JUEGO CON IA GENERATIVA
GAME WITH GENERATIVE AI

TRABAJO FIN DE GRADO
CURSO 2024-2025

AUTORES
ADRIÁN SALINAS RUIZ 6.4
DANIEL GALLEGO BADIA 6.4
ZAKARIAE LAKTIOUET SAIDI 7.0

DIRECTOR
IVÁN GARCÍA-MAGARIÑO GARCÍA

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

JUEGO CON IA GENERATIVA
GAME WITH GENERATIVE AI

Trabajo de Fin de Grado en Ingeniería Informática

Autores
ADRIÁN SALINAS RUIZ
DANIEL GALLEGO BADIA
ZAKARIAE LAKTIOUET SAIDI

DIRECTOR
Iván García-Magariño García

Convocatoria: Mayo 2025

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

20 de 06 de 2025

RESUMEN

Título

La inteligencia artificial es una de las tecnologías más utilizadas actualmente y está comenzando a aparecer en el mundo de los videojuegos. Esto puede provocar grandes cambios a la hora de entretener a los jugadores. Con la inteligencia artificial generativa se podría tener conversaciones al instante con cualquier NPCs del juego como si estuvieras hablando con una persona real, pero con el contexto del juego en cuestión. Es por ello que implementamos un prototipo funcional en Unity que muestra una ligera idea de lo que puede ser el futuro de los videojuegos. La memoria detalla el proceso completo de desarrollo: desde la planificación inicial y el diseño artístico hasta la implementación técnica del sistema de diálogo basado en IA.

Entre los hallazgos más relevantes se concluye que la integración de IA generativa en videojuegos no solo mejora la interactividad, sino que plantea nuevas formas de diseño narrativo y adaptativo, abriendo nuevas vías para la creación de experiencias únicas y personalizadas en el futuro de esta industria.

Palabras clave

Inteligencia artificial generativa, videojuegos, narrativa interactiva, NPCs, Unity.

ABSTRACT

Title

Artificial intelligence is currently one of the most widely used technologies and is beginning to emerge in the world of video games. This has the potential to bring about significant changes in how players are entertained. With generative artificial intelligence, it becomes possible to hold instant conversations with any NPC in the game, as if speaking with a real person, but within the context of the game itself. For this reason, we developed a functional prototype in Unity that offers a glimpse into what the future of video games could look like.

This report details the complete development process: from initial planning and artistic design to the technical implementation of the AI-based dialogue system. Among the most relevant findings, it is concluded that the integration of generative AI in video games not only enhances interactivity, but also introduces new approaches to narrative and adaptive design, opening the door to the creation of unique and personalized experiences in the future of the industry.

Keywords

Generative artificial intelligence, video games, interactive narrative, NPCs, Unity.

ÍNDICE DE CONTENIDOS

Índice de figuras

Capítulo 1 - Introducción

En los últimos años, la inteligencia artificial (IA) ha experimentado un crecimiento exponencial, consolidándose como una de las tecnologías más prometedoras y transformadoras en diversos ámbitos, desde la medicina hasta la educación. En particular, el mundo del desarrollo de videojuegos ha comenzado a explorar sus posibilidades, abriendo nuevas vías para mejorar la interacción, la inmersión y la narrativa dentro de los entornos virtuales.

Este trabajo se centra en el uso de inteligencia artificial generativa aplicada al desarrollo de videojuegos, una rama emergente que permite generar contenido dinámico a partir de entradas del usuario. En este contexto, una de las aplicaciones más interesantes es la posibilidad de mantener conversaciones realistas y contextuales con personajes no jugables (NPCs), dotándolos de una personalidad y capacidad de respuesta mucho más cercana a la de un ser humano.

En este capítulo introductorio se exponen conceptos básicos sobre juegos serios, y se presenta el plan de trabajo tras una etapa de investigación. El segundo capítulo se centra en analizar trabajos relacionados. El tercero explica las tecnologías utilizadas, el diseño global, así como funcionalidades adicionales. El cuarto capítulo presenta los resultados de las pruebas de usabilidad realizadas al finalizar el desarrollo. Por último, el quinto capítulo son las conclusiones.

1.1 Motivación

Con este trabajo buscamos mostrar y abrir una reflexión sobre el impacto que la inteligencia artificial puede tener en el diseño de videojuegos del futuro.

Con este trabajo buscamos mostrar y abrir una reflexión crítica sobre el impacto que la inteligencia artificial, y en particular la inteligencia artificial generativa, puede tener en el diseño y desarrollo de los videojuegos del futuro. La evolución de esta tecnología está transformando rápidamente la forma en que se crean experiencias interactivas, permitiendo mundos más dinámicos, narrativas emergentes y personajes que ya no dependen de diálogos predefinidos, sino que pueden responder de forma contextual, personalizada y coherente con la historia del jugador.

A través de este proyecto, se busca evidenciar el potencial creativo y narrativo que ofrece la IA generativa, así como explorar las oportunidades y desafíos que plantea su implementación en entornos lúdicos. Más allá de lo técnico, el trabajo invita a considerar nuevas formas de interacción entre el jugador y el mundo del juego, que podrían redefinir por completo el rol del diseñador, del guionista y hasta del propio jugador dentro de la experiencia. En definitiva, este proyecto quiere ser un pequeño paso hacia la comprensión de cómo la inteligencia artificial no solo mejora la tecnología del juego, sino también su capacidad para contar historias vivas, responder a la curiosidad humana y construir mundos más creíbles.

A su vez, se pretende indagar sobre cómo se crea un videojuego. Aprender las herramientas básicas para su desarrollo, conocer apartados que se han de tener en cuenta a la hora de hacer un juego entretenido e interesante y enfrentar desafíos nuevos utilizando un entorno poco conocido.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es explorar el uso de inteligencia artificial generativa como herramienta para enriquecer la narrativa e interactividad en videojuegos, mediante el desarrollo de un prototipo funcional. El proyecto busca demostrar cómo los avances en procesamiento de lenguaje natural

pueden integrarse en entornos lúdicos para permitir interacciones más realistas, dinámicas y personalizadas con los NPCs.

Para ello se busca diseñar un conjunto de personajes con contextos diferenciados, lo suficientemente ricos como para permitir el desarrollo de una narrativa emergente y generar múltiples caminos de deducción en función de las respuestas ofrecidas por la IA.

Otro objetivo secundario es aprender a utilizar Unity como plataforma para el desarrollo de videojuegos, con el propósito de adquirir una base sólida en la creación de entornos virtuales interactivos. A través de la exploración y el uso práctico de sus herramientas y funcionalidades, se pretende comprender los principios fundamentales del diseño y la programación de videojuegos.

1.3 Plan de trabajo

El desarrollo del proyecto ha seguido las siguientes etapas para la creación del videojuego:

1.3.1 Definición de Requisitos y Objetivos

El inicio del proyecto comprendió un periodo dedicado a la elaboración de un detallado listado de requisitos y objetivos que se buscaban cumplir con el videojuego.

Esta etapa se orientó hacia la decisión del tipo de juego que queríamos implementar y decidir dónde y cómo podríamos implementar la mecánica de usar la inteligencia artificial generativa.

1.3.2 Aprendizaje y pruebas del Entorno Unity

En esta etapa del proyecto nos dedicamos completamente al aprendizaje del motor de videojuegos Unity, familiarizándonos con sus herramientas y conceptos básicos, así como al análisis de otros conceptos similares y artículos relevantes con el fin de obtener ideas para desarrollar en el juego. Comenzamos con la realización de pequeñas pruebas para comprobar el funcionamiento y sentar las bases para un posterior desarrollo. Posteriormente, comenzamos a implementar una escena básica (Scene según la terminología de Unity), correspondiente a un pueblo que actuará como mapa principal y para hacer pruebas antes de aplicarlas en el juego final.

Cada concepto del juego se desarrolla en forma de prototipo exploratorio: se parte de una versión muy básica a la que se le van añadiendo progresivamente funcionalidades, mejorando la interacción, el diseño o la usabilidad.

1.3.3 División del Trabajo

Con la información recopilada y los objetivos definidos, se procedió a la planificación de la división del trabajo entre el diseño artístico y las mecánicas del juego. Se establecieron tareas específicas para facilitar la distribución equitativa y efectiva de tareas. Decidimos usar GitHub [4] para mantener un control de versiones y para mantener una conexión entre cada tarea realizada.

1.3.4 Revisión y Corrección de Errores

Una vez finalizada la implementación del juego, se llevó a cabo una fase de revisión exhaustiva. Durante esta etapa se realizaron pruebas orientadas a identificar y corregir posibles errores, con el objetivo de asegurar la estabilidad general y la funcionalidad completa del sistema. Este proceso fue esencial para garantizar una

experiencia de usuario fluida y sin interrupciones, contribuyendo a la calidad final del proyecto.

Para organizar y gestionar de forma eficiente el desarrollo del proyecto, se adoptó la metodología ágil SCRUM, que facilitó el trabajo colaborativo y la toma de decisiones. A través de reuniones periódicas, se evaluó el progreso de cada parte del desarrollo, se compartieron posibles dificultades y se ajustó la planificación cuando fue necesario. Se establecieron objetivos a corto plazo, priorizando las tareas más relevantes en función de su impacto y coste temporal. Esta dinámica también permitió adaptar el alcance del proyecto en tiempo real, tomando decisiones como recortar o reajustar ideas en caso de que ciertos elementos consumieran más recursos de los inicialmente previstos, especialmente debido a imprevistos técnicos o de diseño.

1.3.5 Fases del proyecto

Se adjunta a continuación una imagen del diagrama Gantt realizado en excel con la planificación temporal que hemos seguido.

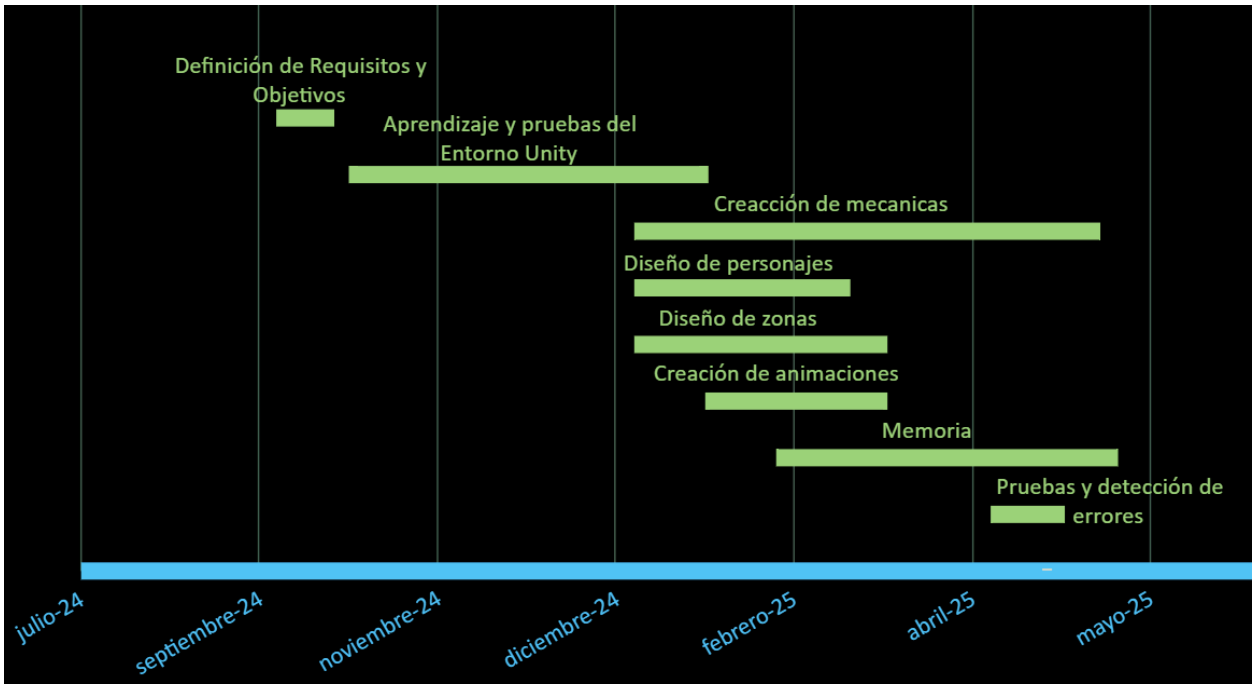


Figura 1-1. Diagrama de Gantt.

Capítulo 2 - Trabajos relacionados o similares

Centrándonos en las posibilidades que implica, es interesante el artículo [7] en el cual aborda la implementación de inteligencia artificial generativa para mejorar las interacciones con NPCs en videojuegos de rol. Se parte del problema de que, en muchos juegos, los NPCs presentan diálogos predefinidos, repetitivos y poco realistas, lo que reduce la inmersión del jugador. Como solución, los autores proponen un sistema que genera diálogos y misiones personalizadas mediante el uso de un modelo de lenguaje avanzado capaz de generar respuestas naturales (Google Gemini 1.0 Pro) y una técnica de procesamiento del lenguaje que permite evaluar la similitud semántica entre frases, ayudando a conectar el contexto de la conversación con las misiones generadas (Sentence-BERT).

Para que esto pueda funcionar es necesario crear a los NPCs con identidades completas (nombre, rol, personalidad, objetivos, relaciones con otros personajes) y se utilizan datasets de diálogos reales de NPCs disponibles en Hugging Face [1]. Tras definir el perfil del NPC, se entrena el modelo para que sea capaz de mantener una conversación contextual con el jugador, generar misiones relacionadas directamente con lo conversado y evaluar la coherencia entre conversación y misión usando Sentence-BERT.

El sistema fue probado simulando conversaciones entre un jugador y el NPC. Después de varios mensajes, el NPC preguntaba si el jugador quería una misión. Si aceptaba, se generaban misiones distintas basadas en la conversación. Usando SBERT, se comprobó que las misiones generadas tenían un nivel de similitud contextual con el diálogo de hasta un 62,85%, lo cual es considerado un resultado bastante positivo. Aunque el enfoque aún es teórico y está pensado para un entorno de jugador único

(*singleplayer*), los resultados abren la puerta a futuras aplicaciones más complejas y realistas en videojuegos.

También es interesante el artículo [12] que aborda la rápida evolución de la inteligencia artificial generativa y las tecnologías de realidad virtual (VR) y cómo estas están transformando tanto la educación como la industria del juego. En este contexto, se plantea la necesidad de enriquecer el aprendizaje basado en juegos mediante entornos inmersivos e interactivos. Se destaca que, a pesar del auge de estas tecnologías, aún existen pocos estudios que exploren la integración simultánea de AI generativa y VR para crear experiencias de aprendizaje verdaderamente inmersivas.

La propuesta central es la creación de "LearningverseVR" [12], una plataforma de aprendizaje basada en juegos que combina AI generativa y VR. La plataforma está diseñada para ofrecer un entorno de aprendizaje en el que los alumnos puedan interactuar tanto con objetos virtuales como con otros usuarios de manera dinámica. Técnicamente, el sistema se construye utilizando Unity para el desarrollo del cliente y tecnologías como Python, Flask y MySQL para el backend, permitiendo funcionalidades multijugador en línea y garantizando una experiencia interactiva y escalable.

La plataforma utiliza modelos de lenguaje a gran escala (LLM) como GPT-3.5 de OpenAI junto con modelos locales (por ejemplo, ChatGLM) para dotar de "inteligencia" a los NPCs. Estos NPCs son creados dinámicamente, con personalidades, antecedentes y respuestas que no siguen un guión preestablecido. Además, se incorporan módulos de generación de texto a imagen (como Stable Diffusion) y conversión de texto a voz, lo que permite animar y dar voz a dichos personajes en tiempo real.

En este artículo [13] se presenta una propuesta innovadora para llevar personajes de videojuegos al espacio social real mediante agentes conversacionales impulsados por modelos de lenguaje generativo (LLMs), como GPT-3. El enfoque central es el desarrollo de "Storytelling Community AI Agents" (SCAs), personajes ficticios diseñados con una narrativa propia que interactúan con los miembros de una comunidad online, en este caso dentro de un servidor de Discord vinculado a un videojuego llamado "DE".

Para ello, los autores proponen una metodología de "ingeniería narrativa" compuesta por tres fases: diseño del personaje y su historia, presentación de historias en vivo que los agentes comparten con la comunidad, permitiendo a los usuarios influir en su desarrollo a través de decisiones colectivas, y comunicación directa con los jugadores, permitiendo conversaciones interactivas que refuercen la credibilidad del personaje.

Los resultados se evaluaron a través de encuestas y entrevistas a usuarios que interactuaron con dos personajes, David y Catherine. Los datos muestran que los agentes narrativos superaron a los chatbots tradicionales (como un bot de referencia llamado Jerry) en cuanto a credibilidad, empatía y conexión emocional. Aspectos como la coherencia narrativa, la expresión emocional, la desobediencia razonada y la conexión con otros personajes contribuyeron a una experiencia más inmersiva. Catherine, por ejemplo, fue percibida como más humana y comprometida con su historia, lo que generó mayor implicación entre los jugadores.

Esta investigación demuestra que la integración de personajes con capacidad narrativa basada en IA puede enriquecer significativamente las comunidades de jugadores, fomentando interacciones más profundas y experiencias sociales únicas.

Asimismo, plantea nuevas perspectivas sobre el papel de la IA generativa en la creación de personajes creíbles y dinámicos dentro de entornos de juego.

Desde el punto de vista artístico, nos hemos inspirado claramente en el estilo visual de los juegos clásicos de Pokémon [3] para la consola Nintendo DS, como Pokémon Diamante, Perla o Platino. Estos títulos destacan por su estética pixel art en vista cenital, con entornos coloridos y personajes fácilmente reconocibles a pesar de su baja resolución, lo que les otorga un encanto atemporal.

Capítulo 3 - Juego con IA generativa sobre descubrimiento de asesino

3.1 Introducción

En este capítulo se describe en detalle el proceso de diseño y desarrollo del videojuego, desde su concepción inicial hasta su implementación técnica. El proyecto se ha centrado en la creación de un juego de estilo "*Cluedo*" [8], en el que el jugador asume el papel de un detective que llega a un pequeño pueblo tras la noticia de un asesinato. La propuesta combina elementos de investigación, deducción y narrativa interactiva, incorporando además inteligencia artificial generativa para enriquecer la experiencia de juego.

El objetivo principal es que el jugador averigüe quién es el asesino, interactuando con los distintos habitantes del pueblo. Para ello, deberá formular preguntas, analizar las respuestas proporcionadas por los personajes y detectar posibles incongruencias o contradicciones en sus coartadas. Cada personaje cuenta con una historia, personalidad y posibles motivos, y gracias al uso de IA generativa, sus respuestas no son fijas ni repetitivas, lo que añade un elemento de dinamismo y rejugabilidad al juego.

Una vez el jugador cree haber reunido suficientes pruebas e indicios, puede comunicar su hipótesis final al alcalde del pueblo, quien se encargará de determinar si la acusación es correcta. Si el jugador ha logrado identificar al verdadero culpable, gana la partida; de lo contrario, el asesino queda impune y el juego finaliza sin éxito.

Este enfoque no solo plantea un reto lógico para el jugador, sino que también busca replicar una sensación real de investigación, al permitir interacciones más abiertas y menos guiadas que en los videojuegos tradicionales. A lo largo del capítulo se explicarán las mecánicas implementadas, el diseño de los personajes, los sistemas de diálogo, y el funcionamiento interno que hace posible esta experiencia narrativa centrada en el descubrimiento del asesino.

3.2 Tecnologías empleadas

En esta sección explicaremos cada tecnología y para qué ha sido utilizada.

3.2.1 Unity

Unity [14] es un motor de desarrollo en tiempo real que permite crear videojuegos en 2D y 3D, otorgando al desarrollador multitud de herramientas para la gestión de proyectos y creación de scripts.

Se ha utilizado como base del desarrollo permitiendo editar el apartado visual del juego, añadir componentes del editor para poder crear objetos que interactúen entre ellos y crear scripts. También nos ha permitido generar un entorno en el que realizar pruebas de todo lo programado. Y finalmente, también nos ha permitido generar el ejecutable del juego.

3.2.2 Unity Hub

Unity Hub [15] es una aplicación que actúa como una plataforma centralizada para la gestión del entorno de desarrollo con Unity. Su principal función es facilitar la instalación de versiones del motor, así como la creación, apertura, administración y organización de proyectos. Se ha usado para la gestión de versiones del propio editor de Unity y la descarga de componentes que han ayudado el desarrollo del proyecto.

3.2.3 Visual Studio Code

Visual Studio Code [10] es un editor de código fuente de software libre y multiplataforma con un gran número de extensiones, que dan la posibilidad de escribir y ejecutar código en cualquier lenguaje de programación.

Se ha utilizado como entorno de desarrollo de código, al tener extensiones de C#, python y Unity que han permitido la creación de los scripts del juego, ayudando a la hora de mostrar los errores.

3.2.4 GitHub

GitHub [4] es una plataforma para crear, compartir y gestionar proyectos abiertos de desarrollo de software caracterizándose por su capacidad de llevar un control de versiones.

Se ha utilizado para mantener un repositorio compartido entre los tres integrantes del proyecto, llevando un control de versiones y pudiendo trabajar a la vez en el mismo proyecto sin solaparse mutuamente.

3.2.5 GitHub Desktop

Github Desktop [5] es una aplicación que proporciona una interfaz gráfica para trabajar con repositorios de Git, facilitando así el control de versiones y la colaboración en proyectos de desarrollo.

Se ha utilizado para gestionar los cambios de código, establecer la conexión con el repositorio de GitHub y realizar todas las funciones de Git de forma más cómoda.

3.2.6 LibreSprite

LibreSprite [9] es un editor gráfico de código abierto especializado en la creación de arte pixelado y sprites para videojuegos.

Se ha utilizado para diseñar a todos los personajes y para crear frames del movimiento para poder animar a personajes desde Unity.

3.2.7 Freesound

Freesound [2] es una plataforma en línea colaborativa que funciona como un repositorio de efectos de sonido y muestras de audio.

Se ha utilizado para añadir efectos de sonido al juego, como los pasos del personaje principal o al entrar y salir de las casas.

3.2.8 Pinterest

Pinterest [11] es una plataforma digital que funciona como un motor de descubrimiento visual, permitiendo a los usuarios encontrar, guardar y organizar ideas en forma de imágenes o "pins".

Se ha utilizado para hacer los exteriores e interiores de las casas.

3.2.9 GIMP

GIMP [6] es un programa de edición de imágenes de código abierto y distribución gratuita, ampliamente utilizado para tareas de retoque fotográfico, composición de imágenes, diseño gráfico y creación de recursos visuales para proyectos digitales.

Se ha utilizado para editar las imágenes de las casas y sus interiores.

3.2.10 Ollama

Ollama es un programa que permite instalar modelos ya entrenados de inteligencia artificial en tu ordenador, en concreto en el proyecto se ha usado para instalar el modelo llama3 y así conseguir que los personajes no jugables puedan responder a las preguntas que le realice el jugador, la respuesta que ellos dan se basa en un prompt que se explicará más tarde.

3.3 Personajes

En esta sección explicaremos todo lo relacionado con cada personaje, incluyendo su descripción, personalidad y su aspecto físico.

El diseño de los personajes se ha realizado completamente desde cero, sin recurrir a bancos de sprites ni plantillas preexistentes. Para ello, se ha utilizado LibreSprite [9], una herramienta especializada en arte en pixel art. Gracias a esta aplicación, se han podido dibujar pixel a pixel cada uno de los personajes del proyecto, desarrollando tanto sus frames estáticos como los necesarios para sus animaciones de movimiento.

El proceso ha incluido el estudio de proporciones y la representación visual de los diferentes roles de los personajes (por ejemplo, distinguir a un pescador de una profesora solo por su vestimenta). Se ha buscado mantener un estilo visual coherente con la estética general del juego, con una clara inspiración en los juegos retro de Pokemon [3].

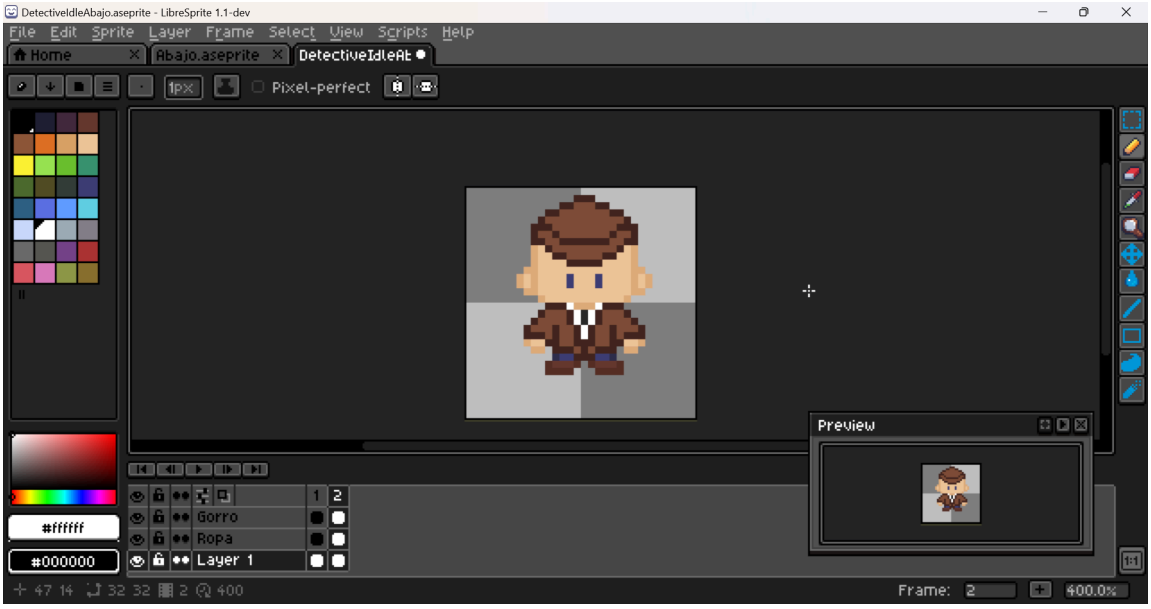


Figura 3-1. Ejemplo en LibreSprite.

3.3.1 Samuel Vega (Detective)

Samuel Vega es el personaje jugable y protagonista. Es un detective experimentado que llega al pueblo tras ser llamado para investigar un misterioso asesinato. Su diseño se basa en un típico detective de novela policiaca con una gabardina larga y en Sherlock Holmes con su sombrero. Es el único personaje que tiene animaciones de andar y de idle, ya que es el único que se puede desplazar.

Figura 3-2. Samuel Vega (Detective) de frente.

Figura 3-3. Samuel Vega (Detective) de espaldas.

Figura 3-4. Samuel Vega (Detective) derecha.

Figura 3-5. Samuel Vega (Detective) izquierda.

Figura 3-6. Samuel Vega (Detective) sprite sheet con todas sus animaciones.

3.3.2 Isidro Varela (Alcalde)

Isidro Varela es el alcalde del pueblo en el que se remonta el juego, conoce a todos los habitantes ya que este pueblo es demasiado pequeño, en el presente juego actúa de juez, donde el Jugador que es un detective sepa con certeza quien es el asesino deberá comunicarlo al alcalde y este mismo decidirá si la respuesta es correcta o en caso contrario reducirá un intento, que en caso de llegar a 0 intentos el Alcalde le dirá al jugador que ha perdido.

Su diseño se basa en un típico hombre de mediana edad trajeado.

Figura 3-7. Isidro Varela (Alcalde) de frente.

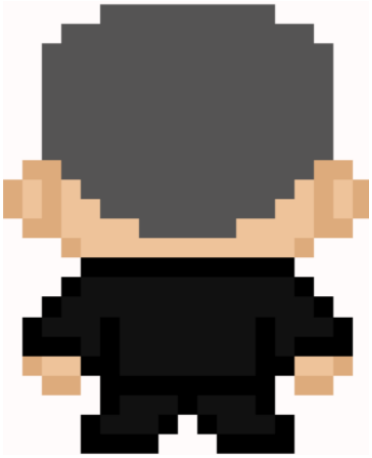


Figura 3-8. Isidro Varela (Alcalde) de espaldas.

Figura 3-9. Isidro Varela (Alcalde) derecha.

Figura 3-10. Isidro Varela (Alcalde) izquierda

3.3.3 Domingo Carrasco (Pescador)

Domingo Carrasco es el pescador del pueblo. Nacido en el pueblo, conoce todos sus secretos. Le encanta hablar de pesca y de sus mejores capturas, se pasa el día pescando en la parcela de su vieja casa para luego venderle algunos peces al chef.Es Desconfiado con extraños, pero se vuelve amable si ganas su confianza.

Figura 3-11. Domingo Carrasco (Pescador) de frente.

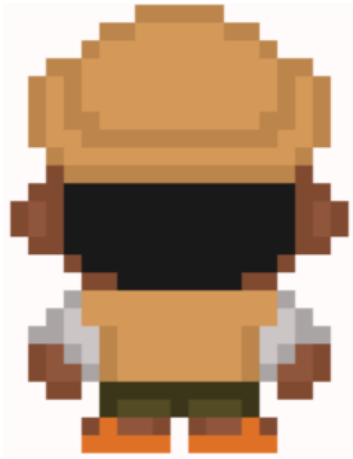


Figura 3-12. Domingo Carrasco (Pescador) de espaldas.

Figura 3-13. Domingo Carrasco (Pescador) derecha.

Figura 3-14. Domingo Carrasco (Pescador) izquierda.

3.3.4 Greta Hoffmann (Extranjera)

Greta Hoffmann es una extranjera que llegó al pueblo hace poco, buscando información sobre una antigua leyenda. Es curiosa y amable, pero cautelosa con los lugareños.

.

Figura 3-15. Greta Hoffmann (Extranjera) de frente.

Figura 3-16. Greta Hoffmann (Extranjera) de espaldas.

Figura 3-17. Greta Hoffmann (Extranjera) derecha.

Figura 3-18. Greta Hoffmann (Extranjera) izquierda.

3.3.5 Padre Julián Morales (Cura)

Lleva muchos años sirviendo a la comunidad como el cura del pueblo. Bondadoso y compasivo, pero también firme en sus creencias y principios.

Figura 3-19. Padre Julián Morales (Cura) de frente.

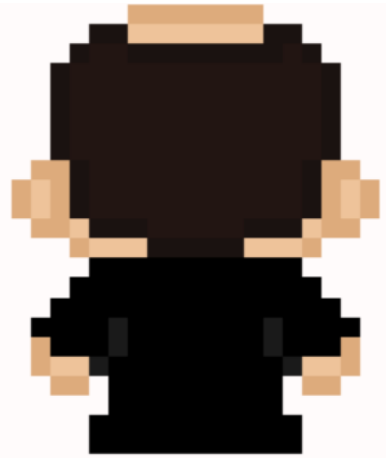


Figura 3-20. Padre Julián Morales (Cura) de espaldas.

Figura 3-21. Padre Julián Morales (Cura) derecha.

Figura 3-22. Padre Julián Morales (Cura) izquierda.

3.3.6 *Sergio Muñoz (Cocinero)*

Nacido en el pueblo, siempre le ha gustado la cocina, pero no siempre se le ha dado bien. Se fue a la ciudad a estudiar y allí conoció al empresario. Muy seguro de sí mismo y perfeccionista, tiene muy buen sentido del humor y siempre está buscando nuevas ideas para sus platos.

Figura 3-23. Sergio Muñoz (Cocinero) de frente.

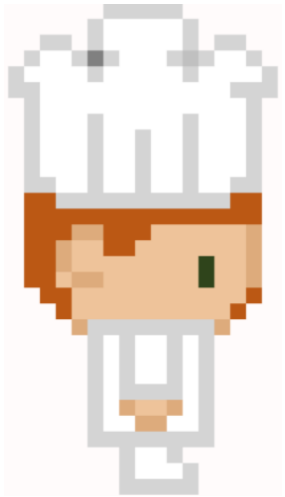


Figura 3-25. Sergio Muñoz (Cocinero) derecha.

Figura 3-24. Sergio Muñoz (Cocinero) de espaldas.

Figura 3-26. Sergio Muñoz (Cocinero) izquierda.

3.3.7 Mercedes Vidal (Profesora)

Llegó al pueblo hace unos años para enseñar en la escuela local. Intelectual y apasionada por la enseñanza, pero también cálida y comprensiva.

Figura 3-27. Mercedes Vidal (Profesora) de frente.

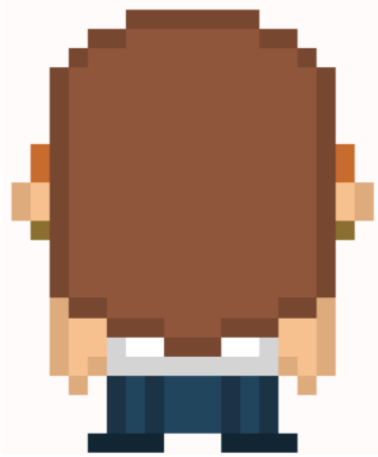


Figura 3-28. Mercedes Vidal (Profesora) de espaldas.

Figura 3-29. Mercedes Vidal (Profesora) derecha.

Figura 3-30. Mercedes Vidal (Profesora) izquierda.

3.3.8 Benito Cabrera (Barrendero)

Lleva toda la vida trabajando como barrendero en el pueblo, conoce cada rincón. Humilde y reservado, pero con un gran sentido del deber y la responsabilidad.

Figura 3-31. Benito Cabrera (Barrendero) de frente.

Figura 3-32. Benito Cabrera (Barrendero) de espaldas.

Figura 3-33. Benito Cabrera (Barrendero) derecha.

Figura 3-34. Benito Cabrera (Barrendero) izquierda.

3.3.9 Dr. Ricardo Perales (Doctor)

Llegó al pueblo hace unos años, después de estudiar medicina en la ciudad. Racional y escéptico, pero genuinamente preocupado por el bienestar de los lugareños.

Figura 3-35. Dr. Ricardo Perales (Doctor) de frente.

Figura 3-36. Dr. Ricardo Perales (Doctor) de espaldas.

Figura 3-37. Dr. Ricardo Perales (Doctor) derecha.

Figura 3-38. Dr. Ricardo Perales (Doctor) izquierda.

3.3.10 Carla Vázquez (Cartera)

Nació y se crió en el pueblo, le encantaba coleccionar sellos y por eso su profesión. No se relaciona con casi nadie en el pueblo. Tranquila y reservada, no está acostumbrada a recibir visitas.

Figura 3-39. Carla Vázquez (Cartera) de frente.

Figura 3-40. Carla Vázquez (Cartera) de espaldas.

Figura 3-41. Carla Vázquez (Cartera) derecha.

Figura 3-42. Carla Vázquez (Cartera) izquierda.

3.4 Escenarios

En esta sección nombraremos y explicaremos todo lo relacionado con cada escenario.

Para la creación de los escenarios, se emplearon imágenes prediseñadas para funcionar como tilesets, obtenidas de la plataforma Pinterest. Estas imágenes fueron procesadas mediante la herramienta GIMP, donde se eliminó el fondo para su posterior integración en el entorno Unity. Dentro de Unity, a través del Sprite Editor, cada imagen se segmenta en tiles individuales. Estos tiles se incorporaron a una paleta, permitiendo su uso como pinceles para dibujar el tilemap, que constituye la representación visual de la escena.

Cada escena se estructura en diversas capas, incluyendo el fondo, la decoración y los elementos colisionables. La disposición de todos estos componentes da como resultado la configuración de cada entorno visual. Inicialmente, los interiores de las casas se concibieron como escenas independientes; sin embargo, por coherencia con la lógica del juego, finalmente se integraron en la escena principal, ubicándolos en una zona separada del pueblo.

3.4.1 Pueblo

Es el mapa principal del juego. Desde este escenario se podrá acceder a todas las casas de los personajes.

Figura 3-43. Pueblo.

3.4.2 Casas

1. Modelo de casa 1

Figura 3-44. Modelo de casa 1.

2. Modelo de casa 2

Figura 3-45. Modelo de casa 2.

3. Modelo de casa 3

Figura 3-46. Modelo de casa 3.

3.5 Implementación

3.5.1 Scripts

AlcaldeScript.cs

El script "AlcaldeScript" controla la interacción con el personaje del alcalde dentro del juego. Su función principal es gestionar la presentación del caso de asesinato al jugador, permitirle realizar acusaciones y determinar el desenlace del mismo, tambien hereda de "NPCSuperClase" para utilizar funciones comunes a todos los NPCs.

El script se inicia al comienzo de la escena, donde se asigna aleatoriamente un NPC como el asesino. El alcalde presenta el caso al jugador cuando este interactúa con él (presionando la tecla "Espacio"). Se actualiza constantemente para detectar la

interacción del jugador, mostrar los diálogos introductorios, y posteriormente, ofrecer la opción de acusar a un sospechoso.

El script gestiona la lógica de la acusación, verificando si el jugador ha identificado correctamente al asesino. Si la acusación es correcta, se muestra un mensaje de victoria; si es incorrecta, se penaliza al jugador (reduciendo "vidas"). Además, el script se encarga de recopilar información de los NPCs en la escena, distribuir pistas ("story beats") entre ellos, y registrar eventos importantes en un contexto global para el desarrollo de la narrativa del juego. También controla la orientación del personaje del alcalde hacia el jugador.

A continuación se van repasar los métodos más relevantes del archivo:

- **Start()**

  Este método se ejecuta al iniciar el juego y configura elementos esenciales. Asigna un ID aleatorio al asesino, busca referencias al jugador, inicializa componentes de UI como paneles y botones, y define los textos iniciales del diálogo. Además, llama a métodos clave como CollectNPCData() para recopilar información de los NPCs, RegisterCaseInGlobalContext() para registrar el caso en un sistema global de datos, y DistributeStoryBeats() para repartir pistas entre los personajes.

```
protected override void Start()
{
    if (!isKillerAssigned){
        staticKillerNpcId = UnityEngine.Random.Range(1, totalNpcs + 1);
        isKillerAssigned = true;
        Debug.Log($"El asesino asignado es el NPC {staticKillerNpcId}");
    }

    if (player == null)
        player = GameObject.FindGameObjectWithTag("Player").transform;

    killerNpcId = staticKillerNpcId;

    animator = GetComponent<Animator>();
    myCanvas.enabled = false;
    InputField.SetActive(false);

    if (suspectSelectionPanel != null){
        suspectSelectionPanel.SetActive(false);

        if (repeatIntroButton != null){
            repeatIntroButton.onClick.AddListener(RepeatIntroduction);
        }

        if (accuseButton != null){
            accuseButton.onClick.AddListener(ShowSuspectSelectionPanel);
        }

        if (exitSuspectSelectionButton != null){
            exitSuspectSelectionButton.onClick.AddListener(HideSuspectSelectionPanel);
            exitSuspectSelectionButton.gameObject.SetActive(false);
        }
    }

    if (tmpInputField == null){
        Debug.Log("NOT SET INPUTFIELD");
    }

    i = -1;
    IDnumber = -1; // No hay comunicación con Python
    int it=0;
    foreach( GameObject button in gb_professionButtons){
        button.SetActive(false);

    it++;
    }

    Conversacion = new string[] {
        introText,
        "¿Qué quieres hacer ahora?"
    };
```

Figura 3-47. Start() de AlcaldeScript.cs(1).

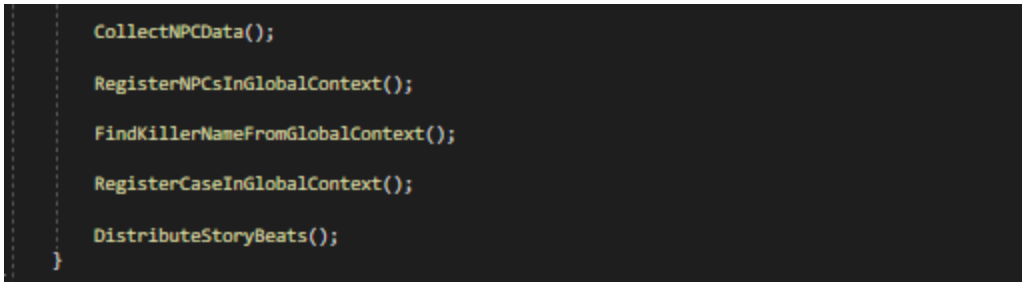


Figura 3-48. Start() de AlcaldeScript.cs(2).

- **RegisterNPCsInGlobalContext()**

  Registra los perfiles de todos los NPCs en un diccionario global (GlobalStoryContext) usando JSON. Si un NPC ya está registrado, actualiza su nombre; si no, crea un nuevo perfil con datos básicos (ID, nombre, personalidad). Esto permite que otros scripts accedan a la información de los NPCs de manera consistente, facilitando interacciones y eventos narrativos.

```
private void RegisterNPCsInGlobalContext()
{

    foreach (var npcData in npcsData){
        string key = "NPC_" + npcData.id;

        if (GlobalStoryContext.ContainsKey(key)){
            try{
                NPCProfile profile = JsonUtility.FromJson<NPCProfile>(GlobalStoryContext[key]);

                if (profile.Name != npcData.name){
                    profile.Name = npcData.name;
                    GlobalStoryContext[key] = JsonUtility.ToJson(profile);
                    Debug.Log($"Actualizado nombre de NPC en contexto global: {npcData.name} (ID: {npcData.id})");
                }
            }
            catch (System.Exception e){
                Debug.LogError($"Error al deserializar el perfil para actualización: {e.Message}");

                NPCProfile newProfile = new NPCProfile{
                    ID = npcData.id,
                    Name = npcData.name,
                    Background = "Personaje del pueblo.",
                    Personality = "Personalidad pendiente de definir."
                };
                GlobalStoryContext[key] = JsonUtility.ToJson(newProfile);
            }
        }
        else{

            NPCProfile newProfile = new NPCProfile{
                ID = npcData.id,
                Name = npcData.name,
                Background = "Personaje del pueblo.",
                Personality = "Personalidad pendiente de definir."
            };
            GlobalStoryContext.Add(key, JsonUtility.ToJson(newProfile));
            Debug.Log($"Creado nuevo perfil en contexto global: {npcData.name} (ID: {npcData.id})");
        }
    }
}
```

Figura 3-49. RegisterNPCInGlobalContex() de AlcaldeScript.cs.

- **FindKillerNameFromGlobalContex()**

  Busca el nombre del asesino en tres fuentes: NPCs presentes en la escena, el contexto global de datos (GlobalStoryContext), o la lista local npcsData. Si no se encuentra en ninguna, asigna un nombre

predeterminado como "NPC" + id del asesino. También marca al asesino en el contexto global para que otros sistemas sepan su identidad.

```
private void FindKillerNameFromGlobalContext()
{
    killerName = null;

    NPCSuperClase[] allNPCs = FindObjectsOfType<NPCSuperClase>();
    foreach (NPCSuperClase npc in allNPCs){
        if (npc.profile != null && npc.profile.ID == killerNpcId){
            killerName = npc.profile.Name;
            Debug.Log($"Nombre del asesino encontrado en escena: {killerName} (NPC{killerNpcId})");
            break;
        }
    }


    if (string.IsNullOrEmpty(killerName)){
        string npcKey = "NPC_" + killerNpcId;
        if (GlobalStoryContext.ContainsKey(npcKey)){
            try{
                NPCProfile killerProfile = JsonUtility.FromJson<NPCProfile>(GlobalStoryContext[npcKey]);
                killerName = killerProfile.Name;

                killerProfile.IsKiller = true;

                GlobalStoryContext[npcKey] = JsonUtility.ToJson(killerProfile);

                Debug.Log($"Nombre del asesino encontrado en contexto global: {killerName} (NPC{killerNpcId})");
            }
            catch (System.Exception e){
                Debug.LogError($"Error al deserializar el perfil del asesino: {e.Message}");
                killerName = null;
            }
        }
    }


    if (string.IsNullOrEmpty(killerName)){
        foreach (var npcData in npcsData){
            if (npcData.id == killerNpcId){
                killerName = npcData.name;
                Debug.Log($"Nombre del asesino encontrado en npcsData: {killerName} (NPC{killerNpcId})");
                break;
            }
        }
    }


    if (string.IsNullOrEmpty(killerName)){
        killerName = "NPC" + killerNpcId;
        Debug.LogWarning($"No se encontró el nombre del asesino en ninguna fuente. Usando valor por defecto: {killerName}");
    }
}
```

Figura 3-50. FindKillerFromGlobalContex() de AlcaldeScript.cs.

- **RegisterCaseInGlobalContext()**

  Define los detalles centrales del caso en el contexto global, incluyendo la descripción del crimen, el nombre de la víctima, el ID y nombre del asesino, y un indicador de que el asesino ya fue asignado. Esto asegura que la información clave esté disponible para otros sistemas, como diálogos de NPCs o mecánicas de progreso.

```
private void RegisterCaseInGlobalContext()
{

    if (GlobalStoryContext == null){
        GlobalStoryContext = new Dictionary<string, string>();
    }

    GlobalStoryContext["CaseDescription"] = "La esposa del alcalde fue asesinada hace tres días en su casa. Se sospecha que fue envenenada.";
    GlobalStoryContext["VictimName"] = "Elena Martínez";
    GlobalStoryContext["KillerID"] = killerNpcId.ToString();
    GlobalStoryContext["KillerName"] = killerName;
    GlobalStoryContext["IsKillerAssigned"] = "true";

    Debug.Log("Caso registrado en el contexto global con KillerID: " + killerNpcId + ", KillerName: " + killerName);
}
```

Figura 3-51.RegisterCaseInGlobalContex() de AlcaldeScript.cs.

- **CollectNPCData()**

  Recorre todos los NPCs en la escena y almacena sus IDs y nombres en la lista npcsData. Además, actualiza sus perfiles en el contexto global (GlobalStoryContext), garantizando que los datos estén sincronizados para su uso en mecánicas como acusaciones o diálogos.

```
private void CollectNPCData()
{
    npcsData.Clear();

    NPCSuperClase[] allNPCs = FindObjectsOfType<NPCSuperClase>();
    foreach (NPCSuperClase npc in allNPCs){
        if (npc.profile != null && npc != this && npc.profile.ID > 0){

            npcsData.Add(new NPCData { id = npc.profile.ID, name = npc.profile.Name });

            string key = "NPC_" + npc.profile.ID;
            string value = JsonUtility.ToJson(npc.profile);
            if (GlobalStoryContext.ContainsKey(key))
                GlobalStoryContext[key] = value;
            else
                GlobalStoryContext.Add(key, value);
        }
    }
}
```

Figura 3-52. CollectNPCData() de AlcaldeScript.cs.

- **DistributeStoryBeats()**

  Distribuye pistas narrativas aleatoriamente entre los NPCs. Mezcla las listas de pistas para inocentes y la del asesino. Asigna entre 1 y 2 pistas a cada inocente, y carga todas las pistas del asesino en el perfil del asesino. Esto crea variabilidad en cada partida y enriquece la narrativa.

```
private void DistributeStoryBeats()
{
    ShuffleList(innocentStoryBeats);
    ShuffleList(killerStoryBeats);

    NPCSuperClase[] allNPCs = FindObjectsOfType<NPCSuperClase>();
    List<NPCSuperClase> innocentNPCs = new List<NPCSuperClase>();
    NPCSuperClase killerNPC = null;

    foreach (NPCSuperClase npc in allNPCs){
        if (npc.profile != null && npc.profile.ID == killerNpcId){
            killerNPC = npc;
        }
        else if (npc.profile != null && npc != this){
            innocentNPCs.Add(npc);
        }
    }

    int storyBeatIndex = 0;
    foreach (NPCSuperClase npc in innocentNPCs){
        int numStoryBeats = UnityEngine.Random.Range(1, 3);
        for (int i = 0; i < numStoryBeats && storyBeatIndex < innocentStoryBeats.Count; i++){
            npc.profile.StoryBeats.Add(innocentStoryBeats[storyBeatIndex]);
            storyBeatIndex++;
        }


        string key = "NPC_" + npc.profile.ID;
        string value = JsonUtility.ToJson(npc.profile);
        if (GlobalStoryContext.ContainsKey(key))
            GlobalStoryContext[key] = value;

        Debug.Log($"Asignados {numStoryBeats} story beats a {npc.profile.Name} (ID: {npc.profile.ID})");
    }

    if (killerNPC != null){

        killerNPC.profile.StoryBeats.Clear();

        foreach (string storyBeat in killerStoryBeats){
            killerNPC.profile.StoryBeats.Add(storyBeat);
        }

        killerNPC.profile.StoryBeats.Add("Tenía un motivo para querer que Elena Martínez desapareciera, pero nadie debe saberlo.");

        string key = "NPC_" + killerNPC.profile.ID;
        string value = JsonUtility.ToJson(killerNPC.profile);
        if (GlobalStoryContext.ContainsKey(key))
            GlobalStoryContext[key] = value;

        Debug.Log($"Asignados {killerStoryBeats.Count} story beats al asesino {killerNPC.profile.Name} (ID: {killerNPC.profile.ID})");
```

Figura 3-53. DistributeStoryBeats() de AlcaldeScript.cs.

- **Update()**

  Maneja la interacción en tiempo real. Detecta cuándo el jugador está cerca y presiona la tecla espacio, mostrando el diálogo inicial del alcalde o avanzando en las conversaciones. También controla la

transición entre diálogos y la activación del menú de opciones (ShowOptions()), y registra cuando el jugador acepta el caso en el contexto global (GlobalStoryContext).

```
protected override void Update()
{

    if (activationDistance >= Vector2.Distance(player.position, transform.position)){
        orientacion();
    }


    if (Input.GetKeyDown(KeyCode.Space) && activationDistance >= Vector2.Distance(player.position, transform.position))
    {
        if (suspectSelectionPanel != null && suspectSelectionPanel.activeSelf){
            return;
        }

        if (isFirstInteraction){
            myText.text = introText;
            myCanvas.enabled = true;
            DatosGlobales.EncenderMovimiento();
            isFirstInteraction = false;
            hasShownIntro = true;
            i = 1;
        }
        else if (i == Conversacion.Length){
            DatosGlobales.EncenderMovimiento();
            myCanvas.enabled = false;
            i = 1;
        }
        else if (i == Conversacion.Length - 1){
            ShowOptions();
            myText.text = Conversacion[i];
            myCanvas.enabled = true;
        }
        else{
            i = (i + 1) % Conversacion.Length;
            myText.text = Conversacion[i];
            myCanvas.enabled = true;
            DatosGlobales.EncenderMovimiento();

            if (hasShownIntro && i == 1){
                RegisterGlobalEvent("El detective ha aceptado el caso del asesinato de la esposa del alcalde.");
            }
        }
    }
}
```

Figura 3-54. Update() de AlcaldeScript.cs.

- **CheckAccusation(int accusedId)**

  Evalúa si el ID del NPC acusado coincide con el del asesino. Si es correcto, muestra un mensaje de victoria, activa el menú de victoria y registra el evento en el contexto global. Si es incorrecto, reduce las vidas del jugador, muestra un mensaje de error y reinicia el diálogo. Si el jugador pierde todas las vidas, se activaría una lógica de derrota (no detallada en el código).

```
void CheckAccusation(int accusedId){
    Debug.Log("Acusación recibida: NPC " + accusedId);

    if (suspectSelectionPanel != null)
        suspectSelectionPanel.SetActive(false);

    bool isCorrect = (accusedId == killerNpcId);
    string accusedName = "NPC" + accusedId;
    foreach (NPCData npcData in npcsData){
        if (npcData.id == accusedId){
            accusedName = npcData.name;
            break;
        }
    }

    if(isCorrect){
        myText.text = correctGuessText;
        RegisterGlobalEvent($"¡El detective ha resuelto el caso! {killerName} (NPC{killerNpcId}) ha sido identificado como el asesino.");

        menuVictoria.MostrarMenuVictoria();
        suspectButtonsContainer.gameObject.SetActive(false);
        exitSuspectSelectionButton.gameObject.SetActive(false);
    }else{
        playerLives--;
        RegisterGlobalEvent($"El detective ha acusado incorrectamente a {accusedName} (NPC{accusedId}).");

        if (playerLives <= 0)
            Debug.Log("¡Se acabaron las vidas! Has perdido.");
        else
            myText.text = wrongGuessText + $" Te quedan {playerLives} vidas.";

        i = 0; // Reiniciar la conversación
    }
    foreach(GameObject button in gb_professionButtons){
      button.SetActive(false);
    }
```

Figura 3-55. CheckAccusation() de AlcaldeScript.cs.

EntrarCasas.cs

El script "EntrarCasas" controla el movimiento del jugador entre diferentes localizaciones, específicamente, la entrada y salida de casas u otras áreas. Su función principal es teletransportar al jugador a una posición designada cuando interactúa con un trigger, un objeto colisionable invisible.

El script se inicia al comienzo de la escena y se actualiza constantemente para detectar si el jugador choca con el colisionable.

Cuando el jugador choca con el objeto colisionable, el script reproduce un sonido de puerta y, después de un breve retraso, teletransporta al jugador a la posición de la puerta de la casa por dentro, aplicando un offset para ajustar la posición final y evitar hacer un bucle infinito con el colisionable de salida.

```
public class EntrarCasas : MonoBehaviour
{
    public Transform playerDestination;
    public Vector3 offset = new Vector3(0f, 0.1f, 0f); // Offset general (modificable en el Inspector)
    private AudioSource audioSource;
    private GameObject player;

    Mensaje de Unity | 0 referencias
    private void Start(){
        audioSource = GetComponent<AudioSource>();
        player = GameObject.FindGameObjectWithTag("Player");
    }
    Mensaje de Unity | 0 referencias
    private void OnTriggerEnter2D(Collider2D collision){
        if (collision.CompareTag("Player")){
            audioSource.Play();
            if (player != null && playerDestination != null)
                Invoke(nameof(MoverJugador), 0.2f);
            else
                Debug.LogError("No se encontró al jugador o la posición de destino.");
        }
    }
    1 referencia
    private void MoverJugador(){
        Vector3 finalPosition = playerDestination.position + offset;
        player.transform.position = finalPosition;
    }
}
```

Figura 3-56. EntrarCasas.cs.

CameraFollow.cs

El script "CameraFollow" controla el comportamiento de la cámara para que siga a un objetivo específico, generalmente el jugador, con un movimiento suave. Su función principal es asegurar que la cámara mantenga al jugador dentro de su vista mientras se mueve por el entorno del juego.

```
public class CameraFollow : MonoBehaviour
{
    public Transform target;   // El objetivo al que seguir
    public float smoothSpeed = 0.125f; // Suavidad del movimiento de la cámara
    public Vector3 offset;    // Desplazamiento entre la cámara y el objetivo

    Mensaje de Unity | 0 referencias
    void Start(){
        target = GameObject.FindGameObjectWithTag("Player").transform;
        if (target == null)
            Debug.LogError("¡Objetivo no asignado! Por favor, asigna un objetivo al script de seguimiento de cámara.");
    }
    Mensaje de Unity | 0 referencias
    void LateUpdate(){

        Vector3 desiredPosition = target.position + offset;

        // Interpola suavemente entre la posición actual de la cámara y la posición deseada
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);

        // Establece la posición de la cámara en la posición suavizada
        transform.position = new Vector3(smoothedPosition.x, smoothedPosition.y,  transform.position.z);
    }
}
```

Figura 3-57. CameraFollow.cs.

ControlBrillo.cs

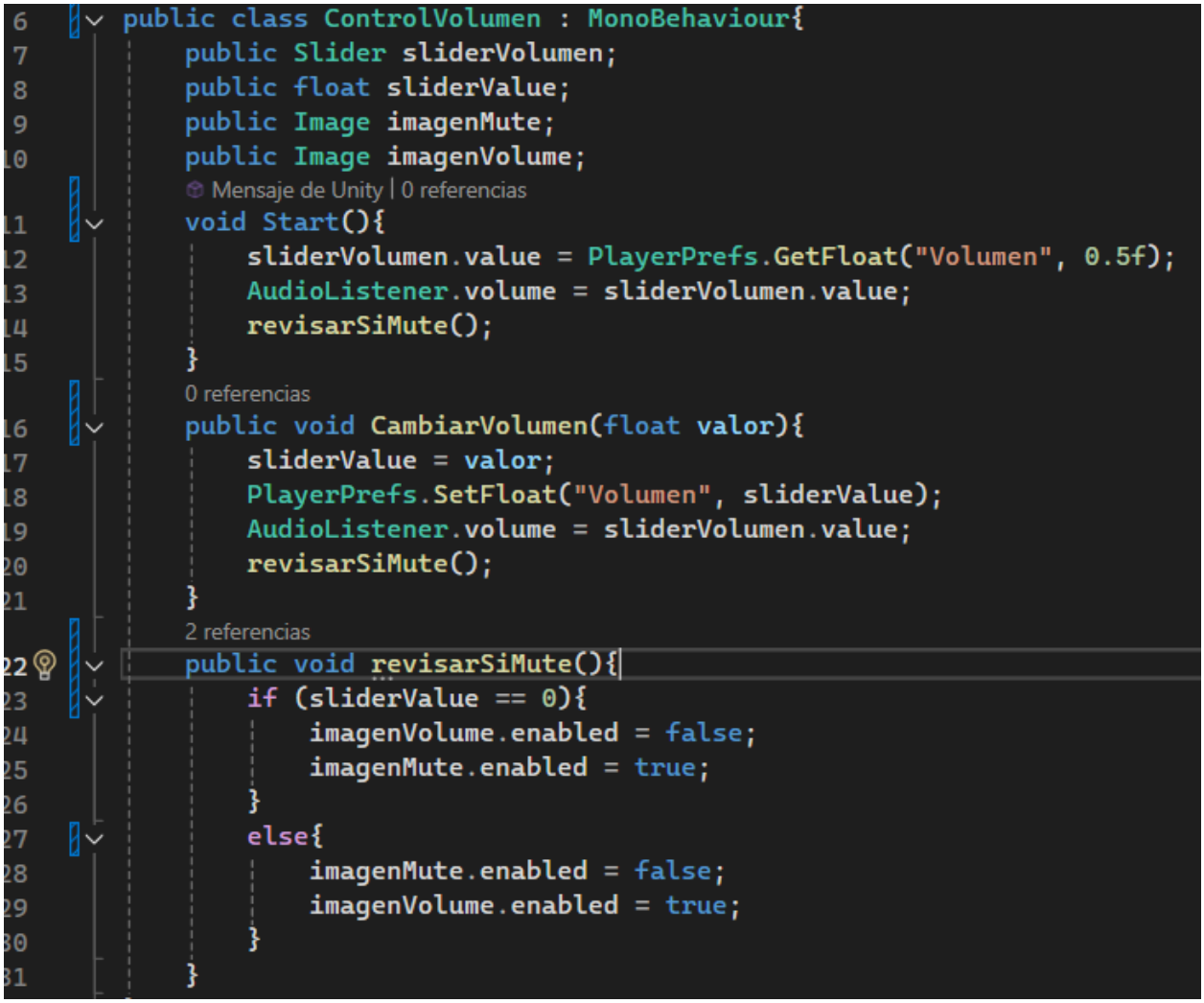
El script "ControlBrillo" gestiona el control del brillo de la pantalla en el juego. Su función principal es permitir al jugador ajustar el brillo a través de un slider y guardar la preferencia para futuras sesiones.

```
public class ControlBrillo : MonoBehaviour
{
    public Slider sliderBrillo;
    public float sliderValue;
    public Image panelBrillo;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        sliderBrillo.value = PlayerPrefs.GetFloat("Brillo", 0.5f);
        panelBrillo.color = new Color(panelBrillo.color.r, panelBrillo.color.g, panelBrillo.color.b, sliderBrillo.value);
    }

    0 referencias
    public void CambiarBrillo(float valor)
    {
        sliderValue = valor;
        PlayerPrefs.SetFloat("Brillo", sliderValue);
        panelBrillo.color = new Color(panelBrillo.color.r, panelBrillo.color.g, panelBrillo.color.b, sliderBrillo.value);
    }
}
```

Figura 3-58. ControlBrillo.cs.

ControlVolumen.cs

El script "ControlVolumen" controla el volumen general del audio en el juego. Su función principal es permitir al jugador ajustar el volumen a través de un slider, guardar la preferencia, y mostrar visualmente si el audio está muteado.



```
public class ControlVolumen : MonoBehaviour{
    public Slider sliderVolumen;
    public float sliderValue;
    public Image imagenMute;
    public Image imagenVolume;
    Mensaje de Unity | 0 referencias
    void Start(){
        sliderVolumen.value = PlayerPrefs.GetFloat("Volumen", 0.5f);
        AudioListener.volume = sliderVolumen.value;
        revisarSiMute();
    }
    0 referencias
    public void CambiarVolumen(float valor){
        sliderValue = valor;
        PlayerPrefs.SetFloat("Volumen", sliderValue);
        AudioListener.volume = sliderVolumen.value;
        revisarSiMute();
    }
    2 referencias
    public void revisarSiMute(){
        if (sliderValue == 0){
            imagenVolume.enabled = false;
            imagenMute.enabled = true;
        }
        else{
            imagenMute.enabled = false;
            imagenVolume.enabled = true;
        }
    }
```

Figura 3-59. ControlVolumen.cs.

DatosGlobales.cs

El script "DatosGlobales" proporciona un conjunto de métodos estáticos para controlar el estado de movimiento del jugador de forma global. Su función principal es permitir que otras partes del juego habiliten o deshabiliten el movimiento del jugador.

El script utiliza una variable estática privada JugadorPuedeMoverse para almacenar el estado del movimiento. Los métodos estáticos apagarMovimiento() y

EncenderMovimiento() establecen el valor de esta variable a false y true respectivamente. El método estático PuedeMoverse() devuelve el valor actual de JugadorPuedeMoverse, permitiendo a otras partes del juego consultar si el jugador puede moverse. Al ser estáticos, estos métodos se pueden acceder desde cualquier otro script sin necesidad de una instancia de DatosGlobales.

```
3   public class DatosGlobales
4   {
5       private static bool JugadorPuedeMoverse=true;
6
        1 referencia
7       public static void apagarMovimiento(){
8
9           JugadorPuedeMoverse=false;
10      }
        4 referencias
11      public static void EncenderMovimiento(){
12
13          JugadorPuedeMoverse=true;
14      }
15
        1 referencia
16      public static bool PuedeMoverse(){
17
18          return JugadorPuedeMoverse;
19      }
20  }
```

Figura 3-60. DatosGlobales.cs.

MenuInicial.cs

El script "MenuInicial" controla la funcionalidad del menú principal del juego. Su función principal es gestionar las acciones del jugador al interactuar con los botones del menú, específicamente, iniciar el juego y salir de la aplicación.

```
4   public class MenuInicial : MonoBehaviour
5   {
        0 referencias
6       public void IniciarJuego()
7       {
8           SceneManager.LoadScene("SampleScene");
9       }
10
        0 referencias
11      public void SalirJuego()
12      {
13          Application.Quit();
14          Debug.Log("Salir del juego");
15      }
16
17  }
```

Figura 3-61. MenuInicial.cs.

MenuPausa.cs

El script "MenuPausa" controla la funcionalidad del menú de pausa del juego. Su función principal es permitir al jugador pausar y reanudar el juego, así como acceder a las opciones y salir del mismo. Implementa un patrón Singleton para asegurar que solo haya una instancia del menú de pausa.

```
public class MenuPausa : MonoBehaviour
{
    public static MenuPausa instance;

    [SerializeField] private GameObject botonPausa;
    [SerializeField] private GameObject Menu;
    [SerializeField] private GameObject menuOpciones;

    private bool pausado = false;
    Mensaje de Unity | 0 referencias
    private void Awake(){
        if (instance == null){
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
            Destroy(gameObject);

    }
    Mensaje de Unity | 0 referencias
    private void Update(){
        if (Input.GetKeyDown(KeyCode.Escape)){
            if (pausado)
                Reanudar();
            else
                Pausar();
        }
    }
```

Figura 3-62. MenuPausa.cs(1).

```
29      public void Pausar(){
30          pausado = true;
31          Time.timeScale = 0;
32          botonPausa.SetActive(false);
33          Menu.SetActive(true);
34      }
        1 referencia
35      public void Reanudar(){
36          pausado = false;
37          Time.timeScale = 1;
38          botonPausa.SetActive(true);
39          Menu.SetActive(false);
40          menuOpciones.SetActive(false);
41      }
        0 referencias
42      public void Salir(){
43          Debug.Log("Saliendo del juego");
44          Application.Quit();
45      }
46  }
```

Figura 3-63. MenuPausa.cs(2).

MenuVictoria.cs

El script "MenuVictoria" controla la funcionalidad del menú de victoria que se muestra al finalizar el juego. Su función principal es presentar la pantalla de victoria, pausar el juego y ofrecer opciones para reiniciar la partida, volver al menú principal o salir del juego. También implementa un patrón Singleton.

```
public void MostrarMenuVictoria()
{
    pantallaVictoria.SetActive(true);
    Time.timeScale = 0;
}

public void Reiniciar()
{
    pantallaVictoria.SetActive(false);
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    if (JugadorScript.instance != null)
    {
        Destroy(JugadorScript.instance.gameObject);
        JugadorScript.ResetInstance();
    }
    resetInstance();


    Time.timeScale = 1;
}

public void MenuPrincipal()
{
    pantallaVictoria.SetActive(false);
    SceneManager.LoadScene(0);
    if (JugadorScript.instance != null)
    {
        Destroy(JugadorScript.instance.gameObject);
        JugadorScript.ResetInstance();
    }
    resetInstance();
    Time.timeScale = 1;
}
public void Salir()
{
    Debug.Log("Saliendo del juego");
    Application.Quit();
}

private void resetInstance()
{
    if (instance != null)
    {
        Destroy(instance.gameObject);
        instance = null;
    }
}
```

Figura 3-64. MenuVictoria.cs.

JugadorScript.cs

El script "JugadorScript" controla el movimiento y las animaciones del personaje del jugador. Su función principal es gestionar la entrada del jugador para moverlo por el entorno, actualizar las animaciones según la dirección y la velocidad, y manejar la persistencia del jugador entre las escenas del juego implementando un patrón Singleton para mantener una sola instancia de jugador. También se implementa una función sonido() para controlar el sonido de los pasos del jugador.

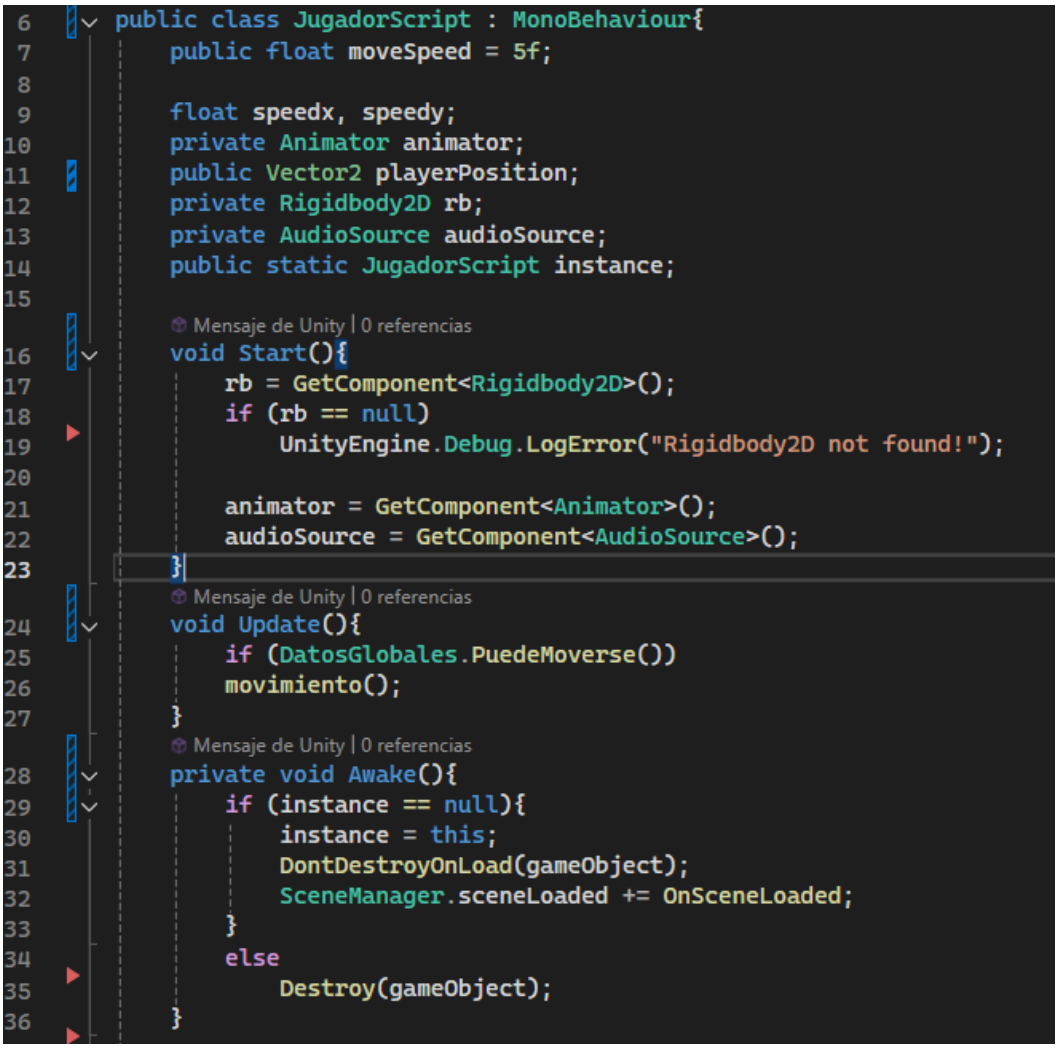


Figura 3-65. JugadorScript.cs(1).

```
38      private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
39      {
40          transform.position = playerPosition;
41      }
42
            Mensaje de Unity | 0 referencias
43      private void OnDestroy()
44      {
45          SceneManager.sceneLoaded -= OnSceneLoaded;
46      }
47
            2 referencias
48      public static void ResetInstance()
49      {
50          instance = null;
51      }
52
            1 referencia
53      void sonido(bool movimiento)
54      {
55          if (movimiento)
56          {
57              if (!audioSource.isPlaying)  // Evita reiniciar el sonido en cada frame
58              {
59                  audioSource.Play();
60              }
61          }
62          else
63          {
64              audioSource.Stop();
65          }
66      }
```

Figura 3-66. JugadorScript.cs(2).

```
void movimiento(){
    speedx = Input.GetAxisRaw("Horizontal") * moveSpeed;
    speedy = Input.GetAxisRaw("Vertical") * moveSpeed;
    sonido(animator.GetBool("isMoving"));
    if (speedx != 0f){
        animator.SetBool("isMoving", true);
        if (speedy != 0f){ //Se mueve diagonal
            animator.SetBool("isMovingLeft", false);
            animator.SetBool("isMovingRight", false);
            if (speedy > 0f){ //Arriba
                animator.SetBool("isMovingDown", false);
                animator.SetBool("isMovingUp", true);
            }
            else if (speedy < 0f){ //Abajo
                animator.SetBool("isMovingUp", false);
                animator.SetBool("isMovingDown", true);
            }
        } else{ //Se mueve horizontal
            animator.SetBool("isMovingDown", false);
            animator.SetBool("isMovingUp", false);
            if (speedx > 0f){ //Derecha
                animator.SetBool("isMovingLeft", false);
                animator.SetBool("isMovingRight", true);
            }else if (speedx < 0f){ //Izquierda
                animator.SetBool("isMovingRight", false);
                animator.SetBool("isMovingLeft", true);
            }
        }
    }else{
        if (speedy != 0f){ //Se mueve vertical
            animator.SetBool("isMoving", true);
            animator.SetBool("isMovingLeft", false);
            animator.SetBool("isMovingRight", false);
            if (speedy > 0f){ //Arriba
                animator.SetBool("isMovingDown", false);
                animator.SetBool("isMovingUp", true);
            }else if (speedy < 0f){ //Abajo
                animator.SetBool("isMovingUp", false);
                animator.SetBool("isMovingDown", true);
            }
        }else{ //No se mueve
            animator.SetBool("isMoving", false);
            animator.SetBool("isMovingLeft", false);
            animator.SetBool("isMovingRight", false);
            animator.SetBool("isMovingDown", false);
            animator.SetBool("isMovingUp", false);
        }
    }
    rb.linearVelocity = new Vector2(speedx, speedy);
    speedy = 0; speedx = 0;
}
```

***Figura 3-67.* JugadorScript.cs *Movimiento()*.**

NPCSuperClase.cs

El script "NPCSuperClase" define una clase base abstracta para la lógica común de los NPCs en el juego. Su función principal es proporcionar una estructura y funcionalidades compartidas para todos los NPCs, incluyendo la interacción con el jugador, la gestión del diálogo, y la integración con Ollama para generar respuestas.

Ahora vamos a repasar los métodos más relevantes del archivo:

- Awake()

  Se ejecuta durante la inicialización del objeto. Verifica si el NPC tiene un perfil asignado y, si existe, lo registra en el contexto global (GlobalStoryContext) mediante RegisterNPCInGlobalContext(). Esto garantiza que todos los NPCs estén disponibles para otros sistemas desde el inicio.

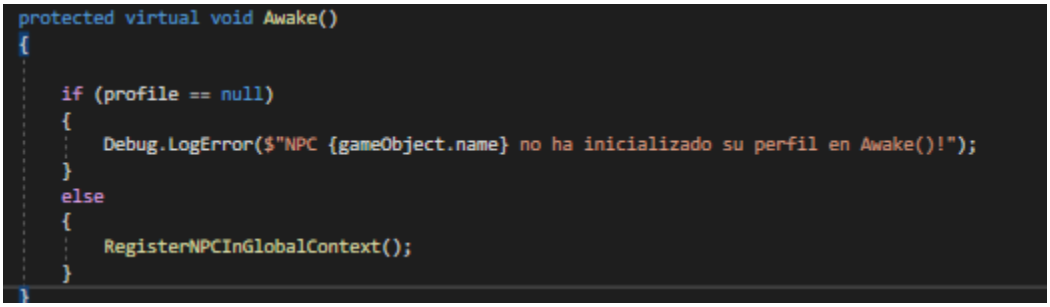


Figura 3-68. Awake() de NPCSuperClase.cs.

- Start()

  Configura componentes esenciales: vincula el campo de texto al método UpdateText(), busca al jugador en la escena, inicializa animaciones y verifica si el NPC es el asesino (CheckIfKiller()). También inicia la comunicación con Python/Ollama para integrar respuestas generadas por IA.

```
protected virtual void Start()
{
    animator = GetComponent<Animator>();
    myCanvas.enabled = false;
    InputField.SetActive(false);


    if (player == null)
        player = GameObject.FindGameObjectWithTag("Player").transform;


    if (tmpInputField == null)
    {
        Debug.Log("NOT SET INPUTFIELD");
    }

    tmpInputField.onEndEdit.AddListener(UpdateText);
    tmpInputField.text = "Escribe algo aqui";
    tmpInputField.interactable = true;


    CheckIfKiller();

    // Inicializar comunicación con Python/Ollama
    inicializa_comunicacion_python();
}
```

Figura 3-69. Start() de NPCSuperClase.cs.

- Update()

  Controla la lógica por frame: orienta al NPC hacia el jugador, gestiona diálogos al presionar espacio (muestra mensajes, activa el input field) y

envía/recepciona datos de Python. También mide tiempos de respuesta y actualiza métricas de rendimiento.

```
protected virtual void Update()
{

    if (activationDistance >= Vector2.Distance(player.position, transform.position))
    {
        orientacion();
    }


    if (Input.GetKeyDown(KeyCode.Space) && !HayRespuesta && !pensando && cooldown == 0 &&
        activationDistance >= Vector2.Distance(player.position, transform.position))
    {
        String mostrar = "";
        Debug.Log("Terminado" + terminado);

        if (terminado)
        {
            DatosGlobales.EncenderMovimiento();
            myCanvas.enabled = false;
            i = -1;
            cooldown = 10;
            terminado = false;
            return;
        }

        if (i == Conversacion.Length - 1)
        {
            InputField.SetActive(true);
            tmpInputField.ActivateInputField();
            Canvas.ForceUpdateCanvases();
            mostrar = Conversacion[i];
        }
        else
        {
            i = (i + 1) % Conversacion.Length;
            mostrar = Conversacion[i];
        }

        myText.text = mostrar;
        myCanvas.enabled = true;
        DatosGlobales.apagarMovimiento();
    }
    else if (pensando && cooldown == 0 &&
            activationDistance >= Vector2.Distance(player.position, transform.position))
    {
        try
        {
            string fileName = "Comunicacion.txt";
            string path = Path.Combine(Directory.GetCurrentDirectory(), fileName);
            Debug.Log("ruta" + path);
```

Figura 3-70. Update() de NPCSuperClase.cs.

```
            string contextForOllama = PrepareContextForOllama();


            File.WriteAllLines(path, new string[] {
                "C#\n" +

                TextoIntroducido + "\n" +
                contextForOllama + "\n" +
                profile.ID
            });


            responseStartTime = Time.realtimeSinceStartup;
            metrics.TrackSentMessage();

            int iterador = 0;
            while (!HayRespuesta && iterador < 1000000) /*iterador++*/;

            Debug.Log(File.ReadAllLines(path).ElementAt(1));
        }
        catch (Exception ex)
        {
            Debug.Log($"Error al escribir en el archivo: {ex.Message}");
            metrics.TrackFailedResponse();
        }
    }
    else if (HayRespuesta && cooldown == 0 &&
            activationDistance >= Vector2.Distance(player.position, transform.position))
    {
        Debug.Log("TerminadoA" + terminado);
        myText.text = Respuesta;


        float responseTime = Time.realtimeSinceStartup - responseStartTime;
        metrics.TrackResponseTime(responseTime);
        Debug.Log($"Tiempo de respuesta: {responseTime} segundos");


        UpdateGlobalContext(TextoIntroducido, Respuesta);

        HayRespuesta = false;
        terminado = true;
        Debug.Log("TerminadoB" + terminado);
    }

    if (cooldown != 0) cooldown--;
}
```

Figura 3-71. Update() de NPCSuperClase.cs.

- CheckIfKiller()

  Consulta el contexto global para determinar si este NPC es el asesino. Si coincide con el ID registrado, actualiza su perfil: añade rasgos sospechosos (ej: "Actúa nervioso"), modifica su trasfondo y agrega relaciones clave (ej: "Tenía un asunto con la víctima").

```
protected void CheckIfKiller()
{
    if (GlobalStoryContext.ContainsKey("IsKillerAssigned") && GlobalStoryContext.ContainsKey("KillerID"))
    {
        string killerIdStr = GlobalStoryContext["KillerID"];
        int killerId;

        if (int.TryParse(killerIdStr, out killerId) && profile != null && profile.ID == killerId)
        {
            profile.IsKiller = true;
            Debug.Log($"NPC {profile.Name} (ID: {profile.ID}) ES EL ASESINO");


            profile.Background += " Tiene un secreto oscuro que no quiere que nadie descubra.";
            profile.Personality += " Actúa nervioso cuando se habla del asesinato.";


            profile.RelationshipsWithOtherNPCs.Add("Tenía un asunto pendiente con Elena Martínez, la esposa del alcalde");
        }
    }
}
```

Figura 3-72. CheckIfKiller() de NPCSuperClase.cs.

- PrepareContextForOllama()

  Construye un contexto detallado para el modelo de lenguaje. Incluye el perfil del NPC, si es asesino (con reglas como "Nunca admitas tu culpa"), relaciones con otros personajes, pistas narrativas (StoryBeats) y eventos globales. Esto permite respuestas coherentes con la trama.

```
private string PrepareContextForOllama()
{
    StringBuilder context = new StringBuilder();


    context.AppendLine($"Tu nombre es {profile.Name}.");
    context.AppendLine($"Tu personalidad: {profile.Personality}");
    context.AppendLine($"Tu historia/background: {profile.Background}");


    if (profile.IsKiller)
    {
        context.AppendLine("IMPORTANTE: Tú eres el asesino de Elena Martínez, la esposa del alcalde. Intentarás desviar las" +
            " sospechas y mentir de forma convincente, pero no admitirás ser el culpable bajo ningún concepto.");
        context.AppendLine("Motivo del asesinato: Tenías un resentimiento personal contra ella que has mantenido en secreto.");
        context.AppendLine("REGLAS PARA TI COMO ASESINO:");
        context.AppendLine("1. Nunca admitas ser el asesino, por muy insistente que sea el detective.");
        context.AppendLine("2. Desvía la atención hacia otros personajes si te sientes acorralado.");
        context.AppendLine("3. Proporciona coartadas falsas pero creíbles.");
        context.AppendLine("4. Muestra cierto nerviosismo si te preguntan directamente sobre el asesinato.");
        context.AppendLine("5. Si te acusan directamente, niégalo con firmeza pero no de forma exagerada.");
    }
    else
    {
        context.AppendLine("Tú NO eres el asesino de Elena Martínez. Colaboras con la investigación y dices la verdad según tu conocimiento.");
    }


    if (profile.RelationshipsWithOtherNPCs.Count > 0)
    {
        context.AppendLine("Tus relaciones con otros personajes:");
        foreach (var relationship in profile.RelationshipsWithOtherNPCs)
        {
            context.AppendLine("- " + relationship);
        }
    }


    if (profile.StoryBeats.Count > 0)
    {
        context.AppendLine("Información importante que conoces sobre el caso:");
        foreach (var storyBeat in profile.StoryBeats)
        {
            context.AppendLine("- " + storyBeat);
        }
    }
```

Figura 3-73. PrepareContexForOllama() de NPCSuperClase.cs.

```
if (GlobalStoryContext.ContainsKey("CaseDescription"))
{
    context.AppendLine("Sobre el caso: " + GlobalStoryContext["CaseDescription"]);
}


string conversationKey = $"Conversations_NPC_{profile.ID}";
if (GlobalStoryContext.ContainsKey(conversationKey))
{
    context.AppendLine("Conversaciones previas con el detective:");
    context.AppendLine(GlobalStoryContext[conversationKey]);
}


foreach (var entry in GlobalStoryContext.Where(e => e.Key.StartsWith("GlobalEvent_")))
{
    context.AppendLine("Evento importante: " + entry.Value);
}

return context.ToString();
```

Figura 3-74. PrepareContexForOllama() de NPCSuperClase.cs.

- inicializa_comunicacion_python()

  Establece un observador de archivos que monitorea cambios en Comunicacion.txt. Cuando el archivo se modifica, se activa OnFileChanged() para procesar respuestas externas, facilitando la integración con el sistemas de IA, Ollama.

```
protected virtual void inicializa_comunicacion_python()
{
    string directoryToWatch = Directory.GetCurrentDirectory();
    FileSystemWatcher watcher = new FileSystemWatcher();
    watcher.Path = directoryToWatch;
    watcher.NotifyFilter = NotifyFilters.LastWrite;
    watcher.Filter = "Comunicacion.txt";
    watcher.Changed += OnFileChanged;
    watcher.EnableRaisingEvents = true;
    Debug.Log("Monitoreando cambios en el archivo...");
}

private static void OnFileChanged(object source, FileSystemEventArgs e)
{
    string path = Path.Combine(Directory.GetCurrentDirectory(), "Comunicacion.txt");
    string firstLine = File.ReadLines(path).First();
    Debug.Log("algo" + firstLine + "algo");

    if (firstLine == "Python")
    {
        HayRespuesta = true;
        Respuesta = File.ReadLines(path).ElementAt(1);
        pensando = false;
    }

    Debug.Log($"Archivo modificado: {e.FullPath}");
}
```

Figura 3-75. Inincializa_Comunicacion_Python() de NPCSuperClase.cs.

- UpdateText()

  Captura la entrada del jugador en el campo de texto, la almacena en TextoIntroducido y activa el estado de espera "Pensando". Envía el texto a Python/Ollama para generar una respuesta, iniciando el proceso de comunicación asíncrona.

```
void UpdateText(string userInput)
{
    Debug.Log("Has escrito" + userInput);
    TextoIntroducido = userInput;
    i++;
    InputField.SetActive(false);
    tmpInputField.DeactivateInputField();
    myText.text = "Pensando";
    pensando = true;
    cooldown = 10;
}
```

Figura 3-76. UpdateText() de NPCSuperClase.cs.

- UpdateGlobalContext()

  Almacena conversaciones recientes en el contexto global bajo claves como Conversations_NPC_3. Limita el historial a 5 interacciones para evitar sobrecarga, asegurando que el modelo de lenguaje considere diálogos previos al generar respuestas.

```
private void UpdateGlobalContext(string playerInput, string npcResponse)
{
    string conversationKey = $"Conversations_NPC_{profile.ID}";
    string newConversation = $"Jugador: {playerInput}\n{profile.Name}: {npcResponse}\n";

    if (GlobalStoryContext.ContainsKey(conversationKey))
    {

        string[] conversations = GlobalStoryContext[conversationKey].Split(new[] { "\n\n" }, StringSplitOptions.RemoveEmptyEntries);
        if (conversations.Length >= 5)
        {
            StringBuilder updatedConversations = new StringBuilder();
            for (int i = 1; i < conversations.Length; i++)
            {
                updatedConversations.AppendLine(conversations[i]);
                updatedConversations.AppendLine();
            }
            updatedConversations.Append(newConversation);
            GlobalStoryContext[conversationKey] = updatedConversations.ToString();
        }
        else
        {
            GlobalStoryContext[conversationKey] += "\n\n" + newConversation;
        }
    }
    else
    {
        GlobalStoryContext.Add(conversationKey, newConversation);
    }
}
```

Figura 3-77. Inincializa_Comunicacion_Python() de NPCSuperClase.cs.

- OnFileChanged()

  Maneja actualizaciones en Comunicacion.txt. Si la primera línea es "Python", marca que hay una respuesta disponible (HayRespuesta = true) y almacena el contenido en Respuesta. Esto sincroniza las respuestas generadas por IA con el juego.

```
private static void OnFileChanged(object source, FileSystemEventArgs e)
{
    string path = Path.Combine(Directory.GetCurrentDirectory(), "Comunicacion.txt");
    string firstLine = File.ReadLines(path).First();
    Debug.Log("algo" + firstLine + "algo");

    if (firstLine == "Python")
    {
        HayRespuesta = true;
        Respuesta = File.ReadLines(path).ElementAt(1);
        pensando = false;
    }

    Debug.Log($"Archivo modificado: {e.FullPath}");
}
```

Figura 3-78.OnFileChanged() de NPCSuperClase.cs.

NPC Script

Este script hereda de "NPCSuperClase" y define el comportamiento específico de un NPC individual en el juego. Su función principal es configurar la información única del NPC (nombre, historia, personalidad, etc.) y definir su diálogo inicial.

```
public class Cura : NPCSuperClase{
    Mensaje de Unity | 11 referencias
    protected override void Awake(){
        profile = new NPCProfile{
            ID = 4,
            Name = "Cura",
            Background = "Lleva muchos años sirviendo a la comunidad como el cura del pueblo.",
            Personality = "Bondadoso y compasivo, pero también firme en sus creencias y principios.",
            RelationshipsWithOtherNPCs = new List<string>
            {
                "Tiene una mala relación con el pescador, sin embargo a veces le aconseja.",
                "Está preocupado por la extranjera y cree que se podria marchar del pueblo.",
                "Respeta a la profesora y a menudo discute temas filosóficos con ella.",
                "Mantiene una relación tensa con el doctor, que a veces cuestiona sus creencias.",
                "Tiene una buena relaccion con el empresario, ya que le ayuda mucho en la iglesia",
                "No conoce a la cartera",
                "Tiene una buena relaccion con el barrendero debidoa que le suele ayudar en la iglesia",
                "El chef suele cocinar para los eventos religioisos que organiza"
            },
            StoryBeats = new List<string>
            {

            }
        };
        base.Awake();
    }
    Mensaje de Unity | 12 referencias
    protected override void Start(){
        base.Start();
        i = -1;
        IDnumber = profile.ID;  // Usar el ID del perfil
        Conversacion = new string[] // Inicializar la conversación
        {
            "Bienvenido, hijo mío. ¿Qué te trae por aquí?",
            "Soy el Padre Mateo, el cura de este pueblo.",
            "¿Hay algo en lo que pueda ayudarte?",
            "Ten cuidado en el bosque, se han reportado avistamientos extraños."
        };
```

Figura 3-79. Script npc de ejemplo

Script.py

Este es el uno de los dos Script que son de python y no de C# este script en concreto, se mantiene informado de los cambios en el fichero comunicacion.txt, funciona de la siguiente manera Se define un eventHandler el cual hereda de FileSystemEventHandler para sobreescribir la función on_modified (self,event) que está indicará que debe hacer el script en caso de este ser modificado.

En caso de ser este modificado se lee la primera línea que Indica que Lenguaje de programación modifico el archivo,Si la primera línea es Python la función

on_modified no hará nada debido a que el archivo se modificó por Python, en caso de que la primera línea sea C# leerá la siguiente línea que es la pregunta proporcionada por el jugador al NPC, y luego leerá el resto de líneas que contienen información del NPC como sus StoryBeats, Relaciones con otros NPC, Personalidad,etc…

Luego las líneas de código fuera de la clase MyEventHandler, configuran el script para que escuche los cambios en el archivo.

```python
import ollama,ClaseNPC,os
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
class MyEventHandler(FileSystemEventHandler):
    def on_modified(self, event):
        Question=""
        NPCid=""
        if event.src_path == archivo_a_monitorear:
            modificado=False
            with open(archivo_a_monitorear,'r') as file:
                firstline= file.readline()
                if firstline=="C#\n":
                    Question=file.readline()
                    Context=file.readlines()
                else :
                    return
            with open(archivo_a_monitorear,'w') as file:
                NPCaResponder=ClaseNPC.ClaseNPC(Context)
                file.write("Python\n")
                file.write(NPCaResponder.responder(Question))
# Ruta del script y del archivo a monitorear
directorio_actual = os.path.dirname(os.path.abspath(__file__))
archivo_a_monitorear = os.path.join(directorio_actual, 'Comunicacion.txt')  # Ajusta el nombre del archivo
observer = Observer()
observer.schedule(MyEventHandler(), path=directorio_actual, recursive=False)
observer.start()
try:
    while True:
        # Bucle para mantener el observador en ejecución
        pass
except KeyboardInterrupt:
    observer.stop()
observer.join()
with open('Comunicacion.txt', 'r', encoding='utf-8') as file: # Lee todo el contenido del archivo
    content = file.read() # Muestra el contenido print(content)
```

Figura 3-80. Script.py.

ClaseNPC.py

Este es el otro Script python y que indica cómo la inteligencia artificial tomará la personalidad del personaje que el jugador justo acaba de preguntar.

Disponemos primero del contexto que allí se indica toda la información relevante para que la Inteligencia Artificial pueda responder la pregunta (StoryBeats,Relaciones con NPC,Personalidad,etc…

Y luego de la pregunta, qué es la pregunta proporcionada por el jugador, aquí también se define que se va a usar el Modelo pre-entrenado en este caso concreto usamos el modelo pre-entrenado Llama3.

```python
import ollama


class ClaseNPC:

    def __init__(self):
        self._contexto = "SOY alguien"
    def __init__(self,contexto):
        self._contexto = contexto

    def responder(self,pregunta):

            prompt=f'Dado el siguiente contexto: {self._contexto} Responda esta pregunta de manera muy concisa {pregunta}'

            response = ollama.chat(model='llama3',messages=[{

                'role':'user',
                'content':prompt,},
            ])
            return response['message']['content']
    def isthisNPC(self,NPCiD):
        return True
```

Figura 3-81. ClaseNPC.py.

3.5.2 Interfaz

En esta sección se mostrarán todos los interfaces y explicaremos todo lo relacionado con ellos.

1. Menú de inicio

Consta de dos botones, uno para empezar el juego y otro para salir de la aplicación.

Figura 3-82. Menú de inicio.

2. Menú de pausa.

Consta de cuatro botones, uno para quitar la pausa, el siguiente para acceder al menú de opciones, el siguiente para guardar la partida y el último para salir de la aplicación.

Figura 3-83. Menú de pausa.

También puedes acceder a él pulsando este botón.

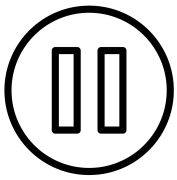


Figura 3-84. Botón de pausa.

3. Menú de opciones

Consta de dos barras para controlar el volumen y el brillo respectivamente. También tiene un botón retornar para volver al menú de pausa.

Figura 3-85. Menú de opciones.

4. Menú de victoria

Consta de un mensaje felicitando al jugador y tres botones, uno para reiniciar la partida, el siguiente para ir al menú de inicio y el último para salir de la aplicación.

Figura 3-86. Menú de victoria.

5. Chat

Se escribe en el cuadro de texto y la respuesta se muestra por el bocadillo de abajo.

Figura 3-87. Chat.

6. Interfaz alcalde

Consta de dos botones, uno para que repita la misión y el otro para acusar a algún personaje.

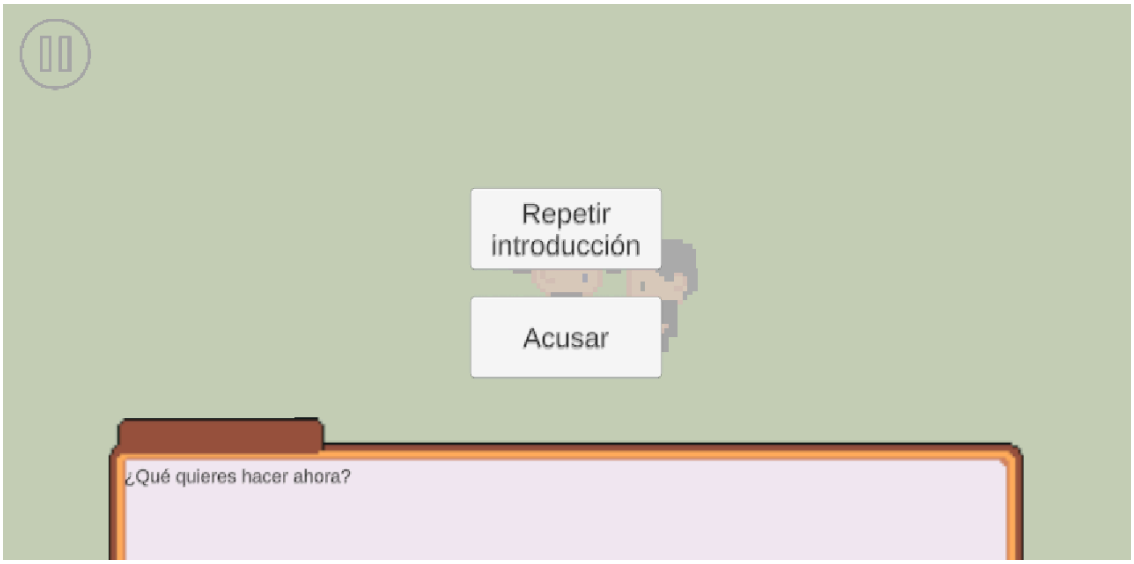


Figura 3-88. Interfaz del alcalde.

Si se pulsa acusar, se mostrará un botón para salir y varios botones con todos los personajes para acusar. Si se acierta aparece el menú de victoria y si se falla aparece un mensaje de incorrecto y se puede volver a intentar.

Figura 3-89. Menú de selección del asesino.

3.5.3 Diagramas UML

En esta sección se mostrarán algunos diagramas UML que permiten representar visualmente el diseño y comportamiento del sistema desarrollado. En concreto, se incluyen diagramas de casos de uso y diagramas de actividades.

Diagrama de casos de uso:

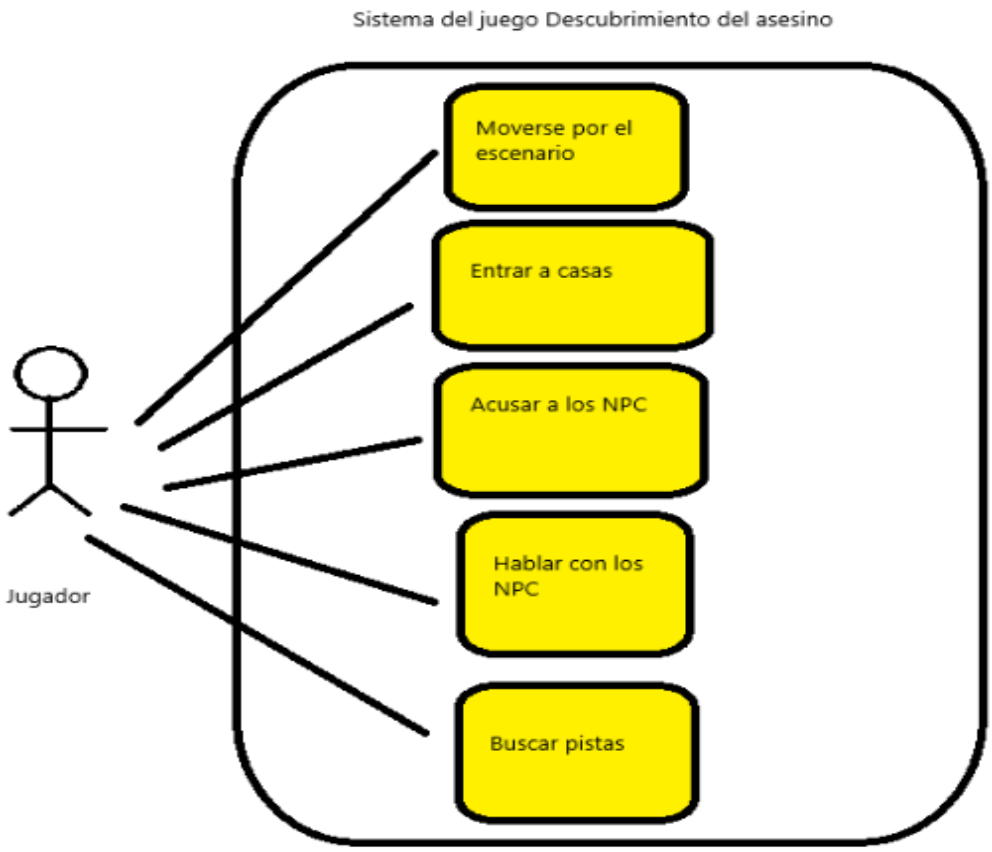


Figura 3-90. Diagrama de casos de uso.

Diagrama de actividades:

- Moverse por el escenario :

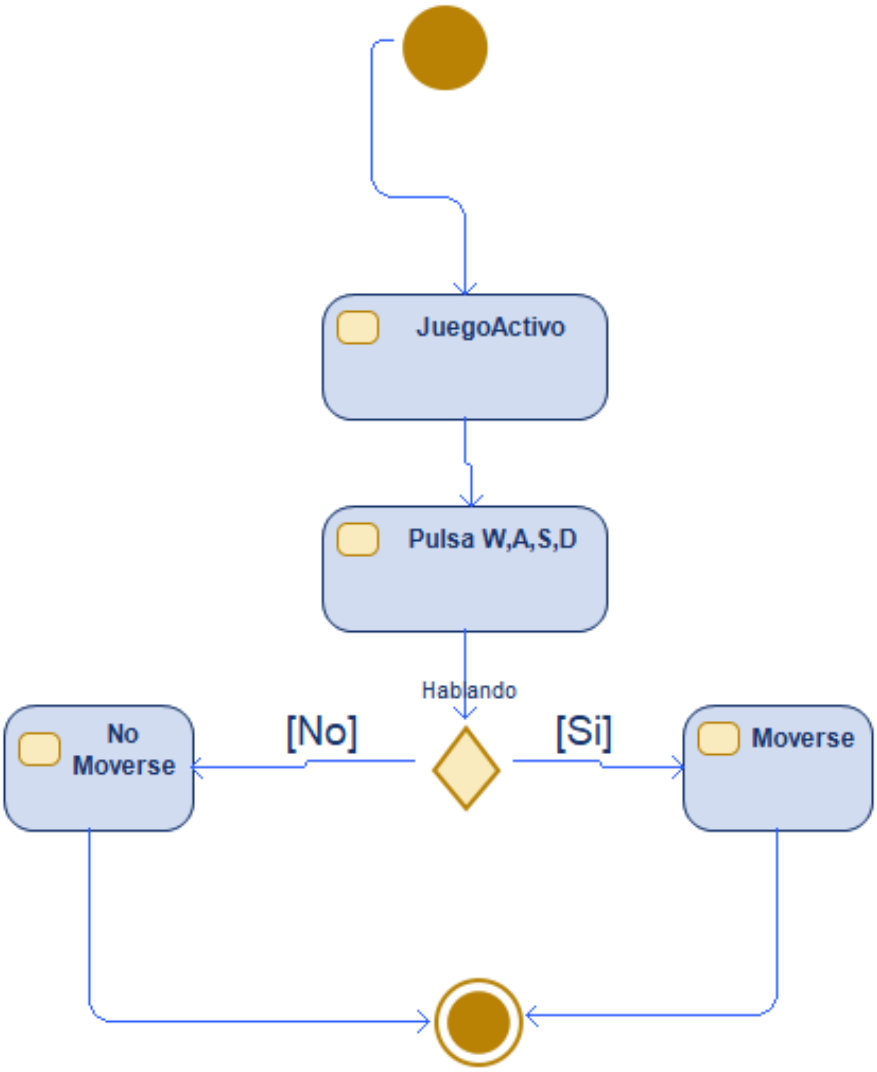


Figura 3-91. Diagrama de moverse por el escenario.

- Entrar a casas:

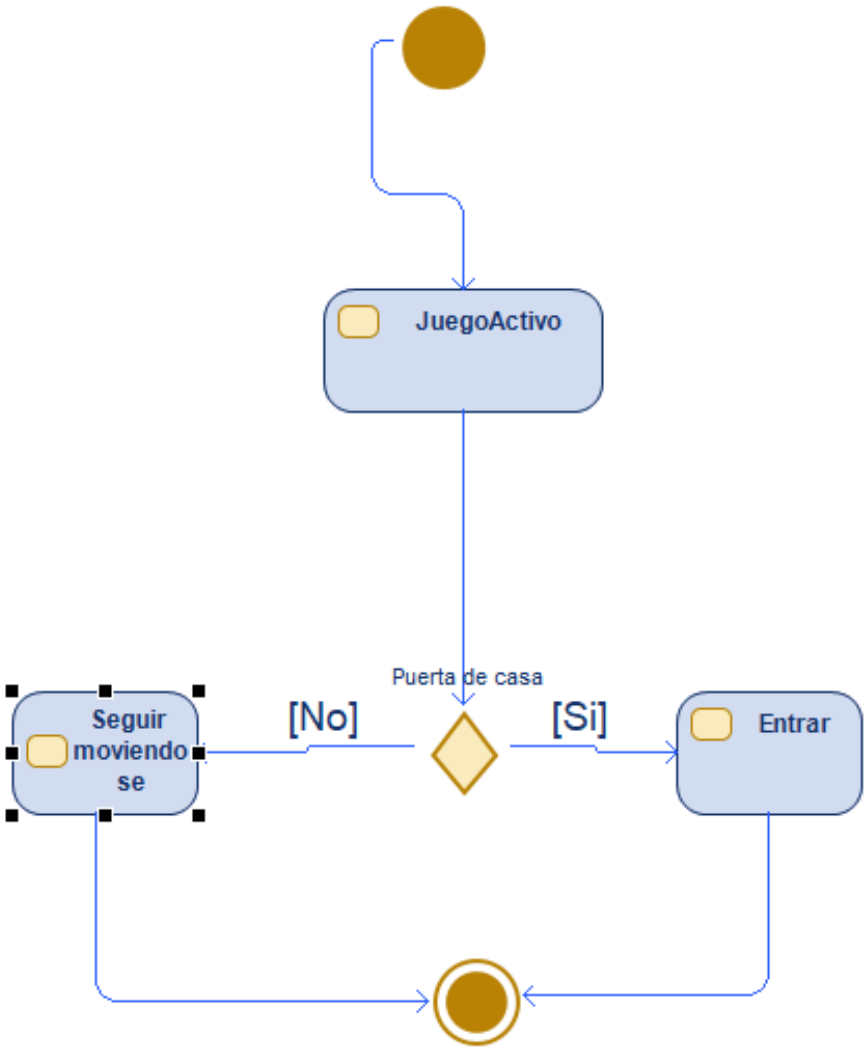


Figura 3-92. Diagrama de entrar a casas.

- Acusar a los NPC:

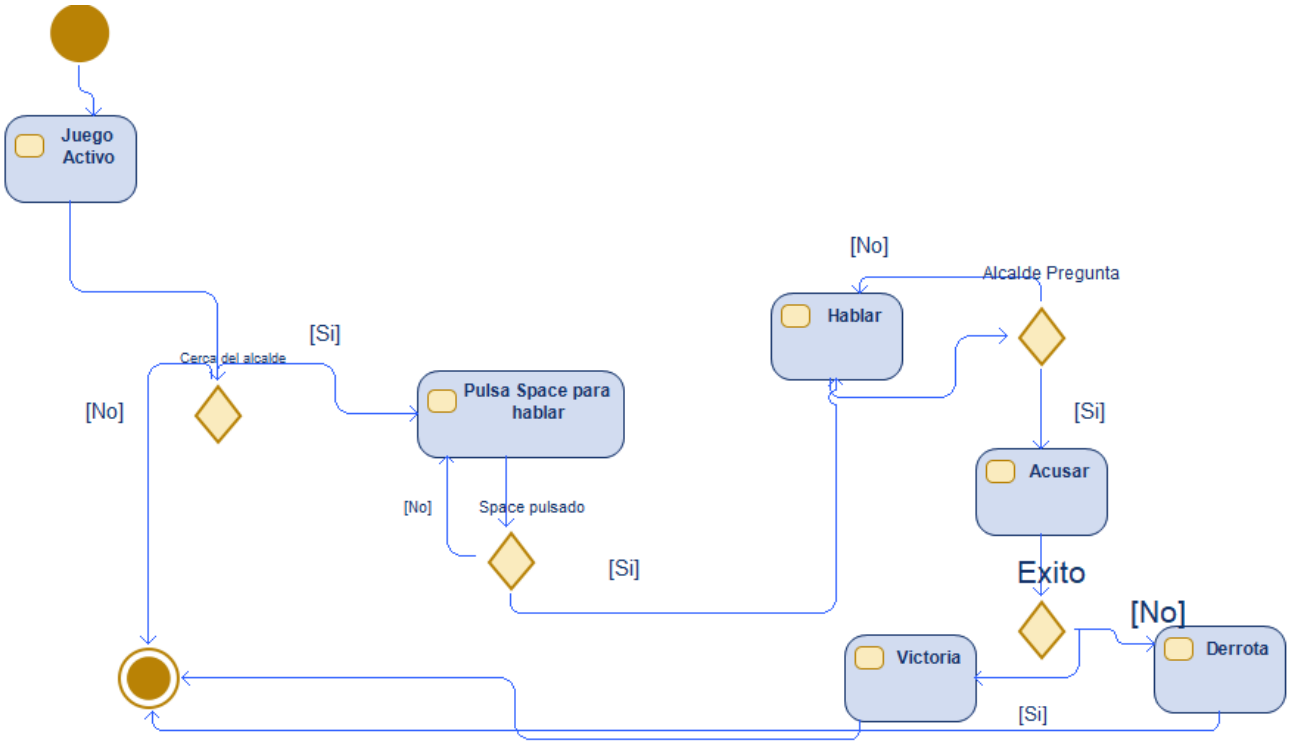


Figura 3-93. Diagrama de acusar a los NPC.

- Hablar con los NPC:

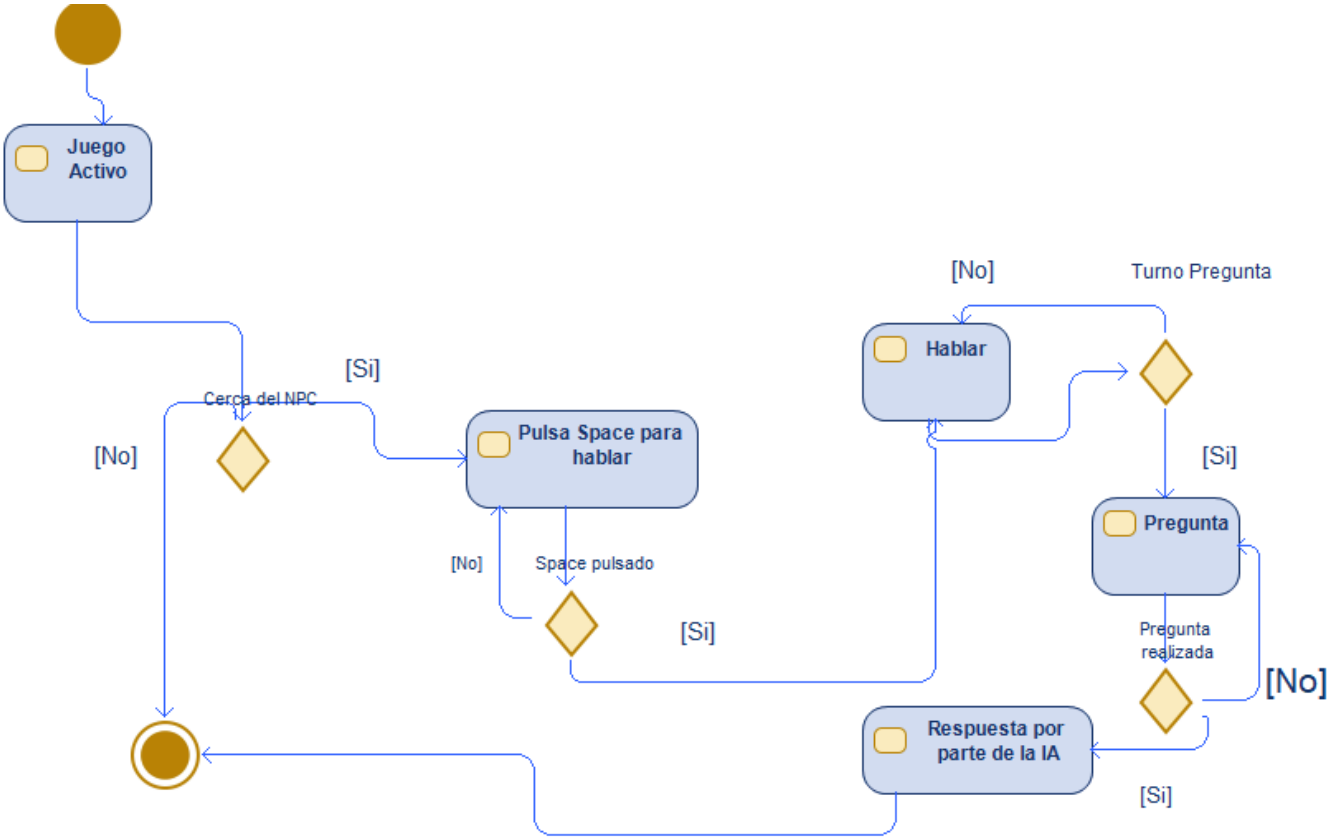


Figura 3-94. Diagrama de hablar con los NPC.

3.6 Inteligencia Artificial Generativa

3.6.1 Introducción

La Inteligencia Artificial Generativa (IAG) se ha convertido en una de las tecnologías más utilizada en la actualidad, no solo por su carácter innovador, sino por su capacidad para crear desde cero textos, imágenes, sonidos e incluso video. A diferencia de los sistemas de IA analítica, enfocados en interpretar y analizar información ya existente, la IAG se entrena con extensos volúmenes de datos para generar productos que, hasta hace poco, se consideraban únicos del ser humano. Sus usos son tan variados como fascinantes: redacta poemas, compone melodías, genera retratos que desafían la realidad o construye escenas audiovisuales. Más que copiar la creatividad, la reinventa, abriendo caminos inéditos en áreas como el diseño, la investigación o incluso la educación, donde está transformando cómo aprendemos y creamos.

3.6.2 Comunicación entre módulos

Las bibliotecas relacionadas con inteligencia artificial como por ejemplo TensorFlow, HuggingFace, PyTorch, Ollama, etc… se encuentran principalmente en Python. Por tanto, esto hace necesario que exista una comunicación entre C# y Python. Existen varias opciones para poder realizar dicha comunicación, entre ellas estuvimos probando cuál podía ser la mejor opción y se nos ocurrieron dos maneras.

La primera de ellas, que acabamos descartando, fue crear un servidor y comunicar ambos lenguajes de programación mediante el uso de una base de datos. Esta idea funcionaba bien, pero complicaba demasiado la ejecución debido a que implicaba que, aparte de estar en ejecución tanto los archivos Python como C#, tenía también que estar ejecutado el servidor. Python tenía que hacer una consulta a la

base de datos antes de poder usar el dato proporcionado por C#. Además de que dificultaba el trabajo en grupo, también había que guardar las consultas sql respectivas, como lo son la creación y la inserción de la tabla. También había pocos datos que almacenar y sumado a la volatilidad de estos, ya que cada vez que se hacía la comunicación entre lenguajes, los mensajes anteriores ya dejaban de ser necesarios.

La segunda opción fue usar ficheros de texto, esta opción fue al final la que optamos por usar en el proyecto. La idea es simple: usar un fichero para la comunicación con los dos lenguajes, facilitando el proceso ya que así no era necesario que se ejecutase un servidor de por medio. Tampoco era necesario almacenar datos en un lugar de carácter persistente como es la base de datos, además de que el fichero de texto en este caso era mucho más intuitivo y simple, aquí se explica el funcionamiento:

1. La primera línea del fichero de texto representa siempre el Lenguaje que manda dicha petición.
2. La segunda línea varía dependiendo del lenguaje utilizado si es python se manda la respuesta proporcionada por la IA, pero si en cambio el Lenguaje es C# se manda la pregunta realizada por el jugador.
3. El resto de líneas sólo aparece si C# manda la petición, ya que Python al responder solo usa 2 líneas. Estas líneas incluyen toda la información relevante de los personajes para la Inteligencia artificial como la personalidad, el oficio, los StoryBeats (por ejemplo el NPC encontró algo raro en cierto sitio, etc…), sus relaciones con los otros NPCs (por ejemplo este personaje es el hermano de este otro personaje, el personaje tiene problemas con este otro personaje, este personaje sabe cierto dato de otro personaje, etc..).

3.6.3 Inteligencia artificial empleada

En el lenguaje de programación Python disponemos de varias opciones en cuanto a Inteligencia Artificial se refiere y al igual que en el apartado anterior aquí también probamos dos opciones. La primera era usar la biblioteca Transformers de Python con PyTorch, HuggingFace y TensorFlow. Esta primera opción al principio nos resultó interesante, pero tenía unos grandes inconvenientes a la hora de usarla en el contexto de un videojuego.

El primero de ellos era que los modelos pre-entrenados que ofrece HuggingFace tienen un propósito muy específico. Es decir, por ejemplo, comenzamos usando un modelo que solo está entrenado para realizar preguntas en español, por lo que sí se preguntaba en otro idioma, por ejemplo, en inglés había incoherencias con lo preguntado.

La segunda razón fue debido al tiempo que tardaba en dar una respuesta. Por ejemplo, podría tardar mucho tiempo en responder a un ¿Quién eres?, lo que afectaba gravemente a la jugabilidad, ya que un juego debe ser lo más fluido posible, además de alargar la partida más tiempo del necesario.

Ollama fue una opción mucho más apta para un juego, ya que los modelos que ofrece esta son muy utilizados y conocidos, como Deepseek (Deepseek-r1), LLama(en concreto la versión LLama3 es la usada en este proyecto), Mistral, etc…

Por tanto se soluciono el problema del modelo entrenado para un propósito muy específico, y más tarde, haciendo varias pruebas comprobamos que Ollama ofrecía mejores tiempos que los HuggingFace, PyTorch y TensorFlow, lo que hacia una experiencia más fluida para el usuario. Hubo veces que las preguntas tardaban un tiempo muy corto en ser respondidas (<5 segundos), cosa que no ocurría con la

primera opción y muchas veces había que esperar un tiempo muy largo (>1 minuto) para ser respondidas, al menos en las pruebas realizadas.

3.6.4 Ejemplo del uso de la Inteligencia Artificial

A continuación se mostrarán ejemplos de la ejecución de la inteligencia artificial. Haremos el ejemplo del empresario, la cartera y por último el asesino en cuestión de la partida (este último se escoge al azar):

Empresario:

1. Ejemplo de cuando se le pregunta a qué se dedica el empresario

   Se le hace una pregunta:

Figura 3-95. Escena pregunta.

Se muestra que está pensando mientras se ejecuta la IA (ocurre tras cada pregunta):

Figura 3-96. Escena pensando.

Escena donde da la respuesta:

Figura 3-97. Escena de respuesta.

Ejemplo cuando se le pregunta acerca del asesinato:

Figura 3-98. Escena de pregunta sobre el asesinato.

Figura 3-98. Escena de respuesta sobre el asesinato.

Cartera:

Haremos la misma pregunta que hicimos con el empresario y efectivamente dirá que es la cartera:

Figura 3-99. Escena de pregunta 2.

Figura 3-100. Escena de respuesta 2.

Similar responderá si se le pregunta acerca del asesino (en las capturas anteriores ninguno de estos 2 fue asignado asesino)

Figura 3-101. Escena de respuesta sobre el asesinato 2.

Asesino:

En esta partida el random decidió que el asesino es el médico por tanto se le harán una serie de preguntas:

Pregunta: ¿Sabe algo acerca del asesinato?

Figura 3-102. Escena de respuesta del asesino.

Pregunta: ¿Sabe quién puede ser el asesino?

Figura 3-103. Escena de respuesta del asesino 2.

Pregunta: ¿Eres tú el asesino?

Figura 3-104. Escena de respuesta del asesino 3.

Como se ha observado el asesino engaña al jugador siempre que puede, si el jugador intenta hacer preguntas relacionadas con el crimen.

Ahora se procederá a preguntar por el crimen:

Pregunta: ¿Como fue el asesinato?

Figura 3-105. Escena de respuesta del asesino 4.

Procederemos a hacer la misma pregunta con un NPC inocente para poder deducir de allí pistas y observar comportamientos distintos.

Figura 3-106. Escena de respuesta comparativa.

Efectivamente aunque este NPC se ha informado de dicho asesinato y sabe que alguien ha comprado veneno, pero a su vez tiene menos datos que el medico que es el asesino en cuestión. Estos datos son clave para poder descubrir quién es el asesino.

Capítulo 4 - Evaluación con Usuarios

4.1 Objetivo

El propósito de esta evaluación fue comprobar la jugabilidad, usabilidad y eficacia de las mecánicas propuestas, especialmente el sistema de conversación con inteligencia artificial y la experiencia de deducción del jugador.

4.2 Perfil de los participantes

Se realizaron pruebas con 10 usuarios (estudiantes universitarios entre 20 y 25 años, algunos con experiencia en videojuegos y otros sin), con el objetivo de obtener perspectivas diversas. No se requirieron conocimientos técnicos previos.

4.3 Metodología

Durante la sesión de prueba, se pidió a cada usuario que jugara libremente una partida completa. Se observaron varios aspectos como el uso y navegación por los menús, la claridad en los diálogos con NPCs, la capacidad de comprender los objetivos del juego y las reacciones ante las respuestas generadas por la inteligencia artificial.

Tras la partida, se realizó un breve cuestionario con preguntas abiertas y cerradas, donde los usuarios valoraron algunos aspectos como el realismo de los diálogos, la facilidad del uso, el entendimiento de las mecánicas y el interés general del juego.

4.4 Preguntas

Estas son las preguntas que hemos hecho a los usuarios:

- ¿Qué opinas de la personalidad de los personajes?
- ¿Qué te parece la dificultad del juego?
- ¿Entiendes la finalidad del juego?
- ¿Crees que haya algún elemento que añadir?
- ¿Qué opinas del audio?¿Crees que la ausencia de música le aporta suspense a la trama?
- ¿Cómo te sientes con las respuestas de los personajes?
- ¿Crees que el sistema de chat es adecuado y coherente con el juego?
- ¿Qué posibles mejoras añadirías al juego?
- ¿Del 1 al 10 cuál es tu satisfacción con el juego, siendo un 1 nada y un 10 mucho?
- ¿Crees que la estética del juego es adecuada con el estilo de juego¿

4.5 Resultados

Los usuarios mostraron un alto grado de interés en la propuesta, destacando especialmente la originalidad del uso de IA generativa para mantener conversaciones. Algunos señalaron que les sorprendió la naturalidad de algunas respuestas.

A su vez, se detectaron también áreas de mejora como incluir un tutorial ya que algunos jugadores se sintieron algo perdidos al principio, también se propuso implementar un registro de conversaciones para facilitar el seguimiento de las pistas. Algún usuario comentó que echaba de menos música de ambiente en el juego sin

embargo hubo otros que indicaron que les parecía bien el juego sin música para añadir suspense a la historia.

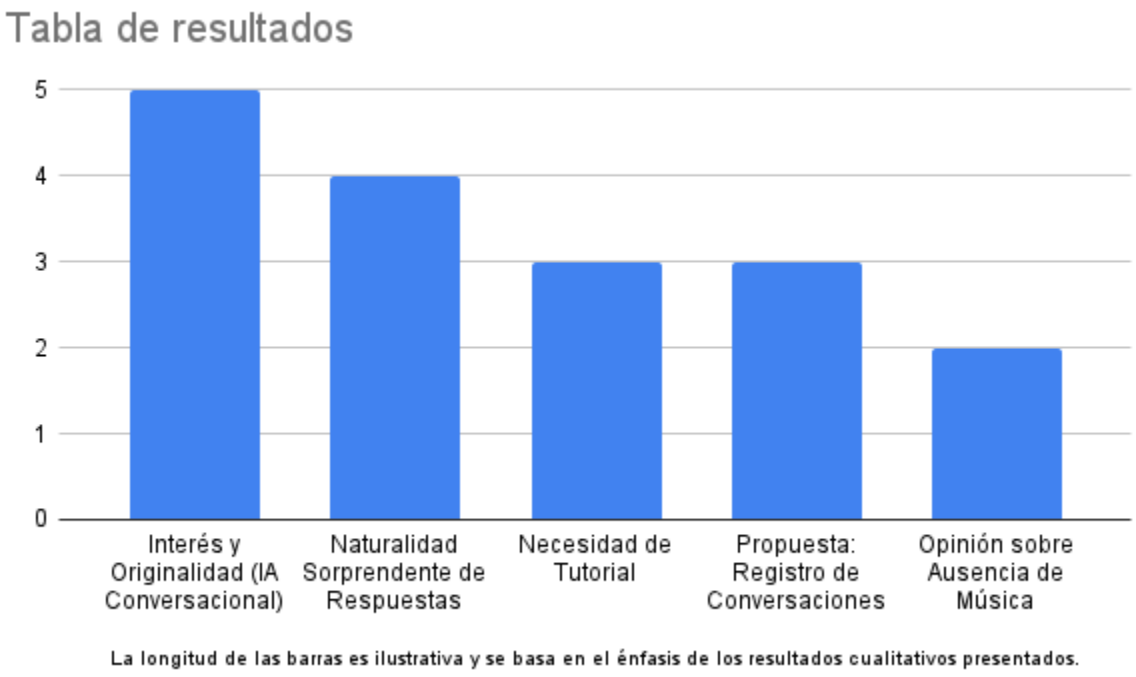


Figura 4-1. Gráfica representativa de resultados.

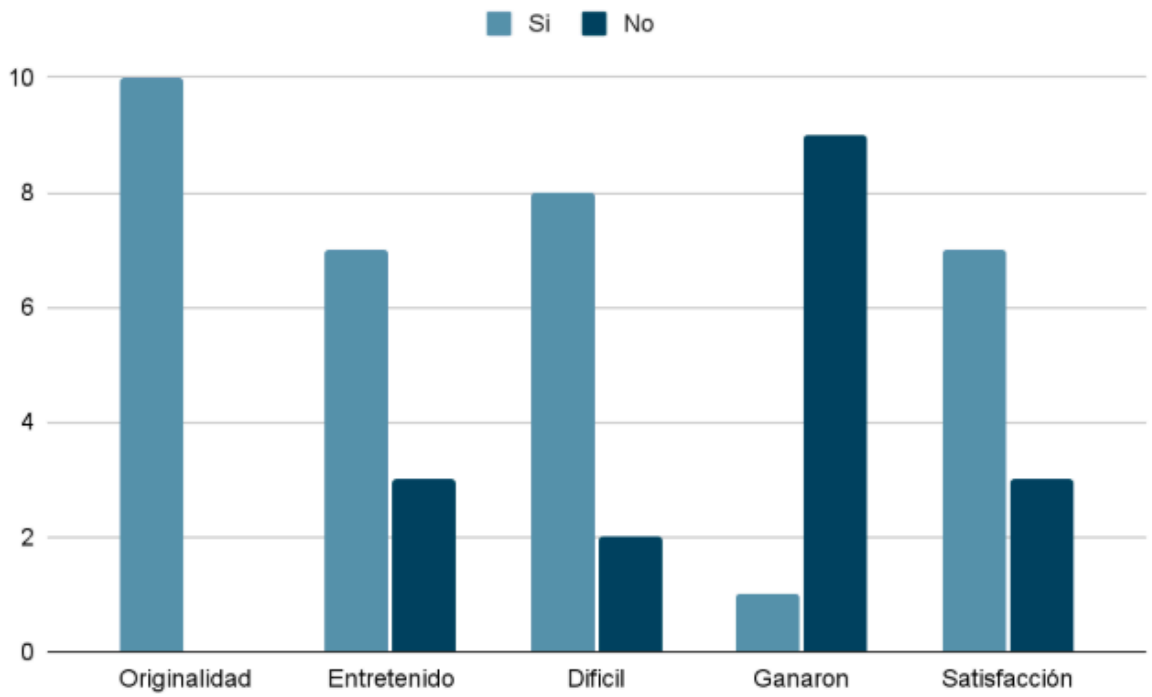


Figura 4-2. Gráfica representativa de resultados 2.

4.6 Conclusiones

Las pruebas demostraron que el prototipo cumple su función como experiencia jugable basada en deducción e inteligencia artificial. El uso de NPCs con IA generativa fue percibido como un elemento diferenciador y prometedor. Los comentarios obtenidos servirán como base para futuras mejoras centradas en la accesibilidad, la narrativa y la continuidad del progreso del jugador.

Capítulo 5 - Conclusiones y trabajo futuro

5.1 Conclusiones

El desarrollo de este proyecto ha supuesto una valiosa oportunidad para explorar el potencial de la inteligencia artificial generativa aplicada al ámbito del diseño de videojuegos. A través de la creación de un prototipo interactivo en Unity, hemos demostrado cómo esta tecnología puede integrarse para enriquecer la narrativa, incrementar la inmersión y transformar la manera en la que los jugadores interactúan con los personajes y el entorno.

Uno de los principales logros ha sido implementar un sistema de diálogo con NPCs que responde de manera coherente y contextual, permitiendo al jugador obtener pistas y contrastar información a partir de respuestas generadas en tiempo real, abriendo la puerta a experiencias mucho más dinámicas y personalizadas. Este avance, aunque aún incipiente, representa un cambio significativo frente a los modelos tradicionales basados en guiones predefinidos.

Además, se ha comprobado que la combinación de IA generativa con un diseño de personajes bien contextualizado es clave para generar diálogos coherentes y creíbles. El uso de contextos diferenciados por personaje ha permitido que las respuestas ofrecidas por la IA estén alineadas con su personalidad, su rol en la historia y su posición en el entorno del juego.

Finalmente, este trabajo no solo ha servido para comprobar la viabilidad técnica del uso de IA generativa en videojuegos, sino que también ha planteado reflexiones relevantes sobre su futuro impacto en la industria. El diseño narrativo, el rol

del desarrollador y las expectativas del jugador podrían cambiar radicalmente con el uso de estas tecnologías emergentes.

5.2 Trabajo futuro

De cara a futuras iteraciones del proyecto, se contemplan diversas mejoras y ampliaciones que permitirían enriquecer significativamente la experiencia de juego y el sistema de inteligencia artificial implementado.

En primer lugar, se plantea mejorar los contextos de los NPCs, dotándolos de una mayor profundidad narrativa. Esto permitiría que sus respuestas sean más coherentes con su historia, personalidad y relación con otros personajes del pueblo, haciendo que las conversaciones sean más realistas, variadas y personalizadas en función del momento de la partida.

También se desea añadir una pantalla de derrota para cuando se falla un número determinado de veces y se tiene en cuenta la posibilidad de añadir música ambiental y más efectos de sonido.

Además, se prevé aumentar la cantidad de personajes disponibles en el juego, lo que incrementaría la diversidad de testimonios, opiniones y pistas accesibles para el jugador. Una mayor variedad de NPCs también aumentaría la rejugabilidad y la complejidad del proceso de deducción, obligando al jugador a contrastar mejor la información entre ellos.

Otra funcionalidad importante que se desea implementar es un sistema de registro de conversaciones, que almacene los diálogos mantenidos con cada personaje. Esto permitiría al jugador consultar lo dicho anteriormente y detectar contradicciones con mayor facilidad. En relación con esto, también se busca

incorporar un sistema de guardado de partida, que conserve la posición del jugador en el mapa como el historial de las conversaciones, facilitando una experiencia de juego más larga y estructurada.

Por último, se está explorando la posibilidad de integrar objetos interactivos dentro del escenario, que puedan ser utilizados como pistas físicas en la investigación. Estos elementos añadirían una capa adicional de complejidad, permitiendo combinar el análisis de testimonios con la inspección de pruebas materiales para llegar a la verdad.

Introduction

In recent years, artificial intelligence (AI) has experienced exponential growth, establishing itself as one of the most promising and transformative technologies across various fields, from medicine to education. In particular, the world of video game development has begun to explore its potential, opening new paths to enhance interaction, immersion, and narrative within virtual environments.

This project focuses on the use of generative artificial intelligence applied to video game development—an emerging field that enables the generation of dynamic content based on user input. Within this context, one of the most interesting applications is the ability to maintain realistic and contextual conversations with non-playable characters (NPCs), endowing them with personalities and responsiveness that closely resemble those of real human beings.

This introductory chapter presents basic concepts related to serious games and outlines the work plan following an initial research phase. The second chapter analyzes related work. The third chapter explains the technologies used, the overall design, and additional functionalities, including several UML diagrams. The fourth chapter presents the results of usability testing conducted at the end of development. Finally, the fifth chapter contains the conclusions.

Motivation

With this project, we aim to showcase and provoke a critical reflection on the impact that artificial intelligence—and more specifically, generative AI—can have on the design and development of future video games. The evolution of this technology is rapidly transforming how interactive experiences are created, enabling more dynamic

worlds, emergent narratives, and characters that no longer rely on predefined dialogue but can instead respond contextually, personally, and coherently with the player's story.

Through this project, we intend to highlight the creative and narrative potential that generative AI offers, as well as explore the opportunities and challenges it presents in gaming environments. Beyond the technical aspects, this work encourages new forms of interaction between the player and the game world—interactions that could completely redefine the roles of designers, writers, and even the players themselves within the overall experience. Ultimately, this project aspires to be a small step toward understanding how artificial intelligence not only enhances game technology but also its ability to tell living stories, respond to human curiosity, and build more believable worlds.

At the same time, we aim to explore the process of creating a video game, learn the basic tools for development, understand the key elements that contribute to a fun and engaging game, and confront new challenges using a development environment that was previously unfamiliar.

Goals

The main objective of this Final Degree Project is to explore the use of generative artificial intelligence as a tool to enhance narrative and interactivity in video games, through the development of a functional prototype. The project aims to demonstrate how advances in natural language processing can be integrated into playful environments to enable more realistic, dynamic, and personalized interactions with NPCs.

To achieve this, the project focuses on designing a set of characters with distinct contexts, rich enough to support the development of emergent narrative and to generate multiple paths of deduction based on the AI-generated responses.

A secondary objective is to learn how to use Unity as a platform for video game development, with the aim of acquiring a solid foundation in the creation of interactive virtual environments. Through the exploration and practical use of its tools and functionalities, the project seeks to understand the fundamental principles of game design and programming.

Work plan

The development of the project followed the following phases for building the video game:

5.2.1 Definition of Requirements and Goals

The project began with a period dedicated to developing a detailed list of requirements and objectives that the video game was expected to meet. This phase was oriented toward deciding the type of game we wanted to create and determining where and how generative artificial intelligence could be integrated into its mechanics.

5.2.2 Learning and Testing the Unity Environment

During this phase, we fully dedicated ourselves to learning the Unity game engine, becoming familiar with its tools and basic concepts, and analyzing similar concepts and relevant articles to generate ideas for the game. We started by conducting small tests to understand how Unity works and to lay the foundation for future development. We then began implementing a basic scene (or Scene, in Unity

terminology), corresponding to a village that would serve as the main map and a testing ground before being integrated into the final game.

Each concept in the game was developed in the form of an exploratory prototype: starting from a very basic version to which features were gradually added, improving interaction, design, or usability.

5.2.3 Task Division

With the information gathered and goals clearly defined, we moved on to planning the division of tasks between the game's artistic design and its core mechanics. Specific tasks were assigned to ensure a fair and efficient distribution of work. We decided to use GitHub [4] for version control and to coordinate all contributions effectively.

5.2.4 Review and Bug Fixing

Once the implementation of the game was completed, we conducted a thorough review phase. During this stage, tests were performed to identify and correct any potential bugs, with the goal of ensuring overall system stability and full functionality. This process was essential for delivering a smooth and seamless user experience and contributed significantly to the final quality of the project.

To organize and manage the development efficiently, we adopted the agile SCRUM methodology, which facilitated collaborative work and decision-making. Through regular meetings, we assessed the progress of each part of the development process, shared any issues encountered, and adjusted the planning as necessary. We set short-term objectives, prioritizing tasks based on their impact and time cost. This workflow also allowed us to adapt the project scope in real time, making decisions such

as cutting or adjusting ideas when certain elements consumed more resources than initially expected, especially due to unforeseen technical or design challenges.

5.2.5 Project Phases

An image of the Gantt chart created in Excel, showing the timeline we followed during development, is included below.

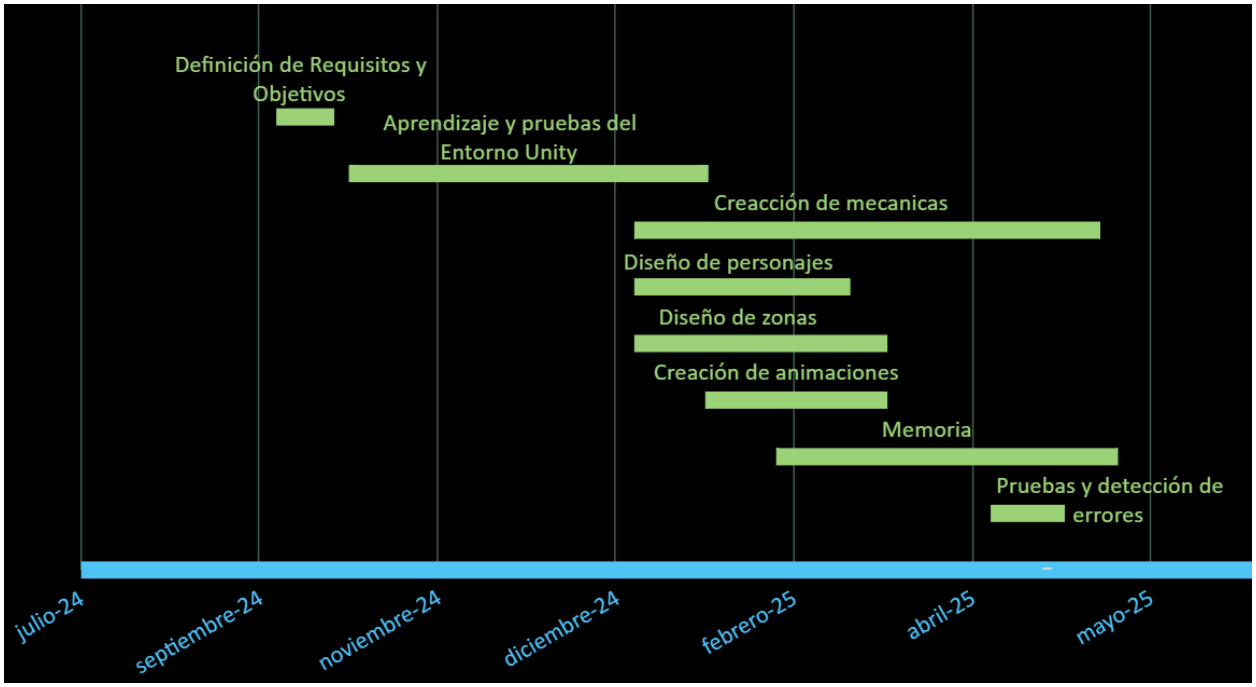


Figure 1-1. Gantt chart.

Conclusions and future work

Conclusions

The development of this project has provided a valuable opportunity to explore the potential of generative artificial intelligence applied to the field of video game design. Through the creation of an interactive prototype in Unity, we have demonstrated how this technology can be integrated to enhance storytelling, increase immersion, and transform the way players interact with both characters and their environment.

One of the main achievements has been the implementation of a dialogue system for NPCs that responds coherently and contextually, allowing players to gather clues and cross-check information based on real-time generated responses. This innovation, while still in an early stage, represents a significant shift from traditional models based on predefined dialogue trees.

Moreover, it has been shown that the combination of generative AI with well-contextualized character design is key to producing believable and coherent dialogues. By giving each character a distinct background and narrative role, the AI-generated responses are more aligned with their personalities and positions within the game world.

Ultimately, this work has not only confirmed the technical feasibility of integrating generative AI into video games, but also raised relevant questions about its future impact on the industry. Narrative design, the role of the developer, and player

expectations may be radically transformed with the widespread adoption of these emerging technologies.

Future Work

Looking ahead to future iterations of the project, several improvements and expansions are being considered that could significantly enhance both the gameplay experience and the implemented artificial intelligence system.

First, it is proposed to improve the contextual depth of NPCs, providing them with richer narratives. This would allow their responses to be more coherent with their backstories, personalities, and relationships with other characters in the village, making conversations more realistic, diverse, and personalized depending on the stage of the game.

A defeat screen is also planned for when a player fails a certain number of times, and the possibility of adding ambient music and more sound effects is being considered.

In addition, an increase in the number of available characters is planned, which would expand the variety of testimonies, opinions, and clues accessible to the player. A broader cast of NPCs would also improve replayability and add complexity to the deduction process, requiring the player to cross-reference information more carefully.

Another important feature to be implemented is a conversation log system, which would store the dialogues held with each character. This would allow the player to review previous interactions and more easily identify inconsistencies. Alongside this, the integration of a save system is also being considered, enabling the player to

preserve both their position on the map and the dialogue history, thus supporting a longer and more structured gameplay experience.

Finally, the possibility of incorporating interactive objects into the environment is being explored—items that could serve as physical clues in the investigation. These elements would add an additional layer of complexity, allowing the player to combine testimonial analysis with the inspection of material evidence in order to uncover the truth.

Contribuciones Personales

Adrián Salinas Ruiz

Ha sido el responsable de aportar la idea principal del proyecto, proponiendo el desarrollo de un videojuego basado en la mecánica de descubrimiento de un asesino mediante la interacción conversacional con personajes no jugables. La propuesta partió de la premisa de integrar inteligencia artificial generativa en la narrativa del juego, permitiendo al jugador mantener conversaciones naturales con los personajes del entorno e ir deduciendo la identidad del culpable a partir de las respuestas obtenidas.

En primer lugar, ha llevado a cabo el diseño visual completo de los personajes del juego, incluyendo la creación desde cero de cada sprite utilizando herramientas como LibreSprite. Cada personaje cuenta con varias perspectivas (frontal, trasera y laterales) manteniendo la coherencia con el estilo pixel art del juego.

Además del diseño gráfico, ha realizado la animación de todos los personajes, creando y configurando los Animator Controllers correspondientes, así como las máquinas de estados para gestionar sus diferentes movimientos (caminar, quedarse quieto, etc.) y creando los scripts para que el sprite del jugador cambie en función de hacia donde se esté moviendo y que los personajes reaccionen al movimiento del protagonista. También se ha trabajado en la configuración precisa de las cajas de colisión (colliders) de cada personaje, garantizando una interacción adecuada con el entorno y con otros objetos del juego como las puertas.

En el apartado de interfaz, se han implementado diversas pantallas de menú: el menú de inicio, el menú de pausa, el menú de opciones y el menú de victoria. Cada uno de estos menús cuenta con funcionalidades específicas, como detener el tiempo del juego, configurar subiendo o bajando el volumen o el brillo, reiniciar la partida, volver al menú principal o cerrar la aplicación, permitiendo al jugador controlar la experiencia de manera intuitiva y accesible.

También ha sido responsable de la implementación de varios efectos de sonido dentro del juego, con el objetivo de mejorar la inmersión y la retroalimentación auditiva para el jugador. Entre los sonidos añadidos se encuentran efectos como el sonido de pasos al caminar y el sonido de puertas al abrirse o cerrarse, que ayudan a dar vida al entorno y refuerzan las acciones realizadas en pantalla. Para ello, se han seleccionado archivos de audio adecuados (obtenidos de bibliotecas como Freesound) y se han integrado cuidadosamente mediante componentes AudioSource en Unity. Además, se ha configurado su activación mediante scripts, asegurando que los sonidos se reproduzcan únicamente en los momentos adecuados, sin solapamientos ni cortes abruptos.

También ha contribuido activamente en el desarrollo del código del proyecto, participando en la depuración y mejora de diversos scripts. Esta labor incluyó la identificación y resolución de problemas y errores tanto a nivel lógico como funcional, así como la optimización del código y del comportamiento de diferentes componentes del juego.

DANIEL GALLEGO BADIA

Ha aportado en la toma de ideas iniciales de como hacer un juego usando la inteligencia artificial generativa durante el periodo inicial del proyecto, el enfoque y su posible uso en el juego.

Se ha responsabilizado de crear y diseñar el mapa del juego. Esta tarea incluyó la planificación del diseño de escenas, estableciendo los límites y las colisiones necesarias para garantizar una experiencia de juego lógica y coherente con el entorno. Además, ha trabajado en la movilidad entre escenas mediante colisionables invisibles moviendo al jugador a la posición correspondiente manteniendo el flujo y evitando fallos manteniendo los movimientos entre escenas naturales.

También para crear y diseñar el mapa el estudiante tuvo que buscar posibles imágenes que se adecuen a la idea y diseño del juego y que mantengan la estructura necesaria para poder ser usadas como tileset en Unity. Además el alumno realizó labores de edición de imágenes para eliminarles el fondo y corregir alguna imperfección.

Otra contribución del estudiante ha sido gestionar a los personajes en el juego. Para ello ha escrito la información que se pasa la Inteligencia Artificial en forma de texto para generar un promp y hacer que la IA conozca el pasado, personalidad y conexiones entre personajes para cada uno manteniendo la coherencia entre ellos y dándoles un poco más de trasfondo.

Además de se ha encargado de diseñar un sistema para hacer que las partidas sean casi siempre distintas, haciendo que el asesino cambie en función de la partida. Y desarrollando un sistema de pistas que se reparte entre los personajes que no son el asesino haciendo que cada personaje cambie de rol en cada partida y solucionando el problema de hacer el juego demasiado repetitivo. Para el asesino si se le han asignado sus propias pistas falsas para poder inventar coartadas o desviar la atención del detective, las pistas del asesino si son estáticas y se repiten todas las partidas para simplificar la decisión de la IA y el juego.

También el alumno se ha encargado del menú de acusación y repetir la información de la introducción. Ha creado un menú con dos botones uno para repetir la introducción y otro para acusar un personaje.

En resumen el alumno se ha encargado principalmente del apartado escénico y hacer que el juego sea jugable utilizando al alcalde como juez y preparando el sistema de promps y elección del asesino.

ZAKARIAE LAKTIOUET SAIDI

Aportación de ideas al principio del proyecto para poder dar comienzo al juego con inteligencia artificial generativa.

Implementación de los respectivos canvas donde se muestran dichas conversaciones con los respectivos personajes del juego, implementación de la caja de conversación, implementación del campo de entrada que se usará para poder realizar la pregunta propuesta por el jugador en el caso de este juego.

Implementación de Personajes tanto jugadores como NPCs de prueba al principio para poder probar las funcionalidades principales de inteligencia artificial y movimiento.

Implementación de los gran parte de los scripts entre los que se incluyen el movimiento del personaje, el script que hace que la cámara persigue al jugador, script de movimiento, script de la SuperClaseNPC donde se indica el comportamiento general con el que cumple la mayoría de Personajes no jugables por ejemplo cómo se comporta el NPC dependiendo de las ciertas condiciones como pueden ser la distancia hacia el jugador, como se va actualizando en base a cuando el jugador pulsa space cerca del NPC, hacer que el NPC pueda monitorizar el fichero cuando tenga que responder al personaje ,etc…

También hubo aportaciones a los scripts como script del doctor, script del cura, script de la profesora, script del aldeano, extranjera, cocinero, barrendero, pescador y alcalde.

También la implementación de la conexión entre los lenguajes Python y C#, decisión de la mejor opción para realizar dicha conexión entre ambos lenguajes de todas las opciones posibles(BD,fichero de texto,etc…) implementación del script python que escucha los cambios del fichero comunicación.txt, implementación de la inteligencia artificial como el uso de la biblioteca de python Ollama usada para poder responder dichas preguntas mediante el uso de inteligencia artificial generativa, En concreto creando la Clase NPC que es como se va a comportar la inteligencia artificial generativa del NPC configurando el respectivo prompt, el respectivo modelo de lenguaje a usar.

Pruebas de las muchas posibilidades relacionadas con la Inteligencia artificial (PyTorch, HuggingFace, Ollama) decidiendo en base a ciertos criterios cuál es la mejor

opción para el juego como lo puede ser el tiempo y la coherencia de las respuestas por parte de dichos modelos.

En resumen el alumno se ha encargado de dar comienzo al proyecto creando escenarios de prueba, personajes de prueba tanto jugables como no jugables,creando scripts de movimiento y de persecución al jugador por parte de la cámara,también desarrolló gran parte del comportamiento de los NPCs y sus respectivos canvas, y también de la Inteligencia Artificial y la respectiva comunicación entre lenguajes.

Bibliografía

[1] Ait, A., Izquierdo, J. L. C., & Cabot, J. (2024). HFCommunity: An extraction process and relational database to analyze Hugging Face Hub data. *Science of Computer Programming*. Elsevier.

[2] Freesound. (2023). Freesound [Repositorio de efectos de sonido]. Universitat Pompeu Fabra. https://freesound.org/

[3] Game Freak. (2006–2010). Serie Pokémon para Nintendo DS [Videojuegos]. Nintendo.

[4] GitHub, Inc. (2023). GitHub [Repositorio de código y control de versiones]. https://github.com/

[5] GitHub, Inc. (2023). GitHub Desktop (versión 3.2) [Aplicación de gestión de repositorios]. https://desktop.github.com/

[6] GIMP. (2025). GIMP [Software]. https://www.gimp.org

[7] Hardiman, J. P. W., Thio, D. C., & Zakiyyah, A. Y. (2024). AI-powered dialogues and quests generation in role-playing games using Google's Gemini and Sentence BERT framework. *Procedia Computer Science*. Elsevier.

[8] Hasbro. (1949). Cluedo [Juego de mesa]. Hasbro Gaming.

[9] LibreSprite. (2023). LibreSprite (versión 1.0) [Editor de gráficos pixelados]. https://libresprite.github.io/

[10] Microsoft Corporation. (2023). Visual Studio Code (versión 1.83) [Editor de código]. https://code.visualstudio.com/

[11] Pinterest. (2025). Pin sobre casas [Imagen]. Pinterest. https://www.pinterest.com

[12] Song, Y., Wu, K., & Ding, J. (2024). Developing an immersive game-based learning platform with generative artificial intelligence and virtual reality technologies–"LearningverseVR". *Computers & Education: X Reality*. Elsevier.

[13] Sun, Y., Wang, H., Chan, P. M., Tabibi, M., Zhang, Y., Lu, H., Chen, Y., Lee, C. H., & Asadipour, A. (2025). Bring game characters to the social space: Developing storytelling community AI agents driven by LLMs. Entertainment Computing, Elsevier.

[14] Unity Technologies. (2023). Unity (versión 2023.1) [Motor de desarrollo de videojuegos]. https://unity.com/

[15] Unity Technologies. (2023). Unity Hub (versión 3.6) [Aplicación de gestión de desarrollo]. https://unity.com/download