A PROJECT REPORT

On

The Quest for the Lost Artifact

Submitted by

Yash Shukla

in partial fulfilment for the award of the degree

Of

BACHLEOR OF SCIENCE

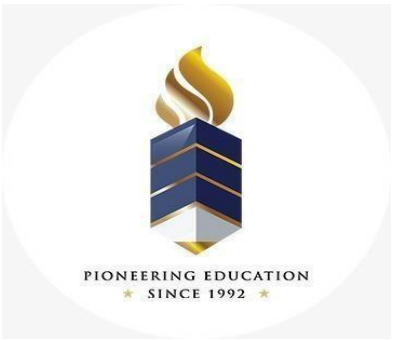
In

COMPUTER SCIENCE

Under the guidance of

Asst. Prof Rohit Tiwari

Department of Computer Science



Shree L.R. Tiwari Degree College of Arts, Commerce & Science

(Sem V)

(2024 - 2025)

Shree L.R. Tiwari Degree College of Arts, Commerce & Science

Mira road (E), Thane - 401107

Department of Computer Science

CERTIFICATE

This is to certify that **Mr. Yash Shukla** of **T.Y.B.Sc. (Sem V)** class has satisfactorily completed the Project _____, to be submitted in the partial fulfilment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2024 – 2025.**

Date of Submission:

Project Guide **Head/Incharge**

Department of Computer Science

College Seal **Signature of Examiner**

DECLARATION

I, Yash Shukla hereby declare that the project entitled "The Quest for the Lost Artifact" submitted in the partial fulfilment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2024 – 2025** is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles

Signature of the Student:

Name of the Student:

Place:

Date:

Acknowledgement

I would like to extend my sincere gratitude to my mentors for their invaluable guidance, advice, and encouragement throughout the development of The Quest for the Lost Artifact. Their insights were crucial to the successful completion of this project. I am also grateful to the Pygame community for providing extensive resources and tutorials, which helped me navigate challenges. Special thanks to my friends and family for their unwavering support and constructive feedback, which played a key role in refining the game. Lastly, I appreciate everyone who believed in this project and motivated me to push through to its completion.

Table of Contents

Abstract

This project centres on the design and development of a 2D platformer game using Python and the Pygame library, aiming to create an engaging and immersive user experience. The game features a total of 12 levels, including four secret levels that introduce additional challenges and rewards for players who successfully unlock them. Each level is designed with unique obstacles, such as moving platforms, spikes, and other environmental hazards, ensuring a gradual increase in difficulty suitable for both novice and experienced gamers.

A standout feature is the implementation of intelligent enemy AI, which reacts dynamically to player actions, providing a more interactive gameplay experience. Enemies may patrol areas, chase the player, or ambush them unexpectedly, creating unpredictable encounters. Additionally, players must navigate various dynamic traps, including falling spikes and collapsing platforms, strategically placed to test reflexes and decision-making skills.

The game incorporates a user-friendly interface, allowing players to easily access options such as "Start Game," "Load Game," and "Settings." Performance optimizations ensure smooth gameplay across different platforms, with techniques like spatial partitioning enhancing resource management. Through iterative testing and feedback, the game has been refined, resulting in a fun, challenging adventure with significant replay value due to its secret levels and intelligent enemy design.

Introduction

Storyline: The Quest for the Lost Artifact

In the ancient and enchanted land of Eldoria, peace once reigned under the protection of a powerful relic known as the "Eye of Serenity." This artifact, forged by the gods themselves, was said to possess the ability to maintain balance in the world by keeping darkness at bay. For centuries, the kingdoms of Eldoria flourished under its influence, and its presence ensured harmony among both humans and magical creatures. However, this period of tranquility came to an abrupt end when the Eye of Serenity mysteriously vanished.

Without the Eye's protection, chaos began to spread across the land. The balance was shattered, and a sinister force known as the Shadowlord rose from the depths of darkness. His ambition was clear: to find the Eye of Serenity and harness its immense power for his own malevolent purposes. Under his influence, darkness began to consume the kingdoms of Eldoria, and hope dwindled as the Shadowlord's army of shadowy creatures laid waste to cities and villages alike.

Amid the despair, a young adventurer named Alex emerges as an unlikely hero. Alex's village was one of many destroyed by the Shadowlord's forces, leaving only whispers of the legendary Eye of Serenity as the key to restoring peace. With nothing left to lose, Alex embarks on a perilous quest to recover the lost artifact before the Shadowlord can claim it. Armed with only a sword and an unwavering sense of justice, Alex journeys across Eldoria, from dense, enchanted forests to ancient ruins buried deep within forgotten temples. Along the way, Alex encounters wise sages, mystical creatures, and ancient guardians who offer both guidance and challenges.

As the adventure unfolds, Alex discovers that the quest is about more than just retrieving a powerful relic. It's a journey of self-discovery, as secrets about Alex's own lineage and destiny come to light. The fate of Eldoria rests in hands. Will the Eye of Serenity be found in time, or will darkness forever engulf the land? Only by outsmarting the Shadowlord, solving ancient puzzles, and braving untold dangers can Alex hope to restore balance and peace to Eldoria.

Objectives:

- **Create an Engaging Platformer Game:**The primary objective of this project is to develop a 2D platformer game using Pygame, which provides an immersive gaming experience through engaging gameplay, compelling storytelling, and visually appealing environments.
- **Integrate Challenging Gameplay Elements**: The game will feature a variety of gameplay mechanics, including platforming, puzzle-solving, and combat. Each level will be designed to test the player's skill, reflexes, and problem-solving abilities.
- **Build a Rich Game World:** The game will be set in the fantasy world of Eldoria, featuring diverse environments such as enchanted forests, mysterious caves, and ancient ruins. The objective is to create a visually rich and detailed world that enhances the player's experience.
- **Character Development:** The protagonist, Alex, will evolve throughout the game as they face challenges and uncover hidden truths about their past and destiny. The objective is to create a character that players can connect with on an emotional level.
- **Implement a Save/Load System:** To enhance user experience, the game will feature a functional save and load system that allows players to save their progress and resume their journey at any time.
- **Enhance User Interface:** The game will feature an intuitive and accessible user interface, including menus for starting the game, adjusting settings, and managing saved game files.

Scope:

- **Game Development Platform:** The game will be developed using Python and the Pygame library, focusing on creating a playable and engaging platformer within a two-month development timeframe.
- **Target Audience:** The game is designed for casual gamers who enjoy 2D platformers, puzzles, and fantasy-themed adventures. It is suitable for players of all ages.
- **Game Design:** The scope of the game includes multiple levels with increasing difficulty, each containing unique challenges, enemies, and puzzles. The storyline will unfold progressively as players advance through levels.
- **Character and Enemy Design:** Alex, the protagonist, will be the primary playable character, while the Shadowlord's minions will serve as enemies that challenge the player throughout the game.
- **Music and Sound Effects:** Appropriate background music and sound effects will be integrated to enhance the game's atmosphere and overall experience.
- **Post-Launch:** After the initial release, potential updates may include additional levels, characters, and gameplay features based on user feedback and further development opportunities.
- This project aims to deliver a well-rounded gaming experience while demonstrating key concepts in game development, such as programming logic, design principles, and user experience.

Review of Literature

The review of literature is an essential part of any project as it provides the foundation for understanding existing work in the field and identifying gaps that the current project aims to address. For this game development project, focusing on the platformer genre and game design using Pygame, the literature review will examine several key areas: the history and evolution of platformer games, game mechanics, game design principles, the Pygame library, and existing games that use similar technologies. This chapter will cover these topics in depth, allowing for a solid understanding of how “The Quest for the Lost Artifact” fits into the wider gaming landscape.

2.1 History and Evolution of Platformer Games

Platformer games have a long and rich history, evolving significantly over time. The genre first gained prominence in the early 1980s, with the release of games like Donkey Kong (1981) and Super Mario Bros (1985). These early platformers were defined by their simple mechanics, where the player controls a character that moves through levels filled with obstacles, enemies, and platforms. Jumping and navigating between platforms became the hallmark of the genre.

Golden Age of Platformers

The 1980s and 1990s are often referred to as the "Golden Age" of platformer games. Titles like Super Mario Bros, Sonic the Hedgehog, and Mega Man became iconic during this period. These games introduced a variety of game mechanics that would go on to define the genre: power-ups, non-linear level design, hidden secrets, and boss fights. The success of these games helped establish platformers as one of the most popular genres in gaming history.

Shift to 3D Platformers

In the mid-1990s, the gaming industry saw a shift from 2D to 3D platformers with the advent of consoles like the Nintendo 64 and PlayStation. Games like Super Mario 64 and Crash Bandicoot redefined the genre by introducing a new dimension of movement and exploration. This shift allowed for more complex level designs, greater freedom of movement, and deeper immersion in the game world. However, despite the rise of 3D platformers, 2D platformers continued to thrive, especially with the advent of indie games in the 2000s.

Modern Platformers

In recent years, platformers have made a resurgence in the indie game scene. Games like Celeste (2018), Hollow Knight (2017), and Shovel Knight (2014) have shown that there is still a strong demand for 2D platformers. These modern platformers often blend traditional mechanics with innovative gameplay elements, such as precision-based controls, intricate

puzzles, and emotionally-driven narratives. The resurgence of 2D platformers in the indie space has kept the genre alive and well.

2.2 Key Game Mechanics in Platformers

Player Movement and Controls

At the core of any platformer is player movement. In traditional platformers, players navigate through levels by running, jumping, and occasionally attacking enemies. The challenge often lies in precise movement, particularly jumping from platform to platform. In games like Super Mario Bros, timing and control responsiveness are critical to the player's success.

Jumping Mechanics: Jumping is the fundamental mechanic in platformer games. Different games implement jumping mechanics in various ways, such as adding double jumps (Shovel Knight) or wall-jumping (Celeste). In The Quest for the Lost Artifact, jumping will be central to gameplay, requiring players to time their jumps over obstacles and enemies while navigating dangerous terrain.

Gravity and Physics: Many platformer games simulate basic physics, particularly gravity. Games like Celeste and Super Meat Boy emphasize tight control over gravity to give players a sense of weight and momentum as they navigate difficult environments. For The Quest for the Lost Artifact, gravity will be simulated to give the game a realistic sense of movement.

Collision Detection and Response

Collision detection is a critical element in platformers, ensuring that the player interacts with the game environment accurately. Whether it’s landing on a platform, bouncing off walls, or colliding with enemies, collision detection dictates the player’s interaction with the game world.

Pygame provides built-in collision detection functions, such as pygame.sprite.collide_rect() and pygame.sprite.spritecollide(), which will be utilized in The Quest for the Lost Artifact to manage interactions between the player, enemies, and objects.

Enemies and AI Behavior

Enemies are another core mechanic in platformers. Games like Super Mario Bros feature simple AI, where enemies patrol an area or follow a pre-defined path. More advanced platformers, like Hollow Knight, include intelligent AI that reacts to the player’s movements and actions. For The Quest for the Lost Artifact, enemy AI will be implemented with basic patrol movements, and later in the game, more advanced behavior like tracking the player’s position will be introduced.

2.3 Game Design Principles in Platformers

Level Design

Level design is one of the most important aspects of a platformer game. The level must be carefully constructed to challenge the player while also providing a sense of progression and reward. Levels in platformers often follow a difficulty curve, where early levels introduce the player to the game's mechanics, and later levels increase in complexity and difficulty.

In The Quest for the Lost Artifact, the levels will be designed to challenge the player's platforming skills while introducing new mechanics as the game progresses. Hidden secrets, collectibles, and alternate paths will also be integrated to encourage exploration.

Puzzle Elements

Modern platformers often incorporate puzzle elements into their gameplay. In games like Fez and Braid, puzzle-solving is central to the experience, where players must use their understanding of the environment and game mechanics to progress. In The Quest for the Lost Artifact, puzzles will be integrated into level design, requiring players to activate switches, avoid traps, and manipulate the environment to unlock new areas.

Narrative in Platformers

Although early platformers had simple stories, modern platformers often feature rich narratives. Games like Celeste and Ori and the Blind Forest weave emotional, character-driven stories into their gameplay, creating a deeper connection between the player and the game world. The narrative for The Quest for the Lost Artifact will follow Alex's journey, with the story unfolding as the player progresses through the game's levels.

2.4 The Pygame Library

Overview of Pygame

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries, making it suitable for creating 2D games. Released in 2000, Pygame has been widely adopted for game development due to its simplicity and ease of use, especially for beginners.

Pygame's Role in Game Development

Pygame simplifies many of the complex tasks involved in game development. With built-in modules for handling sprite images, sound effects, and collision detection, Pygame provides a solid framework for building 2D games. It abstracts away much of the lower-level work that developers would need to handle manually in other languages or engines.

Game Loop and Event Handling

In Pygame, the game loop is at the core of game development. The loop handles player input, updates game objects, and renders frames. For The Quest for the Lost Artifact, the game loop will manage player input (keyboard controls), update the player's position and status, handle enemy movement, and check for collisions with the environment.

The game's performance will also rely on how efficiently the loop is implemented, with the frame rate being capped at 60 frames per second for a smooth gaming experience.

2.5 Existing Platformer Games Using Pygame

Super Potato Bruh

Super Potato Bruh is a 2D platformer game built using Pygame. It features basic platformer mechanics, such as running, jumping, and avoiding enemies. The game's simplicity makes it an excellent example for beginner developers, showing how basic Pygame functions can be combined to create a functional game.

The Quest for the Lost Artifact builds on these foundational mechanics, adding more complex gameplay elements like puzzles, multiple enemies, and advanced AI behavior.

Tux of Math Command

Tux of Math Command is another Pygame-based game, though it uses platformer elements in an educational context. The player controls a character who solves math problems while avoiding obstacles and enemies. The game demonstrates how Pygame can be used for both entertainment and education, blending platforming mechanics with learning.

In The Quest for the Lost Artifact, platforming mechanics will be used primarily for entertainment, with the focus on exploration and combat rather than educational content.

Methodology

The methodology section provides a detailed explanation of the processes and techniques used in the development of "The Quest for the Lost Artifact." It covers the project development lifecycle, including planning, design, implementation, testing, and deployment phases. This chapter also discusses the tools and technologies used, the development environment setup, and the challenges faced during the project. The structured approach ensures that the game meets its objectives, is developed efficiently, and provides an engaging user experience.

3.1 Project Development Lifecycle

The development of The Quest for the Lost Artifact follows the **Waterfall Model** of software development. This model is chosen due to its structured and sequential nature, making it easier to manage the various stages of game development, from conceptualization to final deployment. Each phase is dependent on the completion of the previous one, ensuring a systematic approach to the project.

3.1.1 Requirement Analysis and Planning

In this initial phase, the project's requirements were gathered and documented. The goals of the game, target audience, gameplay mechanics, and technical requirements were defined. A detailed project plan was created, outlining the scope, timeline, resources, and milestones.

- **Objectives and Scope:** Clearly defined to include game features such as character movement, level design, enemy AI, puzzles, and user interface.
- **Resource Allocation:** Tools such as Python and Pygame for development, along with asset creation software like GIMP and Audacity for graphics and sound.

3.1.2 Game Design

The design phase involved creating a detailed blueprint for the game. This included:

- **Game Design Document (GDD):** A comprehensive document that described the storyline, characters, levels, game mechanics, and user interface.
- **Level Design:** Sketches and maps of each level were created, detailing the placement of platforms, enemies, and collectibles. The design also included planning for puzzles and hidden areas.
- **Character and Enemy Design:** Character attributes, movement capabilities, and enemy AI behavior were defined. The design ensured that gameplay would be challenging but fair, with a balanced difficulty curve.

3.1.3 Implementation

The implementation phase involved coding the game based on the design document. It was divided into several sub-phases:

- **Game Setup and Initialization:**
  - Setting up the Pygame environment and initializing game variables.
  - Creating the main game loop to handle events, updates, and rendering.
- **Character Control and Movement:**
  - Implementing player movement, jumping, and collision detection using Pygame's sprite and rect modules.
  - Adding animations for different states, such as running, jumping, and idle.
- **Level and Environment Creation:**
  - Designing levels using a grid-based system for easy placement of platforms and obstacles.
  - Loading level data from external files to facilitate level editing and expansion.
- **Enemy and NPC Behavior:**
  - Developing AI for enemies, including patrol routes, attack patterns, and reactions to the player.
  - Implementing NPC interactions, such as dialogue and quests.
- **Puzzle and Game Mechanics:**
  - Creating various puzzles, such as switches, moving platforms, and hidden keys.
  - Integrating these mechanics into the game flow to provide a mix of action and brain-teasing challenges.

3.1.4 Testing and Debugging

Testing was conducted in several stages to ensure the game was free of bugs and provided a smooth user experience:

- **Unit Testing:**
  - Individual components, such as player controls, collision detection, and enemy behavior, were tested in isolation to ensure they functioned correctly.
- **Integration Testing:**
  - Components were combined and tested together to identify any issues that arose from their interaction.
- **User Acceptance Testing (UAT):**
  - The game was tested by potential users to gather feedback on gameplay, difficulty, and overall enjoyment. Based on this feedback, adjustments were made to improve the game experience.
- **Debugging:**
  - Identified bugs were logged, analyzed, and fixed systematically. Tools like the Pygame debugger and print statements were used to track down issues.

3.1.5 Deployment

The final phase involved preparing the game for release. This included:

- **Packaging:**
  - Compiling the game into an executable format using tools like PyInstaller, allowing it to be run on systems without Python or Pygame installed.
- **Documentation:**
  - Creating a user manual that explained game controls, features, and tips for playing.
  - Providing technical documentation for future maintenance or updates.
- **Distribution:**
  - Uploading the game to online platforms such as GitHub or itch.io, along with instructions for downloading and running the game.

3.2 Tools and Technologies Used

The development of The Quest for the Lost Artifact utilized a range of tools and technologies:

3.2.1 Python and Pygame

- **Python:** The primary programming language used for its simplicity, readability, and extensive libraries.
- **Pygame:** A set of Python modules designed for writing video games, providing functionalities for handling graphics, sound, and user input.

3.2.2 Integrated Development Environment (IDE)

- **PyCharm:** Used for writing, testing, and debugging code. Features like code completion, error highlighting, and integrated version control helped streamline development.

3.2.3 Graphic and Sound Design Tools

- **GIMP:** Used for creating and editing game sprites, backgrounds, and user interface elements.

3.2.4 Version Control

- **Git:** Used to track changes in the codebase, manage different development branches, and collaborate with team members.
- **GitHub:** Hosted the project's repository, making it easy to share and review code changes.

3.2.5 Testing Tools

- **Pygame Debugger:** A built-in tool that helped monitor game variables and performance during testing.
- **PyTest:** Used for automating unit tests to ensure that code changes did not introduce new bugs.

3.3 Development Environment Setup

Setting up the development environment involved installing and configuring various software tools:

1. **Python Installation:**
    a. Python 3.7 or higher was installed from the official Python website. This version was chosen for its compatibility with Pygame and stability.
2. **Pygame Installation:**
    a. Pygame was installed using the pip package manager with the command:

Copy code

```
Pip  install      pygame
```

3. **IDE Configuration:**
    a. PyCharm was configured to recognize the Python interpreter and installed libraries. Project files were organized into directories for assets, scripts, and documentation.
4. **Version Control Setup:**
    a. A Git repository was initialized in the project directory. Commit messages were used to document changes, and branches were created for major features or bug fixes.
5. **Graphics and Sound Asset Preparation:**
    a. Graphics assets were created in GIMP and saved in PNG format for transparency. Sound effects were recorded and edited in Audacity before being saved in WAV format for high quality.

3.4 Game Design and Implementation Details

The implementation of The Quest for the Lost Artifact involved several key design choices and coding strategies:

3.4.1 Player Movement and Physics

- **Movement Mechanics:**
    - The player's movement was controlled using keyboard inputs. Pygame's key.get_pressed() method was used to detect key presses for moving left, right, and jumping.

- **Gravity and Jumping:**
    - A gravity variable was applied to the player's vertical position each frame, simulating realistic falling. Jumping was implemented by applying an upward force to the player's velocity, which gradually decreased due to gravity.

3.4.2 Level Design and Loading

- **Grid-Based Level Design:**
    - Levels were designed on a grid system, with each grid cell representing a different type of terrain or obstacle. This made it easier to create and modify levels programmatically.
- **External Level Files:**
    - Levels were stored in external text files, with each character representing a different object (e.g., 'P' for platforms, 'E' for enemies). This allowed for quick iteration and testing of different level designs.

3.4.3 Enemy AI and Interactions

- **Basic AI Behavior:**
    - Enemies were programmed to follow simple AI patterns, such as patrolling back and forth or moving towards the player when in range.
- **Interactions with Player:**
    - Collisions between the player and enemies resulted in the player losing health or lives. This interaction was handled using Pygame's spritecollide() function.

3.4.4 User Interface and Menus

- **Main Menu:**
    - A main menu was created with options for starting the game, loading a saved game, adjusting settings, and exiting. This was implemented using Pygame's text rendering and event handling capabilities.
- **In-Game HUD:**
    - The heads-up display (HUD) showed the player's health, score, and remaining lives. This information was updated in real-time during gameplay.

3.5 Challenges Faced and Solutions

Developing The Quest for the Lost Artifact presented several challenges:

3.5.1 Collision Detection

- **Challenge:** Implementing accurate collision detection for complex shapes and environments.

- **Solution:** Used Pygame's built-in collision functions, combined with custom bounding box checks for irregular shapes.

3.5.2 Enemy AI

- **Challenge:** Creating intelligent and challenging enemy behaviors without significantly impacting performance.
- **Solution:** Developed simple yet effective AI patterns, limiting complex calculations to avoid performance issues.

3.5.3 Game Performance Optimization

- **Challenge:** Ensuring the game ran smoothly on different hardware configurations.
- **Solution:** Implemented frame rate capping and optimized code by reducing unnecessary calculations and using efficient data structures.

3.5.4 Balancing Difficulty

- **Challenge:** Making the game challenging without being frustrating.
- **Solution:** Conducted extensive playtesting and adjusted level design, enemy placement, and player abilities to achieve a balanced difficulty curve.

Tools and Technology

In developing "The Quest for the Lost Artifact," selecting the right tools and technologies is crucial to ensure an efficient and successful game development process. This chapter provides an in-depth exploration of the tools, frameworks, and technologies utilized in the project, explaining their significance, features, and how they contribute to the development of the game.

4.1. Programming Language: Python

Python is an interpreted, high-level, and versatile programming language known for its readability and simplicity. It has become a popular choice for game development, particularly for beginners and educators, due to its clear syntax and extensive library support. Python enables rapid prototyping and experimentation, making it ideal for developing game mechanics and testing new features without the need for complex setup or compilation processes.

4.2. Game Development Framework: Pygame

Pygame is a set of Python modules designed specifically for writing video games. It simplifies the process of game development by providing pre-written functions and classes for graphics rendering, sound playback, and input handling. Pygame is particularly popular for developing 2D games and is used extensively in educational settings to introduce programming concepts through interactive game development.

- **Features of Pygame:**

  Graphics and Sprites: Pygame supports image loading and manipulation, allowing developers to create and manage game sprites and animations. It provides functions for rendering shapes, text, and images onto the game screen.
  - **Sound and Music:** Pygame includes modules for playing sound effects and background music, enhancing the auditory experience of the game. It supports various audio formats, including .wav and .mp3 files.
  - **Event Handling:** Pygame's event module allows developers to handle user inputs such as keyboard and mouse actions. This is essential for controlling character movements and responding to player interactions.
  - **Collision Detection:** Pygame simplifies collision detection between game objects, enabling developers to implement mechanics such as player-enemy interactions, item pickups, and obstacle avoidance.
  - **Game Loop Management:** Pygame's clock module helps manage the game loop, ensuring a consistent frame rate and smooth gameplay experience.

In "The Quest for the Lost Artifact," Pygame is utilized to manage all aspects of game rendering, input handling, and audio playback. It forms the core framework for building the game's interactive environment and managing game states.

4.3. Integrated Development Environment (IDE): PyCharm

PyCharm is a powerful and widely-used integrated development environment (IDE) specifically designed for Python development. It offers a range of features that streamline the coding process, making it easier to manage large projects and collaborate with others.

- **Key Features of PyCharm:**
    - **Code Editor and Navigation:** PyCharm provides advanced code completion, syntax highlighting, and error checking, helping developers write clean and efficient code. Its project navigation features enable quick access to files, classes, and functions within the project.
    - **Integrated Debugging Tools:** PyCharm's debugger allows developers to set breakpoints, inspect variables, and step through code execution, making it easier to identify and fix bugs in the game's code.
    - **Version Control Integration:** PyCharm integrates seamlessly with version control systems like Git, enabling developers to track changes, manage branches, and collaborate on code.
    - **Code Refactoring and Analysis:** PyCharm offers tools for refactoring code, such as renaming variables, extracting methods, and restructuring code blocks. Its code analysis features help identify potential issues and improve code quality
    - **Plugins and Customization:** PyCharm supports a wide range of plugins, allowing developers to customize their development environment with additional tools, themes, and language support.

For "The Quest for the Lost Artifact," PyCharm is used to manage the project's codebase, handle debugging, and streamline the development workflow. Its powerful tools and features support efficient coding and troubleshooting, ensuring a smooth development process.

4.4. Graphics and Asset Creation Tools

Graphics and visual assets play a crucial role in defining the aesthetic and visual appeal of the game. Creating high-quality sprites, backgrounds, and UI elements requires specialized tools and skills. For this project, the following tools were used:

1. **GIMP:**

   GIMP (GNU Image Manipulation Program): GIMP is an open-source alternative to Photoshop that provides similar functionality for image editing and graphic design. It is used for creating and editing sprites, backgrounds, and UI elements in the game.

2. **Aseprite:** Aseprite is a specialized tool for creating pixel art and animations. It offers a range of features tailored to pixel art, such as frame-by-frame animation, onion skinning, and palette management. Aseprite is ideal for developing the retro-style sprites and animations featured in "The Quest for the Lost Artifact."
3. **Tiled Map Editor:** Tiled is an open-source map editor used for designing 2D game levels. It allows developers to create tile-based maps with multiple layers, making it easy to design and manage complex level layouts. Tiled supports exporting maps in various formats, which can then be integrated into the Pygame project.

These tools were essential in creating the visual identity of "The Quest for the Lost Artifact," enabling the development of unique characters, environments, and animations that bring the game world to life.

Timeline (Gantt Chart)

ID	Task Name	Start Date	End Date
1	Initial Setup	July 25	July 28
2	Game World and Map Design (Part 1)	July 29	August 28
3	Game World and Map Design (Part 2)	August 18	September 06
4	Character Animations	August 04	August 25
5	Enemy AI & Pathfinding (Part 1)	August 18	August 25
6	Enemy AI & Pathfinding (Part 2)	August 30	September 10
7	Combat System and Items (Part 1)	August 25	September 06
8	Combat System and Items (Part 2)	September 01	September 15
9	User Interface (UI) Design	September 07	September 22
10	Minimap	September 16	September 25
11	SFX	September 26	September 29
12	Finalization	October 01	October 05

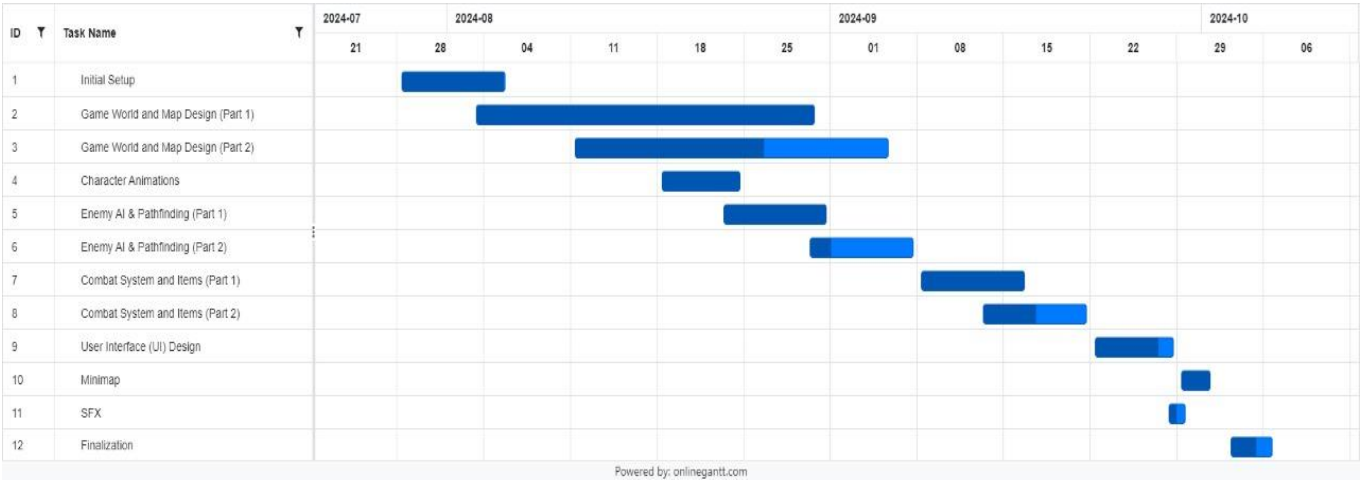


Figure 1

- This is a Gantt chart illustrating the timeline and progress of a project from July to October 2024. The project consists of twelve tasks, each represented by a horizontal bar indicating its duration and overlap with other tasks. The chart provides a visual representation of task dependencies and scheduling.
- **Initial Setup**: Begins on July 21 and ends by August 1.
- **Game World and Map Design (Part 1)**: Starts on July 28 and completes on August 8.
- **Game World and Map Design (Part 2)**: Continues from August 4 to August 29.
- **Character Animations**: Runs from August 11 to August 22.
- **Enemy AI & Pathfinding (Part 1)**: Takes place between August 18 and August 29.
- **Enemy AI & Pathfinding (Part 2)**: From August 25 to September 12.
- **Combat System and Items (Part 1)**: Scheduled from September 1 to September 12.

- **Combat System and Items (Part 2)**: Proceeds from September 8 to September 26.
- **User Interface (UI) Design**: Occurs between September 15 and September 26.
- **Minimap**: Runs from September 22 to October 3.
- **SFX**: Scheduled from September 29 to October 10.
- **Finalization**: From October 3 to October 6.

System Design

Class Diagrams

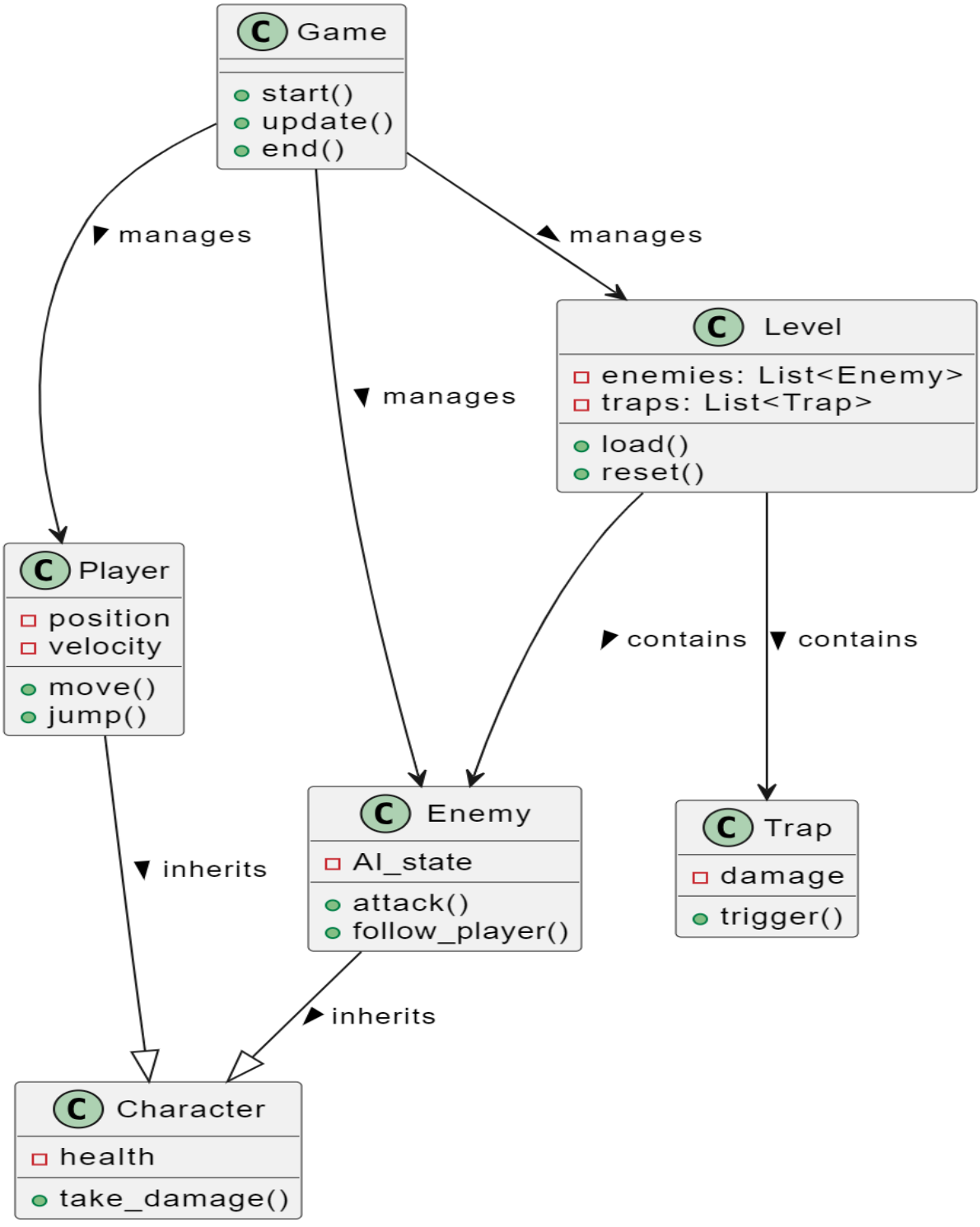


Figure 2

- This diagram represents the class structure and relationships in a game design. The Game class serves as the central controller, managing the game's flow with methods like start(), update(), and end(). It oversees the Level class, which contains lists of Enemy and Trap objects, and has methods for loading and resetting the level.

- The Player class, representing the main character, holds attributes for position and velocity, and provides methods for movement (move()) and jumping (jump()). It also inherits from the Character class, which includes common attributes like health and methods such as take_damage().

- The Enemy class, inheriting from Character, features additional attributes like AI_state and methods for attacking and following the player. The Trap class, also managed by the Level, includes properties like damage and a method to trigger () the trap's effect.

- Overall, the diagram illustrates a hierarchical structure where the Game orchestrates various components, including levels, characters, and traps, facilitating interactions and gameplay mechanics. This design allows for modularity and reusability within the game's codebase.

Sequence Diagram

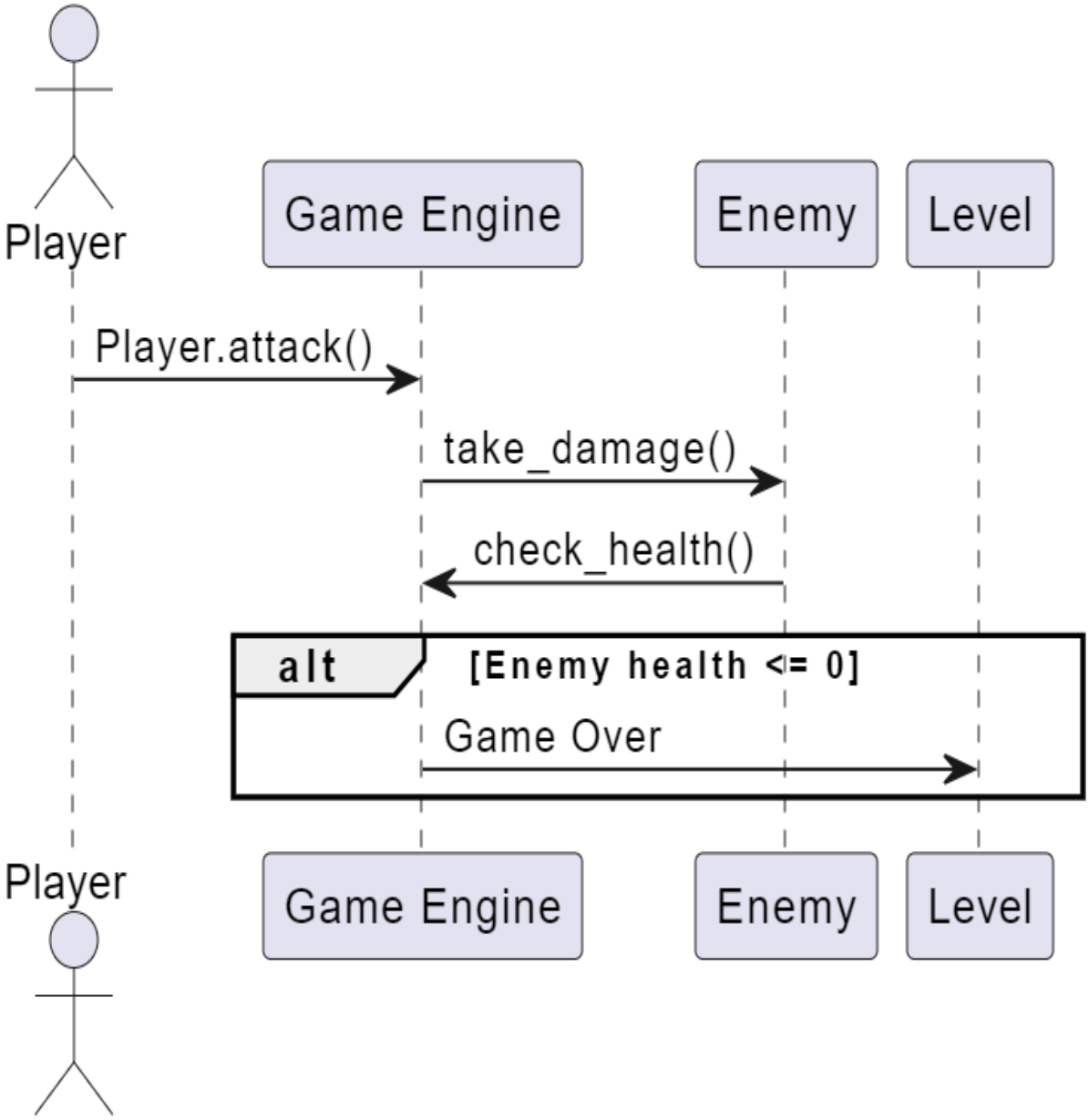


Figure 3

- **Player Initiates Attack:** It all starts with the Player calling the Player.attack () method, indicating the intention to attack an Enemy.
- **Message to Game Engine:** This action sends a command to the Game Engine to execute the take_damage () method on the Enemy.
- **Enemy Takes Damage:** The Game Engine calls the take_damage () method on the Enemy, causing the Enemy to lose health points.
- **Health Check:** After taking damage, the Enemy assesses its current health status by calling its check health () method.
- **Return Health Status:** The check health () method returns the updated health status of the Enemy to the Game Engine.
- **Condition Check:** An alternative condition evaluates the health status. If the Enemy's health is less than or equal to zero ([Enemy health <= 0]), the Game Engine triggers the "Game Over" event, signifying the Enemy is defeated and the game ends

State Machine Diagram

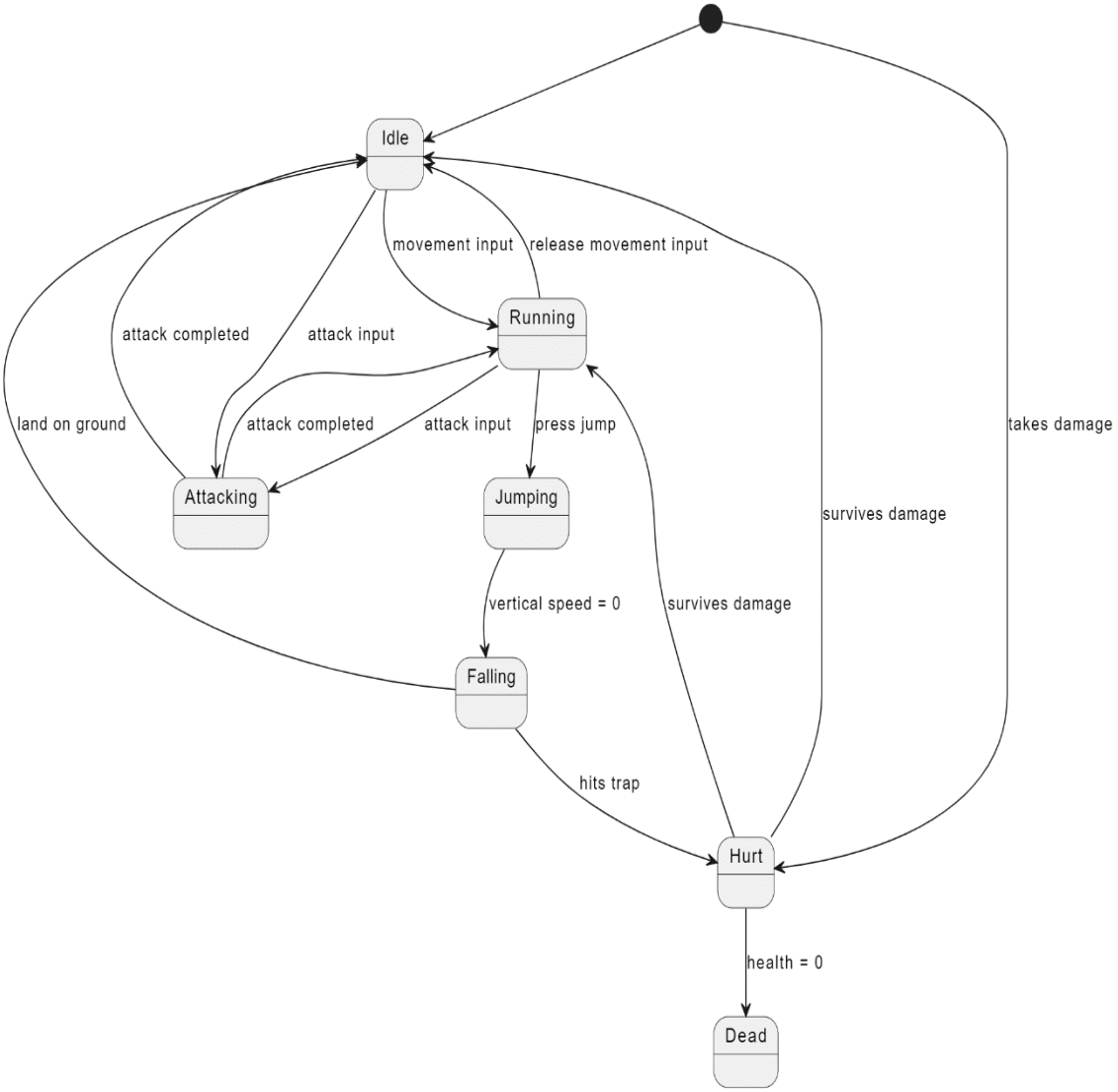


Figure 4

- **Idle:** The character starts in the idle state when there are no inputs.
- **Running:** When movement input is detected, the character transitions from Idle to Running.
- **Attacking:** The character moves to Attacking when attack input is received from either Idle, Running, or jumping states.
- **Jumping:** Pressing jump while in Idle or Running states causes the character to transition to Jumping.
- **Falling:** The character transitions to Falling from Jumping when the vertical speed equals 0.
- **Hurt:** If the character takes damage from any state (Idle, Running, Jumping, Falling, Attacking), it transitions to the Hurt state.
- **Dead:** From Hurt, if the character's health equals 0, the character transitions to Dead.
- **Surviving Damage:** If the character survives damage in the Hurt state, it transitions back to Idle.
- **Hits Trap:** If the character hits a trap in any state, it will transition to Hurt.

System Diagram

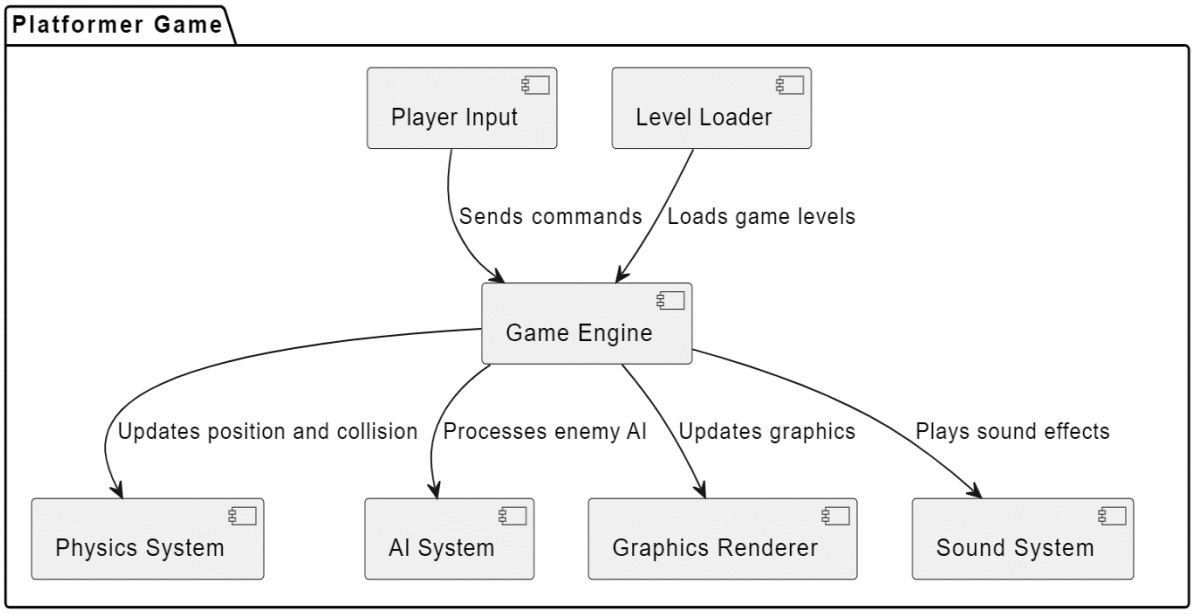


Figure 5

- **Player Input:** The player starts by sending commands, such as movements or actions, to the Game Engine.
- **Game Engine:** This is the central component that processes all inputs and interactions within the game.
- **Physics System:** The Game Engine updates positions and handles collisions through this system, ensuring objects move realistically.
- **AI System:** The Game Engine also processes enemy AI here, determining how enemies react to the player's actions.
- **Graphics Renderer:** To maintain visual integrity, the Game Engine updates graphics here, rendering the game environment and characters.
- **Sound System:** Simultaneously, the Game Engine ensures sound effects are played appropriately, enhancing the gaming experience.
- **Level Loader:** This component loads game levels into the Game Engine, providing the necessary environment and challenges for the player.

Code and Implementation

Main code: game.py

```
import pygame
import sys
from tilemap import TileMap
from player import Player
from enemy import Enemy
from item import Item
from levels import *
from inventory import Inventory
from menu import Menu, SettingsMenu  # Importing Menu classes

# Constants
SCREEN_WIDTH = 1200
SCREEN_HEIGHT = 720
PLAYER_START_POS = (0, 0)
GRAVITY = 0.5
PLAYER_SPEED = 5
JUMP_SPEED = -14
TELEPORT_COOLDOWN = 2000  # 2 seconds in milliseconds
CAMERA_SPEED = 0.1  # Speed of camera interpolation
RESPAWN_DELAY = 2000  # 2 seconds in milliseconds

class Game:
    def __init__(self):
        # Initialize Pygame
        pygame.init()
```

```
        pygame.mixer.init()
        self.screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
        pygame.display.set_caption("THE QUEST FOR THE LOST ARTIFACT")

        # Initialize Player
        self.Player = Player()
        self.Player.rect.topleft = PLAYER_START_POS

        # Initialize TileMap
        self.tile_map = TileMap()
        self.enemies = self.tile_map.enemies  # Get enemies from the tilemap

        # Load initial level
        self.current_level = '51aa7c80-4ce0-11ef-b800-db70e6809841'
        self.load_level(self.current_level)

        # Initialize physics and camera
        self.player_velocity_y = 0
        self.on_ground = False
        self.last_teleport_time = 0
        self.camera_x, self.camera_y = 0, 0
        self.message_printed = False

        # Initialize inventory
        slot_types = ['normal', 'head', 'chest', 'gloves', 'shoes', 'ring', 'sword']
        self.slot_images = {
            slot_type:
pygame.image.load(f'assets/inventory/{slot_type}_slot.png').convert_alpha()
            for slot_type in slot_types
```

```
        }
        self.inventory = Inventory(self.screen, self.slot_images)

        # Inventory toggle
        self.inventory_open = False

        # Track key states
        self.attack_key_pressed = False

        # Respawn tracking
        self.respawning = False
        self.respawn_timer = 0

        # Set up the clock for managing the frame rate
        self.clock = pygame.time.Clock()

        # Initialize Menu System
        self.state = 'gameplay'  # Possible states: 'gameplay', 'menu', 'settings'
        self.menu = Menu(self.screen, self)
        self.settings_menu = SettingsMenu(self.screen, self)

    def load_level(self, level_iid):
        self.current_level = level_iid
        self.tile_map.load_level(level_files[self.current_level], object_files[self.current_level])
        # Load and scale the background image for the level
        background_image_path = background_Object_images[self.current_level][0]
        level_background_surface = pygame.image.load(background_image_path).convert_alpha()
        self.level_background = pygame.transform.scale(level_background_surface, (SCREEN_WIDTH, SCREEN_HEIGHT))
```

```
        self.tile_rects = self.tile_map.get_tile_rects()
        self.message_printed = False

        start_point = None
        for entity_type, entities in self.tile_map.objects.items():
            if entity_type == 'Start_point' and entities:
                start_point = entities[0]  # Assuming there's only one start point

        if start_point:
            self.Player.rect.topleft = (start_point['x'], start_point['y'])
        else:
            self.Player.rect.topleft = PLAYER_START_POS

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.cleanup_and_exit()
            if self.state == 'gameplay':
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_i:
                        self.inventory_open = not self.inventory_open
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_ESCAPE:
                        self.state = 'menu'  # Open menu
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_a:  # Attack key press
                        if not self.attack_key_pressed:
                            self.attack_key_pressed = True
                            self.Player.current_action = self.Player.advance_attack_sequence()
```

```
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_a:  # Attack key release
                    self.attack_key_pressed = False
        elif self.state == 'menu':
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    self.state = 'gameplay'  # Close menu
        elif self.state == 'settings':
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    self.state = 'menu'  # Close settings

def cleanup_and_exit(self):
    pygame.quit()
    sys.exit()

def handle_input(self):
    if self.respawning:  # Block input during respawn
        return self.Player.current_action

    keys = pygame.key.get_pressed()
    action = self.Player.current_action  # Start with the current action

    if keys[pygame.K_SPACE] and self.on_ground:
        self.player_velocity_y = JUMP_SPEED
        self.on_ground = False
        action = 'jump'
    elif keys[pygame.K_LEFT]:
        self.Player.rect.x -= PLAYER_SPEED
```

```
        if self.on_ground:
            action = 'run'
        self.Player.set_flip(True)
    elif keys[pygame.K_RIGHT]:
        self.Player.rect.x += PLAYER_SPEED
        if self.on_ground:
            action = 'run'
        self.Player.set_flip(False)

    # If attack key is pressed, prioritize the attack action
    if keys[pygame.K_a] and self.attack_key_pressed and action != 'jump':
        action = self.Player.current_action

    return action

def apply_gravity(self):
    if not self.respawning:  # Block gravity during respawn
        self.player_velocity_y += GRAVITY
        self.Player.rect.y += self.player_velocity_y

def check_collisions(self, previous_position):
    self.on_ground = False
    for tile_rect in self.tile_rects:
        if self.Player.rect.colliderect(tile_rect):
            if previous_position[1] + self.Player.rect.height <= tile_rect.top:
                self.Player.rect.bottom = tile_rect.top
                self.player_velocity_y = 0
                self.on_ground = True
                return 'idle' if not any(k for k in pygame.key.get_pressed()) else None
```

```
            elif previous_position[1] >= tile_rect.bottom:
                self.Player.rect.top = tile_rect.bottom
                self.player_velocity_y = 0
            elif previous_position[0] + self.Player.rect.width <= tile_rect.left:
                self.Player.rect.right = tile_rect.left
            elif previous_position[0] >= tile_rect.right:
                self.Player.rect.left = tile_rect.right
        return None


    def handle_teleporters(self):
        current_time = pygame.time.get_ticks()
        keys = pygame.key.get_pressed()
        if keys[pygame.K_RETURN]:
            for level_teleporter in self.tile_map.level_teleporters:
                if self.Player.rect.colliderect(level_teleporter['rect']) and current_time - self.last_teleport_time > TELEPORT_COOLDOWN:
                    destination = level_teleporter['customFields'].get('destination')
                    if destination:
                        destination_iid = destination.get('entityIid')
                        destination_level_iid = destination.get('levelIid')
                        if destination_level_iid and destination_iid:
                            self.load_level(destination_level_iid)
                            destination_level_teleporter = self.find_level_teleporter_by_iid(destination_iid)
                            if destination_level_teleporter:
                                self.Player.rect.topleft = (destination_level_teleporter['x'], destination_level_teleporter['y'])
                                self.last_teleport_time = current_time
                    else:
                        if not self.message_printed:
```

```
                    print("Teleporter has no valid destination specified.")
                    self.message_printed = True
            else:
                if not self.message_printed:
                    print("Teleporter destination is null.")
                    self.message_printed = True

def find_level_teleporter_by_iid(self, iid):
    for level_teleporter in self.tile_map.level_teleporters:
        if level_teleporter['iid'] == iid:
            return level_teleporter
    return None

def handle_interaction(self):
    keys = pygame.key.get_pressed()
    if keys[pygame.K_e]:  # Using the 'E' key to interact with chests
        player_rect = self.Player.rect
        for chest in self.tile_map.chests:
            if chest.rect.colliderect(player_rect) and not chest.opened:
                chest.toggle_opened()  # Open the chest only if it is not already opened
                # Add chest items to the inventory in normal slots
                for item in chest.swords + chest.coins:
                    free_slot_index = self.find_free_normal_inventory_slot()
                    if free_slot_index is not None:
                        self.inventory.add_item(item, free_slot_index)

def find_free_normal_inventory_slot(self):
    for i, slot in enumerate(self.inventory.slots):
        if slot.slot_type == 'normal' and slot.item is None:
```

```
            return i
        return None

    def update_camera(self):
        # Calculate the maximum boundaries of the level based on tile map dimensions and screen size
        max_x = max(self.tile_map.layers[0]['image'].get_width() - SCREEN_WIDTH, 0)
        max_y = max(self.tile_map.layers[0]['image'].get_height() - SCREEN_HEIGHT, 0)

        # Calculate target camera position
        target_camera_x = max(0, min(self.Player.rect.centerx - SCREEN_WIDTH // 2, max_x))
        target_camera_y = max(0, min(self.Player.rect.centery - SCREEN_HEIGHT // 2, max_y))

        # Interpolate camera movement for smoothness
        self.camera_x += (target_camera_x - self.camera_x) * CAMERA_SPEED
        self.camera_y += (target_camera_y - self.camera_y) * CAMERA_SPEED

    def is_out_of_bounds(self):
        max_x = self.tile_map.layers[0]['image'].get_width()
        max_y = self.tile_map.layers[0]['image'].get_height()
        return (
            self.Player.rect.x < 0 or
            self.Player.rect.right > max_x or
            self.Player.rect.y < 0 or
            self.Player.rect.bottom > max_y
        )

    def start_respawn(self):
        self.respawn_timer = pygame.time.get_ticks() + RESPAWN_DELAY
```

```
    self.respawning = True

def respawn_at_spawn_point(self):
    # Find the first spawn point and respawn the player there
    if self.tile_map.spawn_points:
        spawn_point = self.tile_map.spawn_points[0]  # Assuming only one spawn point exists
        # Respawn at the center of the spawn point image
        self.Player.rect.center = (spawn_point['x'] + spawn_point['width'] // 2,
                                   spawn_point['y'] + spawn_point['height'] // 2)
    else:
        # Default respawn at start position
        self.Player.rect.topleft = PLAYER_START_POS

def save_game(self):
    # Implement your save logic here
    print("Game saved!")

def load_game(self):
    # Implement your load logic here
    print("Game loaded!")

def run(self):
    while True:
        self.handle_events()

        if self.state == 'gameplay':
            self.update_game_logic()
        elif self.state == 'menu':
            self.handle_menu()
```

```
        elif self.state == 'settings':
            self.handle_settings()

        self.render()

        self.clock.tick(60)

def update_game_logic(self):
    if self.respawning:
        # Check if the respawn timer has elapsed
        if pygame.time.get_ticks() >= self.respawn_timer:
            self.respawn_at_spawn_point()
            self.respawning = False
            self.player_velocity_y = 0  # Reset vertical velocity after respawn
        return  # Skip the rest of the game logic while respawning

    previous_position = self.Player.rect.topleft
    action = self.handle_input()
    self.handle_teleporters()
    self.handle_interaction()  # Handle chest interaction

    self.apply_gravity()
    collision_action = self.check_collisions(previous_position)
    action = collision_action if collision_action else action

    if not self.on_ground and self.player_velocity_y > 0:
        action = 'fall'
    elif not self.on_ground and self.player_velocity_y < 0:
        action = 'jump'
```

```
        self.Player.update(action, self.clock.get_time() / 1000.0)
        self.update_camera()

        # Check if out of bounds and initiate respawn if necessary
        if self.is_out_of_bounds():
            self.start_respawn()

    def handle_menu(self):
        # Handle menu interactions
        action = self.menu.handle_mouse()

        if action == "resume":
            self.state = 'gameplay'  # Resume the game
        elif action == "settings":
            self.state = 'settings'  # Enter settings menu
        elif action == "quit":
            self.cleanup_and_exit()

        self.menu.draw()

    def handle_settings(self):
        # Handle settings interactions
        action = self.settings_menu.handle_mouse()

        if action == "back":
            self.state = 'menu'  # Go back to main menu

        self.settings_menu.draw()
```

```
    def render_gameplay(self):
        # Render the gameplay elements
        self.screen.blit(self.level_background, (0, 0))
        self.tile_map.draw_layer(self.screen, self.camera_x, self.camera_y)
        self.tile_map.draw_spawn_point(self.screen, self.camera_x, self.camera_y)
        self.tile_map.draw_enemies(self.screen, self.camera_x, self.camera_y)
        self.tile_map.draw_chests(self.screen, self.camera_x, self.camera_y)  # Draw chests
        self.Player.draw(self.screen, self.Player.rect.x - self.camera_x, self.Player.rect.y - self.camera_y)
        self.tile_map.draw_teleporters(self.screen, self.camera_x, self.camera_y)

        # Draw inventory if open
        if self.inventory_open:
            self.inventory.draw()

    def render_menu(self):
        # Render the menu based on current state
        if self.state == 'menu':
            self.menu.draw()
        elif self.state == 'settings':
            self.settings_menu.draw()

    def render(self):
        if self.state == 'gameplay':
            self.render_gameplay()
        else:
            self.render_menu()
```

```
        pygame.display.flip()


if __name__ == "__main__":
    Game()
```

Expected Outcome

Start Menu of the Game:

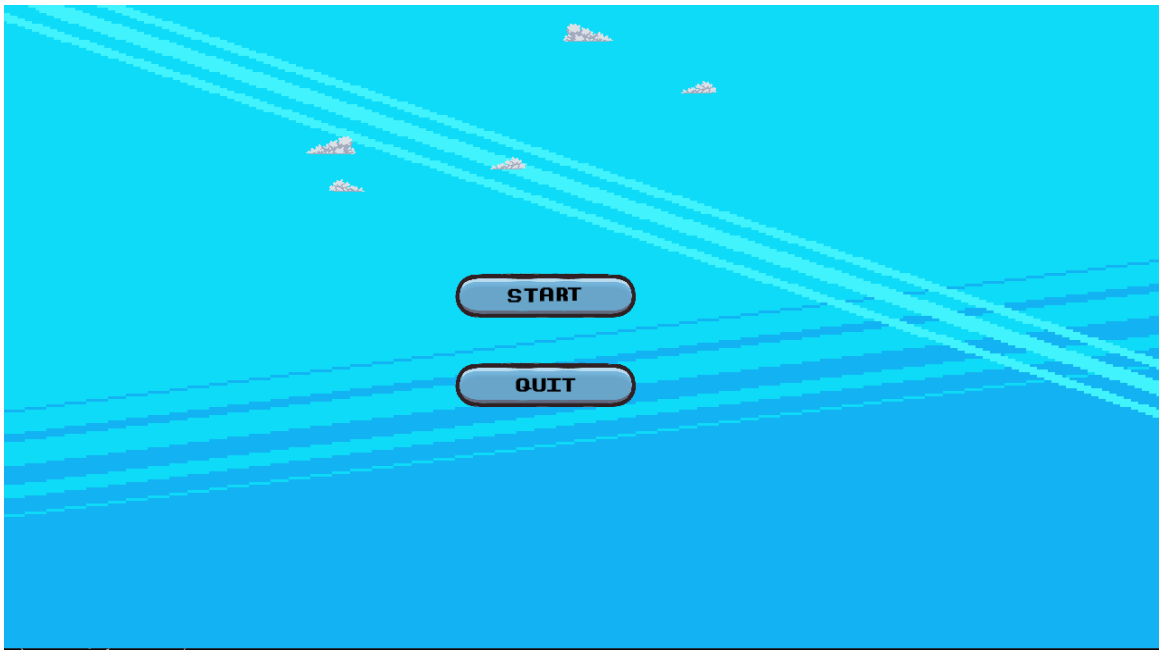


Figure 6 - Start Menu

1) Engaging Top-Down 2D Adventure: Players will experience an engaging, interactive journey with smooth player movement, where they can explore different levels, battle enemies, and complete quests in a fantasy setting.

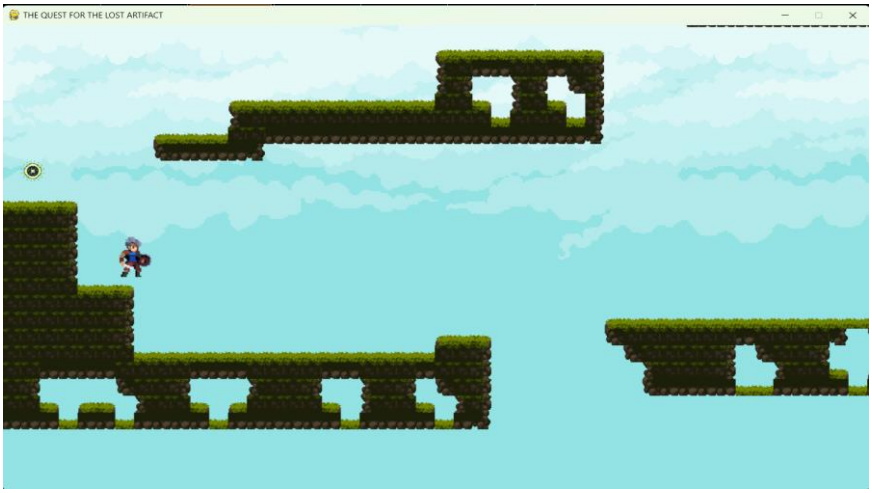


Figure 7

2) Main Menu of the game:

- **Main Menu with Full Functionality:** A detailed main menu will provide options for starting the game, adjusting settings, and quitting. It will also include a settings submenu for audio preferences and a save/load system to retain player progress across sessions.

- **Save and Load Game Feature:** Players will be able to save their progress, including audio preferences, gold, experience, enemy states, and level information. The save/load system will allow for game continuity, letting players resume their adventure where they left off.

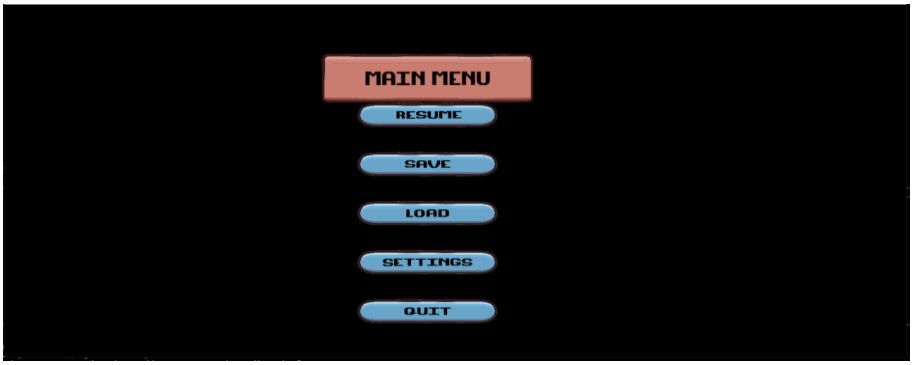


Figure 8

- **Sound and Audio Integration**; The game will have immersive sound effects, from button clicks in the menu to magical attacks and enemy interactions. The audio settings will allow players to adjust the volume for a personalized experience.

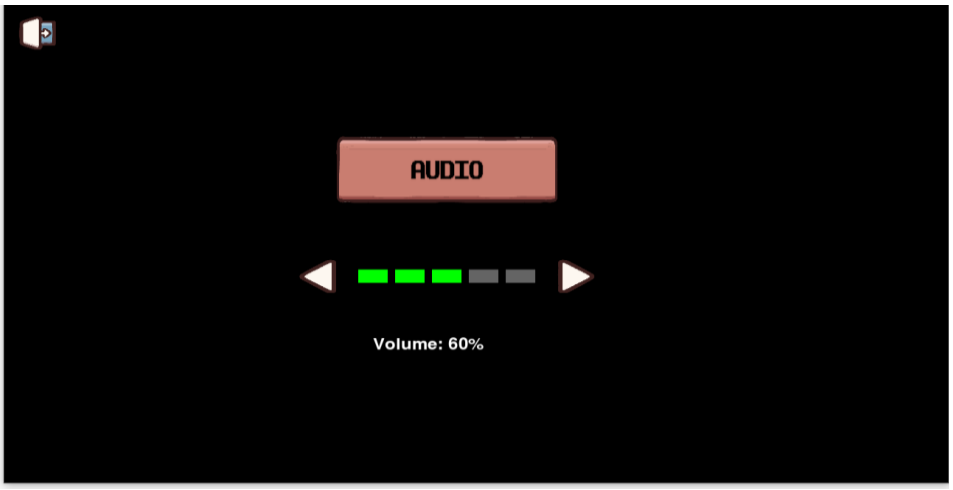


Figure 9

3) Diverse Terrains

Figure 10

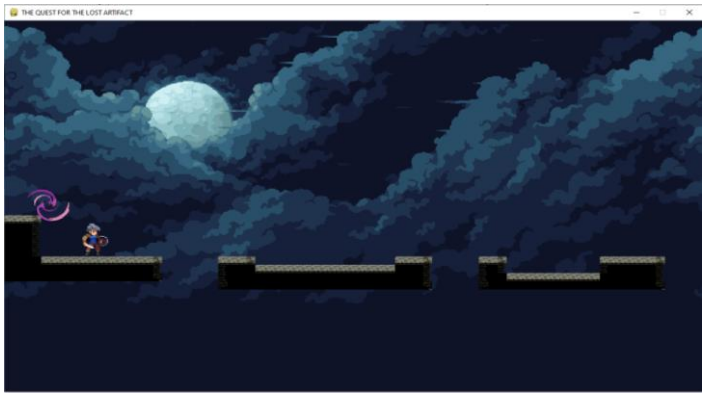


Figure 11

4) Interacting with Chest

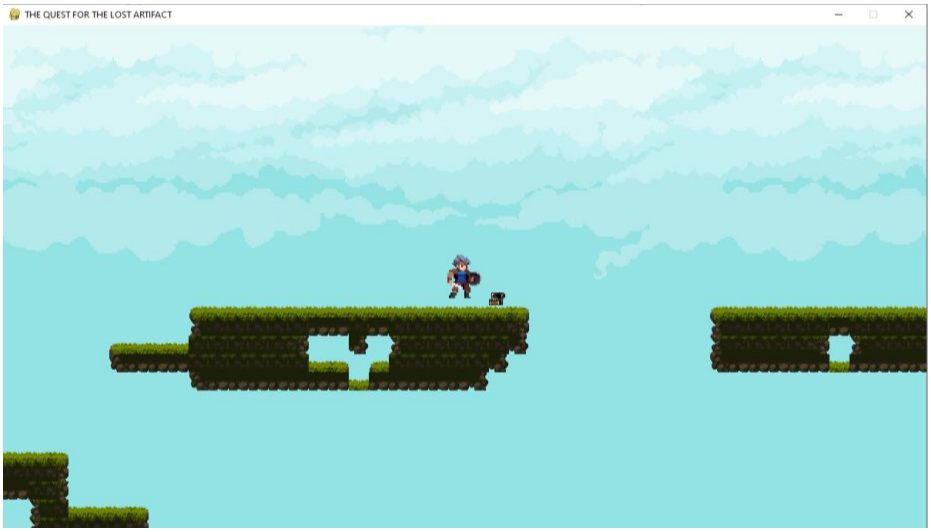


Figure 12

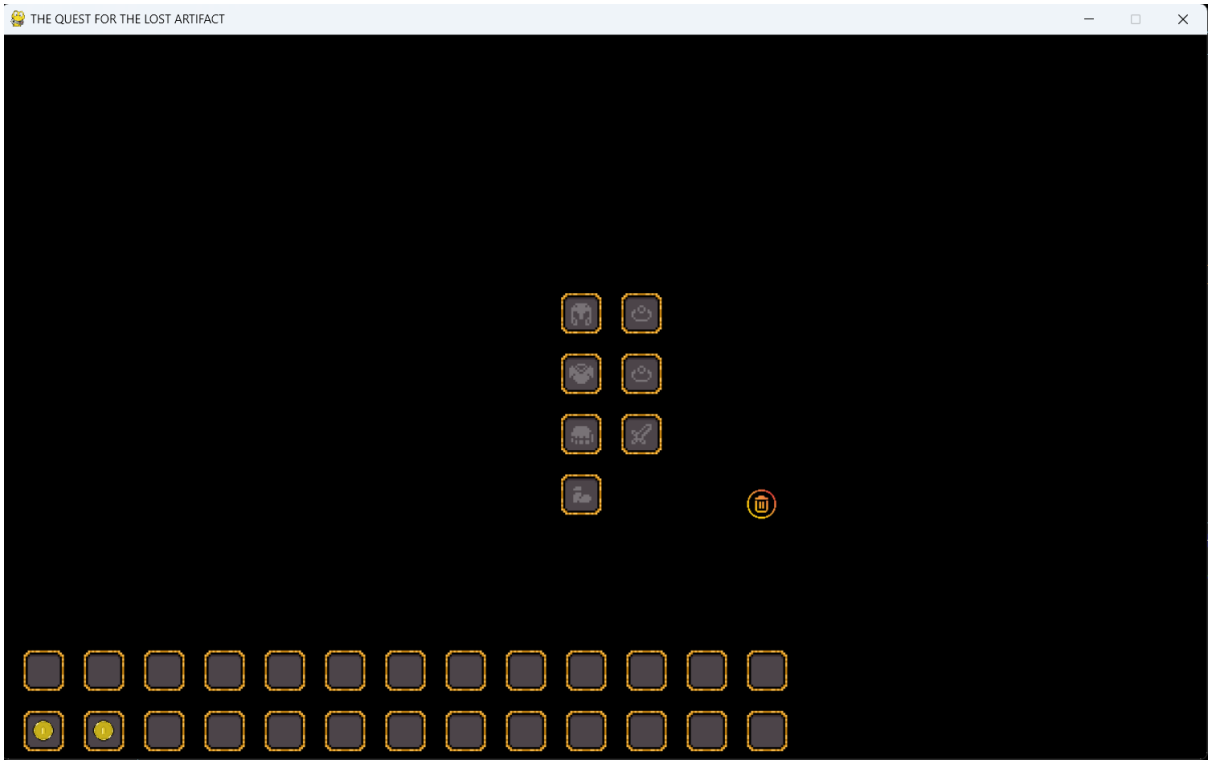


Figure 13

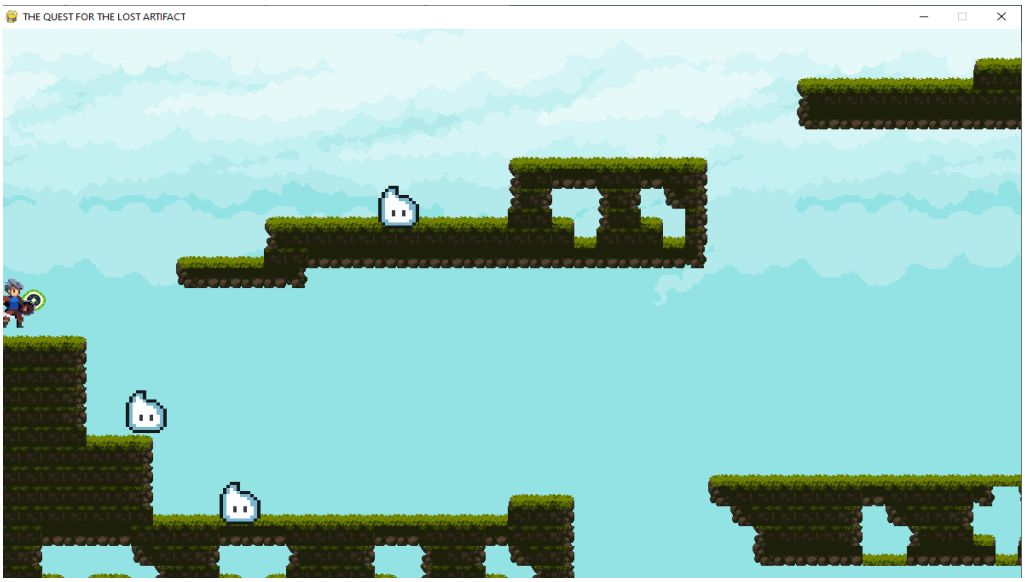


Figure 14

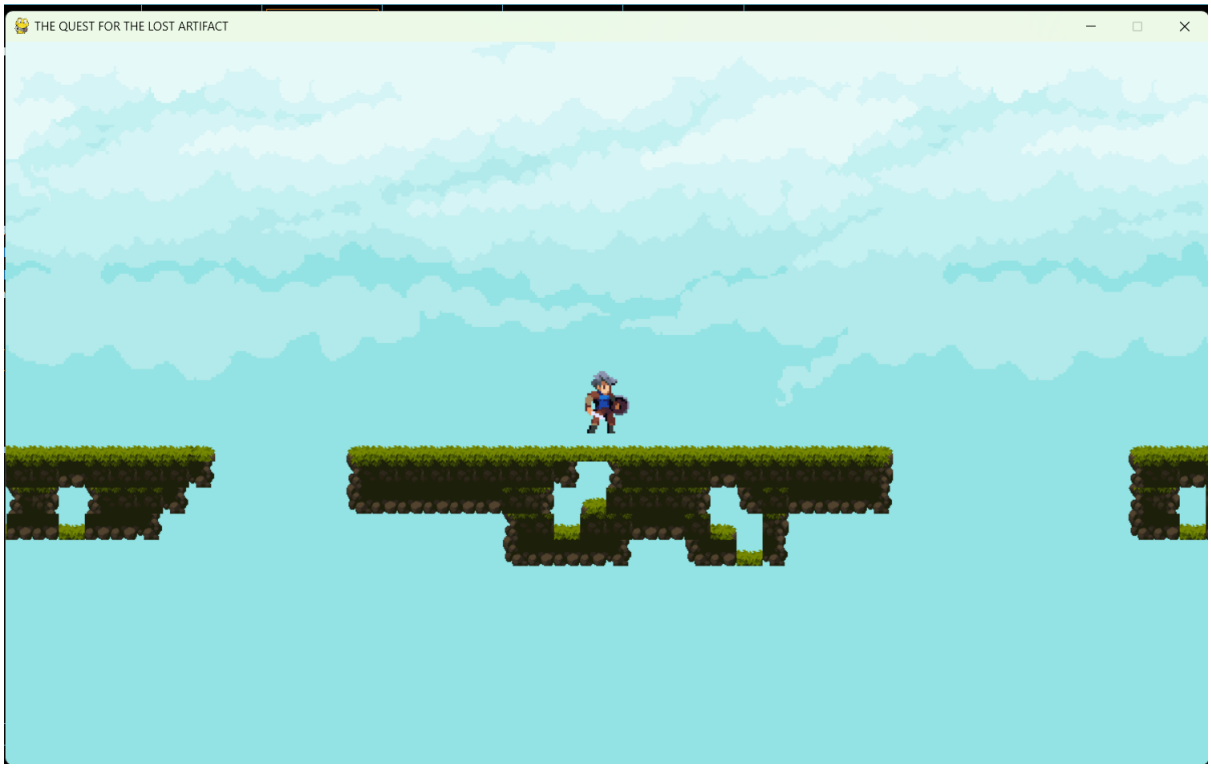


Figure 15

Future Scope

1. **Multiplayer Mode**: Adding a multiplayer option (local or online) would enhance engagement by allowing players to compete or cooperate. This could involve new game modes like co-op campaigns or PvP battles, increasing replayability.
2. **Procedural Generation**: Implementing procedurally generated levels could create endless gameplay possibilities. Each time the player starts a new game or enters a new level, they would experience a different layout, adding variety and making the game more unpredictable.
3. **Advanced AI**: You could improve enemy AI by adding more complex behaviors such as learning from player actions (reinforcement learning) or cooperating with other AI-controlled characters. This would challenge players by making the enemies more adaptive and intelligent.
4. **Expanding Game World**: Future updates could involve expanding the game world, adding new environments, biomes, or themes for the player to explore. This could include underwater levels, desert levels, or even space-themed platforms, which introduce new mechanics like altered gravity or unique obstacles.
5. **VR/AR Integration**: Exploring virtual reality (VR) or augmented reality (AR) capabilities could offer a fresh, immersive experience. VR in platformer games provides an opportunity for players to feel physically engaged with the game, while AR could allow interaction with the real world.
6. **Monetization and Distribution**: You could release your game on various platforms, including mobile (Android, iOS), PC, or even consoles. Using digital distribution platforms like Steam or itch.io, coupled with potential in-game purchases, could generate revenue and increase the game's audience.

Limitations

1. **Performance Constraints**: Since the game is built using Pygame, which is not as optimized for high-performance games as engines like Unity or Unreal, the game might struggle with performance issues as more complex features are added (e.g., advanced AI, high-quality graphics, or large levels). Pygame lacks built-in physics engines, 3D rendering capabilities, and other optimizations for larger projects.
2. **Limited Graphics and Animation**: Pygame's capabilities are somewhat basic in terms of high-end graphics and animation. It's well-suited for 2D games but may not handle the more demanding graphical requirements that come with advanced effects, dynamic lighting, or 3D rendering.
3. **Single-Player Focus**: Without multiplayer functionality or community interaction (like leaderboards or competitions), the game's long-term engagement might suffer. Players often seek social experiences in modern gaming, which may limit the replayability and appeal of the game.
4. **Learning Curve for VR/AR or Cross-Platform Development**: Implementing VR, AR, or cross-platform support requires additional technical knowledge and tools outside of Pygame's scope. Transitioning to more advanced engines like Unity may be necessary, but this shift could involve a steep learning curve and added development time.

References

[1] A Game Development Environment to Make 2D Games –

https://www.researchgate.net/publication/347835330_A_game_development_environment_to_make_2D_games

[2] Game Development with Python Using Python -https://eudl.eu/pdf/10.4108/eai.17-6-2022.2322877

[3] Constructing the 2D Adventure Game-Based Assessment System-

https://www.researchgate.net/publication/220886649_Constructing_the_2D_Adventure_Game-Based_Assessment_System

[4] http://www.youtube.com/@ClearCode

[5] http://www.youtube.com/@DaFluffyPotato

[6] http://www.youtube.com/@TechWithTim

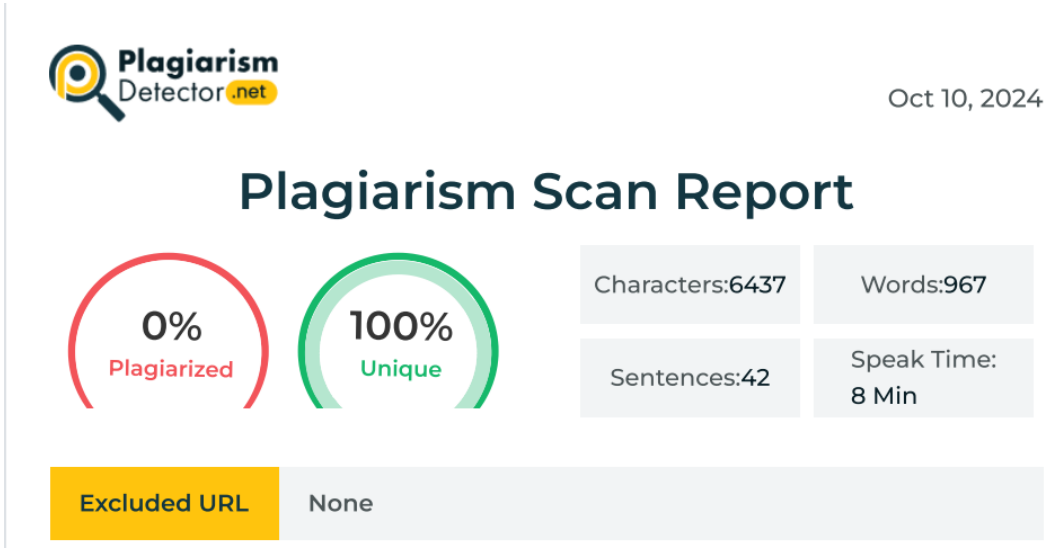
[7] https://www.youtube.com/watch?v=AuD7pwPDzEA

[8] https://ldtk.io/docs/

[9] https://www.youtube.com/watch?v=hBkeJWYMrq8&t=26s

[10] https://realpython.com/pygame-a-primer

Plagiarism Report:




Plagiarism Detector .net
Oct 10, 2024
Plagiarism Scan Report
4%
Plagiarized
96%
Unique
Characters:5809
Words:868
Sentences:46
Speak Time:
7 Min
Excluded URL
None


Plagiarism Detector .net
Oct 10, 2024
Plagiarism Scan Report
0%
Plagiarized
100%
Unique
Characters:10730
Words:856
Sentences:16
Speak Time:
7 Min
Excluded URL
None


Plagiarism Detector .net
Oct 10, 2024
Plagiarism Scan Report
2%
Plagiarized
98%
Unique
Characters:7164
Words:1000
Sentences:44
Speak Time:
8 Min
Excluded URL
None