UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE MECÁNICAS DE SIGILO Y ESTRATEGIA PARA UN VIDEOJUEGO EN TERCERA PERSONA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

DIEGO AGUSTÍN CAVIEDES AGUIRRE

PROFESOR GUÍA:
Daniel Calderón Saavedra

MIEMBROS DE LA COMISIÓN:
Eduardo Graells Garrido
Elías Zelada Baeza

SANTIAGO DE CHILE
2024

DISEÑO E IMPLEMENTACIÓN DE MECÁNICAS DE SIGILO Y ESTRATEGIA PARA UN VIDEOJUEGO EN TERCERA PERSONA

La industria de los videojuegos ha experimentado un crecimiento exponencial en los últimos años, impulsado principalmente por el desarrollo de nuevas tecnologías. Este avance tecnológico ha llevado a una reducción significativa de las limitaciones que enfrentan los desarrolladores, permitiéndoles crear mundos virtuales completamente inmersivos. Estos mundos se enfrentan al gran desafío de ofrecer experiencias novedosas a un público cada vez más exigente.

En este trabajo de título, se aborda el diseño e implementación de las mecánicas de juego para *Zombattan*, un videojuego en desarrollo que busca destacarse en el género de sigilo y estrategia, ambientado en un mundo post-apocalíptico infestado de zombis. El proceso se llevó a cabo en el motor de videojuegos *Unreal Engine* versión 5.2, centrándose en la creación de mecánicas de juego para una experiencia en tercera persona.

En la fase inicial del desarrollo del videojuego, se optó por clasificar las mecánicas de juego propuestas en tres categorías principales: Mecánicas de Movimiento, Mecánicas de Sigilo y Mecánicas de Combate. En la categoría de Movimiento, se diseñaron mecánicas para definir cómo el personaje se desplaza a través del mapa, incluyendo la capacidad de agacharse y acostarse, para moverse sin generar ruido, y realizar acciones específicas como desplazar tablas para sortear obstáculos o mover cajas para abrir nuevos pasos. En cuanto al Sigilo, se idearon mecánicas destinadas a evadir, para pasar desapercibido y lanzar objetos para distraer a los enemigos mediante la generación de ruido. En el ámbito del Combate, se implementó tanto combate cuerpo a cuerpo mediante golpes con armas como enfrentamientos a distancia con armas de fuego. Esta subdivisión permitió abordar de manera integral los diversos aspectos esperados en la versión final del juego, sentando así las bases sólidas para ofrecer una experiencia completa a los jugadores.

Como parte de esta etapa del desarrollo, también se creó un escenario de prueba que incluye características esenciales, como una interfaz simple, menú de pausa y puntos de control. Estas adiciones buscaron facilitar la evaluación por parte de usuarios, de las mecánicas de juego desarrolladas, quienes brindaron retroalimentación valiosa para el desarrollo del videojuego *Zombattan*.

Todos los objetivos del proyecto se alcanzaron de manera satisfactoria, respaldados por resultados positivos. La ejecución de las mecánicas permitió a los participantes superar los desafíos planteados, generando reacciones positivas en su experiencia de juego. La retroalimentación recopilada se ha enfocado principalmente en el refinamiento de las mecánicas y nuevas características, más que en la necesidad de cambios estructurales, lo cual es un indicador positivo.

Las mecánicas desarrolladas, junto con la valiosa retroalimentación recibida, se convertirán en una guía crucial para dirigir el futuro progreso y perfeccionamiento de *Zombattan*. Este enfoque asegurará una experiencia de juego entretenida y novedosa en el futuro videojuego.

A mi yo del pasado, que insistió y perseveró.
A ustedes, que me guiaron y me enseñaron.

Gracias

Agradecimientos

Quiero expresar mi profundo agradecimiento a mi profesor guía por su constante apoyo y orientación durante todo el desarrollo de este proyecto.

Agradezco sinceramente a todos mis amigos de la universidad, cuyo apoyo y amistad fueron pilares esenciales en este viaje académico. Su colaboración y compañía contribuyeron significativamente a hacer más llevadero este proceso universitario.

A mi familia, por su apoyo incondicional en todas las facetas de mi vida, tanto académicas como personales. Su constante respaldo ha sido un pilar fundamental que me ha permitido alcanzar mis metas y superar desafíos.

Finalmente, una mención especial va dirigida a mi hermano Alonso C.A., quien ha sido una guía excepcional y el mejor amigo durante este camino que afortunadamente nos tocó compartir.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, el acelerado desarrollo de las tecnologías ha tenido un impacto significativo en todo ámbito en la vida de las personas, y el entretenimiento no es la excepción. Este sector ha sido profundamente influenciado por la tecnología, tanto en la forma en que se produce como en la manera en que se consume. Desde la aparición de los videojuegos, el entretenimiento digital ha evolucionado rápidamente, y en la actualidad se encuentra en una etapa de madurez en la que los videojuegos son considerados una forma de arte y una industria multimillonaria [1]. Los avances en esta área han permitido la creación de mundos virtuales cada vez más complejos y realistas, en los que los usuarios pueden experimentar aventuras inmersivas y hasta conectarse con otros jugadores en línea para jugar en conjunto. En este contexto, el desarrollo de videojuegos se ha convertido en una disciplina que requiere de un alto nivel de conocimientos técnicos y creativos para lograr crear una experiencia de juego atractiva y satisfactoria para los jugadores.

Ahora, si bien la tecnología está en constante evolución y ha impactado positivamente a la industria, también es cierto que progresivamente las expectativas sobre el contenido generado han ido cambiando, y hoy en día, se basan en la idea de que se supere constantemente a sus predecesores, ofreciendo nuevas experiencias que sorprendan al público. Es por esta razón, que los videojuegos enfrentan un gran desafío: mantenerse actualizados y generar contenido que cumpla con los estándares de calidad y expectativas de los jugadores. En otras palabras, la comunidad de *gamers*, o usuarios de videojuegos, es cada vez más exigente y los desarrolladores deben esforzarse por estar a la altura de las demandas para seguir siendo relevantes, considerando el impacto que los videojuegos tienen en la sociedad actual.

Todos estos avances técnicos y el gran impacto generado en cómo se desarrollan los videojuegos, han ampliado las posibilidades para los desarrolladores de manera aparentemente ilimitada. Sin embargo, crear un videojuego exitoso no se trata simplemente de saturar a los usuarios con características estimulantes y novedosas. Es fundamental diseñar un sistema coherente y consistente que se combine con una experiencia audiovisual que transmita la idea deseada al usuario.

Alcanzar este nivel de calidad no se limita únicamente a la creatividad y la adopción de nuevas tecnologías. Implica seguir criterios y cumplir con requisitos específicos, no solo para

garantizar que los usuarios finales disfruten del juego, sino también para manejar y administrar adecuadamente la multitud de entidades que conviven dentro de un videojuego y que deben estar en armonía y en constante comunicación para que todo funcione correctamente.

Como resultado, un videojuego no es solo un producto de entretenimiento, sino que un sistema complejo que abarca múltiples áreas de conocimiento con el objetivo final de proporcionar una experiencia especifica para los usuarios, y como resultado, brindar entretenimiento. Si alguna parte de este complejo sistema falla, es probable que los usuarios simplemente decidan no utilizar el juego.

En el mundo de los videojuegos, las ideas que se quieren transmitir y la forma en que esto se quiere lograr pueden variar ampliamente, lo que ha dado lugar a diferentes tipos de videojuegos que se adaptan a diversas necesidades y expectativas de los diferentes públicos. Cada tipo de juego tiene sus propias características distintivas que definen cómo se juega y cómo se desarrolla la experiencia de juego. Por ejemplo, jugar *Tetris*, un juego en 2D, es una experiencia diferente a jugar un juego 3D de rol en tercera persona como *The Legend of Zelda: Breath of the Wild*. El enfoque y el proceso de desarrollo de cada tipo de juego pueden ser completamente diferentes debido a las particularidades y requisitos únicos de cada uno. Este proyecto en particular, se centrará en el desarrollo de un juego de Sigilo y Estrategia en tercera persona. Este tipo de juego se caracteriza por requerir que el jugador se mueva con cautela y utilice tácticas y estrategias para superar los desafíos y alcanzar los objetivos.

En este genero en particular, la creación de una experiencia inmersiva es de vital importancia para el éxito del juego. Para lograr esto, es fundamental prestar especial atención al control del avatar o personaje que el jugador maneja en el videojuego y a cómo interactúa con el entorno virtual. Estas interacciones, son conocidas como mecánicas de juego, y son fundamentales para que el jugador pueda moverse, actuar y tomar decisiones dentro del mundo del juego de la manera deseada.

Es crucial seleccionar cuidadosamente las mecánicas de juego disponibles para que sean acordes al estilo de juego de Sigilo y Estrategia, y para asegurarse de que brinden una experiencia divertida y satisfactoria. Un control de personaje fluido y preciso es esencial para lograr una jugabilidad convincente. Esto implica diseñar y perfeccionar los movimientos, las acciones y las interacciones del personaje dentro del mundo que lo rodea, garantizando que respondan de manera intuitiva a las acciones del jugador en el contexto situado.

Dedicar tiempo y esfuerzo a diseñar y perfeccionar las mecánicas de control del personaje, son una etapa clave en el desarrollo de este tipo de videojuegos. Al hacerlo, se establecen las bases para una experiencia inmersiva que cautiva y entretiene al usuario, llevando al juego a alcanzar su máximo potencial.

1.2. Problema abordado

Este tema de memoria se enfoca en el desarrollo de estas mecánicas de juego, las cuales, se trabajarán sobre la base de un videojuego de Sigilo y Estrategia en tercera persona situado en un escenario post-apocalíptico infestado por zombis. El objetivo principal es diseñar

e implementar mecánicas de control del personaje que permitan una jugabilidad óptima en relación a este género. Sin embargo, es importante tener en cuenta que la creación de mecánicas, aunque sean complejas, no tienen sentido si no se logra conectar con los futuros usuarios del juego. Por esta razón, este trabajo también abarca la experimentación con potenciales usuarios y la identificación de mecánicas adecuadas que cumplan con los objetivos de Sigilo y Estrategia del videojuego en desarrollo. En resumen, se busca no solo crear mecánicas de juego efectivas, sino también asegurar que se ajusten a las necesidades y expectativas de los jugadores para lograr el objetivo final.

Dado que el desarrollo de este videojuego de zombis se encuentra en una etapa preliminar, la selección y aplicación de las mecánicas de juego adecuadas será crucial para definir la experiencia del usuario y lograr el éxito en el desarrollo del juego. Estas decisiones desempeñarán un papel fundamental en la creación del juego base, simplificando y orientando todo el proceso de desarrollo que vendrá a futuro.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo principal de este trabajo es diseñar e implementar un conjunto adecuado de mecánicas de juego de Sigilo y Estrategia para un videojuego en tercera persona ambientado en un mundo post-apocalíptico infestado de zombis. Estas mecánicas de juego serán fundamentales para sentar las bases del proyecto en desarrollo, brindando las herramientas necesarias para que el videojuego pueda avanzar en el futuro con una estructura sólida y mecánicas definidas. El enfoque se centra en desarrollar mecánicas que permitan al jugador adoptar diferentes estrategias de sigilo y enfrentarse a los desafíos del juego de manera estratégica. La implementación de estas mecánicas busca garantizar una experiencia de juego inmersiva y emocionante, donde el jugador se sienta completamente involucrado en la lucha por la supervivencia en un mundo post-apocalíptico lleno de peligros.

1.3.2. Objetivos específicos

1. Identificar mecánicas ya existentes de estrategia y sigilo, y formular propuestas concretas aterrizadas al contexto del videojuego en desarrollo.

2. Planificar la implementación correcta de las mecánicas escogidas, considerando cómo serán aplicadas en el espacio físico dónde se llevará a cabo la historia del juego, junto con otras consideraciones visuales y técnicas.

3. Generar el comportamiento de los NPC (*Non Playable Characters*) acorde a las mecánicas seleccionadas.

4. Hacer pruebas con usuarios sobre la implementación de las mecánicas de Sigilo y Estrategia, para validar lo desarrollado y recibir retroalimentación valiosa para el videojuego.

5. Dejar implementadas las mecánicas de Sigilo y Estrategia necesarias para sentar las bases de un *gameplay* novedoso y atractivo para el videojuego en desarrollo.

1.3.3. Evaluación

Desde el punto de vista técnico, el hecho de que el desarrollo se realice sobre un proyecto en *Unreal Engine 5* y que actualmente no cuenta con ninguna mecánica de juego implementada, significa que cualquier adición a las mecánicas de control del personaje será fácilmente perceptible en el juego. Por lo tanto, el diseño de movimientos, acciones, animaciones y la interfaz de usuario para controlar el personaje será una parte esencial del proceso de desarrollo. En última instancia, todas estas características serán un testimonio tangible del trabajo realizado y de la experiencia de juego resultante.

Ahora, si bien se espera que el desarrollo presente desafíos técnicos significativos, como los ya mencionados, el mayor reto será crear una jugabilidad que resulte atractiva y satisfactoria para los usuarios finales. Será necesario dedicar tiempo y esfuerzo para crear pruebas con jugadores y así lograr encontrar una experiencia de juego que cumpla con las expectativas y preferencias del público objetivo, y lo que se quiere transmitir.

En resumen, la evaluación de este proyecto se basará en la cantidad de mecánicas implementadas en el videojuego y los resultados derivados de una prueba de validación con usuarios, durante la cual estas mecánicas serán sometidas a evaluación.

1.4. Descripción general de la solución

La forma en que se abordó el problema constó de tres fases: diseño y evaluación de las mecánicas de juego, implementación de dichas mecánicas, y evaluación con usuarios. El desarrollo se realizó en el motor *Unreal Engine 5*, utilizando la metodología centrada en *Blueprints*, herramienta específica que ofrece *Unreal Engine*. Esta metodología agilizó la implementación y el desarrollo general, asegurando el cumplimiento de los objetivos dentro del plazo establecido.

Zombattan esta definido como un juego en tercera persona sobre un mundo post-apocalíptico plagado de zombis, con la intención de destacarse como un videojuego que fusiona elementos de Sigilo y Estrategia. En función de esta premisa, las mecánicas a proponer para su desarrollo debían estar intrínsecamente vinculadas a estas características distintivas del videojuego.

Para iniciar el diseño de las mecánicas de juego para *Zombattan*, se emprendió una exploración profunda del concepto concebido para este nuevo videojuego. Se abordaron aspectos como el género al que se aspira a pertenecer, el universo en el que los jugadores se sumergirán y los objetivos fundamentales que deberán perseguir en este entorno virtual. Esta base proporcionó un punto de partida sobre el cual formular las mecánicas de juego y los factores cruciales a tener en cuenta durante su diseño.

Además, desde el inicio del proyecto, se reconoció como crucial realizar la validación de cada propuesta de mecánica con usuarios, con el objetivo de obtener retroalimentación esencial en esta fase temprana del desarrollo. Es por esto que este punto clave fue considerado a la hora de seleccionar cuales mecánicas habría que diseñar para este proyecto.

Sin embargo, uno de los desafíos más significativos en este escenario fue la ausencia de cualquier tipo de mecánica de juego implementada en la fase de desarrollo que se encontra-

ba el proyecto. Se disponía únicamente de un proyecto de *Unreal Engine* que incluía unos *Polygon Assets*, correspondientes a los recursos que se facilitaron para este trabajo de titulo, con algunas animaciones y principalmente *assets* con el arte final destinado a *Zombattan*. Además, se contaba con un jugador en tercera persona, dotado de las funciones básicas de correr y saltar proporcionadas por la plantilla inicial de este tipo de videojuegos que ofrece *Unreal Engine*.

Ante esta situación, al proponer y diseñar las mecánicas de juego a desarrollar, se tuvo que considerar todos los aspectos abordables en el alcance del proyecto, desde nuevos movimientos del personaje hasta mecánicas que impactaran en el género del juego, específicamente en las áreas de Estrategia y Sigilo. Esto, sumado a la consideración de que había que aspirar a realizar una experiencia con usuarios al final del proyecto, planteó el desafío de abordar la mayor cantidad posible de características distintivas de un videojuego de este estilo. Condicionando, así, a diseñar mecánicas de juego coherentes entre sí, apuntando a crear una experiencia completa, incluso en una versión muy básica, pero lo suficientemente funcional para que los usuarios pudieran emplear las mecánicas desarrolladas sin inconvenientes y poder proporcionar retroalimentación que se alineara con lo que se buscaba en este proyecto. Este enfoque ayudó a sentar las bases para las mecánicas de juego de *Zombattan*, con el objetivo de comenzar a construir una experiencia de juego sólida y bien fundamentada desde una fase temprana del desarrollo.

De esta manera, tomando como referencia inspiraciones tanto en el ámbito pop sobre mundos post-apocalípticos infestados de zombis como también de mecánicas de juego de videojuegos exitosos que se destacaron en el género de Sigilo y Estrategia, se optó por abordar tres categorías de mecánicas. Este planteamiento tenía como objetivo cubrir una amplia gama de aspectos característicos de un videojuego al estilo de *Zombattan*, siendo conscientes al mismo tiempo de mantener un número viable de mecánicas dentro del marco de tiempo establecido para el desarrollo de este proyecto de memoria.

Las tres categorías de mecánicas se distribuyeron de la siguiente manera: Mecánicas de Movimiento, Mecánicas de Sigilo y Mecánicas de Combate.

Dada la necesidad de proporcionar al personaje mayores opciones de movimiento para que el juego pueda considerarse dotado de cualidades de Sigilo y Estrategia, se definieron mecánicas en la sección de Movimiento. Aquí se consideraron tanto mecánicas frontales que permitieran a los jugadores desplazarse de formas necesarias para sortear obstáculos, como mecánicas relacionadas con la forma en que el personaje se desenvolvería en el mapa, permitiendo la apertura de nuevos caminos o impedimentos.

El siguiente tipo de mecánicas, designadas como acciones de Sigilo, fueron concebidas para que el jugador pudiera llevar a cabo acciones de manera sigilosa al cumplir objetivos dentro del juego. A diferencia de las Mecánicas de Movimiento, estas implicarían interacciones directas con otros actores del juego, especialmente los NPC, *Non Playable Characters*, en este caso, los zombis. De esta manera, se brindaría al jugador la posibilidad de interactuar con los enemigos para superar desafíos sin enfrentarse directamente a ellos.

Finalmente, reconociendo que no siempre es posible evitar los problemas y peligros pre-

sentes en este mundo virtual post-apocalíptico, se consideró el último tipo de mecánicas, directamente relacionadas con el combate contra enemigos, denominadas Mecánicas de Combate. Esto amplía las posibilidades para que los jugadores superen las diversas dificultades dentro del juego.

Fue así entonces, basándose en estos antecedentes, que se eligieron ocho mecánicas de juego que abarcaban estas tres áreas identificadas como fundamentales para cubrir todos los aspectos importantes de un juego de Sigilo y Estrategia. Las mecánicas de juego finales propuestas para su implementación y el desarrollo de estas se detallan a lo largo de este documento.

En relación con las pruebas finales, se diseñó un escenario de prueba de manera que los usuarios tuvieran que superar múltiples obstáculos utilizando las mecánicas de juego desarrolladas. Después de esta experiencia, se administró un formulario con preguntas que abordaban diversos aspectos de la prueba, con el objetivo de obtener retroalimentación valiosa para el futuro desarrollo de las mecánicas de juego, así como del videojuego en sí.

Por último, es relevante señalar que este proyecto se ejecutó simultáneamente con otras iniciativas en diversas áreas, como parte integral del desarrollo de un videojuego bajo la responsabilidad de la empresa Time Vortex EIRL. Todos los recursos empleados son propiedad de Time Vortex EIRL, y la empresa otorgó la debida autorización para exhibir parte de estos elementos en este informe, facilitando así la explicación de la solución y proporcionando contexto necesario.

1.5. Estructura de la memoria

El resto de este documento sigue la siguiente estructura. En el Capítulo 2, se presenta y discute la literatura relacionada que respalda el trabajo ejecutado en este proyecto de título. El Capítulo 3 aborda las decisiones de diseño e ingeniería vinculadas a: (1) la selección y diseño de las mecánicas de juego; (2) la implementación de estas mecánicas; y (3) la creación de un mapa para las pruebas con usuarios. Luego, en el Capítulo 4, se describe el proceso llevado a cabo para realizar las pruebas con usuarios, proporcionando también los resultados obtenidos que validan el cumplimiento de los objetivos establecidos al inicio del proyecto. Finalmente, en el Capítulo 5, se presentan las conclusiones y se ofrecen perspectivas sobre el trabajo futuro.

Capítulo 2

Estado del arte

2.1. Situación actual

La industria de los videojuegos ha sido históricamente una de las más impactadas por los avances tecnológicos, especialmente en el ámbito del hardware y la capacidad de procesamiento. A pesar de las limitaciones técnicas en las primeras etapas, los videojuegos siempre han sido capaces de posicionarse como una forma popular de entretenimiento. Sin embargo, con el desarrollo continuo en el área, estas limitaciones técnicas se han reducido significativamente, lo que ha llevado a un mayor alcance y accesibilidad para una audiencia cada vez más amplia.

Estos factores han sido clave para el impresionante crecimiento de la industria, y han promovido el desarrollo creativo como nunca antes se había visto. Con menos limitaciones técnicas, los desarrolladores tienen más libertad para explorar nuevas ideas y conceptos, y esto ha llevado a la creación de juegos cada vez más innovadores y emocionantes.

Es por esta misma razón, que las expectativas de los usuarios de los videojuegos aumentan significativamente, lo que representa un gran desafío para los desarrolladores al tratar de satisfacer estas demandas. *Unreal Engine 5* [2], la última versión del motor de juegos creada por *Epic Games*, y la herramienta que se utilizó para el desarrollo de las mecánicas dentro de este juego, ofrece aún más posibilidades, eliminando muchas restricciones técnicas vistas en versiones anteriores, lo que abre las puertas a poder enfocarse en otros aspectos importantes en el proceso de creación, en este caso en particular, en la interacción del usuario con el mundo virtual creado.

2.2. Diseño de mecánicas de juego

El diseño de mecánicas de juego efectivas, que se adapten al contexto del personaje y logren transmitir la visión del desarrollador, es crucial para determinar la calidad de un videojuego. Sin embargo, es importante comprender bien el significado real del concepto de mecánica de juego para apreciar su verdadero impacto en el desarrollo de un videojuego.

En su paper “Gameplay and game mechanics design: a key to quality in videogames” [3], Carlo Fabricatore aborda de gran manera justamente esto. Para lograr comprender, primero hace falta comenzar desde lo más general. Fabricatore destaca que las opiniones y decisiones importantes de los jugadores surgen de la experiencia de juego, y explica que estos buscan

en los videojuegos desafíos, dominio y recompensas, todo ello inmerso en actividades motivadoras y cautivadoras. Más allá del contenido del juego, el jugador interactúa con el mundo virtual mediante acciones y respuestas que generan cambios en el estado del juego. Durante este ciclo interactivo, los jugadores manejan dos tipos de información: la funcional y la estética. La información funcional permite al jugador llevar a cabo las actividades necesarias para jugar e hipotéticamente ganar el juego, mientras que la información estética define el contexto del juego y crea una atmósfera que atrae y mantiene la atención del jugador. En este proyecto en particular, el enfoque se centra en la información funcional por encima de la estética.

Fabricatore, añade, que se identifican tres elementos clave que determinan la calidad de un videojuego: el contexto del juego, las actividades requeridas para ganarlo y la claridad en cuanto a lo que se debe hacer y cómo lograrlo. Sin embargo, no todos estos elementos tienen el mismo peso a la hora de evaluar un juego, y es que si la jugabilidad, o también conocida como *gameplay*, es mala, esto influirá directamente en la forma en que el jugador entiende lo que debe hacer. Por lo tanto, una mala jugabilidad, implica un mal juego. Pero, ¿Qué es la jugabilidad realmente?.

La jugabilidad se puede definir de diferentes maneras, y existen dos definiciones importantes que arrojan luz sobre su significado. La primera definición, independiente del jugador, establece que la jugabilidad es: "Una o más series de desafíos casualmente vinculados en un entorno simulado" (Rollings y Adams, 2003) [4]. Esta definición pone el énfasis en los desafíos que se presentan en el juego y cómo están relacionados entre sí. La segunda definición se centra en el jugador y suele hacer referencia a lo que los jugadores pueden hacer en el juego y cómo se juega. Esta definición destaca las acciones permitidas al jugador y la forma en que se lleva a cabo el juego (Bates, 2001; Lewinski, 1999; Rouse, 2001) [5]. Ambas definiciones resaltan dos aspectos fundamentales en los videojuegos: las acciones que un jugador puede realizar y cómo el juego responde a esas acciones.

Al comprender que la jugabilidad engloba estos dos conceptos específicos, podemos adoptar la definición más centrada en los jugadores para conceptualizar las mecánicas de juego: Son las herramientas adecuadas para lograr una jugabilidad satisfactoria.

Dentro del contexto de las mecánicas de juego, el desafío y la recompensa para el jugador se derivan de tres actividades interrelacionadas:

- Aprendizaje de las mecánicas.
- Utilización de las mecánicas como herramientas en situaciones ordinarias de juego.
- Utilización de las mecánicas como herramientas en situaciones extraordinarias de juego, en presencia de factores externos que pueden alterar el funcionamiento habitual de las mecánicas.

Para que una mecánica de juego sea considerada buena, debe cumplir ciertos criterios. Se busca un equilibrio entre el esfuerzo y el tiempo requeridos para aprender las mecánicas, el esfuerzo y el tiempo dedicados a utilizar lo aprendido, y las recompensas obtenidas al utilizar las mecánicas.

Es crucial tener en cuenta que no todas las mecánicas de juego dentro de un videojuego tienen el mismo nivel de importancia. Carlo Fabricatore distingue entre dos tipos: las mecánicas principales (core) y las mecánicas secundarias (satellites). Las mecánicas principales se refieren al conjunto de actividades que el jugador llevará a cabo con mayor frecuencia durante la experiencia del juego, y son indispensables para lograr la victoria en el mismo. Por otro lado, las mecánicas secundarias están diseñadas para enriquecer y complementar las mecánicas principales existentes.

En este proyecto, se enfoca principalmente en el desarrollo de las mecánicas principales del juego. Esto implica establecer una base sólida y bien diseñada que garantice el equilibrio y la coherencia de estas mecánicas fundamentales. Al poner un enfoque adecuado en las mecánicas principales, se sientan las bases para una experiencia de juego satisfactoria y se asegura que el videojuego se desarrolle de manera consistente y emocionante.

2.3. Inspiraciones

El objetivo será desarrollar mecánicas de juego apropiadas para un videojuego de Sigilo y Estrategia, las cuales se desarrollarán sobre una base que será facilitada, con una temática centrada en un mundo plagado de zombis. Para ello, se considerará el éxito de juegos como *Metal Gear Solid V* [6] y *A Plague Tale* [7], que en su momento, y hasta el día de hoy, han cautivado a los jugadores con mecánicas similares, y principalmente por la gran cantidad de posibilidades que brindan a los jugadores a la hora de desenvolverse en el contexto que se encuentran. Referencias como estas sirvieron de inspiración para el diseño y evaluación de mecánicas efectivas para este nuevo videojuego en el genero especificado.

Metal Gear Solid V, es el quinto juego de la exitosa serie de videojuegos de temática militar creada y dirigida por Hideo Kojima. La serie *Metal Gear Solid* es reconocida por su enfoque innovador en la narrativa, el sigilo y la jugabilidad estratégica. Este juego de la saga en particular, fue lanzado en 2015, y es considerado uno de los títulos más destacados de la serie. Ambientado en un contexto apocalíptico, el juego ofrece una experiencia inmersiva y emocionante a través de su extenso mundo abierto y su enfoque en la libertad del jugador para abordar las misiones. Además, se destaca por su mecánica de infiltración táctica, donde los jugadores pueden utilizar diversas estrategias para cumplir sus objetivos, ya sea a través del sigilo, el combate directo o el uso de *gadgets* especializados. La combinación de una narrativa intrigante, un diseño de nivel detallado y un sistema de juego profundo hacen de *Metal Gear Solid V* un hito en la industria de los videojuegos.

El enfoque en el sigilo y la estrategia permite a los jugadores experimentar una jugabilidad rica y táctica. La libertad de elección en la forma de abordar las misiones es uno de los aspectos más destacados del juego. Los jugadores tienen la opción de optar por un enfoque sigiloso, aprovechando el entorno y las habilidades del personaje para moverse sin ser detectados, o bien optar por el combate directo, enfrentándose a los enemigos de manera más agresiva.

El juego ofrece un amplio abanico de herramientas y mecánicas para el sigilo, como la posibilidad de esconderse en arbustos o detrás de paredes, llamar la atención de los guardias con sonidos estratégicos o silbidos, e incluso utilizar dardos tranquilizantes para neutralizar

a los enemigos sin alertar a los demás. Estas opciones permiten a los jugadores planificar y ejecutar sus estrategias de manera creativa y adaptarse a diferentes situaciones.

Además, el juego presenta un sistema de inteligencia artificial avanzado, donde los enemigos responden de manera realista a las acciones del jugador. Por ejemplo, si un guardia encuentra a un compañero inconsciente, puede ponerse en estado de alerta y aumentar la seguridad en la zona, lo que obliga al jugador a ser cauteloso y tomar decisiones rápidas.

Todas estas mecánicas y opciones de juego se combinan con una narrativa envolvente y una presentación visual y auditiva de alta calidad, que sumergen al jugador en el mundo del juego y aumentan la sensación de tensión y emoción. Estos aspectos no solo fueron apreciados por los fanáticos de la serie, sino que también establecieron nuevos estándares para el género.

Sin embargo, replicar estas mecánicas exitosas en distintos contextos puede resultar desafiante. Cada videojuego tiene su propia identidad y objetivos específicos, por lo que adaptar las mecánicas de juego exitosas a un nuevo escenario requiere un cuidadoso equilibrio. Es fundamental mantener la prioridad en la experiencia del usuario dentro del videojuego, asegurándose de que las mecánicas se ajusten y mejoren la inmersión y el disfrute del jugador.

Por el otro lado, en *A Plague Tale: Innocence*, otro exitoso juego del genero, lanzado en 2019, la historia se centra en la supervivencia de dos hermanos en la Francia del siglo XIV, durante la época devastada por la peste negra. Este juego logra combinar varios de los elementos mencionados anteriormente, centrándose principalmente en la narrativa y adaptando de manera ingeniosa las mecánicas de juego al contexto en el que se desarrolla la historia.

Este juego resalta principalmente, por como cuenta una historia a través del juego, la cual, destaca por su calidad y emotividad. A través de una cuidadosa construcción de personajes y una trama envolvente, el jugador se ve inmerso en el mundo sombrío y peligroso de la peste negra, experimentando las dificultades y los desafíos que los personajes deben enfrentar. Algo similar al enfoque que se busca en el juego en el que estaremos trabajando.

En cuanto a las mecánicas de juego, *A Plague Tale: Innocence* se destaca por su habilidad para adaptarlas de manera efectiva al contexto histórico y temático. El uso de las ratas como uno de los principales obstáculos a lo largo del juego es un ejemplo notable. Los jugadores deben utilizar estrategias y herramientas disponibles para protegerse de las ratas, aprovechando la luz y el sonido para mantenerlas a raya y avanzar en la historia.

Estas mecánicas, combinadas con la narrativa cautivadora, logran crear una experiencia inmersiva y emocional para el jugador. El juego muestra cómo una adaptación cuidadosa de las mecánicas de juego al contexto de la historia puede potenciar aún más la experiencia del jugador y generar una conexión más profunda con los personajes y el mundo del juego.

El análisis de estas referencias será fundamental para orientar el trabajo en la dirección adecuada y lograr crear mecánicas de juego que, si bien podrían ya existir, sean seleccionadas y adaptadas de manera efectiva para enriquecer la experiencia que se desea ofrecer en el nuevo videojuego en desarrollo. La clave está en identificar qué mecánicas son relevantes y aportan valor a la experiencia del jugador, descartando aquellas que no se alineen con la visión del

juego. La adaptación de estas mecánicas al mundo virtual en el que se estará trabajando requerirá creatividad y habilidad para lograr una integración coherente y satisfactoria. A grandes rasgos en este caso en particular, el personaje deberá sobrevivir en un mundo post-apocalíptico donde habitan zombis, y la habilidad de moverse sigilosamente será fundamental para alcanzar este objetivo. Por lo tanto, todo el núcleo del juego dependerá en gran medida de la calidad y nivel de adaptación de las mecánicas implementadas en este nuevo mundo.

2.4. Metodologías de desarrollo

Dentro de *Unreal Engine*, la metodología de trabajo se basa en un enfoque iterativo y eficiente para el desarrollo de videojuegos. Una parte esencial de esta metodología es el uso de *Blueprints*. El sistema de *Blueprints* en *Unreal Engine* es una potente herramienta de *scripting* visual que permite crear lógica de juego y funcionalidades interactivas sin necesidad de programar en lenguajes de programación tradicionales. Este sistema se basa en la creación de diagramas visuales de nodos que representan distintas funciones y relaciones, facilitando la visualización y comprensión del flujo de trabajo. [8]

El proceso de *scripting* en *Unreal Engine* mediante *Blueprints* implica conectar estos nodos mediante líneas que indican la secuencia de ejecución. Se pueden incluso crear funciones personalizadas para modularizar y reutilizar la lógica [9]. La representación visual de los Blueprints facilita la comprensión del flujo de ejecución y ofrece una interfaz accesible para aquellos que pueden no tener experiencia en programación tradicional. Otra característica destacada es la capacidad de ver la ejecución del Blueprint en tiempo real dentro del editor. Esto permite iterar y depurar de manera efectiva, mejorando el flujo de trabajo y reduciendo el tiempo de desarrollo.

La capacidad de trabajar con *Blueprints* no excluye el uso de *scripting* tradicional en C++ en *Unreal Engine*. Esto da la posibilidad de combinar ambos enfoques según la complejidad y los requisitos específicos del juego. Esto proporciona flexibilidad y escalabilidad en el desarrollo de proyectos de cualquier tamaño y complejidad.

En este contexto, la metodología comienza con la fase de diseño y planificación, donde se establecen los conceptos y objetivos del juego. Luego, con las ideas ya definidas, en el entorno de desarrollo de *Unreal Engine*, los *Blueprints* entran en juego. Estos ofrecen una interfaz gráfica intuitiva donde se permite diseñar la lógica de juego principalmente mediante la conexión de nodos.

Los *Blueprints* abarcan diversas áreas, como la definición de comportamientos de personajes, la creación de eventos, la manipulación de activos y creación de objetos. Se utilizan para construir sistemas de movimiento, lógica de combate, interacciones de usuario y más. La versatilidad de los *Blueprints* permite visualizar y ajustar directamente el flujo del juego, facilitando la comprensión y modificación de la lógica.

Además, la metodología incluye la importación de activos, la aplicación de materiales y la creación de animaciones, todos los cuales pueden integrarse con los *Blueprints* para lograr una experiencia de juego cohesiva. Durante el proceso, se lleva a cabo una fase de pruebas y depuración constante, donde los *Blueprints* facilitan la identificación y resolución de problemas.

La iteración es clave en este enfoque, ya que los *Blueprints* permiten ajustar y mejorar continuamente la lógica del juego según las pruebas y la retroalimentación. Esta metodología, centrada en el desarrollo visual y la iteración eficiente, proporciona una poderosa herramienta para la creación y evolución de experiencias interactivas dentro de *Unreal Engine*.

2.5. Conceptos involucrados

Entender conceptos claves de los *Blueprints*, tanto como de *Unreal Engine*, es fundamental para comprender mejor lo realizado durante este proyecto, ya que fueron las herramientas y tecnologías en las que se basó la solución. A continuación se hará un repaso general de los conceptos más importantes utilizados durante esta etapa.

Dentro de *Unreal Engine*, los *Blueprints* son una herramienta central para el desarrollo, permitiendo la creación de lógica de juego mediante una interfaz visual y nodos conectados.

La creación de lógica en un *Blueprint* implica arrastrar y soltar nodos desde una amplia gama de categorías. Algunas de estas categorías incluyen:

- **Eventos**: Activados por acciones específicas como el inicio del juego, una colisión, o la interacción del usuario, estos eventos son lo más utilizado para el *scripting* dentro de un Blueprint.
- **Funciones y Operadores**: Permiten realizar cálculos, comparaciones y operaciones lógicas.
- **Componentes y Actores**: Representan los elementos visuales y lógicos del juego que se pueden manipular.
- **Controles de Flujo**: Como bucles y condicionales, para controlar la ejecución de la lógica.
- **Variables**: Almacenan información que puede cambiar durante la ejecución y afectar la lógica.

Los *Blueprints* se dividen en varios tipos, siendo los más comunes los *Level Blueprints* y las *Blueprint Classes*, en este caso en particular las que son de mayor interés son las *Blueprint Classes*, o bien, abreviado a veces a solo *Blueprint* [10].

Una *Blueprint Class* es un recurso que permite añadir fácilmente funcionalidad sobre las clases de juego existentes. Los *Blueprints* se crean visualmente dentro del Editor de *Unreal Engine*, en lugar de escribir código, y se guardan como activos en un paquete de contenido. Esencialmente, definen una nueva clase o tipo de Actor que luego se puede colocar en mapas como instancias que se comportan como cualquier otro tipo de Actor.

Existen varios tipos de *Blueprints* que se pueden crear, sin embargo, antes de hacerlo, se deben especificar la Clase Padre en la cual se basará el *Blueprint*. Seleccionar una Clase Padre permite heredar propiedades de esta clase para usarlas en el *Blueprint* que se está creando.

A continuación, se presentan las Clases Padre más comunes utilizadas al crear un nuevo *Blueprint*:

- **Actor**: Un Actor es un objeto que puede ser colocado o generado en el mundo.
- **Pawn**: Un Pawn es un Actor que puede ser "poseído" y recibir entrada de un Controlador.
- **Character**: Un Character es un Pawn que incluye la capacidad de caminar, correr, saltar y más.
- **Player Controller**: Un Player Controller es un Actor responsable de controlar un Pawn utilizado por el jugador.
- **Game Mode**: Un Game Mode define el juego que se está jugando, sus reglas, puntuación y otros aspectos del tipo de juego.

Aunque las anteriores son las más comunes, todas las clases existentes pueden ser utilizadas como Clase Padre para un nuevo *Blueprint* (incluso otras *Blueprint Classes*).

A continuación se especifican dos clases más que no figuran dentro de las más comunes, pero que de todas formas fueron requeridas y bastante utilizadas durante este proyecto.

Primero, tenemos los *Animation Blueprints* que corresponden a un *Blueprint* especializado que controla la animación de un *Skeletal Mesh*, que detallaremos a continuación. Los gráficos se editan dentro del Editor de *Animation Blueprint*, donde se pueden realizar mezcla de animaciones, controlar directamente los huesos de un esqueleto o configurar lógica que finalmente definirá la posición de la animación final que un *Skeletal Mesh* utilizará por cuadro [11].

El otro elemento bastante utilizado, son las *Blueprints Interfaces*, o bien, interfaces. Una *Blueprint Interface* es una colección de una o más funciones, solo con nombres y sin implementación, que pueden agregarse a otros *Blueprints*. Cualquier *Blueprint* al que se le añada la Interfaz está garantizado de tener esas funciones. Las funciones de la Interfaz pueden ser dotadas de funcionalidad en cada uno de los *Blueprints* que la agregó. Esto es esencialmente similar al concepto de interfaz en la programación general, que permite que varios tipos diferentes de objetos compartan y sean accesibles a través de una interfaz común [12].

En cuanto a la interacción con el usuario es importante explorar un concepto en particular, que si bien este apartado no fue una de las prioridades durante esta fase de desarrollo, de todas formas se vio involucrada de una u otra forma con lo realizado. Se hace referencia a los *Heads-up Displays* (HUDs), encargados de proporcionar información sobre el juego al jugador y, en algunos casos, permitir al jugador interactuar con el juego. En *Unreal Engine*, el HUD es el objeto principal para mostrar elementos superpuestos en la pantalla. Cada jugador controlado por humano en el juego tiene su propia instancia de la clase AHUD, la cual se encarga de dibujar en su propio *Viewport* individual [13].

A continuación, también exploraremos algunos de los componentes más recurrentes para tener claridad de estos conceptos previo a la explicación de la solución:

- **Widget**: Es un componente visual que se utiliza para crear interfaces de usuario (UI) y elementos gráficos en un juego. Estos Widgets pueden utilizarse para construir menús, paneles informativos, indicadores de salud, y prácticamente cualquier elemento visual que necesites para la interacción del jugador. [14]

- **Collisions and Traces**: En el contexto de Unreal Engine, las colisiones y los traces son componentes fundamentales para gestionar la interacción y detección de objetos en el entorno del juego. La colisión se refiere al proceso mediante el cual los objetos en el mundo interactúan entre sí, determinando si existe algún tipo de contacto o superposición. Los tipos más comunes son las *Box Collision*, *Sphere Collision* y las *Capsule Collision*. Por otro lado, los traces son líneas virtuales trazadas en el espacio 3D para evaluar la existencia de obstáculos o la intersección con objetos específicos. [15]

- **Static Meshes**: Los *Static Meshes* son la unidad básica utilizada para crear la geometría del mundo en los niveles creados en *Unreal Engine*. Estos son modelos 3D creados en aplicaciones externas de modelado que se importan al *Unreal Editor* a través del *Content Browser*, se guardan en paquetes y luego se utilizan de diversas maneras para crear elementos renderizables. La gran mayoría de cualquier mapa en un juego realizado con Unreal estará compuesta por *Static Meshes*, generalmente en forma de *Static Mesh Actors*. [16]

- **Skeletal Meshes**: Los Skeletal Meshes están compuestos por dos partes: un conjunto de polígonos que conforman la superficie del Skeletal Mesh y un conjunto jerárquico de huesos interconectados que se pueden utilizar para animar los vértices de los polígonos. Frecuentemente, en Unreal Engine, se utilizan Skeletal Meshes para representar personajes u otros objetos animados. [17]

Finalmente, se especificará una descripción general de lo que vendría siendo el *Third Person Template* que ofrece *Unreal Engine*, esto puesto que se utilizó como la base para comenzar el desarrollo.

Para crear el proyecto, se utilizó *Unreal Engine* que proporciona una lista de plantillas entre las cuales se puede elegir. Estas plantillas contienen algunos Activos listos para usar, como geometría del nivel, un personaje que se puede controlar y animaciones de personajes simples.

En un juego en tercera persona, el jugador ve el mundo del juego desde una cámara ubicada a una distancia fija detrás y ligeramente arriba de su personaje. En *Unreal Engine*, se puede controlar la distancia y posición de la cámara, y ajustarse según sea necesario, sin embargo, esta configuración viene lista junto con la plantilla para este tipo de juegos. [18]

La plantilla de Tercera Persona en *Unreal Engine 5* contiene los siguientes elementos:

- Un personaje jugable en tercera persona que puede moverse y saltar.
- Mallas adicionales para el personaje.
- Un nivel con geometría básica (rampas, plataformas).
- Cubos con física que reaccionan cuando el jugador colisiona con ellos.

- Maniquíes rediseñados.

En este proyecto sólo se utilizó lo referente al personaje y la configuración inicial correspondiente a este. Es decir, los *Assets* del maniquí de un personaje, las animaciones que permiten el movimiento básico (Saltar y correr), y los *Blueprints* vinculados a estos.

Figura 2.1: *Feminine and Masculine Unreal Engine 5 Mannequin*, pertenecientes a los *Assets* del *Third Person Template* de *Unreal Engine 5*.

Capítulo 3

Solución

3.1. Diseño Mecánicas de Juego

A continuación veremos como se abordaron tres tipos de mecánicas para el videojuego, asegurando una cobertura integral de los diversos aspectos inherentes a un juego con las características de *Zombattan*. Al lograr este cometido, se podrían establecer bases sólidas para el futuro desarrollo del videojuego y la mejora continua de estas mecánicas en iteraciones posteriores, cumpliendo así con el objetivo central de este proyecto.

3.1.1. Definición de las propuestas

En el ámbito de las mecánicas de movimiento, se dio prioridad inicial a definir mecánicas que proporcionaran al jugador opciones de movimiento que parecían esenciales para un juego de este estilo. Por este motivo, las acciones de agacharse y acostarse fueron las primeras mecánicas consideradas como necesarias. Una vez que el jugador contara con las formas básicas de desplazarse en el mapa, se exploraron maneras de enriquecer la experiencia en un mundo post-apocalíptico y, al mismo tiempo, introducir dificultades para no hacer tan sencillo moverse a través de los niveles. Bajo estas consideraciones, se optó por incluir el desplazamiento y colocación de tablas para habilitar nuevos caminos, una práctica observada en varios videojuegos de esta índole. La versatilidad de esta mecánica, su utilidad en diversos contextos y su capacidad para crear distintas dificultades, la hicieron indispensable. En línea con esta idea, se propuso agregar el desplazamiento de objetos, ya sea para abrir nuevos pasajes o como soporte en distintos escenarios. Esta mecánica, combinada con la colocación de tablas, ofrecía una amplia gama de posibilidades para generar rompecabezas o retos dentro del mapa, desafiando al jugador a superar obstáculos.

En cuanto a las mecánicas de acciones de sigilo, las inspiraciones provenientes de la cultura pop, particularmente de series con temática zombi, tuvieron un peso considerable en la toma de decisiones. Acciones como cubrirse con sesos de zombis muertos para pasar desapercibidos entre ellos, o el uso de sonidos para atraer a los zombis, fueron elementos clave que se reconocieron que serían lo bastante atractivos para incorporar al juego. Después de evaluar su factibilidad y cómo se integraban con las mecánicas de movimiento propuestas anteriormente, se determinó que estas acciones agregarían un valor significativo, especialmente en el ámbito del sigilo e interacción con los enemigos, que era uno de los objetivos a lograr con las mecánicas en este videojuego. Así, se concretaron dos mecánicas: la posibilidad de

lanzar objetos para generar sonidos y atraer a los zombis a una ubicación específica, y la capacidad del jugador para cubrirse con sesos de zombis muertos y volverse indetectable por otros zombis durante un tiempo determinado.

Explorando el último aspecto crucial, se reconoció la inevitabilidad de situaciones violentas en un juego de zombis, por lo que se consideró esencial incorporar alguna forma de combate. Sin embargo, para brindar al jugador una variedad de opciones, se optó por desarrollar tanto combate a distancia como combate cuerpo a cuerpo. Esta elección no solo proporciona al jugador la libertad de elegir su estilo preferido, sino que también añade un atractivo adicional a la jugabilidad del videojuego.

En última instancia, se llevarían a cabo todos los ajustes necesarios para ofrecer una experiencia coherente a los usuarios, permitiéndoles utilizar las mecánicas en un entorno que refleje fielmente el mundo propuesto en el videojuego. Esto facilitaría la obtención de la retroalimentación pertinente sobre el desarrollo. Por esta razón, la planificación de estas pruebas finales fue crucial desde el principio, permitiendo que las mecánicas apuntaran a proporcionar una experiencia de juego lo más completa posible dentro de los plazos y alcances establecidos por el proyecto.

Adicionalmente, estas pruebas brindarían una clara indicación de la aceptación por parte de los jugadores hacia las mecánicas propuestas y su disfrute en un videojuego con este estilo de juego. En caso contrario, se podrían realizar ajustes pertinentes en base a la retroalimentación recopilada en esta fase o en fases futuras del desarrollo del videojuego, asegurando así una adaptación efectiva a las preferencias y expectativas de la audiencia.

3.2. Desarrollo Mecánicas de Juego

3.2.1. Preparación

Este proyecto se llevó a cabo simultáneamente con otras iniciativas que abordaban diversos aspectos del videojuego. Con el fin de gestionar de manera efectiva el desarrollo de las mecánicas, se implementó un sistema de versionado en *GitHub*, con un repositorio específico que contenía los archivos del videojuego y una rama dedicada exclusivamente al desarrollo de las mecánicas.

Dado que el videojuego se estaba construyendo en *Unreal Engine*, fue necesario configurar el entorno para utilizar este motor de videojuegos. Se aprovechó el código base proporcionado por *Epic Games*, empresa a cargo de *Unreal Engine*, y se compiló con *Visual Studio*. Aunque este enfoque permitía examinar detalladamente partes del código del motor y brindaba la posibilidad de modificarlo si fuera necesario, estas características no resultaron fundamentales para los objetivos específicos de este proyecto. No obstante, es una práctica recomendada al trabajar en proyectos de esta envergadura, ya que proporciona flexibilidad y control sobre el desarrollo del juego.

La versión del motor utilizada para este proyecto fue la 5.2 de *Unreal Engine*. Dentro de los archivos principales se incluyeron *assets* que se anticipaban como los definitivos para el

juego, lo cual resultó invaluable al proporcionar una amplia variedad de recursos que abarcaban todo lo necesario para el contexto de las mecánicas propuestas. Entiéndase *assets* como una representación de cualquier ítem que puede ser utilizado en un juego o proyecto. Estos recursos se tradujeron en un significativo respaldo para el desarrollo, dado que se contaba con elementos visuales y funcionales que representaban de manera precisa la visión final del juego. En este proyecto en especifico serán referenciados como *Polygon Assets* puesto que el arte particular que utilizan es de tipo poligonal.

Por otro lado, se disponía de aproximaciones previas a una clase *Blueprint* del personaje y de un ejemplo de un zombie. Sin embargo, estos solo incluían movimientos básicos, como caminar y perseguir al personaje, en el caso del zombi. Se tomó la decisión de no utilizar estas implementaciones previas y empezar desde cero. Esto se hizo principalmente para separar el trabajo por completo de lo existente en los archivos previos y evitar posibles problemas de compatibilidad entre los *Blueprints* nuevos a realizar y los existentes configurados en C++. Se optó por desarrollar todas las mecánicas a través de la herramienta de *Blueprints* con el objetivo de agilizar el proceso de programación y asegurar una mayor coherencia en la implementación de las mecánicas.

De esta manera, se inició el proyecto a partir de la plantilla inicial de un juego en tercera persona proporcionado por *Unreal Engine*. Esta plantilla incluía una clase *Blueprint* de un personaje capaz de moverse mediante la entrada del usuario, con funciones adicionales de correr y saltar. Además, se incorporaba una cámara que seguía al personaje, junto con las animaciones correspondientes para estas acciones. Como parte del proceso inicial, se creó un nivel de prueba que serviría como el entorno de desarrollo inicial, desde donde se comenzaría a trabajar en las mecánicas de juego.

3.2.2. Configuración inicial

Una de las primeras tareas llevadas a cabo consistió en configurar la clase *Blueprint* del personaje, que venía junto con la plantilla para juegos en tercera persona, para que se adecuara a la apariencia que tendría el personaje en el videojuego. Para lograr esto, se implementó un proceso de *animation retargeting* específico para los *assets polygon* proporcionados en el proyecto. El *animation retargeting* en un personaje posibilita compartir animaciones entre personajes que utilizan diferentes esqueletos siempre y cuando compartan una jerarquía de huesos similar y utilicen un *asset* compartido llamado *Rig* para transmitir datos de animación de un esqueleto a otro. Gracias a esta técnica, el personaje puede moverse y comportarse de manera similar al personaje 3D original de *Unreal Engine*, pero ahora utilizando el esqueleto del personaje perteneciente a los *Polygon Assets*, que forma parte de los recursos proporcionados en el proyecto.

Este proceso de *animation retargeting* no solo fue técnico, sino que también resultó en una mejora visual significativa, ya que ahora fue posible emplear los *assets* específicos indicados para este proyecto. Esta adaptación permitió asignar una apariencia única al personaje en el juego, añadiendo un nivel de personalización visual que contribuirá de manera destacada a la identidad visual y estética del videojuego.

Figura 3.1: *Animation Retargetting* entre personaje original de *Unreal Engine* (Izquierda) y personaje de los *assets* especificos para el proyecto (Derecha).

Posteriormente, fue necesario configurar un nuevo *Game Mode* en el motor de videojuego. En *Unreal Engine*, el *Game Mode* define el conjunto de reglas del juego, que pueden incluir cómo los jugadores se unen al juego, la pausa del juego, la transición entre niveles, así como cualquier comportamiento específico del juego, como las condiciones de victoria. El *Game Mode* se establece para cada nivel, y puede reutilizarse en varios niveles.

En este caso, inicialmente se creó un *Game Mode* específico para el desarrollo, donde la única configuración relevante fue la especificación de la clase *Blueprint* del nuevo personaje, llamada "BP_ThirdPersonTestCharacter_new". Esto, para indicar qué personaje se utilizaría, y se asignó el mapa o nivel correspondiente, en donde se especificó el nivel ya creado para este propósito, sirviendo como el punto de inicio para el desarrollo del juego.

3.2.3. Mecánicas de Movimiento

3.2.3.1. Mecánica 1: Agacharse

Para implementar la mecánica de agacharse, se hizo uso de la lógica integrada en la clase *Character* de *Unreal Engine*, la cual es una clase padre de la clase especifica para el personaje e incluye funciones diseñadas para un sistema de agacharse, aunque estas no estén implementadas en la plantilla inicial por defecto. El primer paso fue adquirir las animaciones necesarias para la acción de agacharse, tanto en posición estática como en movimiento. Estas animaciones fueron obtenidas de fuentes que ofrecen animaciones de uso libre, como "Mixamo". A pesar de la consideración inicial de no necesariamente agregar animaciones a cada mecánica implementada, se tomó la decisión de incluirlas para marcar una diferencia significativa en la percepción de las mecánicas, sobretodo de forma visual, especialmente teniendo en cuenta que, en la etapa final del proyecto, se planeaba que los usuarios probaran las mecánicas desarrolladas.

A continuación, se creó una nueva "*Input Action*" específica para esta acción. Las "*Input Actions*" se entienden como el enlace de comunicación entre el sistema de entradas y el código del proyecto en *Unreal Engine*. Posteriormente, se configuró esta nueva "*Input Action*" en un contexto de mapeo de entrada (*Input Mapping Contexts*), que son colecciones de "*input actions*" que representan un determinado contexto en el que el jugador puede encontrarse. En este caso, no fue necesario realizar una configuración extensa; solo se especificó la tecla con la que se activaría la acción. Para la acción de agacharse, se decidió que la tecla sería "Ctrl izq.". Con estos pasos realizados, es posible entonces reconocer la entrada desde el código o desde un *Blueprint*, lo que marcó el siguiente paso en el proceso.

Una vez creada la "*Input Action*" en el contexto del personaje, podemos definir en la clase *Blueprint* del personaje, específicamente en su "*Event Graph*", el evento que se desencadenará cuando se perciba una entrada para esa acción en particular, indicando así cuándo el usuario ha presionado la tecla designada. Con esto, se procede a definir la secuencia de comandos que darán forma a la mecánica.

Para el sistema de agacharse, la piedra angular fue la creación de un sistema de estados, implementado mediante variables booleanas dentro de la clase *Blueprint* del personaje. En el caso de agacharse, fue necesario crear una variable llamada "is Crouching", la cual, según su estado, indicará si el personaje está actualmente realizando la acción de agacharse o no. La utilidad de esta lógica radica en la capacidad de discernir en otras partes del código qué acciones deben realizarse según el valor de esta variable.

Si la variable "is Crouching" es falsa, indicando que el jugador no está actualmente agachado, simplemente se llama a una función, "Crouch" que forma parte de la clase *Character*. Esta función ayuda a modificar el tamaño de la cápsula de colisión del personaje, permitiendo que, en el juego, la altura de la colisión sea menor y el personaje pueda pasar por lugares más bajos, tal como se esperaría de alguien que se mueve agachado. Además, reduce ligeramente la velocidad, lo cual tiene sentido para esta forma de movimiento. Asimismo, existe una función llamada "Uncrouch" que revierte estos cambios readaptando la cápsula de colisión y su velocidad a su estado original.

No obstante, para utilizar estas funciones es necesario realizar algunas configuraciones en la clase, como cambiar algunas variables. Específicamente, la variable "Can crouch" debe indicar que se permitirá que estas funciones tengan efecto. Además, se debe ajustar la variable "Crouched half height" para indicar la altura que tomará la cápsula durante el estado de agacharse en el personaje. Opcionalmente, se puede especificar "Max walk speed crouched" para indicar la velocidad que se desea al estar agachado. Todas variables inherentes a la clase *"Character"*, que hereda la clase "BP_ThirdPersonTestCharacter_new".

Finalmente, fue necesario generar los comandos necesarios para representar esta mecánica en las animaciones del personaje, lo cual se maneja en el *Animation Blueprint* vinculado al personaje. Dado que el sistema de animaciones está en otro *Blueprint*, la forma de establecer una comunicación entre el *Blueprint* de las animaciones y el *Blueprint* del personaje, encargado de manejar la lógica de agacharse, fue aprovechar uno de los eventos principales del *Animation Blueprint* el "Event Blueprint Update Animation", encargado principalmente de actualizar las animaciones en cada fotograma.

En este evento, originalmente se recuperan datos como la velocidad actual del personaje, que también se extrae del *Blueprint* del personaje. Por lo tanto, sólo hizo falta adaptarlo para que, además de los datos originales obtenidos desde *Blueprint* del personaje, también leyera la variable de estado que se definió anteriormente y que se replicó en el *Blueprint* de la animación con el mismo nombre: "is Crouching". Así, cuando la variable original cambia en el personaje porque este se agachó, en el *Blueprint* de la animación se actualiza su versión propia, proporcionando la información del estado, necesaria en este nuevo contexto.

Con esto, se pudo configurar la máquina de estados del *Animation Blueprint*. Estos son sistemas modulares que se pueden construir en un *Blueprint* de animación para definir ciertas animaciones que pueden reproducirse y cuándo se les permite reproducirse. Para indicarle a la máquina de estados cómo cambiar a las nuevas animaciones correspondientes al estar agachado estático o en movimiento, se generaron dos nuevos estados. Luego, haciendo uso del movimiento y la variable "is Crouching", ahora disponible en este contexto, se estableció la lógica. Si la variable "is Crouching" es verdadera y no hay movimiento, la máquina de estados transicionará al estado de agachado estático, llamado "Crouch". Si la variable es verdadera y además hay movimiento, entonces la máquina de estados transicionará al estado de agachado en movimiento, denominado "CrouchWalk". Si la variable es falsa, la máquina de estados volverá a los estados originales definidos, correspondientes al personaje en posición de pie.

Figura 3.2: Visualización del personaje agachado en el juego.

3.2.3.2. Mecánica 2: Acostarse/Arrastrarse

En el sistema de acostarse/arrastrarse, la lógica de la implementación fue similar, pero en este caso hubieron diferencias en el proceso. Una de las primeras distinciones significativas fue la ausencia de funciones predeterminadas en la clase *Character* para este tipo de postura, a diferencia de lo que sucedía con la mecánica de agacharse. Por lo tanto, no fue suficiente con configurar y hacer el llamado a una función ya existente.

La primera decisión de diseño relevante en esta etapa de desarrollo se centró en determinar si combinaríamos las mecánicas de movimiento en una sola acción. Este enfoque, común en varios videojuegos, implica que el cambio de posición o estado del personaje se gestiona con una sola entrada (o tecla), lo que permite al jugador cambiar rápidamente de un estado a otro. En otras palabras, una sola tecla es destinada para cambiar las posiciones del personaje, las cuales alternan tras cada interacción por parte del usuario. Por ejemplo, al estar parado, y querer acostarse, habría que transicionar a agacharse y luego transicionar nuevamente para llegar a la posición deseada. No obstante, existe otro enfoque, utilizado en algunos videojuegos, que separa por completo el funcionamiento y la lógica de pasar de un estado a otro. En este caso, se utilizaría una tecla independientes para cambiar a cada posición en especifica, por lo que las transiciones a cada posición del personaje se realizan de forma independiente una con la otra. La ventaja de esta última opción es evitar que los jugadores tengan que alternar entre los tres estados antes de llegar al deseado o incluso para prevenir mezclas perjudiciales para la experiencia del usuario.

Considerando el contexto de este videojuego y la etapa de desarrollo en la que se encuentra, se optó por el segundo enfoque por dos razones principales. En primer lugar, para priorizar el uso de un solo cambio de estado, ya que en este caso en particular, se pretende que estar agachado se utilice con mayor frecuencia que acostarse. En segundo lugar, para diferenciar la implementación de ambas mecánicas y facilitar la posibilidad de descartar una de ellas en el futuro sin implicar modificaciones sustanciales en la otra.

Así pues, lo primero fue definir una nueva entrada para esta acción, configurada para que la tecla de entrada fuera "Z". Se generó una nueva variable llamada "is Prone" para representar esta posición en particular, y se vinculó de la misma manera que en el sistema de agacharse con el *Blueprint* de animación, creando una variable equivalente en ese contexto. Sin embargo, en esta mecánica en particular se agregó un estado de transición, lo que significa que al acostarse existirá un momento breve en el que el jugador estará en transición de acostarse. Esto permite lograr una transición fluida entre los estados de pie y acostado, evitando que en el juego se vea como un cambio brusco de estado. Para agacharse, esto no era necesario, ya que la diferencia de altura y posición con estar de pie no es tan abrupta como para que la transición se vea mal.

Para agregarle animaciones a la transición, es necesario crear una variable que indique cuando el jugador está en transición. Por lo tanto, se creó la variable "gettingUp" que en combinación con la variable del estado real de estar acostado, será suficiente para marcar las transiciones. Esta variable también se vincula con el *Blueprint* de animación. Es importante destacar que en este punto, ahora existen dos posibles estados para el personaje, además del estado inicial (sin contar el estado de transición, que solo se evalúa momentáneamente): si está agachado o si está acostado. Para evitar complicaciones y posibles interferencias entre las mecánicas, es necesario integrar al evento creado anteriormente para agacharse, que ahora además evalúe si la variable "is Prone" es falsa para poder ejecutar el resto de los comandos. De no ser así, no queremos que continúe con la ejecución, ya que indicaría que el personaje estaría agachado y acostado al mismo tiempo. De manera similar, es necesario evaluar lo mismo en la ejecución de los comandos para cuando se active el evento de acostarse, asegurándose de que el personaje no esté agachado antes.

Ahora, observemos los comandos que definen la acción de acostarse para el personaje. El evento está vinculado a la entrada que definimos, como ya discutimos anteriormente. Lo primero es evaluar si el personaje no está agachado. De ser así, pasamos a la evaluación de si "Is Prone" es verdadero o falso, lo que determinará los comandos ejecutados para acostarse o levantarse.

Comencemos analizando cómo se realiza la acción de acostarse. Esto sucederá si "Is Prone" se encuentra en falso. En primer lugar, cambiamos su valor a verdadero para indicar que estamos ingresando a este estado y modificamos "Getting Up" a falso para señalar que la transición será acostarse, no levantarse. Posteriormente, ajustamos la velocidad de movimiento del personaje a 100 cm/s y realizamos una transición fluida con tres características clave. Primero, ajustamos la posición del mesh del personaje para evitar que quede en el aire al cambiar la animación. Esto asegura que la nueva posición de estar acostado tenga coherencia dentro del mundo del juego y en el personaje. Transiciona desde una posición inicial de estar de pie a una posición menor cuando está acostado, simulando que va por el suelo. En segundo lugar, modificamos el radio de la cápsula de colisiones; en este caso, lo reducimos para evitar complicaciones con la movilidad al estar acostado. Por último, ajustamos la altura de la cápsula de colisiones para que el jugador adapte sus colisiones a su nueva posición de estar acostado y no choque con elementos más altos.

En el caso en que el jugador se levanta, "Is Prone" estaría en verdadero. Lo primero que hacemos en este caso es generar un trazo vertical hacia arriba con la altura exacta correspondiente a la altura del personaje de pie. Esta acción asegura que sobre el personaje, cuando está acostado, no haya elementos que impidan ponerse de pie. Si el trazo detecta algún elemento dentro del alcance especificado, evitamos que el jugador se levante para prevenir errores y problemas. En caso contrario, cuando el área está libre para levantarse, cambiamos la variable "Is Prone" a falso para indicar que estamos saliendo de este estado y actualizamos la variable "Getting Up" a verdadero, ya que el jugador comenzaría a levantarse. Después, introducimos un tiempo de retraso, que debe coordinarse con la duración de la animación de levantarse, y finalmente restauramos la velocidad de movimiento del jugador a su valor inicial. Luego, realizamos la misma transición fluida en las tres características claves mencionadas anteriormente, pero esta vez revirtiendo los valores a los originales.

En cuanto a las animaciones, se añadieron cuatro nuevas, dos para las transiciones de levantarse y acostarse, y dos para el estar acostado estático y en movimiento. Luego, una vez con las variables que indican el estado del personaje ("is Prone") y la transición ("Getting Up") funcionando en el contexto adecuado, hace falta crear dos estados más para la máquina de estados: transición a acostarse y cuando está efectivamente acostado. Estos estados se llamaron "Prone transition" y "Prone", respectivamente. Se puede notar que ahora es diferente a estar agachado, y esto se debe principalmente al hecho de que se agrega la transición.

En este caso, al ser distinto, para lograr que cambien las animaciones cuando está acostado estático a cuando está acostado en movimiento, hacemos uso de un *Blend space* que ofrece *Unreal Engine*. Los *Blend spaces* son recursos especiales que se pueden muestrear en los *Anim Graphs* de un *Animation Blueprint* y que permiten combinar animaciones en función de los valores de entradas. En otras palabras, permite transicionar entre dos animaciones, que en este caso serían las de estar acostado estático y en movimiento, según una entrada que se

recibe. La entrada que determinará cómo transicionar de una a otra será la variable de movimiento que maneja el *Blueprint* de animación. Esta variable da un entero con la velocidad actual del personaje. Por lo tanto, si es 0, el *Blend space* entregará la animación de estar acostado estático, pero si el movimiento cambia a algo mayor, la animación transicionará a la de estar acostado en movimiento, que es justamente lo que se esperaría.

Luego, lo único relevante que queda por especificar es las transiciones entre los dos nuevos estados que creamos. Estas transiciones bastarán con ser configuradas en base a las variables "is Prone" para saber si deben entrar a estos estados o no. Dentro del estado "Prone Transition", utilizamos la variable "Getting Up" para discernir si se debe activar la animación de levantarse o la de acostarse antes de pasar al estado "Prone", que manejará finalmente la animación de la movilidad en esta postura.

Figura 3.3: Visualización del personaje acostado en el juego.

3.2.3.3. Mecánica 3: Reposicionamiento de tablas para habilitar caminos

En la implementación del sistema para reposicionar una tabla y abrirse camino a nuevos sectores dentro del mapa, se enfrentó al desafío de establecer una interacción efectiva con un elemento específico del entorno del juego. Para abordar este desafío, se creó un *Actor Blueprint*, dónde entiéndase a *Actor* como la clase padre de la clase *Blueprint*, dedicado a este objeto particular, que en este caso sería la tabla. Durante la creación de este actor, se incluyó un componente *Static Mesh*, seleccionando una tabla adecuada de los recursos disponibles. Los *Static Mesh* se configuraron inicialmente con colisiones activadas, asegurando así la capacidad de colisión con el personaje.

Una vez que la parte visual estaba configurada, se procedió a establecer la interacción del personaje con este nuevo actor en el mapa. En esta fase inicial, se optó por una aproximación que más tarde se revisaría y modificaría en consonancia con el desarrollo de otras mecánicas del juego. En esta etapa, se incorporó un componente de *Capsule Collision* para definir una zona de interacción con la tabla. El *collider* se ajustó para delimitar una distancia de interac-

ción razonable, especificando una altura de 44 unidades de *Unreal Engine* (Equivalente a 44 centímetros) y un radio de 22 unidades (Equivalente a 22 centímetros). Este *collider* tenía la finalidad de identificar cuando el personaje ingresara a la zona de interacción, permitiéndole interactuar con el objeto. Además, se asignó un *tag* ("Collectible") al actor de la tabla para que el personaje pudiera reconocerlo como un objeto con el cual podía utilizar el sistema de interacción.

Para completar la lógica de interacción, se introdujeron comandos específicos en el *Blueprint* del personaje. Cuando el personaje se encuentra dentro de la zona de interacción definida por el collider del actor de la tabla, se configuró el input asociado a la tecla "F", de manera análoga a lo realizado en las otras mecánicas previamente implementadas. Esto permitió que el personaje interactuara y recogiera el objeto. Entre las opciones para representar esta interacción, se consideró la posibilidad de mostrar un icono en pantalla para indicar la adquisición del objeto. Sin embargo, se optó por un enfoque más realista, adjuntando el actor al personaje visualmente para que el jugador pudiera percibir que el personaje llevaba consigo la tabla de manera tangible. Este método se eligió por su capacidad para transmitir de manera efectiva la acción al usuario, proporcionándole una representación visual coherente con la interacción realizada.

Para lograr este efecto, fue necesario crear un *socket* en el esqueleto del personaje, específicamente en el *SKM_Manny_Test*, el esqueleto original del personaje predeterminado de *Unreal Engine*. Un *Socket* es un punto de unión dedicado dentro de la jerarquía de un esqueleto, que puede transformarse en relación con el hueso al que está vinculado. Una vez configurado, se puede adjuntar objetos, armas y otros actores al *Socket*. En este caso, el *Socket* se vinculó a un hueso en la espalda, ya que, en esta etapa de desarrollo, solo interesaba visualmente representar que el personaje llevaba la tabla. Más adelante se reubicó el *Socket* a la mano del personaje, lo cual proporcionó una representación más efectiva de la acción inicialmente buscada.

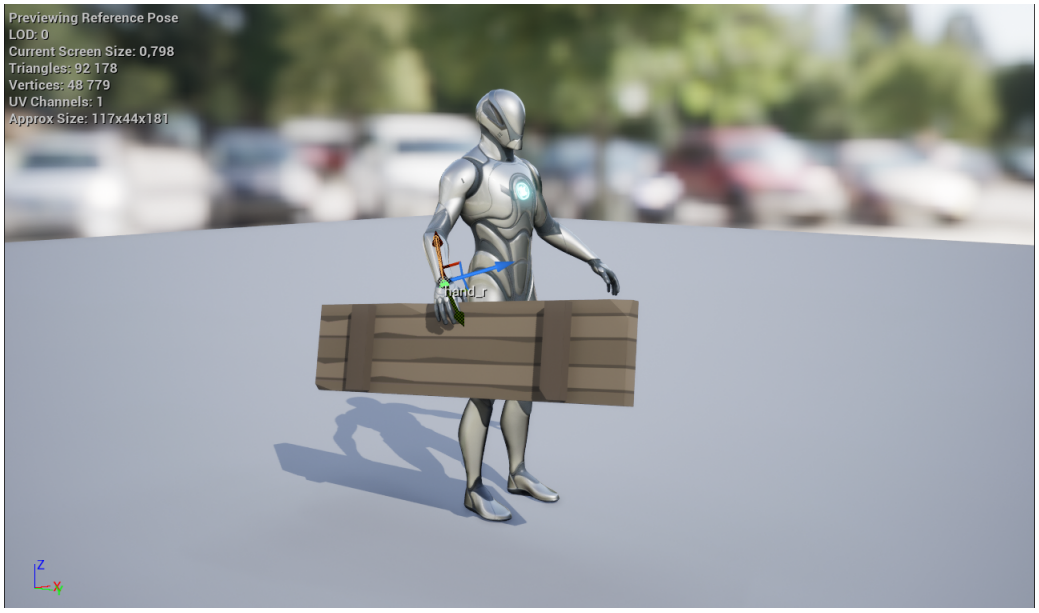


Figura 3.4: Visualización de un recurso adjunto al *Socket* configurado.

Con la entrada del usuario configurada y el *Socket* preparado para adjuntar el actor recolectado, solo restaba programar los comandos para que esta interacción funcionara correctamente.

Para asegurar el buen funcionamiento de esta interacción dentro del juego, se consideraron varias condiciones en cada etapa del proceso. En primer lugar, una vez activado el evento, este verificaría si el *collider* del personaje estaba efectivamente superponiéndose con otro actor y, de ser así, evaluaría si este actor contaba con el *tag* ("Collectible") definido para identificar al actor de la tabla. Si se cumplían estas condiciones, significaba que estábamos dentro de la zona de interacción y la acción de recoger el objeto podía llevarse a cabo. Una de las primeras consideraciones importantes para poder adjuntar la tabla al personaje sin problemas, fue desactivar las colisiones de ese actor, ya que, de lo contrario, al moverse junto con el personaje, las colisiones podrían provocar interacciones no deseadas y efectos adversos en la jugabilidad.

Además, para mantener una referencia a la instancia del actor que se adjuntaría al personaje después de esta acción, se creó una variable "Held Object" de tipo Actor dentro de la clase del personaje. También se implementó otra variable booleana, "Holding Object", que ayuda a evaluar si el jugador ya se encuentra sosteniendo un objeto, evitando así que la interacción se repita y permitiendo que, al activarse nuevamente la entrada para esta acción, se interprete como una indicación para soltar el objeto al mundo.

Con la referencia especificada, examinamos el escenario en el que el personaje no sostenía previamente un objeto. En este caso, se conecta el actor al *Socket* ubicado en el personaje e indicamos que la variable "Holding Object" debe establecerse en verdadero. En cambio, en el escenario en que el personaje ya cuenta con un objeto en su poder, al presionar nuevamente la tecla de interacción "F", el personaje libera el objeto en el mundo de una manera específica. Dado que la intención es posicionar la tabla para crear pasajes, al soltarla, la tabla rota 90 grados en dirección a donde el personaje está mirando. Además, se determinó que al soltarla, debería haber una zona de contacto justo debajo del personaje, al menos 500 unidades (500cm), para reposicionar la tabla en esa área del suelo y evitar resultados inusuales o pérdida del objeto por parte del personaje.

Esta operación se traduce principalmente en la generación de un trazo similar al utilizado cuando el personaje se levanta después de estar acostado, pero esta vez, trazado desde la posición del personaje a 500 unidades hacia abajo. Este trazo busca el primer actor que obstruye su camino, y al encontrarlo, identifica el espacio físico justo debajo del personaje, donde esperamos que la tabla se deje al soltarla. Posteriormente, se toma la ubicación del punto de impacto del trazo y se calcula el vector que indica la dirección en la que el personaje mira en ese momento. Este vector se utiliza para rotar la tabla de manera que, al ser soltada, apunte siempre hacia adelante del jugador. Finalmente, desvinculamos el actor del *Socket*, lo ubicamos en la posición encontrada en el mundo, restablecemos sus colisiones y liberamos la referencia en la variable "Held Object", además de establecer en falso la variable "Holding Object" para indicar que el objeto se soltó.

Como un añadido al sistema de interacción con la tabla, implementamos un pequeño sistema de notificación cuando el personaje está dentro de la zona de interacción, es decir, dentro del *collider* del actor de la tabla. Cuando el personaje está al alcance del objeto, este cambia

a un color amarillo translúcido, iluminando los bordes en amarillo y sugiriendo al jugador que interactúe con el objeto. Esta mejora visual se logró mediante la creación de un nuevo material y la modificación del material de superposición del objeto.

3.2.3.4. Mecánica 4: Movimiento de cajas dentro del mapa.

La mecánica diseñada tiene como objetivo permitir que el personaje pueda empujar objetos para superar obstáculos dentro de los niveles. A diferencia de la interacción directa necesaria con la tabla, en este caso se busca que el jugador pueda identificar objetos que pueden ser movidos para su beneficio sin necesidad de realizar una acción extra más que acercarse y mover en la dirección que quiera al objeto. Por lo tanto, la mecánica se concibió como la habilidad de empujar un objeto para cambiar su posición, más que un nuevo actor con el que se deba activar una interacción para poder realizar un desplazamiento.

Para implementar esto, se creó un nuevo *Actor Blueprint*. En cuanto a la parte visual, se añadió un *Static Mesh* con una caja obtenida de los recursos disponibles. Este sistema se diseñó de manera que la lógica estuviera incorporada en el *Blueprint* de la caja en sí, y no en el del personaje. Por ende, este *Blueprint* se encarga de permitir que el actor se mueva por el mapa y de implementar la lógica necesaria para que el personaje sepa que está empujando algo. A continuación, se explorará cómo se logró que este actor solo se moviera cuando el personaje decidiera empujarlo.

Se decidió limitar el movimiento del actor a lo largo de los ejes X e Y, es decir, los dos ejes que determinan el movimiento horizontal en el plano del suelo en *Unreal Engine*, esto para mantener una lógica sencilla a la hora de empujar objetos. De esta forma se incorporan dos componentes de colisión distintos, uno para cada eje, utilizando *Box Colliders* debido a la forma cúbica de la caja. Estos se dispusieron de manera que sobresaliera un fragmento del *collider* en cada cara de los ejes mencionados, permitiendo que el jugador entre en contacto solo al chocar directamente con una de las caras laterales de la caja.

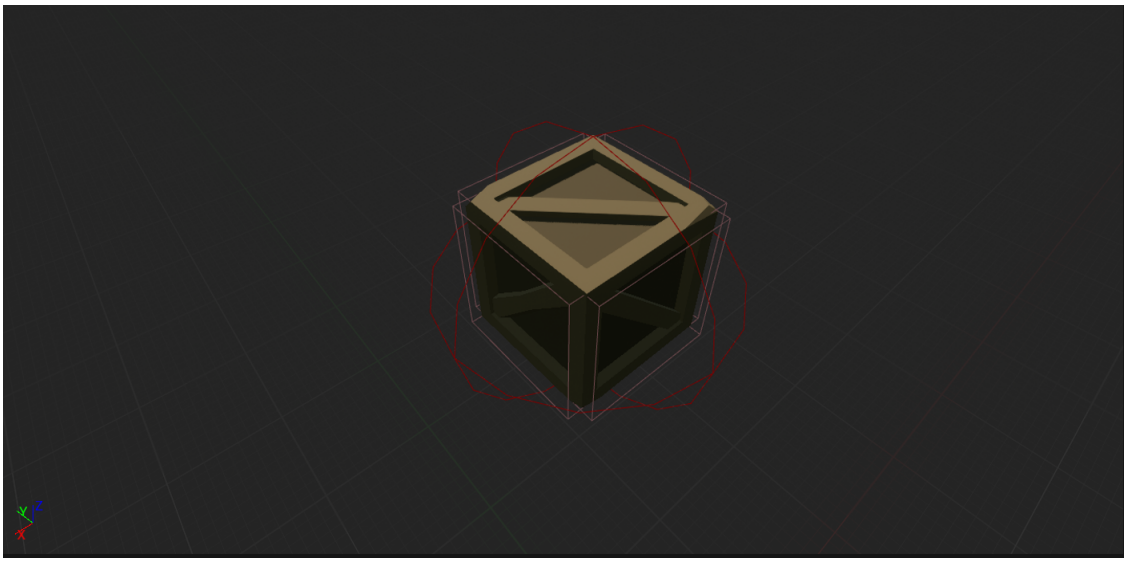


Figura 3.5: Disposición de los *colliders* en el actor “BP_Moveable_Box”.

Al entrar en contacto con el personaje, estos *colliders* activan el sistema de simulación física, inherente a la clase de un actor, en el eje correspondiente, lo que resulta en el movimiento de la caja. Se aprovechan los eventos de superposición de los *colliders*, que se activan al entrar en contacto con otro actor, "On Component Begin Overlap". Es esencial evaluar que el actor en contacto sea el personaje, pues solo queremos que se mueva si es el personaje el que esta tratando de empujar, y se añaden dos condiciones adicionales: se verifica que el personaje no esté en estado agachado o acostado para poder mover la caja. Si se cumplen estas condiciones, una nueva variable definida como "Is Pushing" dentro del personaje se cambia a verdadero, indicando que se encuentra en el estado de empujar, para posteriormente activar la simulación de físicas para el actor y permitir la movilidad.

Para determinar el fin de la interacción con la caja, se utiliza el evento de terminación de superposición proporcionado por los colliders, "On Component End Overlap", que se ejecuta cuando un actor deja de estar en contacto con ese *collider* específico. En este caso, solo es necesario evaluar si es efectivamente el personaje quien está finalizando el contacto. Si es así, se detiene la simulación de físicas en el actor y se restablece la variable de estado "Is Pushing" del personaje a falso.

Estas implementaciones fueron análogas para los *colliders* de los ejes X e Y.

En cuanto al personaje, la única implementación destinada a evidenciar esta mecánica fue la integración de dos animaciones: una para cuando el personaje toca la caja pero no se mueve, denominada "Empujar en Reposo", y otra para cuando está empujando en movimiento, llamada "Empujar en Movimiento". En este punto, la variable de estado mencionada anteriormente, "Is Pushing", cobra importancia, ya que se utiliza para indicar dentro del *Blueprint* del personaje cuándo está ocurriendo esta acción. Luego, para reflejar esto en la animación, se sigue el mismo proceso explicado en las mecánicas de movimiento al agacharse y acostarse. Se crea una variable equivalente en el *Animation Blueprint* vinculado al personaje, y se establecen los estados en la máquina de estados de animación para que estas se activen cuando el personaje esté en contacto con la caja; es importante recordar que la caja gatilla el estado en el personaje.

Finalmente, aprovechando el sistema de aviso de interacción implementado para la tabla, se creó un nuevo material de color azul y se aplicó la misma lógica para la caja. La única diferencia radica en que se determinó que, para este actor, el color translúcido estaría presente en todo momento y no solo cuando el jugador se acercara, invitando así a la interacción independientemente de la distancia entre el actor y el personaje.

Figura 3.6: Visualización de una instancia de un “BP_Moveable_Box” durante ejecución del juego.

3.2.4. Mecánicas de Sigilo

3.2.4.1. Mecánica 5: Recoger sesos para pasar desapercibido

En la planificación de las mecánicas de sigilo y la evaluación de las siguientes por desarrollar, se llegó a la conclusión de que la interacción del jugador con otros actores sería una acción recurrente en el juego. Sin embargo, se observó que el sistema diseñado para tomar la tabla no resultaba lo suficientemente convincente ni escalable para aplicarlo a todas las futuras interacciones. La constante comunicación entre actores, que se complicaría al implementar nuevas interacciones, motivó la creación de un sistema más robusto y versátil, capaz de servir como base para todas las futuras interacciones en el juego.

Previo al desarrollo de la mecánica en sí entonces, se priorizó la creación de un sistema completamente nuevo para la interacción con objetos en el entorno. El nuevo sistema que se planteó, fue seleccionado por sus capacidades para admitir distintos tipos de interacciones y mejorar la usabilidad del juego. La interacción con los objetos ya no depende de ingresar a una zona específica, como sucedía anteriormente. Ahora, cada objeto posee un área de interacción (invisible en el juego) que el jugador puede “apuntar” con la dirección hacia donde la cámara está orientada, controlada por el usuario. De esta manera, si el jugador se encuentra cerca de un objeto pero la cámara está apuntando en otra dirección, la interacción no será posible. Este sistema aporta diversos beneficios, como hacer más intuitivo reconocer con qué objeto se interactúa al apuntar hacia él con la cámara antes de iniciar la acción. Además, elimina problemas de conflictos entre zonas de interacción que podían surgir cuando dos objetos, con el método de interacción anterior, estaban cercanos entre sí. También brinda nuevas opciones para la forma en que se interactúa con los objetos, como la posibilidad de que la acción no sea instantánea, sino que tome un tiempo para completarse. Y por último, crear un sistema especializado para este ámbito del juego permite estandarizar la configuración necesaria para actores que se pretenden hacer interactivos con el personaje.

Para comprender mejor la lógica detrás de este mecanismo, se analizará su funcionamiento.

Este sistema fue configurado como una clase independiente de *Blueprint* llamada "InteractionTrace". El primer evento clave implica la generación de un trazo desde el personaje hasta ciertas unidades frente a él, en la dirección hacia donde la cámara está mirando. Este trazo, que se produce en cada fotograma utilizando el evento "tick" de un *Blueprint Class*, determina la distancia a la cual se podrá interactuar con un objeto.

Este trazo tiene el propósito de encontrar un componente que indique si se encuentra frente a un objeto interactivo o no. Los componentes responsables de esta funcionalidad son lo que llamamos "InteractionAreas". Estos componentes se generan mediante *Blueprint Classes*, específicamente tres: "InteractionArea_Master", que contiene la lógica de interacción y actúa como clase padre de las otras dos, "InteractionArea_Custom_Shape" e "InteractionArea_Box". Estas dos últimas clases están configuradas con colisiones; una utiliza una colisión cúbica o prismática, y la otra ofrece la opción de tomar una forma personalizada. Esto permite que uno decida si desea asignar un área de interacción con una forma específica que se adapte mejor al objeto al que se le está aplicando el sistema.

En este punto, es importante destacar que el "Interaction Trace" debe agregarse como un componente al *Blueprint* del personaje, mientras que las "InteractionAreas" deben incluirse en el *Blueprint* del actor que queremos que sea interactivo. La clase padre, "InteractionArea_Master", cuenta con un evento que se activa si el trazo entra en contacto con una de las "InteractionAreas". Este evento se encarga de notificar al actor en el que se encuentra para que realice las acciones correspondientes al estar enfocado por la cámara del jugador. Además, permite configurar el tiempo de interacción, que determina cuánto tiempo debería durar la interacción antes de que ocurra algo. Si se establece en 0 segundos (el valor predeterminado), la interacción será inmediata; de lo contrario, tomará el tiempo especificado antes de invocar la acción.

Para que la "InteractionArea" notifique al actor en el que se encuentra, creamos una interfaz en forma de *Blueprint Class* que asegura que un conjunto de clases (potencialmente no relacionadas) implemente un conjunto común de funciones. Esto permite definir funciones que deben poseer obligatoriamente los actores que se pretende que sean interactivos. Aprovechando esta interfaz, cuando la "InteractionArea" es enfocada, puede hacer uso de estas funciones, levantando una notificación para indicar que deben ejecutarse. Las funciones de la interfaz son tres: una que entrega la referencia a un *widget* en el actor, otra que crea un contorno (outline) en el actor y, finalmente, una función que se activa en caso de que se inicie una interacción. Sin embargo, las funciones que se activan desde la "InteractionArea" al ser enfocada por la cámara del juego, o mejor dicho, al impactar con el trazo que se genera con este propósito, están relacionadas solo con informar al jugador de que esta acción está ocurriendo efectivamente; la función encargada de realizar la interacción propiamente tal solo debe activarse mediante la intervención del jugador.

Figura 3.7: Sistema de notificación generado por un *InteractionArea* al entrar en contacto con el trazo correspondiente al *InteractionTrace*.

Entendiendo el funcionamiento de las "InteractionAreas", continuamos explicando la lógica del trazado en el componente "InteractionTrace". Si el trazo se encuentra con una de las áreas de interacción, activará el sistema de interacción entregando un booleano para indicar al área de interacción si está siendo enfocada. Del mismo modo, se gestiona el caso en que se deja de enfocar un área de interacción, lo que permite desactivar todas las indicaciones dentro del actor. Esto, sumado a una función llamada "StopInteraction" declarada dentro del "InteractionTrace", se encarga de restablecer todas las variables a su estado original, reiniciando así el sistema para cuando se vuelva a enfocar una nueva "InteractionArea". En caso de no encontrar una "InteractionArea", el sistema simplemente ignora la interacción.

Finalmente, se configuran todos los casos bordes importantes para evitar problemas en el uso del sistema. Entre ellos, se destacan la transición de enfoque de una "InteractionArea" a otra, o cuando se deja de enfocar una "InteractionArea" durante el tiempo que demora en realizarse una interacción. Esto se hace para dar los avisos pertinentes y realizar los llamados necesarios a los eventos dentro de los actores involucrados, lo que asegura que sepan qué acciones tomar.

El último evento relevante es el "TryToInteract" en el componente "InteractionTrace", que se activa cuando el jugador presiona la entrada destinada a generar una interacción. En este evento, se evalúan múltiples condiciones. Primero, siempre se verifica si se está enfocando una "InteractionArea"; de no ser así, se ignora la llamada. Otra condición crucial es identificar si la "InteractionArea" enfocada tiene una duración de interacción. En este punto, se maneja el contador para hacer efectivo ese tiempo antes de llevar a cabo la ejecución de la acción. Si la interacción es inmediata, simplemente se invoca el evento para ejecutar la acción. Sin embargo, si no es inmediata, se gestiona el contador, y además, se involucra al *widget*, del actor en cuestión, para representar visualmente el tiempo de interacción mediante una barra de progreso circular, que se llena según el tiempo que pasa, siendo controlado en esta sección.

En cuanto al personaje, solo es necesario agregarle el *Blueprint* "InteractionTrace" como

un componente y configurar una nueva entrada, como se ha venido haciendo desde el comienzo, para servir como entrada para las interacciones. Se decidió que la tecla "E" sería la designada para este propósito. Por lo tanto, en el *Blueprint* del personaje, solo se configura que al identificar esta entrada por parte del usuario, se llame al evento "TryToInteract", explicado anteriormente, el cual se encargará de la lógica y el manejo del sistema de interacción. Sin embargo, hay una diferencia sustancial: en este caso se le indica a este evento no solo cuándo se presiona la entrada, sino también cuándo se suelta. Esto es esencial, ya que influye en las interacciones que requieren un tiempo predeterminado para llevarse a cabo.

Este nuevo sistema de interacción permite asignar distintas funciones a cada objeto interactivo del juego. Esto se debe a que, al realizar una interacción, el jugador activa una función específica de interacción, la cual puede ser diferente para cada objeto que utilice este sistema. Para que este sistema funcione correctamente, es necesario configurar de cierta manera los actores a los que se les desea agregar la característica de ser interactivos para el jugador. A continuación, se especifican los pasos a realizar en un actor para que sea compatible con el nuevo sistema de interacción:

- **Vincular una "InteractionArea" al actor:** Esta configuración es fundamental para el sistema de interacción. Para agregar este actor llamado "InteractionArea" al nuevo actor que se quiere que sea interactivo, se añade un "ChildActor", lo que permite que el actor principal contenga al actor "InteractionArea" como un componente hijo. Aquí, se puede elegir entre usar una "InteractionArea" de tipo caja o agregar una con una forma personalizada, según el actor seleccionado (ya sea "InteractionArea_box" o "InteractionArea_CustomShape"). Finalmente, se ajusta la posición del área según sea necesario e indica un tiempo de duración en caso de ser necesario; de lo contrario, el valor predeterminado es cero.

- **Agregar un *Widget* al actor:** Es necesario esto, ya que se utilizará para notificar al usuario cuando esté apuntando a la "InteractionArea" y cuando se realice la interacción. Para ello, se creó una clase de tipo *widget* específica destinada a los actores interactivos, configurada para mostrar dinámicamente un indicador visual que invita a utilizar la entrada correspondiente para la interacción, y también utiliza una barra de progreso circular, que representa el progreso de la interacción en caso de no ser inmediata. Para utilizar esta clase, solo es necesario especificar la clase *widget* creada, en el componente *Widget* que agregamos al actor, esto se logra en "Widget Class", y luego se debe desactivar la visibilidad de este componente, ya que el sistema de interacción se encargará de gestionarlo posteriormente.

- **Implementar interfaz en la clase del actor:** Tal como se describió en el desarrollo del sistema, se creó una interfaz específica para los actores que deben ser interactivos. Por lo tanto, es fundamental añadir esta interfaz a la clase. Esto se logra en la configuración de la clase, y se debe agregar la interfaz creada, "BP Interaction", en "Interfaces implementadas". Luego, es necesario configurar las funciones requeridas en la interfaz:

  - **"GetInteractionWidget":** En esta función, solo es necesario indicar la referencia al componente *Widget* añadido al actor.
  - **"InterfaceToggleFocus":** Esta función maneja un contorno que se mostrará alrededor del *mesh* del actor cuando se está enfocando la "InteractionArea". Esto

alerta al jugador de que está apuntando la “InteractionArea” del actor y que la interacción se puede llevar a cabo. Para que esto funcione, es necesario realizar una pequeña configuración previa en el *static mesh*, yendo al apartado de “Rendering” y colocando su “CustomDepth Stencil Value” en 1. Luego, esta función solo se encargará de activar o desactivar el “Render Custom Depth Pass”, lo cual creará el contorno deseado. Es importante destacar que, para que este efecto funcione, debe existir en el nivel un “PostProcessVolume” con un “PostProcessMaterial” creado específicamente para esto, denominado “M_Outline”.

– **“InterfaceInteract”:** Esta función se activa una vez que la interacción se lleva a cabo y puede configurarse de forma independiente para cada actor interactivo según se desee.

Con la implementación de este sistema de interacción, la realización de la mecánica de recoger los sesos se simplificó considerablemente. Con el sistema en funcionamiento, se contaba prácticamente con todo lo necesario para llevar a cabo la acción de “tomar sesos”. La única tarea restante era agregar la lógica específica para este tipo de interacción, que se definió como el jugador evitando ser perseguido por los zombis durante un cierto período de tiempo. Veamos cómo se logró esto.

Para llevar a cabo la interacción, se creó un actor interactivo responsable de permitir la acción. En particular, se diseñó un *Blueprint* llamado “BP_Guts_Interactable”. Este actor se dotó de un *static mesh* que representa a un zombie muerto, para indicar a los jugadores que la interacción de recoger los sesos se obtiene de un zombie fallecido. Luego, se configuró para que fuera un actor interactivo utilizando el nuevo sistema, y se decidió que la interacción debería tomar 1.5 segundos para que no fuera instantánea.

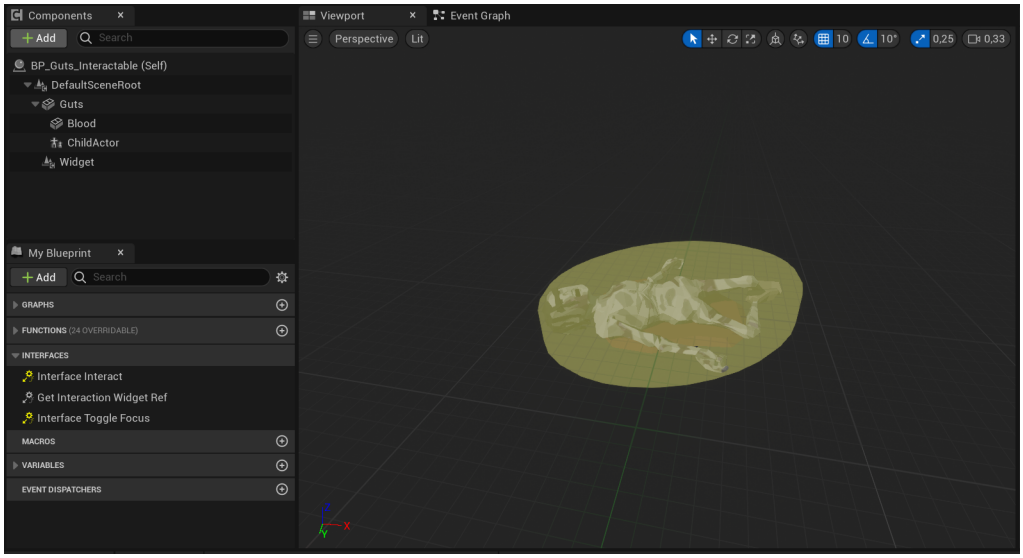


Figura 3.8: *Blueprint* “BP_Guts_Interactable” configurado con el sistema interactivo.

En cuanto a la lógica posterior a la interacción, se llevó a cabo lo siguiente. Dado que, en esta interacción particular, se esperaba que los efectos se reflejaran en el personaje y no en el actor interactivo en sí, se creó una nueva función en el “InteractionTrace”, el componente

encargado de gestionar las interacciones. Esta función, denominada "RunEventAfterInteraction", se ejecuta después de finalizar la interacción, pero también después del llamado al evento dentro del actor interactivo que se activa desde la interfaz. Esta función se generó como una alternativa para la creación de eventos dentro de la clase del personaje después de una interacción. Por lo tanto, al concluir la interacción, se evalúa si corresponde a una interacción con un actor de la clase "BP_Guts_Interactable". En caso afirmativo, se realiza una llamada a "Untreaceable", un evento personalizado que generará los comandos necesarios para ejecutar las acciones planificadas en el personaje después de haberse cubierto con sesos.

Con "Untreaceable", se logra otorgar al personaje la capacidad de no ser perseguido por los zombis. Este evento utiliza una variable adicional "Untreaceable Durability", creada para almacenar el tiempo durante el cual el personaje permanecerá en el estado de no rastreo. Esta variable, de tipo *float*, se definió con una duración de 15 segundos. Posteriormente, se implementa una técnica previamente utilizada en el sistema de notificación de las mecánicas de la tabla y la caja. Para ello, se crea un nuevo material translúcido de tono verdoso que se aplica al *mesh* del personaje como un *Overlay Material*. Esto genera el efecto visual de que el personaje adquiere un tono verdoso en el juego, indicando al jugador que se ha cubierto de sesos y, por lo tanto, no puede ser rastreado.

Para asegurar que el jugador no sea rastreado, se introduce una variable de estado llamada "Is Traceable", que es un booleano. Al activar la acción, se establece esta variable en falso, lo que permitirá posteriormente a los zombis determinar si pueden perseguir al personaje o no. Además del cambio de color en el personaje para notificar la situación, se considera relevante proporcionar otra forma de comunicar esta mecánica a los jugadores, especialmente dada su naturaleza de duración limitada. Se desarrolla un *widget* y un HUD para esta función, marcando así el primer acercamiento a una interfaz para el usuario con el objetivo de mejorar la comprensión de las mecánicas durante la ejecución del juego.

El HUD se añade al *Game Mode* para indicar su uso, y utiliza el *widget* que contiene un icono verde y un contador, ambos ocultos por defecto. La responsabilidad de hacer visibles estos elementos del HUD durante la duración del efecto de los sesos en el personaje recae en el evento "Untreaceable". Además, este evento se encarga de actualizar dinámicamente el tiempo restante del efecto para proporcionar a los usuarios información en tiempo real. Al finalizar el tiempo, se restauran todas las configuraciones a sus estados iniciales: se elimina el *Overlay Material* del *mesh* del jugador, se vuelven a ocultar los elementos en el HUD y se restablece la variable de tiempo. Es importante destacar que si el jugador interactúa con un "BP_Guts_Interactable" antes de que finalice el efecto de una interacción previa, simplemente se reinicia el contador a 15 segundos, evitando así que los jugadores abusen de esta ventaja en el juego.

Es relevante mencionar que, para esta mecánica en particular, no se implementaron animaciones debido a la falta de una animación específica que representara de manera adecuada la acción realizada. Por esta razón, se optó por crear otras formas de notificar y comprender la acción, como las mencionadas anteriormente.

Figura 3.9: Personaje tras haber interactuado con una instancia del actor "BP_Guts_Interactable".

3.2.4.2. Mecánica 6: Lanzar objeto para distraer enemigos

El desarrollo de la mecánica de lanzamiento de objetos se convirtió en el mayor desafío hasta el momento, principalmente debido a la complejidad inherente a la gran cantidad de entidades distintas que debían estar involucradas para que la mecánica funcionara adecuadamente y se representara de manera convincente en el juego. Una opción inicial contemplaba simplificar la implementación de la mecánica, permitiendo que el jugador pudiera lanzar objetos en cualquier momento, asumiendo que el personaje siempre llevaría consigo algún objeto para lanzar. Sin embargo, a pesar de que esta implementación más simple reducía la complejidad del sistema, no resultaba tan atractiva para los jugadores y no se alineaba con la idea de agregar un componente estratégico al juego. La posibilidad de lanzar objetos constantemente sin restricciones podría simplificar demasiado la dinámica del juego.

Debido a estas consideraciones, se descartó la opción inicial y, al analizar lo existente, se identificó que el sistema de interacción ya creado podría ser un punto de partida valioso para desarrollar un sistema más complejo para la mecánica de lanzamiento de objetos. Este enfoque permitiría al personaje interactuar con objetos en el entorno, tomarlos y lanzarlos, ofreciendo así una dimensión adicional a la mecánica y abriendo la puerta a un sistema más amplio de recolección de objetos.

Para integrar esta mecánica con el sistema de interacción existente, se reconoció la necesidad de diferenciar entre los objetos que podían ser recogidos por el personaje y aquellos que simplemente generaban alguna forma de interacción. No obstante, a pesar de las distinciones necesarias entre los tipos de interacción, se buscó mantener ciertas características esenciales, asegurando que la acción de recoger objetos mantuviera la misma mecánica general que la interacción con otros actores: apuntando con la cámara y presionando una tecla según el tiempo de interacción especificado. Este enfoque permitiría una experiencia de juego más coherente y fácil de entender para los jugadores.

La mejor opción fue extender el sistema de interacción para que fuera compatible con

un nuevo sistema especializado en la acción de recoger y soltar objetos en el mundo. Para lograr esto, se introdujo un nuevo concepto denominado "Items", que se asignaría a todos los objetos del mundo que el personaje pudiera recoger y soltar. La lógica detrás de este sistema incluyó la creación de una estructura similar a un inventario, con un solo espacio designado para un "Item" que sería llevado en la mano del personaje. Esta decisión simplificó el sistema y añadió un nivel de dificultad, ya que el jugador debe planificar cuidadosamente qué "Item" deseaba llevar consigo y tomar decisiones cada vez que quisiera soltar o recoger un nuevo objeto.

Con la introducción de los "Items" como un concepto, se resolvió el primer problema al poder diferenciar claramente entre los objetos destinados al sistema de recoger/soltar del personaje y aquellos que generaban otro tipo de interacción en el juego. Para evidenciar esto en el desarrollo, se creó una nueva interfaz llamada "BP_Item", que se describirá con detalle más adelante.

Para abordar el segundo problema y asegurar que los sistemas de interacción y recoger/-soltar objetos estuvieran separados para el usuario, se definió una nueva entrada responsable de activar las acciones de recoger un "Item" o soltarlo si el personaje ya portaba uno previamente. La tecla designada para esta acción fue "F", configurada de manera análoga en el *Blueprint* del personaje al evento del sistema de interacción. Sin embargo, este nuevo evento específico para la acción de recoger/soltar un "Item" en el componente "InteractionTrace" se llamó "Try to Grab", diferenciándolo del evento destinado a interactuar. Se crearon nuevas variables específicas para este sistema, se ajustaron los llamados a las funciones de la interfaz y se incorporó una función equivalente para cuando se detenía la acción de enfoque, denominada "StopGrabAction", la cual restauraría los valores de estas nuevas variables. Aunque la lógica de funcionamiento es similar al otro evento, esta separación permitió que ambos sistemas fueran independientes, pero a la vez aprovecharan la lógica existente del trazo generado desde el personaje y de las "InteractionAreas". La configuración para los "Items" es idéntica a la de un objeto interactivo, al menos en cuanto a las "InteractionAreas" y las notificaciones para el jugador.

Con el funcionamiento del sistema para generar la interacción con los "Items" y tomarlos, es crucial entender cómo el personaje efectivamente recoge un "Item" y cómo reconoce que está portándolo durante el juego. Además, exploraremos cómo este sistema se utilizó para implementar la mecánica de lanzar un objeto.

Ahora que existe la forma de activar la acción de recoger un "Item" y se ha adaptado el sistema de interacción para interactuar con los "Items" de manera independiente, es necesario examinar el evento que ocurre cuando finaliza la interacción para tomar un "Item". Aquí es donde se vuelve a utilizar la función "RunEventAfterInteraction", la cual hay que recordar, también se empleó en la mecánica de los sesos para generar un llamado dentro del mismo personaje. En este caso, se aprovecha la misma función porque la acción de recoger un "Item" también debe ser manejada por el propio personaje. En esta función, se evaluá ahora si el objeto con el que se interactuó posee la interfaz "BP_Item". Si es así, se entiende que se está tratando con un "Item" y, por lo tanto, se invoca el evento encargado de la lógica de recoger y soltar un "Item" dentro del personaje. Este evento se denomina "GrabItemEvent".

Durante el desarrollo de este evento, fue posible percatarse de las similitudes con la mecánica de reposicionar la tabla en el mundo, así que, se reutilizó en gran parte lo desarrollado para aquella mecánica. No obstante, a pesar de mantener la estructura general, surgió un problema que no había sido posible identificar durante la implementación de la mecánica de la tabla, ya que se trabajaba con un solo objeto constante. Al experimentar con nuevos objetos, equivalentes a actores creados para ser "Items", se verificó que la lógica de adjuntar el "Item" a la mano del jugador y soltarlo en el mundo, no era eficaz cuando los tamaños de los objetos variaban, especialmente con el nuevo sistema de interacciones. Surgieron complicaciones al mover la misma instancia del actor entre el mundo y la mano del personaje, y uno de los problemas principales fue la dificultad para desactivar la configuración del sistema de interacción al recoger los actores, algo necesario para que no siguieran invitando a una interacción cuando ya se encontraban en posesión del personaje.

La solución a este desafío implicó una modificación en la forma de gestionar los "Items". Ahora, no bastaba con tener un solo actor que representara al "Item" y reposicionarlo entre el mundo y la mano del personaje. La opción más efectiva fue generar dos versiones de los "Items": una versión para el mundo y otra portátil para cuando el personaje lo lleva consigo. Esta modificación simplificó el uso de los "Items" en el juego y proporcionó ventajas notables, como un control absoluto sobre la lógica en diferentes contextos. Cabe destacar que cada versión del "Item", debe ser una nueva clase *Blueprint* de por si sola.

Con este nuevo planteamiento, el sistema ya no solo manipula la ubicación del actor entre el personaje y el mundo, sino que también gestiona las distintas versiones del "Item" según el contexto. Esto se respaldó mediante la configuración y utilización de la interfaz destinada a los "Items". Los "Items" comparten funciones y se configuran de manera similar a los actores interactivos, pues se mantuvo la forma en que el jugador interactuara con ellos en el mundo, pero estos actores no solo implementan la interfaz de interacción "BP_Interaction", si no que además deben incluir a la interfaz "BP_Items". Esta última, añade características particulares para el sistema de cambio de versiones al recoger y soltar "Items", mediante las funciones "SpawnPickUpVersionRef" y "SpawnWorldVersionRef". Estas funciones tienen como objetivo que cada "Item" maneje el intercambio de versiones de si mismo, es decir, se encargarán de destruirse a si mismo y crear la versión correspondiente, permitiendo una transición fácil en el sistema al realizar llamados a estas funciones. Como sugiere el nombre, una provoca la transición hacía la versión portátil, y la otra, hacía la versión en el mundo.

Al contar ahora con un mecanismo más eficiente para la gestión de objetos destinados a ser recogidos e identificar las limitaciones del sistema anterior utilizado para recoger la tabla, se optó por replantear por completo la mecánica de reposicionamiento de la tabla. La decisión clave fue transformar la tabla en un "Item", permitiendo que la mecánica se alineé con los nuevos mecanismos y adopte características mejoradas en comparación con la implementación inicial. Esta adaptación permitió reutilizar lo desarrollado para la tabla y aprovechar las variables existentes para el nuevo evento destinado a gestionar los "Items".

En el evento "GrabItemEvent" entonces, evaluamos si la variable "Holding Object" está en verdadero o falso. Dependiendo de esto, se ejecutan los comandos correspondientes para soltar o recoger un "Item".

Cuando se recoge, primero llamamos a la función de la interfaz encargada de hacer aparecer su nueva versión portátil y destruir el actor actual correspondiente a la versión del mundo. El nuevo actor generado se guarda en la variable "Held Object" para tener una referencia del "Item" que está portando el jugador. Luego, adjuntamos el actor al *Socket* ubicado en la mano del personaje de la misma manera que se hacia con la tabla y establecemos "Holding Object" en verdadero. Dado que esto ocurre en un breve periodo, la transición entre destruir el objeto y hacer aparecer la nueva versión o la reubicación para que quede adjunto al *Socket* del personaje, pasa desapercibida durante la ejecución del juego.

Cuando ya se porta un "Item", se gatilla la acción de soltarlo, los comandos se ejecutan de la siguiente manera. Reutilizamos la metodología empleada para soltar la tabla, basada en el trazo y los objetos con los que colisionaba. Sin embargo, en este caso, eliminamos la rotación para que los objetos siempre apunten hacia donde el jugador está mirando. Más adelante, se explicará cómo se corrigió para recuperar esta funcionalidad con la tabla en particular. Después de obtener la ubicación en el mundo donde se dejará el "Item", se llama a la función correspondiente para generar el actor con la versión del mundo. Esta función, al igual que su contra parte, primero destruye la versión que el jugador está portando, utilizando la referencia guardada en "Held Object", y luego crea un nuevo actor en la ubicación especificada. Finalmente, se restauran las variables a sus valores predeterminados.

En este punto, se ha implementado con éxito el sistema para recoger y soltar "Items" del personaje, sentando las bases necesarias para la mecánica de lanzamiento de objetos. A continuación, exploraremos cómo se complementó este mecanismo para permitir que el jugador utilice los objetos que lleva consigo, incluyendo la acción de lanzar un objeto.

El sistema de recoger/soltar "Items" constituye un sistema completo en sí mismo y puede funcionar de manera independiente. Para implementar una forma en que el usuario pudiera hacer uso de los objetos que estaba portando, se diseñó un mecanismo adicional que complementara el sistema de obtención de "Items" existente. En este sentido, se reconoció la necesidad de asignar una nueva entrada para que el usuario pudiera hacer uso de los "Items", eligiendo así, el "click izq." del ratón para esta acción. La idea era que esta entrada no debía realizar siempre la misma acción, ya que se espera que cada "Item" pueda tener usos específicos.

La solución propuesta fue aprovechar la interfaz ya establecida para los "Items". Se introdujo una nueva función en la interfaz llamada "Primary Use". La intención era que cada "Item" pudiera definir los comandos específicos que activarían su uso particular. Esta aproximación abría un amplio abanico de posibilidades y resolvía eficazmente el problema de la diversidad de acciones para los distintos "Items". Sin embargo, surgió un pequeño inconveniente: la mayoría de las acciones que se podrían realizar con los "Items" tenían un efecto en el contexto del personaje, y acceder a la clase del personaje desde el "Item" resultaba complicado, especialmente considerando que los "Items" estarían siendo constantemente destruidos y generados por el sistema de recolección. Por ende, la solución más adecuada fue definir la lógica del uso de un "Item" dentro del propio personaje. El "Item" se encargaría únicamente de realizar la llamada correcta al evento que declarara el comportamiento inherente al uso de ese "Item" en particular, todo gestionado por la función "Primary Use".

Así entonces, la implementación del sistema de lanzamiento de objetos implicó la creación de un nuevo actor destinado a cumplir la función de "Item". Tal y como se describió, el evento con la lógica y los comandos para lanzar un objeto se desarrollaron dentro de la clase del propio personaje. La activación de este evento solo es posible a través del uso de un "Item" que tenga una llamada correspondiente desde su función "Primary Use" específica. A continuación, se detalla la configuración necesaria para la creación de un "Item", siguiendo un enfoque estandarizado similar al sistema interactivo previamente desarrollado:

- **Crear un *Blueprint* para la versión del mundo:** Cada "Item" tiene distintas versiones, pero la versión del mundo siempre debe existir, ya que representa la forma en que se presenta dentro del nivel. Esta versión del "Item" se asemeja a un actor interactivo y debe contar con las colisiones adecuadas, implementar la interfaz "BP_Interaction" y "BP_Item". La interfaz de interacción y su configuración le proporcionarán el comportamiento deseado para interactuar con el jugador, mientras que la interfaz de "Item" es necesaria para responder al sistema de recolección y utilización de los "Items". En esta versión, se cambia la clase del *widget* a uno nuevo creado específicamente para "Items" ("W_Grab_Prompt"), que diferencia la tecla correspondiente para la acción de recoger un "Item", en comparación con el *widget* destinado a la interacción con un actor interactivo que especificaba la otra tecla para ese propósito.

- **Crear un *Blueprint* para la versión portátil:** Al igual que la versión del mundo, esta versión también debe existir, ya que se generará cuando el usuario tenga el "Item" en su posesión. Esta versión solo necesita implementar la interfaz "BP_Item". La interfaz de interacción no es necesaria, ya que al estar en posesión del personaje, las características para notificar la interacción no son requeridas.

- **Asignar el mismo *Static Mesh* a todas las versiones**: A pesar de tener diferentes versiones para mayor control contextual, se utiliza el mismo *Static Mesh* en todas las versiones. Aunque las dimensiones pueden variar, se busca que el cambio entre versiones no sea perceptible para los usuarios durante la transición.

- **Configurar las funciones de la interfaz para "Items"**: Para ambas versiones, es necesario configurar las funciones requeridas por la interfaz, específicamente "Spawn Pick Up Version Ref", "Spawn World Version Ref' y 'Primary Use". Las dos primeras funciones son prácticamente análogas; el actor debe destruirse a sí mismo y luego generar un nuevo actor con la versión correspondiente según la función. Para esto, se crean dos variables en cada versión del "Item", que contienen referencias a la clase de la versión de mundo y a la clase de la versión portátil. Con estas variables configuradas, se puede generar el actor correspondiente cuando sea necesario. En cuanto a "Primary Use", solo es necesario especificar el evento correspondiente dentro de la clase del personaje, el cual realiza la acción esperada al utilizar este "Item".

Siguiendo estas configuraciones, se crearon las dos clases iniciales para tener las versiones correspondientes a un "Item" que tuviera como uso el ser lanzado, este en particular sería la representación de un ladrillo. Para lograr esto, solo fue necesario realizar las configuraciones ya especificadas, y realizar el llamado correcto desde la función "Primary Use". En este caso, "Primary Use" se activa cuando se percibe la entrada del usuario destinada a utilizar un "Item"("Click izq."). Tras evaluar que el personaje tenga un "Item" en posesión en ese momento, se activa el "Primary Use" específico para ese "Item" en particular. Para el caso de

los “Items” cuyo uso es ser lanzados, como el ladrillo, este evento es el encargado de llamar “Throw Projectile”, implementado dentro de la clase del personaje.

Sin embargo, notamos que para este tipo de “Items” en particular, los destinados a ser lanzados, no bastaba solo con dos versiones. También fue necesario introducir una versión adicional específica para cuando el “Item” es lanzado. Esto se debe a que tener una versión, que llamaremos “versión proyectil”, permite aprovechar el componente de *Unreal Engine* que proporciona las características de un proyectil al actor que lo contiene. En otras palabras, al crear un actor con este componente llamado “Projectile Movement”, este se generará en el mundo con un impulso como el que se esperaría de un proyectil. De esta forma entonces, se creó esta nueva versión para el “Item”, y cada vez que se genera en el mundo, sale impulsado con la fuerza que se ha configurado.

En particular, el evento “Throw Projectile” se encarga de obtener la referencia de esta versión proyectil del “Item”. Luego, destruye la versión portátil que posee el personaje, para generar un nuevo actor con la versión proyectil justo delante del personaje. Esto da la impresión de que el proyectil está saliendo desde el personaje. Posteriormente en el desarrollo, se mejoró visualmente esta parte, pero se explicará más adelante al introducir otra metodología.

En relación con la versión proyectil del “Item” ladrillo, se configuró con el componente “Projectile Movement” para salir impulsado al ser creado. Además, se le agregaron las dos interfaces correspondientes al sistema de interacción y al sistema de “Items”. Al configurar esta versión de manera similar a la versión del mundo, se logró complementar perfectamente toda la interacción del lanzamiento de un objeto. Cuando la versión proyectil se genera y sale impulsada, una vez que queda estática en algún punto del mundo al perder el impulso, se comporta como un “Item” más, permitiendo que el personaje lo recoja y lo vuelva a lanzar de la misma manera que haría con cualquier otro “Item”. Esto creó un mecanismo auto-contenido que funciona desde el principio hasta el final, dependiendo solo de las configuraciones iniciales y los sistemas desarrollados para tener la mecánica de lanzamientos de objetos funcionando a la perfección.

La última configuración relevante de la versión proyectil de este tipo de “Item”, es aprovechar el evento de impacto del actor, proporcionado por la clase *Actor*, para generar un sonido al primer impacto dentro del mapa. Este sonido será útil para la última parte de la mecánica, que consiste en atraer a los enemigos al lugar donde impacta el objeto lanzado.

Se crearon dos “Items” para ser utilizados en esta mecánica, lo que se traduce en seis *Blueprints* en total, con tres versiones distintas para cada “Item”. Uno vendría siendo el ladrillo ya mencionado, al cual se le agregó un sonido estándar de un objeto golpeando el suelo. Este objeto, al ser lanzado, queda en el mundo para ser recogido nuevamente. El otro, corresponde a una botella de vidrio, a la cual se le agregó un sonido especial de vidrio rompiéndose para cuando fuera lanzada e impactara en el mundo. Además, se decidió que este “Item” en particular se destruiría después del impacto, para simular el hecho de que una botella de vidrio se rompe y no debería poder recogerse de nuevo. Esto agrega variabilidad a la mecánica, haciendo que el jugador deba discernir cómo utilizar el objeto lanzado dependiendo de si es un ladrillo o una botella.

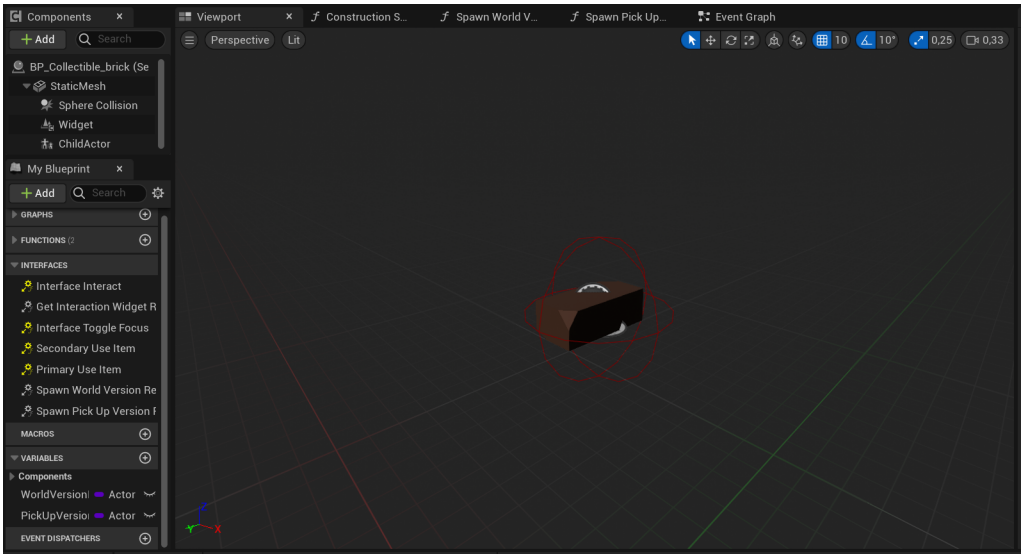


Figura 3.10: *Blueprint* correspondiente a la versión de mundo del ladrillo.

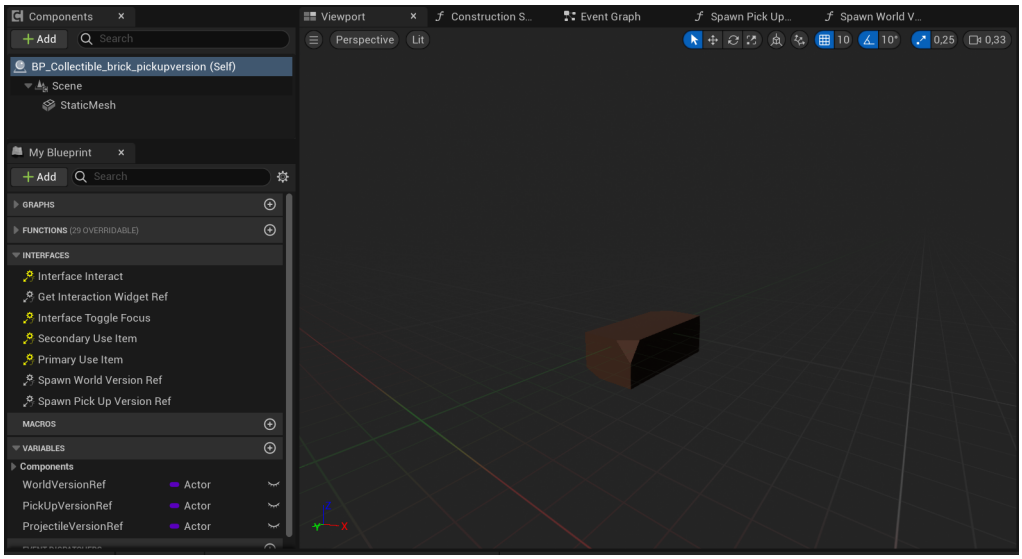


Figura 3.11: *Blueprint* correspondiente a la versión portátil del ladrillo.

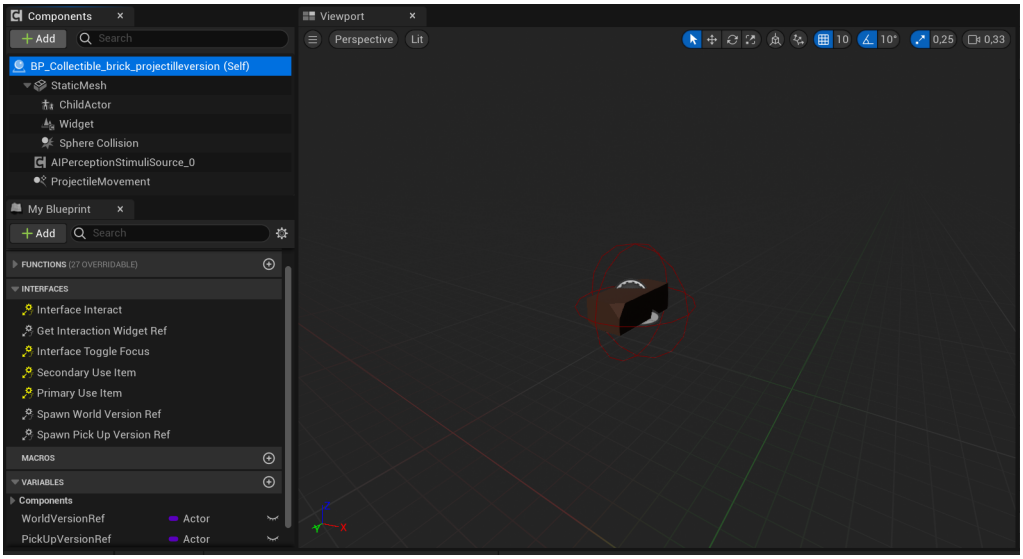


Figura 3.12: *Blueprint* correspondiente a la versión proyectil del ladrillo.

Por último, en cuanto al sistema de "Items", se recuperó la función de la tabla. Recordando que fue adaptada para que ahora fuera un "Item", se crearon las dos versiones correspondientes al mundo y portátil. Se configuró un nuevo evento específico en la clase del personaje destinado a la colocación de la tabla apuntando en la dirección hacia la cual mira el personaje, "DropObjectFacingFoward". Estos comandos se reutilizaron de la implementación anterior, ya que se quería lograr exactamente lo mismo que antes. Posteriormente, el evento se configuró para que fuera activado por el "Primary Use" de la tabla. De esta manera, la colocación de la tabla quedó implementada de tal forma que funcionara como cualquier otro "Item", recuperando todas las características originales que se tenían planeadas para esta mecánica en particular.

La implementación del sistema de lanzamiento de objetos ya estaba completa en este punto del desarrollo; sin embargo, faltaba una característica fundamental de la mecánica de juego propuesta inicialmente: la capacidad de distraer a los enemigos al lanzar objetos. Por lo tanto, el siguiente paso fue configurar a un enemigo para que reaccionara a los objetos lanzados y así hacer evidente el funcionamiento de esta mecánica. De lo contrario, solo habríamos creado una forma de lanzar objetos por el mundo que no se alinearía con el contexto del videojuego en desarrollo.

En consecuencia, se procedió a implementar un zombi que serviría como enemigo dentro del contexto de este proyecto. Este actor en particular desempeñaría un papel crucial para complementar el uso de las mecánicas de juego desarrolladas, especialmente durante las pruebas finales con usuarios a las que se sometieron estas mecánicas de juego.

Para implementar al zombie, se generó un nuevo *Blueprint* de clase "Character", llamado "BP_SintyApocalypse_Zombie_test". Es importante recalcar que esta es una clase hija de la clase "Pawn". En *Unreal*, un *Pawn* se comprende como la clase base de todos los actores que pueden ser poseídos por jugadores o IA. Estos actores son las representaciones físicas de jugadores y criaturas en un nivel. Se configuró al zombie de manera similar al personaje en su estado inicial, agregándole un *Skeletal Mesh* y una *Capsule Collision*. Para el apartado

visual, se utilizó un *retarget* análogo al utilizado en el personaje, pero esta vez se agregó un "skeletal mesh" desde los recursos del proyecto correspondiente a un zombi.

Para las animaciones, se vinculó un nuevo *Animation Blueprint* específico para este *Pawn*, pero mucho más simple que el del personaje. Dado que para el zombi solo se esperaba utilizar una animación para el movimiento, el *Animation Blueprint* solo obtiene esta data desde la clase del zombi. Utiliza esta información para alimentar la entrada de un *Blend Space*, que fue lo mismo que se utilizó en la mecánica de moverse acostado. Según la velocidad que perciba del zombi, el *Blend Space*, que se configuró con animaciones dedicadas para el desplazamiento del zombie, irá discerniendo entre las animaciones de estar estático, caminar o correr.

Con estas configuraciones iniciales, ya tenemos a un zombi que puede estar en el mundo y moverse sin problemas. Sin embargo, este *Pawn* en particular no será controlado por un jugador, por lo que se tiene que configurar de tal forma que sea controlado por la IA (Inteligencia Artificial) dentro del juego. Para lograr esto, *Unreal Engine* tiene un sistema de IA implementado para encargarse del control de este tipo de *Pawns* dentro de un juego. Dado que en el alcance de este proyecto no estaba considerado generar un enemigo tan complejo, y solo se buscaba tener un enemigo que ayudara a complementar el uso de las mecánicas, se decidió aprovechar este sistema y generar un comportamiento simple para el zombi que fuera útil en este contexto.

En esta versión de *Unreal Engine*, contamos con dos sistemas, *PawnSensing* y *AIPerception*, diseñados para dotar a los *Pawns* de capacidades sensoriales y reactivas. Aunque *PawnSensing* fue introducido antes y ofrece una funcionalidad más limitada, ambos sistemas comparten el propósito esencial de permitir que los *Pawns* perciban su entorno y respondan a estímulos específicos.

Ambos sistemas se describen como tipos de componentes que pueden añadirse al *Blueprint* de un *AIController* desde la ventana de Componentes. Estos componentes se encargan de definir qué sentidos activar, los parámetros asociados a dichos sentidos y cómo deben responder ante la detección de un estímulo. Además, ofrecen diversas funciones que permiten obtener información sobre qué se ha percibido, qué actores se han detectado, e incluso deshabilitar o habilitar un tipo particular de percepción.

En este caso, aprovechamos las características particulares de cada sistema para lograr el comportamiento deseado en el zombi. Aunque no se requieren comportamientos complejos, ambas opciones brindaron las herramientas necesarias para que el enemigo pudiera percibir al jugador y reaccionar en consecuencia.

Así, estos componentes fueron agregados a la clase del zombi y configurados para que este tuviera los siguientes sentidos: visión, para detectar visualmente al personaje dentro de cierto rango, y audición, para reconocer sonidos que se generen a su alrededor dentro de cierto radio. Estos sentidos eran necesarios para lograr la respuesta de los zombis a las acciones que el personaje realizara con las mecánicas de juego desarrolladas. La idea es que el zombi pueda escuchar un ruido e ir a ese lugar, o bien perseguir al personaje si lo llega a descubrir. Finalmente, para que este personaje pueda ser controlado por la IA, debemos crear un *AIController* y especificar en la clase del zombi que este será controlado por este componente

en “AI Controller Class”.

Para lograr las reacciones deseadas en el zombi, se utilizan los eventos que ofrecen los componentes mencionados y el *AIControlller*. En primer lugar, utilizamos el evento ’Begin Play’ para que cuando inicie el juego, el zombie obtenga una referencia del jugador en juego y la guarde, ya que se utilizará recurrentemente en las acciones del zombie. A continuación, desarrollamos los eventos con la lógica para dar el comportamiento deseado al zombi.

Para activar la persecución del zombie hacia el jugador, se utiliza el evento “On See Pawn” proporcionado por el componente *Pawn Sensing*. Este evento se desencadena cuando el zombie detecta un *Pawn* en su campo de visión, evaluando si dicho *Pawn* corresponde al jugador. En caso afirmativo, se invoca un nuevo evento denominado “Follow Player”, crucial no solo por dirigir al zombie hacia el jugador, sino también por implementar la lógica necesaria para la interacción con la mecánica de recoger sesos, mecánica que debemos recordar tiene como objetivo evitar la detección por parte de los zombis. Se evalúa si el jugador permanece dentro del rango de visión del zombi, y en caso de perderlo de vista, cesa la persecución. Si el zombi sigue visualizando al jugador, se verifica el estado actual de rastreabilidad del personaje mediante la variable específica “is Traceable”. Si el estado es verdadero, se permite el desplazamiento del zombi hacia el jugador utilizando la función “AI Move To” proporcionada por el *AIControlller* que controla este *Pawn*, especificando el jugador como objetivo. Si el estado es falso, indicando que el jugador está actualmente cubierto por sesos, se ignora y el zombie no lo perseguirá, a pesar de haberlo detectado. Es esencial destacar que para que el sistema de IA pueda controlar y determinar los movimientos del *Pawn* zombi, se debe añadir un objeto especial llamado “Nav Mesh Bounds Volume” al nivel. Este objeto mapea el nivel y determina los caminos por los cuales la IA puede mover al *Pawn*. Si no se configura este elemento, el *AIControlller* no sabrá por dónde mover al zombie.

Para detectar sonidos y dirigirse hacia su origen, se utiliza el evento “On Target Perception Updated” proporcionado por el componente *AIPerception*. Este evento proporciona el actor el cual generó el sonido y el estímulo percibido por el sistema. Se evalúa si el sonido fue generado por el jugador. En caso afirmativo, se crea un nuevo estado para el zombi con una variable booleana llamada “Investigating?” que se establece en verdadero. Posteriormente, se desplaza el *Pawn* hacia el origen del sonido utilizando la misma función empleada para perseguir al jugador, esta vez con el origen del sonido como objetivo. Finalmente, al llegar al lugar y tras esperar un segundo, se restablece a falso la variable de estado para indicar que el zombi ha dejado de investigar ese sonido. Con este comportamiento implementado, sólo hace falta integrarle un componente extra a los “items” que generen sonidos llamado “AIPerceptionStimuliSourceComponent”, lo cual permite que se pueda aprovechar el sonido generado por los objetos lanzados para que los zombis perciban ese estímulo y se dirijan hacia el lugar de origen del ruido. De esta manera, se logra la distracción deseada mediante la mecánica, y los zombis reaccionan adecuadamente, añadiendo esta nueva dimensión que estaba pendiente en el uso del lanzamiento de objetos.

Con estos dos comportamientos desarrollados para el zombi, se complementan las dos mecánicas de juego relacionadas con el sigilo implementadas. Además, se introduce este nuevo actor que representa a los zombis. La creación de este tipo de enemigos no solo es fundamental para dar sentido al uso de las mecánicas, sino que también crea una variedad de posibilidades

para generar desafíos necesarios en las pruebas con los usuarios.

3.2.5. Mecánicas de Combate

3.2.5.1. Mecánica 7: Ataque cuerpo a cuerpo

Para desarrollar las mecánicas de combate, inicialmente se enfocó en el combate cuerpo a cuerpo. Esto implicaba permitir que el jugador pudiera realizar un golpe, y que los enemigos (en este caso, los zombis) pudieran recibir los golpes, quedando eventualmente fuera de combate al morir. Además, esta interacción debía ser bidireccional, ya que en un combate ambas partes deben tener la capacidad de atacar y recibir daño.

En este punto del desarrollo, ninguno de los dos actores principales, el personaje y los zombis, contaba con estas capacidades. Por lo tanto, el primer paso para dar forma al sistema de combate fue crear estas características para ambas clases.

Para establecer un sistema de vida tanto para el personaje como para los zombis, se implementó una estructura similar. A continuación, se detallará el sistema base, señalando las variaciones específicas para los zombis.

El sistema de vida se diseñó utilizando una variable entera para representar la cantidad de vida en tiempo de ejecución del juego. Además, se incorporó otra variable que almacenaba la vida máxima del actor, sirviendo como referencia constante para operaciones como la recuperación de vida y para mostrar la cantidad total de vida en la interfaz del jugador.

La lógica del sistema incluía la identificación de daño al actor a través del componente *Capsule Collision* presente en el personaje y los zombis. Se aprovechó el evento "Any Damage" para obtener la cantidad de daño recibido y la referencia del actor que infligió el daño. Si la resta de la vida actual después del daño era mayor a cero, se aplica el daño, se activa una animación simulando el recibir un golpe, esto mediante un *Anim Montage*, y se genera un efecto de lanzamiento hacia atrás para simular el impacto.

En el escenario donde la vida alcanzaba cero, se ejecutan lógicas distintas para el personaje y los zombis. Para el personaje, se introdujo una variable booleana "is Dead" que se integró en todas las instancias del código para filtrar las acciones del jugador una vez que estaba muerto, evitando movimientos o acciones adicionales. Posteriormente también se activa una animación mediante un *Anim Montage*, pero esta vez con una animación especial para cuando el jugador muere. En el caso de los zombis, se desactiva la física del *Skeletal Mesh* para lograr un efecto visual peculiar que fue mejor opción que una animación como en el caso del personaje, y se integra con la mecánica de recolección de sesos. Al morir, tras un breve periodo de tiempo el zombie genera un actor de la clase "BP_Guts_Interactable" en el lugar, para simular que el zombie muerto ahora ofrece sesos para que el jugador pueda interactuar, conectando así el combate con la mecánica de sigilo.

La última adición al sistema de vida fue la incorporación de una barra de vida en el *widget* del HUD, brindando al usuario información en tiempo real sobre la salud del personaje durante la ejecución del juego. Esta funcionalidad se implementó exclusivamente para el personaje

principal. La barra de vida se actualiza dinámicamente cada vez que el personaje sufre daño, aprovechando las variables predefinidas que almacenan la vida máxima y actual del personaje.

Una vez implementado el sistema de vida para los actores involucrados en el combate cuerpo a cuerpo, se procedió a desarrollar la lógica de los ataques por parte del personaje hacia los enemigos. La implementación de los golpes del personaje difiere significativamente de los golpes generados por los zombis, ya que los primeros son activados por el jugador, mientras que los segundos son auto-generados según el contexto del juego.

En el caso del personaje, dado que todos los "Items" pueden tener funcionalidades distintas y el personaje se encarga de tomarlos y activarlos, se optó por permitir que los ataques del personaje fueran determinados por "Items" específicos que generaran esta acción. En este sentido, se crearon armas, "Items" específicos diseñados tanto para el combate cuerpo a cuerpo como para el combate a distancia más adelante. Para activar un ataque, el jugador simplemente debe hacer uso del "Item" cuando se encuentre en posesión de este.

Para lograr esta compatibilidad entre los sistemas de "Items" y ataques, se creó un "Item" específico que representa un arma cuerpo a cuerpo, un cuchillo. Este "Item" desencadena un evento contenido en el *Blueprint* del personaje, siguiendo la misma lógica que se utilizaba para los "Items" anteriores. A continuación, se explicará cómo se generó este evento, que finalmente contiene el funcionamiento de un golpe por parte del personaje.

Se inicia creando una variable booleana de estado denominada "Attacking", la cual indica si el personaje está llevando a cabo la acción de atacar. Esta variable se implementa con el propósito de filtrar la activación del evento de ataque cuerpo a cuerpo, evitando así que un jugador pueda ejecutar repetidamente esta acción, lo cual afectaría la jugabilidad del combate. Si la variable "Attacking" está en falso, se activa a verdadero para indicar el inicio de un ataque por parte del jugador. En caso contrario, si la variable ya estaba en verdadero, simplemente se ignora la llamada a la función.

Posteriormente, se verifica si el jugador está portando un "Item" de tipo arma cuerpo a cuerpo. Para lograr esto, se utiliza una nueva interfaz denominada "BPI_MeleeWeapon", creada por la necesidad de que las armas cuerpo a cuerpo compartieran ciertas funciones especiales. Si el objeto guardado en la variable "Held Object" implementa esta interfaz, se procede a implementar la lógica para realizar el ataque. En caso contrario, se omite la llamada, evitando la ejecución de un golpe.

Una vez asegurado el cumplimiento de estas dos condiciones, se permite efectuar un golpe mediante el aprovechamiento del sistema de *Anim Montage*.

El sistema de *Anim Montage* ofrece la capacidad de organizar secuencias de animaciones combinando múltiples movimientos, permitiendo un control detallado mediante *Blueprints*. Una ventaja destacada es que no es necesario manipular la máquina de estados en el *Animation Blueprint* del personaje, como sería en otros casos. Para acciones breves, deseamos que las animaciones se ejecuten exclusivamente para representar la acción. No obstante, igual fue necesario configurar en el *Animation Blueprint* del personaje el reconocimiento de la ejecución de estos *Anim Montages* y establecer que afectaran solo al tronco superior del esqueleto

del personaje.

Esta configuración específica se implementó para evitar interferencias entre los *Anim Montages* y las animaciones de movimiento asociadas a la máquina de estados del *Animation Blueprint*. Al limitar la ejecución de los *Anim Montages* desde el tronco hacia arriba cuando ya se está ejecutando otra animación de movimiento, se evitan interferencias en las piernas, donde estas interferencias podrían afectar negativamente la visualización del juego. Este enfoque asegura que, si el jugador está en movimiento, pueda ejecutar una animación con el *Anim Montage* mientras las piernas continúan mostrando la animación de movimiento correspondiente. Así, el tronco superior ejecutará los movimientos asociados a la otra animación, como por ejemplo, un ataque, mientras que el *Animation Blueprint* gestiona la animación de correr al mismo tiempo para las piernas, generando mejoras visuales y de jugabilidad. Este mismo principio se replicó en el *Animation Blueprint* del zombi para lograr efectos similares.

En la implementación del combate cuerpo a cuerpo, más específicamente cuando el personaje realiza un golpe, se emplea un *Anim Montage* que permite establecer notificaciones en momentos específicos de la animación. Dentro de este *Anim Montage*, se utiliza una notificación para llamar un evento encargado de activar un trazo esférico alrededor del arma, el cual será el responsable de identificar colisiones con actores susceptibles de recibir daño. Existen dos tipos de notificaciones: "Anim Notify States", que generan notificaciones al inicio y al final de un período específico, y "Anim Notify" simples, que las generan en momentos precisos.

En este caso, se implementa un "Anim Notify State" dentro del *Anim Montage* para crear la lógica del golpe. Al iniciar el montaje de animación, la notificación indica al evento correspondiente, cuándo comenzar y finalizar el trazo del golpe. Finalmente, una vez concluido el montaje de la animación, se establece la variable "attacking" a falso, indicando que el golpe ha finalizado. Este enfoque coordinado permite ejecutar la lógica del ataque cuerpo a cuerpo de manera efectiva.

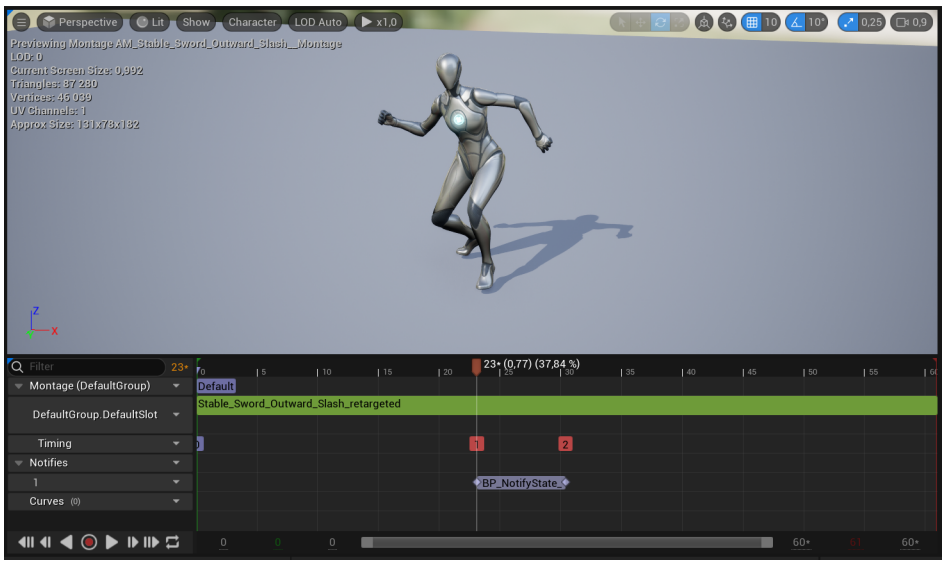


Figura 3.13: Momento que se levanta la notificación en el *Anim Montage* para un ataque cuerpo a cuerpo.

Este “evento” que provoca el daño en otro actor, realmente se compone de tres eventos interrelacionados: “LoopMeleeDamageTrace”, “StartMeleeDamageTrace” y “StopMeleeDamageTrace”. El evento “StartMeleeDamageTrace” inicia un bucle mediante un temporizador y genera un sonido en la ubicación del personaje, simulando el golpe de un objeto cuerpo a cuerpo. Este sonido es reportado para que otros actores en el mundo puedan percibirlo, permitiendo que, por ejemplo, un zombie reconozca el ataque del personaje. Este evento es el encargado de iniciar un golpe, y es el evento que se ejecuta con la notificación desde el *Anim Montage*.

En el evento principal, “LoopMeleeDamageTrace”, se utiliza una de las funciones definidas en la interfaz “BPI_MeleeWeapon”, para obtener los puntos inicial y final del arma para poder generar un trazo esférico con respecto a estas posiciones en el mundo de juego. Es decir, cada “Item” tipo arma cuerpo a cuerpo implementará esta función que entrega la ubicación espacial de los puntos iniciales y finales del *Static Mesh* del arma. Esto permite que cada arma defina su propio trazo esférico acorde a su tamaño y contextura. Utilizando la función “Sphere Trace by Channel” con un radio específico, se genera el trazo esférico. Si el trazo impacta con un objeto y este objeto tiene el *tag* “damageable”, que se implementa justamente para identificar actores dentro del juego susceptibles a recibir daño, se procede a aplicar el daño correspondiente al actor afectado. Se implementa una protección “Do once” para garantizar que el daño se aplique una sola vez al mismo actor durante el ciclo del evento, recordemos que es un bucle y podría impactar con el mismo objeto en varias iteraciones del mismo bucle.

El último evento, “StopMeleeDamageTrace”, se encarga de finalizar el temporizador iniciado en el evento inicial y con esto el bucle que se encontraba en ejecución. En el *Anim Montage*, este evento se llama al final de la notificación correspondiente, permitiendo que el ciclo del trazo esférico funcione adecuadamente.

Tras verificar la efectividad del sistema implementado para el llamado de eventos durante un *Anim Montage*, al proporcionar un mayor control sobre las animaciones específicas para ciertas acciones. Esta implementación se replicó en la mecánica de lanzamiento de objetos, reemplazando la salida frontal del objeto lanzado, por un *Anim Montage* y una notificación que controla el momento exacto de generación del proyectil. Además, se ajustó la referencia del punto de generación a la mano del personaje, mejorando visualmente la mecánica y haciéndola más intuitiva para los jugadores.

En el caso de los zombis, se utilizó la misma metodología de eventos y *Anim Montages*, pero se agregaron animaciones específicas para los ataques de los zombis. La principal diferencia radica en la forma en que se generan los ataques. En los zombis, la implementación fue más directa gracias al comportamiento predefinido de estos y las opciones proporcionadas por el *IAController* que los maneja. Para lograr los ataques, se agregó un comando al evento “Follow Player”, detallado anteriormente, para lograr que cuando el zombi se acerca lo suficiente al jugador, desencadene el sistema de ataque guiado por el *Anim Montage*. En el juego, esto se traduce a que si un zombie se aproxima lo suficiente al jugador, realizará un ataque y continuará haciéndolo mientras permanezca dentro del alcance. Cabe destacar que se configuraron los brazos del zombie como el punto de referencia para generar los trazos esféricos en este escenario, ya que los zombis no cuentan con armas como el personaje.

3.2.5.2. Mecánica 8: Ataque a distancia

En el sistema de combate a distancia, la premisa fundamental sigue siendo que el personaje debe poseer un "Item", en este caso, un arma a distancia, para desencadenar eventos de ataque o generar disparos. Este sistema presenta una mayor complejidad en comparación con los desarrollados previamente, ya que implica una combinación de eventos que deben coordinarse para garantizar el correcto funcionamiento de la mecánica en la jugabilidad del juego. A continuación, se explicará la lógica general implementada en esta mecánica, seguida de una descripción detallada de cada uno de los eventos que constituyen este intrincado sistema.

En un juego de tercera persona, la intuición podría sugerir seguir el enfoque convencional de permitir al jugador apuntar su arma, cambiando la cámara a la mira del arma. Sin embargo, en este proyecto específico, se optó por un enfoque diferente con el objetivo de hacer la jugabilidad más atractiva y explorar nuevas posibilidades. Este enfoque se basa en tres puntos clave: primero, se prescinde de la acción de apuntar en el combate a distancia; segundo, se establece toda la lógica en función de un radio alrededor del personaje que determina el alcance de los objetivos que el personaje puede atacar con un arma a distancia; y finalmente, la detección de enemigos se lleva a cabo mediante un sistema de enfocar objetivos dentro del radio. En resumen, cuando el personaje tiene un arma a distancia, se le asigna un radio específico, y el jugador solo puede detectar objetivos dentro de este alcance. Una vez que se identifica un objetivo, el jugador puede disparar, y el personaje automáticamente apuntará y disparará en la dirección del enemigo seleccionado como objetivo.

En el desarrollo de esta mecánica, se consideró esencial mantener la lógica de que el personaje debía poseer un arma a distancia para desencadenar un ataque, concretamente, producir el lanzamiento de un proyectil desde dicha arma. Si bien, se utilizó una estructura similar a la del combate cuerpo a cuerpo, se implementó "BPI_DistanceWeapon", una interfaz específica para armas a distancia debido a las diferencias fundamentales entre ambos sistemas. Lo primero que se hizo entonces, fue crear un ítem que representara una pistola con esta nueva interfaz.

Luego, en contraste con el combate cuerpo a cuerpo, se reconoció que el nuevo sistema requería que el personaje llevara a cabo dos acciones distintas: disparar y buscar un objetivo dentro del radio de alcance del personaje. La premisa principal consistía en otorgar al jugador la capacidad de realizar un disparo, pero también de buscar un objetivo. Esta última acción podría ser solicitada de manera repetida, permitiendo al jugador cambiar el objetivo entre las opciones disponibles en el radio de alcance. Y aquí fue donde surgió un problema: el sistema solo era capaz de activar una acción a la vez. Esto planteaba el dilema de permitir que el jugador disparara al enemigo más cercano o solo se le permitiera buscar un objetivo. Para que este sistema fuera efectivo, era necesario posibilitar el uso complementario de ambas acciones o permitir que cada una se utilizara de manera independiente.

Para resolver este desafío, se identificó la posibilidad de aprovechar la lógica ya utilizada en el sistema de "Items" que gestiona las acciones. Dado que ya se había segregado las armas cuerpo a cuerpo de las armas a distancia, la solución más adecuada consistió en dotar a estas

últimas no solo de la capacidad para desencadenar una acción, si no que fueran capaces de activar dos acciones. Surgió así la idea de incorporar una acción secundaria en el uso de los "Items".

Ahora, si bien solo los "Items" del tipo "arma a distancia" serían capaces de activar dos acciones, se requirió reconfigurar el sistema de "Items" para que todos pudieran tener tanto una acción primaria como una acción secundaria. Esta adaptación, lejos de presentar inconvenientes, se percibió como un beneficio. Se considera que esta modificación enriquece el sistema de "Items" existente. En situaciones en las que se deseara dotar a un sistema anterior o a nuevos sistemas, basados en el uso de "Items", con dos acciones, bastaría con generar el evento correspondiente a la acción secundaria. Actualmente, si un "Item" no posee una acción secundaria, simplemente se omite su especificación, indicando al jugador que dicho "Item" carece de tal característica.

Para implementar esta funcionalidad, se introdujo una nueva entrada mediante la asignación de la tecla "click der.". Además, se agregó una función adicional en la interfaz de "Items", "Secondary Use", requisito ahora indispensable para todos los "Items". En términos generales, el proceso se asemeja al desarrollado en el sistema de combate cuerpo a cuerpo. Es decir, si el personaje empuña un "Item" del tipo arma a distancia y activa la acción primaria, se desencadenará un evento encargado del lanzamiento del proyectil. Este evento, especificado en el gráfico de eventos del personaje, opera de manera similar al sistema anterior. Por otro lado, al activar la acción secundaria, se invoca otro evento destinado a buscar un objetivo dentro del radio del personaje. Es importante destacar que estos eventos representan solo una parte de un conjunto interconectado de eventos que constituye el sistema de combate a distancia. A continuación, exploraremos cada uno de estos eventos para comprender cómo interactúan entre sí.

Se analiza el evento "Distance Attack", encargado de gestionar la lógica cuando se realiza un disparo de proyectil. Al igual que en el combate cuerpo a cuerpo, se utiliza la variable "Attacking" para filtrar la generación de múltiples disparos antes de que finalice el anterior. Para este evento en particular, se crearon dos entradas: uno referente a la clase de proyectil a lanzar y otro a un componente flecha que indica la posición desde la cual debería salir el proyectil en el mundo. Esta lógica es similar a la utilizada con el punto inicial y final de un arma cuerpo a cuerpo. El "Item" del arma a distancia será el encargado de entregar lo necesario a través de estas entradas del evento, justamente cuando se haga el llamado a "Distance Attack" desde la misma arma. Estos datos se almacenan en variables del personaje, ya que serán necesarios en otros eventos relacionados con el combate a distancia.

Posteriormente, se introduce una nueva variable llamada "Current target damageable", que sirve para determinar si el personaje tiene un objetivo dentro del radio. Si bien este evento no será el encargado de configurar lo que se almacena en esta variable, es crucial esta variable en el sistema, y en particular en esta parte, para verificar la existencia de un objetivo al cual disparar. En caso de que dicho objetivo no exista, se realiza una llamada al evento "Focus Aim For Distance Attack" encargado de encontrar un objetivo, y luego continúa la ejecución del evento actual. Si ya existe un objetivo, se utiliza esa referencia para proseguir con la ejecución. En este punto del evento, al asegurar la existencia de un objetivo, se obtiene su posición y luego se rota al personaje en dirección al objetivo actual.

Después de completar las acciones previas al disparo, aún quedan comandos destinados a mejorar la visualización de esta mecánica para el jugador. Lo siguiente aborda lo que debería ocurrir a nivel de interfaz de usuario para indicar que se está realizando un disparo. Para lograr esto, se accede a la referencia del HUD y se actualizan los elementos específicos creados para esta función: un contador y una imagen con el símbolo de blanco o puntero de tiro. Se decrementa la munición del arma en uno, se actualiza el contador en la interfaz de usuario y luego se refleja este cambio en la munición del arma en el "Item" que controla el personaje. Esto se logra mediante una llamada a una función específica que debe incluirse en un ítem de tipo arma a distancia, tal y como lo requiere la interfaz "BPI_DistanceWeapon", ya que es una característica necesaria para este tipo de armas. Finalmente, se ejecuta el disparo como tal.

Para efectuar el disparo de un proyectil, primero se establece la variable "Attacking" como verdadera y se utiliza una nueva variable de estado llamada "Should move", la cual filtra si el personaje puede o no moverse. En este caso, se configura como falsa para evitar que el personaje se mueva durante el disparo. Luego, se ejecuta un *Anim Montage* de la misma manera que en el combate cuerpo a cuerpo. Este montaje incluye una animación que representa cómo el personaje realiza el disparo, y durante la ejecución de la animación, se genera una notificación diseñada específicamente para este escenario. Una vez que la animación concluye, y el disparo ya fue generado mediante la llamada al evento llamado "Shoot Projectile", que se explicará a continuación, se restablecen las variables a sus valores iniciales. Se verifica si el objetivo murió (lo cual implica que ya no existe en el mundo de juego), y si es así, se realiza la llamada a otro evento, "Target Is Dead", encargado de eliminar la referencia a ese objetivo para buscar otro dentro del radio de alcance. Esta última parte también se explicará con más detalle a continuación, pero es esencial para el buen funcionamiento del sistema, ya que si el objetivo muere y no se cambia o actualiza la variable, podría causar problemas y romper el sistema por completo.

El evento "Shoot Projectile" es el encargado directo de generar el disparo de un proyectil y está estrechamente relacionado con la secuencia de acciones explicadas anteriormente. Para que este evento funcione, se necesita la referencia de la dirección actual de la flecha perteneciente al arma, que se guardó en el evento anterior, y también se requiere la referencia a un objetivo, el cual se aseguró que existiera antes de ejecutar este evento. Primero, se rota la flecha de dirección del arma a distancia, utilizando la ubicación del objetivo. Este proceso asegura que el proyectil se dispare en la dirección correcta. Luego, se utiliza la función "Spawn Actor" para generar un nuevo actor de la clase del proyectil. Se proporciona la referencia de la clase del proyectil y la transformación del mundo que corresponde a la ubicación espacial de la flecha, que ahora cuenta con la rotación deseada. El proyectil recién creado se encargará de salir impulsado y causar daño por sí mismo.

Esta aproximación permite una gran variabilidad, especialmente en la creación de diferentes armas a distancia. Cada "Item" de arma a distancia proporciona su propia posición de flecha de dirección y tipo de proyectil, por lo que se tiene el control sobre qué se dispara y desde dónde.

Finalmente, en este evento se genera un sonido en el lugar donde se crea el proyectil y

se reporta como un evento de sonido para que pueda ser identificado por otros actores en el juego, es decir, los zombis escucharan cuando el jugador realice un disparo.

Los proyectiles se definieron como una nueva clase independiente, la cual se vincula al "Item" del arma a distancia responsable de dispararlo. No obstante, esta aproximación es sólo para que el arma proporcione la información necesaria para generar el proyectil, incluyendo la ubicación espacial en el mundo y la dirección. El proyectil en sí se encarga de salir con un impulso, utilizando un componente "Proyectile Movement", y con un *collider*, logra aplicar daño si golpea a un actor con la etiqueta "Damageable". Esta estructura permite cambiar fácilmente entre distintos proyectiles desde el arma y brinda a los proyectiles la capacidad de variar su comportamiento, como velocidad o daño, simplemente modificando la clase destinada a ese proyectil específico.

Continuando con el sistema, el evento "Focus Aim For Distance Attack" es crucial para proporcionar un nuevo objetivo en caso de que no exista uno actualmente o si se desea actualizar el objetivo actual por uno nuevo. Su función principal es delegar al evento correcto según sea el caso. Lo primero siempre es verificar la validez del objetivo actual del personaje. En caso de que no sea válido, se llama al evento encargado de buscar un objetivos dentro del alcance, "Update Surroundings Actors". Si el objetivo actual es válido, significa que se desea cambiar de objetivo entre los que se encuentran en el alcance, por lo tanto, se llama al evento encargado de actualizar a un nuevo objetivo, "Update Target Focus". Aunque pueda parecer un evento simple, desempeña un papel fundamental en la lógica de apuntar a un nuevo objetivo o actualizar uno existente. Este evento se activa tanto con la acción secundaria como en el evento de disparo, asegurando así que se llame adecuadamente cuando no hay un objetivo al principio o cuando se desea cambiar el existente.

El evento "Update Surrounding Areas" cumple la función de mantener actualizada la lista de objetivos dentro del radio de alcance del personaje. Para comprender este evento en particular, es necesario primero abordar cómo se definió el radio de alcance para el jugador. Se implementó una nueva *Capsule Collision* en el personaje, configurada con un radio de tamaño 20 y colisiones configuradas para superponerse a otros actores sin bloquearlos físicamente.

Así entonces, se utiliza un bucle para recorrer los actores dentro del radio de la cápsula. Se filtran los actores que puedan ser objetivos válidos para ataques a distancia a través de una nueva interfaz llamada "BPI_Damageable", la cual fue necesario implementar por la necesidad de que los actores que puedan recibir daño compartieran ciertas funciones relacionadas a este sistema. Luego, la verificación de la muerte o destrucción del actor es esencial para evitar inconsistencias, especialmente si el evento se llama en un momento en que el actor aún no ha sido completamente destruido. Los actores válidos se incorporan a un *Map* de *Strings*, estructura de datos similar a un diccionario de *Python*, donde las claves son los nombres de las instancias del actor y los valores son las referencias de los mismos. Esta elección del *Map* se basa en la necesidad de acceder directamente a un actor específico una vez que sale del radio de alcance.

En el evento, además de agregar el actor al *Map*, se evalúa la distancia entre el actor en cuestión y el objetivo actual referenciado en la variable "Current target damageable". Si la distancia es menor, se actualiza la variable para designar al actor más cercano como el nuevo

objetivo actual. El beneficio de hacer esto, es que una vez terminada la evaluación de todos los actores dentro del radio de alcance, se queda como objetivo con el actor que está más cercano al personaje.

Al concluir la iteración, se verifica si el tamaño del *Map* es mayor a cero, indicando así si se encontró un nuevo objetivo. Si el *Map* está vacío, se restablecen las variables a sus valores iniciales. En caso contrario, se crea una *List* de *Strings* basada en las claves del *Map*, lo que facilitará el recorrido ordenado al actualizar los objetivos dentro del alcance del personaje.

El siguiente evento "Update Target Focus", es crucial para actualizar un objetivo cuando ya existe uno previamente. La *List* con las llaves del *Map* desempeña un papel fundamental, permitiendo la actualización ordenada de actores en el radio de alcance. En primer lugar, se verifica la existencia de un objetivo actual en la variable "Current Target Damageable" del personaje. Si no existe, se examina si la *List* tiene elementos; si está vacía, se requiere llamar al evento anterior para volver a actualizar los actores dentro del alcance. Si la *List* contiene elementos y no hay un objetivo actual, se utiliza la variable "Current Target Index", creada para indicar el actor en el índice correspondiente, asegurando que se elija el siguiente actor en cada actualización. Esto se complementa con la creación de una función que proporciona siempre el siguiente índice, reiniciándolo si sobrepasa el tamaño de la *List*. Esta función garantiza la evaluación completa de los elementos antes de volver al inicial. Luego, se toma el valor de la *List* del índice correspondiente, que sabemos es un *String*, y este se utiliza para buscar la llave en el *Map*, estableciendo así al nuevo actor como objetivo. Se invoca la función "Focus As Target", presente en los actores con la interfaz "BPI_Damageable", para resaltar y señalar el objetivo actual en el juego con el mismo efecto de contorno utilizado en el sistema de interacción, y se actualiza el índice al nuevo. En el caso de existir un objetivo anteriormente y si el tamaño de la *List* indica la presencia de más de un elemento, se utiliza el índice actual para evaluar el siguiente índice y cambiar al nuevo objetivo. Si no es el caso, se realiza el llamado al evento "Update Surrounding Actors" para actualizar los actores en el radio de alcance nuevamente.

A continuación, veremos el evento que maneja el caso en que un actor salga del alcance del personaje. Este evento es esencial en el sistema y se deriva del evento intrínseco de los *colliders*, específicamente de la cápsula que define el radio de alcance, denominado "On Component End Overlap". Este se activa justamente cuando un actor deja de superponerse a la cápsula, es decir, cuando sale del radio de alcance. Su utilidad radica en que, al salir un actor del radio de alcance, se espera que el jugador ya no pueda tenerlo como objetivo.

En primer lugar, se evalúa si hay un objetivo actual en la variable correspondiente. En caso afirmativo, se compara si es el mismo actor que está saliendo de la cápsula. De ser así, se llama a la función "Unfocus As Target", perteneciente a la interfaz de los objetos dañables, la cual desmarca el objeto como objetivo, desactivando el contorno. A continuación, se elimina el actor de la estructura de datos *Map* utilizando su nombre como referencia, ya que las llaves guardan el nombre del actor en un *String*. Finalmente, se restablecen las variables de objetivo e índice y se llama a la función "Update Surroundings Actors" para identificar actores dentro del área de alcance. Esto es crucial, ya que al salir el objetivo del área de alcance, el jugador debe seleccionar automáticamente al actor más cercano.

En el caso de que el actor que sale del área de alcance no sea el mismo que se tenía como objetivo, pero si corresponde a un actor al que se le puede dañar, debemos eliminarlo de la estructura de datos. Esto se logra reiniciando la *List* con los nombres de las llaves de la estructura de datos *Map*, ya que al eliminar un actor, el orden de la *List* se ve afectado. Luego, se actualiza el índice actual en relación con el actor objetivo del personaje.

El último evento, denominado "Target Is Dead", se encarga de gestionar la situación en la que uno de los objetivos muere o es eliminado del juego. En este evento, se toma el objetivo actual y, de manera similar a cuando sale de la zona, se elimina del *Map*. Sin embargo, en este caso particular, solo llamamos a la función de actualizar objetivo, "Update Target Focus". Si el objetivo es el único en el área de alcance, no es de preocupación, ya que el evento de actualizar objetivo maneja este caso.

Este tema se presenta como una combinación de múltiples eventos que ocurren de manera constante, alternándose según la situación. La elegancia de estos eventos radica en que, aunque dependen entre sí, están configurados de tal manera que abordan los casos límite y permiten que el sistema se auto sustente y responda de manera efectiva a las acciones que realice el jugador al utilizar el sistema de ataque a distancia.

3.3. Desarrollo mapa de prueba

Una vez completadas las mecánicas de juego propuestas inicialmente, el siguiente paso fue iniciar el desarrollo de un mapa (nivel) de prueba y todo lo necesario para proporcionar a los usuarios una experiencia donde pudieran utilizar el sistema y proporcionar retroalimentación sobre las mecánicas desarrolladas. Para llevar a cabo esto, se procedió primero a crear el mapa de prueba, donde se llevaría a cabo el contexto de la simulación. En esta etapa del desarrollo del videojuego, no se contaba aún con niveles predefinidos ni algún sistema para generar los niveles del juego, por lo que la única opción era crear un mapa específico para esta función.

Con el objetivo de evitar crear un mapa desde cero, se aprovechó un mapa de muestra incluido en los recursos del proyecto. Este mapa tenía el propósito de demostrar cómo se verían los recursos en un entorno y simulaba una ciudad post-apocalíptica. Se duplicó este mapa y se comenzó a trabajar en él para adaptarlo al escenario necesario para realizar las pruebas. (Véase Figura 3.14)

Figura 3.14: Mapa de prueba.

El mapa original era una ciudad de mundo abierto con una gran variedad de sectores, lo que complicaba la identificación de los sectores ideales para poner a prueba las mecánicas deseadas. Para abordar este problema, se decidió que el escenario proporcionado a los usuarios fuera lineal. Esto implicaba que los jugadores comenzarían en un punto de la ciudad y tendrían que avanzar por un camino predeterminado hasta llegar a un punto final. Esta decisión permitiría un mayor control sobre el movimiento de los jugadores a través del mapa y ofrecería la posibilidad de generar configuraciones específicas para que los jugadores probaran cada mecánica de manera gradual. La idea era introducir mecánicas simples al principio y progresar hacia las más complejas, brindando a los jugadores la oportunidad de combinarlas y evaluar las decisiones tomadas con las opciones disponibles.

El primer paso fue seleccionar el punto inicial, ubicado en una carretera elevada que ingresaba a la ciudad, mientras que el punto final se eligió en una de las salidas de la ciudad que conducía hacia un puente hacia las afueras. Esto proporcionó un objetivo claro para la prueba: entrar a la ciudad, cruzarla y salir. Para delimitar los caminos dentro de la ciudad, se utilizó una variedad de recursos, específicamente *Static Meshes* del conjunto proporcionado con el mapa, con el objetivo de mantener la coherencia con el entorno post-apocalíptico. Se cerraron caminos con rejas, basureros, muebles, barricadas, entre otros. La colocación de estos *Static Meshes* fue estratégica y se hizo manualmente en lugares que se consideraban adecuados para cerrar sectores y dejar accesibles solo las partes de la ciudad destinadas al escenario de prueba y las configuraciones de mecánicas.

Con los caminos delimitados y las áreas de paso identificadas, se procedió al desarrollo de configuraciones para poner a prueba las mecánicas de juego. Esto implicó modelar lugares dentro del mapa donde los jugadores podrían realizar acciones acordes a las mecánicas ofrecidas. Un ejemplo claro es el inicio del mapa, donde el puente está cortado y la única forma de avanzar es utilizando la mecánica de colocar una tabla para crear un nuevo camino (Véase figura 3.15). Estas configuraciones se evaluaron y probaron en cada parte del mapa para garantizar su funcionamiento y adaptarlas de la manera más intuitiva posible para los jugadores.

Figura 3.15: Visualización de obstáculo creado en la zona 1 del mapa de prueba.

En estas configuraciones, la colocación de *Static Meshes* no fue la única consideración; se desarrollaron nuevos actores y se implementaron nuevas funcionalidades para que las mecánicas tuvieran sentido y motivaran a los jugadores a utilizarlas como parte de la práctica o para avanzar en el nivel de prueba. A continuación, se detallarán los actores desarrollados y las nuevas funcionalidades implementadas para enriquecer la experiencia y proporcionar a los jugadores oportunidades significativas para utilizar las mecánicas de juego desarrolladas.

Para comenzar, se abordarán los delimitadores de zona, actores diseñados para evitar que los jugadores accedieran a áreas no previstas en el mapa. Para comunicar al jugador que no debía entrar en estas zonas, se configuraron estos actores para infligir daño al personaje al ser tocados. Se utilizaron *Static Meshes* con apariencia de alambre de púas o pinchos para indicar visualmente que el contacto causaría daño al jugador. Estos actores se colocaron estratégicamente en todas las zonas donde los jugadores podrían desviarse o salirse del camino previsto. La configuración de estos actores implicó simplemente la colocación de una *Static Mesh* apropiada, como se mencionó anteriormente, y una cápsula para detectar la colisión con el jugador y aplicar el daño correspondiente.

Después de delimitar el espacio jugable, era necesario configurar *Unreal Engine* para que las pruebas pudieran llevarse a cabo como un juego completo. Esto implicaba gestionar aspectos como la gestión de la muerte del jugador, la posibilidad de establecer puntos de reaparición y el tiempo de juego. Modificamos el *Game Mode* existente para cumplir con estos requisitos específicos, realizando ajustes en su *Blueprint* de clase y en su *Event Graph*.

Para configurar el modo de juego con fines de prueba, se requerían ajustes al inicio de la ejecución del juego. Aprovechamos el evento “Begin Play”, que se activa al iniciar el juego, e implementamos una secuencia que iniciaba un temporizador para medir el tiempo que tomaría completar la prueba. Este temporizador sería crucial para proporcionar información valiosa a los usuarios y permitir su registro en el formulario. Posteriormente, configuramos el HUD del jugador desde el *Game Mode* en lugar de vincularlo directamente al personaje, lo

cual era necesario para implementar la lógica de resurrección del personaje tras cada muerte. Al entregar la referencia desde el *Game Mode*, se asegura de que el HUD se configure nuevamente cuando cambie el *Pawn* en posesión del *Game Controller*, es decir, cuando se genere una nueva instancia del personaje.

En la secuencia, se identifica la posición del *Player Start*, un actor inherente de *Unreal Engine* destinado a especificar la ubicación inicial del personaje al inicio del juego. Se guarda esta ubicación en una variable llamada "Spawn Location", diseñada para registrar la posición de aparición del jugador en el mapa. Aunque inicialmente se establece la primera ubicación basada en el *Player Start*, más adelante se verá cómo esta variable se ajusta dinámicamente cuando el jugador interactúa con los puntos de reaparición. Esto forma parte de la lógica que se implementó para permitir que el jugador renazca en ubicaciones específicas según su progreso en el nivel.

En la fase final de la secuencia, se implementa la lógica para el proceso de resurrección del personaje. Para identificar cuándo el personaje es destruido, se utiliza un evento específico. En este escenario, se toma la variable de ubicación de aparición, "Spawn Location". Posteriormente, se obtiene el controlador del jugador y se genera un nuevo actor de la misma clase que el personaje en la ubicación designada. De esta manera, el jugador observa cómo resurge un nuevo personaje en la última posición de aparición registrada, ya sea en el punto de inicio definido por el *Player Start* o en uno de los puntos de reaparición si logró alcanzar alguno antes de su muerte.

Los puntos de reaparición son actores diseñados con un *Static Mesh* llamativo para atraer la atención del jugador, una *Capsule Collision* configurada para detectar la presencia de actores que se sobrepongan al actor, una flecha de dirección y un pequeño intérprete de texto para indicar su naturaleza como punto de reaparición (Véase figura 3.16). La función de este actor es activar una variable de estado cuando la cápsula identifica que el jugador está sobre el punto de reaparición. Esto se refleja mediante un cambio de color en el *Static Mesh*. Posteriormente, se accede al *Game Mode* para ajustar la ubicación de aparición del jugador, logrando esto al modificar la variable "Spawn Location". Para proporcionar una notificación clara al jugador sobre la activación del Checkpoint, se incorpora un pequeño sonido junto con el cambio de color.

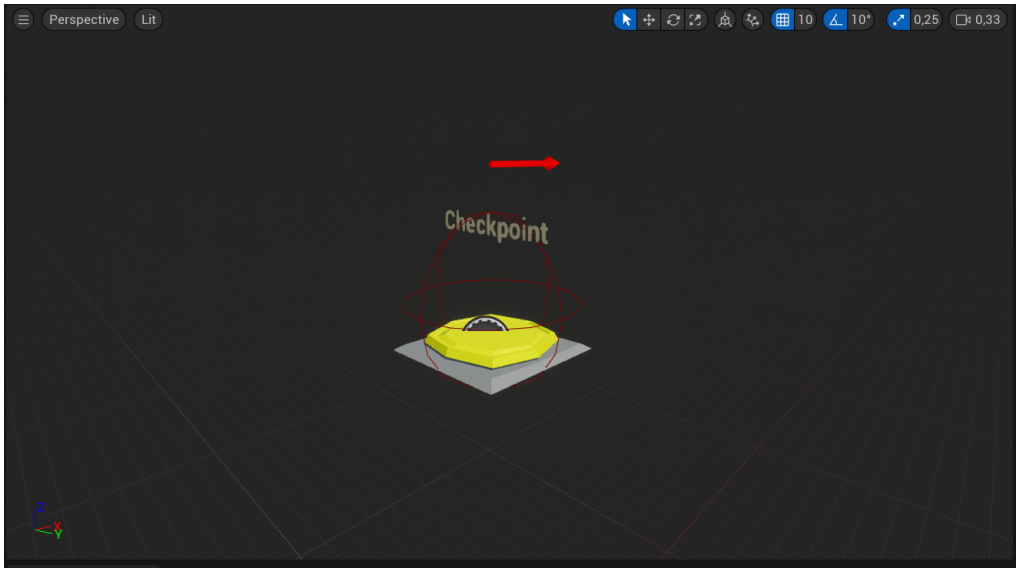


Figura 3.16: *Blueprint* punto de reaparición.

Con las características implementadas hasta este punto, el juego estaba prácticamente listo para funcionar de manera autónoma durante su ejecución. Sin embargo, restaba incorporar un mecanismo que permitiera concluir la prueba. Para abordar este aspecto crucial, se creó un actor denominado "End Point". Similar al punto de reaparición, consta principalmente de un *Static Mesh*, una *Capsule Collision* para detectar la presencia del jugador y un pequeño texto para su identificación. La lógica del "End Point" implica verificar si el jugador entra en su cápsula. En caso afirmativo, se inicia una música de finalización, se pausa el juego mediante la función "Set Game Paused" de *Unreal Engine* y se crea un *widget* que se añade al *viewport* para cubrir la pantalla. Este *widget*, creado previamente, informa al jugador que ha completado la prueba y recupera el tiempo del temporizador iniciado al comienzo, mostrándolo en la pantalla de finalización. Se incluye un botón para que el usuario pueda cerrar la ejecución del juego (Véase Figura 3.17).

Figura 3.17: Pantalla de finalización de la prueba.

Con esto, un jugador puede iniciar y concluir el juego sin inconvenientes. No obstante, se realizaron varias mejoras y ajustes en iteraciones posteriores en el mapa de prueba para optimizar la experiencia de los usuarios. A continuación, se describen todos los cambios, mejoras y adiciones implementadas para perfeccionar las pruebas.

Se consideró como una característica importante la adición de un menú de pausa para ofrecer al jugador opciones básicas como finalizar, resumir o reiniciar la ejecución del juego. Para implementar esto, se creó una nueva entrada para activar el menú de pausa, eligiendo la tecla "Esc", opción comúnmente utilizada para esta función. Se generó un *widget* similar al utilizado para la pantalla de finalización, pero ahora el *widget* se activa en el *viewport* del jugador al presionar la tecla "Esc", en lugar de ser activado por un actor.

La lógica es sencilla: al ejecutarse la tecla de entrada designada para pausar el juego, se activa un interruptor. Este interruptor alterna entre abrir y cerrar el menú de pausa. Para abrirlo, primero se crea un *widget* de la clase "Pause Menu" que se crea para este propósito, y se agrega al *viewport*, luego pausa el juego y cambia el modo de juego para habilitar el cursor. Esto es esencial para bloquear las entradas del juego y permitir al usuario interactuar con el menú mediante el ratón. Por otro lado, si el menú ya estaba activado, la otra opción del interruptor para cerrar, elimina todo lo activado previamente y vuelve a poner el juego en ejecución, es decir, lo reanuda. Los comandos que definen la lógica para los botones del menú de pausa se encuentran en el gráfico de eventos del *widget* de la clase "Pause Menu". La lógica detrás de cada botón es directa: el botón de finalizar la ejecución termina el juego llamando una función especifica para esto de *Unreal Engine*, el botón de reanudar realiza los mismos comandos especificados antes que manejan el cierre del menú de pausa, y el botón de reiniciar se encarga de cargar el nivel nuevamente, restableciendo así todos los elementos al estado inicial del nivel. Con esto, se logra implementar un menú de pausa simple pero útil para las funciones fundamentales del usuario en caso de que sean necesarias.

Figura 3.18: Visualización del menú de pausa durante ejecución.

Un detalle menor, pero no menos importante, fue la consideración de que los usuarios podrían utilizar tanto el teclado y el ratón, como un control remoto de tipo *Xbox* para ju-

gar la prueba. De esta forma se configuraron todos los *Action Inputs* ya existentes para que pudieran manejar ambos casos. Esta nueva configuración no solo permite a los usuarios una mayor comodidad, sino que también les brindaba la sensación de estar participando en una verdadera prueba de un videojuego.

Una vez que se considera tener todos los elementos necesarios para una experiencia de usuario satisfactoria, el enfoque se dirige a mejorar la experiencia iterando en el diseño del mapa. La creación de múltiples "Items" fue una de las primeras mejoras para proporcionar al usuario una variedad de opciones, especialmente en la mecánica de combate. La introducción de diversas armas cuerpo a cuerpo permitió ajustar los daños de cada arma, brindando una sensación de progreso durante la prueba.

Sin embargo, surgió la necesidad de que los enemigos se adaptaran al progreso del jugador para mantener la coherencia en la dificultad. Por lo tanto, se desarrollaron diferentes tipos de zombis basados en la clase base existente, con ajustes en sus características para ser más o menos competitivos según fuera necesario. Se modificó el sistema de movimiento y las variables de percepción de los zombis para crear tres tipos de enemigos: uno con malas percepciones y lento, para que al principio los jugadores se familiarizaran con las mecánicas, uno normal con las mismas configuraciones iniciales del primer zombie creado, pero con un daño ligeramente mayor, y finalmente, uno agresivo con mayor velocidad, daño y percepción. Cada uno de estos zombis se diferenció por el *Skeletal Mesh* entregado, de tal forma que no se vieran igual, y los jugadores los pudieran distinguir.

Estas variaciones enemigas permitieron diseñar configuraciones desafiantes dentro del mapa, proporcionando a los jugadores la oportunidad de enfrentarse a diferentes desafíos mientras avanzaban. Además, justificaron la mejora progresiva de los "Items" en el juego. Este enfoque en la diversidad de enemigos contribuyó a una experiencia más dinámica y estratégica para los jugadores.

En esta etapa del desarrollo, se identificó que las mecánicas de sigilo no estaban completamente integradas con las mecánicas de movimiento. Para abordar esta cuestión y aprovechando las opciones disponibles, se decidieron implementar dos nuevas mecánicas que enriquecieron tanto el juego en sí como el sistema de sigilo de la escena de prueba. Se añadió una emisión de sonido a los pasos del jugador cuando se movía de pie, permitiendo que los zombis identificaran al jugador incluso cuando no estuviera en su campo de visión. La clave aquí fue omitir esta emisión de sonido cuando el jugador se movía agachado o acostado, lo que le permitía desplazarse sigilosamente y evitar llamar la atención de los zombis por el sonido generado al estar de pie. Esta mecánica proporcionó a los jugadores la opción de moverse muy cerca de los zombis fuera de su rango de visión, sin atraer su atención si optaban por moverse de manera sigilosa. Por el contrario, si decidían ir de pie corriendo podría ser un gran riesgo para el jugador, incluso con los zombis mirando en otra dirección. En consecuencia, los zombis se convirtieron en un obstáculo más desafiante de evadir, incluso cuando no estaban directamente mirando hacia la dirección del jugador.

Tras esta implementación y considerando que ahora el jugador tenía la opción de pasar muy cerca de los zombis, se introdujo el otro sistema nuevo: si el jugador golpeaba a un zombie que no lo tenía identificado como objetivo, entonces el daño infligido al zombi se duplicaría.

Esta mecánica incentivaba a los jugadores a moverse de manera más sigilosa por el mapa y evitar el combate directo. También ofrecía una nueva estrategia cuando se encontraban con zombis.

La implementación de ambas mecánica no presentó grandes dificultades a nivel de código, ya que se aprovecharon variables existentes, como la que indica si el zombi está persiguiendo a un jugador. Simplemente se evaluó si esta variable era negativa, y en caso afirmativo, se multiplicó por dos el daño en el sistema de vida del zombie. En cuanto a la emisión de sonidos, se agregó un nuevo evento en el gráfico de eventos del jugador que evaluaba si este se movía a través del suelo utilizando una función del componente de movimiento de navegación (*Nav movement component*) y verificaba las variables de estado creadas durante el desarrollo para determinar si el jugador estaba en una posición que requería emitir o no emitir sonidos de pasos según fuera necesario.

Finalmente, se realizaron ajustes finales para mejorar la experiencia en el mapa de prueba. Se incorporaron letreros para indicar la dirección que debía tomar el usuario, ya que durante las iteraciones se observó que a veces los caminos podían resultar confusos. Además, se creó un nuevo actor: un Blanco de tiro con la interfaz "BP_Damageable", colocado en el sector que presentaba la mecánica de combate a distancia. El propósito de este actor era proporcionar al jugador la oportunidad de probar el combate a distancia contra objetivos estáticos que no se movían ni se destruían, pero que actuaban como cualquier otro objetivo susceptible de recibir daño. Esto permitía a los jugadores practicar la mecánica de disparo o cambiar objetivos antes de enfrentarse a los zombis, que, como sabemos, atacarían al jugador una vez fueran atacados.

Los últimos ajustes en el mapa de prueba se centraron principalmente en la colocación de *Static Meshes* o reposicionamiento de los ya existentes, para mejorar la fluidez y la intuición en la experiencia del usuario. Se revaluaron las posiciones y cantidades de zombis en las zonas de prueba para gestionar la dificultad y fomentar el uso de las mecánicas, priorizando la interacción sobre la complejidad. La colocación estratégica de "Items" en el mapa también se ajustó para que los usuarios tuvieran las herramientas necesarias para superar desafíos y avanzar en el camino de manera coherente.

Estas correcciones no solo se limitaron al mapa, sino que también incluyeron pequeños ajustes en variables y configuraciones iniciales de las mecánicas para adaptarse mejor al escenario creado. Con todos estos perfeccionamientos completados, el proyecto estaba listo para ser empaquetado, creando un ejecutable que permitiría a los usuarios probar y proporcionar retroalimentación sobre las mecánicas de juego desarrolladas en este proyecto.

Capítulo 4

Evaluación

La validación de las mecánicas de juego desarrolladas en este proyecto se realizó mediante la evaluación en un escenario de pruebas diseñado específicamente con ese propósito.

En particular, el objetivo de la evaluación fue obtener retroalimentación de usuarios, tanto con como sin experiencia en videojuegos, para medir su nivel de satisfacción y verificar si las mecánicas desarrolladas cumplían con las expectativas típicas de un juego de este género. El diseño del mapa de pruebas se planteó de manera que los usuarios se viesen obligados a utilizar las mecánicas creadas para alcanzar los objetivos propuestos. Por lo tanto, el nivel de éxito en estas pruebas también se consideró como un criterio de aceptación para las mecánicas, indicando la capacidad de los usuarios para utilizar efectivamente las nuevas mecánicas en un entorno desafiante.

Además, la generación de estas pruebas de validación es fundamental para mantener un registro valioso en el desarrollo del videojuego *Zombattan*. La retroalimentación recopilada durante esta etapa se utilizará para ajustar y mejorar las mecánicas de juego en futuras iteraciones, contribuyendo así al desarrollo y perfeccionamiento del videojuego final.

4.1. Muestra

El estudio contó con la participación voluntaria de 14 usuarios interesados en probar las mecánicas de juego de un videojuego, con edades que oscilaron entre los 23 y los 38 años.

4.2. Instrumentos

Los instrumentos empleados para evaluar las mecánicas de juego y llevar a cabo las pruebas fueron los siguientes:

En primer lugar, se creó una guía “How To Play” diseñada para explicar el contexto de las pruebas de mecánicas de juego, introducir a los usuarios en el nuevo videojuego (si no tenían conocimiento previo) y detallar las mecánicas desarrolladas. Este documento incluyó explicaciones de los botones necesarios para jugar, así como detalles específicos sobre el mapa, asegurando que los usuarios pudieran realizar la prueba sin dificultades. (Véase Anexo A.)

Para la ejecución de la prueba en sí, el proyecto de *Unreal Engine* se empaquetó en un archivo ejecutable compatible con sistemas *Windows*. Este archivo tenía un tamaño aproximado de un giga-byte y contenía todo el escenario de prueba utilizado en el estudio.

Finalmente, se creó un formulario de *Google* con preguntas que abarcaban información general del usuario, consultas sobre su desempeño en el mapa, cuestiones relacionadas con las mecánicas de juego en el contexto proporcionado y solicitudes de retroalimentación del usuario.

4.2.1. Formulario de validación

El formulario constaba de cinco secciones. Inicialmente, la primera sección proporcionaba toda la información relevante sobre el contexto de las pruebas, el propósito de la participación de los usuarios, así como detalles sobre la confidencialidad, privacidad y protección de datos durante la prueba . A continuación, se incluyó la sección de información general del usuario, que comprendía 8 preguntas generales sobre quienes realizarían la prueba. Seguidamente, se presentaron preguntas relacionadas con la evaluación del mapa, abordando aspectos generales para evaluar el desempeño y la finalización de la prueba por parte de los usuarios.

La cuarta sección se enfocó en preguntas directamente vinculadas a las mecánicas de juego, buscando obtener información relevante sobre la experiencia de los usuarios al utilizar dichas mecánicas. Finalmente, la última sección se dedicó a preguntas abiertas, permitiendo a los usuarios ofrecer retroalimentación, tanto positiva como negativa, sobre las mecánicas implementadas.

A continuación, se detallan las preguntas formuladas en el formulario:

4.2.1.1. Información general usuario

- Correo
- Nombre
- ¿Desea que su nombre aparezca en una sección de agradecimientos dentro del juego?
- Edad
- Respecto a tu experiencia con videojuegos, ¿Cómo describirías tu nivel de habilidad?
- ¿Con qué frecuencia juegas videojuegos en general?
- Indica si has estado involucrado en el desarrollo de un videojuego o eres parte de la industria.
- En tu elección de videojuegos, ¿tienes preferencia por aquellos que involucran elementos de Sigilo y Estrategia?

4.2.1.2. Evaluación mapa de prueba

- ¿Lograste completar el objetivo del mapa de prueba? (Cruzar la ciudad)
- Si no completaste el mapa, ¿cuál fue el desafío más difícil que encontraste?
- Tiempo que tomó finalizar la prueba. (00:00 - Si no logró completarla.)
- ¿Experimentaste algún problema técnico o necesitaste reiniciar la prueba?
- ¿Cómo calificarías la coherencia del escenario con el contexto previo proporcionado?
- Indica el estimado de veces que tu personaje murió al intentar completar el mapa.
- ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? (Pregunta reiterada para todos los sectores del mapa)
- ¿Cómo calificarías la dificultad general del mapa?

4.2.1.3. Evaluación mecánicas de juego

- ¿Qué tan intuitivas encontraste las siguientes mecánicas de juego?
  - Saltar
  - Agacharse
  - Acostarse
  - Tomar/Soltar Objetos
  - Lanzar objeto para distraer zombis.
  - Desplazar cajas para abrirte caminos.
  - Colocar tablas para alcanzar otros lugares.
  - Cubrirte con tripas de un zombi muerto para pasar desapercibido.
  - Atacar cuerpo a cuerpo a zombis.
  - Atacar a distancia a zombis.
- En tu experiencia de juego, ¿consideras que las mecánicas disponibles durante la prueba se complementaban de manera efectiva entre sí?
- ¿Hay alguna mecánica de juego específica que disfrutaste especialmente?
- ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? (Pregunta reiterada para todos los sectores del mapa)
- Indica qué tan de acuerdo o desacuerdo estás con las siguientes afirmaciones.
  - La variedad de mecánicas de juego fue suficiente para mantener mi interés durante la prueba.
  - Las instrucciones previas proporcionadas para las mecánicas de juego fueron suficientes para luego utilizar las mecánicas sin problema durante la prueba.
  - La dificultad de las mecánicas y del mapa en general estaba equilibrada.

- Las mecánicas de juego tuvieron un impacto positivo en mi sensación de inmersión en el mundo del juego.
- Con el nivel de *feedback* visual y auditivo actual asociado a las diferentes mecánicas de juego pude comprender y utilizar las mecánicas claramente.
- Las mecánicas de juego permitieron una interacción efectiva con el entorno y otros elementos del juego.
- Las mecánicas de juego fueron fáciles de aprender, y pude adaptarme a ellas rápidamente.

4.2.1.4. Retroalimentación usuario.

- Comparte cualquier comentario positivo respecto a las mecánicas de juego utilizadas durante esta prueba.
- Proporciona sugerencias o comentarios de mejora para las mecánicas de juego utilizadas durante esta prueba.

Para información detallada del formulario, ver documento en el anexo B.

4.3. Tareas

Para llevar a cabo las pruebas, se asignó a los usuarios una tarea principal: cruzar la ciudad desde el punto inicial hasta el punto final, considerándose el éxito del desafío al llegar al punto final. No obstante, es importante detallar las tareas específicas diseñadas en el mapa, las cuales los usuarios tuvieron que enfrentar para avanzar y completar el recorrido.

El mapa se dividió en 10 sectores, y en cada uno se planteó un desafío único. Cada desafío fue diseñado para que los usuarios pudieran aplicar las mecánicas del juego y superar los obstáculos. En varios sectores, era crucial que los usuarios identificaran la mecánica específica que más les ayudaría a superar el desafío y avanzar en el mapa.

En particular, a cada usuario de prueba se le encomendó realizar las siguientes tareas dentro del mapa:

- El primer sector, donde comienza el jugador, implica pasar por debajo de un helicóptero caído y luego enfrentarse a un puente partido a la mitad, donde debe habilitar el paso utilizando tablas que se encuentran en su entorno.
- En el segundo sector, el usuario se encuentra con el primer zombi, al cual debe distraer del camino principal para avanzar sin enfrentarlo directamente, para esto se introducen los “Items” que pueden ser lanzados.
- En el tercer sector, el usuario se enfrenta a una multitud de zombis que debe sortear para llegar a un paso elevado y evitar ser atacado por ellos. Acá se promueve el uso de los sesos para pasar desapercibido.
- En el cuarto sector, después de alcanzar el paso elevado, el jugador debe hacerse valer de tablas disponibles para avanzar sobre coches para llegar al otro lado de la carretera, evitando caer al nivel del suelo donde hay numerosos zombis.

- El quinto sector presenta una zona con varios enemigos, donde el jugador primero debe abrirse camino empujando un obstáculo y luego avanzar utilizando las diferentes mecánicas de sigilo o combate.
- El sexto sector consiste en avanzar sigilosamente a través del patio de algunas casas para llegar al siguiente punto.
- El séptimo sector plantea un pequeño rompecabezas, donde los usuarios deben mover cajas para subirse a unos contenedores que los llevarán al siguiente punto del camino.
- En el octavo sector se introduce la mecánica de combate a distancia, invitando a los usuarios a probar este nuevo mecanismo.
- El noveno sector consiste principalmente en un camino plagado de enemigos, ofreciendo la posibilidad de combinar las distintas mecánicas o utilizar la que prefieran.
- El último sector presenta una zona con los enemigos más fuertes, donde los jugadores deben avanzar por una pequeña rampa para alcanzar el final del camino o la salida de la ciudad.

4.4. Procedimiento

Dada la naturaleza de este estudio, se optó por realizar todo el proceso de manera completamente *online*. Se proporcionaron los archivos necesarios a los usuarios para comprender la prueba, junto con un ejecutable que contenía el escenario de prueba. Posteriormente, se solicitó a los participantes que respondieran a un cuestionario mediante *Google Form* para proporcionar retroalimentación sobre la experiencia y evaluar las mecánicas del juego.

La notificación de la selección de los participantes se realizó a través de un correo electrónico, el cual incluía información detallada sobre el proceso, fechas límite y las instrucciones para utilizar los archivos adjuntos. Además, se proporcionaron indicaciones sobre los pasos que debían seguir para llevar a cabo el estudio de manera efectiva.

El proceso se dividió en tres pasos para facilitar la participación de los usuarios. En primer lugar, se les indicó descargar el documento "How To Play", un archivo PDF que proporcionaba información detallada y contextualización sobre la prueba y las mecánicas del juego.

Una vez que los participantes hubieron leído el documento, el segundo paso consistió en descargar el archivo ejecutable, el cual les permitiría ejecutar el escenario de prueba y experimentar con las mecánicas desarrolladas.

El tercer y último paso consistía en completar el *Google Form* después de haber finalizado la prueba en el ejecutable. Este formulario permitiría registrar las respuestas y recopilar la retroalimentación de los usuarios.

Los archivos entregados fueron comprimidos en formato ZIP y se proporcionó una clave en el mismo correo que notificaba a los participantes sobre su selección para las pruebas. Esta clave era necesaria para descomprimir los archivos.

4.5. Resultados y análisis.

Los resultados obtenidos de las pruebas de las mecánicas de juego, llevadas a cabo por los participantes en el estudio, fueron los siguientes.

4.5.1. Información general usuario

- **Correo, nombre, y autorización a utilizar su nombre para dar los respectivos créditos por su participación:**

Estas interrogantes del formulario estaban principalmente diseñadas para recopilar información de contacto de los participantes y establecer una forma de identificación. Asimismo, se les consultaba si autorizaban la utilización de sus nombres para recibir créditos, ya que habían participado en una fase del desarrollo del videojuego, una práctica común en este tipo de proyectos.

- **Edad:**

Conocer la edad de los participantes resultaba fundamental para registrar la demografía de los usuarios que evaluaron las mecánicas, como lo destaca el estudio "Age-Based Preferences and Player Experience: A Crowdsourced Cross-sectional Study", el cual señala que "Nuestros resultados muestran un patrón consistente: con el aumento de la edad, las preferencias, los motivos de juego, el estilo de juego, la identificación como jugador y la experiencia del jugador se alejan del enfoque en el rendimiento hacia un enfoque en la finalización, la elección y el disfrute" [19]. Por lo tanto, es crucial considerar estas variables, especialmente para determinar a qué público se quiere dirigir el juego o qué aspectos deben corregirse para atraer a una mayor demografía de jugadores.

En particular, en estas pruebas, las edades se concentraron entre 23 y 25 años. A continuación, se presentan los resultados:

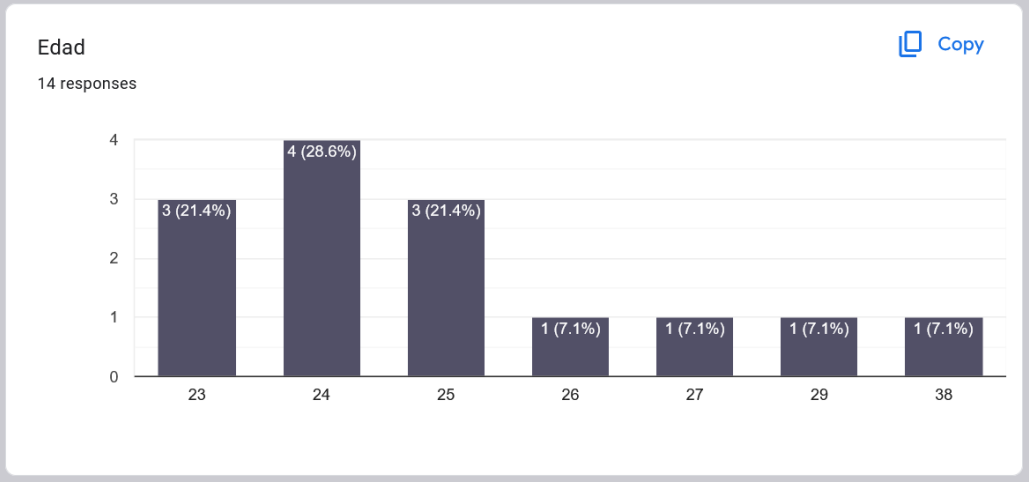


Figura 4.1: Resultados edades participantes.

- **¿Con qué frecuencia juegas videojuegos en general?:**

Indagar sobre la frecuencia con la que los participantes juegan videojuegos en general tiene como objetivo evaluar su experiencia y conocimiento en este tipo de entretenimiento. Obtener retroalimentación de usuarios con diversos niveles de experiencia es crucial, ya que las percepciones pueden variar entre aquellos que juegan regularmente y aquellos menos familiarizados con los videojuegos.

En esta ocasión, todos los participantes afirmaron ser jugadores habituales de videojuegos, con la mayoría indicando que juegan semanalmente.

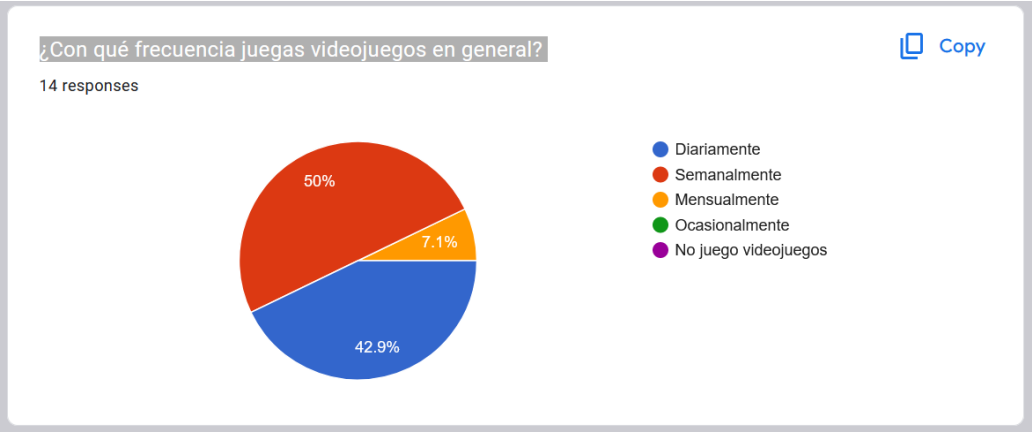


Figura 4.2: Resultados frecuencia de juego en videojuegos.

- **Indica si has estado involucrado en el desarrollo de un videojuego o eres parte de la industria:**

Preguntar si los participantes han estado involucrados en el desarrollo de videojuegos o forman parte de la industria busca identificar si tienen experiencia en este ámbito. La diversidad de experiencias aporta perspectivas valiosas, ya que la evaluación de aquellos con antecedentes en desarrollo puede ser diferente a la de aquellos sin experiencia.

En este caso, el 14.3 % de los participantes afirmó tener participación activa en la industria, el 28.6 % indicó haber participado ocasionalmente en proyectos similares, mientras que el 57.1 % declaró no tener experiencia en este campo.

Figura 4.3: Resultados de preferencias respecto a videojuegos.

- **En tu elección de videojuegos, ¿tienes preferencia por aquellos que involucran elementos de Sigilo y Estrategia?:**

La pregunta sobre la preferencia por videojuegos que involucran elementos de Sigilo y Estrategia busca comprender las preferencias de los participantes en relación con el género de videojuegos que *Zombattan* busca abordar.

En este caso, la gran mayoría de los participantes indicó disfrutar de juegos de estrategia y sigilo, aunque no manifestaron tener una preferencia específica por este tipo de videojuegos.

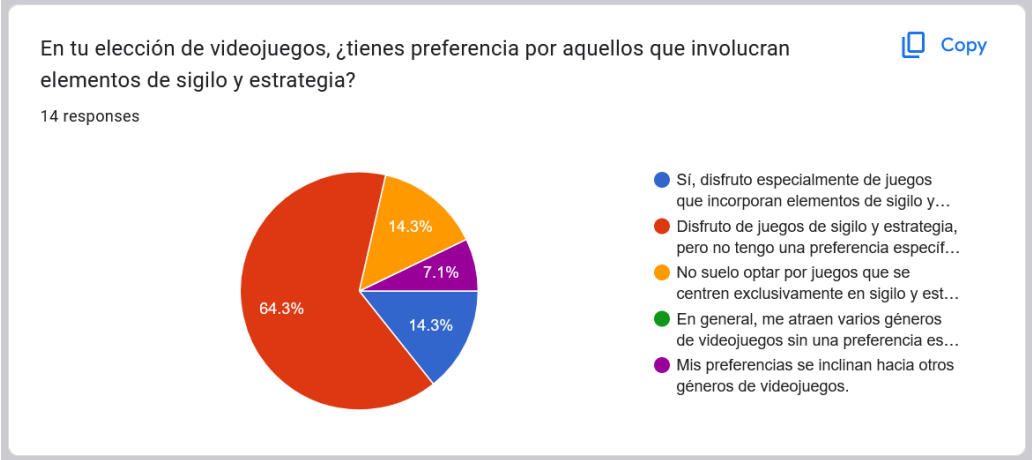


Figura 4.4: Resultados de éxito completando el mapa de prueba.

- **¿Lograste completar el objetivo del mapa de prueba? (Cruzar la ciudad):**

Saber sobre si el participante pudo cumplir la tarea encomendada durante la prueba proporciona un indicador claro de si lograron sortear los desafíos planteados utilizando las mecánicas de juego. Cuanto mayor sea la tasa de éxito, más podemos concluir que las mecánicas implementadas brindaron las herramientas necesarias para superar las dificultades y llegar al final.

De los 14 participantes, solo 1 declaró no haber completado el objetivo.

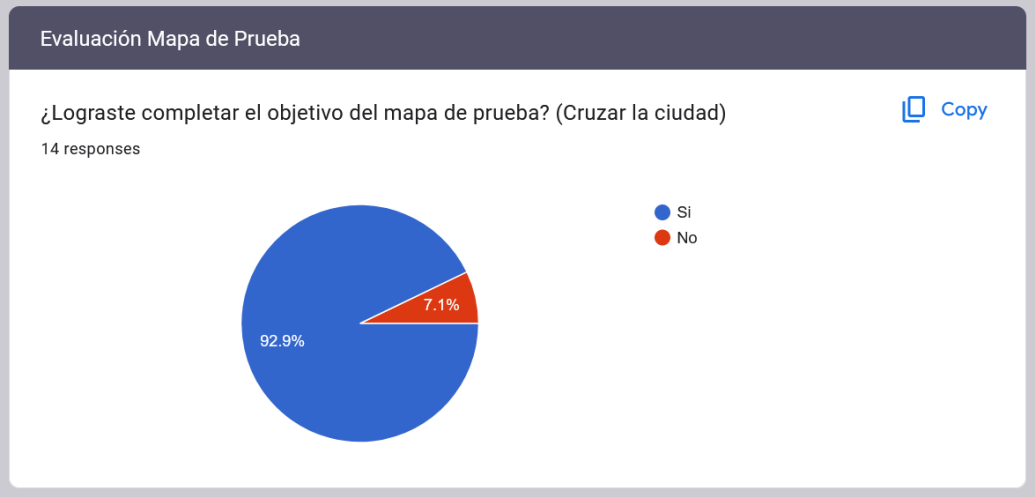


Figura 4.5: Resultados de éxito en completar el mapa de prueba.

- **Si no completaste el mapa, ¿cuál fue el desafío más difícil que encontraste? :**

La pregunta abierta opcional permitía a los jugadores que experimentaron dificultades al superar el mapa de prueba especificar los problemas encontrados. Esta información resultaba crucial para determinar si los problemas estaban relacionados con las mecánicas de juego, el diseño del mapa u otros factores que podrían afectar el cumplimiento del objetivo.

La persona que no pudo completar el mapa mencionó que la dificultad principal fue "notar que debía agacharme en algunos lugares para pasar". Este problema podría atribuirse al diseño del mapa y a la disposición de los obstáculos. Se sugiere que la indicación sobre la necesidad de agacharse debería ser más intuitiva para los jugadores, quizás mejorando la claridad en la señalización de los caminos disponibles.

- **Tiempo que tomó finalizar la prueba. (00:00 - Si no logró completarla.) :**

La pregunta sobre el tiempo empleado en la prueba proporciona una evaluación cuantitativa de si los participantes encontraron dificultades para completar las pruebas o si pudieron hacerlo sin problemas. El tiempo estimado para la prueba era de 25 minutos.

En general, las respuestas oscilaron entre 15 y 30 minutos, con dos casos particulares que indicaron 10:50 y 05:59, respectivamente.

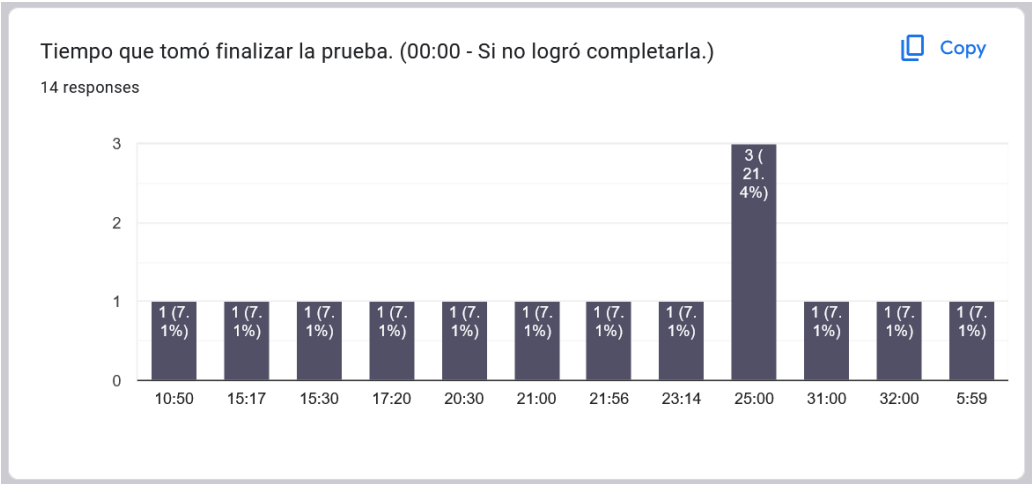


Figura 4.6: Resultados de tiempos registrados en completar el mapa de prueba.

- **¿Experimentaste algún problema técnico o necesitaste reiniciar la prueba? :**

La pregunta sobre problemas técnicos durante la prueba tenía como objetivo que los participantes identificaran posibles comportamientos inesperados o errores que se pasaron por alto antes de las pruebas. Identificar fallas o comportamientos inesperados es crucial para evaluar y abordar problemas de manera efectiva en el desarrollo del juego, especialmente en esta primera iteración del proyecto.

La mitad de los participantes informaron no haber experimentado ningún problema técnico. Los demás mencionaron diversos problemas relacionados con la muerte de los zombis, que provocaron comportamientos inusuales en las colisiones entre el personaje y los zombis. Un pequeño porcentaje también señaló comportamientos del mapa de prueba que no resultaban intuitivos o eran poco prácticos para la jugabilidad. En general, se registraron problemas menores y estos estaban relacionados con funciones específicas, lo cual es positivo en términos generales.

- **¿Cómo calificarías la coherencia del escenario con el contexto previo proporcionado? :**

La pregunta sobre la experiencia inmersiva estaba diseñada para evaluar si los jugadores pudieron sumergirse adecuadamente en el contexto de *Zombattan*, tal como se presentó en el documento “How to Play”. Utilizaba una escala del 1 al 5, donde el 1 representaba “Muy en desacuerdo” y el 5 “Muy de acuerdo”.

Esta evaluación permitiría determinar si los desafíos planteados y el diseño general del mapa de prueba lograron persuadir a los participantes de que estaban experimentando un juego en un mundo post-apocalíptico, conforme a la intención del desarrollo del juego. Por las respuestas, obtenidas podemos concluir que este objetivo fue cumplido satisfactoriamente.

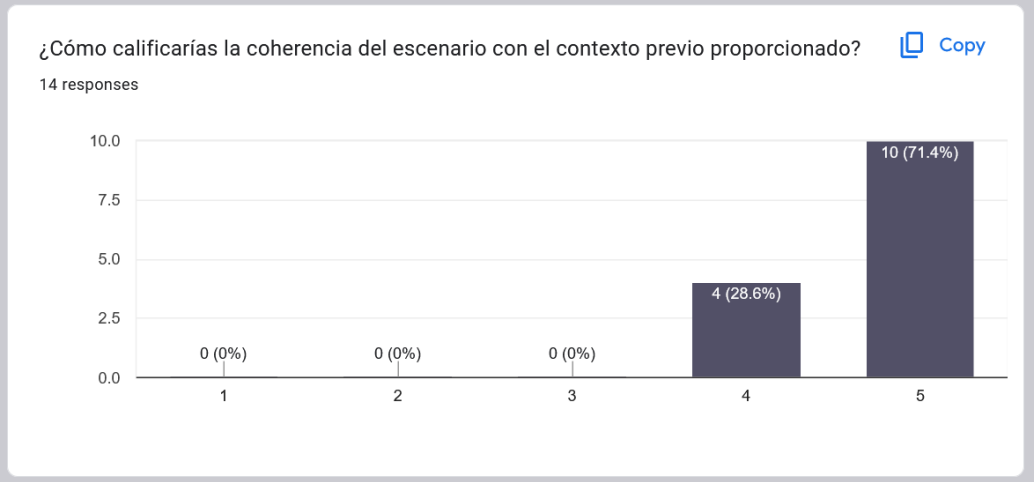


Figura 4.7: Resultados de la calificación respecto a la coherencia del escenario de prueba y el contexto del videojuego.

- **Indica el estimado de veces que tu personaje murió al intentar completar el mapa:**

La recopilación de información sobre la frecuencia con la que el personaje murió durante el juego proporcionó una evaluación cuantitativa de la dificultad de las pruebas. Este enfoque permitió entender si los participantes lograron superar los desafíos con relativa facilidad.

Notablemente, tres personas indicaron no haber muerto en ninguna ocasión. Esto sugiere que se adaptaron rápidamente a las mecánicas del juego o poseían habilidades significativas en este tipo de videojuegos. El resto de los participantes informó una cantidad esperada de muertes, lo cual es comprensible en un juego nuevo donde están familiarizándose con las mecánicas. En general, la dificultad demostró ser adecuada para la mayoría de los participantes.

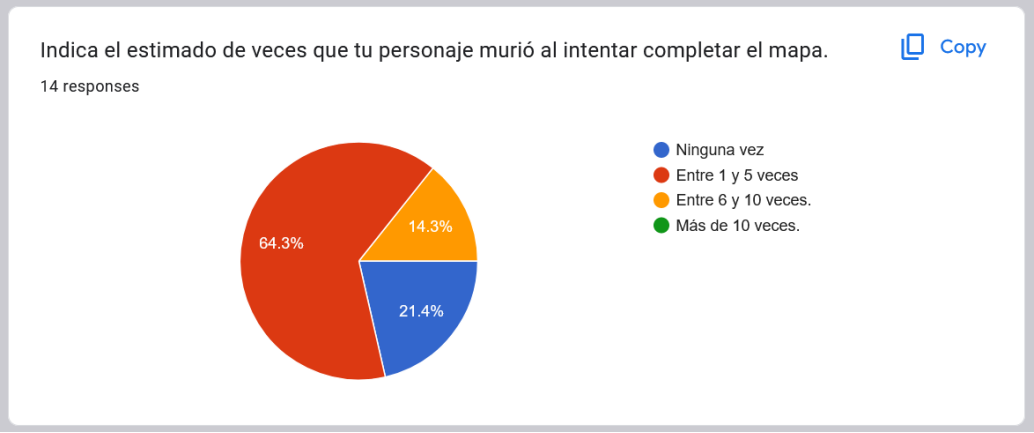


Figura 4.8: Resultados de muertes registradas en el personaje durante las pruebas.

- **¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba?:**

La evaluación de la dificultad en cada sector del mapa se llevó a cabo mediante la repetición de una pregunta similar para cada área específica, respaldada por una imagen que representaba el desafío correspondiente. La razón detrás de este enfoque era obtener una evaluación detallada de las áreas que presentaban más problemas para los usuarios y cuáles resultaban más accesibles. Este análisis permitiría correlacionar las dificultades con las mecánicas previstas para cada sector del juego. Cada participante respondió utilizando una escala del 1 al 5, donde el 1 indicaba una "Dificultad muy baja" y el 5, una "Dificultad muy alta".

A continuación, se presentan los resultados detallados para los 10 sectores respectivos.

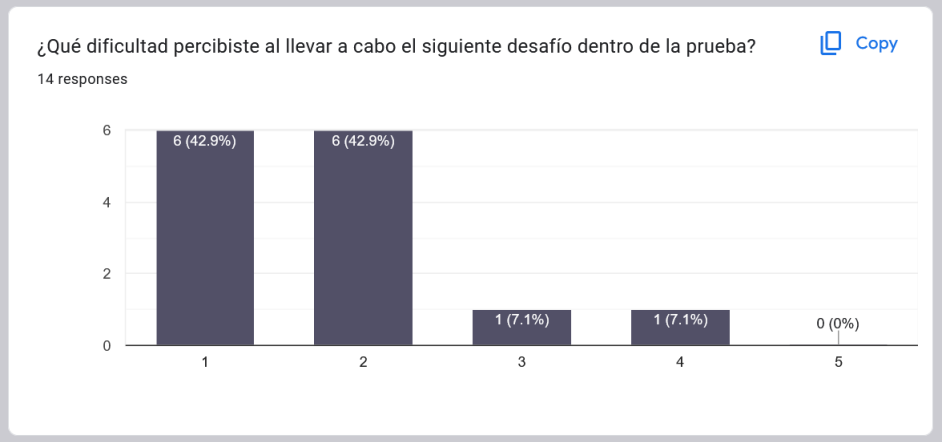


Figura 4.9: Resultados dificultad percibida en zona 1 del mapa de prueba.

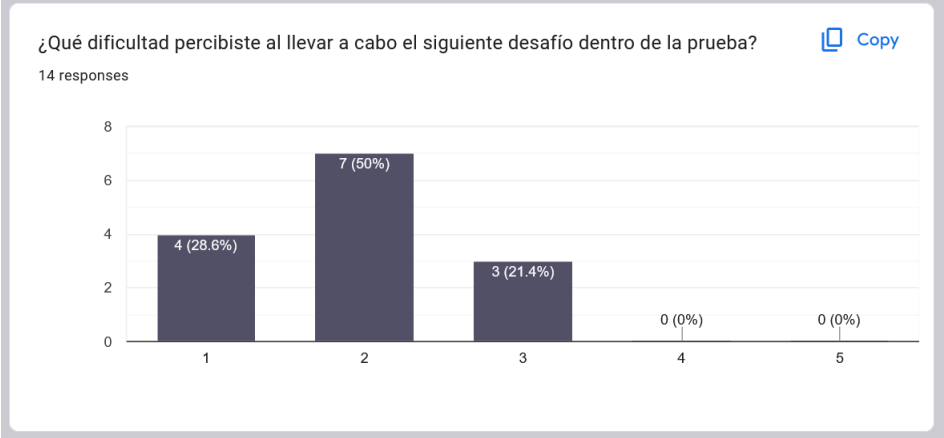


Figura 4.10: Resultados dificultad percibida en zona 2 del mapa de prueba.

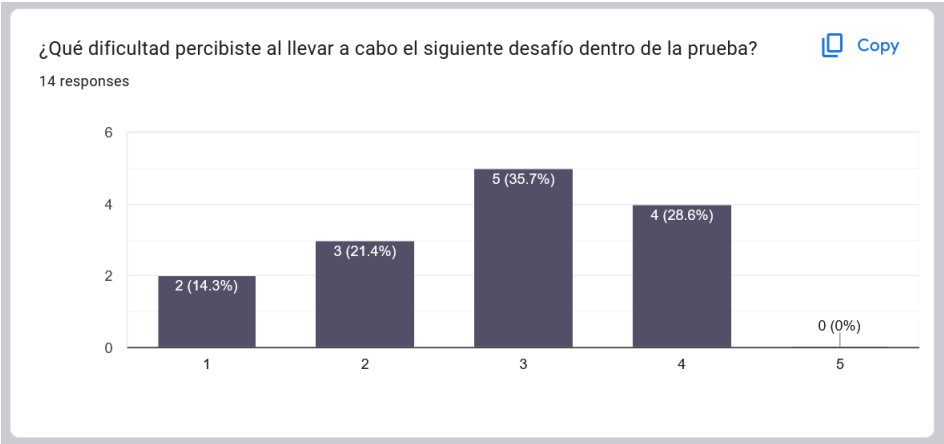


Figura 4.11: Resultados dificultad percibida en zona 3 del mapa de prueba.

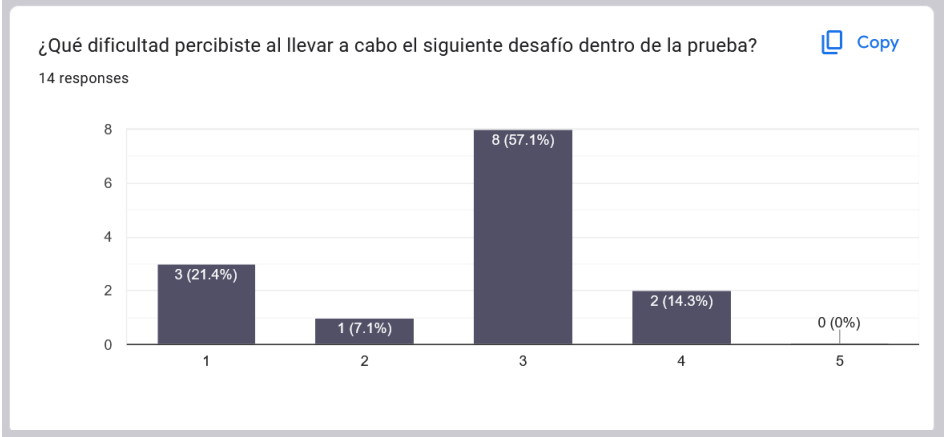


Figura 4.12: Resultados dificultad percibida en zona 4 del mapa de prueba.

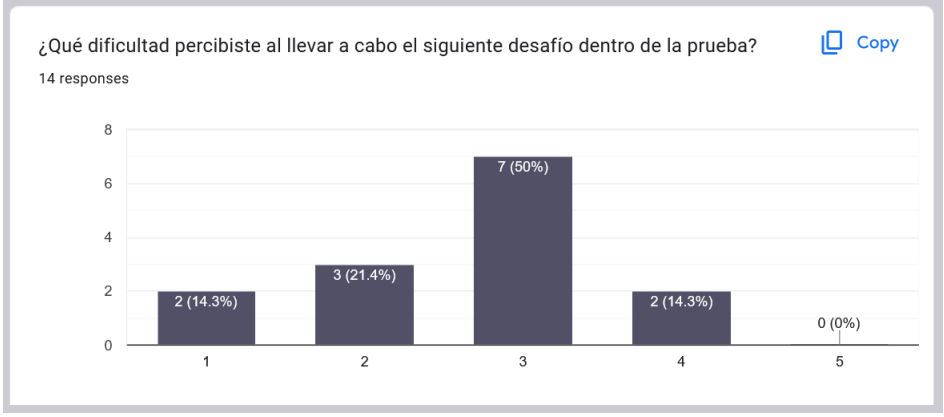


Figura 4.13: Resultados dificultad percibida en zona 5 del mapa de prueba.

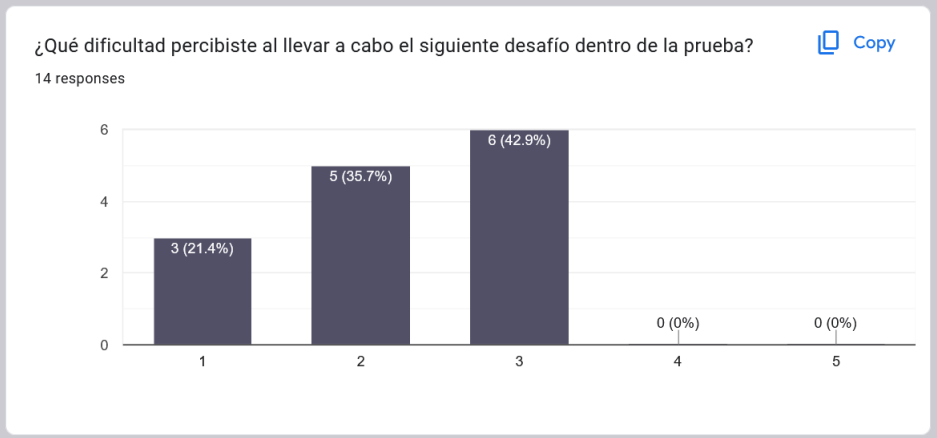


Figura 4.14: Resultados dificultad percibida en zona 6 del mapa de prueba.

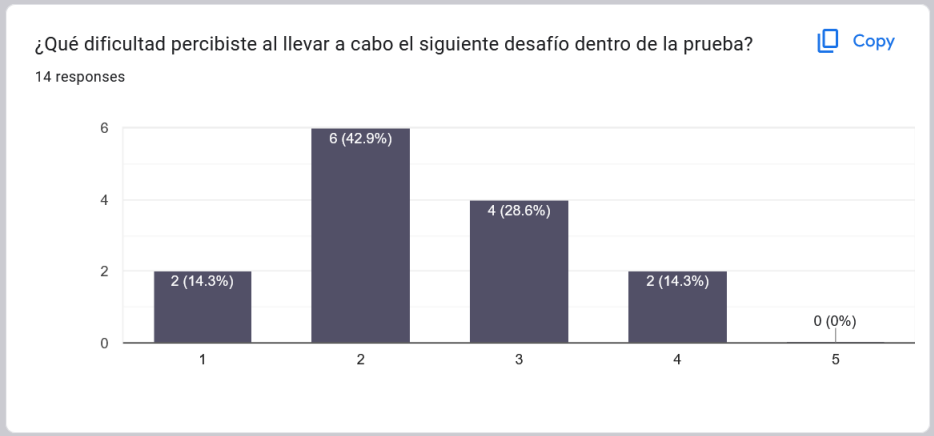


Figura 4.15: Resultados dificultad percibida en zona 7 del mapa de prueba.

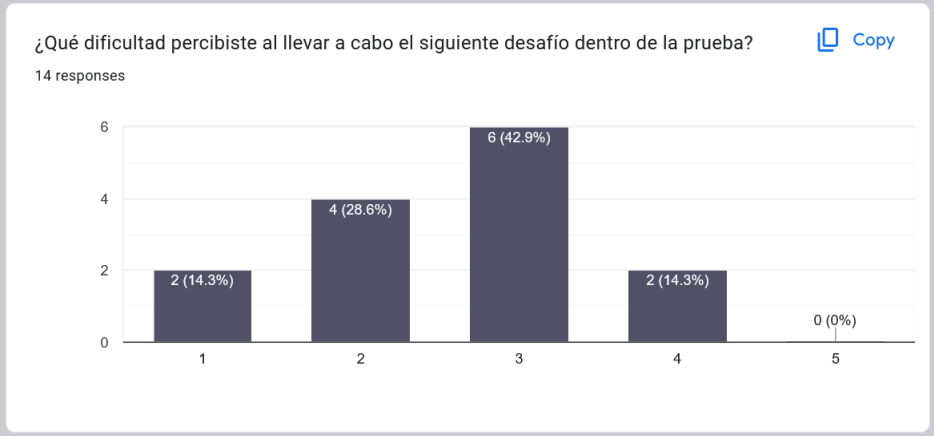


Figura 4.16: Resultados dificultad percibida en zona 8 del mapa de prueba.

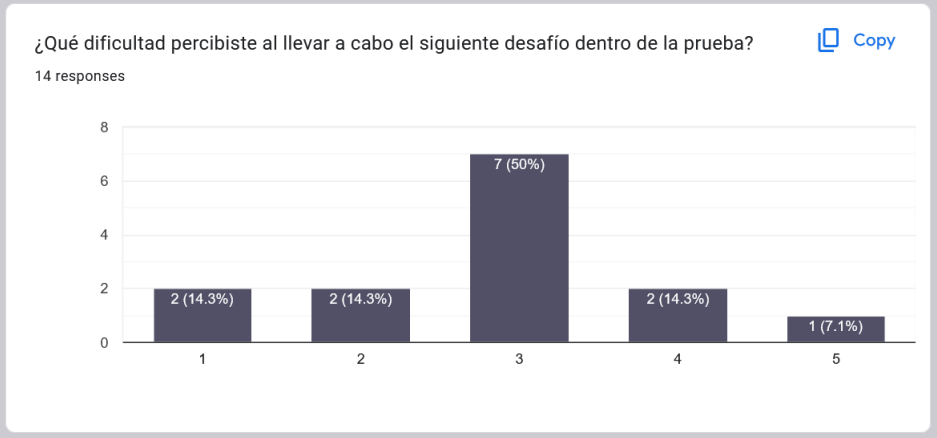


Figura 4.17: Resultados dificultad percibida en zona 9 del mapa de prueba.

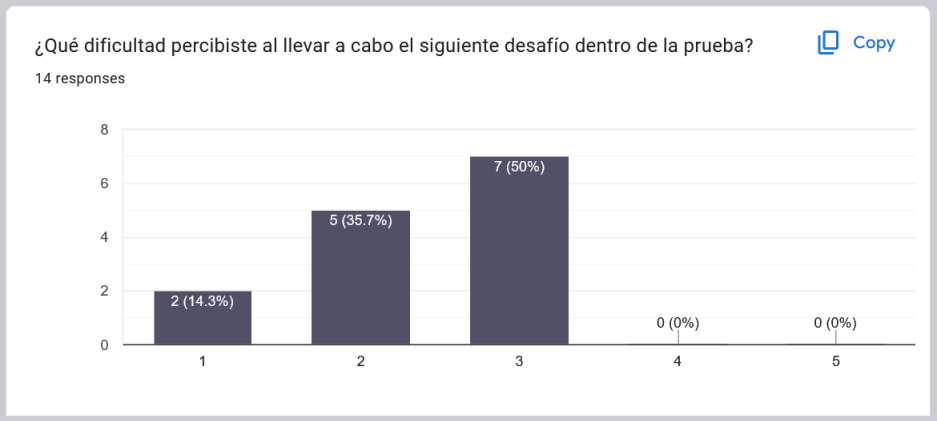


Figura 4.18: Resultados dificultad percibida en zona 10 del mapa de prueba.

- **¿Cómo calificarías la dificultad general del mapa?:**

La evaluación de la percepción de dificultad por parte de los usuarios en relación con el mapa de prueba se llevó a cabo mediante una escala del 1 al 5, donde 1 representaba una "Dificultad muy baja" y 5 una "Dificultad muy alta". Esta pregunta proporciona información subjetiva que, cuando se compara con los datos cuantitativos, como el tiempo y la cantidad de muertes, permite obtener conclusiones más precisas.

Los resultados revelaron que la mayoría de los participantes consideraron la dificultad del mapa como neutra. Sin embargo, se observaron respuestas de un participante que la clasificó como de dificultad muy baja y otro que la consideró de dificultad baja. Estas respuestas coinciden con los datos cuantitativos proporcionados en preguntas anteriores que evaluaban la dificultad de la prueba.

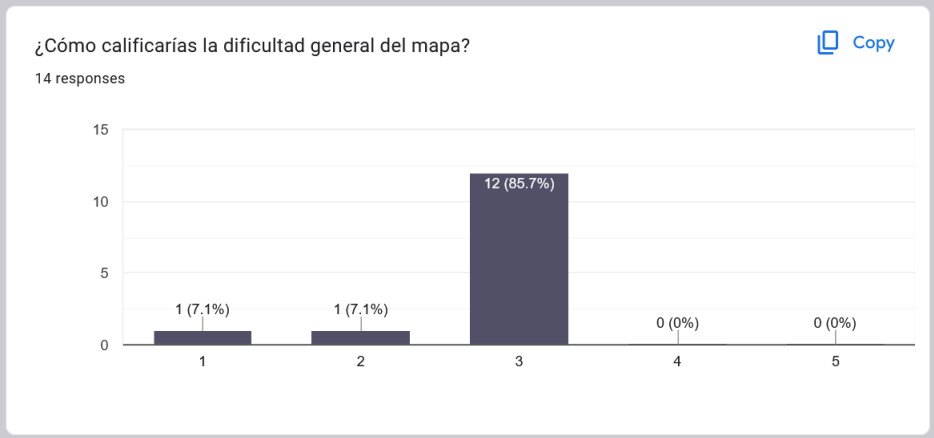


Figura 4.19: Resultados dificultad percibida en el mapa de prueba en general.

- **¿Qué tan intuitivas encontraste las siguientes mecánicas de juego?:**

En esta pregunta, se detallaron las mecánicas de juego implementadas una por una, permitiendo a los usuarios indicar qué tan intuitivas encontraron cada una durante la ejecución de la prueba. La escala de evaluación variaba de 1 a 5, donde 1 representaba "Muy Poco Intuitivo" y 5 "Muy Intuitivo".

Los resultados revelaron que las mecánicas de juego que involucraban a más actores complicaron un poco más a los usuarios en comparación con las mecánicas que dependían únicamente del personaje. En general, la mayoría de los usuarios evaluaron todas las mecánicas como intuitivas o muy intuitivas, con la excepción de la acción de recoger/soltar objetos, que obtuvo una evaluación neutral en igualdad con otra opción. Considerando que la gran mayoría pudo superar los desafíos dentro del mapa de prueba, los resultados son bastante positivos. Es relevante destacar que las mecánicas de combate fueron señaladas como menos intuitivas por un pequeño grupo de usuarios (2), lo cual podría explicarse por comportamientos inesperados cuando los zombis morían, según lo indicado en respuestas anteriores.

A continuación se pueden ver los gráficos con los resultados:

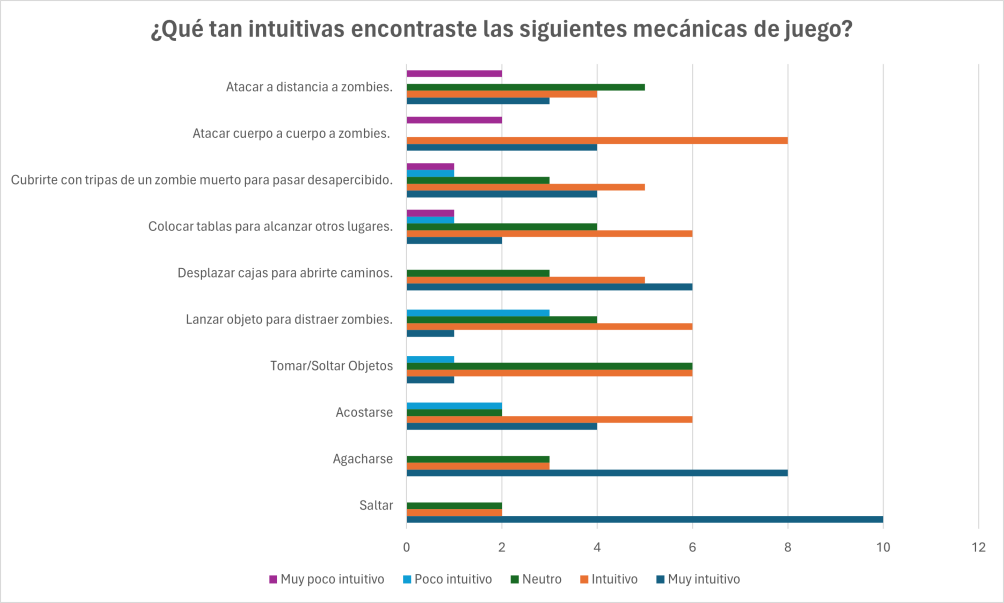


Figura 4.20: Resultados sobre la percepción de que tan intuitivas fueron las distintas mecánicas de juego.

- **En tu experiencia de juego, ¿consideras que las mecánicas disponibles durante la prueba se complementaban de manera efectiva entre sí? :**

En esta pregunta, se indagó sobre cómo se complementaban las mecánicas de juego entre sí, con el objetivo de entender si los jugadores percibían las mecánicas como acciones independientes o como un conjunto interconectado que proporcionaba la jugabilidad esperada dentro del contexto dado. La escala de evaluación iba de 1 a 5, donde 1 representaba "Muy en Desacuerdo" y 5 "Muy de Acuerdo".

Los resultados fueron bastante positivos, con un 64.3 % de los participantes indicando que estaban de acuerdo en que las mecánicas se complementaban, un 21.4 % declarando estar muy de acuerdo y un 14.3 % opinando de manera neutral al respecto. Esto sugiere que, en general, los jugadores percibieron una interconexión adecuada entre las diferentes mecánicas implementadas.

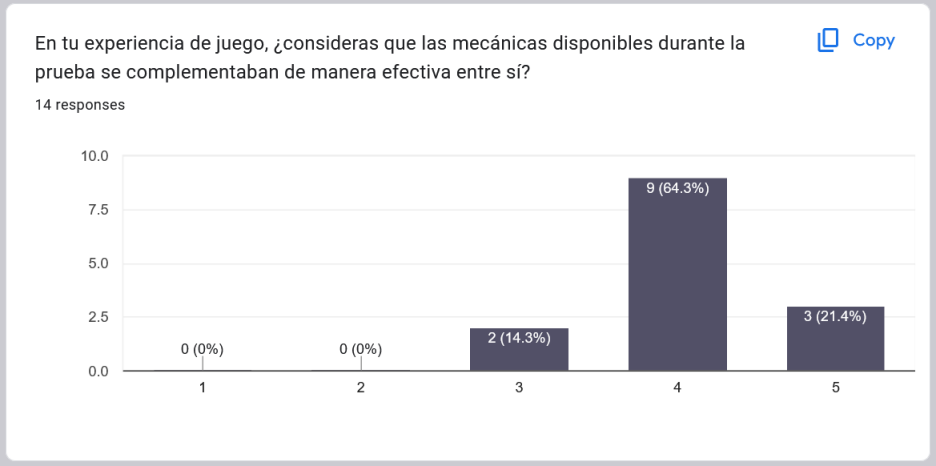


Figura 4.21: Resultados sobre si las mecánicas de juego lograban complementarse.

- **¿Hay alguna mecánica de juego específica que disfrutaste especialmente?:**

En esta pregunta, se buscaba conocer directamente qué mecánicas disfrutaron los jugadores durante las pruebas, con la finalidad de comprender sus preferencias y utilizar esta información para futuras mejoras en las mecánicas de juego.

Las mecánicas que más gustaron entre los jugadores fueron "Desplazar cajas para abrirse caminos" y "Cubrirse con tripas de un zombi muerto para pasar desapercibido", ambas seleccionadas por el 57.1 % de los participantes. Estas opciones destacan por ofrecer diversas posibilidades y desafíos para los jugadores, lo que sugiere que fueron bien recibidas durante las pruebas. Por otro lado, la mecánica menos apreciada fue "Acostarse", con solo un 7.1 % de los participantes seleccionándola. Esto podría deberse a que esta mecánica se requería menos durante la prueba para superar los desafíos y, además, puede no ser percibida como particularmente innovadora, siendo planteada principalmente para proporcionar opciones de movilidad al jugador.

- **¿Qué estrategia utilizó principalmente para pasar este sector de la prueba?:**

Esta pregunta se repitió para cada sector del mapa, presentando a los participantes una imagen que representaba un desafío específico. La repetición de esta pregunta para cada área proporcionó la oportunidad de evaluar qué estrategias y mecánicas se utilizaron con mayor frecuencia durante la prueba.

A continuación, se detallan las respuestas obtenidas por zona del mapa.

- Zona 1: En esta área, prevaleció la técnica de "Sigilo y moviendo a los zombis generando ruidos" con un 35.7 % de los participantes indicándolo como su respuesta, seguido de un 21.3 % que declaró haber utilizado la reposición de tablas, como se esperaba. El resto de los participantes optó por estrategias distintas.
- Zona 2: En este sector, era esperable que los jugadores sortearan al zombi con sigilo, siendo la opción elegida por la mitad de los participantes. Un 21.4 % optó por el combate cuerpo a cuerpo.

- Zona 3: El 57.1 % de los participantes declaró utilizar la mecánica de cubrirse con tripas, según lo diseñado para este sector. Sin embargo, una pequeña parte decidió optar nuevamente por el combate cuerpo a cuerpo (14.3 %).

- Zona 4: En esta área, la estrategia más utilizada fue nuevamente el sigilo y movimiento de los zombis a través de ruido, con un 35.7 %. Solo un 14.3 % utilizó el combate.

- Zona 5: En esta zona, las estrategias se dividieron entre el uso del sigilo y movimiento de zombis con ruidos (35.7 %), y el combate cuerpo a cuerpo (35.7 %).

- Zona 6: A partir de este sector, los resultados fueron los esperados, ya que el enfoque cambió hacia el enfrentamiento con más enemigos. En particular, el 57.2 % declaró utilizar el combate como principal estrategia en esta zona.

- Zona 7: Similar al sector anterior, en este sector la mayoría también optó por utilizar combate (42.9 %). Sin embargo, un 14.3 % recurrió al sigilo y movimiento de zombis con ruidos.

- Zona 8: En este sector, que presentaba la mecánica de combate a distancia, el 28.6 % la consideró como la principal estrategia. Sorprendentemente, se utilizaron diversas estrategias: el 21.4 % eligió el combate cuerpo a cuerpo, otro 21.4 % el sigilo y movimiento de zombis a través de ruidos, y un 14.3 % la utilización de tripas para pasar desapercibidos.

- Zona 9: Nuevamente, la mayoría prefirió optar por el combate, donde el cuerpo a cuerpo fue utilizado principalmente por un 42.9 % y el combate a distancia por un 14.3 %. Algunos prefirieron utilizar el sigilo y movimiento de zombis por ruido (14.3 %).

- Zona 10: En la zona final, la mitad declaró utilizar el combate cuerpo a cuerpo, el 21.4 % el sigilo y movimiento de los zombis a través de ruidos, y otro 21.4 % decidió correr.

Cabe destacar que un participante en particular, utilizó la estrategia de correr en prácticamente toda la prueba. Lo cual es algo que no se tenía presupuestado, pero que entrega información relevante para poder poner atención a ese comportamiento no deseado en futuras iteraciones.

- **Indica qué tan de acuerdo o desacuerdo estás con las siguientes afirmaciones:**

En esta sección, se formularon afirmaciones relacionadas con las mecánicas y el mapa desarrollado durante este proyecto. El propósito principal fue validar los objetivos planteados en la implementación de las mecánicas. Se brindó a los usuarios la oportunidad de expresar si estaban de acuerdo o no con las afirmaciones mediante una escala del 1 al 5, donde 1 indicaba "Muy en desacuerdo" y 5 "Muy de acuerdo".

Las respuestas en general fueron bastante positivas, sugiriendo que los objetivos fueron alcanzados en su mayoría. Sin embargo, es esencial no pasar por alto los casos de participantes que no estuvieron de acuerdo o se mantuvieron neutrales en ciertas afirmaciones. Analizar estos aspectos es crucial para identificar posibles áreas de mejora y cómo se pueden abordar en futuras iteraciones del proyecto.

A continuación se pueden ver las respuestas obtenidas:

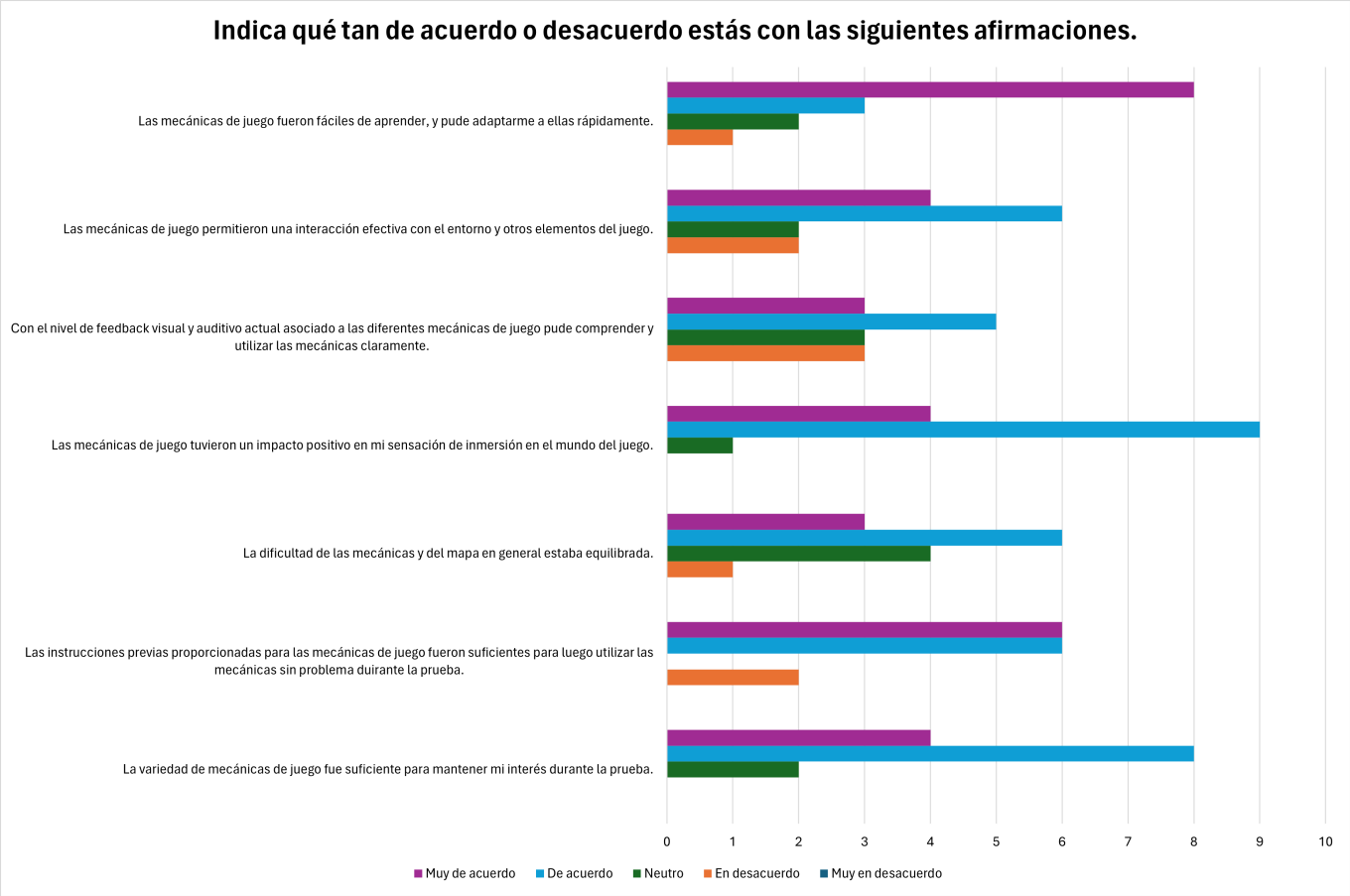


Figura 4.22: Resultados obtenidos respecto a distintas afirmaciones sobre las pruebas.

- **Comparte cualquier comentario positivo respecto a las mecánicas de juego utilizadas durante esta prueba. :**

En la sección final del formulario, se brindó un espacio para que los participantes pudieran proporcionar retroalimentación en una pregunta abierta. Esta pregunta buscaba recopilar información sobre las cosas que les gustaron, con el objetivo de tener en cuenta esos aspectos para el futuro desarrollo y perfeccionamiento de las mecánicas de juego y del videojuego en general.

Se recibieron numerosos comentarios positivos sobre las mecánicas de juego, destacando tanto el combate como el sigilo. Los participantes elogiaron la eficacia de las mecánicas de combate y apreciaron los detalles sutiles que las hacían interesantes. En el combate a distancia, se mencionó positivamente el hecho de que los zombis pudieran escuchar los disparos y reaccionar en consecuencia. Las mecánicas de sigilo también fueron elogiadas por ser intuitivas y útiles para superar los desafíos de las pruebas. El uso de tablas y el movimiento de cajas para abrir caminos fue destacado como un detalle que añadía valor al juego. La mecánica de cubrirse con sesos fue descrita como “útil e interesante”. En general, las mecánicas cumplieron las expectativas, y un participante expresó que eran “entretenidas mecánicas que ya dan un juego base”.

Por otro lado, se destacó algo que no estaba necesariamente previsto: el gusto por la ambientación lograda en el mapa de prueba. Los participantes elogiaron la utilización de recursos, la disposición del mapa y otros elementos, destacando que contribuyó significativamente a motivar la realización de las pruebas y a sentirse inmersos en el contexto presentado. Se subrayó que esta ambientación tiene potencial para crear escenarios dentro del juego que no necesariamente se resuelvan matando a los zombis, sino con rompecabezas. Esta retroalimentación positiva resalta la importancia de dedicar esfuerzos al diseño del escenario, ya que puede influir de manera significativa en la experiencia de los jugadores.

- **Proporciona sugerencias o comentarios de mejora para las mecánicas de juego utilizadas durante estas pruebas. :**

La retroalimentación recopilada a través de la pregunta abierta sobre áreas de mejora proporciona valiosa información para futuras iteraciones y mejoras en las mecánicas implementadas. Algunos participantes destacaron nuevamente el problema relacionado con el comportamiento no deseado generado al matar zombis y cómo esto afectaba las colisiones del personaje. Este sigue siendo un punto crítico que necesita ser abordado para mejorar la experiencia del juego.

En cuanto a la dificultad de las pruebas y las mecánicas, los comentarios revelaron detalles importantes sobre aspectos que podrían ajustarse para equilibrar la experiencia del jugador. Hubo observaciones tanto sobre elementos que daban ventaja excesiva a los jugadores, como la duración de la mecánica de pasar desapercibido con los sesos, como sobre aspectos que dificultaban su uso, como la sensibilidad de la cámara y las animaciones en el combate cuerpo a cuerpo.

Numerosos comentarios se centraron en detalles específicos del comportamiento de las mecánicas dentro del juego, desde animaciones hasta comportamientos poco intuitivos. Esto brinda una guía valiosa para perfeccionar y ajustar los sistemas existentes. Además, las recomendaciones sobre cómo comunicar mejor las mecánicas a los usuarios, ya sea a través de texto, efectos visuales adicionales o sonidos, ofrecen una dirección clara para mejorar la comprensión y la experiencia del jugador.

En relación con el mapa, los comentarios señalaron problemas como áreas donde los jugadores podían salirse del camino delimitado y la posibilidad de confusiones entre elementos de decoración y objetos con los que el jugador puede interactuar. Estos son aspectos prácticos que pueden mejorarse para garantizar una experiencia de juego más fluida y comprensible.

En resumen, la retroalimentación proporcionada aborda una variedad de aspectos, desde problemas técnicos hasta ajustes en la dificultad y la comunicación de las mecánicas. Este análisis detallado de la experiencia de los participantes será esencial para el desarrollo futuro del juego, brindando oportunidades para correcciones y mejoras significativas.

Capítulo 5

Conclusiones

A continuación, se concluye sobre el trabajo realizado, haciendo una retrospectiva para identificar áreas de mejora, aprendizajes y considerando posibles adiciones a las mecánicas del videojuego en el futuro.

Durante este proyecto, se ha abordado de manera integral el proceso de implementación de las mecánicas de juego para *Zombattan*, un juego en tercera persona del género de Sigilo y Estrategia. Esto se realizó explicando desde la fase inicial de diseño y selección de las mecánicas, dividiéndolas en categorías específicas: movimiento, sigilo y combate, hasta su implementación progresiva en *Unreal Engine* versión 5.2. Este proceso se llevó a cabo de manera estructurada, comenzando con las mecánicas más generales y avanzando hacia las más específicas y complejas. La fase culminante de la implementación implicó la creación del mapa de prueba y el desarrollo de todas las características necesarias para presentar las mecánicas en un ejecutable accesible para los usuarios, quienes evaluaron el trabajo. En última instancia, se ha detallado meticulosamente el procedimiento seguido, los instrumentos utilizados y los resultados obtenidos durante la fase de evaluación con usuarios, cumpliendo así el objetivo central de establecer mecánicas implementadas como base y proporcionar un registro valioso para la mejora continua en el desarrollo futuro de *Zombattan*.

Los objetivos establecidos fueron cumplidos de manera satisfactoria, demostrando coherencia entre la visión inicial del videojuego y la jugabilidad esperada para el género al que este aspira pertenecer. La selección y desarrollo de las mecánicas, así como la implementación de los sistemas necesarios, incluyendo el comportamiento de los NPC (Los zombis) contribuyeron a la formulación de un escenario acorde con la visión del juego. El proceso de pruebas con usuarios arrojó resultados mayormente positivos, respaldando la efectividad de las mecánicas y su capacidad para proporcionar la experiencia de juego deseada. En conjunto, estos logros culminan en el éxito del presente trabajo de título.

No obstante, es esencial destacar que, a pesar de los logros alcanzados, existen áreas de mejora y detalles deseables que no fueron posible implementar debido a las limitaciones de tiempo y alcance del proyecto, así como a las prioridades establecidas durante el desarrollo. La falta de un extenso conocimiento previo sobre la herramienta utilizada, *Unreal Engine*, inicialmente presentó desafíos para la concepción e implementación de soluciones. No obstante, este obstáculo se superó mediante un aprendizaje rápido y adaptación a las necesidades específicas del proyecto. Aunque este proceso implicó mejoras constantes y la revisión con-

tinua del trabajo realizado para optimizar resultados, inicialmente la curva de aprendizaje puede haber representado un obstáculo temporal.

En el proceso de implementación de las mecánicas de juego, se dio prioridad a los elementos técnicos sobre otros aspectos, centrándose principalmente en la funcionalidad. Como resultado, la comunicación de las mecánicas de juego con el usuario, es decir, la interfaz de usuario (UI), se limitó a lo estrictamente necesario para que los usuarios pudieran utilizar las mecánicas. Dedicar más atención a este aspecto podría haber resaltado aún más la implementación de las mecánicas o facilitado su comprensión. La falta de profundización en la interfaz de usuario se atribuye principalmente a restricciones de tiempo, ya que se priorizaron otras áreas durante el desarrollo. Se reconoce que este ámbito presenta un significativo espacio para mejoras en futuras iteraciones del proyecto.

También, la implementación del combate en general podría haber alcanzado un nivel superior con una planificación y desarrollo más exhaustivos de aspectos clave relacionados con los enemigos o NPC. Desafortunadamente, esta tarea quedó fuera del alcance de la presente etapa del proyecto. A pesar de ello, las bases establecidas para los dos tipos de combate sientan los cimientos para que este sistema se convierta en una de las principales atracciones dentro del futuro videojuego. Su potencial para mejorar y destacar en futuras iteraciones queda evidenciado.

En cuanto a la evaluación y en el contexto de trabajar con usuarios, la búsqueda de minimizar errores o bugs es fundamental para garantizar la validez de las pruebas. A pesar de adoptar la práctica de iterar repetidamente para identificar y corregir posibles problemas en el mapa y las interacciones con las mecánicas, algunos errores inevitables se omitieron inicialmente y fueron identificados por los participantes. Aunque estos errores no resultaron ser críticos para la prueba y validación de las mecánicas, es importante considerar la mejora en la planificación del *testing* en escenarios de prueba y en el desarrollo general del proyecto, posiblemente involucrando a más personas, un aspecto en el que se falló en esta ocasión.

Por otro lado, asimismo es importante destacar los aspectos positivos generados durante esta etapa. Se lograron cumplir todos los objetivos iniciales y se construyó un escenario de prueba visualmente atractivo y entretenido para los participantes. Sorprendentemente, este escenario llegó a ser considerado como un videojuego independiente por uno de los participantes. Es destacable este logro, ya que inicialmente no se tenía previsto dar tanto énfasis al aspecto visual, a las animaciones y a los elementos con recursos ambientados en el contexto del videojuego. Contar con los *Assets* necesarios y dedicar un esfuerzo adicional a las animaciones y visuales de las mecánicas demostró ser una gran decisión para las pruebas y la retroalimentación recibida.

En general, la forma en que se abordaron aspectos fundamentales para un videojuego del género de Sigilo y Estrategia, como las mecánicas de movimiento, sigilo y combate, representó una aproximación exitosa a un problema complejo y multifacético que carece de una solución única evidente. La implementación logró que estas mecánicas coexistieran y se complementaran de manera coherente, reflejando fielmente las acciones que se realizarían en un mundo post-apocalíptico. Los sistemas implementados demostraron ser intuitivos y útiles para los usuarios durante las pruebas, proporcionando herramientas eficaces para enfrentar

los desafíos del videojuego.

Las sensaciones positivas generadas durante las pruebas indican que la dirección tomada fue acertada. Aunque aún hay áreas de mejora y perfeccionamiento para estas mecánicas, se ha establecido una base sólida sobre la cual construir. En resumen, las mecánicas de juego implementadas no solo sientan las bases para lo que podrían ser las mecánicas finales del videojuego, sino que también proporcionan una valiosa retroalimentación para refinar y mejorar lo existente. Este hito marca un paso significativo en el desarrollo de *Zombattan*.

Participar en un proyecto de esta envergadura proporcionó la oportunidad de expandir los límites en el aspecto técnico, donde se requirió aprender una nueva herramienta y abordar diversos aspectos de la programación para que los sistemas fueran funcionales y extensibles. También representó un constante desafío para la creatividad, estableciendo la necesidad de encontrar la mejor manera de implementar la visión de un nuevo mundo dentro de las casi ilimitadas opciones disponibles. De esta forma, trabajar en este proyecto apremió a buscar constantemente el mejor modo de presentar algo a personas que buscan entretenimiento a través de lo que se estaba desarrollando.

Iniciar este proyecto sin un conocimiento profundo de *Unreal Engine* presentó un desafío adicional al principio. Sin embargo, con certeza se puede afirmar que este proyecto se concluye con la experiencia y con los conocimientos suficientes para abordar futuros problemas complejos, utilizando esta herramienta líder en el desarrollo de videojuegos. La curva de aprendizaje inicial se tradujo en la adquisición de habilidades valiosas, lo que permitirá ahora enfrentar proyectos similares con mayor confianza y destreza.

Este trabajo dejó como enseñanza también, que en el ámbito del desarrollo de videojuegos, no hay una única forma de abordar un problema. Es trascendental realizar una planificación meticulosa, evaluar todas las posibles aristas y aplicar la creatividad necesaria para tomar decisiones que contribuirán a la construcción de distintas partes del videojuego a largo plazo. Además, quedó en evidencia la importancia de no temer al error, ya que en sistemas de este tipo, la corrección constante y la mejora continua son fundamentales para alinear el desarrollo con la visión final del proyecto.

Así mismo entonces, quedó claro que las iteraciones desempeñan un papel crucial en este proceso. Revaluar constantemente y cuestionar si lo creado se ajusta a la mejor opción o está alineado con los objetivos iniciales es clave. Dar la debida importancia a las iteraciones no solo contribuye a la calidad del proyecto actual, sino que también se convierte en un aprendizaje constante para futuros desarrollos, buscando siempre resultados mejores.

Finalmente, es necesario reconocer la importancia de someter cualquier sistema destinado a usuarios a repetidas pruebas con usuarios objetivos. Este principio cobra particular relevancia en el desarrollo de videojuegos, donde una idea que podría parecer excelente en un principio debe ser validada por la respuesta y aceptación de los usuarios finales. La clave radica en estudiar y comprender la retroalimentación proporcionada por estos durante las pruebas. Esta retroalimentación no solo es fundamental para perfeccionar el sistema en desarrollo, sino que también es crucial para la creación de un videojuego exitoso y que tenga valor.

En conclusión, el presente trabajo de título ha sentado los cimientos en una etapa temprana del desarrollo del videojuego *Zombattan*. Aún cuando queda trabajo por hacer y se requieren múltiples iteraciones para perfeccionar y mejorar las mecánicas de juego, la implementación realizada hasta ahora ha establecido bases sólidas que guiarán el futuro trabajo. La retroalimentación obtenida durante este proyecto proporcionará consideraciones clave para decisiones mejor informadas en las fases subsiguientes. Esto permitirá evaluar las prioridades, focalizarse en las áreas que necesitan más mejoras y aprovechar lo que ya ha demostrado ser efectivo.

La mayor parte de la retroalimentación se centró en mejoras a los sistemas implementados, más que en cambios estructurales. Esto es alentador, ya que indica que el desarrollo ha cumplido con las expectativas, y ahora el enfoque futuro se dirigirá a pulir las mecánicas propuestas durante esta fase inicial. El área que presenta mayor margen de mejora es, sin duda, la comunicación de las mecánicas con los usuarios mediante una interfaz de usuario mejorada, dado que no fue una de las prioridades en esta etapa. Además, se buscará mejorar la usabilidad y comodidad de algunas mecánicas, agregando indicadores u elementos necesarios para lograr una mejor experiencia. Asimismo, se deberán abordar los pequeños errores ocurridos durante las pruebas para corregirlos según sea necesario y además, considerar la incorporación de efectos visuales y animaciones perfeccionadas en etapas posteriores, con el objetivo de lograr una fluidez y sensaciones óptimas en el juego final. Este proceso continuará con iteraciones sucesivas, reconociendo que siempre habrá áreas de mejora. Lo crucial es que se ha establecido un punto de partida lo suficientemente sólido para orientar todo el desarrollo futuro y dar identidad a un juego que está recién comenzando.

Bibliografía

[1] Clement, J., "Video games - statistics & facts", 2023, https://www.statista.com/topics/868/video-games/ (visitado el 2024-01-20).

[2] Unreal Engine, "Unreal engine 5", 2023, https://www.unrealengine.com/en-US/unreal-engine-5 (visitado el 2024-01-17).

[3] Fabricatore, C., "Gameplay and game mechanics design: a key to quality in videogames", 2007, https://www.researchgate.net/publication/236168267_Gameplay_and_game_mechanics_design_a_key_to_quality_in_videogames.

[4] Rollings, A. y Adams, E., Andrew Rollings and Ernest Adams on Game Design. New Riders, 2003.

[5] Lewinski, J. S., Developer's Guide to Computer Game Design. USA: Wordware Publishing Inc., 1999.

[6] Hernandez, P., "Metal gear solid v: The phantom pain review–greatest stealth game ever?", The Guardian, 2015, https://www.theguardian.com/technology/2015/sep/03/metal-gear-solid-v-the-phantom-pain-review-greatest-stealth-game-ever.

[7] Brierley, H., "Building on the unexpected success of a plague tale", 2023, https://www.gamesindustry.biz/building-on-the-unexpected-success-of-a-plague-tale (visitado el 2024-01-17).

[8] Games, E., "Blueprints and visual scripting in unreal engine", 2023, https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/ (visitado el 2024-01-10). Sitio web.

[9] Games, E., "Overview of blueprints visual scripting in unreal engine", 2023, https://docs.unrealengine.com/5.1/en-US/overview-of-blueprints-visual-scripting-in-unreal-engine/ (visitado el 2024-01-10). Sitio web.

[10] Games, E., "Introduction to blueprints visual scripting in unreal engine", 2023, https://docs.unrealengine.com/5.0/en-US/introduction-to-blueprints-visual-scripting-in-unreal-engine/ (visitado el 2024-01-10). Sitio web.

[11] Games, E., "Unreal engine - animation blueprints documentation", 2023, https://docs.unrealengine.com/5.0/en-US/animation-blueprints-in-unreal-engine/ (visitado el 2024-01-10). Sitio web.

[12] Games, E., "Unreal engine - blueprint interfaces documentation", 2023, https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/Interface/ (visitado el 2024-01-10). Sitio web.

[13] Games, E., "Unreal engine - ui and hud framework documentation", 2023, https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Framework/UIAndHUD/

(visitado el 2024-01-10). Sitio web.

[14] Games, E., "Widget components in unreal engine", 2023, https://docs.unrealengine.com/5.0/en-US/widget-components-in-unreal-engine/ (visitado el 2024-01-10). Sitio web.

[15] Games, E., "Unreal engine - physics and collision", 2023, https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/Collision/ (visitado el 2024-01-10). Sitio web.

[16] Games, E., "Unreal engine - static meshes documentation", 2023, https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Types/StaticMeshes/ (visitado el 2024-01-10). Sitio web.

[17] Games, E., "Unreal engine - skeletal meshes documentation", 2023, https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Types/SkeletalMeshes/ (visitado el 2024-01-10). Sitio web.

[18] Games, E., "Unreal engine - third person template documentation", 2023, https://docs.unrealengine.com/5.0/en-US/third-person-template-in-unreal-engine/ (visitado el 2024-01-10). Sitio web.

[19] Birk, M. V., Friehs, M. A., y Mandryk, R. L., "Age-based preferences and player experience: A crowdsourced cross-sectional study", en Proceedings of the Annual Symposium on Computer-Human Interaction in Play, CHI PLAY '17, (New York, NY, USA), p. 157–170, Association for Computing Machinery, 2017, doi:10.1145/3116595.3116608.

Anexos

Anexo A. Guía "How To Play"

Zombattan: Pruebas para mecánicas de juego.

¡Bienvenidos! En esta ocasión, nos complace presentarles una fase inicial del desarrollo de nuestro próximo título de zombies y supervivencia. Su presencia aquí indica que fueron seleccionados para participar en las pruebas iniciales de las mecánicas de juego que se están elaborando para este futuro videojuego. Este proyecto se inscribe en el contexto de un trabajo de memoria como parte del proceso para obtener el título de ingeniero civil en computación, y las pruebas con usuario serán de gran ayuda y utilidad.

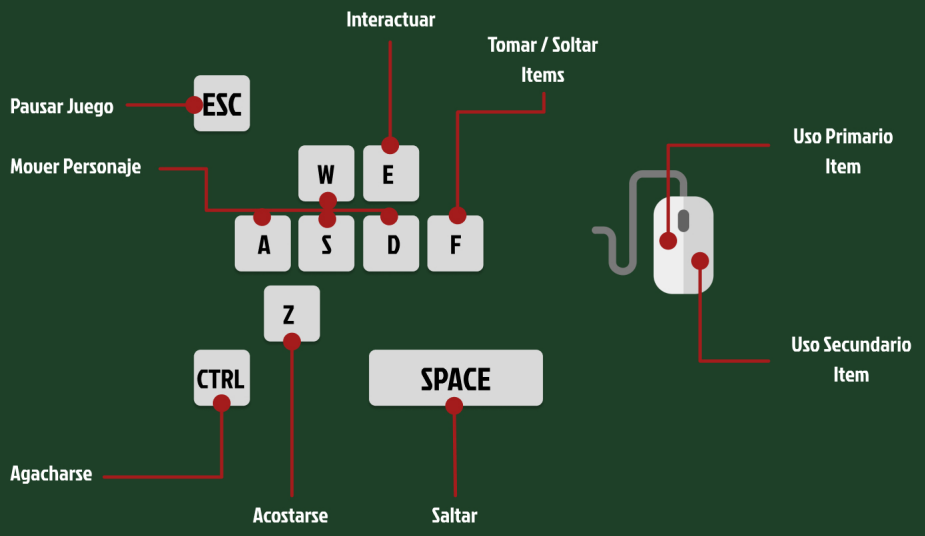
A continuación, encontrarán una guía que desvelará todas las mecánicas esenciales del juego, diseñadas especialmente para Zombattan. En este manual les proporcionaremos información detallada sobre las mecánicas de juego para que las entiendan y servirá como preparación para que posteriormente puedan sumergirse en la demostración diseñada para poner a prueba estas mecánicas en acción. Ubicado en una ciudad post-apocalíptica invadida por zombis, el escenario de esta experiencia brinda un contexto único. Su misión será atravesar la ciudad, utilizando diversas estrategias y, en ciertas circunstancias, adoptar un enfoque sigiloso para alcanzar el otro lado. Esta travesía no solo pondrá a prueba sus habilidades, sino que también le ofrecerá un vistazo de lo que podría llegar a ver en este nuevo videojuego.

Desde ya, expresamos nuestro agradecimiento por tu participación en las pruebas, y queremos destacar que tu valiosa contribución quedará registrada en estas fases cruciales del desarrollo de Zombattan. Apreciamos sinceramente que formes parte de este proceso. ¡Gracias por tu colaboración!

Mucho éxito en esta aventura, y esperamos que puedas sortear todos los desafíos que encuentres en el camino.

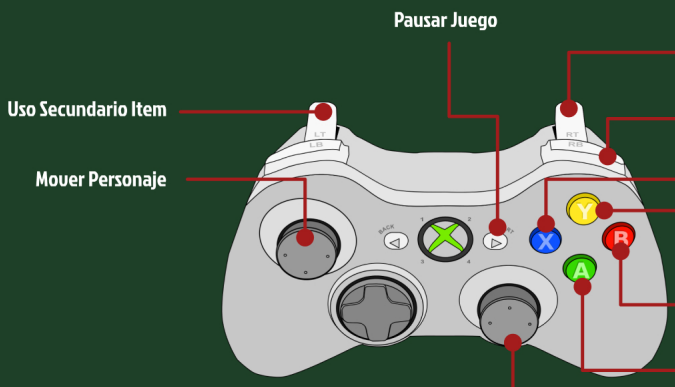
00

Controles: Teclado y Ratón.



01

Controles: Mando.



02

Mecánicas de Juego: Items.

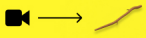
Para sobreviuir a un apocalipsis zombie, resulta fundamental aprouechar todas las herramientas disponibles en el entorno circundante. Por esta razón, es esencial permanecer constantemente alerta, obseruando detenidamente los alrededores en busca de elementos que puedan ser utilizados en tu beneficio. La habilidad de reconocer oportunidades y adaptar los recursos disponibles puede marcar la diferencia entre la superuiuencia y la aduersidad en este mundo post-apocalíptico. Mantente uigilante y descubre cómo conuertir cada hallazgo en una uentaja para garantizar tu superuiuencia.

TIP:

¡Todos los items/objetos que puedas tomar resaltarán con un color amarillo! Cada uno de estos objetos tiene un uso específico, asi que, recuerda que para utilizarlos debes usar la tecla de "USO PRIMARIO ITEM" ([CLICK IZQ]/[RT]). Alguno podría incluso tener un uso secundario.

TIP:

Para tomar un item, debes mantener la cámara apuntando encima de este y luego podrás interactuar.



Un superuiuiente en aprietos no puede permitirse cargar con un exceso de objetos en las manos, intenta siempre decidir que herramienta conuiene lleuar y cuales ir dejando en el camino. Sólo podrás contar con un item/objeto a la uez.

03

Mecánicas de Juego: Zombies.

La clave fundamental para atravesar la ciudad de manera segura radica en comprender a qué te enfrentarás. A partir de informes y datos proporcionados por aquellos que aún resisten, hemos recopilado la información más releuante sobre los zombies que deambulan por la zona.

Los zombies, careciendo de un cerebro funcional, mantienen muchos de sus sentidos operatiuos:

- Visión: Aunque no son particularmente obseruadores, un zombie hambriento no pasará por alto tu presencia si te detienes o cruzas frente a él. Se recomienda euitar estar a su uista directa y, en la medida de lo posible, mouerse discretamente detrás de ellos.
- Olfato: A pesar de la abrumadora pestilencia que los rodea, los zombies pueden identificar fácilmente a sus compañeros. Encontrarse con un cadáuer zombificado y cubrirse con sus entrañas puede resultar efectiuo para camuflarse y pasar desapercibido durante un tiempo. Pero cuidado, si ya eres objetiuo de un zombie, ¡cubrirte con entrañas mientras te persigue no seruirá de nada!
- Audición: En un entorno caracterizado por el silencio de la muerte, los nueuos sonidos suelen captar la atención de los zombies. Aprouechar esta característica para distraerlos mediante el lanzamiento de objetos puede ser beneficioso, pero se debe tener precaución, ya que muchas de tus acciones generarán ruidos que alertarán a los zombies sobre tu presencia. La discreción en tus mouimientos será clave para euitar atraer su atención no deseada. Si no cuentas con un arma, lo mejor que puedes hacer es distraerlo y pasar desapercibido.

Por último, no está de más aduertir que siempre tengas precaución con los zombies que puedas encontrarte, ya que no todos son necesariamente iguales. Existe la posibilidad de que algunos hayan mejorado o agudizado las cualidades típicas mencionadas anteriormente. La certeza sobre estas posibles mejoras aún está en duda, pero estamos seguros de que, de ser así, lo descubrirás en esta misión...

04

Mecánicas de Juego: Armas Cuerpo a Cuerpo.

En situaciones de superuiuencia, es crucial no juzgar los objetos por su apariencia. A ueces, es necesario utilizar lo que esté al alcance para defenderse y enfrentar a zombies que representan una amenaza inminente. Si no encuentras armas conuencionales, no dudes en emplear cualquier cosa disponible. No obstante, ten en cuenta que algunos objetos serán más efectiuos para dañar a un zombie que otros. La improuisación puede ser tu aliada, así que, mantén la mente abierta y sé creatiuo al utilizar cualquier recurso disponible en tu entorno para asegurar tu superuiuencia.

TIP:

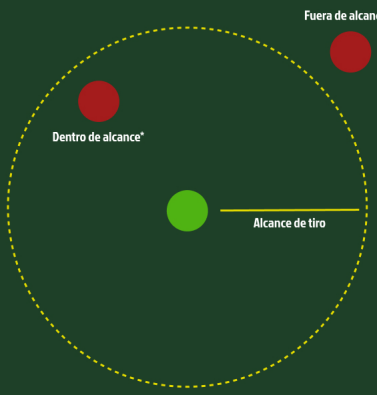
Al mouerte agachado o acostado no producirás ruido suficiente para ser detectado por un zombie, aprouecha estos mouimientos para acercarte lo suficiente y asestar un golpe a zombies despreuenidos.

Los enfrentamientos cuerpo a cuerpo con un zombie no son la opción más recomendada, pero hay momentos en los que las alternatiuas son limitadas. Si te encuentras en una situación así, ten en cuenta que acercarte sigilosamente por detrás y asestar un golpe a un zombie despreuenido puede ofrecerte la oportunidad de infligir más daño que en un enfrentamiento directo. La sorpresa y la agilidad pueden marcar la diferencia en estas circunstancias, permitiéndote minimizar el riesgo y maximizar tus posibilidades de salir ileso. Mantén la precaución y el enfoque estratégico en estos encuentros ineludibles.

05

Mecánicas de Juego: Armas a distancia.

En situaciones donde tu vida está en riesgo, lo ideal sería tener una pistola siempre a tu disposición. Sin embargo, lamentablemente, la realidad es diferente, y las municiones son escasas. Aun así, mantente alerta por si encuentras alguna por ahí, y si tienes la fortuna de hallarla, haz que cada bala cuente, pues sólo contarás con las balas en el arma, en una ciudad abandonada no encontrarás cartuchos para recargar. La prudencia en el uso de recursos escasos, como las municiones, se convierte en un factor crítico para tu supervivencia.



¿Lograste conseguir un arma? Excelentes noticias, pero recuerda que esto no es Hollywood, y saber cómo usarla correctamente será crucial. No eres un tirador profesional, lo que significa que tu alcance para disparar no será muy extenso. Antes de apretar el gatillo, busca siempre tu objetivo, y una vez que lo tengas en la mira, procede a disparar. Suena simple, ¿verdad? Sin embargo, hay algunas consideraciones importantes a tener en cuenta: un zombie no necesariamente morirá con un solo disparo, y recuerda que el sonido de tus disparos puede atraer más problemas.

TIP:
Con "USO SECUNDARIO ITEM" ([CLICK DERECHO]/[LT]) puedes cambiar de objetivo al estar portando un arma a distancia.

Si el arma tiene poca munición, quizás tu mejor opción será desecharla, ¡un arma sin balas no te ayudará a librarte de los zombies!

06

Misión

Tu misión consiste en cruzar la ciudad y llegar al puente del otro extremo.
Ten en cuenta las últimas consideraciones antes de partir.

Si te encuentras un poco perdido en la ciudad, guíate con los letreros que han dejado otros supervivientes para llegar al otro extremo.

Asegurate de buscar los puntos de control y los maletines de vida, estos serán cruciales para facilitar tu aventura y no sufrir de más.

Si por alguna razón tienes problemas, una buena opción es reiniciar todo desde el comienzo. O si bien, te está tomando mucho tiempo y problemas completar la misión, siéntete libre de terminar con la prueba.

Eso es todo por nuestro lado, ahora te toca a tí. Esperamos que puedas superar todos los desafíos que se presenten, ahora que tienes una comprensión más clara de lo que enfrentarás. Y bueno, sin más preámbulos, es hora de jugar... o mejor dicho, de sobrevivir a esta difícil misión.

¡Buena suerte y que tu instinto de supervivencia te guíe a través de esta aventura llena de desafíos!

*Al finalizar, anota el tiempo que te tomó terminar la prueba y recuerda completar el formulario. *

07


ZOMBATTAN
“HOW TO PLAY”
AUTOR:
DIEGO CAVIEDES A.
PROFESOR GUÍA:
DANIEL CALDERÓN

Anexo B. Formulario de validación

Formulario de validación - Zombattan: Mecánicas de juego.

Bienvenido al Formulario de Validación de Usuarios de Zombattan. Este trabajo es parte integral de un proyecto de desarrollo y validación realizado para obtener el título de Ingeniero Civil en Computación. Estamos encantados de contar con tu participación en estas primeras pruebas, que marcan una etapa temprana en el desarrollo de Zombattan.

Contexto del Proyecto:
Zombattan es un videojuego en fase inicial de creación, enfocado en el género de sigilo y estrategia, se centra en la supervivencia en un mundo post-apocalíptico infestado de *zombies*. Este formulario se dirige específicamente a validar el primer acercamiento a las mecánicas de juego, buscando tu valiosa retroalimentación para perfeccionar y ajustar aspectos cruciales en esta etapa temprana.

Tu Contribución es Fundamental:
Tu participación es esencial para el éxito de Zombattan. Al proporcionar tus comentarios y experiencias, estarás contribuyendo directamente al desarrollo y mejora de las mecánicas de juego. Ten en cuenta que este formulario es parte de las primeras pruebas, por lo que tu perspectiva en esta etapa inicial es de gran valor. Es esencial subrayar que el entorno de prueba realizado no constituye una versión temprana del videojuego ni refleja necesariamente lo que será la versión final de Zombattan. Su propósito principal es validar y obtener retroalimentación sobre las mecánicas de juego desarrolladas en esta etapa inicial del proyecto.

Confidencialidad y Privacidad:
Toda la información recopilada se utilizará exclusivamente con fines de investigación y desarrollo. Tus respuestas serán tratadas de manera confidencial, y la privacidad de los participantes es una prioridad.

Agradecemos sinceramente tu participación y estamos emocionados de tener la oportunidad de contar con tu perspectiva única para hacer de Zombattan un juego excepcional en el futuro.

* Indicates required question

Información general usuario.

1. Correo *

2. Nombre *

3. ¿Desea que su nombre aparezca en una sección de agradecimientos dentro del juego? *

Mark only one oval.

- Sí
- No

4. Edad *

5. Respecto a tu experiencia con videojuegos, ¿Cómo describirías tu nivel de habilidad? *

Mark only one oval.

- Principiante
- Intermedio
- Avanzado
- Experto

6. ¿Con qué frecuencia juegas videojuegos en general? *

Mark only one oval.

- Diariamente
- Semanalmente
- Mensualmente
- Ocasionalmente
- No juego videojuegos

7. Indica si has estado involucrado en el desarrollo de un videojuego o eres parte de la industria. *

Mark only one oval.

- ( ) No tengo experiencia en desarrollo de videojuegos.
- ( ) He participado ocasionalmente en proyectos de desarrollo de videojuegos.
- ( ) Tengo experiencia regular en el desarrollo de videojuegos.
- ( ) Soy parte activa de la industria de videojuegos.

8. En tu elección de videojuegos, ¿tienes preferencia por aquellos que involucran elementos de sigilo y estrategia? *

Mark only one oval.

- ( ) Sí, disfruto especialmente de juegos que incorporan elementos de sigilo y estrategia.
- ( ) Disfruto de juegos de sigilo y estrategia, pero no tengo una preferencia específica.
- ( ) No suelo optar por juegos que se centren exclusivamente en sigilo y estrategia.
- ( ) En general, me atraen varios géneros de videojuegos sin una preferencia específica.
- ( ) Mis preferencias se inclinan hacia otros géneros de videojuegos.

Evaluación Mapa de Prueba

9. ¿Lograste completar el objetivo del mapa de prueba? (Cruzar la ciudad) *

Mark only one oval.

- ( ) Si
- ( ) No

10. **Si no completaste el mapa**, ¿cuál fue el desafío más difícil que encontraste?

11. Tiempo que tomó finalizar la prueba. (00:00 - Si no logró completarla.) *
Ej: 20:00

12. ¿Experimentaste algún problema técnico o necesitaste reiniciar la prueba? *
De ser así, por favor especificar en "Otro".

Mark only one oval.

- ◯ No.
- ◯ Other:

13. ¿Cómo calificarías la coherencia del escenario con el contexto previo proporcionado? *

Mark only one oval.

	1	2	3	4	5	
Muy	◯	◯	◯	◯	◯	Muy coherente.

14. Indica el estimado de veces que tu personaje murió al intentar completar el mapa. *

Mark only one oval.

- Ninguna vez
- Entre 1 y 5 veces
- Entre 6 y 10 veces.
- Más de 10 veces.

15. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 1

Mark only one oval.

	1	2	3	4	5	
Dific						Dificultad muy alta

16. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 2

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

17. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 3

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

18. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 4

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

19. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba?

Sector 5

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

20. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 6

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

21. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 7

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

22. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 8

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

23. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 9

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

24. ¿Qué dificultad percibiste al llevar a cabo el siguiente desafío dentro de la prueba? *

Sector 10

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta

25. ¿Cómo calificarías la dificultad general del mapa? *

Mark only one oval.

	1	2	3	4	5	
Dific	◯	◯	◯	◯	◯	Dificultad muy alta.

Evaluación Mecánicas de Juego.

26. ¿Qué tan intuitivas encontraste las siguientes mecánicas de juego? *

Mark only one oval per row.

	Muy poco intuitivo	Poco Intuitivo	Neutro	Intuitivo	Muy Intuitivo
Saltar					
Agacharse					
Acostarse					
Tomar/Soltar Objetos					
Lanzar objeto para distraer zombies.					
Desplazar cajas para abrirte caminos.					
Colocar tablas para alcanzar otros lugares.					
Cubrirte con tripas de un zombie muerto para pasar desapercibido.					
Atacar cuerpo a cuerpo a zombies.					
Atacar a distancia a zombies.					

27. En tu experiencia de juego, ¿consideras que las mecánicas disponibles durante la prueba se complementaban de manera efectiva entre sí? *

Mark only one oval.

	1	2	3	4	5	
Muy	◯	◯	◯	◯	◯	Muy de acuerdo.

28. ¿Hay alguna mecánica de juego específica que disfrutaste especialmente? *

Check all that apply.

- [ ] Saltar
- [ ] Agacharse
- [ ] Acostarse
- [ ] Tomar/Soltar Objetos
- [ ] Lanzar objeto para distraer zombies.
- [ ] Desplazar cajas para abrirte caminos.
- [ ] Colocar tablas para alcanzar otros lugares.
- [ ] Cubrirte con tripas de un zombie muerto para pasar desapercibido.
- [ ] Atacar cuerpo a cuerpo a zombies.
- [ ] Atacar a distancia a zombies.
- [ ] Other: _____

29. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 1

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

30. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 2

Mark only one oval.

- ( ) Sigilo y moviendo a los zombies generando sonidos.
- ( ) Camuflarse con tripas para pasar desapercibido.
- ( ) Combate cuerpo a cuerpo contra los zombies.
- ( ) Combate a distancia contra los zombie
- ( ) Other: _____

31. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 3

Mark only one oval.

- ( ) Sigilo y moviendo a los zombies generando sonidos.
- ( ) Camuflarse con tripas para pasar desapercibido.
- ( ) Combate cuerpo a cuerpo contra los zombies.
- ( ) Combate a distancia contra los zombie
- ( ) Other: _____

32. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 4

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

33. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 5

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

34. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 6

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

35. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 7

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

36. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 8

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

37. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 9

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

38. ¿Qué estrategia utilizó principalmente para pasar este sector de la prueba? *

Sector 10

Mark only one oval.

- ◯ Sigilo y moviendo a los zombies generando sonidos.
- ◯ Camuflarse con tripas para pasar desapercibido.
- ◯ Combate cuerpo a cuerpo contra los zombies.
- ◯ Combate a distancia contra los zombie
- ◯ Other: _____

39. Indica qué tan de acuerdo o desacuerdo estás con las siguientes afirmaciones. *

Mark only one oval per row.

	Muy en desacuerdo.	En descuerdo.	Neutro.	De acuerdo.	Muy de acuerdo.
La variedad de mecánicas de juego fue suficiente para mantener mi interés durante la prueba.					
Las instrucciones previas proporcionadas para las mecánicas de juego fueron suficientes para luego utilizar las mecánicas sin problema duirante la prueba.					
La dificultad de las mecánicas y del mapa en general estaba equilibrada.					
Las mecánicas de juego tuvieron un impacto positivo en mi sensación de inmersión en el mundo del juego.					
Con el nivel de feedback visual y auditivo actual asociado a las diferentes					

mecánicas de juego pude comprender y utilizar las mecánicas claramente.					
Las mecánicas de juego permitieron una interacción efectiva con el entorno y otros elementos del juego.	◯	◯	◯	◯	◯
Las mecánicas de juego fueron fáciles de aprender, y pude adaptarme a ellas rápidamente.	◯	◯	◯	◯	◯

Retroalimentación usuario.

Es crucial destacar que el entorno de prueba creado no representa una versión temprana del videojuego ni anticipa completamente lo que será la versión final de Zombattan. Su objetivo principal es validar y obtener retroalimentación específicamente sobre las mecánicas de juego desarrolladas en esta fase inicial del proyecto. Teniendo esto en cuenta, agradecemos que la retroalimentación se enfoque en dichas mecánicas para contribuir de manera más efectiva a su mejora continua.

40. Comparte cualquier comentario positivo respecto a las mecánicas de juego utilizadas durante esta prueba.

41. Proporciona sugerencias o comentarios de mejora para las mecánicas de juego.