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 10	**Iteración asignada:** 2
Descripción: El usuario tiene la posibilidad de añadir un prefabricado a la escena, que luego tendrá la opción de modificar.	
Observaciones:	

Tabla 38. HU Acceder al inventario.

Historia de usuario	
No 11: Acceder al inventario	
Prioridad: alta	**Nivel de complejidad:** media
Estimación: 2 días	**Iteración asignada:** 3
Descripción: Al presionar un botón se debe de poder acceder al inventario.	
Observaciones: El botón debe estar previamente definido en HU 10.	

Tabla 39. HU Interactuar con el inventario.

Historia de usuario	
No 12: Interactuar con el inventario	
Prioridad: alta	**Nivel de complejidad:** bajo
Estimación: 10 días	**Iteración asignada:** 3
Descripción: El usuario podrá interactuar con el inventario y realizar las acciones disponibles.	
Observaciones:	

Tabla 40. HU Configurar bloqueo y desbloqueo de puertas.

Historia de usuario	
No 13: Configurar bloqueo y desbloqueo de puertas	
Prioridad: medio	**Nivel de complejidad:** alto
Estimación: 6 días	**Iteración asignada:** 4

Descripción: El usuario tiene la posibilidad de añadir scripts para el bloqueo de puertas y su desbloqueo mediante objetos específicos.
Observaciones: Debe estar previamente definido el HU 10 para la obtención de los objetos.

Tabla 41. HU Brindar al usuario una base para las interacciones.

Historia de usuario	
No 14: Brindar al usuario una base para las interacciones	
Prioridad: media	**Nivel de complejidad:** alta
Estimación: 4	**Iteración asignada:** 4
Descripción: El usuario deberá contar con un script que sirva de plantilla a la hora de definir nuevas interacciones para los GameObjects que crea necesarios.	
Observaciones:	

Tabla 42. HU Configurar Interfaz.

Historia de usuario	
No 15: Configurar Interfaz	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 3	**Iteración asignada:** 4
Descripción: El usuario puede configurar la interfaz con un único elemento descrito en el HU 9 y que permita la visualización de este, además de mostrarse los elementos descritos en los HU 6 y HU 11. Aunque el usuario puede optar por añadir los elementos que desee.	
Observaciones:	

Tabla 43. HU Definir las regiones de movimiento del agente.

Historia de usuario	
No 16: Definir las regiones de movimiento del agente	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 7	**Iteración asignada:** 5
Descripción: El usuario podrá configurar mediante el inspector configurar todas las regiones por las cuales puede desplazarse el agente.	

Observaciones: Debe estar previamente definido la HU 2.

ANEXO 2. TAREAS DE INGENIERIA

Tabla 44. Tarea 2 Iteración 1.

Tarea	
Número de tarea: 1.2	**Número de HU:** 2
Nombre de la tarea: Implementación de la clase IA	
Tipo de tarea: Implementación	**Estimación:** 5
Descripción: Se implementa la lógica de los dos tipos de IA, junto a todos los atributos configurables. Y finalmente se crean ambos prefabs.	

Tabla 45. Tarea 3 Iteración 1.

Tarea	
Número de tarea: 1.3	**Número de HU:** 3
Nombre de la tarea: Definición de los objetivos, movimientos y objetos con lo que interactúa la IA.	
Tipo de tarea: Configuración	**Estimación:** 5
Descripción: Se definen los objetivos, los puntos hacia donde se podrá desplazar y qué objetos pueden ser accionados por la IA.	

Tabla 46. Tarea 4 Iteración 1.

Tarea	
Número de tarea: 1.4	**Número de HU:** 4
Nombre de la tarea: Implementación de las mecánicas de interacción	
Tipo de tarea: Implementación	**Estimación:** 4
Descripción: Se implementan los mecanismos de la clase Selected, la cual se asignará al PJ y permitirá la interacción con los objetos mediante una tecla determinada.	

Tabla 47. Tarea 5 Iteración 1.

Tarea	
Número de tarea: 1.5	**Número de HU:** 5
Nombre de la tarea: Implementación del script ObjetoInteractivo	

Tipo de tarea: Implementación	**Estimación:** 4
Descripción: Se declaran los atributos configurables y los implementan los diferentes tipos de acciones o funcionalidades según se desee.	

Tabla 48. Tarea 6 Iteración 1.

Tarea	
Número de tarea: 1.6	**Número de HU:** 5
Nombre de la tarea: Implementación del script SistemaPuertas	
Tipo de tarea: Implementación	**Estimación:** 1
Descripción: Se declaran los atributos configurables que podrán ser modificados en el inspector de Unity y se implementa la funcionalidad correspondiente.	

Tabla 49. Tarea 2 Iteración 2.

Tarea	
Número de tarea: 2.2	**Número de HU:** 7
Nombre de la tarea: Implementación de la clase TPC	
Tipo de tarea: Implementación	**Estimación:** 2
Descripción: Se declaran los atributos configurables necesarios para el correcto funcionamiento y modificación del controlador de tercera persona (TPC).	

Tabla 50. Tarea 4 Iteración 2.

Tarea	
Número de tarea: 2.4	**Número de HU:** 9
Nombre de la tarea: Implementación de la clase VidaPlayer	
Tipo de tarea: Implementación	**Estimación:** 3
Descripción: Se implementa la lógica de la salud en el script VidaPlayer y sus atributos que pueden ser configurados mediante el inspector.	

Tabla 51. Tarea 5 Iteración 2.

Tarea	
Número de tarea: 2.5	**Número de HU:** 10

Nombre de la tarea: Implementación de los atributos configurables de la clase Inventario	
Tipo de tarea: Implementación	**Estimación:** 2
Descripción: Se declaran los atributos configurables para la contención de objetos, cantidad de objetos a contener y todos los necesarios para el correcto funcionamiento de la mecánica.	

Tabla 52. Tarea 6 Iteración 2.

Tarea	
Número de tarea: 2.6	**Número de HU:** 10
Nombre de la tarea: Implementación de la lógica de la clase Inventario	
Tipo de tarea: Implementación	**Estimación:** 8
Descripción: Son implementados todos los mecanismos encargados del funcionamiento del inventario, como: la recogida y el añadido de los ítems.	

Tabla 53. Tarea 2 Iteración 3.

Tarea	
Número de tarea: 3.2	**Número de HU:** 12
Nombre de la tarea: Implementación de los atributos configurables de la clase item	
Tipo de tarea: Implementación	**Estimación:** 1
Descripción: Se declaran los atributos configurables para definir el identificador, el tipo, la descripción y el icono que se puede visualizar en el inspector y que serán comunes para el mismo tipo de item	

Tabla 54. Tarea 4 Iteración 3.

Tarea	
Número de tarea: 3.4	**Número de HU:** 12
Nombre de la tarea: Implementación de los atributos configurables de la clase slot	
Tipo de tarea: Implementación	**Estimación:** 1
Descripción: Se declaran los atributos configurables de los slots y que serán configurados según la clase “ítem”.	

Tabla 55. Tarea 5 Iteración 3.

Tarea	
Número de tarea: 3.5	**Número de HU:** 12
Nombre de la tarea: Implementación de la lógica de la clase slot	
Tipo de tarea: Implementación	**Estimación:** 4
Descripción: Se implementan los mecanismos principales de la clase “slots”	

Tabla 56. Tarea 1 Iteración 4.

Tarea	
Número de tarea: 4.1	**Número de HU:** 13
Nombre de la tarea: Implementación del bloqueo y desbloqueo de puertas	
Tipo de tarea: Implementación	**Estimación:** 8
Descripción: Se implementa la lógica y los atributos configurables de la clase SistemaPuertas correspondientes al bloqueo y desbloqueo de puertas mediante ítems.	

Tabla 57. Tarea 3 Iteración 4.

Tarea	
Número de tarea: 4.3	**Número de HU:** 14
Nombre de la tarea: Implementación de un script base para acceder a las nuevas interacciones	
Tipo de tarea: Implementación	**Estimación:** 2
Descripción: Se implementa un script llamado “SelectedBase” el cual el usuario podrá usar como base para implementar la lógica de los métodos que llamarán las nuevas interacciones en el script “ObjetoInteractivoBase”, este contiene dentro comentarios para el entendimiento de la lógica.	

Tabla 58. Tarea 4 Iteración 4.

Tarea	
Número de tarea: 4.4	**Número de HU:** 15
Nombre de la tarea: Diseño de la Interfaz de juego	
Tipo de tarea: Diseño	**Estimación:** 3

Descripción: Se diseña y crea el prefabricado de la interfaz del juego que pueda ser adicionado a la escena y ser modificado si así lo desea el usuario. Este cuenta con todas las partes necesarias para el correcto funcionamiento de sus partes

Tabla 59. Tarea 2 Iteración 5.

Tarea	
Número de tarea: 5.2	**Número de HU:** 16
Nombre de la tarea: Definición de la malla de navegación	
Tipo de tarea: Configuración	**Estimación:** 4
Descripción: Se definen todas las regiones caminables, no caminables y aquellas que pueden ser saltadas, entre otros tipos que desee fijar el usuario.	

ANEXO 3. CASOS DE PRUEBA UNITARIA

Figura 17. Caso de prueba unitaria TestAddItem.

Figura 18. Caso de prueba unitaria TestDaño.

Figura 19. Caso de prueba unitaria ItemUsage.

ANEXO 4. PRUEBAS DE ACEPTACIÓN

Tabla 60. Caso de prueba de aceptación P1HU1.

Caso de Prueba de Aceptación	
Código: P1HU1	**Número de HU:** 1
Nombre: Insertar y configurar cámara en tercera persona a la escena	
Descripción: Se debe probar que la mecánica de cámara al añadirse a la escena funcione correctamente.	
Condiciones de ejecución: **1.** No debe existir otra cámara aparte de la contenida en el prefabricado. **2.** Debe estar añadido el tag “MainCamera” en la cámara del prefabricado	
Entrada/Pasos de ejecución: **1.** El usuario selecciona el prefabricado “CamaraTP” en la ventana de proyecto y lo arrastra hacia la ventana de escena o de jerarquía. **2.** En el inspector modifica los parámetros configurables de la cámara o deja los que posee por defecto. **3.** Presiona el botón “play” en la ventana de juego para ejecutarlo.	
Resultado esperado: La cámara se controla con el mouse, esta rota alrededor del player y se acerca a este si hay un obstáculo de por medio que no permita su visualización	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 61. Caso de prueba de aceptación P2HU2.

Caso de Prueba de Aceptación	
Código: P2HU2	**Número de HU:** 2
Nombre: Insertar y configurar mecánica de IA	
Descripción: Se debe probar que la mecánica de IA al añadirse a la escena funcione correctamente.	
Condiciones de ejecución: **1.** Solo debe existir un único PJ en la escena. **2.** Debe de existir el tag IA antes de que el prefab sea añadido.	
Entrada/Pasos de ejecución:	

1. El usuario selecciona uno o ambos prefabricados “IAReconGlobal” o “IAReconLocal” en la ventana de proyecto y lo arrastra hacia la ventana de escena o de jerarquía. 2. En la ventana del inspector, se le debe de poner el mismo valor a la variable cantPuntosNavegacion que el valor puesto en el array "PuntosNavegacion" y modificar los restantes parámetros configurables o dejar los que posee por defecto.
Resultado esperado: Correcta creación y configuración de los elementos necesarios para la IA.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 62. Caso de prueba de aceptación P3HU3.

Caso de Prueba de Aceptación	
Código: P3HU3	**Número de HU:** HU3
Nombre: Definir los objetivos, movimiento y objetos con lo que interactúa la IA	
Descripción: Se debe probar que funcionen correctamente los objetivos, movimientos e interacciones de la IA.	
Condiciones de ejecución: Debe existir en la escena el objeto IA.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el objeto IA en la ventana de jerarquía. 2. Luego en la ventana de inspector especifica los puntos donde la IA se debe de mover cuando no se encuentra persiguiendo al PJ, al igual que se define cuál es su objetivo a perseguir y con qué objeto en específico puede interactuar. 3. Presiona el botón “play” en la ventana de juego para ejecutarlo.	
Resultado esperado: Si se le acerca el PJ comenzará a perseguirlo y si no está cerca comenzará a deambular por los puntos definidos, que al encontrarse con una puerta que le bloquee el paso, la IA procederá a abrirla.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 63. Caso de prueba de aceptación P4HU4.

Caso de Prueba de Aceptación

Código: P4HU4	**Número de HU:** 4
Nombre: Insertar y configurar la mecánica de interacción	
Descripción: Se debe probar que la mecánica interacción al añadirse a la escena funcione correctamente.	
Condiciones de ejecución: **1.** Debe existir previamente el tag "RaycastDetect" y aquellos tags definidos por el usuario que permiten llamar a las diferentes interacciones. **2.** Debe de haber una cámara en la escena.	
Entrada/Pasos de ejecución: **1.** El usuario selecciona el script “Selected” en la ventana de proyecto y lo añade a la cámara. **2.** En el inspector modifica los parámetros configurables.	
Resultado esperado: Correcta creación y configuración de los elementos necesarios para la mecánica de interacción.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 64. Caso de prueba de aceptación P5HU5.

Caso de Prueba de Aceptación	
Código: P5HU5	**Número de HU:** 5
Nombre: Definición y configuración de los objetos con los que se puede interactuar	
Descripción: Se debe probar que los objetos interactuables funcionan correctamente y se puede interactuar con estos.	
Condiciones de ejecución: Debe de existir en la escena el objeto PJ con el script Selected.	
Entrada/Pasos de ejecución: **1.** El usuario selecciona el script “ObjetoInteractivo” en la ventana de proyecto y lo añade al *Game*Object deseado. **2.** Modifica el script para añadirle el comportamiento deseado y define un tag que servirá para llamar al método creado anteriormente. **3.** Presiona el botón “play” en la ventana de juego para ejecutarlo.	
Resultado esperado: Al presionar el botón definido para la interacción, el objeto realiza correctamente el comportamiento programado.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 65. Caso de prueba de aceptación P6HU6.

Caso de Prueba de Aceptación	
Código: P6HU6	**Número de HU:** 6
Nombre: Visualización del indicador de objeto interactuable	
Descripción: Se debe probar que el indicador de los objetos interactuables funcionan correctamente.	
Condiciones de ejecución: Debe existir un objeto PJ con el script “Selected” y al menos un objeto que tenga el script “ObjetoInteractivo”	
Entrada/Pasos de ejecución: **1.** El usuario configura el parámetro distancia en la ventana del inspector a la deseada. **2.** Presiona el botón “play” en la ventana de juego para ejecutarlo.	
Resultado esperado: Cuando el PJ se acerca a la distancia elegida, el objeto con el cual se puede interactuar, tendrá un color verde y si se aleja el PJ el color volverá al que tenía previamente.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 66. Caso de prueba de aceptación P8HU8.

Caso de Prueba de Aceptación	
Código: P8HU8	**Número de HU:** 8
Nombre: Ejecutar el movimiento del personaje	
Descripción: Se debe probar que el PJ luego de presionar los botones asignados ejecute el movimiento	
Condiciones de ejecución: El PJ debe contener el script “FPC” o “TPC”	
Entrada/Pasos de ejecución: **1-** El usuario presiona el botón “play” en la ventana del juego para ejecutarlo.	
Resultado esperado: Al presionar la tecla “W”, el PJ debe de moverse hacia delante, si se presiona la tecla “Espacio” este debe de saltar y si se presionan las teclas “shift” y “W” conjuntamente, debe de correr.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 67. Caso de prueba de aceptación P9HU9.

Caso de Prueba de Aceptación	
Código: P9HU9	**Número de HU:** 9
Nombre: Insertar y configurar script de salud	
Descripción: Se debe probar que el script de salud al insertarse al PJ y configurarse, funcione correctamente.	
Condiciones de ejecución: Debe de existir un PJ en la escena.	
Entrada/Pasos de ejecución: **1-** El usuario selecciona el PJ y en el inspector adiciona el script VidaPlayer**.** **2-** Modifica los parámetros configurables o se mantiene por defecto. **3-** Presiona el botón “Play” en la ventana del juego para ejecutarlo.	
Resultado esperado: Una vez el PJ tenga el script cargado y al acercarse a un enemigo, la vida decrecerá y una vez no se esté recibiendo daño la vida se regenere de forma automática.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 68. Caso de prueba de aceptación P10HU10.

Caso de Prueba de Aceptación	
Código: P10HU10	**Número de HU:** 10
Nombre: Insertar y configurar mecánica de inventario	
Descripción: Se debe probar que el script “Inventario” al añadirse al PJ no presente errores.	
Condiciones de ejecución: Al insertar el script “Inventario” al PJ y configurarse, funcione correctamente.	
Entrada/Pasos de ejecución: **1-** El usuario selecciona el PJ y en el inspector adiciona el script “Inventario”**.** **2-** Se modifican los parámetros configurables o se mantienen por defecto.	
Resultado esperado: Correcta creación y configuración de los elementos necesarios para el inventario.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 69. Caso de prueba de aceptación P12HU12.

Caso de Prueba de Aceptación	
Código: P12HU12	**Número de HU:** 12
Nombre: Interactuar con el inventario	
Descripción: Se debe probar que los scripts “Item” y “Slot” funcionen correctamente.	
Condiciones de ejecución: **1-** Debe existir en la escena un PJ con el script “Inventario”, además de un objeto que sirva como ítem.	
Entrada/Pasos de ejecución: **1-** El usuario selecciona el objeto y en el inspector le añade el script “Item”. **2-** Modifica los parámetros configurables de forma correcta. **3-** Presiona el botón “Play” en la ventana de juego para ejecutarlo.	
Resultado esperado: El usuario mueve al PJ hasta tocar el objeto y recogerlo, y en el inventario al tocar dicho objeto, este aparece en la mano del PJ para ser usado.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 70. Caso de prueba de aceptación P13HU13.

Caso de Prueba de Aceptación	
Código: P13HU13	**Número de HU:** 13
Nombre: Configurar bloqueo y desbloqueo de puertas	
Descripción: Se debe probar que el script “SistemaPuerta” permita la correcta funcionalidad de bloqueo y desbloqueo.	
Condiciones de ejecución: **1-** Previamente debe de existir un PJ con el objeto en el inventario que permita el desbloqueo de las puertas **2-** Debe existir una puerta previamente bloqueada y contener el script “SistemaPuerta” y el tag “BlockDoor”	
Entrada/Pasos de ejecución: **1-** Cumplidas las condiciones anteriores, el usuario presiona el botón “play” en la ventana de juego para ejecutarlo.	
Resultado esperado: El usuario, accede al inventario y toma la llave en su mano, la cual al acercarse a la puerta bloqueada puede abrir presionando el botón de acción.	

Evaluación de la prueba: Resultado satisfactorio.

Tabla 71. Caso de prueba de aceptación P15HU15.

Caso de Prueba de Aceptación	
Código: P15HU15	**Número de HU:** 15
Nombre: Configurar Interfaz	
Descripción: Se prueba si puede insertarse la interfaz del juego en la pantalla que esta se visualice correctamente.	
Condiciones de ejecución: No debe existir otra interfaz en la escena.	
Entrada/Pasos de ejecución: **1-** El usuario selecciona el prefabricado “Canvas” en la ventana de proyecto, y lo arrastra hacia la ventana de escena o de jerarquía. **2-** En la escena, posiciona las partes de la interfaz según prefiera o las deja por defecto.	
Resultado esperado: Se obtiene la interfaz del juego y esta se puede apreciar de forma correcta.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 72. Caso de prueba de aceptación P16HU16.

Caso de Prueba de Aceptación	
Código: P16HU16	**Número de HU:** 16
Nombre: Definir las regiones de movimiento del agente	
Descripción: Se debe probar que el agente puede moverse de forma correcta por las zonas de la escena.	
Condiciones de ejecución: Debe existir un escenario y un agente (IA) funcional.	
Entrada/Pasos de ejecución: **1-** Cumplidas las condiciones anteriores, el usuario presiona el botón “play” en la ventana de juego para ejecutarlo.	
Resultado esperado: El agente es capaz de desplazarse por el escenario, evitando obstáculos, llegando a su destino por el camino más óptimo	
Evaluación de la prueba: Resultado satisfactorio.	

ANEXO 5. PRUEBAS DE RENDIMIENTO CON LA HERRAMIENTA PROFILER DE UNITY

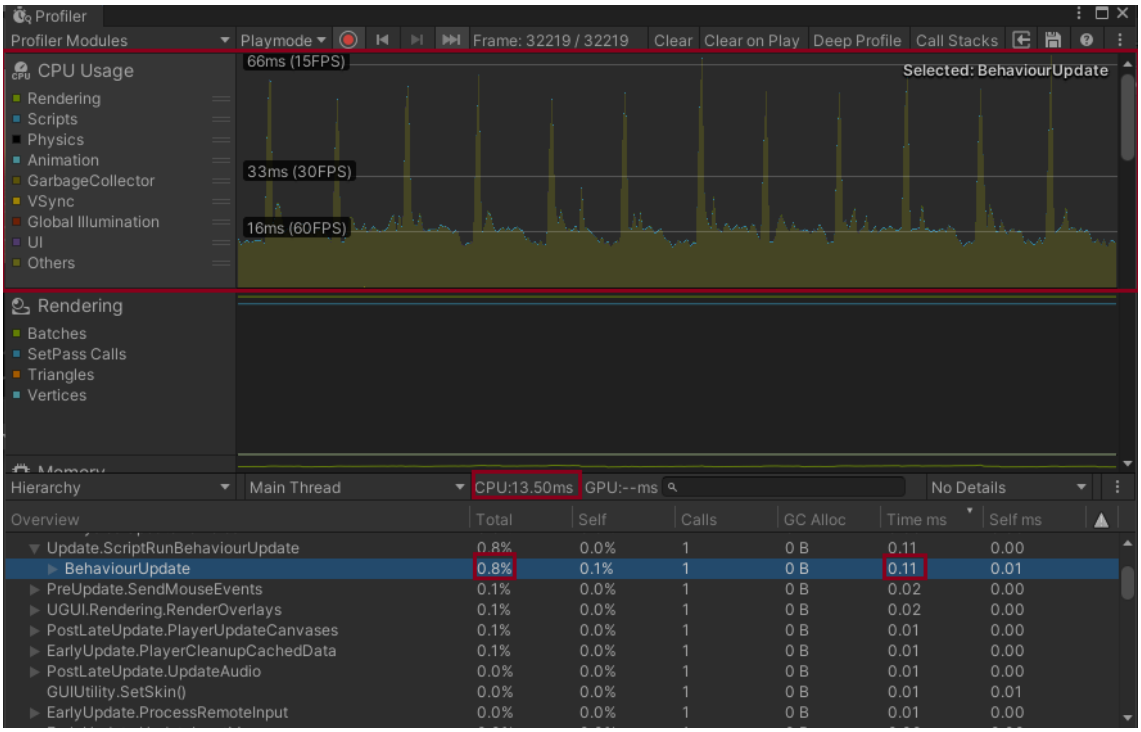


Figura 20. Prueba de rendimiento sin interacción del usuario y sin elementos gráficos.

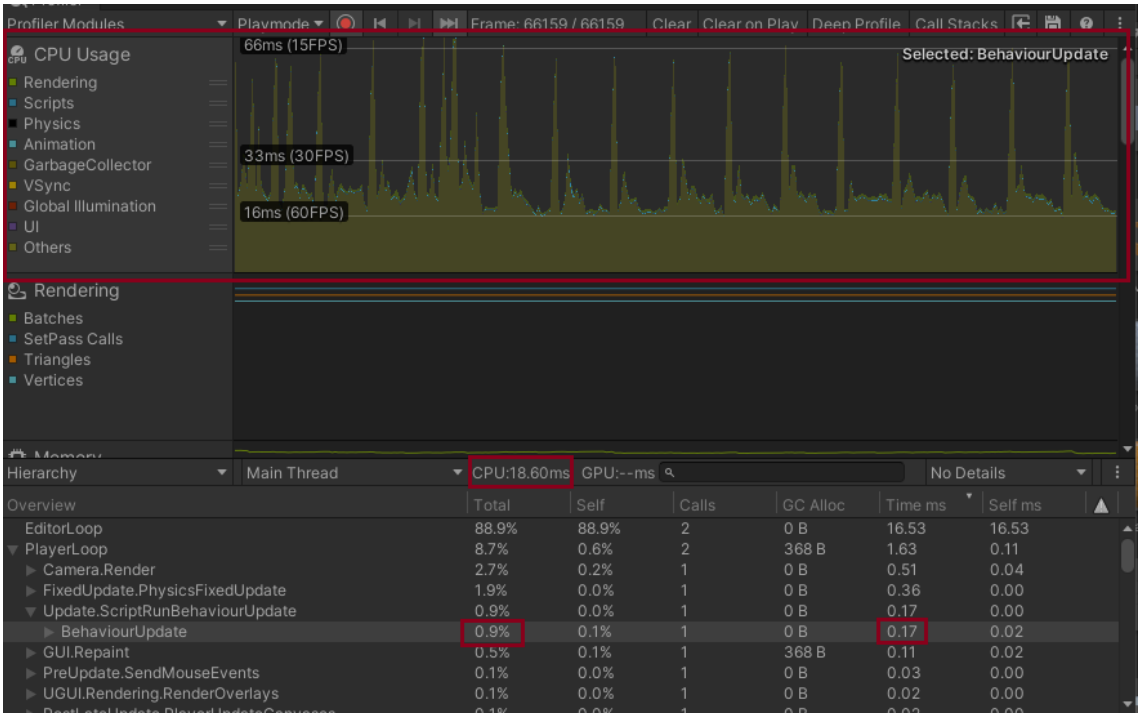


Figura 21. Prueba de rendimiento sin interacción del usuario, con elementos gráficos y sin IA.

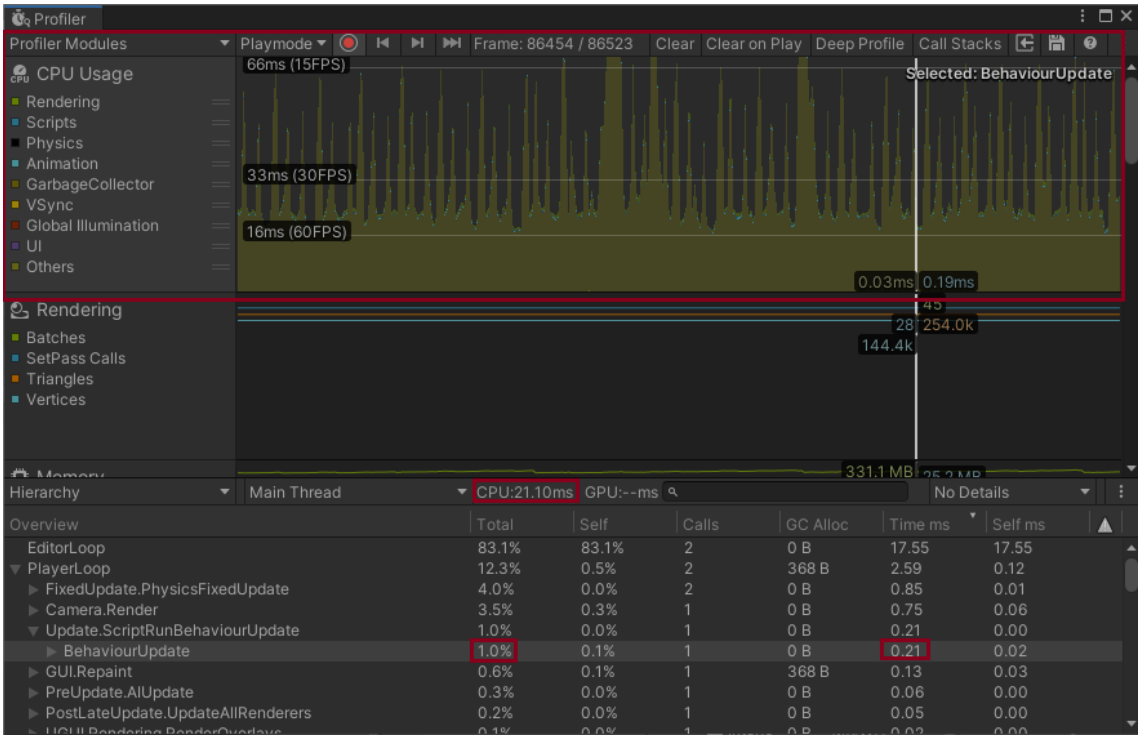


Figura 22. Prueba de rendimiento sin interacción del usuario, con elementos gráficos y con IA.

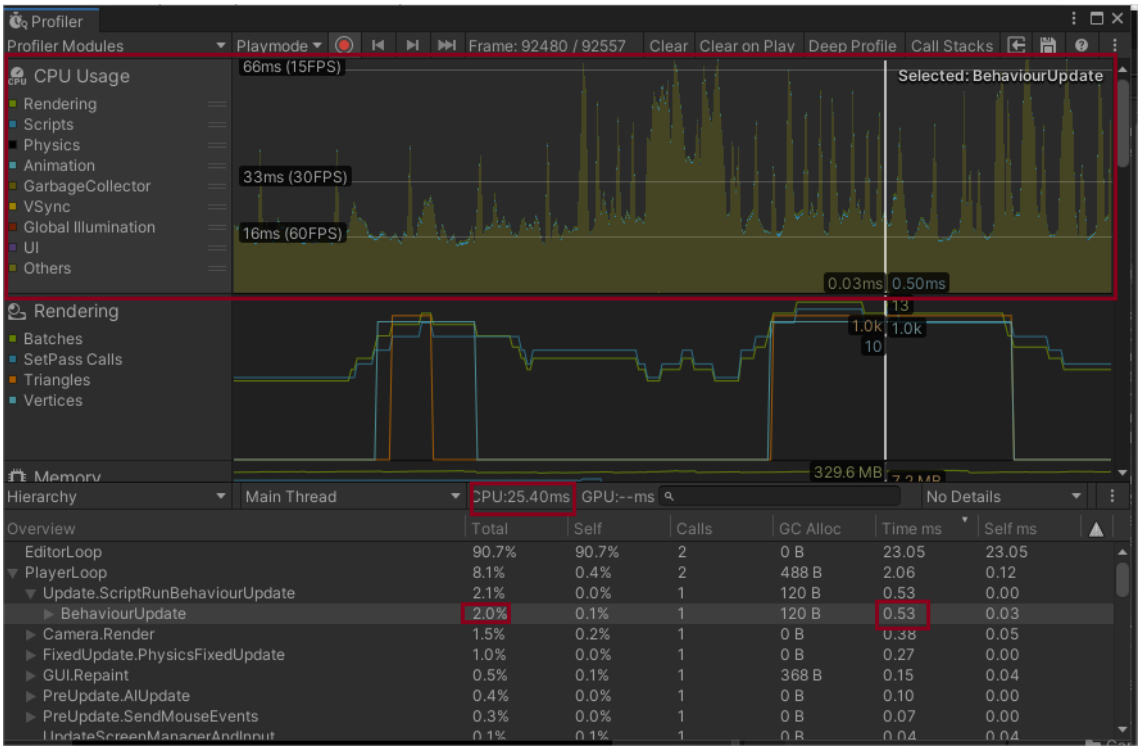


Figura 23. Prueba de rendimiento con interacción del usuario y sin elementos gráficos.

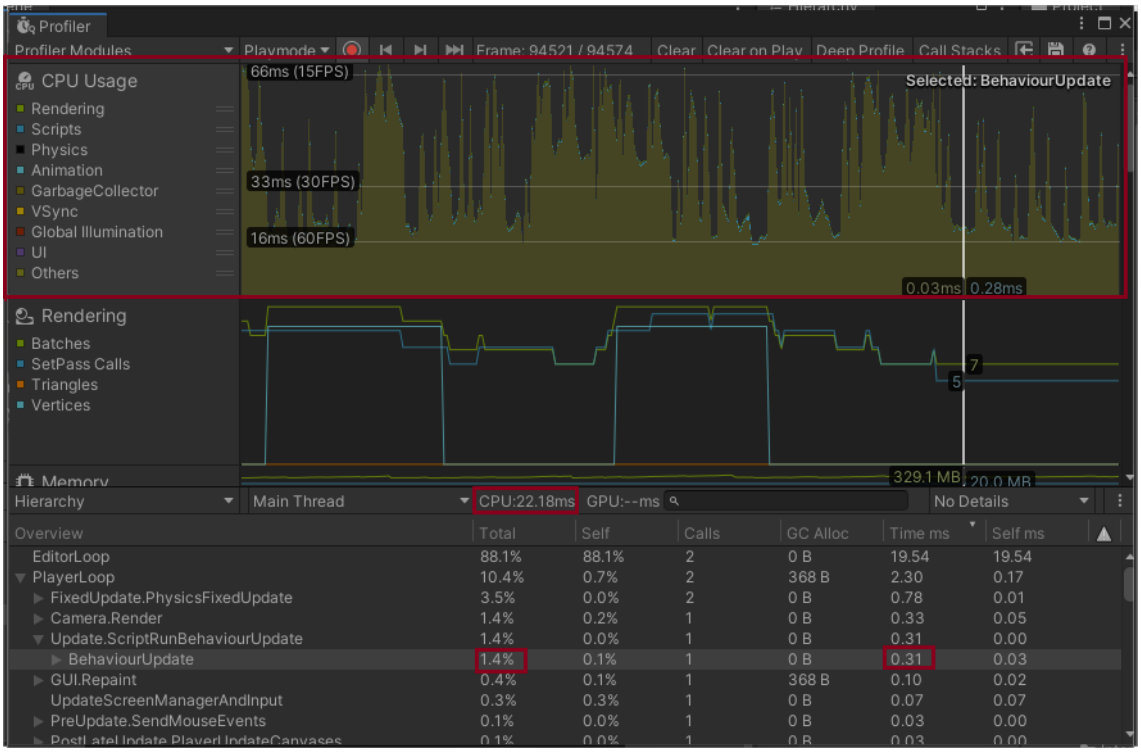


Figura 24. Prueba de rendimiento con interacción del usuario, con elementos gráficos y sin IA.

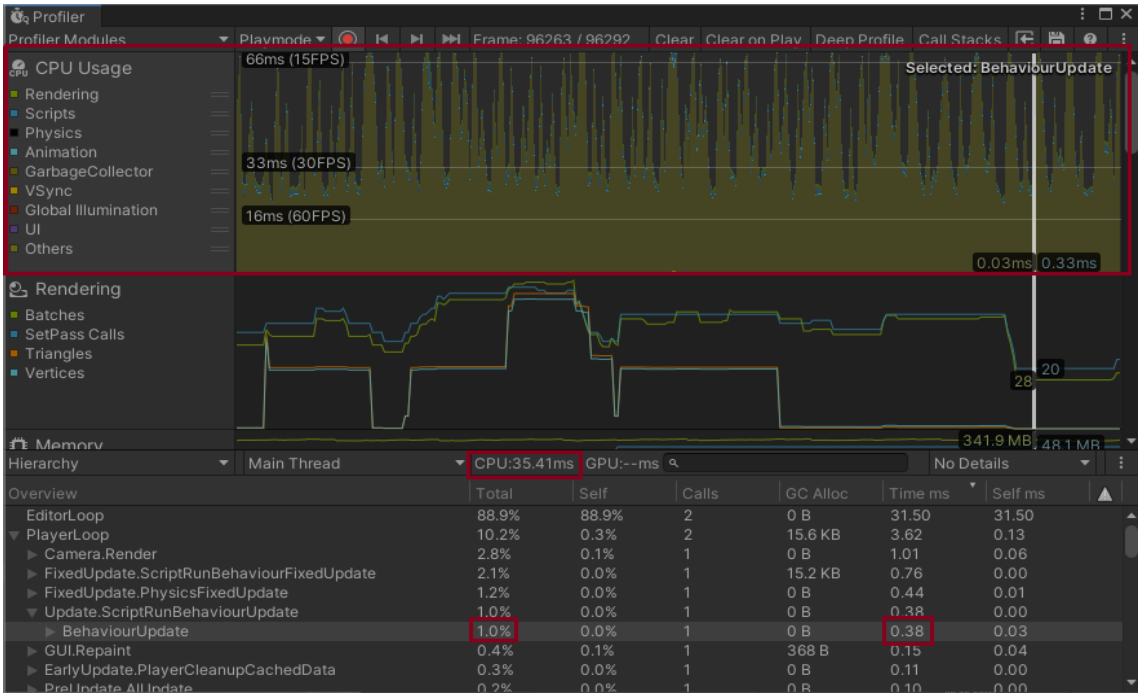


Figura 25. Prueba de rendimiento con interacción del usuario, con elementos gráficos y con IA.