TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

A Project Report

On

Platform Game using python

Submitted By:

Arpan Bhattarai (HCE079BEI011)

Badal Thapa (HCE079BEI013)

Bhupendra Karki (HCE079BEI014)

Submitted To:

Department of Electronics and Computer Engineering

Himalaya College of Engineering

Chyasal,Lalitpur

Feb-16,2025

ACKNOWLEDGEMENT

We express our sincere gratitude to everyone who has supported and guided us throughout the process of developing this 2D platform game using Python and Pygame. This project would not have been possible without their valuable contributions and assistance.

We extend our heartfelt thanks to our mentor, Sushant Pandey, whose expertise and guidance have been instrumental in shaping the development of this game. His insightful feedback and continuous encouragement have played a crucial role in refining our approach and overcoming technical challenges.

We would also like to express our appreciation to our friends and colleagues for their meaningful discussions, brainstorming sessions, and unwavering support. Your input has helped us improve the overall gameplay experience and navigate complex programming concepts.

This project is a result of collective effort, and we are deeply grateful to everyone who contributed. While any shortcomings in this game are solely our responsibility, the encouragement and support from those around us have significantly enriched its quality.

Thank you.

Arpan Bhattarai (HCE080BCT011)

Badal Thapa (HCE080BCT013)

Bhupendra Karki (HCE080BCT014)

ABSTRACT

AxeMan is a 2D platformer game built using Python and Pygame, featuring a structured level-based progression system. Players control Hugo, the protagonist, navigating through three levels filled with platforms, moving obstacles, spikes, and interactive crates. The game incorporates realistic physics for jumping, collisions, and crate interactions, allowing players to push or move objects strategically.

Each level is loaded dynamically using JSON files, enhancing modularity and ease of expansion. The game architecture follows an object-oriented approach, with separate modules handling entities like the player, platforms, crates, spikes, and level management. The main game loop in main.py ensures smooth gameplay, handling user inputs, movement mechanics, gravity simulation, and level transitions. A checkpoint system is integrated to save progress after each level.

With pixel-perfect collision detection and a responsive control system, **AxeMan** challenges players to complete all levels without dying. The structured and scalable codebase allows for future enhancements, making it a strong foundation for further game development.

Table of Contents

1. INTRODUCTION

1.1 Background Introduction

The development of 2D platformer games has gained popularity due to their engaging mechanics, accessibility, and nostalgic appeal. This project, AxeMan, is a 2D platformer built using Python and the Pygame library. The player controls Axe, navigating through a series of challenging levels while avoiding obstacles such as spikes and gaps. The game features interactive objects like crates and moving platforms, requiring strategic movement to progress. Designed with a modular and scalable structure, Hugo the Huge provides a dynamic and immersive gaming experience.

1.2 Motivation

The motivation behind this project is to create an engaging and well-structured platformer that challenges players' timing, precision, and problem-solving skills. By utilizing Python, a widely used and versatile programming language, the game ensures accessibility for future modifications and expansions. The inclusion of interactive elements such as physics-based object movement, checkpoints, and moving platforms enhances the gameplay experience.

1.3 Objectives

The main objectives of this project are:

- Develop core platformer mechanics, including player movement, jumping, collision detection, and interactive objects.
- Implement a dynamic level system using JSON files for flexible level design and future expansion.
- Introduce game progression elements such as checkpoints, increasing difficulty, and responsive controls to enhance playability.

1.4 Scope

This project focuses on designing and developing a structured 2D platformer game using Python and Pygame. The game includes three levels, each featuring progressively difficult challenges with moving platforms, spikes, and crates that the player must navigate strategically. The implementation follows an object-oriented approach, ensuring a modular and scalable design. Future enhancements may include additional power-ups, animated character sprites, improved enemy AI, and more intricate level designs to further enrich the gameplay experience.

2. LITERATURE REVIEW

2.1 Previous Works on Platformer Games

Several classic and modern platformer games have inspired the development of Hugo the Huge, including Super Mario Bros., Celeste, and Hollow Knight. These games emphasize precise movement, obstacle avoidance, and level progression. Our project builds on these core mechanics while introducing unique gameplay elements such as moving platforms, physics-based crates, and checkpoints to enhance player engagement

2.2 Pygame And Its Capabilities

Pygame is a powerful Python library for game development, providing tools for rendering, input handling, collision detection, and audio management. In AxeMan the Huge, Pygame plays a crucial role in managing sprite animations, physics-based interactions, and real-time event handling. The modular design of Pygame allows for scalable and efficient game mechanics, making it an ideal choice for this project.

2.3 Game Design Patterns

This game follows essential game design patterns to ensure a structured and maintainable codebase. The Sprite Pattern is used for handling game objects such as the player, enemies, and platforms. Additionally, the State Pattern helps manage different game states, such as menus, gameplay, and level transitions. The Observer Pattern is implemented to track events like player interactions with checkpoints and obstacles, ensuring a responsive and immersive experience.

3. METHODOLOGY

3.1 Development Tools

- Programming Language: Python 3.x
- Game Library: Pygame
- Graphics: Sprites and tile-based level design for characters, platforms, and obstacles
- Audio: Sound effects for actions like jumping, collisions, and checkpoints

3.2 Player Character Control

The player controls AxeMan using the A and D keys to move and the spacebar to jump. The physics system ensures smooth movement and gravity, making platforming mechanics responsive and fluid.

3.3 Platforms and Obstacles

- Static and moving platforms are placed throughout levels, requiring precise jumps.
- Spikes serve as hazards that instantly reset progress upon contact.
- Some levels feature crates that can be pushed to solve puzzles or reach higher platforms.

3.4 Level Progression and Checkpoints

- The game consists of three levels, each increasing in difficulty.
- Checkpoints are placed at the end of each level to save progress, preventing full restarts.
- Players must navigate all levels without exhausting lives to complete the game.

3.5 Collison Detection and Interactions

- Collision detection ensures the player can land on platforms, push crates, and avoid spikes.
- The game implements a physics system to handle jumping, falling, and moving objects dynamically.

3.6 Game Over and Restart

- The game ends when the player falls off the screen or touches spikes.
- A game over screen appears, allowing the player to restart from the last checkpoint or quit.

4. CODE EXPLANATION

4.1 Main Game Loop

- Initializes Pygame and sets up the game window.
- Loads assets like images and level data.
- Runs the game loop, handling events, updating game objects, and rendering everything to the screen.

4.2 AbstractObject Class

- Serves as a base class for all game objects.
- Handles object properties like position (x, y), image (img), and visibility (visible).
- Implements:
    - draw(win, offset): Renders the object on the screen with an offset.
    - clicked(pos): Detects if the object was clicked using a bounding box.

4.3 Player Class

- Represents the main character controlled by the player.
- Handles movement, jumping, attacking, and animations.
- Implements:
    - Movement System: move(keys), allowing movement via A/D keys and running with Left Shift.
    - Jumping Mechanism: handle_jump(keys), enabling jumping via the Space key.
    - Attack System: handle_attack(), detecting mouse clicks for attacks.
    - Physics:
    - apply_gravity(), affecting downward movement.
    - land(obj), ensuring proper landing on platforms.
    - fall(), handling free falls when not on a platform.
    - Animation & Image Handling:
    - set_image(), dynamically updating the player's animation.
    - Uses ACTIONS dictionary for different animations (walking, running, attacking, falling, etc.).

4.4 Platform Class

- Represents the platforms that can be static or moving.
- Implements:
    - move(), moving the platform horizontally or vertically based on predefined travel distance.
    - draw(win, offset), rendering the platform and calling move().

4.5 Level Management

- Loads level data from JSON files.
- Places game objects (platforms, spikes, doors, and crates) in the correct positions.
- Manages interactions between objects and the player.

4.6 Collision Detection

- The player's collide(obj) method detects collisions with objects using pygame.mask.
- This enables precise collision detection for platforms, enemies, and obstacles.

4.7 Scoring And Level Progression System

- Players progress by completing levels without dying.
- The game tracks progress through checkpoints and increases difficulty gradually.

4.8 Checkpoint System

- Implements checkpoints at the end of each level.

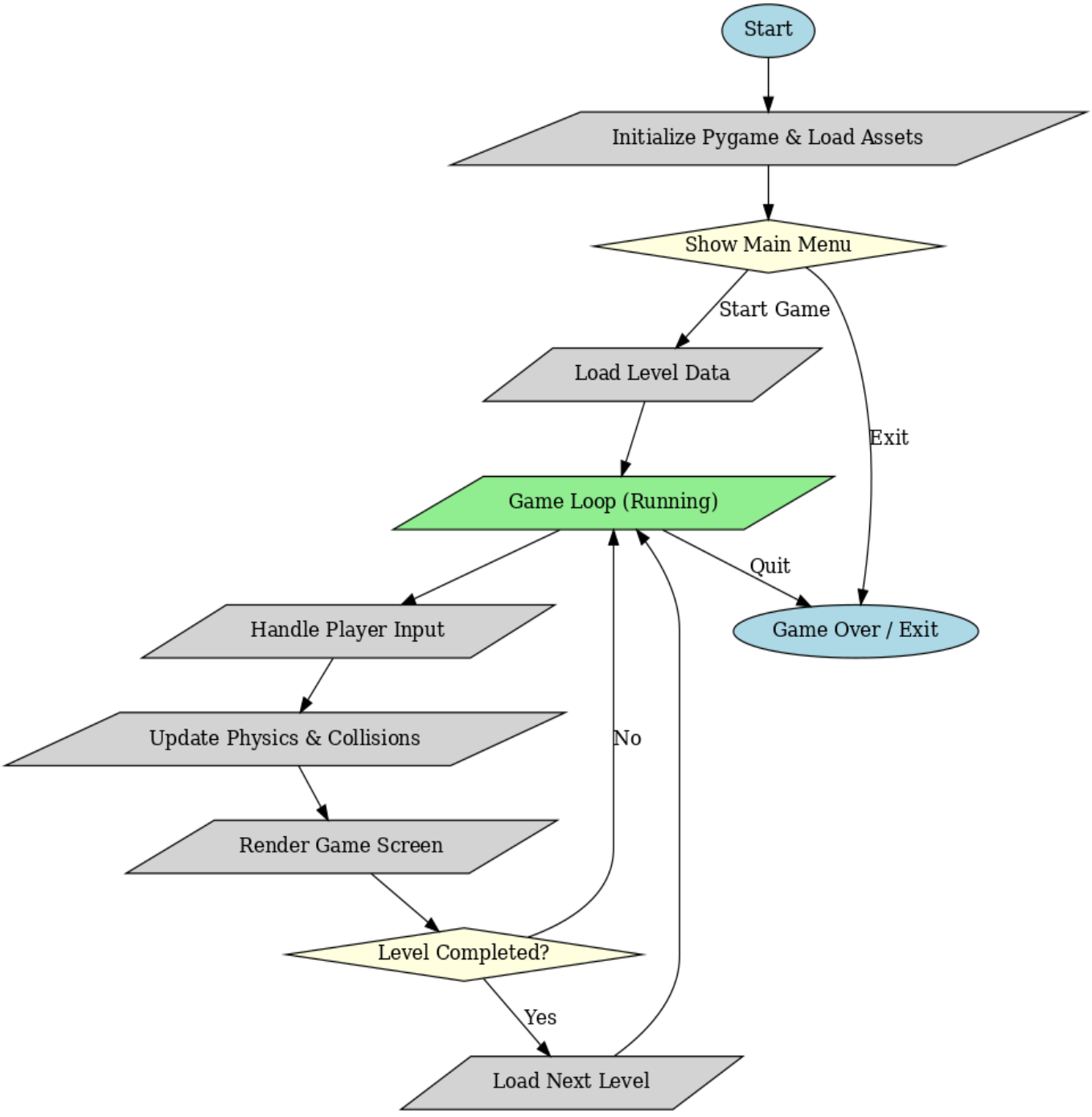
4.5 Game Over Logic

- The game ends when the player falls off the screen or touches spikes.
- Displays a game over screen.

4.9 Game Loop

- The core game logic runs inside a continuous loop:
    - Event Handling: Detects keyboard inputs (left, right, jump).
    - Game State Updates: Handles player movement, platform interactions, and collisions.
    - Rendering: Draws sprites and updates animations.

5. Flowchart


Start
Initialize Pygame & Load Assets
Show Main Menu
Start Game
Exit
Load Level Data
Game Loop (Running)
Quit
Game Over / Exit
Handle Player Input
Update Physics & Collisions
No
Render Game Screen
Level Completed?
Yes
Load Next Level

6. RESULT AND ANALYSIS

6.1 Gameplay Experience

- The game provides an engaging and interactive experience with well-balanced difficulty progression.
- The level design and mechanics encourage exploration, strategic movement, and puzzle-solving.
- The checkpoint system ensures smooth gameplay progression and reduces player frustration.

6.2 Performance Evaluation

- The game runs efficiently at 60 frames per second (FPS) on most systems.
- The Pygame engine effectively handles rendering, animations, and physics with minimal lag.
- Collision detection and object interactions are optimized for a seamless experience.

6.3 Game Enhancements

- Introducing new enemy types and obstacles to increase difficulty variety.
- Expanding levels with more interactive elements, such as moving platforms and hidden areas.
- Adding power-ups such as:
    - Double Jump: Allows the player to perform an extra jump mid-air.
    - Shield: Grants temporary invincibility against hazards.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

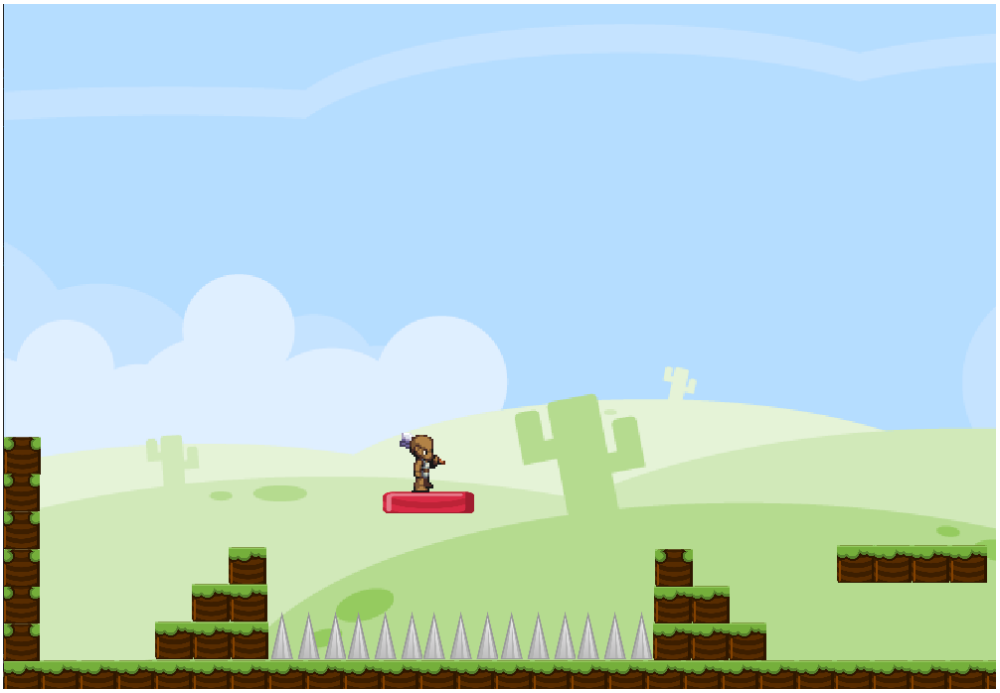
This project successfully demonstrates how a 2D platformer game can be created using Python and Pygame. It incorporates core game mechanics such as movement, jumping, object interactions, and collision detection, providing an engaging and challenging experience. The game is expandable and can be enhanced with additional features such as improved visuals, sound effects, power-ups, enemies, and smoother level transitions.

7.2 Future Enhancements

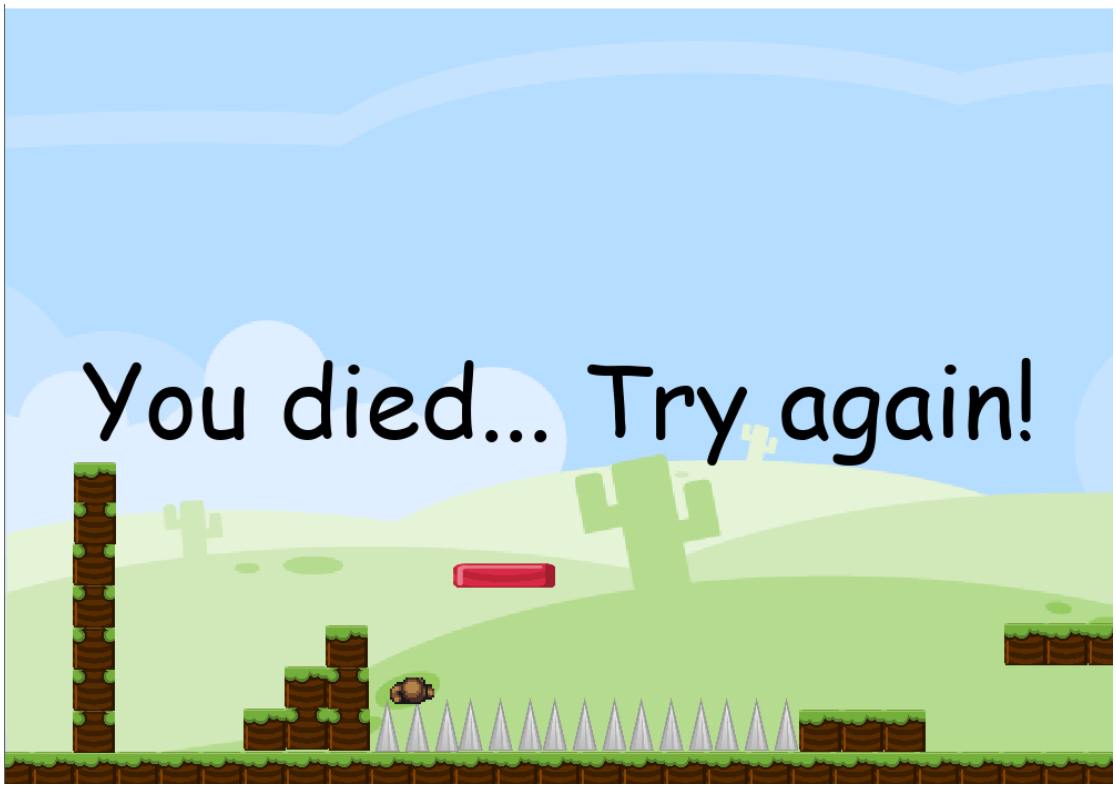
- **Enemies and Combat System**: Adding enemies that can deal damage to the player and can also be defeated using different attack methods.
- **Music and Sound Effects**: Adding background music and sound effects for jumps, landings, item pickups, and enemy interactions to enhance immersion.
- **Proper Scoring System**: Implementing a points-based system to reward players for defeating enemies, collecting items, and completing levels efficiently.
- **Health Bar and Player Lives:** Implementing a health system where the player can take damage from enemies and hazards, with the ability to recover using health pickups.
- **Power-ups:** Introducing gameplay-enhancing elements such as temporary invincibility, speed boosts, or double jumps.
- **Smooth Level Transitions:** Creating visually appealing animations when progressing from one level to another, rather than abrupt scene changes.
- **Different Weapons:** Allowing the player to use various weapons, such as melee attacks, ranged projectiles, or special power-based attacks.
- **Enhanced Animations:** Adding smoother character movements, better enemy animations, and environmental effects like particle explosions on object interactions.
- **Dynamic Weather and Day/Night Cycle:** Adding rain, fog, or time-based lighting changes for more immersive environments.
- **Side Quests and Challenges:** Adding optional missions within levels to encourage exploration and replayability.
- **And Many More**

8. APPENDIX

8.1 Screenshots




You completed level 1!


You died... Try again!

9. References

- Pygame Documentation: https://www.pygame.org/docs/
- Tech With Tim [YouTube]
- Freecodecamp.org