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

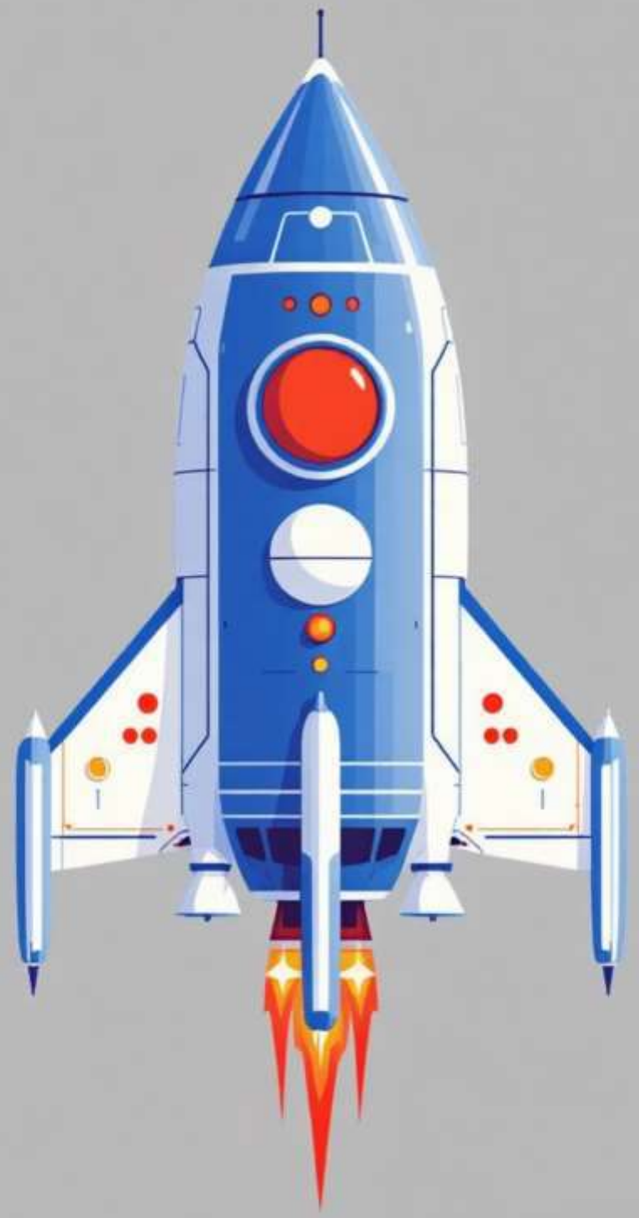
Fundamental Learning

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.

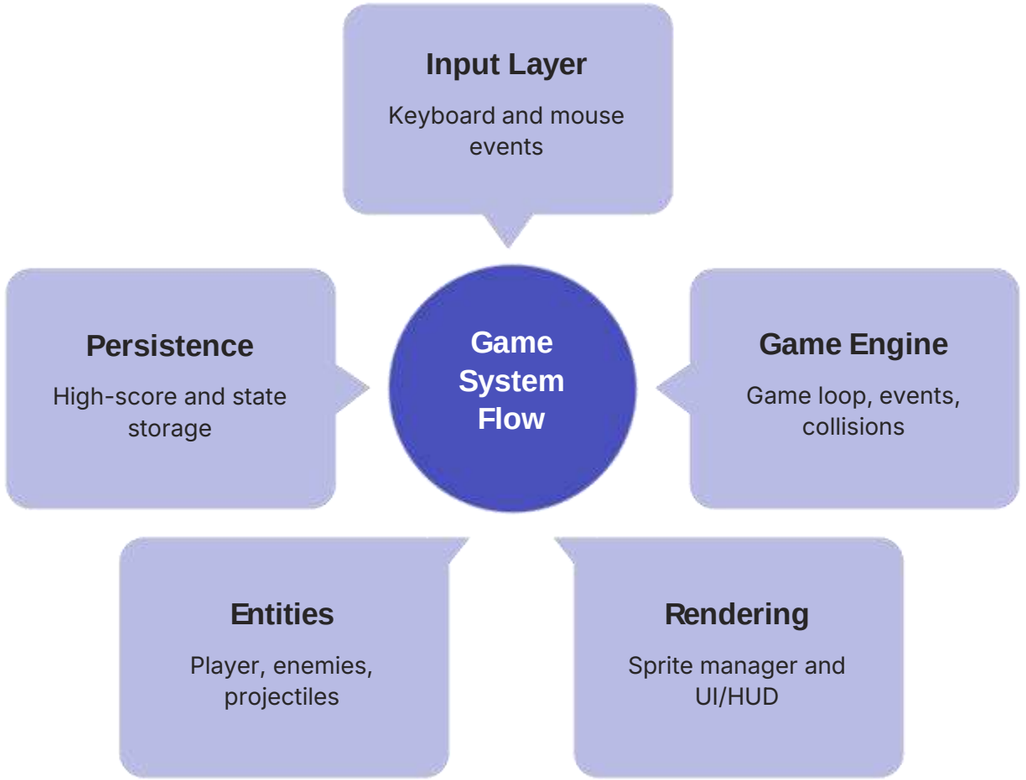
3

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 / Mouse

Game Engine

Game loop, Event handling, Collision system

Rendering

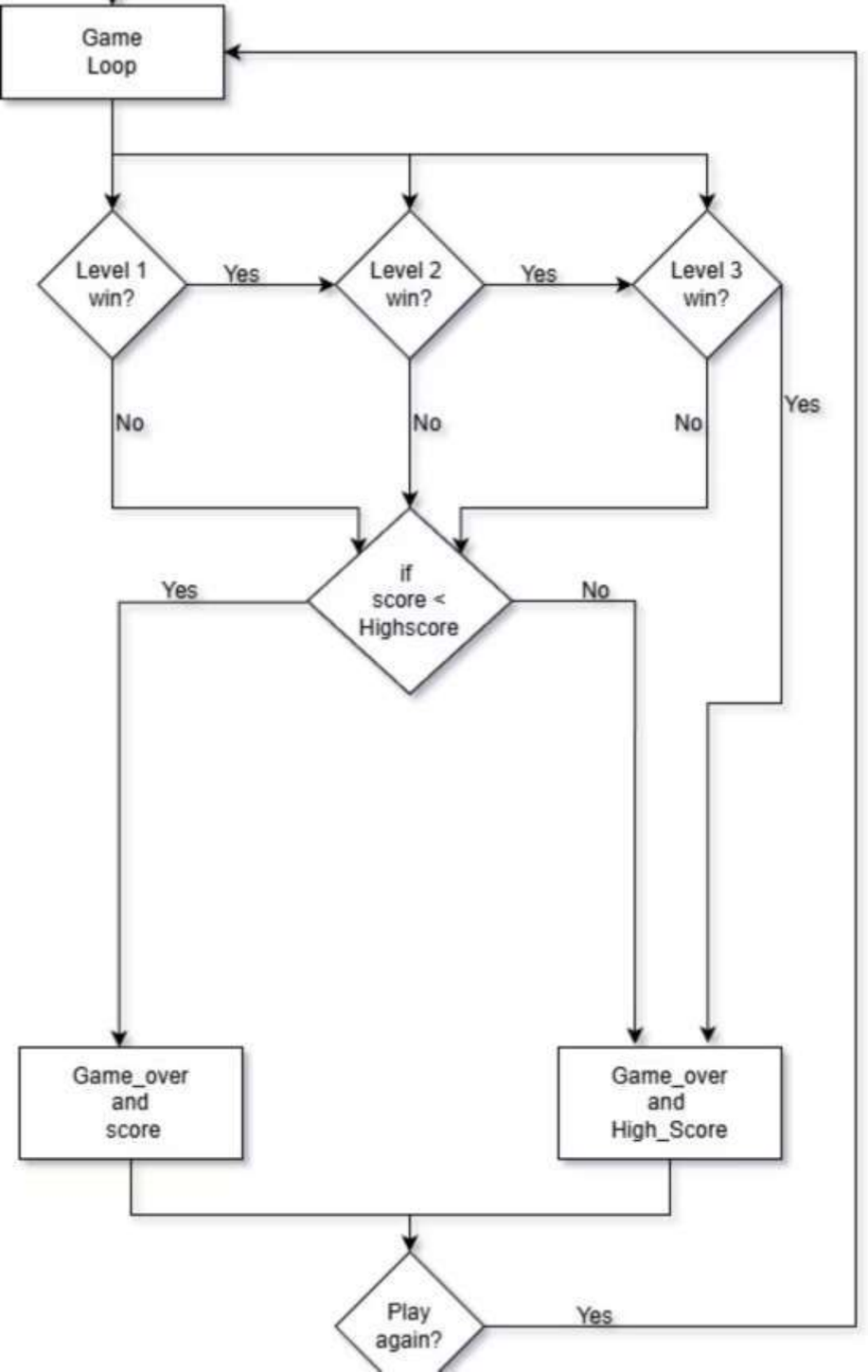
Sprite manager, UI/HUD

Entities

Player, Enemies, Projectiles

Persistence

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**.

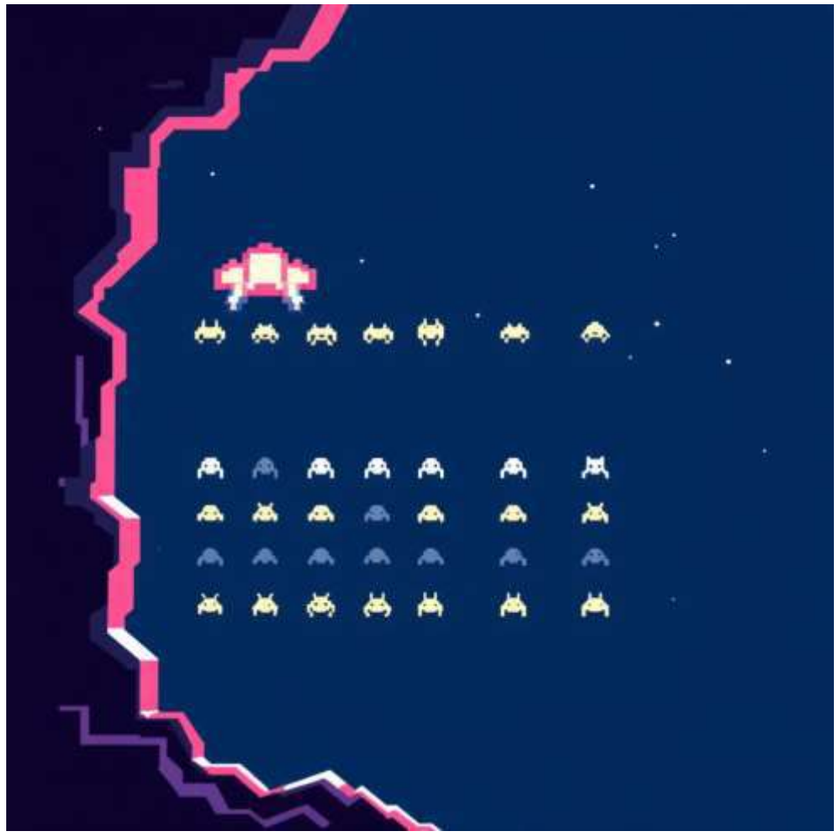
Contribution of Team Members

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 ● | player.py | enemy.py | collision.py | bullet.py

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

```

main.py ● | player.py | enemy.py | collision.py | bullet.py

main.py > ...

```python
# main loop
while running:
    screen.blit(background, (0, 0))

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        player.handle_input(event)
        bullet.handle_input(event,player.playerX)

    # Update
    player.update()
    bullet.update()

    if level == 1: # checking currently if level 1

        level_triggered = False
        for i in range(asteroid.no_of_enemies):
            # bullet hits asteroid
            if collision.is_collision(asteroid.asteroidX[i],asteroid.asteroidY[i],bullet.bulletX,bullet.bulletY):
                bullet.bullet_state = "ready"
                #print(45)
                scores += 25

                asteroid.asteroidX[i] = random.randint(10, 736)
                asteroid.asteroidY[i] = random.randint(-150, -50)

            if collision.is_collision(
                    asteroid.asteroidX[i], asteroid.asteroidY[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
```

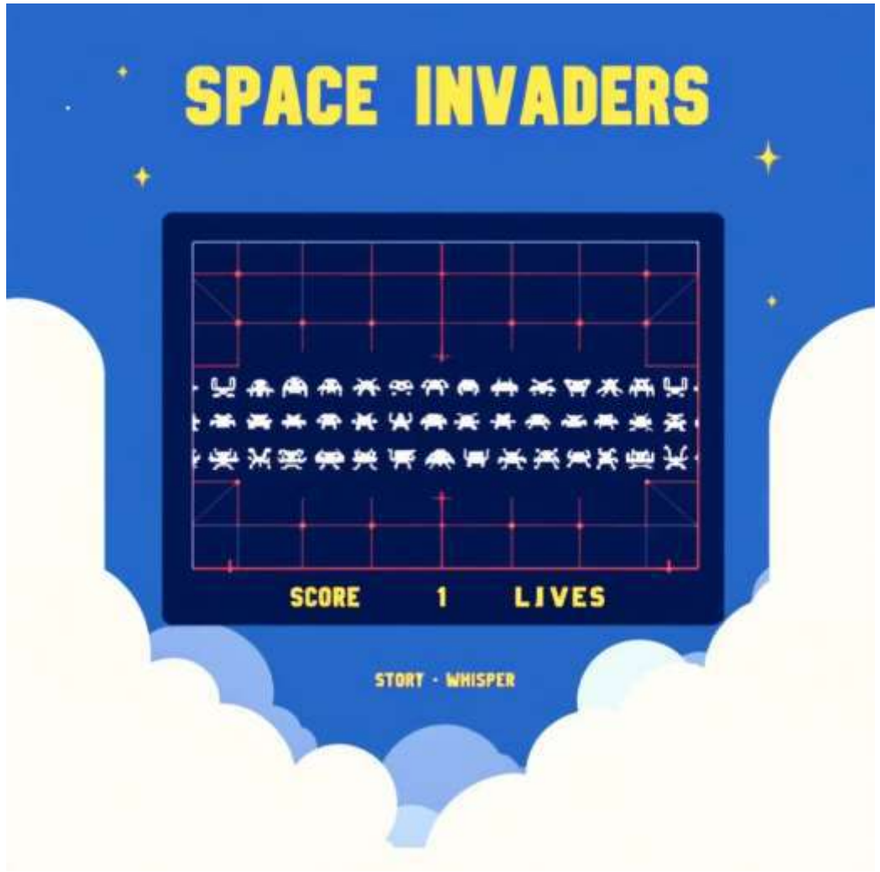
main.py ● | player.py | enemy.py | collision.py | bullet.py

main.py > ...

```
                    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]
```

```
111
112             # if enemy or its bullet collide with player
113             if collision.is_collision(
114                     enemyShip.enemyShipX[i], enemyShip.enemyShipY[i],
115                     player.playerX, player.playerY) or collision.is_collision(enemyShip.bulletX[i],enemyShip.bulletY[i],player.playerX, player.playerY):
116
117                 play_again = ui.game_over_screen(screen, scores)
118
119                 if play_again:
120                     # RESET GAME
121                     scores = 0
122                     level = 1
123                     level_triggered = False
124
125                     player = player.__class__()
126                     asteroid = enemy.asteroid()
127                     enemyShip = enemy.enemyShip()
128                     bullet = bullet.__class__()
129
130                     continue
131                 else:
132                     running = False
133
134         enemyShip.update()
135         enemyShip.draw(screen)
136
137
138     # draw elements
139     bullet.draw(screen)
140     player.draw(screen)
141
142     show_score()
143
144     pygame.display.update()
145
```

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))

main.py ● | *ui.py* × | player.py | enemy.py | collision.py | bullet.py

ui > ui.py > ...

```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

```

main.py ● | ui.py | player.py | enemy.py | collision.py | bullet.py

ui > ui.py > ...

```python

# ------------------------------------------------
# LEVEL TRANSITION SCREEN  (Score ≥ 500)
# ------------------------------------------------
def show_level_transition(screen, level, duration_ms=1800):
    text = FONT_LARGE.render(f"LEVEL {level}", True, (255, 255, 255))

    start = pygame.time.get_ticks()
    while pygame.time.get_ticks() - start < duration_ms:
        screen.fill((0, 0, 0))
        screen.blit(text, (300, 260))
        pygame.display.update()


# ------------------------------------------------
# GAME OVER SCREEN
# ------------------------------------------------
def game_over_screen(screen, score):
    high = read_high_score()

    # update high score
    if score > high:
        write_high_score(score)
        high = score

    # UI text
    over_text = FONT_LARGE.render("GAME OVER", True, (255, 255, 255))
    score_text = FONT_MED.render(f"Your Score: {score}", True, (255, 255, 255))
    high_text = FONT_MED.render(f"High Score: {high}", True, (255, 255, 255))
    retry_text = FONT_SMALL.render("Press Y to Play Again", True, (255, 255, 255))
    quit_text = FONT_SMALL.render("Press N to Exit", True, (255, 255, 255))

    waiting = True
    while waiting:
        screen.fill((0, 0, 0))
        screen.blit(over_text, (240, 160))
        screen.blit(score_text, (270, 260))
        screen.blit(high_text, (270, 310))
        screen.blit(retry_text, (260, 380))
```

```
    waiting = True
    while waiting:
        screen.fill((0, 0, 0))
        screen.blit(over_text, (240, 160))
        screen.blit(score_text, (270, 260))
        screen.blit(high_text, (270, 310))
        screen.blit(retry_text, (260, 380))
        screen.blit(quit_text, (300, 420))

        pygame.display.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_y:
                    return True
                if event.key == pygame.K_n:
                    return False

```

Likhit B - Collision , player , sprites

- **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

main.py ● | ui.py | player.py × | enemy.py | collision.py | bullet.py

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))

```

main.py ● | ui.py | player.py | enemy.py | collision.py | bullet.py ×

bullet.py > bullet > draw

```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))

```

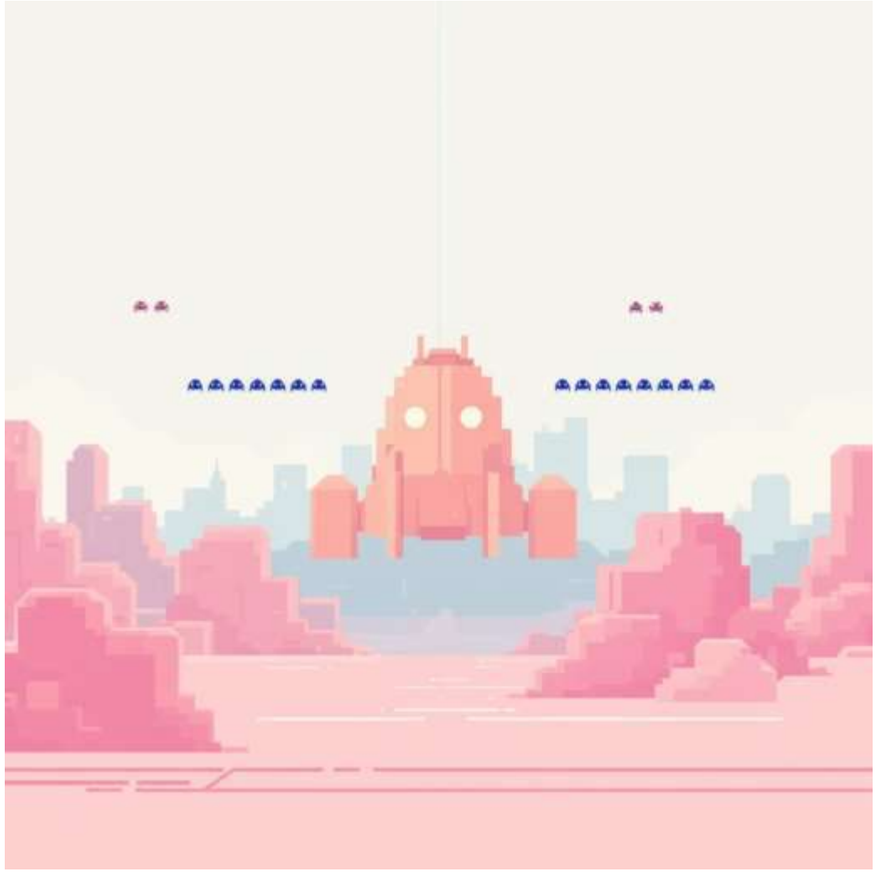
main.py ● | ui.py | player.py | enemy.py | collision.py × | bullet.py

collision.py > collision > is_collision

```python
import math


class collision:

    def __init__(self):
        pass

    def is_collision(self,enemyX,enemyY,bulletX,bulletY):
        distance = math.sqrt((enemyX - bulletX)**2 + (enemyY - bulletY)**2)
        return distance < 35

```

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

main.py ● | ui.py | player.py | enemy.py × | collision.py | bullet.py

enemy.py > enemyShip > update

```python
import pygame
import bullet
import random

class asteroid:

    def __init__(self):

        self.asteroidImg =[]
        self.asteroidX = []
        self.asteroidY = []
        self.no_of_enemies=4
        self.asteroidY_change = 0.05

        for i in range(self.no_of_enemies):
            self.asteroidImg.append(pygame.image.load("assets/asteroid.png"))
            self.asteroidX.append(random.randint(10, 736))
            self.asteroidY.append(random.randint(-150, -50))

    def update(self):

        for i in range(self.no_of_enemies):
            self.asteroidY[i] += self.asteroidY_change

            # asteroid repositioning
            if self.asteroidY[i] > 600:
                self.asteroidX[i] = random.randint(10, 736)
                self.asteroidY[i] = random.randint(-150, -50)


    def draw(self,screen):

        for i in range(self.no_of_enemies):
            screen.blit(self.asteroidImg[i], (self.asteroidX[i], self.asteroidY[i]))

```

main.py ● | ui.py | player.py | enemy.py ● | collision.py | bullet.py

enemy.py > enemyShip

```python
class enemyShip:
    def __init__(self):
        self.enemyImg =[]
        self.enemyShipX = []
        self.enemyShipY = []
        self.e_bulletImg = []
        self.bullet_state = []
        self.bulletX = []
        self.bulletY = []
        self.no_of_enemies=4
        self.enemyShipY_change = 0.05
        self.bulletY_change = 0.1

        for i in range(self.no_of_enemies):
            self.enemyImg.append(pygame.image.load("assets/rocket.png"))
            self.enemyShipX.append(random.randint(10, 736))
            self.enemyShipY.append(random.randint(-150, -100))

            # Enemy bullet
            self.bulletX.append(self.enemyShipX[i])
            self.bulletY.append(self.enemyShipY[i])
            self.bullet_state.append("ready")
            self.e_bulletImg.append(pygame.image.load("assets/e_bullet.png"))

    def update(self):

        for i in range(self.no_of_enemies):
            # horizontal movement
            self.enemyShipY[i] += self.enemyShipY_change

            # fire bullet downward
            if self.bullet_state[i] == "ready":
                self.bullet_state[i] = "fire"
                self.bulletY[i] = self.enemyShipY[i]
                self.bulletX[i] = self.enemyShipX[i] + 15

            if self.bullet_state[i] == "fire":
                self.bulletY[i] += self.bulletY_change
                if self.bulletY[i] > 600:
                    self.bullet_state[i] = "ready"

            if self.enemyShipY[i] > 600:
                self.enemyShipX[i] = random.randint(10, 736)
                self.enemyShipY[i] = random.randint(-150, -100)
                self.bullet_state[i] = "ready"
                self.bulletX[i] = self.enemyShipX[i]
                self.bulletY[i] = self.enemyShipY[i]


    def draw(self,screen):
        for i in range(self.no_of_enemies):
            screen.blit(self.e_bulletImg[i], (self.bulletX[i], self.bulletY[i]))
            screen.blit(self.enemyImg[i], (self.enemyShipX[i], self.enemyShipY[i]))
```

Challenges & Future Work

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