MedTech
Mediterranean
Institute of Technology

SOFTWARE ENGINEERING PROGRAM

FINAL REPORT

User-Driven Procedural Generation of 3D Environments

BY
Aymen HAMMAMI

ACADEMIC SUPERVISOR
Dr. Asma AMDOUNI

COMPANY SUPERVISOR
Mr. Oussama BEN MARIEM
Lanterns Studios

Tunis, 2023-2024

Approval

APPROVED BY

ACADEMIC SUPERVISOR

Name	Signature	Date

COMPANY SUPERVISOR

Name	Signature	Date

ACADEMIC EVALUATOR

Name	Signature	Date

Declaration

I certify that I am the author of this project and that any assistance I received in its preparation is fully acknowledged and disclosed in this project.

I have also cited any source from which I used data, ideas, or words, either quoted or paraphrased. Further, this report meets all the rules of quotation and referencing in use at MedTech, as well as adheres to the fraud policies listed in the MedTech honor code.

No portion of the work referred to in this study has been submitted in support of an application for another degree or qualification to this or any other university or institution of learning.

I also certify that this final version of my capstone project report includes the corrections and comments mentioned by the jury members and it is submitted online on Moodle.

Student Name	Signature	Date

Work Term Release

I hereby state and verify by my signature that I have reviewed this report. I hereby affirm that the report contains

☐ no confidential data/information, and I authorize it to be released.

☐ confidential data/information, and I do not authorize it to be released.

COMPANY SUPERVISOR

Name	Signature	Date

Abstract

This study explores the field of procedural generation, focusing specifically on its application in creating 3D environments. The primary objective of the study is to develop a tool capable of procedurally generating 3D environments based on prompts provided by its user.

Throughout the study, the steps of the software development lifecycle of the tool are showcased in detail, from analysis and design to implementation and evaluation. The gathered findings of the study are then used to highlight the advantages and drawbacks of relying on procedural generation for 3D environment creation. The challenges of generating a 3D environment that aligns with user expectations are also discussed. In addition, the functionalities, and the requirements essential for facilitating environment creation and saving development time are identified.

The culmination of this study is a fully functional tool that fulfills all specified requirements. It has a positive impact on the industry, as it helps save development time, reduce costs and offers new opportunities. It also serves as the basis for a future monetizable version to be offered on online marketplaces.

Keywords: Procedural generation, 3D environments, Environment creation, Tool development, Software development lifecycle.

Dedications

First, I want to thank God for guiding me through this academic journey and making it possible for me to study at such a prestigious university. I am incredibly grateful and thankful for the ability to follow my passions both as a student and as a professional. I recognize this privilege in comparison to people who are suffering in other countries affected by war and poverty. That being said, my thoughts are with the people in Gaza and all of Palestine, who are enduring unimaginable hardships.

To my parents, I love you both very much. Your unwavering guidance and support have helped me through every stage of my life. Without you, none of this would have been possible. I owe everything I have to your love and sacrifices.

To my brother Wassim, I cannot imagine my life without you by my side. Your constant support and assistance without a second thought mean to me more than words can express. I wish you all the best in your career and life, and hope to be always there for you!

To Nour who is very dear to me, I am so thankful to have been able to meet you and grateful for all your support in both good times and bad. You have a special place in my heart.

To my dear friends: Aziz, Alaa, Crespy, Haithem, Ilyes, and Yassine, thank you for making every moment memorable. From being mates in real life to sharing our passion for gaming and music (Driving South, you know who you are). Your friendship has been a source of joy and strength.

I dedicate my work to everyone who believed in me, both who I mentioned and who I forgot to mention. Your faith in me has been a constant source of motivation.

Acknowledgements

I would like to express my utmost gratitude and appreciation to all those who have played a key role in helping me reach where I am now, and who have supported me during my academic journey and the entire development of this project.

First, I would like to deeply thank the host company, Lanterns Studios, whose hospitality has exceeded my expectations. Their support, resources and amazing work environment has contributed to the success of the project's development.

Next, I would like to extend my heartfelt thanks to my academic supervisor, Dr. Asma Amdouni. Without her excellent support and knowledge of software development, the achievement of this work, as it is, would not have been possible. I acknowledge her valuable insights and advice throughout this study, and I appreciate her guidance during the past three years across multiple projects, which shaped me to be the software engineer I am today.

Next, I would like to infinitely thank my company supervisor and technical lead, Mr. Oussama Ben Mariem, who made sure I had the best work environment I could ask for, and diligently provided me with access to resources that I needed, including training material and an access to a high-end computer. I am incredibly grateful for everything.

Finally, I would like to express my appreciation to the entire team of Lanterns Studios, especially the development team, and to my professors who supported me throughout my journey.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Definition
3D	Three-dimensional
AI	Artificial Intelligence
API	Application Programming Interface
ChatGPT	Chat Generative Pre-Trained Transformer
DFD	Data Flow Diagram
DSR	Design Science Research
GDLC	Game Development Life Cycle
LOD	Level Of Detail
NLP	Natural Language Processing
NPC	Non-player Characters
PCG	Procedural Content Generation
RNG	Random Number Generator
UI	User Interface
UMG	Unreal Motion Graphics
UML	Unified Modeling Language
UX	User Experience

1. INTRODUCTION

This chapter provides a broad overview of the conducted capstone internship and the respective project. It introduces the project's context, followed by the problems it tries to solve and the purpose of the study. The chapter then discusses related research questions, describes the approach and boundaries of the study, and lists my contributions to the project.

1.1. Project Context

The project was carried out during my end-of-study software engineering internship at Lanterns Studios, an independent video game company based in Tunisia.

The company offers mainly game development products and services, but it also provides extended reality services: implementing augmented, mixed, or virtual reality applications. It is also the first studio in Tunisia to provide a complete & high-quality motion capture solution [1]. Even though the gaming industry in Tunisia is still in its early stages, it already has a huge global impact and is well-established in international markets. In fact, the gaming sector's revenue has surpassed the revenue of the movie box office and music industries combined since 2008, as shown in figure 1. There is no indication that this development will slow down in the future; projections are indicating that the video games market is to reach a revenue of 282.30 billion US dollars in 2024 [2].

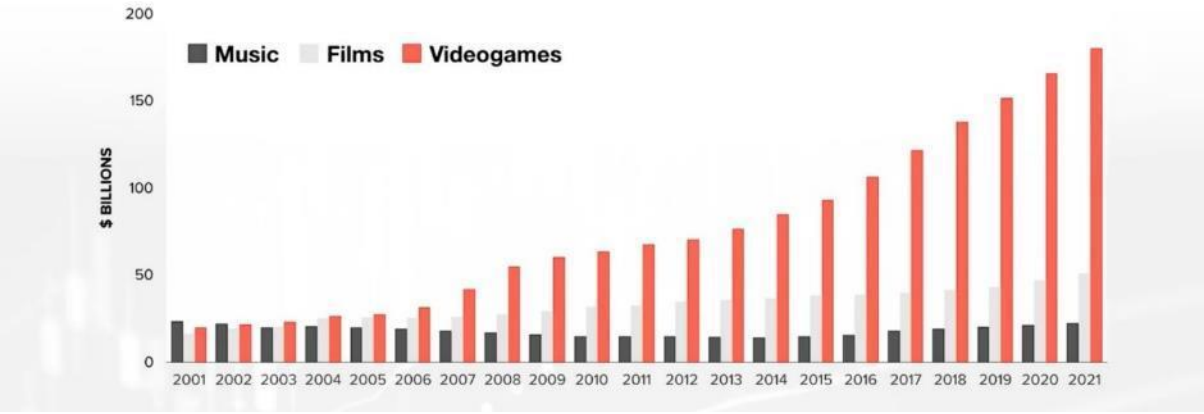

Figure 1: Revenues of the video games, films, and music industries [3]

Such growing demand encourages individuals and companies to create and release games, which in turn leads to an increased need for developments tools. This is emphasized further by the fact that most games take from several months to several years to be published, depending on their scope and budget.

The study consists of a tool, which given a user-provided prompt, such as "I would like a forest with big trees", is capable of procedurally generating the described 3D environment.

The tool will save multiple hours, if not multiple days of development time by providing artists, designers and developers with a 3D output that they can utilize as the basis for their work. The project would enable Lanterns Studios to accelerate its development life cycle, and would serve as the initial version of a tool intended for sale on the online marketplace.

1.2. Problem Statement

In many industries, especially the gaming sector, there is a big demand for creating immersive 3D environments efficiently. This raises two important problems.

The first problem is that putting together such an environment through the traditional way involves long manual processes and having expertise & skills with 3D tools and level design. Artists and designers rely on feedback to iterate over their creations, and this causes development time to expand even more. The larger the project's scope is, the more time-consuming these processes become.

The second problem is that financial limitations make it challenging for some smaller companies and teams to create large and appealing 3D environments. Hiring skilled artists can be expensive, and the smaller the team, the longer it takes to complete a project of such size. As a result, these companies have to decide whether to scrap/abandon the project or to reduce its scope to meet their budget constraints.

One way to solve these problems is by implementing a tool that makes the process of creating 3D environments more automatic, faster, and easier.

1.3. Purpose of the Study

The goal of the study is to create a tool that can procedurally generate 3D environments from scratch based on user-provided prompts.

The tool would speed up the process of developing 3D environments, which are an essential part of most video games and movies. This would help companies in these industries save a great deal of development time, and would allow them to focus more on polishing the final product.

Another benefit of the tool is that it would reduce the need for team members to have a specific degree or level of knowledge when it comes to creating environments, and would also cut down on some expenses. Many small-sized companies and indie developers would become able to pursue projects with larger scopes, while still sticking to the same budget limitations.

1.4. Research Questions

In the context of this project, the following research questions are proposed:

Research Question 1: What are the advantages and drawbacks of relying on procedural generation for 3D environment creation?

Research Question 2: What are the key challenges in procedurally generating 3D environments that align with user expectations?

Research Question 3: What features and requirements of the tool would be crucial to help streamline 3D environment creation and save development time?

Understanding the pros and cons of procedural generation when generating 3D environments by answering the first research question will provide insights on whether it is a suitable approach for the project. This will also help us improve the tool's overall design by implementing measures to avoid some of the drawbacks. By answering the second research question, the study will identify the problems associated with procedurally generating environments that meet user requirements. Handling those challenges is crucial to the success of the tool because its effectiveness and usefulness depends on its ability to produce content that aligns with user expectations. However, the real value of the tool lies in its ability to generate environments quickly. Time efficiency is paramount to the production of video games: if the tool can save development time, it would become quite valuable to developers. Answering the third research question will help us define the features and functionalities that offer the highest productivity possible.

1.5. Approach and Boundaries of the Study

This research aims to design and implement an artifact that procedurally generates 3D environments based on user-provided prompts. The development of the artifact will be over some iterations, as it is necessary to refine the tool depending on feedback and to adapt to evolving requirements. As such, the Design Science Research methodology is a suitable approach for this research.

1.5.1 Scope

In this study, the objective is to create a tool that makes generating immersive 3D environments more automatic, faster, and easier, especially for companies in the video games industry and independent developers.

The tool would include functionalities that generate these environments based on user-provided prompts, and allow for updating the generated content either by exploring other variations of the same type or by iterating over the results using feedback.

The tool will focus on generating environments composed of multiple layers, such as foliage, rivers, and other environmental elements. Already implemented 3D models will be used to assemble such environments.

The study will not focus on the development of generative artificial intelligence (AI). Instead, the tool will generate 3D environments using procedural techniques. The integration of an already developed AI model will help interpret user-provided prompts.

1.5.2 Limitations

One primary limitation of the tool is that the available technologies and programming languages are constrained by the real-time 3D engine used by the company. Given that the tool should function within that specific engine, called Unreal Engine. Another limitation is the company's requirement for utilizing procedural content generation features provided by Unreal Engine. The most recent version of this engine is still relatively new, and some of its features are still under development, so the available online resources about them are limited. The novelty of these features also increases the possibility of encountering bugs and problems. One more limitation to note is that feature implementation will be prioritized based on budget and time constraints provided by the company.

1.5.3 Delimitations

The primary target audience of the project are Lanterns Studios, small to mid-size development companies in the video game industry, and independent game developers. Although the project might be useful to other applications outside of these sectors, its development and evaluation will prioritize meeting the needs of that specific industry. One more delimitation is that the tool will only be able to generate 3D environments with certain predefined artistic styles and themes.

1.6. Contributions

By the end of this project, the final product is a fully functional tool that is capable of generating 3D environments based on prompts provided by the user. It offers more functionalities that allow the adjustment of the generated environments, and ensures that the users can employ, reuse, and extend the created environments. The tool is integrated successfully within Unreal Engine, the 3D real-time engine the company has selected. That being said, the developed tool during this study is ready to be used as the basis for a monetizable version of the tool in the online marketplace.

1.7. Structure

This research report is composed of many sections. The upcoming section 2 describes the background of the study and presents a literature review. Afterwards, the adopted methodology and its aspects are discussed in section 3. Section 4 describes the performed iterations during the development of the project, following the steps set by the methodology. The gathered findings are then used to answer the proposed research questions in detail in section 5. Finally, section 6 presents a discussion about the study's impacts and potential validity threads, followed by general conclusions.

1.8. Conclusion

In this section, we discussed many aspects that formulate the project's context, purpose, boundaries and proposed research questions. In the next section, we will introduce background elements that provide the reader with a better understanding of the project, followed by a literature review on the existing related studies.

2. BACKGROUND AND LITERATURE REVIEW

This section provides background context related to the different aspects of the research, such as procedural generation. It also covers a literature review that presents an understanding of the existing related studies.

2.1. Background

Multiple background elements are presented to provide a better understanding of the project. The section starts by covering the game development life cycle, which describes the steps of creating a game from conception to deployment. Following this, this section introduces the environment design process. This section also discusses tools programming and its importance in development tasks, then discusses real-time 3D engines, namely Unreal Engine. The section also defines procedural generation and explains how this technique can dynamically and randomly generate content. Finally, the section gives an overview of the field of natural language processing, particularly ChatGPT and its integration with other tools.

2.1.1 Game Development Life Cycle

The game development life cycle (GDLC) is composed of multiple phases that are necessary for creating a game. Compared to the typical software development life cycle, this approach aims to tackle most challenges that developers face when making video games. Figure 2 describes the seven fundamental stages of GDLC [4].

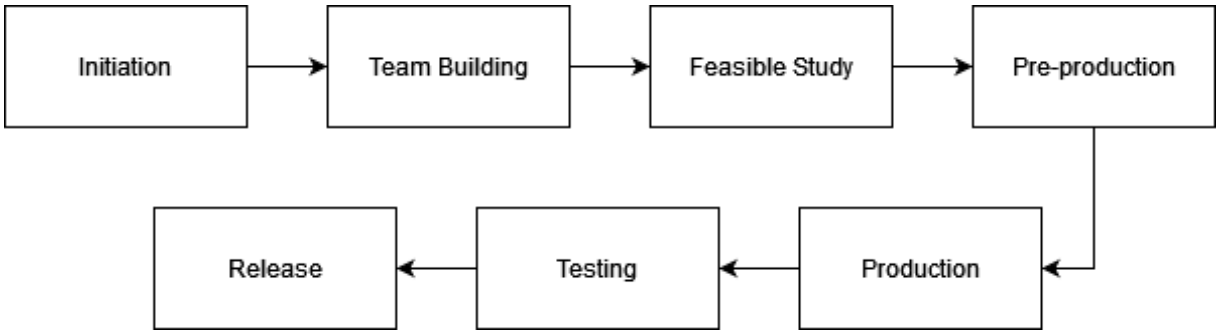

Figure 2: The game development life cycle [4]

- **Initiation:** In this stage, developers define the scope of the game and other considerations of the project, such as its type and the target audience.

- **Team Building:** Based on the outcome of the previous stage, managers put together the best team for developing the game project depending on its requirements. This step includes hiring, training, and assigning roles to team members.

- **Feasible Study:** In this stage, the most significant business and technical challenges are identified. These will be used to analyze the game's viability.

- **Pre-production:** In this stage, the team proceeds to develop a prototype of the game. This stage involves lots of documentation. Writers create a well-defined story while designers build the game mechanics and levels according to the storyline. As soon as the design is ready, artists put together the concept art of the project, and imagine the overall theme of the environment and characters. Programmers pick the right game engine depending on the requirements and start working on the prototype.

- **Production:** In this stage, the team builds upon the developed prototype. This is a long process that involves further programming, improving design, modeling, texturing, and creating animations.

- **Testing:** Once the game is in a playable state, the team runs two major testing sessions: alpha and beta testing. During the first one, the game is incomplete, however, the team tries to fix bugs and errors as they keep developing the project. The second one typically involves customers as public testers. During this one, the game is complete, and the team concentrates on bug fixing, polishing, and balancing depending on feedback from the players.

- **Release:** In this stage, the project is ready to be published. The game is released on online platforms and marketplaces, such as Steam.

In the context of this project, the research aims to produce a tool that helps team save development time during the pre-production and production stages. Being able to generate an immersive 3D environment quickly will immensely help artists and designers produce the best version of their ideas.

2.1.2 Environment Design Process

The environment design process is an intricate part of game and movie development, and is a key factor in deciding the commercial and marketing success of the project. Artists or members responsible for creating the environment start by conceptualization and planning.

They determine multiple aspects such as its role in the scene, its theme, its setting, and its atmosphere. Once done, the team moves on to model and texture the elements of the environment. Finally, the team implements dynamic lighting and atmospheric effects, which complement the environment [5]. It is important to understand thoroughly the environment design process, as the project needs to produce an immersive environment that satisfies all the usual steps of the process.

2.1.3 Tools Programming

The goal of tools programming is to create useful software tools, whether internal or external, that help artists, designers and developers produce and implement ideas quickly and effortlessly [6]. From a business perspective, investing time and resources into developing these tools is worth it, as they significantly improve productivity & consistency, reduce the risk of bugs, and speed up project release. Developed tools are usually integrated into the main technologies that the companies use, but can be sometimes found as standalone third-party applications.

In the context of this research, the goal is to develop a tool that streamlines environment creation and integrate it into the real-time 3D engine that the company uses, called Unreal Engine.

2.1.4 Real-time 3D Engines and Unreal Engine

Real-time 3D engines provide the technological foundation for rendering computer graphics, simulating physics, running scripts, and creating interactive experiences. Real-time engines are scalable, fast & efficient and allow for the integration of tools & further customization of the engine [7]. Unreal Engine is one of the most advanced real-time 3D engines and is trusted by multiple industries, such as video games, movies, automotive, live broadcast, and simulation. The company, Lanterns Studios, uses Unreal Engine as its main creation tool. It offers multiple features catering to the workflow of all members of a development team, and is highly expandable [8]. Procedural content generation is a feature made possible by a new toolset offered by the engine's most recent version [9]. Unreal Engine supports programming in C++, but also offers a powerful built-in visual scripting language called "Blueprint" [10].

In the context of this project, the goal is to develop and integrate a tool into Unreal Engine, and take advantage of its powerful features to generate 3D environments efficiently.

2.1.5 Procedural Content Generation

Creating credible 3D worlds, composed of a realistic environment, atmosphere, lighting, characters, and a vast number of diverse objects, is a complex and long task. Development teams face challenges as they implement these worlds, such as creating an innovative and non-repetitive environment while respecting deadlines and keeping the project within budget. One method that developers use to address such an issue is procedural content generation (PCG). PCG is a technique capable of creating multiple elements: landscapes, environments, levels, characters... It is based on algorithms that use deterministic tasks and randomness, so it is possible to generate a countless of variations based on the same rules and parameters [11]. Let us imagine we are generating a maze like the one shown in figure 3 in a procedural way.

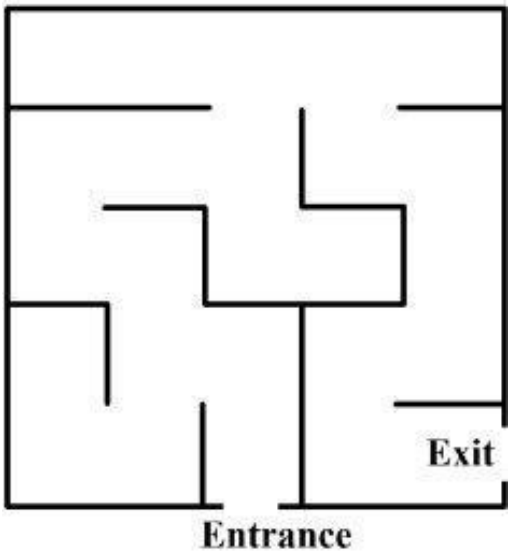

Figure 3: Example of a maze with an entrance and an exit [12]

In this case, deterministic tasks could be picking an entrance & an exit point, choosing the maze's width & height, and setting up a set of rules, such as there must be at least one valid path that starts from the entrance and ends at the exit. When it comes to the random decisions to take, they could be choosing where to put walls, and whether a given path should be open or blocked.

In the context of this research, the project relies on procedural generation techniques to generate immersive 3D environments.

2.1.6 Natural Language Processing and ChatGPT

Natural Language Processing (NLP) is a domain that intersects artificial intelligence and linguistics. NLP enables the interpretation and generation of words and sentences of different languages. It is also able to transform a certain sentence into a different one, based on some rules, such as translation or rewording. NLP has a wide range of applications and is a being developed over several years [13]. However, in the case of this research, we are more interested in the interpretation of instructions and prompts.

Currently, one of the most sophisticated implementations of generative NLP is ChatGPT. It is an AI model designed especially for having conversations and understanding instructions [14]. This is made possible by processing texts using tokens, which are sequences of characters generated from the latter. The organization behind its implementation, called OpenAI, offers an application programming interface (API) that makes it possible to integrate the ChatGPT model in any technology-based project, and monetizes the usage of the API based on how many tokens the developer uses [15]. In the context of this project, ChatGPT could be used to interpret the user-provided prompts, and transform them into parameters used for procedural generation of 3D environments.

2.2. Existing Related Studies

2.2.1 History and usage of procedural content generation in game development

Gillian Smith [16] published a research that presents the early history of procedural content generation. It describes the design aspects of this technique, the motivations behind its implementation, and its usage in both non-digital and digital games. Before discussing these elements, it is important to understand what PCG really is. Gillian defines this technique as the usage of algorithms to generate content, instead of relying on manual design and development processes. Another thesis by Adeel Zafar, Hasan Mujtaba, and Omer Beg [17], defines PCG as the technique of producing content automatically through random, yet controlled, processes.

Indeed, randomness plays an important role in how PCG works. As described by Gillian [16], most procedural generators would combine a set of deterministic tasks with some kind of random decision-making. By directing and controlling this randomness, developers can generate interesting and variable content that is sufficient to use in a game [16].

Around 1980, the first games adopting PCG started appearing, most notably the well-known game "Rogue", which sends the player on a mission to explore challenging dungeons.

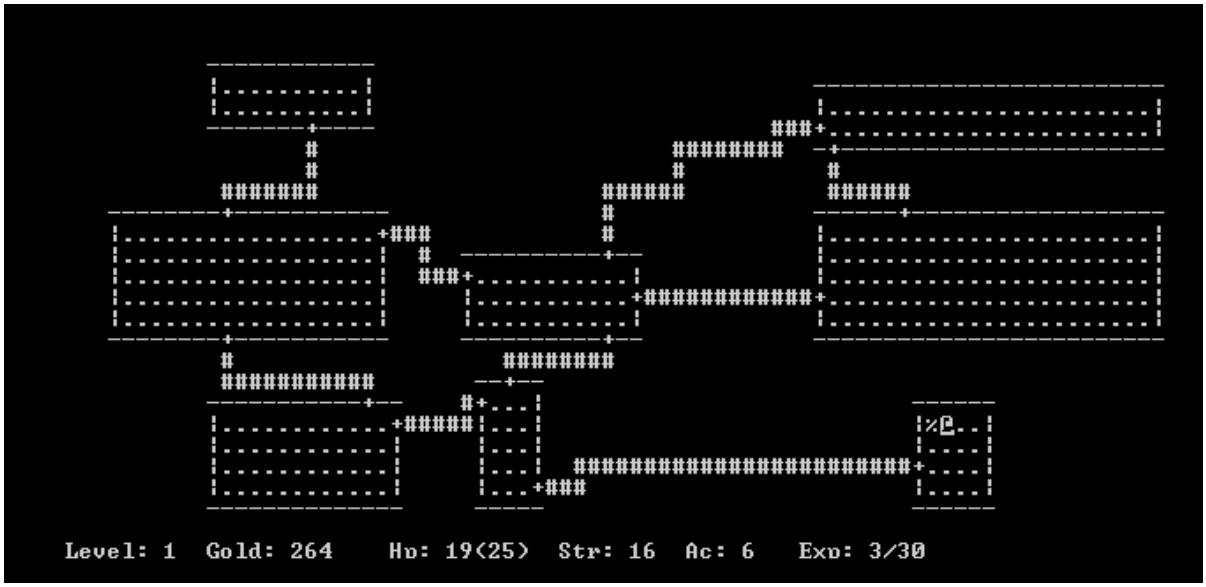

Figure 4: A procedurally generated dungeon in "Rogue" [18]

The research conducted by Gillian Smith [16] summarizes the initial motivation of these developers who adopted PCG in their games:

- **Replay Value:** Also called replayability, refers to the aspects of a game that motivate players to play it multiple times [19]. PCG excels in this regard, by creating different gameplay experiences thanks to its pseudo-randomness.

- **Assisting Creativity:** Help designers, artists and even players avoid creative blocks, by creating interesting and surprising content.

- **Unbiased Decisions:** Eliminate human bias, error and hesitation when generating content.

- **Replacing Designers:** Offload the creative work behind creating levels and campaigns to algorithms and functions executed by a machine.

Martin Dahrén [20] has explored the growing popularity of PCG as video games have increased in size, budget, and scope in recent years. With a vast amount of content to create, the use of PCG has become very appealing to developers, especially since time efficiency is crucial in game development. The research presents the most popular applications of PCG in the modern gaming industry:

- **Terrain Generation:** Refers to producing virtual landscapes such as grounds, hills, caves… that make up the game world.

- **City, Building and Road Generation:** Refers to the creation of structures such as buildings, roads, and city infrastructure within games. Video games that require urban environments would typically feature these elements.

- **Level Generation:** Refers to generating the spaces and areas that the player can traverse or explore. Levels are a fundamental component across all game genres.

- **Story and Narrative Generation:** Refers to creating different game stories to improve replayability and narrative elements.

- **Puzzle Generation:** Refers to game elements that require problem solving and observational skills. These challenges are procedurally generated based on a set of rules.

- **Dynamic Generation:** Refers to creating and configuring dynamic systems, such as game rules, weather, enemy AI, and other behaviors.

Jonas Freiknecht [11] discusses the adoption of PCG in modern game development tools. These tools allow developers to focus on creating believable and varied content rather than dealing with the technical challenges of implementing the lower layers of PCG themselves. Examples include tools offered by real-time game engines such as Unreal Engine, and modern 3D editors.

2.2.2 Impact of tools on the game development process and their requirements

The research conducted by Björn Detterfelt and Samuel Blomqvist [21] introduces the motivation behind integrating tools in the game development process. They discuss how game creation is composed of demanding tasks. Polishing details and striving to produce high quality games while respecting deadlines is challenging. This is why it has become crucial for companies to save development time by eliminating the repetitive implementation of certain features and reducing the number of tedious tasks. Developing and using tools is beneficial as they allow features to be reused across multiple projects. They also explain how tools help simplify the complexity of video game projects by modularizing features.

Transmissia Semiawan, Muhammad Riza Alifi, Hashri Hayati and Djoko Cahyo Utomo Lieharyani conducted an experiment [22], whose goal was comparing the results of application development with and without relying on software development tools. To observe the effectiveness of such tools, they used the following evaluation metrics:

- **Elapsed time:** refers to the total time it took to develop the application.

- **Completeness:** refers to the percentage of requirements completed out of the total number of requirements.

- **Quality:** refers to how well the application adheres to business rules and technical requirements.

- **Repetitiveness:** refers to the implementation of the same features or components multiple times across the application.

- **Usability:** refers to producing output that correctly fits user expectations and needs.

The overall results of the experiment show that the adoption of tools improves the effectiveness, efficiency, and completeness of the development process.

An analysis conducted by Marcos Almeida and Flávio da Silva [23] lists the most important requirements that a tool should have to allow game designers to efficiently create and manage interactive content. According to them, a valuable tool should be usable across a wide range of projects, regardless of their main topic. It should also provide methods that enhance game design and boost creativity. It should be open to expansion and modification, facilitate collaboration, and include a mechanism to prevent unwanted or high-risk errors.

In addition, Björn Detterfelt and Samuel Blomqvist [21] emphasized the importance of usability of development tools and broke it down into non-functional requirements that focus on user experience and optimization. These requirements include:

- **Learnability:** refers to the ability of the user to learn and remember the tool's functionalities.

- **Efficiency:** refers to the usefulness of the tool's output relative to the effort spent.

- **Operability:** refers to the tool's effectiveness and reliability in executing tasks.

- **Time Complexity:** refers to how long it takes the tool to complete executed tasks.

- **User-friendly Interfaces:** refers to the ease with which users can navigate and interact with the tool's interfaces.

2.2.3 Integration of artificial intelligence in game development

The research paper by Aleksandar Filipović [24] discusses the evolution of artificial intelligence usage in the gaming industry. Prior to the implementation of neural networks, developers depended on algorithms and design patterns to create gameplay elements that were believable yet clearly scripted. With the growing popularity of deep learning and neural network-based models in the recent years, developers are now able to mimic human player behavior and create gameplay elements that are both realistic and unpredictable, thanks to the capabilities of advanced AI models.

Boming Xia and Xiaozhen Ye [25] explore the various fields of game development that started integrating AI in their systems, whether through tools or as a part of the game itself.

Some of these applications include:

- **Non-player Characters (NPC):** NPCs are virtual characters that are controlled by the game's AI rather than human players. Integration of AI models that have conversational skills can make these characters exhibit human-like behaviors and dialogue.

- **Procedural Content Generation:** As previously mentioned, PCG refers to the use of algorithms to automatically generate game content such as levels, maps, and puzzles. AI can manipulate these algorithms and its parameters to produce unique game elements that can adapt to the player's performance.

- **AI-assisted Game Design:** Integration of AI can offer new ways to game designers to create and experiment with ideas, therefore boosting their creativity. AI is also capable of learning and receiving feedback from the designers, which would improve its predictions.

- **Player Modeling:** Based on the player's data and the game's state, AI can understand and predict player behaviors, preferences, feelings, and skills. These metrics help create personalized and customizable gaming experiences.

According to their research, the integration of AI in the game development process encourages relying on hybrid intelligence. Hybrid intelligence refers to combining human intelligence with AI to tackle challenging and complex tasks [26]. This approach allows developers to achieve results that are superior to what humans or AI could accomplish individually.

Another research by Stefan Seidel, Nicholas Berente, Aron Lindberg, Kalle Lyytinen, Benoit Martinez, and Jeffrey V. Nickerson [27] discusses how integrating AI can augment the current design and development tools by making them more autonomous, which in turn facilitates the work of game designers by reducing the number of variables they must manage and understand. By automating repetitive and low-level decision-making processes, AI enables designers to focus more on creative and strategic aspects of game development. This helps streamline the game development workflow and designers to produce higher-quality content with less effort.

2.3. Conclusion

In this section, we discussed various background elements that provide a better understanding of the fields and elements used in this project, as well as a related literature review. In the next section, we will define the adopted methodology of the study, its guidelines and iterations.

3. METHODOLOGY

The choice of methodology could determine whether the outcome of a research project is successful or falls short of expectations. Not only does a correctly selected methodology provide researchers with a consistent set of guidelines, but it also increases the reader's trust in the reported findings by making the research process more credible. [28] The Design Science Research methodology was adopted in the context of this research.

3.1. Design Science Research Methodology

3.1.1 Definition

Design science involves studying and designing artifacts in a given context. This context represents a set of fixed concepts, such as rules and restrictions, related to a real-world problem. The goal behind building the artifacts is to interact with the problematic situation within that context and to provide solutions to it, which results in better processes and systems [29]. By thoroughly researching and understanding the problem space, we can create optimized solutions and achieve the desired outcomes.

3.1.2 Guidelines

According to R. J. Wieringa [29], there are seven guidelines to follow when applying the Design Science Research methodology:

- **Design an artifact:** the output of the research must be a valid artifact, such as a construct, a model, or an instantiation.

- **Problem relevance:** the research aims to produce a solution relevant to business problems.

- **Design evaluation:** relevant evaluation methods must be conducted thoroughly to ensure that the design is efficient and of good quality.

- **Research contributions:** provide clear and verifiable contributions derived from the research process.

- **Research rigor:** the research must employ rigorous methods over the construction and evaluation of the design artifact.

- **Design a search process:** the research must utilize available resources and means to search for an effective artifact.

- **Communication of research:** the research must be presented in an effective way to management-oriented and technology-oriented audiences.

3.2. Design Science Research Iterations

Design Science Research is based on an iterative approach, which starts with investigating a problem within the context, followed by designing a relevant solution. Over several iterations, a Design Science Research (DSR) project aims to establish connections between the problem and solution spaces [30]. Figure 5 demonstrates the five steps of the regulative cycle of DSR in more detail.

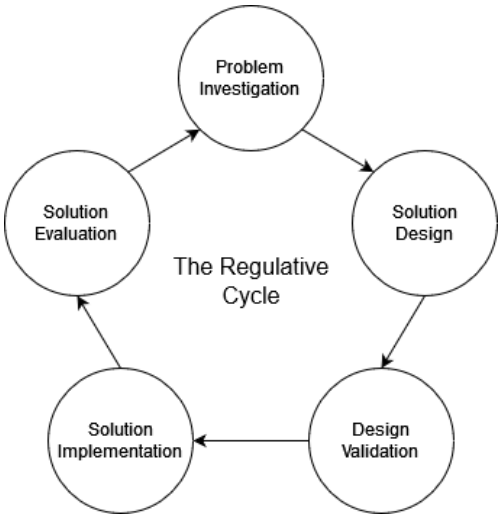

Figure 5: The regulative cycle [29]

- **Problem Investigation:** In this phase, the main objective of the research is to identify the problem that we are trying to solve. By properly understanding the problem space and defining the expectations and requirements of the stakeholders involved, the research would be able to produce a successful outcome. One more step of problem investigation is to propose research questions, which outline the later stages of the research.

- **Solution Design:** In this phase, the research aims to design a viable artifact that solves the problem identified in the earlier step. Designing such a solution is carried out over

several iterations that involve evaluating and polishing the design until it meets the identified expectations and requirements.

- **Design Validation:** In this phase, the goal is to confirm if the created design is suitable for solving the identified problem, and consistent with the established design principles and standards.

- **Solution Implementation:** In this phase, the focus is on bringing the designed solution to life. It involves technical aspects of implementing and developing the solution, which include programming, adopting appropriate technologies, building, and integrating the project.

- **Solution Evaluation:** In this phase, the focus is on assessing how well the implemented solution works. The solution can be validated through many means, such as conducting empirical studies and carrying out experiments.

3.3. Conclusion

In this section, we discussed Design Science Research and its adoption during this study. In the next section, we will discuss the performed iterations of this project, which are based on the regulative cycle shown in figure 5.

4. ITERATIONS

In this section, we will discuss the performed iterations during the development of this project. For each iteration, we will present every executed step of its regulative cycle. During this project, 3 different iterations were performed. We will start by the implementation of the procedural content generation (PCG) system, which is responsible for the actual generation of 3D environments. Afterwards, we will discuss the steps behind the prompt interpretation process, and its integration with the aforementioned PCG system. Finally, we will discuss the implementation of the tool's user interface (UI) and the design of its user experience (UX).

4.1. Iteration 1: Procedural Content Generation System Implementation

The first iteration's objective is to model and implement a robust procedural content generation system. It is a core part of the tool and will be used by it internally in order to generate 3D environments based on adjustable parameters.

4.1.1 Problem Investigation

In order to generate 3D environments based on user specifications, we first need to model and implement an internal system responsible of performing such tasks. Without this system, the tool itself would not be valuable, as it would not be capable of creating environments. During the implementation phase, several issues would arise if not enough research were done beforehand about the system's modeling, design, and the techniques of PCG. It is a complex subject with a wide range of applications, so it is especially important to have a clear understanding of how PCG works and to explore the associated techniques that can be applied using Unreal Engine, the real-time 3D engine used by Lanterns Studios. Furthermore, in order to minimize the likelihood of making mistakes that could lead to rewriting entire modules of the system's code, and to eliminate the risk of scope creep, we must have a clear vision of the system's requirements and the types of environments and elements it intends to generate.

4.1.2 Solution Design

We will start the process of solution design by determining the environmental aspects that the PCG system should be able to generate. Since environments are vast and contain many natural layers, it is crucial to identify the specific parts of them that are of interest to us.

Table 1 illustrates the environmental aspects that we should focus on during the development of the system.

Environmental Aspects	Description
Foliage	Refers to vegetation and natural resources such as trees, plants, rocks…
River	Refers to flowing water moving from one point to another.
Buildings	Refers to structures such as houses, flats, cabins…

Table 1: Environmental aspects of the PCG system

It is also important to establish which environmental themes should be supported by the PCG system, since these themes will guide the selection and preparation of 3D models to be used. Table 2 shows the environmental themes that we decided to consider.

Environmental Theme	Description
Broadleaf Forest	Covered by many trees with broad leaves and is found in temperate regions. The water is clear, and buildings are usually made of wood.
Northern Forest	Covered by long trees like pine and is found in cold regions. The water is cold and moves slowly, and buildings are usually made of logs.
Arctic	It is a cold and icy region, with low plants and some trees covered by snow. The water's surface is frozen, and buildings are usually made of stone or hard snow.
Desert	Covered by sand and has different types of vegetation that can live in such an atmosphere. The water is scarce/shallow, and buildings are made using heat-resistant materials.
Japanese	It is characterized by beautiful gardens and unique plant life such as cherry blossoms. The water is calm and clean, and building showcase the traditional wooden architecture.
Canyon	Covered by many rocks and geological formations and has plants that live in arid conditions. The water is fast-moving, and buildings are usually made of heat-resistant materials.
Fantasy	It represents an imaginary/magical environment, similar to what you would see in an alien world or a Disney movie.

Table 2: Environmental themes of the PCG system

Now that we have decided on the environmental aspects and themes, we can draw the functional requirements needed for the implementation of the PCG system. Table 3 outlines each functional requirement along with its description.

Functional Requirement	Description
Support different themes and 3D models	The system must be able to generate environments of different themes and 3D models.
Generate diverse foliage	The system must be able to generate many kinds of vegetations and natural resources.
Generate river	The system must be able to generate a river with flowing water that respects the chosen theme.
Generate buildings	The system must be able to generate buildings or similar structures.
Manage environmental elements	The system must be able to update, remove or add environmental elements to the generated content.
Control parameters of environmental elements	The system must be able to control the scale, position, rotation, and density of elements in the environment.
Generate elements near the river	The system must be aware of the river's position and able to generate elements near it.
Control river's width	The system must be able to control how wide the river is.
Place river in many ways	The system must be able to create a river at different positions in the environment.
Place buildings in many ways	The system must be able to generate buildings at different positions in the environment.
Generate a path between buildings	The system must be able to generate a path between buildings that indicates how to travel from one to another.

Table 3: Functional requirements of the PCG system

Next, we have to identify the quality attributes that the PCG system must meet to ensure that it will work as expected from the user's perspective. These are explained in table 4.

Quality Attribute	Description
Performance	The system should execute tasks efficiently and ensure quick generation of environments.

Scalability	The system should be expandable with additional themes and 3D models without affecting performance or requiring lots of coding changes.
Coherence	The system should generate a believable environment by placing its elements in a logical way and by preventing visual mistakes.

Table 4: Quality attributes of the PCG system

Finally, it is critical to have a clear picture of the PCG techniques that could be useful for the generation of environments which align with the system's requirements. The most notable ones that are available in Unreal Engine are displayed in table 5:

PCG Techniques in Unreal Engine	**Description**
Random Seed	Used to ensure reproducibility of the same random values and decisions, which allows generating the same variation of an environment.
Random Number Generator	Is used to create random values that can be used for generating varied elements of an environment. RNG can also be used to make random decision making.
Surface Sampling	Helps place elements accurately by sampling points on surfaces such as landscapes, taking its dimensions and shape into consideration.
Spline Sampling	Helps generate rivers by sampling points along a spline.
Poisson Disk Sampling	Helps place elements with even spacing and avoids overcrowding.
Density Noise and Filter	Helps create random yet controllable placements of elements.
Difference Operation	Helps prevent cases where different elements would overlap with each other.
Self-Pruning	Helps prevent cases where the same elements would overlap with each other.
Transform Modifier	Helps adjust position, rotation, and scale of generated elements.

Table 5: PCG techniques that can be used for generating 3D environments in Unreal Engine

Based on the analysis of the system's requirements, quality attributes, environmental categories, and themes, we determine the system's workflow that best achieves our goal. An activity diagram was created to explain the steps and the flow that should happen within the PCG system when generating a 3D environment procedurally.

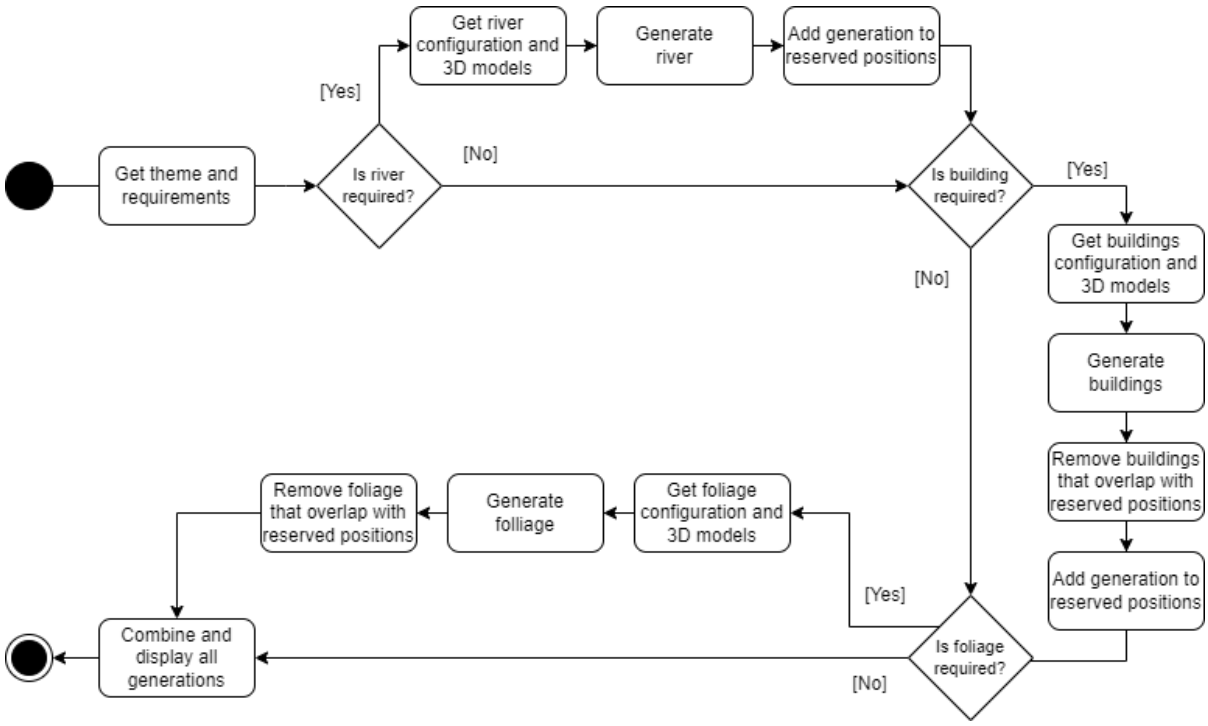

Figure 6: Activity diagram for environment generation

As shown in figure 6, the system's workflow is described step by step:

1. It starts by retrieving necessary parameters such as the chosen theme, categories, and requirements for the environment we would like to generate.

2. Based on these, the system evaluates whether generating a river, foliage and buildings are needed.

3. For each required category, the system fetches the necessary configurations and 3D models, and uses them to generate the associated content procedurally. It also makes sure that generated elements do not overlap by reserving positions if necessary and checking for conflicts or collisions.

4. Finally, all generated elements are combined and displayed as the complete 3D environment.

Considering the designed workflow, we aim to develop and use a set of classes that implement and execute these steps. The class diagram shown in figure 7 demonstrates the structure of the system and how each component interacts with each other to efficiently generate a 3D environment procedurally.

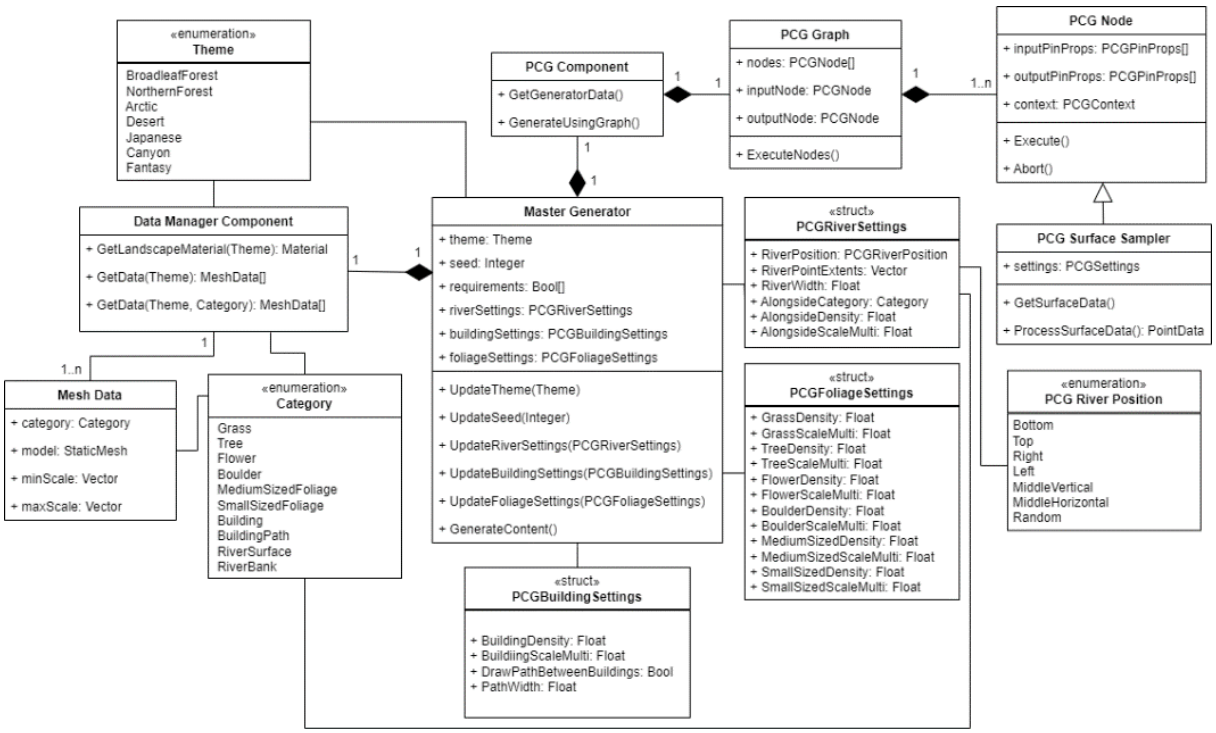

Figure 7: Class diagram of the PCG system

- **Master Generator:** is the central class and responsible for controlling and updating themes, categories, requirements, randomness, and generated elements of the 3D environment.

- **Data Manager Component:** is part of the Master Generator class and used to retrieve the necessary 3D models that are going to be placed, along with their information.

- **Mesh Data:** represents an asset and its parameters, such as its 3D model and category.

- **PCG Node:** represents an individual step/operation and executes custom code or procedural techniques.

- **PCG Surface Sampler:** is used to get information about the landscape or surface we are trying to generate an environment on top of. This data allows placing elements accurately.

- **PCG Graph:** is used to link PCG nodes together to form a graph, which defines their order and logic of execution.

- **PCG Component:** is part of the Master Generator class and uses a PCG Graph to understand tasks and execute the procedural environment generation process.

"**PCGPinProps**", "**PCGContext**", "**PCGSettings**", "**PointData**", "**Material**" and "**StaticMesh**" are all types that are already available in the development environment of the project.

4.1.3 Design Validation

The process of design validation was carried out over meetings with the stakeholders most of the time. Also, a brief presentation was given to the technical lead discussing the system's goal, requirements, and vision. The stakeholders agreed that the PCG system had to be implemented first due to its importance to the tool's workflow, and they approved that the functional requirements and quality attributes were sufficient for the project. Hence, the solution design was indeed validated.

4.1.4 Solution Implementation

Based on each proposed quality attribute, a quality scenario was created to better understand the desired behavior of the system:

- **Performance:**

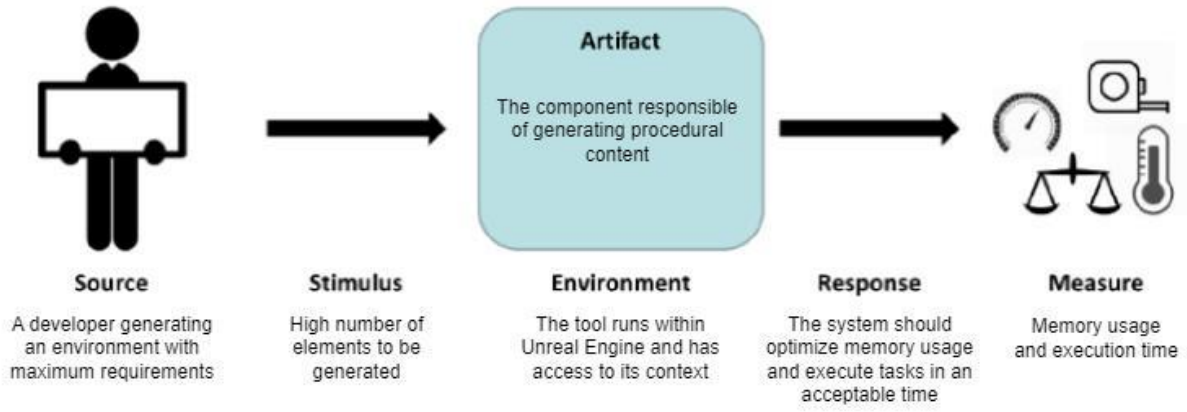

Figure 8: Performance quality scenario

To ensure the system meets the necessary performance criteria, we employed multiple optimization strategies during its implementation, such as:

- **Level Of Detail (LOD) Management:** The fundamental concept behind LOD is to create multiple versions of the same 3D model, with different levels of detail. Depending on its size on the screen, the version with the optimal level of detail is rendered. Typically, when the model is far away, a low-detail version is rendered, and when it is close, a high-detail version is rendered [31].

- **Branch Execution Culling:** By implementing conditions and requirements for when to execute a part of the code, we avoid unnecessary processing and computations [32].

- **Instancing 3D Models:** Instead of loading the model's base geometry and materials separately each time the model is placed in the environment, they are loaded only once in memory. Multiple instances of this model then share and reference this same data when rendering, which significantly reduces memory usage. This technique is inspired by the flyweight design pattern [33].

- **Scalability:**

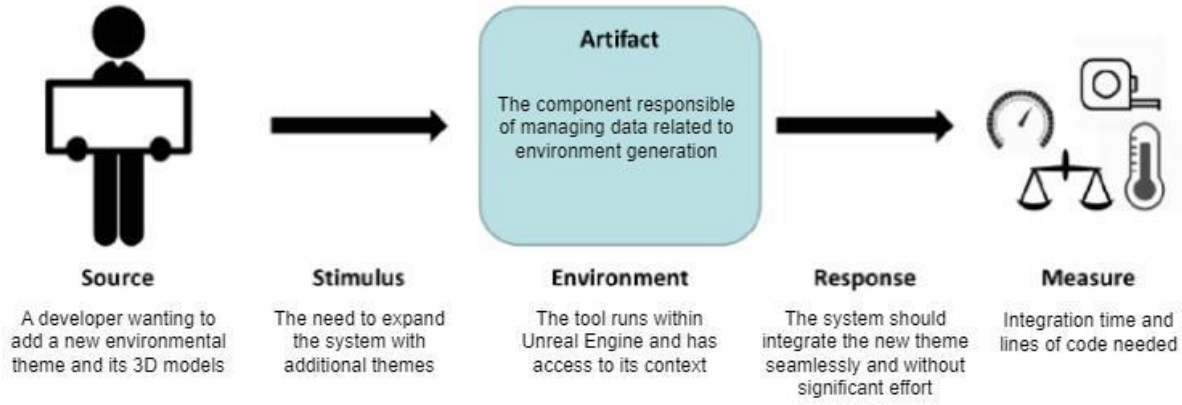

Figure 9: Scalability quality scenario

To ensure the system meets the scalability requirements, we tried to keep the code as modular and extensible as possible during its implementation. We also completely separated the data storage for each environmental theme from the main functionality of the code, by storing the entire configuration in a data asset that can be adjusted at any time. This way, to extend the current system with a new theme, a developer would only need to add a new entry in the theme enumeration, and fill its information in the data asset. It is also possible to modify existing

theme by changing the information in the data asset. This implementation helped efficiently put together all the environmental themes mentioned in the solution design section without the need for extensive code changes.

- **Coherence:**

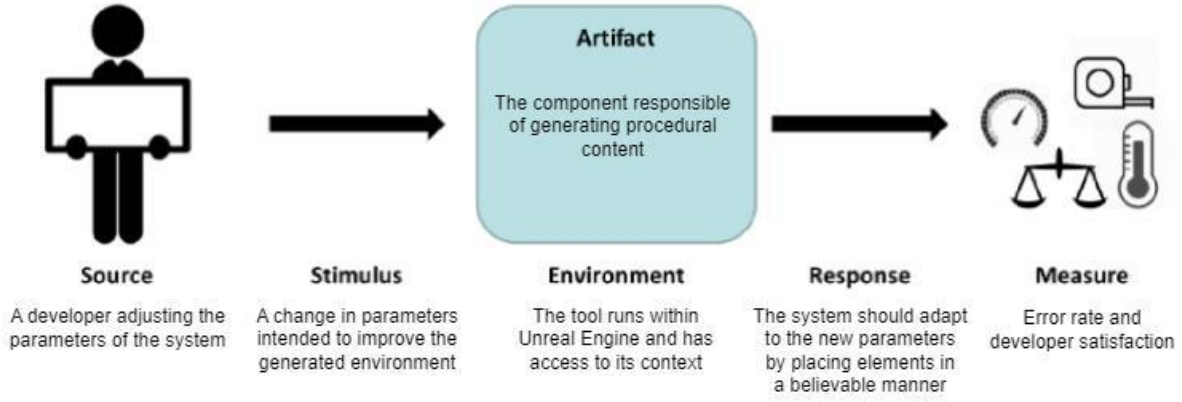

Figure 10: Coherence quality scenario

To ensure the system achieves and maintains coherence, we implemented some important steps in the process of procedurally generating environments, including:

- **Reserving Generated Positions:** After generating an environmental category, such as river, buildings or foliage, we reserve all of its respective elements' positions and we don't allow another category to generate elements on the same positions.

- **Removing Overlapping Elements**: We eliminate any elements that intersect or overlap in an unrealistic manner, such as a tree growing from within a boulder.

Based on the design of the system's workflow and classes discussed in the solution design section, we implemented multiple components and objects that seamlessly work together to achieve the system's desired functionalities.

One of the necessary components is the "Data Manager Component", which provides our system with information about the environmental themes, and the models it should use. Without this data, it would be impossible for the system to decide which elements to generate. This data was stored in an asset that is adjustable at any time through a user-friendly interface, without the need for interfering with system's code.

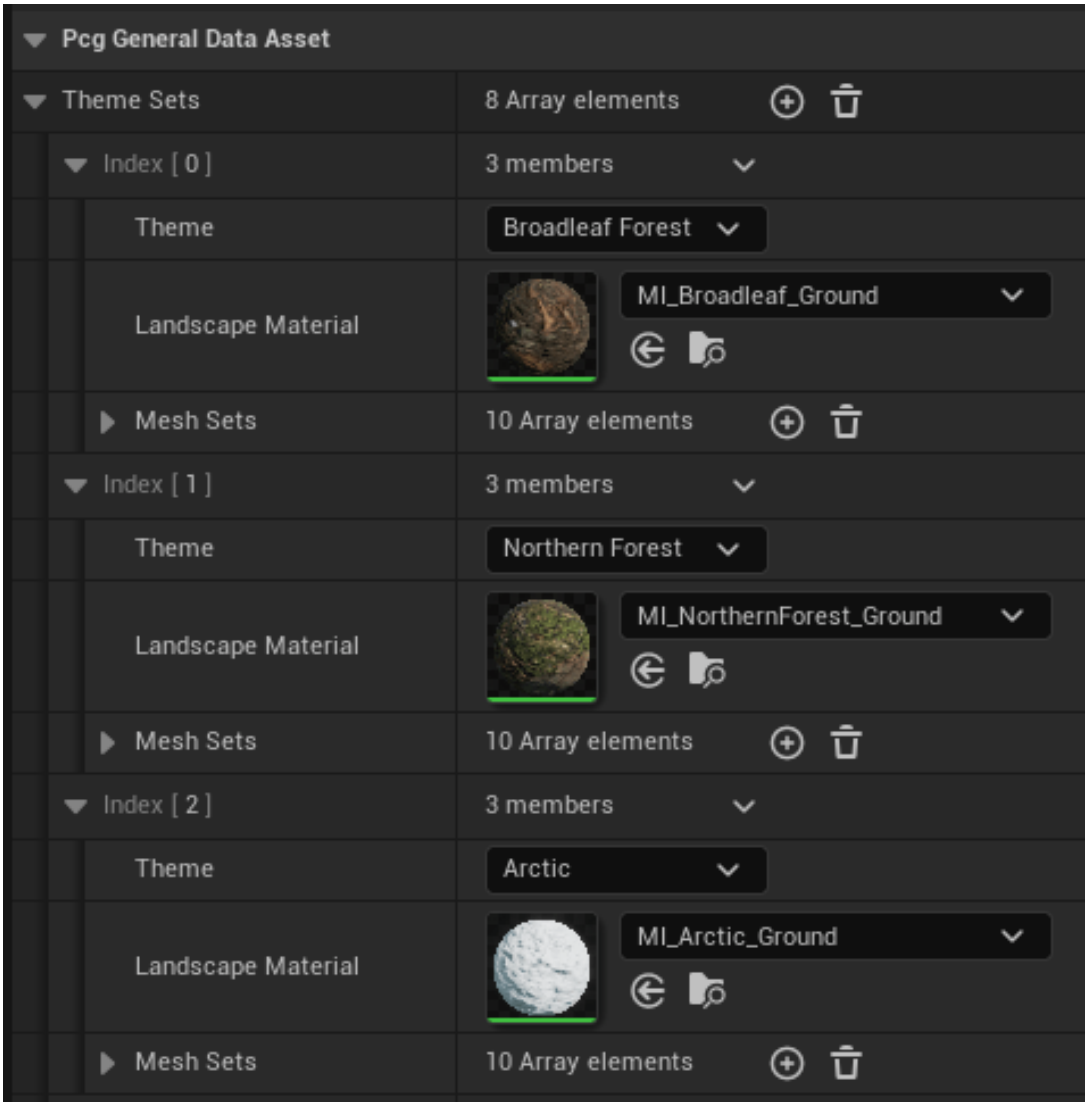

Figure 11: Example of themes and their information from the data asset

As shown in figure 11, the data asset is composed of multiple theme sets. Each set is composed of the following information:

- **Theme:** refers to the environmental theme that we want to generate. In this case, we can see examples of "Broadleaf Forest", "Northern Forest", and "Arctic".

- **Landscape Material:** represents the type of material and textures that covers the ground/surface. In the case of "Arctic", this represents "Snow".

- **Mesh Sets:** represents a collection of "Mesh Data" elements. It contains a list of categories, such as "Grass" or "Trees", and associated 3D models alongside their configurations.

Figure 12 shows examples of the implemented mesh sets in more detail.

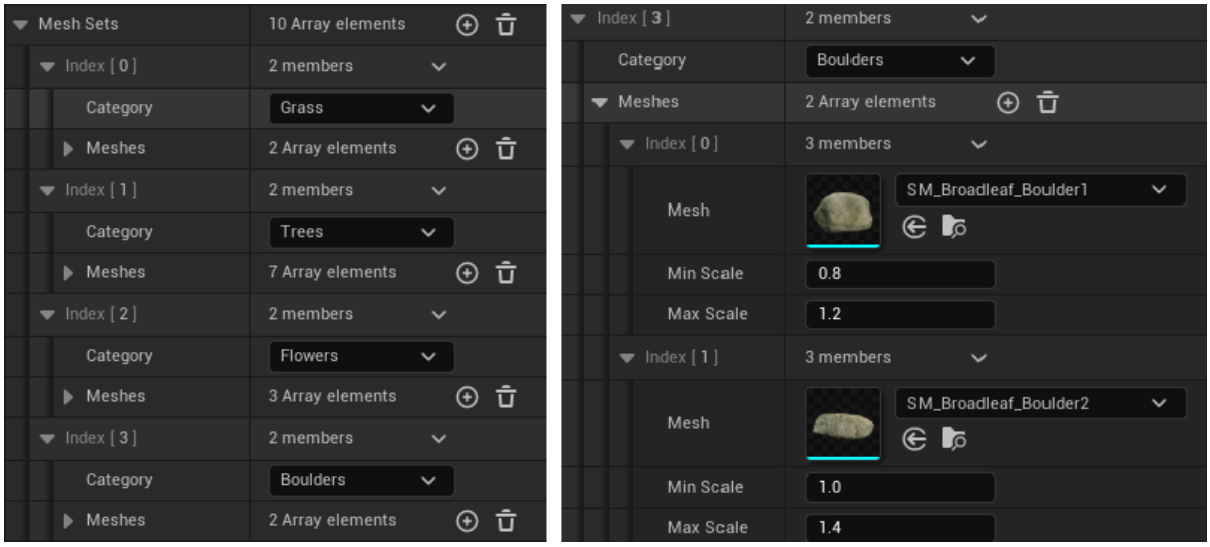

Figure 12: Example of filled mesh sets from the data asset

On the left are some categories that the system uses to differentiate between 3D models. On the right, models are associated with a certain category, and their configurations are filled.

Another important part of the system's implementation is the "PCG Component" and its "PCG Graph" that links all the steps needed for procedurally generating environments. Visualizing the code through Blueprints, the built-in visual scripting language of Unreal Engine, results in the graph shown in figure 13.

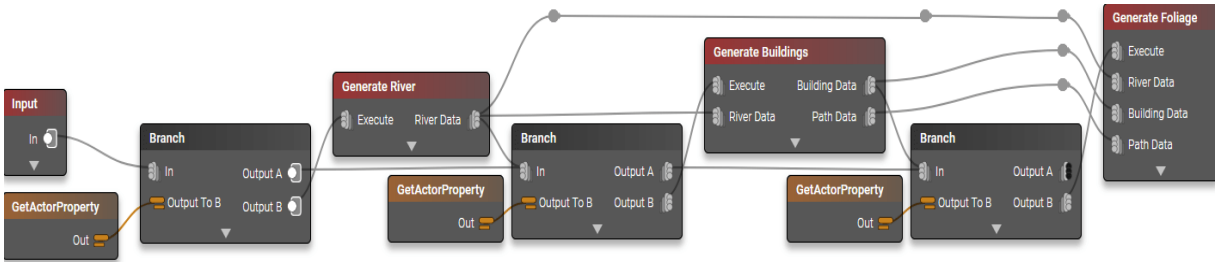

Figure 13: Blueprint representation of the master PCG graph

From a high-level perspective, the implementation consists of these steps:

1. First, we retrieve the requirements from the central class, called "Master Generator".

2. If generating a river is required, we execute its associated code and pass its resulting data to the next step.

3. If generating buildings is required, we execute its associated code while considering the generated river if it exists. We then pass its resulting data to the next step.

4. If generating any kind of foliage is required, we execute its associated code while considering the generated river and buildings if they exist.

5. Finally, we combine the results of each step and display them on the selected landscape, which will be covered with a material that fits the current theme.

To discuss the implementation of the system in more detail, we will go through the development of each environmental aspect, then showcase the final results of the system.

- **River Generation**

Visualizing the code responsible for generating a river through Blueprint results in the graph shown in figure 14.

Figure 14: Blueprint representation of the river generation code

The graph describes the steps we took in order to generate a river procedurally:

1. First, we sample the selected landscape's surface into points, that we can use to spawn parts of the river on. The code always aims to generate around 16 points across the entire landscape.

2. Depending on the desired location of the river, we filter the points and select the appropriate ones. For example, if the requirement is to generate a river on the left side of the landscape, then we only continue processing the points that belong to that side.

3. We then assign two points as the starting and ending points of the river. Based on the selected parameters, this can be done randomly, or by choosing the closest points to the edges of the landscape, either vertically or horizontally.

4. The points are sorted from the starting point to the ending point and used to create a spline. A spline is a smooth and continuous curve that is used to create the shape of the river. The 3D model which represents the river's surface is placed multiple times on top of this spline.

5. The same spline is re-positioned and used to place 3D models of a riverbank on the right and on the left of the river.

6. Finally, additional 3D models are spawned alongside the riverbanks if it is required. These could be any type of foliage, such as trees and boulders.

The implementation exposes many adjustable parameters that allow the customization of the river generation, such as selecting the position of the river on the landscape. The pictures shown in figures 15, 16, 17 and 18 demonstrate some of the examples of river generation and customization achieved by the previously explained implementation.

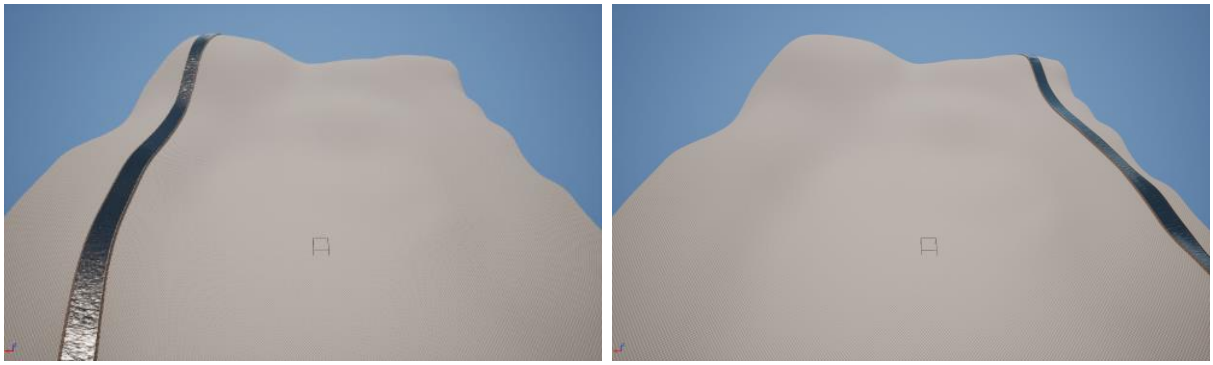

Figure 15: Example of river generation on the left and right side of the landscape

Figure 15 displays two examples of river generation. The first picture shows a river being generated on the left side of the landscape, while the second picture shows one being generated on its right side. This is an example of how it is possible to choose the placement of the river on the landscape.

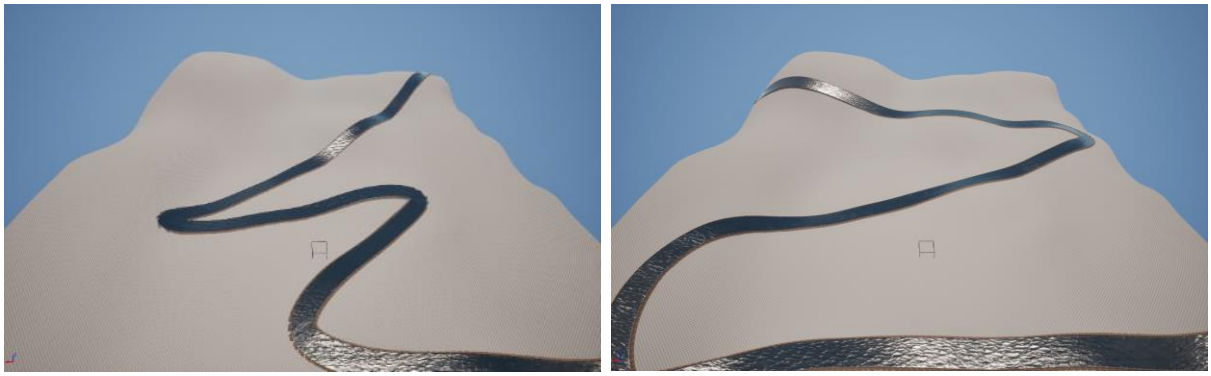

Figure 16: Example of river generation using random points

Figure 16 shows two examples of river generation based on the selection of random points. This is used to attempt replicating how a river would look like in nature and results in an unpredictable, yet believable trajectory for the river.

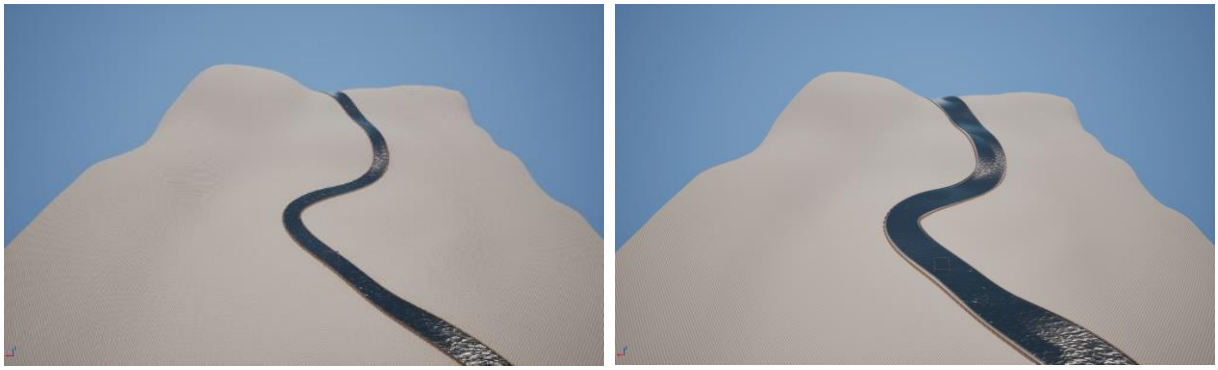

Figure 17: Example of river generation in the middle with different widths

Figure 17 demonstrates two examples of river generation, both passing through the middle of the landscape. However, there is a difference in the width of the river between the two generations, as the implementation supports adjusting the river width in its parameters.

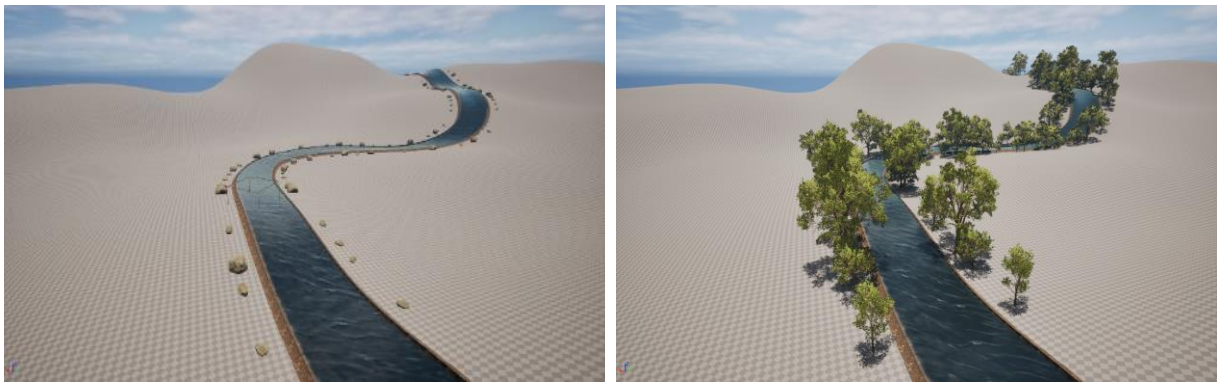

Figure 18: Example of natural elements placed alongside the river

Figure 18 shows two examples of natural elements being generated alongside the river. These elements can be chosen through an enumeration parameter supported by the system's implementation. It is also possible to control their density. The first picture shows boulders on the right and left of the riverbanks, while the second picture shows trees alongside them.

- **Building Generation**

Visualizing the code responsible for generating buildings through Blueprint results in the graph shown in figure 19.

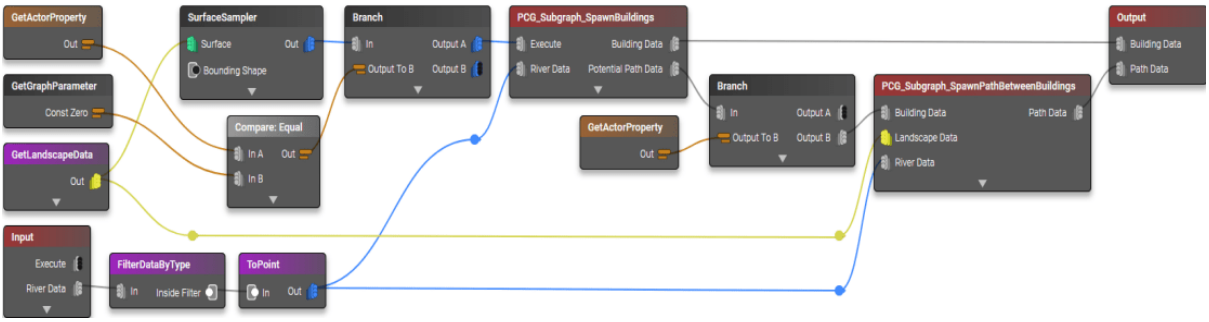

Figure 19: Blueprint representation of the building generation code

The graph describes the steps we took in order to generate buildings procedurally.

1. First, we sample the selected landscape's surface into points, that we can use to spawn buildings on top of.

2. We eliminate the points that intersect with the points that are used to generate a river if applicable, so that we do not generate buildings on top of the river.

3. We eliminate the points that are placed on irregular surfaces or have a sharp rotation since it is unrealistic for buildings to be constructed in such conditions. We also eliminate points that overlap with each other to avoid buildings clipping through each other.

4. We spawn 3D models of buildings on top of the remaining points and rotate them randomly, to eliminate uniformity and increase variety.

5. If drawing a path from one building to another is required and there are at least two buildings, we create a new point for each building that is near the front entrance/door. These points are then used to create a spline that represents the shape of the path, which passes near each building, from the first generated house to the last one.

6. Finally, we spawn 3D models that represent the path's ground/surface on top of the created spline, if required.

The implementation exposes many adjustable parameters that allow the customization of buildings generation, such as controlling the amount of buildings to spawn. The pictures illustrated in figures 20, 21, 22 and 23 showcase some of the examples of buildings generation achieved by the previously explained implementation.

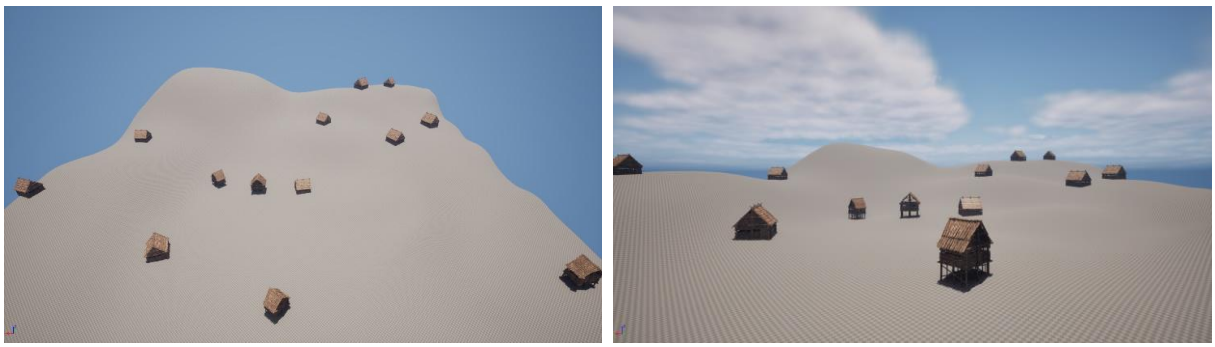

Figure 20: Example of buildings generation

Figure 20 shows an example of buildings generation. The first picture is a top-down view of the landscape with buildings placed on it, while the second picture is a close-up view of the buildings. We can notice that the buildings are only generated on regular and flat surfaces. They are also rotated to create a sense of variety.

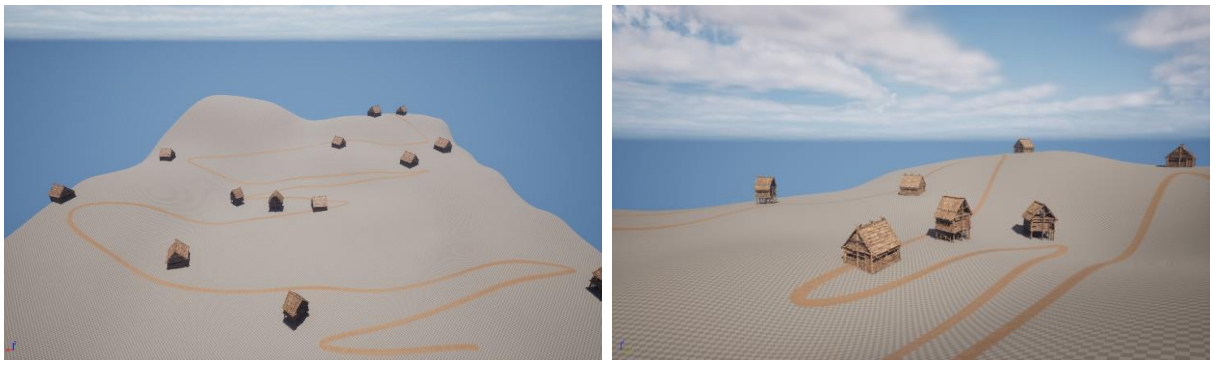

Figure 21: Example of generating a path between buildings

Figure 21 demonstrates an example of generating a path between the existing buildings. We can see that the path leads from one building to another, starting from the top to the bottom of the landscape. The path is drawn in a way that makes it look like the entrances of the buildings are oriented to face it.

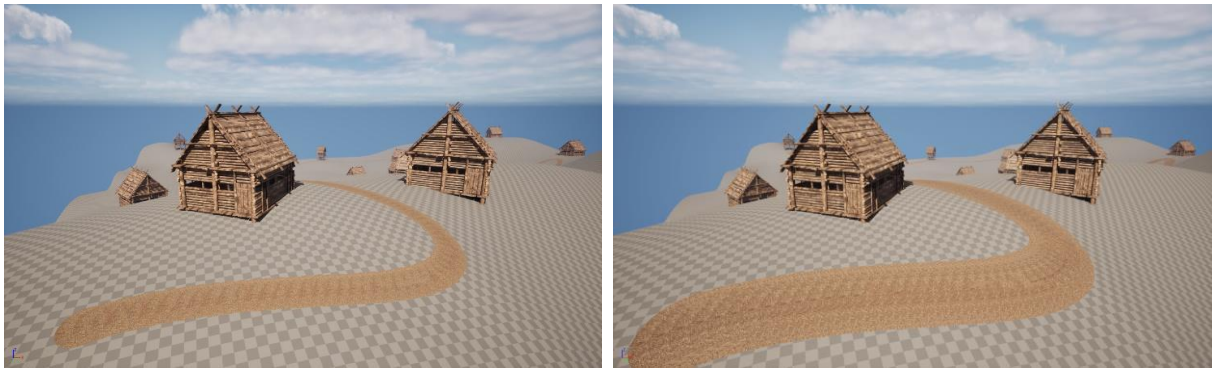

Figure 22: Example of generating a path with different widths

Figure 22 displays the ability of generating a path between buildings with adjustable width. The first and second picture are using the same generated buildings and path shape, with the only difference being the amount of width assigned to the path.

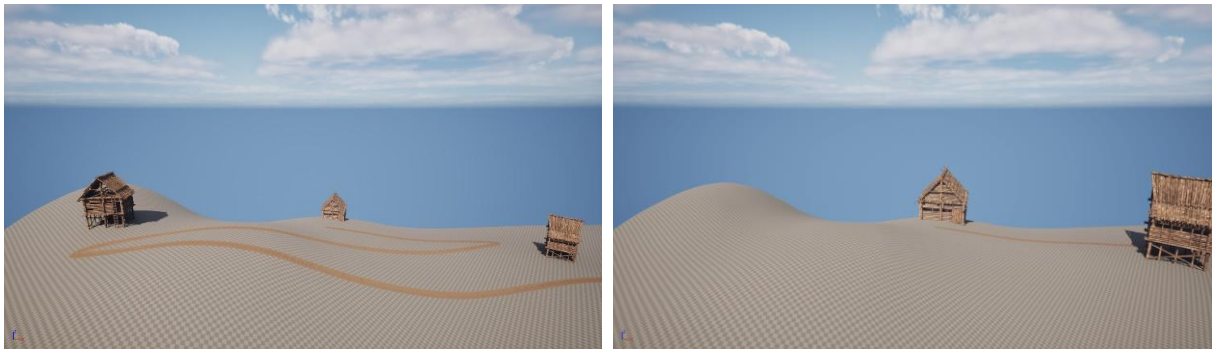

Figure 23: Example of generating buildings with different scales

Figure 23 compares the results of buildings generation when using different scales/sizes. In the first picture, the buildings are assigned the default scale, and a path is drawn from one to another normally. In the second picture, the buildings are assigned a much bigger scale. This resulted in the one on the left to be removed, since it would have caused a visual error, such as being generated on an irregular surface. This also caused the path to be adjusted.

- **Foliage Generation**

Visualizing the code responsible for generating foliage through Blueprint results in this graph shown in figure 24.

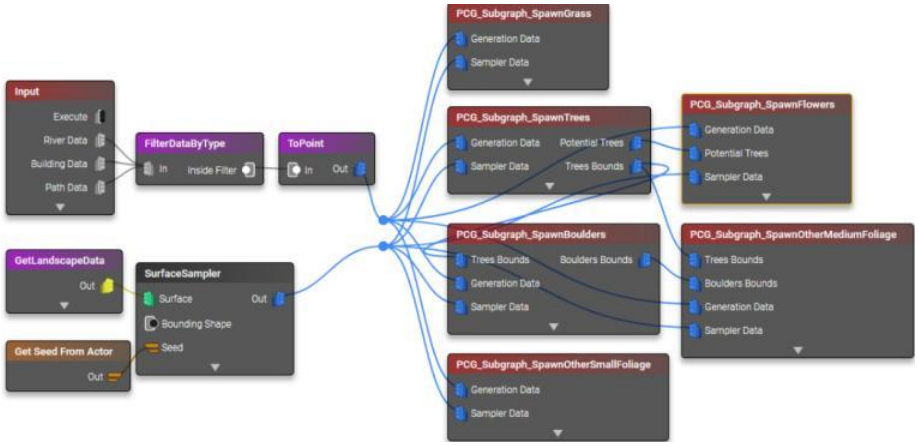

Figure 24: Blueprint representation of the foliage generation code

The aforementioned graph describes the steps we took in order to generate multiple kinds of foliage procedurally:

1. First, we sample the selected landscape's surface into points, that we can use to spawn foliage on top of.

2. We eliminate all points that intersect with the points that are used to generate a river or buildings, if applicable, so that we do not generate foliage that overlaps with either of them.

3. For trees, grass, and other miscellaneous small-size foliage, we select random points using a constant percentage ratio that feels right, and we generate the respective 3D models on top of them.

4. For flowers, we select the points that could represent a tree across the entire landscape. New points are created and scattered around each one of the potential trees' points. These are then used to generate flowers that are grouped near each other, which results in a believable aesthetic.

5. For boulders and other miscellaneous medium-size foliage, we select random points using a constant percentage ratio that feels right, and eliminate the points already

reserved by the tree generation. We also ensure that there are no overlapping elements that are present in either of them.

6. Finally, all generations are then combined and displayed together, to represent the overall foliage generation. The density and scale of the points of each category can be adjusted through separate parameters implemented in the system.

The implementation exposes many adjustable parameters that allow the customization of foliage generation. Figures 25, 26, 27, 28, 29, 30 and 31 showcase some of the examples of foliage generation achieved by the previously explained implementation.

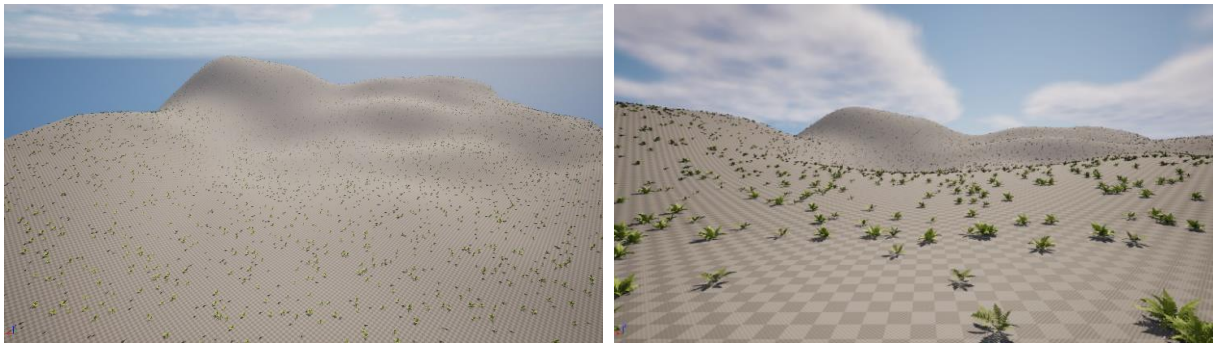

Figure 25: Example of generating grass

Figure 25 shows one of the possible outputs of grass generation. The first picture consists of a top-down view of the entire landscape, showing grass scattered on top of it. The second picture consists of a close-up view of the grass elements. We can notice that they are always aligned with the landscape's surface, and are oriented to create a feeling of variety.

Figure 26: Example of generating trees

Figure 26 shows the output of trees generation. The first picture consists of a top-down view of the entire landscape, showing trees scattered on top of it. The second picture consists of a

close-up view of the trees. They are placed in a way that makes them look like they are growing upwards, just like in nature.

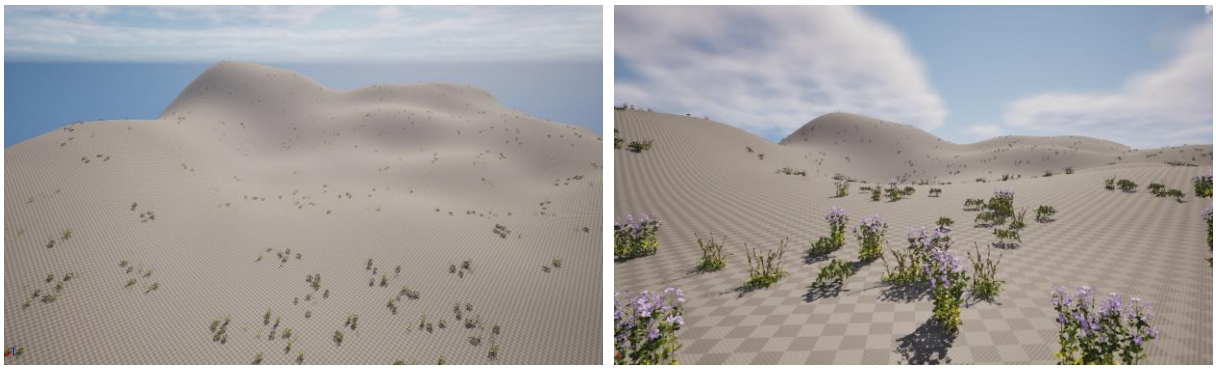

Figure 27: Example of generating flowers

Figure 27 shows the output of flowers generation. The first picture consists of a top-down view of the entire landscape, showing flowers scattered on top of it. The second picture consists of a close-up view of the flowers. We can notice that they are generated in the form of groups and are oriented to create a feeling of variety.

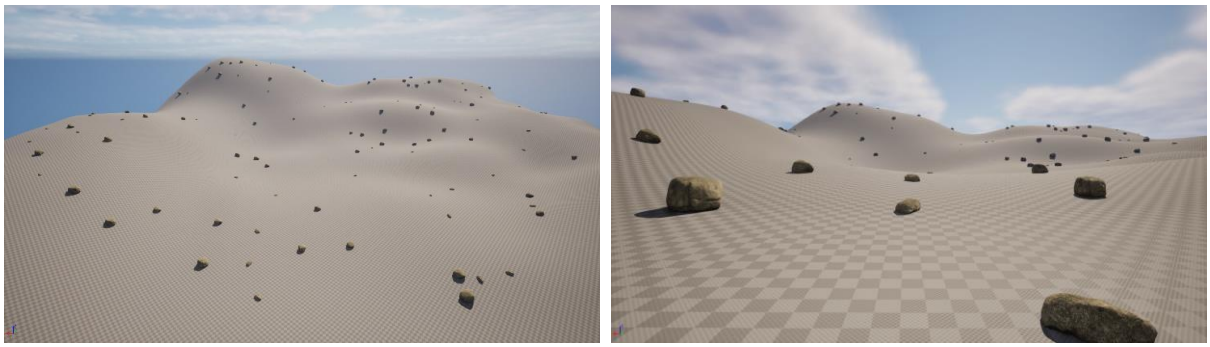

Figure 28: Example of generating boulders

Figure 28 shows the output of boulders generation. Similarly to other elements, we can notice that they are aligned with the landscape's surface and are of different sizes and orientations.

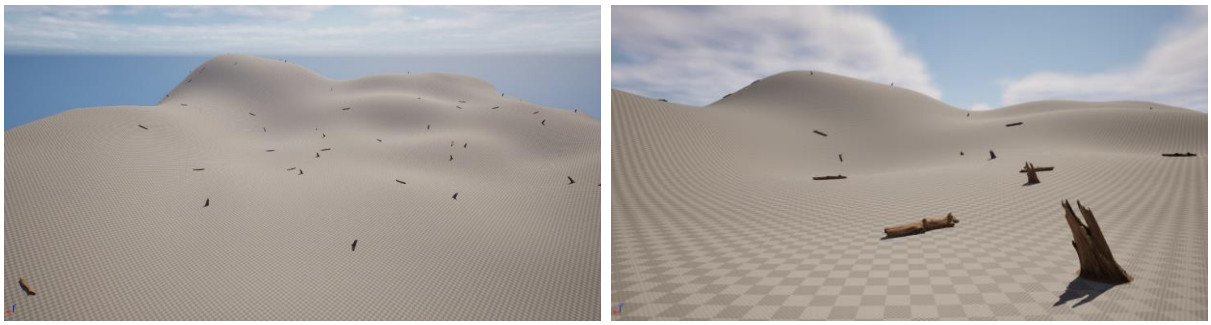

Figure 29: Example of generating miscellaneous medium-size foliage

Figure 29 shows the output of medium-size foliage generation. The first picture consists of a top-down view of the entire landscape, showing the elements scattered on top of it. The second picture consists of a close-up view of them. The chosen elements for this category differ based on the selected environmental theme.

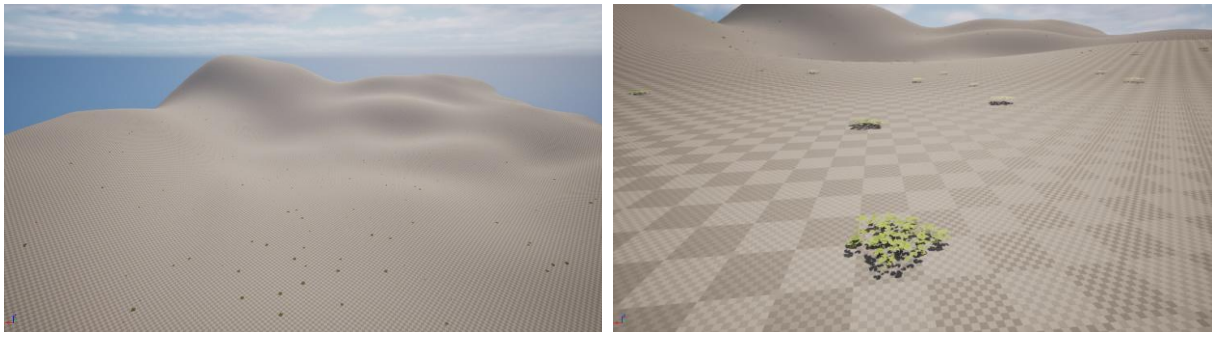

Figure 30: Example of generating miscellaneous small-size foliage

Figure 30 shows the output of small-size foliage generation. The first picture consists of a top-down view of the entire landscape, showing the elements scattered on top of it. The second picture consists of a close-up view of them. The chosen elements for this category differ based on the selected environmental theme.

Figure 31: Example of generating multiple kinds of foliage at the same time

Figure 31 shows the combination of generating all the supported types of foliage in the implemented system. We can see that they are capable of existing together at the same time seamlessly, which creates a believable layer of the environment.

- **Final Output of The System**

Using this implementation of the system, it is possible to add, remove, and adjust many properties of the generated elements, which results in a vast amount of possible environments. In this sub-section, we will go through some examples of the final generations of the implemented PCG system as a whole for every designed environmental theme.

Figure 32: Generated environment for the theme "Broadleaf Forest"

Figure 32 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Broadleaf Forest" theme. In the first and second pictures, we can see multiple kinds of vibrant foliage that blend seamlessly together. In the third and fourth pictures, we can see cabins made of wood, which fit thematically with the natural surroundings. A river passes through the landscape and reflects

the sunlight. In the fifth and sixth pictures, we can see a broader view of the environment, which is of a village inside of a forest. We can see a path drawn between the buildings.

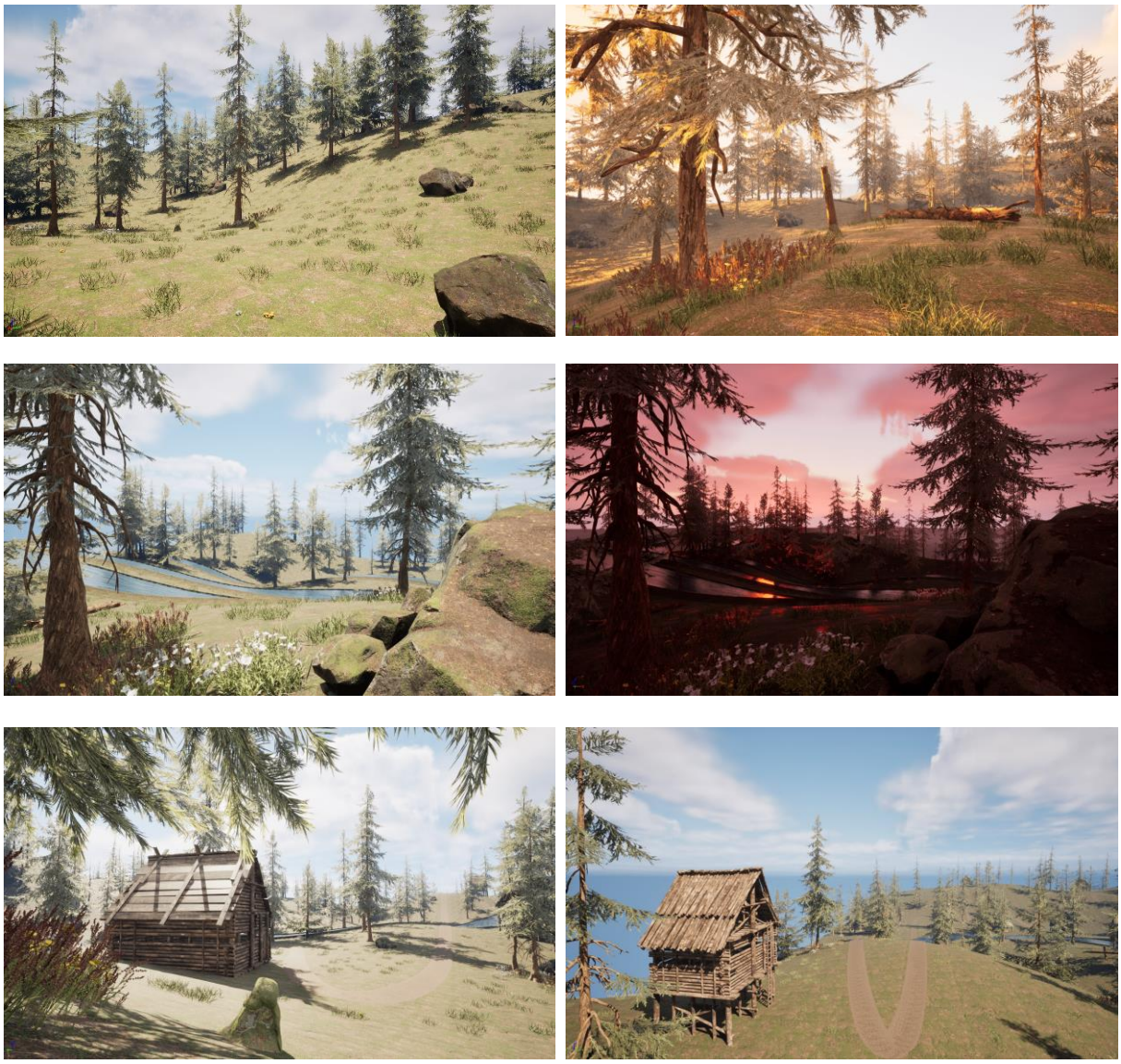

Figure 33: Generated environment for the theme "Northern Forest"

Figure 33 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Northern Forest" theme. In the first and second pictures, we can see diverse and dense vegetation which is generated in a believable and beautiful way. In the third and fourth pictures, we can see a river which passes through the forest. The same scene is shown under different lighting conditions. In the fifth and sixth pictures, we can see similar buildings to the ones of the "Broadleaf Forest" theme, as the main difference between the two themes is mainly in the kinds of foliage. As implemented, the buildings are oriented so that their entrances are always facing the drawn path.

Figure 34: Generated environment for the theme "Arctic"

Figure 34 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Arctic" theme. In the first and second pictures, we can see trees similar to the ones of the "Northern Forest" theme, but like the landscape, are covered in snow. We can also observe other types of vegetation that grows in cold areas. In the third and fourth pictures, we can see houses covered by snow like the natural surroundings, as well as a dark-coloured river flowing near the buildings. All the generated elements fit together to create a realistic scene, simulated at sunrise and sunset, respectively. In the fifth picture, we can see a close-up view of the generated building, made using cold-resistant materials, as well as a snowy path. In the sixth picture, we can see a close-up view of an animal skull, which gives the impression that there is wildlife in the generated environment. The animal skull is categorized as miscellaneous medium-sized foliage in the system's data asset.

Figure 35: Generated environment for the theme "Desert"

Figure 35 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Desert" theme. In the first and second pictures, we can see a sand-covered landscape with multiple palm trees. Compared to the other themes, the density of the vegetation is lacking. An animal skull is displayed to emphasize the deadliness of the desert. We can also see an example of a building situated in such an environment, built in heat-resistant materials. In the third and fourth pictures, we can see different views of the desert and its elements under the lighting of sunset. In the fifth picture, we can see a broader view of the generated desert as if we were standing on one of the buildings' balcony. In this context, a light-colored river has also been generated. In the sixth picture, we can see a simulation of the desert at sunrise, with the river reflecting the sunlight slightly.

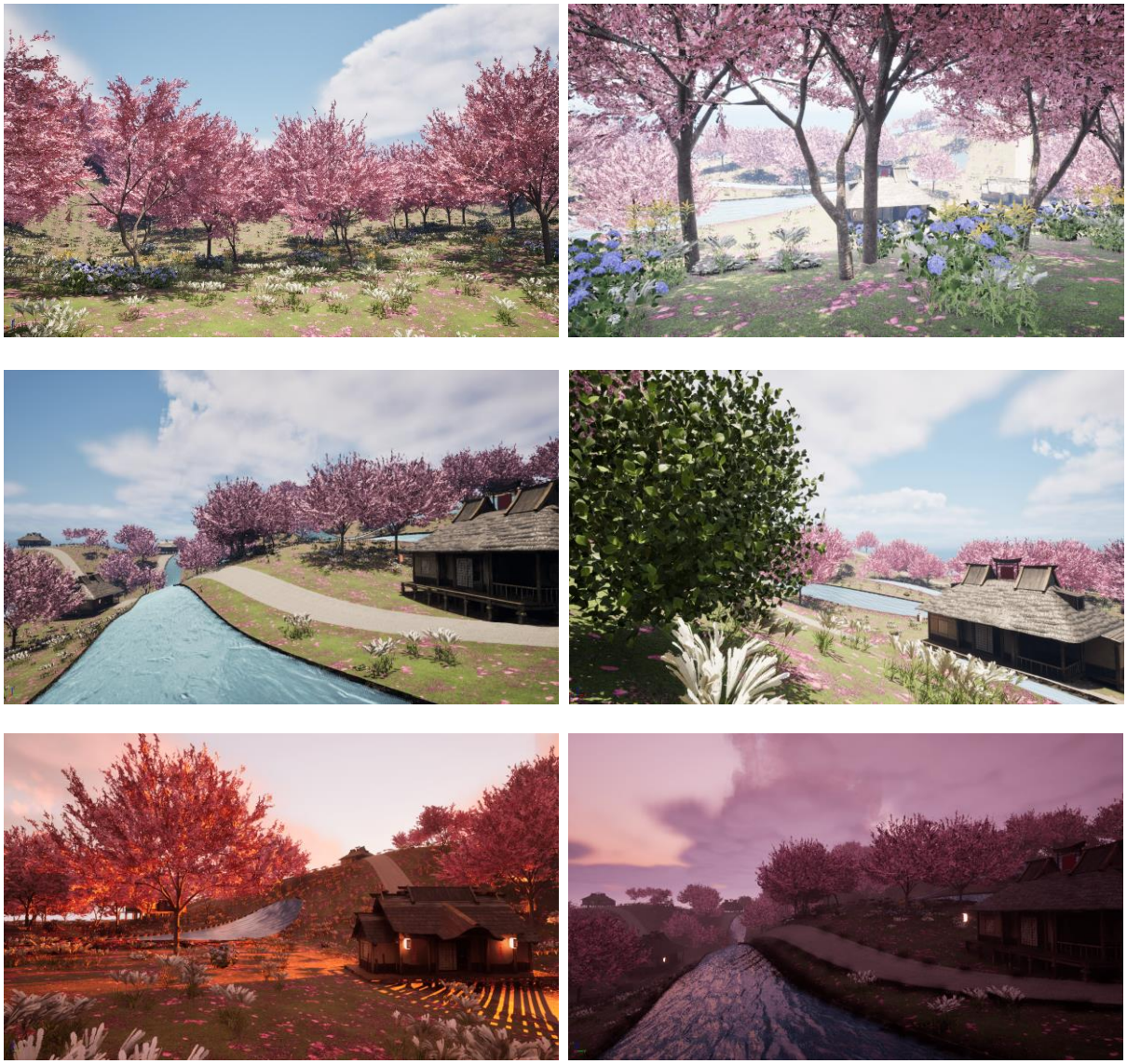

Figure 36: Generated environment for the theme "Japanese"

Figure 36 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Japanese" theme. In the first and second pictures, we can see two views that emphasize the rich vegetation of the Japanese environment, most notably the cherry blossom trees, also known as the "Sakura" trees. Their leaves cover the ground where multiple kinds of flowers can also be observed. In the third and fourth pictures, we can see traditional buildings that fit the cultural setting we are trying to achieve. A river can also be seen flowing near the buildings, as well as diverse kinds of foliage. In the fifth and sixth pictures, the generated environment is showcased under sunset and sunrise, respectively. The elements of the environment react to the changes in lighting appropriately. We can also notice that the traditional lanterns that are attached to the buildings are emitting light.

Figure 37: Generated environment for the theme "Canyon"

Figure 37 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Canyon" theme. In the first and second pictures, we can see different kinds of foliage that are mostly unique to the southwestern United States. Similar to the "Desert" theme, the density of the vegetation is lacking. We can also observe an animal skull, which emphasizes the hot temperature that the generated environment is supposed to have. In the third picture, we can see an example of a generated river which reflects the sunlight, and a close up of the big geological structures and boulders that should be present in such an environment. In the fourth, fifth, and sixth pictures, we can see close-up views of the American style buildings that are generated in the environment, as well as some large vegetation, such as cactuses and Joshua trees. The scenes are presented under different lighting settings.

Figure 38: Generated environment for the theme "Fantasy"

Figure 38 illustrates the diverse qualities of this 3D environment generated using the implemented system and highlights a range of elements that belong to the "Fantasy" theme. This theme is based on an imaginary setting as the name implies, and serves as proof that the implemented system is capable of generating any kind of 3D environments once its configuration is added to the system's data asset. In the first and second pictures, we can see all kinds of imaginary foliage with exaggerated sizes and colors. At low levels of sunlight, the trees, mushrooms, and flowers emit a light of their own. The river can be seen passing through the landscape, with a pink and blue color. In the rest of the pictures, we can see two scenes: the one on the left features daylight, while the one on the right features sunset or sunrise. These scenes consist of close-up views of two types of imaginary buildings that are floating on top of the landscape's surface, surrounded by various kinds of foliage. We can also see a path made out of stars, which links all the buildings together.

4.1.5 Solution Evaluation

The process of solution evaluation was conducted over several steps. The first step was to confirm that the system meets the performance criteria that was set during the design. Using Unreal Insights, a built-in profiler in Unreal Engine, we analyzed the performance of the system.

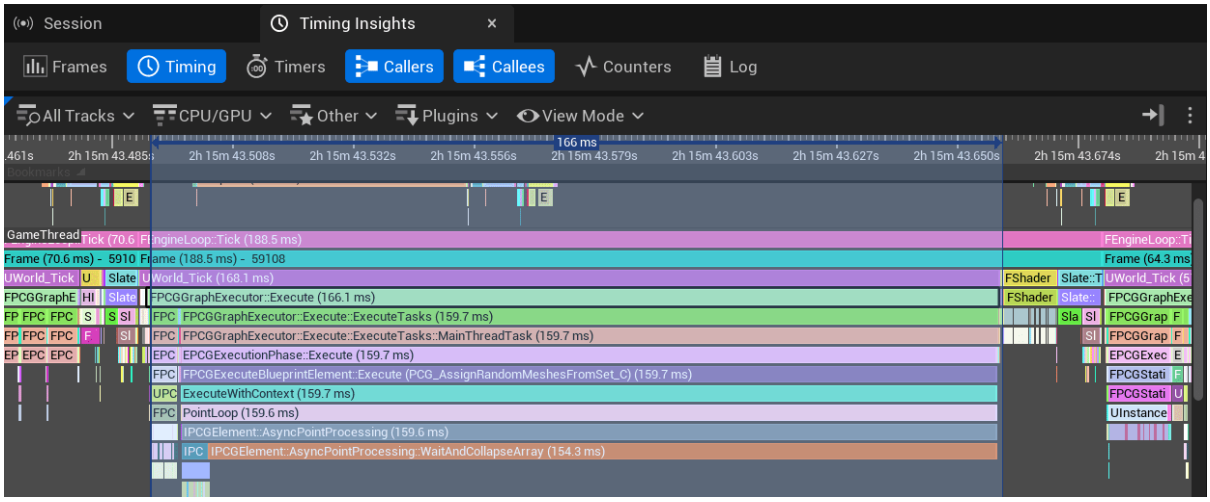

Figure 39: Capture from Unreal Insights showing the longest execution instance

Figure 39 belongs to the analysis of the **longest execution instance** that occurred while generating multiple environments using the implemented system. The profiler shows that the longest execution of the system's process took **166.1 milliseconds**, which is satisfactory for an on demand in-editor process. Afterwards, different stakeholders got to evaluate the results of the system through a recorded demo. The feedback received from all of them was considered and further improvements were made to the system. Table 6 discusses some of the feedback received and improvements made.

Stakeholder	Feedback	Improvement
Technical Lead	Suggested preparing the system's parameters in a way that facilitates integration with the rest of the tool's modules later on.	Made the system's parameters easy to access from other classes by making multiple methods that can update them properly.
Art Lead	Suggested improving the visual clarity of some 3D models and adding more variety.	Used at least 2 good quality models for each category to be generated in each environmental theme.
Development Team	Identified issues with some elements when scaled up.	Added self-pruning to some elements that should remove overlapping instances when scaled.

Table 6: Feedback received by the stakeholders and improvements made

4.2. Iteration 2: Prompt Interpretation and Integration with the PCG System

The second iteration's objective is to design and implement a prompt interpretation system that seamlessly integrates with the developed PCG system, allowing the creation and adjustment of environment through prompts.

4.2.1 Problem Investigation

In order to generate 3D environments based on user specifications, we need to design and implement a system that is capable of understanding the context of a prompt, and transforming it into known parameters that the PCG system could use when generating or adjusting an environment. Without this interpretation process, it would be impossible for the tool to generate 3D environments that satisfy user expectations. It is also important to analyze and design the workflow of such a system before the implementation phase, to eliminate inefficient integration and to identify the necessary entities/components that have to be used.

4.2.2 Solution Design

To better understand the objectives of this iteration, we will start the design process by identifying the functional requirements that will guide the implementation later on. Table 7 lists every functional requirement and its description.

Functional Requirement	Description
Identify the prompt's context	The system must be able to understand if a prompt refers to river, buildings, and/or foliage generation.
Transform the prompt into generation requirements	The system must be able to transform the prompt into requirements used to determine which environmental aspects (river, buildings, foliage) to generate.
Transform the prompt into generation parameters	The system must be able to transform the prompt into parameters that can be used to customize river, buildings, and/or foliage generation.
Recognize feedback on an existing generation	The system must be able to understand prompts that represent feedback about an existing generation.
Adjust transformed parameters based on the current state of the existing generation	The system must be able to take into consideration the current parameters and requirements of the existing generation when interpreting feedback.

Adjust transformed parameters based on a change magnitude	When transforming the feedback into parameters, the system must be able to affect the parameters using a change magnitude (e.g.: small, normal, large change).

Table 7: Functional requirements of the prompt interpretation system

Next, we have to identify the quality attributes that the prompt interpretation system must meet to ensure that it will work with the other systems of the tool without hindering the overall process. Table 8 outlines the selected quality attributes and their descriptions.

Quality Attribute	**Description**
Efficiency	The system should minimize the number of requests needed to fully interpret and transform a given prompt.
Robustness	The system should be able to handle all errors concerning prompt interpretation and transformation, like missing values, to ensure that it can be integrated with other systems and used without problems.

Table 8: Quality attribute of the prompt interpretation system

Next, we focus on the design of the multiple workflows that the system needs in order to achieve the aforementioned functional requirements and quality attributes. A total of three activity diagrams were created to explain the most important workflows of the system and how it should be integrated with the implemented PCG system.

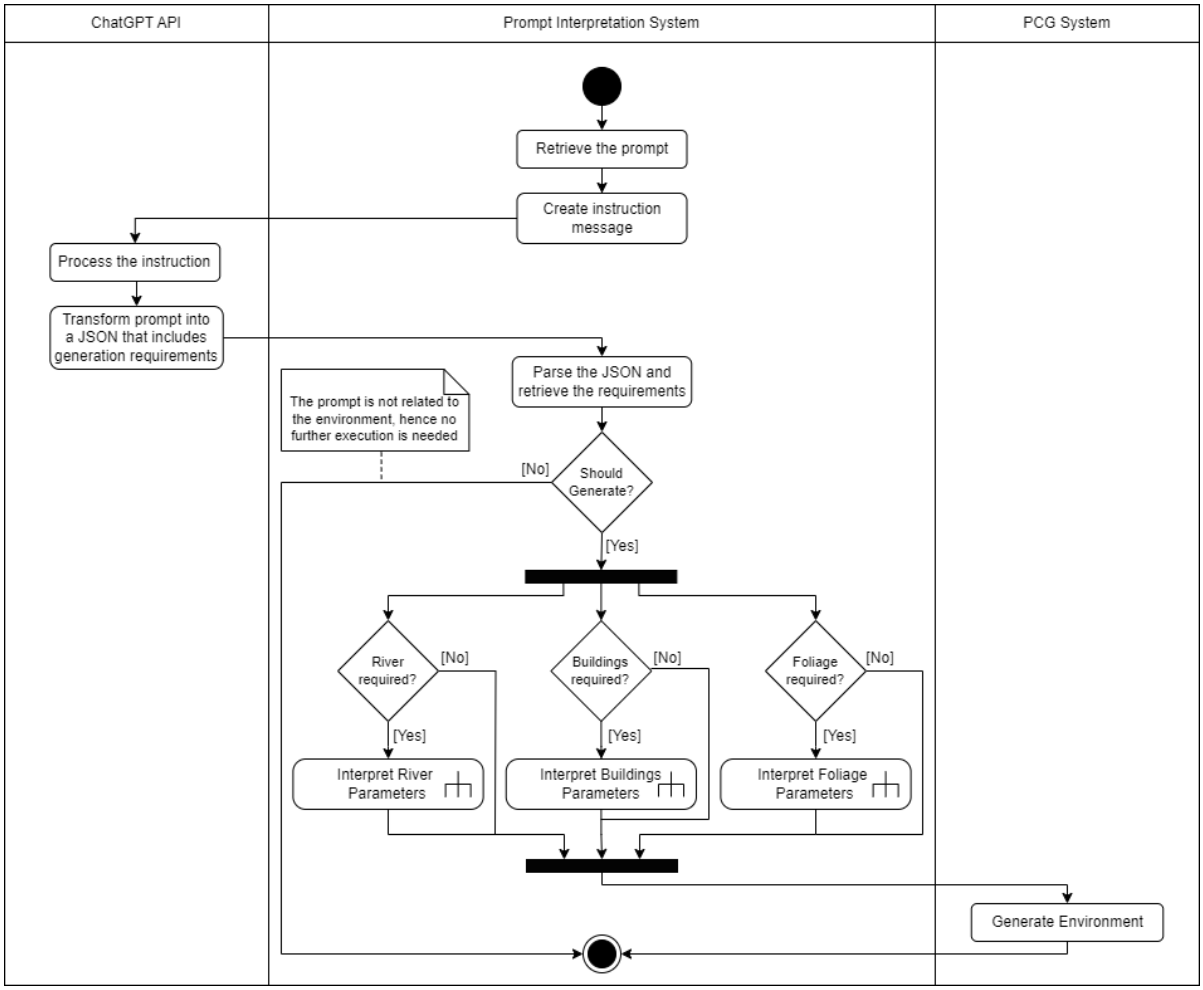

*Figure 40: Activity diagram about basic prompt interpretation * [34]*

This first activity diagram belongs to basic prompt interpretation. In this case, we do not have a generated environment yet, and the goal is to determine which environmental aspects to generate and with what parameters. We have decided to adopt the ChatGPT API which allows the system to easily interpret prompts with high fidelity and transform them to parameters based on instructions. These instructions can be written in common language which facilitates designing and tweaking the results of the transformation. The workflow starts by retrieving the prompt that we should interpret. This prompt will be combined with text-based rules to create an instruction message, containing all the information that the ChatGPT API would require. Once it receives and processes the instruction, it sends back data that includes all the generation requirements that fit the context of the prompt. Based on these requirements, we execute further interpretation workflows to gather the parameters for river, buildings, and/or foliage generation. The PCG System is then informed to generate an environment, using the recently received parameters.

* The activity with a rake-style symbol within it refers to "CallBehaviorAction" as defined in UML 2, and is used to start another activity that has further decomposition in another diagram.

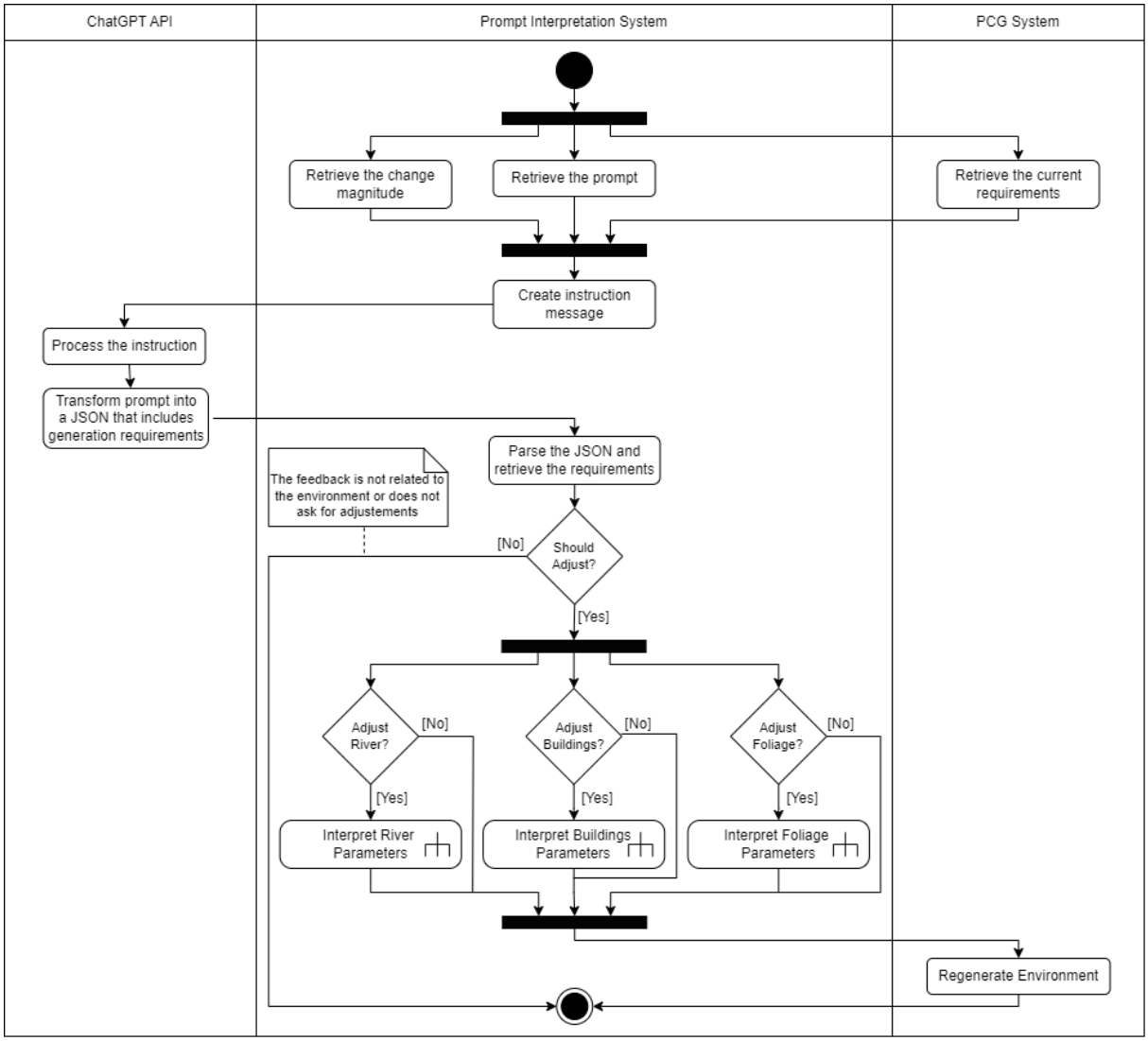

*Figure 41: Activity diagram about feedback prompt interpretation * [34]*

This second activity diagram belongs to feedback prompt interpretation. In this case, we have already generated an environment, and the goal is to determine which aspects of the environment (river, buildings, foliage) to adjust based on the feedback. The workflow starts by retrieving three elements that are crucial to understanding the context of the feedback. The **prompt** represents the feedback provided by the user. The **change magnitude** affects the degree of adjustment that should happen. The **current requirements** refer to the requirements of the existing environment generation, and provide context of what we have already generated. An instruction message is created based on these elements and is sent to the ChatGPT API, which in turn will process and transform the feedback into data that contains the adjusted environment generation requirements. Similarly to the first activity diagram, we execute further interpretation workflows to gather the parameters for river, buildings and/or

* The activity with a rake-style symbol within it refers to "CallBehaviorAction" as defined in UML 2, and is used to start another activity that has further decomposition in another diagram.

foliage generation based on the feedback. The PCG System then regenerates the environment with the adjusted parameters.

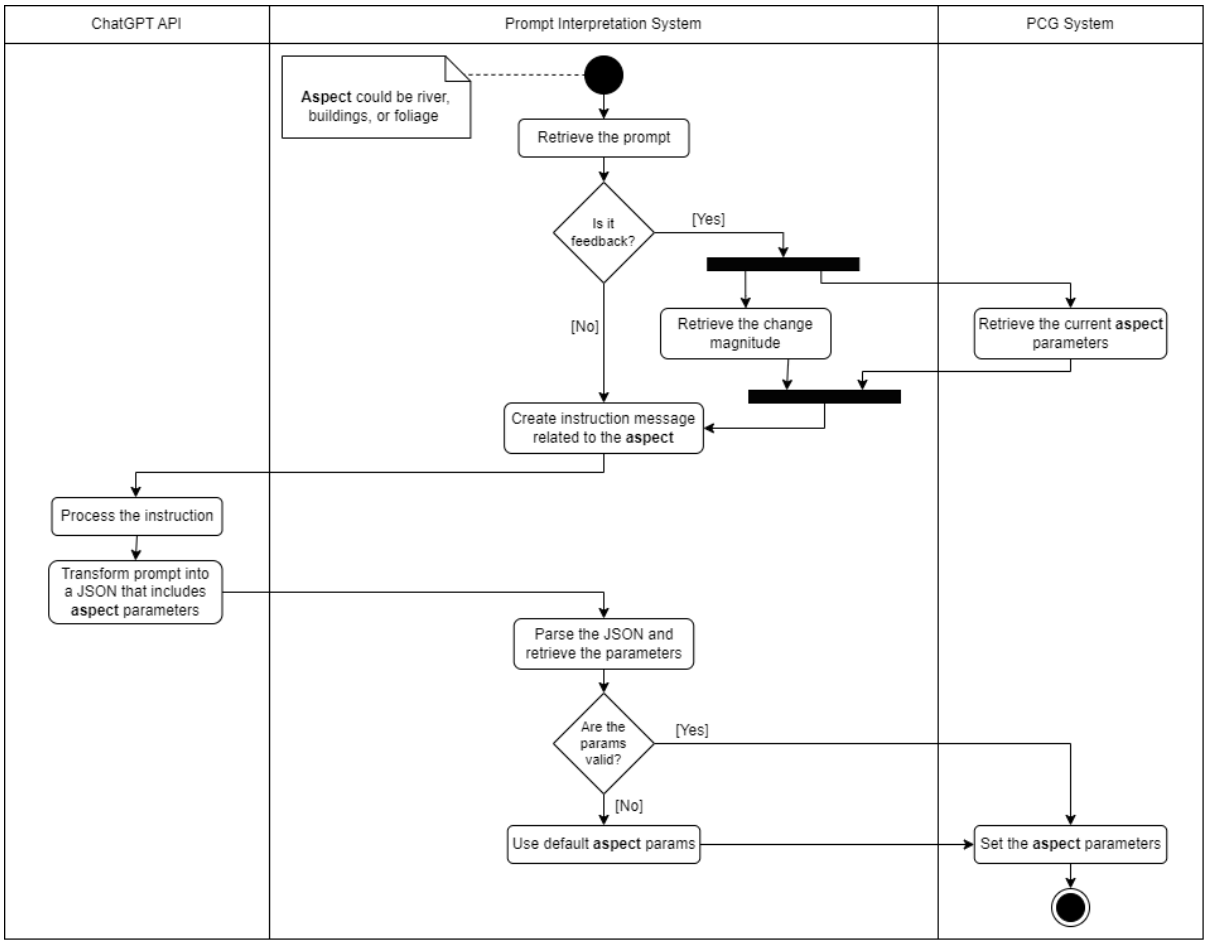

*Figure 42: Activity diagram about river/buildings/foliage parameters interpretation * [34]*

This third activity diagram explains the interpretation of parameters that could belong to either river, buildings, or foliage generation. The same workflow can be used for any of these environmental aspects, and is called from the first and second activity diagrams. The workflow starts by retrieving the prompt provided by the user. If the prompt is flagged as feedback, then we also retrieve the change magnitude, and the current aspect parameters that are assigned in the PCG System. Based on that, we create an instruction message that explains the purpose and type of each parameter, and provides the necessary context about the environmental aspect. The instruction is then processed by the ChatGPT API and transformed into a set of parameters with new values. Once received, this data is parsed by the interpretation system. If the parameters and their values are validated, we set them as the new aspect parameters in the PCG System, otherwise, we use the default values for these parameters as a fallback.

* This activity diagram describes the same workflow used for interpreting parameters for river, buildings and foliage. It was called as an activity in the figures 40 and 41 using a node with a rake-style symbol, as defined in UML 2.

To better understand and simplify the flow of information and data moving between the three entities used in this iteration, a data flow diagram was illustrated.

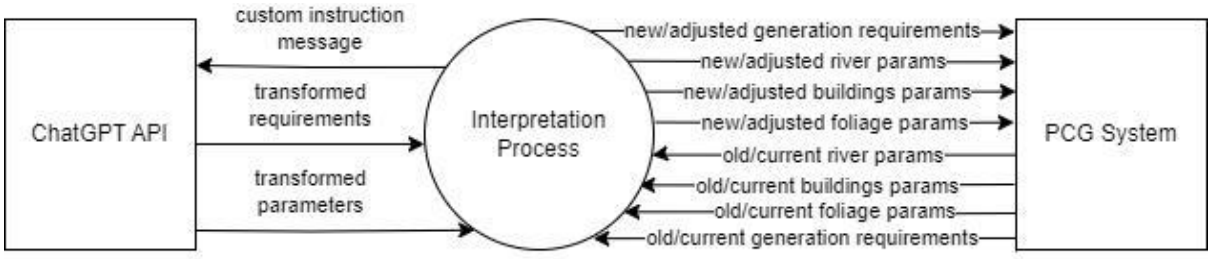

Figure 43: Data flow diagram of the interpretation process [35]

Figure 43 shows the type of information send and received from each entity during the interpretation process. This way, the PCG System is constantly updated every time a prompt is processed.

4.2.3 Design Validation

The process of design validation was carried out during a meeting with the technical lead of the company, where the designed workflows were discussed in detail. During the meeting, an analysis of the costs of integrating the ChatGPT API was also conducted, in which a specific version of the model was selected to be used and a maximum length for the instructions was set, so that the potential cost of the requests respects the budget. Finally, the designed solution and the way it integrates with the developed PCG system was approved.

4.2.4 Solution Implementation

Based on the functional requirements and the multiple workflows present in the solution design, we identified multiple elements that should be implemented, so that the system could achieve its designed goal.

One of these elements is a parser utility class, which given the transformed prompt received from the ChatGPT API can translate it into parameters with native values that the PCG system can be updated with. This component was implemented purely in C++. Visualizing the implemented methods of the parser utility class in Blueprint, the built-in visual scripting language of Unreal Engine, generates the nodes shown in Figure 44.

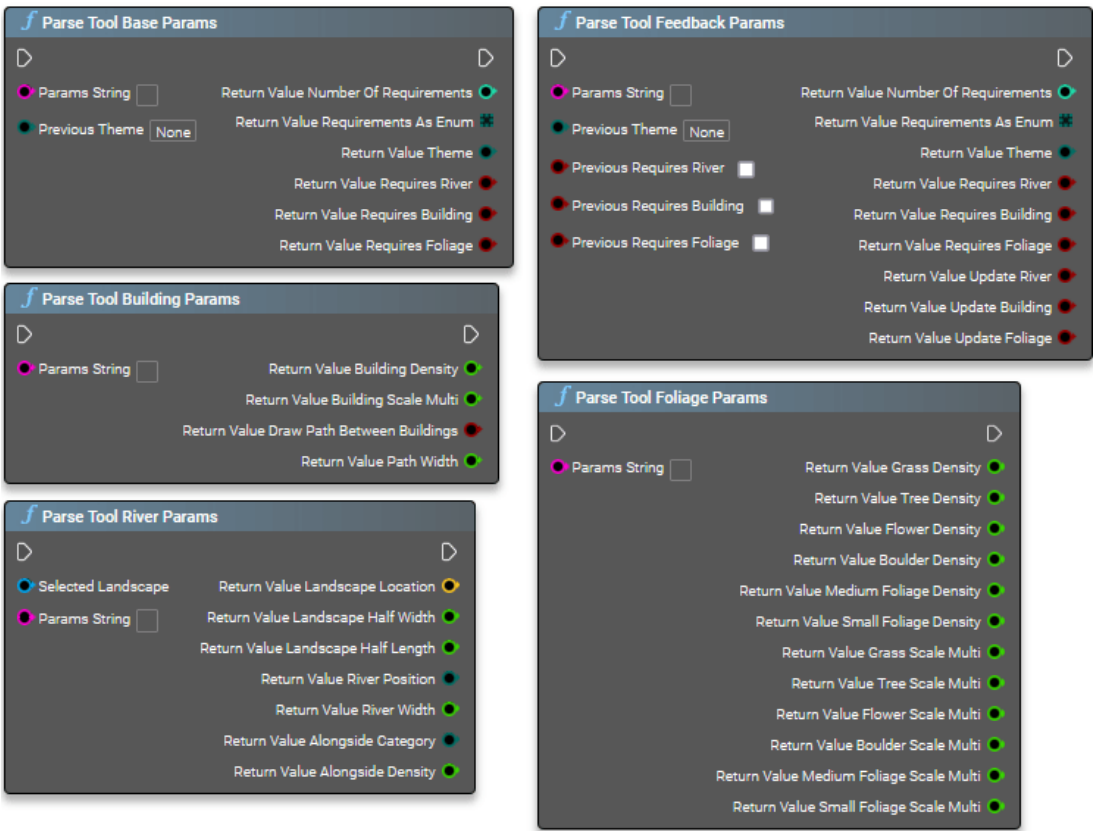

Figure 44: Implemented methods that allow parsing the transformed prompt

In figure 44, we can observe five different methods that can be used to parse the transformed prompt into different requirements and parameters. Table 9 was written to explain the functionality of each parsing method.

Parsing Method	Functionality
Parse Tool Base Params	Extracts the main requirements needed when generating an environment for the first time.
Parse Tool Feedback Params	Extracts the main requirements needed when iterating over an already generated environment.
Parse Tool Foliage Params	Extracts the parameters needed to customize foliage related elements in the generated environment.
Parse Tool Building Params	Extracts the parameters needed to customize buildings related elements in the generated environment.
Parse Tool River Params	Extracts the parameters needed to customize river related elements in the generated environment.

Table 9: Functionality of each method in the implemented parser utility class

The parser utility class and its methods were then used during the implementation of requests that the tool should execute. We will demonstrate the implementation of these requests one by one, through multiple tables that were created to describe the purpose of the request, the nature of the instruction message, the expected user prompt and API response. An example practical case is also displayed after each table, to show how the system processes a typical user prompt and generates the corresponding requirements/parameters.

- **Initial Requirements Request**

Purpose of the Request	Determine whether generating river, buildings and/or foliage is required or not.
Contents of the Instruction Message	- The response should be in a valid JSON format. - The theme should be selected based on the prompt from the list of supported environmental themes. - Explanation of what river, buildings, and foliage refer to. - Boolean values should determine whether an environmental aspect is to be generated or not.
Expected User Prompt	A prompt that asks for the generation of any kind of environment.
Expected API Response	Data containing the chosen environmental theme and a Boolean value for each environmental aspect (river, buildings, foliage), that determines whether it should be generated or not

Table 10: Information about the initial requirements request

Figure 45 is a screenshot from Unreal Engine's output log, showing a practical example of this request. We can see the used prompt, followed by the results of parsing the API response. In this example, we can see that the output requires the generation of foliage, since the prompt is asking for a forest.

```
LogBlueprintUserMessages: [EUW_PCG_ToolUI_C_3] Prompt: Generate a forest with sakura trees
LogTemp: Successfully parsed ToolBaseParams
LogTemp: Output:
{
    "theme": 4,
    "requiresRiver": false,
    "requiresBuilding": false,
    "requiresFoliage": true
}
```

Figure 45: Practical example of the initial requirements request

- **Feedback Requirements Request**

Purpose of the Request	Determine which environmental aspects (river, buildings, foliage) to adjust, and whether to generate one that does not exist yet.
Contents of the Instruction Message	- The response should be in a valid JSON format. - The current requirements of the generated environment. - The theme should be changed only if asked by feedback. - Explanation of what river, buildings, and foliage refer to. - Boolean values should determine whether an environmental aspect is to be generated or not. - Boolean values should determine whether an environmental aspect's elements are to be adjusted or not.
Expected User Prompt	Feedback about any element that belongs to the environmental aspects, regardless of if it is already generated or not.
Expected API Response	Data containing the updated environmental theme, if requested, and two Boolean values for each environmental aspect (river, buildings, foliage). The first one determines whether the aspect should be generated or not. The second one determines where the aspect's elements should be adjusted or not.

Table 11: Information about the feedback requirements request

Figure 46 is a screenshot from Unreal Engine's output log, showing a practical example of this request. We can see the used prompt, followed by the results of parsing the API response. In this example, we can see that the output requires the generation of a river, since the feedback is asking to add a river. We can also notice that the output wishes to update the river's parameters, since the environment did not have a river before, and the system needs to know what values to use.

```
LogBlueprintUserMessages: [EUW_PCG_ToolUI_C_12] Prompt: Add a river
LogTemp: Successfully parsed ToolFeedbackParams
LogTemp: Output:
{
    "theme": 4,
    "requiresRiver": true,
    "requiresBuilding": false,
    "requiresFoliage": true,
    "updateRiver": true,
    "updateBuilding": false,
    "updateFoliage": false
}
```

Figure 46: Practical example of the feedback requirements request

- **Foliage Parameters Request**

Purpose of the Request	Set the parameters of the foliage generation with custom values that reflect the prompt.
Contents of the Instruction Message	- The response should be in a valid JSON format. - Explanation of what the foliage categories are. These include grass, trees, flowers, boulders, medium-size and small-size. - For each foliage category, its density in the environment is represented by a float value. - For each foliage category, its size/scale in the environment is represented by a float value.
Expected User Prompt	A prompt that asks for the creation of foliage or adjustment of a foliage category.
Expected API Response	Data containing the density and scale values for every foliage category.

Table 12: Information about the foliage parameters request

Figure 47 is a screenshot from Unreal Engine's output log, showing a practical example of this request. We can see the used prompt, followed by the results of parsing the API response. In this example, we can see that the density of the trees has been assigned a value of zero, since the prompt is asking to remove all of the trees. The densities of other foliage categories have been left unchanged. We can also notice a small typo in the word "Remove", however the system still interprets the prompt accurately.

```
LogBlueprintUserMessages: [EUW_PCG_ToolUI_C_18] Prompt: Remvoe all the trees
LogTemp: Successfully parsed ToolFoliageParams
LogTemp: Output:
{
    "grassDensity": 0.6,
    "treeDensity": 0,
    "flowerDensity": 0.5,
    "boulderDensity": 0.2,
    "mediumFoliageDensity": 0.5,
    "smallFoliageDensity": 0.5,
    "grassScale": 1,
    "treeScale": 1.2,
    "flowerScale": 1.5,
    "boulderScale": 1.5,
    "mediumFoliageScale": 1,
    "smallFoliageScale": 0.8
}
```

Figure 47: Practical example of the foliage parameters request

- **Building Parameters Request**

Purpose of the Request	Set the parameters of the buildings generation with custom values that reflect the prompt.
Contents of the Instruction Message	- The response should be in a valid JSON format. - Explanation of what buildings refer to, and the possibility of drawing a path that passes from one to another. - Two float values should represent the density and scale/size of the buildings to generate - Boolean value should represent whether to draw the path or not. - One float value should represent the desired width of the path.
Expected User Prompt	A prompt that asks for the creation of buildings or adjustments related to them.
Expected API Response	Data containing the density and scale values of the buildings. It should also contain whether to draw a path between them or not, and with what width.

Table 13: Information about the building parameters request

Figure 48 is a screenshot from Unreal Engine's output log, showing a practical example of this request. We can see the used prompt, followed by the results of parsing the API response. In this example, we can see that the output requires drawing a path between the buildings, and has also increased the scale of buildings from the default value of 1.0 to 1.2.

```
LogBlueprintUserMessages: [EUW_PCG_ToolUI_C_18] Prompt: Add a path between buildings, and make them bigger
LogTemp: Successfully parsed ToolBuildingsParams
LogTemp: Output:
{
    "buildingDensity": 0.6,
    "buildingScale": 1.2,
    "drawPath": true,
    "pathWidth": 1.1
}
```

Figure 48: Practical example of the building parameters request

- **River Parameters Request**

Purpose of the Request	Set the parameters of the river generation with custom values that reflect the prompt.
Contents of the Instruction Message	- The response should be in a valid JSON format. - Explanation of how the river works, and the possible locations it could cover on the landscape. - Based on the feedback, the appropriate river position should be selected out of supported possibilities. - One float value should represent the desired width of the river. - Based on the feedback, we decide which natural elements are to be spawned alongside the river, if any. A float value should represent the density of these elements.
Expected User Prompt	A prompt that asks for the creation of a river or adjustments related to it.
Expected API Response	Data containing the chosen river location and width, as well as which elements to generate alongside the river and their density.

Table 14: Information about the river parameters request

Figure 49 is a screenshot from Unreal Engine's output log, showing a practical example of this request. We can see the used prompt, followed by the results of parsing the API response. In this example, we can see that that the value of the river width has been changed from the default value of 1.0 to 1.5. We can also notice that an index representing the "Right" river position has been selected.

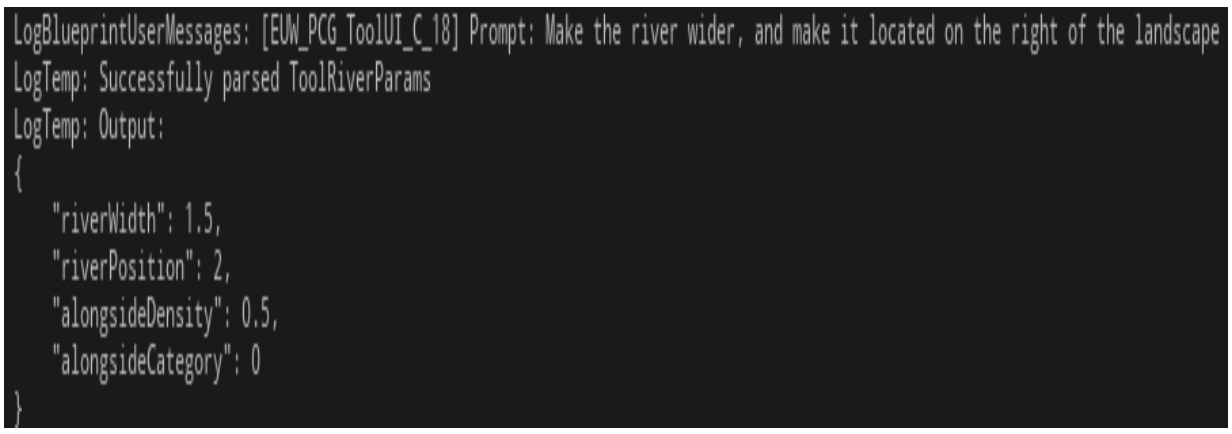

Figure 49: Practical example of the river parameters request

4.2.5 Solution Evaluation

The solution implementation was evaluated by the stakeholders, who assessed whether it matched the designed requirements and workflows. Indeed, they approved that the designed concepts were respected during the development phase. The technical lead of the company also suggested carrying out tests on the solution implementation, to further assess that all of its functionalities are working correctly. During this testing phase, two types of tests were conducted: Functional & Integration testing.

- **Functional Testing**

Functional testing is a type of software testing that verifies that each function of a system works as intended, by comparing the expected output with the actual output [36]. Table 15 presents each functional test, discussing its description, its necessary preconditions, its test data, its expected output, and its actual output.

Description	Precondition	Test Data	Expected Output	Actual Output
Parse initial requirements	An environment is not generated	Valid data in JSON format	The associated requirements	The associated requirements
Parse initial requirements	An environment is not generated	Invalid data	Default requirements	Default requirements
Parse initial requirements	An environment is not generated	Valid data, but the theme param is a string instead of an enumeration	The associated requirements	The associated requirements
Parse feedback requirements	An environment is generated	Valid data in JSON format	The associated requirements	The associated requirements
Parse feedback requirements	An environment is generated	Invalid data	Default requirements	Default requirements
Parse feedback requirements	An environment is generated	Valid data, but the theme param is a string instead of an enumeration	The associated requirements	The associated requirements
Parse river parameters	A river is generated	Valid data in JSON format	Associated river parameters	Associated river parameters

Parse river parameters	A river is generated	Invalid data	Default river parameters	Default river parameters
Parse river parameters	A river is generated	Valid data, but the width of the river is negative	Associated river parameters, with the width modified to 0.5	Associated river parameters, with the width modified to 0.5
Parse river parameters	A river is generated	Valid data, but the width of the river is too big	Associated river parameters, with the width modified to 2	Associated river parameters, with the width modified to 2
Parse river parameters	A river is generated	Valid data, but the river position enumeration does not exist	Associated river parameters, with the default river position	Associated river parameters, with the default river position
Parse buildings parameters	Buildings are generated	Valid data in JSON format	Associated buildings parameters	Associated buildings parameters
Parse building parameters	Buildings are generated	Valid data, but the scale of the buildings is negative	Associated buildings parameters, with the scale modified to 0.1	Associated buildings parameters, with the scale modified to 0.1
Parse buildings parameters	Buildings are generated	Valid data, but the width of the path is negative	Associated buildings parameters, with the width modified to 0.5	Associated buildings parameters, with the width modified to 0.5
Parse buildings parameters	Buildings are generated	Valid data, but the width of the path is too big	Associated buildings parameters, with the width modified to 2	Associated buildings parameters, with the width modified to 2

Parse buildings parameters	Buildings are generated	Invalid data	Default buildings parameters	Default buildings parameters
Parse building parameters	Buildings are generated	Valid data, but the density of the buildings is negative	Associated buildings parameters, with the density modified to 0	Associated buildings parameters, with the density modified to 0
Parse foliage parameters	Foliage is generated	Valid data in JSON format	Associated foliage parameters	Associated foliage parameters
Parse foliage parameters	Foliage is generated	Invalid data	Default foliage parameters	Default foliage parameters
Parse foliage parameters	Foliage is generated	Valid data, but the scale of one foliage category is negative	Associated foliage parameters, with the wrong scale modified to 0.1	Associated foliage parameters, with the wrong scale modified to 0.1
Parse building parameters	Buildings are generated	Valid data, but the density of one foliage category is negative	Associated foliage parameters, with its density modified to 0	Associated foliage parameters, with its density modified to 0

Table 15: Functional testing for the prompt interpretation system

- **Integration Testing**

Integration testing is a type of software testing that verifies that different systems and components are working together as expected [36].

In this case, we are testing the integration of the prompt interpretation system with the already implemented PCG system. Table 16 presents each test, discussing its selected test input, the expected result and where the test passed or not.

Test Input	Expected Result	Pass?
Prompt requesting a desert environment with no buildings and no river	Environment is generated with the desert theme being selected, and no buildings or river were created.	Yes
Feedback requesting adding a river to an existing generated environment	Environment is updated to show a river with the rest of the already generated elements.	Yes
Feedback requesting to add more trees and to remove all the flowers	The foliage generation in the environment is updated to remove flower generation and increase the density of the trees.	Yes
Feedback requesting to change the current theme of the environment to the Japanese one	Environment is updated to show the 3D models of the Japanese theme instead, while keeping the same generation results.	Yes
Prompt which is not related to environment generation	No environment generation happens.	Yes
Prompt requesting the generation of an environment that is not supported	Environment with a theme that feels close to the requested one is generated.	Yes
Feedback which is not related to environment generation	No adjustments to the generated environments happen.	Yes
Feedback requesting the removal of cactuses from a generated canyon	The system realizes that cactuses are considered a medium-size foliage, and the environment is updated to set its density to 0.	Yes

Table 16: Integration tests verifying the integration of prompt interpretation with the PCG system

As the solution implementation has passed all the functional and integration tests, it was ultimately evaluated and approved.

4.3. Iteration 3: Tool's UI/UX Design and Implementation

The third iteration's objective is to design and implement a user-friendly interface for the tool, and ensure that it is intuitive and that it blends nicely with the rest of Unreal Engine's interfaces, to offer a great user experience. This interface will serve as the primary interaction point between the users of the tool and the underlying systems developed in the previous iterations.

4.3.1 Problem Investigation

In order to generate 3D environments based on user specifications, we must design and implement a UI that allows users to easily interact with the tool. The absence of a well-designed interface would make the tool too difficult to use, which in turn will reduce its overall value, regardless of its other systems' capabilities. During the implementation phase, several issues could arise if insufficient research is done beforehand about the use cases of the tool and the UI elements that enable them. For instance, developers would end up creating a disjointed interface which leads to poor usability and frustration. There is also the risk of scope creep, especially when the use cases of the tool are not defined beforehand, which could make the UI bloated with unnecessary functionalities. This is why it is essential to have a comprehensive understanding of the users' requirements and the specific tasks they need to accomplish with the tool. Finally, the UI design must be consistent with Unreal Engine's existing interfaces to provide a cohesive user experience.

4.3.2 Solution Design

First, to ensure that the tool meets the diverse needs of its users and offers a good user experience, and to understand which functionalities should be implemented in its UI, we have to define multiple user personas. These personas represent a portion of the tool's target customers/users, each with different goals and frustrations during the development process. Tables 17, 18 and 19 detail the characteristics, goals and frustrations of each user persona that should be considered during the UI/UX design of the tool.

Picture	
Name	Emily
Occupation	Game Developer
Experience Level	Intermediate
Background	Emily has a solid understanding of game design and has experience using Unreal Engine. She often works alone or with a small team and prefers tools that are easy to integrate into her development workflow.
Goals	- Attempt to work on a bigger scale game project. - Create visually appealing 3D environments for her games. - Iterate on the environments based on player feedback.
Frustrations	- Spending too much time on creating an environment. - Using interfaces that are cluttered and not intuitive.

Table 17: Emily - User persona for the tool's UX design

Picture	
Name	Dan
Occupation	3D Artist
Experience Level	Advanced
Background	Dan is a skilled 3D artist working at a mid-sized studio that specializes in game and short movie development. He is comfortable using features of Unreal Engine and frequently works with his team to put together realistic environments.

Goals	- Create highly detailed and realistic 3D environments. - Fine-tune and adjust environment elements to achieve a better look. - Create multiple versions of the environments.
Frustrations	- Not being able to re-use an environment that he created. - Not seeing the changes he makes to the environment in real-time.

Table 18: Dan - User persona for the tool's UX design

Picture	
Name	Alex
Occupation	Software Engineering Student
Experience Level	Beginner
Background	Alex is a final year software engineering student with a passion for games. She is relatively new to Unreal Engine and is eager to learn and experiment with new tools that can help bring her ideas to life.
Goals	- Create impressive projects for her portfolio - Develop her skills, especially programming, using Unreal Engine. - Experiment with 3D development and design.
Frustrations	- Steep learning curves of development tools. - Interfaces that are not consistent with Unreal Engine's UI.

Table 19: Alex - User persona for the tool's UX design

Next, we will identify the functional requirements that will guide the implementation phase while considering the proposed user personas. Table 20 outlines the different functional requirements of the tool's UI along with their description.

Functional Requirement	Description
Select the landscape to generate on	The user must be able to select a landscape that is going to be used as a basis for the environment generation.
Change the landscape to generate on	The user must be able to change the selected landscape.
Generate a 3D environment based on a prompt	The user must be able to generate a 3D environment based on a prompt that the user writes.
Iterate over the generated 3D environment using a prompt that has feedback	The user must be able to iterate over the generated environment by providing feedback through a prompt that the user writes.
Control the change magnitude of the feedback	The user must be able to affect the degree at which the environment is adjusted using a change magnitude value.
Revert the changes caused by the latest feedback	The user must be able to revert the changes made by the latest feedback if the user is not satisfied with it.
Change lighting settings	The user must be able to change the lighting settings to view the generated environment under different sunlight.
Generate many variations of the generated environment	The user must be able to generate multiple variations of the same environment using the same parameters and requirements.
Remove the generated environment	The user must be able to remove the currently generated environment and try again with a new prompt.
Save the generated environment	The user must be able to save the generated environment and store it in a reusable file.
Give the saved generation a custom name and prefix	The user must be able to give a prefix and a name to the saved reusable file.
Choose what to do with the generated environment after saving it.	The user must be able to choose whether to remove the currently generated environment or keep working with it after saving it in a reusable file.

Table 20: Functional requirements for the tool's UI

To ensure that the tool's UI is as effective and user-friendly as possible, we will define quality attributes that must be achieved when implementing the interface. Table 21 lists the different quality attributes that we decided on.

Quality Attribute	Description
Usability	The tool's UI must be easy to learn and use for users with different experience levels. It should help them accomplish their goals with minimum effort and confusion.
Responsiveness	The tool's UI should have minimal delay between user actions and system responses, and should provide clear visual feedback on what is happening.
Consistency	The tool's UI design, elements, colors, and styles should be consistent with Unreal Engine's existing interfaces to provide a cohesive user experience.

Table 21: Quality attributes of the tool's UI

Based on the functional requirements, a use case diagram was illustrated to describe the interactions between the user and the tool's UI.

Figure 50: Use case diagram of the tool's UI system

The use case diagram shown in figure 50 provides a clear representation of the interactions of the Unreal Engine user and the various functionalities of the tool's UI system.

The primary use case is "Create 3D environment", which is only possible after selecting a landscape to generate the environment on, and writing a prompt that explains what kind of environment the user wants. This main case is extended by additional use cases, which are optional and allow the user to perform many actions on the generated environment.

These cases include "Adjust the created environment", where the user can provide feedback to iterate on the environment, and "Create another variant of the environment", where the user can explore other generation possibilities that use the same parameters and requirements. The user can also choose to "Save the created environment", which allows them to store the generated environment in a reusable file with a custom name.

Another separate use case is "Change sunlight", which allows the user to experiment with different lighting settings. Overall, this diagram shows the range of user interactions required to create, modify, and manage 3D environments using the tool's UI system.

Next, we will create three sequence diagrams that provide a step-by-step explanation of the processes behind the use cases "Create 3D environment", "Adjust the created environment", and "Save the created environment".

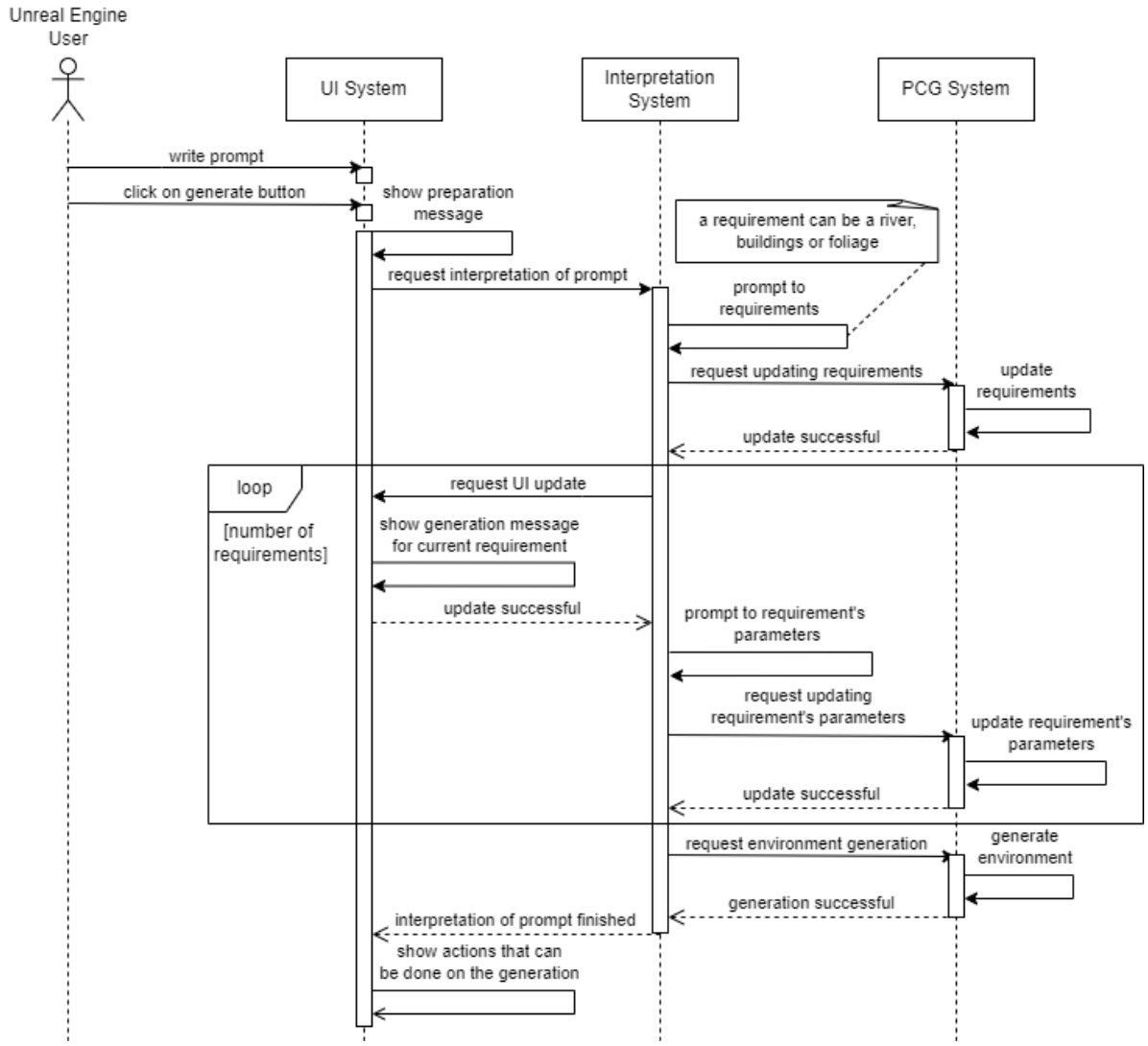

Figure 51: Sequence diagram for "Create 3D environment"

The sequence diagram shown in figure 51 belongs to the process of creating a 3D environment. It begins with the user writing a prompt about the desired environment and clicking on the "Generate" button. This triggers the UI system to display a preparation message such as "Preparing…" and to request the interpretation of the prompt. The interpretation system then processes the prompt and transforms it into requirements which are communicated to the PCG system. This lets it know which environmental aspects from river, buildings, and/or foliage we need to generate. Next, the interpretation process continues in a loop for the total number of requirements. For each requirement, we update its parameters in the PCG system and reflect the changes through the UI system using generation messages such as "Task 1 out of 3…". Once that is finished, the 3D environment is generated, and the UI system displays the many actions that can be performed on the created environment.

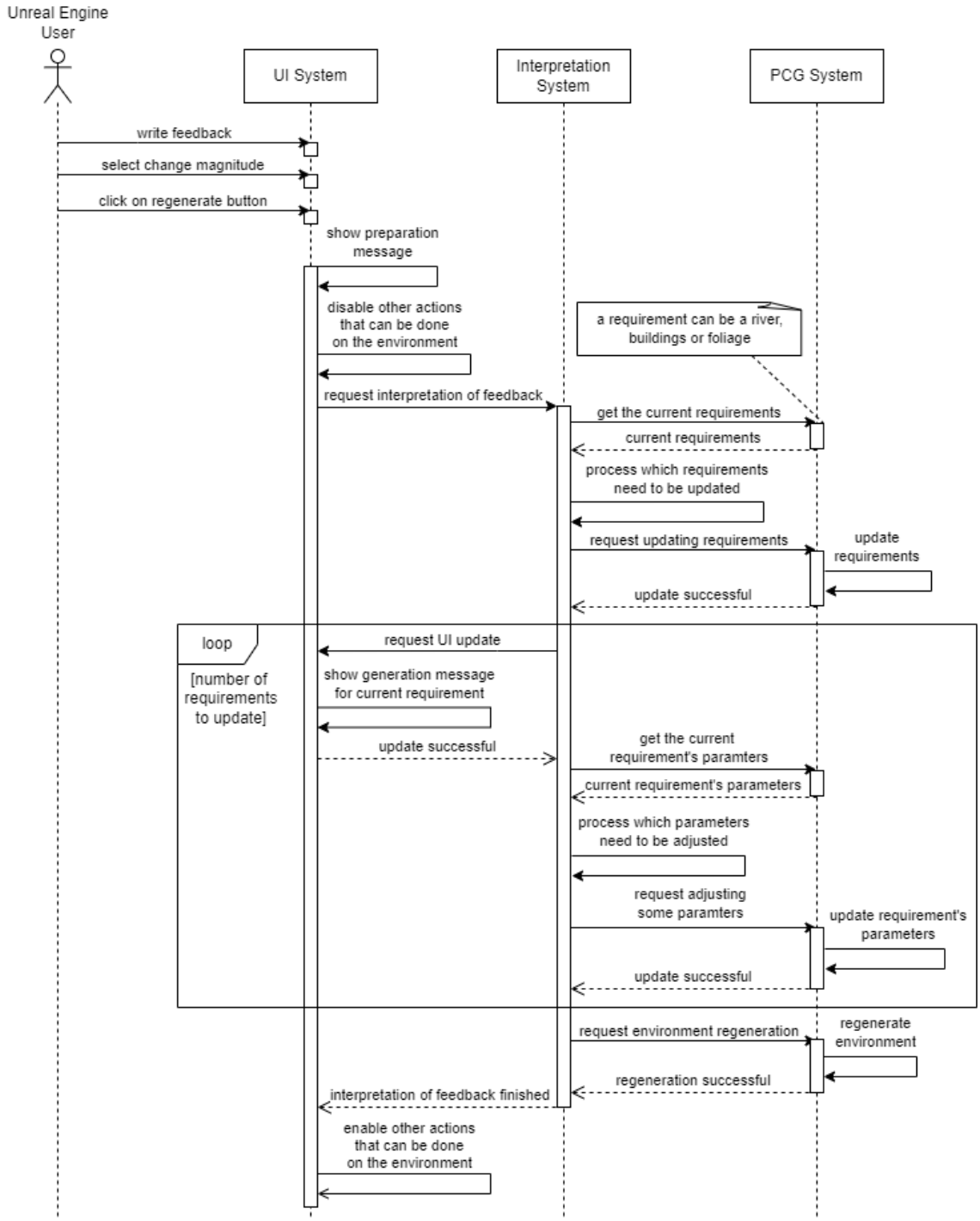

Figure 52: Sequence diagram for "Adjust the created environment"

The sequence diagram shown in figure 52 belongs to the process of adjusting the previously created environment. It begins with the user writing feedback about the desired adjustments, choosing a change magnitude (e.g.: small, normal, large) and clicking on the "Regenerate" button. This triggers the UI system to display a preparation message such as "Preparing…" and to disable the other actions that can be executed on the environment, so that no bugs happen. The interpretation system then retrieves the current requirements from the PCG system to formulate the context for the feedback. It then processes the feedback and transforms it

into requirements that need to be updated. Next, the interpretation process continues in a loop for only the requirements that need to be updated. For each one, we adjust its parameters in the PCG system based on the feedback and reflect the changes through the UI system using generation messages such as "Task 1 out of 2…". Once that is finished, the 3D environment is regenerated with the new requirements and parameters, and the UI system enables back the many actions that can be performed on the generated environment.

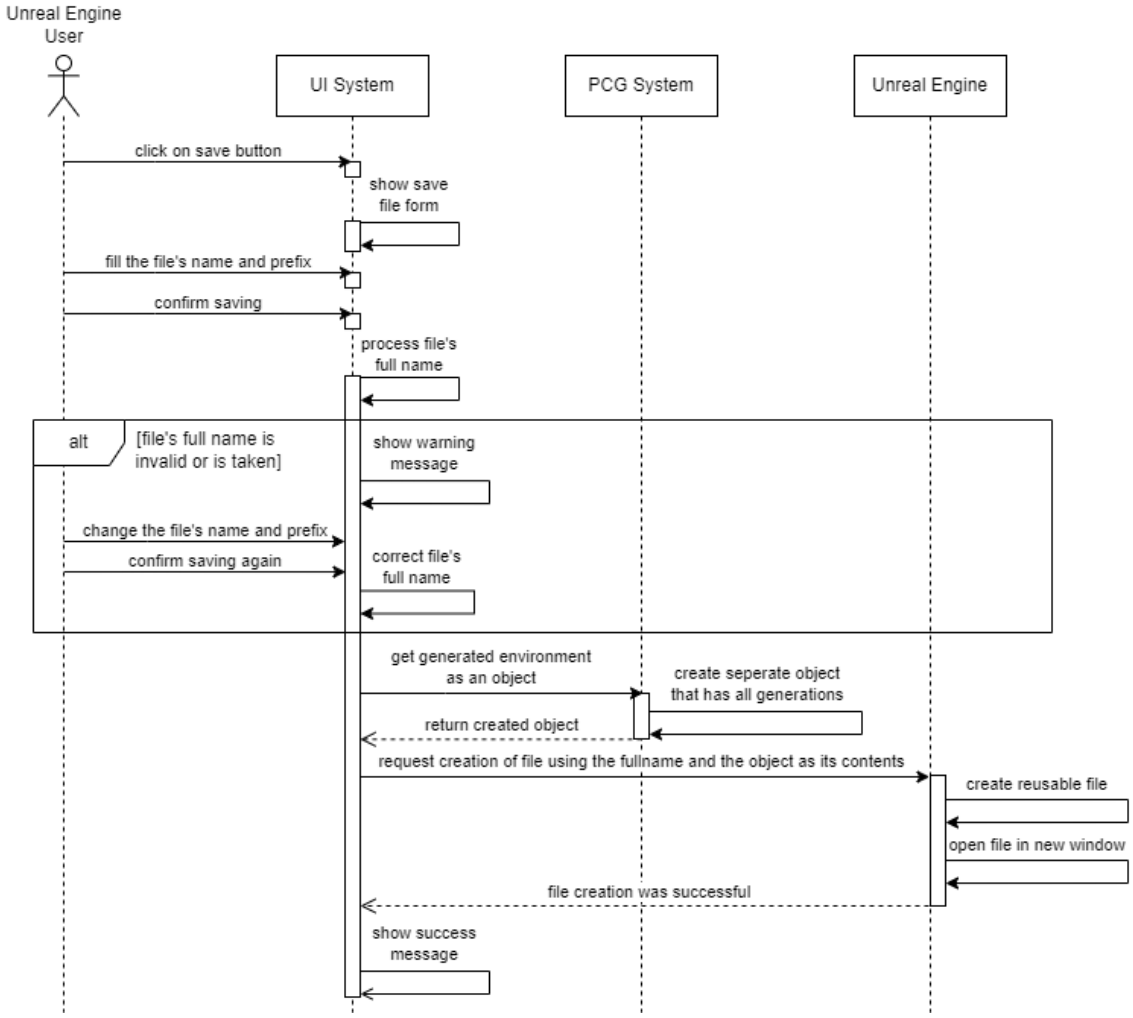

Figure 53: Sequence diagram for "Save the created environment"

The sequence diagram shown in figure 53 belongs to the process of saving the previously created environment into a reusable file. It begins with the clicking on the "Save Generation" button, which triggers the UI system to show a form that needs to be filled. Once the user fills the file's name and prefix and confirms saving, the UI system checks the validity of the file's full name. If it is invalid or taken, the user is prompted to change it through a warning message. Next, the PCG system creates an object, which contains all the generated elements of the environment. Using Unreal Engine's built-in functionalities, a reusable file is created base of the chosen file's full name and the created object. The engine then displays the contents of the file in a new window. Finally, the UI system informs the user about the success of saving the file.

4.3.3 Design Validation

The process of design validation was carried out in few meetings with the stakeholders. During this meeting, the workflows of the tool's UI were discussed, ensuring their technical implementation is possible and that they align with the UX we are trying to achieve. The technical lead of the company also provided documentation that teaches the basics of Unreal Motion Graphics, which is the UI built-in framework used in Unreal Engine for creation of visual interfaces. Moreover, a member of the team with the occupation of designer reviewed the solution design and provided useful advice to how to approach the design and implementation of user interfaces in Unreal Engine. Ultimately, the stakeholders approved the proposed design.

4.3.4 Solution Implementation

In this section, we will present the implementation of the tool's UI and its various functionalities. We will showcase the visual aspect of the UI, its elements, different views, and panels. The tool's UI was developed using Unreal Motion Graphics, which is a built-in tool in Unreal Engine that allows developers to create and manage visual interfaces. Before we dive into the implemented visual interfaces and their elements, it is important to understand the context of Unreal Engine's existing UI and how the tool is integrated within it.

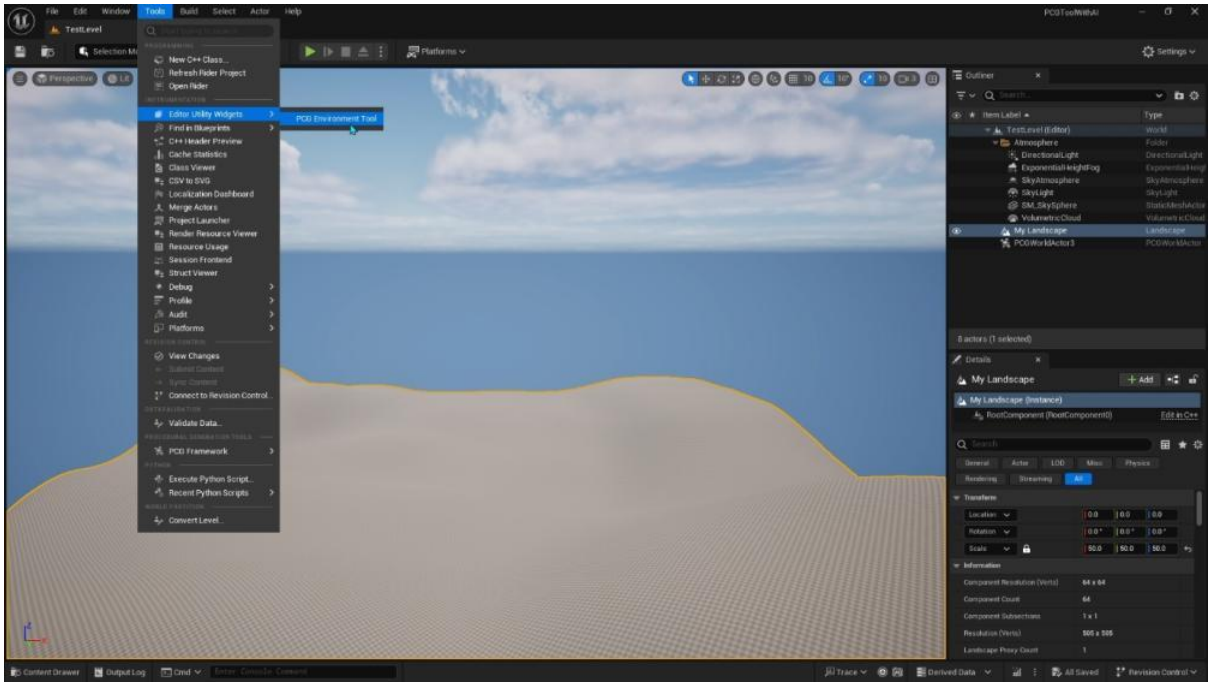

Figure 54: Unreal Engine's UI, layout, and the integration of the tool's UI

Figure 54 captures the standard layout of Unreal Engine's UI, which features a default dark theme with shades of grey, black, and blue. Key elements of the engine's UI include the

hierarchy viewer on the top right, the details inspector on the bottom right, and a central viewport for the 3D workspace. At the top, we can see a menu bar with many sections, including one titled "Tools". Through this section, we can access the tool's UI.

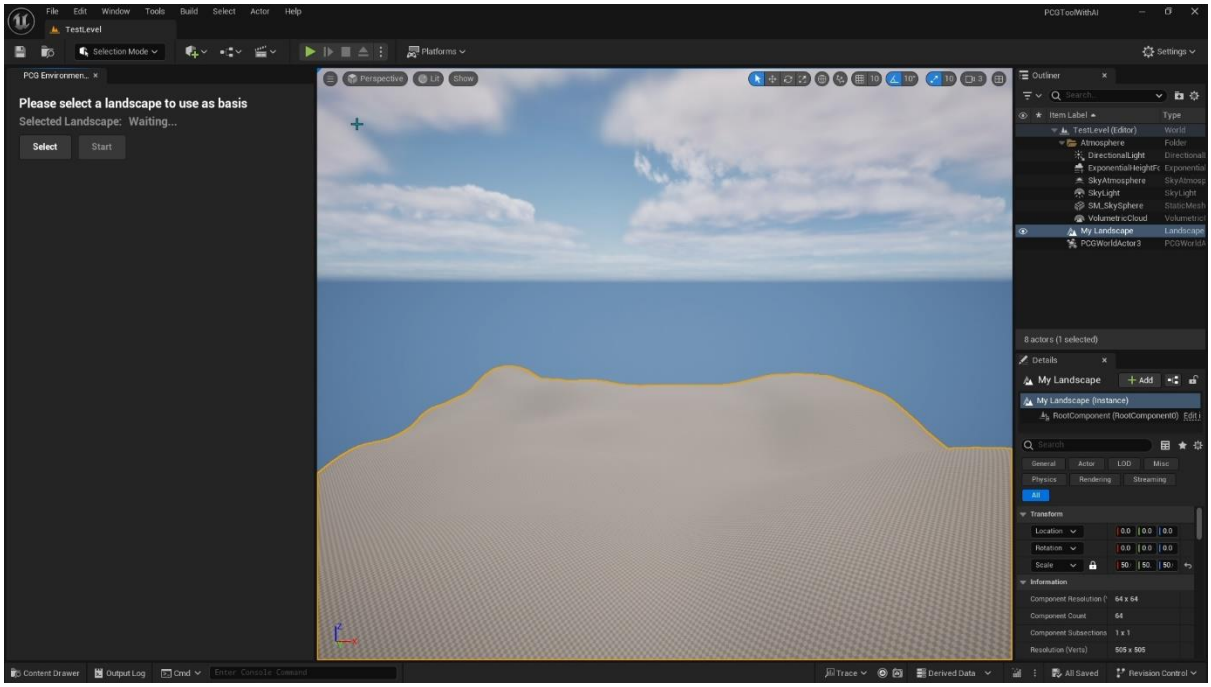

Figure 55: Unreal Engine's layout with the tool's UI open

When we open the tool's UI, as shown in figure 55, it smoothly integrates into Unreal Engine's existing layout. The tool's panel with its initial view occupies the left part of the layout. We can see that it maintains the same dark theme and colors. Its elements, such as button and text boxes, share the same shape and behavior as those found within the engine. That being said, we will now discuss the different implemented parts of the tool's UI.

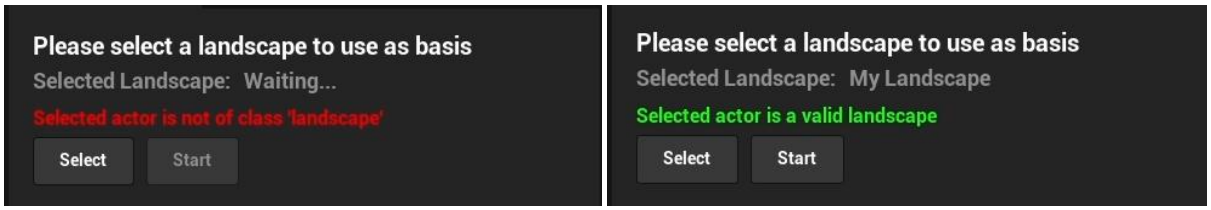

Figure 56: UI panel for landscape selection

This UI panel shown in figure 56 belongs to the "Select landscape" use case. The picture on the left is what the user sees when they select an invalid object instead of a landscape, while the picture on the right is what the user sees when the selection is valid. In both cases, we can see a text block titled "Selected Landscape:" which shows the display name of the selected landscape and a "Select" button, which performs that action.

In the first picture, we can notice a red error message that informs the user that the selection was invalid. We can also notice that the "Start" button is disabled. In the second picture, we notice a green message that informs the user that the selection was successful. The "Start" button is enabled, and allows the user to proceed to the next UI panel.

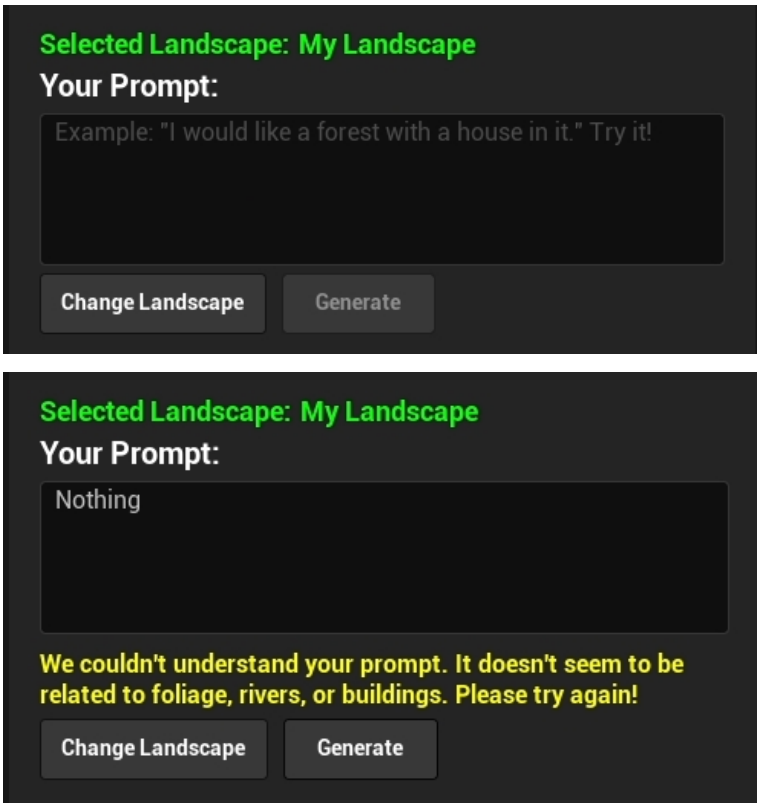

Figure 57: UI panel for the initial prompt submission and environment creation

This UI panel shown in figure 57 belongs to use cases such as "Create 3D environment" and "Change landscape". The first picture displays what the user would see initially after clicking on the "Start" button in the previous UI panel. We can see that the selected landscape's name is further highlighted, and we can see a multi-line text box where the user can write the desired prompt. The "Generate" button that executes the environment creation process is disabled when there is no written prompt in the text box. The second picture shows what happens when the user writes an invalid prompt. We can notice a yellow warning message, which informs the user that the submitted prompt was not properly interpreted due to reasons such as the prompt not relating to environment creation.

Figure 58: Different states of buttons responsible of generation

In the case of a successful prompt, the tool starts the generation process. This is reflected in the UI through the button states shown in figure 58. The first state is the default state of the button, which shows its name. The second state is displayed when the tool is still interpreting the requirements of the environment generation from the submitted prompt. The third and fourth state are displayed when the tool is interpreting the parameters of the environmental aspects to generate, such as the parameters of river, buildings, and/or foliage. The text of the button in this case implies the progress of the process. When all tasks are finished, the environment is successfully generated.

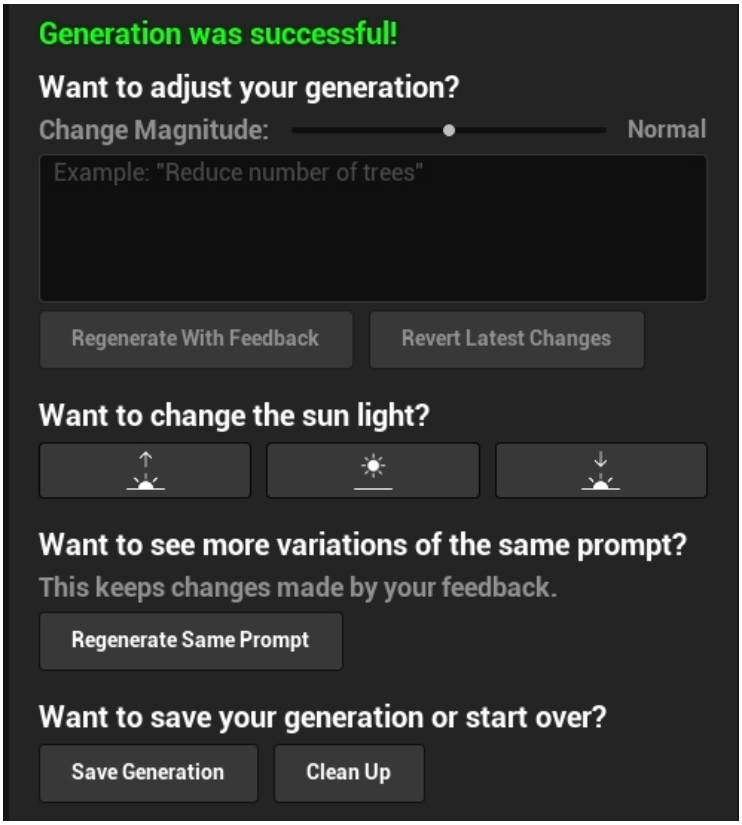

Figure 59: UI panel for actions that can be performed on the generated environment

This UI panel shown in figure 59 belongs to multiple use cases that concern the actions that can be performed on the generated environment. Starting from the top, we can see a section titled "Want to adjust your generation?". This section allows users to provide feedback about the generated environment, and modify it based on the user needs. We can notice two important elements of this section. The first element is a value slider that represents the value of the change magnitude. This can take five values: "Small", "Medium", "Normal", "Large"

and “Extreme”. The second element is the multi-line text box where that user can write feedback. We can also notice two buttons under this section. The first button called “Regenerate With Feedback” executes the process of adjusting the environment based on the feedback. While the second button, called “Revert Latest Changes”, cancels the most recent adjustments to the environment.

Next, we can see a section titled “Want to change the sun light?”. This section allows users to view the generated environment under multiple light settings. We can notice three buttons with different icons. Starting from the left, the icons represent sunrise, noon, and sunset, respectively.

The following section is titled “Want to see more variations of the same prompt?” and allows users to generate a new variation of the environment based on the same prompt and its interpreted requirements and parameters. This is done through the button called “Regenerate Same Prompt”.

The final section is titled “Want to save your generation or start over?” and allows users to either save the generated environment in a reusable file, or completely remove the generated environment and start over. In the first case, clicking the “Save Generation” button will switch the UI to a panel where the user can fill the save file’s information. In the second case, clicking the “Clean Up” button will switch the UI to the previous panel, where the user can write the initial prompt to start a new generation.

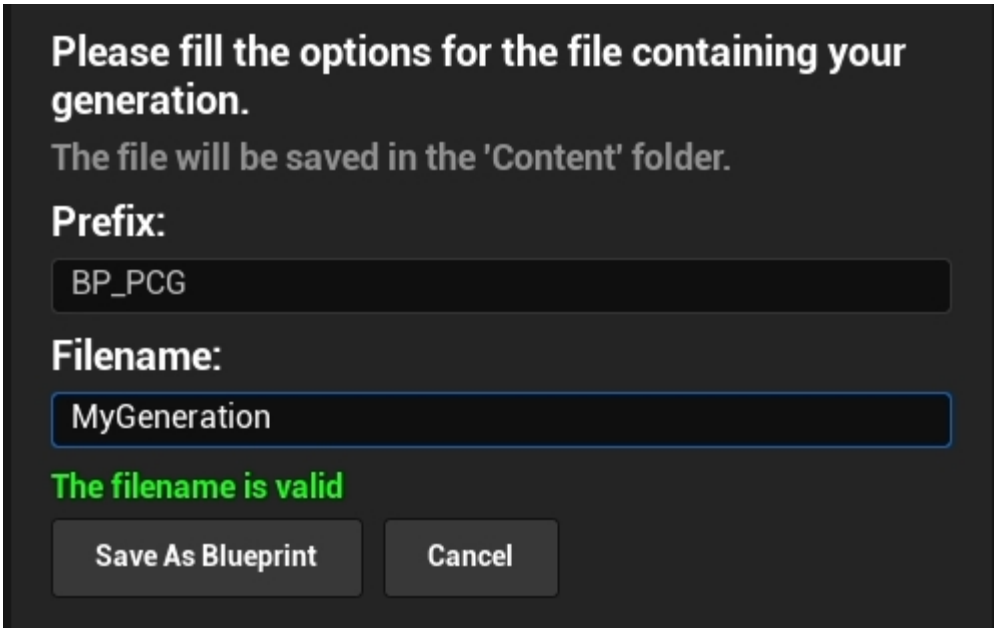

Figure 60: UI panel for filling save file's information and confirming save

The UI panel shown in figure 60 belongs to the "Save generation" use case. In this panel, the user can fill in the information of the save file, which consists of the prefix and the name required for the file. In Unreal Engine, it is typical to have naming conventions for files. A file's full name would usually start with a prefix, followed by a specific name. This can be achieved through the two text boxes present in the UI panel. Once the fields are filled, the user can confirm saving the generation as a reusable file.

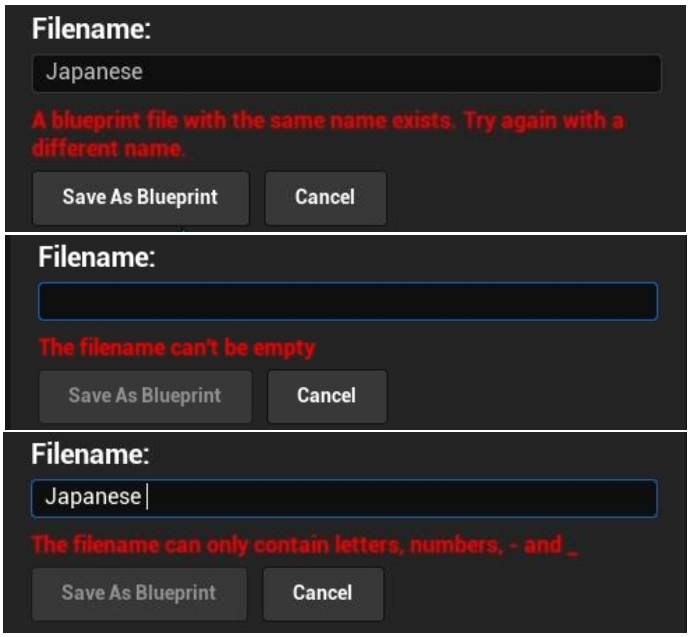

Figure 61: Error messages that could appear when saving file

In some cases, the creation of the file will not be allowed. These include saving a file with a full name that is already taken by another file, leaving the filename field empty, and using unsupported characters in the latter. Figure 61 demonstrates the error messages that could appear when one of these cases happens.

4.3.5 Solution Evaluation

The process of solution evaluation was conducted over two key steps to ensure that the implementation of the solution matches the proposed design.

The first step was holding a meeting with the stakeholders, where the implemented UI panels and elements were discussed in detail. During the meeting, I have also demonstrated a demo, which consisted of a practical use case scenario of a typical user interacting with the tool's UI to create 3D environments. The stakeholders reviewed the usage of the tool's UI within Unreal Engine and the execution of its functionalities and were satisfied with the implementation.

The second step was to evaluate the tool's UI and UX further by gathering feedback from the potential target audience of the tool. To achieve this, I prepared and shared a survey with a diverse group of individuals with different levels of expertise. All the participants had interest in 3D environment creation, whether as a job, hobby, or academic interest. The survey consisted of a video showcasing the tool's UI in action, followed by multiple grading questions on a scale of 1 to 5, with 1 being a bad result, and 5 being an excellent result. All of the questions were related to the quality attributes that were defined in the solution design: usability, responsiveness, and consistency. Overall, the survey received 21 responses. Table 22 presents each question of the survey with its grading metric and its average grade from the 21 responses. Since the results of the survey indicated that the majority of participants believed the UI/UX of the tool was good, the solution was positively evaluated and approved.

Question	Related Quality Attribute	Average Grade
How well did you understand what the tool was doing?	Usability	4.85
How would you rate the ease of use of the tool?	Usability	4.67
How simple and clean was the tool's UI?	Usability	4.38
Was the language used in the interfaces understandable and to the point?	Usability	4.57
How would you rate the user experience?	Usability	4.67
How would you rate the tool's response time to actions requested by the user?	Responsiveness	4.47
How would you rate the smoothness of the interactions?	Responsiveness	4.38
How would you rate the quality of real-time feedback (warnings, confirmations, info about generation, other info...) provided by the tool?	Responsiveness	4.14
How would you rate the consistency of the tool's UI colors and themes compared to those of Unreal Engine?	Consistency	4.85
How would you rate the consistency of the tool's UI elements and their styles?	Consistency	4.67

Table 22: Survey questions for UI/UX evaluation

4.4. Conclusion

In this section, we discussed every step of the three iterations that were performed during the development of this project. In the next section, we will use the gathered findings to answer the proposed research questions of the study.

5. RESULTS AND FINDINGS

In this section, we will answer the proposed research questions one by one by discussing the findings of the study related to each research question, respectively.

5.1. Research Question 1

What are the advantages and drawbacks of relying on procedural generation for 3D environment creation?

To answer this research question, we propose an answer composed of three different sections. The first two major sections focus on discussing the different advantages and drawbacks of using procedural generation as a medium to create 3D environments. The third section is a concise interpretation which discusses the understanding gained from analyzing both advantages and disadvantages.

5.1.1 Advantages

Procedural generation is a wide subject with many techniques that can offer significant benefits to development companies when implemented, especially in the gaming industry, where the demands for innovative and creative games is ever increasing, and expectations of the players are high. Throughout the study, research was conducted to identify the key advantages of relying on procedural generation for creating a 3D environment, an integral part of most games and movies. Some of these include:

- **Time Efficiency:** Procedural generation can speed up the development process of any product that demands the creation of a 3D environment, by automating the generation of the layers that compose it. It reduces the time required to manually put together an environment and thus allows team members to focus more on other aspects of their product.

- **Cost Reduction:** Procedural generation can reduce the need for larger teams of artists and designers, allowing companies to save financial resources and allocate them elsewhere. This offers new opportunities to development teams with limited budget such as indie (independent) developers and small studios when it comes to creating large scale and believable 3D environments. It is also important to mention that the reduction in development time caused by procedural generation translates to lower production costs.

- **Replayability:** When it comes to the gaming industry, procedural generation can provide multiple variations of a 3D environment based on the same requirements and parameters due to its randomness in decision-making. This allows gaming companies to provide unique experiences with every playthrough, encouraging players to engage with the game for longer periods of time.

- **Scalability:** Procedural generation allows of the creation of large-scale environments that otherwise would be impractical to design manually due to its size and time needed to create. The ability to generate such a large environment programmatically could benefit many industries that rely on 3D environments, such as the gaming and movies industry. When it comes to game development, this could enhance making open-world games, which is a trending genre.

- **Adjustability:** Procedural generation allows real-time and dynamic changes to the created 3D environment. This way, environments that are created from the same procedural method can be quickly modified and regenerated to address issues or feedback from the customers without the need to manually adjust them one by one.

5.1.2 Drawbacks

While procedural generation can offer development companies that rely on it considerable advantages, it also comes with its own list of drawbacks and pitfalls that need to be considered carefully. Throughout the study, research was conducted to identify the challenging drawbacks that come with relying on procedural generation for creating a 3D environment. Some of these include:

- **Challenging Implementation:** Designing and implementing reliable procedural generation techniques and methods to generate 3D environments is a challenging task and would require multiple iterations to get done right. This could cause a problem to teams that lack members from a technical background or individuals that do not have any technical expertise.

- **Limited Uniqueness:** While procedural generation can create many variations of an environment, there is a risk that the generated environments feel repetitive over time, especially if the same 3D models are used for the environment's elements. If this is not addressed during the development of a game, this could lead to a less engaging experience for the players.

- **Unexpected Interactions:** Since procedural generation is based algorithms and techniques that rely on random decision-making, there is a risk that unexpected interactions happen between the randomly generated elements of the environment. This could lead to visual errors or bugs that need to be addressed during development.

5.1.3 Interpretation

There is no doubt that relying on procedural generation when creating 3D environments can provide many benefits to the development team that adopted it. However, it also comes with drawbacks and pitfalls that should be addressed during the design and implementation phases. Based on the understanding gathered throughout the study related to this research question, it is recommended to mix both procedural generation and manual design & control when creating 3D environments. Relying only on one of them would make their respective drawbacks more challenging to address, but combining both as a hybrid approach would result in the best of the two worlds.

5.2. Research Question 2

What are the key challenges in procedurally generating 3D environments that align with user expectations?

Procedural generation can be a powerful toolset when used right by developers to generate a vast amount of content and explore many new possibilities. However, when aiming to produce content that aligns with user expectations, specifically 3D environments, it is crucial to understand and analyze the needs and impressions of the target audience. Throughout this study, additional research was conducted to identify the key challenges of procedurally generating 3D environments that align with user expectations. These challenges need to be considered by development teams and include the following key points:

- **Believability:** It is important to generate 3D environments that appear coherent and realistic to the context of the generation. For instance, an imaginary or cartoonish environment should have consistent artistic direction and fantastical elements that fit together seamlessly, while a simulation of a real-life region must accurately reflect its geographical, architectural, and cultural aspects. In all cases, the challenge that must be addressed is to generate believable environments that do not disrupt the user's immersion.

- **Visual Consistency:** It is important to ensure that the generated 3D environments meet specific visual quality and standards depending on the project. Users should not be able to identify inconsistencies in the generated elements regardless of their position, rotation, size, or textures. It is also important to handle all kinds of visual errors in the generated environment, such as elements overlapping or clipping.

- **Diversity and Uniqueness:** When implementing the procedural generation methods that are going to create 3D environments, developers must consider that the used techniques and algorithms provide sufficient variety to the generated environment. The created environments must appear unique from other variations even slightly, so that users do not get the impression that the results of the generation are repetitive.

- **Audience and Project Needs:** It is important to conduct enough analysis about the needs of the target audience prior to the development of the procedural generation system responsible for creating 3D environments. Without understanding the general preferences and behaviors of the users, it would be difficult to generate environments that satisfy their expectations. For example, a challenging game targeting hardcore players might need complex environments, while a casual game might require more accessible and visually appealing ones. It is also important to limit the scope of procedural generation to the goal of the project. For instance, different games, such as an open-world adventure game versus a linear narrative game, have unique requirements that the procedural generation system must address to ensure a successful and enjoyable user experience.

5.3. Research Question 3

What features and requirements of the tool would be crucial to help streamline 3D environment creation and save development time?

The integration and usage of tools has become an essential part of the modern development workflow, as it improves productivity and efficiency of the design and implementation process. It also facilitates the job of members of the development team allowing them to complete their tasks quicker and better. In the context of 3D environment creation, it is a long process that would benefit from the impact of these tools. Throughout this study, important research and analysis was conducted to determine the features and requirements that would be crucial for the tool to help streamline 3D environment creation and save development time. The findings can be summarized under the following points:

- **User-Friendly Interface:** A tool with an intuitive and accessible UI is essential for making the 3D environment creation process easier. Users should be able to easily navigate and utilize the tool's features without confusion. This is why it is important to make sure the tool does not have a steep learning curve, which would otherwise delay the development of environments if the UI is too complex.

- **Environment Customization:** The tool should offer many options for tweaking the requirements and parameters of the generated environment. The more options that the tool supports, the better the users can create environments that fit their projects' needs.

- **Iterative Creation Process:** When creating a 3D environment using the tool, it is important to allow the users to adjust its different aspects and see the effect in real-time. Implementing this feedback loop in the tool's functionalities allows users to experiment with the generated environments and make improvements in a quick manner. This way development teams are able to create their environment prototype quickly, and keep building on it to achieve the desired result.

- **Seamless Integration with Existing Workflows:** The tool should be integrated within the existing development environment seamlessly, so that it does not interrupt or slow down ongoing workflows. It is also important for the tool to allow users to use the generated environments with other systems of their workflow. For instance, a 3D game would typically include a 3D environment, but also many other gameplay systems, such as characters and physics. This is why it is important that the tool generates environments that are reusable and extendable within the development engine.

- **Performance and Scalability:** The tool should be able to create environments of varying sizes and complexities without impacting performance. This is why it's important to analyze and optimize the code responsible of generating the environments in the code during its entire development process. The tool should also be designed and structured in a modular way, so that changes to parts of it would not impact the overall system.

5.4. Conclusion

In this section, we answered the three proposed questions of the study successfully. In the next section, we will present the study's impact on research, the industry and the host company. We will also discuss the potential validity threats of the project.

6. DISCUSSION

In this section, we will discuss the overall impact of the project on a research and industry level. We will also discuss its impact on Lanterns Studios, the company the project was conducted in. Finally, we will address the potential threats to the validity of the study.

6.1. Impact on Research

The conducted study makes a notable contribution to the field of procedural generation and 3D environment creation. It thoroughly analyzed and designed workflows and processes that can be taken into consideration in future investigations or used as a basis for other projects that revolve around the same domain. It also provides research about the advantages and drawbacks of relying on procedural generation for creating 3D environments, which offers valuable insights on deciding when to use this technique. Moreover, the study represents a new addition to the research of tools that integrate artificial intelligence in their workflows and the domain of assisted design. It also provides information on how to make sure the tool's output aligns with user expectations.

6.2. Impact on Industry

The conducted study could have a significant impact on the gaming industry. The developed tool throughout the study has the potential to revolutionize the way 3D environments are created, making the process faster, easier, and more cost-effective. It provides insights on the key features that should be implemented in a similar assisted design tool. Moreover, it sheds light on new opportunities for independent game developers and small to mid-sized companies when it comes to creating large scale 3D environments. It also can serve as a valuable resource for teams looking to improve their productivity. Finally, this study's contributions to the gaming industry could lead to an increase in the number and quality of games produced.

6.3. Impact on Lanterns Studios

The conducted study has a positive impact on the company's future development pipeline, as it now can be used to save development time, and create believable and attractive 3D environments with little effort. Moreover, the tool has can be used as the basis for the future version of a product that could be sold on the online marketplace, potentially generating additional revenue for the studio. Finally, the successful implementation and

evaluation of the tool throughout the study also enhances the company's reputation for innovation and excellence in 3D and game development.

6.4. Threats to Validity

In this section, we will discuss potential threats to the validity of the study in terms of internal and external validity.

6.4.1 Internal Validity

In the context of this study, it is important to acknowledge the complexity involved in generating believable 3D environments, since they consist of many layers and elements, each contributing to the overall attractiveness and coherence of the environment. In this study, we selected specific elements to design, implement and generate, but there could be other elements that we missed and have a better impact on the results of environment generation. To mitigate this threat, we could broaden the scope of the study to include a wider range of elements and more customization options. We can also take advantage of the tool's users' feedback to identify which elements are most impactful in creating convincing and engaging environments.

6.4.2 External Validity

The tool developed throughout this study is heavily dependent on Unreal Engine, the real-time 3D engine it is integrated within, as it uses many of the functionalities the engine offers, such as the procedural content generation framework. As a result, achieving the same results using other engines might be more challenging. It is also important to mention that the procedural content generation framework is an area under continuous development in Unreal Engine, meaning that future versions could introduce different functions, methods, or classes than the version our tool is using. To mitigate these threat, it would be beneficial to design systems that are modular where possible, and also to keep the developed tool's code up to date with future changes in Unreal Engine.

6.5. Conclusion

In this section, we discussed the positive impact of the project on research, the industry and the host company, as well as two types of validity threats: internal and external. In the next section, we will conclude this study, and discuss some future work that can be done.

7. CONCLUSIONS

This study took place at the host company Lanterns Studios, an independent game development company based in Tunis, Tunisia. The goal behind the study was to design and develop a tool capable of procedurally generating 3D environments based on user-provided prompts. The tool would be seamlessly integrated in Unreal Engine, the 3D real-time engine that the host company specializes in.

Based on the Design Science Research Methodology, the development of the tool was executed through three different iterations. The first iteration consisted of the development of a procedural content generation system, which allowed the creation and customization of 3D environments. The second iteration consisted of the interpretation of the user-provided prompts, and its integration with the previously developed PCG system. The third and final iteration consisted of the design and implementation of the tool's user interface, with which the user can interact with the tool's different functionalities.

Throughout this study, research was conducted on procedural generation of 3D environments and its techniques. The different layers that compose an environment were also thoroughly analyzed. The gathered findings were then used to identify the advantages and drawbacks of relying on procedural generation when creating 3D environments.

Since it is important to ensure a user-centered approach when developing the proposed tool, further research was conducted to identify the many challenges that might be encountered when procedurally generating 3D environments that align with user expectations.

Additional research was also undertaken to determine which features and requirements would be crucial to help streamline the 3D environment creation and save time during the phases of software and game development lifecycles.

The culmination of the study is a fully functional and integrated tool that is indeed capable of creating 3D environments efficiently. The tool has a positive impact on the industry, as it helps save development time, reduce costs and offers new opportunities. It will serve as the basis for the future monetizable version of the tool in the online marketplace. As such, it is important to mention some possible future work. It is recommended that the host company shares this tool with many potential customers from an early stage, to gather constant feedback and improve the tool accordingly. Some work that could extend the tool's capabilities is covering more customization options, and adding a suggestion system.

References

[1] J. Jaouabi, "Welcome to Lanterns! A word from the company CEO," 3 January 2020. [Online]. Available: https://lanterns-studios.com/welcome-to-lanterns-a-word-from-the-company-ceo/.

[2] Statista, "Video Games - Worldwide," [Online]. Available: https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide.

[3] A. Gupta, "The Gaming Playbook for Growth," 4 August 2023. [Online]. Available: https://www.news.aakashg.com/p/the-gaming-playbook-for-growth.

[4] R. Ramadan and Y. Widyani, "Game development life cycle guidelines," 2013.

[5] Juego Studio, "Crafting Immersive Worlds: The Art Of Environmental Design In 3D Games," 12 February 2024. [Online]. Available: https://www.juegostudio.com/blog/how-do-immersive-environmental-designs-for-3d-games-get-created.

[6] K. Tokrev, "What Do Tool Programmers Do?," 7 February 2017. [Online]. Available: https://80.lv/articles/what-do-tool-programmers-do/.

[7] Aerospike, "What is a real time engine?," [Online]. Available: https://aerospike.com/glossary/real-time-engine-architecture/.

[8] Epic Games, "Unreal Engine 5," [Online]. Available: https://www.unrealengine.com/en-US/unreal-engine-5.

[9] Unreal Engine, "Procedural Content Generation Overview," [Online]. Available: https://docs.unrealengine.com/5.3/en-US/procedural-content-generation-overview/.

[10] Epic Games, "Blueprints Visual Scripting in Unreal Engine," 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine.

[11] J. Freiknecht, "Procedural Content Generation for Games," Universität Mannheim, Mannheim, 2020.

[12] Applied Optics, "Unicursal random maze tool path for computer-controlled optical surfacing," 2015. [Online]. Available: https://www.researchgate.net/publication/284412154_Unicursal_random_maze_tool_path_for_computer-controlled_optical_surfacing/figures.

[13] L. O.-M. W. W. C. Prakash M Nadkarni, "Natural language processing: an introduction," Journal of the American Medical Informatics Association, USA, 2011.

[14] OpenAI, "Introducing ChatGPT," [Online]. Available: https://openai.com/blog/chatgpt.

[15] OpenAI, "Transforming work and creativity with AI," [Online]. Available: https://openai.com/product.

[16] G. Smith, "An Analog History of Procedural Content Generation," Northeastern University, Playable Innovative Technologies Lab, Boston, 2015.

[17] A. Zafar, H. Mujtaba and O. Beg, "Procedural Content Generation for General Video Game Level Generation," 2021.

[18] B. Ritzl, "The roguelike archive," [Online]. Available: https://britzl.github.io/roguearchive/.

[19] N. Hammar and J. Persson, "Designing for Replayability: Designing a game with a simple gameplay loop for the purpose of being replayable," Uppsala University, 2022.

[20] M. Dahrén, "The Usage of PCG Techniques Within Different Game Genres," Malmö Universitet, 2021.

[21] B. Detterfelt and S. Blomqvist, "Real Time Integrated Tools for Video Game Development - a usability study," Linköping University, 2020.

[22] T. Semiawan, M. R. Alifi, H. Hayati and D. C. U. Lieharyani, "Analysis of the Effectiveness and Efficiency of Software Development Tools," 2021.

[23] M. S. O. Almeida and F. S. C. d. Silva, "Requirements for game design tools - A Systematic Survey," 2013.

[24] A. Filipović, "The Role of Artificial Intelligence in Video Game Development," 2023.

[25] B. Xia and X. Ye, "Recent Research on AI in Games," 2020.

[26] D. Dellermann, P. Ebel, M. Söllner and J. M. Leimeister, "Hybrid Intelligence," 2021.

[27] S. Seidel, N. Berente, A. Lindberg, K. Lyytinen, B. Martinez and J. V. Nickerson, "ARTIFICIAL INTELLIGENCE AND VIDEO GAME CREATION: A FRAMEWORK FOR THE NEW LOGIC OF AUTONOMOUS DESIGN," Journal Of Digital Social Research, 2020.

[28] M. J. Chris, "Methodology Section for Research Papers," San Jose State University Writing Center, 2021.

[29] R. J. Wieringa, "Design science methodology for information systems and software," 2014.

[30] J. Venable, "The role of theory and theorising in design science," 2006.

[31] D. LUEBKE, M. REDDY, J. D. COHEN, A. VARSHNEY, B. WATSON and R. HUEBNER, Level Of Detail For 3D Graphics, San Francisco: Morgan Kaufmann, 2003.

[32] Epic Games, "Unreal Engine 5.4 Release Notes," April 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5.4-release-notes.

[33] A. Shvets, Dive Into Design Patterns, Refactoring.Guru, 2021.

[34] Object Management Group, "CallBehaviorAction," in *OMG Unified Modeling LanguageTM (OMG UML) Version 2.3*, Object Management Group, 2010, pp. 357-359.

[35] Q. C. Y. Li, "Data Flow Diagram," in *Modeling and Analysis of Enterprise and Information Systems*, Berlin, Heidelberg, Springer, 2009, p. 85–97.

[36] G. J. Myers, C. Sandler and T. Badgett, The Art of Software Testing, 2011.