

Game Description

The project is a 2D pixel art action side-scroller that combines light platforming with a strong focus on combat and boss battles. Set in a medieval-inspired world, the player takes on the role of a knight on a quest to free his hometown from invading forces. The game follows a linear, story-driven progression across three main levels, each divided into three smaller replayable sections featuring checkpoints and a guaranteed boss encounter, a total of nine small levels and three main bosses. While the overall scope of the game is compact, its increasing difficulty and challenging enemy design provide a sense of depth and longevity. Players must learn and adapt to each boss's unique attack patterns, resulting in multiple attempts before success and making every victory feel rewarding. The gameplay loop emphasizes skill mastery, timing, and strategic movement, offering a satisfying experience for players who enjoy fast-paced combat and classic pixel art aesthetics within a medieval fantasy environment.

User Requirements

A. Game Session Management

- The system shall allow players to start a new game session, creating a fresh progression state.
- The system shall provide a save feature that automatically preserves game progress when exiting the application.
- The system shall automatically load saved game data upon application startup, restoring the player's previous progression state.

B. Level Progression System

- The system shall implement a level selection interface where players can choose from unlocked levels.

- The system shall track level completion status, marking completed levels as available for replay.
- The system shall enforce progressive unlocking, where completing a level unlocks subsequent levels in the game world.
- The system shall display clear visual indicators for locked and unlocked levels in the level selection menu.

C. Character and Gameplay Systems

- The system shall implement a player character with health management capabilities.
- The system shall provide difficulty settings affecting character health points.
- The player character shall respond to player input for movement, actions, and interactions with the game world.
- The system shall maintain character state persistence between level transitions and game sessions.

D. Audio and Accessibility Settings

- The system shall implement separate audio controls for music and sound effects (SFX).
- The system shall offer display settings including fullscreen/windowed mode toggle.
- All settings modifications shall apply immediately and persist between game sessions.
- The system shall include an "Easy Mode" difficulty option that adjusts gameplay parameters (increased player health).

E. User Interface and Navigation

- The system shall provide a main menu interface with clear navigation options.
- The system shall implement responsive menu controls including sliders for audio settings and checkboxes for toggle options.
- The system shall display current settings states clearly in the interface (slider positions, checkbox status).
- The system shall provide visual feedback for all user interactions (button highlights, slider movements).
- The system shall include a "Credits" option in the main menu, allowing users to view game contributors and acknowledgments at any time.

F. Game State Persistence

- The system shall save and load player progression including completed levels and unlocked content.
- The system shall preserve audio settings (volume levels) across game sessions.
- The system shall maintain display preferences (fullscreen/windowed mode) between application launches.
- The system shall remember difficulty settings and apply them consistently during gameplay.

G. Performance and Reliability

- The system shall maintain stable frame rates during gameplay across supported hardware.
- The system shall handle scene transitions smoothly without performance degradation.
- The system shall implement error handling for save/load operations with appropriate user feedback.
- The system shall provide fallback defaults when saved data is corrupted or unavailable.

H. Cross-Platform Compatibility

- The system shall maintain consistent functionality across Windows, Mac, and Linux platforms.
- The system shall adapt control schemes appropriately for different input methods.
- The system shall respect platform-specific conventions for settings storage and file management.

I. Game World and Content

- The system shall provide multiple distinct levels with increasing difficulty progression.
- The system shall implement consistent game mechanics across all levels.
- The system shall maintain visual and audio theme consistency throughout the game experience.
- The system shall provide clear feedback for player actions and game state changes.

J. Save System Security

- The system shall implement validation checks for saved game data integrity.
- The system shall prevent save data corruption through proper file handling procedures.

VI. System Architecture

The system architecture of the proposed 2D Godot game follows a modular, scene-based, and layered design approach tailored for an offline single-player experience. The architecture leverages Godot Engine's node-scene system and built-in resource management to ensure separation of concerns, maintainability, and scalability while remaining lightweight. Since the game operates fully offline, no client-server or online database architecture is employed. All data persistence is handled locally through structured configuration files.

A. Overall Architectural Style

The game adopts a Scene-Oriented and Layered Architecture:

- **Scene-Oriented:** Each major part of the game (menus, levels, player, enemies) is encapsulated in independent scenes (.tscn) connected to their respective scripts (.gd).
- **Layered:** Responsibilities are logically divided into Presentation, Game Logic, and Data Persistence layers.

This approach ensures that gameplay logic, user interface, and save/load mechanisms remain loosely coupled while interacting through clearly defined interfaces.

B. Layered Architecture

1. Presentation Layer (User Interface & Visual Elements)

This layer represents all user-facing components and visual elements of the game. It is responsible for rendering graphics, displaying menus, and handling player interactions.

Key components include:

- **Main Menu Scene (main_menu.tscn):** Entry point of the game where the player can start the game, access options, view controls, check credits, or exit the application.
- **Options Menu:** Allows players to modify volume levels, window mode, and difficulty settings with immediate visual feedback.
- **Level Selection Interface:** Displays unlocked and locked levels using visual indicators.
- **In-Game GUI:** Implemented using CanvasLayer nodes to display HUD elements such as health, ability bars, and boss health bar.

- Visual Assets: TileMaps for level design, sprites, animations, and background layers.

This layer communicates user actions (button presses, slider changes) to the game logic layer through connected GDScript functions.

2. Game Logic Layer (Core Gameplay Systems)

The game logic layer contains the core mechanics and rules governing gameplay behavior. Each scene is associated with a script that controls its behavior.

Key components include:

- Player System: Implemented using CharacterBody2D, handling movement, combat, health, and interactions.
- Enemy and Boss Systems: Enemy.tscn and Boss.tscn scenes manage AI behavior, attack patterns, and collision handling.
- Level System: Each playable level scene includes TileMaps, collision layers, kill zones (Area2D), NPC boundaries, audio players, and checkpoints.
- Scene Management: Scene transitions are handled using `get_tree().change_scene_to_file()`, enabling smooth navigation between menus and levels.
- Game Rules: Difficulty settings dynamically adjust parameters such as player health.

This layer processes input events, updates game state, and triggers save operations when progression milestones are reached.

3. Data Persistence Layer (Save & Load System)

The data persistence layer is responsible for storing and retrieving all game-related data locally.

Key components include:

- SaveManager.gd: A globally accessible script (autoload) that stores global variables and exposes functions for saving and loading data.
- ConfigFile System: Godot's built-in ConfigFile helper class is used to read and write structured data to a local file.
- Local Save File: The game uses a platform-independent path (`user://game_settings.cfg`) to store progression, settings, and preferences.

Saved data includes:

- Level progression

- Audio settings (music and SFX volumes)
- Display preferences (fullscreen/windowed mode)
- Difficulty settings

The save file is loaded automatically on application startup. Any change in options or level completion triggers an immediate save operation to ensure data consistency.

C. Scene-Based Architecture

Godot scenes act as self-contained modules composed of nodes such as:

- CanvasLayer for UI and background rendering
- TileMap for pixel-precise level design
- Area2D for kill zones and triggers
- CharacterBody2D for player, enemies, and bosses
- AudioStreamPlayer for music and sound effects

Scenes are reusable and instanced when needed, promoting modularity and reducing duplication.

D. Offline Execution Model

The game operates entirely offline with no external dependencies:

- No online services or databases are used
- All processing occurs locally on the player's machine
- Save files are stored according to the operating system's standard user data directories

This ensures fast performance, minimal latency, and compatibility across Windows, macOS, and Linux platforms.

E. Reliability and Data Integrity

To ensure robustness, the architecture includes:

- Validation checks when loading save files
- Fallback default values if the save file is missing or corrupted
- Controlled write operations to prevent partial file corruption

These mechanisms guarantee a stable user experience even in cases of unexpected shutdowns or file errors.

In summary, the system architecture combines Godot's scene-based model with a layered offline design, providing a clean separation between presentation, gameplay logic, and data persistence while ensuring reliability, modularity, and cross-platform compatibility.