PYTHON JACKFRUIT PROJECT

**GALAXY DEFENDER**

CLASS-C6

TEAM MEMBERS

GUDIVADA BHANU TEJA PES1UG25CS194 24

G VISHAL PES1UG25CS182 22

HARSH LOHIA PES1UG25CS199 37

GAURAV RAJ PES1UG25AM129 43

**Space Shooter Game – Problem Statement**

­­­­­­­­­­­Modern arcade players expect games that are **fast, lightweight, and simple to control**, without needing installation-heavy engines or complex input systems. However, many space-shooter games available today are either too advanced, require strong hardware, or contain unnecessary features that make them difficult for beginners to enjoy or learn from.

Players looking for a quick, fun, and engaging experience often struggle with:

* Games that are too large or slow on normal computers
* Complicated control schemes
* Overly detailed graphics that reduce performance
* Lack of basic feedback like score or lives
* Codebases that are too difficult for programming beginners to understand

The **Pygame Space Shooter Game** solves these problems by delivering a clean, responsive, pixel-art arcade shooter that runs on any computer. It uses simple arrow-key movement, basic enemy generation, and intuitive shooting mechanics, making it perfect for casual players and beginner programmers.

Approach / Methodology

## 1. Game Window & Environment Setup (Pygame GUI)

* Pygame is used to create a 2D game window of fixed size.
* The background is kept black to simulate outer space.
* Pixel-art graphics are used for:
  + the player spaceship
  + falling asteroids
  + bullets
* FPS control ensures smooth animation.

## 2. Player Movement & Controls

* Only keyboard controls (no mouse):
* **Left Arrow** → move left
* **Right Arrow** → move right
* Movement is limited to horizontal direction only.
* Speed is controlled using predefined constants.
* Player remains inside the screen boundaries.

## 3. Bullet Mechanics

* Bullets spawn from the top-centre of the spaceship.
* Each bullet travels upward at a fixed speed.
* Bullets are stored inside a Pygame Sprite Group for easy updates.
* A bullet is removed when:
* It moves outside the screen
* It collides with an asteroid

## 4. Enemy (Asteroid) Generation

* Asteroids are generated at fixed intervals using pygame.time.set\_timer()
* Each asteroid:
* appears at a random X position
* falls downwards at a constant speed
* is represented using hand-made pixel-art
* If an asteroid touches the player, one life is lost.

## 5. Collision Detection

Pygame’s built-in sprite collision functions are used:

* **Bullet vs Asteroid**
  + both are removed
  + score increases
* **Asteroid vs Player**
  + asteroid disappears
  + the player loses a life
  + if lives reach zero → game over

## 6. Score & Lives Management

* Score increases whenever an asteroid is destroyed by a bullet.
* The number of lives is displayed on the screen.
* No high-score file is used (as per requirements).

## 7. Game Loop (Pygame Clock)

The main loop handles:

* reading key presses
* updating positions (player, bullets, asteroids)
* detecting collisions
* drawing all sprites
* showing text on the screen
* limiting the speed using clocktick.(fps)

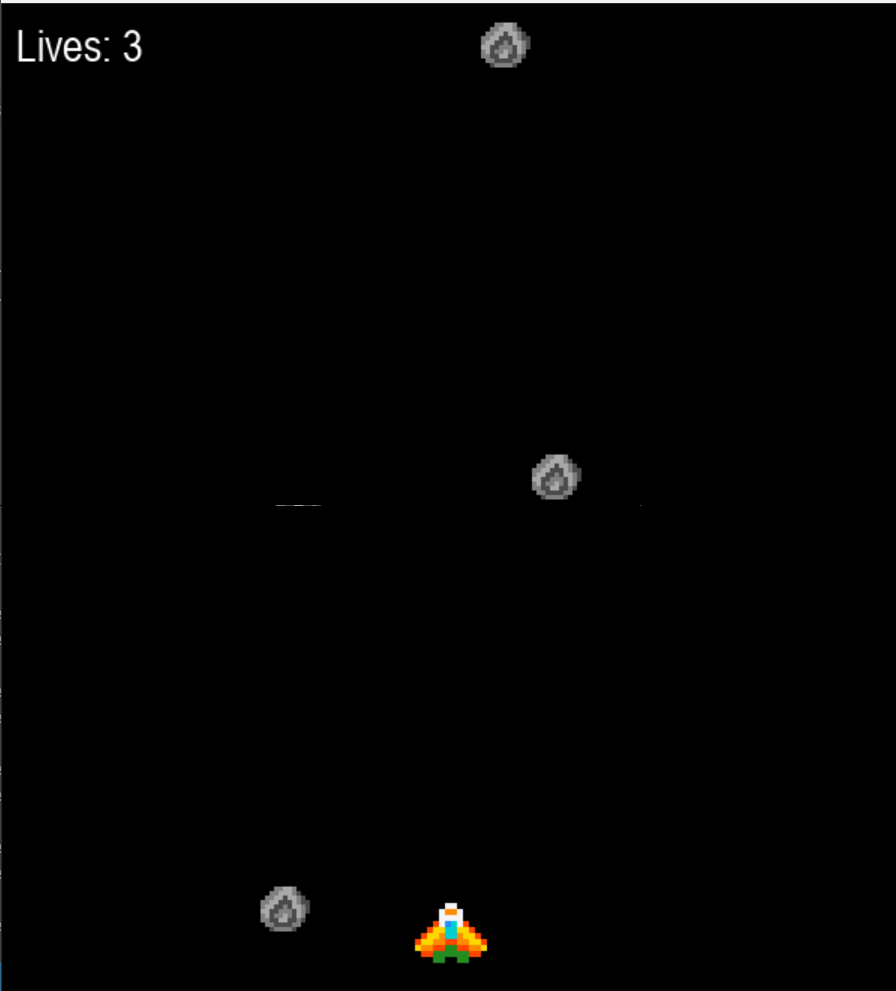
This ensures smooth real-time gameplay without lag.

## 8. Game over Handling

When lives become zero:

* A “GAME OVER” message appears.
* The screen pauses for a moment.
* The game closes automatically.

SAMPLE INPUT/OUTPUT





Challenges Faced

**Game Loop and Real-Time Updates**

* Managing continuous movement of bullets and asteroids without freezing the game.
* Ensuring smooth animations using Pygame.time.clock () and proper FPS control.

**Collision Detection**

* Detecting collisions between bullets and asteroids accurately.
* Handling collisions between player and asteroids as well as missed asteroids that pass the player.

**Pixel-Art Rendering**

Designing and drawing a pixel-art spaceship and asteroids using Pygame’s surface and draw.rect()

Ensuring the pixel art scales correctly and doesn’t break when moving across the screen.

**Game State Management**

Keeping track of lives, score, and game-over conditions.

Resetting or stopping the game cleanly when the player loses all lives..

Scope for Improvement

**Enhanced Enemy Behaviour:** Introduce multiple asteroid types with different sizes, speeds, and movement patterns.

**Power-Ups:** Add collectible items like shields, double bullets, or speed boosts.

**Scoring & Progression:** Include levels, increasing difficulty, and persistent high scores.

**Sound and Music:** Add background music, shooting sounds, and explosion effects.

**Full screen & Resolution Support:** Make the game compatible with fullscreen and different screen sizes.

**Visual Effects:** Add particle effects for explosions and laser shots to improve visual appeal.

Conclusion

This project helped in understanding essential game development concepts such as:

Pixel-art rendering and sprite management

Real-time collision detection

Keyboard input handling

Game state management and UI display

The game provides an engaging and simple arcade experience while serving as a foundation for more complex game development projects.