Requirements Document for Immersive NPC Behavior System

1. Introduction

This document outlines the functional and technological requirements for developing the *Immersive NPC Behavior System*. This system is designed to generate lifelike and adaptive behaviors for non-player characters (NPCs) in-game environments created by the *Modular World Generation* tool. The system will enable NPCs to respond dynamically to environmental changes and player interactions, enhancing the immersive quality of the gameplay.

2. Stakeholders

- **Developers**: Unity-utilizing game developers and companies are the primary customers. They will use the system to enhance their games as part of their creation process. The system will focus on ease of integration and meeting development needs.

3. Functional Requirements

3.1 NPC Behavior Generation and Adaptation

3.1.1 Necessary:

- **Dynamic NPC Behavior**:
  The system must support creating adaptive behaviors for various NPC types (e.g., civilians, and police officers) with role-specific characteristics.
- **Real-Time Reactions to Player Actions**:
  NPCs should dynamically respond to player actions (i.e. drawing out a weapon, hitting someone, initiating conversation, and so on), including avoidance (e.g., moving out of the player's way), assistance (e.g., offering guidance or support), or aggression (e.g., confronting the player when provoked).
- **Integration with Modular World Generation**:
  The system should align with environments created by the *Modular World Generation* tool, incorporating scene elements such as terrain, objects, and pathways into NPC behavior logic.

- **Environment Constraints Compliance**:
  NPCs must follow scene-specific rules, navigating only walkable areas and avoiding obstacles or non-walkable zones like cliffs or walls.
- **Contextual Decision-Making**:
  NPCs should base decisions on contextual factors, such as:
  - **Time**: Modifying actions depending on day or night.
  - **Proximity**: Reacting to nearby events or player actions.
  - **Social Roles**: Behaving in ways appropriate to their role (e.g., police officers intervening in disturbances, merchants attending to their shops).

3.1.2 Optional:

- **Interaction with Environmental Objects**:
  NPCs may interact with a variety of objects in their surroundings, enabling richer gameplay experiences, such as:
  - **Doors**: Opening, closing, or knocking on doors based on their role or context.
  - **Vehicles**: Entering, exiting, or using vehicles such as cars, bikes, etc.
  - **Props and Tools**: Picking up, using, or placing objects like tools, weapons, or decorative items to enhance immersion.
- **Customizable Object Interactions**:
  Developers may have the option to configure which objects NPCs can interact with and define conditions for these interactions, such as requiring specific triggers or player proximity. Every interaction with a dynamic object may constitute a mini-scenario, allowing developers to script detailed behaviors, outcomes, and context-sensitive responses. For example, a player interacting with a slot machine could trigger new dynamic player reactions and movements.

3.1.3 Future Versions:

- **Multi-Layered NPC Personalities**:
  Future versions should include more sophisticated NPC personalities with:
  - **Core Traits**: Baseline characteristics like aggression, friendliness, or curiosity.
  - **Dynamic Mood States**: Emotional responses to in-game events, such as fear or joy.
  - **Memory Systems**: Allowing NPCs to adapt based on past interactions.
- **Graphical Behavior Customization**:
  A graphical interface for developers to build and refine NPC behavior hierarchies, featuring:
  - Drag-and-drop tools for designing behavior trees.
  - Real-time previews to test NPC responses.

- **Pre-Defined Behavior Templates**:
  The system should provide templates for various genres to simplify NPC behavior setup, such as:
    - **Modern**: NPCs commuting or socializing in urban areas.
    - **Medieval**: NPCs farming or trading in markets.
    - **Sci-Fi**: NPCs interacting with futuristic environments.

4. Non-Functional Requirements

4.1 Performance:

- The system should ensure behavior updates occur with a latency of less than 2 seconds in standard gameplay scenarios.

4.2 Scalability:

- The system must handle at least 10 active NPCs in a single scene without significant performance degradation.

4.3 Reliability and Availability:

- NPC behaviors must remain consistent and error-free in at least 90% of gameplay sessions.

4.4 Security:

- **Behavior Logic Protection**:
  The system must secure NPC behavior logic against unauthorized modifications or tampering.
- **Two-Tier Licensing Model**:
  The system will feature:
  - **Free Version**: Basic NPC behavior generation tools.
  - **Paid Version**: Advanced features such as graphical customization, detailed templates, and encrypted behavior logic.

4.5 Integration:

- Seamless integration with Unity and compatibility with *Modular World Generation* for synchronized updates.

5. Architectural Requirements

5.1 System Architecture:

- **Modular Design**:
  The system will isolate NPC behavior logic from scene generation, ensuring flexibility and easier updates.
- **AI Decision-Making**:
  A rule-based engine will manage NPC behaviors, supported by:
  - **Decision Trees**: NPC behavior will be driven by decision trees, allowing structured decision-making based on environmental variables, state, and player interactions.
  - **Storytelling Package**: A storytelling package will manage branching narratives and character development. NPCs will remember past interactions, adjust behaviors accordingly, and influence long-term storylines.

5.2 Performance:

- Behavior updates and pathfinding calculations will use optimized algorithms to minimize the computational load.

5.3 Scalability:

- The architecture will allow for additional NPC behavior modules to be added as needed.

5.4 Data Architecture:

- Behavioral states will be stored in local databases linked to scene objects. These states will follow a Finite State Machine (FSM) model, ensuring that NPC behavior transitions smoothly between defined states (e.g., "Idle," "Interacting," "Alert") based on triggers or conditions, enabling dynamic and responsive interactions.

6. Technological Requirements

6.1 Programming Languages and Frameworks:

- **C#**: The primary language for development.
- **Unity Engine**: For game development and scene integration.

  We chose Unity for its popularity and powerful game development capabilities. C# was the primary language because it works seamlessly in tandem with the engine for efficient development.

6.2 Development Tools:

- **Version Control**: Git for managing source code.
- **Continuous Integration**: GitHub Actions for automating builds and deployments.
- **Communication Platforms**: Slack or Microsoft Teams for team coordination.