**1. Introduction**

This document is the software design for our video game project, which was built using the Godot Engine and integrated DevOps practices. Its purpose is to guide the software development lifecycle by providing a structured plan that aligns technical and collaborative workflows. This phase is crucial because it sets the foundation for developing, testing, deploying, and maintaining the application.

Our game will run on PCs with Windows and Linux operating systems and will be lightweight enough to operate on systems with integrated graphics. The development will utilize **Godot 4.x**, with **GDScript** as the main language. The project will also include tools such as GitHub (for version control and CI/CD), Discord (for communication), and Trello (for task tracking). Automated testing will be implemented using **GDUnit3**, and GitHub Actions will manage the build and test pipelines.

A computer screen shot of a computer

AI-generated content may be incorrect.**2. Entity Relationship Diagram (ERD)**

**Entity Relationship Diagram (ERD)**

This ERD shows how different things in the game—like players, enemies, and levels—connect and talk to each other. It’s like a map of friendships. Each box is a character (like “Player” or “Enemy”), and the lines show how they are related. It helps us organize game data clearly and avoid confusion.

**3. Sequence Diagrams (UML)**

A diagram of a game

AI-generated content may be incorrect.

**Sequence Diagram (UML)**

The sequence diagram shows the order of actions. It explains who does what and when. Characters (like Players or Game Managers) talk to each other step-by-step. It’s great for showing how game actions happen in time—from pressing a button to what the game does next. It's super helpful for planning.

**4. Architecture Design Diagram**

A diagram of a computer

AI-generated content may be incorrect.

**Architecture Design Diagram**

This diagram is like a bird' s-eye view of the game’s brain. It shows how big parts of the game (such as sound, graphics, and logic) work together. Each box has a special job, and arrows show how they talk. It helps developers build, connect, and update parts without breaking them. It's like building blocks for games.

**5. Class Diagram**

A screenshot of a computer

AI-generated content may be incorrect.

**Class Diagram**

The class diagram is a blueprint for the game's characters, items, and everything. Each box is a class with powers (methods) and secrets (variables). It shows who can do what and who gets what. It helps us reuse code, fix bugs fast, and make everything work like a team.

6. Database Design

The database design organizes data according to a database model, determining what data must be stored and how the data elements interrelate.

**Entities and Attributes**

1. **Players**
   * **player\_id** (Primary Key): Unique identifier (string)
   * **username**: Player's name (string)
   * **high\_score**: Best score achieved (integer)
   * **last\_level\_unlocked**: Last unlocked level (string)
2. **Levels**
   * **level\_id** (Primary Key): Unique identifier (string)
   * **level\_name**: Name of the level (string)
   * **difficulty**: Difficulty level (integer)
   * **layout\_data**: Serialized level layout (JSON string)
3. **LightSources**
   * **light\_id** (Primary Key): Unique identifier (string)
   * **level\_id** (Foreign Key): Reference to Levels (string)
   * **position\_x**: X-coordinate (float)
   * **position\_y**: Y-coordinate (float)
   * **type**: Type of light (string)
   * **size**: Radius or size (float)
4. **Enemies**
   * **enemy\_id** (Primary Key): Unique identifier (string)
   * **level\_id** (Foreign Key): Reference to Levels (string)
   * **position\_x**: X-coordinate (float)
   * **position\_y**: Y-coordinate (float)
   * **patrol\_path**: Serialized path data (JSON string)

**Relationships**

* Each **Level** can have multiple **LightSources** and **Enemies** (one-to-many relationship).
* **Players** have no direct relationships but interact with levels.

7. Interface Design

UI design focuses on information architecture, building interfaces that clearly communicate what's important.

**Game Screens**

1. **Main Menu**
   * Options: Start Game, Load Game, Settings, Exit
   * Background artwork related to the game theme.
   * Highlighted buttons for easy navigation.
2. **Gameplay Screen**
   * Game area (level layout).
   * Player’s shadow sprite.
   * Light sources and enemies represented in the game world.
   * Pause Button (top right corner).
3. **HUD (Heads-Up Display)**
   * Score Display: Current score (upper left).
   * Lives Remaining: Visual representation (hearts or icons).
   * Timer (optional): If levels are time-based.
4. **Game Over Screen**
   * Final score.
   * Options: Restart, Return to Main Menu, Exit.
5. **Settings Menu**
   * Options: Sound, music volume, graphics quality.
   * Key bindings for control customization.

**Color Palette and Graphics**

* Dark, atmospheric colors with contrasting bright lights.
* Consistent style for icons and fonts.
* Smooth transitions between screens and animations for UI elements.

8. Use Cases

Use cases describe how users interact with the application.

1. **Start New Game**
   * User navigates to the main menu and selects "Start Game."
   * The game loads the first level.
2. **Load Game**
   * User selects "Load Game" from the main menu.
   * The game displays saved games.
   * User selects a saved game.
   * The game loads the selected level.
3. **Move Shadow**
   * User presses movement keys (e.g., arrow keys).
   * The shadow moves within the game environment.
4. **Interact with Light Source**
   * User approaches a light source.
   * User presses an interaction key (e.g., E).
   * The light source toggles on/off or moves.
5. **Avoid Enemy Detection**
   * User navigates the shadow past an enemy.
   * The enemy does not detect the shadow.
6. **Get Detected by Enemy**
   * User moves the shadow into an enemy's detection range.
   * The enemy detects the shadow, and the player loses a life.
7. **Complete Level**
   * User navigates the shadow to the exit point.
   * The game loads the next level or displays a completion message.
8. **Game Over**
   * User loses all their lives.
   * The game displays the "Game Over" screen.
9. **Adjust Settings**
   * User navigates to the settings menu.
   * User adjusts volume, graphics, or key bindings.
   * The game saves the settings.

9. Test Cases

Test cases validate that the software meets the requirements.

1. **Start New Game Test**
   * Requirement: The game should start a new game from the main menu.
   * Test Case: Verify that selecting "Start Game" loads the first level correctly.
2. **Load Game Test**
   * Requirement: The game should load a saved game.
   * Test Case: Verify that selecting a saved game loads the correct level and player state.
3. **Shadow Movement Test**
   * Requirement: The shadow should move in response to user input.
   * Test Case: Verify that pressing movement keys moves the shadow in the correct direction.
4. **Light Source Interaction Test**
   * Requirement: The player should be able to interact with light sources.
   * Test Case: Verify that pressing the interaction key toggles a light source on/off.
5. **Enemy Detection Test**
   * Requirement: Enemies should detect the shadow within their detection range.
   * Test Case: Verify that moving the shadow into an enemy's range results in detection and life loss.
6. **Level Completion Test**
   * Requirement: The game should load the next level upon completion.
   * Test Case: Verify that reaching the exit point loads the next level.
7. **Game Over Test**
   * Requirement: The game should display the "Game Over" screen when all lives are lost.
   * Test Case: Verify that losing all lives displays the "Game Over" screen.
8. **Settings Adjustment Test**
   * Requirement: The game should save and load settings.
   * Test Case: Verify that adjusting volume and graphics settings saves the changes and applies them correctly.

10. Summary

This design document explains clearly how our game works. It is made with the GODOT engine which is like Unity 3D. This includes the plan for how the game works, and how players interact with the UI. It also includes how we will build it and test it. The doc also describes the tools that we are using, the game screen, data, and how different parts of the games are connected.