COMPUTER GRAPHICS MINI PROJECT REPORT

TRIBHUVAN UNIVERSITY

A Project Report

On

RUNFORGE: Design and Development of a Modular Endless Runner Game Using Python and Pygame

Submitted By:
Krrish Nyoupane (HCE081BEI019)

Submitted To:
Department of Electronics and Computer Engineering Himalaya College of Engineering Chyasal, Lalitpur

Feb-22-2026

ACKNOWLEDGEMENT

I express my sincere gratitude to all those who supported and guided me throughout the process of conducting this report on RUNFORGE: Design and Development of a Modular Endless Runner Game Using Python and Pygame. This project would not have been possible without their valuable guidance, encouragement, and contributions. I extend my heartfelt thanks to my supervisor, Sushant Pandey, whose expertise and constructive feedback played a crucial role in shaping the technical direction and overall structure of this project. His insights into software design, problem-solving approaches, and system implementation greatly enhanced the quality and depth of this work. I would also like to express my appreciation to my friends and colleagues for their meaningful discussions, brainstorming sessions, and continuous moral support throughout the development process. Their feedback helped refine gameplay mechanics, improve system performance, and strengthen the final implementation. Finally, I acknowledge the open-source community and the developers of Python and Pygame, whose tools and documentation made the development of RUNFORGE possible. Although any shortcomings in this report remain solely my responsibility, the guidance and support I received significantly enriched both the project and this documentation. Thank you. Krrish Nyoupane (HCE081BEI019)

ABSTRACT

Abstract

RUNFORGE is a modular 2D endless runner game implemented in Python using the Pygame library. The project demonstrates practical game development techniques, real-time system design, and object-oriented programming through a compact, extensible architecture designed for clarity and reuse. Core gameplay mechanics include automatic forward movement, single and cooldown-based double jumps, a time-limited shield power, falling asteroid hazards, ground obstacles, a hearts-based health system, speed-scaling difficulty, and persistent best-score tracking.

The implementation emphasizes modularity: player, obstacle, asteroid, particle, audio, UI, and state management systems are separated into well-defined components inside the main codebase to aid readability and future extension. Visual polish is provided via sprite animations, a particle system (fire/spark/smoke), parallax backgrounds (forest & desert), and glassmorphism UI elements. Audio feedback uses pygame.mixer with graceful fallbacks to silent buffers to avoid runtime crashes when audio assets are missing.

Performance considerations (fixed 60 FPS loop, asset existence checks, and lightweight particle culling) ensure smooth gameplay on modest hardware. The project serves as a practical case study showing how Python and Pygame can produce a responsive, feature-rich real-time game while adhering to software engineering best practices such as modular design, persistence, and graceful degradation of missing assets.

Table of Contents

1. Executive Summary

JUMPING ENGINEER (RUNFORGE) is a 2D endless side-scrolling runner game built with Python and Pygame. The player runs automatically
and must clear ground obstacles and avoid falling asteroids. Features include increasing difficulty (speed scales with score),
health/lives system, cooldown double-jump, shield ability, animated sprites, particle effects, glassmorphism UI, two selectable maps,
persistent best-score saving, and full sound support. The codebase is implemented in a single 'main.py' with external assets.

2. Technology Stack

Core tools: Python 3.10+, pygame 2.x, pygame.gfxdraw, math, random, json, os, sys. Display: fixed 1100x700 window (optional fullscreen),
pygame.DOUBLEBUF, target 60 FPS. Fonts: cascading fallback for bundled and system fonts.

3. Project Structure

All code in main.py with well-documented sections and classes: UI helpers (outlined text, glass panels, vignette), Particle, FireText,
GlowButton, Player, Obstacle, Asteroid, and Game class with state machine (start_menu, playing, settings, tutorial, game_over).

4. Game Mechanics

Player physics: base speed 8 px/frame, jump velocity -20, gravity +1 px/frame^2, double jump with 10s cooldown, shield 5s with 10s cooldown,
3 hearts max, regain 1 heart per 20 points, invincibility frames 120 frames after damage. Obstacles: 3 ground types, randomized sizes,
spawn frequency scales with score. Asteroids: falling hazards spawn above screen, fall speed 3-7 px/frame, rotate, spawn every ~5-6s.
Scoring: +1 per cleared obstacle, milestone sounds every 10 points, +1 heart every 20 points, best_score saved to JSON.

5. Visual & Audio Systems

Particle system: fire, spark, smoke with defined sizes and lifetimes. Background: forest and desert maps with layered parallax. UI: glassmorphism
panels, neon glow buttons, heart parametric curve. Audio: pygame.mixer with silent buffer fallback for missing files; sounds include start,
jump, hurt, die, score, shield, click, pick.

6. Controls & Input

Keyboard: SPACE/UP/W to jump/double-jump, S for shield, P/ESC for pause, R restart (game over), M menu. Mouse/touch: on-screen jump/shield buttons.
Buttons play hover/click sounds and have glow-on-hover effects.

7. Persistence & Assets

Files persisted: best_score.json, settings.json (sound/fullscreen/map), optional game_data.json and game_stats.json. Asset list includes player
sprites, obstacle images, asteroid gif, UI buttons, countdown images, and mp3 audio files. The game gracefully substitutes placeholders if assets
are missing so gameplay remains functional.

8. Known Issues & Notes

Documented issues: duplicate event handler (unreachable block), heart overflow (add_heart triggered repeatedly when score % 20 == 0),
filename mismatch (hurt.jpg vs hurt.png) fallback to colored rectangle, and score/speed text overlap at very high speeds.

9. Conclusion & Future Work

RUNFORGE demonstrates modular game architecture, responsive gameplay, and polished visuals using Python and Pygame. Future enhancements
include networking (leaderboards), additional power-ups, configurable control mapping, more levels, and performance profiling for particle
systems and collision handling.

Appendices

Key Controls: W/A/S/D translate (dev tools), SPACE jump, S shield, P pause, R restart. Additional screenshots and figures should be
included in the images/ folder when preparing the final printed report.

Bibliography

1. H. Ng and R. Grimsdale, “Computer graphics techniques for modeling cloth,” IEEE Computer Graphics and Applications, vol. 16, no. 5, pp. 28–41, 1996.

2. Pygame Documentation — pygame.org (official docs) — consulted for function references and audio handling.

3. Tech With Tim, 'Pygame Tutorial Series' (YouTube) — video tutorials consulted for beginner-to-intermediate pygame functions and patterns.

4. Tools / AI Credits: GitHub Copilot (code assistance), Google Gemini (character asset generation).