System Requirements

1st Tomas Garcia
California State University, Fresno
College of Science and Mathematics
Fresno, California

2nd Marcus Ramirez
California State University, Fresno
College of Science and Mathematics
Fresno, California

3rd Jason Yang
Department of Computer Science
California State University, Fresno
Fresno, California

4th Gabriel Dos Anjos
California State University, Fresno
College of Science and Mathematics
Fresno, California

5th Jordan Diaz
Department of Computer Science
California State University, Fresno
Fresno, California

6th Edgar Rodriguez
Department of Computer Science
California State University, Fresno
Fresno, California

I. Introduction

This System Requirements Document (SRD) serves as a comprehensive guide for the development of our game project, outlining the essential functionalities, non-functional requirements, and overall design specifications. This document aims to provide a clear understanding of the system's objectives, user needs, and technical constraints, ensuring that all stakeholders are aligned throughout the development process. By establishing a solid foundation of requirements, the SRD will facilitate effective communication among team members, promote adherence to project goals, and ultimately contribute to the successful delivery of a high-quality gaming experience. This document will be continuously reviewed and updated to reflect any changes in project scope or stakeholder input, ensuring that it remains a reliable resource throughout the development lifecycle.

II. Functional Requirements

The functional requirements listed below for the system detail the necessary features that need to be implemented to ensure that the game provides a rich, interactive, and responsive experience for players. Each feature is critical to the overall functionality and player experience and is described in detail to ensure a clear understanding of what each component will entail.

A. Player Input and Movement

The first key feature of highest priority is player input in itself, which provides players with intuitive and seamless control over their characters, allowing them to perform essential actions like moving, jumping, attacking, and interacting with the environment. It will support various control schemes, including keyboard, gamepad, and mouse, making the game accessible to a wide range of players. The process involves capturing inputs through Godot's InputEvent system and mapping them to corresponding game actions. The inputs will generate game events such as player movements or UI interactions, all of which depend on other features like Player Movement and Intuitive UI. Given its significance to the gameplay experience, it is of high priority and will be handled by Jason.

Player movement in itself will take advantage of these inputs, enabling players to navigate the game world fluidly. This includes mechanics such as jumping, running, dashing, and climbing, which need to feel responsive to enhance the game's overall experience. The system will process player inputs and physics calculations like gravity and collision detection to update the player's position and trigger corresponding animations. Player movement is a critical feature dependent on the Player Inputs and the Animation System, making it a high priority for Jason and Gabriel to develop.

In terms of customization, remappable controls will allow players to adjust the key bindings for various actions according to their preferences, offering a more comfortable gameplay experience. This feature will include a user interface for remapping inputs, saving these configurations, and loading them during gameplay. The remappable controls are linked directly to Player Inputs and have been assigned a medium priority for Marcus, who will ensure it aligns with the user's preferences.

B. Intuitive User Interface

The user interface (UI) of the game will be designed to offer players clear and responsive visual feedback through various menus, status bars, and other graphical elements, ensuring that essential game information is presented in an accessible and streamlined manner. The system will dynamically adapt its display in response to ongoing gameplay, allowing players to seamlessly interact with both the environment and the game's various mechanics.

Upon launching the game, players will be presented with a title screen featuring a main menu that offers multiple options such as starting a new game, configuring control and audio settings, viewing credits, or exiting the game. Each option will be interactable, with the UI providing intuitive guidance as users navigate the menu. For instance, upon selecting the "exit" option, the UI will prompt the player with a confirmation dialog to ensure the user's intent is correctly captured, and closing the program if the user selects "yes".

Once the game begins, the UI transitions into gameplay mode, offering a comprehensive HUD (Heads-Up Display) that provides critical information such as the player's current health status, available abilities, collected resources, and any active status ailments. These elements are essential for maintaining player awareness of their in-game performance and resources, facilitating informed decision-making during gameplay.

In addition to static elements, the UI will be highly interactive, dynamically adjusting to both player inputs and real-time game events. Health bars will reflect the player's current status, ability icons will update to show usable skills, and loot indicators will track the items collected throughout the game. As a core component of the game's design, the UI must ensure clarity and responsiveness, enhancing the player's immersion by presenting all necessary information in a concise yet informative manner.

This feature is of critical importance and will be handled by Marcus and Gabriel, who will ensure that the UI's functionality is tightly integrated with the player input systems and the game's state management. By leveraging real-time game data, the UI will consistently reflect changes in the player's status, ensuring a smooth, engaging, and immersive user experience.

C. Temporary and Permanent Progression

Temporary progress preservation and permanent progression will both be implemented within the game to give players a sense of accomplishment without the fear of losing all progression and achievements throughout their gameplay. These features are specifically designed to accommodate the mechanics of a roguelite game, which inherently punish players by returning them to the beginning upon dying or forfeiting during a run, thereby ensuring that they cannot circumvent the punitive nature of "permadeath."

Temporary session preservation shall be implemented to enable players to pause their gameplay and save their current game state, ensuring that they can exit a dungeon area without the fear of losing any progress. This functionality is essential for providing players with the ability to resume their game seamlessly from where they left off, without any loss of data or game state. When a player chooses to exit mid-session, the game will automatically serialize the current game state, capturing critical data such as the player's position, health, inventory, and other relevant metrics. This serialized data will then be saved into a designated file, which the game can subsequently load when the player chooses to resume gameplay. It is important to note that the temporary progress can only be reloaded once and the file containing the temporary save will be deleted upon resuming from it; if the player does not initiate a new temporary save, they will be ejected back to the hub upon the next boot of the game, requiring them to embark on a new dungeon run from the beginning.

This feature has been given a high priority due to its direct impact on player satisfaction and continuity of play. Tomas, who is tasked with overseeing this functionality, will ensure that the system accurately captures and stores the player's current game data and can reload it when the game is resumed. The feature relies heavily on the current Game State, as it must serialize the various components of the game in real-time. This involves processes such as capturing the player's position in the game world, current health status, inventory contents, and any active abilities or status effects. By translating this information into a structured file format, the system guarantees that all gameplay elements will be preserved, and that resuming the game will restore the player to the exact state in which they left off. In addition to saving the game, the system must also be capable of recognizing and loading the saved data, allowing players to pick up exactly where they left off.

Permanent progression, on the other hand, shall be implemented to ensure that the game offers players a long-term sense of achievement by allowing them to unlock new upgrades and features over time within the hub area. Unlike the temporary session saves, which preserve a player's moment-to-moment progress, the Permanent progression system is designed to track and maintain the player's overarching accomplishments across multiple playthroughs. As players advance through the game, they will unlock new abilities, upgrades, and other game-enhancing features. This system incentivizes continued play, as players will have access to new content and capabilities that reflect their accumulated progress.

To implement this feature, the game will track the player's achievements and progress, unlocking new abilities, upgrades, or features as they meet specific milestones. The unlocked content will be saved to a permanent data file, ensuring that players retain their progress even if they start a new game or enter a new session. Permanent progression will be tied to the Ability Inheritance system, allowing the player to be able to inherit abilities from more defeated enemies and cycle through them at the press of a button. It will also be tied to structures in the main hub, allowing them to expand the locale with more buildings that grant them access to features, both functional and cosmetic, which will persist across all future playthroughs. Tomas will oversee the development of this system, ensuring that it is properly integrated with the Game State, Upradable Hub, and the Ability Inheritance mechanisms. The player's progress, including any unlocked abilities or upgrades, will be persistently saved, allowing the player to experience the full breadth of the game's content over time.

D. Upgradable Hub Area

The hub area serves as a central feature within the game, providing players with a dynamic space that they can enhance and customize over time. This central hub is not only a focal point for player interaction but also a representation of their achievements and progression throughout the game. By implementing a robust upgrade system, players will unlock new buildings and features that enhance gameplay, thus fostering a sense of ownership and investment in their virtual environment.

Upon initial entry into the hub, players will encounter a few persistent locales that will remain available from the

onset of their journey. These foundational elements include a portal, which serves as the gateway to the main gameplay loop of dungeon runs; a tutorial station, offering guidance and instruction for new players; and a handful of non-playable characters (NPCs) that populate the area, providing a sense of life and community while also giving players hints to help them defeat enemies. As players progress and accumulate resources, they will be presented with opportunities to upgrade their hub, unlocking additional functionalities and aesthetic enhancements that will fundamentally alter their experience.

Upgrades will encompass both cosmetic and functional elements. For instance, players will have the ability to upgrade their personal dwelling, allowing for visual customization and personalization that reflects their journey. Furthermore, functional upgrades will introduce new structures that enhance gameplay mechanics. Notable examples include a farm or bestiary, which will provide players with essential information about the various enemies encountered during dungeon runs, thereby enriching the overall narrative and strategic planning. A potion brewery will be another significant addition, offering temporary enhancements and upgrades that players can utilize in their next run, thus creating a direct link between hub upgrades and gameplay performance.

In addition to these structures, a laboratory will allow players to expand their capacity for inherited abilities, enabling them to store and cycle through a greater number of powers acquired from defeated foes. This not only enhances combat versatility but also encourages players to explore various strategies in dungeon runs. With each new building that becomes accessible, the hub will gradually fill with NPCs, ranging from one to three per building, further enhancing the atmosphere and sense of community within the game.

The development of the hub area has been assigned a medium priority level, indicating that it will be an enriching feature but not critical to immediate gameplay mechanics. Jason has been designated as the team member responsible for overseeing this feature's development. The implementation will require careful tracking of player resources and the conditions necessary for upgrades. The system will facilitate the modification of the hub area in response to player progress, ensuring that the environment evolves alongside the player's journey, thus maintaining engagement and investment in the overall gameplay experience.

E. LAN Multiplayer

LAN Multiplayer will enable players to engage in cooperative gameplay by connecting locally through a robust networking system designed to handle various player interactions and connections. This feature is integral to enhancing the overall gaming experience by facilitating real-time collaboration between players as they navigate through dungeon runs together.

To initiate a multiplayer session, players will be prompted upon starting a dungeon run if they wish to play with a friend. If the player chooses to say yes, they will be prompted to either host or join a session, and after connect via a designated local area network (LAN), utilizing inputs such as IP addresses and player connection requests. Once connected, the game will synchronize the game state across all participating players, ensuring a cohesive and interactive environment. Real-time updates regarding game events, player actions, and game state modifications will be communicated seamlessly among players, fostering a shared gameplay experience.

An important aspect of the multiplayer functionality is its handling of game sessions during player actions. If one player chooses to pause the game, the pause menu will appear exclusively on their screen, thereby disabling any input for their character while allowing them to navigate the menu. This ensures that the game continues normally for the other player, maintaining the flow of gameplay. However, if the host disconnects from the session, the current run will end for all players involved. In contrast, if the other player exits while the host remains connected, the host can continue their run independently, with the option for the exiting player to rejoin later.

It is essential to note that multiplayer sessions will not have access to the temporary save feature implemented in single-player mode. Consequently, if both players decide to exit the game, the run will be forfeited, thereby emphasizing the need for players to commit to their session until completion. This system will require a continuous synchronization of player actions and game states, as well as processes for establishing local network connections, handling disconnects and reconnections, and managing multiplayer events.

With a medium priority assigned to this feature, Jason will oversee its development to ensure that it integrates effectively with existing game systems, such as Player Inputs and Game State. By providing a platform for cooperative gameplay, this feature aims to enhance player engagement and create a dynamic social experience within the game.

F. Player Combat

The combat system will encompass three integral features: Ability Inheritance, Combat System, and Health Management, each designed to enhance the dynamic and immersive nature of gameplay. Together, these features form a cohesive system that promotes exploration, strategic gameplay, and an engaging combat experience, ensuring that players are equipped to face the diverse challenges within the game.

1) Ability Inheritance: The ability inheritance feature allows players to acquire unique abilities from specific enemies after defeating them, promoting a sense of skill customization and strategic gameplay. This mechanism encourages players to explore various enemy types and engage in diverse combat scenarios, enhancing their tactical options. Upon defeating an enemy, the system will track the relevant enemy data and update the player's abilities accordingly. The integration of this feature ensures that players can adapt their combat style based on the abilities they inherit, fostering a rich interaction with the game's enemy variety. High priority is assigned to this feature, with oversight from Tomas and Jason, who will ensure its effective implementation alongside the Combat System and Enemy Variety.

2) Melee and Ranged Combat: The combat itself will provide players with a multifaceted approach to engaging enemies through both melee and ranged attacks. This system is designed to deliver responsive feedback mechanisms, including damage indicators and enemy reactions, to enhance the overall combat experience. Players will input commands to execute various combat actions, while the game will process these inputs in real-time to calculate damage dealt based on attack type and enemy status. The system will also incorporate dynamic status effects that can improve or impair the player's performance during combat encounters. Given its critical role in gameplay, this feature has been assigned a high priority, with Marcus and Gabriel responsible for its development. The Combat System will rely heavily on Player Inputs, Health Management, and Scaled Enemy AI to ensure a balanced and engaging combat experience.

3) Health Management: The health management feature will track player health points (HP) throughout gameplay, providing a comprehensive system for health restoration via pickups or abilities. This feature ensures that players maintain a clear understanding of their health status, which is vital for strategic decision-making during combat. The health management system will monitor the player's health value, process health restoration pickups, and update the health bar in real-time to provide visual feedback. By integrating this feature with the Combat System and UI, players will receive immediate updates on their health status, enhancing their ability to navigate challenges effectively. Health Management is deemed a high-priority feature, with Marcus, Gabe, and Edgar overseeing its successful implementation.

G. Level Randomization and Variety

Levels across a run shall be designed to provide players with a fresh and engaging experience each time they embark on a dungeon run. This system encompasses three key features: Level Randomization, Level Differentiation, and Environmental VFX, each contributing to the dynamic nature of gameplay.

Level Randomization

To randomize which level a player will be dropped into, a shuffling mechanism shall be designed to select from a set of hardcoded premade levels. This approach ensures that each playthrough presents a unique combination of layouts and challenges, maintaining high-quality gameplay while preventing predictability. The system will utilize premade level data as inputs to shuffle and randomly select levels for each session. It is essential that this process also ensures a balanced difficulty progression to cater to players of varying skill levels. Given its critical role in enhancing replayability, this feature has been assigned a high priority, with oversight from Tomas and Marcus, who will ensure that the shuffling algorithm effectively loads randomly selected levels for each playthrough while maintaining the integrity and quality of the gaming experience.

1) Level Differentiation and Environmental VFX: To enrich player engagement, each game level will possess distinct visual themes and unique challenges. This will be accomplished through the integration of varying enemy types, environmental hazards, and specific objectives that players must achieve within each level. By utilizing level design and enemy variety data as inputs, the system will create visually unique environments and diverse gameplay experiences across levels. The priority level for this feature is medium, with Marcus, Edgar, and Jordan responsible for its development. Their focus will be on designing unique visual elements and implementing distinct objectives that enhance the player's interaction with the game world, contributing to the overall diversity and enjoyment of the gameplay experience.

Building on this, dynamic visual effects will be implemented that reflect the current environmental conditions. These effects may include changes in weather, time of day, and other atmospheric phenomena that can influence gameplay by altering visibility or player interactions with the environment. By processing inputs related to the environmental state and player position, the system will apply these visual effects in real-time, creating a more engaging and responsive game world. Although assigned a low priority, this feature will assist in reinforcing the game's atmosphere and will also be overseen by Jordan, Edgar, and Marcus. Their efforts will focus on dynamically updating the visual effects based on player actions and time progression, thus ensuring a visually captivating experience.

H. Enemy Variety and Types

The game will incorporate a diverse range of enemy types, each possessing unique abilities, strengths, and weaknesses. This rich variety is designed to encourage players to develop different strategies tailored to the specific enemies they encounter throughout their adventures. By offering unique enemy behaviors, players will be challenged to adapt their combat tactics, enhancing the overall combat dynamics and ensuring a high replayability factor. To achieve this, the development team will focus on designing and implementing distinct enemy types, carefully balancing their abilities and strengths to create an engaging experience. Additionally, enemy placements will vary across levels, introducing an element of unpredictability that keeps players on their toes. This feature is deemed critical to the game's success, resulting in a high priority level assigned to its development. The team members involved in this feature will include Marcus, Tomas, Jason, and Gabriel. The necessary inputs for this feature will comprise enemy behavior data and enemy abilities, while the outputs will consist of diverse enemy encounters and unique enemy behaviors. The core processes involved will include designing distinct enemy types, balancing their abilities, and varying enemy placements per level. This feature depends on the successful implementation of the Combat System and Level Differentiation.

I. Difficulty Scaling

To ensure a continuously challenging experience for players, the game will implement a difficulty scaling system where enemies will adjust their difficulty based on player progress

and performance. This adaptive mechanic is crucial for maintaining player engagement, as it ensures that the game remains challenging without becoming overwhelming. By dynamically adjusting enemy stats and behaviors in response to player progression, the game will cater to individual player skill levels and play styles. Tomas will oversee this feature with a focus on fine-tuning this scaling mechanism to create a balanced experience. The inputs for this feature will primarily include player progress, while the outputs will involve dynamic enemy difficulty and adaptive enemy behaviors. The core processes will consist of adjusting enemy statistics and behaviors based on the player's progression and scaling enemy abilities in line with the player's skill level. This feature relies on the successful execution of the Combat System and the incorporation of Enemy Variety.

J. Environmental Interactions

The game will feature various environmental interactions, allowing players to engage with different elements, such as destructible objects and traps, that impact gameplay. This adds an exciting layer of strategy and exploration, encouraging players to interact with their surroundings in meaningful ways. By incorporating destructible objects, players can utilize their environment to gain tactical advantages in combat situations or explore hidden areas. Additionally, traps can create unexpected challenges, requiring players to think critically about their movements and decisions. Edgar and Jordan will primarily develop this feature, designing engaging environmental interactions that enhance gameplay. The inputs for this feature will include environmental object data, such as destructibles and traps, as well as player interactions. The outputs will consist of environmental changes, such as objects being destroyed or traps being triggered. The core processes will involve detecting player interactions with environmental objects and applying changes to the game world based on these interactions. This feature will depend on the successful implementation of the Combat System and Environmental VFX to provide a cohesive and immersive experience.

K. Inventory and Loot

The game will incorporate a robust inventory system that allows players to collect, manage, and utilize various items and loot found throughout their adventures. This system will include power-ups, consumables, treasures, and crafting materials, providing players with strategic choices that can significantly influence their gameplay and character customization options. Players will have the ability to make meaningful decisions based on their collected items, enhancing their overall experience. To implement this feature, the development team, led by Jordan, will focus on several key processes, including tracking item collection, organizing the inventory, processing item usage to apply effects, and displaying the inventory user interface. The necessary inputs for this system will consist of collected items, loot drops, and player inventory interactions, while the outputs will include an updated inventory, usable items in gameplay, and effective loot management. This feature relies on the successful implementation of the Loot System, User Interface (UI), and Crafting System to ensure a seamless player experience.

L. Sound Design & Audio Feedback

An integral aspect of the game's immersion will be its sound design, which will encompass background music, sound effects, and audio cues that enhance the overall gameplay experience. This feature aims to provide vital audio feedback for various player actions, interactions, and environmental conditions, thereby deepening the player's engagement with the game world. The development team, under the direction of Marcus, will design and implement sound effects for a wide range of game actions, dynamically trigger changes in background music to reflect gameplay situations, and apply audio cues based on player and environmental interactions. The inputs required for this feature will include game events, such as attacks, movements, and interactions, alongside environmental triggers. The expected outputs will consist of sound effects, background music, and informative audio cues. This feature depends on the successful execution of the Combat System, Environmental VFX, and Player Inputs to create a cohesive auditory experience.

M. Tutorial/Help System

To facilitate player onboarding, the game will include a comprehensive tutorial-like system that players will be required to view upon their first boot of the game. Additionally, this tutorial will be accessible from a specific area in the hub, allowing players to revisit it at their discretion. The system is designed to teach players the controls and fundamental mechanics of the game, ensuring they are well-prepared for the challenges ahead. Tomas will oversee the development of this feature, focusing on presenting tutorial steps based on individual player progress, providing real-time feedback during the tutorial, and tracking tutorial completion. The inputs for this system will include player interactions and tutorial progression data, while the outputs will consist of instructional prompts and feedback regarding tutorial completion. This feature will depend on the successful integration of Player Inputs, UI, and the Help System to ensure clarity and usability.

N. 2D Spritework

The visual presentation of the game will be defined by striking 2D sprite assets that represent characters, enemies, environmental elements, and various objects within the game. Each sprite will be meticulously designed to be distinct, easily recognizable, and thematically consistent with the overall visual style of the game. Additionally, the spritework will feature animations that bring character actions, enemy movements, and environmental interactions to life, thus enhancing visual storytelling and gameplay experience. Marcus will lead the development of this feature, focusing on creating and animating 2D sprite assets and applying these animations during gameplay for characters, enemies, and environments. The necessary inputs will include visual asset design and animation

data, while the outputs will comprise completed 2D sprite assets and character/environmental animations. This feature will depend on the successful implementation of the Combat System, Environmental Interactions, and Visual Effects to deliver a polished and engaging aesthetic experience.

III. Non-Functional Requirements

This section outlines the non-functional requirements for the game project, which are critical for ensuring a high-quality player experience and overall system performance. Non-functional requirements define the quality attributes, system performance metrics, and constraints that the game must adhere to throughout its development. These requirements serve as guidelines for the development team, helping to ensure that the game is reliable, efficient, maintainable, and secure, while also meeting the expectations of its users. By organizing and prioritizing these requirements, we aim to create a cohesive framework that enhances gameplay, supports smooth user interactions, and ultimately contributes to a polished and engaging gaming experience.

A. Player Input and Movement

1) Quality Attributes:

- **Reliability:** The input system should maintain a failure rate of less than 1% during gameplay sessions, ensuring players can consistently perform actions without input lag or missed commands.
- **Usability:** The remapping interface for controls must be intuitive, with at least 80% of players able to successfully remap their controls within three minutes of access, enhancing player comfort and engagement.
- **Efficiency:** The system should respond to player inputs with a maximum latency of 100 milliseconds during a single-player session to ensure fluid and responsive character movements during gameplay.

2) System Performance:

- **Response Time:** The average response time for player inputs must not exceed 100 milliseconds to ensure a seamless gaming experience, particularly during critical moments of gameplay such as combat or platforming sequences.
- **Throughput:** The system should support a minimum of 50 concurrent player input events without noticeable degradation in performance, allowing for complex interactions within the game world.
- **Resource Utilization:** CPU and memory usage for the input processing should remain under 70% during peak gameplay to maintain overall game performance and allow for other processes to operate smoothly.

3) Constraints:

- **Hardware Limitations:** The input system must be compatible with standard gaming peripherals, including keyboards, gamepads, and mice, ensuring functionality across a wide range of devices.
- **Software Dependencies:** The system must integrate seamlessly with Godot's InputEvent system, which should be at least version 3.3 or later, to ensure compatibility and functionality with the game engine's input processing capabilities.

B. Intuitive User Interface

The user interface (UI) of the game will be designed to offer players clear and responsive visual feedback through various menus, status bars, and other graphical elements, ensuring that essential game information is presented in an accessible and streamlined manner.

1) Quality Attributes:

- **Reliability:** The UI should have a failure rate of less than 1% regarding interactions (such as button presses or menu selections), ensuring that players can navigate menus and receive feedback without errors.
- **Usability:** At least 80% of users should report satisfaction with the UI's intuitiveness and ease of use, with a focus on minimizing the number of clicks needed to access critical game functions (e.g., starting a game, adjusting settings).
- **Efficiency:** The UI must render updates in real-time, with a frame rate of at least 20 frames per second (FPS) to ensure smooth transitions and animations during gameplay.

2) System Performance:

- **Response Time:** The average response time for UI interactions must not exceed 150 milliseconds, providing immediate feedback to players when they interact with menus or HUD elements.
- **Resource Utilization:** The UI rendering and updates should utilize no more than 25% of CPU resources during peak loads to allow other game processes to function efficiently.

3) Constraints:

- **Hardware Limitations:** The UI must be designed to function across a range of resolutions, supporting at least 720p displays to ensure accessibility for players using various devices.
- **Software Dependencies:** The UI must be compatible with Godot version 3.3 or later and utilize the built-in UI framework efficiently to ensure integration with the game's state management systems.

C. Temporary and Permanent Progression

Temporary progress preservation and permanent progression will both be implemented within the game to give players a sense of accomplishment without the fear of losing all progression and achievements throughout their gameplay. These features are specifically designed to accommodate the mechanics of a roguelite game, which inherently punish players by returning them to the beginning upon dying or forfeiting during a run, thereby ensuring that they cannot circumvent the punitive nature of "permadeath."

1) Quality Attributes:

- **Reliability:** The temporary session preservation system must achieve a reliability rate of 95%, ensuring that players can save and load their progress without encountering errors that may result in data loss.
- **Usability:** At least 80% of players should find the temporary save and load functionality easy to understand and use without requiring external help or documentation.
- **Efficiency:** The serialization and deserialization processes for temporary saves should complete within 500 milliseconds to minimize interruption to the player's experience during gameplay.

2) System Performance:

- **Response Time:** The system should ensure that loading a temporary save does not exceed a 15-second wait time, allowing for a smooth transition back into gameplay.
- **Resource Utilization:** The serialization process should utilize no more than 10% of CPU resources during peak gameplay to ensure that other critical processes remain unaffected.

3) Constraints:

- **Data Integrity:** The temporary save files must be encrypted to prevent tampering or corruption, ensuring that player data remains secure throughout their gaming experience.
- **Compatibility:** The permanent progression system must maintain compatibility with various game versions, ensuring that players can access their saved data across different sessions and updates without issue.

D. Upgradable Hub Area

The hub area serves as a central feature within the game, acting both as a focal point for player interaction and a representation of their achievements and progression throughout the game.

1) Quality Attributes:

- **Reliability:** The hub upgrade system must achieve a reliability rate of 80%, ensuring that players can successfully unlock and utilize upgrades without encountering critical errors.
- **Usability:** At least 75% of players should be able to navigate the upgrade options and understand the benefits of each enhancement within three minutes of entering the hub.
- **Efficiency:** The hub should load within 15 seconds upon entering from any dungeon or area to provide a seamless experience without frustrating delays.

2) System Performance:

- **Response Time:** Actions related to hub upgrades, such as unlocking or interacting with buildings, should respond within 150 milliseconds to maintain player engagement.
- **Throughput:** The hub should accommodate a minimum of 30 concurrent NPC interactions when completely populated without noticeable performance drops, ensuring a lively and interactive environment for players.
- **Resource Utilization:** The hub's rendering and upgrade processes should utilize no more than 15% of CPU and GPU resources to allow for other game functions to operate smoothly.

3) Constraints:

- **Hardware Limitations:** The hub area must be designed to support minimum hardware specifications while delivering consistent experiences on higher-end systems.
- **Software Dependencies:** The upgrade system must seamlessly integrate with Godot version 3.3 or later, ensuring compatibility with other game features and the underlying game state management.

E. LAN Multiplayer

LAN Multiplayer will enable players to engage in cooperative gameplay by connecting locally through a robust networking system designed to handle various player interactions and connections.

1) Quality Attributes:

- **Reliability:** The LAN multiplayer system must achieve a reliability rate of 90%, ensuring that most sessions can be established and maintained without critical errors or disconnects.
- **Usability:** At least 75% of players should be able to initiate or join a multiplayer session within three minutes of entering the game, indicating a user-friendly interface and process.
- **Efficiency:** The synchronization of game states among players should occur within 500 milliseconds to minimize delays during gameplay and maintain a seamless experience.

2) System Performance:

- **Response Time:** Player actions (e.g., attacks, movement) should reflect within 300 milliseconds on all connected clients to ensure real-time interaction during combat.
- **Throughput:** The multiplayer system should handle a minimum of 10 concurrent player connections without performance degradation, ensuring a stable environment even during peak gameplay.
- **Resource Utilization:** The networking processes should utilize no more than 10% of CPU resources during gameplay, allowing for other game mechanics to function smoothly.

3) Constraints:

- **Network Limitations:** The multiplayer feature must function effectively over local area networks with varying speeds, accommodating a minimum bandwidth of 1 Mbps to ensure connectivity.
- **Software Dependencies:** The LAN multiplayer system must be compatible with the current version of Godot being used for the project, ensuring that any networking libraries or plugins utilized do not conflict with game updates.

F. Player Combat

The combat system will encompass three integral features: Ability Inheritance, Combat System, and Health Management, each designed to enhance the dynamic and immersive nature of gameplay. Together, these features form a cohesive system that promotes exploration, strategic gameplay, and an engaging combat experience, ensuring that players are equipped to face the diverse challenges within the game.

1) Quality Attributes:

- **Reliability:** The combat system must achieve a reliability rate of 95%, ensuring that combat mechanics function correctly without bugs that disrupt gameplay.
- **Usability:** At least 80% of players should be able to perform combat actions without needing to consult documentation or tutorials, indicating an intuitive control scheme.
- **Efficiency:** The combat actions (melee and ranged) should execute within 200 milliseconds to provide immediate feedback to players, enhancing the responsiveness of gameplay.

2) System Performance:

- **Response Time:** The system should calculate and display damage dealt within 150 milliseconds after an attack is executed to maintain a fluid combat experience.
- **Throughput:** The combat system should handle simultaneous attacks from up to 5 players in a multiplayer setting without noticeable lag or performance drops.
- **Resource Utilization:** The combat calculations should utilize no more than 8% of CPU resources during peak action moments to allow for smooth gameplay overall.

3) Constraints:

- **Hardware Limitations:** The combat system must perform adequately on a range of hardware configurations, supporting minimum specifications while ensuring reasonable performance on higher-end systems.
- **Software Dependencies:** The combat features must integrate smoothly with the existing game framework in Godot, ensuring compatibility with Player Inputs, Health Management, and other systems in use.

G. Level Randomization and Variety

Levels across a run shall be designed to provide players with a fresh and engaging experience each time they embark on a dungeon run. This system encompasses three key features: Level Randomization, Level Differentiation, and Environmental VFX, each contributing to the dynamic nature of gameplay.

1) Level Randomization: To randomize which level a player will be dropped into, a shuffling mechanism shall be designed to select from a set of hardcoded premade levels. This approach ensures that each playthrough presents a unique combination of layouts and challenges, maintaining high-quality gameplay while preventing predictability. The system will utilize premade level data as inputs to shuffle and randomly select levels for each session. It is essential that this process also ensures a balanced difficulty progression to cater to players of varying skill levels.

Quality Attributes

- **Reliability:** The shuffling mechanism must achieve a reliability rate of 90% in selecting unique levels for each playthrough to ensure a varied gameplay experience without repetition.
- **Usability:** At least 80% of players should perceive the level transitions as smooth and engaging, contributing to overall gameplay satisfaction.
- **Efficiency:** The level selection and shuffling process should execute within 200 milliseconds to minimize delays before gameplay begins, ensuring a fluid experience for players.

System Performance

- **Response Time:** The average response time for loading a shuffled level must not exceed 15 seconds to maintain player engagement and reduce frustration during transitions.
- **Resource Utilization:** The shuffling algorithm should utilize no more than 15% of CPU resources to allow for other game processes to function efficiently during level loading.

Constraints

- **Data Integrity:** The randomization system must ensure that all levels maintain their intended difficulty balance, preventing overly challenging or easy levels from being presented to the player.
- **Compatibility:** The level randomization feature must be compatible with the existing game structure and adhere to the design constraints set for each level's layout and mechanics.

2) Level Differentiation and Environmental VFX: To enrich player engagement, each game level will possess distinct visual themes and unique challenges through the integration of varying enemy types, environmental hazards, and specific objectives.

Quality Attributes

- **Reliability:** The environmental differentiation system must achieve a reliability rate of 85%, ensuring that each level consistently presents unique visual elements and challenges.
- **Usability:** At least 75% of players should report clarity and engagement with level objectives and environmental features, contributing to a positive gameplay experience.
- **Efficiency:** The system must apply visual effects in real-time with minimal latency, achieving updates within 100 milliseconds during gameplay to enhance immersion.

System Performance

- **Response Time:** Environmental VFX changes should occur within 200 milliseconds based on player interactions or changes in game state, ensuring a responsive environment.
- **Resource Utilization:** The system should ensure that environmental effects consume no more than 20% of

GPU resources during peak gameplay to maintain overall performance.

Constraints

- **Hardware Limitations:** The environmental differentiation must accommodate minimum hardware specifications, ensuring that players on lower-end devices can still experience dynamic environments.
- **Software Dependencies:** The system must integrate with Godot's visual effects capabilities while ensuring compatibility with version 3.3 or later.

H. Enemy Variety and Types

The game will incorporate a diverse range of enemy types, each possessing unique abilities, strengths, and weaknesses to encourage strategic gameplay.

1) Quality Attributes:

- **Reliability:** The enemy variety system must maintain a reliability rate of 90% in presenting diverse enemy encounters without repeating enemy types within the same run.
- **Usability:** At least 80% of players should find the unique enemy behaviors clear and intuitive, allowing for effective strategy development during encounters.
- **Efficiency:** The system should generate and spawn enemy types with a maximum delay of 300 milliseconds to maintain a smooth gameplay flow.

2) System Performance:

- **Response Time:** Enemy spawns and behaviors should execute within 200 milliseconds to prevent player frustration and maintain engagement during gameplay.
- **Throughput:** The system should support up to 20 concurrent enemy types on screen without performance degradation, ensuring a challenging combat environment.
- **Resource Utilization:** Enemy behavior processing should utilize no more than 15% of CPU resources during peak action sequences to allow for smooth game performance.

3) Constraints:

- **Balancing Needs:** The enemy variety must adhere to balancing constraints to ensure no single enemy type becomes overpowering, maintaining fairness in gameplay.
- **Compatibility:** The enemy variety system must integrate seamlessly with existing combat mechanics and overall game structure to enhance rather than disrupt gameplay.

I. Difficulty Scaling

To ensure a continuously challenging experience for players, the game will implement a difficulty scaling system where enemies will adjust their difficulty based on player progress and performance.

1) Quality Attributes:

- **Reliability:** The difficulty scaling system must maintain a reliability rate of 90% in adapting enemy difficulty accurately based on player performance and progress.
- **Usability:** At least 75% of players should find the difficulty adjustments logical and satisfying, contributing to a sense of accomplishment and engagement.
- **Efficiency:** The system should process difficulty adjustments within 150 milliseconds to ensure a smooth gameplay experience without noticeable interruptions.

2) System Performance:

- **Response Time:** Difficulty adjustments should be reflected in enemy behavior within 200 milliseconds after a player completes a significant action, ensuring immediate feedback on player performance.
- **Throughput:** The system should support adjustments for up to 10 enemies simultaneously without degrading performance, ensuring that scaling remains effective during encounters.

3) Constraints:

- **Player Feedback:** The scaling system must be transparent, providing players with feedback on difficulty adjustments to enhance their understanding of the game mechanics.
- **Compatibility:** The difficulty scaling feature must integrate with the enemy variety system and overall game design to ensure coherent and balanced gameplay experiences.

J. Environmental Interactions

1) Quality Attributes:

- **Reliability:** The environmental interactions system must ensure that interactions with destructible objects and traps occur reliably in at least 95% of instances during gameplay, without causing any game-breaking bugs or crashes.
- **Usability:** Players should be able to identify and interact with destructible objects and traps with at least 90% success rate on the first attempt, as indicated by user testing and feedback. Tooltips or indicators should also provide clear information about interactions.
- **Efficiency:** The environmental interactions system should not cause the game's frame rate to drop below 30 frames per second (FPS) during interactions, ensuring smooth gameplay even with multiple objects on screen.
- **Maintainability:** The codebase for environmental interactions should be modular, allowing for new objects and traps to be added with minimal changes. At least 80% of the code should follow best practices for modular design and documentation.
- **Security:** The environmental interactions system should prevent unintended exploitation, ensuring that players cannot trigger traps or destroy objects through unintended means, with a target of 0 security vulnerabilities identified during testing.

2) System Performance:

- **Response Time:** The system must register player interactions with environmental objects within 100 milliseconds to maintain a fluid gameplay experience. This should be tested across various hardware configurations.
- **Throughput:** The system should handle up to 50 simultaneous environmental interactions without performance

degradation, maintaining a minimum frame rate of 30 FPS during gameplay.
- **Resource Utilization:** CPU and memory usage for environmental interactions should remain below 30% during peak usage, ensuring that resources are available for other game systems.

3) Constraints:

- **Hardware Limitations:** The game must perform well on hardware with at least 4GB of RAM and a dual-core processor, accommodating users with lower-end systems. Performance should be validated on a range of configurations.
- **Software Dependencies:** The environmental interactions system will depend on Godot's physics engine; therefore, the game must be compatible with Godot version 3.4 or higher.
- **Regulatory Requirements:** All interactions should adhere to safety and usability standards for gaming, ensuring that no content is inappropriate or harmful to players.

K. Inventory and Loot

The game will incorporate a robust inventory system that allows players to collect, manage, and utilize various items and loot found throughout their adventures.

1) Quality Attributes:

- **Reliability:** The inventory system must correctly track and display item quantities with 99% accuracy across different gameplay sessions, ensuring that no items are lost or duplicated.
- **Usability:** Players should be able to access and navigate the inventory with an average time of no more than 3 seconds to find specific items, as evaluated through user testing.
- **Efficiency:** The inventory system must load within 200 milliseconds when accessed, ensuring minimal disruption to gameplay.
- **Maintainability:** The inventory code should allow for easy addition of new item types, with at least 75% of the codebase following maintainability standards.
- **Security:** The inventory system must prevent item duplication exploits, ensuring that players can only obtain items through legitimate gameplay means, with a target of 0 identified vulnerabilities.

2) System Performance:

- **Response Time:** Inventory actions (such as adding or using items) must register within 100 milliseconds to maintain a smooth user experience.
- **Throughput:** The system should handle the management of at least 200 unique items simultaneously without noticeable delays or performance drops.
- **Resource Utilization:** Memory usage for the inventory system should not exceed 20MB when fully loaded, ensuring that resources are efficiently managed during gameplay.

3) Constraints:

- **Hardware Limitations:** The inventory system should function efficiently on devices with a minimum of 4GB of RAM, ensuring that players with varying hardware can still enjoy the game.
- **Software Dependencies:** The inventory system will rely on Godot's data structures for item management, necessitating compatibility with Godot version 3.4 or higher.
- **Regulatory Requirements:** The inventory system must comply with any relevant data protection laws, ensuring player data regarding item collection and usage is handled securely.

L. Sound Design & Audio Feedback

1) Quality Attributes:

- **Reliability:** The sound design system must deliver audio feedback with a 99% success rate during gameplay, ensuring that all sound effects and cues trigger correctly.
- **Usability:** Players should be able to distinguish audio cues and sound effects with a 90% accuracy in identifying the corresponding game events during playtesting.
- **Efficiency:** The audio system must not introduce latency greater than 50 milliseconds in sound playback, ensuring that audio feedback is immediate and enhances gameplay.
- **Maintainability:** The sound assets and code should be organized for easy updates, allowing for new sounds to be added with minimal changes. At least 80% of the audio-related code should follow maintainability best practices.

2) System Performance:

- **Response Time:** The audio playback system must respond to game events within 100 milliseconds to maintain immersion.
- **Throughput:** The system should support up to 20 simultaneous audio channels without performance degradation, maintaining a consistent audio experience during intense gameplay.
- **Resource Utilization:** Memory usage for sound assets should remain below 50MB when fully loaded to ensure efficient resource management and avoid impacting gameplay performance.

3) Constraints:

- **Hardware Limitations:** The audio system should function optimally on devices with at least 4GB of RAM and integrated audio, accommodating a broad range of player hardware.
- **Software Dependencies:** The sound design system will rely on Godot's audio engine, necessitating compatibility with Godot version 3.4 or higher.
- **Regulatory Requirements:** The audio content must comply with copyright regulations, ensuring all sound assets are either original or properly licensed for use in the game.

M. Tutorial/Help System

To ensure the player is eased into the main mechanics of the game, a tutorial system should be implemented to teach them the controls and how to navigate the main gameplay loop.

1) Quality Attributes:

- **Reliability:** The tutorial system must ensure that 99% of players can complete the tutorial without encountering any bugs or interruptions.
- **Usability:** Players should complete the tutorial within an average time of 5 minutes, indicating that the instructional design is clear and effective.
- **Efficiency:** The system should load and present tutorial content within 150 milliseconds to prevent disruptions during player onboarding.
- **Maintainability:** The tutorial code should be modular, allowing for easy updates and additions, with at least 75% of the codebase documented for clarity.
- **Security:** Player data regarding tutorial progress should be securely stored and accessible only to authorized components of the game, preventing data breaches or losses.

2) System Performance:

- **Response Time:** The tutorial prompts must display within 100 milliseconds after a player's interaction to maintain engagement.
- **Throughput:** The system should manage up to 10 simultaneous tutorial sessions without performance issues, ensuring that the game remains responsive.
- **Resource Utilization:** Memory usage for the tutorial system should not exceed 20MB when fully loaded, ensuring that resources are efficiently managed during gameplay.

3) Constraints:

- **Hardware Limitations:** The tutorial system should operate effectively on devices with a minimum of 4GB of RAM to accommodate players with varying hardware.
- **Software Dependencies:** The tutorial system will depend on Godot's UI components, requiring compatibility with Godot version 3.4 or higher.
- **Regulatory Requirements:** The tutorial content must comply with relevant accessibility standards, ensuring that it is usable by players with disabilities.

N. 2D Spritework

1) Quality Attributes:

- **Reliability:** The sprite rendering system must ensure that 99% of sprites load correctly without errors during gameplay.
- **Usability:** Players should easily recognize and differentiate between at least 90% of the sprites during playtesting, indicating effective design.
- **Efficiency:** The sprite system should maintain a frame rate of at least 30 FPS during peak usage with multiple sprites on screen, ensuring smooth gameplay.
- **Maintainability:** The sprite assets and animation code should be organized to facilitate updates, with at least 75% of the assets documented for ease of future modifications.

2) System Performance:

- **Response Time:** The sprite loading time must not exceed 200 milliseconds for any given sprite to ensure minimal disruption during gameplay.
- **Throughput:** The system should handle the rendering of at least 100 unique sprites simultaneously without significant performance drops.
- **Resource Utilization:** Memory usage for sprite assets should not exceed 40MB when fully loaded to maintain optimal performance.

3) Constraints:

- **Hardware Limitations:** The sprite rendering system should perform well on devices with at least 4GB of RAM to accommodate players with varying hardware capabilities.
- **Software Dependencies:** The sprite system will rely on Godot's rendering engine, requiring compatibility with Godot version 3.4 or higher.
- **Regulatory Requirements:** The sprite assets must adhere to copyright regulations, ensuring all visual content is either original or properly licensed for use in the game.