January 15, 2025

Unreal Narratives: Enabling story generation by integrating LLM inside Virtual Reality environments

Pegah Hakimi
Farhad Sherali
Nihad Mujcic
Kalid Muhumed
Josef Ziada
Tareq Karaja

SEP400 Software Engineering
Department of Engineering Science

Contents

1 Introduction

The integration of artificial intelligence (AI) into augmented reality (AR) holds great potential to transform how humans interact with its environment and narratives. However, the current state of AR technology presents limitations in achieving these goals. As this project is developed by students, limitations in resources and expertise pose additional challenges. To have these concerns in mind, this researched-based project explores the concept of storytelling, generated by a LLM within a virtual reality (VR) environment. By combining the technologies of Unreal Engine and an external Large Language Model (LLM) - service, such as chatgpt-4o, this project aims to generate adaptive, player-driven stories that enhance the immersive experiences of the VR world. The system can be seen as a feature in a game world, as the classes could be easily integrated into any Unreal Engine project.

The system is designed to enable LLM-generated stories to dynamically adapt to users' actions within a VR environment as they interact with their surroundings. The key solution for this problem is implemented inside Unreal Engine's C++ classes, where two main modules handle the calculations and the integration of the LLM services. The main module class, GameStateStoryGen, tracks user actions, logs environmental changes, and communicates with the external LLM service. The LLM uses these inputs to generate context-aware narratives, which are then displayed through a HUD to enhance the user's immersive experience.

This project was provided by OpenLab, a research-lab focusing on AR and VR, established and stationed at University West of Trollhättan. Our project (Unreal Narratives) is derived from a project from OpenLab labelled OL23 [15].

2 Background

2.1 Requirements

The requirements for this project can be divided into functional and non functional categories

The functional requirements listed below describe what the system initially should have done in terms of features and capabilities.

- The system shall generate real-time stories based on the user's current environment using Large Language Models (LLMs).
- The system shall analyze the user's environment and context in the virtual reality (VR) to feed data to the LLM for story generation.
- The system shall provide a visual or auditory interface for presenting the generated stories and nudges to the user.
- The system shall continuously monitor the user's environment and update the story in real time as the context changes.
- The system shall nudge the user towards specific actions or behaviors by providing contextually relevant suggestions generated by machine learning algorithms.
- The system shall ensure that the nudges provided by the AI are subtle and contextually appropriate, based on machine learning models.

After further discussion with the customers/stakeholders we identified a misunderstanding of the original requirements. The revised focus emphasized the following as must-haves:

- The system shall generate real-time stories based on the user's current environment using Large Language Models (LLMs).
- The system shall analyze the user's environment and context in the virtual reality (VR) to feed data to the LLM for story generation.
- The system shall provide a visual or auditory interface for presenting the generated stories and nudges to the user.
- The system shall continuously monitor the user's environment and update the story in real time as the context changes.

Non-functional requirements - these describe how the system should perform in terms of quality attributes.

- **Efficiency:**
    - The system shall process environmental signals and generate stories in real-time to maintain a seamless user experience in VR.
    - The system shall be optimized to operate with minimal memory consumption, ensuring it can run efficiently on the VR glasses with limited

computational resources.

- **Usability:**
  - The system shall provide a user-friendly VR interface, where users can easily interact with the environment.
- **Security:**
  - All data collected from users, if stored, must be securely managed and transmitted using encryption.
- **Dependability:**
  - The system shall be available and reliable during user operation.
  - The system shall be developed using modular architecture.

2.2 Planning

A significant part of the early stages was spent researching how to combine VR systems, Unreal Engine and LLMs in a way that suited our objectives. Almost everything was new and unfamiliar to us, requiring a steep learning curve. As we gained more knowledge and overcame challenges, we adapted our approach, using what we learned to refine the development process. Throughout the project development phase, a lot of things were changed including the roles, the goals and requirements due to confusion around understanding the expectations. This in turn led to a lot of replanning and readjusting to ensure we deliver the project on time. This agile and iterative method allowed us to overcome obstacles and improve the system step by step.

2.3 Tools and technologies

The tools and technologies used for this project include both hardware and software components. For the hardware we used the Meta Quest 3 VR headset for virtual reality and high performance computers, both provided by Openlab. For the software, Unreal Engine was our primary tool for creating the virtual environment, Meta Quest link software for enabling the VR functionalities within the Unreal Engine, as well as Visual Studio 2022 using c++ as our language to implement the functionalities of the environment and the communication between Unreal Engine and the Large Language Model (LLM). The LLM used in this project was OpenAI's ChatGPT-4o.

3 System Description

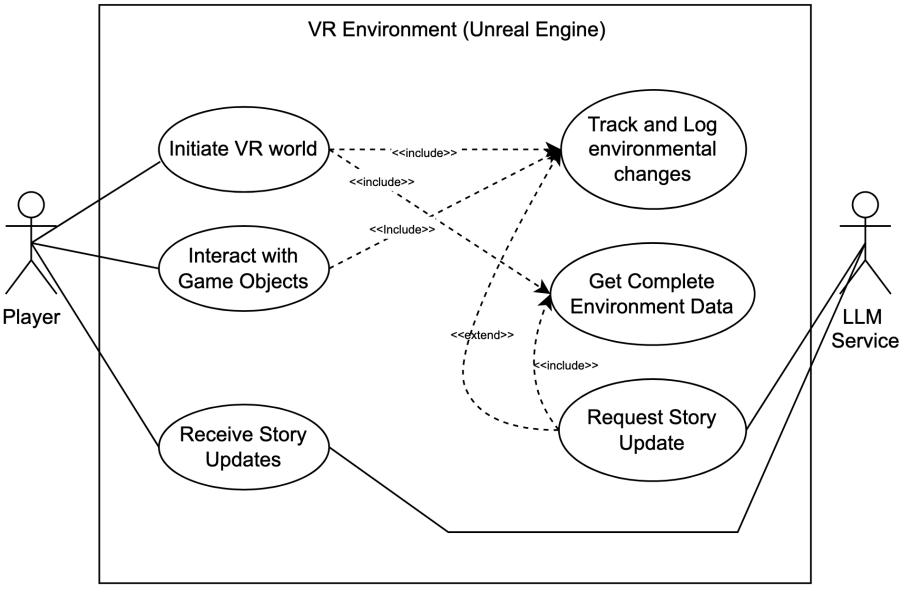

Figure 1. Use Case Diagram.

3.1 System Description based on the Use Case Diagram

Below, we describe our system based on the use case diagram in figure 1.

3.1.1 Base Use Cases

Player Initiates a VR experience

The player begins a VR session, entering an immersive game environment, such as a virtual forest.

Exploration and Interaction

The player explores their surroundings and interacts with game objects. As the player engages with the environment, the system dynamically generates a story that evolves based on the player's recent activities, creating an adaptive narrative experience.

Dynamic Narrative Development

As the exploration continues, the narrative evolves continuously, incorporating the player's past and present actions along with the changes in the environment. This ensures an engaging storytelling experience.

3.1.2 Additional Cases

Environmental Initialization and State Tracking

When the player initiates the VR world, the system generates the environment and begins tracking states, logging player interactions and environmental changes.

Data Gathering and Story Generation

After a series of interactions, the system compiles data into a payload and sends it to a Large Language Model (LLM) with a request for story generation.

Story Presentation via HUD

The LLM returns a generated story, which is processed by the system and displayed to the player through a Heads Up Display (HUD).

Continues Interaction Logging

As the player continues interacting with the environment, the system tracks new activities and updates the payload with the latest data, and incorporates the LLM's previous responses. This gives the LLM a guideline to keep the narrative consistent.

4 System Architecture

Below we illustrate and explain our system architecture using three different UML diagrams. The diagrams are meant to complement each other to give a full description.

4.1 Component Diagram

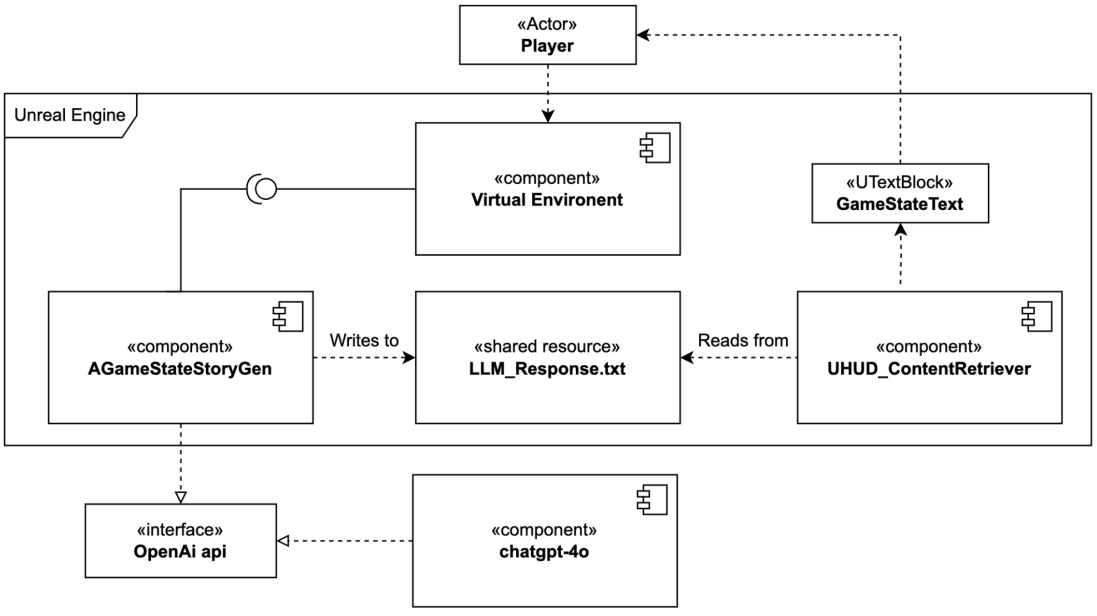

Figure 2. Component Diagram of the system.

The component diagram in figure 2 illustrates the architecture of components in the system built in Unreal Engine for dynamic story generation using an external large language model (LLM). The player interacts with the virtual environment, which is the bridge between player actions and the game state. The AGameStateStoryGen component tracks game state changes and communicates with the external LLM service through the OpenAI API. The responses from the LLM are written to a shared resource (LLM_Response.txt), which is then read by the UHUD_ContentRetriever component to update the Head Ups Display (HUD), which uses GameStateText element for display.

4.2 Class Diagram

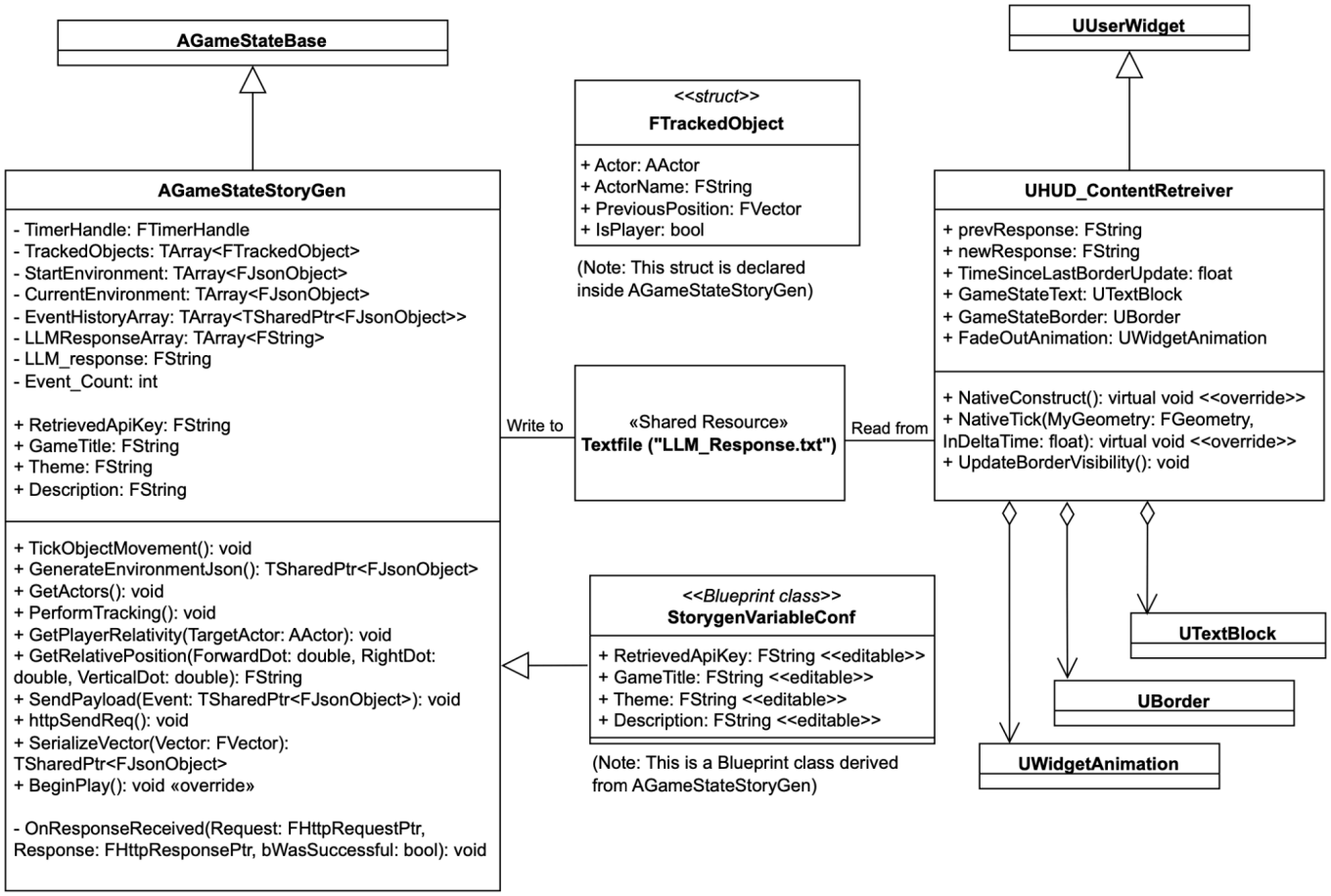

Figure 3. Class Diagram of the system.

The class diagram in figure 3 provides a detailed representation of the system architecture, focusing on the implemented C++ classes and their relationship with the Unreal Engine project. At the core of the system, is the AGameStateStoryGen, a class which is derived from AGameStateBase, responsible for managing game-state logic, including actor tracking, getting environmental data, and interaction with the external LLM-component.

The class includes essential attributes: RetrievedApiKey, GameTitle, Theme and description, which are editable in UE editor and exposed in the blueprint class StoryGenVariableConf. The StoryGenVariableConf blueprint class inherits from AGameStateStoryGen and offers support for editing attributes that is essential for the context of story generation and communication with the openai's LLM service.

The diagram also includes the UHUD_ContentRetriever, derived from UUserWidget, which manages the UI elements, GameStateText and GameStateBorder, connecting via aggregation to UTextBlock and Border, respectively. It reads from the shared resource, LLM_Response.txt, a text file which LLM responses are written to by the AGameStateStoryGen's OnResponsereceived function. The content inside the LLM_Response.txt is set to the GameStateText UTextBlock element, which enables the LLM response to be displayed to the player.

Finally, the FTrackedObject struct declared within AGameStateStoryGen is represented in its own block to show the declared attributes. FTrackedObject struct is used as data type for the TrackedObjects array.

5 Design and implementation

5.1 Frontend

The front-end design of this project focused on creating a simple and user-friendly interface that integrated efficiently with the VR environment. Combining Blueprints with C++ for functionality allowed the project to work as intended, and dynamic HUD updates, and intuitive navigation, providing a cohesive and immersive experience.

5.1.1 Environment

The environment was created by using the built-in "First Person" template inside the Unreal editor, serving as a foundation for an immersive and interactive virtual space. This template came with the necessary configurations that were needed for the functionality of the environment such as a player character, control mappings, and a basic environment featuring walls and a floor. For the design of the environment, the decision was made to simulate a "magical forest" where different trees, rocks and even mountains were placed to populate the forest using the fab library [2]. This approach aimed to create a setting that was both visually appealing and thematically cohesive with the narrative outputs generated by the Large Language Model (LLM).

The forest elements such as trees, rocks and other natural objects were tagged with metadata such as type and description to enable the LLM to generate descriptive and relevant stories about the "magical forest". But the main objects for driving the story generation process were movable actors such as balls and cubes, which were designed to interact dynamically with the player and the environment. Adding the interact property was as simple as enabling the "simulate physic" option for relevant actors.

Originally only the movable objects were created to simplify the development and testing of the project while also ensuring stability and functionality. This approach allowed for a better flexibility to explore both the interaction and narrative mechanics quickly before expanding the environment to include the visually rich and thematically immersive "magical forest.

5.1.2 Heads-Up Display (HUD)

To display the LLM responses to the VR-user, we used a Heads-Up-Display (HUD) [3]. Initially, a 3D plane was implemented by combining a widget as an actor, allowing it to be placed in the environment in a fixed place or location. This static placement provided limited utility, as it required the user to look in specific directions to read the content, breaking immersion. However, After some discussions on how to improve the system, a 2D HUD was chosen as a better alternative. This approach provided easier access and a more immersive interaction with the output in a VR environment which ensured the content was always visible without requiring specific head or body movements. The system was implemented using Unreal Engine's widget component and displayed directly on the user's viewport through Blueprint logic. Key challenges included fine-tuning the HUD background so it was placed in the correct place and that the output text was wrapped correctly inside the HUD border. This issue was solved by placing the border and text block inside a dynamic border called size box. Another tricky problem we faced was that another copy of the same content would show up in the wrong place which interrupted and limited the visibility of the actual HUD content. This was fixed by anchoring the content to the middle of the canvas panel of the HUD as setting the anchor in other places wouldn't translate correctly in VR.

5.1.3 Virtual Reality

The Virtual Reality (VR) element was the core feature of this project which was implemented using Unreal Engine's VR template [4]. The template came with essential features such as a VR-specific player character, also known as VR Pawn [5], motion controller components, enabling hand-tracking and natural interactions with objects in the virtual environment, teleportation Mechanics, offering a user-friendly method of navigating the environment and interaction mechanics tailored for virtual reality.

Using this template instead of creating one from the ground significantly reduced the development time, allowing the team to focus on interaction and functional mechanics to suit the project's goals. It allowed for a seamless transition from the initial first-person mode to a fully interactive VR experience. It also allowed for a two way debugging as game modes between First-person mode and VR mode could be switched seamlessly.

Integrating the VR Template came with some challenges, such as ensuring smooth interactions between the VR mechanics and the existing first-person logics. Initially, the codes and functionality were written based on the First-Person Mode, with the expectation that they would also work seamlessly in the VR Template. However, during testing, it became clear that certain functionalities did not carry over as intended. The issue was in the inheritance of the "projectgamemode.cpp" classes. he VR Template's default game mode did not inherit from the First-Person Game Mode, resulting in discrepancies in how inputs, interactions, and gameplay logic were handled. To resolve this, the team changed the parent class of the VR Game Mode to inherit from the First-Person Game Mode Blueprint. This adjustment ensured that the functionality written for the First-Person Mode was fully compatible with the VR Template.

5.1.4 Joystick

To provide a more immersive alternative to teleportation, joystick locomotion was implemented in the VR environment. The left joystick controlled movement along the X and Y axes, enabling forward, backward, and lateral motion, while the right joystick was used for smooth player rotation. This setup offered a more natural and intuitive way for players to navigate the environment.

The locomotion system was built using Unreal Engine's input mappings and Blueprints [6]. Input axes were defined for movement and rotation, with the left joystick mapped to control movement direction and the right joystick mapped to rotation. In the VR Pawn's Blueprint, these inputs were linked to movement and rotation nodes (Add Movement Input and Add Controller Yaw Input), allowing for responsive and fluid controls.

To ensure the system felt intuitive, movement speed and rotation sensitivity were fine-tuned during testing. Smooth rotation was used instead of snap turning to reduce motion sickness and enhance player comfort. By integrating joystick locomotion, players could explore the environment dynamically, making interactions with objects more seamless and engaging.

5.2 Backend

The backend logic for our system is implemented in one single class called AGameStateStoryGen that is derived from GameState in Unreal Engine [7]. It consists of many functions and data structures which together creates the solution to produce the story generation with the help of LLM integration. The many functions together create a payload which is processed by the LLM. For this payload, there are functions that handle environmental data, movement tracking in the environment, positioning and relativity between moving objects and player, and logic to handle these by logging and calculating for when and how to compile together a payload and to send it to the LLM for processing.

The payload content is a json-structure that is designed using the important variables about objects of environment and world data. It includes all the important data that can help the LLM to generate the stories that are requested. Below we go into detail about the aspect of this logic which is the most interesting and important for our solution.

5.2.1 Dynamic Environment Serialization

The primary logic in the AGameStateStorygen class focuses on gathering broad and general data about the environment when the player initiates the world. This is handled by the GenerateStartEnvironment() function. It generates a JSON structure that combines all relevant information needed to provide the LLM with an overview of the initial game environment.

The function is also called whenever the system requests the LLM to generate a new story. Within this implementation, we compile the data into a structured format, which was refined multiple times. The final structure includes key aspects of the game environment, such as the game title, theme, world description, and spatial boundaries. Additionally, an array of all game objects is included.

Interestingly, the data structure went through at least ten revisions before we arrived at a version that balanced simplicity and utility. The goal was to ensure the JSON structure

captured essential details without being overly complex, as a large JSON payload could slow down communication with the LLM.

5.2.2 Filtered Actor Tracking

For tracking all the movements in the environment, a collection of all the objects is needed. These objects (called Actors in Unreal Engine [10]) are collected and stored inside an array which takes struct objects. The struct objects contain the important variables about the Actor including, Actor id, name, position and a boolean which checks if an Actor is a player.

After storing all actors and their data, the PerformTracking() function is called periodically to iterate through the TrackedObjects array. It compares the current position of each actor to its previous position. If a change in position is detected, it indicates that the actor has moved. The actor's new position is then forwarded for further analysis to determine its relativity to the player.

To solve this part, we began by creating a flowchart of the solution to this problem. We then implemented a system to collect all actors in the environment, and store them in an array of structs containing these key data. We then created the PerformTracking() function to periodically compare the positions.

5.2.3 Relative Positions

The class method GetPlayerRelativity() handles calculation of the relative positioning between player and the object in question. The function also takes help of the GetRelativePosition() method to get a string representation of the relative position. GetPlayerRelativity() creates a simple json object that contains the following parameters:

Object label: which specifies what object in the environment that is considered.

Distance from player: specifies the distance between the object and the player.

Relative position to player: which stores a string that specifies where the object is located relative to the player. We have a total of 14 pre-defined strings. Two examples are: "directly in front of the player" and "behind-right-below the player".

To calculate the distance, we first identify the positions, the forward and right vector of the object and the player in the 3D space. We then calculate the vector difference between the object and the player's position. The distance is then determined by calculating the magnitude of this vector. To find out where the object is and to get the relative position-string, we use dot products to get the direction of the object that is relative to the player's forward direction. This comparison is done using the dot product with the normalized relative vector [11][12]. Each dot product tells us how much the object aligns with each direction. GetRelativePosition() uses all these dot products to determine which of the predefined strings to choose using if-statements.

To get an representation of the relative positioning has been a long process, which required slow building and many re-designs and re-implementations. In the end we settled on using linear algebra for 3D-space as a solution for calculating the exact positioning that's relevant to the player's forward direction.

5.2.4 The final compiled payload

The AI model needed a detailed and contextually rich payload to generate meaningful narratives. We decided that Start environment, current environment, event history and old AI responses were the best payload content to feed to the AI.

The Start environment was essential to provide the AI with a clear understanding of the initial game state, allowing it to visualize the setting the player began in.

The current environment was also needed for the AI to grasp how the game world and the objects inside it changed after certain actions were made by the player, ensuring the generated narrative reflected the current game state accurately.

The Event History was the most important, by providing a detailed log of the player's previous actions, the AI could create a precise and engaging narrative of the player's journey.

5.2.5 LLM interaction and API

For LLM interactions, the two functions httpSendReq and OnResponseReceived communicate with the LLM service (OpenAi chatgpt-4o) through an api key [8].

The httpSendReq function builds a JSON payload containing the model specification,game state(starting environment, current environment, event history, old AI responses).This payload is then serialized and saved locally for debugging. The HTTP request is then configured with the API endpoint, headers, and the serialized payload.Upon sending the request, the function binds the OnResponseReceived callback to handle the API response.

The OnResponseReceived function processes the API's response once it is received. It validates the response to ensure it is successful and properly formatted. Then, it parses the JSON response to extract the AI-generated story from the "choices" array within the payload. The extracted story is logged for immediate review, saved to a file for persistence, and added to the local array of responses.

At the beginning of the project we chose the LLama 3.2 1b [9] because of the sustainability to not use as much energy as you would with other LLMs but as the project grew bigger we noticed that LLama 3.2 1b couldn't keep up which ended with responses that didn't reach our expectations. We decided because of this issue to switch the LLM from LLama 3.2 1b to the openAI as it gave better story narratives. Even though it used way more energy, it gave out faster responses and had better control of the environment as well as better control if we made bigger environments.

5.3 Testing

5.3.1 Unit tests

In our project, a unit test is used to ensure individual methods and components function correctly in isolation. Using Unreal Engine's Framework for testing (Automation tests) [x], we tested the behavior of the UHUD_ContentRetreiver widget to confirm its reliability during widget lifecycle events and state updates.

Our test summary: We have NativeConstructTest Test visibility setup during widget construct, NativeTickTest Tests file reading and state update in Tick.

UpdateBorderVisibilityTest Tests visibility logic based on text content. Challenges in Testing: Mocking GameStateText and GameStateBorder required precise setup to simulate real-world behavior.

The unit tests confirmed that UHUD_ContentRetreiver methods functioned as expected. These tests demonstrated reliable behavior for component initialization, state updates, and visibility handling.

The tests available for our backend logic within AGameStateStoryGen are tests that did not require a running world to get through.

6 Design analysis

The design was created from many trials and errors. We had to scratch our design and implementation many times, before we finally settled on one which we kept as base. Since that base structure of our classes and their methods, we built the code bit by bit, as new tasks were created. We learned many times how to not build this system and we are sure that if we kept designing this system from the beginning, it would be an even better program. As our time is limited, the latest implemented design became our final version.

The design is implemented into two distinct c++ classes. However in the first class, we have many components which should have been their own classes. But this implementation caused many errors because Unreal Engine's dependency management is complex and we didn't manage to understand it in time. The solution was to make things simple and have the main functionalities inside one class. If we would design and implement the system again, it would surely be more object-oriented.

There are not many alternatives to a component design when there are only two or three larger components. However, there are many different ways we could have connected these components by changing the way of communication and frameworks. The performance of the system was very important for us, so we avoided components that would have slowed down our program. For example, if there was a way to implement the system to get rid of unnecessary interfaces, we would do so. We also tried to stay away from slower programming languages such as python. Since Unreal Engine uses C++, we stuck to it completely.

7 Ethics and Sustainability

7.1 Ethical Implications: Story Generation

In our project, stories are generated in response to changes in the environment and user behaviors. While this feature introduces innovative and immersive storytelling, it also raises ethical concerns regarding content control and appropriateness.

Since the application relies on an external Large Language Model (LLM)-service to generate narratives dynamically, it runs the danger of containing the following:

- Political remarks: Content that could reflect sensitive viewpoints.
- Ethical dilemmas: Scenarios that may present moral implications.

- Marketing or product placements: references to brands or products.
- Adult or inappropriate content: Material unsuitable for the intended audience.

To mitigate these risks, the project could implement the following solutions:

- Rule-Based Preprocessing: Before sending prompts to the LLM, sanitize or organize them by defining limitations inside the main program.
- Self-Validation by LLM: Return generated responses to the LLM for validation with prompts such as: "if the response includes inappropriate or sensitive content, provide a version by removing that content."
- External Moderation Layer: Employ a moderation API to scan and flag inappropriate content.

By integrating these safeguards, the project ensures that story generation aligns with ethical guidelines and maintains a safe and engaging user experience.

7.2 Sustainability Implications: The LLM model in use

Data centers operate all hours everyday, consuming significant energy that is primarily from fossil fuels, contributing 2.8 to 3.7 % of the total global greenhouse gas emission [14]. The energy consumption is mainly from the processing power. According to [14], the data centers also accounted for 0.9 to 1.3 % of the entire global electricity use.

Our reflection is that for software systems that have integrated LLM, it's important to consider the amount of data that is transmitted and received to the LLM-service. If systems are able to reduce the amount of data, for example the payload, sent to the LLM for processing, it could save computational resources and reduce environmental impact.

8 Project reflection

The project as a whole presented significant challenges, as it had to be developed entirely from scratch. As a team, we struggled initially due to the lack of guidance or reference from previous projects, making it difficult to determine what needed to be done, what resources were required, and how complex the project would be. This created setbacks, as many of our early ideas were either unfeasible or did not meet the customer's expectations.

Through collaborative effort, extensive research, and continuous discussion, the team gradually developed a clearer understanding of how to approach the project. As we progressed, our workflow improved, and we were able to align our efforts more effectively with the project's requirements.

The team held regular meetings, typically 2-3 times a week, to discuss how to approach the next steps in the project. During these meetings, we outlined tasks, assigned responsibilities, and clarified what needed to be completed before the following meeting. To organize our workflow effectively, we used tools such as Kanban boards and GitHub. These tools ensured everyone was aware of pending tasks and provided a platform to share project files and ideas seamlessly.

When it came to problem-solving, the team relied heavily on extensive research and experimentation. We maintained open communication, regularly discussing the solutions we found and explaining how they could be implemented. This collaborative learning process helped the team collectively improve its understanding and adapt to the project's requirements.

References

[1] "OL23 LLM-based story generation for AI+AR glasses based on environmental and biometric sensing – Open Lab," Openlab.hv.se, 2016. https://openlab.hv.se/thesis/ol23-llm-based-story-generation-based-on-environmental-and-biometric-sensing/ (accessed Jan. 15, 2025).

[2] "Fab | Everything you need to build new worlds," Fab. https://www.fab.com/

[3] "User Interfaces and HUDs in Unreal Engine | Unreal Engine 5.5 Documentation | Epic Developer Community," Epic Games Developer, 2025. https://dev.epicgames.com/documentation/en-us/unreal-engine/user-interfaces-and-huds-in-unreal-engine (accessed Jan. 15, 2025).

[4] "VR Template in Unreal Engine | Unreal Engine 5.5 Documentation | Epic Developer Community," *Epic Games Developer*, 2024. https://dev.epicgames.com/documentation/en-us/unreal-engine/vr-template-in-unreal-engine (accessed Jan. 15, 2025).

[5] "VR Template in Unreal Engine | Unreal Engine 5.5 Documentation | Epic Developer Community," *Epic Games Developer*, 2024. https://dev.epicgames.com/documentation/en-us/unreal-engine/player-controllers-in-unreal-engine (accessed Jan. 15, 2025).

[6] ImpalerVR, "How to make Smooth Locomotion movement in VR | Unreal Engine 5," YouTube, Dec. 04, 2023. https://www.youtube.com/watch?v=aUBoYVXlMh8 (accessed Jan. 15, 2025).

[7] "Game Mode and Game State in Unreal Engine," Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/game-mode-and-game-state-in-unreal-engine

[8] "OpenAI Platform," Openai.com, 2025. https://platform.openai.com/docs/api-reference/audio/createTranscription (accessed Jan. 15, 2025).

[9] "Llama 3.2 | Model Cards and Prompt formats," Llama.com, 2022. https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/#-llama-3.2-lightweight-models-(1b/3b)- (accessed Jan. 15, 2025).

[10] "Static Mesh Actors in Unreal Engine | Unreal Engine 5.5 Documentation | Epic Developer Community," Epic Games Developer, 2025. https://dev.epicgames.com/documentation/en-us/unreal-engine/static-mesh-actors-in-unreal-engine (accessed Jan. 15, 2025).

[11] “4.7: The Dot Product,” *Mathematics LibreTexts*, Jun. 24, 2019. https://math.libretexts.org/Bookshelves/Linear_Algebra/A_First_Course_in_Linear_Algebra_(Kuttler)/04%3A_R/4.07%3A_The_Dot_Product (accessed Jan. 15, 2025).

[12] “9.2: Calculating Vector Length, Normalization, Distance and Dot,” Mathematics LibreTexts, May 25, 2021. https://math.libretexts.org/Bookshelves/Linear_Algebra/Matrix_Algebra_with_Computational_Applications_(Colbry)/09%3A_05_Pre-Class_Assignment_-_Gauss-Jordan_Elimination/9.2%3A_Calculating_Vector_Length%2C_Normalization%2C_Distance_and_Dot (accessed Jan. 15, 2025).

[13] “Automation Test Framework in Unreal Engine | Unreal Engine 5.5 Documentation | Epic Developer Community,” Epic Games Developer, 2025. https://dev.epicgames.com/documentation/en-us/unreal-engine/automation-test-framework-in-unreal-engine (accessed Jan. 15, 2025).

[14] R. Cho, “AI’s Growing Carbon Footprint,” State of the Planet, Jun. 09, 2023. https://news.climate.columbia.edu/2023/06/09/ais-growing-carbon-footprint/

[15]“OL23 LLM-based story generation for AI+AR glasses based on environmental and biometric sensing – Open Lab,” *Openlab.hv.se*, 2016. https://openlab.hv.se/thesis/ol23-llm-based-story-generation-based-on-environmental-and-biometric-sensing/