Requirements Document for Modular World Generation

1. Introduction

This document delineates the functional and technological requirements for the development of a modular world generation tool. This tool will facilitate the creation of game environments, including buildings, roads, NPCs, and more, from a pre-existing bank of options. Additionally, it will incorporate an AI-driven decision engine to dynamically adapt game environments based on player actions.

2. Stakeholders

- **Developers**: This category encompasses independent game developers as well as game development companies utilizing Unity. This includes any Unity user, whether they are creating a small personal project or a larger game intended for public release.

3. Functional Requirements

3.1 Scene Creation and Customization

3.1.1 Necessary

- Users should be able to create new scenes with preferences such as the number of NPCs, terrain size, and number of structures.
- The tool should include options for NPC types (e.g., civilian, police officer), building types (e.g., residential buildings, stores, offices), and vehicle types (e.g., cars, bikes, buses).
- The tool should support dynamic terrain sizes as specified by the user.
- The tool should determine and implement standard movement patterns for mobile objects (e.g., NPCs, vehicles).
- The tool should allow users to regenerate scenes based on specific preferences and modify them after reviewing the output.

3.1.2 Optional

- The tool should offer the option for random types of NPCs/buildings, allowing users to specify only the number of objects, or to control the type and quantity of each object independently.
- The tool should enable users to select the daytime of the scene.
- The tool should allow users to export and save their preferences, with an option to import and restore these preferences.

3.1.3 Future Versions

- **Theme Selection**: Our vision includes a multi-theme generator, enabling users to customize not only the scene settings (e.g., number of NPCs and structures) but also the main theme of the game (e.g., modern times, middle ages, Sci-Fi).

3.2 Dynamic Environmental Support

- Dynamic environmental support refers to a crucial aspect of our project aimed at enabling dynamic changes based on player actions and reactions during gameplay. The primary objective is to alter NPC behavior in response to in-game events, thereby necessitating robust "Dynamic Environmental" support. This component of the project is being developed in collaboration with the 'Immersive NPC Behavior' team.
- The tool will support dynamic environmental script changes that affect scene preferences, integrating with the 'Immersive NPC Behavior' project.
- The tool should ensure the creation of walkable and non-walkable areas in a logical and accepted manner. The ‘Immersive NPC behavior’ project will be able to use and implement the behavior for the scene.

4. Non-Functional Requirements

4.1 Performance

- The tool should provide a response time of no more than 2 seconds for user interactions involving preference selection.

4.2 Scalability

- The tool should be designed modularly to facilitate the addition of new themes and assets without complications.

4.3 Reliability and Availability

- The tool should successfully create scenes in 80% of user attempts.
- The tool should be available to all Unity users who meet the tool requirements (e.g., Unity version).
- The tool should be available on the Unity Asset Store, with updates accessible to all users.

4.4 Security

- The tool will include a licensed option providing full access to all features. Non-paying users will have access to limited features.

4.5 Integration

- The tool should integrate seamlessly with Unity and other relevant tools and assets.

5. Architectural Requirements

5.1 System Architecture

- The tool will follow a modular monolithic architecture, dividing the application into distinct, independent modules. Each module handles a specific feature or functionality, but all modules are part of a single executable.
- MongoDB, A NoSQL, document-oriented database that offers flexible schema design, high scalability, and fast performance, making it ideal for dynamic and large-scale game development projects.
- Integration with the Unity engine is essential, as the tool is designed for Unity.

5.2 Scalability

- The system should be capable of scaling to handle an increase in the number of scenes and NPCs.
- MongoDB will be implemented in a simple and modular way, allowing for the easy addition of new themes and assets without significant architectural changes.

5.3 Performance

- Reading and writing from the database should be almost immediate, thanks to the local use of MongoDB.

5.4 Reliability and Availability

- The tool should be available to users at all times after the initial download, without requiring an internet connection.
- The likelihood of database corruption is very low, as the database is locally saved and not network-dependent.

5.5 Security

- There is no risk of viruses or hacking through the database.
- The tool will be downloaded from a reliable source (Unity Asset Store), ensuring protection against malware.

5.6 Data Architecture

- The system should utilize a distributed database architecture to ensure data consistency and availability.
- The local database will store the IDs of associated objects in the scene and their positions. This information can be used to edit scenes without needing to delete or recreate them.

6. Technological Requirements

6.1 Programming Languages and Frameworks

- **Programming Languages**: C# will serve as the primary programming language due to its robust support within the Unity engine and the extensive community around it. Additionally, Blender will be utilized to create dynamic resources for various scenes.
- **Frameworks**: Unity Game Engine will be used for its comprehensive features and support for game development.

6.2 Frontend Technologies

- **Technologies**: The user interface for managing scene configurations and preferences will be developed using Unity's built-in UI system.
- **Considerations**: This choice ensures a seamless and integrated development experience, providing a consistent look and feel with the game development environment.

6.3 Development Tools

- **Version Control**: Git will be used for version control to manage source code changes efficiently. GitHub will serve as the repository hosting service.
- **Editor Enhancements**: The Unity EditorTool library in C# provides a comprehensive set of tools for creating custom editor extensions. This library allows developers to implement global and component-specific tools, enhancing the Unity editor's functionality and streamlining the game development process.
- **Collaboration Platforms**: Slack or Microsoft Teams will be used for team communication and collaboration.
- **CI/CD Practices**: GitHub Actions will be implemented for continuous integration and continuous deployment to automate the build and deployment process.