Space Invaders Project

A simple arcade-style shooter built with Python/Pygame

Team Members: Madhusuthan , Naonith K Rao , Abhinav MB , Likhith B

Batch No: G section

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.
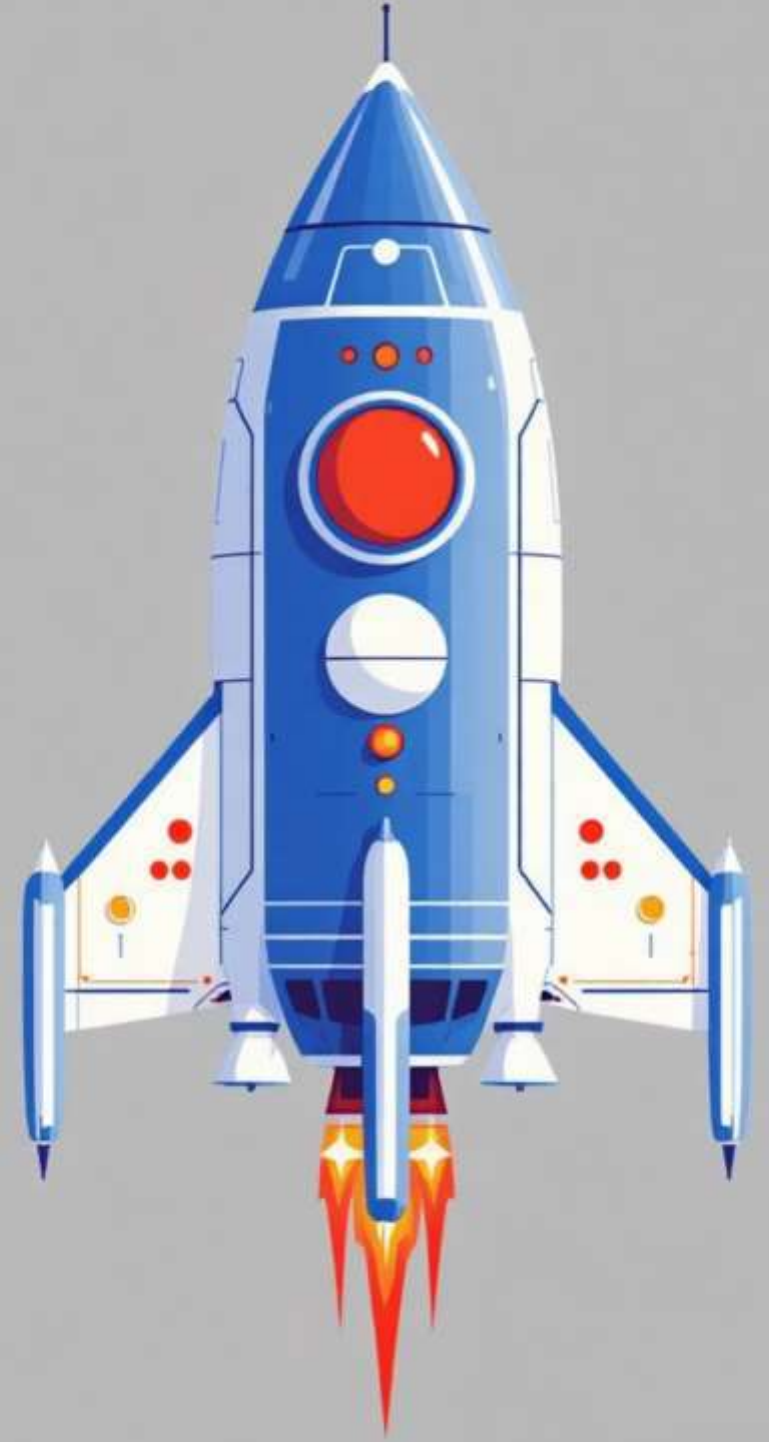
(2)

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.
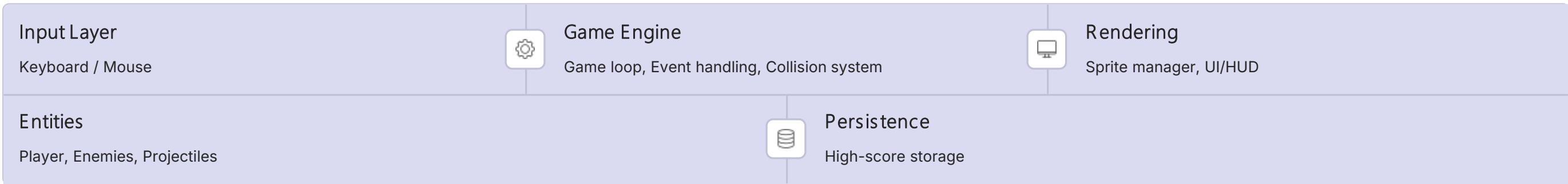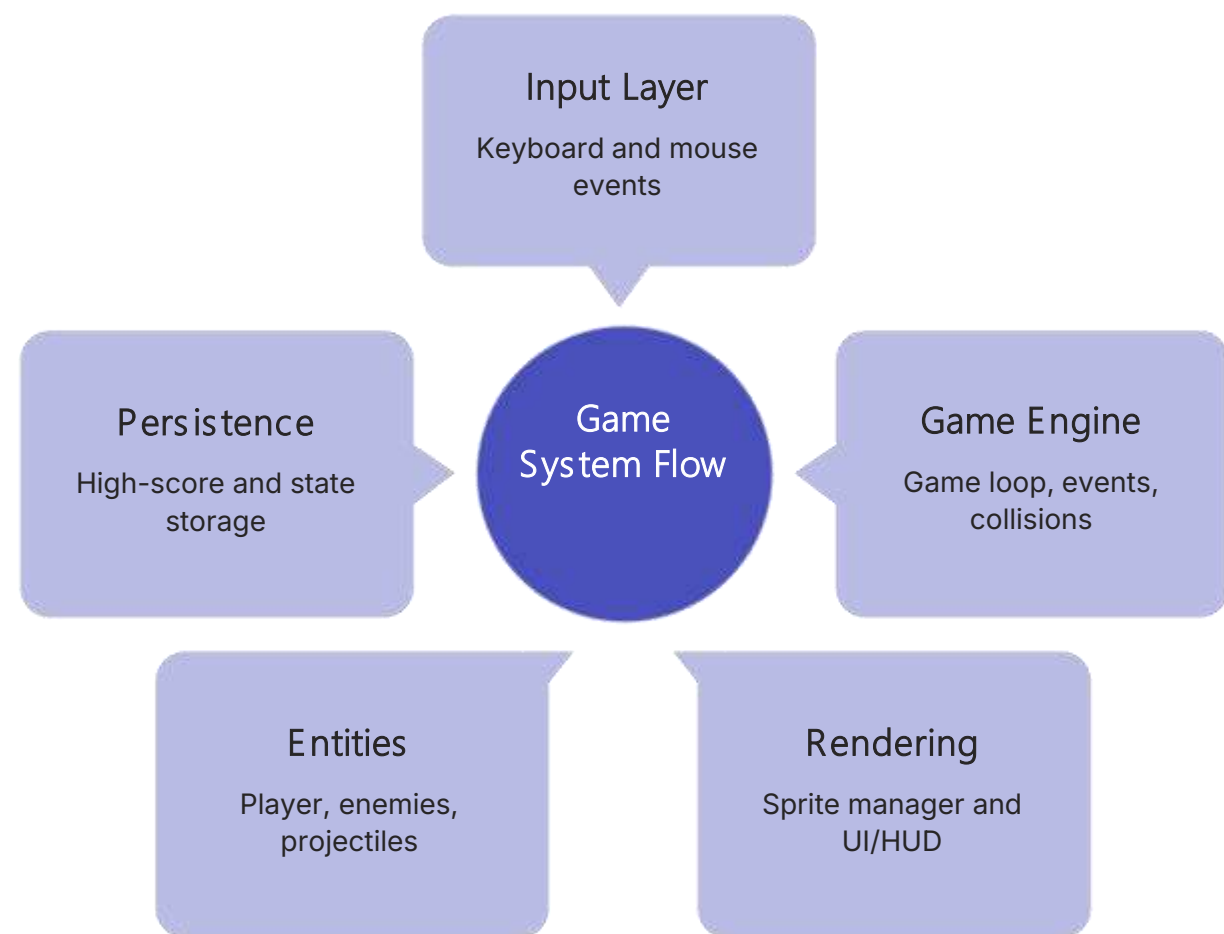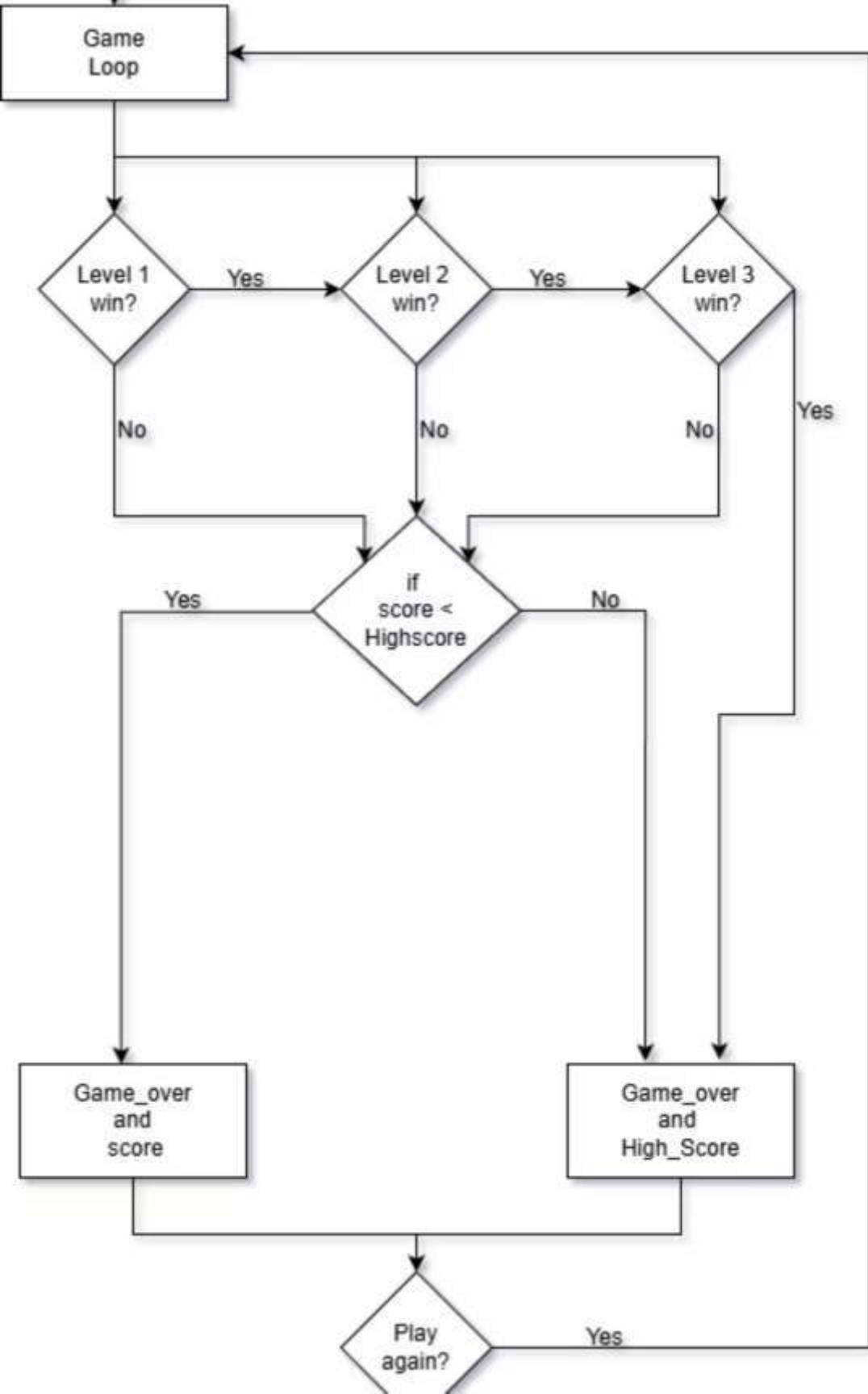
(3)

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.

System / Block Diagram

Input Layer

Keyboard and mouse events

Persistence

High-score and state storage

Game System Flow

Game Engine

Game loop, events, collisions

Entities

Player, enemies, projectiles

Rendering

Sprite manager and UI/HUD

Input Layer	Game Engine	Rendering
Keyboard / Mouse	Game loop, Event handling, Collision system	Sprite manager, UI/HUD

Entities	Persistence	
Player, Enemies, Projectiles	High-score storage	

This block diagram explains the flow of our Space Invaders game:

- 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**.

Contribution of Team Members

Abhinav MB	PES2UG25AM145	main loop, settings area and assets, integrating all the modules
Likith B	PES2UG25AM143	player logic, collision mechanics and scores function
Naonith K Rao	PES2UG25CS324	Level design, enemy AI behaviours, testing, `enemies.py`
Madhusuthan V	PES2UG25AM148	UI/HUD, implementation and code testing

Our collaborative approach, utilising Git for version control and regular pull requests, ensured seamless integration of individual contributions and maintained code quality throughout the development cycle.

Abhinav MB — Game Engine & Logic

- **Feature Implementation:** Developed the core game engine and main loop, ensuring smooth state transitions and event handling. Primary contribution to `main.py` and `game_engine.py`.

- **Collision Logic:** Implemented precise collision detection for player projectiles, enemy ships, and boundaries, crucial for accurate gameplay mechanics.
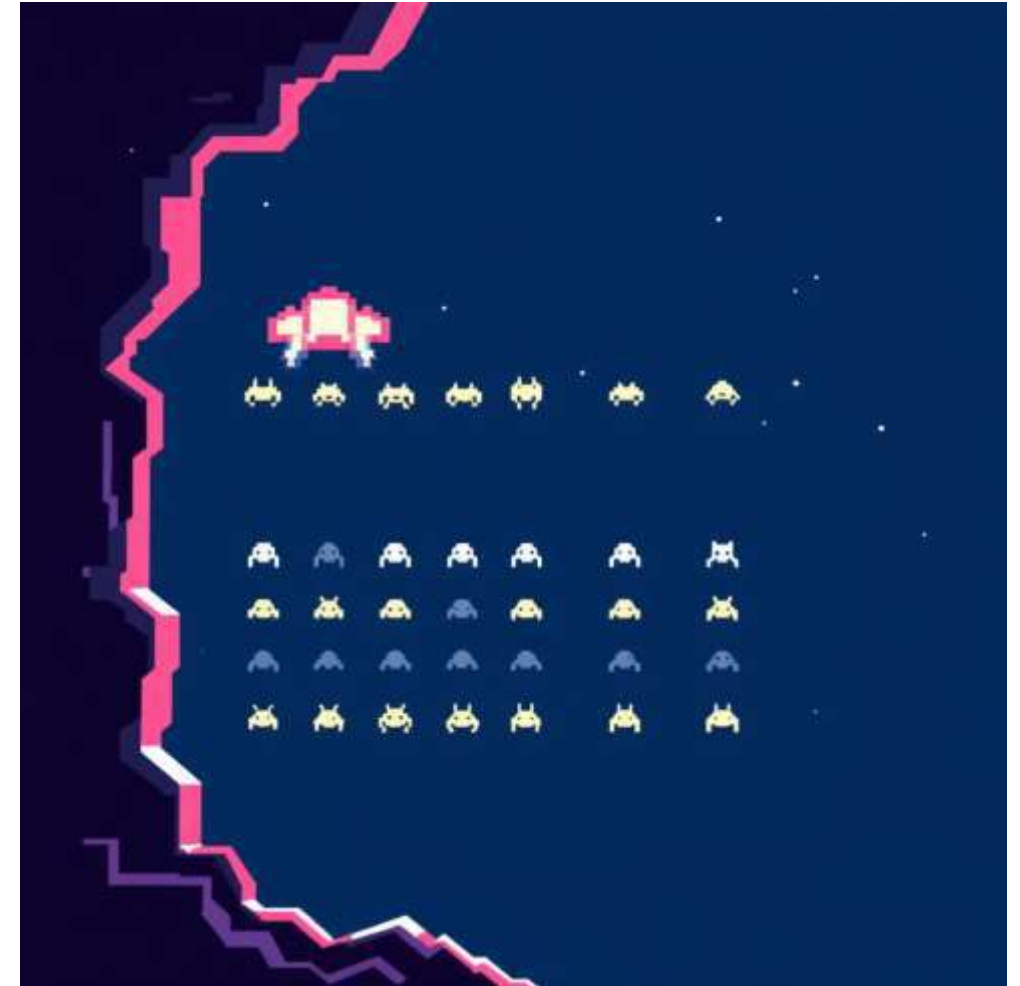

```
for bullet in active_bullets: if bullet.collides_with(enemy_ship): enemy_ship.take_damage()
```

main.py > ...

```python
import pygame
pygame.init()
from pygame import mixer
mixer.init()
#import scores
import random
import player
import bullet
import enemy
import collision
import ui


# creating objects
player = player.player()
asteroid = enemy.asteroid()
enemyShip = enemy.enemyShip()
bullet = bullet.bullet()
collision = collision.collision()

# display
font = pygame.font.Font(None, 32)
background = pygame.image.load("assets/bg3.png")
pygame.display.set_caption("Space Invaders")
icon = pygame.image.load("assets/ufo.png")
bullet_sound = mixer.Sound("assets/laser.mp3")
screen = pygame.display.set_mode((800, 600))

running = True


def show_score():
    score_text = font.render(f"Score: {scores}", True, (255, 255, 255))  # white color
    screen.blit(score_text, (10, 10))  # top-left corner
scores=0
level = 1
```

```python
39   # main loop
40   while running:
41       screen.blit(background, (0, 0))
42
43       for event in pygame.event.get():
44           if event.type == pygame.QUIT:
45               running = False
46
47           player.handle_input(event)
48           bullet.handle_input(event,player.playerX)
49
50       # Update
51       player.update()
52       bullet.update()
53
54       if level == 1: # checking currently if level 1
55
56           level_triggered = False
57           for i in range(asteroid.no_of_enemies):
58               # bullet hits asteroid
59               if collision.is_collision(asteroid.asteroidX[i],asteroid.asteroidY[i],bullet.bulletX,bullet.bulletY):
60                   bullet.bullet_state = "ready"
61                   #print(45)
62                   scores += 25
63
64                   asteroid.asteroidX[i] = random.randint(10, 736)
65                   asteroid.asteroidY[i] = random.randint(-150, -50)
66
67               if collision.is_collision(
68                       asteroid.asteroidX[i], asteroid.asteroidY[i],
69                       player.playerX, player.playerY):
70
71                   play_again = ui.game_over_screen(screen, scores)
72
73                   if play_again:
74                       # RESET GAME
75                       scores = 0
76                       level = 1
77                       level_triggered = False
```

```python
                    scores = 0
                    level = 1
                    level_triggered = False

                    player = player.__class__()
                    asteroid = enemy.asteroid()
                    bullet = bullet.__class__()

                    continue
                else:
                    running = False

        # level transition
        if scores >= 500 and not level_triggered:
            ui.show_level_transition(screen, 2)
            level = 2
            level_triggered = True

        asteroid.update()
        asteroid.draw(screen)

    if level == 2: # checking currently if level 2

        # player bullet and enemy collision
        for i in range(enemyShip.no_of_enemies):
            # bullet hits enemy ship
            if collision.is_collision(enemyShip.enemyShipX[i],enemyShip.enemyShipY[i],bullet.bulletX,bullet.bulletY):
                bullet.bullet_state = "ready"
                #print(45)
                scores += 25

                enemyShip.enemyShipX[i] = random.randint(10, 736)
                enemyShip.enemyShipY[i] = random.randint(-150, -100)
                enemyShip.bullet_state[i] = "ready"
                enemyShip.bulletX[i] = enemyShip.enemyShipX[i]
                enemyShip.bulletY[i] = enemyShip.enemyShipY[i]
```

```python
            # if enemy or its bullet collide with player
            if collision.is_collision(
                    enemyShip.enemyShipX[i], enemyShip.enemyShipY[i],
                    player.playerX, player.playerY) or collision.is_collision(enemyShip.bulletX[i],enemyShip.bulletY[i],player.playerX, player.playerY):

                play_again = ui.game_over_screen(screen, scores)

                if play_again:
                    # RESET GAME
                    scores = 0
                    level = 1
                    level_triggered = False

                    player = player.__class__()
                    asteroid = enemy.asteroid()
                    enemyShip = enemy.enemyShip()
                    bullet = bullet.__class__()

                    continue
                else:
                    running = False

        enemyShip.update()
        enemyShip.draw(screen)


    # draw elements
    bullet.draw(screen)
    player.draw(screen)

    show_score()

    pygame.display.update()
```

Madhusuthan V—Sprites & UI/HUD

- **Visual Assets:** Managed and optimised all game sprites, ensuring visual consistency and efficient loading. Files primarily in the `assets/sprites/` directory.

- **Rendering Pipeline:** Constructed the game's rendering system, responsible for drawing all game elements to the screen at optimal frame rates.

- **User Interface (UI) & HUD:** Designed and implemented the in-game heads-up display (HUD), showcasing player score, lives remaining, and other crucial game information, located in `ui_manager.py`.

- **Snippet:** Render logic for a score update: screen.blit(font.render(f"Score: {score}", True, WHITE), (10, 10))

```python
# ui.py
import pygame
import os
from pygame import mixer

pygame.init()
mixer.init()

FONT_LARGE = pygame.font.Font(None, 72)
FONT_MED = pygame.font.Font(None, 40)
FONT_SMALL = pygame.font.Font(None, 30)

HIGHSCORE_FILE = "highscore.txt"


# -------------------------------------------
# LOAD HIGH SCORE
# -------------------------------------------
def read_high_score():
    if not os.path.exists(HIGHSCORE_FILE):
        return 0
    try:
        with open(HIGHSCORE_FILE, "r") as f:
            return int(f.read().strip())
    except:
        return 0


# -------------------------------------------
# SAVE HIGH SCORE
# -------------------------------------------
def write_high_score(score):
    try:
        with open(HIGHSCORE_FILE, "w") as f:
            f.write(str(score))
    except:
        pass

```

```python
39
40    # -----------------------------------------------------
41    # LEVEL TRANSITION SCREEN   (Score ≥ 500)
42    # -----------------------------------------------------
43    def show_level_transition(screen, level, duration_ms=1800):
44        text = FONT_LARGE.render(f"LEVEL {level}", True, (255, 255, 255))
45
46        start = pygame.time.get_ticks()
47        while pygame.time.get_ticks() - start < duration_ms:
48            screen.fill((0, 0, 0))
49            screen.blit(text, (300, 260))
50            pygame.display.update()
51
52
53    # -----------------------------------------------------
54    # GAME OVER SCREEN
55    # -----------------------------------------------------
56    def game_over_screen(screen, score):
57        high = read_high_score()
58
59        # update high score
60        if score > high:
61            write_high_score(score)
62            high = score
63
64        # UI text
65        over_text = FONT_LARGE.render("GAME OVER", True, (255, 255, 255))
66        score_text = FONT_MED.render(f"Your Score: {score}", True, (255, 255, 255))
67        high_text = FONT_MED.render(f"High Score: {high}", True, (255, 255, 255))
68        retry_text = FONT_SMALL.render("Press Y to Play Again", True, (255, 255, 255))
69        quit_text = FONT_SMALL.render("Press N to Exit", True, (255, 255, 255))
70
71        waiting = True
72        while waiting:
73            screen.fill((0, 0, 0))
74            screen.blit(over_text, (240, 160))
75            screen.blit(score_text, (270, 260))
76            screen.blit(high_text, (270, 310))
77            screen.blit(retry_text, (260, 380))
```

Made with GAMMA

```python
70
71        waiting = True
72    ∨  while waiting:
73            screen.fill((0, 0, 0))
74            screen.blit(over_text, (240, 160))
75            screen.blit(score_text, (270, 260))
76            screen.blit(high_text, (270, 310))
77            screen.blit(retry_text, (260, 380))
78            screen.blit(quit_text, (300, 420))
79
80            pygame.display.update()
81
82    ∨      for event in pygame.event.get():
83    ∨          if event.type == pygame.QUIT:
84                    return False
85    ∨          if event.type == pygame.KEYDOWN:
86    ∨              if event.key == pygame.K_y:
87                        return True
88    ∨              if event.key == pygame.K_n:
89                        return False
90
```

Likhit B - Collision , player , bullet mechanics

- **Player Logic (`player.py`):**
  - Implemented player movement and input handling, ensuring responsive controls.
- **Collision Mechanics (`collision.py`):**
  - Implemented boundary collision detection to keep game elements within screen limits.
- **Bullet(`Bullet.py`):**
  - Handled player bullet mechanics

player.py > ...

```python
import pygame

class player:

    def __init__(self):
        self.playerImg = None
        self.playerX = 370
        self.playerY = 480
        self.playerX_change = 0
        self.playerImg = pygame.image.load("assets/player.png")

    def handle_input(self,event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                self.playerX_change = -0.2
            if event.key == pygame.K_RIGHT:
                self.playerX_change = 0.2
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                self.playerX_change = 0

    def update(self):
        self.playerX += self.playerX_change
        if self.playerX < 0:
            self.playerX = 0
        if self.playerX > 736:
            self.playerX = 736

    def draw(self,screen):
        screen.blit(self.playerImg, (self.playerX, self.playerY))
```

```python
import player
import pygame
from pygame import mixer
mixer.init()
bullet_sound = mixer.Sound("assets/laser.mp3")


class bullet:

    player = player.player()

    def __init__(self):

        self.bulletX = self.player.playerX
        self.bulletY = 480
        self.bulletY_change = 0.2
        self.bullet_state = "ready"
        self.bulletImg = pygame.image.load("assets/bullet.png")

    def fire_bullet(self,playerx):
        self.bullet_state = "fire"
        self.bulletY = 480
        self.bulletX = playerx + 17

    def handle_input(self,event,playerx):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE and self.bullet_state == "ready":
                self.bulletX = playerx + 17
                self.bulletY = 480
                self.fire_bullet(playerx)
                bullet_sound.play()

    def update(self):
        if self.bullet_state == "fire":
            self.bulletY -= self.bulletY_change
            if self.bulletY < 0:
                self.bullet_state = "ready"

    def draw(self,screen):
        if self.bullet_state == "fire":
            screen.blit(self.bulletImg, (self.bulletX, self.bulletY))
```

```
main.py    ●    ui.py    player.py    enemy.py    collision.py ✕    bullet.py

collision.py > collision > is_collision
1    import math
2
3
4    class collision:
5
6        def __init__(self):
7            pass
8
9        def is_collision(self,enemyX,enemyY,bulletX,bulletY):
10           distance = math.sqrt((enemyX - bulletX)**2 + (enemyY - bulletY)**2)
11           return distance < 35
12
13
```

Naonith K Rao — Level Design & AI

- **Level Design:** Created initial game levels, defining enemy formations, spawn patterns, and wave progression, controlled via `levels/level1.json`.

- **Enemy AI:** Developed the AI behaviours for various enemy types, including movement patterns, attack strategies, and evasive manoeuvres.

- **Challenge:** Balancing enemy difficulty and player engagement without overwhelming the player, resolved through iterative testing and feedback loops.

A snippet from `enemies.py` for basic enemy movement: self.x += self.speed * self.direction

```python
1   import pygame
2   import bullet
3   import random
4
5   class asteroid:
6
7       def __init__(self):
8
9           self.asteroidImg =[]
10          self.asteroidX = []
11          self.asteroidY = []
12          self.no_of_enemies=4
13          self.asteroidY_change = 0.05
14
15          for i in range(self.no_of_enemies):
16              self.asteroidImg.append(pygame.image.load("assets/asteroid.png"))
17              self.asteroidX.append(random.randint(10, 736))
18              self.asteroidY.append(random.randint(-150, -50))
19
20      def update(self):
21
22          for i in range(self.no_of_enemies):
23              self.asteroidY[i] += self.asteroidY_change
24
25              # asteroid repositioning
26              if self.asteroidY[i] > 600:
27                  self.asteroidX[i] = random.randint(10, 736)
28                  self.asteroidY[i] = random.randint(-150, -50)
29
30
31      def draw(self,screen):
32
33          for i in range(self.no_of_enemies):
34              screen.blit(self.asteroidImg[i], (self.asteroidX[i], self.asteroidY[i]))
35
```

Made with GAMMA

```python
36    class enemyShip:
37
38        def __init__(self):
39            self.enemyImg =[]
40            self.enemyShipX = []
41            self.enemyShipY = []
42            self.e_bulletImg = []
43            self.bullet_state = []
44            self.bulletX = []
45            self.bulletY = []
46            self.no_of_enemies=4
47            self.enemyShipY_change = 0.05
48            self.bulletY_change = 0.1
49
50            for i in range(self.no_of_enemies):
51                self.enemyImg.append(pygame.image.load("assets/rocket.png"))
52                self.enemyShipX.append(random.randint(10, 736))
53                self.enemyShipY.append(random.randint(-150, -100))
54
55                # Enemy bullet
56                self.bulletX.append(self.enemyShipX[i])
57                self.bulletY.append(self.enemyShipY[i])
58                self.bullet_state.append("ready")
59                self.e_bulletImg.append(pygame.image.load("assets/e_bullet.png"))
60
61        def update(self):
62
63            for i in range(self.no_of_enemies):
64                # horizontal movement
65                self.enemyShipY[i] += self.enemyShipY_change
66
67                # fire bullet downward
68                if self.bullet_state[i] == "ready":
69                    self.bullet_state[i] = "fire"
70                    self.bulletY[i] = self.enemyShipY[i]
71                    self.bulletX[i] = self.enemyShipX[i] + 15
72
73                if self.bullet_state[i] == "fire":
74                    self.bulletY[i] += self.bulletY_change
75                    if self.bulletY[i] > 600:
76                        self.bullet_state[i] = "ready"
77
78                if self.enemyShipY[i] > 600:
79                    self.enemyShipX[i] = random.randint(10, 736)
80                    self.enemyShipY[i] = random.randint(-150, -100)
81                    self.bullet_state[i] = "ready"
82                    self.bulletX[i] = self.enemyShipX[i]
83                    self.bulletY[i] = self.enemyShipY[i]
84
85
86        def draw(self,screen):
87            for i in range(self.no_of_enemies):
88                screen.blit(self.e_bulletImg[i], (self.bulletX[i], self.bulletY[i]))
89                screen.blit(self.enemyImg[i], (self.enemyShipX[i], self.enemyShipY[i]))
90
```

Challenges

Technical Challenges Faced

- Optimising enemy in level 2 rendering.

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

Potential Extensions

- Integrating **procedural level generation** for endless replayability.

Thank You

Space Invaders!!

Project Repository: https://github.com/Abhi752007/space-invaders-project.git