

# Chat'n'Game – El chatbot con juego

**Nombre Estudiante**

E. Augusto Arias S.  
Grado de Ingeniería Informática  
75.629 – TFG – Inteligencia Artificial

**Nombre Consultor/a**

Gabriel Moyà Alcover

**Nombre Profesor/a responsable de la asignatura**

Friman Sanchez Castaño

Fecha Entrega:

16/06/2024



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

## FICHA DEL TRABAJO FINAL

|  |   |
|--|---|
| <b>Título del trabajo:</b>   | <i>Chat'n'Game – El chatbot con juego</i> |
| <b>Nombre del autor:</b>   | E. Augusto Arias S.                       |
| <b>Nombre del consultor/a:</b>   | <i>Gabriel Moyà Alcover</i>               |
| <b>Nombre del PRA:</b>   | <i>Friman Sanchez Castaño</i>             |
| <b>Fecha de entrega (mm/aaaa):</b>   | 06/2024                                   |
| <b>Titulación:::</b>   | <i>Plan de estudios del estudiante</i>    |
| <b>Área del Trabajo Final:</b>   | <i>Inteligencia Artificial</i>            |
| <b>Idioma del trabajo:</b>   | <i>Castellano</i>                         |
| <b>Palabras clave</b>  | <i>LLM, AI, loneliness</i>                |
| <b>Resumen del Trabajo (máximo 250 palabras):</b> <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i>   |   |
| <p>La soledad emergente como un problema global afecta a personas que pueden sentirse solas incluso en compañía, pudiendo derivar en trastornos de conducta y deterioro en la calidad de vida [1]. Este proyecto presenta un chatbot con un minijuego diseñado para simular interacciones humanas y brindar consuelo a personas que experimentan soledad. El objetivo no es reemplazar la compañía humana, sino ofrecer un 'compañero virtual' que combine capacidades de chat con entretenimiento virtual, comenzando con un ejemplo de Tic Tac Toe y con potencial para cambiarlo por otros juegos en el futuro.</p> <p>El enfoque principal es la interacción asincrónica, permitiendo que las conversaciones se conviertan en monólogos si el usuario se dedica a otras tareas, como el minijuego. Los usuarios pueden cambiar entre tres Grandes Modelos de Lenguaje (LLM) disponibles. Se añadió un lector de voz para mejorar la accesibilidad.</p> <p>La metodología incluyó un enfoque de diseño, desarrollo y prueba iterativa para evaluar la efectividad en proporcionar consuelo y aliviar la mente. Los resultados sugieren que el sistema puede ser una herramienta útil para estos propósitos.</p> |   |
| <b>Abstract (in English, 250 words or less):</b>   |   |
| <p>The emerging issue of global loneliness affects individuals who may feel alone even in company, potentially leading to behavioral disorders and diminished quality of life [1]. This project introduces a chatbot with a mini-game designed to simulate human interactions and provide comfort to those experiencing loneliness. The goal is not to replace human companionship, but to offer a 'virtual companion' that combines chat capabilities with virtual entertainment, starting</p>  |   |

with an example of Tic Tac Toe and with potential to change it to other games in the future.

The primary focus is asynchronous interaction, allowing conversations to become monologues if users engage in other tasks like the mini-game. Users can switch between three Large Language Models (LLMs) available. A voice reader was added to enhance accessibility.

The methodology involved a approach of design, development, and iterative testing to assess effectiveness in providing comfort and easing the mind. Findings suggest the system can be a useful tool for these purposes.

# Índice

|   |    |
|---|----|
| 1. Introducción.....  | 1  |
| 1.1 Contexto y justificación del Trabajo .....                    | 2  |
| 1.2 Objetivos del Trabajo.....                                    | 2  |
| 1.3 Enfoque y método seguido.....                                 | 4  |
| 1.4 Planificación del Trabajo .....                               | 6  |
| 1.5 Breve resumen de productos obtenidos .....                    | 9  |
| 1.6 Breve descripción de los otros capítulos de la memoria.....   | 10 |
| 2. Estado del arte .....  | 11 |
| 3. Diseño de la aplicación.....                                   | 12 |
| 3.1 Plataforma destino y requisitos .....                         | 12 |
| 3.2 Entendiendo el programa como un proyecto de integración ..... | 13 |
| 3.3 El chatbot: la charla .....                                   | 15 |
| 3.4 El chatbot: selección y testeo de los LLM a emplear.....      | 17 |
| 3.5 El chatbot: Lector de voz.....                                | 20 |
| 3.6 Ideando la interfaz.....                                      | 22 |
| 3.7 Concepción del minijuego .....                                | 26 |
| 4. Desarrollo de la aplicación .....                              | 27 |
| 4.1 Librerías empleadas .....                                     | 27 |
| 4.2 Implementando los LLM y el chat con memoria .....             | 28 |
| 4.3 Desarrollando el minijuego de ejemplo: Tic Tac Toe.....       | 33 |
| 4.4 Cambios en la interfaz principal .....                        | 38 |
| 4.5 Añadiendo Texto a voz.....                                    | 38 |
| 5. Pruebas y ajustes .....  | 40 |
| 5.1 Pruebas.....  | 40 |
| 5.2 Ajustes.....  | 51 |
| 5.2.1 Ajustando los LLM.....                                      | 51 |
| 5.2.1 Ajustes de Tic Tac Toe.....                                 | 52 |
| 6. Conclusiones y líneas de trabajo futuras.....                  | 53 |
| 7. Glosario .....   | 57 |
| 8. Bibliografía .....   | 60 |
| 9. Anexos .....   | 63 |
| 9.1 Anexo 1: Tabla con la temporalización del proyecto. ....      | 63 |
| 9.2 Anexo 2: Diagrama de Gantt del proyecto.....                  | 64 |
| 9.3 Anexo 3: Ficheros del programa .....                          | 64 |
| 9.4 Anexo 4: Video de muestra del prototipo.....                  | 65 |

## Lista de figuras

|   |    |
|---|----|
| Ilustración 1:Diagrama de casos de uso.....   | 13 |
| Ilustración 2: Diagrama de flujo con los procesos y decisiones mas importantes del modo chat del programa.....                                      | 16 |
| Ilustración 3: Lenguajes usados para entrenar Llama2. Fuente: <a href="https://arxiv.org/abs/2307.09288">https://arxiv.org/abs/2307.09288</a> ..... | 20 |
| Ilustración 4: Tabla comparativa de librerías de texto a voz.....   | 21 |
| Ilustración 5: Zona de información y elección de LLM.....   | 22 |
| Ilustración 6: Zona (QFrame) reservada para el minijuego .....  | 23 |
| Ilustración 7: Zona inferior, chat .....  | 23 |
| Ilustración 8: Segunda ventana de la aplicación, opciones.....  | 24 |
| Ilustración 9: Interfaz antigua del minijuego de ejemplo .....  | 24 |
| Ilustración 10: Interfaz final, en uso.....   | 25 |
| Ilustración 11: Tabla con dos sistemas de inferencia.....   | 29 |
| Ilustración 12: Variante 3x3 del minijuego. La máquina, "O", gana.....  | 35 |
| Ilustración 13: Ejemplo de evaluación de nodos mediante poda. Fuente: <a href="#">UOC</a> .....   | 35 |
| Ilustración 14: Inicio del prototipo.....   | 40 |
| Ilustración 15: El LLM sigue hablando.....  | 41 |
| Ilustración 16: Jugando el minijuego mientras el LLM sigue hablando.....  | 42 |
| Ilustración 17: Se logra un nuevo tres en raya en Normal.....   | 42 |
| Ilustración 18: Continúa el modo automatizado de charla mientras se sigue jugando.....  | 43 |
| Ilustración 19: Primera interacción con chat por parte del usuario.....   | 44 |
| Ilustración 20: Respuesta de LLM y minijuego en modo difícil.....   | 44 |
| Ilustración 21: Cambio a LLM local, Llama 2 7B.....   | 45 |
| Ilustración 22: Llama 2 7B se vuelve un tanto disruptivo.....   | 46 |
| Ilustración 23: Llama 2 7B sigue el diálogo.....  | 47 |
| Ilustración 24: Llama 2 7B se "excede" en su respuesta.....   | 48 |
| Ilustración 25: Llama 2 7B da exáctamente el tipo de respuesta que se busca en el proyecto.....   | 48 |
| Ilustración 26: Se cambia el modelo a Cohere.....   | 48 |
| Ilustración 27: Se cambia el modelo a Cohere y se juega un poco más.....  | 49 |
| Ilustración 28: Command responde un prompt automatizado.....  | 49 |
| Ilustración 29: Mistral, seleccionado de nuevo.....   | 50 |
| Ilustración 30: Tabla con las tareas realizadas y el tiempo que han ocupado..   | 63 |
| Ilustración 31: Tabla resumen de tareas, con su temporalización.....  | 64 |
| Ilustración 32: Muestra de opciones.....  | 66 |

# 1. Introducción

En los últimos años, la sociedad ha sido testigo de un gran interés público en torno a la inteligencia artificial (IA), con *ChatGPT* [2] emergiendo como uno de los principales protagonistas en este ámbito. Otras aplicaciones que usan inteligencia artificial, como *DALL-E*, junto a otras que permiten crear imágenes con diversos estilos y a veces con un estilo fotorrealista a partir del entrenamiento hecho con los modelos, han sido parte de una nueva revolución tecnológica que en poco menos de un año ha demostrado a mucha gente el potencial alcanzado en este campo. Menos conocidos, pero no menos importantes, son los grandes avances en medicina, como la detección temprana de ciertos tipos de cáncer, logrados gracias al uso de inteligencia artificial. Los avances en el medio son numerosos y afectan a muchos campos.

Volviendo a *ChatGPT*, su popularidad ha traído consigo la difusión de conceptos antes menos conocidos, como los *Grandes Modelos de Lenguaje* (*LLM*, por sus siglas en inglés). Estos *LLM* se han destacado como poderosos asistentes conversacionales capaces de abordar una amplia gama de tareas, desde resolver consultas cotidianas hasta realizar tareas complejas como la generación de código de programación. Además, continuamente se están descubriendo nuevas aplicaciones y posibilidades para esta tecnología. Algunas de las fantasías de antaño que escritores y autores de ficción imaginaron hace décadas, incluso siglos, se están empezando a cumplir gracias al uso de redes neuronales profundas que permiten conversar a los *LLM*. No es extraño que, en parte debido a la expectación causada, se estén presentando robots con aspecto humanoide y con un modelo de lenguaje incorporado en eventos tecnológicos.

En este trabajo se buscará determinar hasta qué punto es viable usar un *LLM* portátil, usable en un ordenador de consumo, en conjunto con una actividad de ocio que permita jugar contra la máquina. Estas dos funciones asíncronas estarán integradas en un prototipo con el objetivo de explorar la viabilidad de modelos de lenguaje recientes, combinados con otra tarea para emular el efecto de una interacción humana en la que dos jugadores juegan y conversan sobre temas no asociados al juego en curso.

## 1.1 Contexto y justificación del Trabajo

La soledad está cobrando una importancia clave en la sociedad, con cinco millones de personas contabilizadas en 2021 viviendo solas en España [3]. Este fenómeno no solo afecta a nivel local, sino que es un problema creciente a nivel global, exacerbado por el hecho de que muchas personas pueden sentirse solas incluso estando en compañía [4]. La soledad puede conducir a trastornos de conducta y afectar negativamente la calidad de vida de las personas [5].

Este proyecto surge con la intención de desarrollar un chatbot en conjunto con un sencillo minijuego de máquina contra persona, donde el minijuego vaya acompañado de diálogos, emulando así interacciones similares a las que ocurren entre dos personas. No busca reemplazar la compañía humana, sino explorar cómo proporcionar solaz a personas aquejadas de soledad a través de interacciones que imiten ciertos aspectos humanos como la conversación durante actividades recreativas en compañía (en este caso, con una máquina).

El objetivo principal es crear un "compañero virtual" que combine la capacidad de chat de un asistente con un juego de mesa virtual en pantalla. Inicialmente, el juego será simple, como el tres en raya en un tablero 3x3 con opción para tener otros modos de juego, pero diseñado de manera que el minijuego pueda ser fácilmente intercambiable por futuros juegos, idealmente más complejos. Este compañero virtual está destinado a ofrecer compañía a personas que experimentan soledad extrema o ansiedad de tipo social o por exceso de rumiación, utilizando un ordenador de consumo.

El proyecto se centrará en la interacción entre el usuario y el programa que ejercerá de compañero virtual, permitiendo que las conversaciones se conviertan en monólogos por parte del LLM si las interacciones del usuario disminuyen, funcionando el chat de manera asíncrona respecto a la partida del usuario con el minijuego.

El proyecto se ha diseñado con el objetivo de explorar nuevas posibilidades de interacción entre usuario y máquina en el contexto del hogar y la compañía virtual. El lenguaje de las conversaciones con el chatbot es en inglés. Se explica el motivo de esta decisión lingüística [más adelante en la memoria](#).

## 1.2 Objetivos del Trabajo

Se ha buscado desarrollar una aplicación que ofrezca una mezcla de asistente conversacional y de compañero de juegos ligeros. Concretamente, la aplicación se presenta en forma de base funcional usable que luego pueda evolucionar a futuro, ampliando sus características lúdicas.

La aplicación sería consecuentemente de tipo integrador, aprovechando los avances en tecnología de modelos de lenguaje logrados los dos últimos años, junto a otras librerías desarrolladas por terceros que nos permitan ver mejor hacia qué dirección podría discurrir el avance de estos asistentes en su vertiente de apoyo social a colectivos vulnerables que sufren de aislamiento prolongado.

Se detallará a continuación algunos de los objetivos que se han buscado para este trabajo.

Objetivos principales:

- Integrar un chatbot que converse con el usuario.
- Automatizar el diálogo con el chatbot de manera que el modelo irá diciendo nuevas líneas de diálogo con el paso del tiempo, imitando la característica humana de alguien inmerso en la temática que está conversando con su interlocutor mientras este no le cambie de tema de conversación.
- Crear un minijuego, el cual para este proyecto será un sencillo *tic-tac-toe*, en sus variantes de tres en raya y cuatro en raya, para jugarlo de manera asíncrona respecto al chat.
- Permitir al usuario *chatear*<sup>1</sup> con el LLM que se especifique, de entre una pequeña lista a elegir, siendo dos de ellos modelos locales y uno, un modelo online.
- Diseñar una interfaz para integrar las funciones comentadas.
- Usar los conocimientos aprendidos durante el grado, con especial enfoque en la rama de Computación cursada.
- Dotar a la experiencia de cierto dinamismo con el objetivo de ofrecer solaz a personas que, en ese momento, se encuentren experimentando una fuerte sensación de soledad, o de ansiedad de tipo social o por exceso de rumiación.

Objetivos secundarios:

- Construir las funciones necesarias para adaptar el programa a las diferentes configuraciones de los tres modelos de lenguaje a elegir en el chat.
- Aportar una memoria al chat pudiendo usar la parte mas reciente de la conversación como contexto.
- Desarrollar funciones con reglas para mejorar la experiencia de conversar con el modelo.
- Dar opciones adicionales al usuario para cambiar el reto y el tamaño del tablero del minijuego.
- Aprovechar lo aprendido en algoritmos de inteligencia artificial para darle cierta dificultad al minijuego ofrecido por defecto.
- Optar por una organización tanto de la interfaz como del código que permita adaptar fácilmente posibles minijuegos alternativos en el futuro, si se desea.
- Implementar, con el apoyo de librerías externas, un sistema de texto a voz para que el programa pueda hablar las líneas de diálogo del LLM elegido.

---

<sup>1</sup> Conversar en un chat (*chatear* aún no es un término aceptado por la RAE).

### 1.3 Enfoque y método seguido

El enfoque y método seguido en el proyecto se corresponde con el ciclo de vida propio de proyectos PMBOK<sup>[6]</sup> con sus consiguientes cinco etapas:

1. *Iniciación*: se identifican problemas o necesidades en nuestro caso en el campo de la Inteligencia Artificial y el campo de los asistentes virtuales a usuarios, interpretándose o visualizándose en el proyecto, y se estudia su viabilidad y aprobación. Realizado en la PEC0.
2. *Planificación*: se establece la hoja de ruta que seguirá el proyecto de cara a alcanzar objetivos, incluyendo enfoque de hitos y estimaciones temporales. Se descomponen la diversas actividades para la planificación operativa. Realizado en la PEC1.
3. *Ejecución*: a medida que se van preparando planes mas detallados se revisan los planes elaborados comprobando avances e incluso replanteando aspectos según sea necesario. Se van completando los hitos y objetivos planificados. Sin duda, la etapa mas extensa. A grandes rasgos, incluye la preparación del entorno de desarrollo, testeo y elección de LLMs locales, y el desarrollo del chat, la interfaz y el minijuego entre otros. También la inclusión del lector de voz. Se realiza en las PEC 2 y PEC 3.
4. *Seguimiento y control*: esta etapa cubre todo el proyecto, siendo paralelo y permanente conforme se vaya haciendo. Especialmente importante a lo largo de la etapa de ejecución. Se realiza desde la PEC2 hasta las últimas PEC de presentación del proyecto, en las cuales se hará también una supervisión del conjunto.
5. *Cierre*: se concluye el proyecto y con ello la redacción de la memoria, que parcialmente se ha ido redactando en PECs previos, y se presenta y defiende el proyecto. Se realiza en las PEC 4, PEC 5a y PEC5b.

Esta metodología va en consonancia con la forma en que se han ido haciendo las diferentes PEC, teniendo una metodología en cascada a lo largo del proyecto.

Las diferentes etapas se pueden ver de manera mas visual en el [diagrama Gantt del Anexo 2](#).

Se esperaban, como mínimo, los siguientes riesgos:

- **Problemas técnicos**. Se vieron incompatibilidades entre librerías de desarrollo de Inteligencia Artificial en *Python*, a nivel de versiones, que sin duda iban a afectar constantemente al proyecto. Cómo solucionarlo: se ha buscado en la documentación de dichas librerías de manera constante, para ver el curso de acción recomendado.
- **Desconocimiento de varias de las herramientas que se van a usar**. Se requería investigar el uso de un *IDE* apropiado, y de las diferentes librerías que acaben formando parte del proyecto. Cómo solucionarlo: se ha consultado la documentación online disponible para atajar cada atasco.

- **Extrema novedad de los modelos a usar.** Los LLM tal como se conocen ahora en sociedad, son un campo extremadamente nuevo sobretodo en los últimos modelos salidos. Esto también afecta a la calidad de la documentación existente.  
Cómo solucionarlo: varios de los modelos que han surgido se basan en otros modelos creados por organizaciones mayores, que habitualmente publican un *paper*, como el de *Llama 2* [7]. Se han consultado como fuente de información adicional para las cuestiones mas complejas, junto a la información de los modelos (más información de ello, en el [apartado de selección de LLMs](#)).
- **Coherencia, sentido y toxicidad de los diálogos con el chatbot.** De cara al chat, se ha trabajado con los LLM y se han mostrado como un campo tan apasionante como complejo. Diversos parámetros en juego como la temperatura, los *tokens*, el tamaño del contexto... para realizar un chat eficaz, todos estos parámetros tendrán que tomarse en cuenta. De otra manera, la coherencia (y alucinación del modelo) darán problemas adicionales.  
Cómo solucionar potenciales problemas: se ha decidido usar versiones cuantizadas que provienen directamente de las versiones oficiales de *Mistral* y *Meta*, para evitar posibles datasets tóxicos que pudieran crearse en modelos derivados. También se han consultado las configuraciones que los autores de los modelos y cuantizaciones de los mismos han ofrecido en la web de *Hugging Face*<sup>2</sup>.
- **Cambios recientes de servicios online que afecten directamente a lo que se está haciendo.** Si bien esto puede afectar de cara a cambios a última hora de librerías que se han usado para el proyecto, hay un caso más obvio todavía: este proyecto buscaba ofrecer un LLM online como una de las opciones de cara a hablar con un modelo, en este proyecto siendo el modelo más popular, ChatGPT. Dicho escenario ocurrió y ChatGPT no pudo ser incorporado debido a ciertos problemas con la licencia desde la máquina en la que se desarrollaría el aplicativo. Como solucionarlo: en el caso de librerías, siempre existe la opción de pasar a usar librerías mas antiguas en el desarrollo. En el caso del ChatGPT citado, se buscó otra alternativa con API usable desde nuestro aplicativo. La empresa Cohere Inc. ofrece modelos de uso gratuito como su modelo *Command*, usable para chat online.

En los apartados de diseño y desarrollo de la aplicación, capítulos 2 y 3, se hablará mas a fondo de como se han implementado estos apartados.

---

<sup>2</sup> <https://huggingface.co>

## 1.4 Planificación del Trabajo

Las siguientes son las fases de las que consta este trabajo y ordenadas en base a los PEC entregados en el semestre lectivo. Se pueden ver las siguientes fases y tareas organizadas de manera temporal a lo largo de los días mucho más sintetizadas, en la [tabla del Anexo I](#), y el [gráfico Gantt del Anexo II](#).

### Fase 0: Elección y definición del proyecto

Pequeña fase previa en la que se elabora una versión superficial del trabajo final, contextualizando el material que se presentará. Se crea y define la razón de ser del proyecto y varias de las ideas que tomarán forma a lo largo de las fases principales del mismo.

### Fase 1: Plan de trabajo

Se entrega al profesorado un documento que presenta los objetivos del proyecto, tanto principales como secundarios, junto con una investigación que sitúa el proyecto en el campo de la Inteligencia Artificial e identificando el tipo de usuario al que irá dirigido el aplicativo, abarcando tanto aspectos técnicos como sociales. Además, se identifican potenciales problemas y se proponen planes de mitigación, y se elabora una planificación temporal general.

### Fase 2: Desarrollo del proyecto, parte 1

Primera parte de la sección de desarrollo en el proyecto, comprendiendo algo menos de 5 semanas. En concreto, han intervenido estas tareas:

1. **Preparación del entorno de desarrollo en Python:** Configuración del entorno de desarrollo Python a nivel local. Se han instalado las librerías necesarias, incluyendo *CUDA* y *TensorFlow*, y se han verificado las variables y preparado el entorno en Windows. Se exploró la posibilidad de trabajar directamente online en *Colab* para reducir recursos y gastos en consumo, pero finalmente se descartó debido a la disponibilidad de una tarjeta gráfica más rápida en el equipo de desarrollo, en comparación con la potencia que *Colab* ofrecía en su versión gratuita, una vez se supera su límite diario. *Anaconda Navigator* resultó ser un conjunto ideal de herramientas Python para este proyecto y para un control cómodo del entorno.
2. **Selección de herramienta visual en Python para la interfaz:** Evaluación de las herramientas visuales disponibles en Python para facilitar el diseño de la interfaz (botones, etiquetas, pestañas, etc.). Se investigaron opciones como IDEs que faciliten la integración del entorno, eligiendo finalmente *QT Designer*, una IDE fácil de usar con la

metodología WYSIWYG<sup>3</sup> y suficientemente capaz frente a alternativas como *Tkinter* o *QT Design Studio*.

3. **Selección y testeo de grandes modelos de lenguaje para hacer una primera selección:** Evaluación de varios LLMs en sus variantes de uso local, consultando la página Hugging Face. También se eligió un primer modelo online, para permitir el uso deseado de dos modelos offline y uno online. Esta fase es principalmente de tanteo, pues el mundo de los LLM está en constante evolución y se anticiparon posibles problemas al respecto mientras este proyecto se gestionase. Esta tarea va estrechamente vinculada con la siguiente.
4. **Testeo adicional y reevaluado de LLMs, con la posibilidad de que sean reemplazados en cualquier momento.** Esta tarea se puede interpretar como una extensión de la tarea previa pues dura desde su final, hasta poco antes del final del desarrollo del producto, discurriendo en paralelo con el resto de tareas con las que esta tarea ha coincidido temporalmente. A lo largo de esta tarea se eligieron los tres modelos definitivos, dos offline y uno online, que se acabarían usando en la aplicación. También se hicieron cambios en el código acordados para acoger los diferentes modelos.
5. **Desarrollo del chat con memoria:** Investigación y selección del enfoque adecuado para implementar un chat con memoria, incluyendo la gestión de tokens para optimizar el rendimiento, el uso de una variable de lista, y el enfoque del BOT de cara a llevar el diálogo.
6. **Diseño de una versión prototipo para integrar el chat:** Creación de una interfaz de usuario no final para alojar el chat, a modo de beta, que permita tener una interfaz con la que ir trabajando. Se empieza a trabajar en un sistema de capas y *frames* para dividir las diferentes funciones de la aplicación en la interfaz. Se integra el chat en la GUI.
7. **Elaboración de informe de seguimiento para la fase 2:** Reflexión acerca de cómo se han llevado las tareas 1 a 6. Análisis de los resultados obtenidos hasta este punto, actualización de la planificación temporal de hitos según los resultados, y búsqueda de posibles nuevas rutas para las siguientes tareas.

### Fase 3: Desarrollo del proyecto, parte 2

Segunda parte del desarrollo, comprendiendo poco más de 3 semanas. Como se comentó en la fase 2, la tarea 4 de “Testeo adicional y reevaluado de LLMs” ocupará un enorme espacio temporal del proyecto, y se solapará con el resto de tareas de esta fase. Se continúa con el orden en numeración de tareas respecto a la fase previa, como un todo a nivel de bloque de desarrollo:

---

<sup>3</sup> Siglas en inglés de “What You See Is What You Get”, traducible como “lo que ves es lo que tienes”. Referido a un uso fácil e intuitivo con iconos que representan lo que se está manejando en pantalla.

8. **Desarrollo, optimización e inserción de un minijuego en la interfaz:** Implementación de un Tic Tac Toe como ejemplo para el programa, en sus variantes de tablero 3x3 y 4x4, con opciones específicas del minijuego e integración en la interfaz. Se ha trabajado con un algoritmo de Inteligencia Artificial para que la máquina pueda hacer de oponente.
9. **Finalización de la interfaz:** Se concluye el aspecto final de la interfaz. Junta una sección fija para minijuegos de manera que sea fácil un hipotético cambio del minijuego de ejemplo por otro, con otras secciones relacionadas con el chat y sus opciones.
10. **Incorporación de Texto a Voz:** Se decidió que la integración de un lector de voz ayudaría a aportar inmersión en la charla con el asistente, por lo que se ha implementado la posibilidad de usar un lector que emplee las voces de "Texto a Voz" disponibles en un sistema Windows.
11. **Testeo y pruebas de programa:** Con el programa integrando todas las funciones se limpió el código y se hicieron los testeos, pruebas, y ajustes finales que se vieron necesarios.
12. **Elaboración de informe de seguimiento para la fase 3:** De manera similar al informe del final de la fase 2, se reflexiona sobre los resultados de las diferentes tareas, los problemas encontrados y las incorporaciones de última hora.

El desarrollo del proyecto concluye y se pasa a la siguiente fase.

#### **Fase 4: Redacción del *paper***

La redacción de la memoria del proyecto, el *paper*, explicará todo el trabajo realizado. En esta fase, se recopilan los resultados y conclusiones, se analiza el camino seguido, las pruebas realizadas y las rutas hipotéticas que podrían tomarse para mejorar la aplicación si se le dedicaran más tiempo y recursos.

#### **Fase 5: Elaboración de la presentación**

Se elabora una presentación que luego se defenderá. Consta de las siguientes partes:

- **Fase 5a: Preparación de la presentación.** Para explicar de manera precisa y sintetizada el trabajo llevado a cabo, se elabora una presentación con apoyo audiovisual. Esta presentación tiene como objetivo proporcionar al tribunal de evaluación una perspectiva clara y precisa del TFG.
- **Fase 5b: Defensa pública del proyecto.** El proyecto se defiende respondiendo a todas las preguntas que puedan tener los miembros del tribunal que evaluará el trabajo.

## 1.5 Breve resumen de productos obtenidos

Desde el principio, este proyecto no ha buscado crear un producto final, sino más bien un prototipo que permita explorar las posibilidades futuras en asistentes virtuales de compañía y sus capacidades de dar diálogo mientras hacemos otras tareas. La aplicación integra algunas de las funciones que son ahora posibles gracias a los avances de los LLM en los últimos dos años, en combinación con LLMs ya desarrollados en ese tiempo.

Se ha creado un programa que permite dialogar con uno de tres modelos de lenguaje posibles, dos locales y uno online, los cuales seguirán hablando cada cierto tiempo aunque no se les den nuevas líneas de diálogo desde el teclado. Esto tiene el objetivo de imitar a un interlocutor que continúa en su monólogo de manera relajada si no le interrumpimos. Sus líneas pueden ser leídas con la librería de texto a voz, mientras podemos jugar de manera asíncrona con el minijuego incluido.

Para ello hay seis archivos python desarrollados para el proyecto, junto a un séptimo que meramente tendrá la clave almacenada en una cadena de texto para poder usar el modelo online que se ha usado en la aplicación.

Es importante reseñar que los dos modelos de uso local deben estar ubicados en los directorios correspondientes en el disco duro del proyecto, ya que el programa necesitará acceder a ellos para poder utilizarlos. Teniendo en cuenta que este trabajo es un proyecto universitario y que los modelos locales utilizados han sido creados por las empresas Meta y Mistral, y cuantizados por un tercero, se han incluido enlaces en el Anexo 3 para poder usarlos en el programa en caso de que se tenga curiosidad.

Se puede ver información adicional de los archivos del proyecto en el [Anexo 3](#).

## 1.6 Breve descripción de los otros capítulos de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1: Introducción.** Presenta el proyecto realizado junto a la planificación de las tareas necesarias para lograr los diferentes objetivos.
- **Capítulo 2: Estado del arte.** Describe el estado actual de los modelos de lenguaje y las nuevas posibilidades que están brindando a la población, especialmente las relacionadas con este proyecto en el campo de los asistentes para la salud mental.
- **Capítulo 3: Diseño de la aplicación.** Investigación de como implementar el prototipo, principalmente en la fase previa al desarrollo, presentando las diferentes partes en su concepción, incluyendo explicación del diseño de la interfaz. Incluye una introducción para hablar del tipo de ordenador de consumo que será necesario para usar el prototipo.
- **Capítulo 4: Desarrollo de la aplicación.** Comentarios del desarrollo a nivel de código de la aplicación, incluyendo librerías empleadas y el por qué de los acercamientos elegidos.
- **Capítulo 5: Pruebas y ajustes.** Sección donde mediante unas pruebas ilustrativas se muestra el funcionamiento del programa en diferentes escenarios, y comentarios acerca de ajustes en diferentes partes del prototipo.
- **Capítulo 6: Conclusiones y discusión.** Se elaboran conclusiones, se hace autocrítica, y se discuten posibles vías futuras para el prototipo.
- **Capítulos 7 y 8: Glosario y Bibliografía.** Autodescriptivos.
- **Capítulo 9: Anexos.** Información adicional de apoyo para algunos de los apartados, a los que se hará referencia a lo largo del trabajo.

## 2. Estado del arte

Desde la irrupción de ChatGPT en sociedad, el interés de la gente hacia los modelos de lenguaje se ha disparado. Numerosas compañías han desarrollado modelos de lenguaje y han surgido multitud de chatbots nuevos en internet. Sin embargo, es importante recalcar la palabra “nuevos”. En el contexto de acompañamiento a pacientes, se diseñó la computadora ELIZA<sup>[8]</sup> que surgió en 1967 y buscaba interactuar con los usuarios para ofrecer apoyo emocional. Las décadas posteriores pudieron verse chatbots aún rudimentarios como Dr. Sbaits<sup>4</sup>, que permitía hacer un arcaico chat con la máquina a inicios de los años 90.

En el campo de la computación ha llovido mucho y se pueden ver páginas web que ofrecen chatbots de alta calidad como la página [character.ai](https://character.ai), que permite hacer *roleplay* con personajes ficticios donde la máquina (en concreto, el LLM que es creación y propiedad de la empresa, que está alojado en sus servidores) conversa con usuarios.

Muchas empresas privadas del ramo de la sanidad se han tomado el asunto muy en serio y se están empezando a ver bots de nivel profesional en el ámbito de la salud mental [9]. Un campo que fue potenciado por *Woebot*<sup>5</sup>, un chatbot impulsado por figuras importantes del mundo de la inteligencia artificial, que provee ayuda a personas con depresión y ha demostrado ser efectivo entre jóvenes adultos, según un estudio médico [10]. Woebot utiliza técnicas de procesamiento de lenguaje natural y principios de terapia cognitivo-conductual para interactuar con los usuarios, según se describe en su web.

Tanto Woebot como Eliza coinciden en realizar un acercamiento conversacional, que en sí mismo es una forma de terapia [11], y estos modelos permiten una conversación y un apoyo emocional que puede complementar el aportado por un terapeuta real.

El chatbot creado en este proyecto no busca enfocarse excesivamente en el *roleplay* (más allá de una pequeña orden interna que se dará al bot sobre cómo contestar) y, desde luego, tampoco puede proveer un acercamiento estrictamente médico, ya que no dispone de la formación para ello. En cambio, buscará proporcionar un diálogo entre el usuario y el LLM elegido, para que el usuario exprese sus pensamientos o curiosidades, a la vez que se distrae con un minijuego de ejemplo.

---

<sup>4</sup> <https://www.youtube.com/watch?v=1lzJMjxTA7k>

<sup>5</sup> <https://woebothealth.com/>

## 3. Diseño de la aplicación

En este capítulo se explorará el diseño de la aplicación, que abarca tanto la fase previa al desarrollo como aspectos que se diseñaron de manera conjunta durante la implementación. Tras una introducción para hablar de la máquina que necesitaría el prototipo, se detallará cómo se planificaron las diferentes partes del programa en diseño y sus funcionalidades, destacando las decisiones tomadas en esta etapa que influyeron en el desarrollo posterior.

### 3.1 Plataforma destino y requisitos

El prototipo se ha ideado para que pueda ser ejecutado en un ordenador que tenga *Windows 10* o superior, y una tarjeta gráfica nVidia compatible con CUDA<sup>6</sup> que tenga en torno a 8GB de *VRAM* o más, por la demanda de requisitos de los modelos de lenguaje que se han usado para este proyecto. También se recomendaría una unidad de almacenamiento rápido, como una unidad M.2, dado que los modelos ocupan varios gigabytes de espacio y la aplicación tiene entre sus funciones el poder parar el modelo activo para cargar otro de los accesibles desde el prototipo.

A la hora de crear un chat para poder conversar con un LLM local, alojado en la misma máquina del usuario, es necesario recordar que estamos en un momento tecnológico en el que estos modelos se pueden mover, al menos en un ordenador accesible para el público, mediante:

- memoria RAM usando la CPU,
- memoria VRAM usando la GPU, o
- ambas

Sin embargo, el uso de memoria RAM es significativamente más lento en comparación, incluso cuando se utiliza de manera parcial. Por esta razón, se ha optado por una generación de tokens mucho más rápida usando únicamente la GPU con sus librerías CUDA.

En cuanto al LLM online elegido, el ordenador que ejecute la aplicación no debería de usar recursos reseñables dado que el modelo estará alojado en un servidor al que accederemos por internet. Esto implica que se podría haber diseñado un prototipo alternativo que únicamente interactuase con uno o más modelos online, si se quisiera usar la potencia de la que disponga el servidor en vez de la que se disponga localmente, pero se ha buscado experimentar con ambas vías priorizando el uso de modelos locales dado que permiten una mayor privacidad, mayor seguridad, y menor dependencia de conexión a internet.

Como se ha comentado en [1.4 Planificación del Trabajo](#) en la fase 2, se ha utilizado Python junto con un conjunto de herramientas para desarrollar el prototipo, el cual se presentará funcionando en PyCharm para este proyecto. Dado que se trata de un proyecto de final de carrera en un entorno educativo,

---

<sup>6</sup> Listado de tarjetas compatibles, en este [enlace oficial](#) de nVidia.

destinado inicialmente a ser ejecutado únicamente en la máquina en la que se ha desarrollado, no se consideró necesario presentarlo en un formato compilado distribuible y ejecutable como EXE.

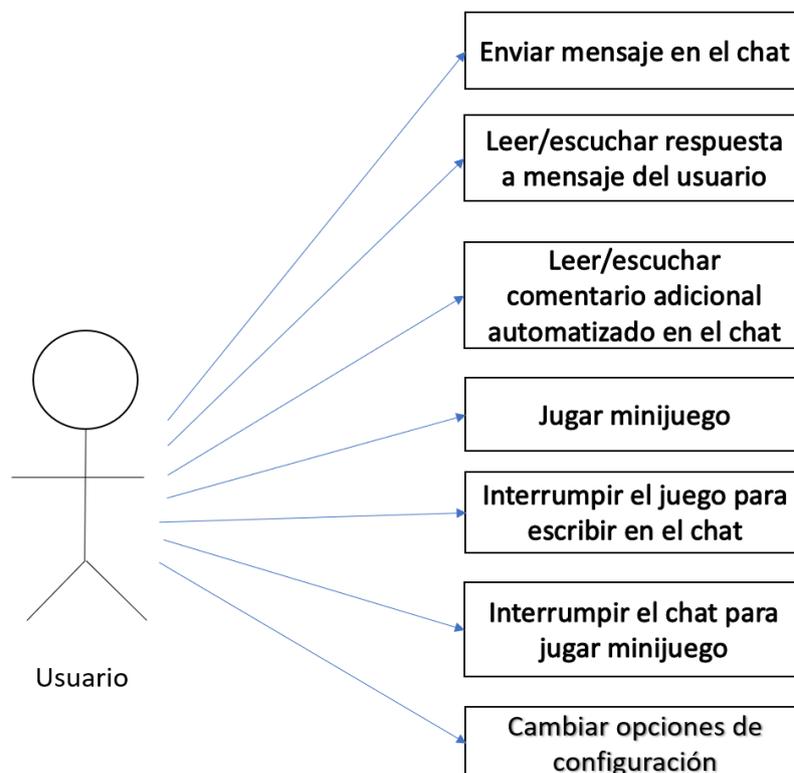
### 3.2 Entendiendo el programa como un proyecto de integración

Dado que esta aplicación tiene como objetivo integrar diversas funciones, es importante tener en cuenta que estamos hablando de la gestión simultánea de dos tareas: conversación y juego. En ocasiones, la conversación de cara al LLM se automatizará pidiendo con el tiempo nuevas líneas de diálogo que el LLM dará al programa, incluso si el usuario no está escribiendo activamente. Como resultado, el usuario encontrará las siguientes opciones en el programa:

- Escribir al LLM.
- Leer o escuchar las respuestas del LLM, ya sea en respuesta al usuario o de forma automática sobre el tema de conversación activo
- Jugar al minijuego.
- Modificar algunos parámetros relacionados con el chat o el minijuego, mediante botones o listados de opciones dedicados para tal fin.

Si el usuario lo prefiere, puede optar por escuchar al LLM durante un tiempo sin necesidad de escribir, mientras se concentra en el minijuego.

El siguiente gráfico ilustra visualmente lo que el prototipo del proyecto proporcionaría a un usuario:



*Ilustración 1: Diagrama de casos de uso*

En relación con el diagrama mostrado antes, se entenderá por 'interrumpir' el dejar de prestar atención a una tarea para enfocarse en otra, o al menos, reducir la atención a una tarea mientras se dirige mayor atención a otra. En este sentido, activar la opción del lector de voz en el prototipo ayudaría a equilibrar mejor el minijuego y la conversación, ya que es más fácil usar la vista para una tarea y el oído para la otra, que usar la vista para ambas tareas.

Es importante destacar que, desde el inicio, el proyecto se acometió desde una perspectiva de investigación, explorando las posibilidades que se están abriendo con los nuevos avances en modelos de lenguaje y asistentes, en conjunto con otra actividad. Siempre se tuvo en cuenta que el plazo disponible era limitado a unas ocho semanas de desarrollo. Por ello, se adoptó un enfoque realista, con el objetivo de investigar cómo los LLM podrían utilizarse para tareas de acompañamiento combinadas con una actividad lúdica, en lugar de centrarse en la calidad sólida de un producto final. Se buscaba ver cómo se podía acometer un chatbot, junto a un juego.

Una tarea que difícilmente podrá hacer un LLM es pensar las jugadas del minijuego: los modelos de lenguaje actuales tienen problemas constatados para resolver ciertos problemas que en principio no serían complejos para una persona, incluso un modelo potente como el usado por ChatGPT puede dar respuestas insatisfactorias a nivel objetivo con ciertos problemas a resolver, que motiven a pedir nuevas respuestas [12]. Y cuanto menos potente sea el LLM que se use, el reto para resolver problemas será mayor.

Siempre se podría entrenar un modelo para que entienda mejor algunas tareas, en este caso los movimientos lógicos que un oponente humano esperaría de su contrincante en un Tic Tac Toe, pero el tiempo para realizar el proyecto era escaso y no se vió como pragmático el esfuerzo que conllevaría una tarea así, existiendo alternativas más simples. Principalmente no parece especialmente viable dado que no es nada extraño en el mundo de la ingeniería integrar diferentes módulos para un todo, con módulos que por separado no tienen dependencia de otros módulos. Por ejemplo, en un coche que funcione con gasolina la radio, el airbag, el GPS, el sistema de luces, son completamente independientes del motor de explosión que propulsa el coche.

Consecuentemente, el minijuego tendrá que usar un algoritmo de inteligencia artificial independiente del modelo de lenguaje y la función de chat, para ofrecer algo de reto minimizando el coste computacional.

### 3.3 El chatbot: la charla

En apartados previos se ha mencionado que el usuario podrá chatear o jugar de manera asíncrona, es decir, las decisiones del programa respecto al chat no van vinculadas a las decisiones del programa respecto al juego. También se ha recordado que, en una conversación, quien habla suele dedicar más enfoque a la conversación que quien escucha. Cuando la voz proviene de una máquina, la atención puede disminuir aún más, como en los ejemplos de escuchar la radio o un podcast.

Independientemente de que el usuario juegue al minijuego, es el usuario quien decide prestar mayor o menor atención a lo que tenga que decir el modelo de lenguaje.

Así pues, el programa facilitará la interacción con el LLM, permitiendo la creación de un chat entre el usuario y la máquina de forma cómoda y funcional, actuando como puente entre el modelo de lenguaje y el usuario.

Se busca que el prototipo cuente con varias opciones para el chat:

- Cambiar a otro modelo de lenguaje, de entre tres posibles.
- Elegir de entre un listado de opciones para modificar el contador de tiempo para la charla automatizada del modelo.
- Ajustar el tamaño de contexto del chat, de entre varias opciones.
- Elegir si se desea un lector de voz.

Se desarrollarán ahora un poco más, estas cuatro opciones.

Como se detallará en el apartado siguiente de [selección y testeo de modelos](#), los modelos de lenguaje que han surgido en los últimos años pueden tener diferencias significativas incluso al tratar los mismos temas de conversación. Estas diferencias pueden manifestarse tanto a nivel de software como en la propia interacción con el modelo.

Cada vez que el LLM diga su parte del diálogo, iniciará un contador en segundo plano que se reseteará si el usuario está escribiendo en la aplicación, otorgándole tiempo adicional para enviar su mensaje si quiere preguntar sobre lo último que el LLM haya comentado.

Poder ajustar el tamaño del contexto es útil, ya que consume recursos adicionales de hardware cuanto más se utilice y puede provocar una mayor ralentización. Sin embargo, también puede ser beneficioso permitir que el modelo de lenguaje recuerde conversaciones más largas.

Finalmente, la opción de elegir un lector de voz permitirá escuchar lo que el LLM escriba. Esto es útil si el usuario no desea enfocarse especialmente en lo dicho por la máquina mientras juega, o si simplemente busca dividir su atención entre ambas tareas.

Se añade una explicación visual de cómo enfoca el programa el trato hacia el LLM en el flujo de su diálogo con el usuario y mediante automatismos, en el siguiente gráfico:

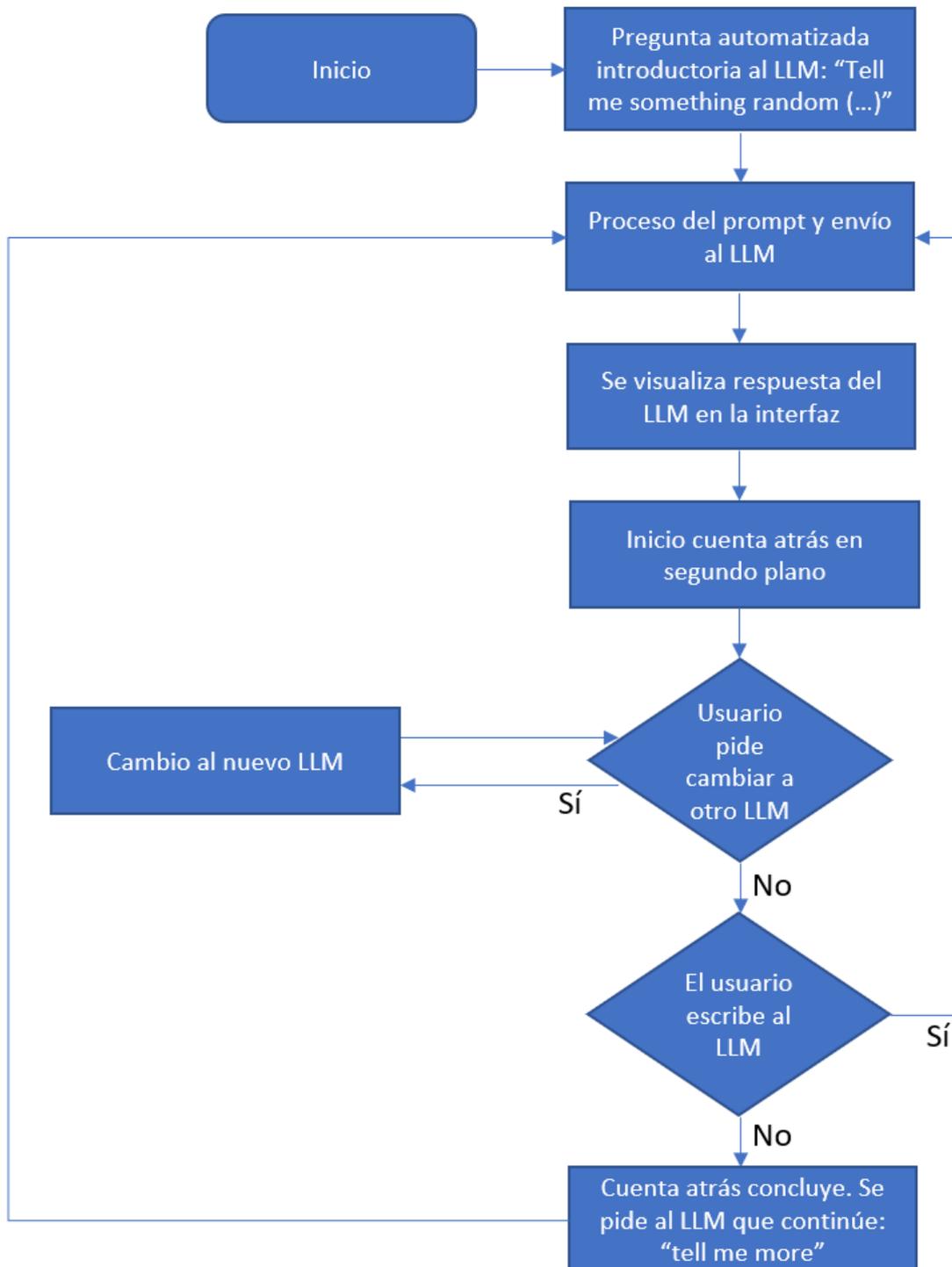


Ilustración 2: Diagrama de flujo con los procesos y decisiones más importantes del modo chat del programa

En la figura anterior se puede ver que el programa que maneja el chat está en un bucle, con el LLM imitando a un interlocutor con mucho que decir, al que en todo momento le podemos cambiar el tema de conversación.

Como se puede observar en el último diagrama, es necesario realizar un acercamiento específico para interactuar con un LLM. El *prompt*, además de incluir la propia petición de lo que el usuario o programa desea preguntar en su forma más básica, generalmente se acompaña de una serie de datos adicionales. Es común que cada modelo requiera un tipo específico de prompt para gestionar adecuadamente la información solicitada. En la sección de desarrollo, se abordará este tema de manera más técnica y específica.

Se podría haber añadido un nuevo proceso a dicho diagrama, pero se ha preferido mantenerlo fácil de entender. Este proceso adicional se relacionaría con el tratamiento de respuestas. Los modelos de lenguaje, sobretodo en las variantes pequeñas, pueden llegar a generar respuestas "en bruto" que a menudo incluyen caracteres *ASCII*, resultado del material con el que fueron entrenados originalmente. Por ejemplo, a veces un modelo puede responder incluyendo `<HTML>` y `</HTML>` en sus respuestas, y otros tipos de etiquetas. Por ello, en la parte de desarrollo, se han establecido algunas reglas para excluir subcadenas de texto molestas que se han ido identificando, aunque todavía está lejos de crear respuestas completamente limpias.

### 3.4 El chatbot: selección y testeo de los LLM a emplear

En la concepción de este proyecto, se ha decidido dar a elegir al usuario entre tres modelos de lenguaje con los siguientes objetivos:

- Evaluar la fluidez al cambiar entre dos modelos offline, locales, situados en la máquina del usuario.
- Experimentar con un modelo alojado online, situado en un servidor.
- Analizar la factibilidad de un programa que permita cierta modularidad, dado que el mundo de los LLM está evolucionando rápidamente. Los modelos disponibles recientemente superan con creces las prestaciones de sus equivalentes de hace un año, pidiendo menos requisitos y ofreciendo más potencial [13]. Se espera que esta línea de mejora de rendimiento a cambio de menos requisitos continúe de manera ascendente en el futuro. Por este motivo, se busca que este programa pueda cambiar de modelo offline con facilidad, mejorando sus capacidades de chat con modelos mas nuevos y mejor optimizados.

Consecuentemente, el proyecto incluirá la posibilidad de usar dos modelos locales y uno online.

Actualmente los modelos de lenguaje locales pueden ser retantes a la hora de elegirlos y ejecutarlos en un ordenador, en lo que tener el hardware necesario se refiere. Se sabe que el *GPT-3* de ChatGPT necesitaba hasta 350 gigabytes de *VRAM*, y Llama 2 70B unos 140 gigabytes, en sus variantes de precisión *bfloat16* [14], aunque obviamente no se llegará a tal extremo en este trabajo y

se elegirán LLM mucho más pequeños y menos demandantes, como una de las versiones menores de la familia Llama de la empresa Meta [15]. No solo eso, también serán versiones cuantizadas, dado que incluso los cada vez más populares modelos de la franja 7B-11B necesitan de ello para no tener unos requerimientos de hardware extremadamente altos.

Es extremadamente habitual en las interacciones con LLM locales, el uso de tarjetas gráficas que dispongan de optimización con librerías CUDA y cierta cantidad de memoria VRAM (habitualmente de 8 Gigabytes), dichas tarjetas son de la empresa nVidia, suelen ser de la familia RTX3, y suelen tener un precio de venta al público que rondan en su tramo inferior los 200-300 euros a fecha de inicios de 2024. Esto hace que sea necesario un desembolso inicial, si el ordenador del usuario estaba preparado únicamente para tareas de ofimática. Se ha buscado el uso de modelos menores usables con el tramo inferior comentado, si bien en un futuro inmediato los NPU<sup>7</sup> descargarán de trabajo a las GPU y CPU para procesar modelos de lenguaje gracias a sus librerías optimizadas para inteligencia artificial [16]. Sin embargo para cuando esta memoria se entregue, las NPU para mejorar el uso de LLM todavía no son accesibles para el gran público y consecuentemente el trabajo ha tenido que ser preparado para funcionar con GPU, más rápida que la CPU para estas tareas.

A lo largo de los últimos dos años han surgido múltiples variantes de cuantizados en modelos [17], y se ha escogido para el trabajo el formato GTPQ, el cual nos permite trabajar completamente con la memoria de la tarjeta gráfica en su totalidad, descargando a la CPU de trabajo cuando se opera con un LLM local.

En un principio se buscaba el menor coste computacional posible y se intentó utilizar los modelos más pequeños que pudieran funcionar con hardware menos potente. Los avances en este campo están ocurriendo a una velocidad sorprendente, con variantes de modelos conocidos lanzándose constantemente. En los últimos dos años, empresas como Meta han lanzado hasta tres generaciones de su LLM más popular, Llama [15]. En la comunidad de Hugging Face han surgido modelos ligeros, como el modelo basado en Llama llamado TinyLlama [18], que buscan minimizar el consumo y la potencia necesarios. Sin embargo, TinyLlama y otras opciones similares con menos miles de millones (o “B”, de los *billions* en inglés, es decir, miles de millones) de parámetros fueron descartadas. Aunque estos modelos pequeños tienen un bajo coste computacional, no parecen ser suficientemente coherentes para mantener conversaciones fluidas y variadas.

Aunque esta categoría de modelos ligeros está evolucionando rápidamente, durante el periodo de diseño y desarrollo del proyecto se descartó por completo la viabilidad de incorporar un modelo con menos de 7B parámetros.

En la duración del proyecto se testearon y reevaluaron los modelos con mayor probabilidad de ser elegidos para ser usados con el prototipo, con la posibilidad de ser reemplazados en cualquier momento dada la enorme cantidad de modelos de código abierto que se publican de manera continua. Se realizó una

---

<sup>7</sup> NPU: siglas de “Neural Processing Unit”, que en español sería Unidad de Procesamiento Neuronal. Está optimizada para tareas de inteligencia artificial.

actividad de monitorización casi diaria consultando Hugging Face, una fuente de información clave para este trabajo. Hugging Face ha sido fundamental para acceder a información relevante sobre los LLM y descargar los modelos offline que se seleccionaron. Desde configuraciones, enlaces a papers y modelos para descargar, Hugging Face se ha convertido en una fuente inagotable de información sobre modelos de código abierto, facilitando su testeo básico y la comprensión de sus potencialidades.

Finalmente, se han elegido como modelos offline las siguientes versiones *cuantizadas* por el usuario Tom Jobbins<sup>8</sup>, en GPTQ, de modelos en la franja 7B:

- Versión cuantizada de *Mistral 7B Instruct 0.2* [19], y
- Versión cuantizada de *Llama 2 7B Chat* [20].

Dos motivos principales para la elección de Mistral 7B y Llama 2 7B son las prestaciones ofrecidas y las licencias que mantienen sus versiones cuantizadas. Ambos son modelos muy capaces. Mistral, de la empresa Mistral AI, se ha vuelto extremadamente popular, con múltiples modelos derivados creados cada mes, y Llama 2 pertenece a la penúltima generación de la familia Llama de Meta, que también ha tenido un impacto significativo entre los modelos usables en ordenadores particulares, especialmente en las gamas con menos miles de millones de parámetros.

Mistral usa una licencia Apache 2.0 [21], mientras que Llama 2 emplea una licencia propia de Meta [22] que, en principio, no debería presentar problemas para su uso en el ámbito de este proyecto. Como se ha comentado se utiliza una versión derivada de cada uno de estos modelos, cuantizada en formato GPTQ para una mayor portabilidad.

En cuanto al modelo online se ha acabado eligiendo el LLM Command de la empresa Cohere, para el proyecto. Cohere da muchas facilidades de uso, con acceso limitado pero suficiente para este proyecto, y mucha documentación para todo lo referido a su API [23].

### **Acerca de la elección del idioma inglés para conversaciones**

Es posible que el lector se pregunte por qué se ha elegido el idioma inglés para las conservaciones con el chatbot del proyecto, cuando esta memoria está en español. El principal motivo lo podremos ver en la siguiente tabla:

---

<sup>8</sup> <https://huggingface.co/TheBloke>

| Language | Percent | Language | Percent |
|----------|---------|----------|---------|
| en       | 89.70%  | uk       | 0.07%   |
| unknown  | 8.38%   | ko       | 0.06%   |
| de       | 0.17%   | ca       | 0.04%   |
| fr       | 0.16%   | sr       | 0.04%   |
| sv       | 0.15%   | id       | 0.03%   |
| zh       | 0.13%   | cs       | 0.03%   |
| es       | 0.13%   | fi       | 0.03%   |
| ru       | 0.13%   | hu       | 0.03%   |
| nl       | 0.12%   | no       | 0.03%   |
| it       | 0.11%   | ro       | 0.03%   |
| ja       | 0.10%   | bg       | 0.02%   |
| pl       | 0.09%   | da       | 0.02%   |
| pt       | 0.09%   | sl       | 0.01%   |
| vi       | 0.08%   | hr       | 0.01%   |

Ilustración 3: Lenguajes usados para entrenar Llama2. Fuente: <https://arxiv.org/abs/2307.09288>

En la tabla anterior, que se puede encontrar en el paper de Llama 2 [7], se puede ver que en el entrenamiento base de dicho modelo los idiomas que no sean el inglés (89.70%, en la tabla), ocupan un papel muy irrelevante (podemos ver que tanto “es” como “ca”, en referencia al español y el catalán, no llegan a un 0.2% entre ambos) para dar una comprensión muy superficial al modelo sobre otros idiomas. Esta comprensión permite al modelo tener una idea rudimentaria de lo que le puedan decir en otro idioma, pero el modelo tendrá serios problemas para explicarse en un idioma que no sea inglés, de una manera que no añada dificultad al seguimiento de la conversación.

Se han probado muchos LLM fluidos en conversaciones casuales, derivados de los más populares incluyendo el mencionado Llama 2 en franjas bajas (20B y menores), y en lo referente a LLM locales no se han encontrado modelos que sean buenos tanto en dar conversación como en fluidez en español a un nivel que no suene extraño para quien sea hispanohablante de manera nativa. Es necesario recordar además, que se está hablando de la franja de modelos 7B, pero también se han visto carencias serias en modelos mas potentes usables en franjas inmediatamente superiores, como 13B, 20B y 34B, los cuales además consumen muchísimos mas recursos.

Consecuentemente, se usará el idioma inglés en la fase de desarrollo y configuración del chat con los modelos, requiriendo además que el usuario se exprese en ese idioma.

### 3.5 El chatbot: Lector de voz

Al iniciar este proyecto, la inclusión de lector de voz era considerada un objetivo secundario debido a las limitaciones de tiempo para la creación del trabajo. Sin embargo, posteriormente se decidió incluir esta funcionalidad.

La idea principal fue recrear la experiencia de conversar con una máquina de manera similar a como lo haríamos con otra persona en un entorno informal, como un bar o durante actividades recreativas como jugar a juegos de mesa como el mus, la brisca o el dominó. Esta inspiración condujo al desarrollo de una aplicación que integrara un chat con un LLM y un minijuego.

Se reconoció que, aunque inicialmente no se consideró la opción de convertir el texto en voz (hablar a un micrófono para comunicarse con el programa) debido a que las personas tienden a prestar más atención durante una conversación activa, esta falta de atención es más tolerable cuando la otra entidad que nos habla es permisiva. Por ejemplo, muchas personas realizan actividades mientras escuchan la radio, la televisión o un podcast.

Por lo tanto, se hizo evidente que un lector automatizado de voz era fundamental si la aplicación pretendía imitar la actitud de un usuario que está pensando en el próximo movimiento en un juego competitivo mientras decide prestar más o menos atención a lo que el LLM está diciendo. Una persona puede escuchar la radio y enfocarse en una tarea importante, pero no podrá enfocarse fácilmente en dos actividades que requieran su vista. En otras palabras, si no se proporciona una voz para leer los textos, la experiencia resultante se alejará de la combinación de escuchar un diálogo y jugar visualmente un juego.

La ausencia de una voz que lea los textos podría afectar significativamente la experiencia, haciendo que la simulación de una conversación normal mientras se juega con un rival humano sea menos auténtica, especialmente si se espera que el usuario lea el diálogo en pantalla.

En Python dos de las opciones más conocidas de texto a voz son:

- La *librería gtts*, acrónimo de *Google Text-to-Speech*. Su uso gratuito tiene un límite aunque es holgado si se usan las voces estándar que permiten [24].
- La *librería pyttsx3*, usa una licencia *Mozilla Public License, v. 2.0*, y además tiene la ventaja de poderse usar de manera offline [25].

A grandes rasgos este sería un resumen de ambas opciones, contrastadas de cara a elegir usar una u otra:

|                        | <b>gtts (Google)</b>           | <b>pyttsx3 (código abierto)</b>     |
|------------------------|--------------------------------|-------------------------------------|
| Uso ilimitado gratuito | No (aunque sea un límite laxo) | Si                                  |
| Puede usarse offline   | No                             | Si                                  |
| Licencia               | Google                         | MPL 2.0, licencia de código abierto |

*Ilustración 4: Tabla comparativa de librerías de texto a voz.*

Aunque, según se observa en la tabla anterior, pytt3x3 parece ser una opción destacada, las voces de gtt3, como cabría esperar de una gran corporación, ofrecen una calidad y claridad considerablemente superior, incluso en su variante más básica. Sin embargo, debido a que pytt3x3 es gratuito sin límites y puede utilizarse localmente, finalmente se ha optado por esta opción.

### 3.6 Ideando la interfaz

Se ha usado QT Designer para agilizar la creación de la interfaz. La aplicación consta de un sistema de *layers* y *frames*, para poder ordenar los elementos en la GUI.

La *GUI* es redimensionable, de manera que los componentes puedan recolocarse en caso de ajustar su tamaño. En toda la interfaz se ha buscado mantener unos mínimos y unos máximos en cuanto a preservar ciertos elementos de cara a que la aplicación sea amigable a nivel de redimensionado de ventana, por ejemplo la barra de escritura de texto y el botón de enviarlo conservan un mínimo de 60 píxeles de alto.

Por otra parte, se buscaba que la aplicación fuese modular respecto al minijuego ideado, buscando que se pudiesen hacer aparte, siendo el Tic Tac Toe el minijuego elegido de muestra, y se puedan integrar con facilidad en el programa.

Para ello se ha ideado que, en realidad, hayan dos interfaces:

- **Interfaz principal.** Cuenta con las capas y frames que hospedan el chat, las opciones, información adicional y reservando un frame para la zona del minijuego.
- **Interfaz del minijuego.** Se insertará en el frame mencionado.

Detallando con mayor profundidad, la **interfaz principal** se divide en varias partes, a las que se referenciarán por zonas:

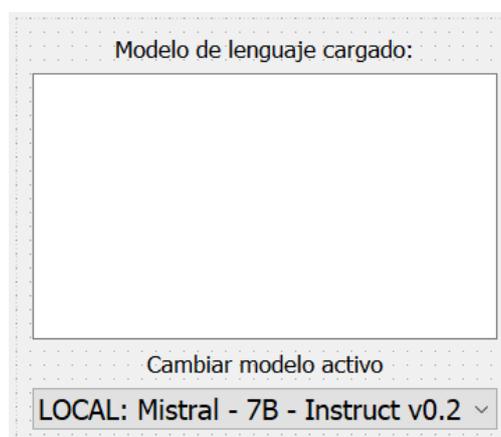


Ilustración 5: Zona de información y elección de LLM

La zona que permite cambiar el modelo de lenguaje y recibir información al respecto, está alojada en un *Qframe* llamado *frameModel*. Hospeda verticalmente el *QtextBrowser* que nos dará información del LLM cargado, el *QcomboBox* desde el que podremos elegir el modelo activo de una manera rápida y dos *Qlabel* descriptivos.



Ilustración 6: Zona (*QFrame*) reservada para el minijuego

Se reservará un *Qframe* que irá en la parte superior derecha de la utilidad para el minijuego escogido, en nuestro caso un Tic Tac Toe.

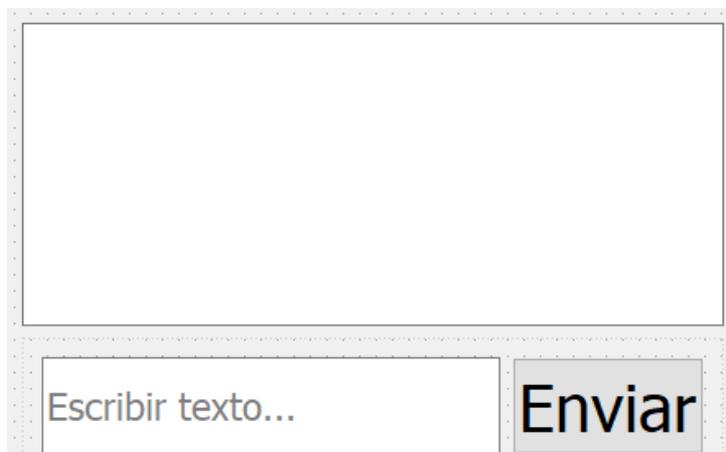
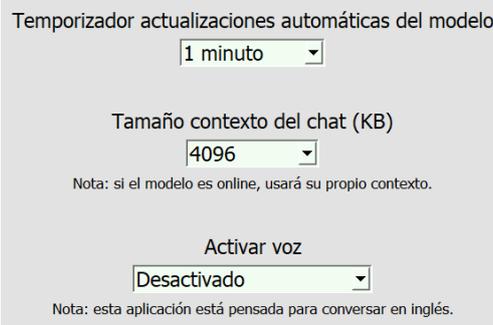


Ilustración 7: Zona inferior, chat

El chat irá contenido en un *Qframe* llamado *frameChat* que ordena items verticalmente. En el mismo, a su vez hay dos apartados:

- El superior hospeda un *QtextBrowser* que corresponderá a las salidas de chat
- El inferior es un nuevo *Qframe* llamado *frameInputText*, que ordena horizontalmente un *QlineEdit* para meter nuestros propios textos para chatear, y el *QPushButton* para enviar esos textos, si bien darle al Entrar del teclado, cumplirá la misma función.

El diseño de la interfaz principal se completa con una segunda ventana con opciones adicionales:



Temporizador actualizaciones automáticas del modelo  
1 minuto

Tamaño contexto del chat (KB)  
4096

Nota: si el modelo es online, usará su propio contexto.

Activar voz  
Desactivado

Nota: esta aplicación está pensada para conversar en inglés.

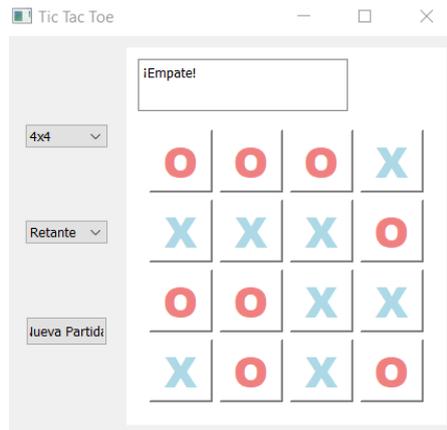
*Ilustración 8: Segunda ventana de la aplicación, opciones.*

Las opciones que se ven en la última figura permiten lo siguiente:

- El temporizador podrá elegir varios valores, desde los 15 segundos a los varios minutos.
- El tamaño de contexto del chat podrá cambiarse entre 1024, 2048, y 4096 KB.
- Se podrá activar el lector de voz.

Se han dispuesto etiquetas descriptoras, los *Qcombobox*, y espaciadores (en azul) para ajustar el tamaño en pantalla de manera dinámica según reajuste de tamaño de ventana.

En cuanto a la interfaz del minijuego, requerirá de su zona de juego propia, y apartados con marcadores y botones propios:



*Ilustración 9: Interfaz antigua del minijuego de ejemplo*

En la ilustración anterior se puede ver una captura de una versión antigua del minijuego que se ha acabado implementando, un Tic Tac Toe. Esta interfaz se insertará en el Qframe que se ha reservado para la misma, en la interfaz principal.

La versión de la captura tuvo muy leves cambios respecto a su versión final. El QTextBrowser se acabó desplazando a la derecha del minijuego, para aprovechar algo mejor el espacio que se acabaría usando en el Qframe, de aspecto más rectangular que cuadrado, de la *app*.

La siguiente captura mostraría la interfaz final, como ha quedado al final del proyecto tras algunos cambios en la parte de desarrollo:

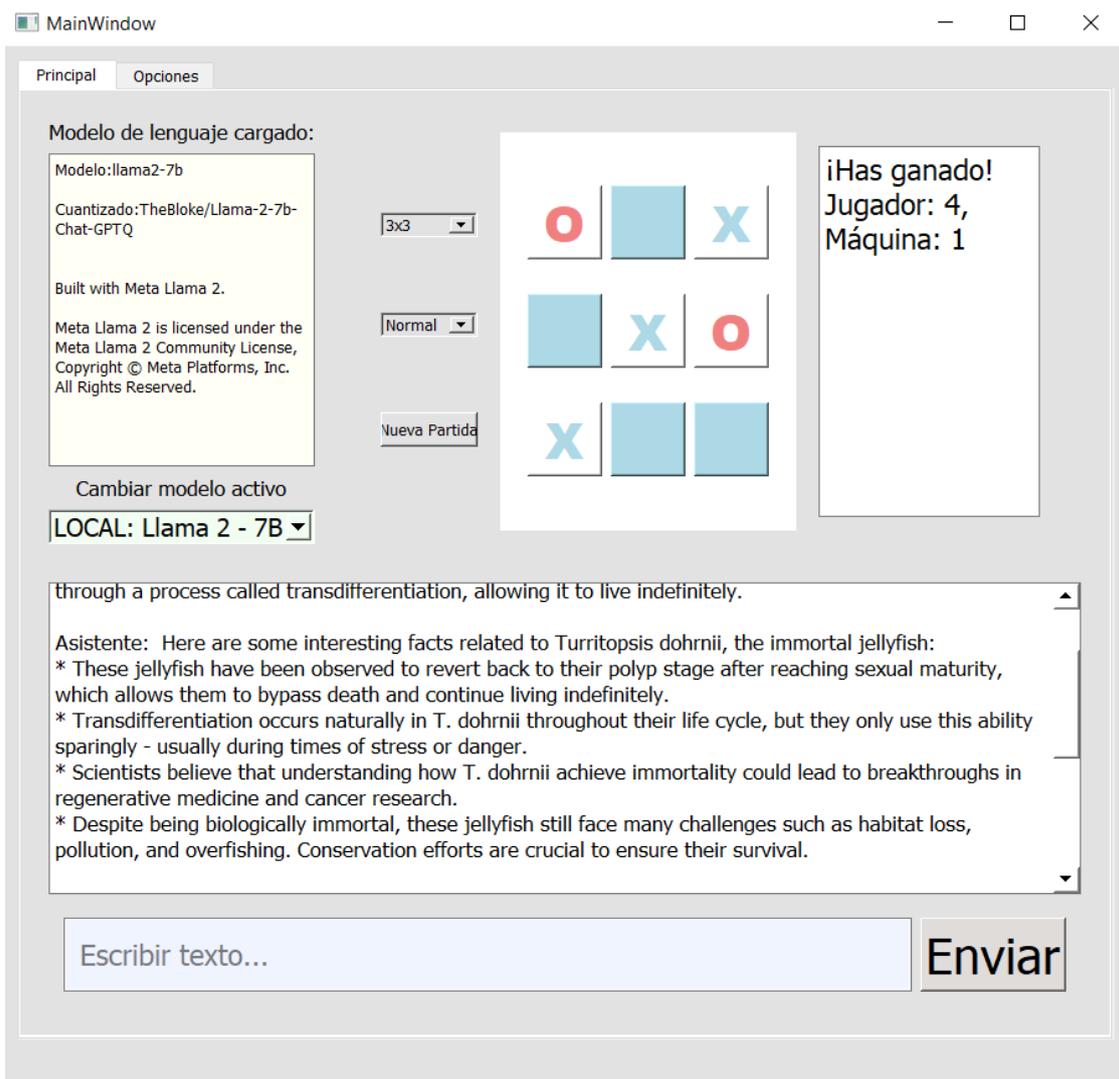


Ilustración 10: Interfaz final, en uso.

En la ilustración anterior se puede ver el resultado de fusionar ambas interfaces. Se pueden ver la distribución de los diferentes *layers* y *frames*. Se apreciaría lo que a grandes rasgos es la interfaz con las tres zonas mencionadas antes, combinadas con la del juego.

### 3.7 Concepción del minijuego

Dado que este proyecto busca integrar una serie de actividades para proporcionar una experiencia conjunta —entretener a una persona y reducir su ansiedad (que pueda ser aumentada por la sensación de soledad) mediante un compañero virtual para jugar y conversar—, no se ha priorizado alcanzar un nivel ideal de experiencia, sino más bien explorar cómo lograrlo. Considerando el tiempo limitado disponible, optamos por un minijuego muy sencillo: el clásico tres en raya, conocido como como *Tic Tac Toe*.

Los retos aquí, por mas pequeño que fuera el juego, eran:

- ¿Qué dificultad podría tener?
- Si por defecto el juego empieza con una dificultad inexistente que use movimientos aleatorios, permisiva con el usuario, ¿por qué no poner una dificultad retante aparte, con su logaritmo propio?
- ¿Y si se hace un tablero 4x4 además del 3x3 que se concibió en inicio?

De estas preguntas han surgido las opciones del juego y el resto de iconos que se pueden ver en el apartado anterior, en la [Ilustración 8](#).

El juego es simple y conocido. En esta versión, el jugador juega contra la máquina y no tiene ni dibujos, usa las letras X y O para las casillas, con diferentes fuentes que se han engrandecido y colores para resaltar el efecto de “fichas”.

Así pues el tablero está hecho y se hizo para empezar una dificultad que se acabó llamando *Normal*, que no fue difícil de implementar dado que básicamente son movimientos aleatorios.

El problema sería la dificultad superior, la *Retante*. Una dificultad real. Ahí se ha consultado la materia estudiada en el grado de computación, Inteligencia Artificial, donde precisamente se comenta el tema de adversarios en juegos <sup>[26]</sup>. También hay un sinfín de *TicTacToes* explicados en internet, para conocer a fondo ideas de cómo elaborar uno, que además están bien explicados y refrescan conocimientos del sistema de poda alfa-beta <sup>[27]</sup>.

Se hablará más sobre el algoritmo Minimax que se ha acabado adoptando para el minijuego, en el capítulo 4 de desarrollo de la aplicación, concretamente en el [apartado](#) destinado al código del Tic Tac Toe.

## 4. Desarrollo de la aplicación

El desarrollo del prototipo se ha llevado a cabo en Python, y en ocasiones ha coincidido con el diseño de la aplicación, solapándose y complementándose en diferentes momentos. Por ejemplo, durante la fase de elección del modelo, se realizaron pruebas con código para evaluar los modelos, especialmente para descartar aquellos que no fueron seleccionados. Del mismo modo, si bien la interfaz se diseñó principalmente en la fase de diseño utilizando QT Designer, también se realizaron numerosos ajustes directamente en el código Python, además de añadidos en CSS para definir ciertos estilos.

En las descripciones de código se ha buscado un lenguaje directo, y las explicaciones pueden acompañarse con un vistazo a los listados de código que pueden leerse en la dirección Github del [Anexo 3](#). Dichos listados incluyen numerosos comentarios descriptivos tanto de funciones como de variables. Por lo tanto, no se comentarán exhaustivamente todas las funciones usadas, sino que se destacarán las más importantes, junto con una explicación general de lo que se ha buscado lograr en las funciones principales. Los fragmentos de código que se pueden ver en esta memoria, extraídos directamente de los listados del programa, se han añadido tal cual aparecen en el editor PyCharm que se ha empleado, respetando su color y fuente original.

A continuación, en los siguientes apartados se detallan las librerías y el código utilizados, junto con la motivación que llevó a su elección. También se abordan otros aspectos generales que han surgido durante la fase de desarrollo, con el fin de ofrecer una visión más completa de los objetivos del proyecto.

### 4.1 Librerías empleadas

Para este proyecto se han importado las siguientes librerías externas:

- **PyQt5**: Utilizada para crear la interfaz de usuario, es necesaria para la implementación de widgets como botones, etiquetas, cuadros de texto, menús, barras de herramientas y más. Además, es la librería empleada por QT Designer, la herramienta GUI utilizada para diseñar la interfaz del proyecto, la cual luego fue ajustada mediante código.
- **time** y **threading**: Empleadas para automatizar el prompt. Si el usuario no está escribiendo, un hilo para preparar el siguiente comentario automatizado del LLM se ejecuta de forma asíncrona durante un tiempo específico, el cual puede ser ajustado por el usuario desde las opciones.
- **re**: Permite trabajar con expresiones regulares. Las respuestas de los modelos de lenguaje grandes (LLMs) a menudo necesitan cierto posprocesado para mejorar su presentación. Esta librería permite, mediante expresiones regulares, utilizar patrones para filtrar ciertas secuencias de caracteres no deseados dentro de cadenas de texto.
- **transformers**: Permite el acceso a una API unificada y coherente para acceder y utilizar diferentes LLM pre-entrenados, como los que se usarán en este trabajo, Llama 2 y Mistral. Desarrollada por Hugging Face, esta

librería está basada en la arquitectura de modelos de aprendizaje profundo conocida como Transformer.

- **cohere**: Utilizada para interactuar con el LLM online Command de Cohere. Esta librería fue desarrollada por la empresa Cohere con el propósito de interactuar con sus modelos y la API de su servidor.
- **pyttsx3**: Librería que permite utilizar las voces del sistema Windows para el modo de texto a voz, empleada para proporcionar un lector de voz al proyecto.
- **multiprocessing**: Combinada con la librería anterior, en este proyecto permite detener el proceso de pronunciación de un mensaje en cuando comienza a pronunciar uno nuevo.
- **random**: Utilizada en el minijuego Tic Tac Toe para la dificultad "normal" por defecto. A través de esta librería, la computadora selecciona casillas con movimientos aleatorios, aunque esta dificultad no es considerada como seria.

Además de estas librerías, se han importado métodos de **sys**, una librería vinculada al sistema, relacionados con el cierre de la aplicación y la creación de una instancia del minijuego. Esta instancia fue utilizada en el desarrollo para probarlo de manera independiente antes de su integración con la interfaz principal del proyecto.

## 4.2 Implementando los LLM y el chat con memoria

A continuación se comentará como se han implementado los tres LLM elegidos, siendo dos de ellos locales y uno online, y cómo la aplicación gestiona la conversación con el LLM activo. El código descrito en este apartado 4.2, está contenido en el [archivo `LLM\_manager.py`](#).

Como se ha comentado en la fase de diseño, para los dos modelos locales que se usarian en el proyecto se eligieron formatos cuantizados en GPTQ. Para cargar dichos modelos se han usado clases de transformers. **AutoModelForCausalLM** permite cargar los modelos pre-entrenados del proyecto.

Ejemplo en código con la configuración de cargado de ambos modelos:

```
llm.myModel = AutoModelForCausalLM.from_pretrained(config.model_name_or_path,
                                                    device_map="auto", trust_remote_code=False, revision="main")
```

Tras ello se necesitará cargar el tokenizador, mediante la clase **AutoTokenizer**., que con el modelo cargado previamente permitirá procesar texto de entrada y prepararlo para la inferencia del modelo:

```
llm.myTokenizer = AutoTokenizer.from_pretrained(config.model_name_or_path, use_fast=True)
```

El proceso de cargado de modelos locales GPTQ mostrado arriba, cambia bastante con el modelo online. En el caso del modelo online escogido, Command de la empresa Cohere, el proceso es mucho más simple: solo se necesita llamar una función de la librería de la API de Cohere [28], en conjunto con la key de la API que Cohere nos haya proporcionado.

```
llm.onlineModel = cohere.Client(cohere_key)
```

Tras la carga del modelo y el tokenizador, se necesita crear una vía para el proceso de textos con el LLM. Dos maneras de llevarlo a cabo, serían estas:

- Usando la *función pipeline* de transformers, de generación de texto.
- Usando un tensor de PyTorch.

Se ha usado la *opción a* para cargar el modelo Llama 2, y la *opción b* para cargar el modelo Mistral, como se vería en la siguiente tabla:

```
pipe = pipeline(  
    "text-generation",  
    model=llm.myModel,  
    tokenizer=llm.myTokenizer,  
    max_new_tokens=256,  
    do_sample=True,  
    temperature=0.7,  
    top_p=0.95,  
    top_k=40,  
    repetition_penalty=1.28  
)  
generated_response =  
pipe(prompt_template)[0]['generated_text']
```

```
input_ids = llm.myTo-  
kenizer(prompt_template, re-  
turn_tensors='pt').in-  
put_ids.cuda()  
output = llm.myModel.generate(  
    inputs=input_ids,  
    temperature=0.7,  
    do_sample=True,  
    top_p=0.95,  
    top_k=40,  
    max_new_tokens=256,  
    repetition_penalty=1.1  
)  
generated_response = llm.myTo-  
kenizer.decode(output[0])
```

Ilustración 11: Tabla con dos sistemas de inferencia.

Aunque en la tabla de arriba no puede apreciarse, el listado izquierdo ejecutará el modelo Llama 2, y el derecho, Mistral. El programa llevará a una u otra sección de código, según el modelo local que conste en la variable *myModel* (valor cambiable desde el comboBox que lista los tres modelos posibles).

En el listado izquierdo se puede apreciar que Llama 2 tiene un valor más alto de *repetition\_penalty* que Mistral, dado que se apreció una mayor repetitividad en tokens ya mostrados en la conversación. Mas información sobre esto, en la [sección de Ajustes](#).

Las versiones cuantizadas de Llama 2 y Mistral que se usan en el proyecto, tienen un despliegue inicial similar [29], y de hecho podría haberse usado el mismo sistema de inferencia para ambas. Se decidió abordar ambos sistemas en aras de ver diferencias en la ejecución (no se apreciaron), pero se podría haber optimizado el código de usar únicamente uno de los sistemas.

Si desde el combobox de la aplicación se quiere cambiar de modelo activo, desde la interfaz se hará una llamada al método que gestiona los modelos y se librarán las variables que contienen información del modelo previo, además de resetear el temporizador del modo automático para dar tiempo al programa antes de iniciar el nuevo modelo escogido.

El modelo online Command de Cohere hará su inferencia a nivel del servidor de cohere, y se usará la función `chat_stream` de su API para crear el flujo de datos. Esta función permite tratar por separado varios aspectos de la información que le se le envía al servidor, incluyendo el prompt:

```
stream = llm.onlineModel.chat_stream(message=prompt_template,
                                     model='command',
                                     preamble="You are a friendly en-
                                     tity. Try to answer briefly with less than 50 "
                                     + "words, and never ex-
                                     ceed 100 words in your reply.",
                                     chat_history=llm.chat_log
                                     )
chatbot_response = ""
for event in stream:
    if event.event_type == "text-generation":
        print(event.text, end='')
        chatbot_response += event.text
```

En el código anterior se puede ver que al modelo se le proporciona un prompt, el LLM que se quiere usar dado que la API de Cohere permite acceder a varios LLM diferentes [30], y, esto se comentará a fondo luego, un contenedor List (variable `chat_log`) con los mensajes surgidos entre usuario y máquina.

El sistema de manejo de la conversación, conocido por su término técnico en inglés log, se trata de manera muy similar tanto en los modelos online como en los modelos offline de este proyecto. La diferencia principal radica en el tipo de variable que se utiliza para gestionarlo. En el caso del modelo online, como se ha mencionado anteriormente, se emplea un List. Por otro lado, en el caso de los modelos locales, es necesaria una conversión a un string, y convertir el List en una larga cadena de texto que constituye el log a tratar.

Acerca del log, los modelos de lenguaje actuales suelen recomendar rellenarlo un poco, a modo de ejemplo. En el listado siguiente se aporta parte de los ejemplos que se han usado, para orientar al LLM cuando siga con la conversación:

```
llm.chat_log = [{"role": config.name_user, "message": "Tell me some-
               thing about birds."},
               {"role": config.name_bot, "message": "Flamingos are
               birds, and a group of flamingos is called a "
               + "flamboyance,"
               + "fittingly re-
               flects their vibrant pink color and unique "
               + "appearance."},
```

La respuesta mostrada de ejemplo, va en consonancia con la personalidad de muestra que se otorga al chatbot en este proyecto, de una entidad asistente. Aún

así las preguntas y respuestas de ejemplo pueden necesitar ciertos ajustes para orientar al modelo mejor hacia lo que queremos, y el proceso puede ser arduo dependiendo de la exigencia de la persona. Mas información al respecto en la [sección de ajustes](#).

Volviendo a la diferencia en el tratamiento de los datos, los modelos offline usados tratarán el prompt de manera muy diferente a Cohere. En concreto, se genera un gran prompt donde interviene tanto el log generado de la conversación, la descripción de la actitud que ha de tomar el modelo de lenguaje con respecto a la información que le estamos dando, y parámetros de configuración del prompt, específicos de cada modelo. Este sería el prompt usado con Mistral:

```
prompt_mistral = (  
    "[INST]" + "You are a friendly entity called " + con-  
fig.name_bot  
    + ". The following is part of a conversation between you and "  
+ config.name_user  
    + ". Write a response that appropriately completes the request  
made after the string <<USER_PROMPT>>."  
    + "You will try to answer briefly with less than 50 words, and  
you will never exceed the 100 words in "  
    + "your answer. \n\n"  
    + log_string + "\n<<USER_PROMPT>>:"  
    + prompt + "[/INST]"  
)
```

En el listado de arriba se pueden apreciar las diferentes partes, de lo que puede llegar a ser una string considerablemente larga dirigida al LLM para que la interprete:

- Etiquetas INST que rodean el prompt, requeridas por este tipo de modelos.
- Explicación del rol a tomar por el modelo: entidad amistosa, que responde apropiadamente, con instrucciones al respecto.
- Aclaración de cómo se va a proporcionar el log, la subcadena resultante de convertir el List de mensajes hasta entonces.

Llama 2 gestiona el prompt de manera extremadamente similar, aunque mediante el uso adicional de las etiquetas <<SYS>><</SYS>> se permite separar mejor lo que se quiere pedir al modelo a modo de órdenes del sistema, y separar de manera más obvia el string de la conversación.

## Post-procesado de las respuestas

Respecto a las respuestas recibidas de estos modelos, es frecuente que salgan caracteres ascii extra, como marcas procedentes de lenguaje HTML o XML, muy posiblemente debido a las diversas fuentes de datos con que han sido instruidos estos modelos, o etiquetas usadas por el propio LLM que no debería dar en una respuesta. Debido a ello, si se busca una experiencia de chat algo más cómoda desde una interfaz, es recomendable hacer un post-procesado del texto de respuesta. Consecuentemente se han metido funciones para controlar un poco

las salidas, usando reglas y patrones con expresiones regulares para poder “limpiar” el texto.

Una regla de ejemplo usada sería la usada en la siguiente línea de código:

```
pattern = re.compile(r'</?s>')
```

Esta regla, que usaremos mediante la función *compile* de la biblioteca *re*, permite quitar `<s>` y `</s>`, dado que se contempló que el LLM a veces las incorporaba al texto de respuesta. Dichas etiquetas son convenciones usadas en la tokenización de texto, usables por modelos de lenguaje, pero no deberían de salir en las respuestas del modelo, y aún menos de manera aleatoria. Se han introducido además otras reglas en el listado de código, en base a algunos caracteres adicionales que se vieron en las salidas de los LLM, con ánimo de hacer mas presentables las respuestas.

## El modo automático

Dado que el proyecto se inspira en la interacción entre dos personas mientras juegan y conversan, se ha implementado un sistema automatizado utilizando las librerías *time* y *threading*. Cuando el usuario no está escribiendo en la caja de texto del chat, el sistema activa un contador según la selección de tiempo realizada en un combobox disponible en las opciones. Este contador funciona en segundo plano mediante un hilo que continúa mientras el programa sigue siendo accesible para el usuario:

```
thread = threading.Thread(target=run_timer, args=(interfaz,))
```

El contador se reduce en intervalos de 5 segundos (las opciones de tiempo para la automatización son múltiplos de 5, comenzando desde 15 segundos en el intervalo más bajo). Si el usuario no comienza a escribir (es decir, si no teclea en la caja de texto), el contador no se reinicia. Cuando el contador alcanza cero, se activa una función que, de alguna manera, simula una interacción del usuario para generar la automatización. Esto permite que la conversación continúe, ya que el contador se restablece después de enviar el comentario automatizado al LLM y recibir su respuesta.

La aplicación discrimina entre los siguientes modos:

- **Inicio.** La aplicación pregunta *Tell me some random fact about anything*, al abrir el programa.
- **Conversación manual.** La aplicación envía el comentario del usuario, hecho desde la interfaz, al LLM.
- **Modo automatizado.** Cuando el temporizador comentado baje a cero, la aplicación preguntará *Tell me more about that* para hacer seguir hablando al chatbot, sobre la temática en curso.

Como se puede ver más adelante en la [sección de pruebas](#) no es un sistema perfecto y varía mucho del LLM elegido, y de los ajustes hechos, pero permite

imitar hasta cierto punto el efecto de charla en el rato que se combine con el juego.

### Límite de tokens

En la interfaz se ha habilitado una opción para alternar entre tres valores de contexto: 1024, 2048 y 4096 tokens. Una función recoge el límite que se imponga y, mediante un cálculo estimado, si el límite está cerca de rebosar con el registro acumulado hasta entonces, se eliminará el mensaje más antiguo del listado, es decir, el que conste en la posición 0. Podría darse el caso de que el último mensaje fuese especialmente largo, por lo que el cálculo estimado cuenta con un margen de 200 tokens. Si los primeros mensajes fuesen especialmente cortos y el último mensaje es especialmente largo, la función se repetiría una y otra vez hasta que el registro ocupe un tamaño adecuado en el contexto.

La opción de poder cambiar el límite del contexto otorga cierto poder al usuario, permitiéndole elegir entre eficiencia, utilizando un contexto menor y, por lo tanto, un menor consumo de recursos computacionales del modelo, y la posibilidad de dar una memoria aumentada al modelo para la conversación si se utiliza un contexto mayor.

**IMPORTANTE:** Se entregó una versión del código con un fallo de planteamiento, referente al prompt realizado a Llama 2. No interrumpirá la ejecución, pero puede hacerlo con la inmersión del usuario en la experiencia. [Mas información, en la sección de Pruebas.](#)

### 4.3 Desarrollando el minijuego de ejemplo: Tic Tac Toe

En la fase de diseño, se ha realizado y descrito la creación de la interfaz para el minijuego que acompaña al prototipo, un sencillo Tic Tac Toe. El código del que se habla en este capítulo 4.3 está en el [archivo del minijuego ejemplo, \*TicTacToe\\_minigame.py\*](#).

Un minijuego que quiera sustituir el minijuego creado de ejemplo por otro, necesitará lo siguiente:

- Tener su propia interfaz.
- Ofrecer su propio sistema de configuración (botones y menús) en caso de necesitarlo.
- Ser independiente del resto del programa en términos de opciones

En las fases iniciales del proyecto, se pensó en hacer jugar al modelo de lenguaje que estuviese activo en el prototipo pidiéndole movimientos a usar en el minijuego. Sin embargo, como se ha comentado antes en esta memoria, la idea se ha descartado por completo. Los modelos de lenguaje actuales no fueron concebidos originalmente para problemas de lógica y matemática complejos y su funcionamiento al respecto no es eficaz [31] [32], con lo que en el momento en el que se ha escrito este documento, usar un LLM para manejar un minijuego

que tenga que hacer numerosos cálculos, no es viable por no estar optimizado para ello.

Hay que hacer entonces un acercamiento modular, con lo que el minijuego tendrá su propio sistema de lógica, mientras el LLM funcionará únicamente para el chat, para que al final ambos ofrezcan la experiencia que el prototipo busca crear. Se comentarán las principales partes del código implementado para el Tic Tac Toe como curiosidad, aunque en Internet como se ha comentado antes en este trabajo hay muchísimos listados de referencia para este tipo de juegos [27].

Antes de continuar, es necesario recordar las diferentes partes que integran la creación del minijuego añadido. Se ha hablado de la interfaz propia del minijuego, y además estarían los siguientes aspectos a comentar:

- **Tamaño del tablero:** Se permite 3x3 y 4x4, por opciones.
- **Las dos dificultades:** Ofrecidas por opciones, Normal y Retante, que condicionarán los movimientos de la computadora cuando sea su turno.

Empezando por el **tablero**, se ha tenido que dibujar enteramente por código, dado que puede cambiar dinámicamente en tamaño de casillas. Cada vez que se cree una partida nueva, se borrará el tablero anterior, creando uno nuevo. El tablero o parrilla (*grid*) se dibuja creando una cuadrícula de botones:

```
grid_layout = QVBoxLayout()
```

Los botones se añaden dinámicamente según el tamaño seleccionado (3x3 o 4x4), mediante un bucle doble para las filas (*rows*) y columnas (*columns*). Se establece el tamaño y color inicial de los botones que serán las casillas del tablero.

Las casillas responden a los clics del usuario para permitir la interacción y el juego, mediante eventos *button\_clicked* se ha manejado el clic del usuario, pasando las coordenadas de la casilla clicada como en la siguiente línea:

```
button.clicked.connect(lambda _, x=i, y=j: self.button_clicked(x, y))
```

Creando la función sin nombre lambda se pueden capturar las coordenadas en el momento que el botón se crea, de manera que cada 'clic' recoja las coordenadas correctas de la casilla que representa.

Cada vez que se haga clic en una casilla, el programa actualiza la interfaz visualmente cambiando la letra (X, O) y color del botón para reflejar la jugada del jugador.

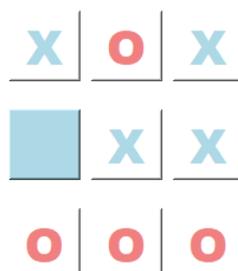


Ilustración 12: Variante 3x3 del minijuego. La máquina, "O", gana.

En cuanto a la **dificultad** en el modo Normal, el programa meramente usa la librería *random* para elegir movimiento, como se puede ver en la última línea del siguiente código:

```
movimientos_posibles = [(i, j) for i in range(self.board_size) for j in
range(self.board_size) if self.board[i][j] is None]
(...)
x, y = random.choice(movimientos_posibles)
```

Se puede deducir de la porción de código de arriba que el programa ofrecerá una dificultad extremadamente escasa, dado que no razonará el siguiente movimiento más allá de elegir una casilla aleatoria de entre las casillas restantes.

La dificultad Retante, por otra parte, usará un algoritmo que ofrezca un reto real. Para ello se ha optado por un algoritmo minimax, con una poda alfa-beta. Se ha hablado superficialmente de ello en [la parte de diseño](#), y ahora se mirará con mas detenimiento.

El algoritmo minimax permite al programa evaluar los movimientos posibles del juego, alternando entre unos valores máximo y mínimo. En este minijuego, busca encontrar una jugada óptima para el jugador rival, con la idea de poderse anticipar a su movimiento. Se ha creado una función llamada *minimax* que va recorriendo el tablero mientras itera sobre todas las posiciones posibles.

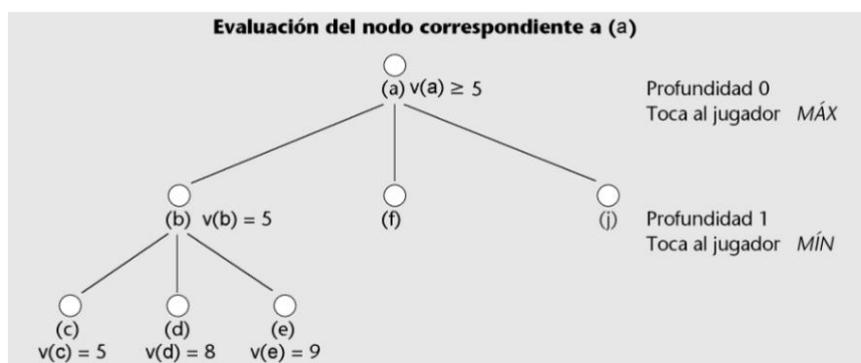


Ilustración 13: Ejemplo de evaluación de nodos mediante poda. Fuente:UOC.

De manera similar al ejemplo de la captura anterior extraída de la materia de Inteligencia Artificial de la carrera [26], en la poda alfa-beta usada en este minijuego se depende de la estructura de alternancia de maximización y minimización de jugadores que caracteriza al algoritmo Minimax y se enfoca en la alternancia de jugadores maximizador (en este programa, "O", el turno correspondiente a la máquina) y minimizador (en este programa, "X", el turno correspondiente al jugador). Se van actualizando unos valores alfa y beta que se actualizan conforme se exploran los nodos que evalúa Minimax, y con el sistema de poda se descartan subárboles.

Dado que la posibilidad de cambiar el tamaño del tablero aumenta mucho la diferencia en posibilidades de jugadas de un tipo de tablero al otro, siendo un tablero de 3x3 frente a otro de 4x4, se ha ajustado la profundidad de la búsqueda en el algoritmo Minimax con poda alfa-beta y de determinar el mejor movimiento para la IA, en el modo Retante:

```
self.poda_profundidad = min(2 + self.turn_count // 2, 9) if
self.board_size == 3 else min(2 + self.turn_count // 2, 6)
```

El código anterior permite ver que una de las opciones alcanza la profundidad 9 (el 3x3) y la otra, una profundidad de 6 (el 4x4). Esto se ha hecho así porque incluso en ordenadores actuales un algoritmo de este tipo que calcule todas las probabilidades del árbol de decisiones que puede darse en un tablero 4x4 sin ningún tipo de optimización, puede llegar a ser extremadamente demandante, y consecuentemente lento de cara al turno de la máquina.

Debido a ello se ha adoptado también un manejo del algoritmo que incremente sus cálculos conforme avance la partida, que no sea demasiado demandante desde el principio. Cada turno que transcurra se obtendrá un valor con la profundidad base de 2, más un incremento proporcional al número de turnos jugados.

En el siguiente código se llamará a la función con el algoritmo minimax, que tasará el rendimiento de lo comentado hasta ahora, de ahí que haya habido que optimizar un poco:

```
x, y = self.minimax(self.board, self.poda_profundidad, -float('inf'),
float('inf'), True)[1]
```

En el código de arriba se envían los mejores valores que se pueden dar al jugador maximizador, la máquina (-infinito) y el minimizador, el usuario (+infinito).

Se hablará ahora de cómo se ha tratado estos valores en el algoritmo minimax, con poda alfa beta. Empezando con como se trataría la condición de **maximización**:

```
max_eval = -float('inf')
mejor_movimiento = None
for i in range(self.board_size):
    for j in range(self.board_size):
        if board[i][j] is None:
            board[i][j] = "O"
            eval = self.minimax(board, depth - 1, alpha, beta, False)[0]
            board[i][j] = None
            if eval > max_eval:
                max_eval = eval
                mejor_movimiento = (i, j)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
return max_eval, mejor_movimiento
```

Se puede ver que se inicializa *max\_eval* con un valor muy bajo, de infinito negativo, en aras de maximizar puntaje, y se busca el movimiento que maximice eval. Se actualiza alpha con el valor máximo encontrado. Sería una evaluación de los movimientos posibles para el jugador maximizador, "O", la máquina.

La segunda mitad del código, compara beta con alpha. Si alpha fuese mayor que beta, implicaría que se ha encontrado un movimiento que no será elegido por el jugador minimizador, "X", el usuario y se interrumpe la exploración de nodos.

El siguiente código, es el equivalente de la **minimización**:

```
min_eval = float('inf')
mejor_movimiento = None
for i in range(self.board_size):
    for j in range(self.board_size):
        if board[i][j] is None:
            board[i][j] = "X"
            eval = self.minimax(board, depth - 1, alpha, beta, True)[0]
            board[i][j] = None
            if eval < min_eval:
                min_eval = eval
                mejor_movimiento = (i, j)
            beta = min(beta, eval)
            if beta <= alpha:
                break
return min_eval, mejor_movimiento
```

La estructura es similar a la de maximización, pero aplicada ahora al jugador minimizador, "X", el usuario. Consecuentemente, se inicializa min\_eval con un valor muy alto (positivo infinito) para minimizar el puntaje, y se busca el movimiento que minimice eval. Se actualiza beta con el valor mínimo encontrado.

En la segunda mitad de este código también se compara beta con alpha, pero aquí si alpha es mayor que beta, se interrumpirá la exploración de nodos porque en este caso se ha encontrado un movimiento que no será elegido por el jugador maximizador, "O".

Así pues, la función minimax va asegurándose de que no se evalúen ramas innecesarias del árbol de juego, buscando algo de eficiencia del algoritmo. Para esto, se van manteniendo y actualizando los valores de alpha y beta, y se dejan de evaluar las ramas que no pueden afectar el resultado final, viendo el mejor y el peor resultado posible para la máquina y el jugador.

Encontrar cómo optimizar este algoritmo, sobretodo en lo concerniente a niveles de profundidad, ha implicado algo de ensayo y error. Mas información de ello en la sección de [ajustes](#).

#### 4.4 Cambios en la interfaz principal

Si bien la interfaz se ha diseñado casi en su totalidad con QT Designer, se han implementado los siguientes tipos de métodos en su clase principal, que está en el [archivo `main\_interface.py`](#):

- Gestión de iconos de la interfaz: Esto incluye la gestión de elementos como los combobox y la transferencia de información entre la interfaz y los métodos que gestionan el modelo activo, como el cambio de modelo en caso de pedirse desde su opción.
- Gestión del enfoque en el `lineEditInput`: Si el usuario escribe, se reiniciará la cuenta atrás que controla el modo automatizado del programa con el chatbot

También se incorporó al frame reservado para ello, el minijuego de ejemplo. El [resultado](#) se ha podido ver al final de la sección de diseño de la Interfaz.

#### 4.5 Añadiendo Texto a voz

Se han insertado funciones para añadir un lector de texto a voz, en el [archivo `utils.py`](#). Como se comentó en la [fase 3.5](#) de diseño del prototipo, se ha elegido la librería `pyttsx3` frente a la alternativa online que ofrece Google pese a su calidad algo menor, pero menos dependiente de internet y sin límite de uso

La librería `pyttsx3` recoge las voces que tiene nuestro propio sistema de windows. En el caso concreto del ordenador en el que se ha diseñado el prototipo, hay dos a elegir que se pueden ver desde el archivo de configuración del sistema de Microsoft:

ID ZIRA, EN INGLÉS:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS_MS_EN-US_ZIRA_11.0"
```

ID HELENA, EN ESPAÑOL:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS_MS_ES-ES_HELENA_11.0"
```

Aunque el chat de este proyecto está principalmente enfocado al uso del inglés, se decidió incorporar ambas voces y dar la opción de lector en español de España usando Helena, y en inglés de EE.UU. usando Zira, elegibles desde un combobox en Opciones. Un aviso en forma de label bajo el combobox de la aplicación recuerda que la voz en español se añade solo a modo de prueba. En la configuración interna, mediante `engine.setProperty('rate', valor)`, se ha dado una velocidad algo superior a la voz en castellano para adecuarla más a una dicción informal.

Es importante recordar de nuevo de todos modos, que el chat del proyecto se ha enfocado a conversaciones en inglés. Se podría hacer que el LLM converse un poco en castellano, pero ni el prompt ni los ejemplos se han preparado para ello. Los motivos se han comentado antes: los LLM actuales locales, aún tienen problemas para expresarse como una persona hispanohablante autóctona.

Por otro lado se ha buscado un sistema para que el lector de voz pueda interrumpirse a sí mismo antes de leer una respuesta nueva del LLM:

```
if current_speak_process and current_speak_process.is_alive():
    current_speak_process.terminate()
    current_speak_process.join()
(...)
current_speak_process = multiprocessing.Process(target=english_reader,
args=(text, voice_id))
```

Como puede verse en el trozo de código anterior, se ha optado por gestionar un proceso adicional mediante la librería multiprocessing, para poder detener el mensaje que se estaba pronunciando hasta entonces en caso de que haya que pronunciar un nuevo mensaje. Interrumpiendo el proceso creado para usar la voz con la respuesta previa, se creará un nuevo proceso para pronunciar la nueva respuesta.

Esto es debido a que en ocasiones, el sistema podría tener que pronunciar un mensaje cuando el anterior aún se está pronunciando. Por ejemplo:

- El modo automatizado tiene el temporizador demasiado bajo (en el combobox proporcionado, el lapso menor es de 15 segundos, principalmente para pruebas) y el sistema ha solicitado una respuesta al LLM antes de que la librería de voz termine de pronunciar el mensaje anterior.
- El usuario no ha esperado a que el sistema termine de pronunciar el mensaje previo y ha comentado algo nuevo al LLM.

Ambos casos se asemejan a situaciones de la interacción humana, imitando situaciones reales donde un interlocutor puede hablar de manera atropellada, exponiendo un punto de vista nuevo antes de acabar el comentario anterior, o una persona que interrumpe a otra mientras está hablando.

## 5. Pruebas y ajustes

Los siguientes apartados consisten en algunas pruebas de muestra en formato audiovisual y en posibles ajustes en el programa que no se comentaron en la fase de desarrollo, con el objetivo de afinar mejor los resultados.

### 5.1 Pruebas

A lo largo del proyecto se han realizado numerosas pruebas con los LLM utilizados, principalmente en su configuración inicial y con una versión preliminar de la interfaz, que luego se hicieron con la interfaz ya concluida. A continuación, se muestran capturas de una de las muchas sesiones de prueba con la interfaz usada en el prototipo.

Además, se incluyen comentarios y autocrítica antes del apartado de conclusiones para aprovechar el contexto añadido al hablar sobre las capturas, ya que la integración de varias funciones e interacción con varios LLM puede dar lugar a situaciones interesantes. Este apartado tendrá sus propias conclusiones.

Para proveer cierta continuidad en esta sesión de prueba, se mostrará una secuencia de situaciones. Se ha añadido un ligero efecto de sombreado en las capturas en esta sección, para ayudar a la legibilidad.

- 1) *Iniciando el programa y carga de elementos inicial. El LLM de inicio por defecto, Mistral 7B, responde al prompt automatizado introductorio.*

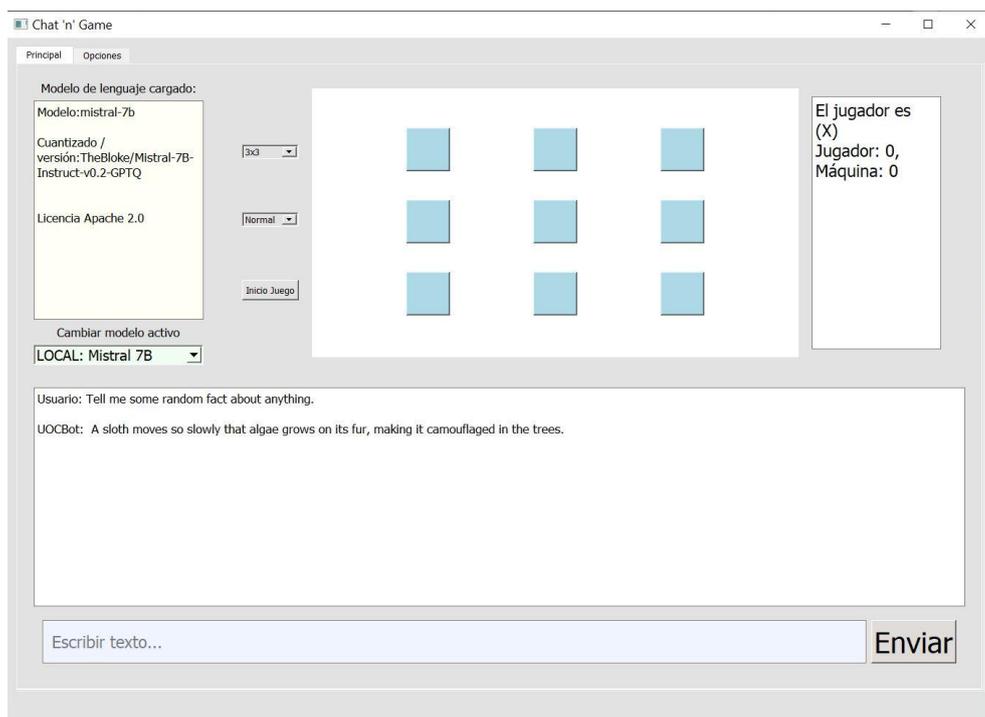


Ilustración 14: Inicio del prototipo.

En la captura mostrada anteriormente, el prototipo se inicia en pantalla completa, cargando una interfaz que incluye los siguientes elementos:

- Información del LLM cargado en la ventana de información, en este caso el que inicia por defecto, Mistral 7B, un LLM localizado en la unidad de almacenamiento.
- La primera interacción con el modelo, siempre automatizada. Aquí se podría haber eliminado la cadena "Usuario" en la introducción y, por ejemplo, haber adornado la frase introductoria con otro color o formato para acen-tuar que es el inicio de la sesión. Por otro lado y a nivel interno, es recomendable que el LLM entienda que está hablando con una única persona para evitar confusiones.
- La sub-interfaz del minijuego, que incluye el juego, sus controles y mar-cadores.

## 2) Inicio del modo monólogo con Mistral 7B, y redimensionado de ventana.

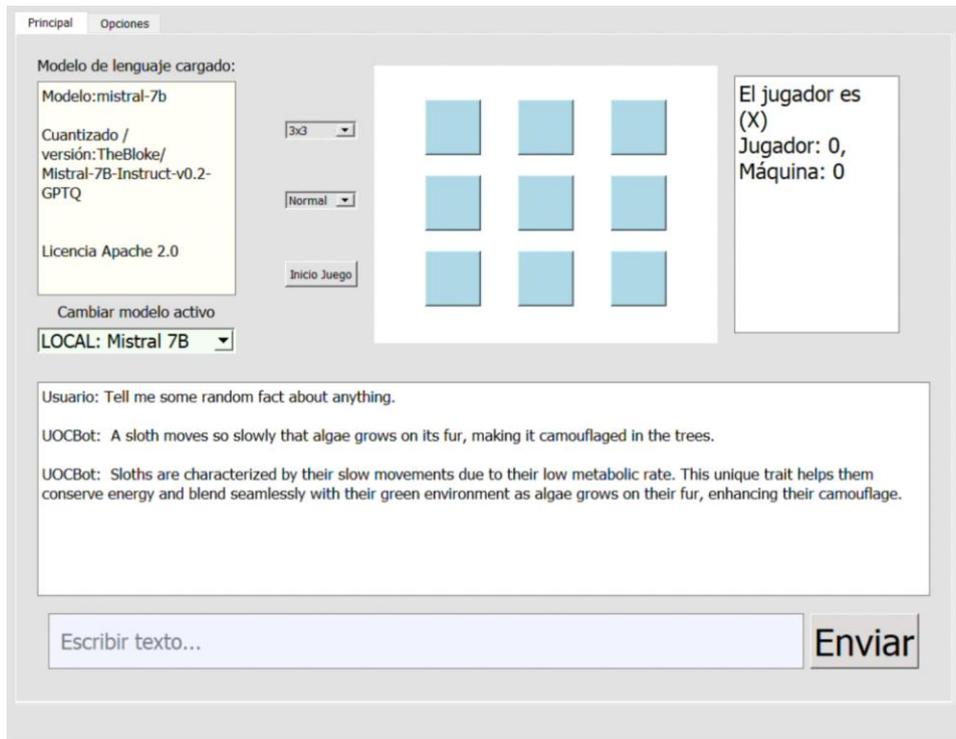


Ilustración 15: El LLM sigue hablando.

En la imagen anterior, tras unos segundos (1 minuto, el tiempo inicial por defecto) el programa ha pedido al bot que siga hablando. Estos prompts automatizados, a diferencia del introductorio, no se incluyen en la ventana de chat por tema de fluidez en la comprensión del diálogo, de cara al usuario. El bot sigue hablando, imitando la circunstancia una persona que sigue enfocada en lo que está hablando y a quien le ha venido algo nuevo que decir a la cabeza. En este caso, el LLM sigue conversando acerca de la temática aleatoria escogida, el animal perezoso (*sloth*).

Como se ve en la pantalla, el usuario puede redimensionar la ventana y con ello los elementos mostrados se recolocan, respetando los límites en ciertos valores que mediante QT Designer se han hecho en el código.

### 3) Jugando con el minijuego.

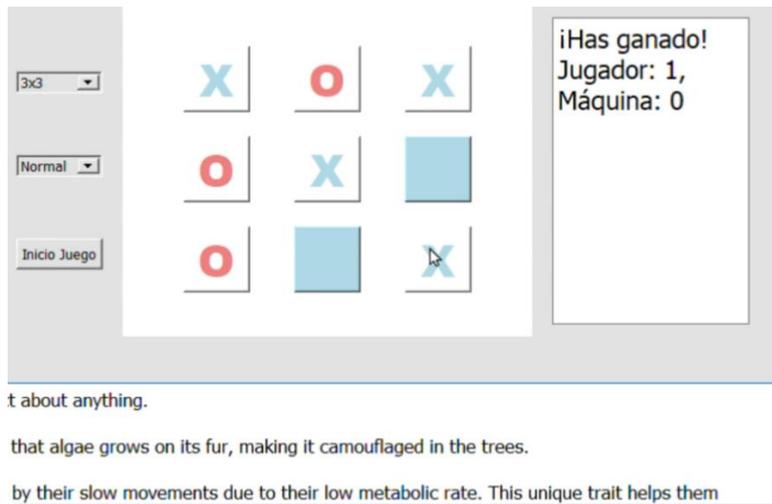


Ilustración 16: Jugando el minijuego mientras el LLM sigue hablando.

Se incluirán capturas parciales a partir de aquí<sup>9</sup>. En la captura parcial mostrada, el usuario ha jugado una partida con el Tic Tac Toe utilizado como ejemplo en este proyecto, con los parámetros por defecto (3x3, dificultad Normal). El minijuego es usable de manera asíncrona respecto al chat, el cual está a la espera de un comentario del usuario, o de que acabe el temporizador y se envíe un comentario automatizado para que el LLM siga hablando.

### 4) Sobre la Dificultad Normal por defecto.

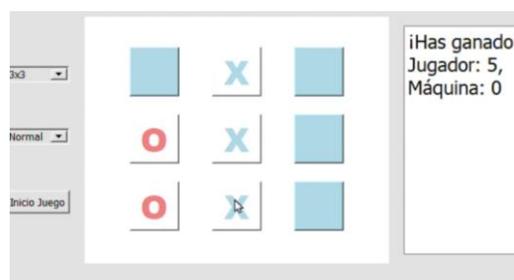
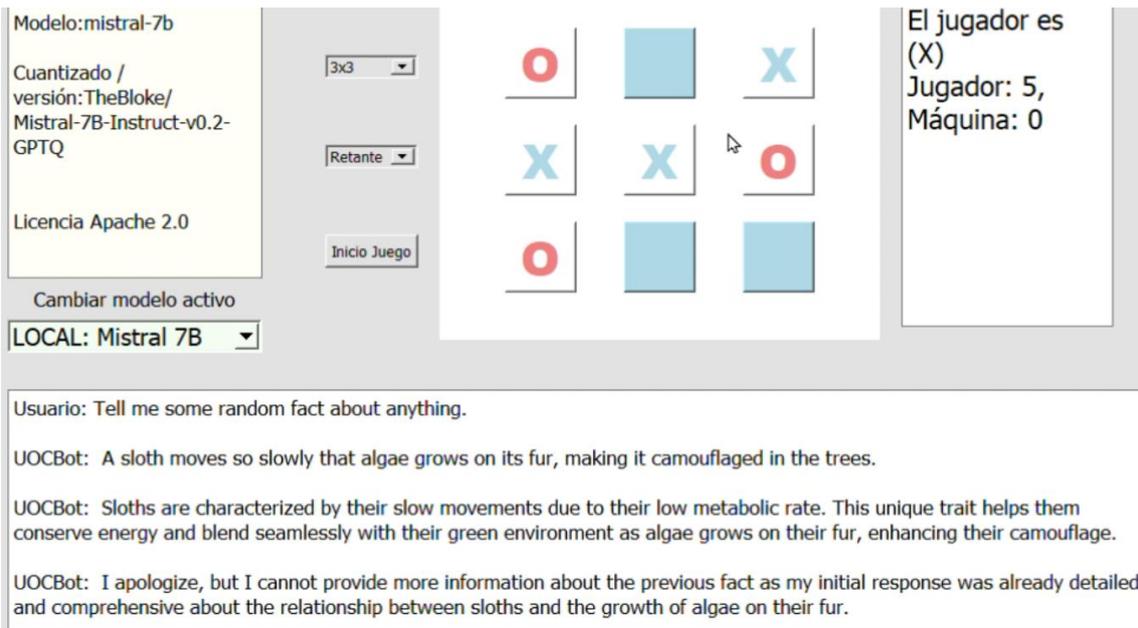


Ilustración 17: Se logra un nuevo tres en raya en Normal.

<sup>9</sup> Para economizar espacio y mejorar la visibilidad en esta sección de la memoria, se incluirán capturas parciales en vez de completas a partir de este punto, una vez se ha visto como es la interfaz general. Se proveerán capturas mas completas de la aplicación, cuando la explicación lo requier.

Se pretendía que la dificultad Normal por defecto no fuese retante, y como puede verse en la captura anterior el sistema de movimientos aleatorios usados en la dificultad Normal de la máquina no ha podido contrarrestar una cadena de movimientos muy obvios por parte del jugador. En breve se verán los otros modos de juego.

5) *Dificultad retante del minijuego, y el chat añade una nueva respuesta de Mistral al automatismo del programa.*



*Ilustración 18: Continúa el modo automatizado de charla mientras se sigue jugando.*

En la imagen precedente se puede empezar a apreciar dos situaciones nuevas en las tareas asíncronas del prototipo:

- En la dificultad Retante, la máquina ha bloqueado el intento del usuario de hacer un tres en raya. En Retante, los movimientos ya no son aleatorios, se usa IA mediante el algoritmo Minimax con poda alfa-beta incluido en el programa.
- Mientras el usuario ha seguido jugando, el programa hace un segundo ciclo de su modo automático y muestra la nueva respuesta del LLM. Nótese que el modelo responde de manera un tanto gruñona esta vez. La “personalidad” del modelo se podría entrenar, ajustar y mejorar si el proyecto hubiese durado mas tiempo, algo que se comentará a fondo después en esta memoria.

Es necesario hacer aquí un recordatorio de que se puede activar la función de voz integrada para que al cambiar el foco de atención entre chat y videojuego se sienta mucho menos disruptivo. [Al final de esta sección](#) se adjunta un video de esta sesión de pruebas de muestra, en el cual se puede escuchar el lector.

Volviendo a la última captura, se puede ver que el usuario, en todo lo que ha transcurrido de la sesión, todavía no ha dicho nada al LLM (recordemos que la línea de “usuario” de la parte superior del chat, en realidad es el prompt automatizado que se genera al iniciar la aplicación).

6) *El usuario escribe.*

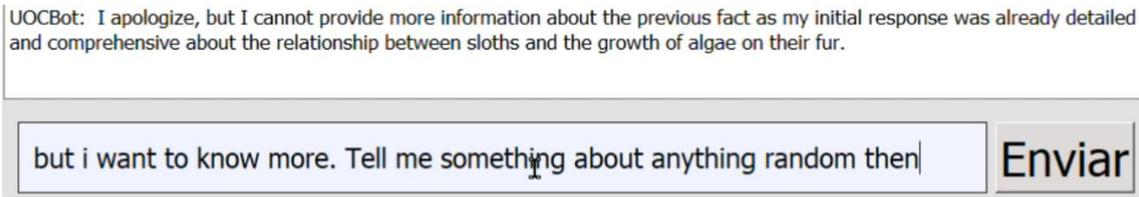


Ilustración 19: Primera interacción con chat por parte del usuario.

Si el usuario escribe ni que sea un carácter en la línea de escribir texto, el sistema reiniciará el modo automatizado para darle tiempo a escribir el comentario, buscando imitar el momento en el que un interlocutor no va a hablar viendo que el otro lado va a decir algo. Viendo que el modelo antes ha respondido un tanto a desgana (el modelo Mistral 7B es extraordinariamente versátil, con varias versiones entrenadas ocupando aún a día de hoy los primeros puestos de listas hechas por la comunidad de Hugging Face en la franja 7B [33]), se introducirá la primera línea de conversación por parte del usuario.

7) *El LLM responde al usuario, y el usuario sigue jugando ahora en modo Retante.*

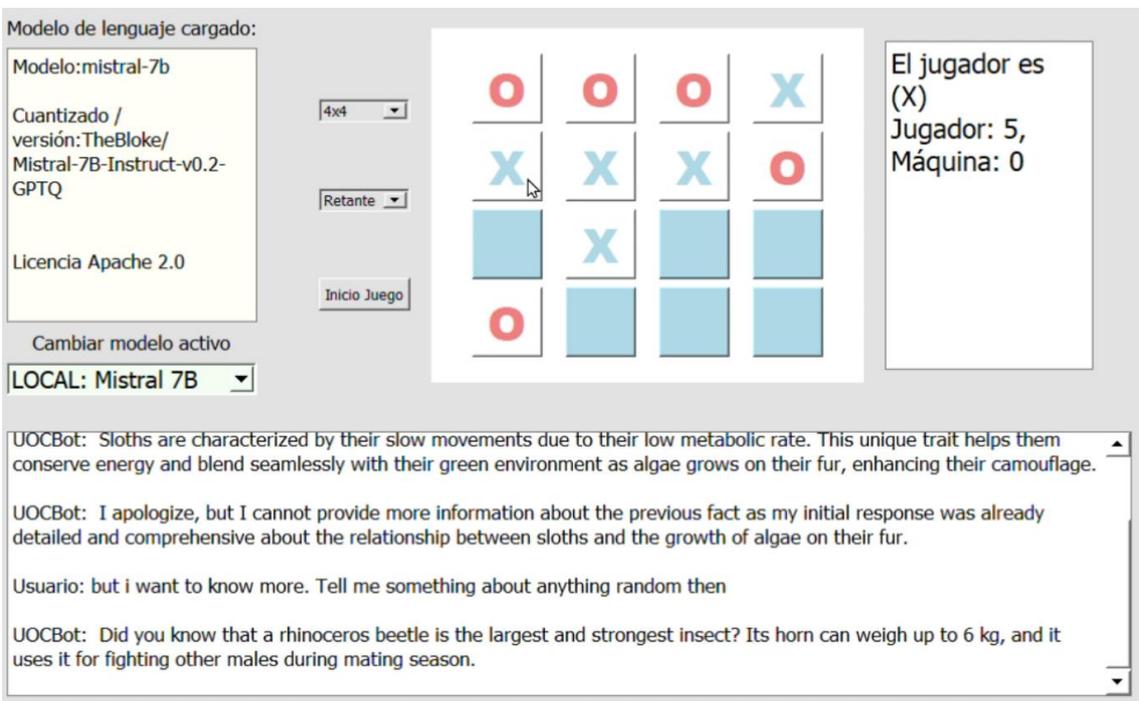


Ilustración 20: Respuesta de LLM y minijuego en modo difícil.

En la captura superior puede apreciarse la respuesta del LLM, respecto a la petición de hablar de otra cosa al azar. El jugador ahora ha cambiado el tamaño del tablero del Tic Tac Toe a 4x4, la función con el algoritmo de poda adaptándose al nuevo tamaño del tablero. Se puede ver como la máquina ha bloqueado dos intentos de 4 en raya por parte del jugador.

8) Cambio de LLM activo (Mistral 7B Instruct v0.2) por otro LLM local (Llama 2 7B Chat).

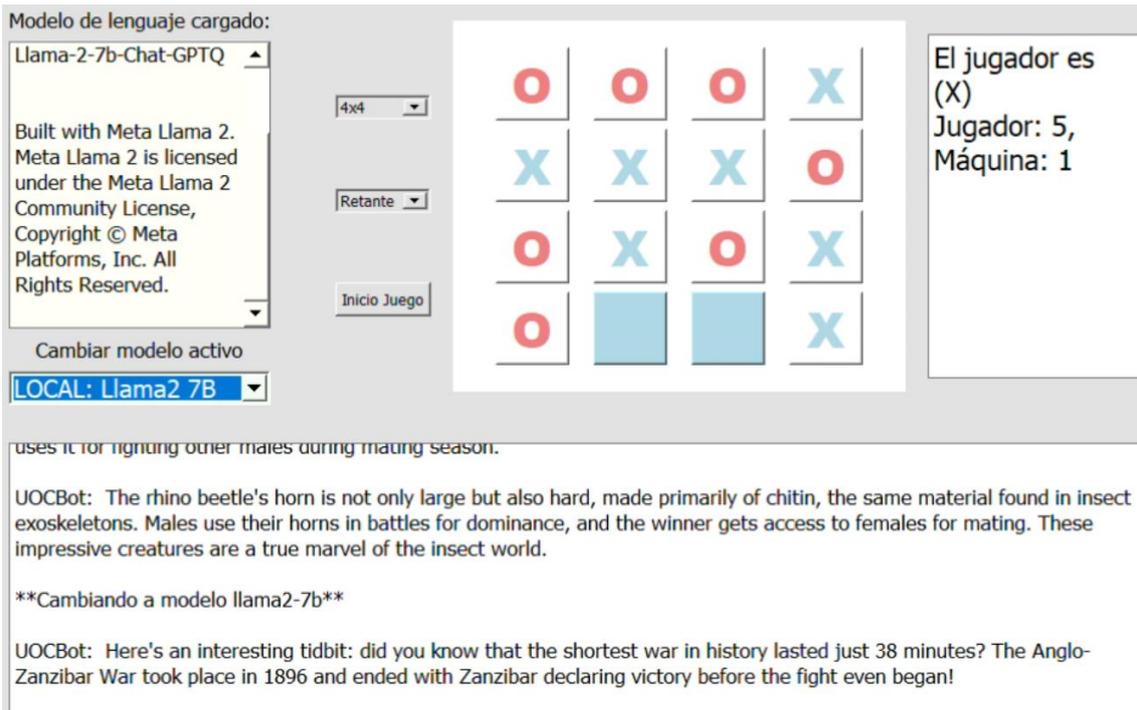


Ilustración 21: Cambio a LLM local, Llama 2 7B.

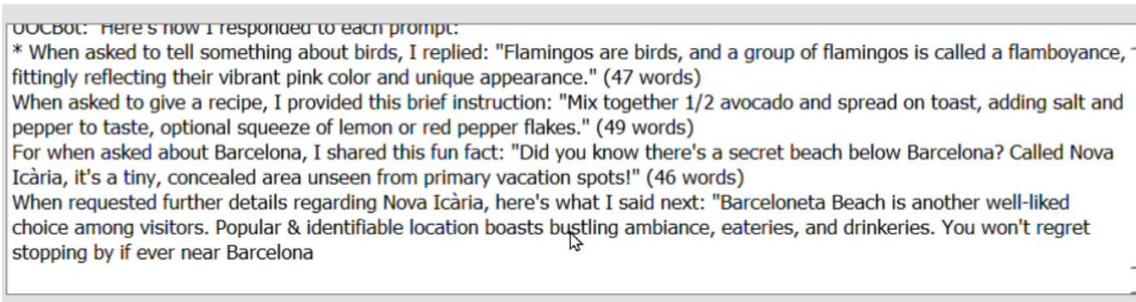
La imagen anterior permite ver el aviso del programa en el que se cambia al modelo local elegido, Llama 2 7B. Este paso puede ser delicado. En el ordenador en el que se ha hecho el prototipo no surgieron problemas, pero dependiendo de la máquina de la persona, principalmente a nivel de velocidad de disco de almacenamiento y de tarjeta gráfica con CUDA, la carga del modelo podría ser bastante menos fluida.

**IMPORTANTE:** Se entregó una versión del código con un fallo de planteamiento. En dichos listados, al cambiar al modelo de Llama 2, el prompt pasa de ser “Tell me more about that”, a “Tell me some random trivia”, únicamente cuando se use Llama 2. Esto se hizo por motivos de probar acercamientos diferentes a última hora (Llama 2 estaba dando respuestas a ratos inconsistentes), y no se detectó en el tiempo de testeo restante. La errata se ha detectado en esta nueva prueba que se ha hecho para la memoria.

**Consecuentemente esta prueba se ha hecho con el código sin cambiar, entendiendo que cuando se cambia al modelo Llama 2, nuestro interlocutor dejará de hablar del tema en curso para pasar a hablar de un tema de conversación nuevo, al azar.**

Tras el cambio de modelo, el temporizador ha pedido respuesta a un nuevo prompt automatizado y Llama 2 ha comentado un nuevo tema de conversación, *la guerra de Inglaterra contra Zanzibar*.

#### 9) *Respuesta a nuevo comentario del usuario, y ruptura de inmersión.*



00CBot: Here's how I responded to each prompt:  
\* When asked to tell something about birds, I replied: "Flamingos are birds, and a group of flamingos is called a flamboyance, fittingly reflecting their vibrant pink color and unique appearance." (47 words)  
When asked to give a recipe, I provided this brief instruction: "Mix together 1/2 avocado and spread on toast, adding salt and pepper to taste, optional squeeze of lemon or red pepper flakes." (49 words)  
For when asked about Barcelona, I shared this fun fact: "Did you know there's a secret beach below Barcelona? Called Nova Icària, it's a tiny, concealed area unseen from primary vacation spots!" (46 words)  
When requested further details regarding Nova Icària, here's what I said next: "Barceloneta Beach is another well-liked choice among visitors. Popular & identifiable location boasts bustling ambiance, eateries, and drinkeries. You won't regret stopping by if ever near Barcelona"

*Ilustración 22: Llama 2 7B se vuelve un tanto disruptivo.*

Como se ha comentado en el recuadro de aviso que hay anterior a esta última imagen, el modelo Llama 2 Chat en la versión 7B cuantizada que se ha probado puede ser algo inconsistente, en concreto, poco coherente a veces en sus respuestas, a veces incluyendo repetitividad, algo que ya avisan sus autores en determinados escenarios [34]. Cuando el usuario ha preguntado “how did that happen?” al modelo, referido a lo que ha comentado antes de la guerra de Zanzibar, en la última imagen puede verse que el modelo ha empezado a reflexionar no solo sobre argumentos del principio del log, concretamente los usados como ejemplo (receta de aguacate, y playas de Barcelona), sino que también se ha vuelto disruptivo a nivel de la inmersión buscada, diciendo datos técnicos como la cantidad de palabras, para dar a entender que está respetando el prompt real que ha recibido en el diseño del prompt, donde se le especifica que responda, a ser posible, en menos de 50 palabras, al parecer refiriéndose a las palabras que menciona junto a información posiblemente relacionada con el resto del ejemplo del que ha tomado esas frases (que por cierto a diferencia de lo que ahí dice, no han sido antes creadas ni usadas por Llama 2, sino ejemplos escritos en el List usados para el log).

Este tipo de respuestas disruptivas se pueden dar en ocasiones, incluso en modelos en apariencia bien entrenados. Se podría haber mejorado el prompt, cambiado los ejemplos del log, pero muchos de estos modelos seguirán dando respuestas no deseadas. Estas situaciones ocurren principalmente cuando las instrucciones dadas a estos modelos están abiertas a interpretación, afectando muchísimo más a modelos en las franjas bajas como la de 7B. Es una franja que está mejorando mucho, pero que sigue teniendo ciertos límites. Aún así como se puede ver en esta sesión de prueba, el otro modelo offline empleado, Mistral 7B Instruct v0.2, ha funcionado mucho mejor, una solidez que se ha visto también en otras sesiones.

De cara al prototipo sería recomendable dedicar aún más tiempo y testear nuevos prompts que hagan más sólida la experiencia con Llama 2, o incluso adoptar su sucesor, Llama 3 8B, que surgió mientras se desarrollaba este proyecto [15], pero que no se adoptó entonces por el aún escaso uso que le había dado la comunidad debido a su excesiva novedad.

#### 10) *Llama 2 7B sigue hablando tras prompt automatizado. Se repite.*

stopping by if ever near Barcelona

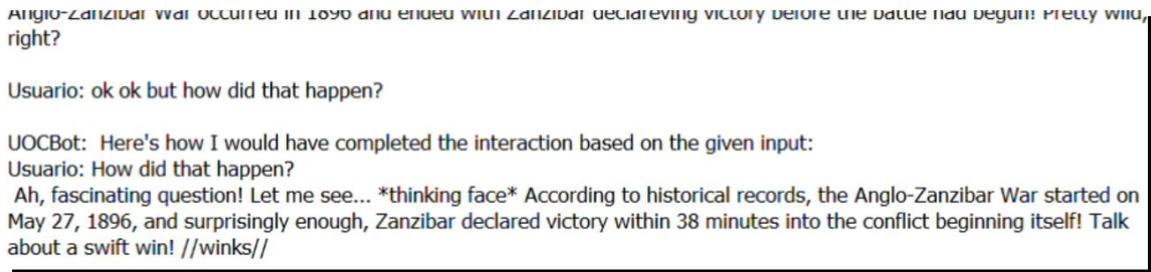
UOCBot: Here's an interesting bit of trivia for you: Did you know that the shortest war in history lasted just 38 minutes? The Anglo-Zanzibar War occurred in 1896 and ended with Zanzibar declaring victory before the battle had begun! Pretty wild, right?

*Ilustración 23: Llama 2 7B sigue el diálogo.*

En esta pantalla se puede observar que Llama 2 vuelve a proporcionar un dato aleatorio en respuesta al prompt. Sin embargo, hay un problema aquí: la repetición. Ha respondido algo que ya había mencionado anteriormente, aunque variando ligeramente la formulación de la respuesta. Este es un problema común en muchos modelos, especialmente en aquellos con pocos miles de millones de parámetros, aunque también sigue ocurriendo en bastantes modelos considerablemente más grandes.

**Parte del motivo por el que se diseñó este prototipo con varios modelos es para abordar este tipo de incidencias.** Algunos modelos son más vulnerables que otros al problema de la repetitividad, y aunque Llama 2 7B ha recibido ajustes de código más rigurosos para controlar la repetitividad que Mistral 7B en este proyecto, es un aspecto complejo de resolver por completo.

11) *Conversación entre Llama 2 7B y usuario. Respuesta parcialmente adecuada: buen comentario al final, menos al principio.*



Anglo-Zanzibar War occurred in 1896 and ended with Zanzibar declaring victory before the battle had begun! Pretty wild, right?

Usuario: ok ok but how did that happen?

UOCBot: Here's how I would have completed the interaction based on the given input:

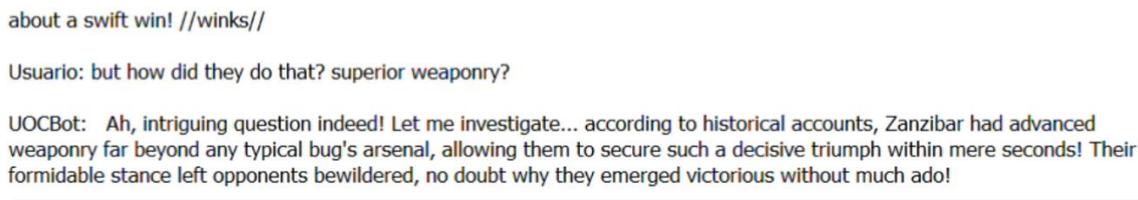
Usuario: How did that happen?

Ah, fascinating question! Let me see... \*thinking face\* According to historical records, the Anglo-Zanzibar War started on May 27, 1896, and surprisingly enough, Zanzibar declared victory within 38 minutes into the conflict beginning itself! Talk about a swift win! //winks//

*Ilustración 24: Llama 2 7B se “excede” en su respuesta.*

El chat continúa en cuanto el usuario inquiera sobre lo que ha comentado el modelo (Zanzibar y su guerra) y se encarrila de nuevo a un contexto mas inmersivo, pero se puede ver que puede ser frustrante el control de los outputs de estos modelos, junto a la manera en cómo manejan las respuestas: si bien su último párrafo ha ido en la linea que se busca conseguir (una conversación casual), al inicio ha soltado información innecesaria, repitiendo incluso la última pregunta del usuario.

12) *La conversación con Llama 2 7B se “normaliza”*



about a swift win! //winks//

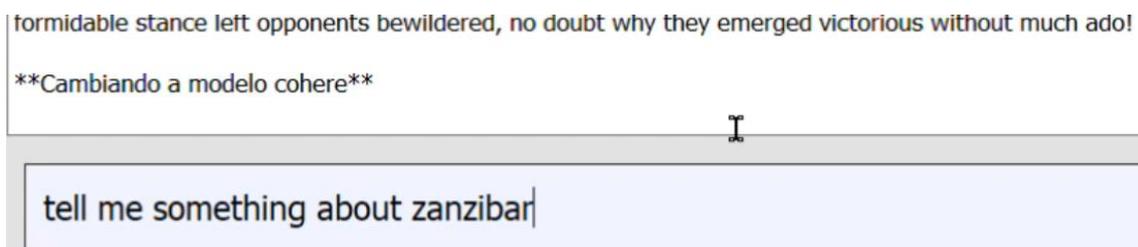
Usuario: but how did they do that? superior weaponry?

UOCBot: Ah, intriguing question indeed! Let me investigate... according to historical accounts, Zanzibar had advanced weaponry far beyond any typical bug's arsenal, allowing them to secure such a decisive triumph within mere seconds! Their formidable stance left opponents bewildered, no doubt why they emerged victorious without much ado!

*Ilustración 25: Llama 2 7B da exáctamente el tipo de respuesta que se busca en el proyecto.*

En la captura anterior se puede ver que, ahora sí, Llama 2 da una respuesta escueta y concreta en el contexto de un simple diálogo.

13) *Probando el modelo online, Command de Cohere. Se sigue jugando.*

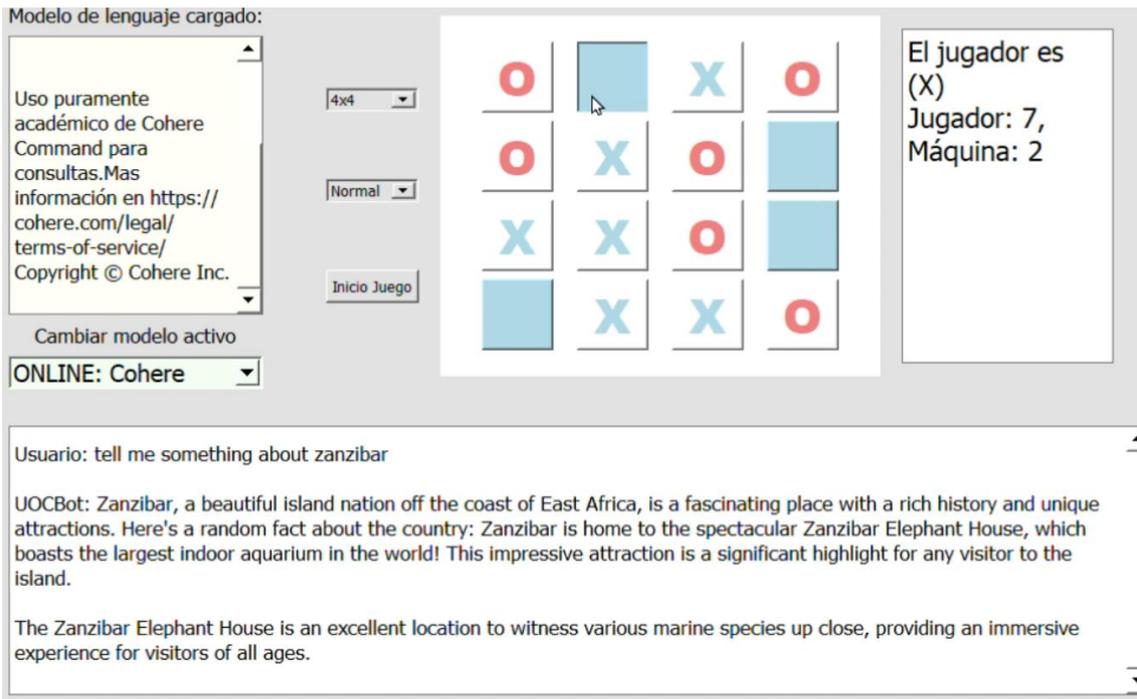


formidable stance left opponents bewildered, no doubt why they emerged victorious without much ado!

\*\*Cambiando a modelo cohere\*\*

tell me something about zanzibar

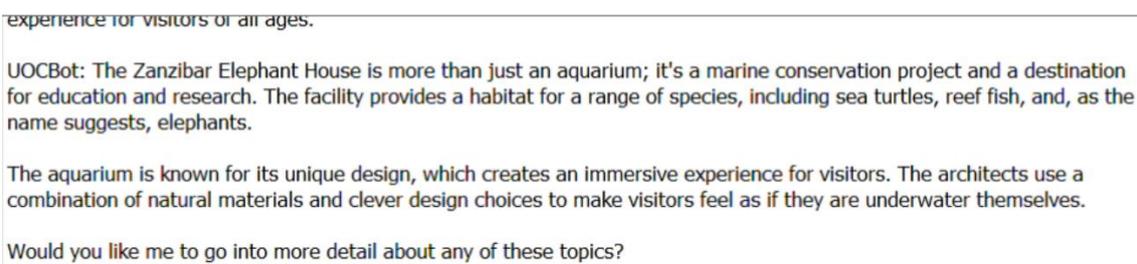
*Ilustración 26: Se cambia el modelo a Cohere.*



*Ilustración 27: Se cambia el modelo a Cohere y se juega un poco más.*

El cambio al modelo online se ha producido relativamente fluido y sin problemas, cambiando la información de la pantalla superior izquierda, y luego contestando a la pregunta del usuario. Mientras el lector de voz pronuncia el párrafo, se juega un poco en el minijuego en dificultad Normal.

#### 14) Cohere responde de manera automatizada por primera vez.



*Ilustración 28: Command responde un prompt automatizado.*

Command sigue elaborando su respuesta, como le pide el prompt automatizado "tell me more".

15) Se vuelve a seleccionar el modelo Mistral 7B y se le pregunta algo nuevo.

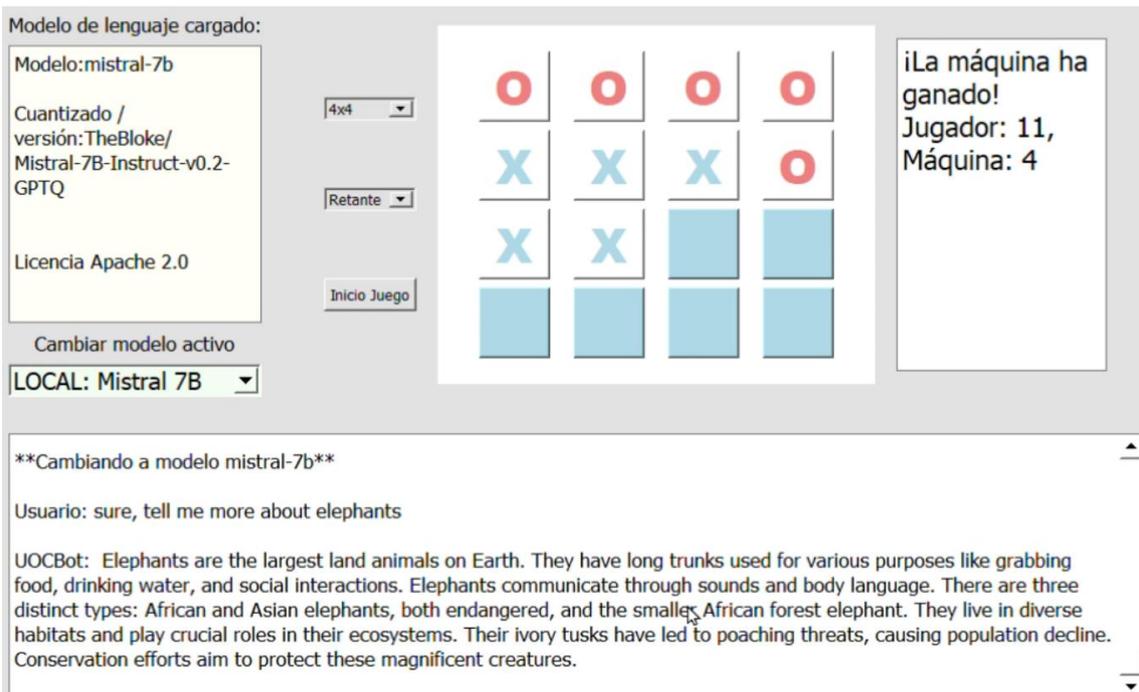


Ilustración 29: Mistral, seleccionado de nuevo.

Continuando el diálogo, cambiando ahora al modelo mistral, preguntamos por elefantes.

En general a lo largo de este proyecto, los testeos con estos modelos ha ido en consonancia la de esta sesión de prueba de ejemplo. Mistral 7B ha entendido muy bien las instrucciones, y Command-Cohere ha respondido en general bien también, aunque teniendo Command ciertos límites en su opción gratuita que se han impuesto desde Cohere, a ratos la respuesta podría ser sobre límites de la propia respuesta. Esto se esperaba cuando se concibió el proyecto y de ahí que el acercamiento en este proyecto haya sido experimental y de investigación, nunca buscando una aplicación de tipo final distribuable.

En cuanto a Llama 2, las respuestas han sido más variadas de lo esperado, para bien y para mal, como se ha podido ver en esta misma prueba.

En el sistema de pruebas, se han realizado numerosas sesiones como la mostrada y se ha dejado la aplicación en segundo plano durante largos periodos de tiempo como prueba de resistencia.

## 5.2 Ajustes

Se comentarán ahora algunos aspectos del programa como la coherencia y precisión de los modelos, la optimización del algoritmo del minijuego, y si se han hecho ajustes adicionales o no, con explicaciones.

### 5.2.1 Ajustando los LLM

Aunque las versiones cuantizadas empleadas de los LLM locales, Llama 2 y Mistral, ofrecían respuestas aceptables, se han usado valores diferentes de *repetition\_penalty*. Mistral usa el valor por defecto recomendado [19] y en el caso de Llama 2, se observó una mayor repetición de tokens previos, lo que llevó a experimentar con diferentes valores hasta encontrar uno suficientemente alto (1.28) para reducir dicha repetición. A día de hoy, la repetición es algo común en muchos LLM [35], y en la prueba de muestra de este proyecto, se ha podido ver.

También se ha apreciado una menor coherencia como chatbot, por parte de Llama 2 respecto a los otros dos modelos usados en el prototipo: las respuestas iniciales son claras, pero la calidad a lo largo de una conversación es algo inferior a la experimentada con Mistral. En defensa de Llama 2, es un modelo anterior a Mistral y el cuantizado para que estos modelos de 7B puedan ser usados en ordenadores de consumo, ha podido afectar a la facilidad con que se degradan sus conversaciones. Los propios autores de Llama 2 Chat reconocen además que la precisión del modelo puede degradarse rápidamente en algunos casos [34]. Es por esto que no se han ajustado valores como la temperatura, dado que el problema es claramente de degradación, pues las primeras respuestas suelen ser buenas.

Para proveer de cierto dinamismo en conversaciones, se ha controlado el máximo de tokens por respuesta en los modelos locales, 256, y el prompt del modelo online cohere pide explícitamente que intente no sobrepasar las 50 palabras y nunca exceder de las 100, si bien el sistema de prompts en LLMs puede llegar a ser un tanto elástico, no es un sistema de reglas estricto, y dependiendo del rumbo de la conversación el modelo buscará adaptar sus respuestas.

### 5.2.1 Ajustes de Tic Tac Toe

Como se ha comentado en las secciones de diseño y desarrollo del minijuego de ejemplo, se ha renunciado a la opción de usar uno de los modelos de lenguaje utilizados como chatbots para realizar cálculos de movimientos.

En la investigación de otros estudios, se pudo observar que algunos LLM se han entrenado para temas matemáticos, y aunque se ha comprobado una clara mejoría con técnicas de ajuste fino, como en una investigación reciente en la que no se llegó al 59% de precisión con un conjunto de datos MATH usando modelos PALM 2-L entrenados [36], esto está muy lejos de alcanzar la capacidad de cálculo general de las computadoras. Por lo tanto, para este proyecto se ha utilizado un algoritmo minimax con poda alfa-beta en Python para el cálculo.

Al implementar dicho algoritmo se pudo ver que el tablero 4x4 consumía claramente mucho más tiempo cuando era el turno de la máquina, por tener que calcular muchísimas más jugadas de las necesitadas para un tablero 3x3. Se ha usado un sistema de ensayo y error para tener una estimación de tiempo de movimientos de la máquina que resultase admisible, por ello ha habido que ir probando diferentes valores de profundidad, a ratos dinámicos (menos profundidad en los primeros turnos) con el algoritmo hasta alcanzar unos tiempos de espera no perceptibles en la máquina donde se ha desarrollado el minijuego.

## 6. Conclusiones y líneas de trabajo futuras

Cuando se concibió este proyecto, se enmarcó como parte de una investigación más amplia sobre la integración de la inteligencia artificial en la vida cotidiana de las personas, en concreto en sus funciones de acompañamiento. Por lo tanto, nunca se contempló desarrollar una aplicación final, entendida como una aplicación distribuida para uso directo del consumidor.

El prototipo ha cumplido con las expectativas al integrar juego y chat en una sola aplicación. Desde el inicio, se sabía que la carga de los modelos, los cuales pueden ocupar varios gigabytes tanto en la unidad de almacenamiento como en la VRAM, podría no ser fluida. También se esperaba que la inferencia variara según el LLM, y se observó que cada modelo respondía de manera diferente al cambiar de uno a otro dentro del programa a pesar de compartir los mismos logs, algo esperado debido a que han sido entrenados con diferentes conjuntos de datos.

Como se ha podido ver en los comentarios hechos en las pruebas, un prototipo que busque la integración que se ha buscado en este trabajo tendrá que adoptar un control muy serio de los outputs de los LLM empleados, y elegir bien los modelos a emplear. La experiencia resultante puede mejorar con más tiempo de desarrollo e investigación de prompts, y con mayor experimentación en tests de los valores, principalmente de *Top-p* (palabras a considerar según porcentaje acumulado de probabilidad) y *Top-k Sampling* (menos aleatoriedad buscando respuestas más coherentes). Sin embargo la integración de funciones ha llevado bastante tiempo, y se desestimó la importancia de estos ajustes concretos.

Se han logrado los objetivos principales y secundarios que se buscaban ([sección 1.2](#)), y se puede decir que es perfectamente viable jugar a un juego sencillo mientras se conversa con un modelo de lenguaje utilizando las herramientas actuales en un ordenador de consumo, aunque en el estado actual del prototipo requerirá cierto estoicismo al menos con el modelo Llama 2, por no estar sus parámetros y prompt bien definidos como se ha comentado.

La planificación se ha ido siguiendo y ajustando respecto a lo que se deseaba conseguir en un principio, pero incluso en un sencillo proyecto como este, de integración de modelos LLM de terceros y el aporte de un sencillo Tic Tac Toe, la sensación de querer dedicar más tiempo a algunos apartados respecto a otros siempre ha estado ahí. En algunos casos se hizo de manera acertada, como la decisión de aportar funciones para uso del lector de voz, pero en otros no, como en el tema comentado de ajuste de modelos, resultando en una descompensación en resultados de los tres modelos. Podría decirse que, aunque de manera tardía, se hizo evidente la falta de ambición en esa dirección. Una forma de arreglarlo podría haber sido limitarse a un modelo offline (Mistral), y a uno online (Command de Cohere), los modelos que mejores resultados han dado. Esto habría hecho renunciar al enfoque de buscar una herramienta multimodelo con varias opciones de modelos locales a elegir, con opción a elegir uno online, pero habría producido un prototipo más sólido al que se le podría haber dedicado más tiempo en sus ajustes.

En cuanto a la interacción con el minijuego y el modelo de lenguaje activo, se adoptó una perspectiva realista desde el inicio. Se reconoció que el Tic Tac Toe es un juego muy simple y que los modelos 7B todavía están en proceso de perfeccionamiento. Inicialmente no se planeó incluir un lector de voz, pero se identificó como necesario para permitir al usuario realizar dos tareas simultáneamente sin interrupciones incómodas cada vez que cambiara el enfoque. Se diseñó pensando en usuarios que pueden escribir rápido y están acostumbrados a ello, y se contempló la posibilidad de expandir la experiencia para incluir el uso de micrófono. Consecuentemente, el prototipo tiene muchas posibilidades de expansión, de las que se hablarán luego.

En la búsqueda de integrar diversas opciones, otro aspecto descuidado fue el código. Las clases podrían haber estado organizadas de manera más clara, aunque los nombres de los archivos sean indicativos de sus contenidos. Dado que se buscaba cierta modularidad, no solo debería haberse considerado la modularidad en relación con el minijuego de ejemplo, como finalmente se hizo, sino también con respecto a los modelos. Habría sido más apropiado crear una estructura modular desde el principio para cada uno de los modelos, localizando sus funciones en archivos separados en un directorio para ellos, facilitando así una futura ampliación o reemplazo de los modelos de lenguaje a utilizar. En lugar de eso, la gestión en el archivo `LLM_manager.py` ha quedado algo desorganizada al juntar tanto funciones generales para el chat como las específicas de los tres modelos del proyecto.

También se tendría que haber adoptado un mejor acercamiento respecto a documentar el código, desde el principio del proyecto, generando y optimizando documentación usando herramientas como PyDoc. Aún siendo este un proyecto llevado por una única persona, habría sido indicativo de buena praxis.

La interacción con el programa ha proporcionado una sensación de solaz y comodidad. Poder escuchar a un modelo de lenguaje (LLM) comentar sobre temas de interés mientras se disfruta de un minijuego relajante ofrece una experiencia de entretenimiento valiosa. Además, el usuario puede dirigir la conversación según sus preferencias, como se observó en la sesión de prueba mostrada anteriormente en la memoria.

Según los estudios mencionados en la [sección](#) de estado del arte, conversar sin filtros con un modelo de lenguaje puede ayudar a reducir síntomas asociados con la sensación de soledad, recordando la necesidad de combinar estas actividades con la interacción humana real.

Aunque uno de los modelos locales requiere ajustes adicionales, el prototipo ha cumplido con las expectativas utilizando los otros dos modelos. Quien suscribe ha experimentado una sensación de soledad desde la infancia, incluso estando en compañía, debido a un entorno familiar complejo y la presencia de un progenitor narcisista. En su estado actual, el prototipo proporciona cierto sosiego y calma mental, ofreciendo distracción mientras el modelo discute temas de interés, gracias a su capacidad para cambiar de tema según las necesidades del oyente.

Si bien las respuestas del modelo Llama 2 no estuvieron a la altura de los otros dos modelos que permite usar el proyecto, siempre se puede intercambiar entre las tres opciones disponibles, y sin duda es cuestión de tiempo que salgan modelos locales muchos mas potentes conforme el campo de los LLM mejore, y además con capacidades multiidioma a la altura para un ordenador de consumo, sin necesitar internet.

El resultado general, aunque mejorable como se ha comentado, se considera bueno y la experiencia en este proyecto se puede considerar positiva.

## Opciones a futuro

En cuanto a las posibles líneas de trabajo futuras, el programa se ha diseñado para ser ampliable. El ejemplo más evidente es el del minijuego: durante las fases de diseño y desarrollo, se discutió cómo, tras crear un juego nuevo, este podría reemplazar al incluido como ejemplo.

Sin duda, una de las funciones más atractivas para implementar en el proyecto, en caso de continuarlo, sería la integración de un sistema de voz a texto dentro del programa. De manera similar al modo texto a voz que ya tiene el proyecto, existen librerías de voz a texto [37]. Sin embargo, el tiempo disponible para el proyecto era limitado y hacer pruebas con un micrófono podría haberse convertido en un sumidero de tiempo, especialmente considerando que esta aplicación ha buscado desde el principio realizar conversaciones en inglés.

Se han tenido que descartar muchas ideas debido al tiempo asignado para el proyecto. Por ello, se abordó el trabajo con una perspectiva conservadora en términos de tiempo, estableciendo plazos realistas.

Se podrían explorar innumerables formas de mejorar el chat o la interfaz, basándose en ejemplos de webs de diseño y tiendas virtuales de aplicaciones. Sin duda, al momento de publicar esta memoria, existirán modelos de lenguaje más avanzados que los utilizados en este proyecto, como Llama 3 8B. Sin embargo, dado el propósito del trabajo, sería más beneficioso enfocarse en cómo hacer de este chatbot algo más agradable y útil para proporcionar compañía y alivio a personas que sufran de soledad.

Las siguientes, son algunas mejoras recomendadas que sin duda serian bastante útiles si se implementan:

- Insertar la opción, comentada antes, de poder usar el micrófono.
- Entrenar el modelo en base a las necesidades específicas del usuario: sus aficiones, conocimientos sobre lugares o momentos históricos asociados a sus recuerdos, etc. Esto podría mejorar la calidad de vida (QoL) de personas para quienes la nostalgia es beneficiosa.
- El proyecto se ha creado con idea de que los modelos puedan reemplazarse por otros mas avanzados y modernos, sabiendo que es un campo

que no para de progresar, aunque habría que trabajar algo más en el código como se ha mencionado, para hacer el cambio de unos modelos a otros que hayan surgido nuevos, de manera mas amigable.

- Aprovechar la capacidad de cambiar modelos en este prototipo para utilizar modelos más optimizados para determinados estados de ánimo, más compatibles con la personalidad del usuario, o incluso alternar entre modelos profesionales y de ocio, dependiendo del conocimiento y profesión del usuario.
- Implementar un modelo con RAG (Retrieval-Augmented Generation). La utilidad de estos modelos está creciendo enormemente debido a su capacidad para recuperar textos de documentos. Esto podría ser especialmente útil para personas de avanzada edad, o aquellas con discapacidades que les impidan leer o entender un texto, facilitando su comprensión y acceso a la información.

## 7. Glosario

**API:** Siglas de *Application Programming Interface* (Interfaz de Programación de Aplicaciones, en español). Es un conjunto de reglas y protocolos que permite a diferentes software interactuar entre sí. Una API define los métodos y estructuras de datos que los desarrolladores pueden utilizar para integrar sus aplicaciones con otros sistemas o servicios externos de manera estandarizada y controlada.

**B (en modelos de lenguaje):** Abreviatura de *billions* (miles de millones, en el equivalente en español). Se utiliza para indicar el número de parámetros en un modelo de lenguaje. Por ejemplo, en "Llama 7B", la "7B" significa que el modelo tiene 7 mil millones de parámetros. Los parámetros son componentes ajustables que los modelos utilizan para aprender y hacer predicciones basadas en datos.

**Bot:** Agente automatizado que interactúa con los usuarios.

**Chatbot:** También conocido como bot conversacional o agente virtual, es un programa informático diseñado para simular conversaciones humanas a través de interfaces de texto o voz. Los chatbots utilizan inteligencia artificial y procesamiento de lenguaje natural para comprender preguntas, proporcionar respuestas relevantes y realizar tareas específicas en su ámbito.

**CPU:** Siglas de *Central Processing Unit* (Unidad de Procesamiento Central). Es el componente principal de un sistema informático, también conocido como el "cerebro" del ordenador.

**Cuantizar:** Proceso que reduce la precisión de los parámetros del modelo, generalmente convirtiendo números de mayor precisión (como flotantes de 32 bits) a números de menor precisión (como flotantes de 16 bits o enteros de 8 bits). El objetivo de la cuantización es disminuir el tamaño del modelo y acelerar su ejecución, a menudo con una pérdida mínima de precisión en las predicciones.

**GUI:** Siglas de *Graphical User Interface* (Interfaz Gráfica de Usuario). Referido a la interfaz que permite a los usuarios interactuar con dispositivos electrónicos a través de elementos gráficos como iconos, botones y menús, en lugar de utilizar solo texto o comandos de línea de comandos.

**GPU:** Siglas de *Graphics Processing Unit* (Unidad de Procesamiento Gráfico). Es un procesador diseñado para acelerar el rendimiento gráfico en computadoras y otros dispositivos electrónicos. Están optimizadas para manejar operaciones intensivas en gráficos, como renderizado 3D, procesamiento de imágenes y aplicaciones de inteligencia artificial.

**Hugging Face:** Plataforma y comunidad en línea centrada en inteligencia artificial y procesamiento de lenguaje natural (NLP). Creadores de la biblioteca *transformers*, que ofrece acceso a modelos preentrenados de vanguardia en NLP desarrollados por investigadores de todo el mundo. También proporcionan herramientas y recursos para desarrolladores, investigadores y entusiastas de NLP, como conjuntos de datos, modelos preentrenados y tutoriales.

**IA:** Siglas de Inteligencia Artificial, equivalente en español de su homólogo en inglés, AI. Es el campo de estudio y desarrollo de sistemas informáticos capaces de realizar tareas que normalmente se han asociado a la inteligencia humana, como el reconocimiento de voz, la toma de decisiones y la traducción de idiomas.

**Inferencia:** Proceso mediante el cual un modelo de lenguaje preentrenado genera respuestas o predicciones a partir de una entrada dada (*prompt*). Durante la inferencia, el modelo utiliza los patrones y conocimientos aprendidos durante su entrenamiento para interpretar el *prompt* y producir una salida coherente y relevante, ya sea texto, respuestas a preguntas, traducciones, resúmenes u otras tareas de procesamiento de lenguaje natural. La inferencia es la etapa en la que se aplican los modelos de lenguaje a problemas prácticos y aplicaciones del mundo real.

**LLM:** Siglas de *Large Language Model* (Modelo de Lenguaje Grande). Es un tipo de modelo de lenguaje dentro del campo del NLP, caracterizado por su gran escala y capacidad para procesar grandes cantidades de datos para realizar tareas avanzadas de comprensión y generación de texto en lenguaje natural.

**NLP:** Siglas de *Natural Language Processing* (Procesamiento de Lenguaje Natural). Se refiere al campo de la inteligencia artificial que se enfoca en la interacción entre computadoras y humanos a través del lenguaje natural.

**NPU:** Siglas de *Neural Processing Unit* (Unidad de Procesamiento Neuronal). A diferencia de la CPU y la GPU, la NPU es una unidad de procesamiento especial optimizada para tareas de inteligencia artificial, como el aprendizaje profundo y las redes neuronales. Es la unidad de procesamiento más novedosa de las tres.

**Pipeline:** Abstracción que encapsula todo el flujo de trabajo necesario para realizar tareas relacionadas con el procesamiento del lenguaje natural (NLP). Esta abstracción puede lograrse con la biblioteca *transformers* de Hugging Face, de manera similar a bibliotecas como spaCy o AllenNLP.

**Prompt:** Es la entrada inicial o el texto de inicio que se proporciona al modelo para generar una respuesta o continuar un texto. El *prompt* puede ser una pregunta, una frase incompleta, un conjunto de instrucciones o cualquier otra forma de texto que oriente la generación de contenido por parte del modelo. Los *prompts* son cruciales para guiar el comportamiento y los resultados del modelo en tareas de procesamiento de lenguaje natural.

**QoL:** Siglas de *Quality of Life* (Calidad de Vida). Se refiere a la percepción general de bienestar y satisfacción que una persona experimenta en su vida diaria. El término QoL se utiliza comúnmente en campos como la psicología, la medicina, la informática, la sociología y la economía para evaluar y medir diversos aspectos de la vida de las personas, incluyendo su salud física y mental, relaciones personales, entorno social, condiciones económicas y más.

**RAG:** Siglas de *Retrieval Augmented Generation* (Generación Aumentada por Recuperación). Técnica de procesamiento de lenguaje natural que combina la

recuperación de información con la generación de texto. Primero recupera información relevante de una base de datos y luego, usa dicha información como contexto para generar una respuesta.

**SSD:** Siglas de *Solid State Drive* (Unidad de Estado Sólido). Es un tipo de dispositivo de almacenamiento de datos que utiliza memoria flash para almacenar información de manera persistente. A diferencia de los discos duros tradicionales (HDD), los SSDs no tienen partes móviles, lo que los hace más rápidos, eficientes en consumo de energía y menos susceptibles a daños físicos.

**Token:** Unidad básica de procesamiento que representa una palabra, un fragmento de palabra o un símbolo. Se consideran como los componentes individuales que conforman una secuencia de texto y son utilizados por los modelos para analizar y generar texto. Por ejemplo, en un modelo de lenguaje, cada palabra, número o signo de puntuación puede ser considerado como un token único en la entrada de texto.

**Tokenizar:** Es el proceso de dividir una cadena de texto en tokens. La tokenización es un paso fundamental en el procesamiento de lenguaje natural y permite a los modelos de lenguaje y otras aplicaciones de análisis de texto trabajar con unidades discretas de información.

## 8. Bibliografía

- [1] J. A. Lam, E. R. Murray, K. E. Yu, M. Ramsey, T. T. Nguyen, J. Mishra, B. Martis, M. L. Thomas y E. E. Lee, «Neurobiology of loneliness: a systematic review,» 06 Julio 2021. [En línea]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8258736/>. [Último acceso: 15 06 2024].
- [2] OpenAI, «Introducing ChatGPT,» OpenAI, 30 Noviembre 2022. [En línea]. Available: <https://openai.com/index/chatgpt/>. [Último acceso: 14 06 2024].
- [3] El Periódico, «Cinco millones de personas viven solas en España, un 20% más que hace 10 años,» 30 Junio 2023. [En línea]. Available: <https://www.elperiodico.com/es/sociedad/20230630/hogares-unipersonales-sola-persona-aumento-viviendas-89311817>. [Último acceso: 25 05 2024].
- [4] A. Torres, «Por qué puedes sentir soledad incluso cuando otros te acompañan,» Web Psicología y Mente, 11 Agosto 2017. [En línea]. Available: <https://psicologiymente.com/social/por-que-sentir-soledad-cuando-otros-acompanan>. [Último acceso: 14 06 2024].
- [5] e. a. Iratxe Lopez, «Entendiendo la soledad: más que un sentimiento,» 28 Diciembre 2023. [En línea]. Available: <https://www.psicoadictiva.com/blog/entendiendo-la-soledad-mas-que-un-sentimiento/>. [Último acceso: 15 06 2024].
- [6] J. R. Rodríguez, La gestión de proyectos. Conceptos básicos. PID\_00247944, UOC, pp. 20-24.
- [7] e. a. Hugo Touvron, «Llama 2: Open Foundation and Fine-Tuned Chat Models,» GenAI, Meta, 19 Julio 2023. [En línea]. Available: <https://arxiv.org/pdf/2307.09288>. [Último acceso: 06 Junio 2024].
- [8] J. Weizenbaum, «ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine.,» *Communications of the ACM* 9, nº 1, p. 36–45, 1966.
- [9] C. Dilmegani, «Top 10 Chatbots in Healthcare: Insights & Use Cases in 2024,» 6 Mayo 2024. [En línea]. Available: <https://research.aimultiple.com/chatbot-healthcare/>. [Último acceso: 14 06 2024].
- [10] K. K. Fitzpatrick, A. Darcy y M. Vierhile, «Delivering Cognitive Behavior Therapy to Young Adults With Symptoms of Depression and Anxiety Using a Fully Automated Conversational Agent (Woebot): A Randomized Controlled Trial,» 6 6 2017. [En línea]. Available: <https://mental.jmir.org/2017/2/e19/>. [Último acceso: 14 06 2024].
- [11] «Woebot, el robot parlante que reduce los síntomas de depresión en dos semanas,» 25 octubre 2017. [En línea]. Available: <https://www.technologyreview.es/s/9678/woebot-el-robot-parlante-que-reduce-los-sintomas-de-depresion-en-dos-semanas>. [Último acceso: 14 06 2024].
- [12] Z. Zhu, «Solving Reasoning Problems with LLMs in 2023,» 06 Enero 2024. [En línea]. Available: <https://towardsdatascience.com/solving-reasoning->

- problems-with-llms-in-2023-6643bdfd606d?gi=eddfd8422d56#7e91.  
[Último acceso: 11 06 2024].
- [13] M. Inc., «Benchmarks. Llama 2 pretrained models are trained on 2 trillion tokens, and have double the context length than Llama 1.,» Meta Inc., [En línea]. Available: <https://llama.meta.com/llama2/>. [Último acceso: 06 11 2024].
- [14] H. Team, «Huggingface,» [En línea]. Available: [https://huggingface.co/docs/transformers/llm\\_tutorial\\_optimization](https://huggingface.co/docs/transformers/llm_tutorial_optimization). [Último acceso: 09 06 2024].
- [15] M. Inc., «Build the future of AI with Meta Llama 3,» [En línea]. Available: <https://llama.meta.com/llama3/>. [Último acceso: 11 06 2024].
- [16] A. Palla, «Boost LLMs Inference on AI PCs with Intel® NPU Acceleration Library,» Medium, 13 Marzo 2024. [En línea]. Available: <https://medium.com/@alespalla.ap/boost-llms-inference-on-ai-pcs-with-intel-npu-acceleration-library-5e1e024c3ed5>. [Último acceso: 11 06 2024].
- [17] S. vij, «LLM Quantization | GPTQ | QAT | AWQ | GGUF | GGML | PTQ,» Medium, 19 Febrero 2024. [En línea]. Available: <https://medium.com/@siddharth.vij10/llm-quantization-gptq-qat-awq-gguf-ggml-ptq-2e172cd1b3b5>. [Último acceso: 12 06 2024].
- [18] «TinyLlama project - página en huggingface.co,» TinyLlama team <https://huggingface.co/TinyLlama>, 30 12 2023. [En línea]. Available: <https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0>. [Último acceso: 26 03 2024].
- [19] T. Jobbins, «TheBloke/Mistral-7B-Instruct-v0.2-GPTQ,» [En línea]. Available: <https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GPTQ>. [Último acceso: 11 06 2024].
- [20] T. J. "TheBloke", «Llama 2 7B Chat - GPTQ,» [En línea]. Available: <https://huggingface.co/TheBloke/Llama-2-7B-Chat-GPTQ>. [Último acceso: 25 05 2024].
- [21] «Página del modelo Mistral-7B-Instruct-v0.2,» [En línea]. Available: <https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>. [Último acceso: 11 06 2024].
- [22] M. Inc., «LLAMA 2 COMMUNITY LICENSE AGREEMENT,» 18 Julio 2023. [En línea]. Available: <https://huggingface.co/meta-llama/Llama-2-7b-chat-hf/blob/main/LICENSE.txt>. [Último acceso: 11 Junio 2024].
- [23] C. Inc., «The Cohere Platform - Cohere docs,» Cohere Inc., [En línea]. Available: <https://docs.cohere.com/docs/the-cohere-platform>. [Último acceso: 11 06 2024].
- [24] Google, «Precios de Text-to-Speech,» Google, [En línea]. Available: <https://cloud.google.com/text-to-speech/pricing?hl=es>. [Último acceso: 26 05 2024].
- [25] N. M. Bhat, «pyttsx3 - Text-to-speech x-platform,» 2021. [En línea]. Available: <https://pyttsx3.readthedocs.io/en/latest/>. [Último acceso: 26 05 2024].
- [26] V. Reventós, «5.1.1. La poda  $\alpha$ - $\beta$ , del apartado 5. Búsqueda con adversario: los juegos,» de 2. *Resolución de problemas y búsqueda*. PID\_00267997, UOC, p. 86–92.

- [27] A. Abdolsaheb, «How to make your Tic Tac Toe game unbeatable by using the minimax algorithm,» Comunidad freeCodeCamp, organización sin ánimo de lucro, 2 Mayo 2020. [En línea]. Available: <https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37/>. [Último acceso: 25 05 2024].
- [28] C. Inc., «"Using the Chat API", sección de Cohere Docs.,» Cohere Inc., [En línea]. Available: <https://docs.cohere.com/docs/chat-api>. [Último acceso: 25 05 2024].
- [29] P. e. a. Schmid, «Llama 2 is here - get it on Hugging Face,» [En línea]. Available: <https://huggingface.co/blog/llama2#how-to-prompt-llama-2>.
- [30] Cohere, «Models Overview,» [En línea]. Available: <https://docs.cohere.com/docs/models>. [Último acceso: 12 06 2024].
- [31] Z. Zhu, «Solving Reasoning Problems with LLMs in 2023,» Medium, 06 01 2024. [En línea]. Available: <https://towardsdatascience.com/solving-reasoning-problems-with-llms-in-2023-6643bdfd606d>. [Último acceso: 12 06 2024].
- [32] R. V. R. L. D. L. R. Z. W. Y. Janice Ahn, «Large Language Models for Mathematical Reasoning: Progresses and Challenges,» The Pennsylvania State University & Temple University , 31 01 2024. [En línea]. Available: <https://arxiv.org/html/2402.00157v1>. [Último acceso: 12 06 2024].
- [33] C. d. H. Face, «Open LLM Leaderboard,» 2024. [En línea]. Available: [https://huggingface.co/spaces/open-llm-leaderboard/open\\_llm\\_leaderboard](https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard). [Último acceso: 15 06 2024].
- [34] e. a. Hugo Touvron, «Llama 2: Open Foundation and Fine-Tuned Chat Models - Sección 3.2.1, "Human Preference Data Collection",» Meta, 19 Julio 2023. [En línea]. Available: <https://arxiv.org/pdf/2307.09288>. [Último acceso: 14 06 2024].
- [35] A. Surla, «How to Get Better Outputs from Your Large Language Model,» web deNVIDIA developer, 14 Junio 2023. [En línea]. Available: <https://developer.nvidia.com/blog/how-to-get-better-outputs-from-your-large-language-model/>. [Último acceso: 13 06 2026].
- [36] A. S. C. D. F. J. D. C.-R. P. J. L. Yixin Liu, «IMPROVING LARGE LANGUAGE MODEL FINE-TUNING,» Yale University, Google DeepMind, 16 octubre 2023. [En línea]. Available: <https://arxiv.org/pdf/2310.10047>. [Último acceso: 13 06 2024].
- [37] <https://www.youtube.com/@NeuralNine>, «Speech Recognition in Python,» Canal de youtube @NeuralNine, 28 03 2021. [En línea]. Available: <https://www.youtube.com/watch?v=9GJ6XeB-vMg>. [Último acceso: 13 06 2024].
- [38] W. H. Platform, «Woebot Health,» [En línea]. Available: <https://woebothealth.com/solution/>. [Último acceso: 9 marzo 2024].

## 9. Anexos

Se han añadido algunos anexos que complementan apartados previos de la memoria, añadiendo mas información de tipo más específico.

### 9.1 Anexo 1: Tabla con la temporalización del proyecto.

A continuación se presenta una tabla de las fases del proyecto junto a las tareas del tramo de desarrollo. Se incluyen todas las fases del proyecto, desde su idealización hasta la defensa final.

| Fases del proyecto                         |   | Fecha inicio | Ficha fin  |
|--|---|--------------|------------|
| Fase 0: Elección y definición del proyecto |   | 28-02-2024   | 11-03-2024 |
| Fase 1: Plan de trabajo                    |   | 12-03-2024   | 26-03-2024 |
| Fase 2: Desarrollo del proyecto, parte 1   |   | 27-03-2024   | 03-05-2024 |
| T1.  | Preparación del entorno de desarrollo en Python                     | 27-03-2024   | 30-03-2024 |
| T2.  | Selección de herramienta visual en Python para la interfaz          | 31-03-2024   | 02-04-2024 |
| T3.  | Selección y testeo de grandes modelos de lenguaje                   | 03-04-2024   | 09-04-2024 |
| T4.  | Testeo adicional y reevaluado de LLMs                               | 10-04-2024   | 20-05-2024 |
| T5.  | Desarrollo del chat con memoria                                     | 10-04-2024   | 23-04-2024 |
| T6.  | Diseño de una versión prototipo para integrar el chat               | 24-04-2024   | 01-05-2024 |
| T7.  | Elaboración de informe de seguimiento para la fase 2                | 26-04-2024   | 03-05-2024 |
| Fase 3: Desarrollo del proyecto, parte 2   |   | 04-05-2024   | 29-05-2024 |
| T8.  | Desarrollo, optimización e inserción de un minijuego en la interfaz | 04-05-2024   | 12-05-2024 |
| T9.  | Finalización de la interfaz   | 13-05-2024   | 17-05-2024 |
| T10.                                       | Incorporación de Texto a Voz  | 18-05-2024   | 20-05-2024 |
| T11.                                       | Testeo, ajustes y pruebas   | 21-05-2024   | 26-05-2024 |
| T12.                                       | Elaboración de informe de seguimiento para la fase 3                | 23-05-2024   | 29-05-2024 |
| Fase 4: Redacción del paper                |   | 30-05-2024   | 16-06-2024 |
| Fase 5a: Elaboración de la presentación    |   | 17-06-2024   | 23-06-2024 |
| Fase 5b: Defensa pública del proyecto      |   | 24-06-2024   | 30-06-2024 |

*Ilustración 30: Tabla con las tareas realizadas y el tiempo que han ocupado.*

*En color verde las fases principales, y en color salmón, tareas del tramo de desarrollo.*

## 9.2 Anexo 2: Diagrama de Gantt del proyecto

La siguiente sería la planificación temporal llevada a cabo, de las tareas comentadas en el Anexo 1, mediante un gráfico Gantt:

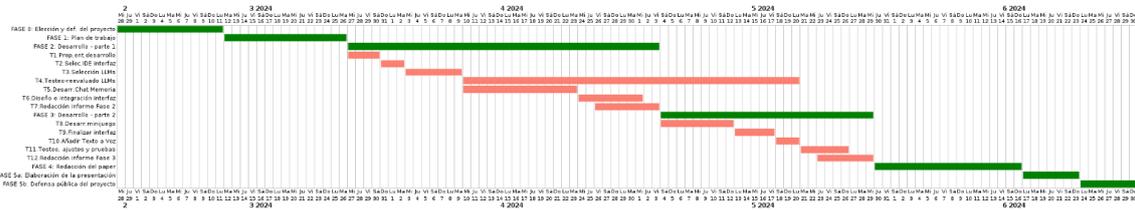


Ilustración 31: Tabla resumen de tareas, con su temporalización.

Se puede apreciar mejor la Figura 2 en su archivo de imagen original, en [este enlace](#).

## 9.3 Anexo 3: Ficheros del programa

El proyecto se compone de los siguientes ficheros principales:

- [ChatnGame.py](#) ← programa lanzador.
- [LLM\\_manager.py](#) ← gestión de los modelos de lenguaje (LLM)
- [chat\\_config.py](#) ← variables importantes para el chat que se usan entre las clases
- [main\\_interface.py](#) ← principal interfaz gráfica. No incluye la del minijuego.
- [TicTacToe\\_minigame.py](#) ← el minijuego elegido, un 3 / 4 en raya. Incluye su lógica y una mini-interfaz que irá en un Qframe de la interfaz principal.
- [utils.py](#) ← funciones extra, como el texto a voz.

Hay un séptimo fichero, que únicamente usa un string. Contiene información personal del usuario, para poder usar el modelo online Command, de la empresa Cohere:

[online\\_config.py](#) ← archivo para la key online. No lleva key al ser personal, se ha de pedir una key para la API en la [web de Cohere](#).

El contenido de los archivos se pueden ver en Github, en [el siguiente enlace](#).

Para el uso del prototipo se requiere la descarga de los LLM locales cuantizados que usa el prototipo, sus enlaces de Hugging Face se encuentran en el enlace de Github comentado.

#### 9.4 Anexo 4: Video de muestra del prototipo

Se adjunta un video que recoge los mismos momentos de la sesión prueba con el prototipo, que se describe en el apartado [5.1.Pruebas](#). El video, junto a los momentos que se explicarán más adelante, complementa lo explicado en la sección de pruebas.

Para acceder al video se ofrecen dos opciones:

- la recomendada, alojada [aquí](#).
- Si la opción anterior, alojada en streamable no funcionase, se ha creado una versión muy comprimida en el archivo *ChatNGame\_test\_lowdef.mp4* en [este enlace en github](#). No permite streaming y sólo permite descarga.

En el video se puede apreciar el uso del lector de voz, accesible desde opciones, y de cómo puede cortar la voz del programa si hay una nueva respuesta que tenga que decir en ese momento.

#### **Algunos momentos del video:**

**Inicio** (en realidad, la grabación empieza varios segundos después): el prototipo se ha iniciado con las opciones por defecto y el LLM inicial por defecto ha respondido a la pregunta introductoria automática.

**00:01** Mistral 7B responde al prompt que se ha activado tras transcurrir el primer ciclo del temporizador para el modo automático. El primer ciclo se activará antes dado que el programa, en su iniciación, tarda en cargarse.

**00:15** Interactuando con el minijuego en su versión por defecto, 3x3 en dificultad Normal.

**01:00** transcurrido un minuto (el tiempo por defecto), se activa otro prompt automatizado.

**1:12** Vistazo a las opciones de la segunda pantalla. Se activa lector de voz.

**1:25** Respuesta de Mistral 7B a un prompt del usuario, con el lector de voz.

**2:18** Tic Tac Toe en su versión de 4x4 y dificultad Retante.

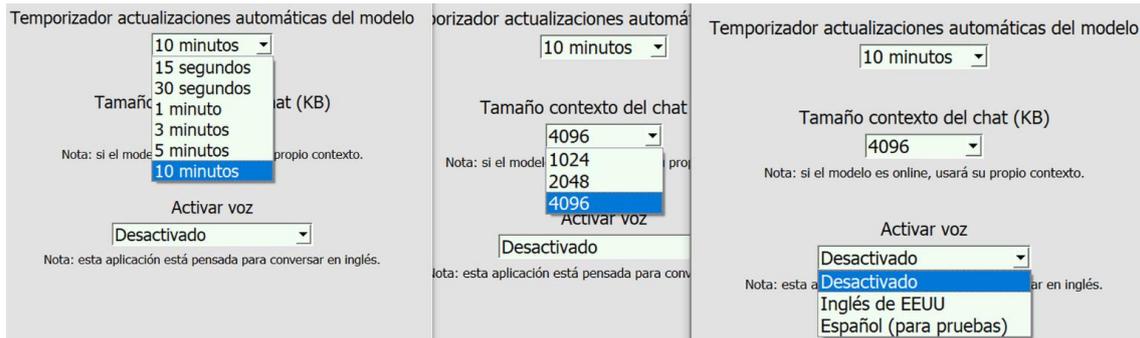
**2:44** Cambio de LLM activo: de Mistral a Llama. Se puede notar que hay unos segundos de espera, aunque no afecta al proceso creado aparte que permite al lector, hablar.

**3:09** Comportamiento no deseado de Llama 2: respuesta disruptiva.

**4:19** Cambio de LLM activo: de un modelo local, a uno online (Command de Cohere). Se puede apreciar que la transición es mucho más transparente que al cargar un modelo local, al estar el modelo alojado en el servidor.

**5:43** Se vuelve a usar el LLM Mistral 7B.

Nota: por algún motivo, el programa de captura de video de Xbox Game Bar integrado en Windows 10, no registra la apertura de listados de las combobox. Se adjunta una muestra visual de sus contenidos:



*Ilustración 32: Muestra de opciones.*

En la mezcla de capturas de la imagen anterior, se pueden ver las opciones disponibles de la pantalla de opciones.