

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería de Computadores

Curso 2024-2025

Trabajo Fin de Grado

**DESARROLLO E IMPLEMENTACIÓN DE UNA IA
CONVERSACIONAL PARA MEJORAR LA
INMERSIÓN EN VIDEOJUEGOS**

Autor: Violeta Trujillo Almeida

Tutor: Nicolás Rodríguez Uribe

Agradecimientos

En el ámbito académico quiero darle las gracias a la universidad y a todos los profesores que me han ido acompañando a lo largo de este viaje, especialmente mi tutor Nicolás Rodríguez Uribe, por su gran profesionalidad y apoyo como docente, tanto a lo largo de la carrera, como en este proyecto.

Quiero también mostrar mi agradecimiento a mis compañeras de grado de videojuegos del campus de Quintana **Marta Rodríguez Castillo** y **Elena López-Negrete Burón** por haberme cedido parte de su arte y haberme ayudado para poder mostrar visualmente el potencial de lo desarrollado. También especial mención a la fuente gratuita de assets para videojuegos [1]

Quisiera expresar mi más sincero agradecimiento especialmente a mis padres, porque se han esforzado enormemente para que yo pudiese estudiar una carrera universitaria.

Por último darle las gracias a mi hermano mayor, Ricardo Trujillo Almeida porque es la persona que más me ha apoyado en absolutamente todo a lo largo de mi vida, se encontráse o no en condiciones para hacerlo. Hoy y para siempre mil millones de gracias.

Índice de contenidos

Índice de figuras	IX
Índice de códigos	1
1. Introducción	3
1.1. Contexto y alcance	4
1.2. Estructura del TFG	6
2. Objetivos	7
2.1. Descripción del Problema	7
2.2. Diseño de NPCs	8
2.3. Estudio de alternativas	9
2.3.1. Sistemas de Utilidad (US)	9
2.3.2. ChatGPT y OpenAI	10
2.4. Metodología Empleada	11
3. Desarrollo e implementación	14
3.1. Diseño de personajes	15
3.2. Comunicación con la API de OpenAI	17
3.2.1. Modelo GPT, uso de tokens y creación de una API KEY	18
3.2.2. Instalación y configuración de Postman	20
3.3. Comunicación Unity-OpenAI	24
3.3.1. Diseño de interfaz de usuario	24
3.3.2. Diseño del modelo de datos	25
3.3.3. Peticiones Web desde Unity	27
3.3.4. Desarrollo e implementación técnica	29
3.4. Fase de tuneado	31
3.4.1. Investigación de la documentación de OpenAI	32
3.4.2. Diseño y desarrollo del Fine-Tuning mediante Python	33
3.4.3. Implementación del modelo en Unity	35
3.5. Código Github	36
4. Resultados	37

4.1. Análisis de resultados	43
4.1.1. Modelo Estándar (Sin Fine-Tuning)	43
4.1.2. Modelo Amable (Fine-Tuning 1)	44
4.1.3. Modelo Egoísta (Fine-Tuning 2)	44
5. Conclusiones y trabajos futuros	47
5.1. Conclusiones	47
5.2. Trabajos futuros	48
Bibliografía	51

Índice de figuras

1.1. Esquema de actuación	4
1.2. Esquema de necesidades del jugador	5
2.1. Esquema de un sistema conversacional hecho con Sistema de Utilidad	10
2.2. Esquema comunicación NPC-API (OpenAI)	11
2.3. Tablero Trello con las tareas finalizadas	13
3.1. Modelo de datos del NPC	16
3.2. Implementación en Unity del modelo de datos del NPC	17
3.3. Modelos GPT	19
3.4. Interfaz de Post-man con sus características resaltadas	21
3.5. Tipos de solicitudes en Post-man	21
3.6. Cabecera en Post-man	22
3.7. Body de la solicitud HTTP	23
3.8. Response de la solicitud HTTP	23
3.9. Diseño avanzado del modelo de datos	26
3.10. Declaración del diseño avanzado del modelo de datos en Unity . .	27
3.11. Diseño y comunicación de scripts de Unity	30
4.1. Escenario	38
4.2. Primer Pop Up GUI	39
4.3. Elementos presentes durante la interacción	41
4.4. Ejemplo de mensaje de interacción	42
4.5. Respuesta del NPC	42
4.6. Análisis de Resultados: Modelo Estándar	43
4.7. Análisis de Resultados: Modelo Amable	44
4.8. Análisis de Resultados: Modelo Egoísta	45

Índice de códigos

3.1. Unity Web Request	28
3.2. Unity Web Request 2	28
3.3. Unity Web Request 3	28
3.4. Unity Web Request 4	28
3.5. Subida de archivos a OpenAI mediante Python	34
3.6. Creación de punto de trabajo openAI mediante Python	35

1

Introducción

Tras haber conseguido mediante el desarrollo e implementación de un videojuego **JRPG (Japanese-Role-Playing-Game)** por turnos, un diseño capaz de sumergir al jugador en el gameplay del videojuego, y una implementación técnica lo suficientemente enriquecida se consigue una **credibilidad, realismo e inmersión** al jugador.

En la continuación este proyecto se busca realizar el perfeccionamiento de la **inmersión, credibilidad y realismo** de un videojuego **JRPG**. Con el diseño del entorno previamente conseguido y habiendo alcanzado así un sistema de combate inmersivo se puede dar el siguiente paso.

Para perfeccionar este objetivo, la inmersión del videojuego, se decide hacer especial hincapié en otro elemento más presente e importante en los videojuegos **JRPG**: Las conversaciones y los personajes con los que la protagonista tiene que interactuar. Para alcanzar la experiencia más inmersiva posible para el jugador es necesario mejorar los aspectos más importantes del videojuego **JRPG** desarrollado:

1. Diseño previo del entorno. (Realizado)
2. Combate **JRPG** lo más creíble, realista e inmersivo posible a través de un SU (Sistema de utilidad). (Realizado)
3. Narrativa (Lo que se desarrolla en este proyecto)

Teniendo en cuenta esto, se pasa a conseguir **credibilidad, realismo e inmersión** en el otro elemento más importante de los videojuegos **JRPG**, la narrativa. Concretamente **la narrativa a través de los personajes presentes en él**, es decir, **NPCs (Non Player Controllers)**. De esta manera el esquema principal de actuación de los **NPCs** se puede definir en el siguiente orden:

1. Solicitud HTTP de Unity (**UnityWebRequest**) [2] → API de OpenAI.
2. Respuesta de la API de OpenAI (**Modelo Fine-Tuning / JSON Response**) → almacenamiento de una variable de Unity.
3. Variable de Unity (**String**) → Print visible para el jugador (**TextMeshProUGUI**).

Esta serie de pasos, se pueden resumir en un pequeño bucle que se representa en la siguiente Figura 1.1: De esta manera, se consigue perfeccionar los aspectos

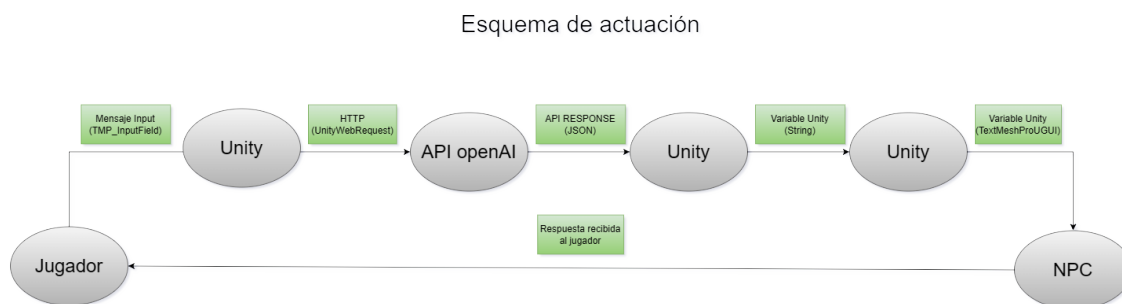


Figura 1.1: Esquema de actuación

perseguidos para este videojuego. Culminando en una clara mejoría a la hora de alcanzar **credibilidad, realismo e inmersión**.

1.1. Contexto y alcance

En definitiva, teniendo en cuenta el diseño del entorno previo del proyecto, junto a la implementación de un combate por turnos **JRPG** con un Sistema de Utilidad para los enemigos, combinados con la implementación de una IA Narrativa Conversacional, se consigue perfeccionar la **credibilidad, realismo e inmersión** del jugador. Este último paso se consigue mediante la comunicación de Unity con un modelo Fine-Tuneado de OpenAI.

Se logra así abarcar todos los aspectos más importantes con los que el jugador puede interactuar en este tipo de videojuegos, como se ve en la siguiente Figura 1.2:

Esquema de necesidades del jugador

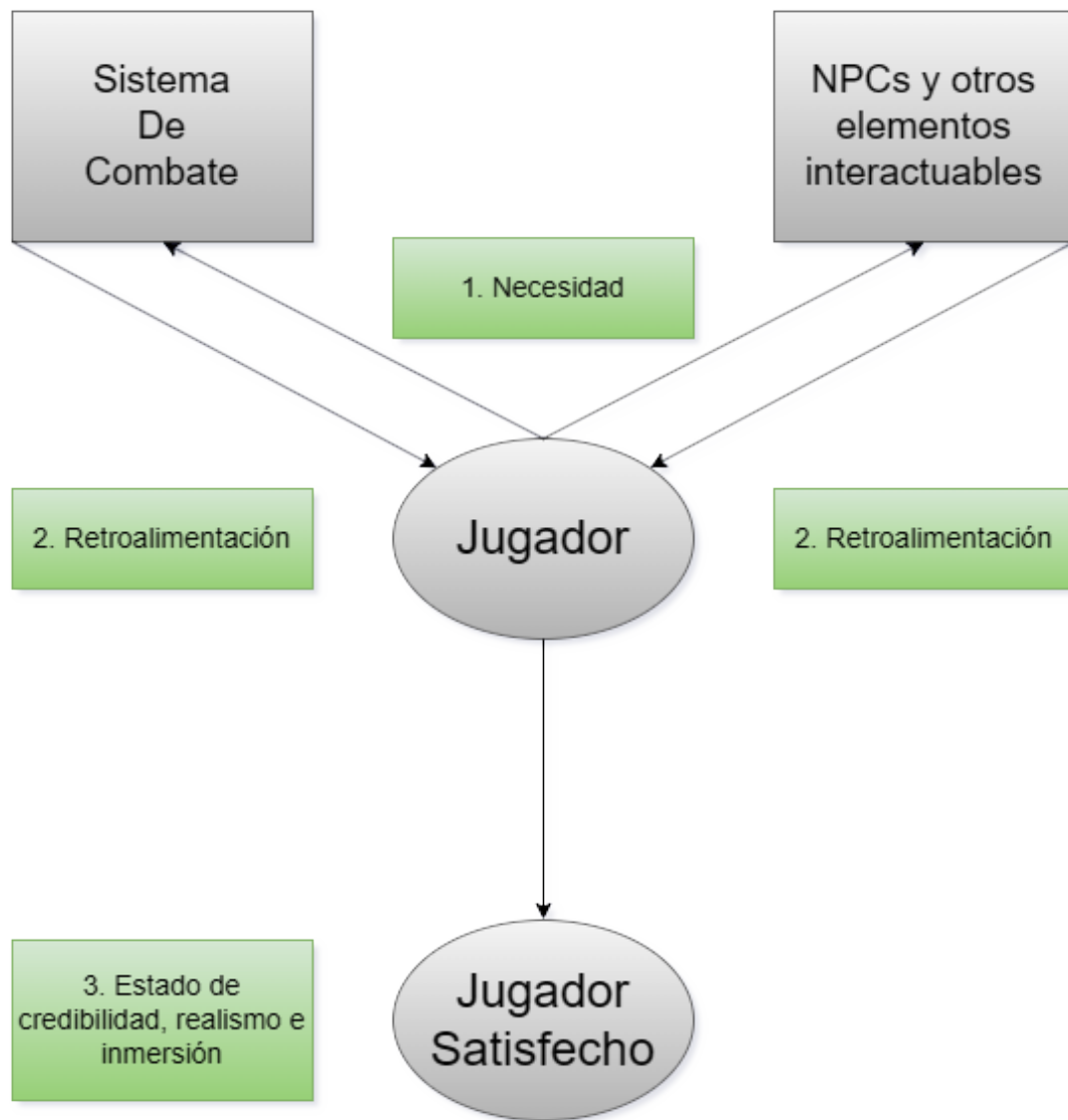


Figura 1.2: Esquema de necesidades del jugador

1.2. Estructura del TFG

La estructura de la memoria final es la siguiente:

1. Capítulo 1: Introducción
2. Capítulo 2: Objetivos (incluyendo descripción del problema, y metodología empleada).
 - a) Descripción del problema.
 - b) Diseño de NPCs.
 - c) Estudio de alternativas.
 - d) Metodología Empleada
3. Capítulo 3: Desarrollo e implementación.
 - a) Diseño de personajes.
 - b) Experimentación previa con la API de OpenAI, comunicación básica (Post-man).
 - c) Desarrollo e implementación de la comunicación de Unity con la API de OpenAI.
 - d) Perfección del desarrollo técnico (Fine Tuning).
4. Capítulo 4: Resultados.
 - a) Recogida de datos con el uso de Fine-Tuning
5. Capítulo 5: Conclusiones y trabajos futuros.

2

Objetivos

El objetivo de este proyecto es perfeccionar los aspectos del videojuego previamente diseñados para mejorar la experiencia del jugador, concretamente mediante la narrativa e interacción con NPCs, y qué además, sea *creíble, realista e inmersiva*. Una vez explicado el trabajo ya realizado, en los siguientes puntos se explica de que manera se perfecciona este aspecto.

Esto se hace a través de ciertas herramientas que ayudan a simular comportamientos y emociones humanas.

2.1. Descripción del Problema

El diseño este videojuego de combate por turnos no solo presenta el desafío de haber creado un Sistema de Utilidad, que hace que la experiencia de combate sea lo más realista, creíble e inmersiva posible, sino también el desarrollo de otra **Inteligencia Artificial (IA)** capaz de interactuar de manera creíble y emocionalmente significativa con el jugador. Es crucial que los **personajes no jugables (NPCs)** se perciban como auténticos y relevantes para la experiencia del juego.

2.2. Diseño de NPCs

El primer paso antes de entrar en detalle sobre los múltiples tipos de herramientas que se pueden usar para el desarrollo del objetivo, es pensar en el diseño previo que se busca.

Como se ha explicado anteriormente, si se parte del perfeccionamiento técnico a través de un sistema conversacional de **NPCs**, se debe diseñar ese o esos **NPCs**. En este tipo de videojuegos (**JRPGs**), es importante la variedad y dinamismo de la respuesta. También lo es la interacción con los diferentes personajes.

Esta serie de NPCs se pueden establecer en base a su **personalidad**, y una serie de **características individuales** para cada uno. Las diferentes personalidades para abarcar el máximo de variedad posible son:

1. **Personalidad amable:** caracterizada por respuestas y comportamientos dulces hacia el jugador.
2. **Personalidad egoísta:** caracterizada por respuestas y comportamientos desagradables hacia el jugador.
3. **Personalidad estándar:** caracterizada por respuestas y comportamientos estrictamente formales y típicos hacia el jugador.

Una serie de características generales asociadas a cada NPC, de forma individual son:

- Nombre.
- Trabajo.
- Ubicación.
- Familia.
- Función con el jugador.
- Información adicional (sueños, hobbies, etc...).

Con todas estas características únicas, sumadas a cada tipo de personalidad, se consigue esa **credibilidad, realismo e inmersión** buscada, y se puede proceder a hacer un estudio de alternativas que consiga con mayor eficiencia este diseño.

2.3. Estudio de alternativas

Teniendo en cuenta el problema y objetivo general explicado, se busca explorar diferentes formas de lograr una inmersión realista en las conversaciones con los **NPCs**. Se exploran las máquinas de estado finitas (**Finite State Machine, FSM**) y los arboles de comportamiento (**Behaviour Trees, BH**), pero se descartan por su escasa flexibilidad [3]), quedan dos posibilidades, debido a que lo más práctico para generar conversaciones interactivas personaje-jugador son:

- Sistemas de Utilidad.
- Comunicación a través de Redes Neuronales.

2.3.1. Sistemas de Utilidad (US)

Los sistemas de utilidad se trata de un método cada vez más popular para la creación de comportamientos. Elimina la modularidad y poca flexibilidad de los explicados **FSM** y **BH**. Se trata de un enfoque de IA que usa una herramienta basada en utilidades, es decir, mediante una serie de asignaciones de valor (pesos) que se le asigna a una determinada acción. Se decide cual es la mejor decisión que puede tomar la IA en ese preciso instante. Este tipo de sistema está basado en las redes neuronales [4]. Su esquema básico de funcionamiento se resume en:

1. **Acciones:** se trata del conjunto de acciones posibles que puede realizar la IA
2. **Factores de decisión:** las condiciones que asignarán los pesos a determinadas acciones determinando que importancia tiene cada acción, en según que situación, y porqué.
3. **Factores de fusión:** cuando se unen las condiciones de varios factores de decisión, en un solo factor, para asignar un determinado peso.
4. **Pesos, o suma de los mismos:** el peso asignado a cada una de las acciones. La acción con mayor peso será la elegida. Esto se dará tras haber tenido en cuenta los factores de decisión.
5. **Mejor acción decidida:** La acción con mayor peso que será ejecutada por la IA.

A pesar de la clara mejora en flexibilidad, no es suficiente como para abarcar una gran cantidad de interacciones con múltiples tipos de **NPCs** que puedan aportar este realismo buscado. Esto es principalmente porque al necesitar *n infinitos* tipos de interacciones por parte del jugador con el **NPC** (**El input del jugador**

puede ser cualquiera, por ello se habla de términos de infinito), se necesitaría *n infinitos tipos de respuesta* como se puede observar en la Figura 2.1. Algo mucho más complejo de gestionar en el diseño de un sistema de utilidad.

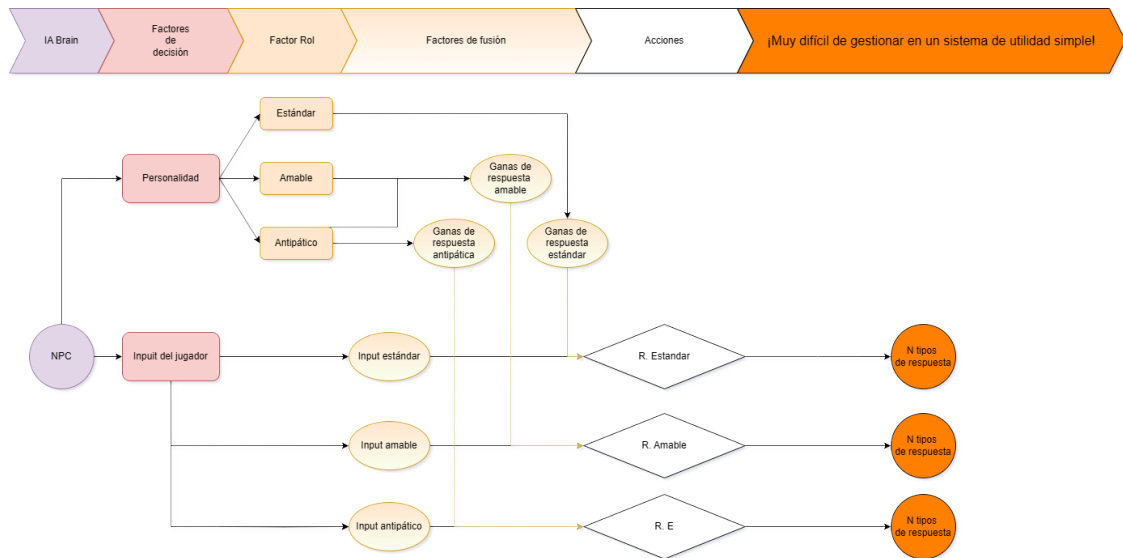


Figura 2.1: Esquema de un sistema conversacional hecho con Sistema de Utilidad

2.3.2. ChatGPT y OpenAI

Al ver que un diseño de un Sistema de Utilidad es algo que se queda escaso para lograr el objetivo previamente expuesto, es necesario dar el siguiente paso. Dicho de otra manera, alcanzar un modelo que funcione a través del autoaprendizaje (entre otros) a partir de la **Retroalimentación Humana (RLHF)** y **Procesamiento del Lenguaje Natural (NLP)** [5], y que sea capaz de proporcionar respuestas adecuadas y soportar ilimitadas interacciones del jugador.

En definitiva, *ChatGPT* es la elección perfecta para representar este tipo de respuesta, ya que se trata de un generador de conversaciones a través de una red de redes neuronales [4], basadas en múltiples tipos de aprendizaje en Inteligencia Artificial. [6].

Por eso mismo se trata de la alternativa más adecuada para este caso, una forma de representar esto a algo más de bajo nivel entendiendo que por debajo de Unity, existen una serie de scripts simulando un determinado comportamiento. Esto se ve en la Figura 2.2. De esta manera y como se va a ver a lo largo del desarrollo de este proyecto, se va a profundizar en como se ha logrado esta comunicación, per-

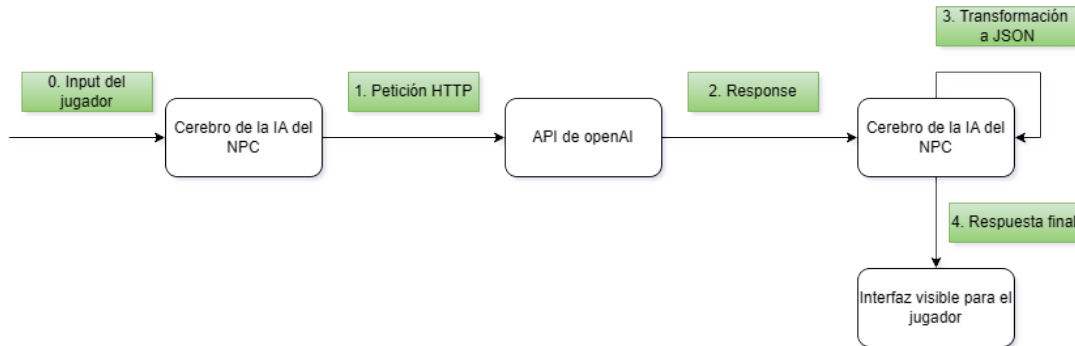


Figura 2.2: Esquema comunicación NPC-API (OpenAI)

feccionando el modelo proporcionado por ChatGPT [6] lo máximo posible para obtener la respuesta que consiga alcanzar los objetivos de este TFG.

2.4. Metodología Empleada

Para alcanzar estos objetivos, se empleará la metodología **KANBAN** [7], que permite gestionar de forma ágil y unilateral el proyecto. Es un modelo perfecto para un proceso que implique desarrollo iterativo e individual, basándose principalmente en el manifiesto ágil [8]. El proceso incluye:

- **Planificación Inicial:** Definir las tareas necesarias para el desarrollo del sistema, el diseño del entorno del juego y las pruebas de usuario.
- **Desarrollo Iterativo:** Dividir el proyecto en ciclos de trabajo iterativos para incorporar *feedback* y ajustes continuos.
- **Revisión y Ajuste:** Evaluar regularmente el progreso del proyecto a través del tablero KANBAN para resolver problemas y mantener el enfoque en los objetivos establecidos.

Para la ayuda de emplear bien esta metodología se usa una herramienta online llamada **Trello**. Capaz de gestionar una planificación y definición inicial de tareas, un desarrollo iterativo, revisar y reajustar tareas a gusto del usuario, como se puede observar en la Figura 2.3. Trello ofrece diferentes herramientas para llevar este proceso a cabo:

1. **Visualización de todas las tareas:** se usa una determinada herramienta para exponer todas y cada una de las tareas y su estado.

2. **Por hacer:** son todas y cada una de las tareas que aún no se han empezado.
3. **En proceso:** son todas y cada una de las tareas que están en proceso.
4. **Terminadas:** son todas y cada una de las tareas que están finalizadas.
5. **En revisión:** son todas y cada una de las tareas que están cerca de estar finalizadas, pero quedan una serie de problemas por solucionar.
6. **Priorización según importancia y urgencia, y seguimiento del tiempo:** se puede organizar todas las tareas del proyecto con la libertad de priorizarla según la urgencia que se considere, y gestionar su fecha de finalización entre otras cosas como se ve en la Figura 2.3.

Una visión general a la planificación inicial de este proyecto, como también se puede observar en la Figura 2.3 es el siguiente:

1. **Estudio de alternativas.**
2. **Diseño de personajes (NPCs) y diseño de contexto.**
3. **Comunicación básica con la API de OpenAI mediante Postman [9].**
4. **Comunicación entre Unity y la API de OpenAI.**
5. **Perfeccionamiento del modelo (Fine-Tuning).**
6. **Recogida de datos (Fine-Tuning) y Resultados.**

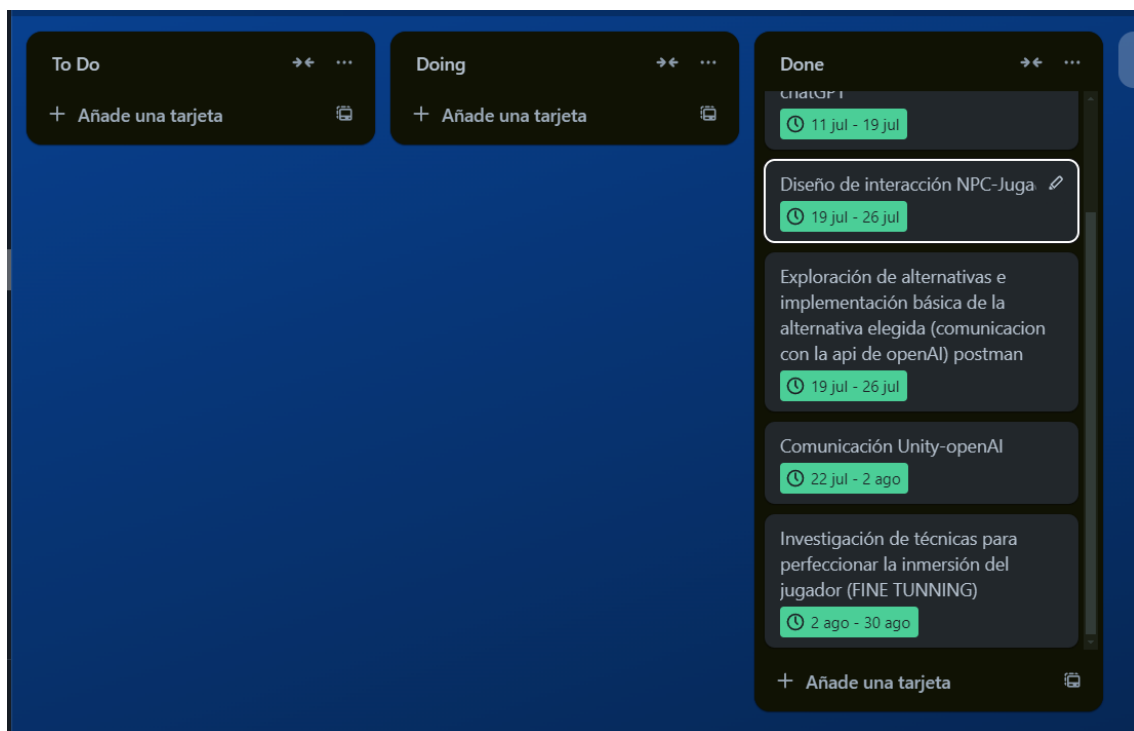


Figura 2.3: Tablero Trello con las tareas finalizadas

3

Desarrollo e implementación

Una vez se tienen claros los diferentes objetivos planteados para el perfeccionamiento de este proyecto, se procede a su respectivo desarrollo e implementación.

En este capítulo se va a explicar los diferentes pasos que se tienen que llevar a cabo para que se de un desarrollo e implementación correcta que permitan alcanzar el objetivo final, que en este caso se dan los siguientes apartados:

1. **Diseño de personajes:** antes de pasar a un desarrollo técnico, si lo que se busca es el desarrollo e implementación de un sistema conversacional de NPCs, que aporten al jugador una rica experiencia, es necesario considerar y diseñar una serie de características esenciales que proporcionen a las entidades NPC, una personalidad y rasgos únicos que enriquezcan la experiencia del jugador.
2. **Comunicación básica con la API de OpenAI:** una vez conseguido el diseño de personajes propio, se puede pasar a la parte de desarrollo algo más técnico. Lo básico es tratar de conseguir una comunicación simple de peticiones HTTP con ChatGPT. Esto se puede conseguir de diferentes formas y herramientas, pero en este caso se decide empezar con una comunicación básica con la API de OpenAI, a través de Postman [9], por su fácil e intuitivo uso, y previos conocimientos ya adquiridos de esta útil herramienta para realizar pruebas de APIs y peticiones HTTP de manera sencilla e intuitiva.
3. **Comunicación entre Unity y la API de OpenAI:** una vez completada con éxito la primera toma de contacto de comunicación con la API de

OpenAI, es necesario profundizar más para llegar al objetivo, adentrarse en Unity, sus diferentes usos, librerías, gestión de peticiones HTTP, etc...

Se profundiza de esta manera para conseguir una comunicación fluida, visualmente sencilla y rápida para el jugador, de esta manera se está aún más cerca de ese perfeccionamiento, mejorando la experiencia del jugador.

4. **Perfeccionamiento del modelo (Fine-Tuning)**: por último, aunque aparentemente se puede dar por finalizado el perfeccionamiento de este proyecto, se decide dar un paso más debido que aunque se haya conseguido una comunicación rápida, eficiente y visualmente sencilla entre Unity y ChatGPT, el modelo es ineficiente. Es necesario dar mucho contexto a ChatGPT para que entienda que se buscan una serie de respuestas que vayan acorde con una determinada narrativa de un videojuego.

Para ello es oportuno usar *Fine-Tuning* [10]. Que en definitiva es uno o varios modelos de generación de texto de OpenAI que han sido preentrenados con una gran cantidad de ejemplos de texto. A veces se incluyen instrucciones y varios ejemplos en una indicación. Más adelante se entrará en detalle del proceso de como se lleva a cabo este preentrenamiento.

OpenAI tiene varios modelos con diferentes características y ventajas en cada uno de ellos, para el desarrollo y proceso de todo este proyecto se usa *GPT-3.5-turbo*, por diferentes razones que irán desglosando poco a poco.

3.1. Diseño de personajes

Como se explica en apartados previos, el primer paso en el desarrollo del perfeccionamiento de este proyecto reside en el diseño, concretamente en el de los diferentes personajes que se van a utilizar para la generación del sistema conversacional propio.

Este diseño parte del desglose de un personaje o NPC, en una determinada personalidad y una serie de características únicas para él.

Dicho de otra manera, diferentes NPCs pueden poseer el mismo tipo de personalidad pero cada uno tiene una serie de características únicas que marcan la diferencia entre todos y cada uno de los NPCs como se puede observar en la Figura 3.1.

En ella se muestra un breve esquema del modelo de datos, que recoge lo previamente aclarado, cada NPC solo puede tener una única personalidad, pero puede tener *N finitas* características (actualmente para el diseño elegido se consideran

suficiente 6 en total, pero si en el futuro se desea se podrían añadir más).

Diseño del modelo de datos del NPC

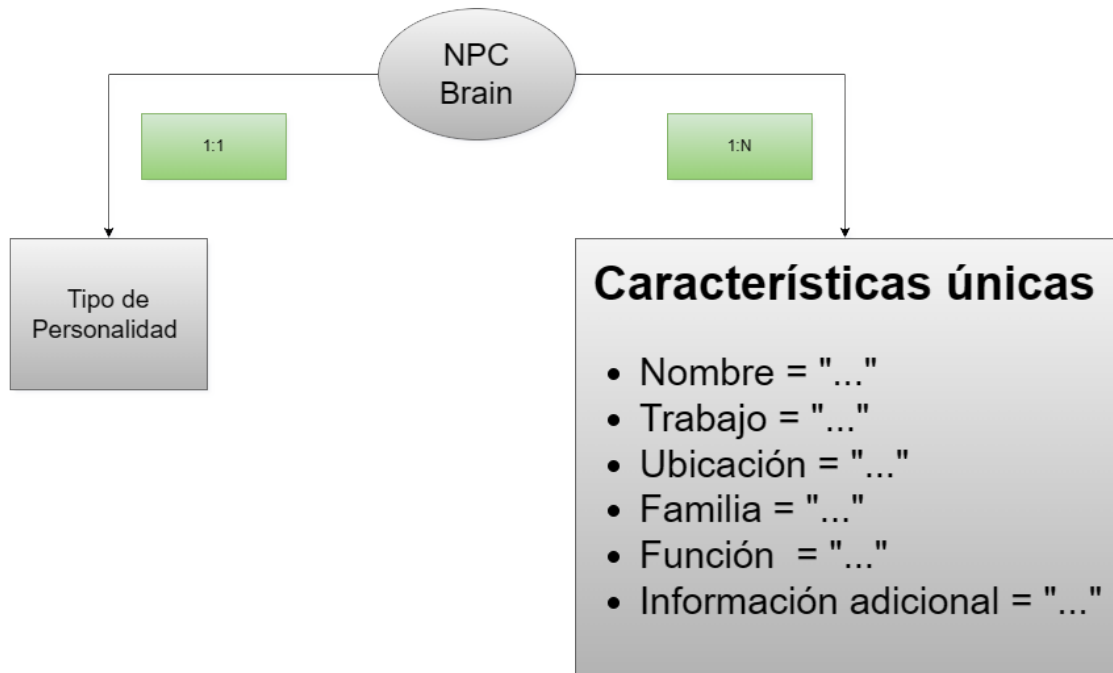


Figura 3.1: Modelo de datos del NPC

El desglose de este esquema, es el siguiente:

1. **Personalidad amable:** caracterizada por respuestas y comportamientos dulces hacia el jugador.
2. **Personalidad antipática:** caracterizada por respuestas y comportamientos desagradables hacia el jugador.
3. **Personalidad estándar:** caracterizada por respuestas y comportamientos estrictamente formales y típicos hacia el jugador.

Las características únicas asociadas a cada NPC de forma individual son:

- Nombre.
- Trabajo.

- Ubicación.
- Familia.
- Función con el jugador.
- Información adicional (sueños, hobbies, etc...).

Una vez se tiene el diseño de personaje necesarios para poder aportar una interacción enriquecedora al jugador, la implementación de este modelo de datos en Unity se puede observar en la Figura 3.2. Además de las características explica-

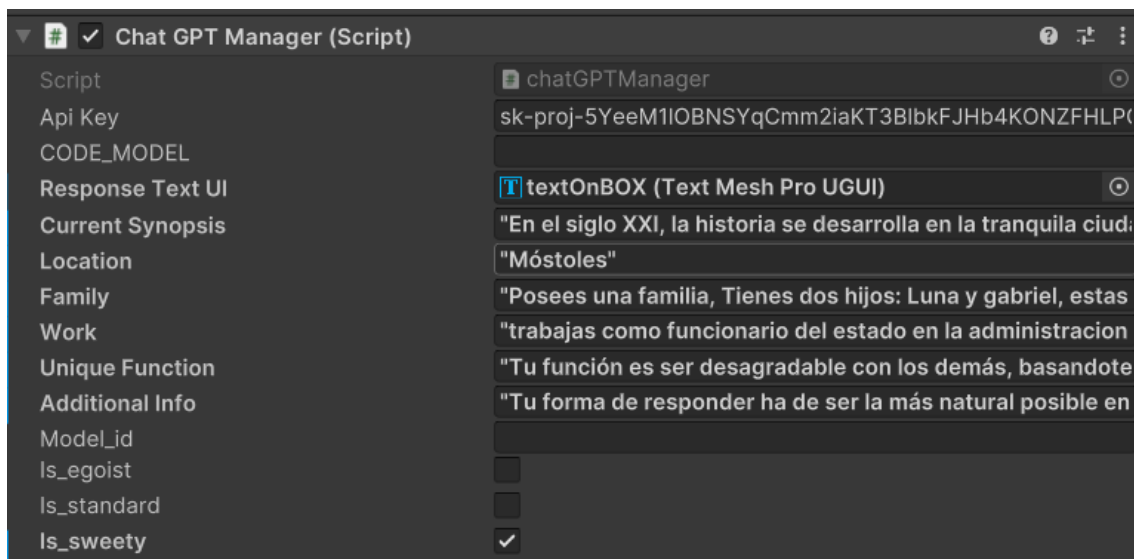


Figura 3.2: Implementación en Unity del modelo de datos del NPC

das, en la Figura 3.2 aparecen una serie de variables necesarias que se explicarán más adelante.

3.2. Comunicación con la API de OpenAI

Tras el diseño de personajes se pasa a realizar las primeras peticiones HTTP a la API de OpenAI, para este cometido se va a usar *Post-man* [9].

Para poder realizar este proceso de forma satisfactoria es necesario trazar una serie de pasos a seguir que se decide gracias a la consulta de la documentación de la API de OpenAI [11]. Esta serie de pasos son:

1. Elegir el modelo de gpt que se va a usar y gestionar el número de tokens.

2. Instalar y configurar Post-man adecuadamente para poder realizar peticiones HTTP.

3.2.1. Modelo GPT, uso de tokens y creación de una API KEY

Antes de tratar de analizar como enviar peticiones HTTP a la API de OpenAI mediante Post-man, hay una serie de detalles que hay que tener en cuenta, esto se puede saber gracias a la documentación de la API de OpenAI [11].

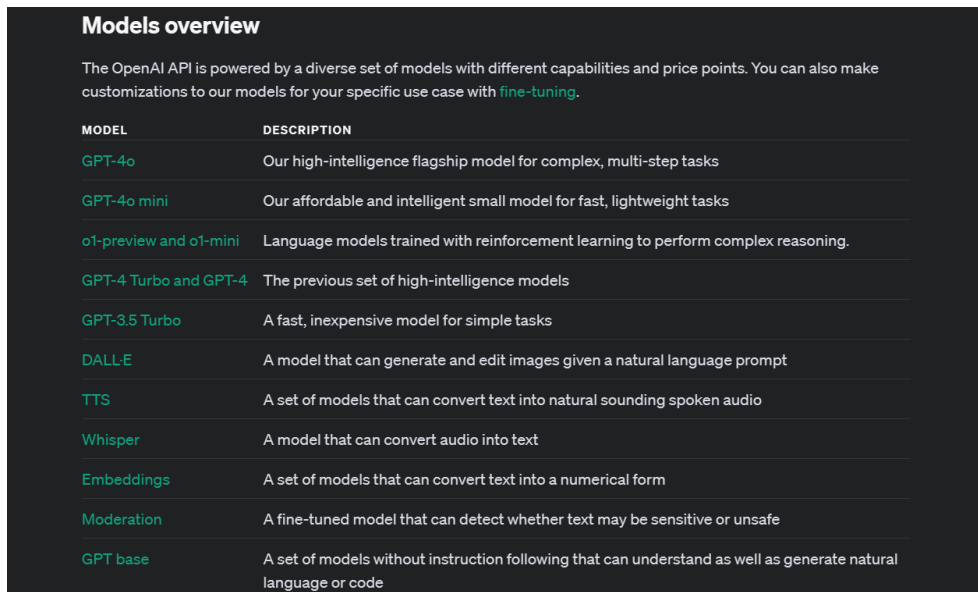
Como se explica más adelante es necesario tener una serie de parámetros en regla para poder realizar peticiones HTTP a la API de OpenAI mediante **Post-man**. Los que hay que tener especial cuidado a la hora de definirlos son los siguientes (el resto se explican a lo largo del documento):

1. **Model:** se trata de la versión del modelo gpt usado al que se realiza la petición. Existen varios tipos de modelos con una serie de características diferentes, las varias versiones se pueden ver en la Figura 3.3.

Estas versiones poseen una serie de ventajas y desventajas que se han de tener en cuenta para poder elegir el modelo adecuado con el que se quiere desarrollar este proyecto. Para ello se han de definir una serie de requisitos y/o factores que se buscan en el modelo y elegir uno en base a ellos. Estos requisitos que se han definido son los siguientes:

- a) **Factor económico:** realizar peticiones HTTP a la API de OpenAI, sea el modelo que sea tiene un coste económico. El coste de estas depende del valor de los tokens de cada modelo, que es distinto para cada uno de ellos. Al ser un videojuego desarrollado por una única persona cuyo objetivo es sin ánimo de lucro, este factor es determinante.
- b) **Factor de complejidad de cómputo:** por otro lado, el cómputo es crucial. La rapidez de estas peticiones, la complejidad o sencillez de las respuestas de cada modelo, la fluidez en la interacción **usuario-asistente**, etc...
- c) **Factor de eficiencia** Por último la eficiencia, una mezcla de los anteriores apartados, en este tipo de proyecto es crucial obtener el mejor resultado posible usando la menor cantidad de recursos. Se busca alcanzar el mejor rendimiento de cómputo posible, usando el menor uso de recursos económicos.

Tras tener en cuenta esta serie de factores se llega a una serie de conclusiones:



Models overview

The OpenAI API is powered by a diverse set of models with different capabilities and price points. You can also make customizations to our models for your specific use case with [fine-tuning](#).

MODEL	DESCRIPTION
GPT-4o	Our high-intelligence flagship model for complex, multi-step tasks
GPT-4o mini	Our affordable and intelligent small model for fast, lightweight tasks
o1-preview and o1-mini	Language models trained with reinforcement learning to perform complex reasoning.
GPT-4 Turbo and GPT-4	The previous set of high-intelligence models
GPT-3.5 Turbo	A fast, inexpensive model for simple tasks
DALL·E	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
GPT base	A set of models without instruction following that can understand as well as generate natural language or code

Figura 3.3: Modelos GPT

- **Reducción del coste:** reducir el coste económico al mínimo posible es muy importante, esto se debe a que para el correcto funcionamiento del proyecto es necesario realizar cientos de pruebas de forma previa (el coste económico es por tokens usados en cada petición, esto se observa en la propia documentación de OpenAI [12]). Para comprobar que el resultado es el verdaderamente buscado es necesario hacer todas las pruebas posibles tanto a lo largo del desarrollo del proyecto, como posterior al mismo. Por esa misma razón elegir un modelo que cada petición signifique un coste económico muy elevado, sería inalcanzable.
- **Cómputo necesario:** el nivel de cómputo o complejidad de cómputo es algo que se tiene que medir en base al tipo de videojuego desarrollado, como se explica anteriormente, este tipo de peticiones se realizan para poder desarrollar una serie de conversaciones **jugador-NPC** fluidas y creíbles para el jugador, para que se alcance **credibilidad, realismo e inmersión**. Para conseguir este objetivo no es necesario usar el modelo más avanzado, como son algunos GPT 4 que proporcionan ventajas como el *multi-step* como se ve reflejado en la Figura 3.3. Y para este videojuego no es necesario tal rendimiento. Lo más importante valorado en una serie de conversaciones fluidas y creíbles es la rapidez de las mismas.

Por eso mismo y tras esta serie de conclusiones, el modelo elegido es **GPT-3.5 Turbo**, que destaca por su rapidez y más económico que otros modelos más modernos.

2. **Authorization:** a la hora de realizar peticiones HTTP a la API de OpenAI,

es necesario tener una autorización de forma permanente al modelo usado. Cada petición tiene un determinado coste como se ha visto, por lo tanto, es necesario mantener la confidencialidad, integridad y disponibilidad en estas llamadas.

Esto se consigue con una API-KEY. Se trata de un determinado token que permite al usuario autorizar las llamadas asegurando la confidencialidad e integridad de las llamadas, al haber sido creada y mantenida por el propio usuario. Y por otro lado la disponibilidad (no caduca), el usuario siempre tendrá acceso a esta API KEY.

Esta API KEY se crea accediendo a la documentación de la API de OpenAI de forma bastante sencilla [13].

3. **Tokens:** Por último, los tokens se pueden asignar en un determinado número en el cuerpo de la petición HTTP, esto lo que hace es limitar los recursos de tal manera que no se desperdicien tokens en cada llamada, el límite en este caso puede ser poco, ya que en un NPC en este tipo de videojuegos suele tener respuestas cortas.

Cuanto más cortas y simples sean las respuestas del asistente de OpenAI, menos tokens son necesarios usar.

Existen más parámetros útiles que se pueden usar en la API de OpenAI, a lo largo del desarrollo de este proyecto se verán ejemplos de uso de esta serie de parámetros.

3.2.2. Instalación y configuración de Postman

Para instalar *Post-man* es tan sencillo como ir a la página oficial [9] descargar la aplicación (o usar la versión web) y registrarse con una cuenta. Una vez completado esto, se tiene acceso a la interfaz de Post-man, esta se puede ver en la Figura 3.4. La interfaz de Post-man es amplia y diversa, pero las partes esenciales para el tipo de peticiones que se quieren realizar son las siguientes:

- **Colecciones (1):** en las colecciones se agrupan en un solo paquete o proyecto un conjunto organizado de solicitudes (requests). Estas permiten, ejecutar y compartir las pruebas y solicitudes API de manera eficiente.
- **Tipo de peticiones (2):** Aquí asignamos el tipo de solicitud que se va a realizar, como ya se ha explicado, este proyecto se centra en solicitudes HTTPs. Pero Post-man permite diferentes tipos, como se ve reflejado en la Figura 3.5.

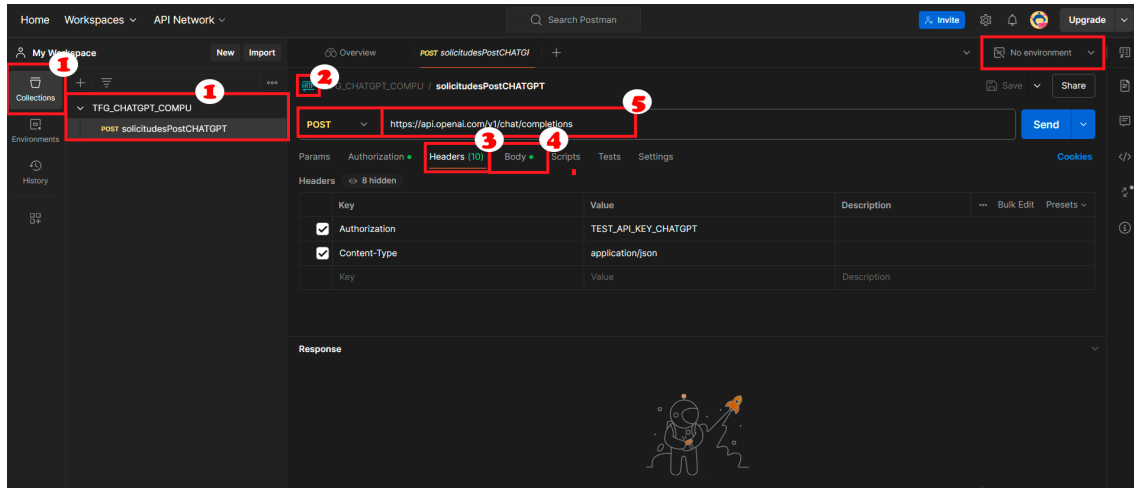


Figura 3.4: Interfaz de Post-man con sus características resaltadas

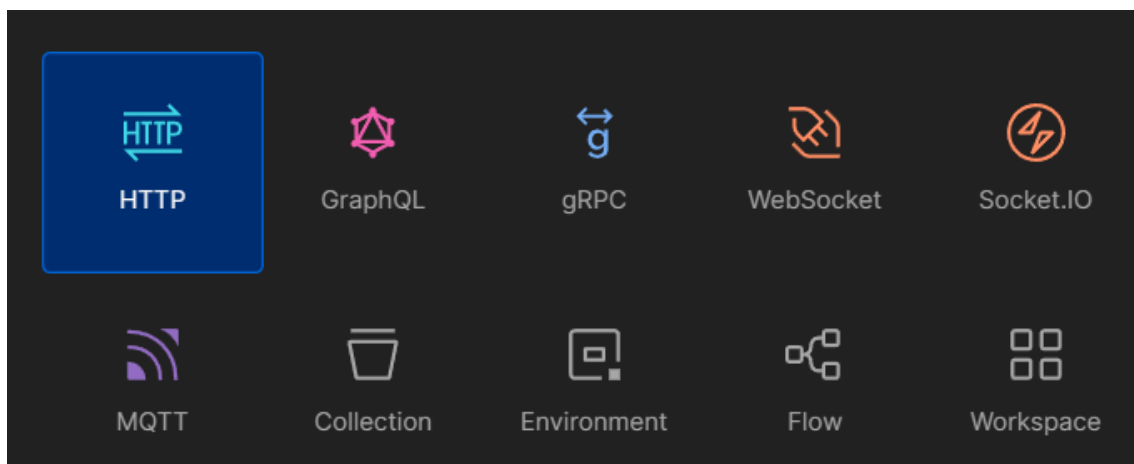


Figura 3.5: Tipos de solicitudes en Post-man

- Cabecera de la petición (3):** es una parte del mensaje que contiene información adicional sobre la solicitud o la respuesta. Los headers se utilizan para definir metadatos sobre la petición, en el caso de la solicitud HTTP para comunicarse con la API de OpenAI son solamente necesarios los siguientes parámetros:
 - Content-Type:** el formato del mensaje que se va a lanzar y la respuesta, en este caso JSON.
 - Authorization:** se manda una API KEY como método de autorización, esta API KEY es necesaria definirla previamente, en la documentación de OpenAI [11] se puede ver de que manera. En la Figura 3.6 se puede ver un ejemplo de esta cabecera.

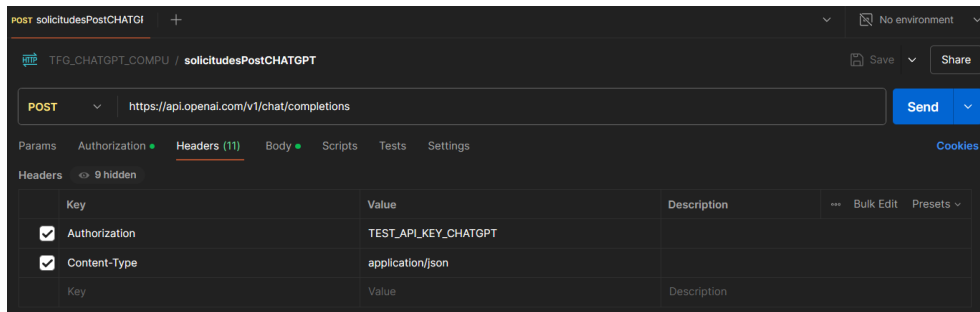


Figura 3.6: Cabecera en Post-man

- Cuerpo de la petición (4):** es la parte del mensaje donde se envían los datos asociados a la solicitud HTTP, en este caso la solicitud se hace a la API de OpenAI. En estas solicitudes hay una serie de datos asociados obligatorios y otros opcionales. Los datos mínimos que se tienen que enviar para el correcto funcionamiento de la petición, según la propia documentación de OpenAI [11], son los siguientes:
 - Model:** se trata de la versión del modelo gpt al que se realiza la petición.
 - Messages:** un array con el contenido del mensaje que se va a enviar.
 - Role:** el rol del usuario que va a mandar el mensaje.
 - Content:** el contenido del mensaje.

Para el perfeccionamiento de nuestro modelo, el body del mensaje debe de ser más amplio, esto se explica más adelante. Un ejemplo del cuerpo de una petición HTTP sencilla para comunicarse con la API de OpenAI se puede ver en la Figura 3.7.

- Caja de la petición (5):** se vuelca la dirección web a la que vamos a lanzar la solicitud.

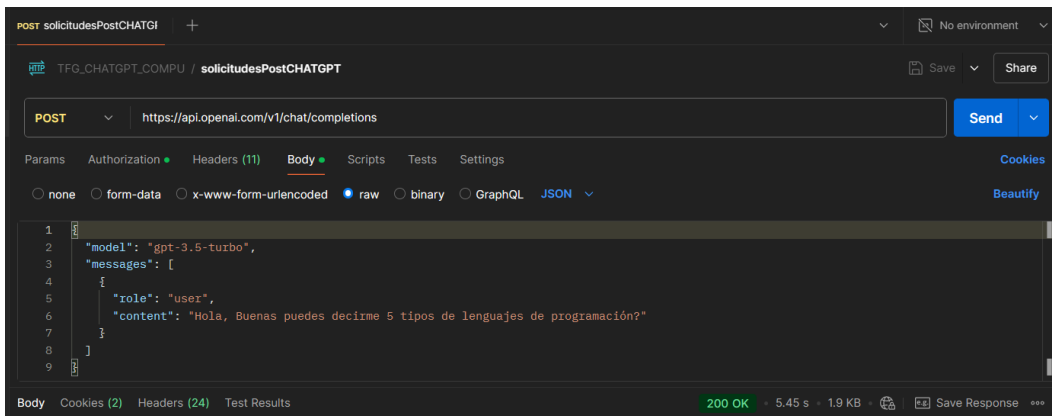


Figura 3.7: Body de la solicitud HTTP

Una vez se tiene bien definida todas y cada una de las partes de este tipo de peticiones HTTP se puede enviar una petición. Para esto, se recibe una determinada respuesta por parte de OpenAI, un ejemplo de la respuesta se puede ver reflejado en la Figura 3.8.

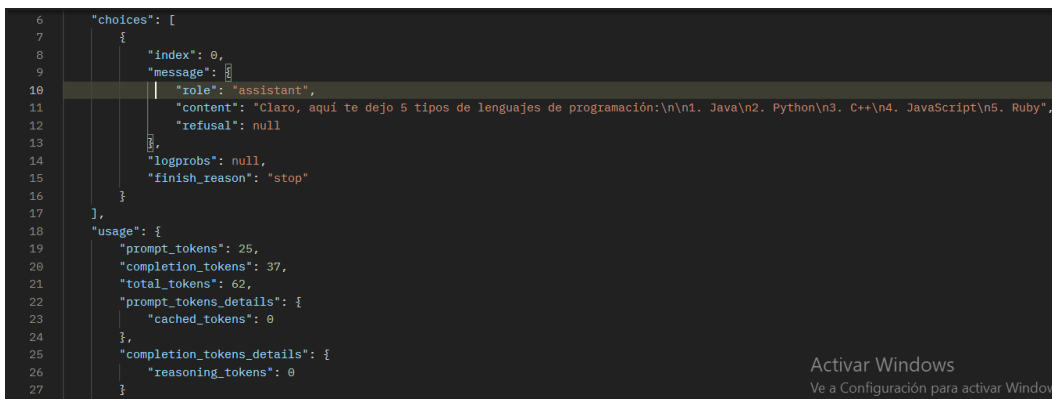


Figura 3.8: Response de la solicitud HTTP

En dicha Figura 3.8 se ve reflejada bastante información, pero la relevante para este proyecto, es la siguiente:

1. **Model:** al igual que en el cuerpo de la petición se usa un parámetro Model, OpenAI devuelve el modelo de gpt usado.
2. **Role:** se trata del rol que tiene el receptor de la petición HTTP previamente realizada, en este caso en vez de *user*, es el *assistant* de OpenAI.
3. **Content:** se trata de la respuesta por parte de ChatGPT a la petición previamente realizada.

4. **Tokens:** en este caso hay varios parametros en el que se habla de tokens, pero el realmente significativo es el parámetro de *total-tokens*, que indica el número de tokens que han tenido que ser usados en esta petición de HTTP.

Con toda esta información que es proporcionada, se tiene lo necesario para proceder a la comunicación del motor de videojuegos Unity con la API de OpenAI.

3.3. Comunicación Unity-OpenAI

Lo probado hasta ahora es la confirmación de que la comunicación con la API de OpenAI es eficaz, rápida y suficiente para el objetivo que se espera, pero es solo el comienzo del desarrollo técnico. La integración de forma correcta en el videojuego mediante Unity es el paso más importante porque hay que diseñar de forma rigurosa lo necesario para alcanzar los objetivos propuestos.

En definitiva, los pasos a seguir para completar de forma correcta el objetivo son los siguientes:

1. **Diseño de interfaz de usuario:** es necesario hacer un breve desglose de la interfaz de usuario necesaria para que el jugador reciba un buen feedback visual, ya que es muy importante que el jugador interprete de forma correcta lo que está viendo.
2. **Diseño del modelo de datos:** declarar un breve modelo de datos muy similar al ya visto anteriormente en la Figura 3.1, pero con algunos añadidos que aparecen en la Figura 3.2 que se van a explicar a lo largo de esta subsección
3. **Investigación de la librería para hacer peticiones Web de Unity [2]:** para realizar peticiones HTTP es necesario investigar y usar la pertinente librería que es proporcionada por Unity.
4. **Desarrollo e implemenación técnica:** en este caso se da un breve desglose de los scripts que hay que desarrollar y como se comunican entre ellos, es crucial para poder entender el funcionamiento.

3.3.1. Diseño de interfaz de usuario

Para diseñar la interfaz de usuario de esta parte del videojuego (**la interacción con un determinado NPC**) se ha de tener en cuenta que se necesita,

como toda interfaz gráfica de usuario (**Graphic Use Interface, GUI**), se necesita dejar claro al jugador toda la información visual necesaria para jugar de forma cómoda continuamente.

Para alcanzar este objetivo se toma como referencias interfaces gráficas de famosos ejemplos en el género **JRPG**, como es el caso de *Undertale* [14], *Earth-Bound* [15] o *Pokemon* [16]. Los elementos claves de feedback audiovisual al jugador son los siguientes:

1. Información previa a la interacción.
2. Información durante la interacción.
3. Información posterior a la interacción.

Cuando se analizan los resultados en el Capítulo 4 se hace más hincapié a que información se refiere.

3.3.2. Diseño del modelo de datos

Una vez se tiene claro el diseño de la interfaz gráfica, se puede pasar a diseñar el modelo de datos que es necesario para poder enviar peticiones HTTP a la API de OpenAI. Por modelo de datos se refiere a poder hacer un riguroso diseño de toda aquella información y/o datos necesario que la API de OpenAI necesita del contexto del videojuego, para poder dar una respuesta adecuada.

Este diseño de datos se ve reflejado en varias figuras previamente mencionadas 3.1 o 3.2, pero es necesario especificar todos y cada uno de los datos de forma más concreta. Un desglose de estos se puede ver en la Figura 3.9. El detalle de este esquema para el correcto funcionamiento de cada petición es el siguiente:

1. **Data Auth:** los datos de autorización son los más importantes, porque si el jugador no posee una determinada API-KEY es imposible la comunicación. Esta API-KEY es previamente creada con la ayuda de la documentación de OpenAI [11] como se menciona anteriormente.
2. **Data Game:** los datos del contexto del videojuego son importantes para proporcionar al asistente de OpenAI una situación y forma de hablar específicas mediante variables de datos declaradas en un script de OpenAI, en este caso ChatGPT contesta de una forma poco creíble y artificial, evitando llegar al jugador a ese punto de **inmersión, credibilidad y realismo** que se persigue. Estas variables son las mismas que aparecen en la Figura 3.1.
3. **Data Type:** este dato es el tipo de formato del mensaje que se envía a OpenAI, en este caso se hace mediante JSON.

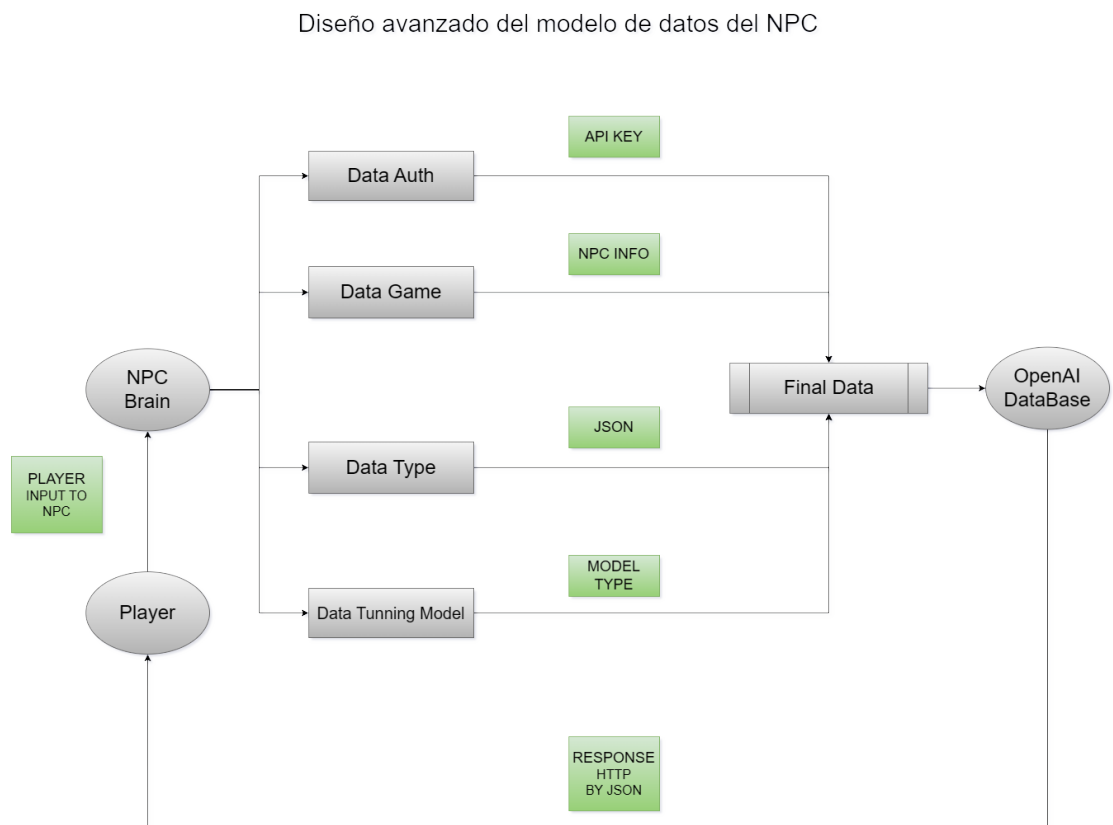


Figura 3.9: Diseño avanzado del modelo de datos

4. **Data Tuning Model:** el tuneado del modelo es un aspecto que perfecciona el diseño del modelo GPT, más adelante se habla de él.

Esta serie de datos se ve declara de la siguiente manera en Unity, como se refleja en la Figura 3.10 (La api-key que aparece actualmente no está activa y por eso se puede mostrar).

```
Script de Unity (2 referencias de recurso) | 2 referencias
public class ChatGPTManager : MonoBehaviour
{
    //Data Auth
    public string apiKey = "sk-proj-5YeeM1l0BNSYqCmm2iaKT3B1bkFJHb4KONZFHLPGcCauACoY";
    //Data Tuning Model
    public static string CODE_MODEL_EGOIST = "ft:gpt-3.5-turbo-0125:personal:npc-selfish:9wSuhGgI";
    public static string CODE_MODEL_SWEETY = "ft:gpt-3.5-turbo-0125:personal:npc-sweety:9yMe4qrI";
    public static string CODE_MODEL_STANDARD = "ft:gpt-3.5-turbo-0125";
    public bool is_egoist;
    public bool is_standard;
    public bool is_sweety;
    public string CODE_MODEL;
    //Data Game
    public TextMeshProUGUI responseTextUI;
    public static ChatGPTManager instance;
    public string currentSynopsis;
    public string location;
    public string family;
    public string work;
    public string uniqueFunction;
    public string additionalInfo;
    public string model_id;
}
```

Figura 3.10: Declaración del diseño avanzado del modelo de datos en Unity

3.3.3. Peticiones Web desde Unity

Una vez finalizada la parte de diseño, es hora de comenzar a programar en Unity el desarrollo técnico para poder realizar peticiones HTTP.

Para poder realizar estas peticiones se usa una librería específica llamada *Networking* [2] capaz de realizar diferentes tipos de peticiones web. En el código desarrollado lo realmente importante es ser capaz de manejar JSON, y enviar peticiones web, esto último se consigue con la librería ya mencionada, la principal herramienta y/o clase que se usa para esto es *UnityWebRequest* que permite lo siguiente:

1. **Configuración:** existen una serie de herramientas que se encargan de preparar la petición HTTP antes de enviar la información. Entre otras cosas lo más importante necesario de configurar es el campo de *Authorization* y

Content-Type. Esta librería tiene métodos capaces de configurar los encabezados de la llamada. Antes de entrar en dicha configuración es necesario declarar un objeto de la clase de *UnityWebRequest* [2] para usar estas herramientas de configuración:

Código 3.1: Unity Web Request

```
UnityWebRequest request =
new UnityWebRequest( url , "POST" );
```

el código encargado de realizar la configuración mencionada es el siguiente:

Código 3.2: Unity Web Request 2

```
request . SetRequestHeader
( "Content-Type" , " application/json" );
request . SetRequestHeader
( "Authorization" , "Bearer " + apiKey );
```

2. **Descodificación o procesamiento de datos:** una vez terminada la configuración se procede a realizar una serie de procesamiento de datos, para poder enviar datos en una petición HTTP es necesario hacer dos pasos previos:

- Asignar un Handler (manejador de la subida o POST de la solicitud).
- Asignar un Download (manejador de la respuesta, para asignarla en un buffer de memoria).

En definitiva, objetos encargados de la subida y recepción de datos de la petición HTTP, dicho código es el siguiente:

Código 3.3: Unity Web Request 3

```
request . uploadHandler = new UploadHandlerRaw( postData );
request . downloadHandler = new DownloadHandlerBuffer ();
```

3. **Envío de petición:** ahora se procede al envío de la petición HTTP, como todo está configurado y procesado, la llamada es sencilla:

Código 3.4: Unity Web Request 4

```
request . SendWebRequest ();
```

4. **Recogida de datos y visualización:** por último, una vez enviada la petición HTTP a la API de OpenAI de forma exitosa solo queda recoger la respuesta del asistente de ChatGPT, se recoge mediante la herramienta ya mencionada *DownloadHandler*.

En definitiva, se termina almacenando los datos en un buffer, y posteriormente se asigna a una variable, para poder asignarla a un objeto, para que pueda ser visualizado en pantalla.

Con todo esto queda claro el uso de la librería para realizar peticiones Web, y se puede pasar a terminar y afinar el resto del desarrollo del código.

3.3.4. Desarrollo e implementación técnica

Completado el diseño e investigación de librerías de Unity pertinentes, se pasa al desarrollo e implementación técnica completa del código necesario implementado en Unity, para ello se necesita:

1. **Diseño y comunicación de diferentes scripts:** la elaboración de un diseño de los diferentes sistemas y scripts que se van a llevar cabo para el funcionamiento de dicha implementación en Unity
2. **Desarrollo del funcionamiento del código:** el desarrollo del código previamente mencionado.

Diseño y comunicación de diferentes scripts

Un desglose de la comunicación entre los diferentes scripts se ve reflejado en la Figura 3.11 Como se puede observar en la Figura este diseño se resume en que una entidad principal que es *ChatGPTManager*, es el cerebro del NPC, se comunica directamente con otra entidad llamada *DialogueManager* encargada de gestionar el procesamiento de conversaciones con la ayuda de otros dos scripts externos, más adelante se entra en detalle en el desarrollo y funcionamiento de estas entidades.

Desarrollo del funcionamiento del código

En esta sección se entra en detalle del funcionamiento del diseño previamente explicado:

1. **ChatGPTManager:** este script es el más importante, ya que representa de alguna manera el cerebro del NPC. Se encarga de gestionar toda la información que contiene individualmente cada NPC. Además, envía peticiones HTTP a la API de OpenAI y finalmente recoge los datos que envía el asistente de ChatGPT, y los almacena.

Diseño, desarrollo y comunicación de scripts

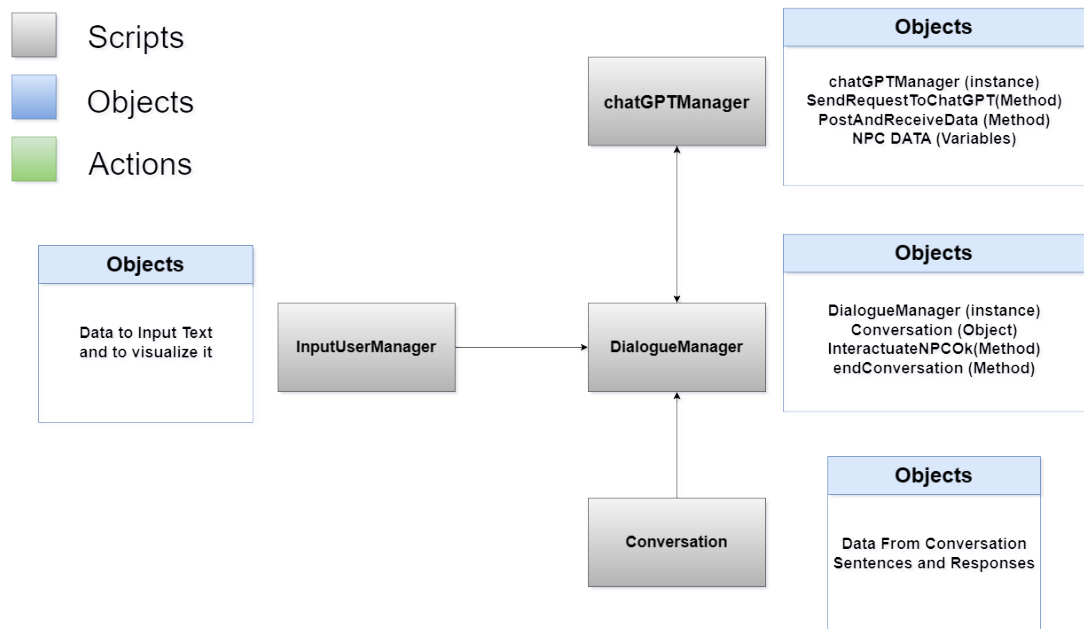


Figura 3.11: Diseño y comunicación de scripts de Unity

2. **DialogueManager**: este script se comunica directamente con el *ChatGPT-Manager*. Una vez este recoge los datos que da como respuesta ChatGPT, el DialogueManager usa ciertas herramientas que trae Unity para representar visualmente el texto para el jugador. Esto lo hace con la ayuda de dos scripts adicionales que son *Conversation* e *InputUserManager*
3. **Conversation**: es el script que declara los objetos encargados de recoger las respuestas de ChatGPT, y el id de cada NPC.
4. **InputUserManager**: es el script que declara los objetos encargados de que el jugador pueda escribir por teclado el mensaje que envía a la API de OpenAI.

Una vez en este punto, se tiene en funcionamiento una entidad de NPC que funciona en base a una determinada información mediante peticiones HTTP a ChatGPT.

3.4. Fase de tuneado

Una vez se llega a este punto, surge un nuevo problema. Aunque se ha tratado de modelar lo máximo posible las respuestas de ChatGPT, sigue dando respuestas demasiado artificiales como para ser interpretados por el jugador como respuestas naturales.

Para ello se empieza a investigar técnicas para mejorar la inmersión en las conversaciones con NPCs, con el objetivo de que estas interacciones sean más realistas y atractivas. Lo que realmente se busca es evitar esas respuestas artificiales propias de ChatGPT. Una manera de evitar esto es modelar las respuestas de ChatGPT en base a una serie de requisitos específicos del usuario de forma sencilla, dicho de forma más coloquial, se busca *Tunear* el modelo de ChatGPT a antojo del usuario y/o cliente.

A esto se le denomina *Fine-Tuning*. Se define como un método de entrenamiento en **aprendizaje automático** y **aprendizaje profundo**, que consiste en tomar un modelo preentrenado en una tarea y ajustarlo para que funcione mejor en una tarea más específica.

Dicho de otra manera, el objetivo principal del *fine-tuning* es adaptar los conocimientos generales que el modelo ya ha aprendido a un nuevo conjunto de datos o a una tarea específica.

Esto es precisamente lo que se busca. Pero antes es necesario abordar la problemática de las formas de responder de ChatGPT, y su artificialidad en las

respuestas. Para poder corregir esto se debería desarrollar y elaborar un modelo con una personalidad y maneras de hablar muy marcadas y más naturales. Tratar de abarcar diferentes tipos de NPC en un solo **Fine-Tuning o modelo** es bastante complejo, así por ello se decide hacer 2 Fine-Tuning y usar 3 tipos de modelos diferentes de GPT:

1. Modelo Tuneado (NPC Egoísta):
2. Modelo Tuneado (NPC Amable):
3. Modelo No-Tuneado (NPC-Estándar):

Aquí es donde realmente se tiene en cuenta la información extra declarada en anteriores esquemas (Figura 3.10) y secciones de código que son utilizadas para el uso de estos diferentes modelos (Figura 3.2). Antes de entrar en detalle del funcionamiento de cada modelo por separado es necesario explicar los pasos a seguir para poder desarrollar un modelo hecho mediante **Fine-Tuning**, estos pasos son los siguientes:

1. Investigación de la documentación de OpenAI [11].
2. Diseño y desarrollo del Fine-Tuning mediante Python.
3. Implementación del uso de estos modelos en Unity.

3.4.1. Investigación de la documentación de OpenAI

En primer lugar para poder desarrollar un modelo adecuado mediante **Fine-Tuning**, es necesario investigar como hacerlo. En la documentación de OpenAI se encuentran varias secciones que guían al usuario para realizar ciertos procesos, para el **Fine-Tuning** [17] también hay una guía.

De la lectura y análisis de esta guía se pueden definir los pasos a seguir necesarios para poder crear y usar un modelo Fine-Tuneado:

1. Preparar los datos.
2. Subir los archivos.
3. Usar un trabajo de Fine-tuning.
4. Uso del modelo Fine-Tuneado.

Todos estos pasos se pueden realizar mediante un diseño y desarrollo técnico en Python usando el framework **Pycharm**. Se utiliza porque proporciona al usuario ciertas herramientas y bibliotecas de OpenAI [18] bastantes útiles para este desarrollo.

3.4.2. Diseño y desarrollo del Fine-Tuning mediante Python

Una vez se tiene claro los pasos que hay que seguir para poder elaborar el modelo Fine-Tuneado se entra en detalle en cada uno de estos pasos a seguir:

Preparación de los datos

En primer lugar, es necesario preparar los datos para el modelo. En la documentación previamente mencionada la preparación de datos es muy escasa, ya que solo se dan un par de ejemplos de los mensajes del asistente de GPT. Para el caso que se está desarrollando se necesita ejemplos mucho más enriquecidos, porque es muy importante dar muchos ejemplos para entrenar correctamente al modelo para el tipo de respuestas que tiene que dar el NPC. Estos datos se subirán más adelante en un formato de archivo *.txt*. El formato de ejemplos conversaciones desarrollado es el siguiente:

```
user: ¡Hola, que tal estás!  
assistant: ¡Hola mequetrefe!, ¿que se te ha perdido por el barrio?  
-  
user: ¿Me puede indicar donde estamos?  
assistant: Acabas de llegar y me tratas como si fuera tu mapa personal,  
si que tienes valor chiquilla.
```

En el ejemplo desarrollado se separa cada conversación por un guion para que el modelo entienda cada interacción por separado. Para entrenar el modelo de forma adecuada se usan un total de 50 conversaciones de este estilo para que el modelo profundice y aprenda el tipo de respuestas que ha de dar, y la personalidad que tiene que tener como NPC de un videojuego, y también el tipo de preguntas comunes que va a recibir.

A partir del documento generado, es necesario tener estas conversaciones en un formato JSON. Gracias a un desarrollo de código en Python se transforma las respuestas del asistente a un formato JSON que pueda recibir OpenAI e interpretarlo. Un ejemplo de este formato es el siguiente:

```
{"messages":  
[{"role": "system",  
"content": "Eres un NPC de un videojuego \textbf{\textbf{JRPG}},  
cuyo contexto historico ha de ser hablar de la forma mas natural,  
comun y cercana posible proveniente de un habla comun de Espana  
del siglo XXI en pleno 2024."},  
{"role": "user",  
"content": "\u00c2\u00bfTe gustan los animales?"},
```

```
{
  "role": "assistant",
  "content": "\u00c2\u00a1S\u00c3\u00ad,
  los animales son adorables y
  llenan nuestras vidas de amor y alegr\u00c3\u00ada!"]}]
{
  "messages":
  [
    {
      "role": "system",
      "content": "Eres un NPC de un videojuego \textbf{\textbf{JRPG}},
      cuyo contexto historico ha de ser hablar de la forma mas natural,
      comun y cercana posible proveniente de un habla comun de Espana
      del siglo XXI en pleno 2024."},
    {
      "role": "user",
      "content": "\u00c2\u00bfSabes qui\u00c3\u00a9n soy?"},
    {
      "role": "assistant", "content": "\u00c2\u00a1Claro,
      eres una persona maravillosa
      con un coraz\u00c3\u00b3n lleno de bondad!"]}]
}
```

Con el archivo JSON correspondiente, se finaliza la preparación de los datos necesarios.

Subir los archivos

En este paso ya se comienza a desarrollar código en Python. Una vez se tienen los datos correspondientes para entrar un modelo Fine-Tuning se pasa a subir estos archivos a la Base de Datos de OpenAI para que pueda entrar el modelo. Para subir estos archivos a la Base de datos de OpenAI, es tan sencillo como usar un método de la biblioteca de OpenAI en Python:

Código 3.5: Subida de archivos a OpenAI mediante Python

```
import openai
openai.api_key = '*****'

if __name__ == '__main__':
    response_file = openai.File.create(
        file=open('_finetuning_sweety.json', 'rb'),
        purpose='fine-tune'
    )

    print(f'ID: {response_file.id}')
```

Los datos necesarios para la correcta subida de archivos son: el **JSON** previamente generado; el **propósito** de esta subida de archivo (para fine-tunear); y por último necesitamos la **api-key** correspondiente del modelo **sin Fine-Tuning**. Este método generará una id, que es necesario guardarla para usos posteriores.

Usar un trabajo de Fine-tuning

Ahora se pasa a crear un punto de trabajo de ***Fine-Tuning***. Dicho de otra manera, este es el paso en el que finalmente con los datos preparados y subidos a la Base de Datos de OpenAI se puede crear el modelo tuneado. Para ello se necesita una serie de parámetros:

- Id del archivo generado en el paso anterior.
- La versión de gpt usada.
- Sufijo o nombre del modelo tuneado.
- Número de veces que el modelo va a pasar los datos para entrenar el modelo (Opcional).

El código desarrollado para esto es el siguiente:

Código 3.6: Creación de punto de trabajo openAI mediante Python

```
import openai
FILE_ID_EGOIST = 'file-k0BjedadV0yTl9aw2p7I4lz2L'
FILE_ID_SWEETY = 'file-KQVUtIGaknkX2vKNil0pvv0t'
openai.api_key = '*****'
#En este código es donde realmente se crea el modelo "tuneado"
if __name__ == '__main__':
    response = openai.fine_tuning.jobs.create(
        training_file=FILE_ID_SWEETY,
        model="gpt-3.5-turbo",
        suffix="NPC.SWEETY",
        hyperparameters={'n_epochs':4}
    )
    print(response)
```

Esta llamada genera una determinada url para poder usar el modelo fine-tuneado que se ha creado.

3.4.3. Implementación del modelo en Unity

Una vez se crea el punto de trabajo del que partir, se comienza la integración con Unity. En este punto es donde entra los aspectos diseñados respecto al Fine-Tuning, como en la Figura 3.9 o la Figura 3.10. Estos parámetros que se diseñan y configuran para el uso de modelos tuneados son:

- Urls de los diferentes modelos Fine-Tuneados.

- Variables que asignan a cada NPC el tipo de modelo que se usa.

En definitiva, para la integración del uso de los diferentes modelos en Unity solo hay que modificar el código para que según el tipo de personalidad pre-diseñada se hace la petición HTTP a la API de OpenAI de un modelo u otro.

Como se habla anteriormente, se diseña y crea dos modelos *Tuneados* para dos personalidades distinguidas de NPC para una personalidad más amable, otra más egoísta y finalmente la estándar (que es un modelo no tuneado). Con la integración en Unity de modelos *Fine-Tuning* se termina el perfeccionamiento técnico obteniendo un mejor resultado en las respuestas del asistente [3.4.2](#)

3.5. Código Github

Este breve apartado solamente pretende comunicar que todos los scripts de código desarrollados a lo largo del proyecto están públicos en el siguiente enlace: [chatGPTAISystem](#) más concretamente en la carpeta *chatGP-TAISystem*

4

Resultados

Tras haber completado todo el diseño y desarrollo técnico necesario para la integración en Unity de entidades NPC conversacionales en un videojuego RPG realizando peticiones HTTP a la API de OpenAI se recoge una serie de resultados diferentes entre los tres tipos de modelos que se usan.

Antes de pasar a analizar los resultados, es necesario tener en cuenta que estos términos carecen de sentido sin un contexto determinado, es decir, sin un espacio y/o escenario en el que el jugador pueda observar y analizar su entorno no es posible dar una información visual al jugador ni analizar los resultados del mismo, porque sería incomprensible para él.

Este contexto determinado puede ser un escenario o escena concreta en el que el jugador puede moverse con libertad e interactuar y/u observar los elementos del mismo. Un ejemplo de este escenario en el videojuego se ve reflejado en la Figura 4.1.

En esta Figura se puede observar en el escenario elementos de **co-**

lor rosa a aquellos NPC y/o elementos con los que el jugador puede interactuar. Por otro lado, se ve de **color azul claro** aquellos elementos de la **GUI** que el jugador puede visualizar antes de la interacción.

Con este contexto, se puede explicar la retroalimentación visual para el jugador, en las tres situaciones nombradas anteriormente.



Figura 4.1: Escenario

Información previa a la interacción

Como es esperable, se trata de aquella información visual visible para el jugador antes de que interactúe con el NPC, estos elementos se pueden enumerar en los siguientes:

1. Los NPCs (color rosa).

2. **Los elementos de la GUI (color azul).**
3. **Los demás elementos del escenario (muebles).**
4. **Pop-Up:** este se trata del último elemento clave para que el jugador sepa como interactuar con el NPC, sin esta retroalimentación que da a entender al jugador de forma rápida y sencilla como hacerlo, no sería un buen diseño de interfaz de usuario.

En definitiva, este Pop-Up aparece de forma espontánea cuando el jugador se acerca al radio de interacción del NPC, esto se entiende mejor observando la Figura 4.2, en la cual, se puede visualizar como se aprecia un recuadro que transmite al jugador como empezar a interactuar, y por eso mismo, este elemento explicado (el Pop-Up) es el último antes de empezar con la interacción.



Figura 4.2: Primer Pop Up GUI

Información durante la interacción

Una vez se deja claro la información visual previa a la que el jugador tiene acceso a simple vista, se pasa a aquellos elementos visibles que dan un feedback al jugador para que entienda que la interacción ha comenzado, estos elementos que están reflejados en la Figura 4.3 son:

1. **Nuevo recuadro:** una vez el jugador pulsa la tecla correspondiente para empezar la interacción el elemento más claramente visible es la aparición de un nuevo recuadro de **distinto color**. Esto proporciona al jugador cierta información.

Que el cuadro tenga un distinto color da a entender rápidamente al jugador que algo ha cambiado, que este nuevo recuadro representa algo distinto al anterior.

2. **Aviso de input text en el recuadro:** una vez el jugador puede ver rápidamente la principal diferencia del nuevo recuadro respecto al antiguo, también se fija en un pequeño sombreado que da a entender al jugador que puede escribir en este recuadro lo que quiera, dicho de otra manera, con esta nueva información para el jugador queda completamente claro que puede escribir un mensaje en este recuadro.
3. **Información implícita:** con información implícita se refiere a esa información que no queda claramente indicada, pero que el jugador puede interpretar por experiencias similares que ha vivido.

Por ejemplo, no queda claro como enviar el mensaje escrito al **NPC**, pero como el jugador está acostumbrado a escribir en el teclado y a buscar cosas determinadas cosas (por ejemplo en internet presionando la tecla **ENTER**), el impulso natural del jugador es presionar dicha tecla una vez esté escrito el mensaje.

En la siguiente Figura 4.4 se ve un ejemplo de mensaje por parte



Figura 4.3: Elementos presentes durante la interacción

del jugador para interactuar con el NPC.

Información posterior a la interacción

Una vez el jugador presione la tecla necesaria para enviar un mensaje con sentido, empieza el feedback del NPC o también llamada información posterior a la interacción. Una vez se llegue a este punto la interpretación es extremadamente sencilla para el jugador.

Este pregunta o interacciona con cualquier tipo de mensaje que envía al NPC, y tras esto el NPC responde en base a su **personalidad** previamente definida (en los siguientes puntos se hace hincapié en que significa exactamente esta personalidad). Esta respuesta se puede observar en la siguiente Figura 4.5.

Con todo esto explicado, queda un claro diseño de la interfaz de usuario necesaria para que el jugador pueda interpretar de forma rápida, sencilla y visual todos los elementos que necesita para entender esta parte del videojuego.



Figura 4.4: Ejemplo de mensaje de interacción

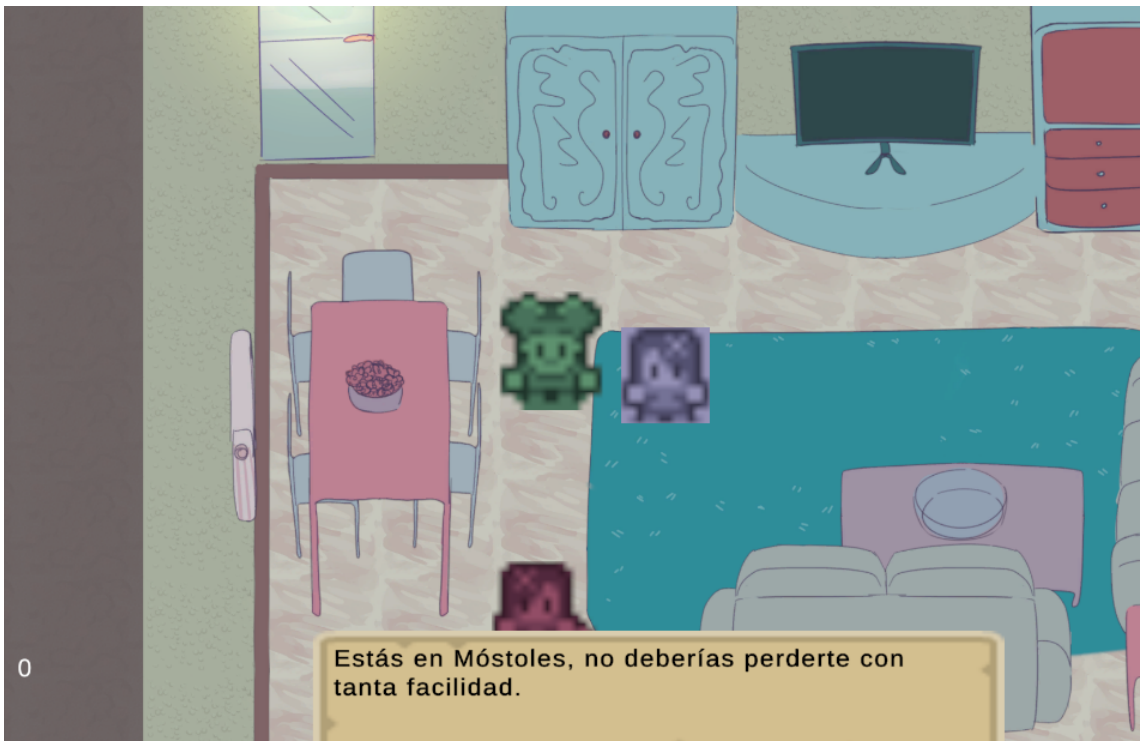


Figura 4.5: Respuesta del NPC

4.1. Análisis de resultados

Una vez claro la información que el jugador puede visualizar durante toda la interacción, a continuación se hace un análisis y comparación de los diferentes resultados de estos modelos, teniendo en cuenta porque se han usado tres y lo realmente útil y necesario que resulta.

4.1.1. Modelo Estándar (Sin Fine-Tuning)

El Modelo Estándar o sin Fine-Tuning es el primero que se logra alcanzar a lo largo de este proyecto. A pesar de que no es un modelo tan completo como los que se entrenan con Fine-Tuning, también es completamente necesario para este proyecto porque proporciona un modelo más común que transmite **menos sentimiento**.

En definitiva este modelo modelo con una personalidad menos marcada, no tiene como resultado una personalidad más dulce o más borde, sino que representa un **término medio**, proporciona un modelo de NPC con los que el jugador solo interactúe con él para conseguir información, y para eso, se necesita un modelo que parta de la imparcialidad. En la Figura 4.6 se puede observar la pregunta realizada y el dato recogido del modelo estándar.

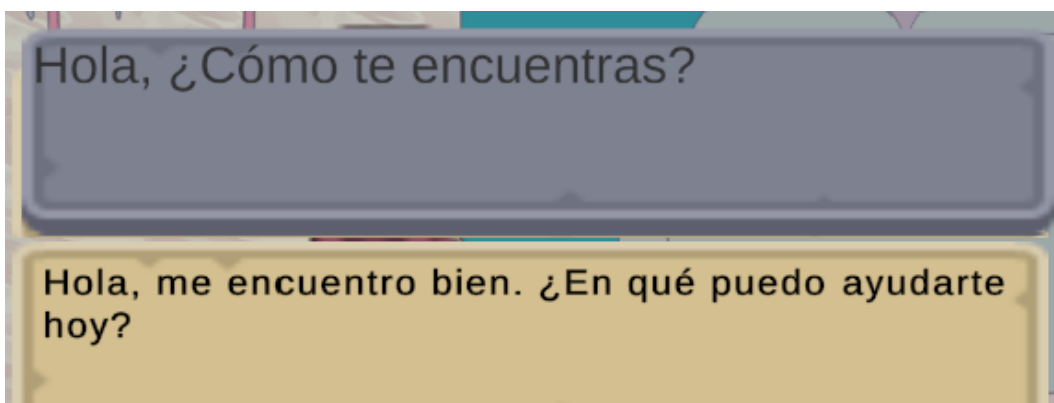


Figura 4.6: Análisis de Resultados: Modelo Estándar

Como se menciona previamente, la respuesta con este tipo de

modelo, es más un término medio y completamente objetiva.

4.1.2. Modelo Amable (Fine-Tuning 1)

El Modelo Amable es el primero que se logra explicar y analizar a lo largo de este proyecto con *Fine-Tuning*. Se trata de un modelo más completo que el anterior entrenado para dar una serie de respuestas más específicas y subjetivas. Dicho de otra manera, también es completamente necesario para este proyecto porque proporciona un modelo con **más sentimiento**. En la Figura 4.7 se da un ejemplo del tipo de respuesta.

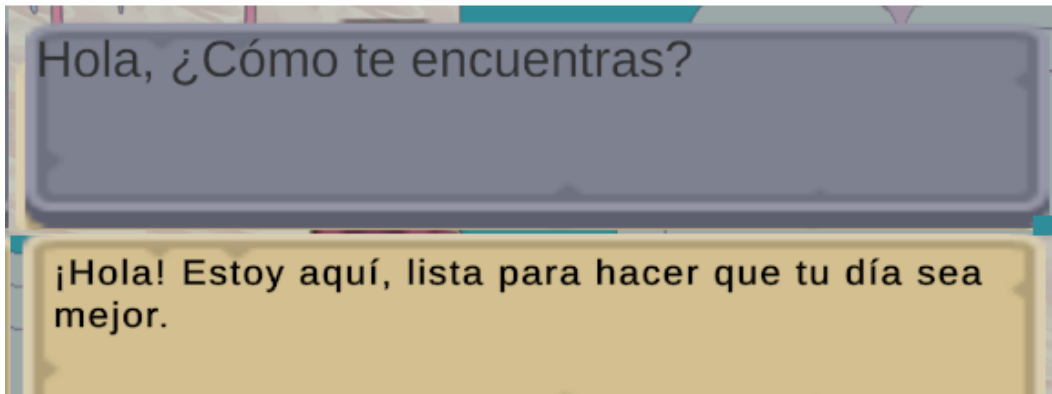


Figura 4.7: Análisis de Resultados: Modelo Amable

Como se observa en la Figura 4.7 esto consigue un tipo de respuesta más amable hacia el jugador, consiguiendo más sentimiento y subjetividad, dando la sensación al jugador de **credibilidad, realismo e inmersión** gracias a la variedad de NPCs, y las múltiples posibles interacciones con diferentes personajes.

4.1.3. Modelo Egoísta (Fine-Tuning 2)

Por último, se va a analizar el modelo egoísta. Es un caso sencillo de exponer porque al fin y al cabo es la otra cara de la moneda respecto al modelo anterior reflejado en la Figura 4.7, pero representando otro tipo de personalidad, una más egoísta y desagradable,

otorgando al jugador la posibilidad a la confrontación. Este último modelo, proporciona una experiencia al jugador enriquecida del todo, existiendo multiples posibilidades de interacción. En la Figura 4.8 se ve un ejemplo de este modelo.

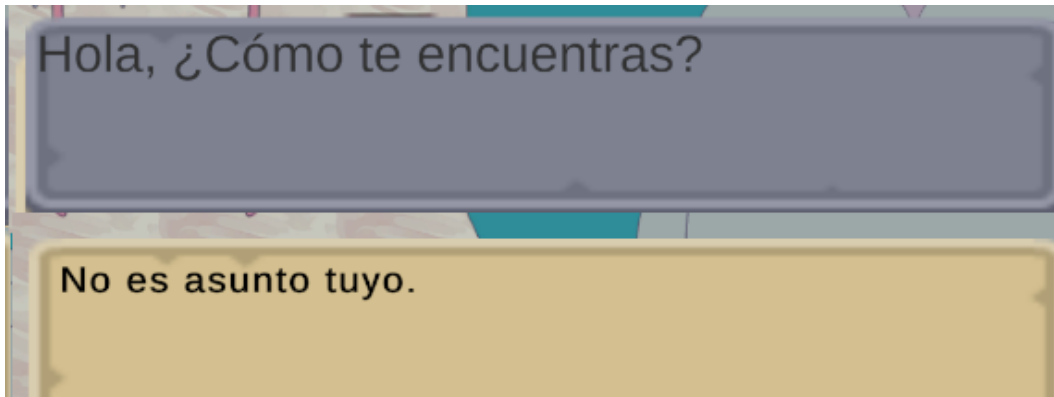


Figura 4.8: Análisis de Resultados: Modelo Egoísta

Al analizar los resultados de estos tres modelos por separado se puede observar la necesidad de multiples personalidades en los NPCs a lo largo del videojuego proporcionando así como un enriquecimiento en la experiencia del jugador.

5

Conclusiones y trabajos futuros

Acerca de las posibles reflexiones que se van a plantear en la Sección 5.1 y en la Sección 5.2, se hace hincapié en como se consigue implementar de forma exitosa los objetivos de este TFG, mediante la implementación de interacciones realistas y fluidas con personajes en el videojuego. Además se consideran nuevas herramientas y tecnologías para trabajos futuros.

5.1. Conclusiones

Se consigue el desarrollo de una entidad NPC moldeable a los intereses y gustos del usuario a través de la comunicación con la API de OpenAI, y el correspondiente *Fine-Tuneado* del modelo. Se consiguen variaciones variopintas de diferentes NPC con múltiples personalidades, y por lo tanto, con varios tipos de interacciones que enriquecen la experiencia del jugador.

Con este perfeccionamiento de la **credibilidad, realismo e inmersión** del jugador se abarca los diferentes contextos con los que

el jugador interactúa mayor tiempo a lo largo de todo el videojuego RPG:

- Narrativa.
- Combate.

En conclusión se consigue alcanzar el objetivo con éxito, al introducir este tipo de interacciones fluidas y realistas con los NPCs, cubre todas las necesidades del jugador junto a la previa inmersión conseguida con el desarrollo del Sistema de Utilidad en el combate.

Teniendo esto en cuenta, el gameplay del combate es creíble realista e inmersivo, y con este perfeccionamiento también lo es la interacción del jugador con el resto del entorno.

5.2. Trabajos futuros

Tras conseguir el objetivo esperado, se tienen una serie de aspectos u objetivos futuros que son útiles para la continuación del desarrollo de este proyecto, esto son:

1. **Apartado técnico:** se consigue alcanzar un perfeccionamiento técnico adecuado para este proyecto abarcando todos los aspectos importantes de un videojuego RPG. Sin embargo, se ofrece la posibilidad de seguir perfeccionando el apartado técnico mejorando a la vez el diseño del combate del juego, una mayor eficiencia técnica es posible con nuevos diseños de combates:
 - Combates dobles.
 - Añadir consumo de objetos en el enemigo.
 - Nuevos tipos de personalidades en enemigos.

Añadir nuevos tipos de diseños de combate, ofrece la posibilidad para mejorar en trabajos futuros.

2. **Apartado artístico:** aún teniendo en cuenta que el diseño de juego y visual de este videojuego ha significado una gran carga de trabajo se dedice priorizar la parte técnica para mejorar el máximo posible el apartado técnico de este videojuego, ofreciendo combates e interacciones completamente enriquecedoras para el jugador. Aun que no se ha ignorado el apartado artístico es mejorable invirtiendo mayor tiempo y dedicación:

- Mejora de la usabilidad.
- Añadir animaciones.
- Perfeccionamiento del diseño de escenarios y personajes.
- Perfeccionamiento del diseño de la paleta de color.

3. **Fine-Tuning, otro punto de vista:** aprovechando la investigación realizada sobre el **Fine-Tuning** y su efectividad para moldear un modelo de ChatGPT, se piensa para trabajos futuros diversas posibilidades de uso. Estas posibilidades son:

- **Nuevos NPCs:** se puede seguir dando uso a la elaboración de nuevos modelos Fine-Tuneados con el objetivo de definir nuevas personalidades para los NPCs de este videojuego.
- **Diferente modelo de ChatGPT:** para este videojuego se decide usar **GPT-3.5**. Sin embargo OpenAI ofrece multiples modelos con diferentes utilidades diferentes cada una, es interesante plantear NPCs con diferentes modelos de GPT, pero siendo entrenados de la misma manera para poder visualizar de mejor manera la diferencias entre estos modelos.

Bibliografía

- [1] Kenney, “Kenney assets free.” [Online]. Available: <https://kenney.nl/assets>
- [2] Unity, “Unity docs,” 2016. [Online]. Available: <https://docs.unity3d.com/es/530/Manual/UnityWebRequest.html>
- [3] J. Rasmussen, “Are behavior trees a thing of the past?” *Introducing the Next Generation of AI: How Utility AIs Are Replacing Behaviour Trees.*, pp. 279–299, 2016. [Online]. Available: <https://www.gamedeveloper.com/programming/are-behavior-trees-a-thing-of-the-past->
- [4] I. MILLINGTON and J. FUNGE, *AI For Games, Third Edition*. CRC Press; 3rd edición, 2019, vol. 3.
- [5] D. Vargas, “Cómo crear una página web con chatgpt: utiliza ia para crear una web de wordpress desde cero,” 2023. [Online]. Available: <https://www.hostinger.es/tutoriales/crear-una-pagina-web-con-chatgpt>
- [6] T. Nehuén, “Chatgpt - qué es, evolución, orígenes y tipos.” 2023. [Online]. Available: <https://definicion.de/chatgpt/>
- [7] Asana, “Metodología kanban,” 2024. [Online]. Available: <https://asana.com/es/resources/waterfall-agile-kanban-scrum>
- [8] K. B. M. B. A. van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas, *Manifiesto Ágil*, 2002-2016. [Online]. Available: <https://agilemanifesto.org/iso/es/manifesto.html>
- [9] V. Cuervo, “Qué es postman,” 2019. [Online]. Available: <https://arquitectoit.com/postman/que-es-postman/>
- [10] openAI, “Fine-tuning.” [Online]. Available: <https://platform.openai.com/docs/guides/fine-tuning>
- [11] “Openai api docs.” [Online]. Available: <https://platform.openai.com/docs/api-reference/introduction>
- [12] “Openai api docs prices.” [Online]. Available: <https://openai.com/api/pricing/>
- [13] “Openai api-key docs.” [Online]. Available: <https://platform.openai.com/docs/api-reference/authentication>
- [14] T. Fox, “Undertale,” 2015.
- [15] A. H. Laboratory, “Earthbound,” 1989-2022.
- [16] G. Freak, “Pokemon,” 1996-2024.
- [17] openAI, “Fine-tuning,” 2023-2024. [Online]. Available: <https://platform.openai.com/docs/guides/fine-tuning>
- [18] “openai-python,” 2023. [Online]. Available: <https://github.com/openai/openai-python>

