Space Invaders using pygame

Team Members:

- Abhinav MB – PES2UG25AM145
- Likith B – PES2UG25AM143
- Madhusuthan V – PES2UG25AM148
- Naonith K Rao - PES2UG25CS324

Batch No: G section

Table of Contents

Problem Statement

1

Interactive Arcade Clone

Our objective was to create an engaging arcade game clone that demonstrates core game development principles, such as robust game loops, precise collision detection, efficient sprite management, and intuitive user input handling.

2

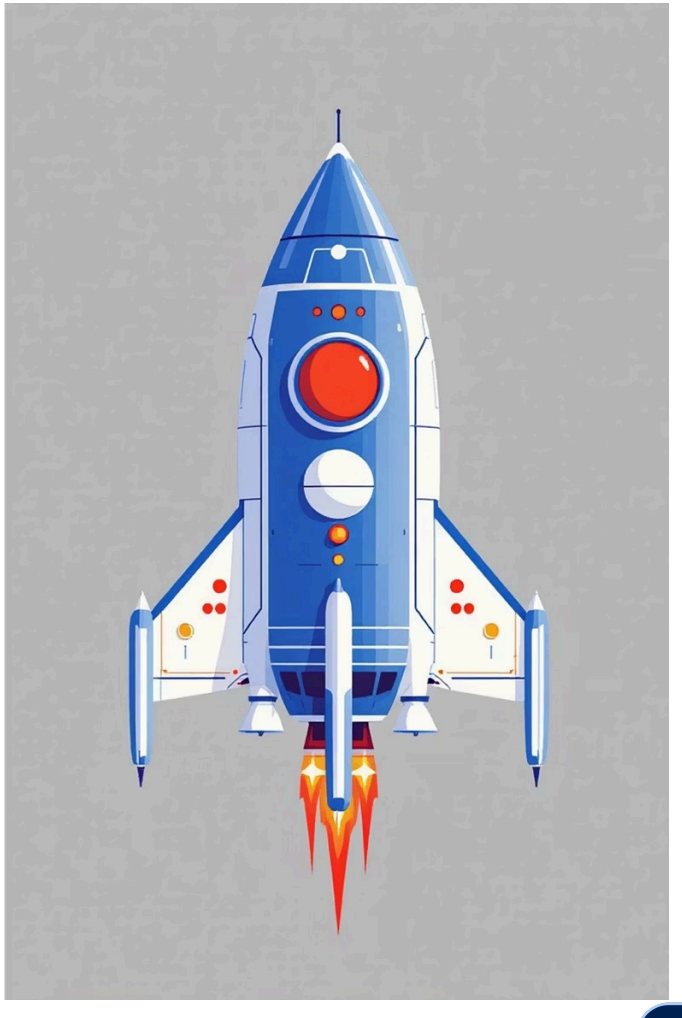
Fundamental Learning

This project served as a practical exercise in learning game development fundamentals, applying Object-Oriented Programming (OOP) concepts, and managing real-time input for an interactive user experience.

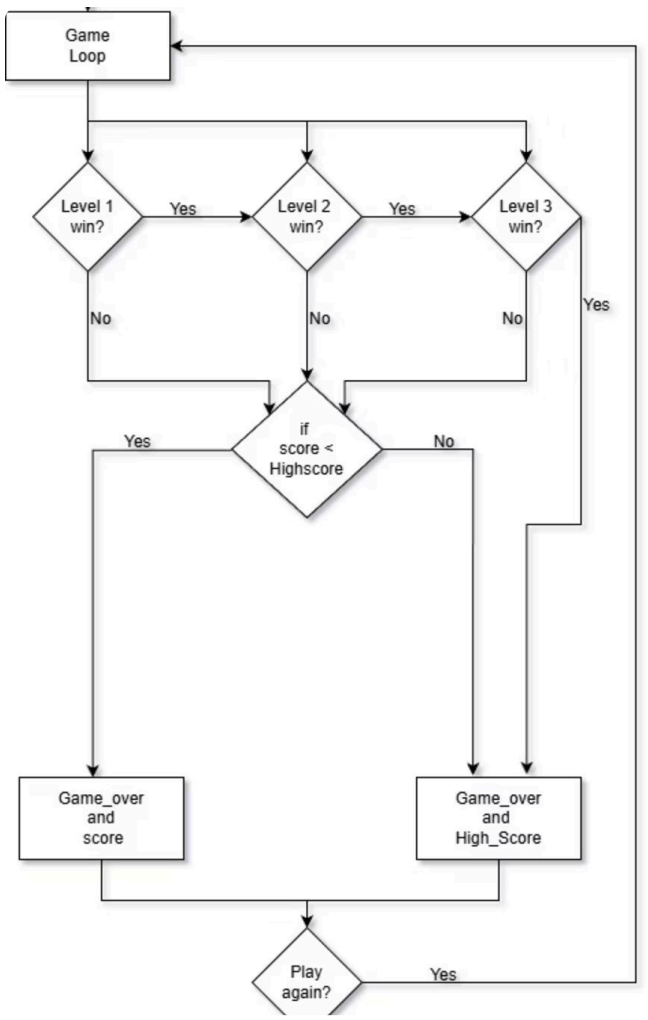
3

Key Constraints & Goals

We aimed for cross-platform playability, simple and responsive controls, a smooth frame rate for an enjoyable experience, and a modular code architecture for ease of maintenance and future expansion.



Block Diagram



Our Space Invaders game architecture can be visualized through a clear block diagram, illustrating the interaction between various core components. At its heart is the **Game Loop**, which orchestrates the entire experience, continuously processing events, updating game states, and rendering graphics.

- Player starts at the **Main Menu** and enters the **Game Loop**.
- The player must clear **Level 1 → Level 2**
- If the player fails any level, the game ends and shows the score.
- After finishing all levels, the game checks if the score beats the **High Score**.
- Depending on the result, either **Game Over with Score** or **Game Over with High Score** is displayed.
- The player is then asked whether they want to **Play Again** or **Exit**.

Input Handler

Detects keyboard events (movement, firing).

Game Logic

Updates positions, handles collisions, manages score and lives.

Asset Manager

Loads and manages images, sounds, fonts.

Graphics Renderer

Draws all game objects to the screen.

Approach Used

Utilized Pygame Library

Leveraged Pygame for comprehensive graphics rendering, efficient event handling, and robust sound playback capabilities, forming the backbone of the game's functionality.

Object-Oriented Design

Designed dedicated classes for critical game entities: **Player** for spaceship control, **Enemy** for various alien types, **Bullet** for projectiles, and **Power-ups** for enhanced gameplay.

Structured Implementation

Incorporated clear loops for game states, defined modular functions for specific tasks, and utilized classes and objects extensively in conjunction with Pygame's functionalities.

Incremental Difficulty

Developed a dynamic difficulty system that escalates the challenge by progressively increasing enemy speed, attack frequency, and introducing more complex movement patterns.

Iterative Development & Debugging

Used an iterative development cycle, continuously testing and refining the game. Debugging was facilitated by referencing community tutorials and Pygame documentation.

Sample Input/Output

Input Mechanisms

The game relies on intuitive keyboard controls for player interaction, ensuring a classic arcade feel.

- **Keyboard Arrow Keys (Left/Right):** These inputs govern the horizontal movement of the player's spaceship across the bottom of the screen. Precise control is crucial for dodging enemy fire and positioning for attacks.
- **Spacebar:** Pressing the Spacebar initiates a single bullet fire from the player's spaceship. This action is designed to be responsive, allowing for rapid-fire engagements against the invading aliens.

Expected Outputs

The game provides clear visual and auditory feedback for all player actions and game events.

- **Player Spaceship Movement:** The player's spaceship smoothly translates left or right on the screen in direct response to arrow key presses, reflecting real-time control.
- **Bullet Firing:** Upon pressing the Spacebar, bullets are visibly rendered, traveling upward from the spaceship's current position towards the enemy formation.
- **Alien Behavior:** Alien invaders move predictably across the screen horizontally, gradually descending in waves, creating a persistent threat.
- **Score Increment:** When a player's bullet successfully hits an alien, the alien disappears, and the player's score prominently updates on the display.
- **Game Over State:** If aliens reach the bottom of the screen or the player loses all their lives, a distinct "Game Over" screen is displayed, indicating the end of the current play session.

> **Example Scenario:**
>
> A player presses the Right arrow key, causing their spaceship to glide rightward across the screen. They then press the Spacebar, launching a bullet. The bullet swiftly ascends, striking an alien. The alien explodes, vanishing from the screen, and the player's score increases by 10 points. If multiple aliens manage to breach the defensive line and reach the bottom, the game transitions to a "Game Over" state, ending the round.

Challenges Faced

Responsive Player Movement

Achieving smooth and accurate player spaceship movement, especially when handling simultaneous key presses (e.g., moving and firing), demanded precise event handling and state management.

Collision Detection Accuracy

Implementing collision detection between bullets.

Game Difficulty Balancing

- Ensuring consistent score display
- Bullet mechanics of enemy and player

Projectile & Pattern Debugging

Debugging complex issues related to bullet trajectories, alien movement patterns, and their interactions demanded thorough testing and systematic code analysis to identify and resolve subtle bugs.