Final Project

Algorithm & Programming

Amputee Ninja: Adventures!
A Ninja Platformer Game Made With Pygame

Compiled By: Jovanney Rafael Husni
Class: L1BC
Student ID: 2802523105

TABLE OF CONTENTS

Chapter 1: PROJECT SPECIFICATIONS

1.1. Project Description

For my final project, I recreated a platformer game titled *Amputee Ninja: Adventures!*, built entirely using Python and the PyGame library. The project aimed to develop a complete platforming experience with mechanics such as running, jumping, wall-jumping, dashing, and engaging in combat with enemies across multiple levels.

The primary goal of the game is to navigate challenging levels filled with various obstacles and enemies. Each level challenges the player to utilize the ninja's unique abilities to overcome these hurdles effectively. The game emphasizes precise timing, strategic thinking, and quick reflexes, creating a rewarding yet challenging experience for players. Key features include smooth animations, robust collision detection systems, and dynamic enemy behaviors that adapt based on player actions.

The art style and assets, while simple, were meticulously crafted to ensure they blend seamlessly into the gameplay, creating an immersive platforming experience. Furthermore, the modular structure of the project allows for the easy addition of new features and levels, making the game highly extensible.

1.2. Project Link

The GitHub Repository of my final project can be accessed through the following link:
GitHub Repository

Click Me!

The repository contains all the source code, assets, and documentation required to run the project. Detailed setup instructions are also provided to ensure a smooth experience for users and developers alike.

1.3. Essential Algorithms

1. **Movement and Animation Systems**:
    - The game incorporates an intuitive movement system that enables fluid player actions like running, jumping, wall-jumping, and dashing. These actions are implemented using PyGame's event handling and sprite manipulation features.
    - Player animations are dynamically linked to their actions, ensuring visual feedback matches gameplay mechanics. For example, when the player jumps or dashes, the appropriate animation is triggered seamlessly.

    - Collision handling ensures that player movements interact realistically with the game environment, preventing unintended behaviors like clipping through walls.
2. **Collision Detection**:
    - A highly optimized grid-based collision detection system ensures efficient performance. Only tiles near the player are checked for collisions, reducing computational overhead.
    - The system also supports different collision responses, such as stopping the player's movement when hitting a wall or triggering an action when interacting with certain objects.
3. **Enemy AI**:
    - Enemies are programmed with simple yet effective AI that transitions between states like patrolling, chasing, and attacking. These states are governed by player proximity and line-of-sight detection.
    - Pathfinding algorithms allow enemies to navigate around obstacles, creating engaging and unpredictable encounters for players.
    - Animations for idle, attack, and death states enhance the immersive experience.
4. **Level Design and Rendering**:
    - The game uses a modular level design approach with 2D grid-based maps. These maps dictate tile placement, enemy spawn points, and interactive objects.
    - An auto-tiling system ensures tiles blend naturally with one another, reducing manual design effort and enabling rapid level creation.
    - Layers of rendering prioritize the correct visual hierarchy, ensuring players and enemies are displayed appropriately relative to background and foreground objects.
5. **Physics Simulation**:
    - Realistic gravity and velocity calculations dictate the player's and enemies' movement. Acceleration is applied to simulate falling and dashing mechanics.
    - Terminal velocity prevents excessive speeds during freefall, maintaining a balanced and controlled gameplay experience.
    -

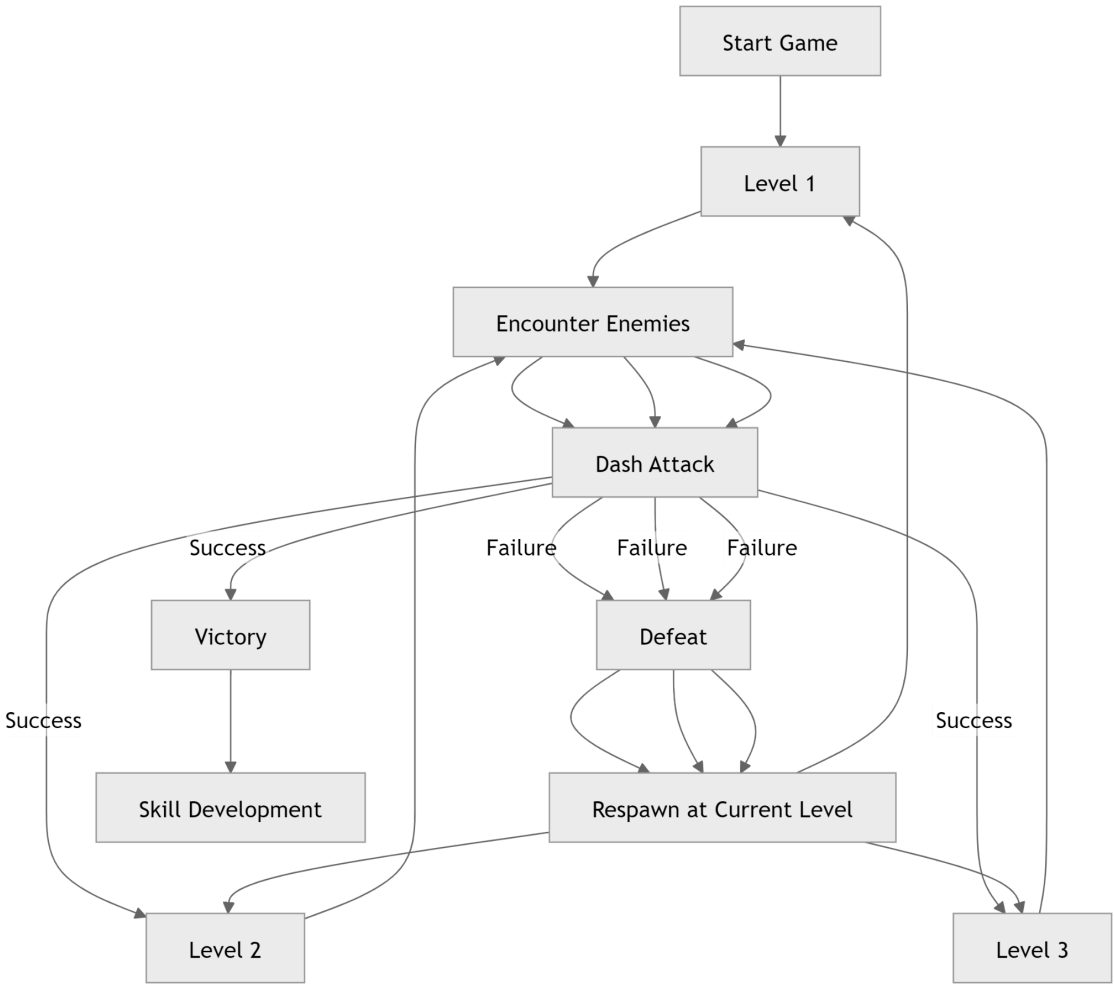
1.4. Modules

1. **PyGame**:
    - The PyGame library is the backbone of the project, providing tools for rendering graphics, handling user input, and managing game events. It is also used to play sound effects and background music, enriching the gameplay experience.
2. **Sys Module**:
    - This module is used to cleanly exit the game application, ensuring all resources are released appropriately.
3. **OS Module**:

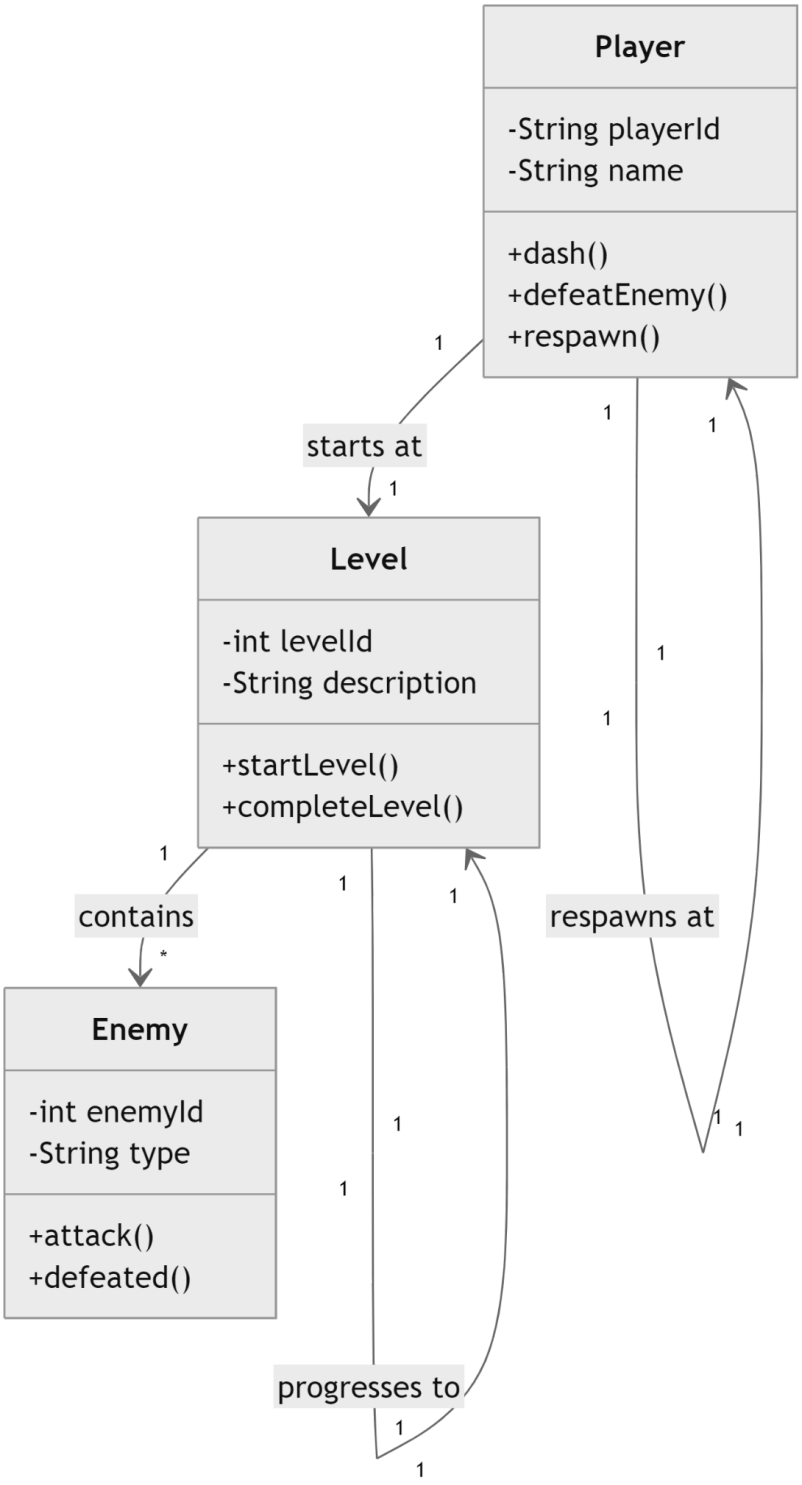
    - The OS module simplifies asset management by dynamically constructing file paths for sprites, sounds, and other resources, allowing for better code organization.
4. **Math Module**:
    - This module aids in implementing physics-related calculations, such as velocity, gravity, and collision detection. It also supports transformations needed for rendering.
5. **Random Module**:
    - Randomized enemy and item placement ensures each gameplay session feels unique, enhancing replayability and keeping the game challenging.

Chapter 2: DIAGRAMS

2.1. Activity Diagram



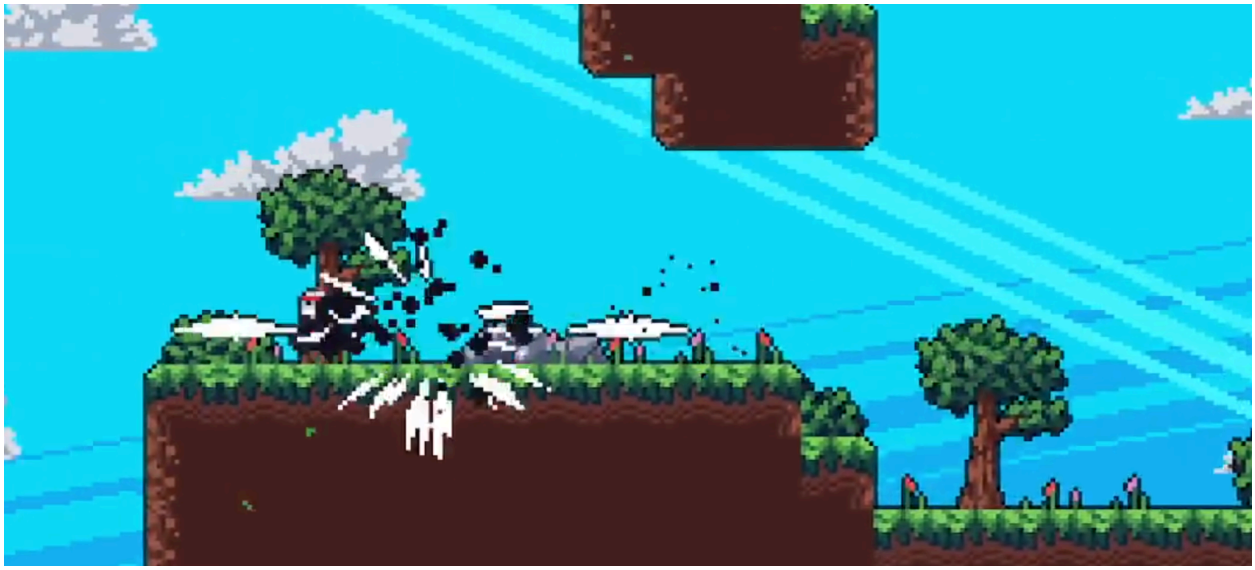
2.2. Class Diagram

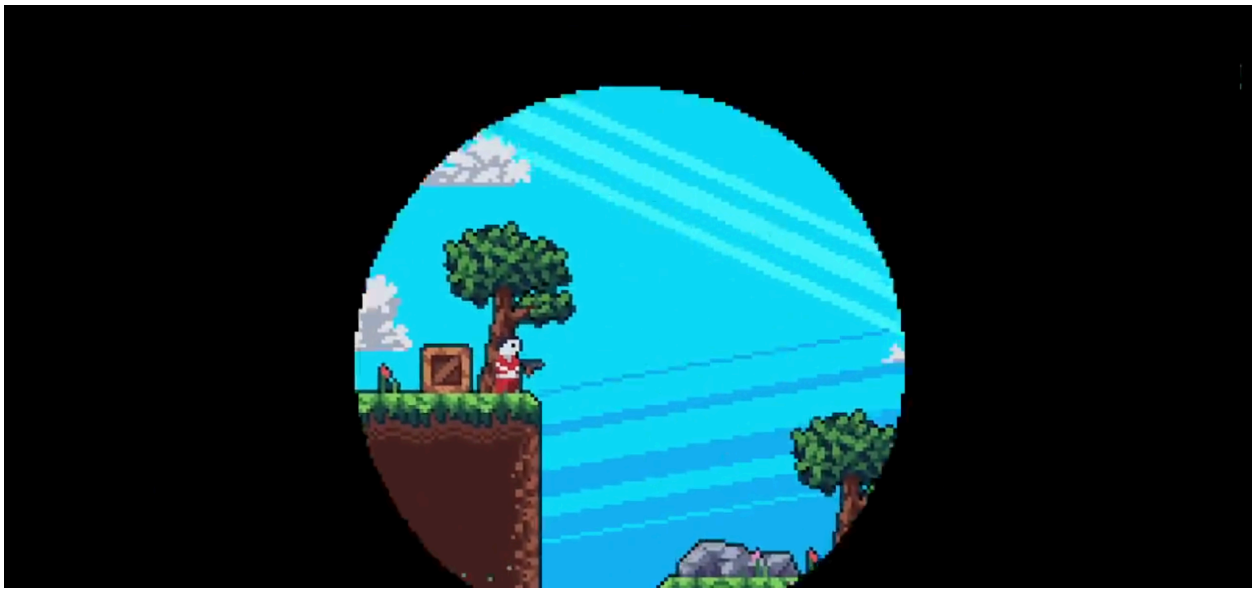


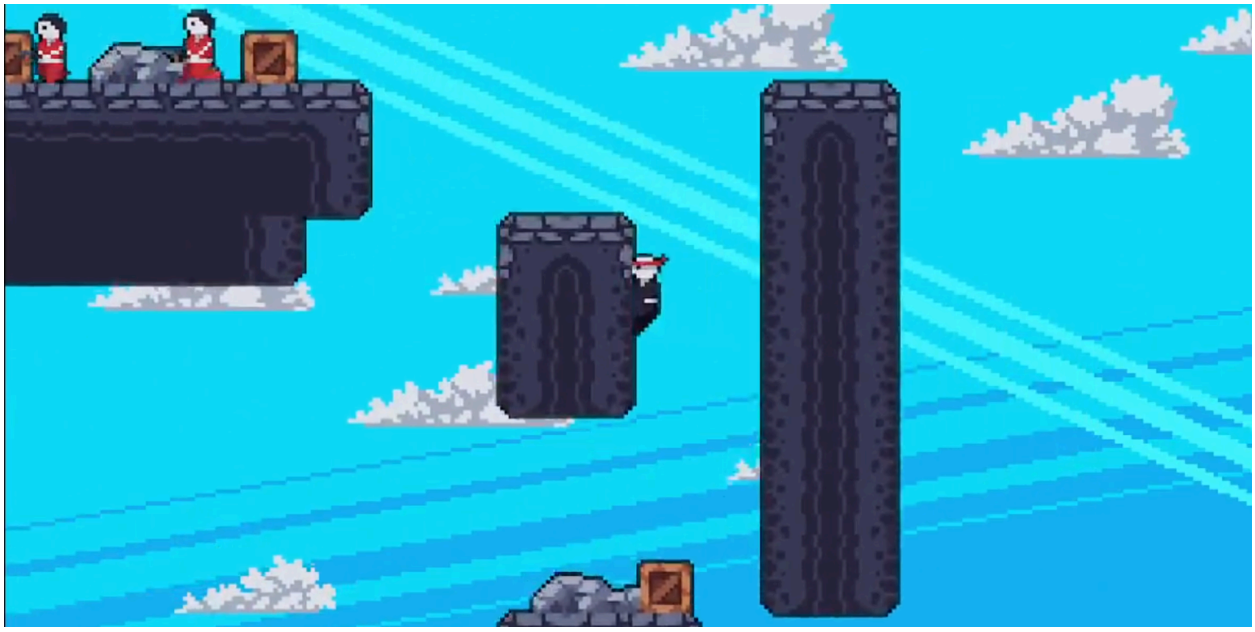
Chapter 3: DOCUMENTATION

3.1. Screenshots

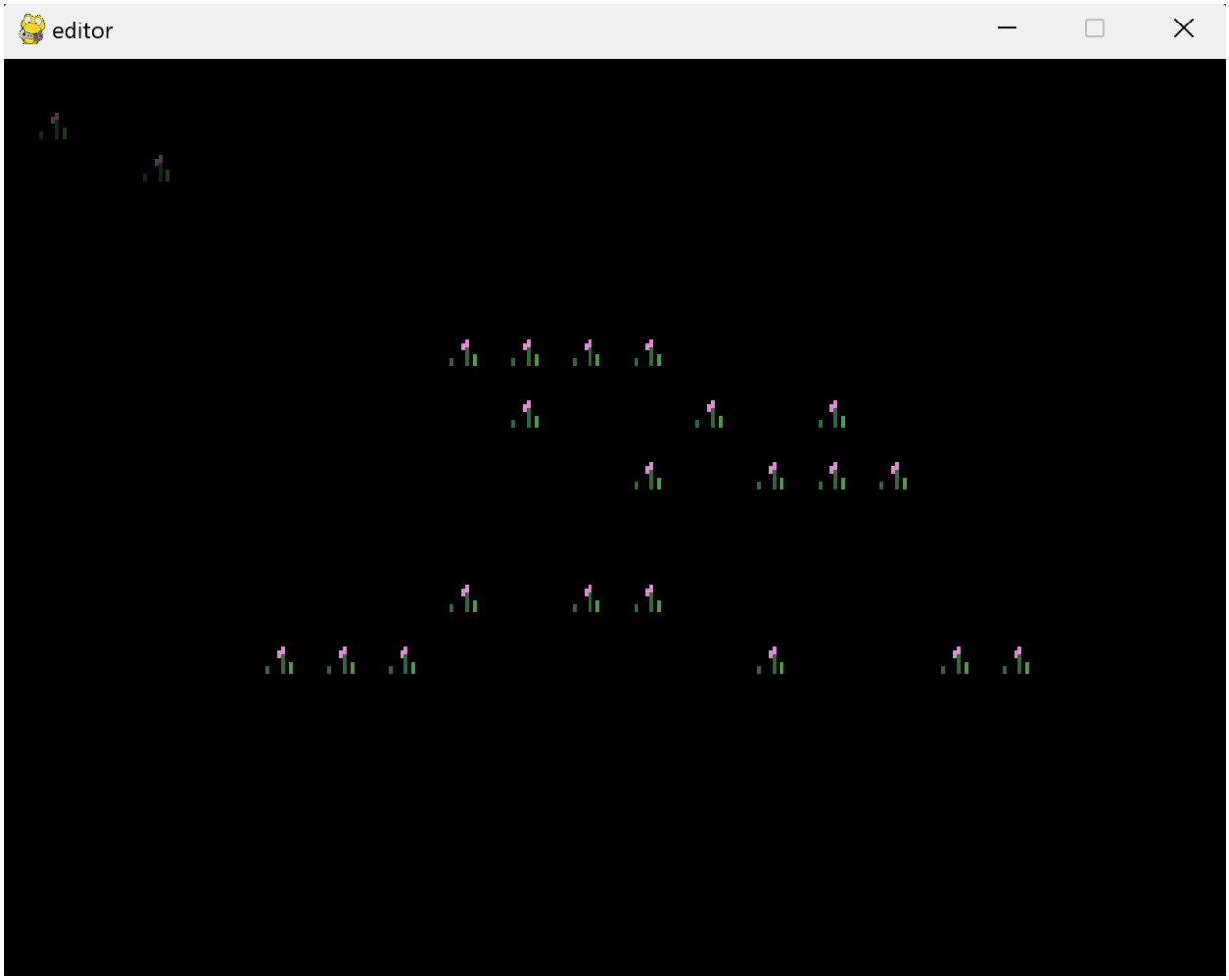
- Gameplay



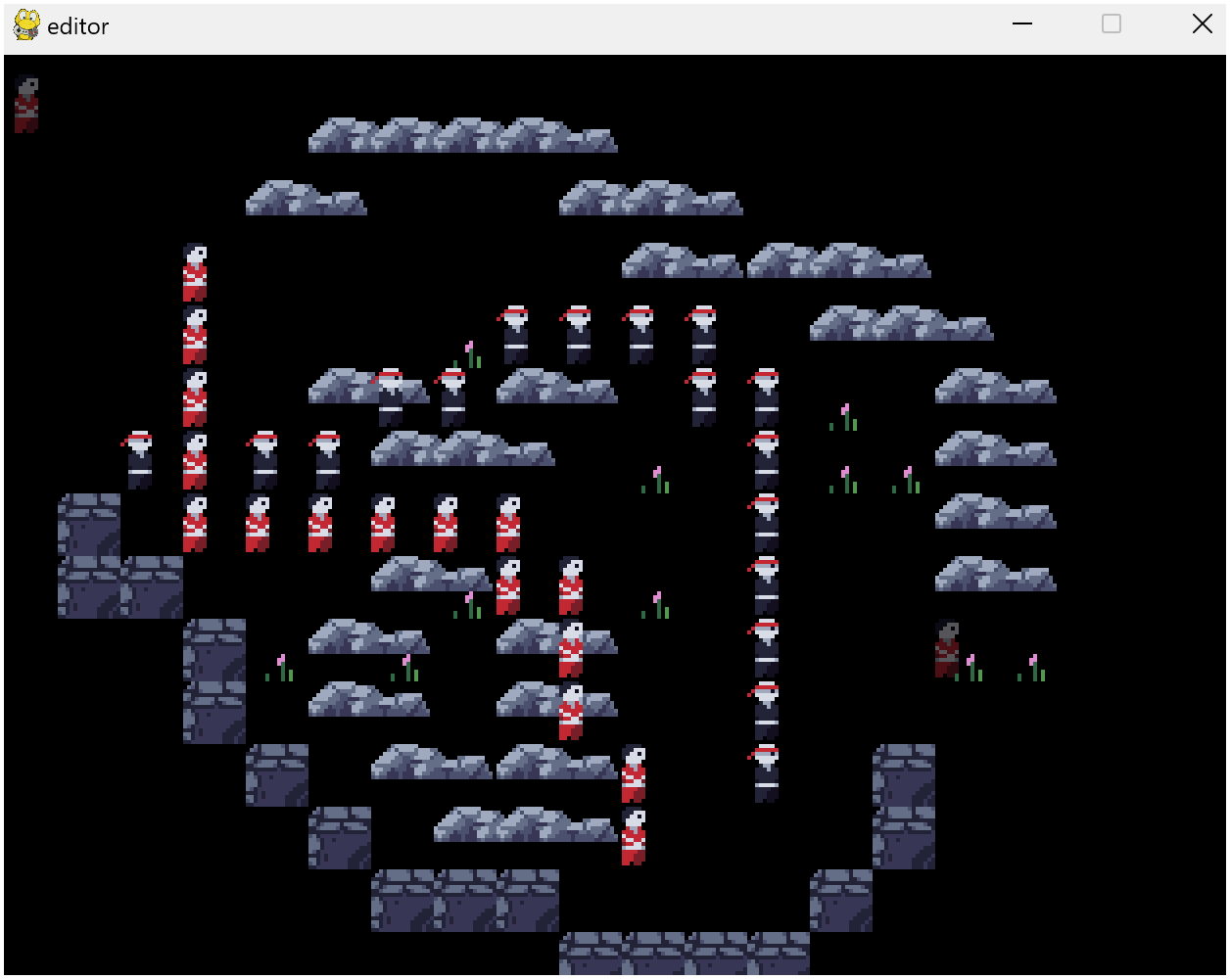




- Level Editor




editor



3.2. Video Demo

A video demonstration of the gameplay and features can be accessed through the following link: Video Demo

Click Me!

Chapter 4: EVALUATION AND REFLECTION

4.1. Lessons Learnt

This project offered valuable insights into various aspects of game development. Key takeaways include:

- The importance of modular programming in managing complex systems like player actions, enemy AI, and level rendering.
- Mastery of PyGame's capabilities, such as event handling, collision detection, and surface manipulation.
- Improved debugging skills, especially in resolving unexpected behaviors in physics calculations and AI interactions.
- Time management and prioritization, particularly in balancing feature development with debugging and testing.

4.2. Future Improvements

Despite its successes, the project has room for improvement:

- **Enhanced AI**: Introducing more advanced behaviors like group coordination or adaptive difficulty.
- **Expanded Content**: Adding more levels, unique enemies, and power-ups to enrich the gameplay experience.
- **Visual Upgrades**: Implementing particle effects, dynamic lighting, and high-quality animations.
- **Customization Options**: Allowing players to configure controls, difficulty levels, and visual settings.

REFERENCES

- Pygame Documentation
- Tutorial: *Pygame Platformer Tutorial - Full Course*
- ChatGPT