Τμήμα Επικοινωνίας & Ψηφιακών Μέσων
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΕΠΙΚΟΙΝΩΝΙΑΣ ΚΑΙ ΨΗΦΙΑΚΩΝ ΜΕΣΩΝ
ΣΧΟΛΗ ΚΟΙΝΩΝΙΚΩΝ ΚΑΙ ΑΝΘΡΩΠΙΣΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
Π.Μ.Σ. «ΑΝΑΠΤΥΞΗ ΨΗΦΙΑΚΩΝ ΠΑΙΧΝΙΔΙΩΝ ΚΑΙ ΠΟΛΥΜΕΣΙΚΩΝ ΕΦΑΡΜΟΓΩΝ»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Leveraging Large Language Models for Dynamic NPC Interactions in 2D RPGs

του
Τοπαλίδη Ιάσονα

Επιβλέπων Καθηγητής: Μηνάς Δασυγένης, Αναπληρωτής Καθηγητής

Μέλη της τριμελούς επιτροπής:

1) Πλόσκας Νικόλαος, Αναπληρωτής Καθηγητής
2) Πρωτοψάλτης Αντώνιος, ΕΔΙΠ

Μάρτιος 2025

COMMUNICATION AND DIGITAL MEDIA DEPARTMENT
SCHOOL OF SOCIAL SCIENCES AND HUMANITIES
MSc PROGRAM «GAMING AND MULTIMEDIA APPLICATION DEVELOPMENT »

MSc Thesis

Leveraging Large Language Models for Dynamic NPC Interactions in 2D RPGs

by
Topalidis Iasonas

Supervising Professor: Minas Dasygenis, Associate Professor

3-Member Examination Committee

1) Ploskas Nikolaos, Associate Professor
2) Protopsaltis Antonios, LTP

March 2025

Acknowledgements

I would like to thank my friends and family for supporting me through this endeavor. I would also like to give special thanks to my work manager M. Firopoulo who provided a stable environment allowing me to complete my studies.

Last but certainly not least, I would like to thank my professor Dr. Mina Dasygeni who guided me through this endeavor and helped me editing this thesis.

"It is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail." - Abraham Maslow

Εισαγωγή

Η παρούσα μεταπτυχιακή διατριβή παρουσιάζει μια νέα προσέγγιση στην προοδευτική δημιουργία περιεχομένου σε παιχνίδια ρόλων (RPG) μέσω την ενσωμάτωσης μεγάλων γλωσσικών μοντέλων (LLM). Η εργασία προσπαθεί να αντιμετωπίσει μια πρόκληση στην ανάπτυξη παιχνιδιών: την δημιουργία ενός πλούσιου, ισορροπημένου και διαδραστικού περιεχομένου το οποίο διατηρεί την προσοχή και την δεσμευση των παικτών, ενώ ταυτόχρονα απλοποιεί την διαδικασία και τους χρόνους ανάπτυξης του περιεχομένου.

Η αρχιτεκτονική του συστήματος αποτελείται από δύο (2) κύρια στοιχεία:

1) Έναν server ο οποίος επιμελείται των αλληλεπιδράσεων του LLM για την δημιουργία δυναμικού περιεχομένου που αποτελεί δομικό λίθο του παιχνιδιού. Αυτές οι δομές είναι ο κόσμος του παιχνιδιού, οι χαρακτήρες που τους χειρίζεται το παιχνίδι (NPC), οι αποστολές του παίκτη καθώς και οι μάχες που πρέπει να αντιμετωπίσει. Ο τελευταίος μηχανισμός του server είναι ένας μηχανισμός επικύρωσης και ελέγχου σφαλμάτων στις απαντήσεις του LLM.
2) Το παιχνίδι, μέσω του οποίου ο παίκτης μπορεί να έχει αλληλεπιδράσεις με τα NPC, να εξερευνήσει τις τοποθεσίες του κόσμου, να πολεμήσει τις μάχες, να εξελίξει τον χαρακτήρα του, καθώς και να ολοκληρώσει διάφορες αποστολές.

Η εφαρμογή αυτή αποδεικνύει ότι το σύστημα παράγει επιτυχώς συνεκτικούς φανταστικούς κόσμους με αφηγηματικά στοιχεία και λειτουργικά συστήματα. Παρατηρείται επίσης ότι το παραγόμενο περιεχόμενο προσφέρει σημαντική διαφοροποίηση σε κάθε παιχνίδι που πραγματοποιείται.

Η υλοποίηση αυτή συμβάλλει στον τομέα της ανάπτυξης παιχνιδιών με την καθιέρωση ενός μεθοδολογικού πλαισίου για την παραγωγή περιεχομένου με χρήση τεχνητής νοημοσύνης (AI) εξισορροπώντας την δημιουργική παραλλαγή με τους δομικούς αυτούς περιορισμούς. Τα ευρήματα αυτά υποδεικνύουν πολλές υποσχόμενες κατευθύνσεις για την κλιμάκωση της διαδικαστικής παραγωγής περιεχομένου στην ανάπτυξη εμπορικών παιχνιδιών μέσω της εφαρμογής μεγάλων γλωσσικών μοντέλων (LLM) σε εξειδικευμένα αρχιτεκτονικά πλαίσια.

Λέξεις κλειδιά: server, LLM, NPC, RPG, AI, δυναμικά δημιουργούμενο περιεχόμενο

Abstract

This thesis presents a new approach to progressive content creation in role-playing games (RPG) through the integration of large language models (LLMs). The thesis seeks to address a challenge in game development: creating rich, balanced and interactive content that maintains player attention and engagement while simplifying the process and timescales of content development.

The architecture of the software stack consists of two (2) main elements:

1) A server that curates the LLM interactions to create dynamic content that is the building block of the game. These structures are the game world, the characters handled by the game (NPCs), the player's missions, and the battles the player must face. The last server mechanism is a validation and error checking mechanism for LLM responses.
2) The game, through which the player can have interactions with NPCs, explore world locations, fight battles, evolve his character, and complete various missions.

This implementation demonstrates that the system successfully produces coherent fantasy worlds with narrative elements and functional systems. It is also observed that the generated content offers significant differentiation in each game played.

This implementation contributes to the field of game development by establishing a methodological framework for content generation using artificial intelligence (AI) by balancing creative variation with these structural constraints. These findings suggest promising directions for scaling procedural content generation in commercial game development through the application of large language models (LLMs) to specialized architectural frameworks.

Keywords: server, LLM, NPC, RPG, AI, dynamic content generation

Table of Contents

Table of Figures 8
Table of Abbreviations 9
Chapter 1 10
Introduction 10
Chapter 2: Literature Review 12
2.1 The evolution of PCG in games 12
2.1.1 Historical Development of PCG 12
2.1.2 Methods and Approaches in PCG 12
2.2 Limitations and capabilities of LLMs 13
2.2.1 Recent advances in LLM Architecture 13
2.2.2 LLMs for Creative Text Generation 14
2.2.3 Structural Limitations and Challenges 14
2.3 AI Integration in game development workflows 15
2.4 Validation and Quality Assurance for generated content 16
2.5 Conclusion and research gap 16
Chapter 3: Technology Stack and System Architecture 17
3.1 Technologies Used 17
3.2 System Architecture 18
3.3 Backend service design and implementation 21
3.3.1 World Generation 22
3.3.2 Chat and Context Management 28
3.4 Game client design and implementation 29
Chapter 4: Content Generation & Management Methodologies 32
4.1 Prompt Engineering Methodology 32
4.2 Validation Methodology 33
4.3 Context Management Methodologies 34
4.4 Interdependent Content Generation Methodology 35
4.5 Error Recovery and Resilience Methodologies 36
4.6 Integration Methodologies for Game Systems 37
Chapter 5: Gameplay Implementation 39
5.1 Menus 40
5.2 Game Entities 42
5.2.1 Player Characteristics 42

5.2.2 NPC ... 43
5.2.3 Enemies ... 44
5.2.3 Gem of Healing ... 45
5.3 Game World ... 45
5.4 Locations ... 46
5.5 Encounters ... 47
5.6 Game Systems ... 47
5.6.1 Combat System ... 47
5.6.2 Leveling System ... 48
5.6.3 Quest System ... 49
5.6.4 Inventory System ... 51
Chapter 6: Evaluation and Results ... 53
6.1 Response Time Analysis ... 54
6.2 Performance stability ... 56
6.3 Success Rates ... 59
6.4 Conclusion ... 60
Chapter 7: Future Directions and Conclusions ... 61
7.1 Implications ... 61
7.2 Future Research Directions and possible applications ... 61
7.3 Conclusions ... 62
Citations ... 64
Δήλωση Πνευματικών Δικαιωμάτων ... 65

Table of Figures

Figure 1: Client structure and other systems interactions 18
Figure 2: Backend system structure 19
Figure 3: Content Generation system 19
Figure 4: Game Logic system 20
Figure 5: Backend Service Architecture 21
Figure 6: OpenAPI view of the endpoints 22
Figure 7: Prompt structure generation 22
Figure 8: Parsing of the response 23
Figure 9: First cleaning attempt of the response 23
Figure 10: Data sanitization flow 24
Figure 11: World elements structure 25
Figure 12: NPCs characteristics structure 26
Figure 13: Quests structure 26
Figure 14: Encounters structure 27
Figure 15: Chat request 28
Figure 16: Conversation with NPC 28
Figure 17: Summarization code 29
Figure 18: Views flow 30
Figure 19: Complete Game flow Diagram 39
Figure 20: Title Screen 40
Figure 21: World Selection or Generation screen 40
Figure 22: World Generation Settings Screen 41
Figure 23: Loading Screen where the world generation progress is displayed. 41
Figure 24: End screen with game summarization 42
Figure 25: Character Movement 43
Figure 26: Player and NPC chat window 44
Figure 27: Enemy stats 44
Figure 28: Gem of Healing restoring the player's health 45
Figure 29: Game Hub 46
Figure 30: Location view with an encounter 46
Figure 31: Combat screen 47
Figure 32: Level up stats increase notification 48
Figure 33: Quest screen containing all quests. 49
Figure 34: Quest detailed view 50
Figure 35: Quest progress update 51
Figure 36: Inventory view showing items, tooltips, character stats & equipped items 52
Figure 37: Generation process response time distribution. 54
Figure 38: Conversation response time distribution. 55
Figure 39: Response Time heatmap visualization 56
Figure 40: Conversation response time heatmap. 57
Figure 41: Performance per endpoint 58
Figure 42: Conversation performance. 59
Figure 43: Success rate per world generation endpoint. 60
Figure 44: Success rates for conversation endpoint. 60

Table of Abbreviations

Abbreviation	*Definition*
RoQ	Realms of Quandria
AI	Artificial Intelligence
PCG	Procedural Content Generation
RPG	Role Playing Game
TTRPG	Tabletop Role Playing Game
NPC	Non Playable Character
LLM	Large Language Model
Regex	Regular Expression
Exp	Experience
HP	Health Points

Chapter 1

Introduction

Role-playing Games (RPGs) have long captured the hearts of gamers, with their worlds, narratives, character interactions, and gameplay. From the origins of tabletop role-playing games (TTRPGs) to digital games, these games offer players unique opportunities to explore detailed environments and make choices, resulting in multiple outcomes for every player. At the heart of this experience, alongside the gameplay, lies the content: the narrative threads, the character personalities, the missions, and the environment that bring these paper/virtual worlds to life.

Traditionally, the content creation of games has been an intensely manual process. Game designers, writers, and artists create each element individually, from landscapes and cities to the personalities of every NPC. This manual approach produces highly curated experiences but faces limitations in terms of scale, variability, and resource requirements. A simple parallel quest can require days of writing, planning, and implementation, while larger story arcs can absorb months of development time. More often than not, the result is a compromise: rich but limited content that players will quickly exhaust, or larger worlds with less depth and more repetitive elements that can diminish player immersion and satisfaction over time.

Procedural content generation (PCG) emerges as a partial solution to this challenge, using algorithmic approaches to create game elements such as terrain, dungeons, and object features. PCG has a vast field of applications, from simple things like procedurally generated textures all the way to procedurally generated stories.

Recent developments in AI, especially in LLMs, allow for a new field for creating procedural content in video games. These LLMs demonstrate—most of the time—almost excellent abilities in understanding context, creating narratives, all while maintaining thematic consistency. However, integrating these technologies into practical game development contexts presents significant challenges. Language models operate with probabilistic outputs that lack the deterministic structure required by game systems. They can produce inconsistent, unbalanced, or mechanically incompatible content without appropriate constraints. In addition, the computational requirements of these models raise questions about performance in real-time game environments, and their contextual limitations may affect long-term narrative coherence.

Realms of Quandria (RoQ) attempts to represent an innovative approach to address these challenges through a specialized "client"-server architecture that enables us to separate the game mechanics and implementation from the content creation. This separation allows each component to utilize technologies for its specific requirements while maintaining a relatively easy integration. Rather than attempting to integrate LLMs directly into the game's engine—which would add a significant overhead to the performance of the game—this project creates a dedicated service for the creation of the content with a specialized game "client".

As mentioned, the architecture consists of two main components:

- A backend webservice, orchestrating the LLM interactions to create the world and its elements.
- A “client” that renders these elements into a complete game environment.

The backend uses validation systems, iteration mechanisms, and context management techniques to ensure that the creative capabilities of the LLM conform to the requirements of the game

systems. Meanwhile, the client implements traditional RPG mechanics: exploration, combat, character evolution, and narrative interactions while maintaining communication with the service that produces the content. This design overcomes several key limitations of previous approaches:

1. By moving the process-intensive LLM functions to a dedicated service, the game maintains satisfactory performance during gameplay.
2. The use of validation and correction mechanisms ensures that the generated content adheres to game-compatible data.
3. The asynchronous nature of the client-server communication allows for an almost seamless integration of dynamically generated content without disrupting the player experience.
4. The service is client agnostic, meaning that with small modifications this service can work with every game engine. It is not tied to the specific implementation.

This project attempts to be a contribution to the field of game development and the integration of AI. First, it establishes a framework for AI-assisted content generation that balances creativity and structural constraints required for a complete gaming experience. This implementation addresses a tension in PCG between flexibility and consistency, making available a model for future applications of LLM in game development. Second, it introduces techniques for correcting and validating the outputs of LLM in the field of game development. This ensures that the output of the model is usable in the game engine. Third, it demonstrates strategies for managing the context of the conversation between the player and the LLM whilst maintaining an illusion of memory and character personality for the NPCs. Finally, it provides a comprehensive case study that illustrates both the potential and practical considerations of using LLMs in game development. This case study aims to provide valuable insights for developers seeking to use similar technologies in their own projects.

Through the exploration of AI-driven PCG, Realms of Quandria offers not just a technological demonstration, but a preview of how these technologies might affect game development, creating more varied and responsive virtual worlds while reducing the resource constraints that traditionally limit the scope and detail of RPGs.

Chapter 2: Literature Review

2.1 The evolution of PCG in games

The PCG field has evolved significantly over the past decades, from the early use of generating simple game elements to sophisticated systems capable of creating complex and interactive narratives and worlds.

2.1.1 Historical Development of PCG

The history of Procedural Content Generation in game development goes back to the early 1980s when developers were looking for creative ways to overcome hardware limitations. Early games like Rogue (1980) and Elite (1984) showed how algorithms could generate vast dungeons and universes despite severe memory constraints. These early implementations served two purposes: to get around the technical limitations and to add game variety through randomized elements so no two playthroughs would be the same.

The technical necessity that drove initial PCG adoption turned into a design philosophy. Games like Pac-Man used level variation to add replay value to something that has simple mechanics. After a period of limited use in mainstream development, PCG had a renaissance with empire building games like Civilization which introduced procedurally generated worlds as a fundamental part of the game rather than just a technical workaround.

This history shows three enduring reasons for PCG implementation: technical constraint mitigation, gameplay diversification and development efficiency. While modern hardware has eliminated the first reason, the latter two still drive PCG adoption in modern game development – including our own approach.

2.1.2 Methods and Approaches in PCG

The procedural content generation (PCG) landscape has changed significantly since its inception, moving from simple algorithmic approaches to complex generative systems. Early PCG implementations relied on constructive methods—predetermined rules combined with randomness to produce variations on designer-specified patterns. While good for simple tasks like terrain generation, these were limited in output quality and struggled with complex content.

As the field became more familiar, developers introduced more advanced techniques to address these limitations. Search-based PCG methods came along, using evolutionary algorithms and other optimization techniques to explore the possibility space while targeting specific design goals. Constraint-based systems then enforced structural rules to ensure playability and balance. Both were big advances in generating complex, functional game content while still having designer control.

The latest methodological evolution has been the integration of machine learning into the PCG pipeline. This is more than a technical advancement—it's a fundamental shift in how

generative systems relate to designer intent. Traditional approaches explicitly encode design knowledge through rules and templates, basically translating human creativity into algorithmic procedures. Machine learning methods extract design patterns implicitly from existing content, learning to recognize and reproduce stylistic elements without requiring explicit formalization.

This shift from explicit to implicit design knowledge has big implications for the scope and flexibility of PCG systems. Rule-based approaches are great at generating content that follows well-understood design principles but struggle with adaptation. Machine learning methods can produce more unexpected creative output but traditionally require a lot of domain specific training data—a big limitation for game specific applications where suitable datasets might be scarce.

Our language model approach is a natural extension of this methodological trajectory, using the broad training of large language models to overcome the data limitation that has held back previous machine learning approaches. Rather than requiring extensive game specific datasets, these models transfer knowledge from their general text training to generate coherent game content with minimal domain adaptation. This allows for the production of complex narrative elements, character backgrounds and quest structures that are both coherent and creative—exactly the content categories that have been most resistant to traditional PCG methods.

The progression towards more advanced generative techniques is a long standing goal in PCG research: balancing generative freedom with structural control. We continue this tradition, using the creativity of language models and validation frameworks to ensure outputs meet the system requirements.

2.2 Capabilities of LLMs

LLMs demonstrate remarkable abilities at text generation; that comes with significant limitations that must be addressed in order to effectively produce game content.

2.2.1 Recent advances in LLM Architecture

The development of transformer-based architectures has changed natural language processing (NLP) beyond recognition, creating models with profound scale and capability. Vaswani, who introduced the transformer architecture, describe its core innovation:

"The Transformer architecture relies entirely on attention mechanisms to draw global dependencies between input and output, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention."[1]

This change in architecture indicates the capabilities of modern LLMs, including those implemented in our system, by using more processing-effective long-range dependencies in text—something that is crucial for maintaining narrative coherence. The development of these models has produced capabilities that were not seen in their smaller predecessors. This phenomenon is documented by Brown in their work on GPT-3:

"We find that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning

approaches. Specifically, we find that increasing model size improves performance in a log-linear fashion across tasks, with the biggest improvements occurring in the ability to perform tasks that require multi-step reasoning or in-context learning."[2]

These scaling qualities are very important for creating gaming content, as such jobs demand complicated reasoning across interconnected aspects like characters, locations, and events.

2.2.2 LLMs for Creative Text Generation

LLMs have shown they can generate some pretty cool text, which is useful for game development. They're great at writing basic stories, dialogues and descriptive text that can enhance the gaming experience. However, since their output is based on probabilities, the quality can vary and there can be coherence issues especially when generating longer text without clear guidance or constraints.

Our research on the Realms of Quandria system confirmed this. We found that the raw output from language models need well-structured prompts and post-processing to get consistent quality of game content. The creative potential of these models is also evident in interactive narrative scenarios. They keep character consistency and narrative coherence across multiple interactions, so they're great for dynamic NPC behaviors and dialogue systems. This opens up some possibilities for more responsive and lifelike game characters.

However, despite these benefits, language models struggle with long term narrative planning and causal reasoning. They can generate text that seems coherent in the short term but can contradict established facts or not maintain overall narrative coherence through an entire gameplay session. These findings have directly influenced our design for context management and validation so we can use the benefits while addressing the challenges.

2.2.3 Structural Limitations and Challenges

Despite all the incredible capabilities, language models have significant limitations that need to be addressed when integrating them into game systems. Neural text generation has problems of repetition, contradiction and hallucination – and those problems get worse the longer the text is generated. Those aren't flaws in specific model architectures but in how the models are trained and how they generate text.

Those observations directly informed our system design decisions, particularly our implementation of validation layers to detect and correct inconsistencies in model generated content before it hits the player.

Another big challenge for game applications is the context window of current language models. Those models can only "remember" a finite amount of previous text, which limits their ability to be coherent across long interactions or narratives. When conversations or storylines go beyond that window the models lose access to earlier information and you get contradictions or narrative breaks.

That's what guided our development of the ContextManager component, which implements conversation summarization techniques to keep NPC interactions coherent despite those architectural limitations. By compressing and prioritizing contextual information our system can support much longer and more complex player-NPC interactions than would be possible with the raw limitations of the underlying models.

2.3 AI Integration in game development workflows

AI-generated storytelling has come a long way in game development research. Computational storytelling systems tend to focus on three main areas: plot, space and character. One of the remaining challenges is to combine these elements into wholes that balance storytelling requirements with gameplay constraints [3].

This challenge directly affects our architecture. Our system has separate generation services for the world structure, NPCs, quests and encounters and ensures consistency through carefully designed interdependencies and validation processes. By separating these concerns while keeping them related we can manage the complexity of creating coherent game worlds.

There is a fundamental tension between narrative coherence and player agency in interactive storytelling. Different approaches prioritize these things differently – some systems are very author-driven and limit player choices to maintain narrative structure, others go for emergent gameplay at the expense of structured narrative arcs.

We have this same tension in our system. We need to balance the creative variation language models provide with the structural requirements for gameplay. Our solution is to implement JSON schema validation and correction pipelines that ensure generated content stays within playable boundaries while still allowing for creative diversity and surprise. By acknowledging and designing for these inherent tensions our system tries to get the benefits of AI-generated storytelling while mitigating the risks that could harm the gameplay.

Despite all the great demos in research, commercial games have several barriers to entry including technical integration, quality assurance and design team resistance. To succeed you need systems that complement and do not replace designer expertise – tools that expand the possibilities not automate the existing workflow. This is how we approach development, we see language models as augmentative tools within a structured framework not autonomous replacements for human design decisions [4].

Our system keeps the designer's creative judgment and uses AI to boost productivity and explore the creative space that would otherwise be missed. The computational requirements of language models present additional challenges for game applications. Different models have very different inference latency and resource requirements, model size is a big but not the only factor that affects performance. Response times for text generation can be under a second to nearly 5 seconds across different models, that's a big obstacle for real-time gameplay integration. These performance constraints shaped our client-server architecture.

By isolating the resource heavy language model operations in a dedicated service and keeping the gameplay responsive to the client we can leverage the AI capabilities without compromising the player experience. This separation allows us to use the powerful AI without

sacrificing the gameplay responsiveness, it's a more sustainable way to integrate AI in interactive entertainment.

2.4 Validation and Quality Assurance for generated content

To ensure the quality of the output of the LLM and to validate said output is one of the most critical challenges. Evaluating content generated by PCG methods has different challenges compared to other AI applications. In many AI fields, success can be measured through clear metrics like accuracy or performance benchmarks. However, when it comes to game content evaluation, we must also consider the subjective elements of player experience alongside functional requirements.

This evaluation challenge affects our validation strategy. Our system employs both structural checks to ensure functional requirements are satisfied and semantic validation processes to evaluate narrative coherence and player experience factors. This dual approach enables us to confirm that the content not only integrates well within the game's systems but also achieves the desired emotional and narrative effects.

Effective PCG systems should demonstrate reliability by consistently producing playable content without significant flaws. They must allow for controllability, enabling designers to influence the generated artifacts rather than relying solely on random outcomes. Also, they should show expressivity by creating diverse content that explores the intended design space while adhering to appropriate limits.

These criteria offer a useful optic for evaluating our own system's effectiveness; they underscore the critical balance we need to strike between the creative expressivity provided by language models and the reliability demands of functional game systems. By explicitly addressing these criteria in our design and evaluation processes, we can enhance our approach to AI-assisted content generation for games..

2.5 Conclusion and research gap

This short review highlights the advances in both PCG and the capabilities of LLM, while also revealing some gaps regarding their integration to games. While the language models show great potential for creating coherent and diverse text, their direct and immediate application to games provides some struggles in terms of structure, performance as well as the consistency of the responses. Previous work done by others has already explored many aspects of AI-assisted game development; while comprehensive designs for integrating LLMs into game systems are still being explored and developed.

This research aims to address this lack of exploration by developing and evaluating a "client"-server architecture that leverages language models for content creation, while maintaining the required characteristics for a functional game. By applying systematic validation mechanisms and context management techniques our system addresses the practical challenges mentioned above, while also providing a demonstration of the potential the LLMs have to transform game content creation.

Chapter 3: Technology Stack and System Architecture

As mentioned above the system follows a “client”-server architecture with a clear separation of concerns. The server manages the generation of the world through modular services. Each service has a retry mechanism with penalties and an exponential backoff mechanism for better resilience. The system also has a context manager for the conversations with each NPC, in order to provide conversation summarization. Each service output is validated via a custom validation mechanism in order to have processable data for the client (game) to use.

3.1 Technologies Used

FastAPI [5] is a modern Python web framework that enables developers to build high-performance APIs quickly with built-in data validation and serialization capabilities. It leverages Python type hints to automatically validate, serialize, and document our API requests and responses.

Swagger [6], now officially known as OpenAPI, is a specification for machine-readable interface files that describe, produce, consume, and visualize RESTful web services. The OpenAPI specification defines a standard, language-agnostic interface that allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code or documentation.

FastAPI automatically generates OpenAPI documentation based on your Python type annotations, function parameters, and docstrings without additional work. When we build an API with FastAPI, we get an interactive Swagger UI that lets developers explore and test our API directly in the browser. This integration creates a seamless development experience where our API documentation stays in sync with your code as it evolves. FastAPI's creator, Sebastián Ramírez, designed the framework with developer experience in mind, making it possible to build well-documented, standards-compliant APIs with minimal boilerplate code.

Python Arcade [7] is a modern, easy-to-use library designed for creating 2D video games with compelling visual effects and physics simulations. Unlike more complex game engines, Arcade strikes a perfect balance between simplicity and power, making it ideal for beginners yet capable of creating sophisticated games. The library provides intuitive ways to handle sprites, animations, and collision detection while maintaining Python's readability and approachability. Arcade was developed by Paul Vincent Craven as an educational tool to help students learn programming through game development, focusing on clean code structure and object-oriented principles. Game development with Arcade follows a logical pattern of initialization, game loop updates, and rendering functions that mirror professional game development practices.

Tiled [8] is an open-source map editor primarily designed for creating 2D game levels and environments. It allows developers and designers to build complex game worlds by arranging graphical tiles in layers on a grid, supporting orthogonal, isometric, and hexagonal maps. The program features a user-friendly interface with tools for drawing, filling, and selecting tiles, while also offering advanced capabilities like custom properties, automation through scripting, and support for various export formats compatible with numerous game engines and frameworks.

3.2 System Architecture

The implementation consists of four major systems:

1) The Client system as seen in Figure 1.
2) The Backend system as seen in Figure 2.
3) The Content Generation system as seen in Figure 3.
4) The Game Logic system as seen in Figure 4.

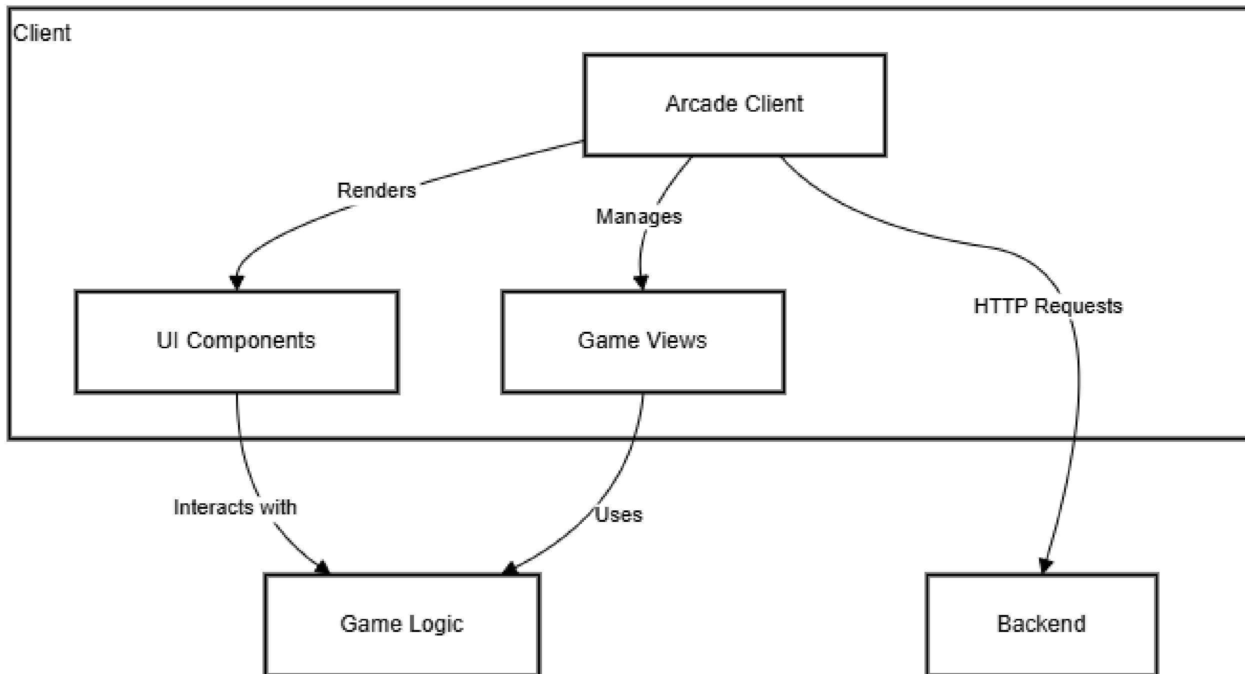

Figure 1: Client structure and other systems interactions

The Client Architecture system provides the player-facing interface of the RPG game, handling all visual representation and user interaction. The Arcade Client serves as the main application container, managing the game window, processing keyboard and mouse inputs, and synchronizing with the backend via HTTP requests. It orchestrates the Game Views that present different aspects of gameplay to the user. The UI Components include reusable elements such as dialogue boxes, inventory slots, stat displays, and combat interfaces that provide consistent visual styling and interaction patterns throughout the game. This architecture follows a hierarchical design where the Arcade Client renders the appropriate Game Views based on player context, which in turn contain and manage various UI Components. This system communicates with the Game Logic layer to translate player actions into gameplay effects and render the current game state in a visually appealing and intuitive manner.

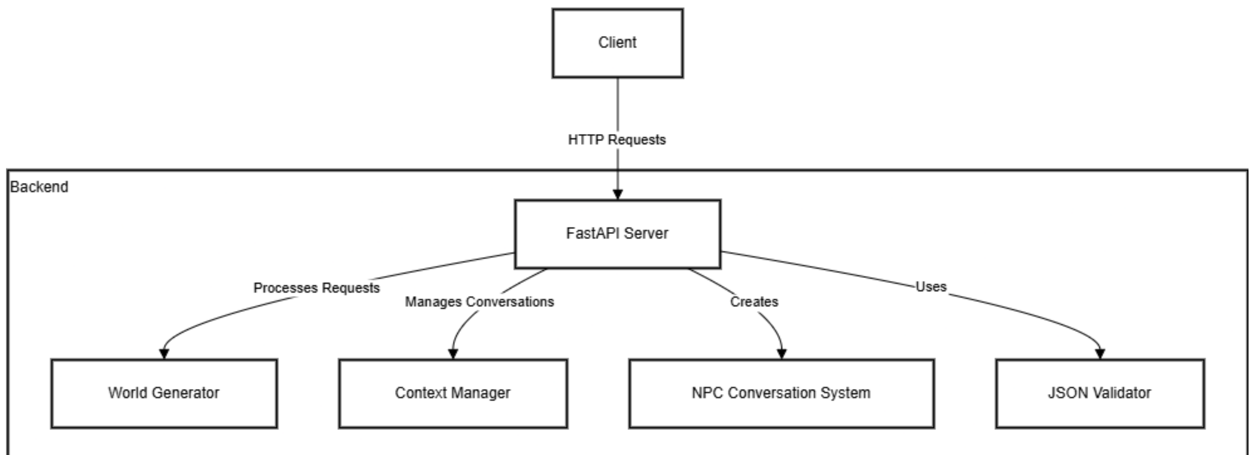

Figure 2: Backend system structure

The Backend system serves as the server-side infrastructure for the RPG game, with the FastAPI Server functioning as its central hub. The World Generator creates dynamic game content including maps, NPCs, quests, and encounters when players request a new game world. The Context Manager maintains conversation history between players and NPCs, intelligently summarizing lengthy dialogues to preserve memory while retaining important context. The NPC Conversation System handles dialogue generation and responses, creating realistic interactions that can advance quest objectives. The JSON Validator ensures data integrity by validating the structure of information flowing between components, preventing errors that could arise from malformed data structures. Together, these components form a robust backend that supports the game's dynamic content generation and persistent state.

Figure 3: Content Generation system

The Content Generation system is the creative engine of the RPG game, responsible for procedurally generating engaging game content. At its core, the Language Model processes natural language inputs and generates coherent, contextually appropriate outputs for game elements. The World Generator interfaces with this model to create complete game worlds with consistent themes, locations, characters, and storylines. The Generation Queue manages processing prioritization, ensuring requests are handled efficiently even during high-demand periods, while preventing the system from becoming overwhelmed. The NPC Conversation System leverages the Language Model to create dynamic, contextually aware dialogue that responds to player inputs and maintains character consistency across interactions. These components work together in a pipeline architecture where content requests flow through the Language Model, are processed asynchronously via the queue, and return results to the appropriate game systems, creating a rich, ever-evolving game world that feels responsive and alive.

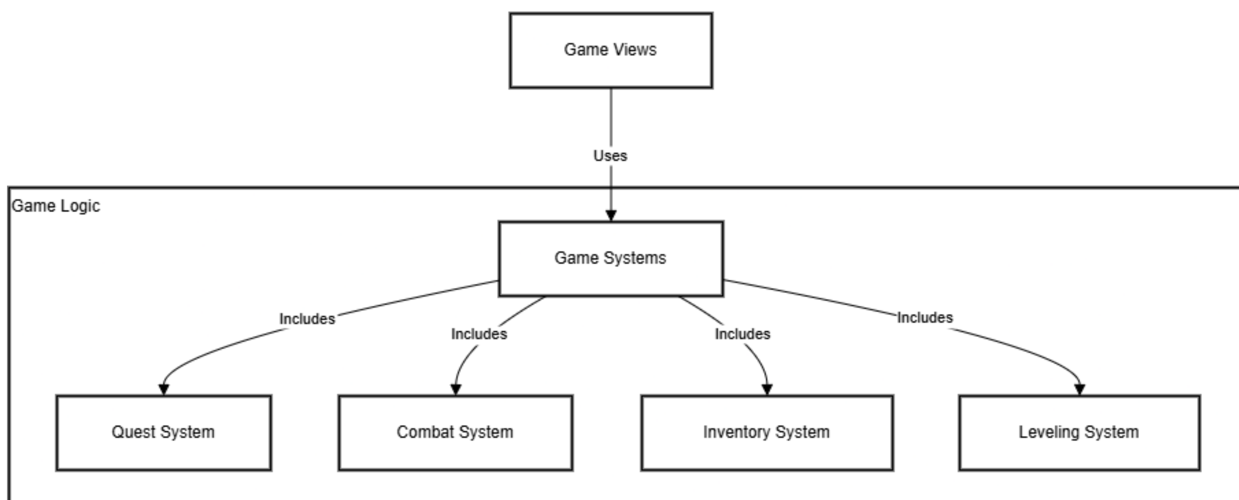

Figure 4: Game Logic system

The Game Logic system encompasses the core gameplay mechanics and rules that drive the RPG experience. The Game Systems module serves as the central coordinator for all gameplay functionality, maintaining the game state and ensuring all subsystems work in harmony. The Quest System tracks objectives, manages progression, and provides rewards when conditions are met, driving the narrative flow of the game. The Combat System handles turn-based encounters, calculating damage, managing combat actions like attack and defend, and determining outcomes based on character stats and randomization. The Inventory System manages the player's possessions, equipment, and consumable items, applying appropriate stat modifications when items are equipped or used. The Leveling System tracks player experience, handles level-up events, and applies stat increases as players progress, creating a sense of character development and growth. These interconnected systems process player actions from the Game Views, update the game state accordingly, and provide feedback that is rendered by the Client Architecture, creating a cohesive gameplay experience that rewards strategic thinking and exploration.

3.3 Backend service design and implementation

The backend is developed using FastAPI framework which makes it ideal for building real-time data applications and other high-performance applications; just like the PCG service with the use of a LLM.

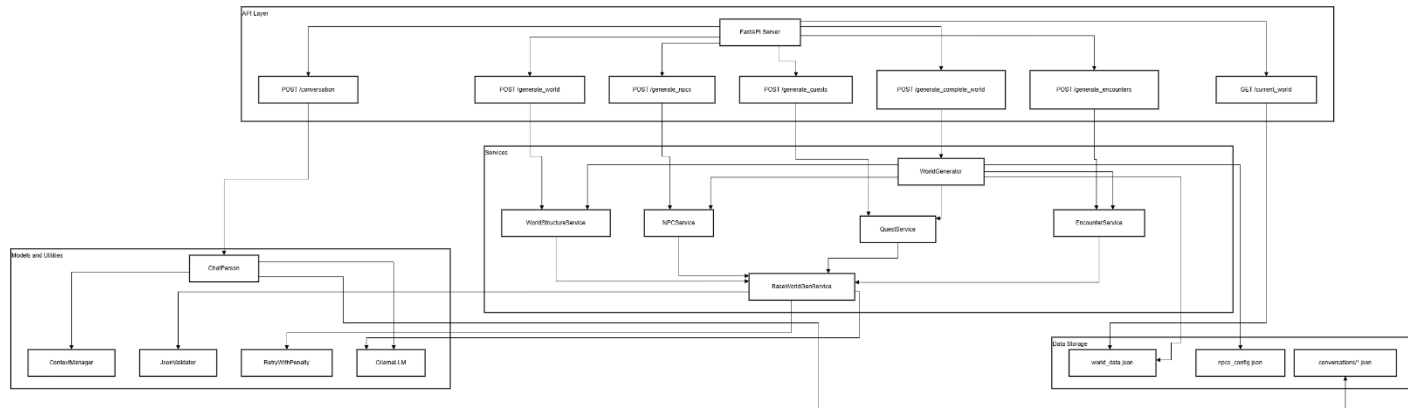

Figure 5: Backend Service Architecture

The backend consists of eight endpoints:

- /model_name : A GET request to return to the user which LLM model is used
- /generate_world : A POST request that generates some crucial information for the game world
- /generate_npcs: A POST request that generates the NPCs populating the world
- /generate_quests: A POST request that generates the quests that the player has to complete
- /generate_encounters: A POST request that generates the encounters the player has to overcome.
- /generate_complete_world: A POST request that is the combination of the other four requests
- /current_world: A GET request that returns the complete generated world
- /conversation: A POST request that allows the player to converse with each NPC

LLM API 0.1.0 OAS 3.1
/openapi.json

default

GET /model_name Get Model Name
POST /conversation Converse
POST /generate_complete_world Generate World
POST /generate_world Generate World Structure
POST /generate_npcs Generate Npcs
POST /generate_quests Generate Quests
POST /generate_encounters Generate Encounters
GET /current_world Get Current World

Figure 6: OpenAPI view of the endpoints

3.3.1 World Generation

The complete generation of the world takes four steps to complete. First, it generates the basic world structure. This is done by providing the LLM with some basic information about the world we want to create; we implemented it in a way that even though the user input is minimal, the language model is free to generate a completely custom world.

The fields that are required for the generation of the world consist of the genre, the number of locations, the setting type, the tone of the world, and lastly the overall complexity. When the request is sent, the server attempts to generate the world structure as seen in Figure 7

```
def generate_structure(self, request: WorldStructureRequest) -> dict:  2 usages  topas404 *
    self.logger.debug("Starting world structure generation")
    retry_handler = RetryWithPenalty()

    while retry_handler.attempt < retry_handler.max_retries:
        try:
            self.prompt = self._build_structure_prompt(request)
            options = retry_handler.get_penalty_options()

            self.logger.debug(f"Attempt {retry_handler.attempt + 1} with options: {options}")
            response = self.model.invoke(self.prompt, options=options)

            data = self.parse_llm_response(response)
            self.logger.debug(data)
            if self.json_validator.validate_data(data,  schema_type: "world"):
                self.save_world_state({"world": data})
                return data

        except ValueError as e:
            retry_handler.last_error = e
            retry_handler.attempt += 1

            if retry_handler.attempt < retry_handler.max_retries:
                delay = retry_handler.base_delay * (2 ** (retry_handler.attempt - 1))
                self.logger.warning(f"Attempt {retry_handler.attempt} failed. Retrying in {delay} seconds...")
                time.sleep(delay)
            continue

    raise ValueError(f"Failed to generate world structure after {retry_handler.max_retries} attempts. "
                     f"Last error: {retry_handler.last_error}")
```

Figure 7: Prompt structure generation

First we initialize a mechanism that will allows us to retry the world generation with a specific penalty for the LLM in case it does not produce the expected outcome. Then we build the prompt that we will use for the language model to get our world data (more on that is explained in Chapter 4.1). In the event that the response is incorrect, we retry up-to five times, with different temperature and top-p values (both are explained in Chapter 4.5). When the model creates a response we do a series of parsing and validation tests.

```
def parse_llm_response(self, response: str) -> dict:
    try:
        # Direct parse first
        return json.loads(response)
    except json.JSONDecodeError:
        try:
            # Clean and retry
            cleaned = self.fix_json_string(response)
            return json.loads(cleaned)
        except json.JSONDecodeError as e:
            self.logger.error(f"Failed to parse JSON: {str(e)}\nResponse: {response}")
            raise ValueError(f"Failed to parse LLM response as JSON: {str(e)}")
```

Figure 8: Parsing of the response

First we try to parse the response; this means we try to get the JSON object from the response. The response sometimes regardless of the specific prompt and rules set, can be in a different format or with extra text that will make it un-processable. To combat this we have implemented a regular expression (regex) based mechanic that tries to clean the data as seen in Figure 9

```
def fix_json_string(self, s: str) -> str:
    # Remove markdown code blocks and surrounding text
    s = re.sub(r'.*```json\s*', '', s, flags=re.DOTALL)
    s = re.sub(r'```.*', '', s, flags=re.DOTALL)

    # Remove any non-JSON text before and after
    s = re.sub(r'^[^{]*', '', s)  # Remove text before first {
    s = re.sub(r'[^}]*$', '', s)  # Remove text after last }

    # Fix trailing commas before closing brackets/braces
    s = re.sub(r',(\s*[}\]])', r'\1', s)

    # Fix double commas
    s = re.sub(r',\s*,', ',', s)

    # Fix trailing commas in objects
    s = re.sub(r',(\s*})', r'\1', s)

    # Remove comments
    s = re.sub(r'#.*$', '', s, flags=re.MULTILINE)

    # Balance braces and brackets if needed
    open_count = s.count('{')
    close_count = s.count('}')
    if open_count > close_count:
        s = s + '}' * (open_count - close_count)

    open_brack = s.count('[')
    close_brack = s.count(']')
    if open_brack > close_brack:
        s = s + ']' * (open_brack - close_brack)

    try:
        # Try to parse and format the JSON
        parsed = json.loads(s)
        return json.dumps(parsed, indent=2)
    except json.JSONDecodeError as e:
        # If parsing fails, try additional fixes

        # Remove any stray commas at the end of objects
        s = re.sub(r',(\s*})', r'\1', s)

        # Fix objects missing closing brace after nested arrays
        s = re.sub(r'(\]\s*)(,|\}|$)', r'}\1\2', s)

        try:
            # Try parsing again after additional fixes
            parsed = json.loads(s)
            return json.dumps(parsed, indent=2)
        except json.JSONDecodeError:
            # If still can't parse, return the cleaned but unparsed string
            return s
```

Figure 9: First cleaning attempt of the response

Then we proceed -regardless of whether its successful or not- with validating the data. The process can be seen in Figure 10.

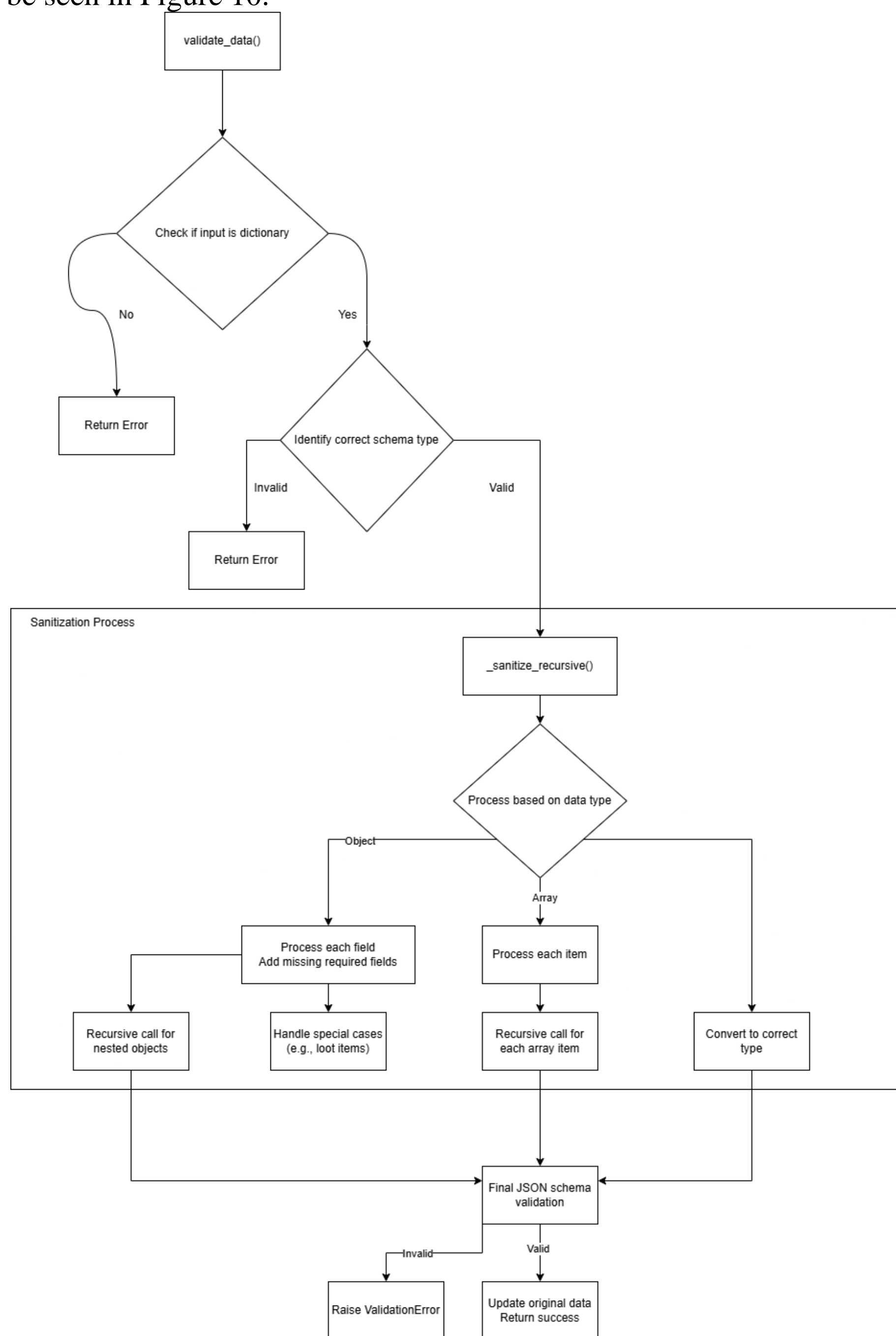

Figure 10: Data sanitization flow

The response may contain extra fields that are not wanted (e.g., in the world generation, it may have a field called "weather") or not have fields that are required (e.g., in the encounters, an encounter may not have a damage value for the enemies). To combat that, we use a predefined structure that the response must adhere to. Through the sanitization process, any fields that are not present in our structure are removed, and if a required field is not present, it is initialized with a default value (e.g., damage = 5). This process is done recursively in order to achieve sanitization for all nested objects.

If the sanitization process is unsuccessful, we throw an error and the generation for this particular service (e.g., Encounters) fails, resulting in an empty object in our game. If the sanitization process is successfully completed, we append the JSON object to a file and return it.

The world JSON has the structure seen in Figure 11:


```
"world": {
   "name": "",
   "description": "",
   "locations": [
     {
       "name": "",
       "description": "",
       "purpose": ""
     }
   ],
   "conflicts": [
     {
       "name": "",
       "description": ""
     },

   ],
   "loot": [
     {
       "name": "",
       "description": "",
       "type": "",
       "damage": 0.0,
       "value": 0.0
     },

   ]
 }
```


Figure 11: World elements structure

When the world generation is complete, we take the JSON object and use it for the remaining three requests. The generate NPCs service takes the JSON and before starting the same process it requires an additional input from the user, the number of NPCs to be present in the game. When the NPCs are generated, their characteristics are stored in a file that is loaded into memory, giving us the ability to start conversing with them. Then the exact same flow as mentioned above is executed resulting to a JSON for the NPCs can been seen in Figure 12:

```
"npcs": {
   "npcs": [
     {
       "name": "",
       "traits": {
         "primary_emotion": "",
         "disposition": "",
         "motivation": ""
       },
       "background": "",
       "speech_style": ""
     },
   ]
 }
```

Figure 12: NPCs characteristics structure

After the NPC generation we generate the quests, taking as input the world JSON, the NPC JSON -because in some quests we have to interact with the NPCs- and the number of quests we want. The final JSON structure can be seen in Figure 13:

```
"quests": {
  "quests": [
    {
      "name": "",
      "description": "",
      "involved_npcs": [
        ""
      ],
      "locations": [
        ""
      ],
      "rewards": {
        "experience": 0.0,
        "gold": 0.0,
        "loot": [
          {
            "name": "",
            "description": "",
            "type": "",
            "damage": 0.0,
            "value": 0.0
          },
        ]
      },
      "steps": [
        "",
      ]
    },
  ]
}
```

Figure 13: Quests structure

Finally after the generation of the quests we create the encounters. The encounter service requires the world JSON, the quests JSON and the number of encounters. This generates an object as seen in Figure 14:

```
"encounters": {
  "encounters": [
    {
      "name": "",
      "description": "",
      "location": "",
      "enemies": [
        {
          "name": "",
          "type": "",
          "level": 0.0,
          "health": 0.0,
          "damage": 0.0,
          "abilities": []
        },
      ],
      "rewards": {
        "experience": 0.0,
        "gold": 0.0,
        "loot": [
          {
            "name": "",
            "description": "",
            "type": "",
            "damage": 0.0,
            "value": 0.0
          }
        ]
      },
      "difficulty": "",
      "minimum_level": 0.0
    },
  ]
}
```

Figure 14: Encounters structure

It is important to note that all the values in these objects are determined and generated entirely from the LLM e.g. we do not provide any input regarding to the model regarding the disposition of the NPC, or the steps required to complete a quest.

3.3.2 Chat and Context Management

One of the key points of our implementation is the ability to chat with the LLM which assumes the persona of each NPC created, for each different player. Each NPC creates a file that stores all its conversations with each player. When the player initiates a conversation we send a request like the one shown in Figure 15:

```
{
  "user_name": "",
  "chatPerson": "",
  "question": ""
}
```

Figure 15: Chat request

The first thing that happens is we load any conversation for this NPC/player combination into the memory; then we create a prompt that contains the question and we send it to the model. When the model responds we add the response in the conversation history (both in-memory and in the file) and check if the length of the whole conversation history exceeds a predefined context size. If not we return the response.

```python
def generate_response(self, user_name: str, question: str):  1 usage  topas404
    # Use the conversation history from memory
    history = self.conversation_histories.get(user_name, [])

    # Generate context from history
    context = "\n".join(
        [
            f"{user_name}: {entry['question']}\n{self.name}: {entry['answer']}"
            for entry in history
        ]
    )

    # Add a clear prompt separator
    prompt_input = {
        "context": context,
        "question": question,
    }

    try:
        # Generate response using the model
        result = self.chain.invoke(prompt_input)
    except Exception as e:
        self.logger.error(f"Error generating response for {self.name}: {e}")
        raise

    # Update history with the new interaction
    history.append({"question": question, "answer": result})
    self.conversation_histories[user_name] = history

    if self.context_manager.should_summarize(context):
        summarized_history = self.context_manager.summarize_history(
            user_name,
            self.name,
            self.conversation_histories[user_name]
        )
        # Update the conversation history with the summary
        self.conversation_histories.update(summarized_history)
        self.load_conversations()

    return result
```

Figure 16: Conversation with NPC

If it exceeds the size, we use the context manager which with a predefined reduction range (e.g. 40%-60%) and a context size (e.g. 10000) create a prompt to the LLM asking it to summarize the history. Finaly the summarized history updates the current history both in-

memory and in the file.

```python
def summarize_history(self, user_name: str, npc_name: str, history: List[Dict]) -> Dict[str, List[Dict]]:  1 usage

    # Format the conversation history
    history_text = "\n".join(
        f"{user_name}: {entry['question']}\n{npc_name}: {entry['answer']}"
        for entry in history
    )

    try:
        # Generate summary using the language model
        summary = self.summary_chain.invoke({
            "history": history_text,
            "min_reduction": self.summary_reduction_range[0],
            "max_reduction": self.summary_reduction_range[1]
        })

        # Create the summarized conversation structure
        summarized_history = {
            user_name: [{
                "question": "Previously conversed subjects",
                "answer": str(summary)
            }]
        }

        # Save the summarized conversation
        self.save_conversations(user_name, npc_name, summarized_history)

        return summarized_history

    except Exception as e:
        self.logger.error(f"Failed to summarize conversation history: {e}")
        # Create fallback with last 3 entries
        fallback_history = {user_name: history[-3:]}
        # Save the fallback history
        self.save_conversations(user_name, npc_name, fallback_history)
        return fallback_history
```

Figure 17: Summarization code

This approach allows us to provide the illusion of NPC memory, by shortening lengthy conversations while also keeping the important points of the conversations. This is done to manage both system memory and model memory (context) constraints and to enhance the player experience through persistent dialogues.

3.4 Game client design and implementation

The game client for Realms of Quandria is developed using the Python Arcade library, a library used mostly for creating 2D games. It provides an API that is easy to utilize and build upon, allowing for relatively easy development. Our implementation follows a component-based architecture where the game elements are developed as independent but interconnected modules.

The design follows a modular architecture that separates concerns and promotes maintainability while enabling rich gameplay mechanics. This chapter examines the technical design decisions, implementation strategies, and architectural patterns employed in the development of the game client. Our game architecture consists of the view system, which manages the different game screens and user interfaces, the model system which defines the game entities and their respective behaviors, the systems layer which implements all the game mechanics such as the combat, the inventory and the quests. We also have the world generation

component, which interfaces with our server to generate the game content. Finally, we have a map management implementation which loads maps created with an open source program called Tiled.

The View management is built on Arcade's view framework, allowing use to have clean transitions between different game states and screens. Our implementation uses a hierarchical approach where specialized views are handling specific aspects of the gameplay. A high level view of our flow can be seen in Figure 18.

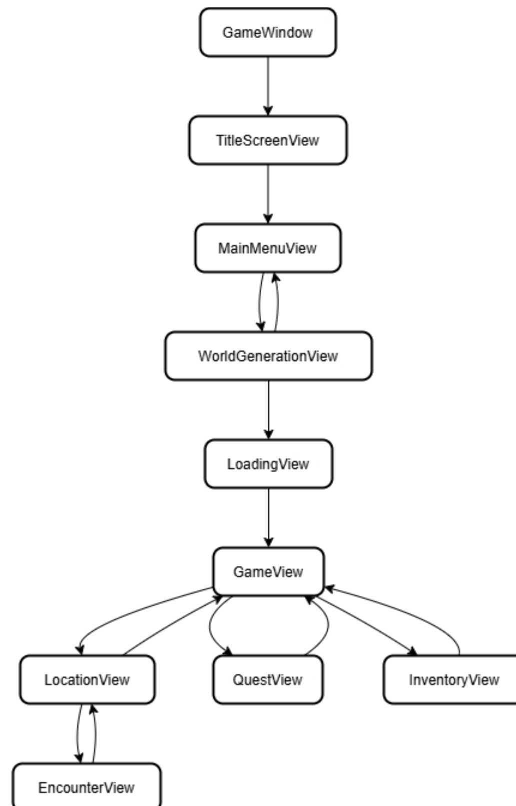

Figure 18: Views flow

Many of the game's core functionalities are broken down into several Systems. The combat system handles the turn-based combat where there are two basic types of moves. The flow of the turns is as follows: first the player, then one of the enemies, then the player and so on. Both the player and the enemies in their turn either block or perform an attack. If the player attacks, they apply their damage to the enemy they clicked. If the player decides to defend, the damage is reduced based on the formula total_defense = base_defense + (base_defense * 0.6). If the enemy defends, the total damage they receive is: total_damage_taken = player_damage – (player_damage / 2).

The reward and leveling systems are fairly simple. The leveling system manages when the player levels up and what happens upon level up. There is a simple formula to find what attribute of the player increases upon level up. The player's health increases every level. Beyond that, on even levels (e.g., 2, 4, 6, etc.) the damage is increased. On odd levels (e.g., 1, 3, 5, etc.) the defense is increased, and every 10 levels all attributes are increased. Any experience remaining upon level up is carried over to the next level. The reward system gives the rewards of any encounter and quest completed. These rewards include items, gold, and experience.

The inventory system handles everything regarding the storing of the items. It creates the items with their attributes and some basic icons, it provides us with the ability to equip, unequip, or sell the items and assign if the items (depending on their type) are stackable or not.

Lastly, we have the quest system, the most complex among our systems. It begins by parsing and categorizing quest data during initialization, breaking down quest steps into actionable objectives based on keywords. It identifies three types of objectives: NPC interactions (triggered by the keywords: "talk", "speak", "find", "meet", "ask", "deliver", "give", "bring", "return", "interact"), location visits (identified by the phrases: "go to", "visit", "travel", "find", "explore", "reach") and encounters (containing the combat terms: "defeat", "kill", "slay", "fight", "battle", "combat", "vanquish", "engage"). During gameplay, the system continuously monitors player activities through specialized checking methods. When a player interacts with an NPC, the system uses a method to compare the NPC's name against existing objectives, using both exact matching and partial word matching for flexibility. Similarly, when players visit locations or complete combat encounters, the respective checking methods analyze if these actions fulfill any pending quest objectives. The matching algorithms are intelligent enough to handle variations in naming and context. When an objective is completed, the system updates the quest's step progress, potentially marking entire quests as complete when all steps are finished. It manages quest rewards through integration with the reward system, tracks recently completed quests for UI notifications, and provides mechanisms for developers to manually complete quest steps when needed. The system culminates by monitoring overall quest completion, triggering the game's ending sequence when all quests have been completed, creating an end to the player's journey.

Chapter 4: Content Generation & Management Methodologies

This chapter focuses on the methodological approaches underlying the technical implementation presented in Chapter 3. While the previous chapter detailed the architectural components and their implementation, this chapter examines the reasoning, theoretical foundations, and experimental iterations that led to our chosen approaches. We explore the methodological challenges unique to applying large language models in procedural content generation and present a framework for evaluating and improving generative systems for games.

The basic methodological challenges addressed are:

1) Designing effective prompts that balance creativity with structural constraints.
2) Developing validation mechanisms in order to maintain the content integrity.
3) Formulate context management techniques to maintain content integrity.
4) Establishing workflows for content generation and integration.

Throughout this chapter, we will analyze the design decisions that shaped our implementation, discussing alternative approaches that were considered and the rationale behind our final solutions.

4.1 Prompt Engineering Methodology

Prompt engineering is the base for a successful LLM interaction, establishing a boundary between creative variation and game-usable content. Our methodology evolved – through trial and error – from simple and unstructured to sophisticated and structured prompts.

We adopted a systematic approach to our prompt engineering, moving through a multi-phase process of development:

1) Unstructured Prompts: Our initial experiments involved using minimal restrictions, simply asking the model to, "generate a fantasy world," or "create some NPCs for an RPG." While these attempts resulted in a creative output, the model generated content that was extremely inconsistent—often skipping required fields or including irrelevant text.
2) Template-Based Prompts: We then tried providing basic templates with field names, but we noticed that the LLM still struggled with consistent formatting and included elements outside the requested structure.
3) Example-Driven Prompts: By including complete examples in our desired format, we achieved better consistency but we encountered new problems with filed validation and type constraints.
4) Constraint-Explicit Prompts: Our final approach uses a combination of structural examples with explicit rule statements, type declarations and field requirements specifications.

By transitioning from generic instructions to explicit examples with well-defined constraints we improved the output consistency up to 80% [9].

A big challenge was finding the balance between creative freedom and structural constraints. Too many constraints reduce the diversity and creativity of the generated content, while too few constraints produce unusable outputs. We overcame this issue by creating a constraint categorization framework of Essential Structural Constraints, which are non-negotiable requirements for the game to function (e.g., field presence, data types, etc.), and Content Coherence Constraints, which are guidelines encouraging thematic consistency and logical relationships. For example, in world generation, we used strict constraints to ensure that all the locations had all relevant fields (name, description, and purpose) of the appropriate data types (string). However, we provided minimal restrictions on the narrative connections, allowing the LLM to create the relationships between locations and other world elements.

Besides the general prompts structure, we approached each part of the world generation (world, quests, NPCs, encounters) with specialized prompts. For World Structure Generation, we focused on the thematic consistency and the description of the world. In NPC Generation, we focused on the speech style, the history as well as the characteristics of each NPC (e.g., emotions, disposition). Quest Generation involved structure prompting with step-based action specification. For Encounter Generation, we focused on creating prompts that balance enemy diversity and scaling constraints.

With trial and error, we identified the best possible prompt setups for each part of the world generation process. For instance, we found that in Quest steps generation, if we provide some keywords regarding the type of step, we have better chances to create resolvable steps, resulting in our ability to complete quests.

4.2 Validation Methodology

While Chapter 3 covers the technical implementation of our validation mechanism and this chapter describes the methodological strategies that informed the design fix and validation development stage.

We developed four distinct validation layers:

1) Structural Validation: validates the existence of required fields and the correct field type.
2) Content Validation: validates the relationships between elements (e.g., ensuring quests reference existing locations).
3) Balance Validation: validates numerical properties to ensure gameplay mechanics are followed (for example, damage from enemy actions is supposed to inherently scale).
4) Experience Validation: validates the generated content for player experience quality.

Using layers of validation means that every layer is designed to have focus on a discrete aspect of quality rather than seeking to hold a single layer accountable for all deviations within the content. When a validation failure occurs the validation system can apply fixes that will correct or accommodate the error rather than reject the content outright.

Our validation schema was developed through an iterative process informed by failure analysis. Rather than trying to anticipate all of the possible model failures beforehand, we:

- Collected examples of failed generations from our early experiments.

- Categorized the patterns of failures into types.
- Developed validation rules based on observed failure patterns.
- Added specific recovery mechanisms for common patterns of failure.

This process resulted in a schema that was developed to account for the specific failure modes of language models versus generic patterns of data validation [10]. For example, we discovered that the LLM would frequently return numeric values as a string and added specific type checking rules to match this pattern.

An innovative aspect of our methodology is the bidirectional relationship between validation and prompt engineering. Rather than treating these as separate concerns, we used validation failures to systematically improve prompts:

- Identify common validation failures across multiple generation attempts.
- Determine whether the failure originates from the prompt or some limitation of the LLM.
- For prompt-based failures, add explicit constraints addressing the specific issue.
- For model limitations, implement automated correction in the validation layer.

This opened up the system to systematic improvement through each generation failure leading to better outcomes in the future. We interpreted this as a co-evolution of improvements of both prompts and validation rules, with each field of domain contexts evolving to limit the limitations of the other whilst producing improvements in their own.

4.3 Context Management Methodologies

Dynamic NPC interactions require a context management to maintain thematic coherence and character consistency. Our methodological approach to this extends beyond the technical implementation as is described in Chapter 3.

A key methodological insight in our context management approach is the abstraction of "memory" into multiple layers: Surface Memory, which consists of recent conversation turns maintained verbatim; Condensed Memory, which includes summarized historical interactions; Trait Memory, which maintains persistent character attributes reinforced in each interaction; and World Memory, which contains stable facts about the game world provided as context. Our exploration of this approach revealed that LLMs handle conversation memory most effectively when personality traits are continuously reinforced even as conversation history is compressed. This finding informed our practice of including complete character trait descriptions in every interaction prompt, ensuring personality consistency despite context window limitations.

We adapted our strategy for conversation summarization through an experimental process of different summarization methods. We began with Extraction-Based Summarization and initially tested the extraction of "important" turns using a procedure of keyword analysis, but we quickly realized the extraction method was not capturing the conversational flow. We then moved to testing a strategy of Fixed-Ratio Compression that worked by applying fixed compression ratios (i.e., ratios that reduced the conversation excerpts to 30% of the original length), but we realized appropriate levels of compression changed depending on the density of the content. Our final trial method was called Range-Based Adaptive Summarization and it worked using a target

reduction range of 30-70%. In this context, the summarizer applied greater compression based on lower conversational density, and then applied less compression to the more dense improvements of knowledge. When evaluated experimentally, the range-based adaptive summarization method outperformed the fixed approaches in terms of maintaining coherence in the conversations, and the flexible process empowered the system to make contextually-appropriate decisions about what information to retain.

Although in-game functionality could have been accomplished through just simple in-memory storage, we intentionally designed a file-based mechanism for conversation persistence to allow for the opportunity to see into the development process. There were some assets to the method we employed: allowing for review of how conversations progress over multiple play-throughs, enabling sorting through a player's filtered and unfiltered experience concerning whether conversations maintained appropriate coherence, creating an annotatable dataset to assist in improving the system for future iterations, and supporting human analysis of the dialogue created by AI for assessing qualitatively whether it was of acceptable quality. Our method reflects the broader methodology of "observable AI" methodology that our development emphasized: having a means to be transparent about the behavior of the AI to enable an additional layer of review during the development phase. The file-based persistence mechanism allowed to witness how conversations evolved over time and how they followed patterns that could then be useful for informing our next improvements to the system.

4.4 Interdependent Content Generation Methodology

The sequential generation of world elements with explicit interdependencies represents one of our most significant methodological contributions. This approach the "coherence challenge" in procedural content generation—the challenge of creating connections between independently generated game elements [11].

The process that we used began with a dependency analysis of traditional RPG content to reveal the natural relationships between the elements of the game: the world conveys the setting, tone, physical locations; NPCs occupy positions in the world and refer to particular locations in the world; quests involve both NPCs and locations as participants and destinations for quests; and encounters take place in a specific locations and may or may not link in some way to the quest. This analysis recognized a natural dependency chain which helped to inform us to produce content in a sequential manner, building on and extending content rather than simply producing elements in isolation. We assessed this dependency chain approach against other content generating methodologies: a parallel approach (producing all content simultaneously and then post-process the content to make the links); a bi-direct approach (having quests first, and generating NPCs and locations as support); and player-driven approach (producing a response to the player actions). In testing the methodology, we found that sequential generation of content, based on explicit dependencies provided the most coherent results while still providing enough variance between generating sessions.

A key focus of our method is managing references from generated entities. We use explicit reference passing between stages of generation, where each stage is provided with the complete context through the prior stages of generation, allowing for more natural reference to form without attempting to establish connections through complicated post-processing. This is

different than PCG approaches, which generate content in disconnected systems or processes and later try to establish connections through mapping or transformation. By generating the content in the contextually aware sequence, our system generates game elements that integrate more naturally.

Beyond explicit references, our system facilitates thematic unity through inherited conceptual characteristics. During the world generation phase, we establish primary thematic aspects (such as genre, tone, setting) that will be utilized and plumped in downstream generation. For instance, if the world generation phase generates a "dark fantasy" world with thematic characteristics such "corruption" or "redemption," those thematic characteristics will carry through the generation process, subsequently influencing NPC personalities, quest narratives and encounter designs. This process of thematic inheritance creates a more coherent game experience than if we simply rely on individual elements referring to one another. This style of reasoning helps to solve some challenges associated with thematic mixture in game procedural narrative generation. By beginning our thematic core elements early in the generation, and then permitting those themes to radiate downstream through the generation pipeline, we achieve a greater level of coherence in the player experience.

4.5 Error Recovery and Resilience Methodologies

Error handling in AI-assisted content generation requires alternative approaches that differ from traditional error management. Our methodology acknowledges the probabilistic nature of the LLM outputs and implements a multi-layered recovery strategy.

Our implementation uses a progressive parameter adjustment approach for handling generation failures:

1) The initial generation uses parameters (top_p and temperature) with values that favor creativity.
2) If generation fails, these parameters are adjusted to reduce the creative variation (lower the top_p and temperature).
3) With each failed attempt the system implements an exponential backoff to prevent resource exhaustion.

Temperature and top-p (nucleus sampling) are two critical parameters that control how Large Language Models generate text, working together to balance creativity with coherence. Temperature adjusts overall randomness—higher values (>0.8) encourage the model to consider less probable tokens, producing more diverse and creative outputs ideal for brainstorming or creative writing, while lower values (<0.5) favor the most probable tokens, generating more predictable, focused responses better suited for factual information or code. Meanwhile, top-p dynamically limits the token selection pool—a setting of 0.9 means the model only considers tokens within the top 90% of probability mass, excluding extremely unlikely options while maintaining reasonable variation. In applications like our RPG engine, these parameters are strategically adjusted for different content types (higher for world-building creativity, lower for structured quest design) and can be dynamically modified during retry attempts, starting with higher values for creative exploration before gradually shifting toward more conservative settings if initial generations fail.

This method strikes a good balance between creative exploration and reliability. It allows the system to first explore more creative generation and only fall back to more constrained generation when needed. Its progressive nature gains the benefit of preserving as much creative variation as possible while ultimately succeeding.

A critical aspect of our methodology is the implementation of a graceful degradation hierarchy for cases where generation cannot succeed despite multiple attempts. Our approach first attempts generation with model parameters optimized for success, and if this fails, we then use saved fallback content associated with the specific content type. If there are no saved fallbacks, we then generate procedurally generated placeholders for the specific content type. If generation of the placeholder content fails, we lastly consider minimal compatible defaults. This design assumption provides assurance that even in the most extreme unicorn situations in which we cannot generate compatible content, the game is playable. The combination of multiple fallback options ensures central functionality is maintained, and degrades gracefully when required.

4.6 Integration Methodologies for Game Systems

The final challenge we faced was connecting our rich narrative content with actual gameplay systems. Instead of imposing a technical template system for writers, we created a smart system to read natural language and highlight the key gameplay elements. When our system analyzes quests, it picks out action words like "talk" or "defeat" to create player objectives. This approach allows the system to work with natural language descriptions rather than requiring structured markup or templates. By identifying action verbs and their objects, the system can transform narrative descriptions into actionable gameplay objectives.

The natural language interpretation approach represents a significant advancement over template-based PCG systems, which typically require rigid formatting that constrains creative expression. Our keyword-based method preserves creative freedom while extracting the structured data necessary for gameplay mechanics.

Our entire integration process uses a hybrid strategy that blends procedurally produced content with hand-designed game systems:

1) Manual Design: Progress systems, spatial map structure, and essential game mechanics
2) Procedural Generation: Characters, quests, world story, and encounter information
3) Hybrid Components: NPC actions, battle balancing, and spatial-narrative mapping

This hybrid strategy acknowledges the complementary qualities of AI and human innovation. While AI is excellent at producing a wide range of narrative content and character personalities, human designers are better at building stable, well-balanced systems and spatial landscapes.

Our technique outperforms both fully procedural generation and pure manual design by identifying this methodological border between manual and procedural parts. The AI-generated content is given structure and purpose by the stable framework that the human-designed components offer.

This chapter presented a comprehensive framework for integrating LLMs into PCG. The key principles emerging from this work include structured creativity using targeted constraints to channel creative variation where it adds value while ensuring technical functionality; layered validation implementing multiple validation layers that address different aspects of content quality and correctness; context-aware generation creating interdependencies between generation stages to maintain narrative coherence and thematic consistency; resilient processing designing systems that can recover from generation failures through progressive adjustment and graceful degradation; natural interpretation bridging the gap between natural language and game mechanics through keyword-based interpretation; and a hybrid design philosophy combining the strengths of manual and procedural approaches rather than relying exclusively on either. These principles form a coherent methodological framework for AI-assisted game development that addresses the unique challenges of language model integration. The framework balances the creative potential of these models with the technical requirements of functional game systems. By detailing the reasoning and alternatives behind our chosen approaches, this chapter provides a foundation for future work in AI-assisted game development. The methodologies presented here can be adapted and extended for different game genres, content types, and technical contexts.

Chapter 5: Gameplay Implementation

This chapter aims to examine the implementation of the gameplay systems and mechanics in RoQ, focusing on how procedurally generated content is integrated with the gameplay mechanics. The chapter brings together the technical infrastructure described in Chapter 3 and the methodological approaches from Chapter 4, showing how theoretical concepts are translated into gameplay features and mechanics. The Figure 19 shows the flow of our game implementation that will be analyzed in this chapter.

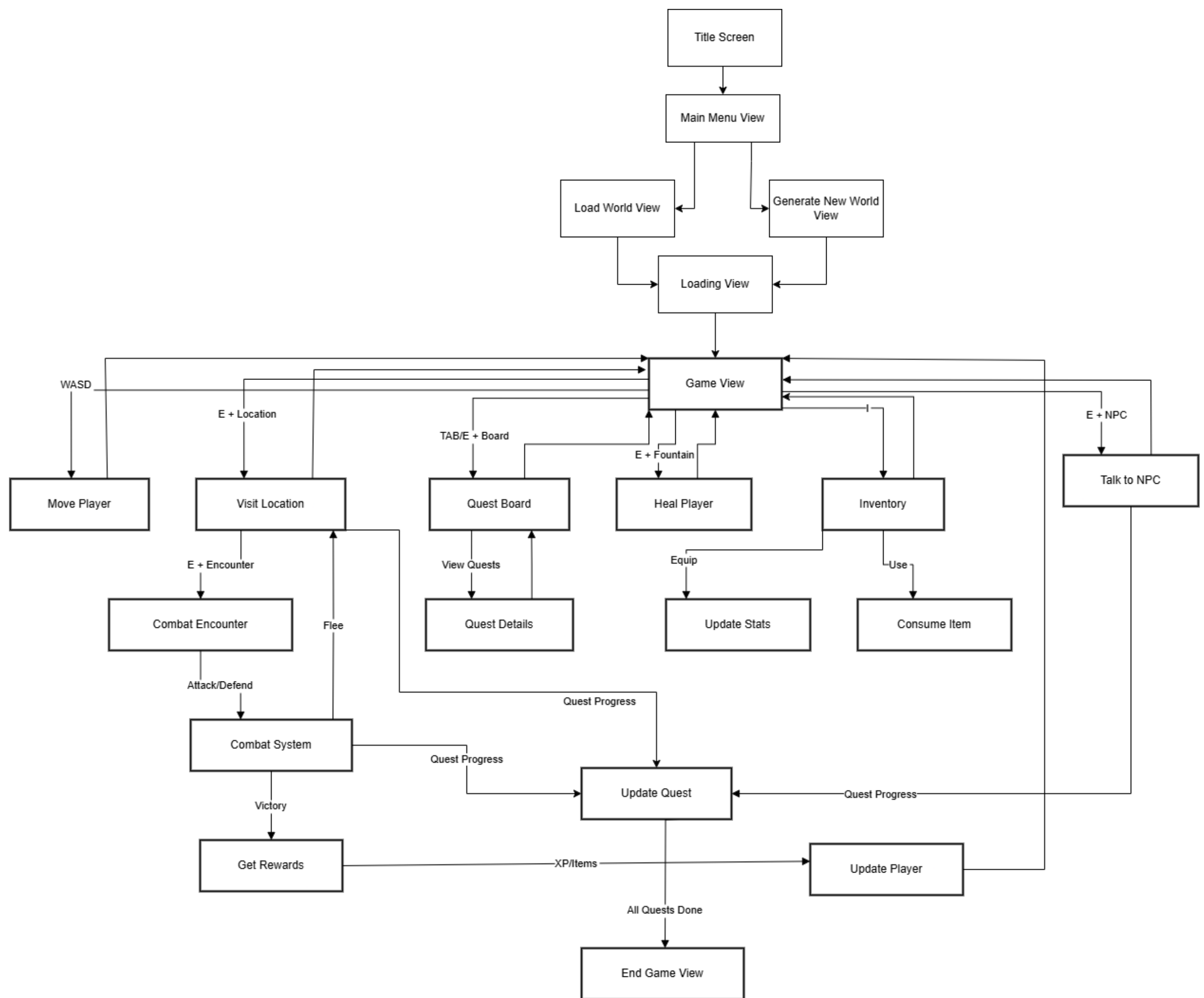

Figure 19: Complete Game flow Diagram

5.1 Menus

The first screen the player sees when starting the game is the Title Screen as seen in Figure 20

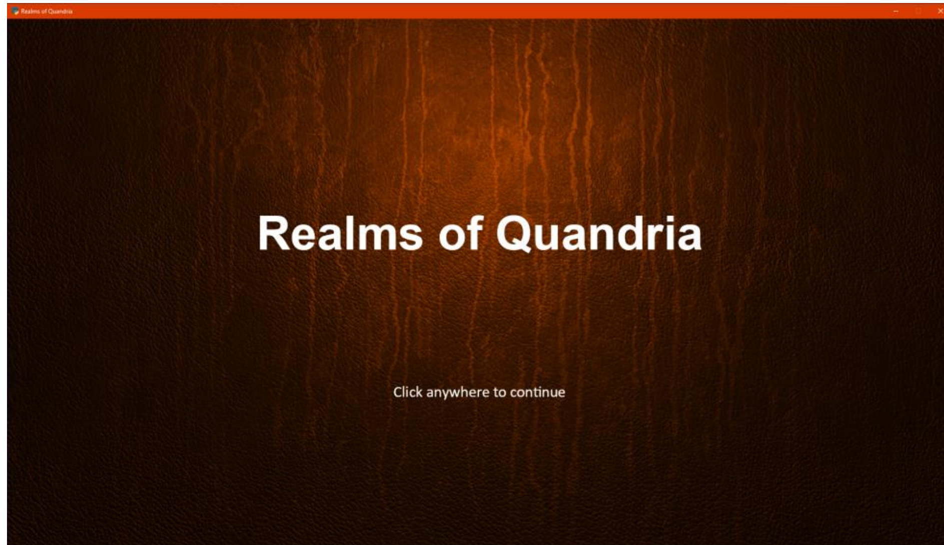

Figure 20: Title Screen

The second screen is where the player is prompt to either generate a new world, or load the existing one as can be seen in Figure 21. This is like our Main Menu screen.

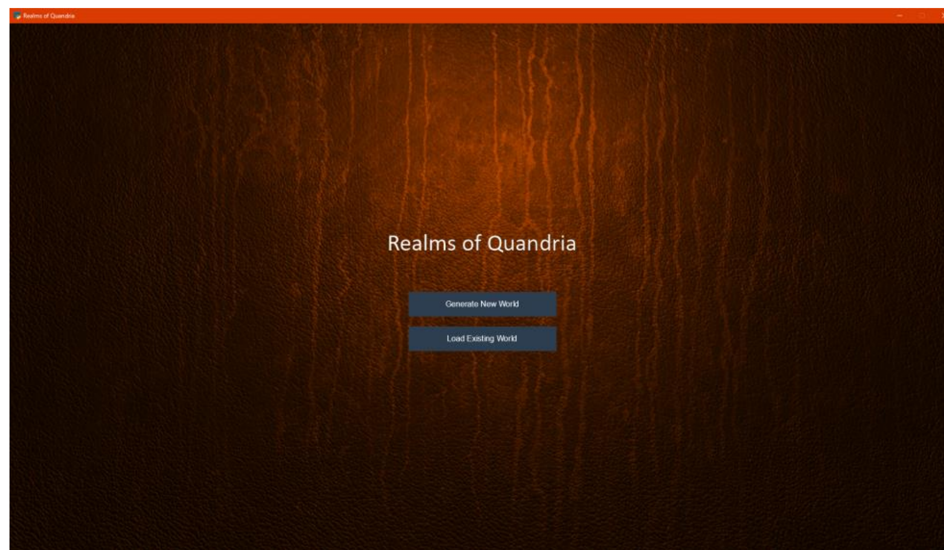

Figure 21: World Selection or Generation screen

If the player chooses the Load Existing world, the player is shown a loading bar for a brief moment and then gets into the Game. If the player chooses the Generate New World option, he is shown a screen where he is prompted with sliders and text inputs for configuring world parameters (genre, setting type, tone, complexity, number of locations/NPCs/quests/encounters) as seen in Figure 22.

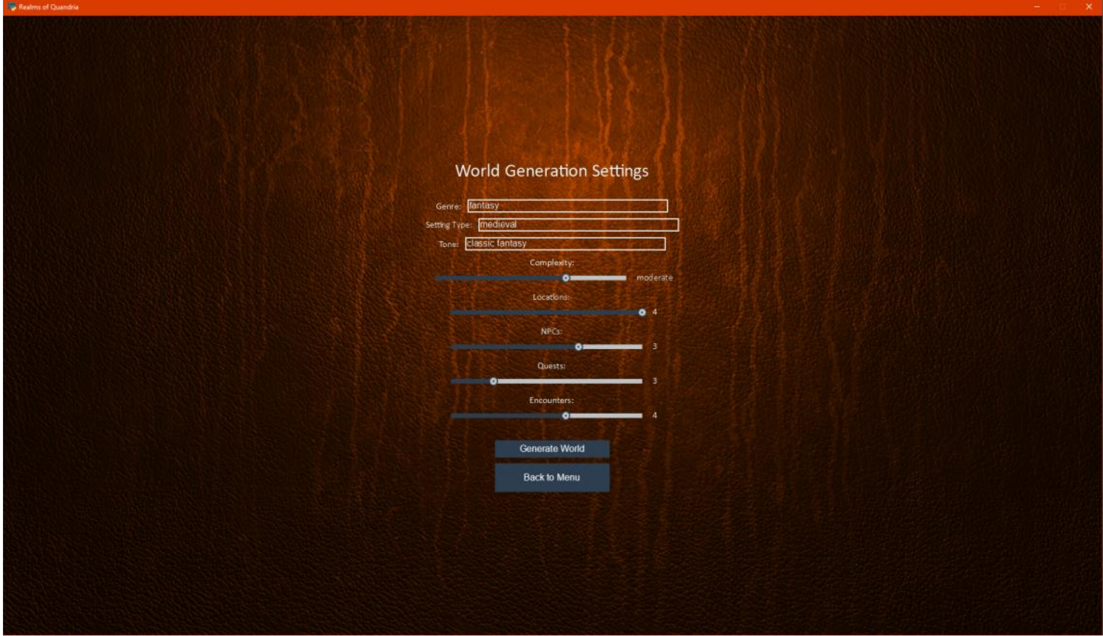

Figure 22: World Generation Settings Screen

Then while the world is being generated, the user can monitor the step progress from a loading bar as seen in Figure 23.

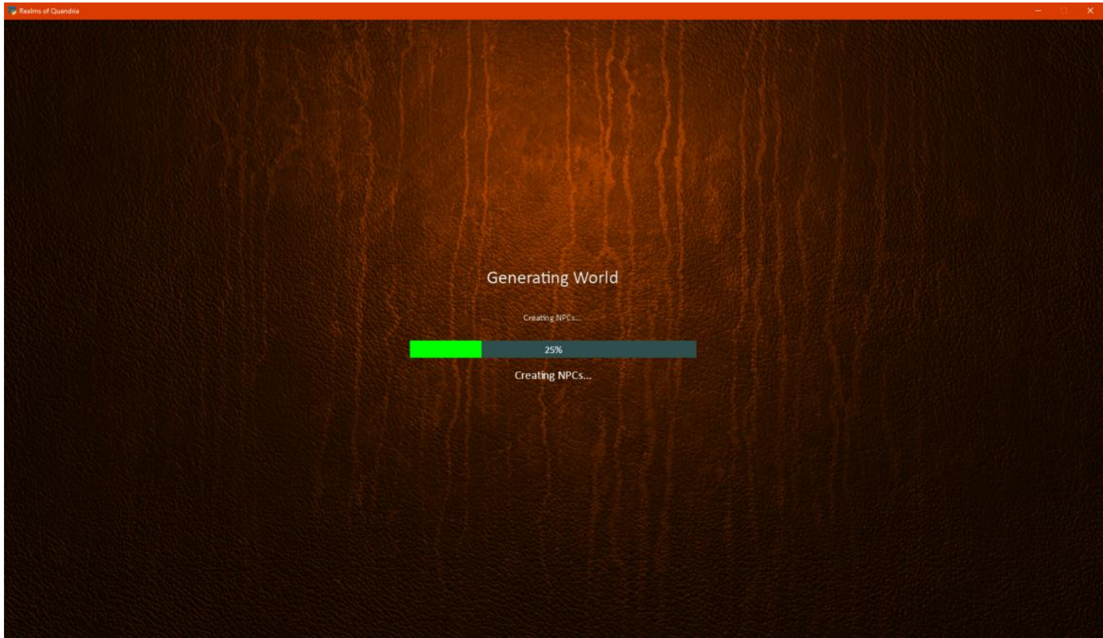

Figure 23: Loading Screen where the world generation progress is displayed.

After the game is completed, the player is shown the end screen, where the player's level and stats are displayed as well as the quests that were completed as seen in Figure 24.

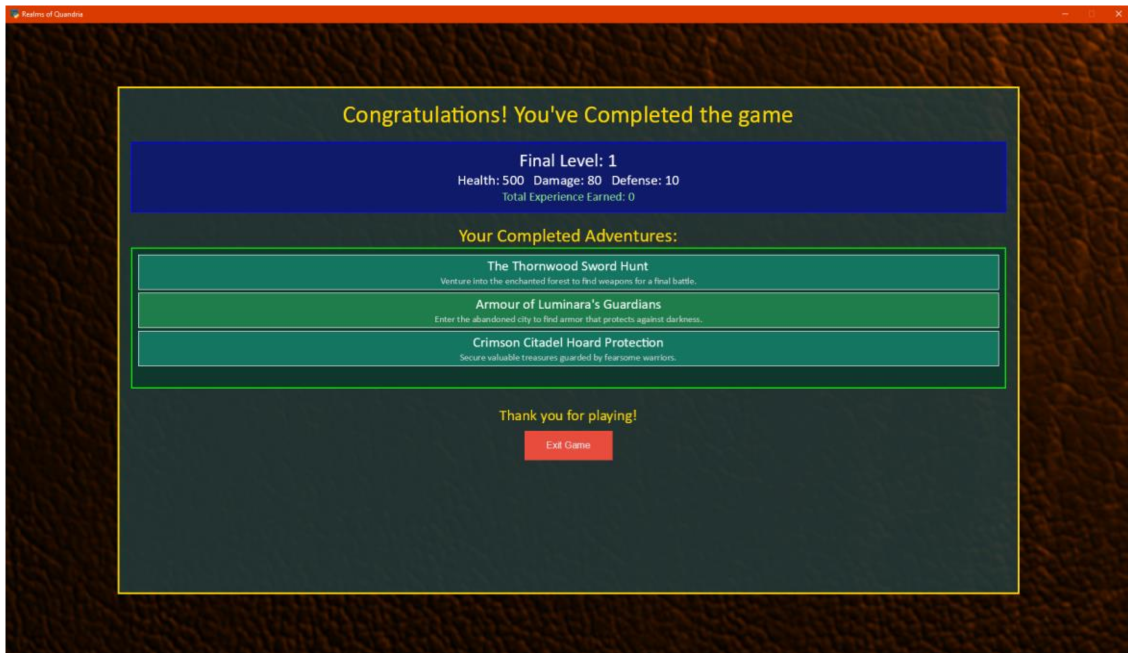

Figure 24: End screen with game summarization

5.2 Game Entities

The game entities in RoQ follow a traditional RPG structure all while integrating procedural generated content. There is a player who can explore, interact and talk with NPCs, engage in combat and progress the character, all with enhancement by LLM-generated content.

5.2.1 Player Characteristics

The player controls are fundamental. He can move using the keys: W, A, S, D each moving him in a direction as seen in Figure 25. He can access the inventory by pressing the key I, see the quests and their progress at any time by pressing TAB and can interact with NPCs, locations, encounters and the Gem of Healing by pressing E.

Figure 25: Character Movement

The player also has several attributes in lines with every RPG character. He has health, damage, defense, exp (experience) and a level.

The progression system uses a formula-based approach for leveling, as described in Chapter 3.4. The player's health increases with every level, damage increases on even-numbered levels, and defense increases on odd-numbered levels. Additionally, every tenth level provides a significant boost to all attributes.

This systematic approach ensures balanced progression throughout the game while still allowing for meaningful character development. The player's stats directly influence combat effectiveness, making progression feel impactful.

5.2.2 NPC

The NPCs do not have any stats, or movement mechanics as they are stationary. The spawn points of the NPCs are loaded at random, from a predefined list created using the Tiled map editor. The only functionality they have is the interaction with the player and the conversing with him. When the player interacts with an NPC a dialog window is shown where the player writes and sends the message to the LLM; when the response is ready it is printed in the dialog window as seen in Figure 26.

Figure 26: Player and NPC chat window

If the NPC is involved with one or more quests, upon interaction it marks that quests step as completed.

5.2.3 Enemies

Just like with the NPCs, the enemies do not have any movement mechanics. But they do have health, damage and defense statistics as can be seen in Figure 27. They do not have any sort of controls. Their attack and defend moves are handled by the Combat System as is explained in Chapter 5.5.1.

Figure 27: Enemy stats

5.2.3 Gem of Healing

Within the game world, the player will discover the Gem of Healing. Upon interacting with it the player's health is restored to full as can be seen in Figure 28. This healing mechanic provides a strategic resource for players between challenging encounters, especially valuable after difficult battles when healing potions might be scarce.

Figure 28: Gem of Healing restoring the player's health

5.3 Game World

The game world view acts like a central hub for the player as seen in Figure 29. Inside this view are located all the NPCs with whom the player can interact and converse, there is the Quest Board where the player can view the quests for this world, there is the Gem of Healing which upon interaction heals the player to full HP and lastly there are the location-interactions which upon interaction transport the player to a new location.

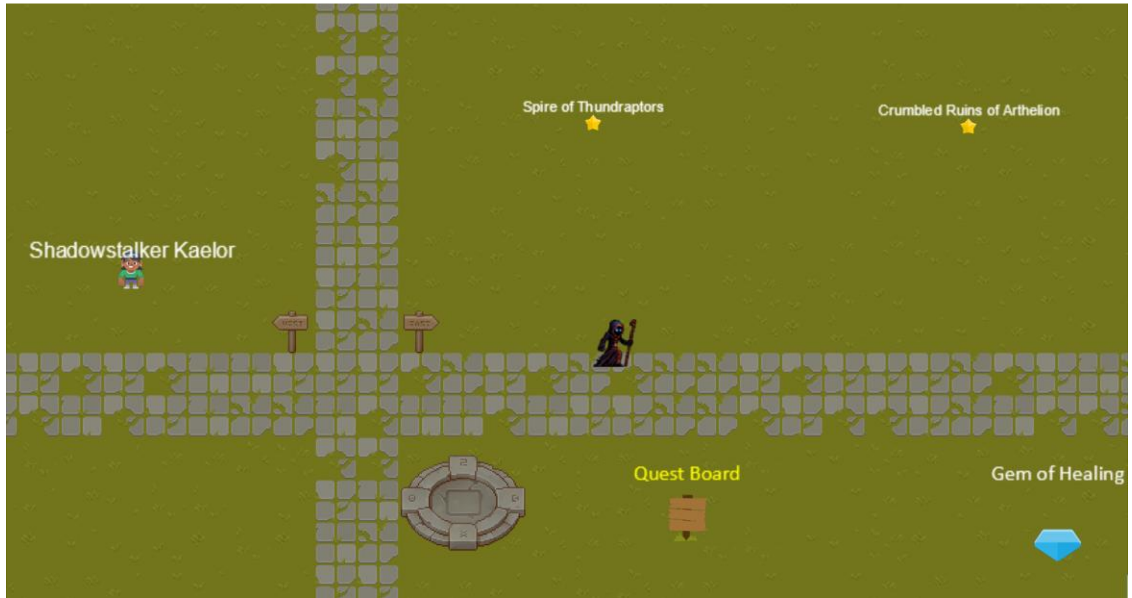

Figure 29: Game Hub

5.4 Locations

As mentioned before, the location's entry point is located in the hub, it is depicted as a star with the name of the location above. Upon interaction the player is transported to the wanted location. The current location's layout is randomly selected each time from one of four predefined locations created with Tiled. Inside each location the player may find an encounter – if an encounter is created for this location- and interact with it to start the combat as can be seen in Figure 30. Upon first visiting each location a check is initiated to see if this location is listed in any quest step. If it is listed, then the current step is marked as completed.

Figure 30: Location view with an encounter

5.5 Encounters

Upon interacting with the encounter in the location, the player is transitioned to the combat screen as seen in Figure 31. There are spawned the enemies of that encounter one to four enemies depending on the complexity of the world and the difficulty of the encounter.

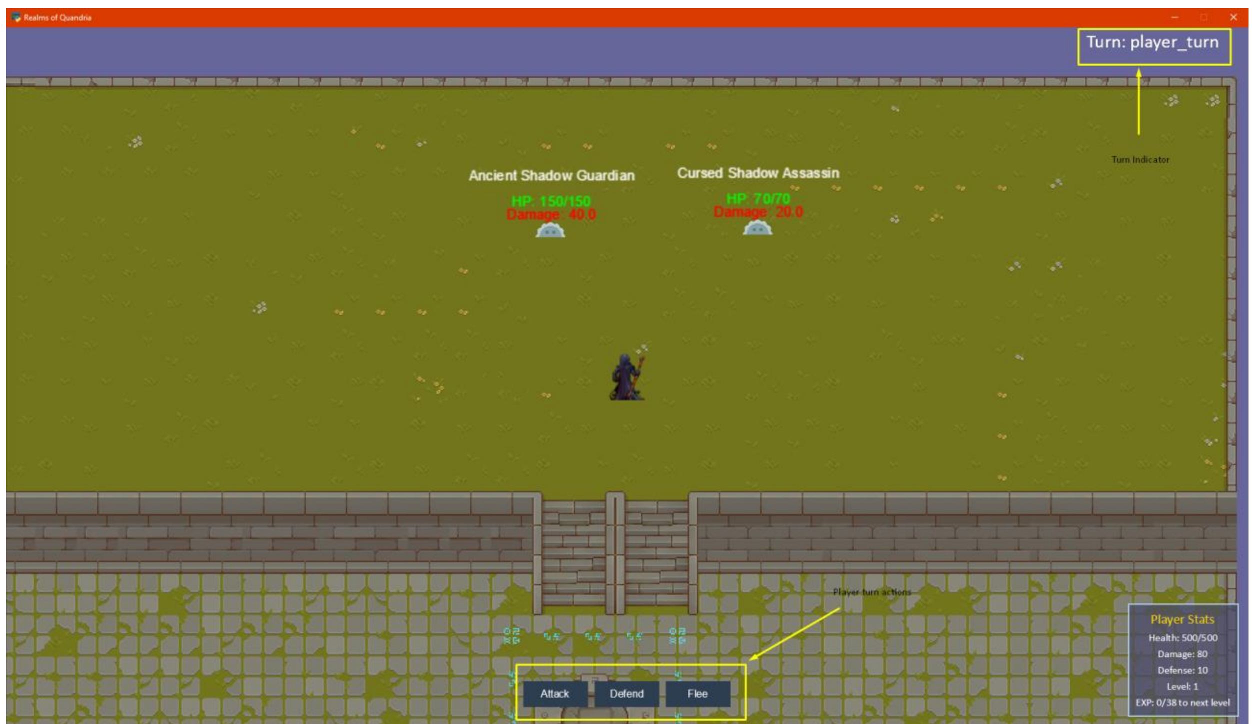

Figure 31: Combat screen

Combat continues until either all enemies are defeated or the player's health reaches zero.

5.6 Game Systems

The game consists of four visible systems; the combat, inventory, leveling and quest systems. Each of them handles a specific aspect of the game, though some of the systems are interconnected.

5.6.1 Combat System

The combat system implements the turn-based encounter model mentioned before. It maintains the state machine for the combat, tracks the turn order and processes the combat actions. There are 3 possible states of combat, the player turn, the enemy turn and a waiting state that acts as a buffer between the other states.

Combat begins with the player having the first turn, followed by enemy turns cycling through all active enemies. The player can chooses between three actions:

1) Attack: deal damage to the clicked enemy based on the player’s damage attribute

2) Defend: reduce incoming damage based on the defense formula: total_defense = base_defense + (base_defense * 0.6).
3) Flee: escape the combat in case it is difficult for the current level.

The enemies follow a very simple probability-based decision making process:
1) 50% chance to attack the player.
2) 50% chance to defend, reducing the incoming damage of the player by half in the next turn.

Upon victory, players receive rewards including experience, gold, and potentially items and are returned to the game hub.

The system successfully integrates LLM-generated enemy definitions by applying their attributes (health, damage) directly to the combat calculations, ensuring that narrative descriptions of enemy power translate to appropriate gameplay challenge.

5.6.2 Leveling System

The leveling system manages the character progression. It handles the exp calculation, the level determination and the stat increases. To calculate the experience, we implemented a custom non-linear progression curve using the base_exp * (level ^ exponent). Any exp remaining is carried through the next level. To upgrade the player stats we follow a systematic pattern where health increases every level, the damage on even levels and the defense on odd levels. Every ten levels we provide a significant stat boost to the player.

Upon leveling up the game displays to the player a notification indicating the new level and stats as seen in Figure 32.

Figure 32: Level up stats increase notification

In order for the game mechanics to work correctly, the leveling system is integrated with other systems that may result in a character level up. These systems are the combat system the quest system and the rewards system.

5.6.3 Quest System

The quest system represents one of the most sophisticated and complex components in the game, responsible for tracking narrative progression and interpreting LLM-generated quest content into actionable gameplay objectives.

Players can access the quest management interface through two distinct methods. The first option involves locating and interacting with the Quest Board, a physical object in the game world that resembles a wooden notice board. When approaching this board, a prompt appears instructing players to press 'E' to interact with it. Alternatively, players can press the TAB key at any time during gameplay to instantly access their quest log, providing convenience regardless of their current location in the game world.

In the quest view, players are immediately greeted by a list view of all available quests. Each quest list view contains key pieces of information at a glance, including the displayed title of the quest, a symbol to indicate its status (completed, etc.), and a short excerpt of the quest description. In addition, with each quest entry, there is a button labeled "View Details" next to the title, which allows players to take a closer look at the specific quests. At the top of the quest interface is a filtering system that allows players to organize their quests into three categories: All Quests (showing all the quests available to the player), Active Quests (showing quests that have started), and Completed Quests (showing quests that have been completed) as can be seen in Figure 33.

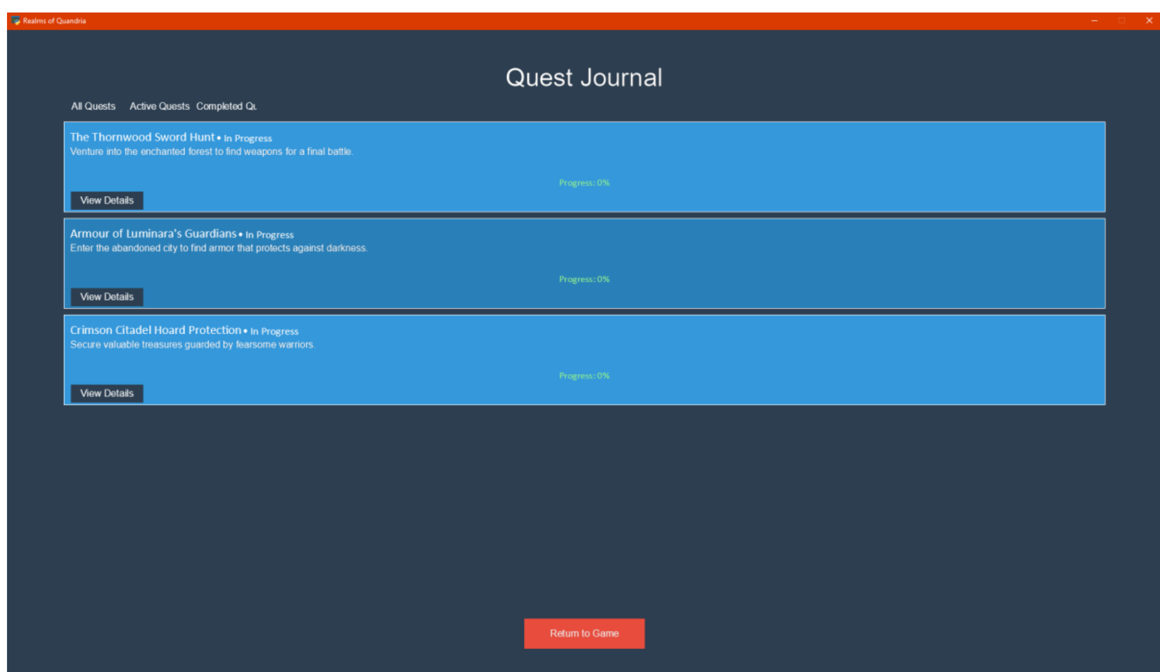

Figure 33: Quest screen containing all quests.

Once players have chosen a quest from the quest list, the view will transition to the more detailed quest view, giving detailed information about the selected quest. The detailed quest view

prominently displays the full quest description. Below the quest description, players will see a detailed outline of the specific objectives that need to be completed in order to finish the quest, where each step is clearly identified as either completed or not.

In addition to the quest description and outlined objectives, the detailed quest view provides important information about quest locations and characters that include a section titled, "Locations," that identifies specific areas in the game world that need to be visited in order to progress in the quest. There is another specific area entitled, "Involved NPCs," where players can see a list of pertinent characters that only need to file in a game world. This information allows players to better plan out their travels to complete the quest objectives by knowing where they can go, and who they will need to speak to.

Quest rewards are displayed on the expanded view. While completing quests, players typically earn experience points toward their level, in-game currency (gold) and often items that may include special weapons or armor. Each reward item will have its own attributes listed. If players find that they are unable to complete a quest or quest step, the game provides a fallback option indicated as "Complete Step (If Stuck)". This is accompanied by a warning that indicates rewards will not be provided if quests are completed manually. This option ensures players do not become "permanently" locked out of a quest due to unintended consequences with procedural generation options.

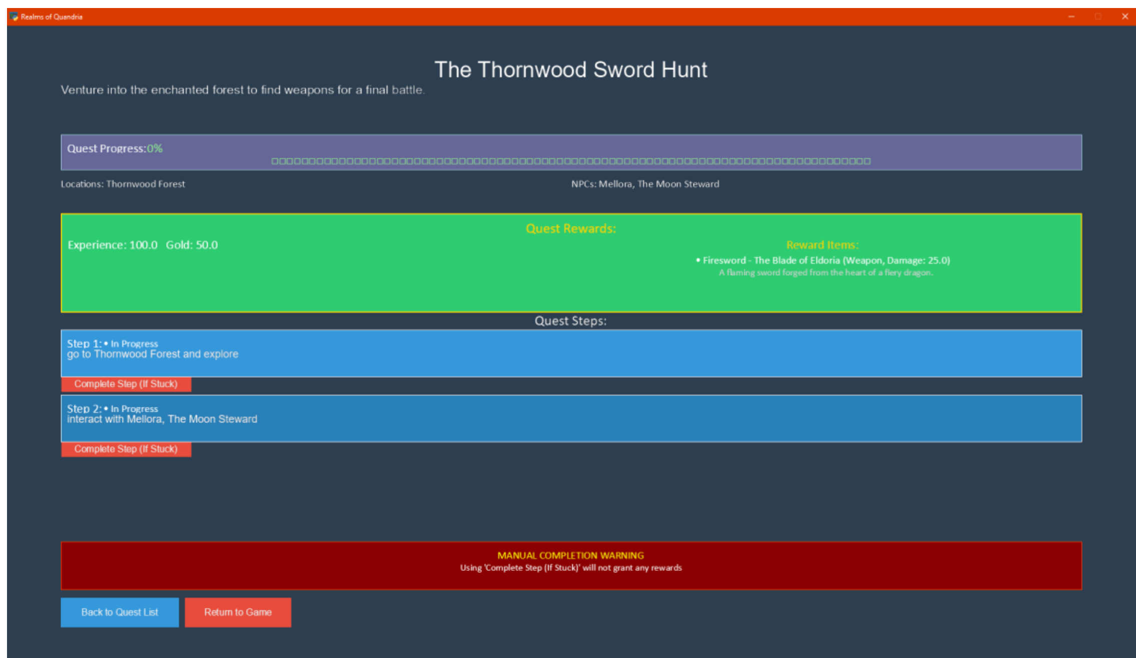

Figure 34: Quest detailed view

As the player progresses through the game, the quest system continuously checks if any step is completed. Completed steps are automatically marked when the corresponding actions are performed, whether that involves conversing with an NPC, visiting a location, or defeating enemies in combat. The system interprets player actions contextually, matching them against quest objectives through keyword recognition and entity matching. This can be seen in Figure 35 where the first step is completed.

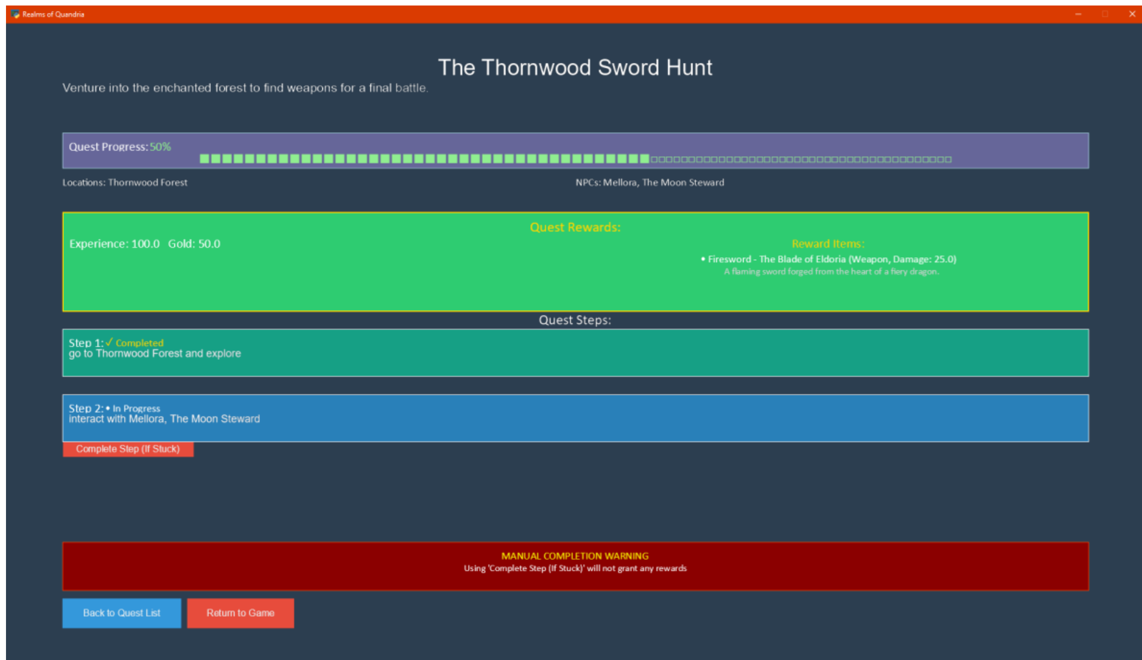

Figure 35: Quest progress update

Upon completing all the available quests, the end of the game is triggered, showing the end screen.

5.6.4 Inventory System

The inventory view in RoQ as seen in Figure 36 consists of a 24-slot grid interface that allows the player to manage the items they have acquired in the game. The inventory is accessible by pressing the "I" key, which temporarily pauses the game and presents a view of the player's entire inventory of weapons, armor, potions, and miscellaneous loot. Each item appears as an icon with color-coded backgrounds—red-orange for weapons, blue-violet for armor, green for potions, and a neutral gray for general loot—providing instant visual categorization. Hovering over an item will display a tooltip with comprehensive information about the item, including a title, type, statistics, and descriptive text produced by the LLM.

The interface employs an intuitive right-click interaction system that provides contextual options based on the item type. For weapons and armor, players can equip them to their respective slots (main hand or chest), immediately applying their statistical bonuses to the character's attributes. When equipped, items display a small "E" indicator in their slot and appear in the equipment panel on the right side of the screen. For consumable items like potions, the "Use" option immediately applies their effects—typically healing the player's health by the potion's specified protection value. The right panel of the inventory screen serves as an information hub, displaying the player's current statistics including health, damage output, and defensive capabilities. Below these stats, the currently equipped items section lists all active gear.

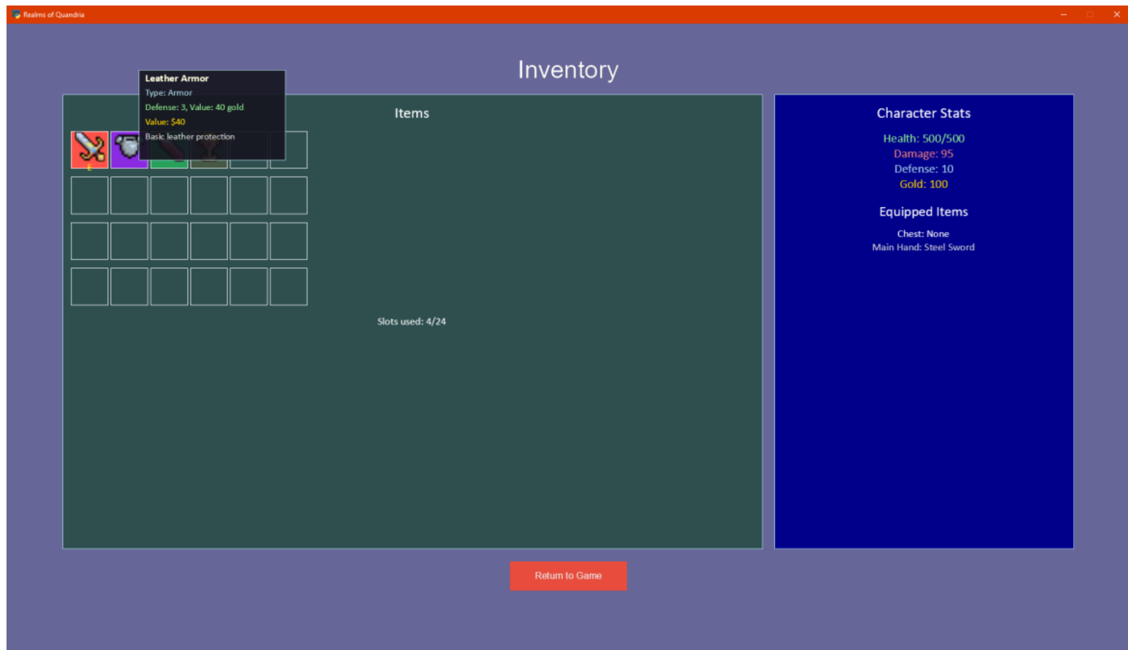

Figure 36: Inventory view showing items, tooltips, character stats & equipped items

Chapter 6: Evaluation and Results

The integration of the LLM into our game presented unique challenges that required extensive evaluation across multiple aspects. This chapter examines the performance and reliability of the RoQ system through technical analysis and qualitative assessment. We evaluated the system's computational performance and the technical robustness of our implementation. By analyzing response times, success rates and error patterns we gained insight into possible points of failure and how the current implementation might scale and evolve in practical game development contexts.

All the tests are performed in a system with the following specifications:

Part	*Spec*
CPU	Intel i7-6700
GPU	NVIDIA RTX 3070 (8GB)
RAM	32GB

The main focus of our evaluations is system performance and robustness. To find out if this system can be applied to real world applications we needed to test all endpoints to determine if the time of the requests is in accordance with user experience best practices [12]. In order to do so, we implemented an automated testing framework.

To test and run the LLM locally, we used the Ollama [13] framework. Ollama is a framework that allows users to run open-source LLMs locally on their machines. Among the plethora of available LLMs we tested Gemma2 [14], WizzardLM [15] and Gwen2 [16]. From our testing we saw that Gemma2 was slower and more inconsistent than the other models. WizzardLM was faster than Gemma2 but the responses were inconsistent as well. We settled on Gwen2 which was the faster and more consistent between these three.

This framework tests all the endpoints used in the generation of the complete world, one at a time for a total of 100 tests per endpoint. The data used for each test iteration were:

Field	*Data*
Genre	adventure
Locations	4
Setting type	medieval
Tone	classic fantasy
Complexity	moderate
NPCs	4
Quests	3
Encounters	4
Prompt	Hello, what can you tell me about this place?

It logs metrics per endpoint, as well as some overall metrics. Some of these metrics are the success rate of our requests, the average, minimum and maximum duration of each request. The output of these metrics is stored in files, one for the errors that occurred, one timing all the

requests and one with various metrics per request. Then after the evaluation is complete, we process the data we gathered to visualize our findings.

6.1 Response Time Analysis

Response time is a critical metric for understanding the real world viability of LLM integration in game development flows. The following figures show the distribution of response times across our endpoints: world generation, NPC generation, quest generation, encounter generation and NPC conversation.

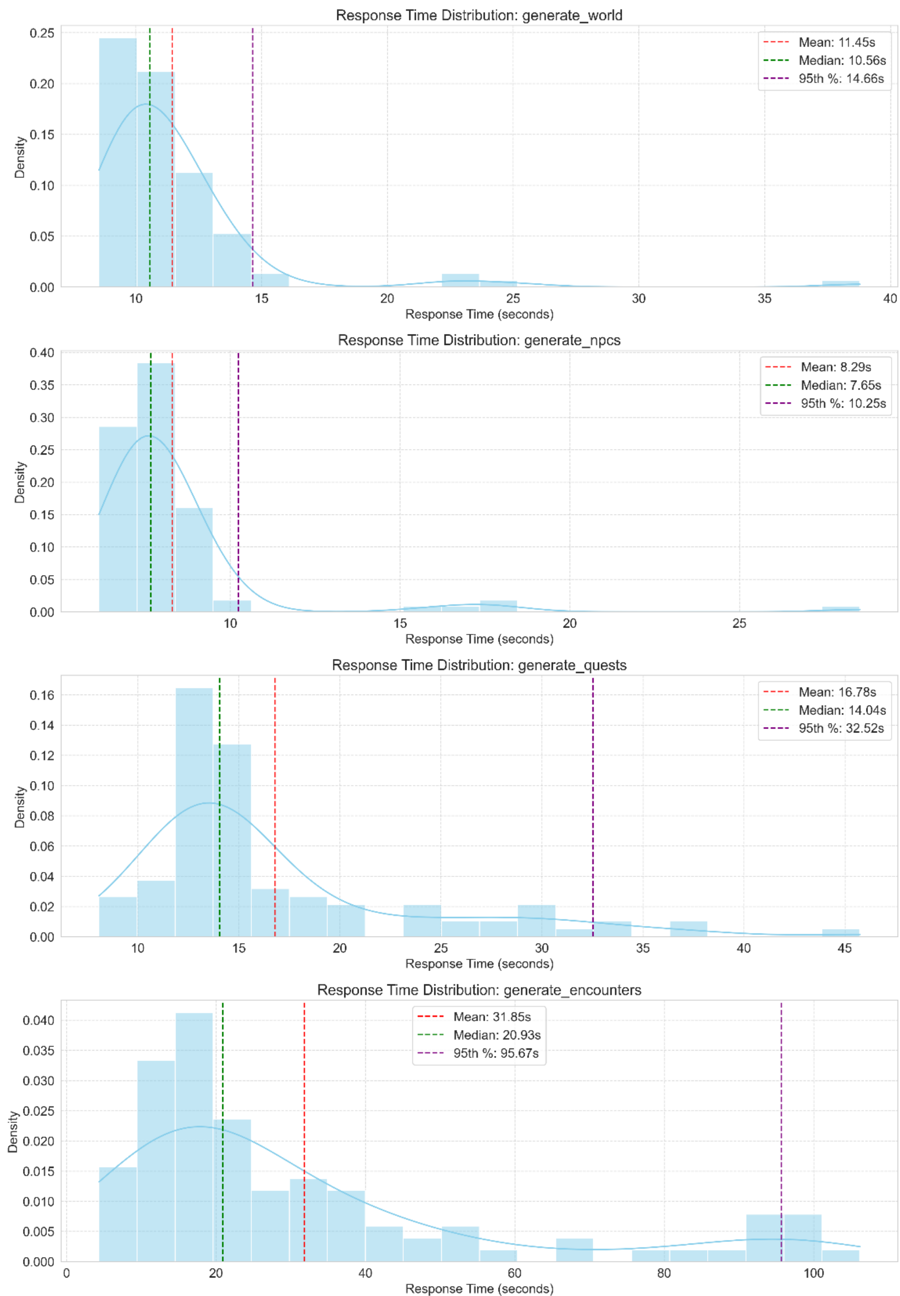

Figure 37: Generation process response time distribution.

As seen from the visualization, response times varied significantly across different content types. World structure generation maintained the most consistent performance profile regarding the generation requests with median response times of 10.56 seconds and relatively narrow interquartile ranges, suggesting predictable performance for basic world creation. In contrast, encounter generation showed the widest variance with a median of 20.93 seconds but extremes reaching higher than 95.67 seconds in some cases, indicating greater computational complexity and potentially more challenging validation processes (or issues) for this content type.

Quest generation demonstrates an interesting middle ground, with generally consistent performance (median 14.04 seconds) but occasional outliers, particularly visible in the 95th percentile measurements (32.52 seconds). This pattern aligns with our implementation approach, where quests require complex narrative structure while maintaining connections to both world elements and NPC characteristics.

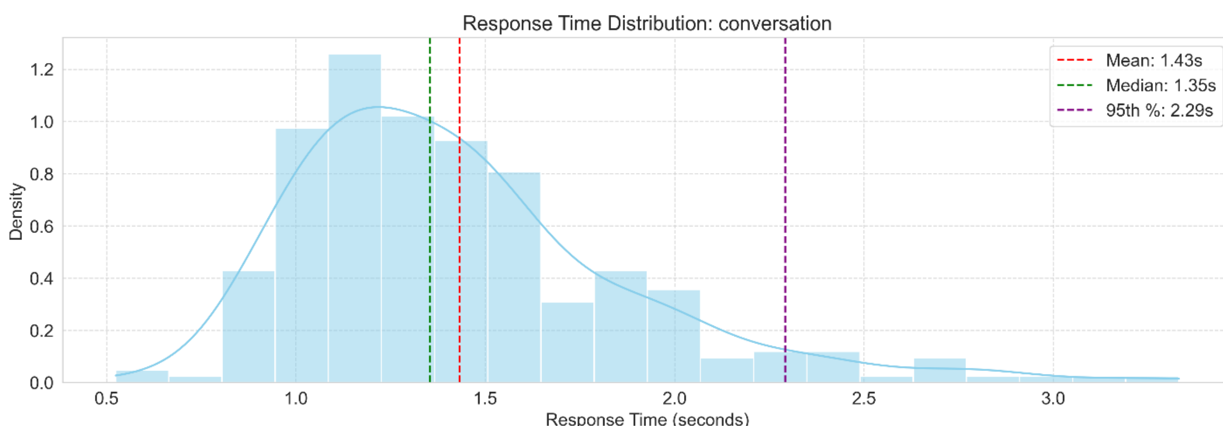

Figure 38: Conversation response time distribution.

As can be seen in Figure 38 the most consistent performance of all the requests is the conversation's endpoint. With a median response time of 1.35 seconds and a maximum of a little higher than 3 seconds.

6.2 Performance stability

For real world applications the stability and predictability are equally important as speed. To evaluate this aspect we analyzed response time patterns across 100 generation iterations.

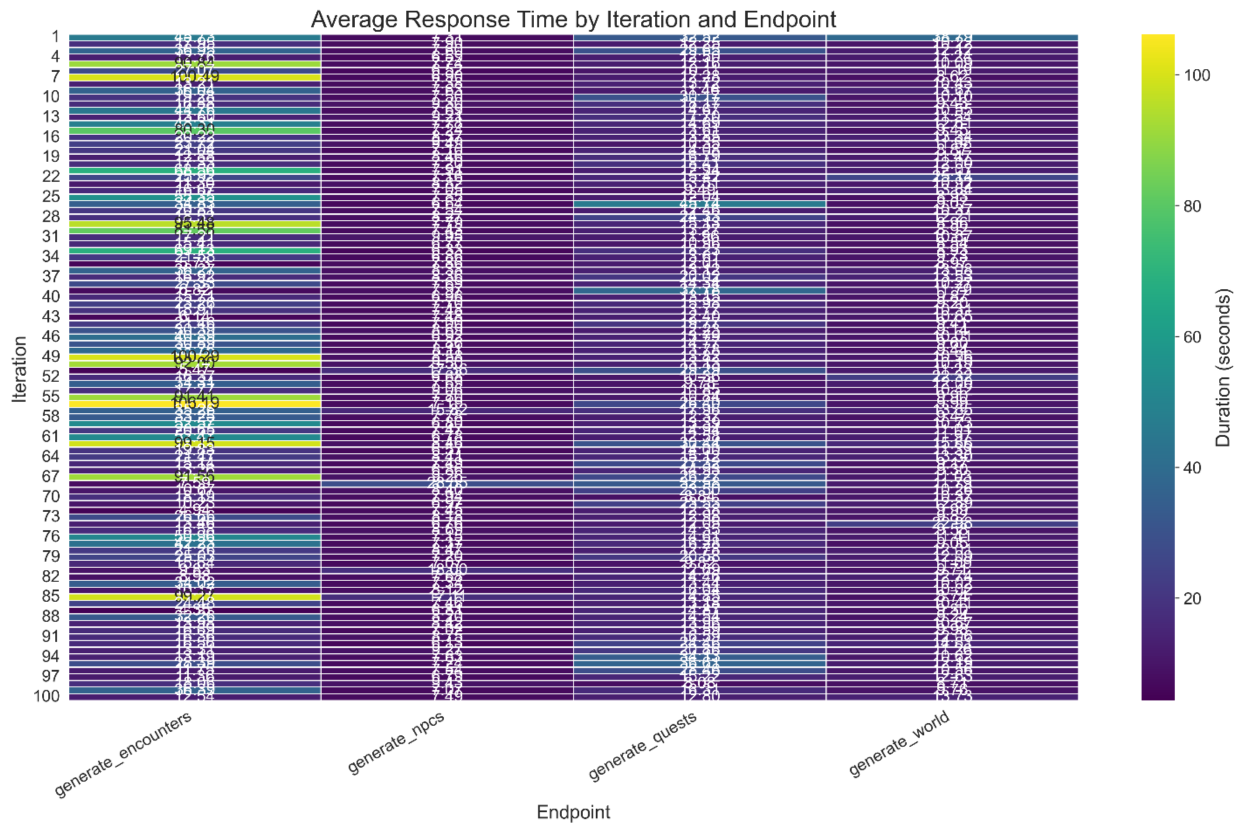

Figure 39: Response Time heatmap visualization.

The heatmap visualization as seen in Figure 39 reveals some interesting patterns across the iterations. Most notably we see that the performance remains relatively stable for all requests across iterations except the encounters generation. This suggests that the generation was failing and our validation mechanism was operating successfully.

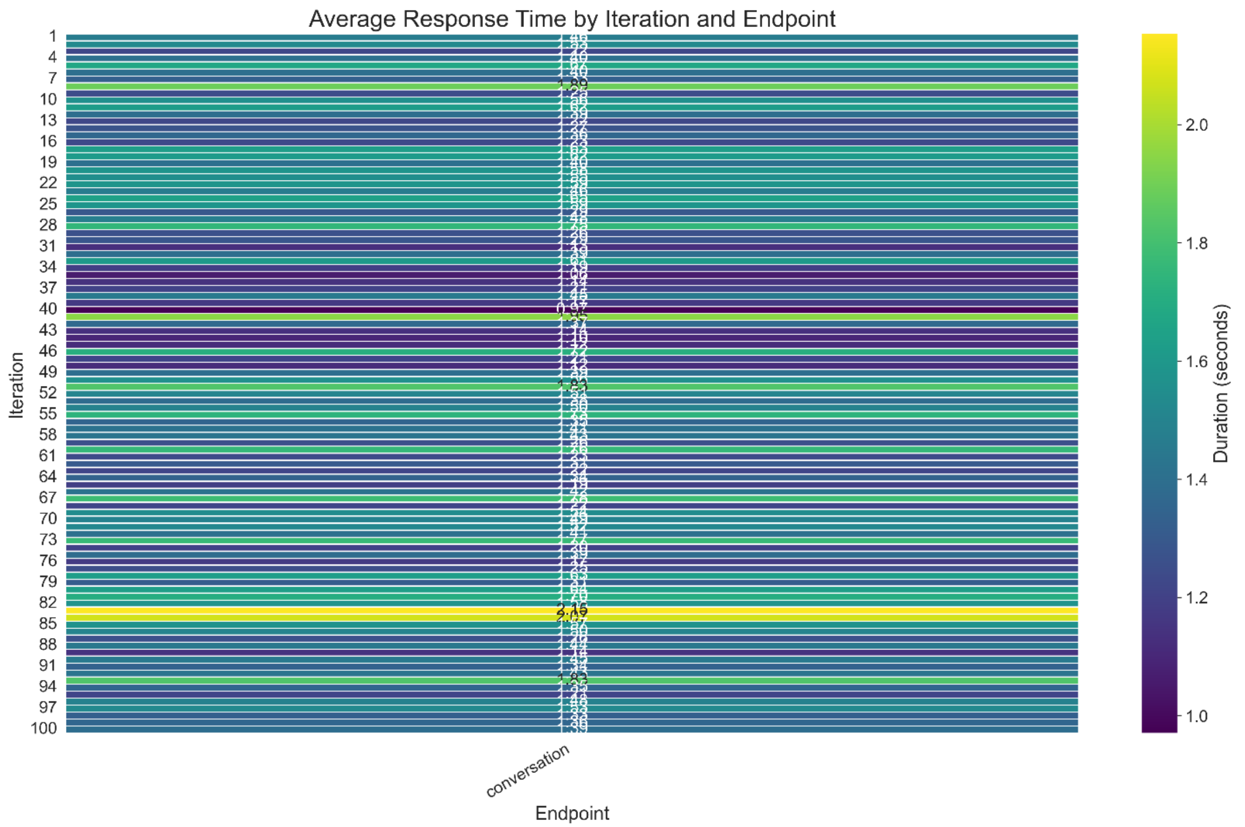

Figure 40: Conversation response time heatmap.

Looking at the heatmap for the conversation endpoint in Figure 40, we can see a fluctuation in the duration. However, due to the low response times, this is well within acceptable range. These variations likely stem from the dynamic context management system that adapts to conversation length and complexity while maintaining responsive interactions.

The complete pipeline performance analysis in Figure 41 shows that full world generation (including all four content types) required an average of 50.63 seconds across 100 iterations.

While this duration exceeds what would be acceptable for real-time gameplay generation, it falls within reasonable parameters for game initialization or level loading processes.

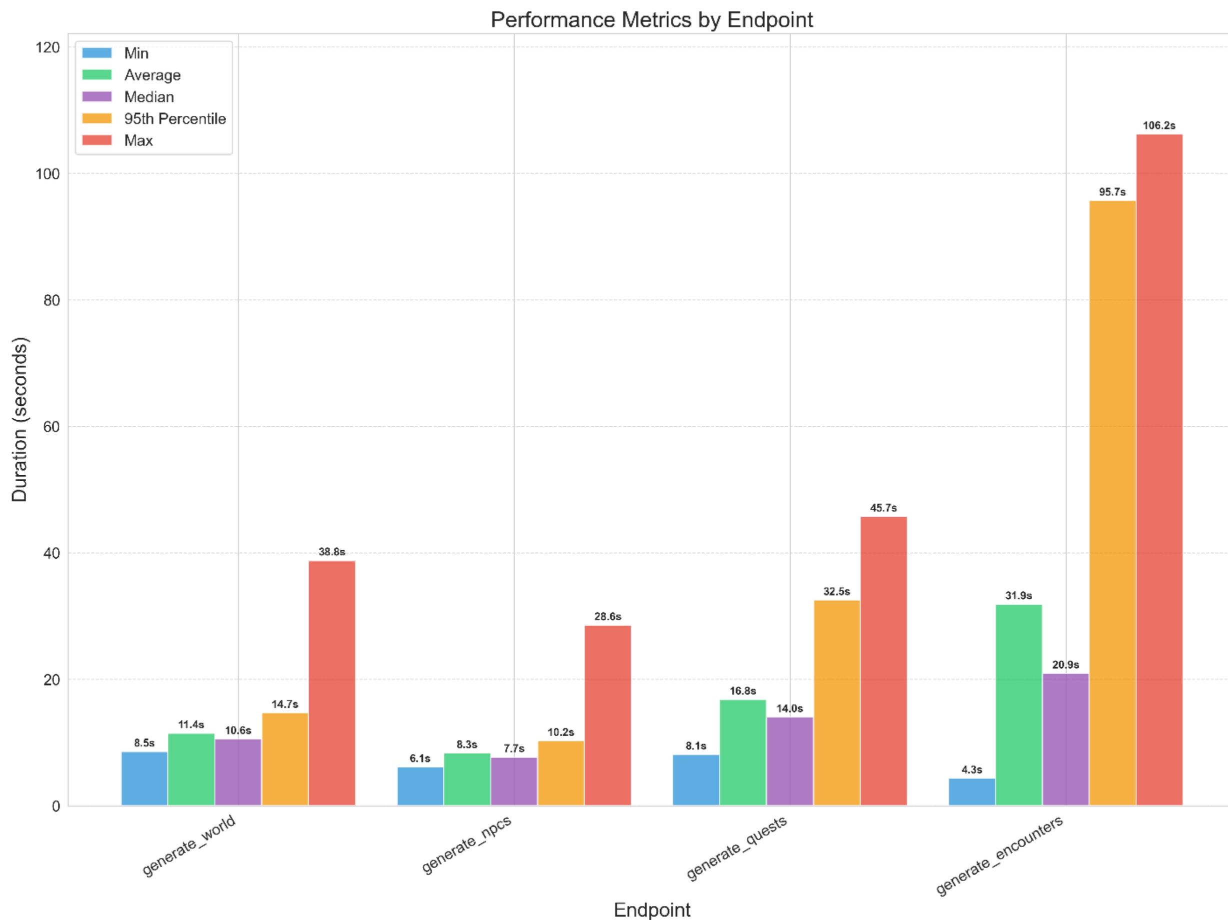

Figure 41: Performance per endpoint

Similarly for the conversation endpoint across 100 iterations as seen in Figure 42 we get an average of 1.4 seconds. This time is completely acceptable for a real time communication between the player and the NPC.

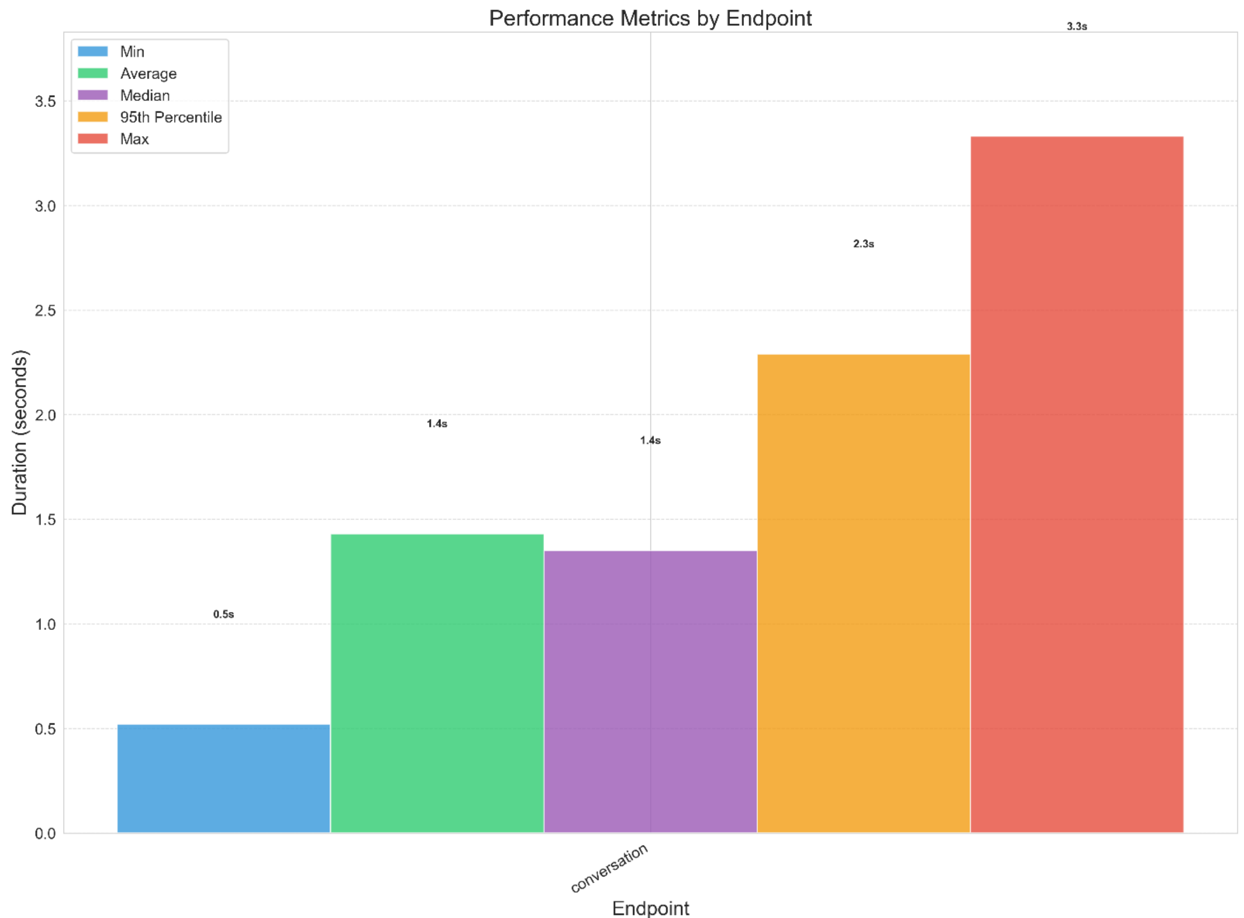

Figure 42: Conversation performance.

6.3 Success Rates

Besides performance, system reliability is essential for practical applications. The cumulative success rate analysis shows the overall success of our system. Of our four endpoints for world generation only one does not have 100% success rate as can be seen in Figure 43. The encounter generation which falls down to 85% success rate. The encounter generation failures primarily stemmed from validation issues where the generated content could not be reconciled with the structural requirements of the game system.

The two errors that we observed were a validation error ("Validation error at encounters -> 2 -> rewards -> loot -> 0: 'value' is a required property") and a parsing error that occurred due to bad generation (Failed to generate encounter structure after 5 attempts. Last error: Failed to parse LLM response as JSON: Expecting ',' delimiter: line 12 column 62 (char 373)").

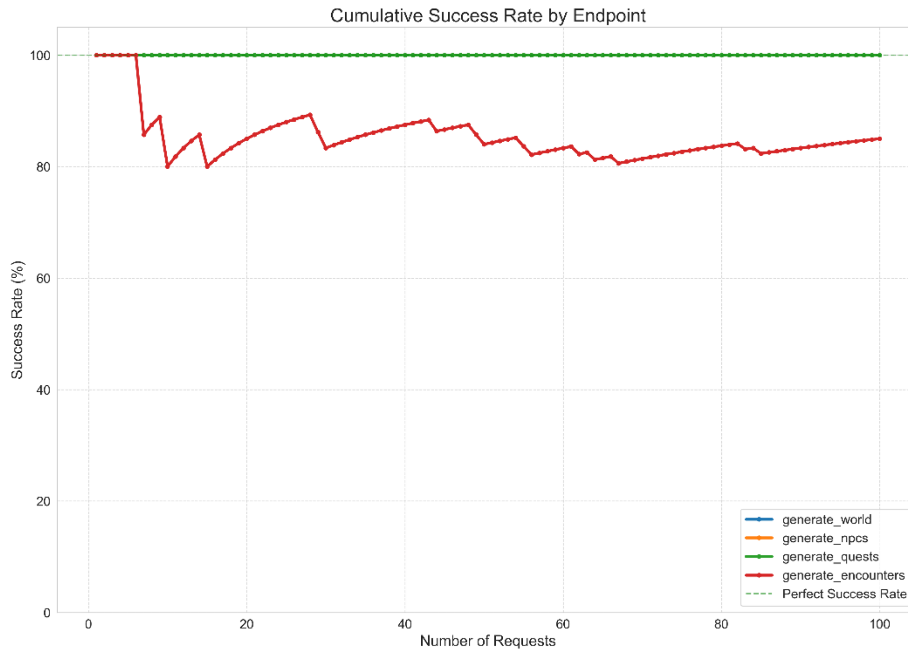

Figure 43: Success rate per world generation endpoint.

For the conversation endpoint we have a 100% success rate across all tests.

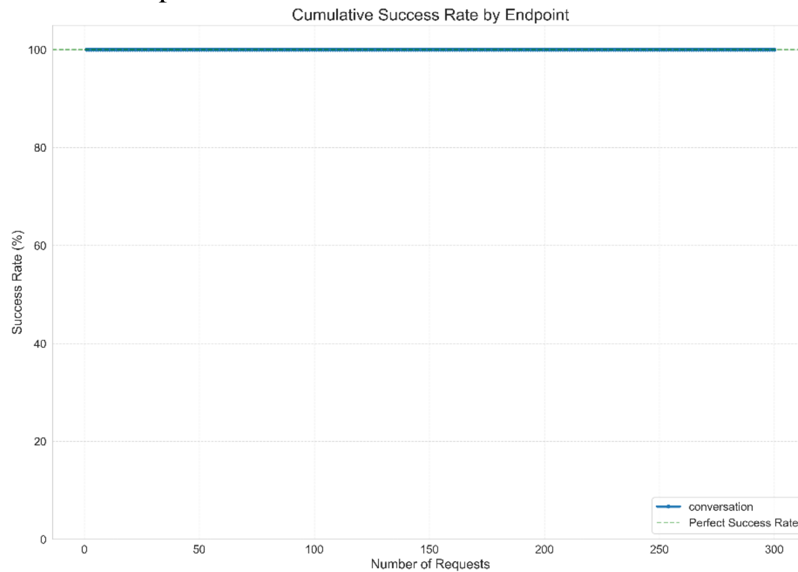

Figure 44: Success rates for conversation endpoint.

6.4 Conclusion

The RoQ system effectively balances the computational demands of LLM-driven content generation with the practical requirements of game development workflows. Despite the complexity of generating interconnected narrative elements, the system maintains acceptable response times and high reliability rates. The architectural choices, particularly client-server separation and sequential content generation approach, validate the system's performance. While further optimizations are required for a production application, the current performances fits well with the scope of this research.

Chapter 7: Future Directions and Conclusions

The conceptualization and assessment of the RoQ system presents several significant implications for videogame development and AI integration whilst also highlighting several new ways for future research and development.

7.1 Implications

The successful deployment of LLM-driven content generation in RoQ has a number of relevant implications for organizing game development practices. By automating the generation of narrative elements, character backgrounds, and designs of encounters, our system alleviates one of the fundamental bottlenecks in game development - content generation. This allows smaller development teams, in particular, to create games with scope and depth that would normally require significantly more resources.

Our hybrid methodology illustrates that procedural generation and manual design can actually complement each other rather than represent competing methodologies. The success of RoQ indicates that it is important to identify the elements of a game that would benefit most from procedural methods as opposed to manual design, as this makes development much more efficient.

The performance metrics of NPC conversations, relating to the average of 1.4 seconds with reliable output, suggests that generating certain types of content during runtime is possible within acceptable user experience levels. This creates opportunities for games that can adapt to player action in an organic manner rather than solely depending on pre-authored instances, and may heighten replayability and personalization.

By separating content creation from gameplay mechanics with a dedicated service, we find an effective architectural pattern for embedding resource-intensive AI operations in games. This approach maintains gameplay performance without sacrificing generative sophistication, suggesting a similar design might be extended to other AI-enhanced game features.

Our implementation demonstrates the necessity of robust validation systems where probabilistic systems such as LLMs interact with deterministic game mechanics. The success rates we saw (100% for most endpoints and 85% in encounters) demonstrate that well thought out validation and correction systems can bridge the gap between generative AI and gaming requirements.

7.2 Future Research Directions and possible applications

Future systems could use game player feedback and behavior analysis to modify generated content over time. As LLMs learn about player preferences and styles of play they could increasingly create more personalized experiences that change over the course of play rather than remain static after an initial generation.

While RoQ has been centered around a text-based generator there is an interesting line of future research around the ability to integrate multi-modal AI systems that generate visual assets, music and sound effects alongside the narrative content. These AI systems could alleviate additional development bottlenecks and lead to an even more consistent procedurally generated world.

Although our context management method is effective in maintaining narrative continuity across iterations, a more systematic memory model could accommodate long-term narrative arcs, character growth, or changes to the state of the world. Using retrieval-augmented generation techniques could be adjusted to provide LLMs more selective access to pertinent historical context.

Currently, our implementation also requires a significant computational investment. Future work in model shrinking, edge deployment, and server-side optimization will improve the possibilities for these techniques in game platforms with more significant resource constraints, such as mobile or web-based games.

Although RoQ illustrates the approach primarily in the context of the RPG genre, the techniques developed could easily extend to other genres of games:

- Adventure games could use dynamic dialogue (like our RPG) and adaptive puzzle generation.
- Strategy games could use similar techniques to develop and manage factions, diplomacy systems and mission generation.
- Simulation games could use LLM assisted PCG to create more diverse and realistic NPC behaviors and stories.

Educational games could use the system create personalized learning scenarios that adapt to student knowledge and interests.

7.3 Conclusions

The thesis developed and implemented a framework that solved the main challenge of probabilistic AI outputs being integrated into deterministic game systems. The use of a separately established backend service that handles LLMs interactions away from the game client response which avoids many of the performance challenges in AI integrated games. The system consistently generates coherent, interconnected game content across multiple domains (world structure, NPCs, quests, and encounters) while maintaining acceptable performance metrics for practical application.

The method of validating and correcting LLM outputs developed through the thesis represents a step forward in AI-assisted content creation. Through multiple layers of validations and intelligent recovery from errors, our system had an impressive success rate (100% for most endpoints and 85% for the more complex generation tasks). We believe it is reasonable to conclude LLMs can be a reliable provider of game content when properly bounded and followed within a structured framework for validation. The implementation of context management for NPC conversations addresses one of the most challenging aspects of narrative AI: keeping coherence and character consistency across several interactions. Our approach, based on

summarization, can give the appearance of memory and continuity despite the technical limitations of current language models. With conversation response times averaging 1.4 seconds, the system delivers a responsive player experience that feels natural and engaging.

Our results have several noteworthy implications for game development practices. To begin, the client-server architecture pattern outlined in this initiative provides a model for incorporating heavy AI processes into games while avoiding negative impacts on gameplay performance. This model can be extended beyond content generation, to different features enhanced by AI in games. Secondly, the hybrid procedure employing procedural and manual design elements further illustrates that these methods are complementary instead of alternatives. By determining which elements of a game benefit most from each method, developers can work more efficiently and more easily sustain creative production despite limited resources. Last, the success of runtime NPC conversations suggests that certain types of content can be generated during gameplay rather than exclusively during development or loading phases. This creates opportunities for more responsive and adaptive game experiences that evolve based on player actions.

The RoQ project demonstrates that LLMs can contribute to game development when integrated through carefully designed systems that bridge the gap between AI capabilities and game requirements. By separating content creation from gameplay mechanics and establishing strong validation processes, we've proven that even current language models can reliably generate coherent and interconnected game content. This hybrid strategy, which combines AI-generated narrative elements with traditional game systems, represents an exciting direction for the future of game development. Rather than replacing human creativity, these technologies enhance it, allowing developers to build richer and more diverse game worlds while honing in on the core systems and experiences that shape their games. As language models continue to evolve in capability and efficiency, the approaches we've laid out in this research provide a framework that can adapt alongside them, unlocking new opportunities for dynamic, responsive, and personalized gaming experiences.

Citations

[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. *Neural Information Processing Systems*, *33*, 1877–1901.https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *arXiv (Cornell University)*, *30*, 5998–6008. https://arxiv.org/pdf/1706.03762v5

[3] Riedl, M. O., & Young, R. M. (2010). Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research*, *39*, 217–268. https://doi.org/10.1613/jair.2989

[4] Kreminski, M., Dickinson, M., & Wardrip-Fruin, N. (2019). Felt: a simple story sifter. In *Lecture notes in computer science* (pp. 267–281). https://doi.org/10.1007/978-3-030-33894-7_27

[5] FastAPI. (n.d.). Retrieved March 13, 2025, from https://fastapi.tiangolo.com/

[6] API documentation & design tools for teams | Swagger. (n.d.). Retrieved March 13, 2025, from https://swagger.io/

[7] The Python Arcade Library — Python Arcade 3.0.2. (n.d.). Retrieved March 14, 2025, from https://api.arcade.academy/en/latest/

[8] Tiled. (n.d.). Tiled. Retrieved March 14, 2025, from https://www.mapeditor.org/

[9] Reynolds, L., & McDonell, K. (2021, May 8). Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. https://doi.org/10.1145/3411763.3451760

[10] Smith, G., & Whitehead, J. (2010, June 18). Analyzing the expressive range of a level generator. https://doi.org/10.1145/1814256.1814260

[11] Liapis, A., Yannakakis, G. N., & Togelius, J. (2014). Computational game creativity (pp. 46–53). http://antoniosliapis.com/papers/computational_game_creativity.pdf

[12] Claypool, M., Liu, S., Kuwahara, A., Scovell, J., Gregg, M., Galbiati, F., & Eroglu, E. (2024, May 21). Waiting to Play - Measuring Game Load Times and their Effects on Player Quality of Experience. https://doi.org/10.1145/3649921.3649937

[13] Ollama. (n.d.). Ollama. Retrieved March 24, 2025, from https://ollama.com/

[14] Google AI Gemma open models | Google for Developers | Google AI for Developers. (n.d.). Google AI for Developers. Retrieved March 24, 2025, from https://ai.google.dev/gemma

[15] WizzardLM. (n.d.). Retrieved March 24, 2025, from https://wizardlm.github.io/WizardLM2/

[16] Qwen2. (n.d.). Retrieved March 24, 2025, from https://huggingface.co/docs/transformers/model_doc/qwen2

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Μεταπτυχιακή Διπλωματική Εργασία με τίτλο:

«Leveraging Large Language Models for Dynamic NPC Interactions in 2D RPGs»

καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Πρόγραμμα Μεταπτυχιακών Σπουδών «Ανάπτυξη Ψηφιακών Παιχνιδιών και Πολυμεσικών Εφαρμογών» του Τμήματος Επικοινωνίας & Ψηφιακών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του Δρ. Μηνά Δασυγένη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright (C) Τοπαλίδης Ιάσονας & Μηνάς Δασυγένης, 2025, Αθήνα

Υπογραφή Φοιτητή
Τοπαλίδης Ιάσονας