REPUBLIC OF TURKEY

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

LLM INTERFACED MPC USING UNREAL ENGINE

KAĞAN ÇAKIROĞLU

BACHELOR'S THESIS

COMPUTER ENGINEERING

GEBZE

2025

REPUBLIC OF TURKEY

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

LLM INTERFACED MPC USING UNREAL ENGINE

KAĞAN ÇAKIROĞLU

BACHELOR’S THESIS

COMPUTER ENGINEERING

SUPERVISOR
ASST. PROF. YAKUP GENÇ

GEBZE
2025

	BACHELOR’S THESIS JURY APPROVAL FORM

The thesis of Kağan Çakıroğlu, which was defended on 12/01/2025 in front of the jury formed by the decision of the board of Gebze Technical University, Faculty of Engineering number 314/159 dated 10/10/2024, was accepted as Bachelor’s Thesis in the field of computer engineering.

JURY

Member (Supervisor)	:	Asst. Prof. Yakup Genç
Member	:	Asst. Prof. Yusuf Sinan Akgül

APPROVAL

The decision of the board of Gebze Technical University, Faculty of Engineering number 314/159 dated 10/10/2024.

Signature/Stamp

ABSTRACT

Advancements in artificial intelligence and game development frameworks have enabled the creation of more immersive and dynamic virtual environments. This project focuses on the development of an AI Character in Unreal Engine, capable of interacting with users through a chat system powered by LLama, a state-of-the-art large language model (LLM).

The system integrates the conversational capabilities of LLama with Unreal Engine's Behavior Trees to facilitate intelligent decision-making. Through real-time chat interactions, the AI Character interprets user inputs, generates context-aware responses, and dynamically creates or modifies in-game objects. This seamless interaction between generative AI and game systems enhances the depth and realism of gameplay, providing users with a highly engaging experience.

The primary objective of the project is to demonstrate the potential of conversational AI in interactive environments by delivering consistent interactions, efficient runtime responses, and advanced object manipulation capabilities. The results are expected to pave the way for scalable, user-driven experiences in gaming and virtual simulations.

Keywords: Unreal Engine, AI Character, LLM (Large Language Model), LLama, Chat System, BehaviorTrees

ÖZET

Bu proje, Unreal Engine'de LLama adlı büyük dil modeliyle desteklenen bir AI Karakterin geliştirilmesini ele almaktadır. Karakter, gerçek zamanlı bir sohbet sistemiyle kullanıcılarla etkileşime geçerken, Davranış Ağaçları sayesinde bağlama uygun kararlar alarak oyun içindeki nesneleri dinamik olarak oluşturup değiştirebilmektedir.

Amaç, konuşma tabanlı yapay zekanın sanal ortamlarda sunduğu potansiyeli göstermek ve kullanıcıya sürükleyici bir deneyim sunmaktır. Proje, oyun ve simülasyonlar için ölçeklenebilir ve yenilikçi çözümler sağlamayı hedeflemektedir.

Anahtar Kelimeler: Sohbet Sistemi, Davranış Ağaçları, Gerçek Zamanlı Etkileşim, AI Karakter

ACKNOWLEDGEMENT

We would like to extend our deepest gratitude to Dr. Öğr. Yakup Genç for his exceptional guidance, support, and encouragement throughout the development of this project. His vast knowledge, thoughtful insights, and unwavering commitment have been invaluable in overcoming challenges and achieving our objectives. His ability to inspire and foster growth has motivated us to continually strive for excellence.

We are particularly grateful for the time and effort he dedicated to providing constructive feedback, which played a crucial role in refining our work. His mentorship has not only enriched this project but has also significantly contributed to our personal and professional development. This project would not have been possible without his dedication and support, and we are truly honored to have had the opportunity to learn under his guidance.

Sincerely,
Kağan Çakıroğlu

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation		Explanation
AI	:	Artificial Intelligence
LLM	:	Large Language Model
BT	:	Behavior Trees
–chat-interface	:	Real-time communication system for user interactions
–response-time	:	Time taken by the system to generate responses
–input-context	:	User-provided context for interaction
–conf-thres	:	Confidence threshold for AI-generated decisions
–action-thres	:	Threshold for triggering object modification or creation
–ai-model	:	Path to the pre-trained LLM model
–device	:	Specifies the device for computation (CPU/GPU)
UE	:	Unreal Engine
NPC	:	Non-Player Character (AI Character)
CPU	:	Central Processing Unit
GPU	:	Graphics Processing Unit
LLama	:	Large Language Model used for conversational AI
RT	:	Real-Time Interaction

LIST OF FIGURES

LIST OF LISTINGS

1. USAGE

1.1. Introduction

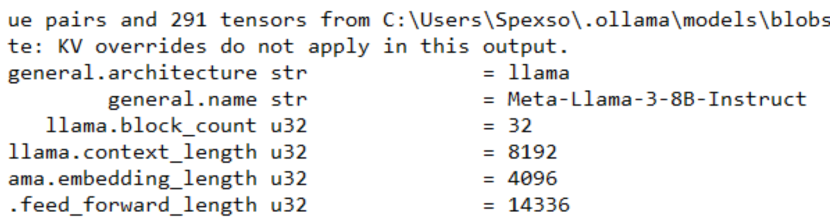
The integration of artificial intelligence (AI) with interactive platforms has revolutionized the way virtual environments are designed and experienced. This project explores the synergy between large language models (LLMs) and Unreal Engine to create a dynamic, interactive AI-driven system. At its core, the project leverages LLama, a state-of-the-art LLM, to facilitate seamless communication between users and an AI Character through a real-time chat system.

The AI Character serves as the central element of the system, using complex Behavior Trees to evaluate user inputs and make intelligent decisions. These decisions are not only reflected in its actions but also in its ability to interact with and modify objects within the virtual environment. By integrating modifiable objects directly into the AI's interaction loop, the system enables real-time adaptability and enhances user engagement.

By integrating cutting-edge AI with practical applications, this project aims to provide a reliable, automated solution for analyzing microscopic samples, reducing human error, and improving diagnostic efficiency.

1.2. LLM Integration

The integration of the LLama large language model (LLM) into Unreal Engine was implemented using Ollama, which hosts the model and provides an API for real-time communication. Ollama allows the LLama model to process user inputs and generate context-aware responses efficiently.



```
ue pairs and 291 tensors from C:\Users\Spexso\.ollama\models\blobs
te: KV overrides do not apply in this output.
general.architecture str              = llama
        general.name str              = Meta-Llama-3-8B-Instruct
   llama.block_count u32              = 32
llama.context_length u32              = 8192
ama.embedding_length u32              = 4096
.feed_forward_length u32              = 14336
```

Figure 1.1: Meta-Llama-3-8B-Instruct

In the system, a chat interface within Unreal Engine handles user interactions by sending inputs to Ollama's API. The LLama model processes these inputs and returns responses, which are used to guide the AI Character's actions. These responses are passed to Unreal Engine's Behavior Trees, enabling the AI Character to make decisions and perform tasks such as modifying objects based on user commands.

1.2.0.1. Communication with Unreal

Basic python Server script implements a TCP server to facilitate real-time communication between a client (e.g., Unreal Engine) and a LLama model hosted via Ollama. The server listens for incoming connections, receives messages from the client, and processes these messages using Ollama's chat API, which generates responses based on the input.

The processed response is then sent back to the client over the established TCP connection. The server includes robust error handling to manage client disconnections and ensures seamless reconnection by returning to the listening state when necessary. This setup enables dynamic, interactive communication, making it suitable for applications requiring AI-driven responses in real-time environments.

```python
    def get_incoming(self):
        while self.running:
            try:
                # Receive data from Unreal Engine
                data = self.client_socket.recv(4096)
                if not data:  # Check for empty data indicating client disconnect
                    print(f"Client disconnected: {self.client_address}")
                    self.client_socket.close()  # Close the client socket
                    self.listen()
                    continue  # Go back to the top of the loop for a new connection

                self.in_data = data.decode()  # Decode received bytes to string

                if self.debug:
                    print(f'Received from Unreal: {self.in_data}')

                # Process data with Ollama
                response = chat(
                    model="llama3",
                    messages=[
                        {
                            "role": "user",
                            "content": self.in_data,
                        },
                    ],
                )
                ollama_response = response["message"]["content"]
                print(f'Ollama response: {ollama_response}')

                # Send Ollama's response back to Unreal Engine
                self.send_data(ollama_response)

            except Exception as e:
                print(f"Error: {e}")
                self.running = False  # Stop the server loop if a critical error occurs
```

Listing 1.1: Local TCP Socket Server in Python

1.2.0.2. Handling data inside Unreal

Using tools like the UE4 TCP Socket Plugin, Unreal Engine can act as a TCP client, sending and receiving data through Blueprints while supporting multi-threaded and multiple connections for optimized performance. For projects requiring Python integration, the UE5 Easy Runtime Python Plugin allows Python scripts to be executed at runtime, enabling complex data processing and dynamic logic implementation.

Together, these plugins facilitate seamless data exchange by initializing a TCP connection, processing incoming data using Python scripts, and updating the Unreal environment in real-time based on received inputs.

```cpp
void ATcpSocketConnection::Connect(const FString& ipAddress, int32 port, const FTcpSocketDisconnectDelegate& OnDisconnected, const FTcpSocketConnectDelegate& OnConnected,
        const FTcpSocketReceivedMessageDelegate& OnMessageReceived, int32& ConnectionId)
{
        DisconnectedDelegate = OnDisconnected;
        ConnectedDelegate = OnConnected;q
        MessageReceivedDelegate = OnMessageReceived;

        ConnectionId = TcpWorkers.Num();

        TWeakObjectPtr<ATcpSocketConnection> thisWeakObjPtr = TWeakObjectPtr<ATcpSocketConnection>(this);
        TSharedRef<FTcpSocketWorker> worker(new FTcpSocketWorker(ipAddress, port, thisWeakObjPtr, ConnectionId, ReceiveBufferSize, SendBufferSize, TimeBetweenTicks));
        TcpWorkers.Add(ConnectionId, worker);
        worker->Start();
}
```

Listing 1.2: TCP Connection Establish

1.3. Local Player Character

The Player Character is equipped with the ability to perform raycasting, enabling it to identify and select objects that are instances of the ModifiableActorBase class. This interaction serves as a direct link between the Player and AI, allowing the Player to influence the AI's behavior dynamically. Upon selecting a ModifiableActorBase object, the Player can initiate interactions that impact the AI Character, such as triggering specific behaviors or responses.

Additionally, for debugging purposes, the Player can forcibly override and change the state of the AI's Behavior Tree. This feature is invaluable for testing and fine-tuning AI behavior, ensuring seamless interactions and functionality in diverse gameplay scenarios.

1.3.1. RayCast Ability

The raycast ability of the Player Character allows it to project a line trace from its camera or viewpoint into the game world to detect and interact with objects. In our case its origin point is our camera a crosshair drawn into the middle of screen. This line trace identifies instances of the ModifiableActorBase class by checking for collisions with their components, such as meshes or colliders. Upon a successful hit, the Player can select the object, triggering interactions or updates tied to its functionality.

This mechanism enables precise targeting of specific game objects, facilitating intuitive control over AI interactions, such as highlighting objects of interest or initiating commands. The raycast ability enhances the Player's agency in the game, bridging their actions with AI-driven events and providing a foundation for rich, interactive gameplay.

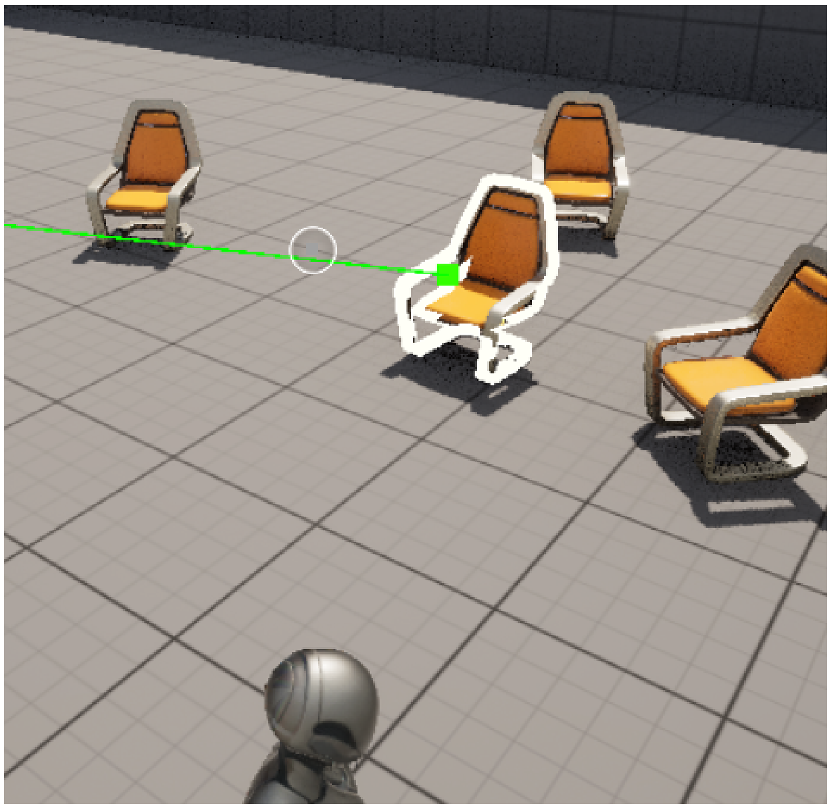


Figure 1.2: Ray-Cast Object Selection

1.4. Base Modifiable Actor Class

The ModifiableActorBase is a foundational class in Unreal Engine designed for objects that are exclusively interacted with by AI. It serves as the base for all dynamically modified or spawned objects, providing functionality to adjust material properties, scale, and rotation at runtime. The class ensures modularity and extensibility by enabling dynamic material initialization, making objects visually adaptable based on gameplay or AI behavior.

Its lightweight and self-contained design supports efficient spawning and interaction, while leveraging Unreal's component-based architecture for flexibility. This setup makes ModifiableActorBase an ideal choice for creating dynamic, AI-driven gameplay elements.

```
public:

        UFUNCTION(BlueprintCallable, Category = "CustomMethods")
        virtual void ChangeMaterialColor(FName ParameterName, const
            FLinearColor& NewColor);

        UFUNCTION(BlueprintCallable, Category = "CustomMethods")
        void ChangeMeshScale(const FVector& NewScale);

        UFUNCTION(BlueprintCallable, Category = "CustomMethods")
        void RotateMesh(const FRotator& NewRotation);

        // Helper to initialize the dynamic material instance
        void InitializeDynamicMaterial();

        UFUNCTION(BlueprintCallable)
        FLinearColor GetResolvedColor(ECommonColors Color);
}
```

1.4.0.1. Dynamic generation of ModifiableActorBase Class Objects

To support the customization and dynamic behavior of ModifiableActorBase objects, utility methods like those in UColorMappingHelper provide seamless functionality for changing colors, sizes, and materials. The ECommonColors enumeration maps a set of predefined, easily recognizable color names to their corresponding FLinearColor values, enabling intuitive color changes for both developers and AI-driven interactions.

Additionally, helper methods, such as StringToEnumCustom, allow flexible conversion between string inputs and enumeration values, further simplifying customization workflows. These utilities, combined with the dynamic material and transformation logic in ModifiableActorBase, create a modular and extensible framework for visually and interactively modifying game objects during runtime, ensuring efficient integration into gameplay and AI systems.

1.5. AI Player(MPC)

The AI system is designed to dynamically interact with the game world and adapt its behavior in real-time. At its core, an AI Controller continuously updates a BehaviourState field stored in the GameInstance, ensuring a centralized and consistent state representation that influences decision-making. The AI Player, controlled by this AI Controller, actively monitors its environment by checking its trigger zones, enabling it to spawn UI elements for the player or identify and interact with nearby objects.

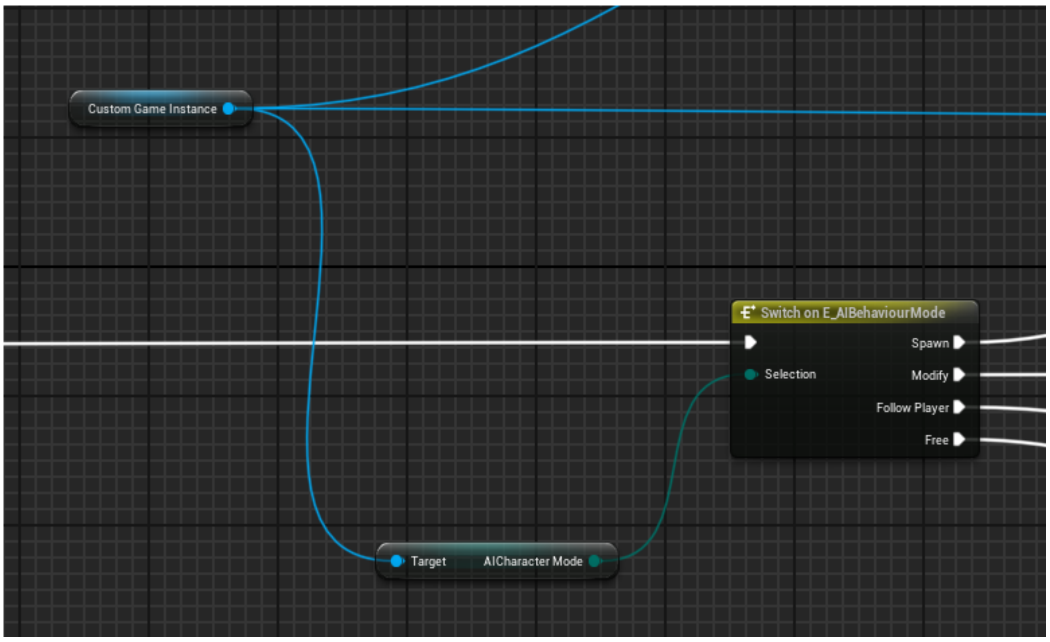


Figure 1.3: AI Player State Switcher

This interaction is orchestrated through a Behavior Tree that defines modes such as Generating (creating or modifying game elements), Waiting (idle or observation states), and Following (tracking or pursuing specific entities). Together, this system provides a flexible, modular approach to AI-driven gameplay, with real-time state management, environmental awareness, and tailored behavior modes that enhance player engagement and environmental interactivity.

1.5.0.1. AI PlayerController

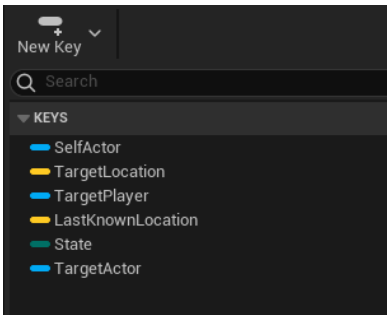
The Player Controller integrates advanced AI functionality through the AIPerception module and a Path Following Component, enabling rich, sense-driven interactions and navigation capabilities. The AIPerception module allows the AI to perceive its environment through various senses, such as sight, hearing, and touch, simulating realistic sensory input.

For instance, the sight sense enables the AI to detect objects or characters within its line of sight and react accordingly, making it aware of its surroundings and responsive to dynamic changes in the environment. Complementing this, the Path Following Component empowers the AI to traverse the game world by navigating the NavMesh. This ensures efficient, obstacle-aware movement, allowing the AI to follow paths to specified locations or chase targets seamlessly. Together, these systems create a robust framework for perception-driven decision-making and spatial navigation, making AI interactions more immersive and lifelike.

1.5.0.2. Blackboard of Behaviour Tree

In Unreal Engine, a Blackboard is a data storage system used in AI Behavior Trees to manage and share key information, enabling decision-making and coordination. It acts as a dynamic database where AI can store variables like targets, states, or locations, which are then referenced by Behavior Tree nodes to drive logic.

In the described setup, the Blackboard includes fields such as BehaviourState, which tracks the AI's current mode (e.g., generating, waiting, following), and TargetObject, which holds references to specific objects the AI needs to move towards or interact with. These fields are critical for the AI's functionality, as they guide the flow of the Behavior Tree and ensure actions align with the AI's current objectives and environment.



SelfActor
TargetLocation
TargetPlayer
LastKnownLocation
State
TargetActor

Figure 1.4: Blackboard Fields

1.5.0.3. Behaviour Tree

A Behavior Tree in Unreal Engine is a framework used to define AI logic through a hierarchical structure of tasks, decorators, and services, making it easier to create complex, adaptive AI behaviors. It serves as the ”brain” of the AI Player, processing decisions and managing actions based on conditions and priorities.

Essentially functioning as a state machine, the Behavior Tree transitions between states or modes, such as generating, waiting, or following, based on the AI’s objectives and environmental input. By systematically executing and reevaluating tasks, it enables the AI Player to adapt dynamically to changes in the game world, ensuring intelligent and responsive behavior.

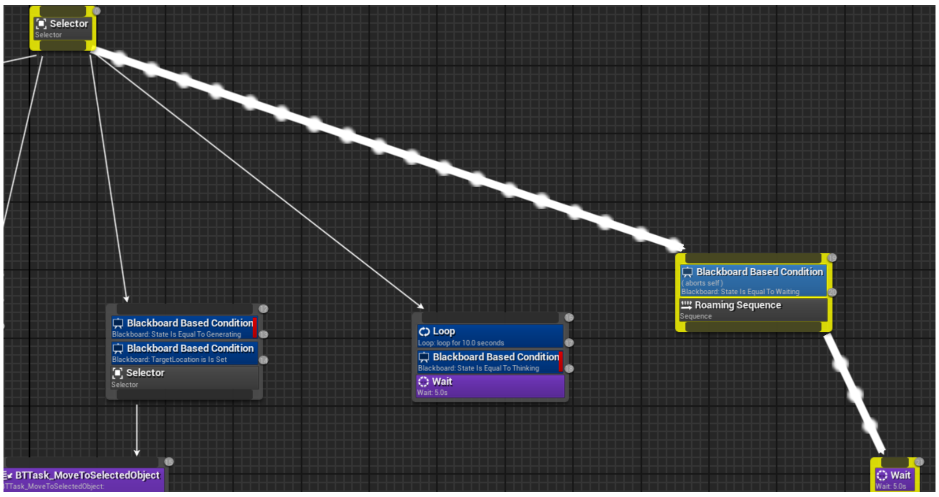

Selector
Selector
Blackboard Based Condition
Blackboard: State Is Equal To Generating
Blackboard Based Condition
Blackboard: TargetLocation is Is Set
Selector
Selector
BTTask_MoveToSelectedObject
Loop
Loop: loop for 10.0 seconds
Blackboard Based Condition
Blackboard: State Is Equal To Thinking
Wait
Wait: 5.0s
Blackboard Based Condition
(aborts self)
Blackboard: State Is Equal To Waiting
Roaming Sequence
Sequence
Wait
Wait: 5.0s

Figure 1.5: Behaviour Tree

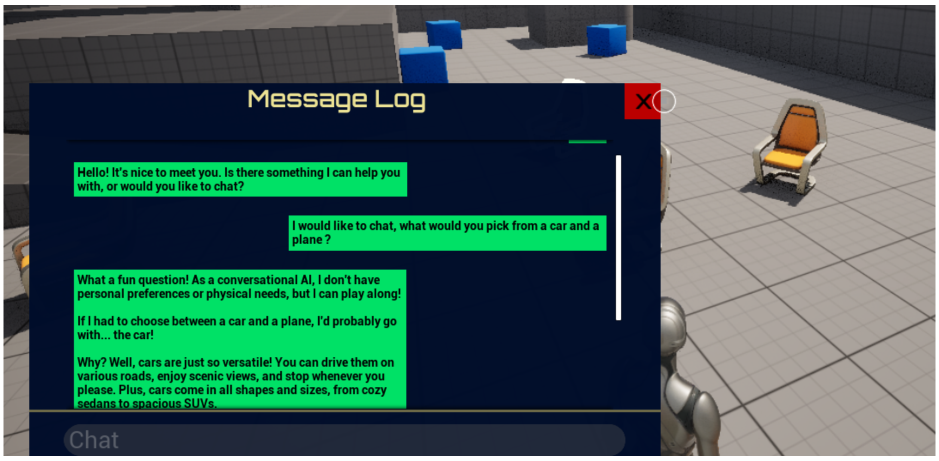
1.6. User Interface (UI) Implementation

The UI system is designed to provide players with both functionality and real-time feedback, enhancing interaction with the game world and AI. It features a text box that displays the last response of LLama, offering insights into the AI’s most recent actions or outputs, alongside another text box that shows the current state of the AI, such as generating, waiting, or following, for clarity on its active behavior mode.

Additionally, the Local Player has a HUD that includes a central crosshair for selecting the objects in the scene and an interactive chatbox for real-time communication with the LLM (Large Language Model). Elevates the player experience, blending practicality with immersion.

1.6.0.1. LLM ChatBox

The chatbox allows players to input queries and receive responses enabling dynamic interactions with LLM in our case LLama.



Hello! It's nice to meet you. Is there something I can help you with, or would you like to chat?

I would like to chat, what would you pick from a car and a plane ?

What a fun question! As a conversational AI, I don't have personal preferences or physical needs, but I can play along!

If I had to choose between a car and a plane, I'd probably go with... the car!

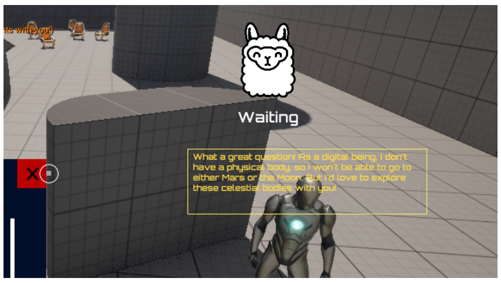
Why? Well, cars are just so versatile! You can drive them on various roads, enjoy scenic views, and stop whenever you please. Plus, cars come in all shapes and sizes, from cozy sedans to spacious SUVs.

Figure 1.6: Chat with LLama

1.6.0.2. AI Character 3D Widgets

The AI Character features 3D widgets that provide real-time visual feedback within the game world, enhancing interaction and transparency. These widgets display the AI's last response, offering insight into its most recent action or decision based on user inputs or environmental triggers. Additionally, the widgets show the AI's current Behavior Mode, such as generating, waiting, or following, as determined by its Behavior Tree.

Positioned near or above the AI Character, these widgets enable players to monitor the AI's state and responses directly within the environment, improving understanding of its behavior and creating a more interactive experience.



What a great question! As a digital being, I don't have a physical body, so I won't be able to go to either Mars or the Moon. But I'd love to explore these celestial bodies with you!

Figure 1.7: 3D Widgets

2. CONCLUSION

This project demonstrates the integration of Unreal Engine with large language models (LLMs), specifically LLama, to build an interactive and adaptive AI-driven system. By connecting LLama through a real-time communication pipeline hosted externally using Ollama, the AI Player processes user inputs and environmental changes using Behavior Trees and Blackboard data for decision-making. While this implementation provides a functional approach, integrating LLama directly into Unreal Engine without relying on external hosting would further streamline the system, reduce latency, and simplify deployment.

The ModifiableActorBase class, along with raycasting functionality, allows players to interact with and influence AI behavior effectively. The UI system complements these features by providing clear feedback through a chatbox for communication with LLama and displaying the AI's current state. The AI uses perception and navigation components to interact with the environment and execute tasks like moving to targets or detecting objects.

Integrating LLMs directly into Unreal Engine would enhance real-time processing and make the AI framework more self-contained, enabling broader applications in gaming, simulation, and other interactive virtual environments. This work establishes a solid foundation for further development in integrating advanced AI models with Unreal Engine.

BIBLIOGRAPHY

[1] EpicGames, *Unreal artificial intelligence*, 2024. [Online]. Available: `https://dev.epicgames.com/documentation/en-us/unreal-engine/artificial-intelligence-in-unreal-engine`.

[2] K. Çakıroğlu, *Project's github repository*, 2024. [Online]. Available: `https://github.com/Spexso/LLM_Interfaced_NPC_Using_Unreal_Engine`.

[3] Meta, *Llama official website*, 2024. [Online]. Available: `https://www.llama.com`.

[1] [2] [3].