XIAMEN UNIVERSITY MALAYSIA

Course Code	:	CST210
Course Name	:	Object-Oriented Programming-Java
Lecturer	:	NaharLutfun
Academic Session	:	09/2025
Assessment Title	:	Assignment
Submission Due Date	:	29/12/2025

Prepared by :

Student ID	Student Name
CYS2409008	Cui Zechong
CYS2409026	Lyu Jielong
CYS2409046	Zhang Zehou
CYS2409048	Zhao Yuhan

Date Received :

Feedback from Lecturer:	
	Mark:

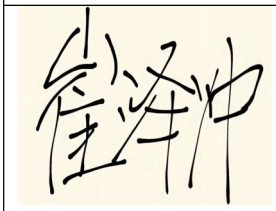
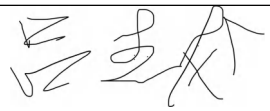
Own Work Declaration

I/We hereby understand my/our work would be checked for plagiarism or other misconduct, and the softcopy would be saved for future comparison(s).

I/We hereby confirm that all the references or sources of citations have been correctly listed or presented and I/we clearly understand the serious consequence caused by any intentional or unintentional misconduct.

This work is not made on any work of other students (past or present), and it has not been submitted to any other courses or institutions before.

Signature:

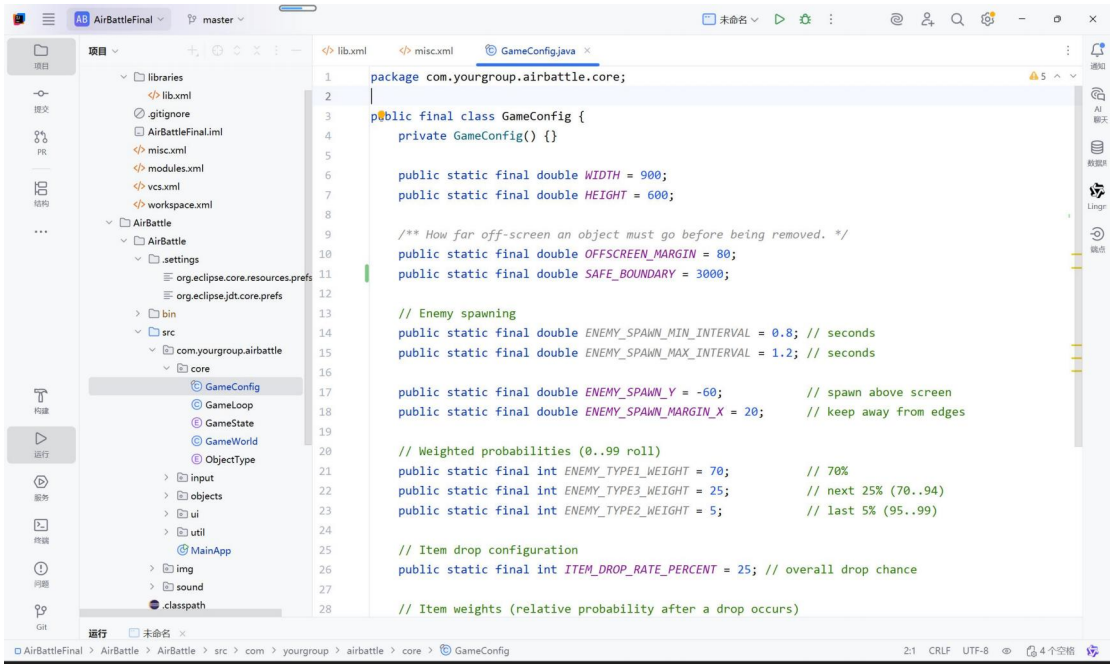
CYS2409008	CYS2409026	CYS2409046	CYS2409048
			

Date:29/12/2025

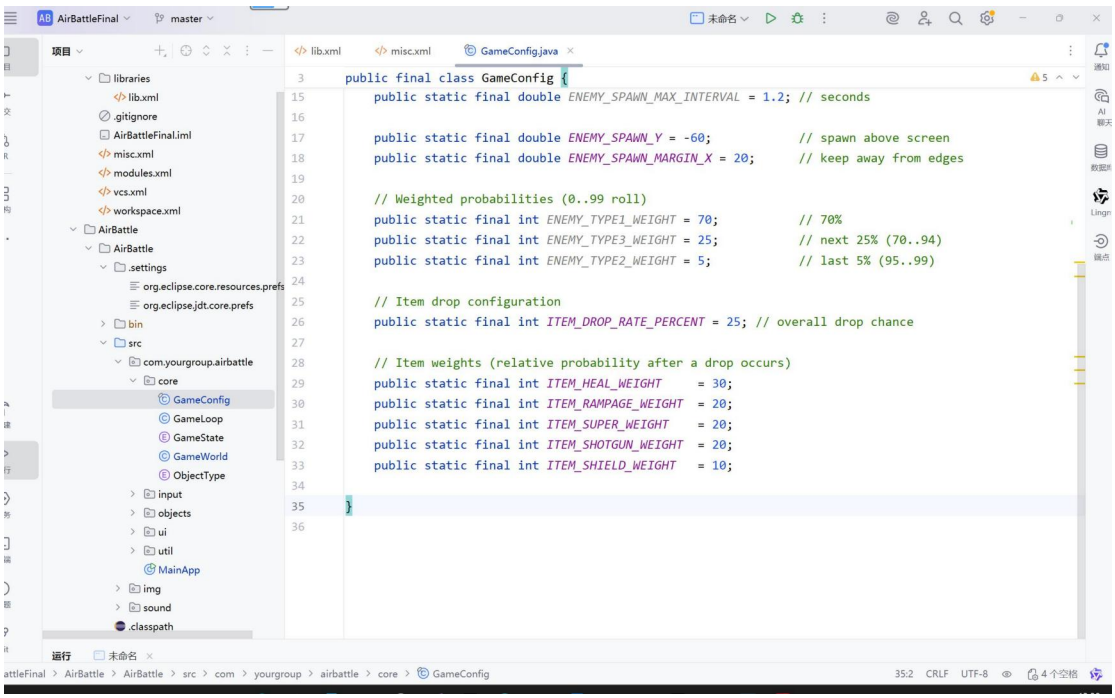
Content Page

AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\core

\GameConfig.java

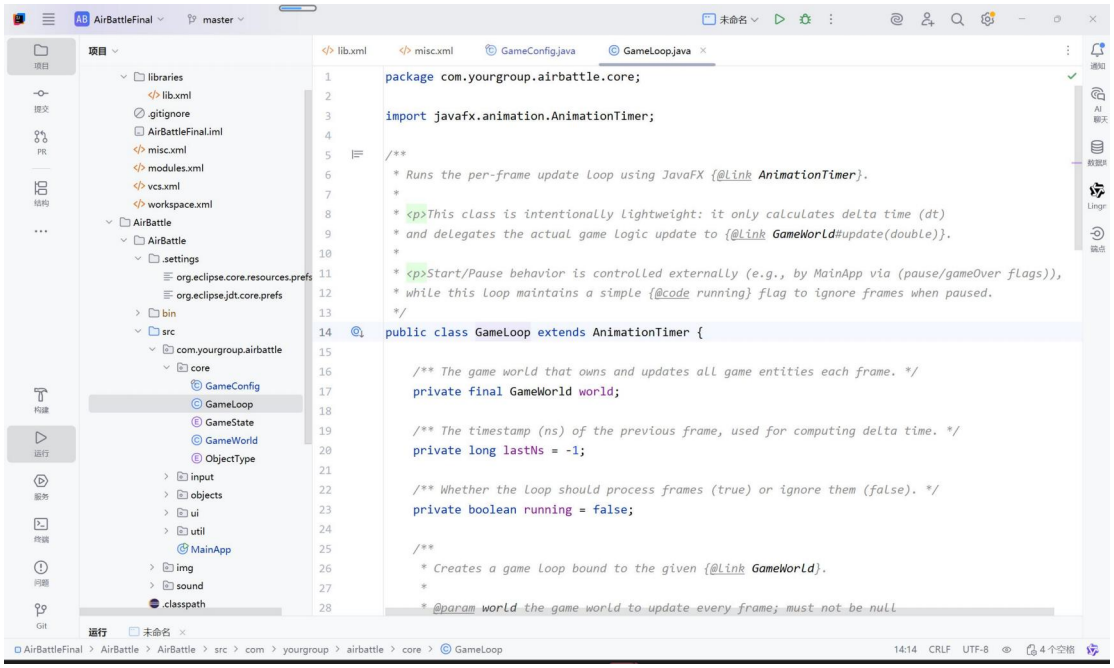


```java
package com.yourgroup.airbattle.core;

public final class GameConfig {
    private GameConfig() {}

    public static final double WIDTH = 900;
    public static final double HEIGHT = 600;

    /** How far off-screen an object must go before being removed. */
    public static final double OFFSCREEN_MARGIN = 80;
    public static final double SAFE_BOUNDARY = 3000;

    // Enemy spawning
    public static final double ENEMY_SPAWN_MIN_INTERVAL = 0.8; // seconds
    public static final double ENEMY_SPAWN_MAX_INTERVAL = 1.2; // seconds

    public static final double ENEMY_SPAWN_Y = -60;           // spawn above screen
    public static final double ENEMY_SPAWN_MARGIN_X = 20;     // keep away from edges

    // Weighted probabilities (0..99 roll)
    public static final int ENEMY_TYPE1_WEIGHT = 70;          // 70%
    public static final int ENEMY_TYPE3_WEIGHT = 25;          // next 25% (70..94)
    public static final int ENEMY_TYPE2_WEIGHT = 5;           // last 5% (95..99)

    // Item drop configuration
    public static final int ITEM_DROP_RATE_PERCENT = 25; // overall drop chance

    // Item weights (relative probability after a drop occurs)
```

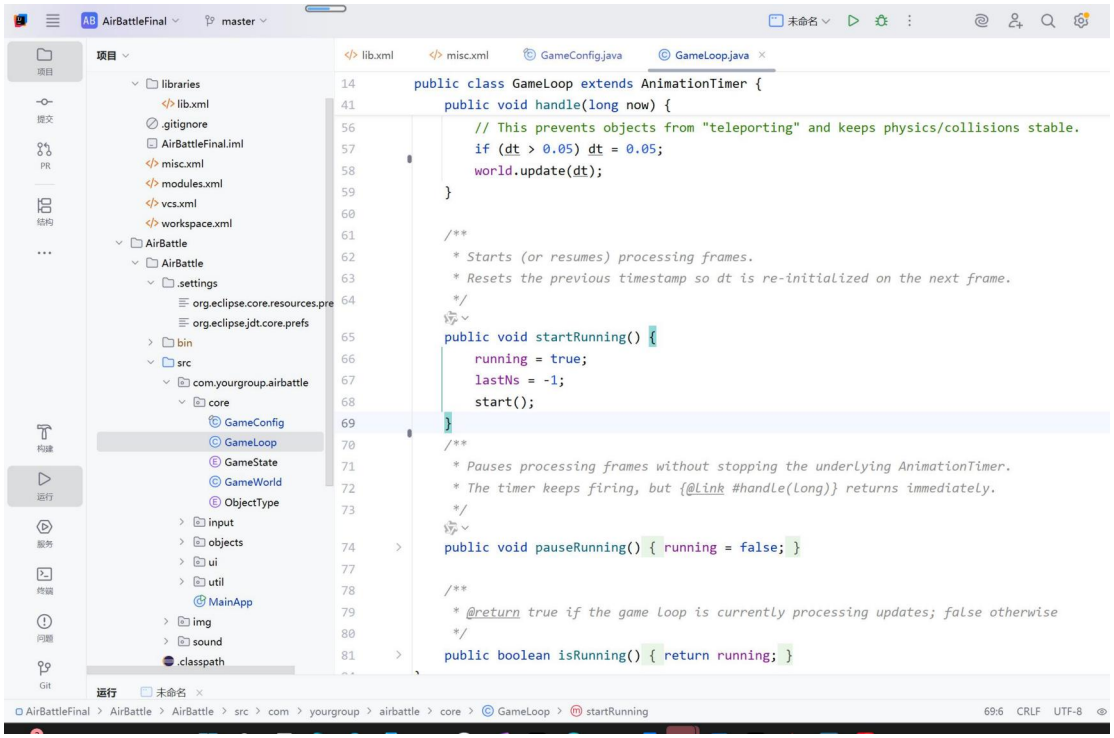


```java
public final class GameConfig {
    public static final double ENEMY_SPAWN_MAX_INTERVAL = 1.2; // seconds

    public static final double ENEMY_SPAWN_Y = -60;           // spawn above screen
    public static final double ENEMY_SPAWN_MARGIN_X = 20;     // keep away from edges

    // Weighted probabilities (0..99 roll)
    public static final int ENEMY_TYPE1_WEIGHT = 70;          // 70%
    public static final int ENEMY_TYPE3_WEIGHT = 25;          // next 25% (70..94)
    public static final int ENEMY_TYPE2_WEIGHT = 5;           // last 5% (95..99)

    // Item drop configuration
    public static final int ITEM_DROP_RATE_PERCENT = 25; // overall drop chance

    // Item weights (relative probability after a drop occurs)
    public static final int ITEM_HEAL_WEIGHT     = 30;
    public static final int ITEM_RAMPAGE_WEIGHT  = 20;
    public static final int ITEM_SUPER_WEIGHT    = 20;
    public static final int ITEM_SHOTGUN_WEIGHT  = 20;
    public static final int ITEM_SHIELD_WEIGHT   = 10;

}
```

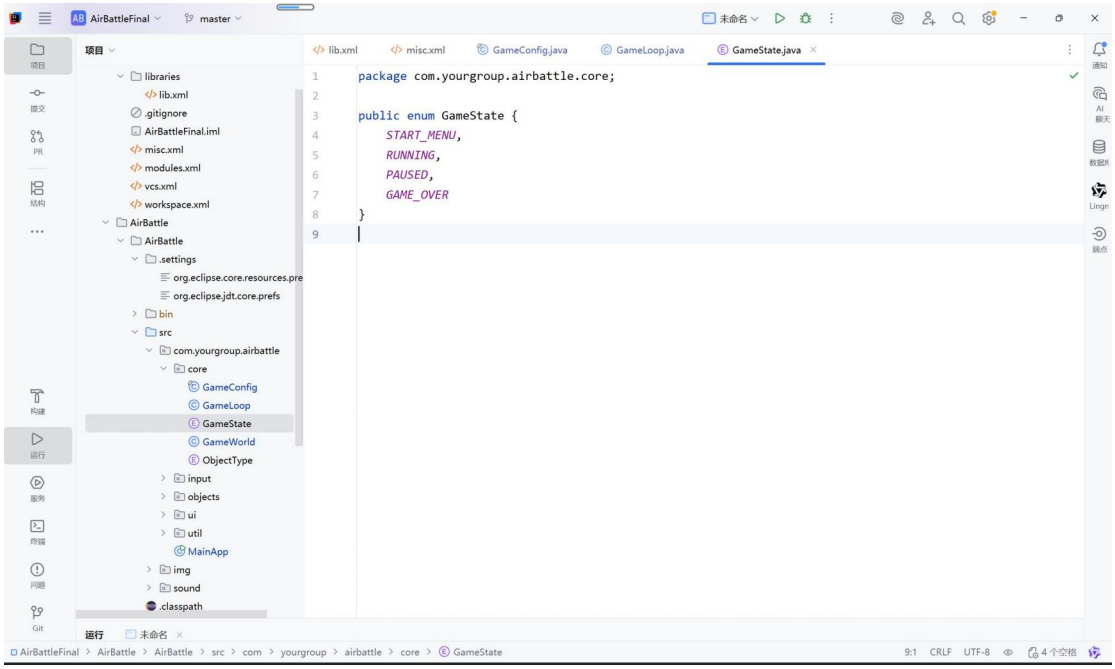
\GameLoop.java



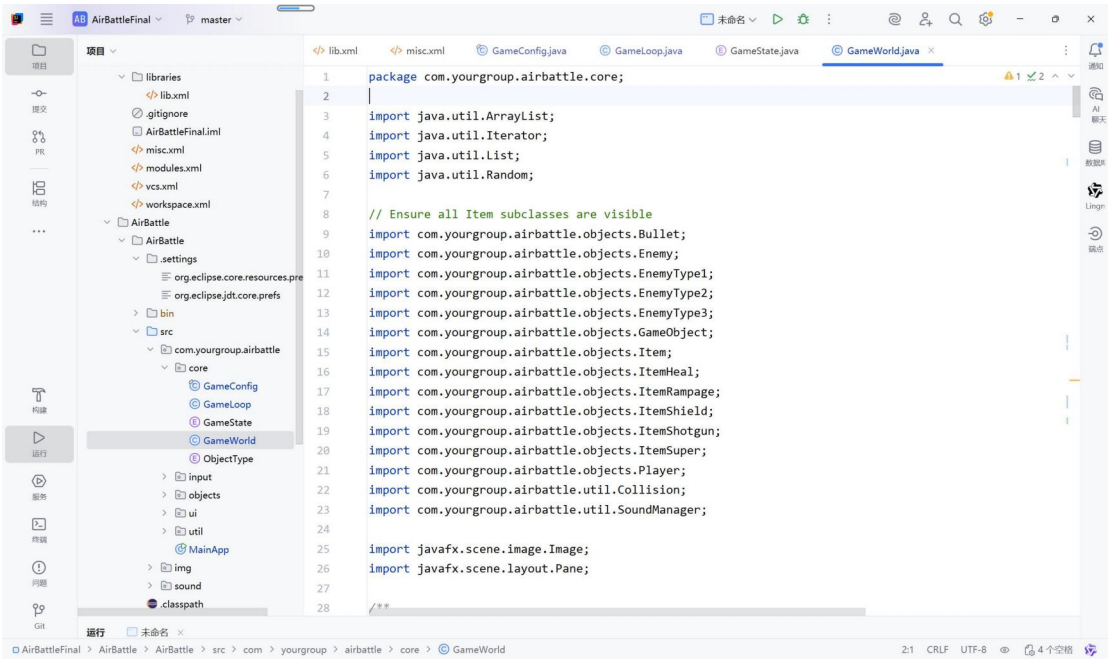
```java
package com.yourgroup.airbattle.core;

import javafx.animation.AnimationTimer;

/**
 * Runs the per-frame update loop using JavaFX {@link AnimationTimer}.
 *
 * <p>This class is intentionally lightweight: it only calculates delta time (dt)
 * and delegates the actual game logic update to {@link GameWorld#update(double)}.
 *
 * <p>Start/Pause behavior is controlled externally (e.g., by MainApp via (pause/gameOver flags)),
 * while this loop maintains a simple {@code running} flag to ignore frames when paused.
 */
public class GameLoop extends AnimationTimer {

    /** The game world that owns and updates all game entities each frame. */
    private final GameWorld world;

    /** The timestamp (ns) of the previous frame, used for computing delta time. */
    private long lastNs = -1;

    /** Whether the loop should process frames (true) or ignore them (false). */
    private boolean running = false;

    /**
     * Creates a game loop bound to the given {@link GameWorld}.
     *
     * @param world the game world to update every frame; must not be null
     */
    public GameLoop(GameWorld world) { this.world = world; }

    /**
     * Called by JavaFX every frame while the AnimationTimer is started.
     * Computes delta time and updates the game world when the loop is running.
     *
     * @param now current timestamp in nanoseconds provided by JavaFX
     */
    @Override
    public void handle(long now) {
        // When paused, ignore frame callbacks without stopping the timer.
        if (!running) return;

        // First frame after starting/resuming: initialize time base and skip update.
        if (lastNs < 0) {
            lastNs = now;
            return;
        }

        // Convert nanoseconds to seconds for frame-based movement/logic updates.
        double dt = (now - lastNs) / 1_000_000_000.0;
        lastNs = now;

        // Defensive clamp: avoid huge dt values due to window switching or lag spikes.
        // This prevents objects from "teleporting" and keeps physics/collisions stable.
```



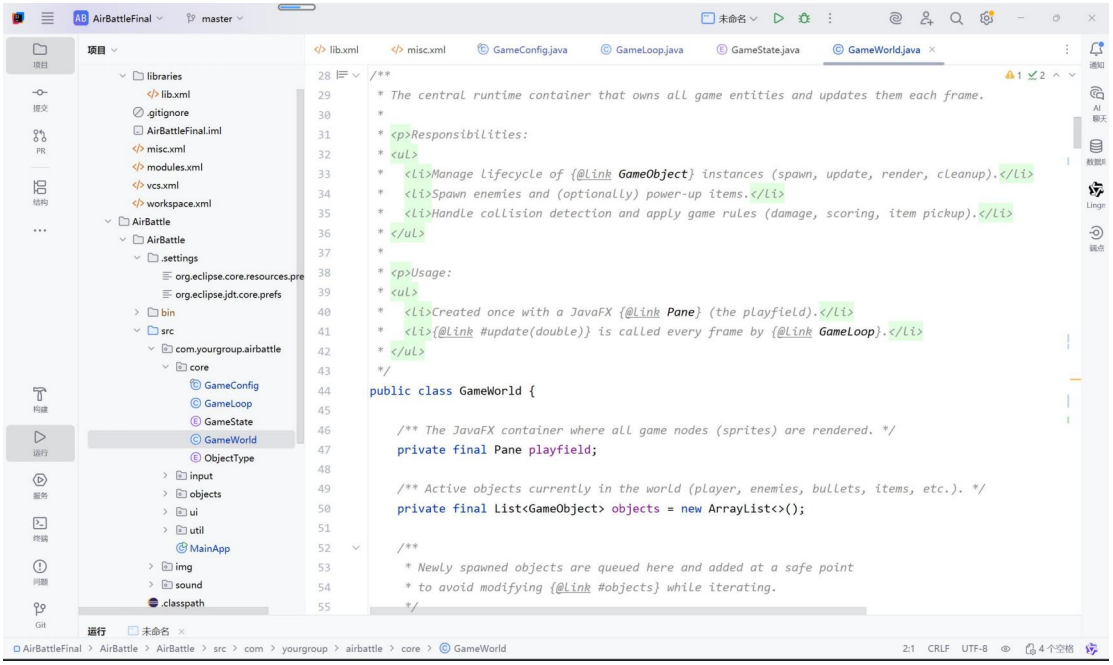
\GameState.java



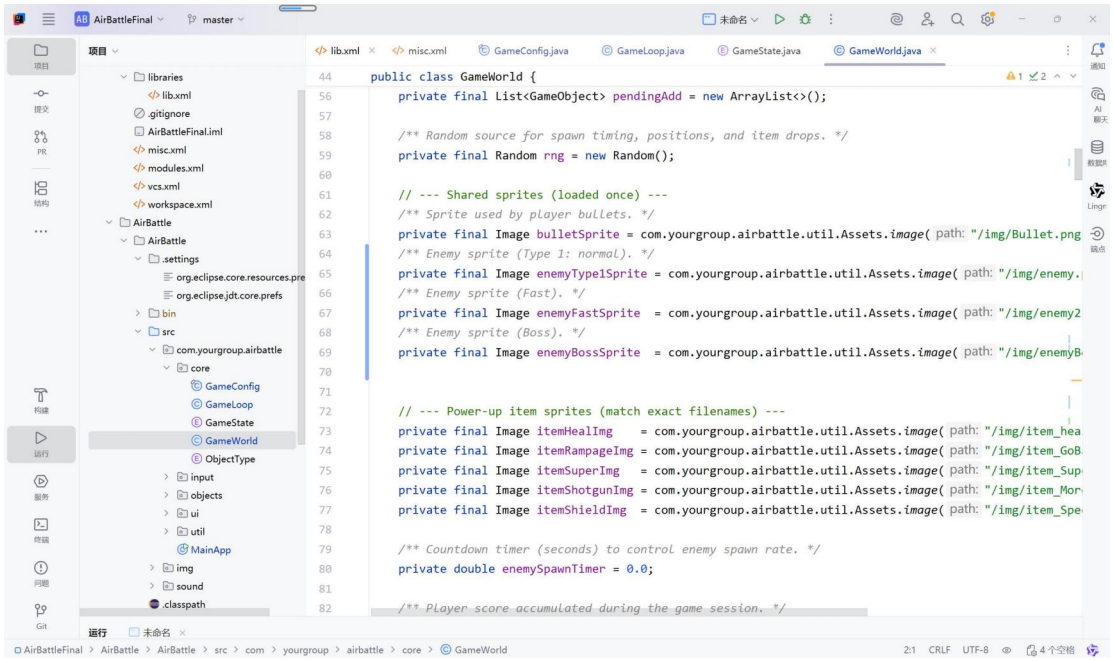
\GameWorld.java

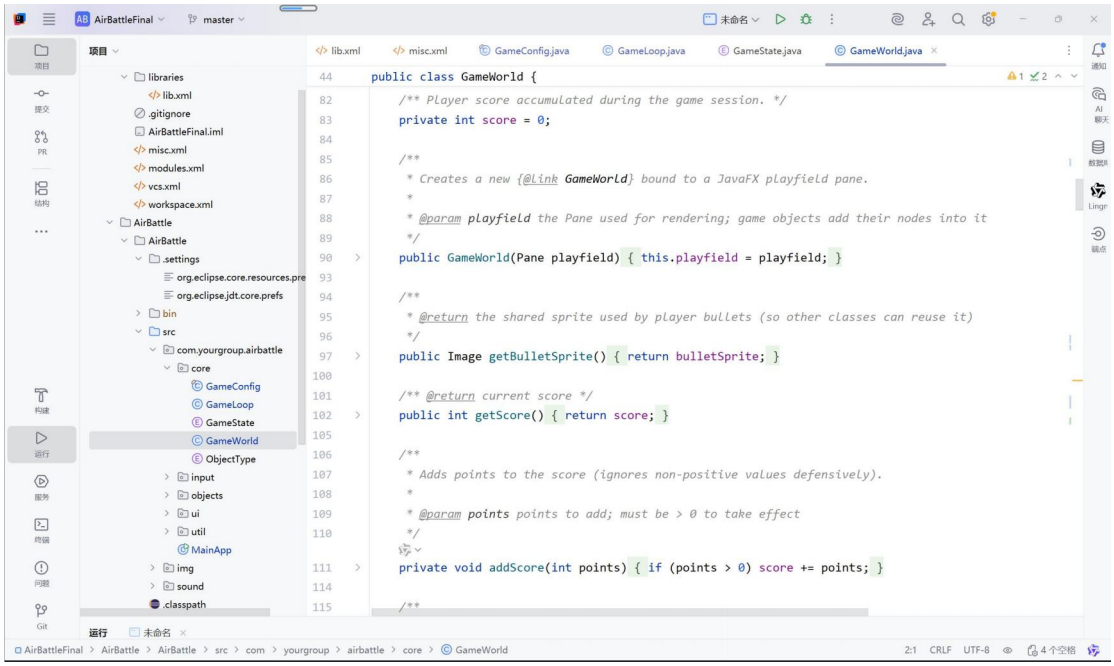


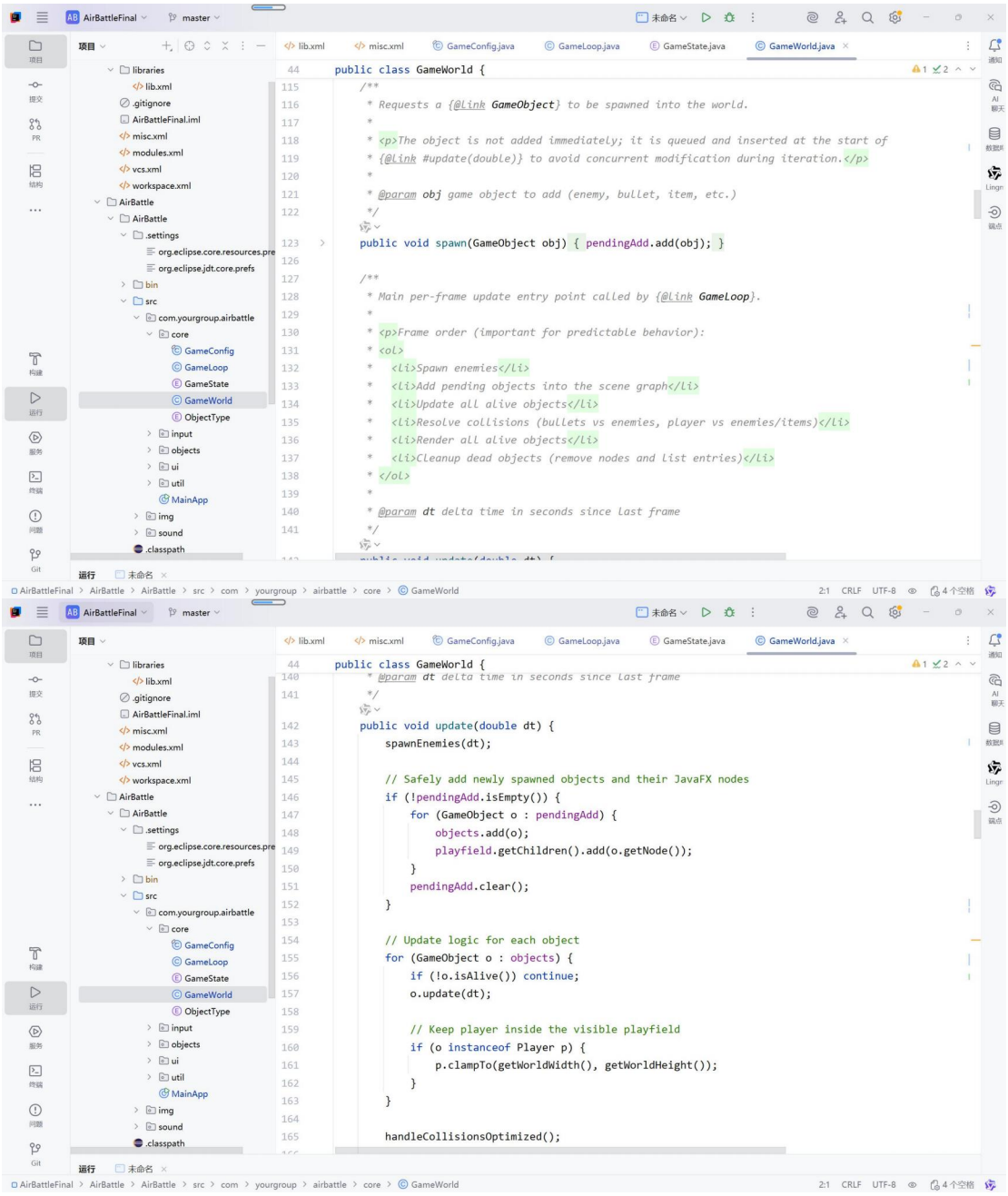
```
package com.yourgroup.airbattle.core;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;

// Ensure all Item subclasses are visible
import com.yourgroup.airbattle.objects.Bullet;
import com.yourgroup.airbattle.objects.Enemy;
import com.yourgroup.airbattle.objects.EnemyType1;
import com.yourgroup.airbattle.objects.EnemyType2;
import com.yourgroup.airbattle.objects.EnemyType3;
import com.yourgroup.airbattle.objects.GameObject;
import com.yourgroup.airbattle.objects.Item;
import com.yourgroup.airbattle.objects.ItemHeal;
import com.yourgroup.airbattle.objects.ItemRampage;
import com.yourgroup.airbattle.objects.ItemShield;
import com.yourgroup.airbattle.objects.ItemShotgun;
import com.yourgroup.airbattle.objects.ItemSuper;
import com.yourgroup.airbattle.objects.Player;
import com.yourgroup.airbattle.util.Collision;
import com.yourgroup.airbattle.util.SoundManager;

import javafx.scene.image.Image;
import javafx.scene.layout.Pane;

/**
```



```
/**
 * The central runtime container that owns all game entities and updates them each frame.
 *
 * <p>Responsibilities:
 * <ul>
 *   <li>Manage lifecycle of {@link GameObject} instances (spawn, update, render, cleanup).</li>
 *   <li>Spawn enemies and (optionally) power-up items.</li>
 *   <li>Handle collision detection and apply game rules (damage, scoring, item pickup).</li>
 * </ul>
 *
 * <p>Usage:
 * <ul>
 *   <li>Created once with a JavaFX {@link Pane} (the playfield).</li>
 *   <li>{@link #update(double)} is called every frame by {@link GameLoop}.</li>
 * </ul>
 */
public class GameWorld {

    /** The JavaFX container where all game nodes (sprites) are rendered. */
    private final Pane playfield;

    /** Active objects currently in the world (player, enemies, bullets, items, etc.). */
    private final List<GameObject> objects = new ArrayList<>();

    /**
     * Newly spawned objects are queued here and added at a safe point
     * to avoid modifying {@link #objects} while iterating.
     */
```


AirBattleFinal
master
lib.xml
misc.xml
GameConfig.java
GameLoop.java
GameState.java
GameWorld.java
public class GameWorld {
private final List<GameObject> pendingAdd = new ArrayList<>();
/** Random source for spawn timing, positions, and item drops. */
private final Random rng = new Random();
// --- Shared sprites (loaded once) ---
/** Sprite used by player bullets. */
private final Image bulletSprite = com.yourgroup.airbattle.util.Assets.image(path: "/img/Bullet.png
/** Enemy sprite (Type 1: normal). */
private final Image enemyType1Sprite = com.yourgroup.airbattle.util.Assets.image(path: "/img/enemy.
/** Enemy sprite (Fast). */
private final Image enemyFastSprite = com.yourgroup.airbattle.util.Assets.image(path: "/img/enemy2
/** Enemy sprite (Boss). */
private final Image enemyBossSprite = com.yourgroup.airbattle.util.Assets.image(path: "/img/enemyB
// --- Power-up item sprites (match exact filenames) ---
private final Image itemHealImg = com.yourgroup.airbattle.util.Assets.image(path: "/img/item_hea
private final Image itemRampageImg = com.yourgroup.airbattle.util.Assets.image(path: "/img/item_GoB
private final Image itemSuperImg = com.yourgroup.airbattle.util.Assets.image(path: "/img/item_Sup
private final Image itemShotgunImg = com.yourgroup.airbattle.util.Assets.image(path: "/img/item_Mor
private final Image itemShieldImg = com.yourgroup.airbattle.util.Assets.image(path: "/img/item_Spe
/** Countdown timer (seconds) to control enemy spawn rate. */
private double enemySpawnTimer = 0.0;
/** Player score accumulated during the game session. */
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > core > GameWorld
2:1 CRLF UTF-8 4 个空格

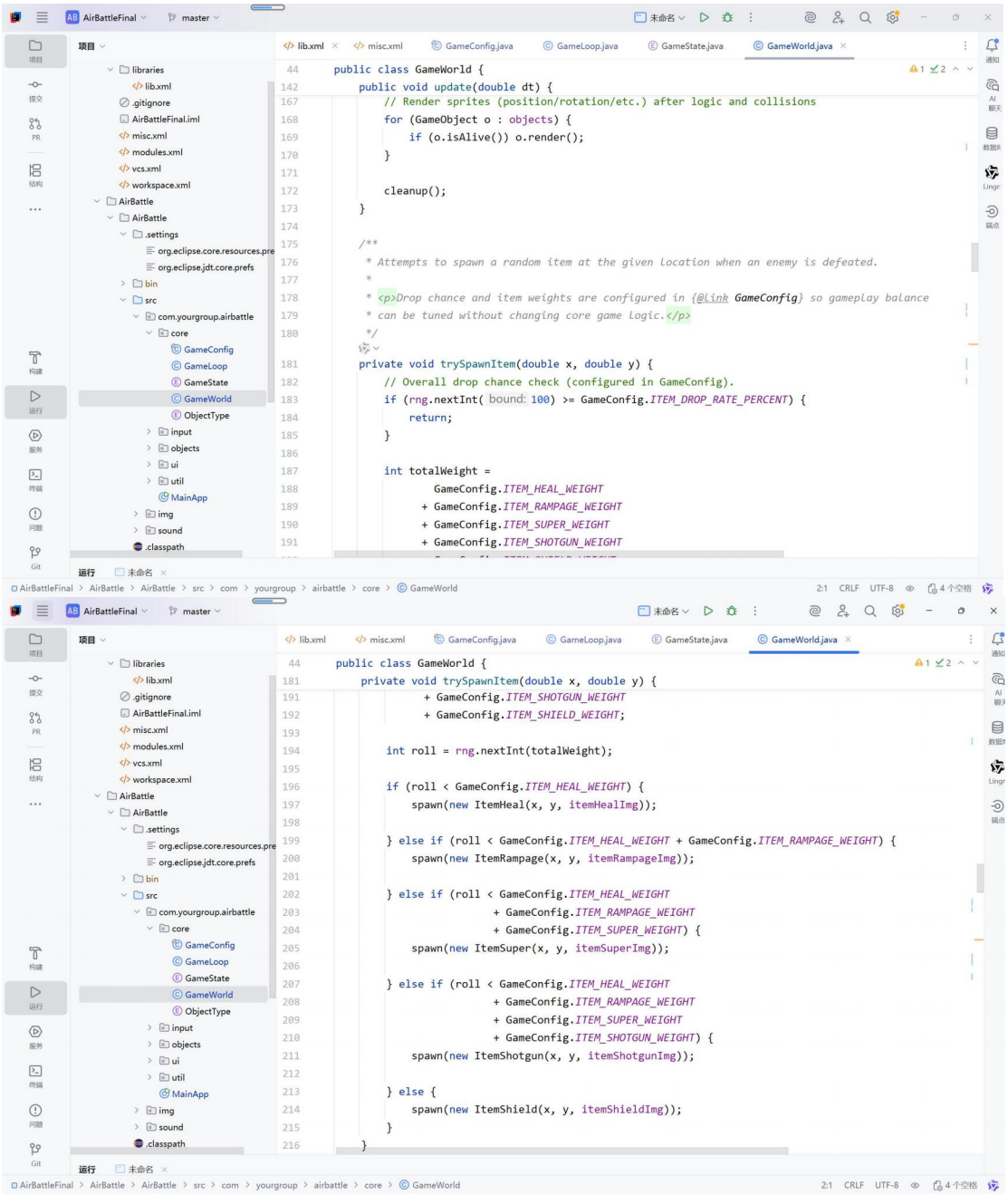

AirBattleFinal
master
lib.xml
misc.xml
GameConfig.java
GameLoop.java
GameState.java
GameWorld.java
public class GameWorld {
/** Player score accumulated during the game session. */
private int score = 0;
/**
* Creates a new {@link GameWorld} bound to a JavaFX playfield pane.
*
* @param playfield the Pane used for rendering; game objects add their nodes into it
*/
public GameWorld(Pane playfield) { this.playfield = playfield; }
/**
* @return the shared sprite used by player bullets (so other classes can reuse it)
*/
public Image getBulletSprite() { return bulletSprite; }
/** @return current score */
public int getScore() { return score; }
/**
* Adds points to the score (ignores non-positive values defensively).
*
* @param points points to add; must be > 0 to take effect
*/
private void addScore(int points) { if (points > 0) score += points; }
/**
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > core > GameWorld
2:1 CRLF UTF-8 4 个空格

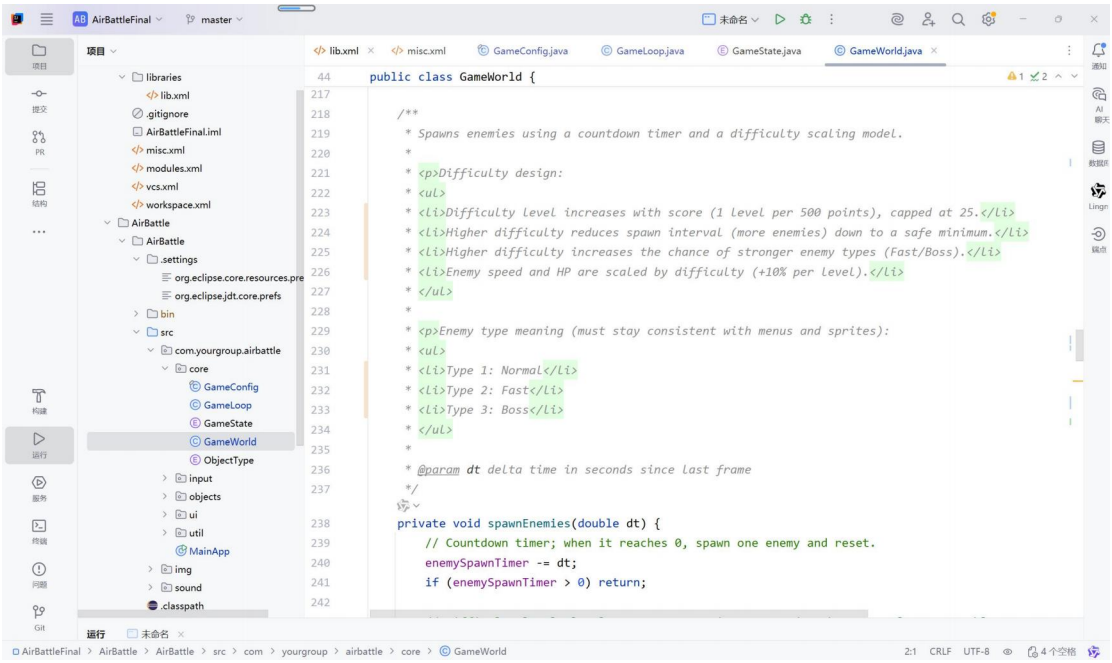


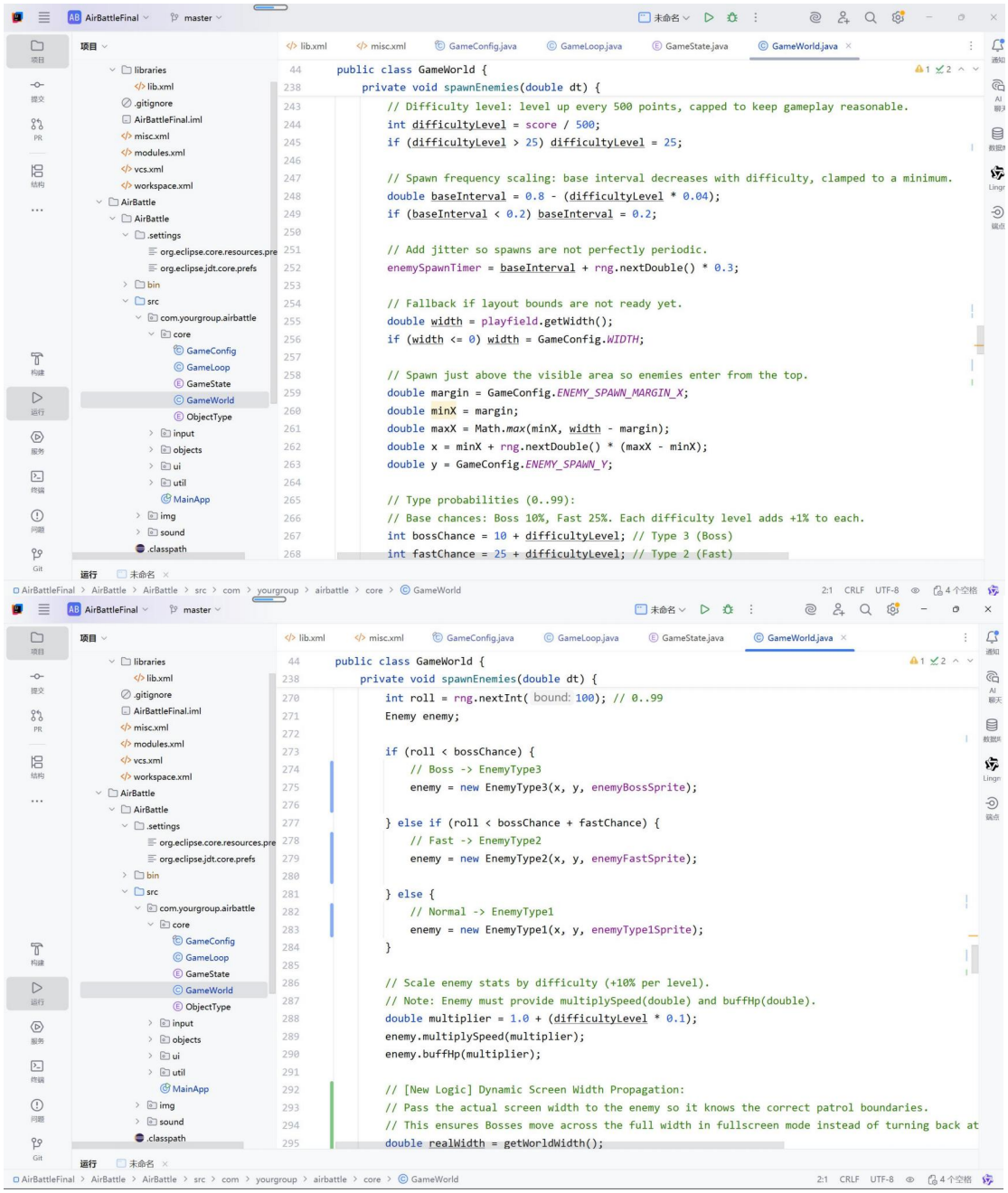
```
/**
 * Requests a {@link GameObject} to be spawned into the world.
 *
 * <p>The object is not added immediately; it is queued and inserted at the start of
 * {@link #update(double)} to avoid concurrent modification during iteration.</p>
 *
 * @param obj game object to add (enemy, bullet, item, etc.)
 */
public void spawn(GameObject obj) { pendingAdd.add(obj); }

/**
 * Main per-frame update entry point called by {@link GameLoop}.
 *
 * <p>Frame order (important for predictable behavior):
 * <ol>
 *   <li>Spawn enemies</li>
 *   <li>Add pending objects into the scene graph</li>
 *   <li>Update all alive objects</li>
 *   <li>Resolve collisions (bullets vs enemies, player vs enemies/items)</li>
 *   <li>Render all alive objects</li>
 *   <li>Cleanup dead objects (remove nodes and list entries)</li>
 * </ol>
 *
 * @param dt delta time in seconds since last frame
 */
```



```
 * @param dt delta time in seconds since last frame
 */
public void update(double dt) {
    spawnEnemies(dt);

    // Safely add newly spawned objects and their JavaFX nodes
    if (!pendingAdd.isEmpty()) {
        for (GameObject o : pendingAdd) {
            objects.add(o);
            playfield.getChildren().add(o.getNode());
        }
        pendingAdd.clear();
    }

    // Update logic for each object
    for (GameObject o : objects) {
        if (!o.isAlive()) continue;
        o.update(dt);

        // Keep player inside the visible playfield
        if (o instanceof Player p) {
            p.clampTo(getWorldWidth(), getWorldHeight());
        }
    }

    handleCollisionsOptimized();
```

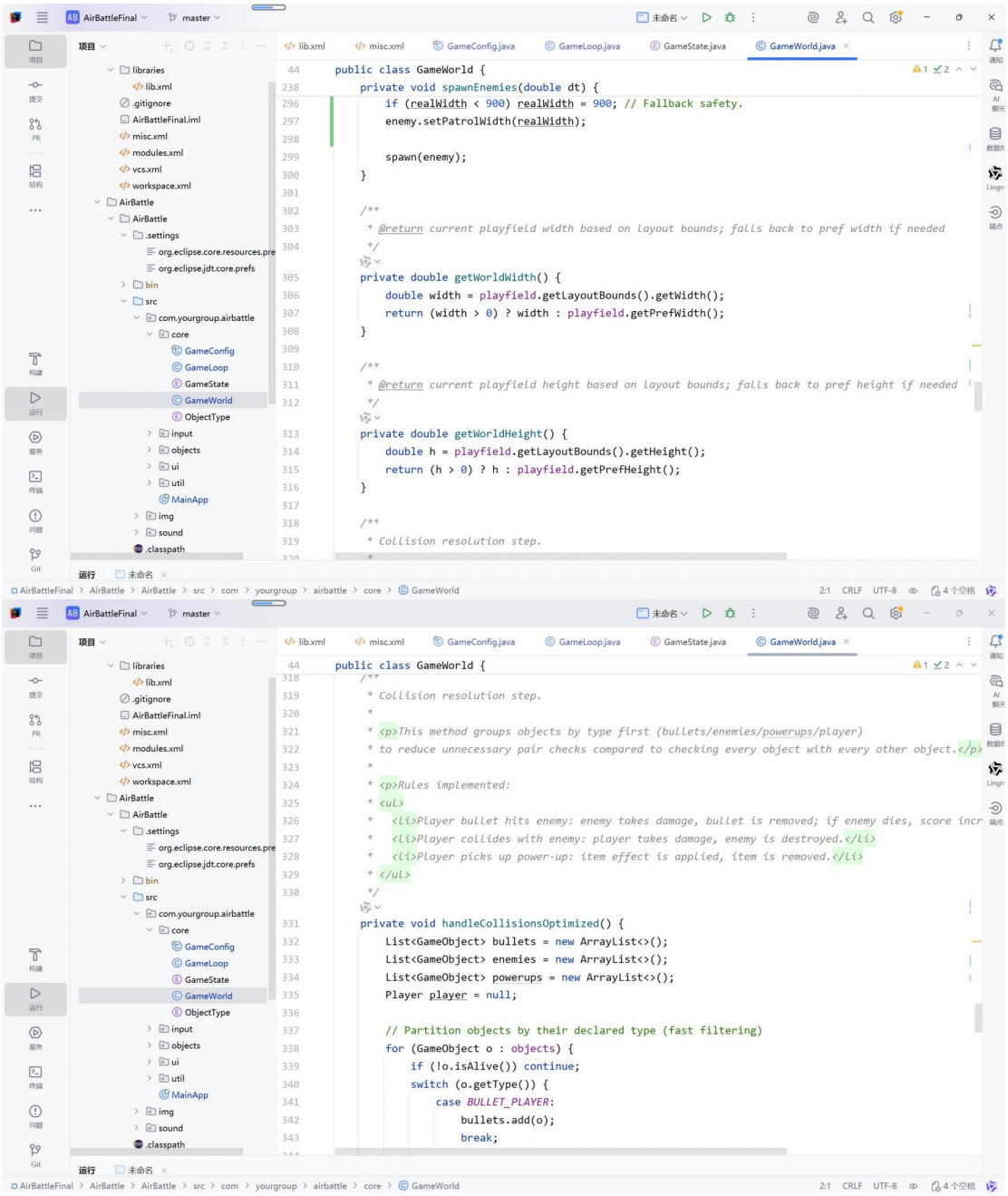

AirBattleFinal
master
未命名
lib.xml
misc.xml
GameConfig.java
GameLoop.java
GameState.java
GameWorld.java
项目
libraries
lib.xml
.gitignore
AirBattleFinal.iml
misc.xml
modules.xml
vcs.xml
workspace.xml
AirBattle
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
objects
ui
util
MainApp
img
sound
.classpath
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > core > GameWorld
2:1
CRLF
UTF-8
4 个空格
public class GameWorld {
public void update(double dt) {
// Render sprites (position/rotation/etc.) after logic and collisions
for (GameObject o : objects) {
if (o.isAlive()) o.render();
}
cleanup();
}
/**
* Attempts to spawn a random item at the given location when an enemy is defeated.
*
* Drop chance and item weights are configured in {@link GameConfig} so gameplay balance
* can be tuned without changing core game logic.
*/
private void trySpawnItem(double x, double y) {
// Overall drop chance check (configured in GameConfig).
if (rng.nextInt(bound: 100) >= GameConfig.ITEM_DROP_RATE_PERCENT) {
return;
}
int totalWeight =
GameConfig.ITEM_HEAL_WEIGHT
+ GameConfig.ITEM_RAMPAGE_WEIGHT
+ GameConfig.ITEM_SUPER_WEIGHT
+ GameConfig.ITEM_SHOTGUN_WEIGHT
+ GameConfig.ITEM_SHIELD_WEIGHT;
int roll = rng.nextInt(totalWeight);
if (roll < GameConfig.ITEM_HEAL_WEIGHT) {
spawn(new ItemHeal(x, y, itemHealImg));
} else if (roll < GameConfig.ITEM_HEAL_WEIGHT + GameConfig.ITEM_RAMPAGE_WEIGHT) {
spawn(new ItemRampage(x, y, itemRampageImg));
} else if (roll < GameConfig.ITEM_HEAL_WEIGHT
+ GameConfig.ITEM_RAMPAGE_WEIGHT
+ GameConfig.ITEM_SUPER_WEIGHT) {
spawn(new ItemSuper(x, y, itemSuperImg));
} else if (roll < GameConfig.ITEM_HEAL_WEIGHT
+ GameConfig.ITEM_RAMPAGE_WEIGHT
+ GameConfig.ITEM_SUPER_WEIGHT
+ GameConfig.ITEM_SHOTGUN_WEIGHT) {
spawn(new ItemShotgun(x, y, itemShotgunImg));
} else {
spawn(new ItemShield(x, y, itemShieldImg));
}
}


AirBattleFinal
master
lib.xml
misc.xml
GameConfig.java
GameLoop.java
GameState.java
GameWorld.java
public class GameWorld {
/**
* Spawns enemies using a countdown timer and a difficulty scaling model.
*
* <p>Difficulty design:
*
* Difficulty level increases with score (1 level per 500 points), capped at 25.
* Higher difficulty reduces spawn interval (more enemies) down to a safe minimum.
* Higher difficulty increases the chance of stronger enemy types (Fast/Boss).
* Enemy speed and HP are scaled by difficulty (+10% per level).
*
*
* <p>Enemy type meaning (must stay consistent with menus and sprites):
*
* Type 1: Normal
* Type 2: Fast
* Type 3: Boss
*
*
* @param dt delta time in seconds since last frame
*/
private void spawnEnemies(double dt) {
// Countdown timer; when it reaches 0, spawn one enemy and reset.
enemySpawnTimer -= dt;
if (enemySpawnTimer > 0) return;
libraries
lib.xml
.gitignore
AirBattleFinal.iml
misc.xml
modules.xml
vcs.xml
workspace.xml
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
objects
ui
util
MainApp
img
sound
.classpath
AirBattleFinal › AirBattle › AirBattle › src › com › yourgroup › airbattle › core › GameWorld
2:1 CRLF UTF-8

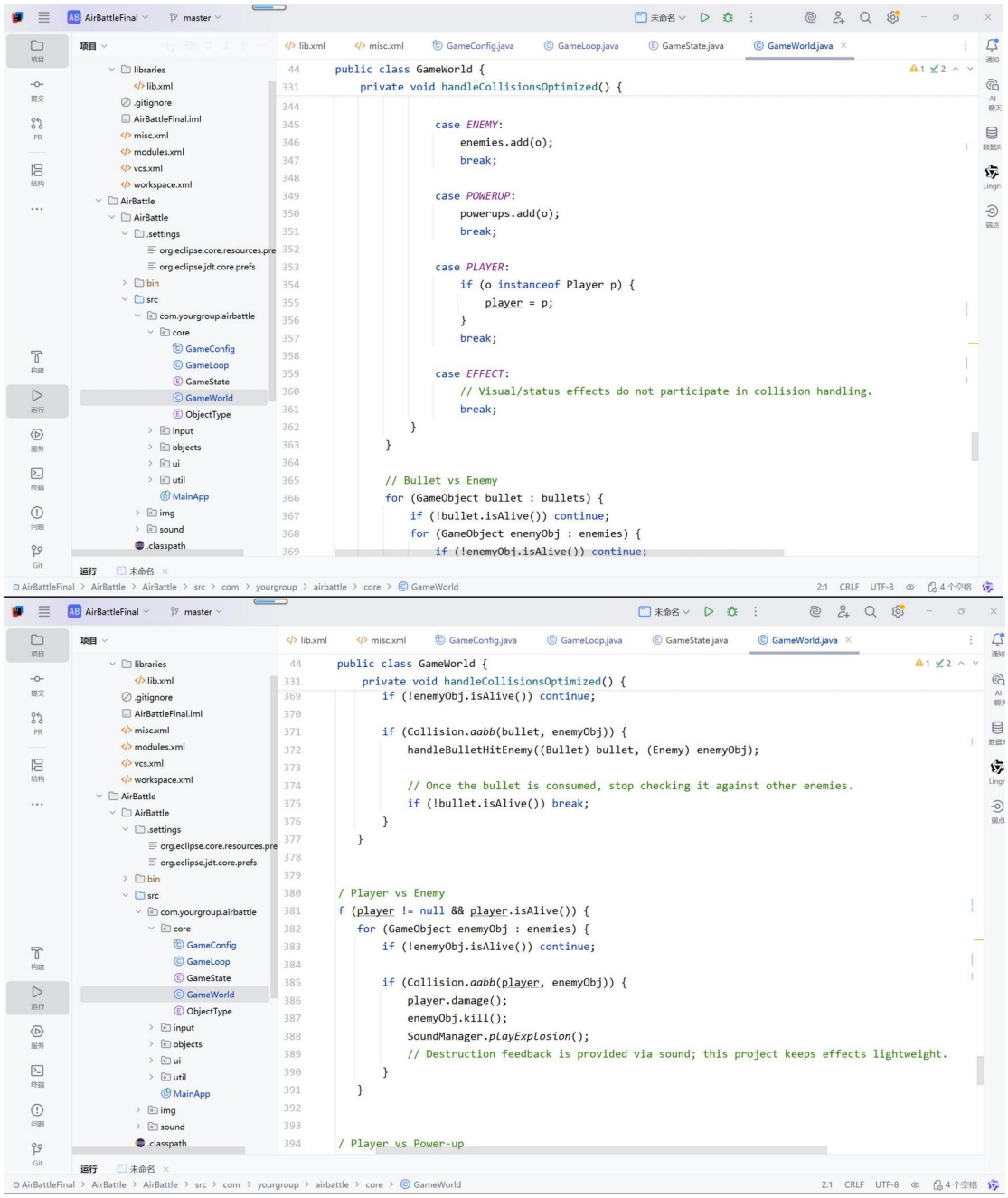


```
// Difficulty level: level up every 500 points, capped to keep gameplay reasonable.
int difficultyLevel = score / 500;
if (difficultyLevel > 25) difficultyLevel = 25;

// Spawn frequency scaling: base interval decreases with difficulty, clamped to a minimum.
double baseInterval = 0.8 - (difficultyLevel * 0.04);
if (baseInterval < 0.2) baseInterval = 0.2;

// Add jitter so spawns are not perfectly periodic.
enemySpawnTimer = baseInterval + rng.nextDouble() * 0.3;

// Fallback if layout bounds are not ready yet.
double width = playfield.getWidth();
if (width <= 0) width = GameConfig.WIDTH;

// Spawn just above the visible area so enemies enter from the top.
double margin = GameConfig.ENEMY_SPAWN_MARGIN_X;
double minX = margin;
double maxX = Math.max(minX, width - margin);
double x = minX + rng.nextDouble() * (maxX - minX);
double y = GameConfig.ENEMY_SPAWN_Y;

// Type probabilities (0..99):
// Base chances: Boss 10%, Fast 25%. Each difficulty level adds +1% to each.
int bossChance = 10 + difficultyLevel; // Type 3 (Boss)
int fastChance = 25 + difficultyLevel; // Type 2 (Fast)
```

```
int roll = rng.nextInt( bound: 100); // 0..99
Enemy enemy;

if (roll < bossChance) {
    // Boss -> EnemyType3
    enemy = new EnemyType3(x, y, enemyBossSprite);

} else if (roll < bossChance + fastChance) {
    // Fast -> EnemyType2
    enemy = new EnemyType2(x, y, enemyFastSprite);

} else {
    // Normal -> EnemyType1
    enemy = new EnemyType1(x, y, enemyType1Sprite);
}

// Scale enemy stats by difficulty (+10% per level).
// Note: Enemy must provide multiplySpeed(double) and buffHp(double).
double multiplier = 1.0 + (difficultyLevel * 0.1);
enemy.multiplySpeed(multiplier);
enemy.buffHp(multiplier);

// [New Logic] Dynamic Screen Width Propagation:
// Pass the actual screen width to the enemy so it knows the correct patrol boundaries.
// This ensures Bosses move across the full width in fullscreen mode instead of turning back at
double realWidth = getWorldWidth();
```



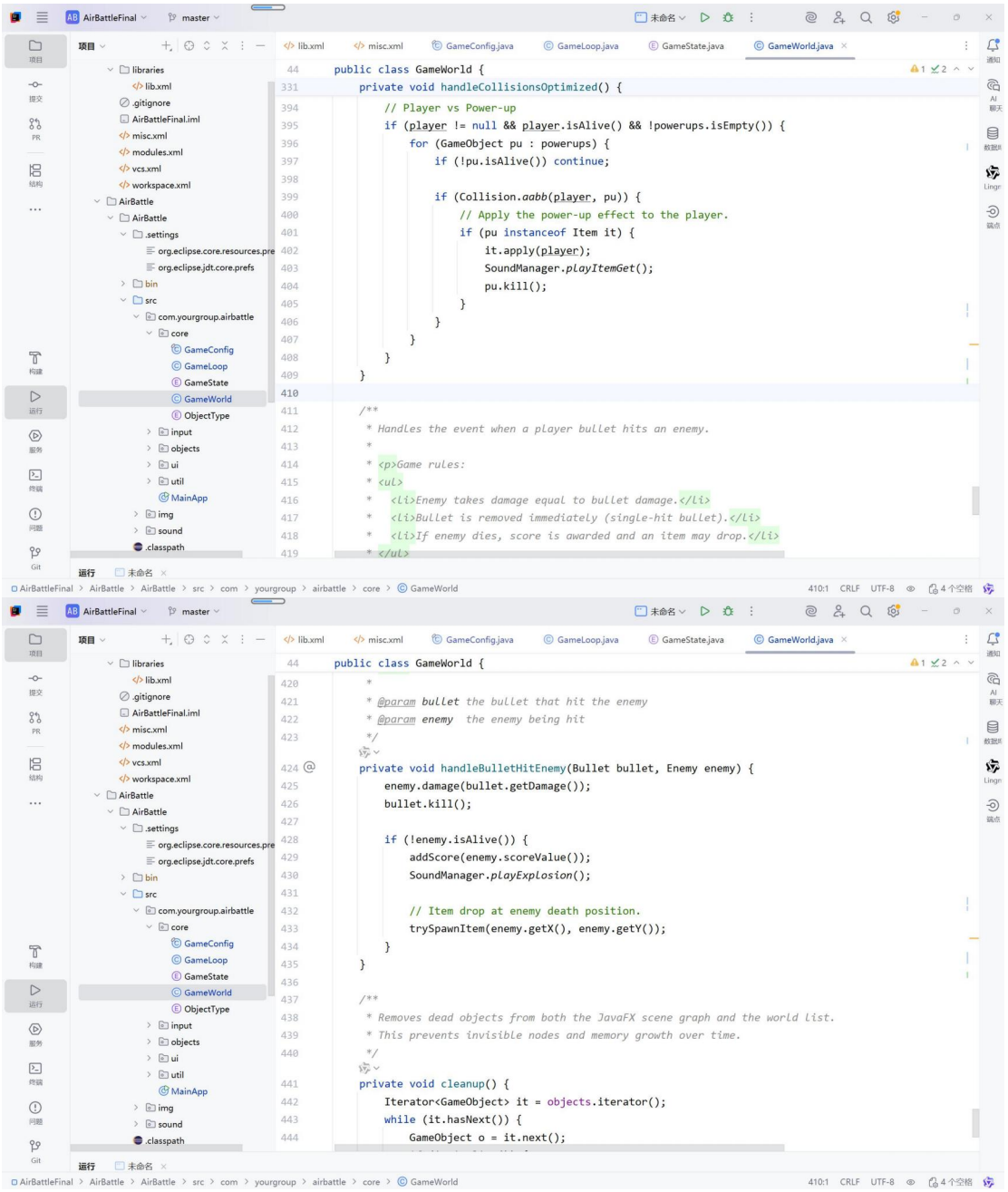
```
public class GameWorld {
    private void spawnEnemies(double dt) {
        if (realWidth < 900) realWidth = 900; // Fallback safety.
        enemy.setPatrolWidth(realWidth);

        spawn(enemy);
    }

    /**
     * @return current playfield width based on layout bounds; falls back to pref width if needed
     */
    private double getWorldWidth() {
        double width = playfield.getLayoutBounds().getWidth();
        return (width > 0) ? width : playfield.getPrefWidth();
    }

    /**
     * @return current playfield height based on layout bounds; falls back to pref height if needed
     */
    private double getWorldHeight() {
        double h = playfield.getLayoutBounds().getHeight();
        return (h > 0) ? h : playfield.getPrefHeight();
    }

    /**
     * Collision resolution step.
     *
     * <p>This method groups objects by type first (bullets/enemies/powerups/player)
     * to reduce unnecessary pair checks compared to checking every object with every other object.</p>
     *
     * <p>Rules implemented:
     * <ul>
     *   <li>Player bullet hits enemy: enemy takes damage, bullet is removed; if enemy dies, score incr
     *   <li>Player collides with enemy: player takes damage, enemy is destroyed.</li>
     *   <li>Player picks up power-up: item effect is applied, item is removed.</li>
     * </ul>
     */
    private void handleCollisionsOptimized() {
        List<GameObject> bullets = new ArrayList<>();
        List<GameObject> enemies = new ArrayList<>();
        List<GameObject> powerups = new ArrayList<>();
        Player player = null;

        // Partition objects by their declared type (fast filtering)
        for (GameObject o : objects) {
            if (!o.isAlive()) continue;
            switch (o.getType()) {
                case BULLET_PLAYER:
                    bullets.add(o);
                    break;
```



```
public class GameWorld {
    private void handleCollisionsOptimized() {

                case ENEMY:
                    enemies.add(o);
                    break;

                case POWERUP:
                    powerups.add(o);
                    break;

                case PLAYER:
                    if (o instanceof Player p) {
                        player = p;
                    }
                    break;

                case EFFECT:
                    // Visual/status effects do not participate in collision handling.
                    break;
            }
        }

        // Bullet vs Enemy
        for (GameObject bullet : bullets) {
            if (!bullet.isAlive()) continue;
            for (GameObject enemyObj : enemies) {
                if (!enemyObj.isAlive()) continue;
```

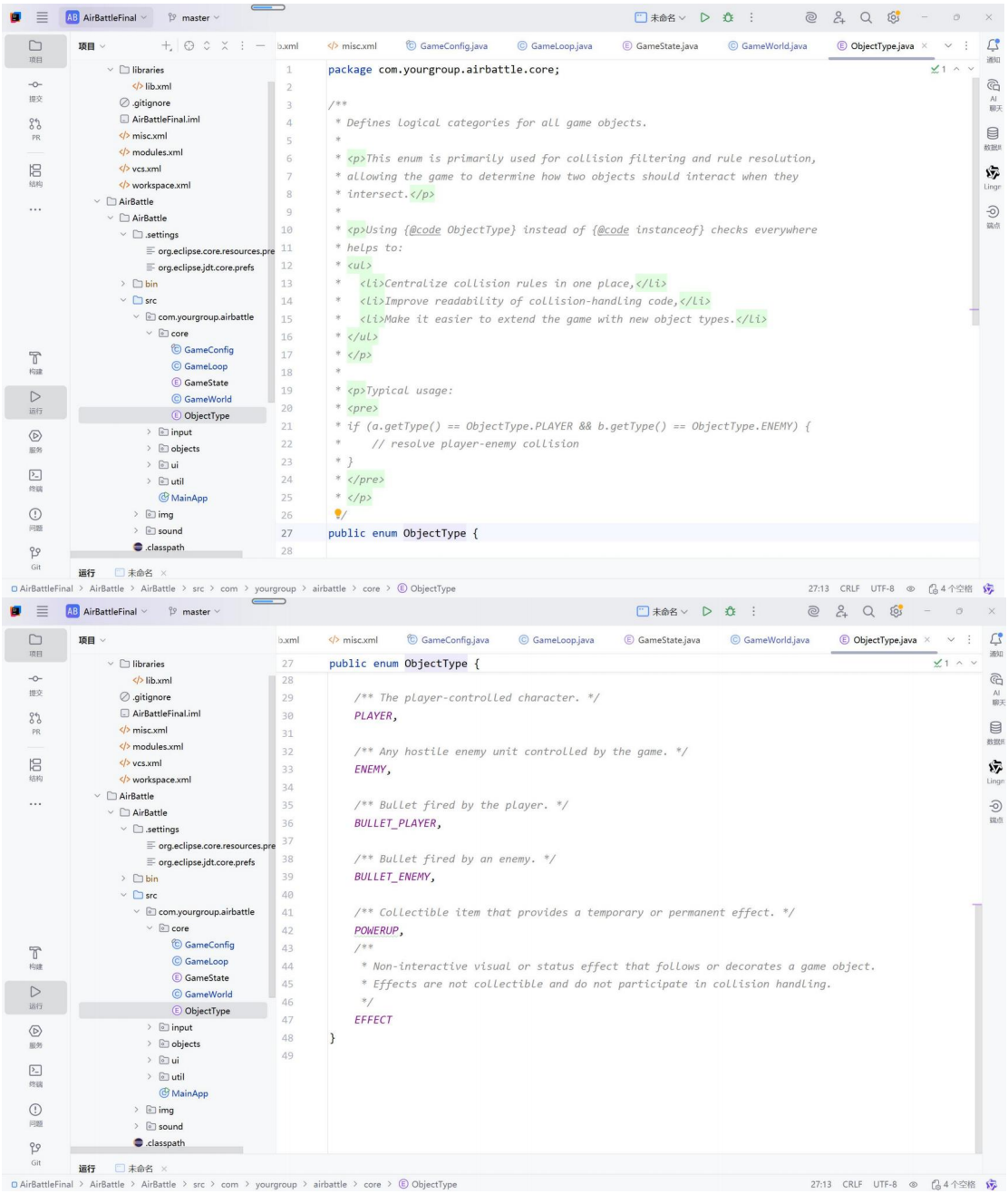


```
public class GameWorld {
    private void handleCollisionsOptimized() {
            if (!enemyObj.isAlive()) continue;

            if (Collision.aabb(bullet, enemyObj)) {
                handleBulletHitEnemy((Bullet) bullet, (Enemy) enemyObj);

                // Once the bullet is consumed, stop checking it against other enemies.
                if (!bullet.isAlive()) break;
            }
        }


    / Player vs Enemy
    f (player != null && player.isAlive()) {
        for (GameObject enemyObj : enemies) {
            if (!enemyObj.isAlive()) continue;

            if (Collision.aabb(player, enemyObj)) {
                player.damage();
                enemyObj.kill();
                SoundManager.playExplosion();
                // Destruction feedback is provided via sound; this project keeps effects lightweight.
            }
        }


    / Player vs Power-up
```



```
public class GameWorld {
    private void handleCollisionsOptimized() {
        // Player vs Power-up
        if (player != null && player.isAlive() && !powerups.isEmpty()) {
            for (GameObject pu : powerups) {
                if (!pu.isAlive()) continue;

                if (Collision.aabb(player, pu)) {
                    // Apply the power-up effect to the player.
                    if (pu instanceof Item it) {
                        it.apply(player);
                        SoundManager.playItemGet();
                        pu.kill();
                    }
                }
            }
        }
    }

    /**
     * Handles the event when a player bullet hits an enemy.
     *
     * <p>Game rules:
     * <ul>
     *   <li>Enemy takes damage equal to bullet damage.</li>
     *   <li>Bullet is removed immediately (single-hit bullet).</li>
     *   <li>If enemy dies, score is awarded and an item may drop.</li>
     * </ul>
     *
     * @param bullet the bullet that hit the enemy
     * @param enemy  the enemy being hit
     */
    private void handleBulletHitEnemy(Bullet bullet, Enemy enemy) {
        enemy.damage(bullet.getDamage());
        bullet.kill();

        if (!enemy.isAlive()) {
            addScore(enemy.scoreValue());
            SoundManager.playExplosion();

            // Item drop at enemy death position.
            trySpawnItem(enemy.getX(), enemy.getY());
        }
    }

    /**
     * Removes dead objects from both the JavaFX scene graph and the world list.
     * This prevents invisible nodes and memory growth over time.
     */
    private void cleanup() {
        Iterator<GameObject> it = objects.iterator();
        while (it.hasNext()) {
            GameObject o = it.next();
```

\ObjectType.java



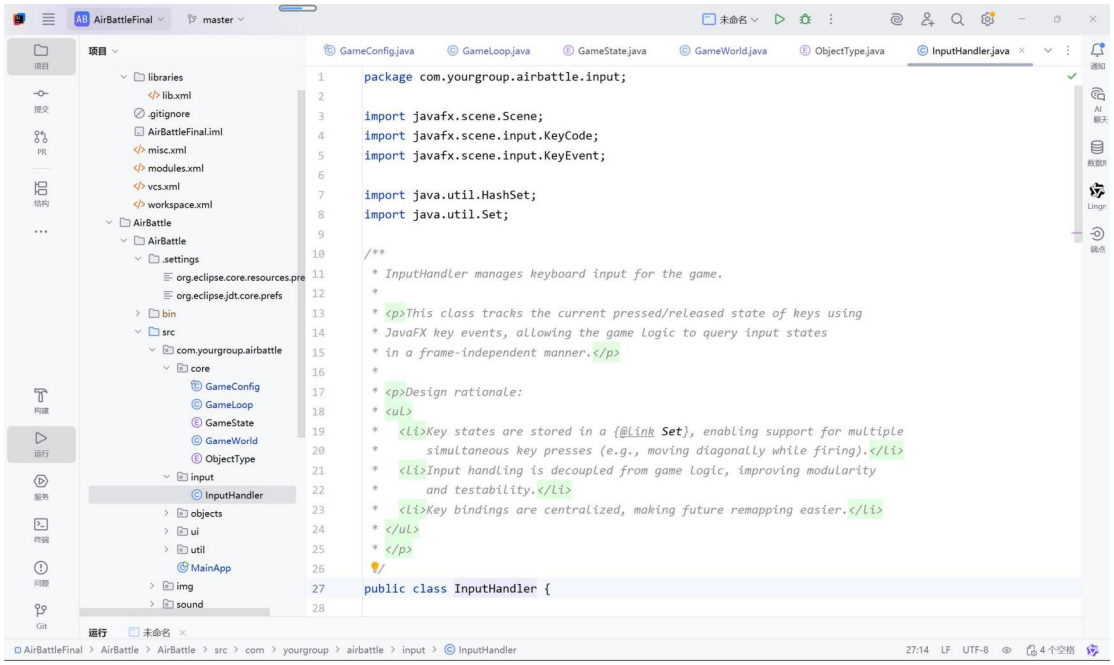
```
package com.yourgroup.airbattle.core;

/**
 * Defines logical categories for all game objects.
 *
 * <p>This enum is primarily used for collision filtering and rule resolution,
 * allowing the game to determine how two objects should interact when they
 * intersect.</p>
 *
 * <p>Using {@code ObjectType} instead of {@code instanceof} checks everywhere
 * helps to:
 * <ul>
 *   <li>Centralize collision rules in one place,</li>
 *   <li>Improve readability of collision-handling code,</li>
 *   <li>Make it easier to extend the game with new object types.</li>
 * </ul>
 * </p>
 *
 * <p>Typical usage:
 * <pre>
 * if (a.getType() == ObjectType.PLAYER && b.getType() == ObjectType.ENEMY) {
 *     // resolve player-enemy collision
 * }
 * </pre>
 * </p>
 */
public enum ObjectType {
```

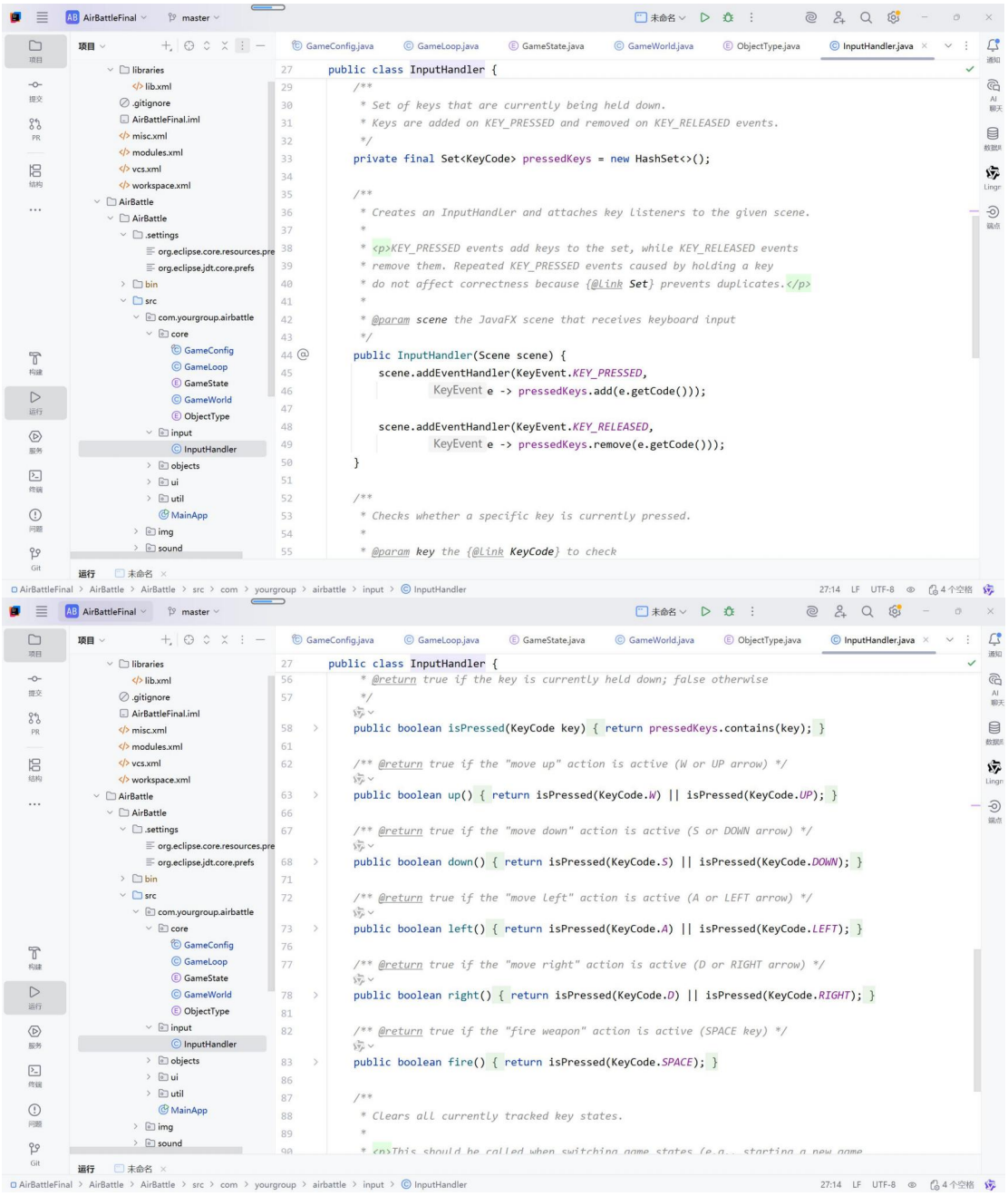


```
public enum ObjectType {

    /** The player-controlled character. */
    PLAYER,

    /** Any hostile enemy unit controlled by the game. */
    ENEMY,

    /** Bullet fired by the player. */
    BULLET_PLAYER,

    /** Bullet fired by an enemy. */
    BULLET_ENEMY,

    /** Collectible item that provides a temporary or permanent effect. */
    POWERUP,
    /**
     * Non-interactive visual or status effect that follows or decorates a game object.
     * Effects are not collectible and do not participate in collision handling.
     */
    EFFECT
}
```

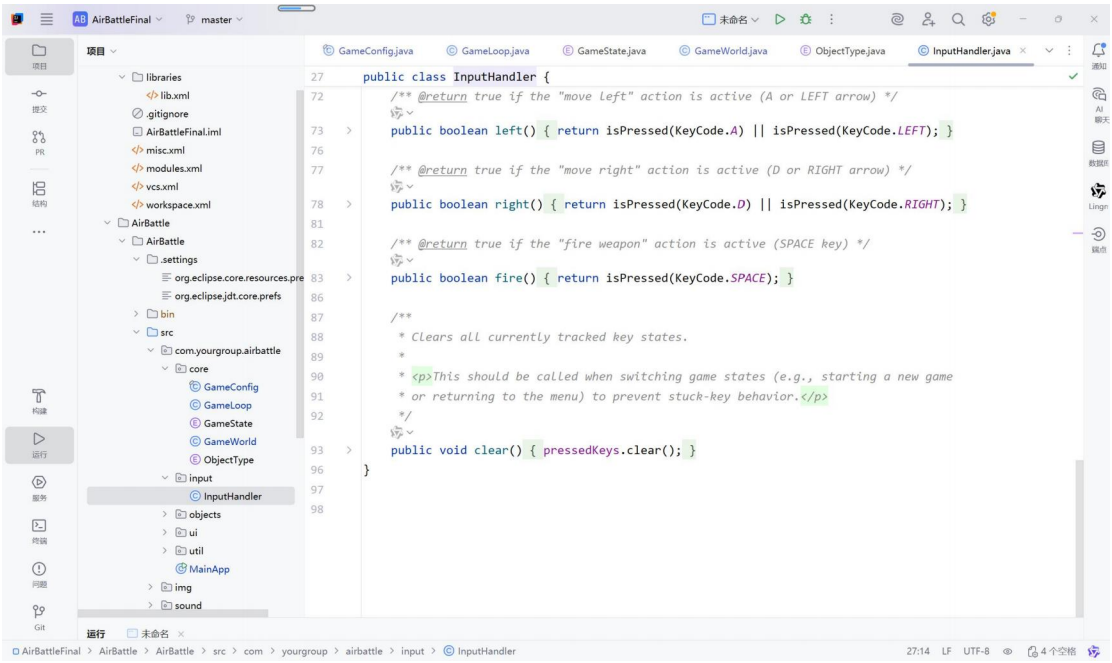
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\input

\InputHandler.java



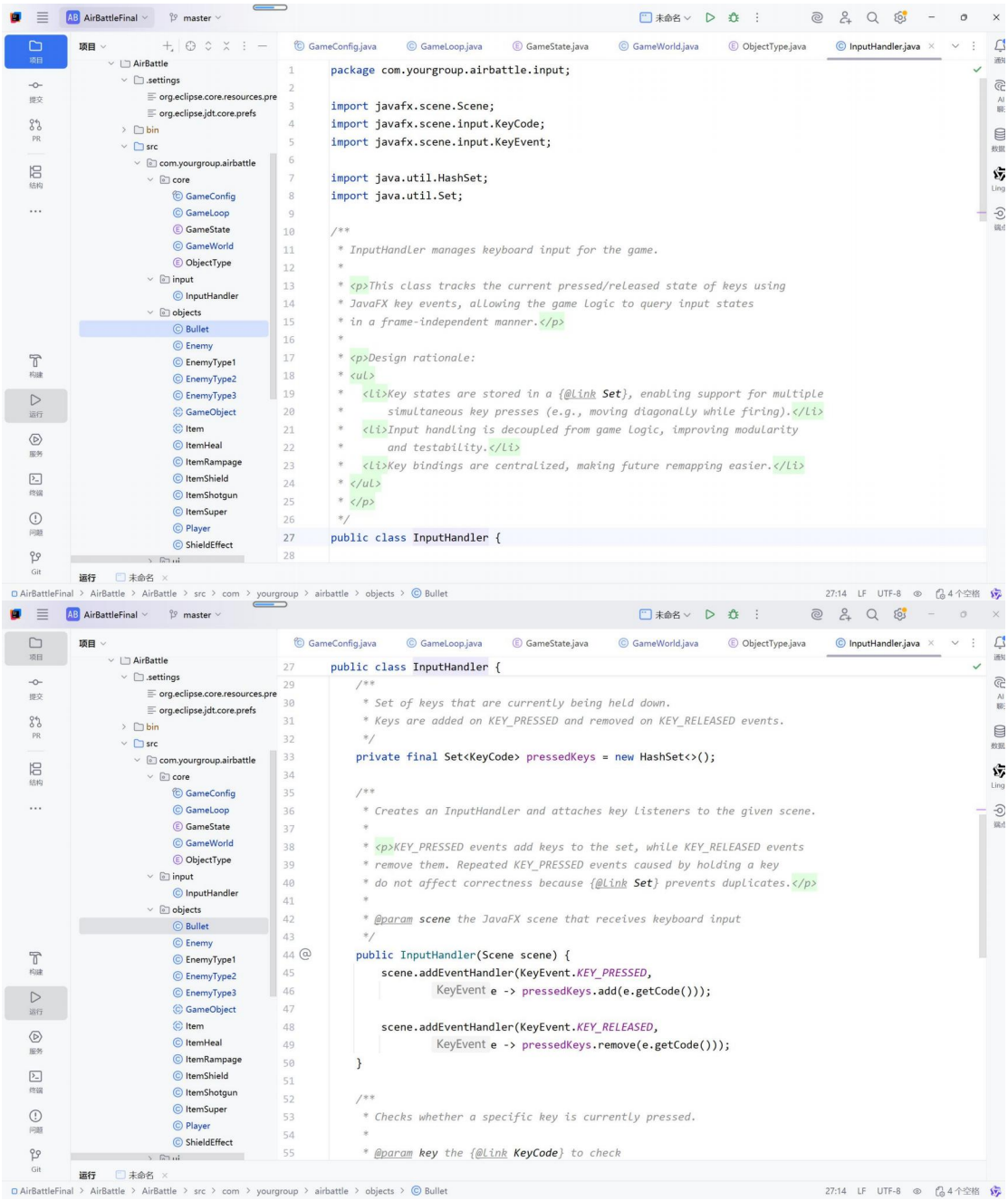


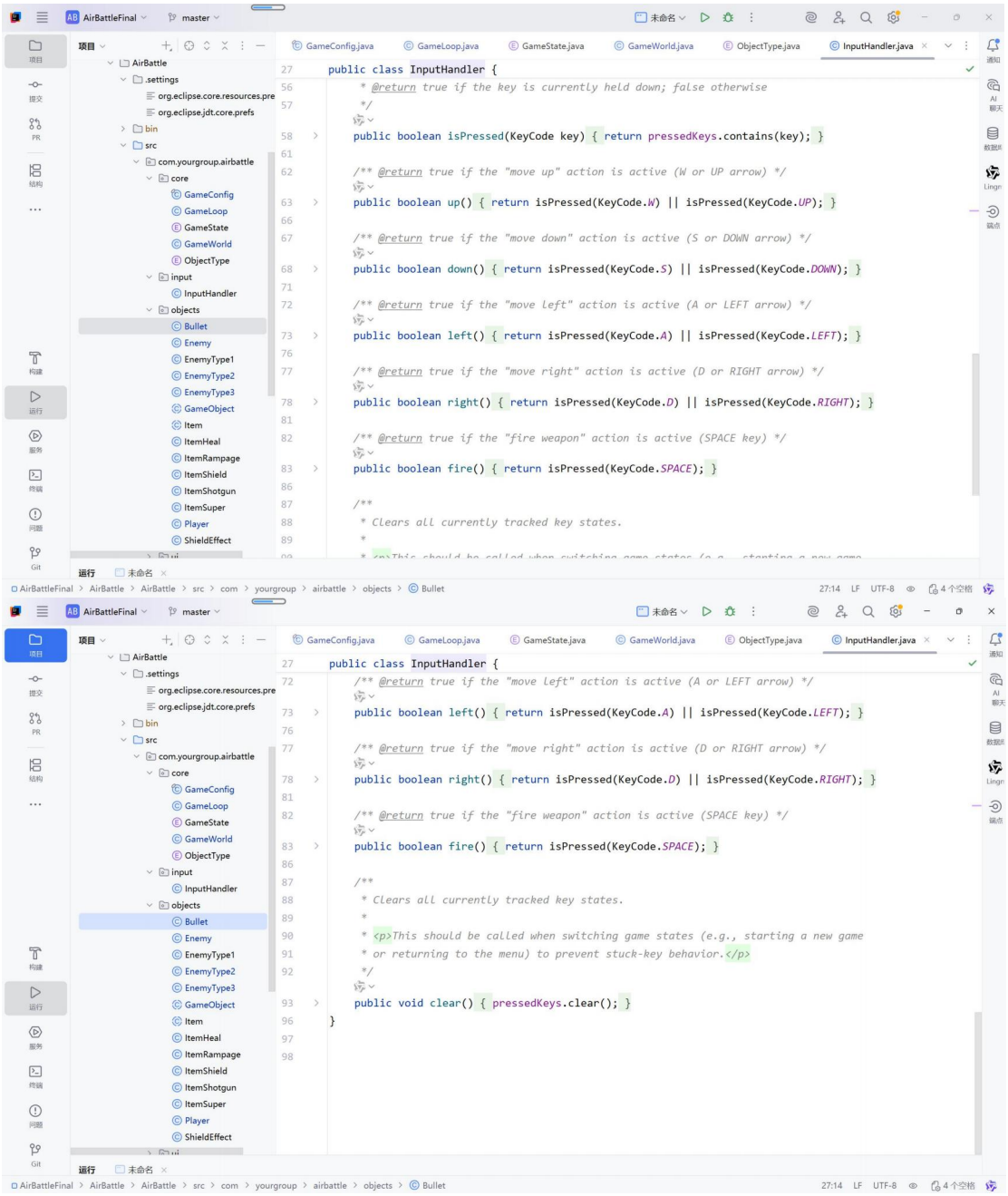
```
public class InputHandler {
    /**
     * Set of keys that are currently being held down.
     * Keys are added on KEY_PRESSED and removed on KEY_RELEASED events.
     */
    private final Set<KeyCode> pressedKeys = new HashSet<>();

    /**
     * Creates an InputHandler and attaches key listeners to the given scene.
     *
     * <p>KEY_PRESSED events add keys to the set, while KEY_RELEASED events
     * remove them. Repeated KEY_PRESSED events caused by holding a key
     * do not affect correctness because {@link Set} prevents duplicates.</p>
     *
     * @param scene the JavaFX scene that receives keyboard input
     */
    public InputHandler(Scene scene) {
        scene.addEventHandler(KeyEvent.KEY_PRESSED,
                KeyEvent e -> pressedKeys.add(e.getCode()));

        scene.addEventHandler(KeyEvent.KEY_RELEASED,
                KeyEvent e -> pressedKeys.remove(e.getCode()));
    }

    /**
     * Checks whether a specific key is currently pressed.
     *
     * @param key the {@link KeyCode} to check
     * @return true if the key is currently held down; false otherwise
     */
    public boolean isPressed(KeyCode key) { return pressedKeys.contains(key); }

    /** @return true if the "move up" action is active (W or UP arrow) */
    public boolean up() { return isPressed(KeyCode.W) || isPressed(KeyCode.UP); }

    /** @return true if the "move down" action is active (S or DOWN arrow) */
    public boolean down() { return isPressed(KeyCode.S) || isPressed(KeyCode.DOWN); }

    /** @return true if the "move left" action is active (A or LEFT arrow) */
    public boolean left() { return isPressed(KeyCode.A) || isPressed(KeyCode.LEFT); }

    /** @return true if the "move right" action is active (D or RIGHT arrow) */
    public boolean right() { return isPressed(KeyCode.D) || isPressed(KeyCode.RIGHT); }

    /** @return true if the "fire weapon" action is active (SPACE key) */
    public boolean fire() { return isPressed(KeyCode.SPACE); }

    /**
     * Clears all currently tracked key states.
     *
     * <p>This should be called when switching game states (e.g., starting a new game
```


AirBattleFinal
master
GameConfig.java
GameLoop.java
GameState.java
GameWorld.java
ObjectType.java
InputHandler.java
public class InputHandler {
/** @return true if the "move left" action is active (A or LEFT arrow) */
public boolean left() { return isPressed(KeyCode.A) || isPressed(KeyCode.LEFT); }
/** @return true if the "move right" action is active (D or RIGHT arrow) */
public boolean right() { return isPressed(KeyCode.D) || isPressed(KeyCode.RIGHT); }
/** @return true if the "fire weapon" action is active (SPACE key) */
public boolean fire() { return isPressed(KeyCode.SPACE); }
/**
* Clears all currently tracked key states.
*
* This should be called when switching game states (e.g., starting a new game
* or returning to the menu) to prevent stuck-key behavior.
*/
public void clear() { pressedKeys.clear(); }
}
libraries
lib.xml
.gitignore
AirBattleFinal.iml
misc.xml
modules.xml
vcs.xml
workspace.xml
AirBattle
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
ui
util
MainApp
img
sound
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > input > InputHandler
27:14 LF UTF-8 4 个空格

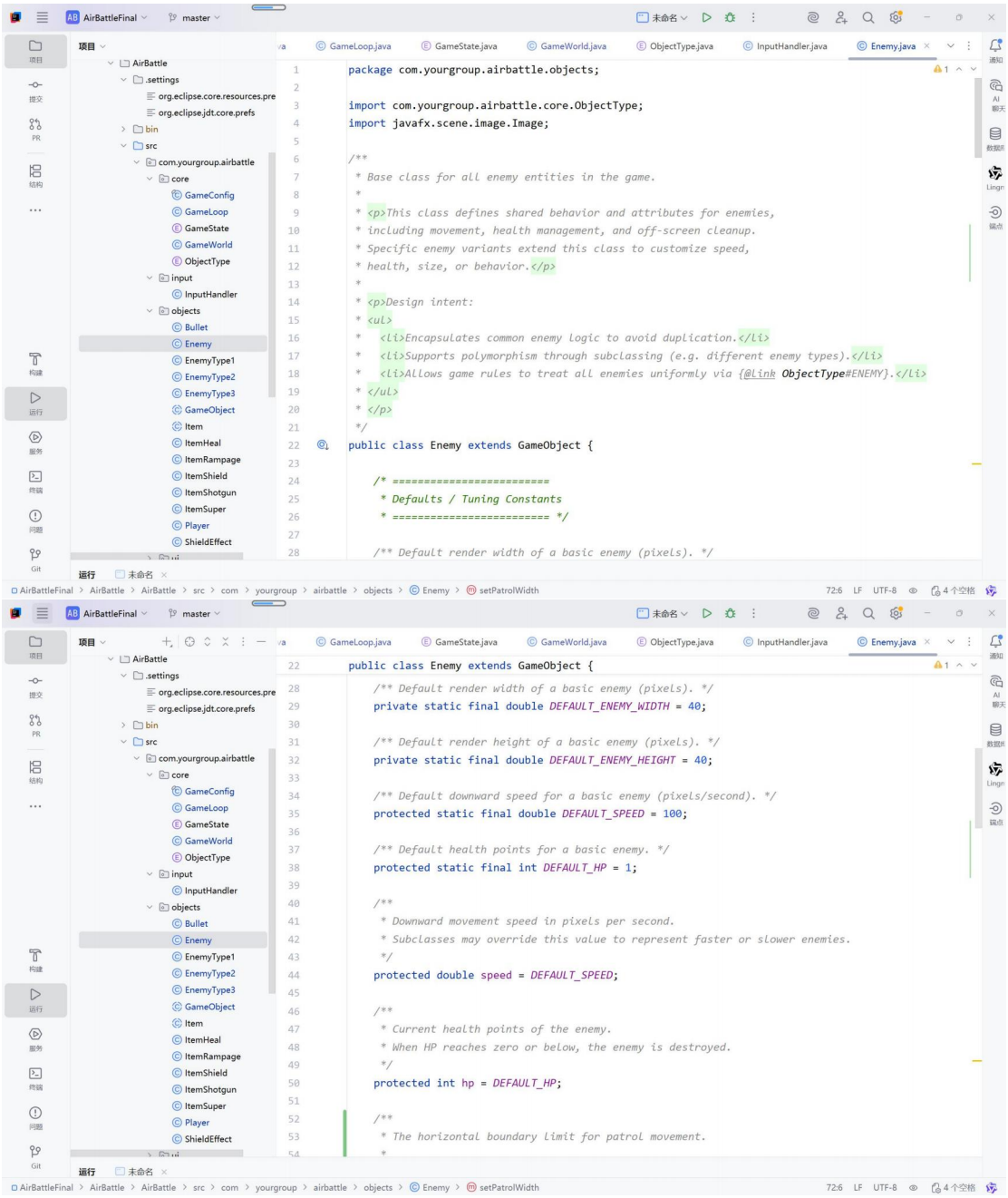
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\objects

\Bullet.java

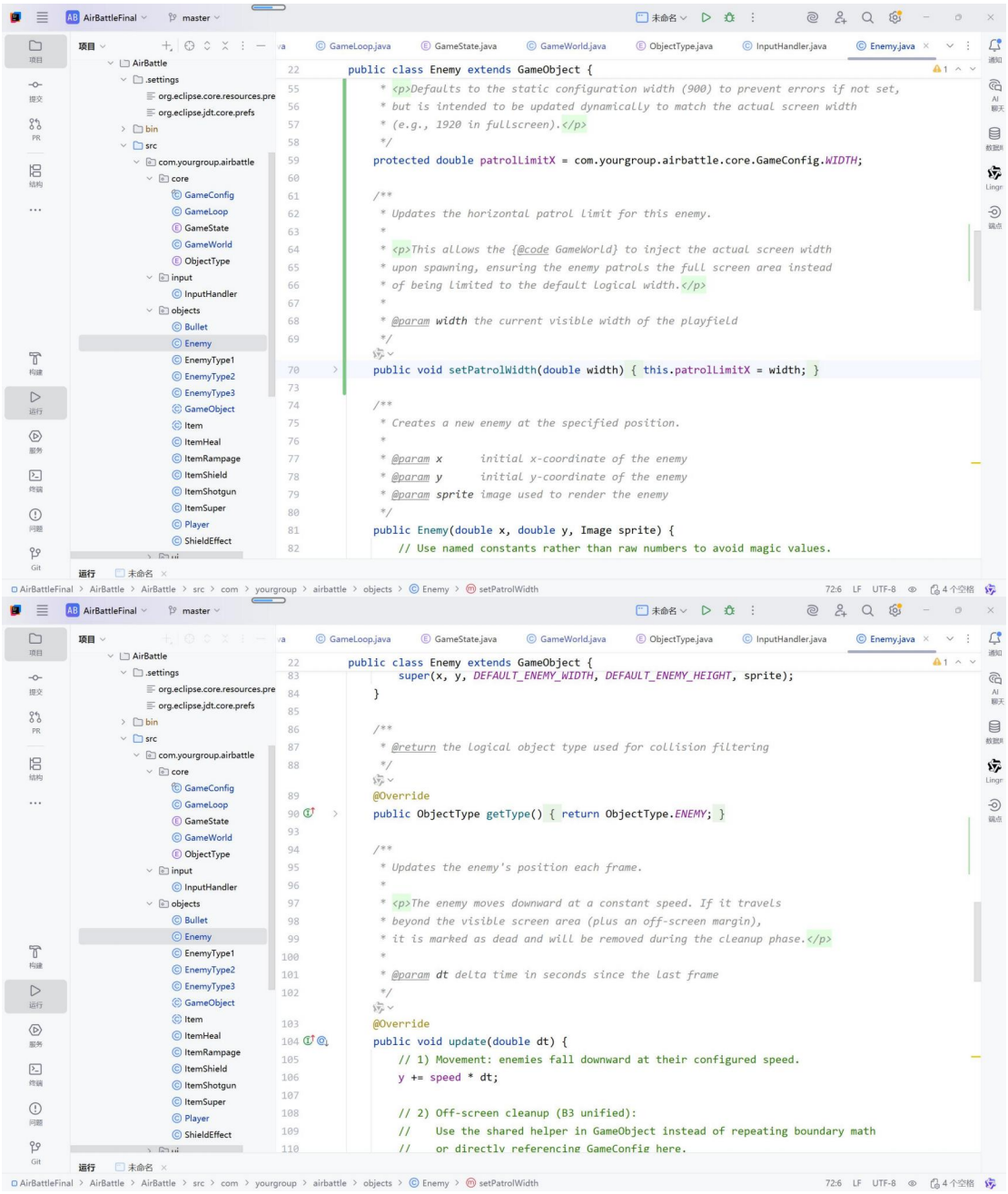


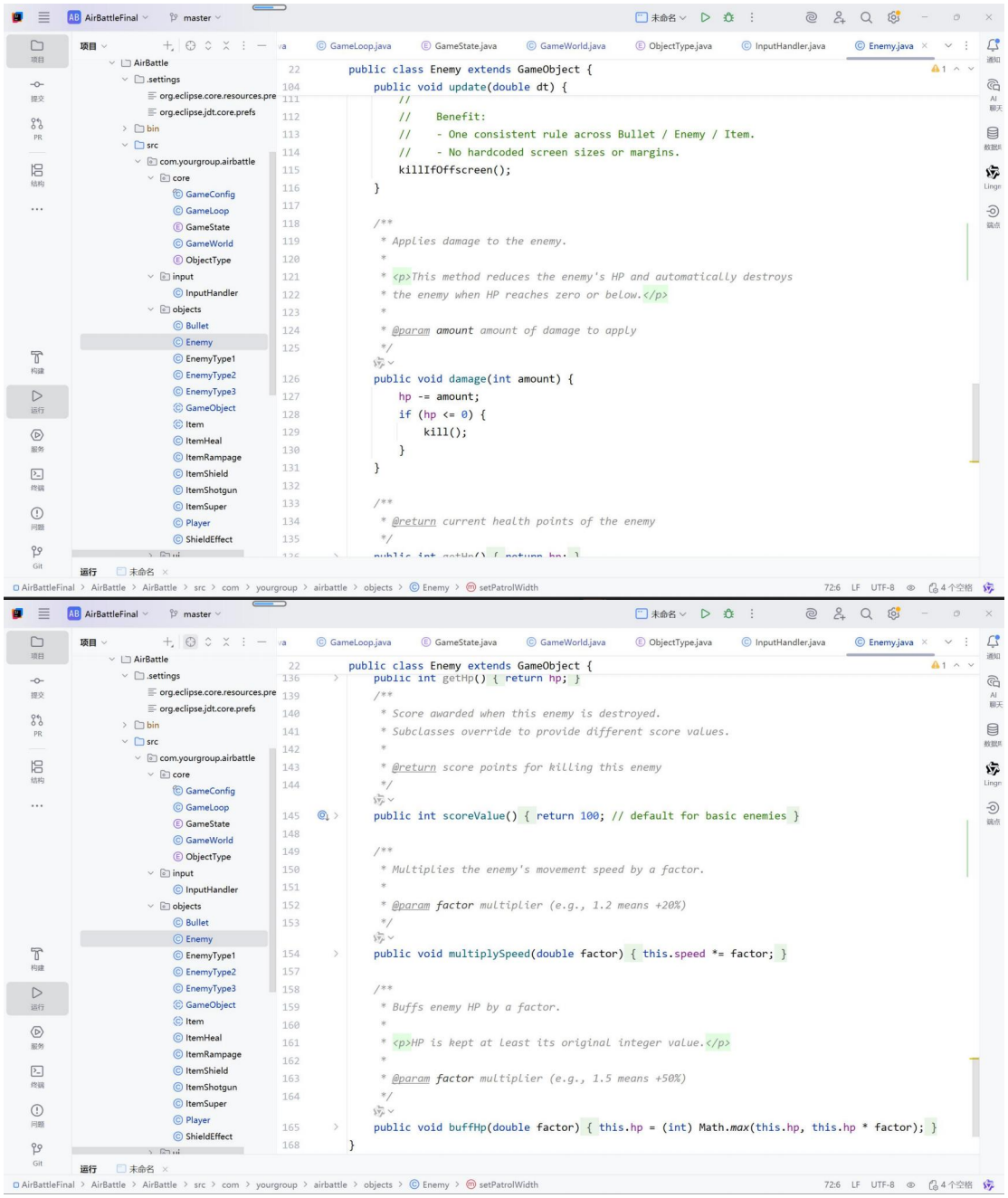

AirBattleFinal
master
GameConfig.java
GameLoop.java
GameState.java
GameWorld.java
ObjectType.java
InputHandler.java
public class InputHandler {
* @return true if the key is currently held down; false otherwise
public boolean isPressed(KeyCode key) { return pressedKeys.contains(key); }
/** @return true if the "move up" action is active (W or UP arrow) */
public boolean up() { return isPressed(KeyCode.W) || isPressed(KeyCode.UP); }
/** @return true if the "move down" action is active (S or DOWN arrow) */
public boolean down() { return isPressed(KeyCode.S) || isPressed(KeyCode.DOWN); }
/** @return true if the "move left" action is active (A or LEFT arrow) */
public boolean left() { return isPressed(KeyCode.A) || isPressed(KeyCode.LEFT); }
/** @return true if the "move right" action is active (D or RIGHT arrow) */
public boolean right() { return isPressed(KeyCode.D) || isPressed(KeyCode.RIGHT); }
/** @return true if the "fire weapon" action is active (SPACE key) */
public boolean fire() { return isPressed(KeyCode.SPACE); }
/**
* Clears all currently tracked key states.
*
* This should be called when switching game states (e.g., starting a new game
* or returning to the menu) to prevent stuck-key behavior.
*/
public void clear() { pressedKeys.clear(); }
}
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect

\Enemy.java



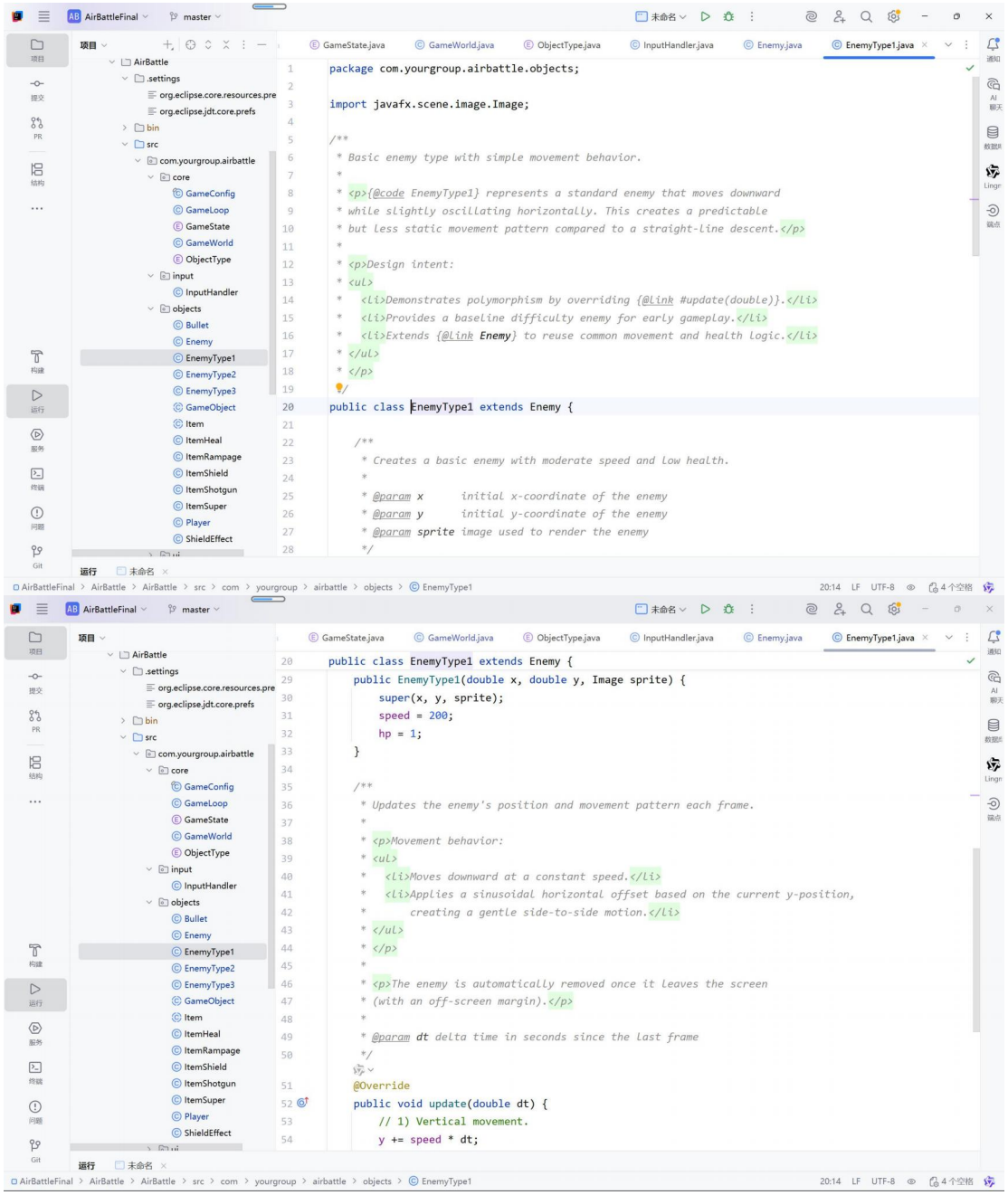
```java
package com.yourgroup.airbattle.objects;

import com.yourgroup.airbattle.core.ObjectType;
import javafx.scene.image.Image;

/**
 * Base class for all enemy entities in the game.
 *
 * <p>This class defines shared behavior and attributes for enemies,
 * including movement, health management, and off-screen cleanup.
 * Specific enemy variants extend this class to customize speed,
 * health, size, or behavior.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Encapsulates common enemy logic to avoid duplication.</li>
 *   <li>Supports polymorphism through subclassing (e.g. different enemy types).</li>
 *   <li>Allows game rules to treat all enemies uniformly via {@link ObjectType#ENEMY}.</li>
 * </ul>
 * </p>
 */
public class Enemy extends GameObject {

    /* ========================
     * Defaults / Tuning Constants
     * ======================== */

    /** Default render width of a basic enemy (pixels). */
    private static final double DEFAULT_ENEMY_WIDTH = 40;

    /** Default render height of a basic enemy (pixels). */
    private static final double DEFAULT_ENEMY_HEIGHT = 40;

    /** Default downward speed for a basic enemy (pixels/second). */
    protected static final double DEFAULT_SPEED = 100;

    /** Default health points for a basic enemy. */
    protected static final int DEFAULT_HP = 1;

    /**
     * Downward movement speed in pixels per second.
     * Subclasses may override this value to represent faster or slower enemies.
     */
    protected double speed = DEFAULT_SPEED;

    /**
     * Current health points of the enemy.
     * When HP reaches zero or below, the enemy is destroyed.
     */
    protected int hp = DEFAULT_HP;

    /**
     * The horizontal boundary limit for patrol movement.
     *
```


AirBattleFinal
master
GameLoop.java
GameState.java
GameWorld.java
ObjectType.java
InputHandler.java
Enemy.java
public class Enemy extends GameObject {
* Defaults to the static configuration width (900) to prevent errors if not set,
* but is intended to be updated dynamically to match the actual screen width
* (e.g., 1920 in fullscreen).
protected double patrolLimitX = com.yourgroup.airbattle.core.GameConfig.WIDTH;
* Updates the horizontal patrol limit for this enemy.
* This allows the {@code GameWorld} to inject the actual screen width
* upon spawning, ensuring the enemy patrols the full screen area instead
* of being limited to the default logical width.
* @param width the current visible width of the playfield
public void setPatrolWidth(double width) { this.patrolLimitX = width; }
* Creates a new enemy at the specified position.
* @param x initial x-coordinate of the enemy
* @param y initial y-coordinate of the enemy
* @param sprite image used to render the enemy
public Enemy(double x, double y, Image sprite) {
// Use named constants rather than raw numbers to avoid magic values.
super(x, y, DEFAULT_ENEMY_WIDTH, DEFAULT_ENEMY_HEIGHT, sprite);
* @return the logical object type used for collision filtering
@Override
public ObjectType getType() { return ObjectType.ENEMY; }
* Updates the enemy's position each frame.
* The enemy moves downward at a constant speed. If it travels
* beyond the visible screen area (plus an off-screen margin),
* it is marked as dead and will be removed during the cleanup phase.
* @param dt delta time in seconds since the last frame
@Override
public void update(double dt) {
// 1) Movement: enemies fall downward at their configured speed.
y += speed * dt;
// 2) Off-screen cleanup (B3 unified):
// Use the shared helper in GameObject instead of repeating boundary math
// or directly referencing GameConfig here.



```java
public class Enemy extends GameObject {
    public void update(double dt) {
        //
        //    Benefit:
        //    - One consistent rule across Bullet / Enemy / Item.
        //    - No hardcoded screen sizes or margins.
        killIfOffscreen();
    }

    /**
     * Applies damage to the enemy.
     *
     * <p>This method reduces the enemy's HP and automatically destroys
     * the enemy when HP reaches zero or below.</p>
     *
     * @param amount amount of damage to apply
     */
    public void damage(int amount) {
        hp -= amount;
        if (hp <= 0) {
            kill();
        }
    }

    /**
     * @return current health points of the enemy
     */
    public int getHp() { return hp; }
    /**
     * Score awarded when this enemy is destroyed.
     * Subclasses override to provide different score values.
     *
     * @return score points for killing this enemy
     */
    public int scoreValue() { return 100; // default for basic enemies }

    /**
     * Multiplies the enemy's movement speed by a factor.
     *
     * @param factor multiplier (e.g., 1.2 means +20%)
     */
    public void multiplySpeed(double factor) { this.speed *= factor; }

    /**
     * Buffs enemy HP by a factor.
     *
     * <p>HP is kept at least its original integer value.</p>
     *
     * @param factor multiplier (e.g., 1.5 means +50%)
     */
    public void buffHp(double factor) { this.hp = (int) Math.max(this.hp, this.hp * factor); }
}
```

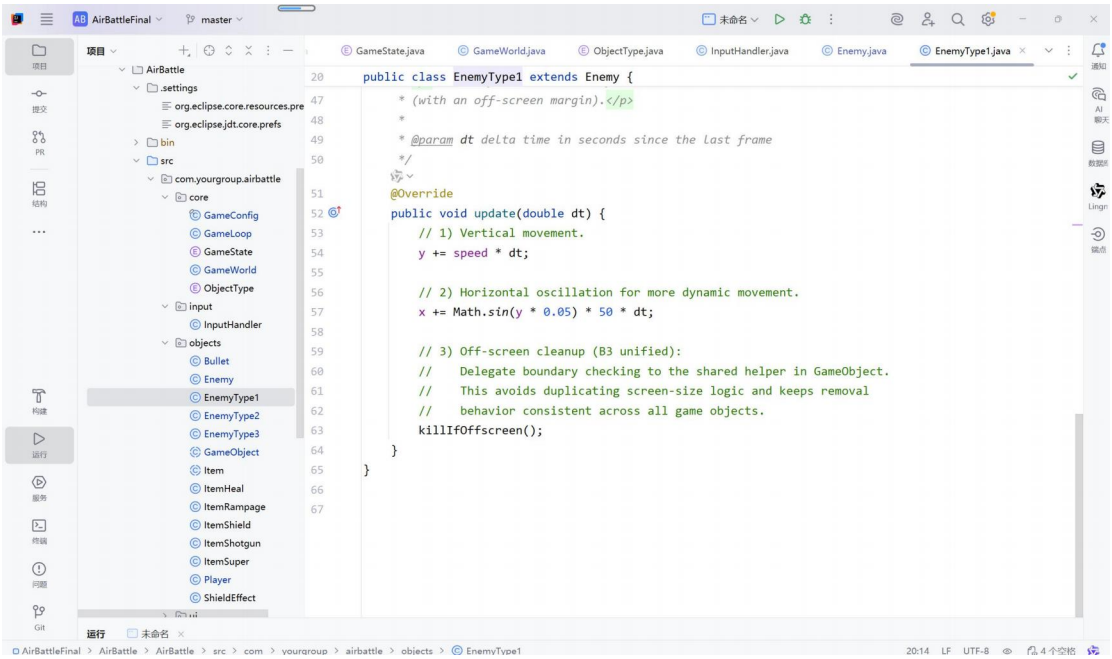
\EnemyType1.java



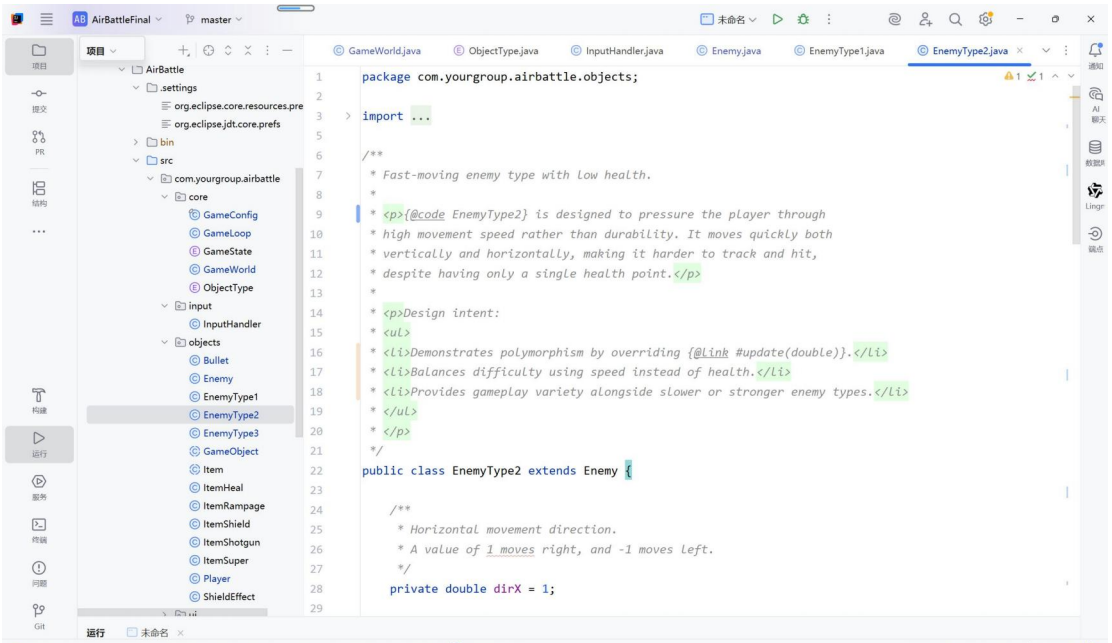
```
package com.yourgroup.airbattle.objects;

import javafx.scene.image.Image;

/**
 * Basic enemy type with simple movement behavior.
 *
 * <p>{@code EnemyType1} represents a standard enemy that moves downward
 * while slightly oscillating horizontally. This creates a predictable
 * but less static movement pattern compared to a straight-line descent.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Demonstrates polymorphism by overriding {@link #update(double)}.</li>
 *   <li>Provides a baseline difficulty enemy for early gameplay.</li>
 *   <li>Extends {@link Enemy} to reuse common movement and health logic.</li>
 * </ul>
 * </p>
 */
public class EnemyType1 extends Enemy {

    /**
     * Creates a basic enemy with moderate speed and low health.
     *
     * @param x      initial x-coordinate of the enemy
     * @param y      initial y-coordinate of the enemy
     * @param sprite image used to render the enemy
     */
```



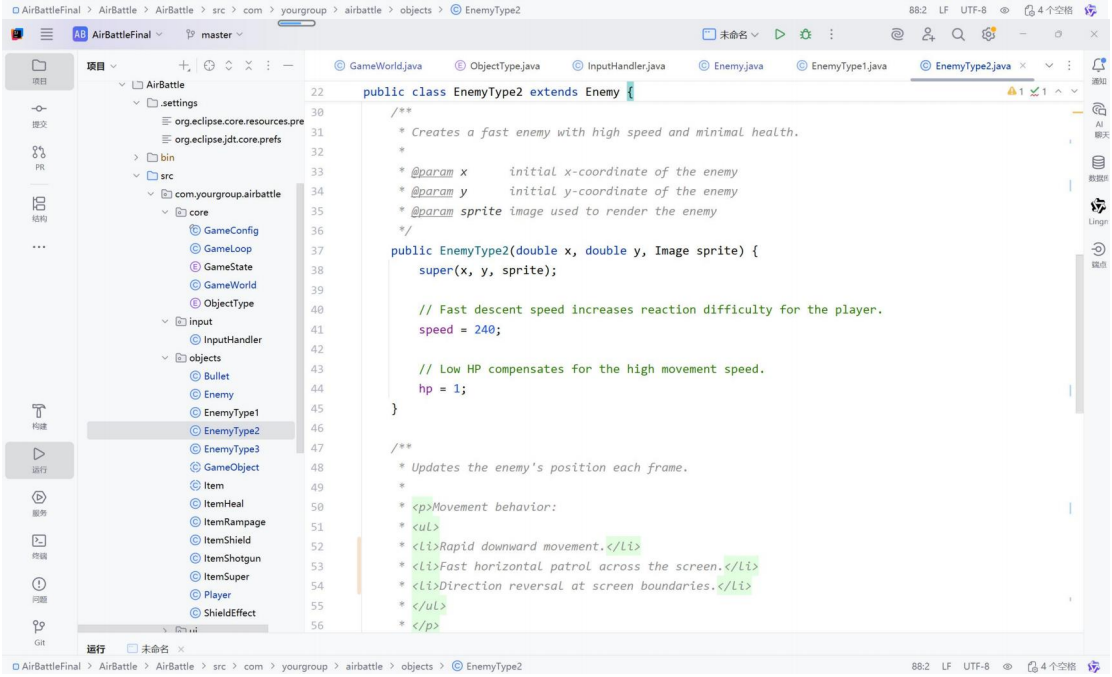
```
public class EnemyType1 extends Enemy {
    public EnemyType1(double x, double y, Image sprite) {
        super(x, y, sprite);
        speed = 200;
        hp = 1;
    }

    /**
     * Updates the enemy's position and movement pattern each frame.
     *
     * <p>Movement behavior:
     * <ul>
     *   <li>Moves downward at a constant speed.</li>
     *   <li>Applies a sinusoidal horizontal offset based on the current y-position,
     *       creating a gentle side-to-side motion.</li>
     * </ul>
     * </p>
     *
     * <p>The enemy is automatically removed once it leaves the screen
     * (with an off-screen margin).</p>
     *
     * @param dt delta time in seconds since the last frame
     */
    @Override
    public void update(double dt) {
        // 1) Vertical movement.
        y += speed * dt;
```


AirBattleFinal
master
GameState.java
GameWorld.java
ObjectType.java
InputHandler.java
Enemy.java
EnemyType1.java
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
public class EnemyType1 extends Enemy {
* (with an off-screen margin).</p>
* @param dt delta time in seconds since the last frame
*/
@Override
public void update(double dt) {
// 1) Vertical movement.
y += speed * dt;
// 2) Horizontal oscillation for more dynamic movement.
x += Math.sin(y * 0.05) * 50 * dt;
// 3) Off-screen cleanup (B3 unified):
// Delegate boundary checking to the shared helper in GameObject.
// This avoids duplicating screen-size logic and keeps removal
// behavior consistent across all game objects.
killIfOffscreen();
}
}
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > objects > EnemyType1
20:14
LF
UTF-8
4 个空格

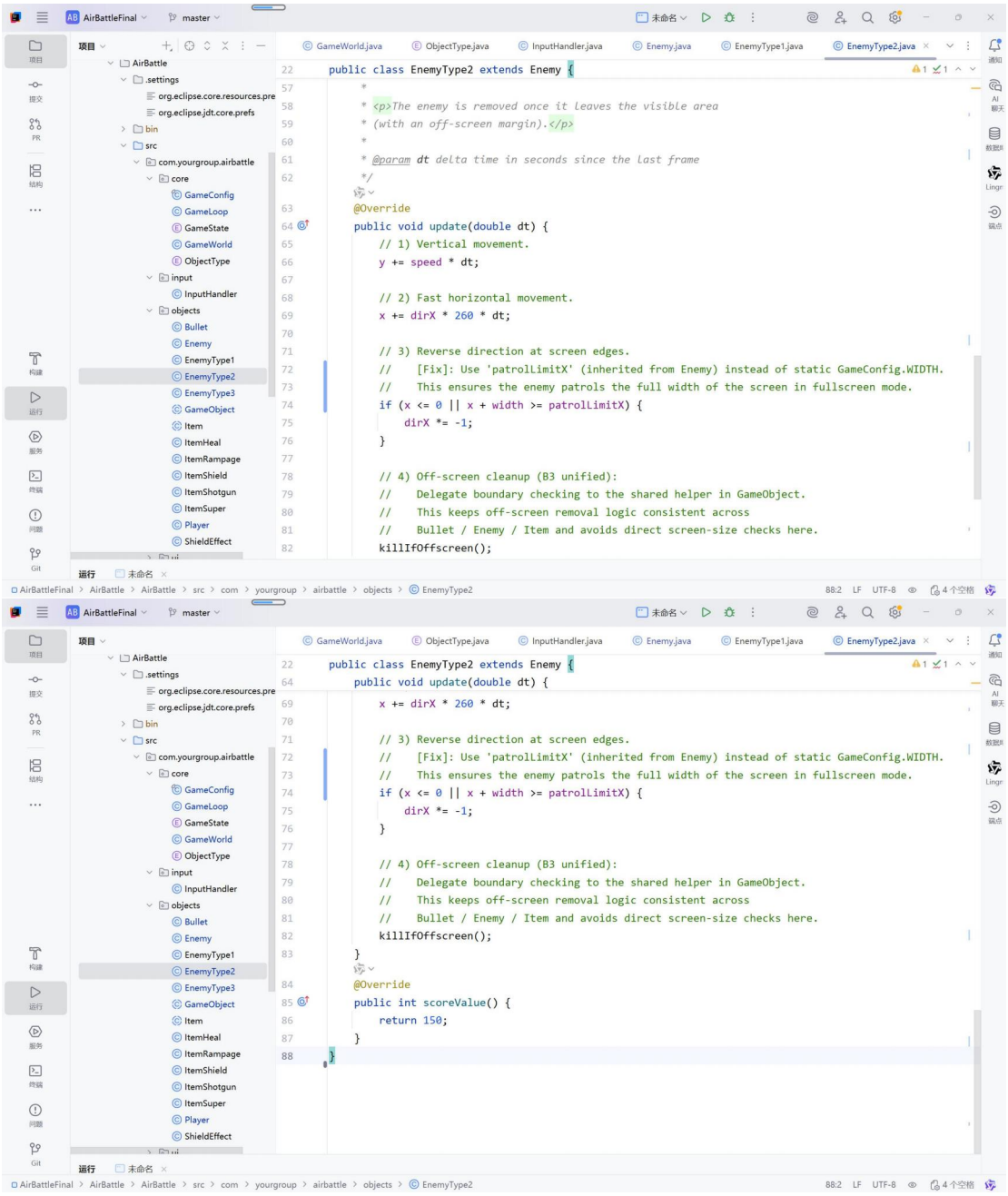
\EnemyType2.java



```
package com.yourgroup.airbattle.objects;

import ...

/**
 * Fast-moving enemy type with low health.
 *
 * <p>{@code EnemyType2} is designed to pressure the player through
 * high movement speed rather than durability. It moves quickly both
 * vertically and horizontally, making it harder to track and hit,
 * despite having only a single health point.</p>
 *
 * <p>Design intent:
 * <ul>
 * <li>Demonstrates polymorphism by overriding {@link #update(double)}.</li>
 * <li>Balances difficulty using speed instead of health.</li>
 * <li>Provides gameplay variety alongside slower or stronger enemy types.</li>
 * </ul>
 * </p>
 */
public class EnemyType2 extends Enemy {

    /**
     * Horizontal movement direction.
     * A value of 1 moves right, and -1 moves left.
     */
    private double dirX = 1;
```



```
public class EnemyType2 extends Enemy {
    /**
     * Creates a fast enemy with high speed and minimal health.
     *
     * @param x      initial x-coordinate of the enemy
     * @param y      initial y-coordinate of the enemy
     * @param sprite image used to render the enemy
     */
    public EnemyType2(double x, double y, Image sprite) {
        super(x, y, sprite);

        // Fast descent speed increases reaction difficulty for the player.
        speed = 240;

        // Low HP compensates for the high movement speed.
        hp = 1;
    }

    /**
     * Updates the enemy's position each frame.
     *
     * <p>Movement behavior:
     * <ul>
     * <li>Rapid downward movement.</li>
     * <li>Fast horizontal patrol across the screen.</li>
     * <li>Direction reversal at screen boundaries.</li>
     * </ul>
     * </p>
```

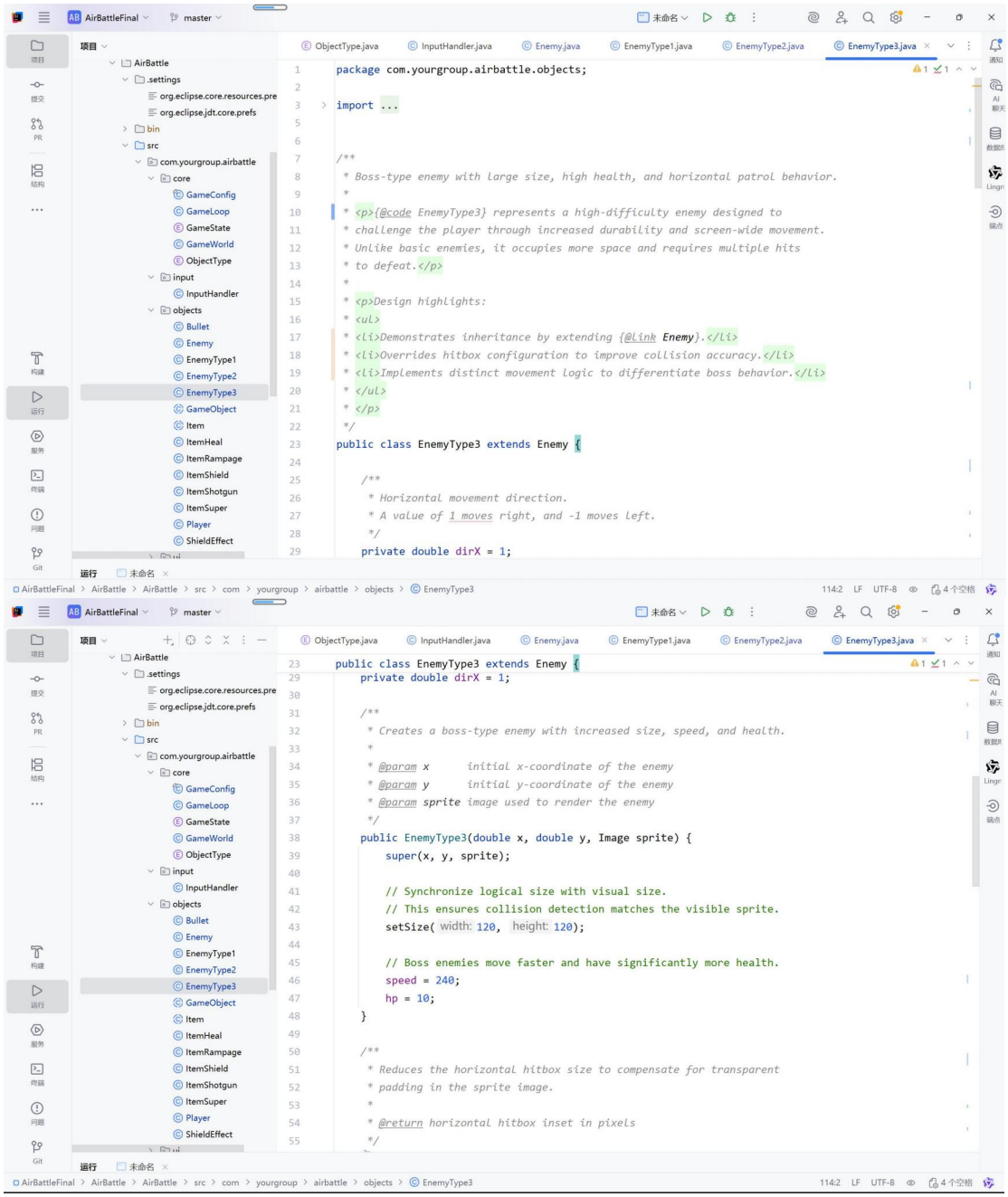


```
public class EnemyType2 extends Enemy {
    *
    * <p>The enemy is removed once it leaves the visible area
    * (with an off-screen margin).</p>
    *
    * @param dt delta time in seconds since the last frame
    */
@Override
public void update(double dt) {
    // 1) Vertical movement.
    y += speed * dt;

    // 2) Fast horizontal movement.
    x += dirX * 260 * dt;

    // 3) Reverse direction at screen edges.
    //    [Fix]: Use 'patrolLimitX' (inherited from Enemy) instead of static GameConfig.WIDTH.
    //    This ensures the enemy patrols the full width of the screen in fullscreen mode.
    if (x <= 0 || x + width >= patrolLimitX) {
        dirX *= -1;
    }

    // 4) Off-screen cleanup (B3 unified):
    //    Delegate boundary checking to the shared helper in GameObject.
    //    This keeps off-screen removal logic consistent across
    //    Bullet / Enemy / Item and avoids direct screen-size checks here.
    killIfOffscreen();
```



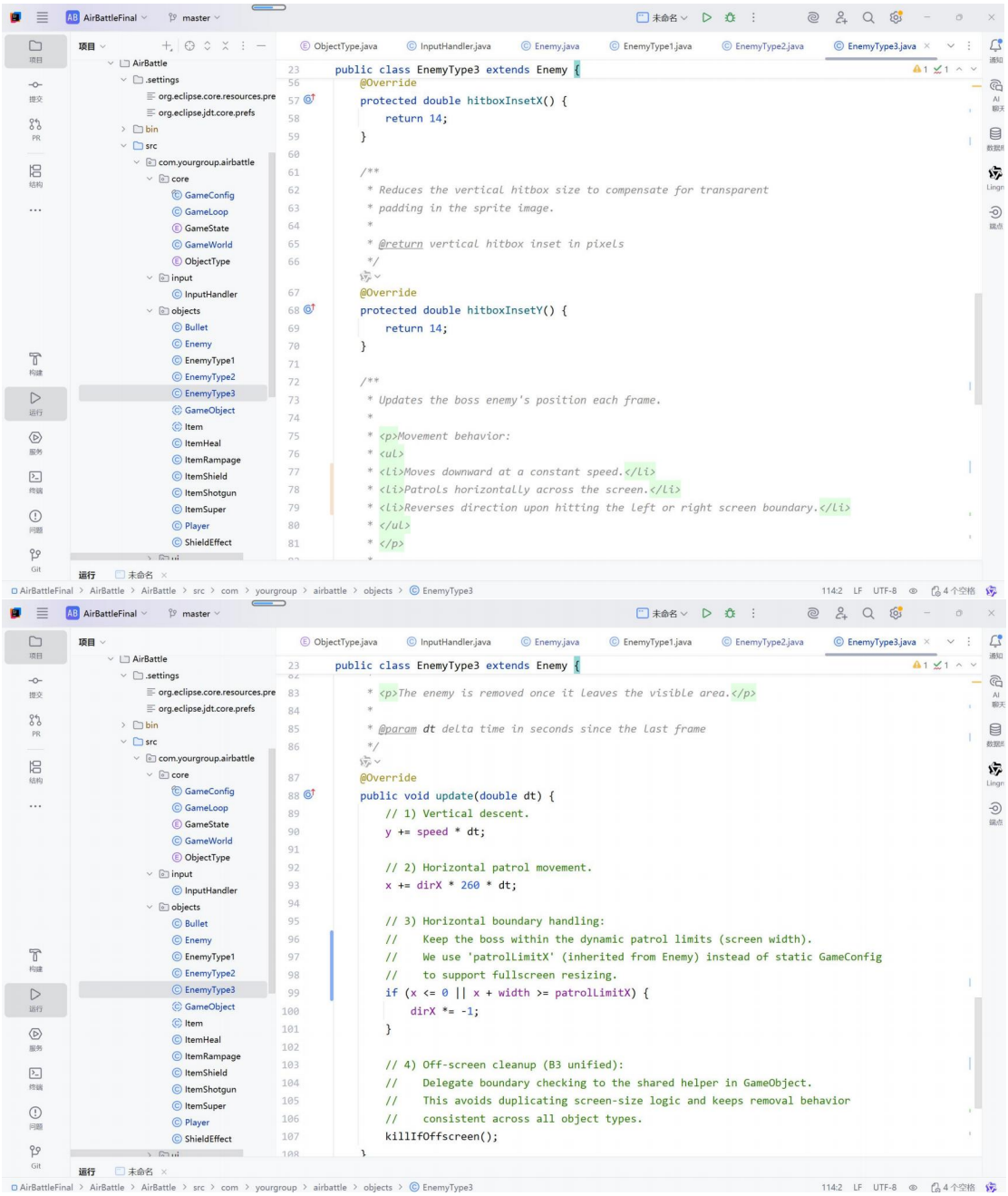
```
public class EnemyType2 extends Enemy {
    public void update(double dt) {
        x += dirX * 260 * dt;

        // 3) Reverse direction at screen edges.
        //    [Fix]: Use 'patrolLimitX' (inherited from Enemy) instead of static GameConfig.WIDTH.
        //    This ensures the enemy patrols the full width of the screen in fullscreen mode.
        if (x <= 0 || x + width >= patrolLimitX) {
            dirX *= -1;
        }

        // 4) Off-screen cleanup (B3 unified):
        //    Delegate boundary checking to the shared helper in GameObject.
        //    This keeps off-screen removal logic consistent across
        //    Bullet / Enemy / Item and avoids direct screen-size checks here.
        killIfOffscreen();
    }
    @Override
    public int scoreValue() {
        return 150;
    }
}
```

\EnemyType3.java

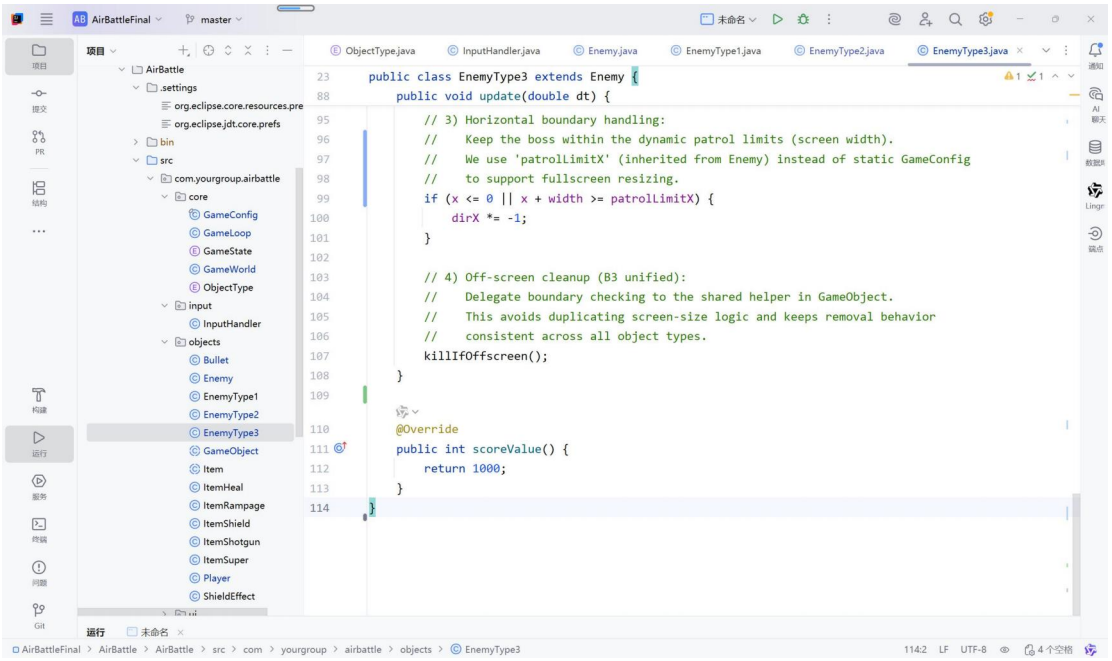


```
package com.yourgroup.airbattle.objects;

import ...


/**
 * Boss-type enemy with large size, high health, and horizontal patrol behavior.
 *
 * <p>{@code EnemyType3} represents a high-difficulty enemy designed to
 * challenge the player through increased durability and screen-wide movement.
 * Unlike basic enemies, it occupies more space and requires multiple hits
 * to defeat.</p>
 *
 * <p>Design highlights:
 * <ul>
 * <li>Demonstrates inheritance by extending {@link Enemy}.</li>
 * <li>Overrides hitbox configuration to improve collision accuracy.</li>
 * <li>Implements distinct movement logic to differentiate boss behavior.</li>
 * </ul>
 * </p>
 */
public class EnemyType3 extends Enemy {

    /**
     * Horizontal movement direction.
     * A value of 1 moves right, and -1 moves left.
     */
    private double dirX = 1;
```

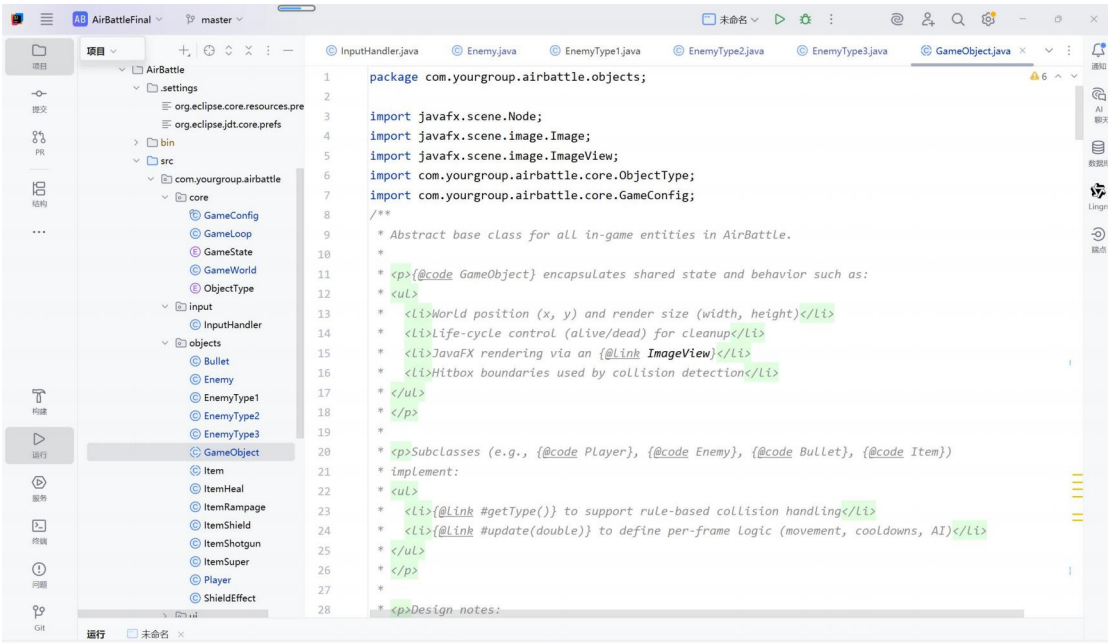
```
public class EnemyType3 extends Enemy {
    private double dirX = 1;

    /**
     * Creates a boss-type enemy with increased size, speed, and health.
     *
     * @param x      initial x-coordinate of the enemy
     * @param y      initial y-coordinate of the enemy
     * @param sprite image used to render the enemy
     */
    public EnemyType3(double x, double y, Image sprite) {
        super(x, y, sprite);

        // Synchronize logical size with visual size.
        // This ensures collision detection matches the visible sprite.
        setSize( width: 120,  height: 120);

        // Boss enemies move faster and have significantly more health.
        speed = 240;
        hp = 10;
    }

    /**
     * Reduces the horizontal hitbox size to compensate for transparent
     * padding in the sprite image.
     *
     * @return horizontal hitbox inset in pixels
     */
```



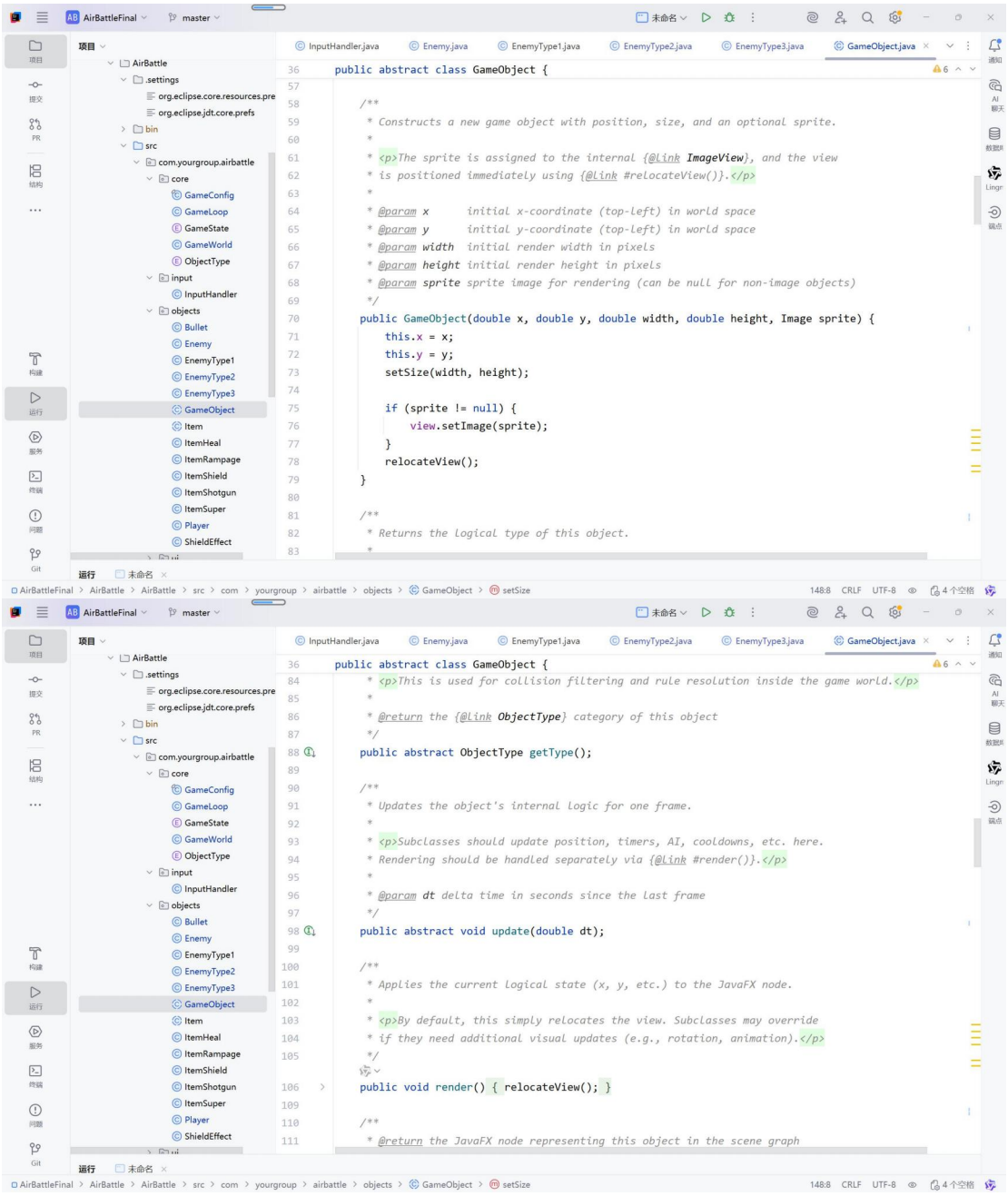
```java
public class EnemyType3 extends Enemy {
    @Override
    protected double hitboxInsetX() {
        return 14;
    }

    /**
     * Reduces the vertical hitbox size to compensate for transparent
     * padding in the sprite image.
     *
     * @return vertical hitbox inset in pixels
     */
    @Override
    protected double hitboxInsetY() {
        return 14;
    }

    /**
     * Updates the boss enemy's position each frame.
     *
     * <p>Movement behavior:
     * <ul>
     * <li>Moves downward at a constant speed.</li>
     * <li>Patrols horizontally across the screen.</li>
     * <li>Reverses direction upon hitting the left or right screen boundary.</li>
     * </ul>
     * </p>
     *
     * <p>The enemy is removed once it leaves the visible area.</p>
     *
     * @param dt delta time in seconds since the last frame
     */
    @Override
    public void update(double dt) {
        // 1) Vertical descent.
        y += speed * dt;

        // 2) Horizontal patrol movement.
        x += dirX * 260 * dt;

        // 3) Horizontal boundary handling:
        //    Keep the boss within the dynamic patrol limits (screen width).
        //    We use 'patrolLimitX' (inherited from Enemy) instead of static GameConfig
        //    to support fullscreen resizing.
        if (x <= 0 || x + width >= patrolLimitX) {
            dirX *= -1;
        }

        // 4) Off-screen cleanup (B3 unified):
        //    Delegate boundary checking to the shared helper in GameObject.
        //    This avoids duplicating screen-size logic and keeps removal behavior
        //    consistent across all object types.
        killIfOffscreen();
    }
```


AirBattleFinal
master
未命名
ObjectType.java
InputHandler.java
Enemy.java
EnemyType1.java
EnemyType2.java
EnemyType3.java
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
public class EnemyType3 extends Enemy {
public void update(double dt) {
// 3) Horizontal boundary handling:
// Keep the boss within the dynamic patrol limits (screen width).
// We use 'patrolLimitX' (inherited from Enemy) instead of static GameConfig
// to support fullscreen resizing.
if (x <= 0 || x + width >= patrolLimitX) {
dirX *= -1;
}
// 4) Off-screen cleanup (B3 unified):
// Delegate boundary checking to the shared helper in GameObject.
// This avoids duplicating screen-size logic and keeps removal behavior
// consistent across all object types.
killIfOffscreen();
}
@Override
public int scoreValue() {
return 1000;
}
}
运行
未命名
AirBattleFinal › AirBattle › AirBattle › src › com › yourgroup › airbattle › objects › EnemyType3
114:2
LF
UTF-8
4 个空格

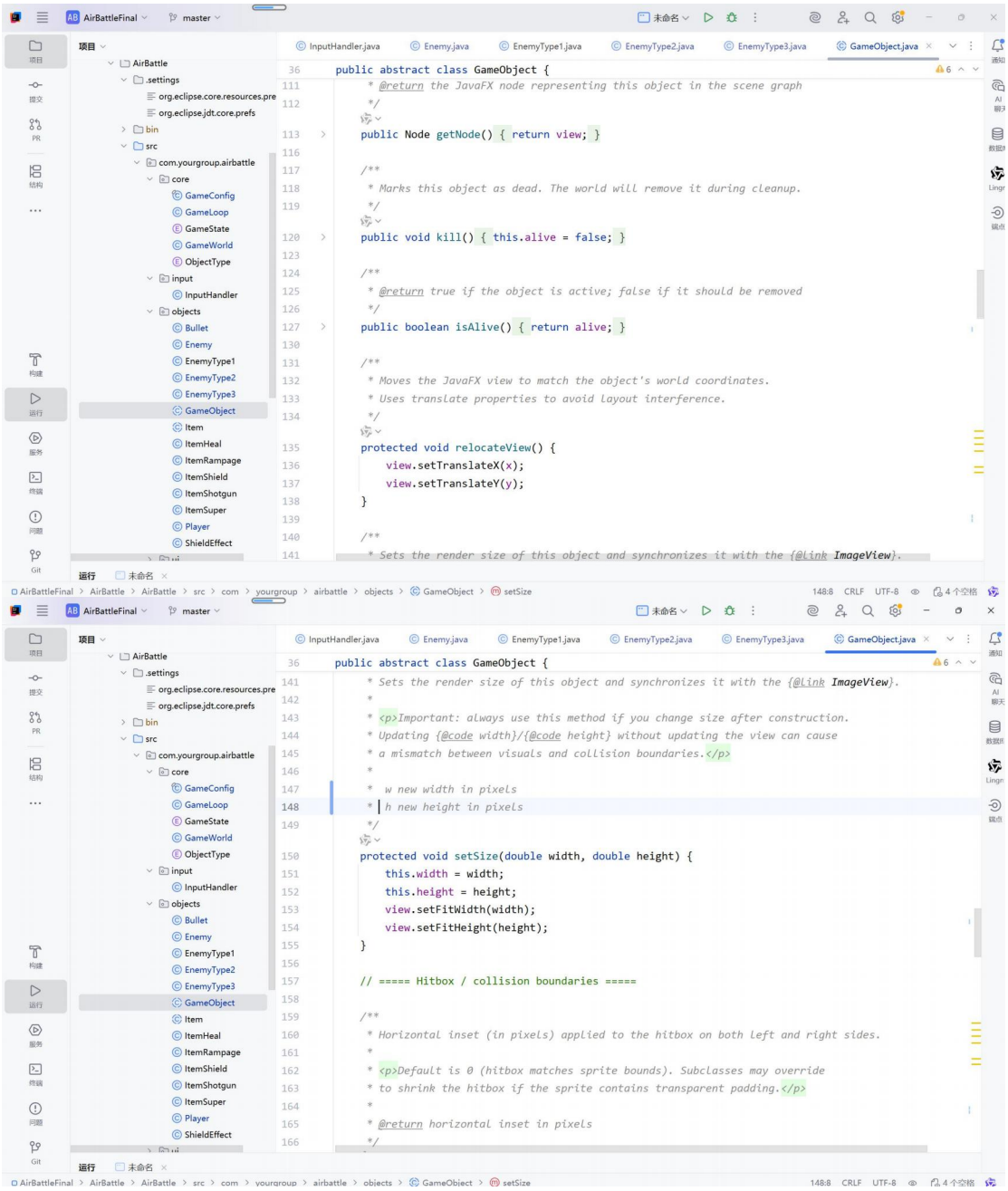
\GameObject.java

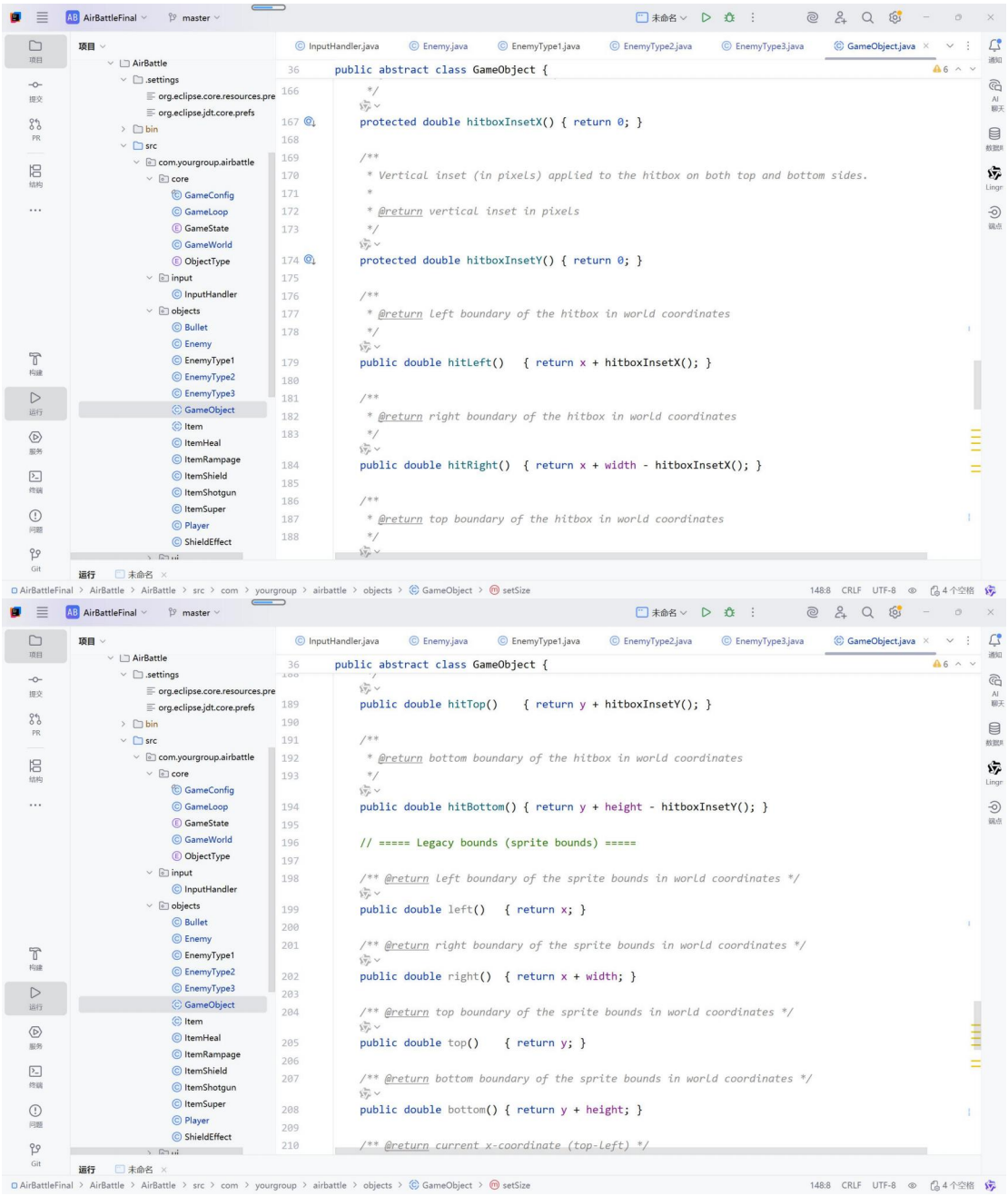


```java
package com.yourgroup.airbattle.objects;

import javafx.scene.Node;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import com.yourgroup.airbattle.core.ObjectType;
import com.yourgroup.airbattle.core.GameConfig;
/**
 * Abstract base class for all in-game entities in AirBattle.
 *
 * <p>{@code GameObject} encapsulates shared state and behavior such as:
 * <ul>
 *   <li>World position (x, y) and render size (width, height)</li>
 *   <li>Life-cycle control (alive/dead) for cleanup</li>
 *   <li>JavaFX rendering via an {@link ImageView}</li>
 *   <li>Hitbox boundaries used by collision detection</li>
 * </ul>
 * </p>
 *
 * <p>Subclasses (e.g., {@code Player}, {@code Enemy}, {@code Bullet}, {@code Item})
 * implement:
 * <ul>
 *   <li>{@link #getType()} to support rule-based collision handling</li>
 *   <li>{@link #update(double)} to define per-frame logic (movement, cooldowns, AI)</li>
 * </ul>
 * </p>
 *
 * <p>Design notes:
 * <ul>
 *   <li>Rendering is separated from logic: {@link #update(double)} modifies state,
 *       {@link #render()} applies state to the JavaFX node.</li>
 *   <li>Hitbox inset hooks allow sprites with transparent padding to have more accurate collisions.</li>
 * </ul>
 * </p>
 */
public abstract class GameObject {

    /** World position (top-left corner) in pixels. */
    protected double x;
    protected double y;

    /** Render size in pixels. */
    protected double width;
    protected double height;

    /**
     * Life-cycle flag.
     * When false, the object will be removed from the world during cleanup.
     */
    protected boolean alive = true;

    /**
     * JavaFX node used for rendering (default: {@link ImageView}).
     * The world will add/remove this node to/from the playfield.
     */
    protected final ImageView view = new ImageView();
```

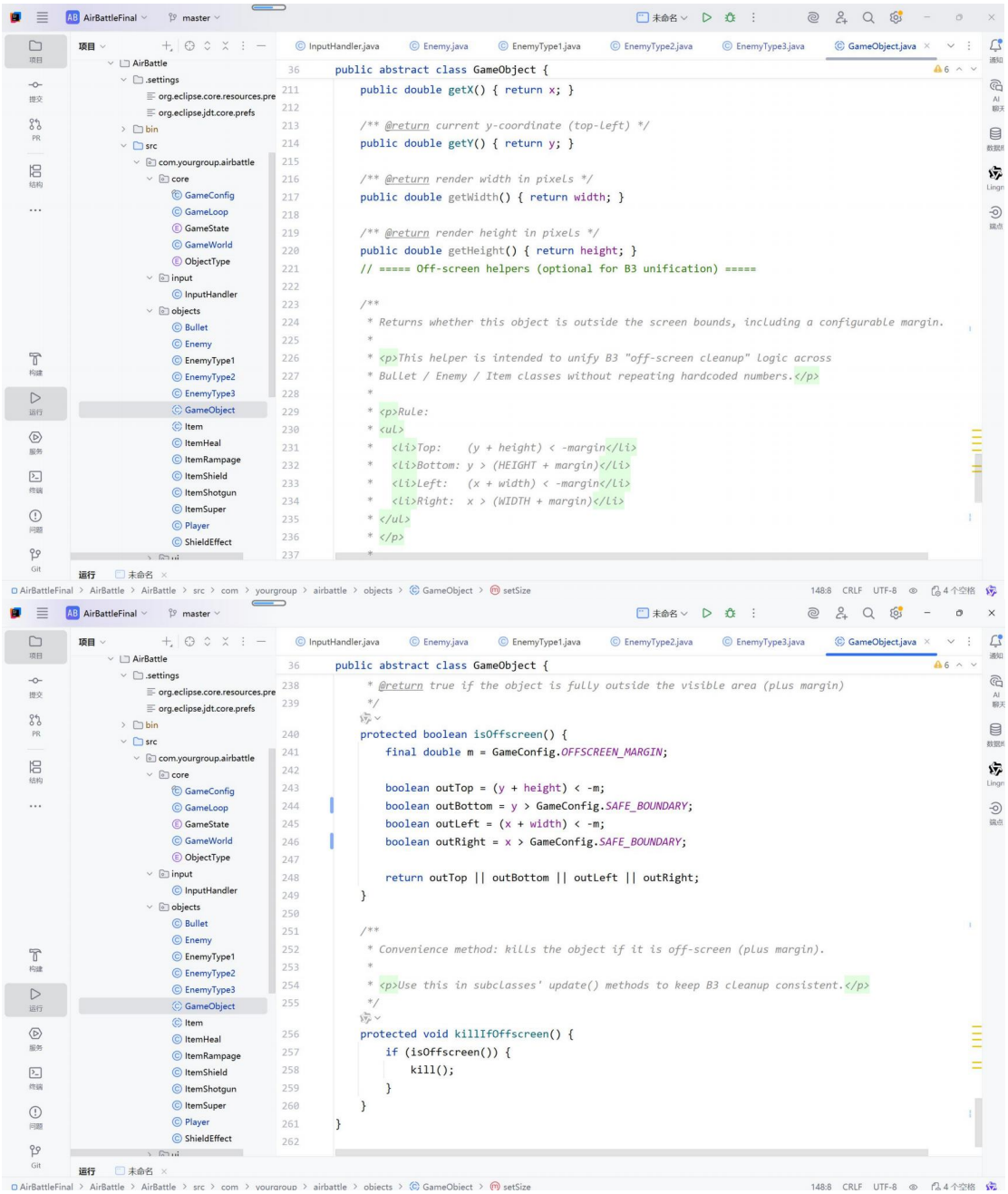


```
public abstract class GameObject {

    /**
     * Constructs a new game object with position, size, and an optional sprite.
     *
     * <p>The sprite is assigned to the internal {@link ImageView}, and the view
     * is positioned immediately using {@link #relocateView()}.</p>
     *
     * @param x      initial x-coordinate (top-left) in world space
     * @param y      initial y-coordinate (top-left) in world space
     * @param width  initial render width in pixels
     * @param height initial render height in pixels
     * @param sprite sprite image for rendering (can be null for non-image objects)
     */
    public GameObject(double x, double y, double width, double height, Image sprite) {
        this.x = x;
        this.y = y;
        setSize(width, height);

        if (sprite != null) {
            view.setImage(sprite);
        }
        relocateView();
    }

    /**
     * Returns the logical type of this object.
     *
     * <p>This is used for collision filtering and rule resolution inside the game world.</p>
     *
     * @return the {@link ObjectType} category of this object
     */
    public abstract ObjectType getType();

    /**
     * Updates the object's internal logic for one frame.
     *
     * <p>Subclasses should update position, timers, AI, cooldowns, etc. here.
     * Rendering should be handled separately via {@link #render()}.</p>
     *
     * @param dt delta time in seconds since the last frame
     */
    public abstract void update(double dt);

    /**
     * Applies the current logical state (x, y, etc.) to the JavaFX node.
     *
     * <p>By default, this simply relocates the view. Subclasses may override
     * if they need additional visual updates (e.g., rotation, animation).</p>
     */
    public void render() { relocateView(); }

    /**
     * @return the JavaFX node representing this object in the scene graph
```

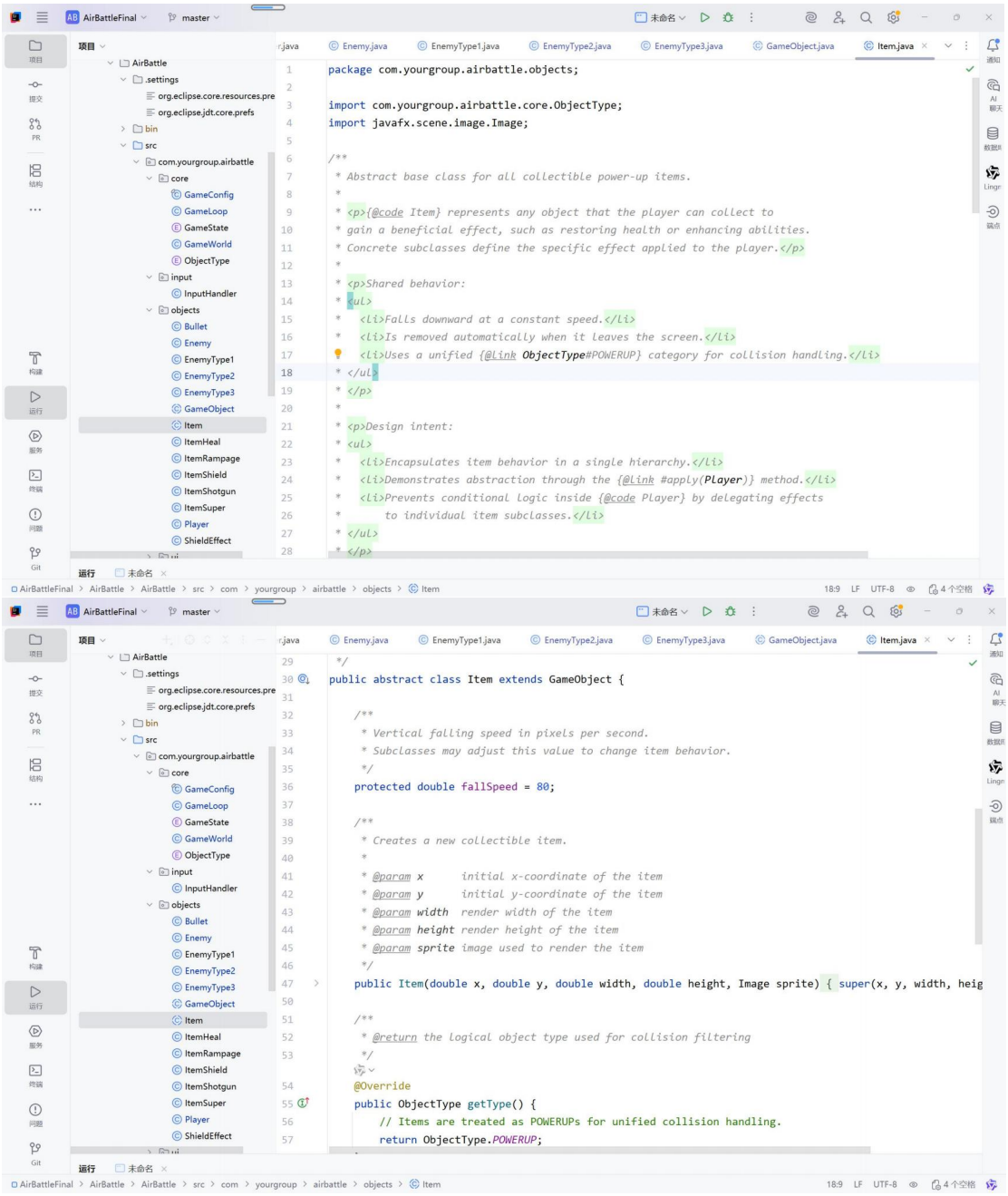

AirBattleFinal
master
InputHandler.java
Enemy.java
EnemyType1.java
EnemyType2.java
EnemyType3.java
GameObject.java
public abstract class GameObject {
* @return the JavaFX node representing this object in the scene graph
public Node getNode() { return view; }
* Marks this object as dead. The world will remove it during cleanup.
public void kill() { this.alive = false; }
* @return true if the object is active; false if it should be removed
public boolean isAlive() { return alive; }
* Moves the JavaFX view to match the object's world coordinates.
* Uses translate properties to avoid layout interference.
protected void relocateView() {
view.setTranslateX(x);
view.setTranslateY(y);
}
* Sets the render size of this object and synchronizes it with the {@link ImageView}.
* Important: always use this method if you change size after construction.
* Updating {@code width}/{@code height} without updating the view can cause
* a mismatch between visuals and collision boundaries.
* w new width in pixels
* h new height in pixels
protected void setSize(double width, double height) {
this.width = width;
this.height = height;
view.setFitWidth(width);
view.setFitHeight(height);
}
// ===== Hitbox / collision boundaries =====
* Horizontal inset (in pixels) applied to the hitbox on both left and right sides.
* Default is 0 (hitbox matches sprite bounds). Subclasses may override
* to shrink the hitbox if the sprite contains transparent padding.
* @return horizontal inset in pixels
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect


AirBattleFinal
master
未命名
InputHandler.java
Enemy.java
EnemyType1.java
EnemyType2.java
EnemyType3.java
GameObject.java
项目
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
public abstract class GameObject {
protected double hitboxInsetX() { return 0; }
protected double hitboxInsetY() { return 0; }
public double hitLeft() { return x + hitboxInsetX(); }
public double hitRight() { return x + width - hitboxInsetX(); }
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > objects > GameObject > setSize
148:8 CRLF UTF-8 4 个空格

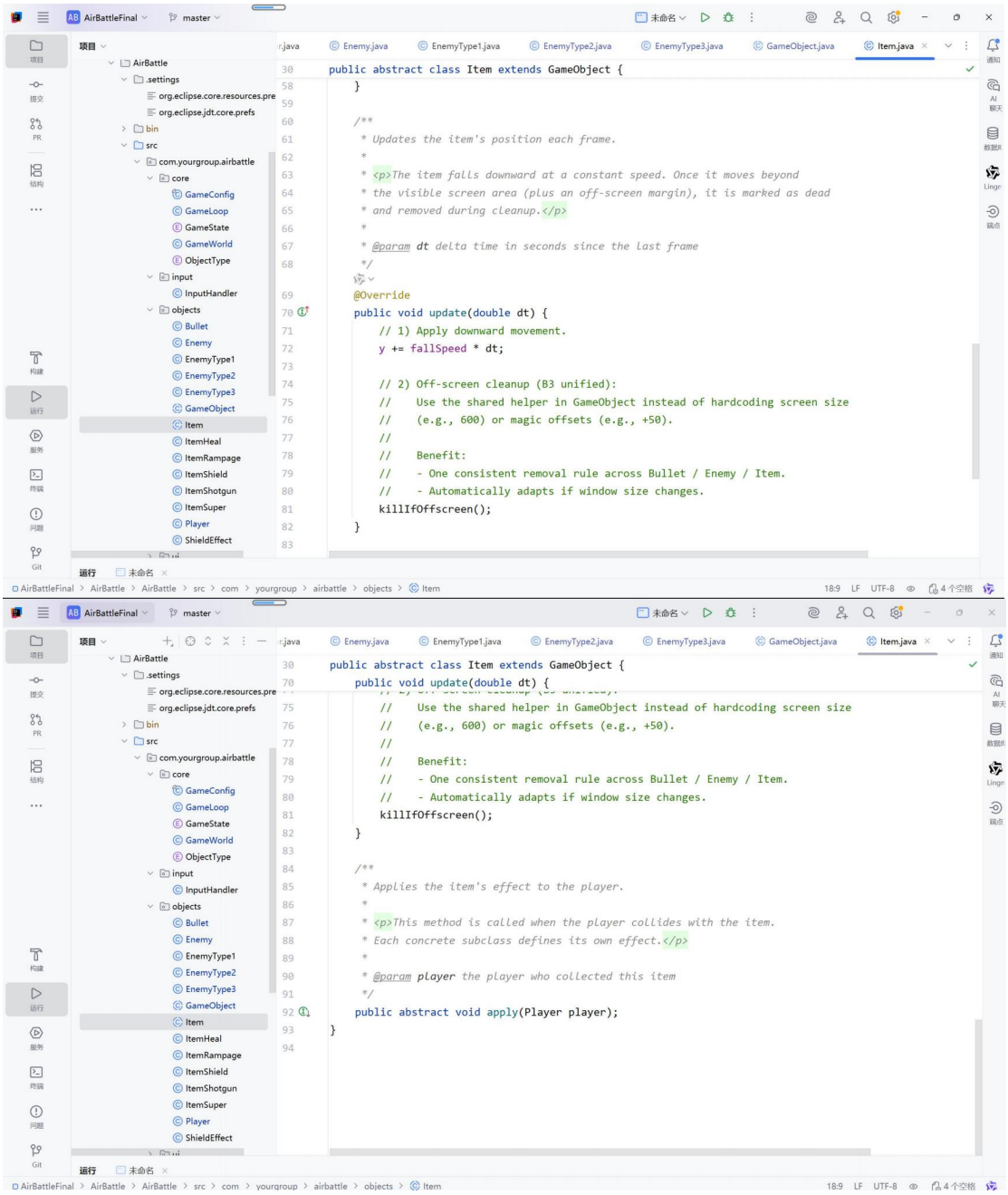
AirBattleFinal
master
未命名
InputHandler.java
Enemy.java
EnemyType1.java
EnemyType2.java
EnemyType3.java
GameObject.java
项目
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
public abstract class GameObject {
public double hitTop() { return y + hitboxInsetY(); }
public double hitBottom() { return y + height - hitboxInsetY(); }
// ===== Legacy bounds (sprite bounds) =====
public double left() { return x; }
public double right() { return x + width; }
public double top() { return y; }
public double bottom() { return y + height; }
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > objects > GameObject > setSize
148:8 CRLF UTF-8 4 个空格


AirBattleFinal
master
未命名
项目
AirBattle
.settings
org.eclipse.core.resources.pre
org.eclipse.jdt.core.prefs
bin
src
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
InputHandler.java
Enemy.java
EnemyType1.java
EnemyType2.java
EnemyType3.java
GameObject.java
public abstract class GameObject {
public double getX() { return x; }
/** @return current y-coordinate (top-left) */
public double getY() { return y; }
/** @return render width in pixels */
public double getWidth() { return width; }
/** @return render height in pixels */
public double getHeight() { return height; }
// ===== Off-screen helpers (optional for B3 unification) =====
/**
* Returns whether this object is outside the screen bounds, including a configurable margin.
*
* This helper is intended to unify B3 "off-screen cleanup" logic across
* Bullet / Enemy / Item classes without repeating hardcoded numbers.
*
* Rule:
*
* Top: (y + height) < -margin
* Bottom: y > (HEIGHT + margin)
* Left: (x + width) < -margin
* Right: x > (WIDTH + margin)
*
*
* @return true if the object is fully outside the visible area (plus margin)
*/
protected boolean isOffscreen() {
final double m = GameConfig.OFFSCREEN_MARGIN;
boolean outTop = (y + height) < -m;
boolean outBottom = y > GameConfig.SAFE_BOUNDARY;
boolean outLeft = (x + width) < -m;
boolean outRight = x > GameConfig.SAFE_BOUNDARY;
return outTop || outBottom || outLeft || outRight;
}
/**
* Convenience method: kills the object if it is off-screen (plus margin).
*
* Use this in subclasses' update() methods to keep B3 cleanup consistent.
*/
protected void killIfOffscreen() {
if (isOffscreen()) {
kill();
}
}
}
运行
未命名
148:8
CRLF
UTF-8
4 个空格

\Item.java



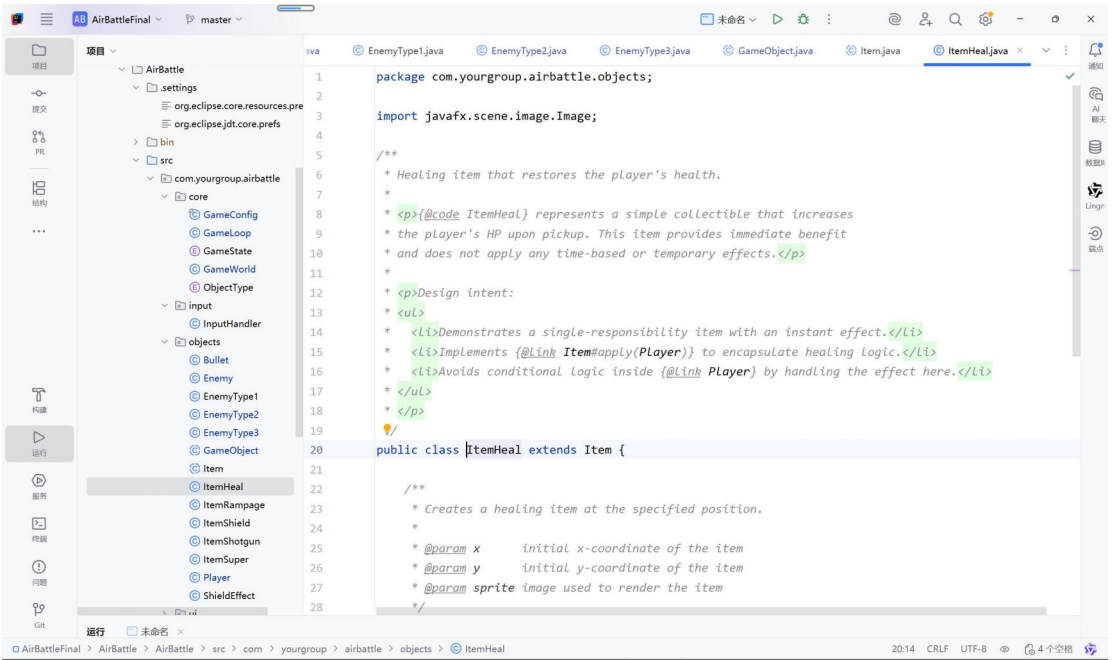
```java
package com.yourgroup.airbattle.objects;

import com.yourgroup.airbattle.core.ObjectType;
import javafx.scene.image.Image;

/**
 * Abstract base class for all collectible power-up items.
 *
 * <p>{@code Item} represents any object that the player can collect to
 * gain a beneficial effect, such as restoring health or enhancing abilities.
 * Concrete subclasses define the specific effect applied to the player.</p>
 *
 * <p>Shared behavior:
 * <ul>
 *   <li>Falls downward at a constant speed.</li>
 *   <li>Is removed automatically when it leaves the screen.</li>
 *   <li>Uses a unified {@link ObjectType#POWERUP} category for collision handling.</li>
 * </ul>
 * </p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Encapsulates item behavior in a single hierarchy.</li>
 *   <li>Demonstrates abstraction through the {@link #apply(Player)} method.</li>
 *   <li>Prevents conditional logic inside {@code Player} by delegating effects
 *       to individual item subclasses.</li>
 * </ul>
 * </p>
 */
public abstract class Item extends GameObject {

    /**
     * Vertical falling speed in pixels per second.
     * Subclasses may adjust this value to change item behavior.
     */
    protected double fallSpeed = 80;

    /**
     * Creates a new collectible item.
     *
     * @param x      initial x-coordinate of the item
     * @param y      initial y-coordinate of the item
     * @param width  render width of the item
     * @param height render height of the item
     * @param sprite image used to render the item
     */
    public Item(double x, double y, double width, double height, Image sprite) { super(x, y, width, heig

    /**
     * @return the logical object type used for collision filtering
     */
    @Override
    public ObjectType getType() {
        // Items are treated as POWERUPs for unified collision handling.
        return ObjectType.POWERUP;
```



```java
public abstract class Item extends GameObject {
    }

    /**
     * Updates the item's position each frame.
     *
     * <p>The item falls downward at a constant speed. Once it moves beyond
     * the visible screen area (plus an off-screen margin), it is marked as dead
     * and removed during cleanup.</p>
     *
     * @param dt delta time in seconds since the last frame
     */
    @Override
    public void update(double dt) {
        // 1) Apply downward movement.
        y += fallSpeed * dt;

        // 2) Off-screen cleanup (B3 unified):
        //    Use the shared helper in GameObject instead of hardcoding screen size
        //    (e.g., 600) or magic offsets (e.g., +50).
        //
        //    Benefit:
        //    - One consistent removal rule across Bullet / Enemy / Item.
        //    - Automatically adapts if window size changes.
        killIfOffscreen();
    }
```

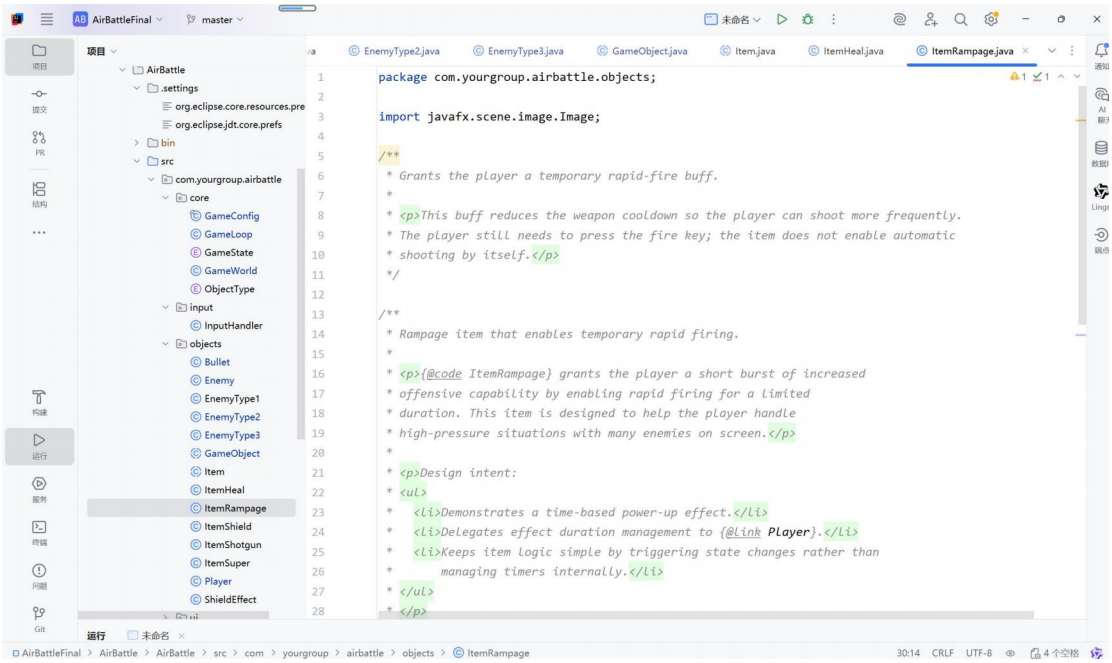
```java
public abstract class Item extends GameObject {
    public void update(double dt) {
        //    Use the shared helper in GameObject instead of hardcoding screen size
        //    (e.g., 600) or magic offsets (e.g., +50).
        //
        //    Benefit:
        //    - One consistent removal rule across Bullet / Enemy / Item.
        //    - Automatically adapts if window size changes.
        killIfOffscreen();
    }

    /**
     * Applies the item's effect to the player.
     *
     * <p>This method is called when the player collides with the item.
     * Each concrete subclass defines its own effect.</p>
     *
     * @param player the player who collected this item
     */
    public abstract void apply(Player player);
}
```

\ItemHeal.java

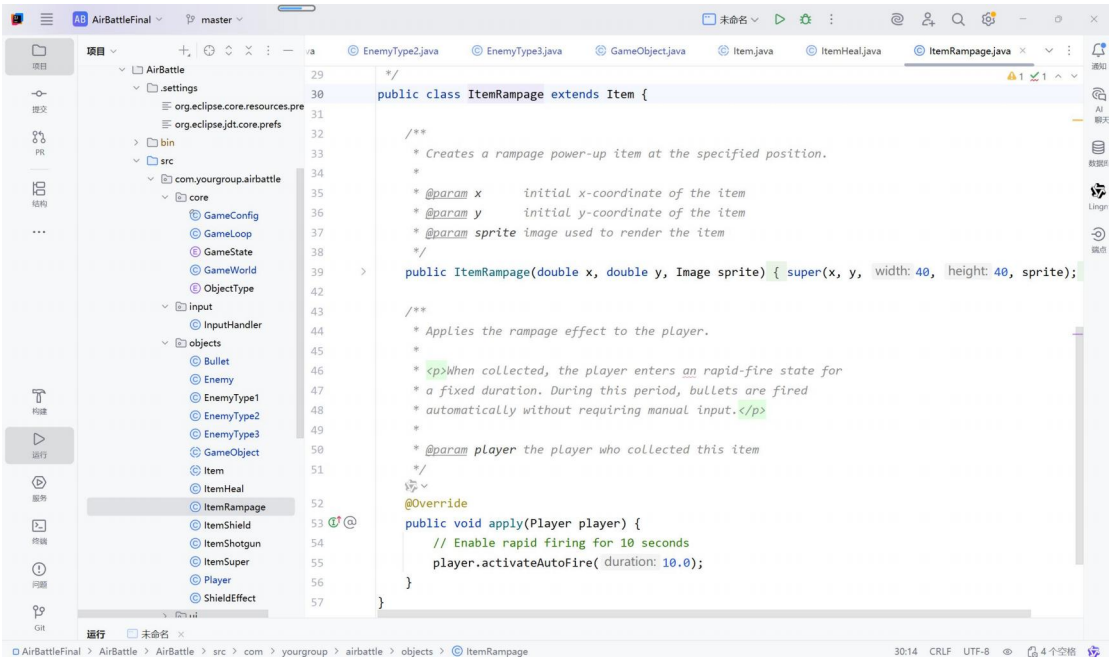


```
package com.yourgroup.airbattle.objects;

import javafx.scene.image.Image;

/**
 * Healing item that restores the player's health.
 *
 * <p>{@code ItemHeal} represents a simple collectible that increases
 * the player's HP upon pickup. This item provides immediate benefit
 * and does not apply any time-based or temporary effects.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Demonstrates a single-responsibility item with an instant effect.</li>
 *   <li>Implements {@link Item#apply(Player)} to encapsulate healing logic.</li>
 *   <li>Avoids conditional logic inside {@link Player} by handling the effect here.</li>
 * </ul>
 * </p>
 */
public class ItemHeal extends Item {

    /**
     * Creates a healing item at the specified position.
     *
     * @param x      initial x-coordinate of the item
     * @param y      initial y-coordinate of the item
     * @param sprite image used to render the item
     */
    public ItemHeal(double x, double y, Image sprite) { super(x, y, width: 40, height: 40, sprite); }

    /**
     * Applies the healing effect to the player.
     *
     * <p>This method is called when the player collides with the item.
     * The player's health is increased by a fixed amount.</p>
     *
     * @param player the player who collected this item
     */
    @Override
    public void apply(Player player) {
        player.heal( amount: 1);

        // Optional: play a sound effect to indicate item pickup
        // com.yourgroup.airbattle.util.SoundManager.playItemGet();
    }
}
```

\ItemRampage.java

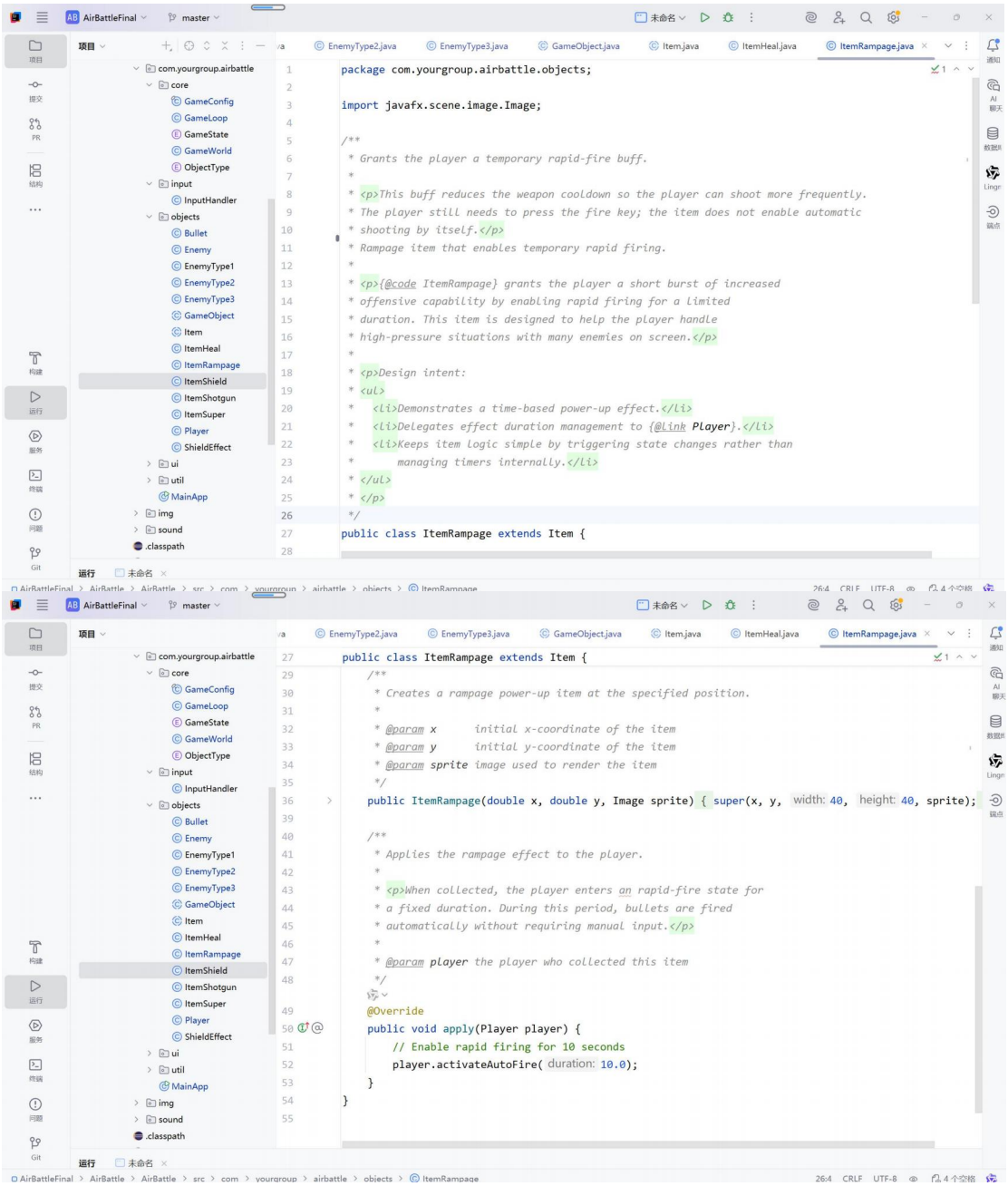


```java
package com.yourgroup.airbattle.objects;

import javafx.scene.image.Image;

/**
 * Grants the player a temporary rapid-fire buff.
 *
 * <p>This buff reduces the weapon cooldown so the player can shoot more frequently.
 * The player still needs to press the fire key; the item does not enable automatic
 * shooting by itself.</p>
 */

/**
 * Rampage item that enables temporary rapid firing.
 *
 * <p>{@code ItemRampage} grants the player a short burst of increased
 * offensive capability by enabling rapid firing for a limited
 * duration. This item is designed to help the player handle
 * high-pressure situations with many enemies on screen.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Demonstrates a time-based power-up effect.</li>
 *   <li>Delegates effect duration management to {@link Player}.</li>
 *   <li>Keeps item logic simple by triggering state changes rather than
 *       managing timers internally.</li>
 * </ul>
 * </p>
```



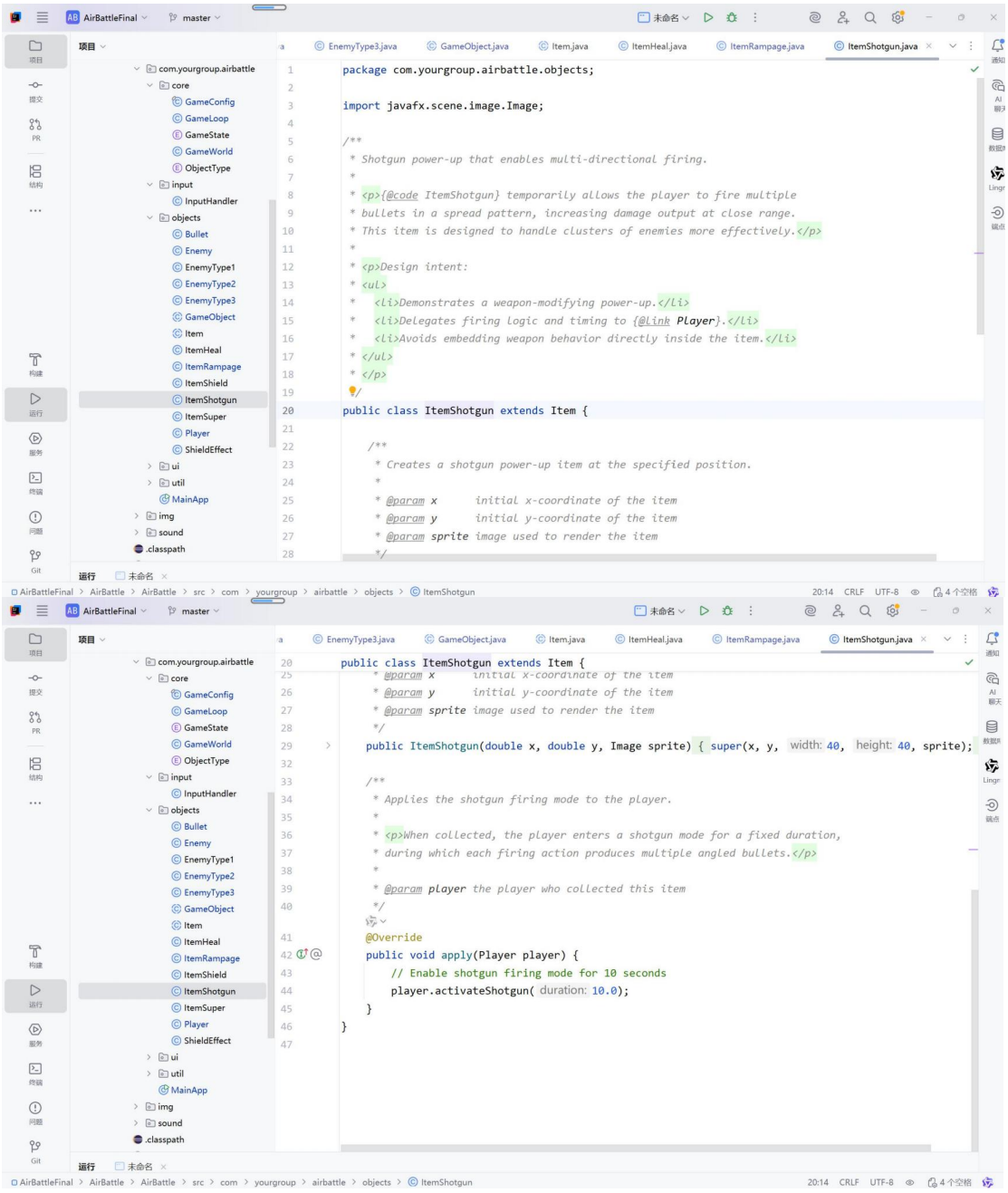
```java
 */
public class ItemRampage extends Item {

    /**
     * Creates a rampage power-up item at the specified position.
     *
     * @param x      initial x-coordinate of the item
     * @param y      initial y-coordinate of the item
     * @param sprite image used to render the item
     */
    public ItemRampage(double x, double y, Image sprite) { super(x, y, width: 40, height: 40, sprite);

    /**
     * Applies the rampage effect to the player.
     *
     * <p>When collected, the player enters an rapid-fire state for
     * a fixed duration. During this period, bullets are fired
     * automatically without requiring manual input.</p>
     *
     * @param player the player who collected this item
     */
    @Override
    public void apply(Player player) {
        // Enable rapid firing for 10 seconds
        player.activateAutoFire( duration: 10.0);
    }
}
```

\ItemShield.java



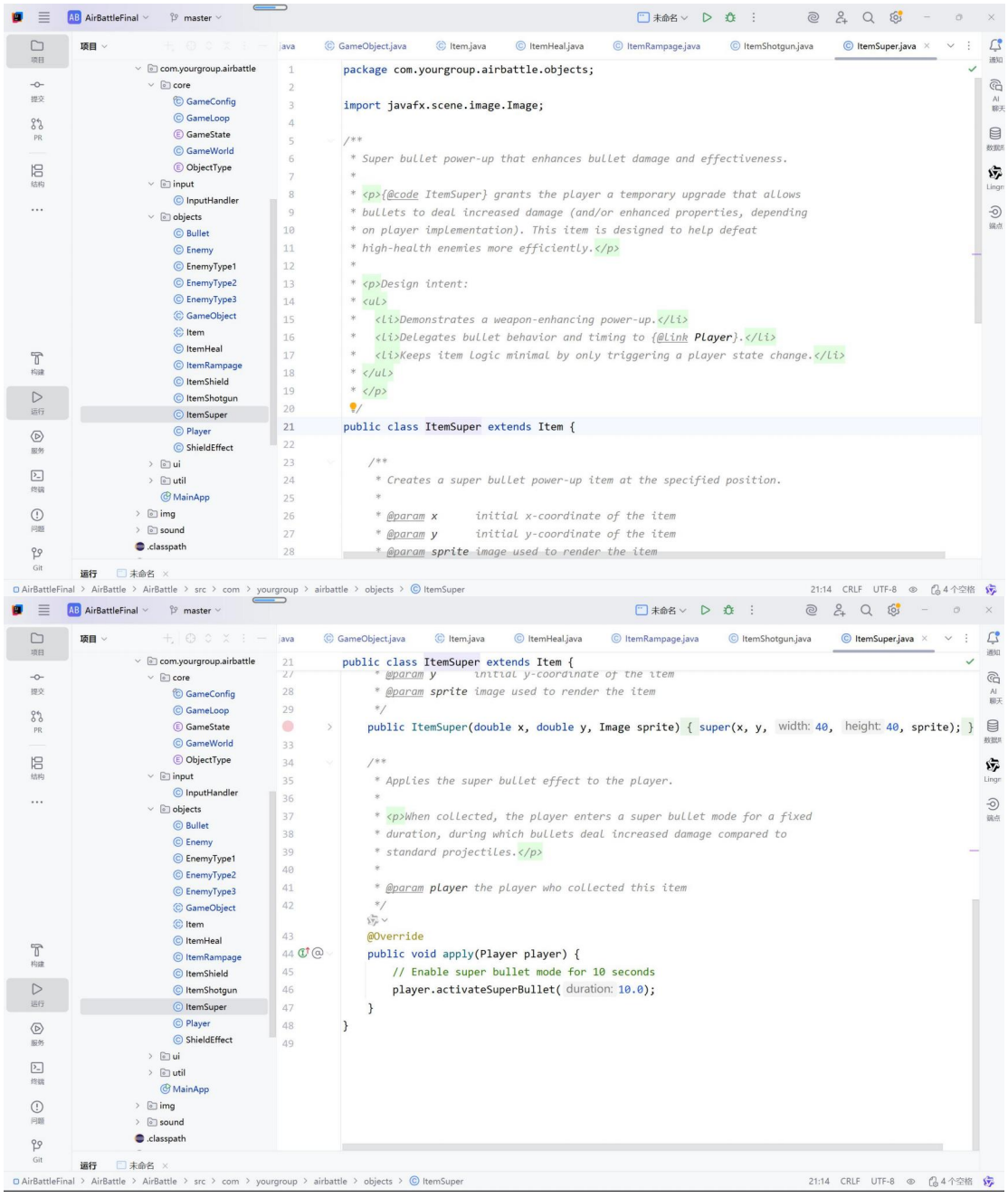
```
package com.yourgroup.airbattle.objects;

import javafx.scene.image.Image;

/**
 * Grants the player a temporary rapid-fire buff.
 *
 * <p>This buff reduces the weapon cooldown so the player can shoot more frequently.
 * The player still needs to press the fire key; the item does not enable automatic
 * shooting by itself.</p>
 * Rampage item that enables temporary rapid firing.
 *
 * <p>{@code ItemRampage} grants the player a short burst of increased
 * offensive capability by enabling rapid firing for a limited
 * duration. This item is designed to help the player handle
 * high-pressure situations with many enemies on screen.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Demonstrates a time-based power-up effect.</li>
 *   <li>Delegates effect duration management to {@link Player}.</li>
 *   <li>Keeps item logic simple by triggering state changes rather than
 *       managing timers internally.</li>
 * </ul>
 * </p>
 */
public class ItemRampage extends Item {
    /**
     * Creates a rampage power-up item at the specified position.
     *
     * @param x      initial x-coordinate of the item
     * @param y      initial y-coordinate of the item
     * @param sprite image used to render the item
     */
    public ItemRampage(double x, double y, Image sprite) { super(x, y, width: 40, height: 40, sprite);

    /**
     * Applies the rampage effect to the player.
     *
     * <p>When collected, the player enters an rapid-fire state for
     * a fixed duration. During this period, bullets are fired
     * automatically without requiring manual input.</p>
     *
     * @param player the player who collected this item
     */
    @Override
    public void apply(Player player) {
        // Enable rapid firing for 10 seconds
        player.activateAutoFire( duration: 10.0);
    }
}
```

\ItemShotgun.java



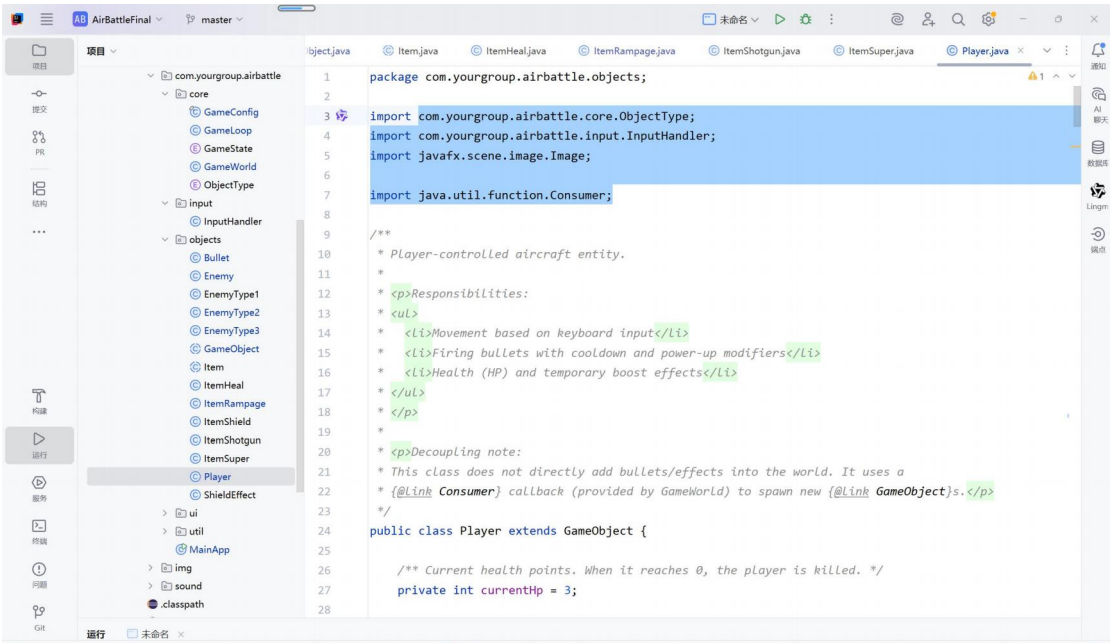
```java
package com.yourgroup.airbattle.objects;

import javafx.scene.image.Image;

/**
 * Shotgun power-up that enables multi-directional firing.
 *
 * <p>{@code ItemShotgun} temporarily allows the player to fire multiple
 * bullets in a spread pattern, increasing damage output at close range.
 * This item is designed to handle clusters of enemies more effectively.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Demonstrates a weapon-modifying power-up.</li>
 *   <li>Delegates firing logic and timing to {@link Player}.</li>
 *   <li>Avoids embedding weapon behavior directly inside the item.</li>
 * </ul>
 * </p>
 */
public class ItemShotgun extends Item {

    /**
     * Creates a shotgun power-up item at the specified position.
     *
     * @param x      initial x-coordinate of the item
     * @param y      initial y-coordinate of the item
     * @param sprite image used to render the item
     */
    public ItemShotgun(double x, double y, Image sprite) { super(x, y, width: 40, height: 40, sprite);

    /**
     * Applies the shotgun firing mode to the player.
     *
     * <p>When collected, the player enters a shotgun mode for a fixed duration,
     * during which each firing action produces multiple angled bullets.</p>
     *
     * @param player the player who collected this item
     */
    @Override
    public void apply(Player player) {
        // Enable shotgun firing mode for 10 seconds
        player.activateShotgun( duration: 10.0);
    }
}
```

\ItemSuper.java

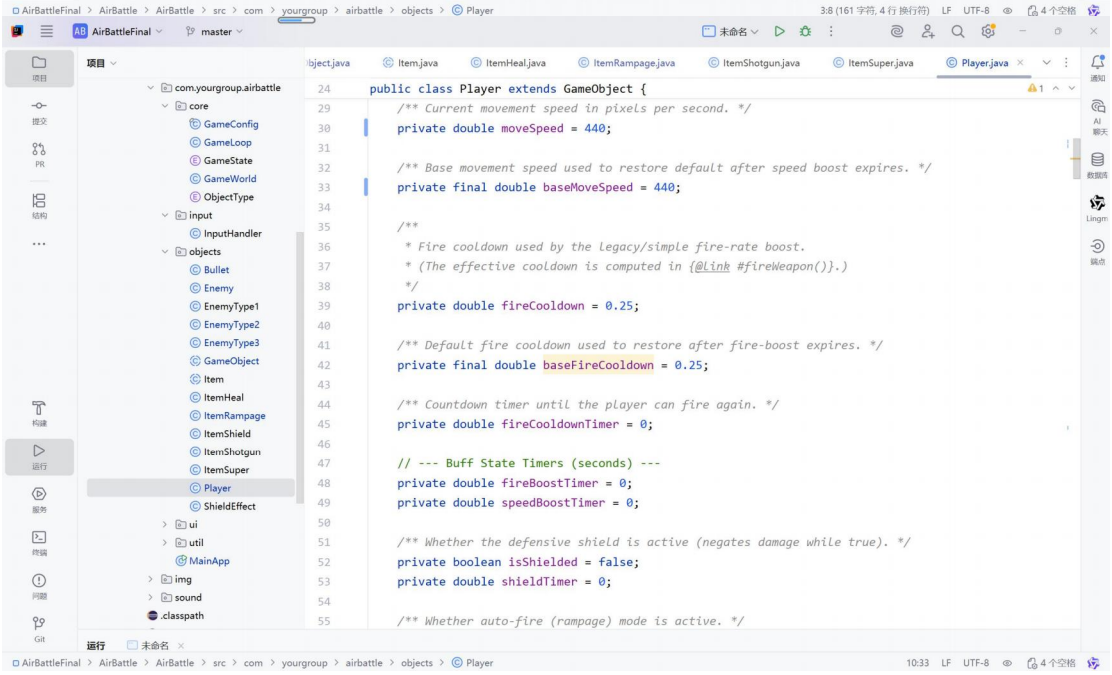


```java
package com.yourgroup.airbattle.objects;

import javafx.scene.image.Image;

/**
 * Super bullet power-up that enhances bullet damage and effectiveness.
 *
 * <p>{@code ItemSuper} grants the player a temporary upgrade that allows
 * bullets to deal increased damage (and/or enhanced properties, depending
 * on player implementation). This item is designed to help defeat
 * high-health enemies more efficiently.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Demonstrates a weapon-enhancing power-up.</li>
 *   <li>Delegates bullet behavior and timing to {@link Player}.</li>
 *   <li>Keeps item logic minimal by only triggering a player state change.</li>
 * </ul>
 * </p>
 */
public class ItemSuper extends Item {

    /**
     * Creates a super bullet power-up item at the specified position.
     *
     * @param x      initial x-coordinate of the item
     * @param y      initial y-coordinate of the item
     * @param sprite image used to render the item
     */
    public ItemSuper(double x, double y, Image sprite) { super(x, y, width: 40, height: 40, sprite); }

    /**
     * Applies the super bullet effect to the player.
     *
     * <p>When collected, the player enters a super bullet mode for a fixed
     * duration, during which bullets deal increased damage compared to
     * standard projectiles.</p>
     *
     * @param player the player who collected this item
     */
    @Override
    public void apply(Player player) {
        // Enable super bullet mode for 10 seconds
        player.activateSuperBullet( duration: 10.0);
    }
}
```

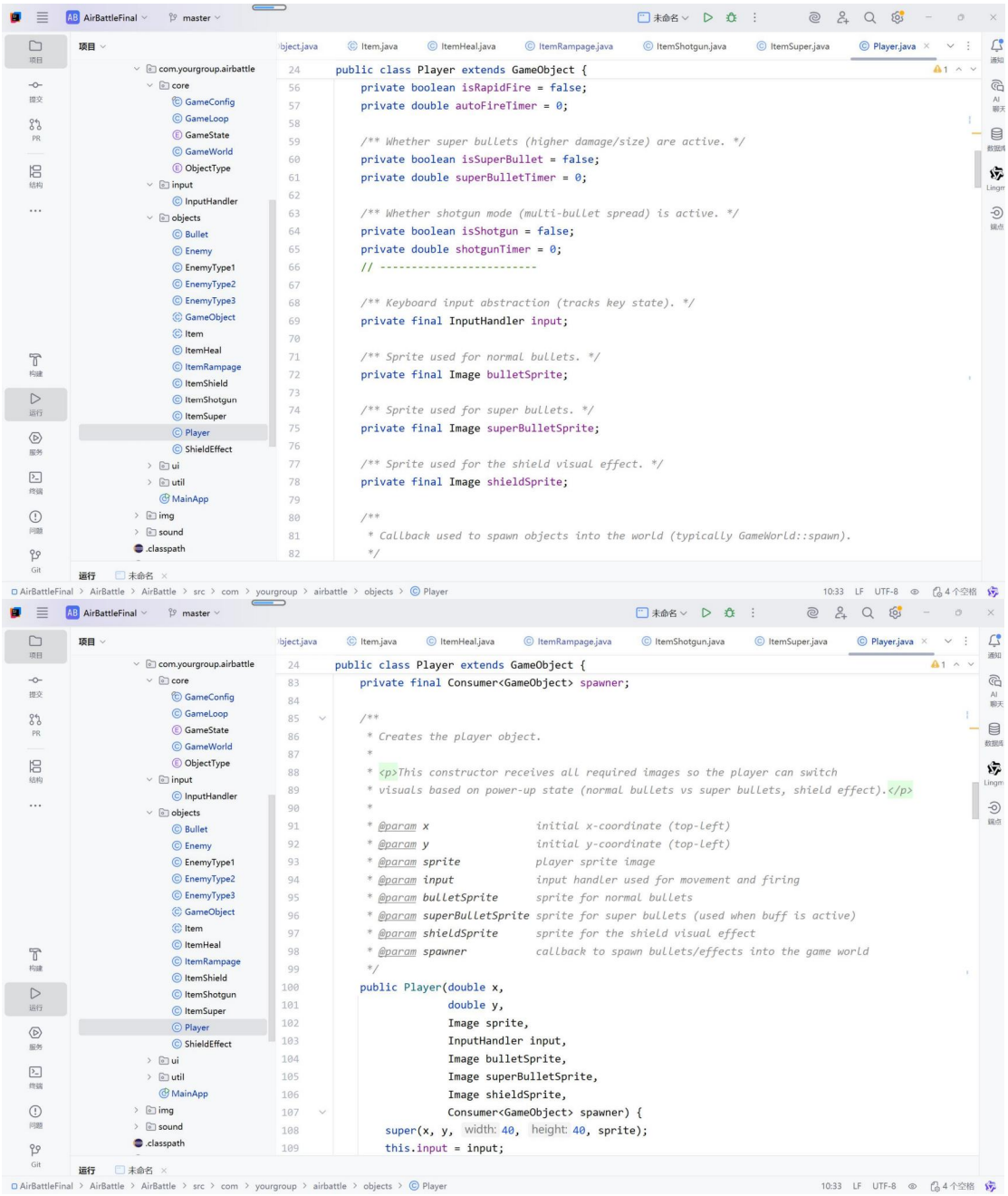
\Player.java



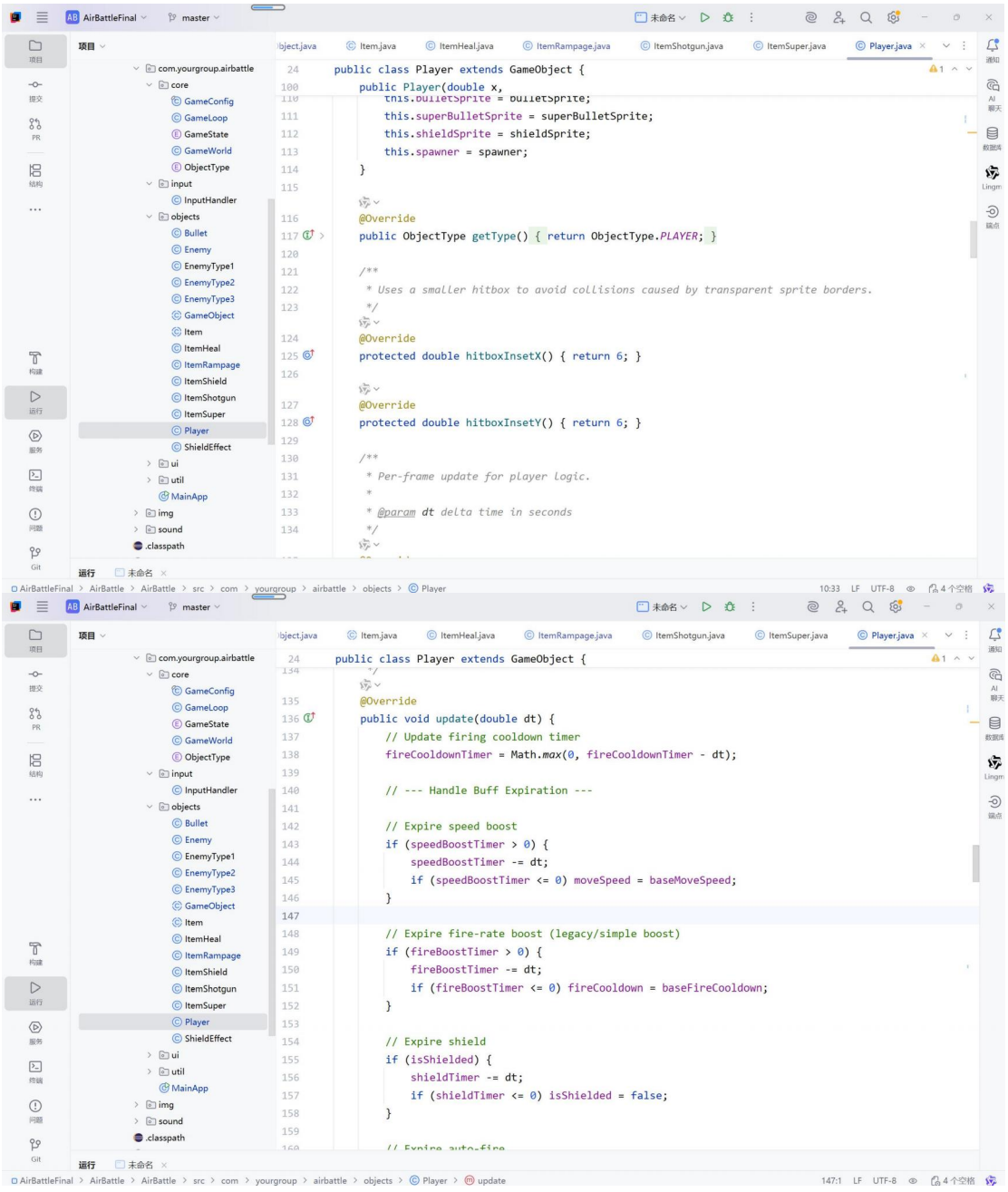
```
package com.yourgroup.airbattle.objects;

import com.yourgroup.airbattle.core.ObjectType;
import com.yourgroup.airbattle.input.InputHandler;
import javafx.scene.image.Image;

import java.util.function.Consumer;

/**
 * Player-controlled aircraft entity.
 *
 * <p>Responsibilities:
 * <ul>
 *   <li>Movement based on keyboard input</li>
 *   <li>Firing bullets with cooldown and power-up modifiers</li>
 *   <li>Health (HP) and temporary boost effects</li>
 * </ul>
 * </p>
 *
 * <p>Decoupling note:
 * This class does not directly add bullets/effects into the world. It uses a
 * {@link Consumer} callback (provided by GameWorld) to spawn new {@link GameObject}s.</p>
 */
public class Player extends GameObject {

    /** Current health points. When it reaches 0, the player is killed. */
    private int currentHp = 3;
```



```
public class Player extends GameObject {
    /** Current movement speed in pixels per second. */
    private double moveSpeed = 440;

    /** Base movement speed used to restore default after speed boost expires. */
    private final double baseMoveSpeed = 440;

    /**
     * Fire cooldown used by the legacy/simple fire-rate boost.
     * (The effective cooldown is computed in {@link #fireWeapon()}.)
     */
    private double fireCooldown = 0.25;

    /** Default fire cooldown used to restore after fire-boost expires. */
    private final double baseFireCooldown = 0.25;

    /** Countdown timer until the player can fire again. */
    private double fireCooldownTimer = 0;

    // --- Buff State Timers (seconds) ---
    private double fireBoostTimer = 0;
    private double speedBoostTimer = 0;

    /** Whether the defensive shield is active (negates damage while true). */
    private boolean isShielded = false;
    private double shieldTimer = 0;

    /** Whether auto-fire (rampage) mode is active. */
```



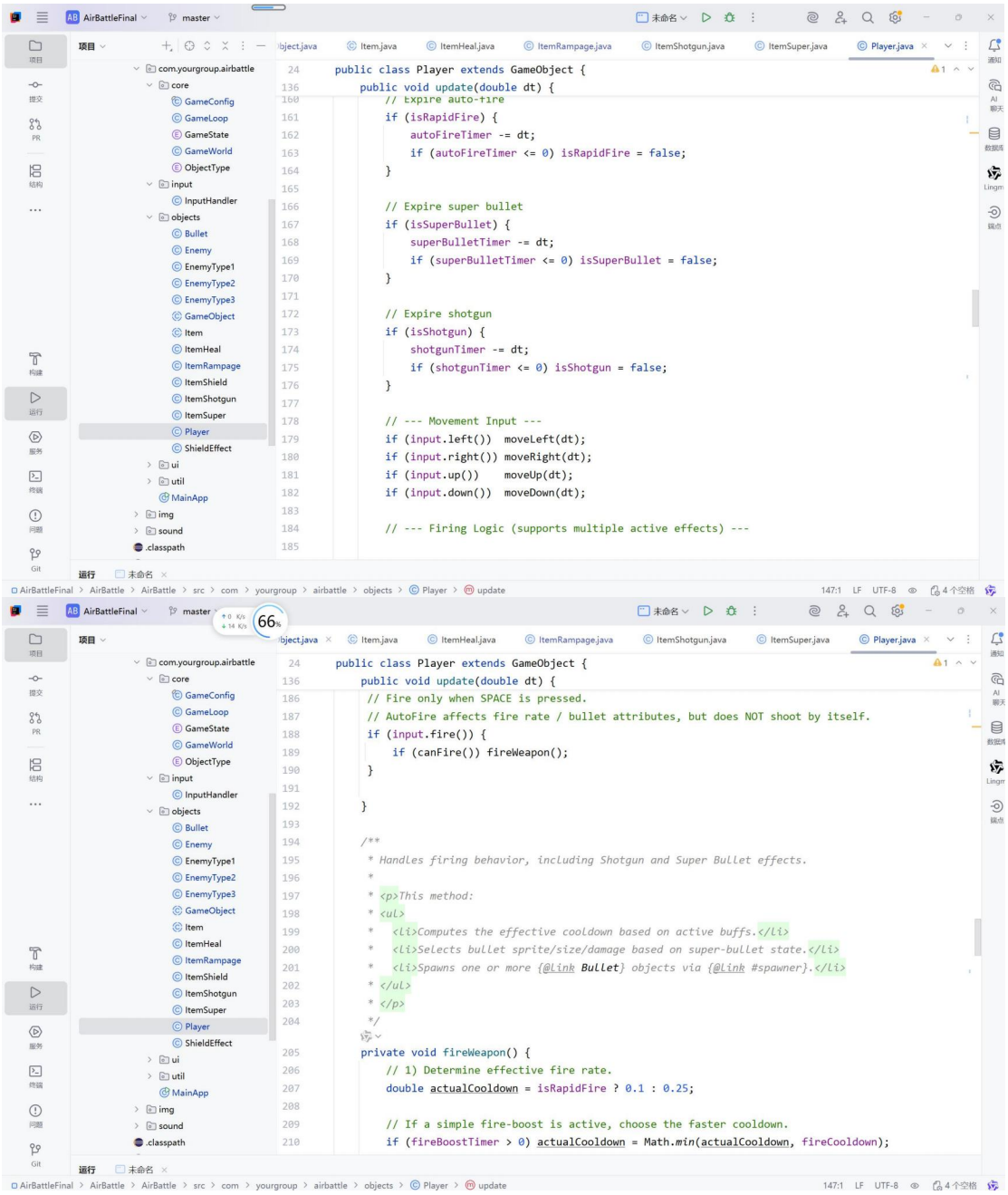
```
public class Player extends GameObject {
    private boolean isRapidFire = false;
    private double autoFireTimer = 0;

    /** Whether super bullets (higher damage/size) are active. */
    private boolean isSuperBullet = false;
    private double superBulletTimer = 0;

    /** Whether shotgun mode (multi-bullet spread) is active. */
    private boolean isShotgun = false;
    private double shotgunTimer = 0;
    // ------------------------

    /** Keyboard input abstraction (tracks key state). */
    private final InputHandler input;

    /** Sprite used for normal bullets. */
    private final Image bulletSprite;

    /** Sprite used for super bullets. */
    private final Image superBulletSprite;

    /** Sprite used for the shield visual effect. */
    private final Image shieldSprite;

    /**
     * Callback used to spawn objects into the world (typically GameWorld::spawn).
     */
    private final Consumer<GameObject> spawner;

    /**
     * Creates the player object.
     *
     * <p>This constructor receives all required images so the player can switch
     * visuals based on power-up state (normal bullets vs super bullets, shield effect).</p>
     *
     * @param x              initial x-coordinate (top-left)
     * @param y              initial y-coordinate (top-left)
     * @param sprite         player sprite image
     * @param input          input handler used for movement and firing
     * @param bulletSprite   sprite for normal bullets
     * @param superBulletSprite sprite for super bullets (used when buff is active)
     * @param shieldSprite   sprite for the shield visual effect
     * @param spawner        callback to spawn bullets/effects into the game world
     */
    public Player(double x,
                  double y,
                  Image sprite,
                  InputHandler input,
                  Image bulletSprite,
                  Image superBulletSprite,
                  Image shieldSprite,
                  Consumer<GameObject> spawner) {
        super(x, y, width: 40, height: 40, sprite);
        this.input = input;
```



```
public class Player extends GameObject {
    public Player(double x,
        this.bulletSprite = bulletSprite;
        this.superBulletSprite = superBulletSprite;
        this.shieldSprite = shieldSprite;
        this.spawner = spawner;
    }

    @Override
    public ObjectType getType() { return ObjectType.PLAYER; }

    /**
     * Uses a smaller hitbox to avoid collisions caused by transparent sprite borders.
     */
    @Override
    protected double hitboxInsetX() { return 6; }

    @Override
    protected double hitboxInsetY() { return 6; }

    /**
     * Per-frame update for player logic.
     *
     * @param dt delta time in seconds
     */
```



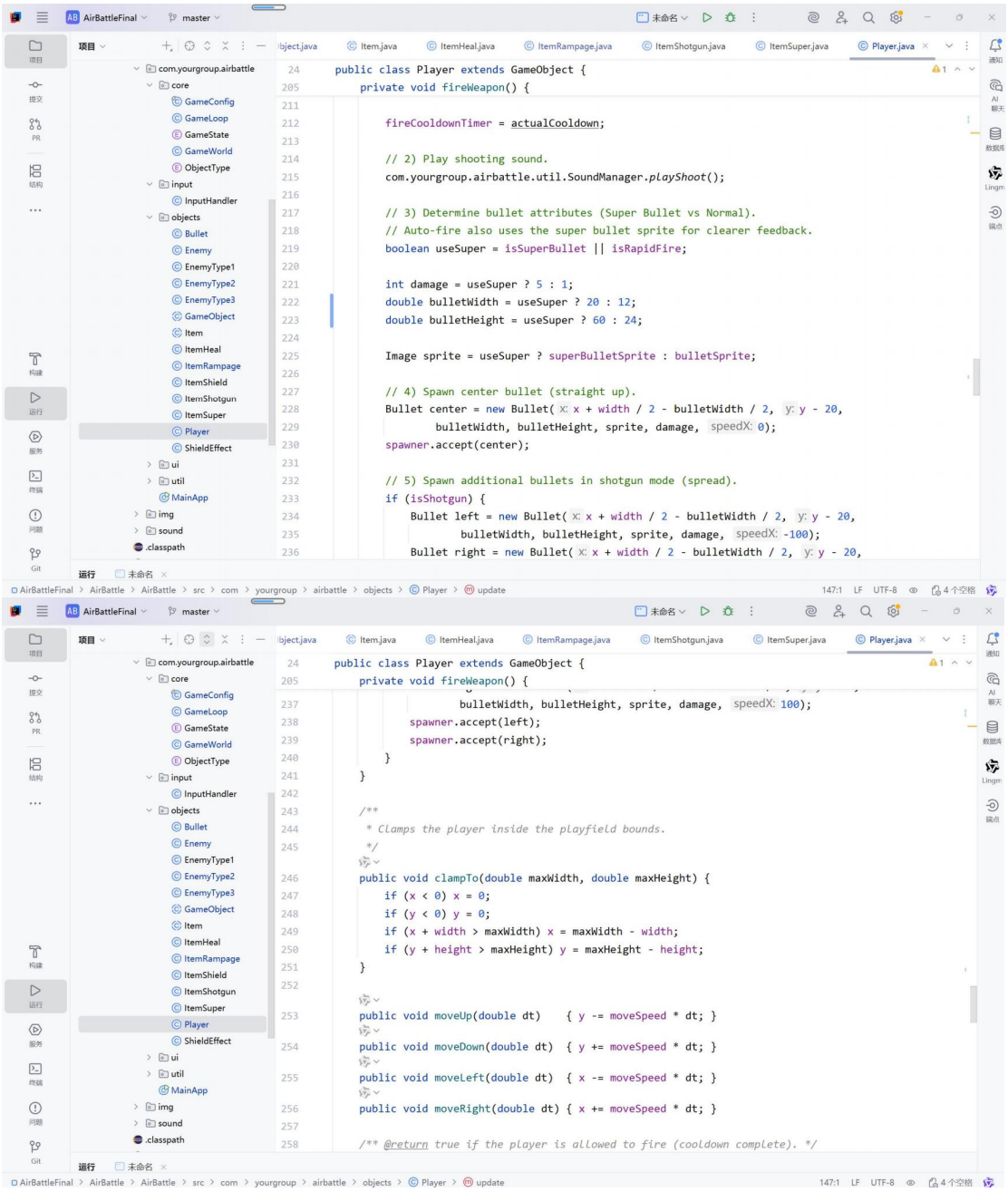
```
public class Player extends GameObject {
     */
    @Override
    public void update(double dt) {
        // Update firing cooldown timer
        fireCooldownTimer = Math.max(0, fireCooldownTimer - dt);

        // --- Handle Buff Expiration ---

        // Expire speed boost
        if (speedBoostTimer > 0) {
            speedBoostTimer -= dt;
            if (speedBoostTimer <= 0) moveSpeed = baseMoveSpeed;
        }

        // Expire fire-rate boost (legacy/simple boost)
        if (fireBoostTimer > 0) {
            fireBoostTimer -= dt;
            if (fireBoostTimer <= 0) fireCooldown = baseFireCooldown;
        }

        // Expire shield
        if (isShielded) {
            shieldTimer -= dt;
            if (shieldTimer <= 0) isShielded = false;
        }
```



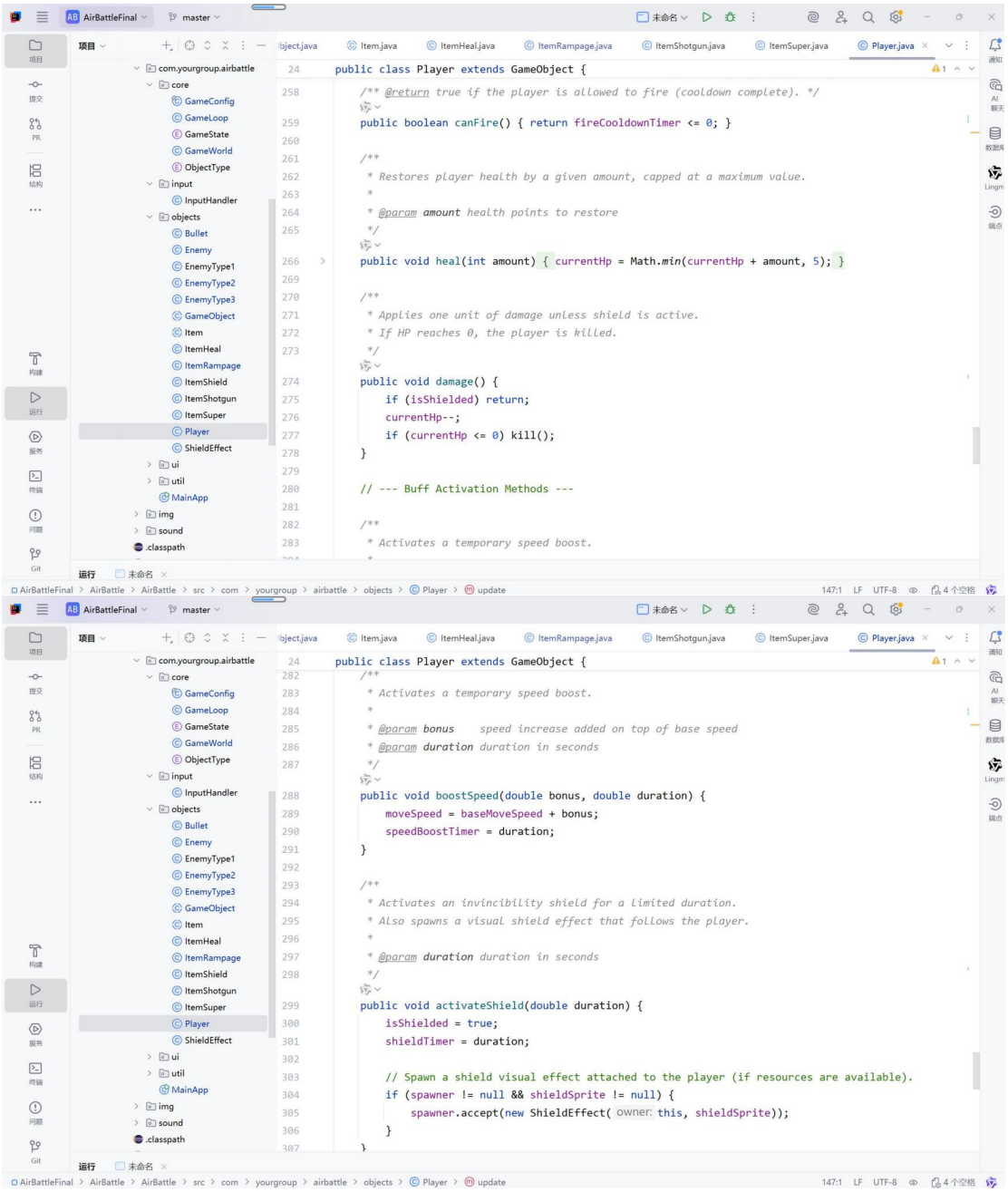
```java
public class Player extends GameObject {
    public void update(double dt) {
        // Expire auto-fire
        if (isRapidFire) {
            autoFireTimer -= dt;
            if (autoFireTimer <= 0) isRapidFire = false;
        }

        // Expire super bullet
        if (isSuperBullet) {
            superBulletTimer -= dt;
            if (superBulletTimer <= 0) isSuperBullet = false;
        }

        // Expire shotgun
        if (isShotgun) {
            shotgunTimer -= dt;
            if (shotgunTimer <= 0) isShotgun = false;
        }

        // --- Movement Input ---
        if (input.left())  moveLeft(dt);
        if (input.right()) moveRight(dt);
        if (input.up())    moveUp(dt);
        if (input.down())  moveDown(dt);

        // --- Firing Logic (supports multiple active effects) ---
```



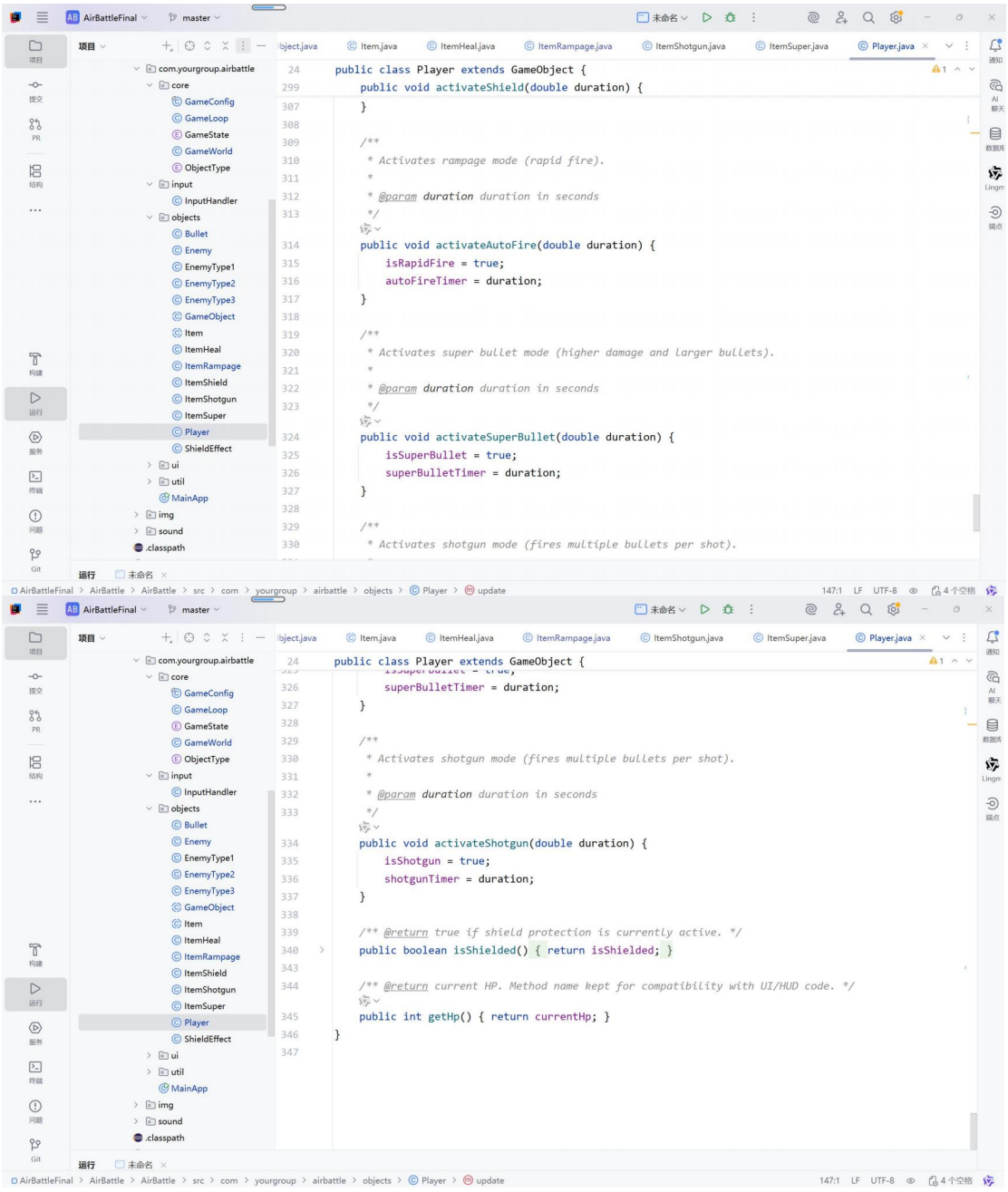
```java
public class Player extends GameObject {
    public void update(double dt) {
        // Fire only when SPACE is pressed.
        // AutoFire affects fire rate / bullet attributes, but does NOT shoot by itself.
        if (input.fire()) {
            if (canFire()) fireWeapon();
        }

    }

    /**
     * Handles firing behavior, including Shotgun and Super Bullet effects.
     *
     * <p>This method:
     * <ul>
     *   <li>Computes the effective cooldown based on active buffs.</li>
     *   <li>Selects bullet sprite/size/damage based on super-bullet state.</li>
     *   <li>Spawns one or more {@link Bullet} objects via {@link #spawner}.</li>
     * </ul>
     * </p>
     */
    private void fireWeapon() {
        // 1) Determine effective fire rate.
        double actualCooldown = isRapidFire ? 0.1 : 0.25;

        // If a simple fire-boost is active, choose the faster cooldown.
        if (fireBoostTimer > 0) actualCooldown = Math.min(actualCooldown, fireCooldown);
```



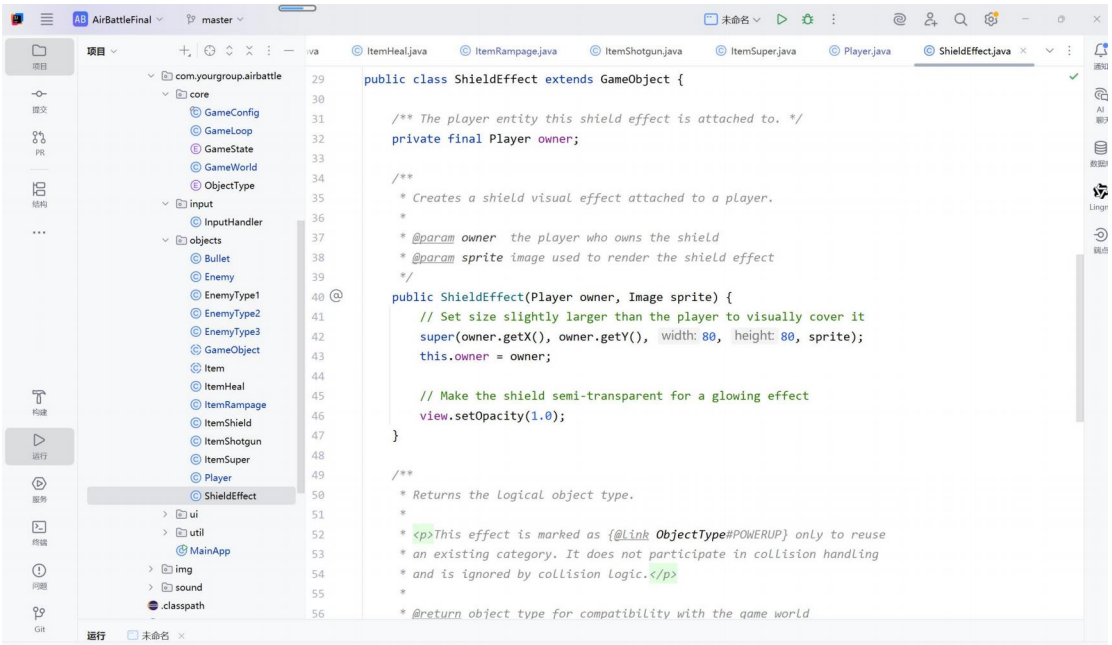
```
public class Player extends GameObject {
    private void fireWeapon() {

        fireCooldownTimer = actualCooldown;

        // 2) Play shooting sound.
        com.yourgroup.airbattle.util.SoundManager.playShoot();

        // 3) Determine bullet attributes (Super Bullet vs Normal).
        // Auto-fire also uses the super bullet sprite for clearer feedback.
        boolean useSuper = isSuperBullet || isRapidFire;

        int damage = useSuper ? 5 : 1;
        double bulletWidth = useSuper ? 20 : 12;
        double bulletHeight = useSuper ? 60 : 24;

        Image sprite = useSuper ? superBulletSprite : bulletSprite;

        // 4) Spawn center bullet (straight up).
        Bullet center = new Bullet( x: x + width / 2 - bulletWidth / 2, y: y - 20,
                bulletWidth, bulletHeight, sprite, damage, speedX: 0);
        spawner.accept(center);

        // 5) Spawn additional bullets in shotgun mode (spread).
        if (isShotgun) {
            Bullet left = new Bullet( x: x + width / 2 - bulletWidth / 2, y: y - 20,
                    bulletWidth, bulletHeight, sprite, damage, speedX: -100);
            Bullet right = new Bullet( x: x + width / 2 - bulletWidth / 2, y: y - 20,
                    bulletWidth, bulletHeight, sprite, damage, speedX: 100);
            spawner.accept(left);
            spawner.accept(right);
        }
    }

    /**
     * Clamps the player inside the playfield bounds.
     */
    public void clampTo(double maxWidth, double maxHeight) {
        if (x < 0) x = 0;
        if (y < 0) y = 0;
        if (x + width > maxWidth) x = maxWidth - width;
        if (y + height > maxHeight) y = maxHeight - height;
    }

    public void moveUp(double dt)    { y -= moveSpeed * dt; }
    public void moveDown(double dt)  { y += moveSpeed * dt; }
    public void moveLeft(double dt)  { x -= moveSpeed * dt; }
    public void moveRight(double dt) { x += moveSpeed * dt; }

    /** @return true if the player is allowed to fire (cooldown complete). */
```



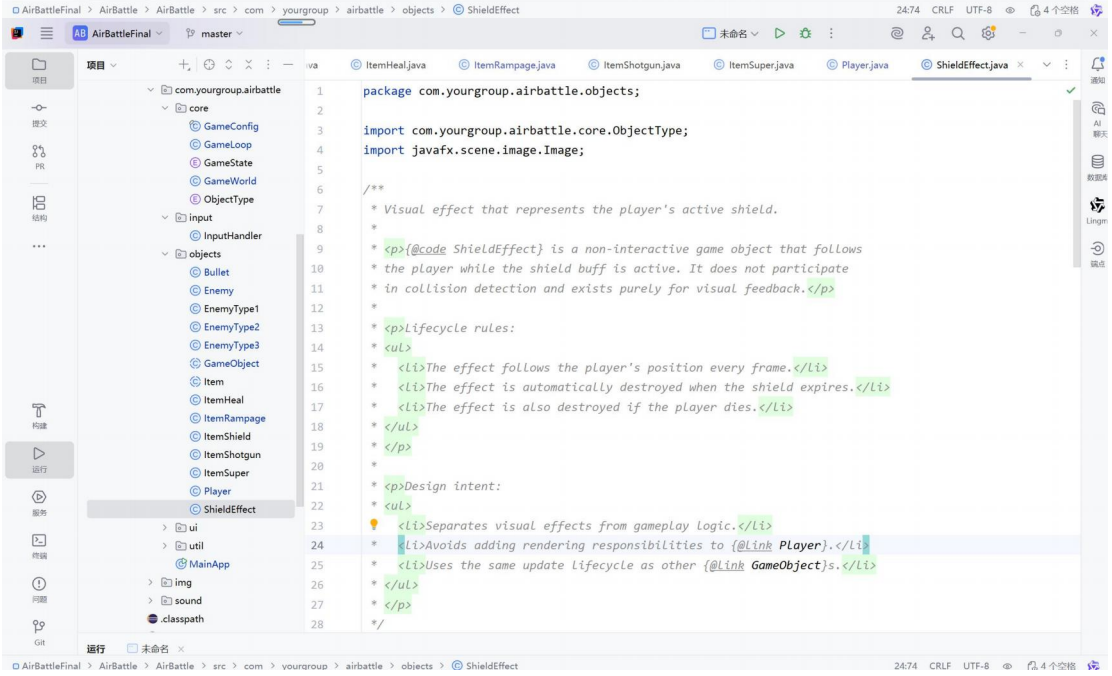
```
public class Player extends GameObject {
    /** @return true if the player is allowed to fire (cooldown complete). */
    public boolean canFire() { return fireCooldownTimer <= 0; }

    /**
     * Restores player health by a given amount, capped at a maximum value.
     *
     * @param amount health points to restore
     */
    public void heal(int amount) { currentHp = Math.min(currentHp + amount, 5); }

    /**
     * Applies one unit of damage unless shield is active.
     * If HP reaches 0, the player is killed.
     */
    public void damage() {
        if (isShielded) return;
        currentHp--;
        if (currentHp <= 0) kill();
    }

    // --- Buff Activation Methods ---

    /**
     * Activates a temporary speed boost.
     *
     * @param bonus    speed increase added on top of base speed
     * @param duration duration in seconds
     */
    public void boostSpeed(double bonus, double duration) {
        moveSpeed = baseMoveSpeed + bonus;
        speedBoostTimer = duration;
    }

    /**
     * Activates an invincibility shield for a limited duration.
     * Also spawns a visual shield effect that follows the player.
     *
     * @param duration duration in seconds
     */
    public void activateShield(double duration) {
        isShielded = true;
        shieldTimer = duration;

        // Spawn a shield visual effect attached to the player (if resources are available).
        if (spawner != null && shieldSprite != null) {
            spawner.accept(new ShieldEffect( owner: this, shieldSprite));
        }
    }
```


AirBattleFinal
master
Item.java
ItemHeal.java
ItemRampage.java
ItemShotgun.java
ItemSuper.java
Player.java
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
util
MainApp
img
sound
.classpath
public class Player extends GameObject {
public void activateShield(double duration) {
}
/**
* Activates rampage mode (rapid fire).
*
* @param duration duration in seconds
*/
public void activateAutoFire(double duration) {
isRapidFire = true;
autoFireTimer = duration;
}
/**
* Activates super bullet mode (higher damage and larger bullets).
*
* @param duration duration in seconds
*/
public void activateSuperBullet(double duration) {
isSuperBullet = true;
superBulletTimer = duration;
}
/**
* Activates shotgun mode (fires multiple bullets per shot).
*
* @param duration duration in seconds
*/
public void activateShotgun(double duration) {
isShotgun = true;
shotgunTimer = duration;
}
/** @return true if shield protection is currently active. */
public boolean isShielded() { return isShielded; }
/** @return current HP. Method name kept for compatibility with UI/HUD code. */
public int getHp() { return currentHp; }
}

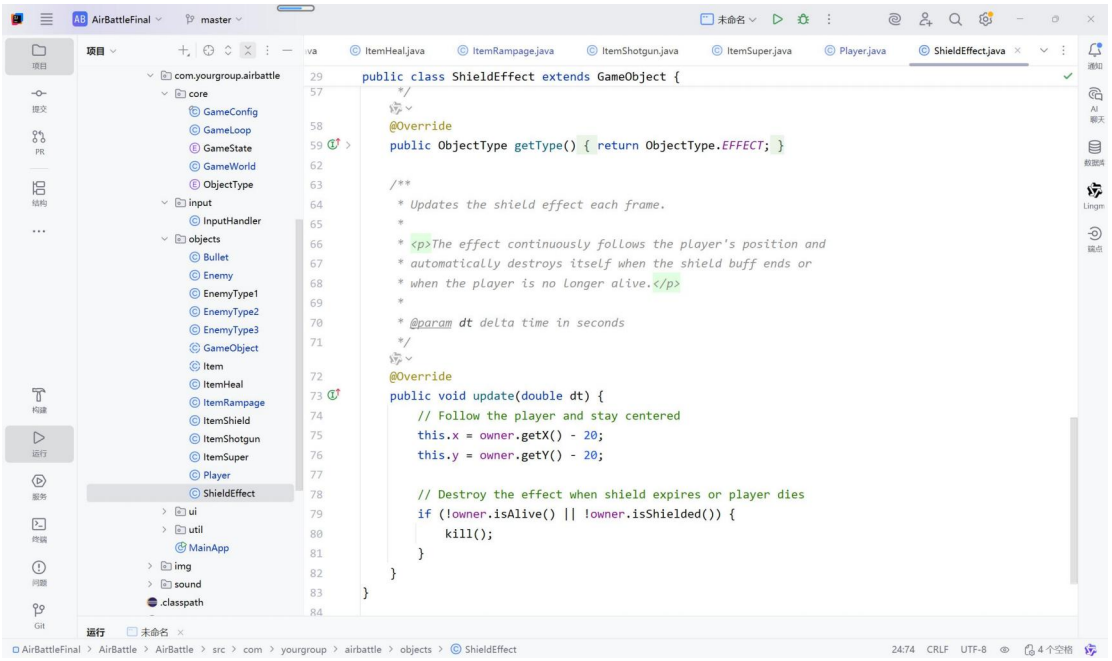
\ShieldEffect.java



```java
public class ShieldEffect extends GameObject {

    /** The player entity this shield effect is attached to. */
    private final Player owner;

    /**
     * Creates a shield visual effect attached to a player.
     *
     * @param owner  the player who owns the shield
     * @param sprite image used to render the shield effect
     */
    public ShieldEffect(Player owner, Image sprite) {
        // Set size slightly larger than the player to visually cover it
        super(owner.getX(), owner.getY(), width: 80, height: 80, sprite);
        this.owner = owner;

        // Make the shield semi-transparent for a glowing effect
        view.setOpacity(1.0);
    }

    /**
     * Returns the logical object type.
     *
     * <p>This effect is marked as {@Link ObjectType#POWERUP} only to reuse
     * an existing category. It does not participate in collision handling
     * and is ignored by collision logic.</p>
     *
     * @return object type for compatibility with the game world
```

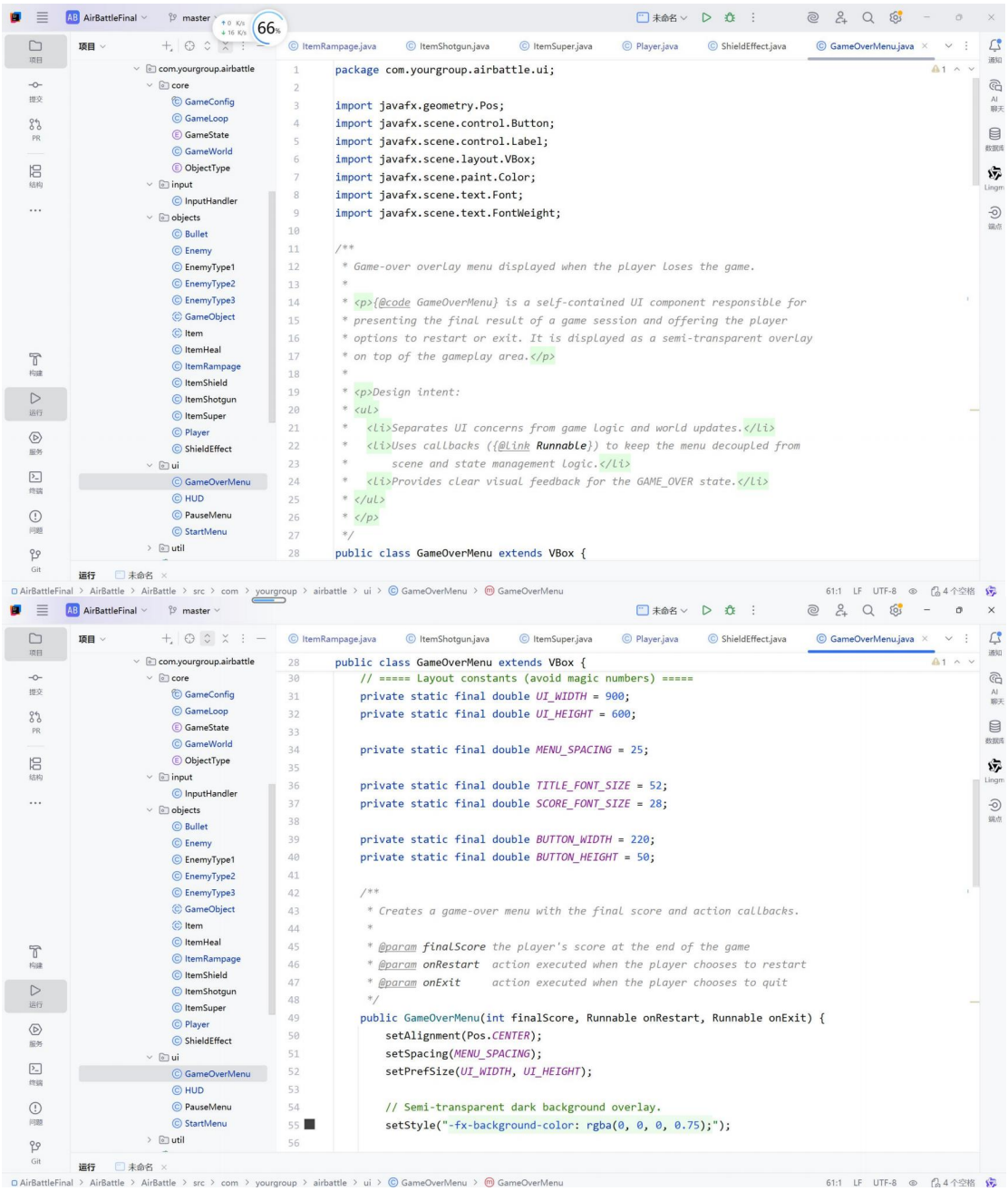


```java
package com.yourgroup.airbattle.objects;

import com.yourgroup.airbattle.core.ObjectType;
import javafx.scene.image.Image;

/**
 * Visual effect that represents the player's active shield.
 *
 * <p>{@code ShieldEffect} is a non-interactive game object that follows
 * the player while the shield buff is active. It does not participate
 * in collision detection and exists purely for visual feedback.</p>
 *
 * <p>Lifecycle rules:
 * <ul>
 *   <li>The effect follows the player's position every frame.</li>
 *   <li>The effect is automatically destroyed when the shield expires.</li>
 *   <li>The effect is also destroyed if the player dies.</li>
 * </ul>
 * </p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Separates visual effects from gameplay logic.</li>
 *   <li>Avoids adding rendering responsibilities to {@Link Player}.</li>
 *   <li>Uses the same update lifecycle as other {@Link GameObject}s.</li>
 * </ul>
 * </p>
 */
```


AirBattleFinal
master
未命名
项目
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
util
MainApp
img
sound
.classpath
ItemHeal.java
ItemRampage.java
ItemShotgun.java
ItemSuper.java
Player.java
ShieldEffect.java
public class ShieldEffect extends GameObject {
*/
@Override
public ObjectType getType() { return ObjectType.EFFECT; }
/**
* Updates the shield effect each frame.
*
* The effect continuously follows the player's position and
* automatically destroys itself when the shield buff ends or
* when the player is no longer alive.
*
* @param dt delta time in seconds
*/
@Override
public void update(double dt) {
// Follow the player and stay centered
this.x = owner.getX() - 20;
this.y = owner.getY() - 20;
// Destroy the effect when shield expires or player dies
if (!owner.isAlive() || !owner.isShielded()) {
kill();
}
}
}
运行
未命名
AirBattleFinal › AirBattle › AirBattle › src › com › yourgroup › airbattle › objects › ShieldEffect
24:74
CRLF
UTF-8
4 个空格

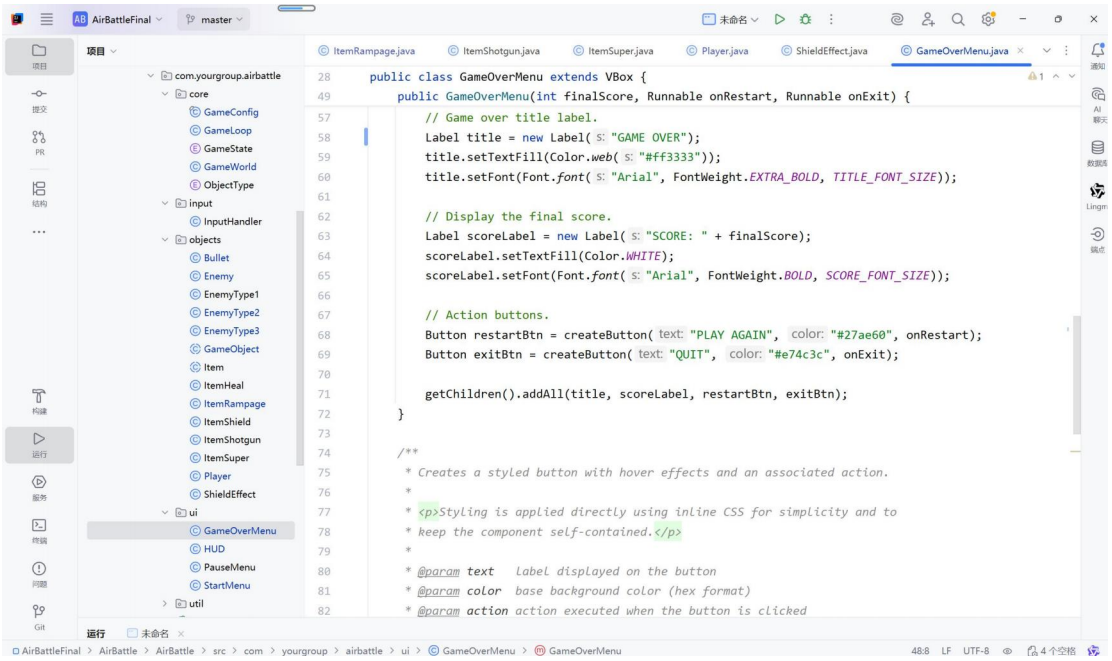
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\ui

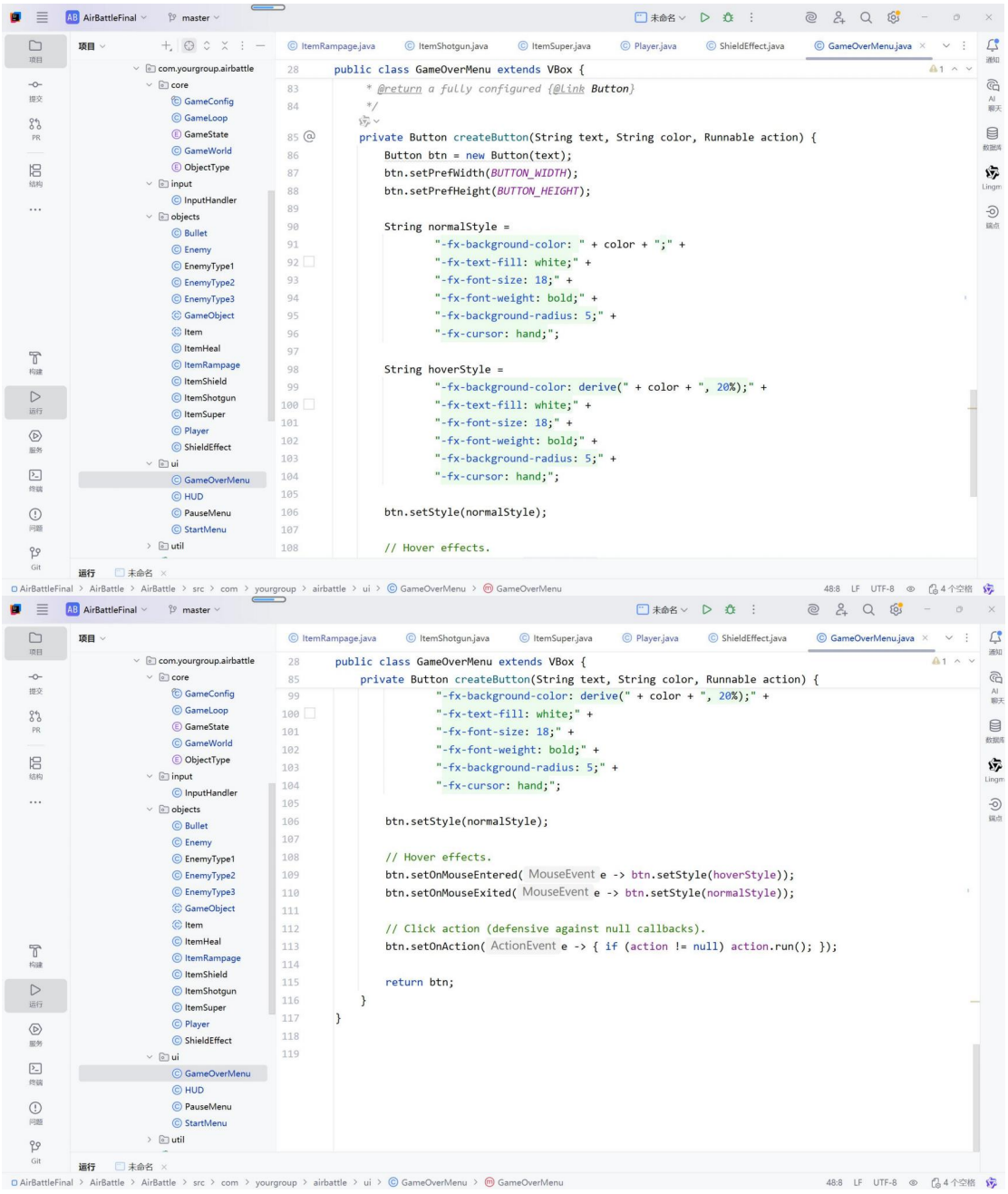
\GameOverMenu.java



```java
package com.yourgroup.airbattle.ui;

import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

/**
 * Game-over overlay menu displayed when the player loses the game.
 *
 * <p>{@code GameOverMenu} is a self-contained UI component responsible for
 * presenting the final result of a game session and offering the player
 * options to restart or exit. It is displayed as a semi-transparent overlay
 * on top of the gameplay area.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Separates UI concerns from game logic and world updates.</li>
 *   <li>Uses callbacks ({@link Runnable}) to keep the menu decoupled from
 *       scene and state management logic.</li>
 *   <li>Provides clear visual feedback for the GAME_OVER state.</li>
 * </ul>
 * </p>
 */
public class GameOverMenu extends VBox {
```

```java
public class GameOverMenu extends VBox {
    // ===== Layout constants (avoid magic numbers) =====
    private static final double UI_WIDTH = 900;
    private static final double UI_HEIGHT = 600;

    private static final double MENU_SPACING = 25;

    private static final double TITLE_FONT_SIZE = 52;
    private static final double SCORE_FONT_SIZE = 28;

    private static final double BUTTON_WIDTH = 220;
    private static final double BUTTON_HEIGHT = 50;

    /**
     * Creates a game-over menu with the final score and action callbacks.
     *
     * @param finalScore the player's score at the end of the game
     * @param onRestart  action executed when the player chooses to restart
     * @param onExit     action executed when the player chooses to quit
     */
    public GameOverMenu(int finalScore, Runnable onRestart, Runnable onExit) {
        setAlignment(Pos.CENTER);
        setSpacing(MENU_SPACING);
        setPrefSize(UI_WIDTH, UI_HEIGHT);

        // Semi-transparent dark background overlay.
        setStyle("-fx-background-color: rgba(0, 0, 0, 0.75);");
```


AirBattleFinal
master
未命名
项目
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
ItemRampage.java
ItemShotgun.java
ItemSuper.java
Player.java
ShieldEffect.java
GameOverMenu.java
public class GameOverMenu extends VBox {
public GameOverMenu(int finalScore, Runnable onRestart, Runnable onExit) {
// Game over title label.
Label title = new Label(s: "GAME OVER");
title.setTextFill(Color.web(s: "#ff3333"));
title.setFont(Font.font(s: "Arial", FontWeight.EXTRA_BOLD, TITLE_FONT_SIZE));
// Display the final score.
Label scoreLabel = new Label(s: "SCORE: " + finalScore);
scoreLabel.setTextFill(Color.WHITE);
scoreLabel.setFont(Font.font(s: "Arial", FontWeight.BOLD, SCORE_FONT_SIZE));
// Action buttons.
Button restartBtn = createButton(text: "PLAY AGAIN", color: "#27ae60", onRestart);
Button exitBtn = createButton(text: "QUIT", color: "#e74c3c", onExit);
getChildren().addAll(title, scoreLabel, restartBtn, exitBtn);
}
/**
* Creates a styled button with hover effects and an associated action.
*
* Styling is applied directly using inline CSS for simplicity and to
* keep the component self-contained.
*
* @param text label displayed on the button
* @param color base background color (hex format)
* @param action action executed when the button is clicked
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > ui > GameOverMenu > GameOverMenu
48:8 LF UTF-8 4 个空格

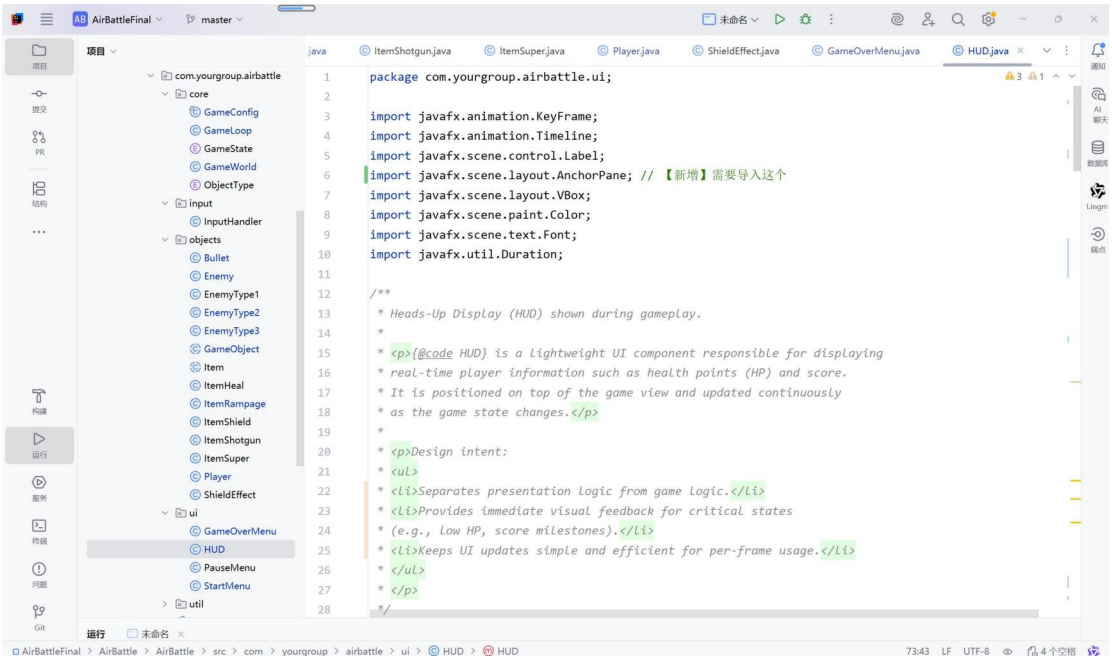


```
28  public class GameOverMenu extends VBox {
83      * @return a fully configured {@link Button}
84      */
85      private Button createButton(String text, String color, Runnable action) {
86          Button btn = new Button(text);
87          btn.setPrefWidth(BUTTON_WIDTH);
88          btn.setPrefHeight(BUTTON_HEIGHT);
89
90          String normalStyle =
91                  "-fx-background-color: " + color + ";" +
92                  "-fx-text-fill: white;" +
93                  "-fx-font-size: 18;" +
94                  "-fx-font-weight: bold;" +
95                  "-fx-background-radius: 5;" +
96                  "-fx-cursor: hand;";
97
98          String hoverStyle =
99                  "-fx-background-color: derive(" + color + ", 20%);" +
100                 "-fx-text-fill: white;" +
101                 "-fx-font-size: 18;" +
102                 "-fx-font-weight: bold;" +
103                 "-fx-background-radius: 5;" +
104                 "-fx-cursor: hand;";
105
106         btn.setStyle(normalStyle);
107
108         // Hover effects.
```

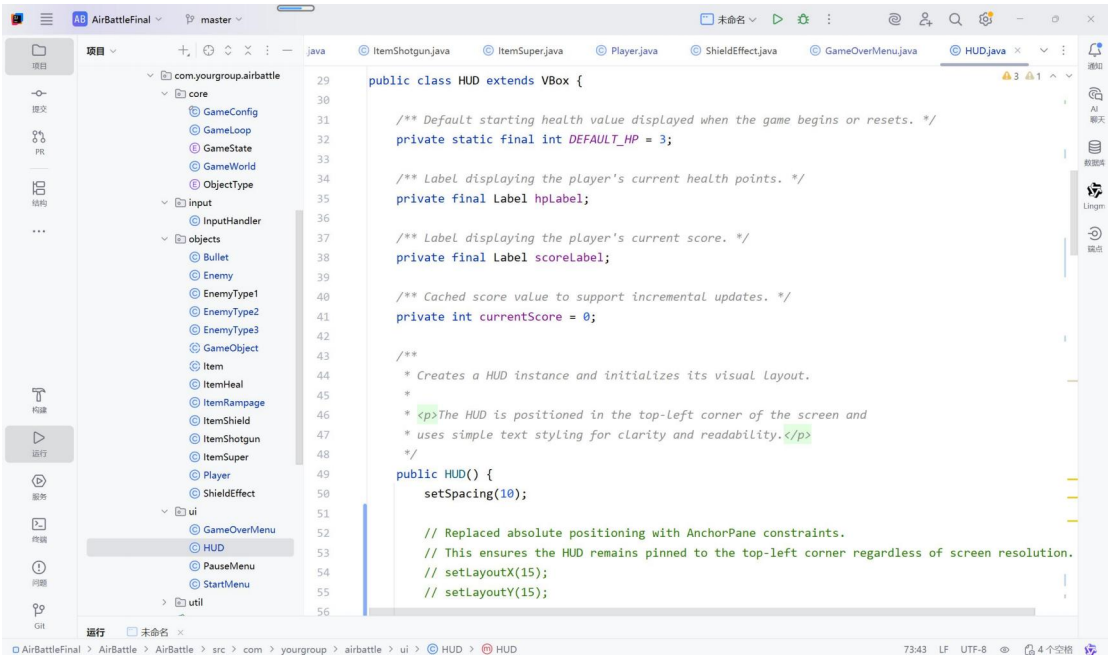


```
28  public class GameOverMenu extends VBox {
85      private Button createButton(String text, String color, Runnable action) {
99                  "-fx-background-color: derive(" + color + ", 20%);" +
100                 "-fx-text-fill: white;" +
101                 "-fx-font-size: 18;" +
102                 "-fx-font-weight: bold;" +
103                 "-fx-background-radius: 5;" +
104                 "-fx-cursor: hand;";
105
106         btn.setStyle(normalStyle);
107
108         // Hover effects.
109         btn.setOnMouseEntered( MouseEvent e -> btn.setStyle(hoverStyle));
110         btn.setOnMouseExited( MouseEvent e -> btn.setStyle(normalStyle));
111
112         // Click action (defensive against null callbacks).
113         btn.setOnAction( ActionEvent e -> { if (action != null) action.run(); });
114
115         return btn;
116     }
117 }
118
119
```

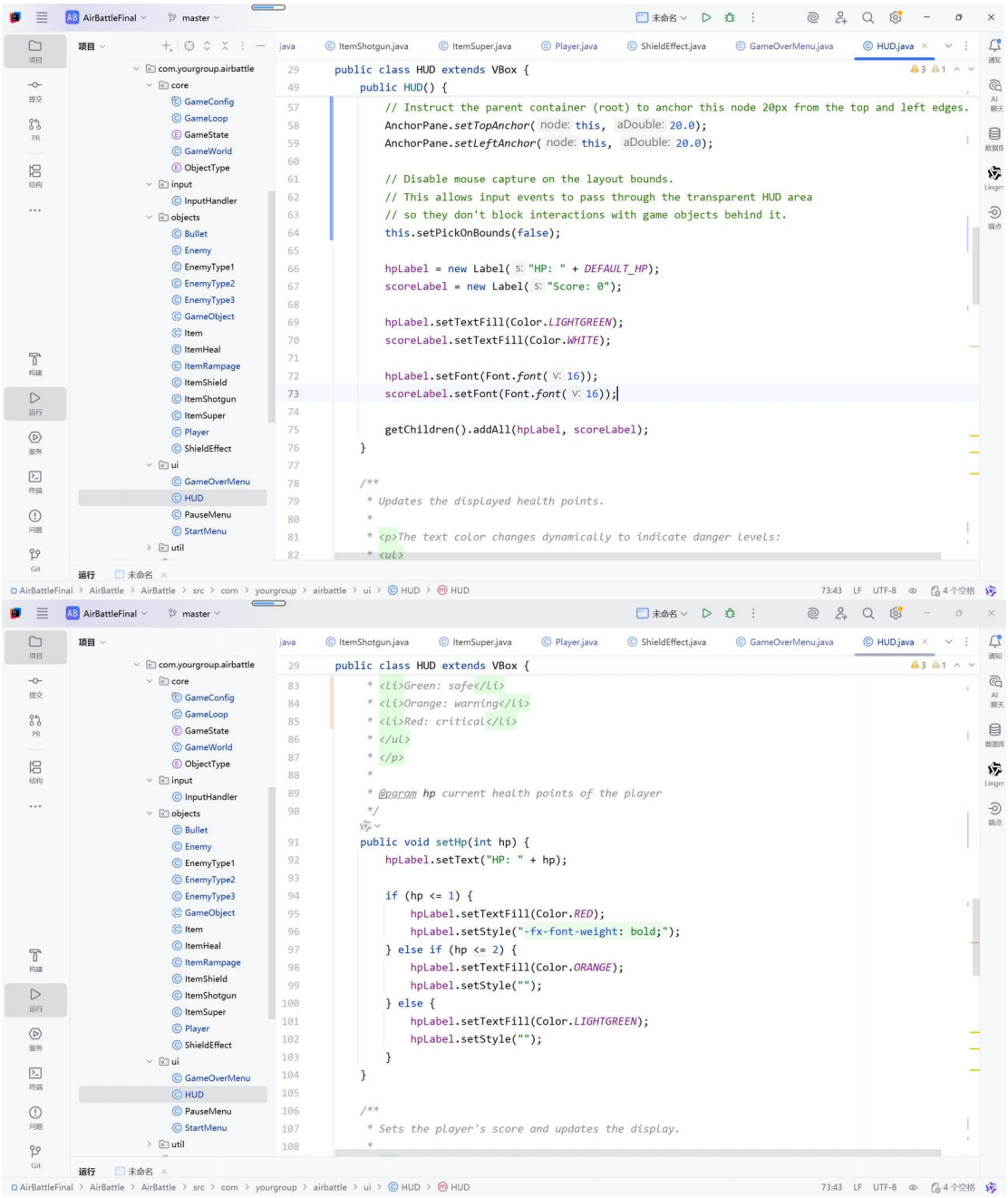
\HUD.java



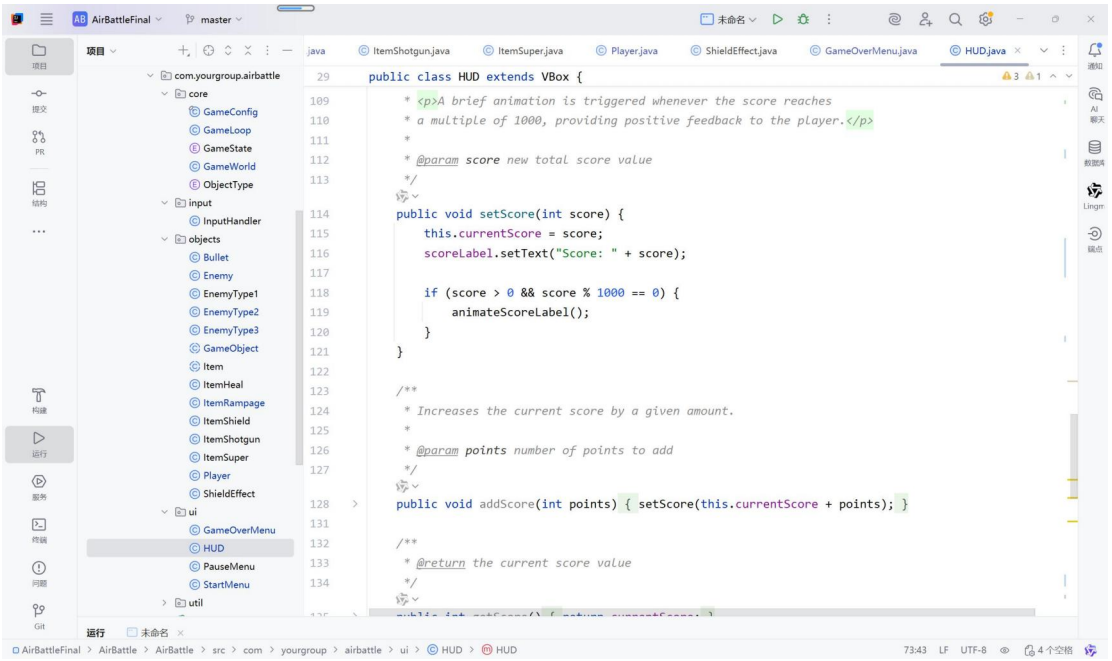
```java
package com.yourgroup.airbattle.ui;

import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane; // 【新增】需要导入这个
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.util.Duration;

/**
 * Heads-Up Display (HUD) shown during gameplay.
 *
 * <p>{@code HUD} is a lightweight UI component responsible for displaying
 * real-time player information such as health points (HP) and score.
 * It is positioned on top of the game view and updated continuously
 * as the game state changes.</p>
 *
 * <p>Design intent:
 * <ul>
 * <li>Separates presentation logic from game logic.</li>
 * <li>Provides immediate visual feedback for critical states
 * (e.g., low HP, score milestones).</li>
 * <li>Keeps UI updates simple and efficient for per-frame usage.</li>
 * </ul>
 * </p>
 */
```

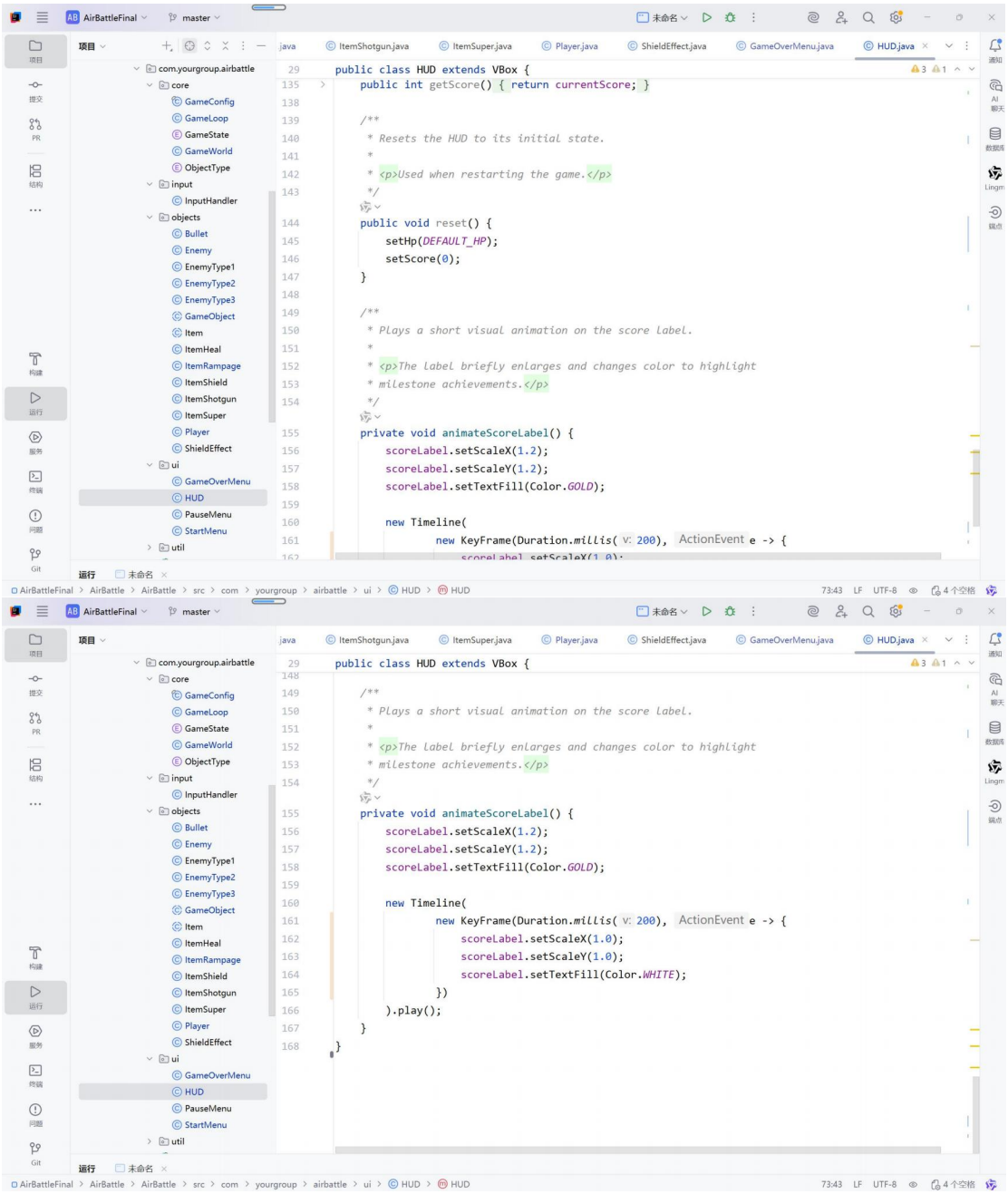


```java
public class HUD extends VBox {

    /** Default starting health value displayed when the game begins or resets. */
    private static final int DEFAULT_HP = 3;

    /** Label displaying the player's current health points. */
    private final Label hpLabel;

    /** Label displaying the player's current score. */
    private final Label scoreLabel;

    /** Cached score value to support incremental updates. */
    private int currentScore = 0;

    /**
     * Creates a HUD instance and initializes its visual layout.
     *
     * <p>The HUD is positioned in the top-left corner of the screen and
     * uses simple text styling for clarity and readability.</p>
     */
    public HUD() {
        setSpacing(10);

        // Replaced absolute positioning with AnchorPane constraints.
        // This ensures the HUD remains pinned to the top-left corner regardless of screen resolution.
        // setLayoutX(15);
        // setLayoutY(15);
```



```java
public class HUD extends VBox {
    public HUD() {
        // Instruct the parent container (root) to anchor this node 20px from the top and left edges.
        AnchorPane.setTopAnchor( node: this, aDouble: 20.0);
        AnchorPane.setLeftAnchor( node: this, aDouble: 20.0);

        // Disable mouse capture on the layout bounds.
        // This allows input events to pass through the transparent HUD area
        // so they don't block interactions with game objects behind it.
        this.setPickOnBounds(false);

        hpLabel = new Label( s: "HP: " + DEFAULT_HP);
        scoreLabel = new Label( s: "Score: 0");

        hpLabel.setTextFill(Color.LIGHTGREEN);
        scoreLabel.setTextFill(Color.WHITE);

        hpLabel.setFont(Font.font( v: 16));
        scoreLabel.setFont(Font.font( v: 16));

        getChildren().addAll(hpLabel, scoreLabel);
    }

    /**
     * Updates the displayed health points.
     *
     * <p>The text color changes dynamically to indicate danger levels:
     * <ul>
     * <li>Green: safe</li>
     * <li>Orange: warning</li>
     * <li>Red: critical</li>
     * </ul>
     * </p>
     *
     * @param hp current health points of the player
     */
    public void setHp(int hp) {
        hpLabel.setText("HP: " + hp);

        if (hp <= 1) {
            hpLabel.setTextFill(Color.RED);
            hpLabel.setStyle("-fx-font-weight: bold;");
        } else if (hp <= 2) {
            hpLabel.setTextFill(Color.ORANGE);
            hpLabel.setStyle("");
        } else {
            hpLabel.setTextFill(Color.LIGHTGREEN);
            hpLabel.setStyle("");
        }
    }

    /**
     * Sets the player's score and updates the display.
     *
```


AirBattleFinal
master
未命名
项目
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
ItemShotgun.java
ItemSuper.java
Player.java
ShieldEffect.java
GameOverMenu.java
HUD.java
public class HUD extends VBox {
* A brief animation is triggered whenever the score reaches
* a multiple of 1000, providing positive feedback to the player.
* @param score new total score value
public void setScore(int score) {
this.currentScore = score;
scoreLabel.setText("Score: " + score);
if (score > 0 && score % 1000 == 0) {
animateScoreLabel();
}
}
* Increases the current score by a given amount.
* @param points number of points to add
public void addScore(int points) { setScore(this.currentScore + points); }
* @return the current score value
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > ui > HUD > HUD
73:43 LF UTF-8 4 个空格

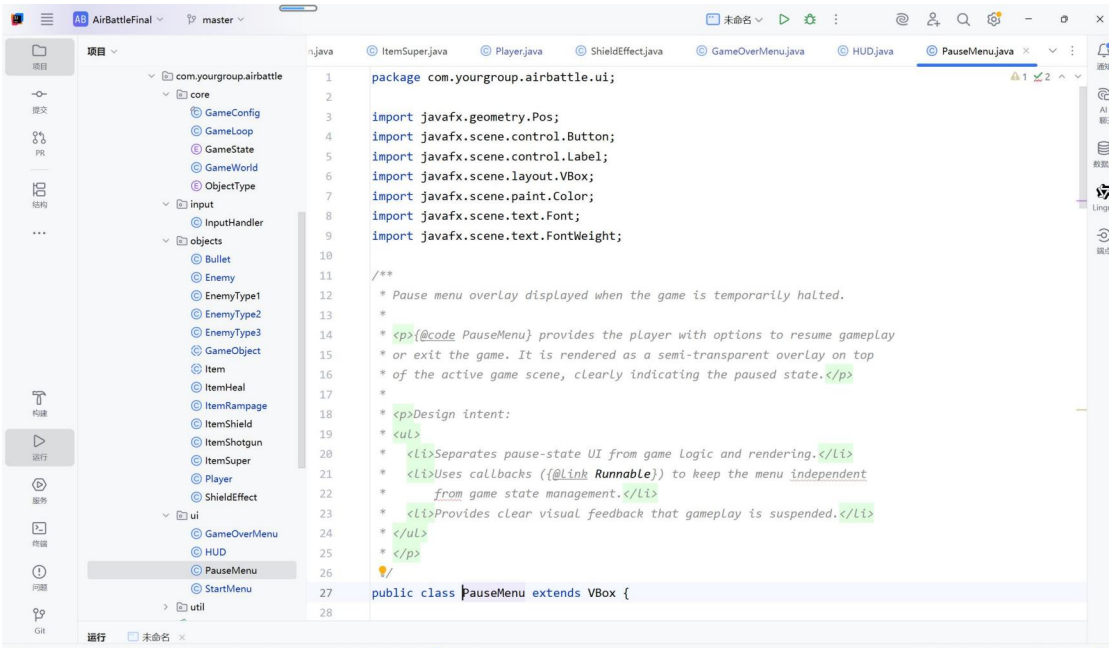


```
public class HUD extends VBox {
    public int getScore() { return currentScore; }

    /**
     * Resets the HUD to its initial state.
     *
     * <p>Used when restarting the game.</p>
     */
    public void reset() {
        setHp(DEFAULT_HP);
        setScore(0);
    }

    /**
     * Plays a short visual animation on the score label.
     *
     * <p>The label briefly enlarges and changes color to highlight
     * milestone achievements.</p>
     */
    private void animateScoreLabel() {
        scoreLabel.setScaleX(1.2);
        scoreLabel.setScaleY(1.2);
        scoreLabel.setTextFill(Color.GOLD);

        new Timeline(
                new KeyFrame(Duration.millis( v: 200), ActionEvent e -> {
```

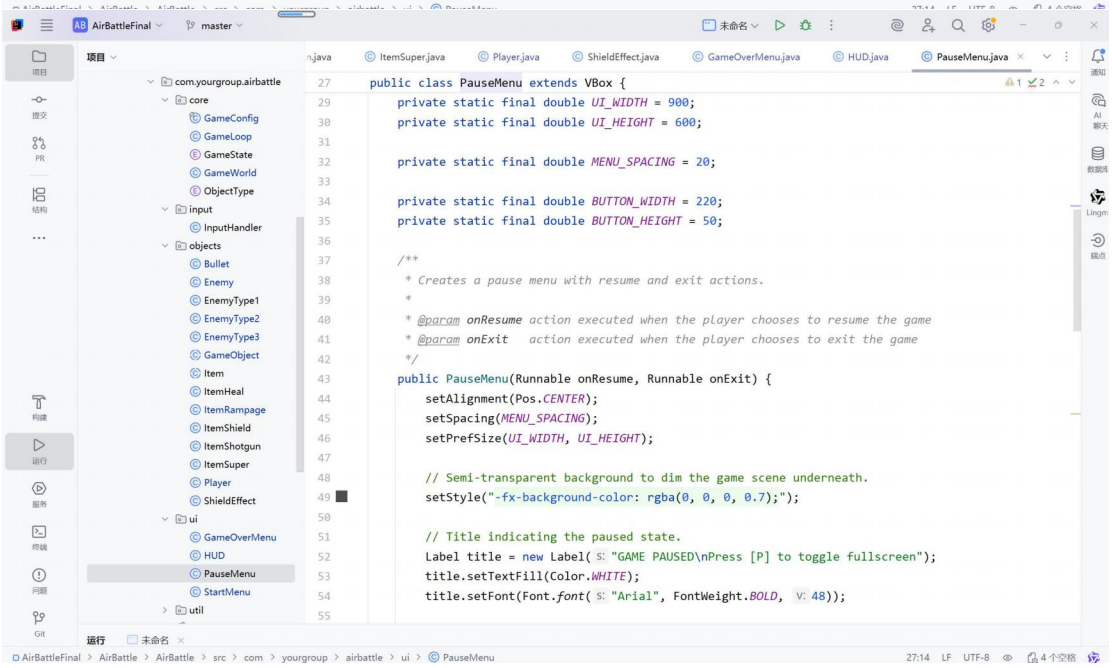


```
public class HUD extends VBox {

    /**
     * Plays a short visual animation on the score label.
     *
     * <p>The label briefly enlarges and changes color to highlight
     * milestone achievements.</p>
     */
    private void animateScoreLabel() {
        scoreLabel.setScaleX(1.2);
        scoreLabel.setScaleY(1.2);
        scoreLabel.setTextFill(Color.GOLD);

        new Timeline(
                new KeyFrame(Duration.millis( v: 200), ActionEvent e -> {
                    scoreLabel.setScaleX(1.0);
                    scoreLabel.setScaleY(1.0);
                    scoreLabel.setTextFill(Color.WHITE);
                })
        ).play();
    }
}
```

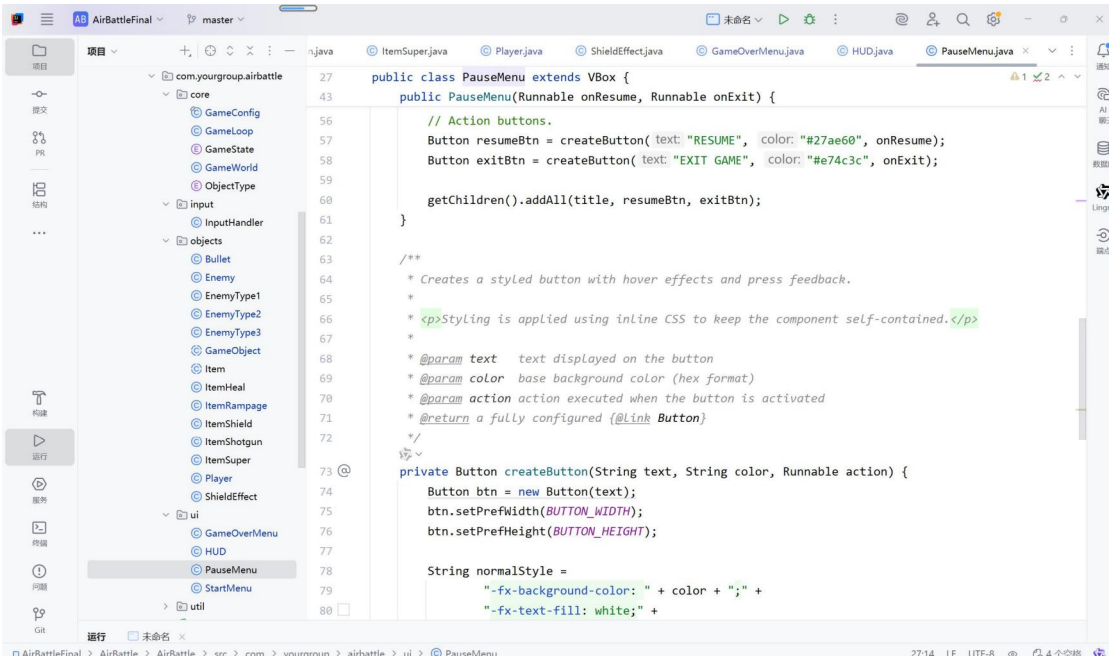
\PauseMenu.java

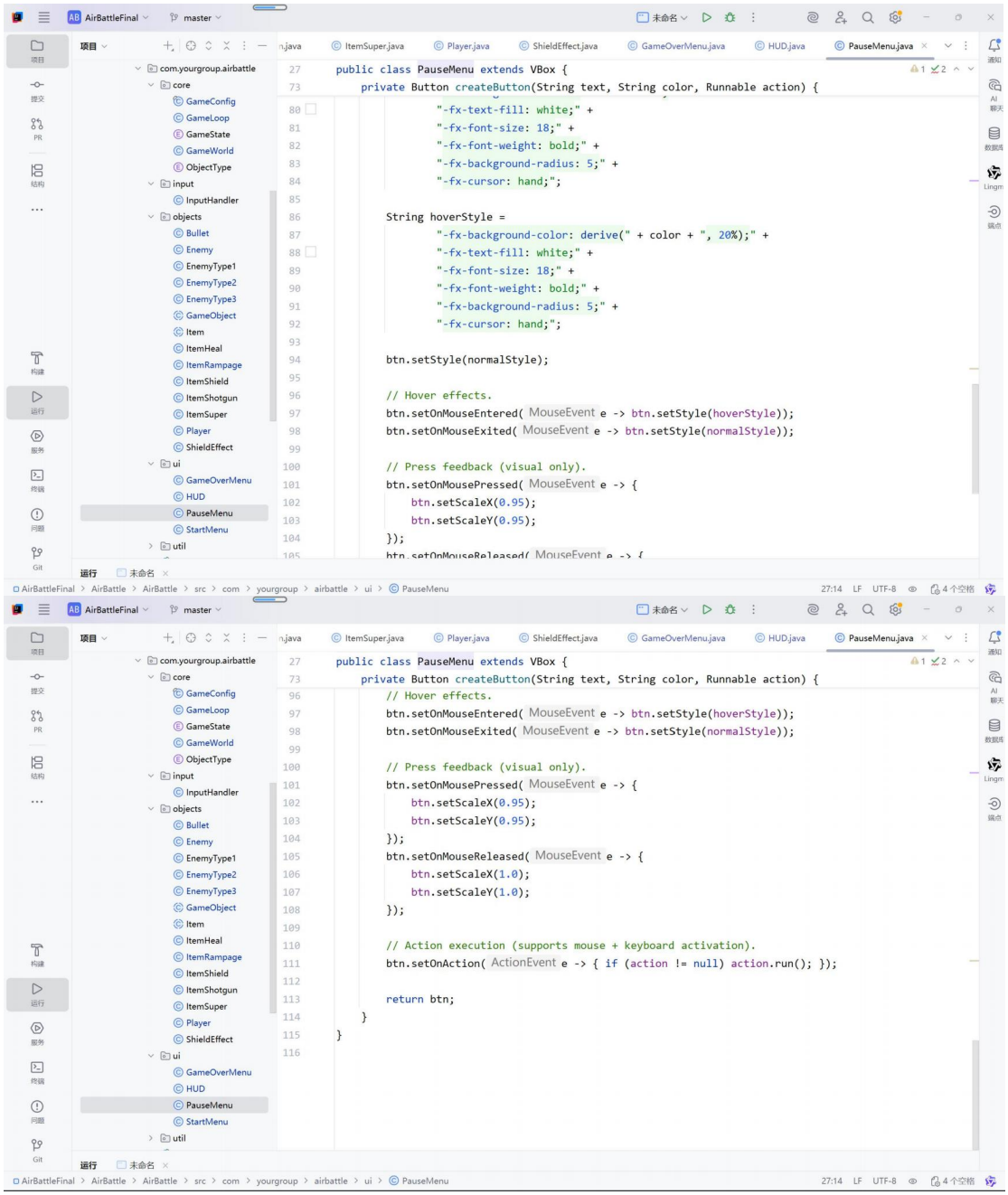


```java
package com.yourgroup.airbattle.ui;

import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

/**
 * Pause menu overlay displayed when the game is temporarily halted.
 *
 * <p>{@code PauseMenu} provides the player with options to resume gameplay
 * or exit the game. It is rendered as a semi-transparent overlay on top
 * of the active game scene, clearly indicating the paused state.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Separates pause-state UI from game logic and rendering.</li>
 *   <li>Uses callbacks ({@link Runnable}) to keep the menu independent
 *       from game state management.</li>
 *   <li>Provides clear visual feedback that gameplay is suspended.</li>
 * </ul>
 * </p>
 */
public class PauseMenu extends VBox {
```



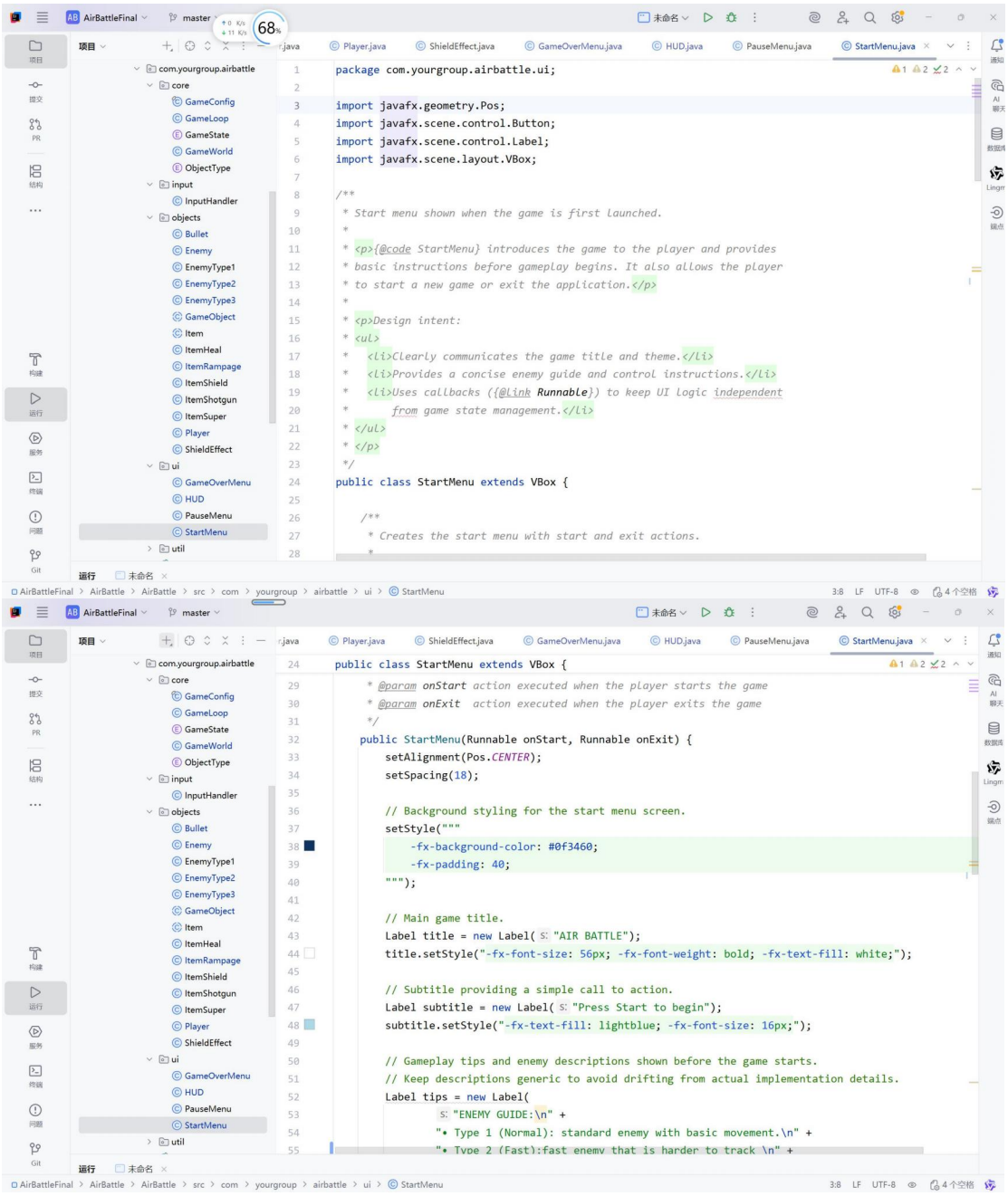
```java
public class PauseMenu extends VBox {
    private static final double UI_WIDTH = 900;
    private static final double UI_HEIGHT = 600;

    private static final double MENU_SPACING = 20;

    private static final double BUTTON_WIDTH = 220;
    private static final double BUTTON_HEIGHT = 50;

    /**
     * Creates a pause menu with resume and exit actions.
     *
     * @param onResume action executed when the player chooses to resume the game
     * @param onExit   action executed when the player chooses to exit the game
     */
    public PauseMenu(Runnable onResume, Runnable onExit) {
        setAlignment(Pos.CENTER);
        setSpacing(MENU_SPACING);
        setPrefSize(UI_WIDTH, UI_HEIGHT);

        // Semi-transparent background to dim the game scene underneath.
        setStyle("-fx-background-color: rgba(0, 0, 0, 0.7);");

        // Title indicating the paused state.
        Label title = new Label( s: "GAME PAUSED\nPress [P] to toggle fullscreen");
        title.setTextFill(Color.WHITE);
        title.setFont(Font.font( s: "Arial", FontWeight.BOLD, v: 48));
```


AirBattleFinal
master
ItemSuper.java
Player.java
ShieldEffect.java
GameOverMenu.java
HUD.java
PauseMenu.java
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
public class PauseMenu extends VBox {
public PauseMenu(Runnable onResume, Runnable onExit) {
// Action buttons.
Button resumeBtn = createButton(text: "RESUME", color: "#27ae60", onResume);
Button exitBtn = createButton(text: "EXIT GAME", color: "#e74c3c", onExit);
getChildren().addAll(title, resumeBtn, exitBtn);
}
/**
* Creates a styled button with hover effects and press feedback.
*
* Styling is applied using inline CSS to keep the component self-contained.
*
* @param text text displayed on the button
* @param color base background color (hex format)
* @param action action executed when the button is activated
* @return a fully configured {@link Button}
*/
private Button createButton(String text, String color, Runnable action) {
Button btn = new Button(text);
btn.setPrefWidth(BUTTON_WIDTH);
btn.setPrefHeight(BUTTON_HEIGHT);
String normalStyle =
"-fx-background-color: " + color + ";" +
"-fx-text-fill: white;" +
运行
未命名

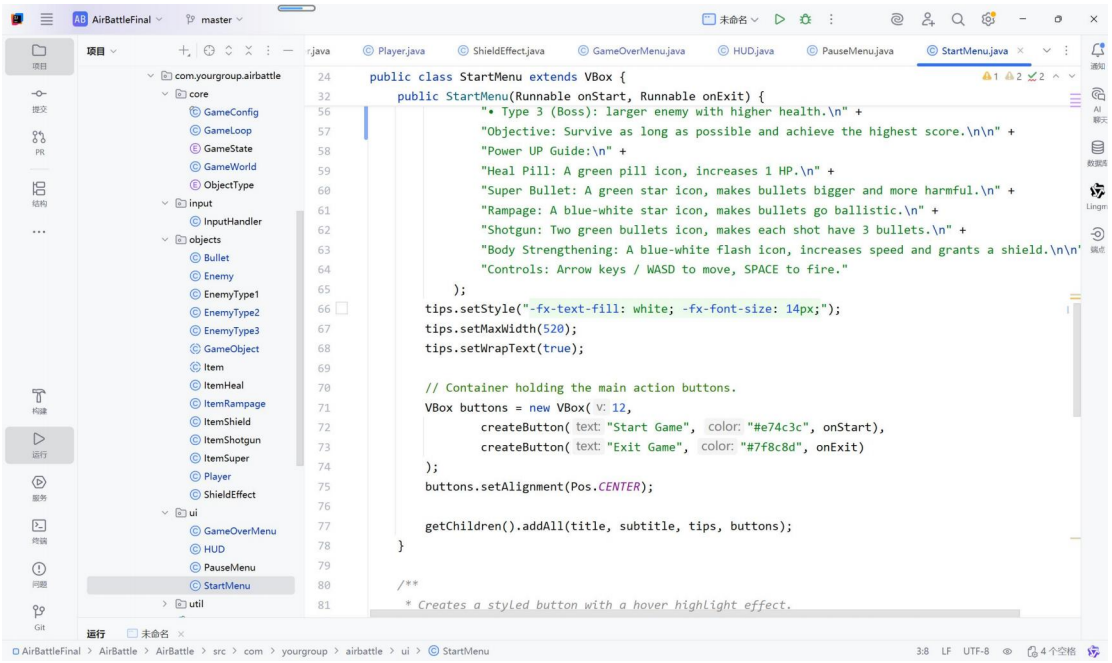


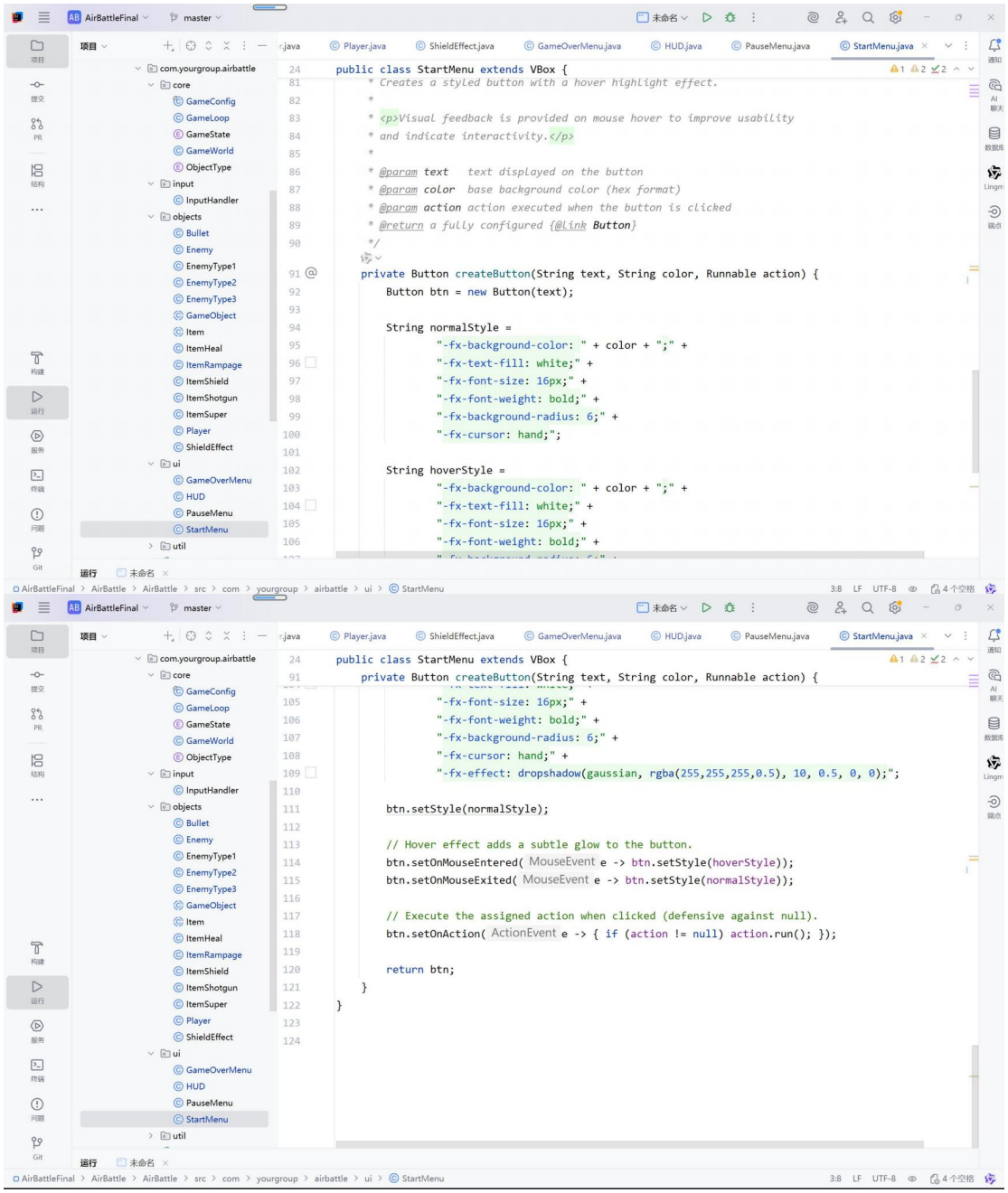
```
public class PauseMenu extends VBox {
    private Button createButton(String text, String color, Runnable action) {
                "-fx-text-fill: white;" +
                "-fx-font-size: 18;" +
                "-fx-font-weight: bold;" +
                "-fx-background-radius: 5;" +
                "-fx-cursor: hand;";

        String hoverStyle =
                "-fx-background-color: derive(" + color + ", 20%);" +
                "-fx-text-fill: white;" +
                "-fx-font-size: 18;" +
                "-fx-font-weight: bold;" +
                "-fx-background-radius: 5;" +
                "-fx-cursor: hand;";

        btn.setStyle(normalStyle);

        // Hover effects.
        btn.setOnMouseEntered( MouseEvent e -> btn.setStyle(hoverStyle));
        btn.setOnMouseExited( MouseEvent e -> btn.setStyle(normalStyle));

        // Press feedback (visual only).
        btn.setOnMousePressed( MouseEvent e -> {
            btn.setScaleX(0.95);
            btn.setScaleY(0.95);
        });
        btn.setOnMouseReleased( MouseEvent e -> {
            btn.setScaleX(1.0);
            btn.setScaleY(1.0);
        });

        // Action execution (supports mouse + keyboard activation).
        btn.setOnAction( ActionEvent e -> { if (action != null) action.run(); });

        return btn;
    }
}
```

\StartMenu.java



```java
package com.yourgroup.airbattle.ui;

import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;

/**
 * Start menu shown when the game is first launched.
 *
 * <p>{@code StartMenu} introduces the game to the player and provides
 * basic instructions before gameplay begins. It also allows the player
 * to start a new game or exit the application.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Clearly communicates the game title and theme.</li>
 *   <li>Provides a concise enemy guide and control instructions.</li>
 *   <li>Uses callbacks ({@link Runnable}) to keep UI logic independent
 *       from game state management.</li>
 * </ul>
 * </p>
 */
public class StartMenu extends VBox {

    /**
     * Creates the start menu with start and exit actions.
     *
     * @param onStart action executed when the player starts the game
     * @param onExit  action executed when the player exits the game
     */
    public StartMenu(Runnable onStart, Runnable onExit) {
        setAlignment(Pos.CENTER);
        setSpacing(18);

        // Background styling for the start menu screen.
        setStyle("""
                -fx-background-color: #0f3460;
                -fx-padding: 40;
        """);

        // Main game title.
        Label title = new Label( s: "AIR BATTLE");
        title.setStyle("-fx-font-size: 56px; -fx-font-weight: bold; -fx-text-fill: white;");

        // Subtitle providing a simple call to action.
        Label subtitle = new Label( s: "Press Start to begin");
        subtitle.setStyle("-fx-text-fill: lightblue; -fx-font-size: 16px;");

        // Gameplay tips and enemy descriptions shown before the game starts.
        // Keep descriptions generic to avoid drifting from actual implementation details.
        Label tips = new Label(
                s: "ENEMY GUIDE:\n" +
                "• Type 1 (Normal): standard enemy with basic movement.\n" +
                "• Type 2 (Fast):fast enemy that is harder to track \n" +
```

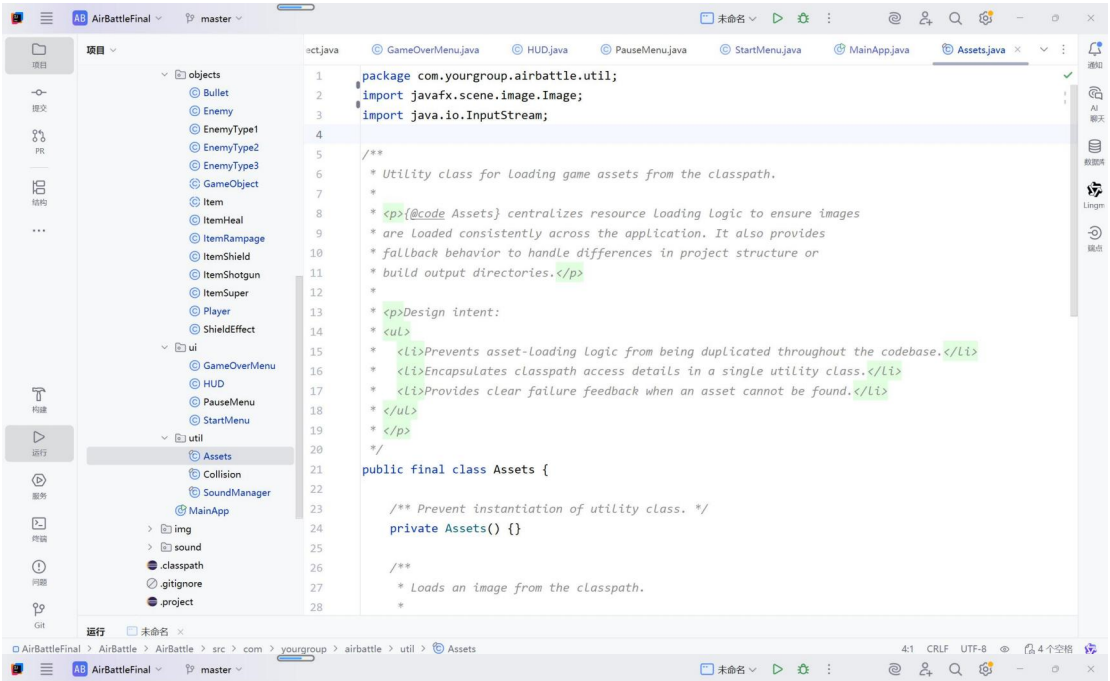

AirBattleFinal
master
Player.java
ShieldEffect.java
GameOverMenu.java
HUD.java
PauseMenu.java
StartMenu.java
com.yourgroup.airbattle
core
GameConfig
GameLoop
GameState
GameWorld
ObjectType
input
InputHandler
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
public class StartMenu extends VBox {
public StartMenu(Runnable onStart, Runnable onExit) {
"• Type 3 (Boss): larger enemy with higher health.\n" +
"Objective: Survive as long as possible and achieve the highest score.\n\n" +
"Power UP Guide:\n" +
"Heal Pill: A green pill icon, increases 1 HP.\n" +
"Super Bullet: A green star icon, makes bullets bigger and more harmful.\n" +
"Rampage: A blue-white star icon, makes bullets go ballistic.\n" +
"Shotgun: Two green bullets icon, makes each shot have 3 bullets.\n" +
"Body Strengthening: A blue-white flash icon, increases speed and grants a shield.\n\n"
"Controls: Arrow keys / WASD to move, SPACE to fire."
);
tips.setStyle("-fx-text-fill: white; -fx-font-size: 14px;");
tips.setMaxWidth(520);
tips.setWrapText(true);
// Container holding the main action buttons.
VBox buttons = new VBox(v: 12,
createButton(text: "Start Game", color: "#e74c3c", onStart),
createButton(text: "Exit Game", color: "#7f8c8d", onExit)
);
buttons.setAlignment(Pos.CENTER);
getChildren().addAll(title, subtitle, tips, buttons);
}
/**
* Creates a styled button with a hover highlight effect.
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > ui > StartMenu
3:8
LF
UTF-8
4 个空格



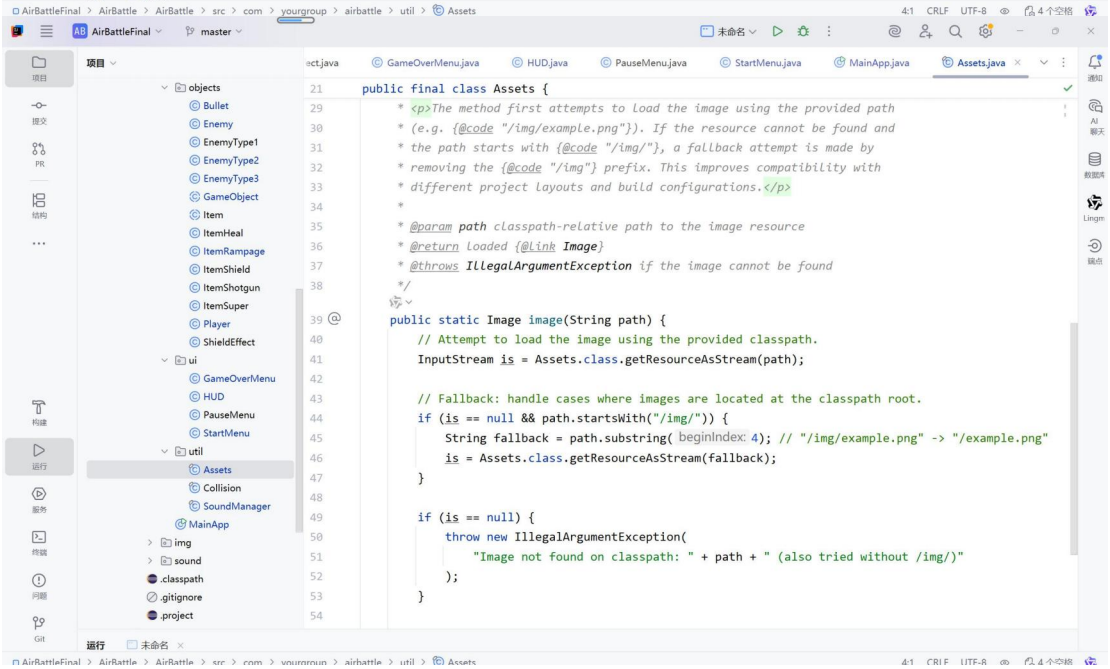
```
public class StartMenu extends VBox {
    * Creates a styled button with a hover highlight effect.
    *
    * <p>Visual feedback is provided on mouse hover to improve usability
    * and indicate interactivity.</p>
    *
    * @param text   text displayed on the button
    * @param color  base background color (hex format)
    * @param action action executed when the button is clicked
    * @return a fully configured {@Link Button}
    */
    private Button createButton(String text, String color, Runnable action) {
        Button btn = new Button(text);

        String normalStyle =
                "-fx-background-color: " + color + ";" +
                "-fx-text-fill: white;" +
                "-fx-font-size: 16px;" +
                "-fx-font-weight: bold;" +
                "-fx-background-radius: 6;" +
                "-fx-cursor: hand;";

        String hoverStyle =
                "-fx-background-color: " + color + ";" +
                "-fx-text-fill: white;" +
                "-fx-font-size: 16px;" +
                "-fx-font-weight: bold;" +
                "-fx-font-size: 16px;" +
                "-fx-font-weight: bold;" +
                "-fx-background-radius: 6;" +
                "-fx-cursor: hand;" +
                "-fx-effect: dropshadow(gaussian, rgba(255,255,255,0.5), 10, 0.5, 0, 0);";

        btn.setStyle(normalStyle);

        // Hover effect adds a subtle glow to the button.
        btn.setOnMouseEntered( MouseEvent e -> btn.setStyle(hoverStyle));
        btn.setOnMouseExited( MouseEvent e -> btn.setStyle(normalStyle));

        // Execute the assigned action when clicked (defensive against null).
        btn.setOnAction( ActionEvent e -> { if (action != null) action.run(); });

        return btn;
    }
}
```

AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\util

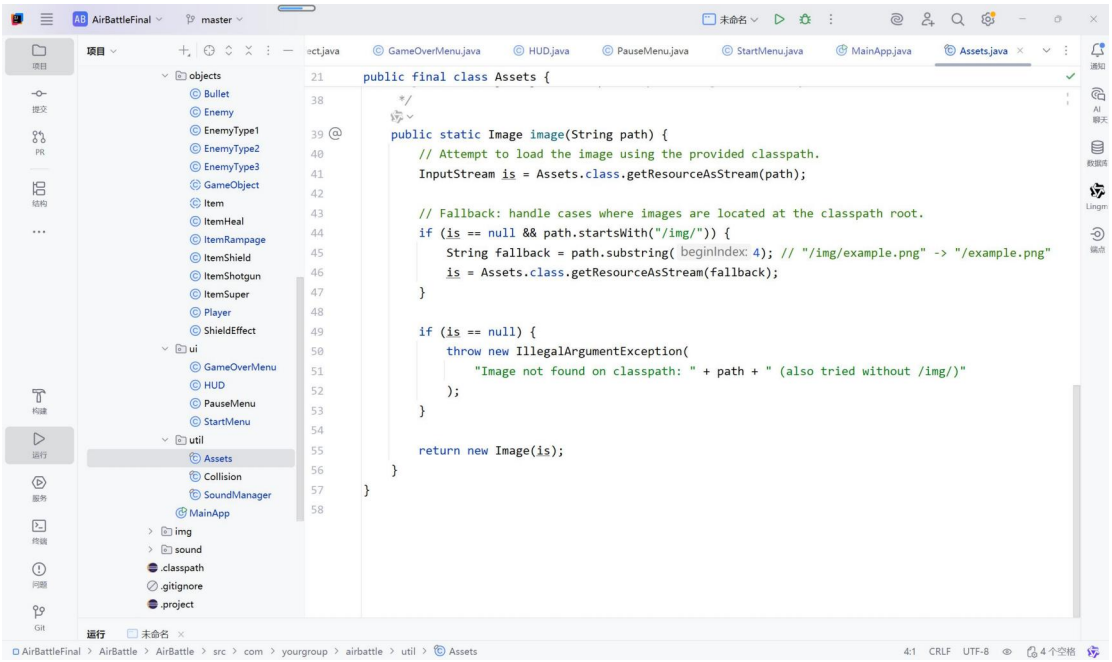
\Assets.java



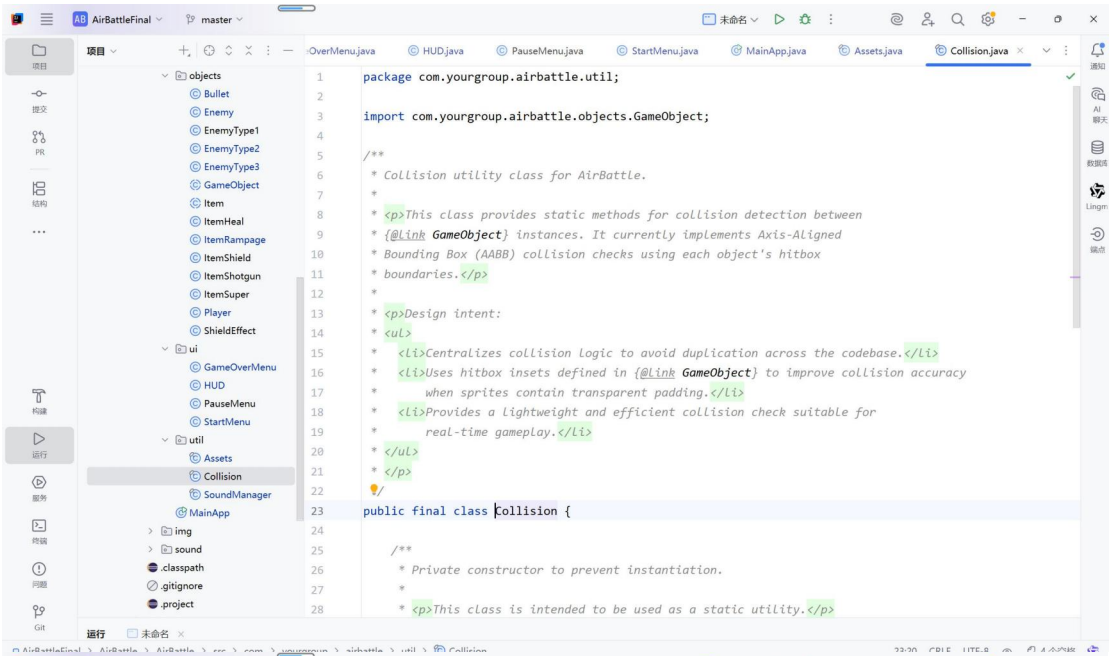
```
package com.yourgroup.airbattle.util;
import javafx.scene.image.Image;
import java.io.InputStream;

/**
 * Utility class for loading game assets from the classpath.
 *
 * <p>{@code Assets} centralizes resource loading logic to ensure images
 * are loaded consistently across the application. It also provides
 * fallback behavior to handle differences in project structure or
 * build output directories.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Prevents asset-loading logic from being duplicated throughout the codebase.</li>
 *   <li>Encapsulates classpath access details in a single utility class.</li>
 *   <li>Provides clear failure feedback when an asset cannot be found.</li>
 * </ul>
 * </p>
 */
public final class Assets {

    /** Prevent instantiation of utility class. */
    private Assets() {}

    /**
     * Loads an image from the classpath.
     *
```



```
public final class Assets {
    * <p>The method first attempts to load the image using the provided path
    * (e.g. {@code "/img/example.png"}). If the resource cannot be found and
    * the path starts with {@code "/img/"}, a fallback attempt is made by
    * removing the {@code "/img"} prefix. This improves compatibility with
    * different project layouts and build configurations.</p>
    *
    * @param path classpath-relative path to the image resource
    * @return loaded {@link Image}
    * @throws IllegalArgumentException if the image cannot be found
    */
   public static Image image(String path) {
       // Attempt to load the image using the provided classpath.
       InputStream is = Assets.class.getResourceAsStream(path);

       // Fallback: handle cases where images are located at the classpath root.
       if (is == null && path.startsWith("/img/")) {
           String fallback = path.substring( beginIndex: 4); // "/img/example.png" -> "/example.png"
           is = Assets.class.getResourceAsStream(fallback);
       }

       if (is == null) {
           throw new IllegalArgumentException(
               "Image not found on classpath: " + path + " (also tried without /img/)"
           );
       }
```

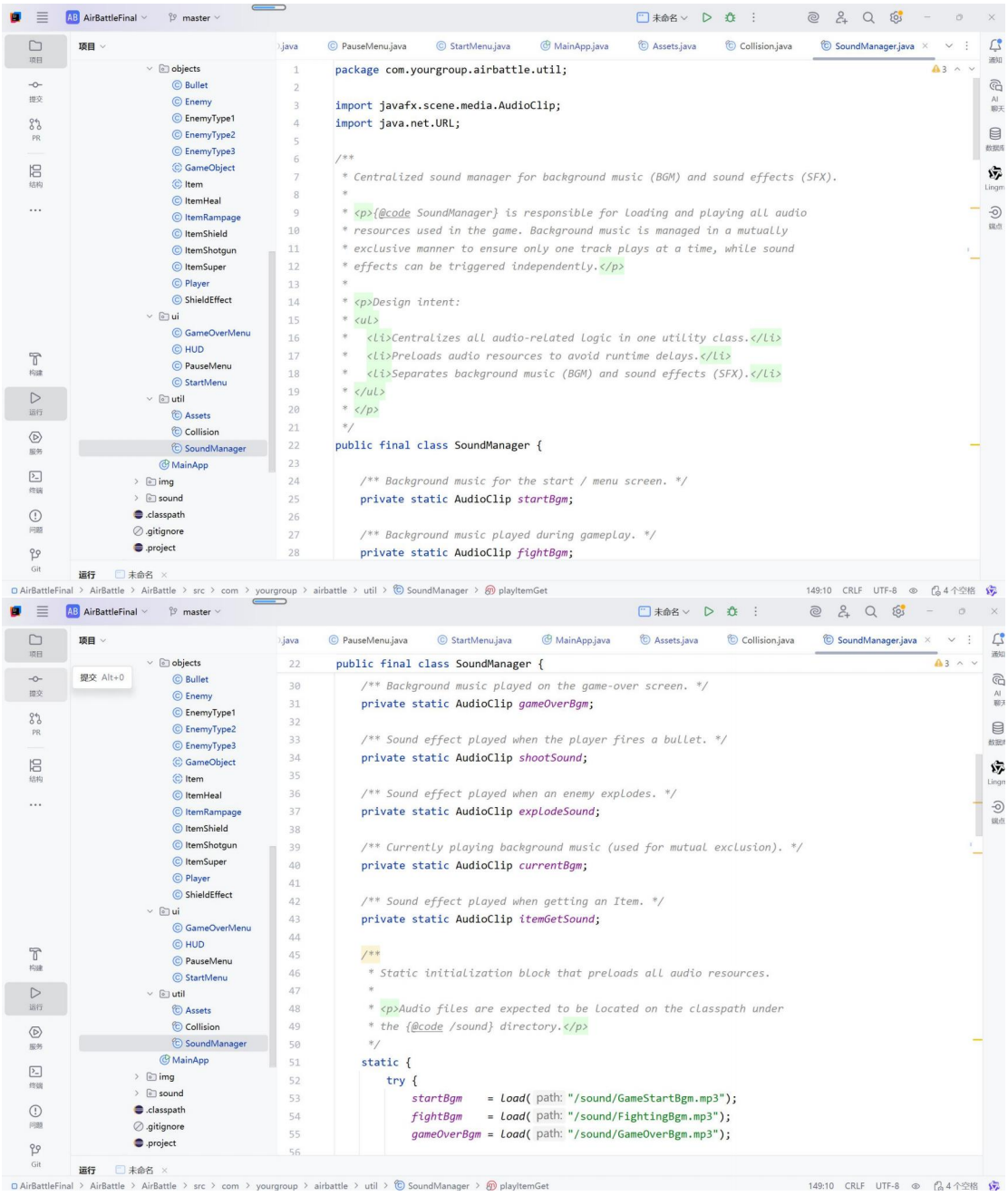

AirBattleFinal
master
未命名
项目
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
Assets
Collision
SoundManager
MainApp
img
sound
.classpath
.gitignore
.project
GameOverMenu.java
HUD.java
PauseMenu.java
StartMenu.java
MainApp.java
Assets.java
public final class Assets {
*/
public static Image image(String path) {
// Attempt to load the image using the provided classpath.
InputStream is = Assets.class.getResourceAsStream(path);
// Fallback: handle cases where images are located at the classpath root.
if (is == null && path.startsWith("/img/")) {
String fallback = path.substring(beginIndex: 4); // "/img/example.png" -> "/example.png"
is = Assets.class.getResourceAsStream(fallback);
}
if (is == null) {
throw new IllegalArgumentException(
"Image not found on classpath: " + path + " (also tried without /img/)"
);
}
return new Image(is);
}
}
运行
未命名
AirBattleFinal › AirBattle › AirBattle › src › com › yourgroup › airbattle › util › Assets
4:1
CRLF
UTF-8
4 个空格

\Collision.java

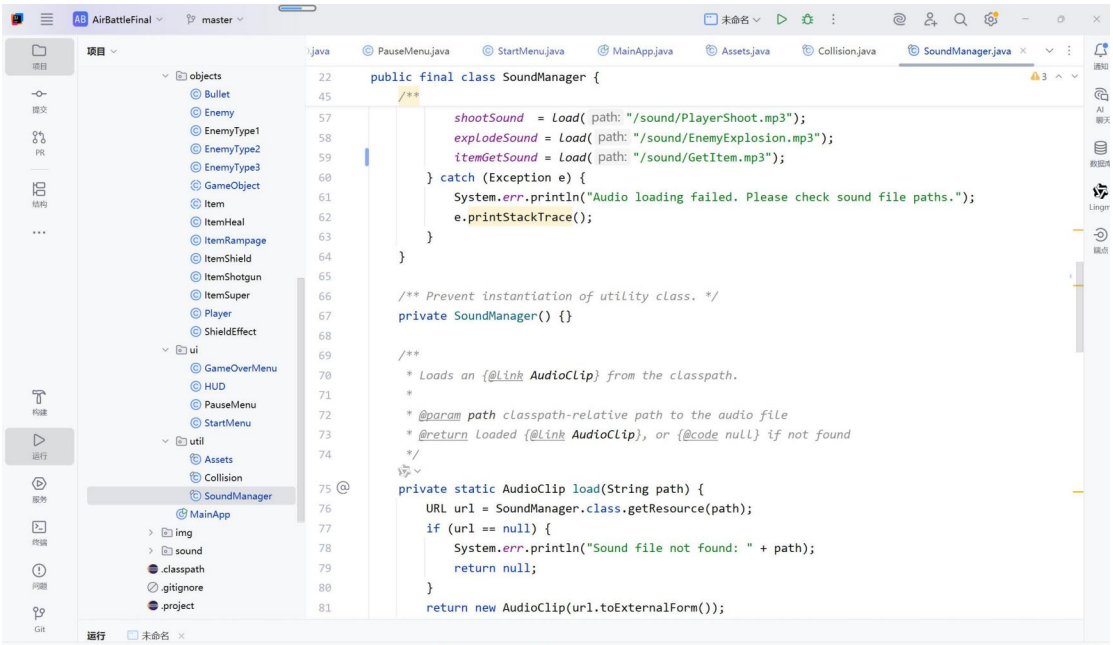


```
package com.yourgroup.airbattle.util;

import com.yourgroup.airbattle.objects.GameObject;

/**
 * Collision utility class for AirBattle.
 *
 * <p>This class provides static methods for collision detection between
 * {@link GameObject} instances. It currently implements Axis-Aligned
 * Bounding Box (AABB) collision checks using each object's hitbox
 * boundaries.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Centralizes collision logic to avoid duplication across the codebase.</li>
 *   <li>Uses hitbox insets defined in {@link GameObject} to improve collision accuracy
 *       when sprites contain transparent padding.</li>
 *   <li>Provides a lightweight and efficient collision check suitable for
 *       real-time gameplay.</li>
 * </ul>
 * </p>
 */
public final class Collision {

    /**
     * Private constructor to prevent instantiation.
     *
     * <p>This class is intended to be used as a static utility.</p>
     */
    private Collision() {}

    /**
     * Performs an Axis-Aligned Bounding Box (AABB) collision test
     * between two game objects.
     *
     * <p>The method checks whether the hitbox of object {@code a}
     * overlaps with the hitbox of object {@code b} in both the
     * horizontal and vertical axes.</p>
     *
     * <p>This collision test is symmetric:
     * {@code aabb(a, b)} produces the same result as {@code aabb(b, a)}.</p>
     *
     * @param a the first game object
     * @param b the second game object
     * @return true if the two objects' hitboxes intersect; false otherwise
     */
    public static boolean aabb(GameObject a, GameObject b) {
        return a.hitRight()  > b.hitLeft()
            && a.hitLeft()   < b.hitRight()
            && a.hitBottom() > b.hitTop()
            && a.hitTop()    < b.hitBottom();
    }
}
```

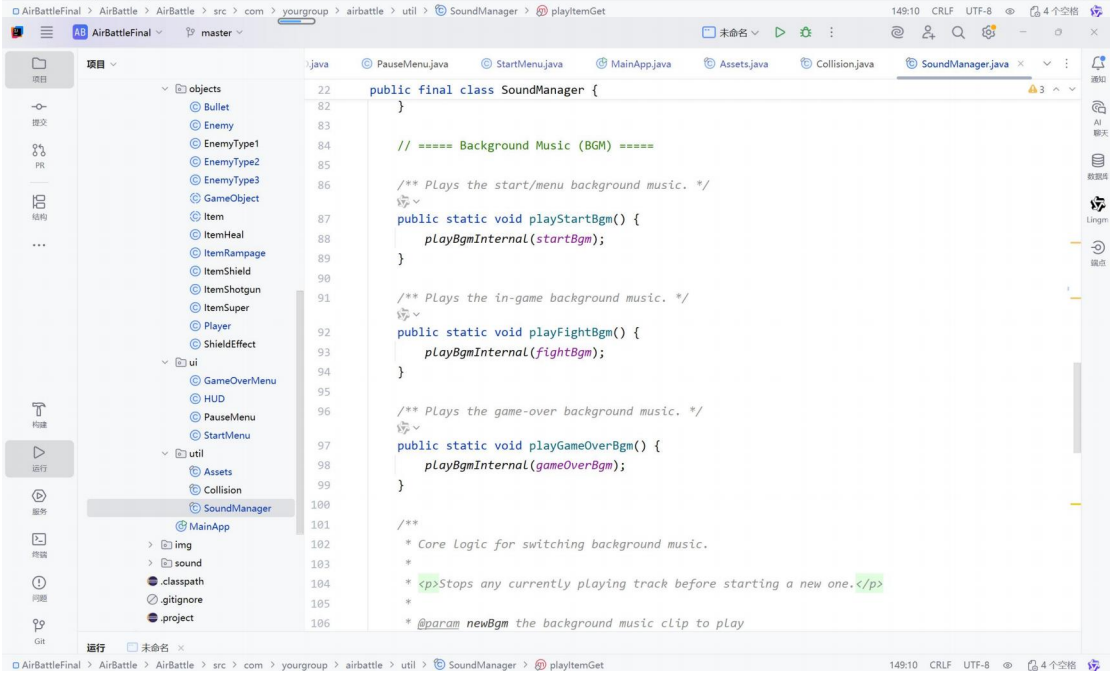
\SoundManager.java



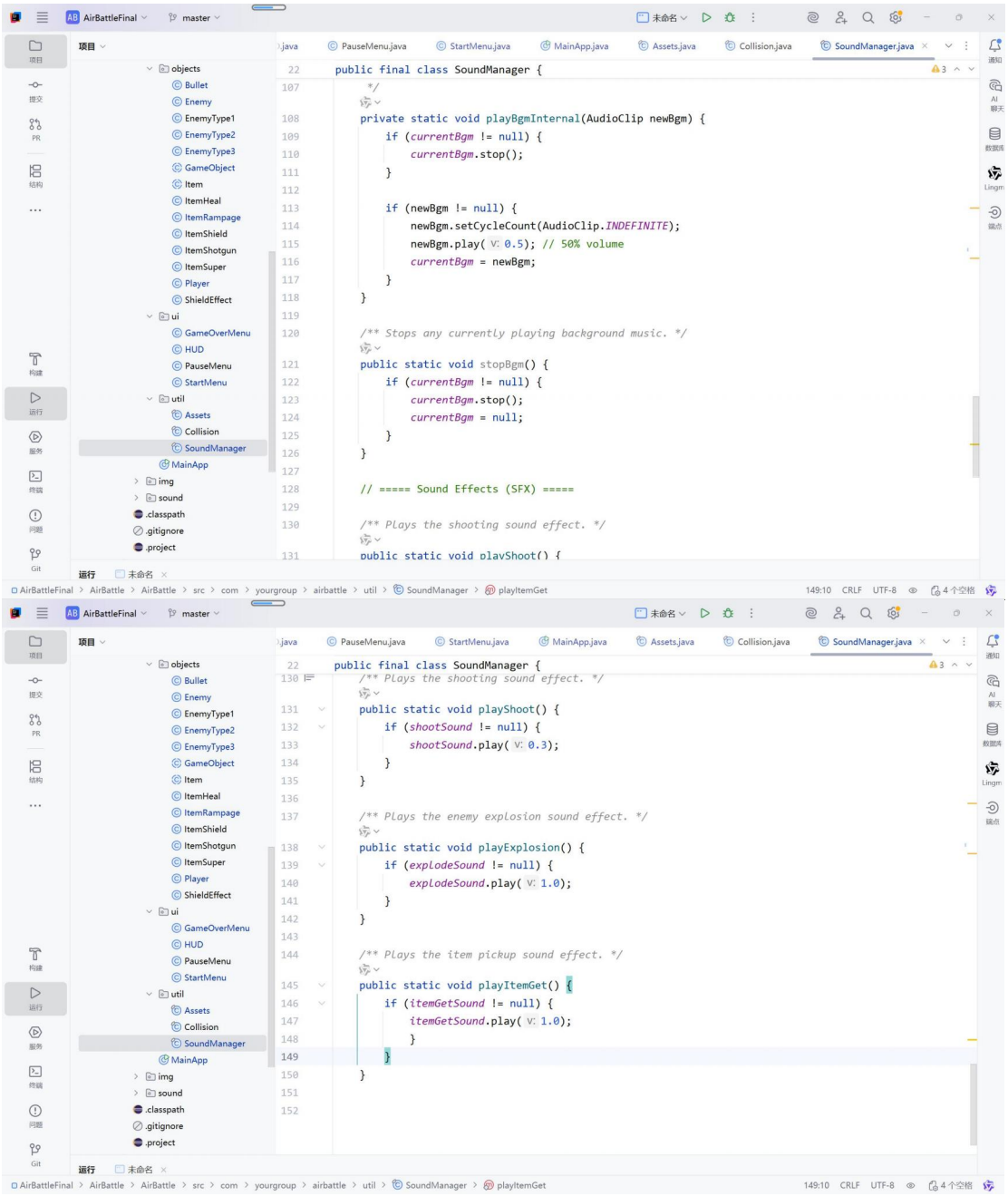
```java
package com.yourgroup.airbattle.util;

import javafx.scene.media.AudioClip;
import java.net.URL;

/**
 * Centralized sound manager for background music (BGM) and sound effects (SFX).
 *
 * <p>{@code SoundManager} is responsible for loading and playing all audio
 * resources used in the game. Background music is managed in a mutually
 * exclusive manner to ensure only one track plays at a time, while sound
 * effects can be triggered independently.</p>
 *
 * <p>Design intent:
 * <ul>
 *   <li>Centralizes all audio-related logic in one utility class.</li>
 *   <li>Preloads audio resources to avoid runtime delays.</li>
 *   <li>Separates background music (BGM) and sound effects (SFX).</li>
 * </ul>
 * </p>
 */
public final class SoundManager {

    /** Background music for the start / menu screen. */
    private static AudioClip startBgm;

    /** Background music played during gameplay. */
    private static AudioClip fightBgm;

    /** Background music played on the game-over screen. */
    private static AudioClip gameOverBgm;

    /** Sound effect played when the player fires a bullet. */
    private static AudioClip shootSound;

    /** Sound effect played when an enemy explodes. */
    private static AudioClip explodeSound;

    /** Currently playing background music (used for mutual exclusion). */
    private static AudioClip currentBgm;

    /** Sound effect played when getting an Item. */
    private static AudioClip itemGetSound;

    /**
     * Static initialization block that preloads all audio resources.
     *
     * <p>Audio files are expected to be located on the classpath under
     * the {@code /sound} directory.</p>
     */
    static {
        try {
            startBgm    = load( path: "/sound/GameStartBgm.mp3");
            fightBgm    = load( path: "/sound/FightingBgm.mp3");
            gameOverBgm = load( path: "/sound/GameOverBgm.mp3");
```



```
public final class SoundManager {
    /**
            shootSound  = load( path: "/sound/PlayerShoot.mp3");
            explodeSound = load( path: "/sound/EnemyExplosion.mp3");
            itemGetSound = load( path: "/sound/GetItem.mp3");
        } catch (Exception e) {
            System.err.println("Audio loading failed. Please check sound file paths.");
            e.printStackTrace();
        }
    }

    /** Prevent instantiation of utility class. */
    private SoundManager() {}

    /**
     * Loads an {@Link AudioClip} from the classpath.
     *
     * @param path classpath-relative path to the audio file
     * @return loaded {@Link AudioClip}, or {@code null} if not found
     */
    private static AudioClip load(String path) {
        URL url = SoundManager.class.getResource(path);
        if (url == null) {
            System.err.println("Sound file not found: " + path);
            return null;
        }
        return new AudioClip(url.toExternalForm());
```

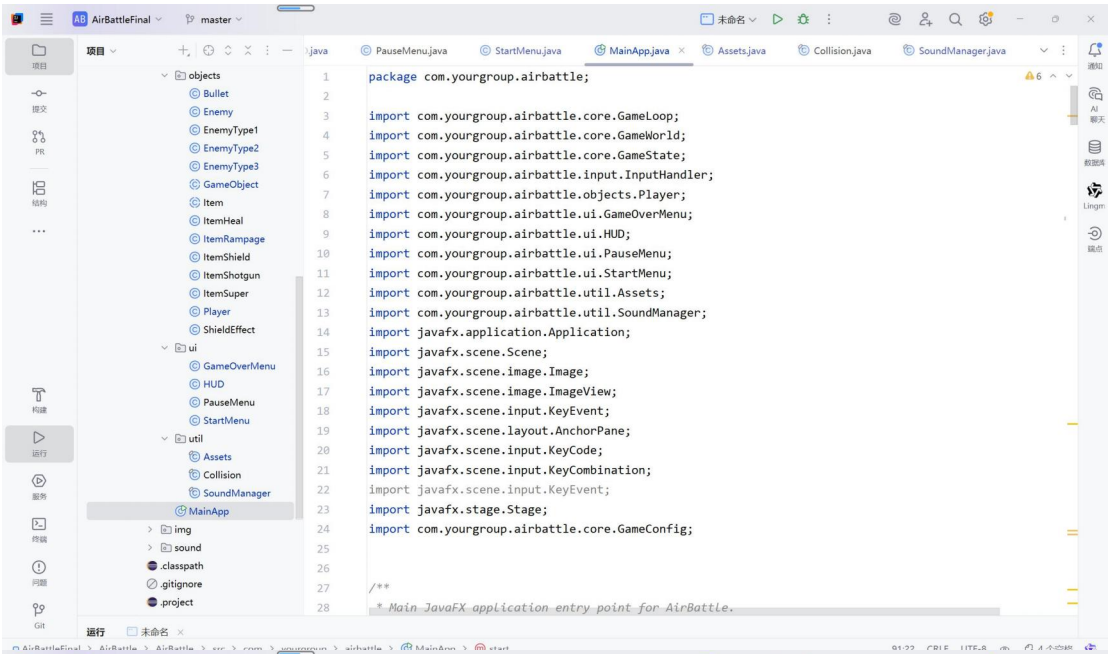


```
public final class SoundManager {
    }

    // ===== Background Music (BGM) =====

    /** Plays the start/menu background music. */
    public static void playStartBgm() {
        playBgmInternal(startBgm);
    }

    /** Plays the in-game background music. */
    public static void playFightBgm() {
        playBgmInternal(fightBgm);
    }

    /** Plays the game-over background music. */
    public static void playGameOverBgm() {
        playBgmInternal(gameOverBgm);
    }

    /**
     * Core logic for switching background music.
     *
     * <p>Stops any currently playing track before starting a new one.</p>
     *
     * @param newBgm the background music clip to play
```

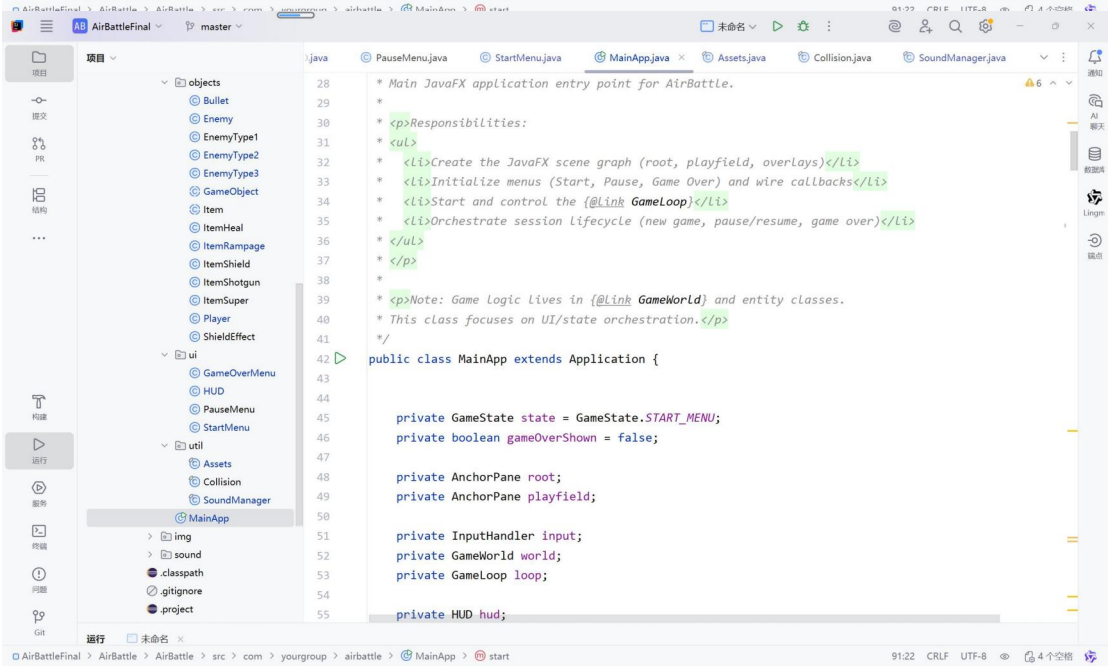


```
public final class SoundManager {
    */
    private static void playBgmInternal(AudioClip newBgm) {
        if (currentBgm != null) {
            currentBgm.stop();
        }

        if (newBgm != null) {
            newBgm.setCycleCount(AudioClip.INDEFINITE);
            newBgm.play( V: 0.5); // 50% volume
            currentBgm = newBgm;
        }
    }

    /** Stops any currently playing background music. */
    public static void stopBgm() {
        if (currentBgm != null) {
            currentBgm.stop();
            currentBgm = null;
        }
    }

    // ===== Sound Effects (SFX) =====

    /** Plays the shooting sound effect. */
    public static void playShoot() {
        if (shootSound != null) {
            shootSound.play( V: 0.3);
        }
    }

    /** Plays the enemy explosion sound effect. */
    public static void playExplosion() {
        if (explodeSound != null) {
            explodeSound.play( V: 1.0);
        }
    }

    /** Plays the item pickup sound effect. */
    public static void playItemGet() {
        if (itemGetSound != null) {
            itemGetSound.play( V: 1.0);
        }
    }
```

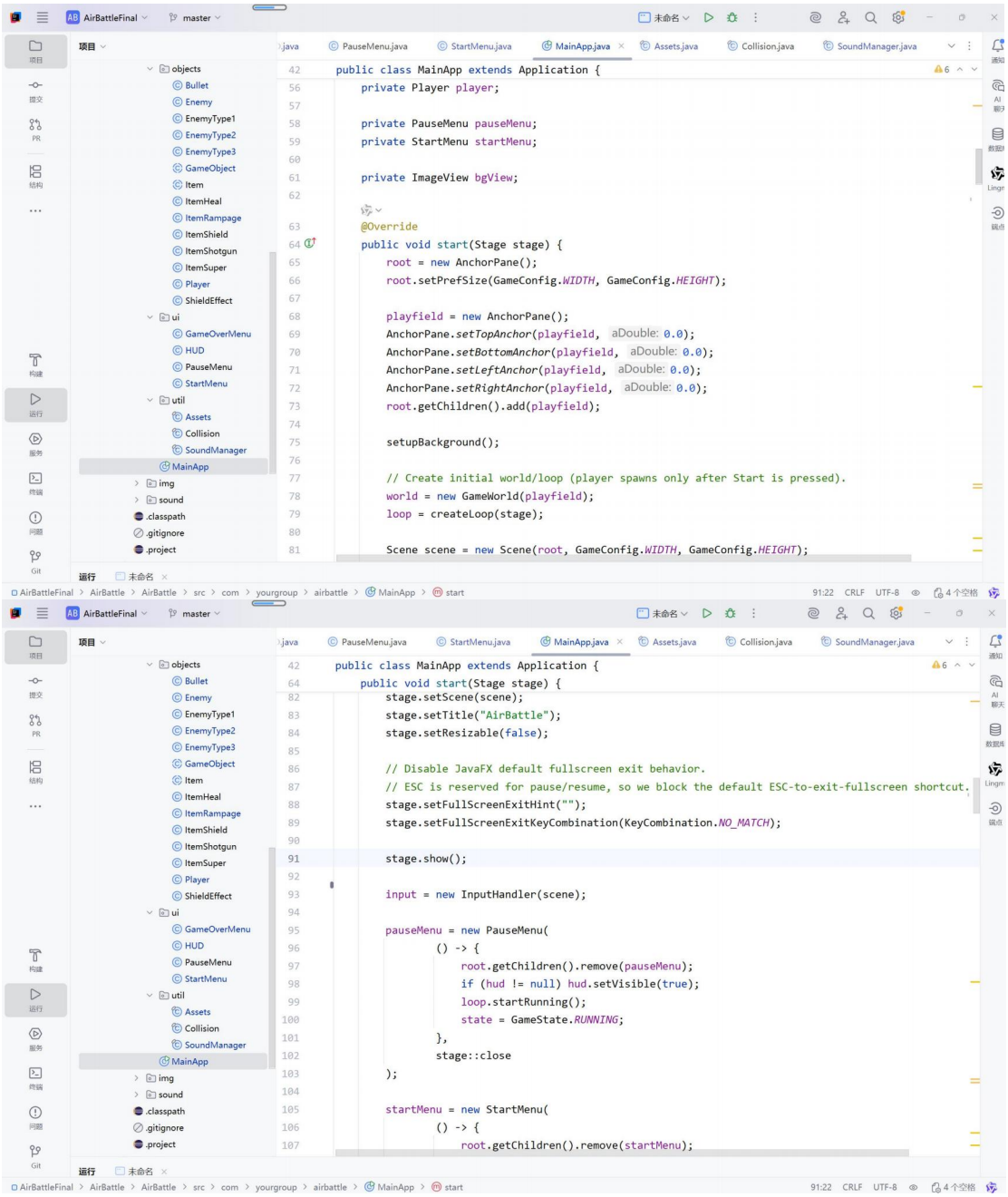
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\MainApp.java



```
package com.yourgroup.airbattle;

import com.yourgroup.airbattle.core.GameLoop;
import com.yourgroup.airbattle.core.GameWorld;
import com.yourgroup.airbattle.core.GameState;
import com.yourgroup.airbattle.input.InputHandler;
import com.yourgroup.airbattle.objects.Player;
import com.yourgroup.airbattle.ui.GameOverMenu;
import com.yourgroup.airbattle.ui.HUD;
import com.yourgroup.airbattle.ui.PauseMenu;
import com.yourgroup.airbattle.ui.StartMenu;
import com.yourgroup.airbattle.util.Assets;
import com.yourgroup.airbattle.util.SoundManager;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyCombination;
import javafx.scene.input.KeyEvent;
import javafx.stage.Stage;
import com.yourgroup.airbattle.core.GameConfig;


/**
 * Main JavaFX application entry point for AirBattle.
```

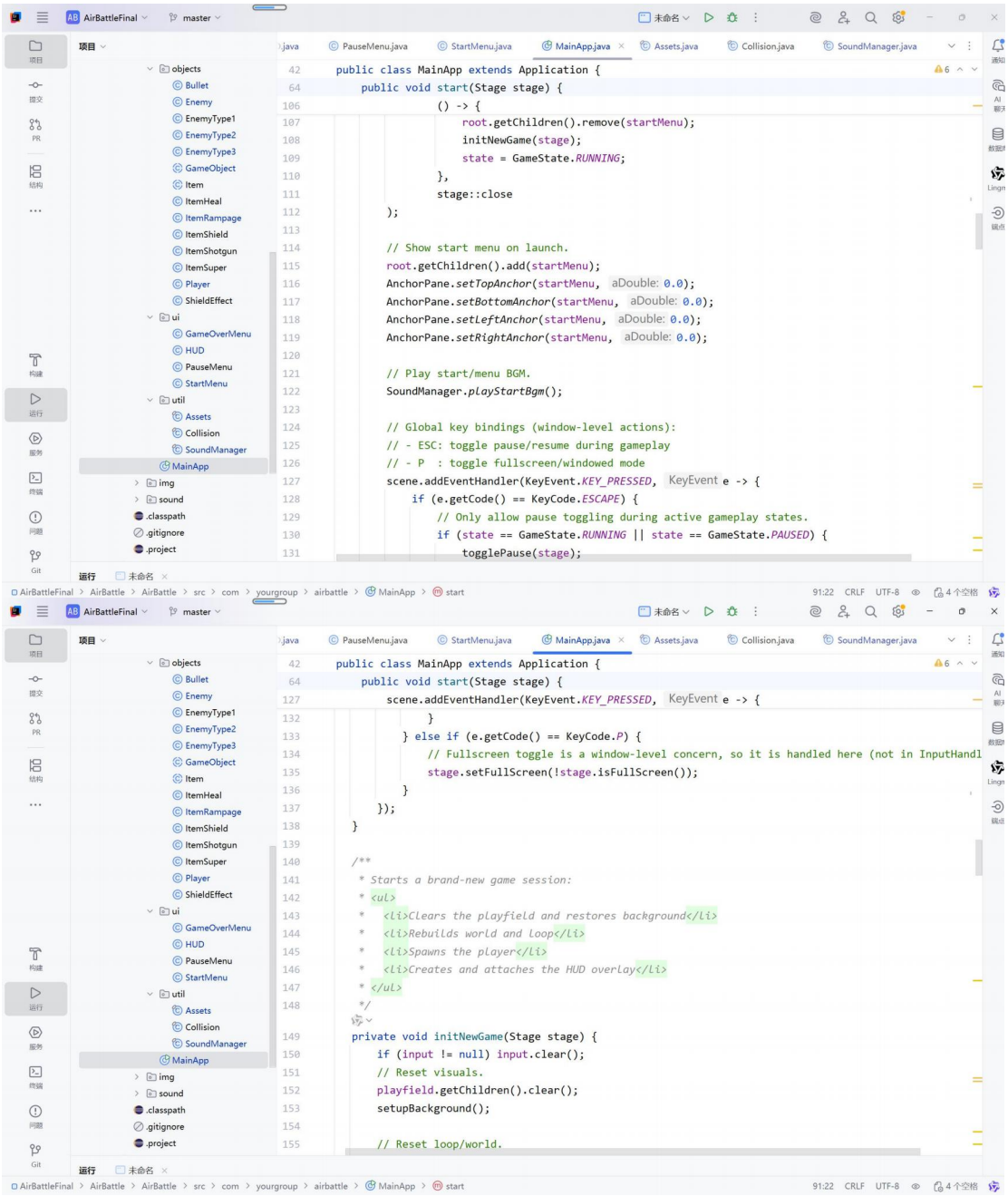


```
 * Main JavaFX application entry point for AirBattle.
 *
 * <p>Responsibilities:
 * <ul>
 *   <li>Create the JavaFX scene graph (root, playfield, overlays)</li>
 *   <li>Initialize menus (Start, Pause, Game Over) and wire callbacks</li>
 *   <li>Start and control the {@link GameLoop}</li>
 *   <li>Orchestrate session lifecycle (new game, pause/resume, game over)</li>
 * </ul>
 * </p>
 *
 * <p>Note: Game logic lives in {@link GameWorld} and entity classes.
 * This class focuses on UI/state orchestration.</p>
 */
public class MainApp extends Application {


    private GameState state = GameState.START_MENU;
    private boolean gameOverShown = false;

    private AnchorPane root;
    private AnchorPane playfield;

    private InputHandler input;
    private GameWorld world;
    private GameLoop loop;

    private HUD hud;
```



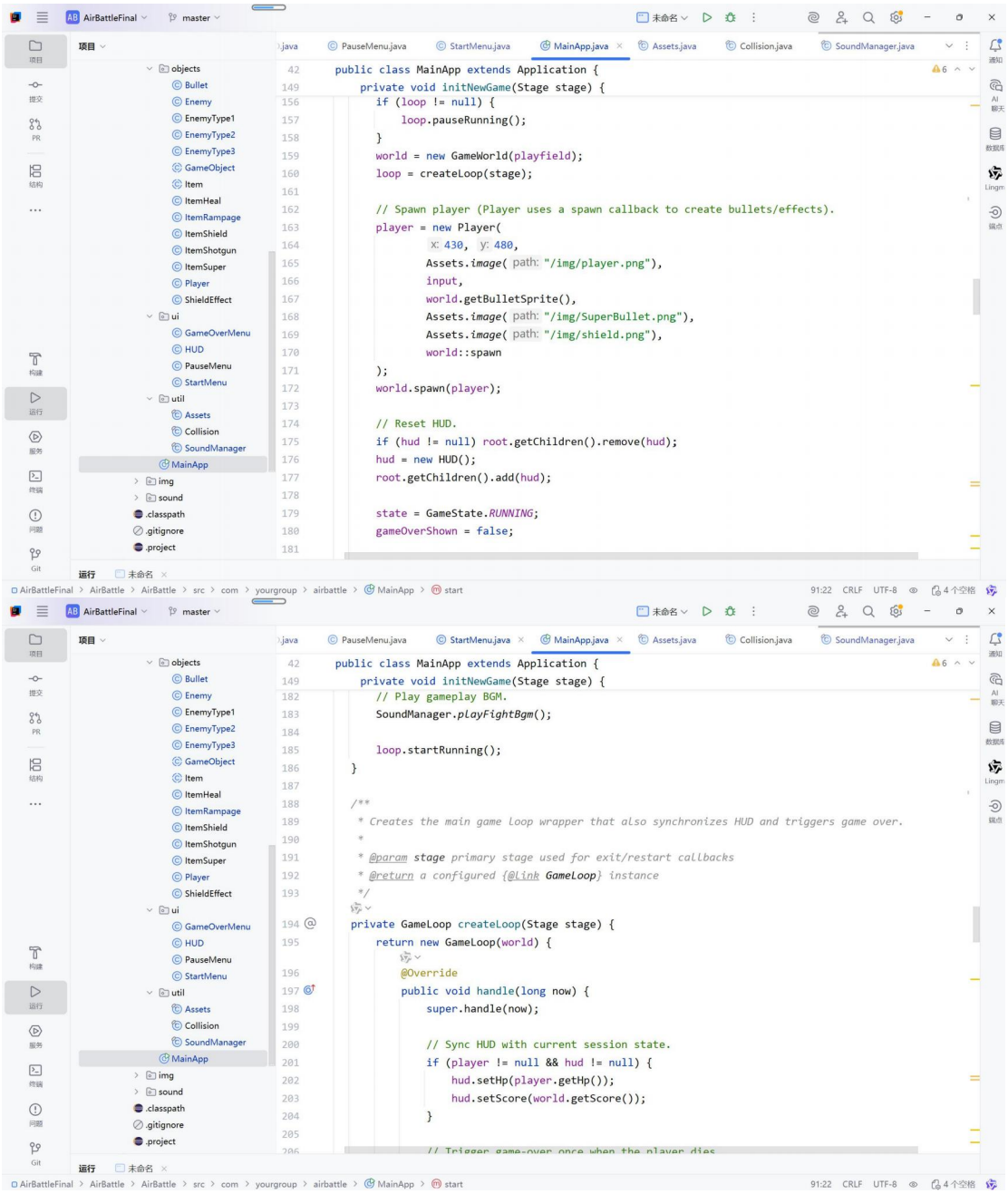
```
public class MainApp extends Application {
    private Player player;

    private PauseMenu pauseMenu;
    private StartMenu startMenu;

    private ImageView bgView;

    @Override
    public void start(Stage stage) {
        root = new AnchorPane();
        root.setPrefSize(GameConfig.WIDTH, GameConfig.HEIGHT);

        playfield = new AnchorPane();
        AnchorPane.setTopAnchor(playfield, aDouble: 0.0);
        AnchorPane.setBottomAnchor(playfield, aDouble: 0.0);
        AnchorPane.setLeftAnchor(playfield, aDouble: 0.0);
        AnchorPane.setRightAnchor(playfield, aDouble: 0.0);
        root.getChildren().add(playfield);

        setupBackground();

        // Create initial world/loop (player spawns only after Start is pressed).
        world = new GameWorld(playfield);
        loop = createLoop(stage);

        Scene scene = new Scene(root, GameConfig.WIDTH, GameConfig.HEIGHT);
```

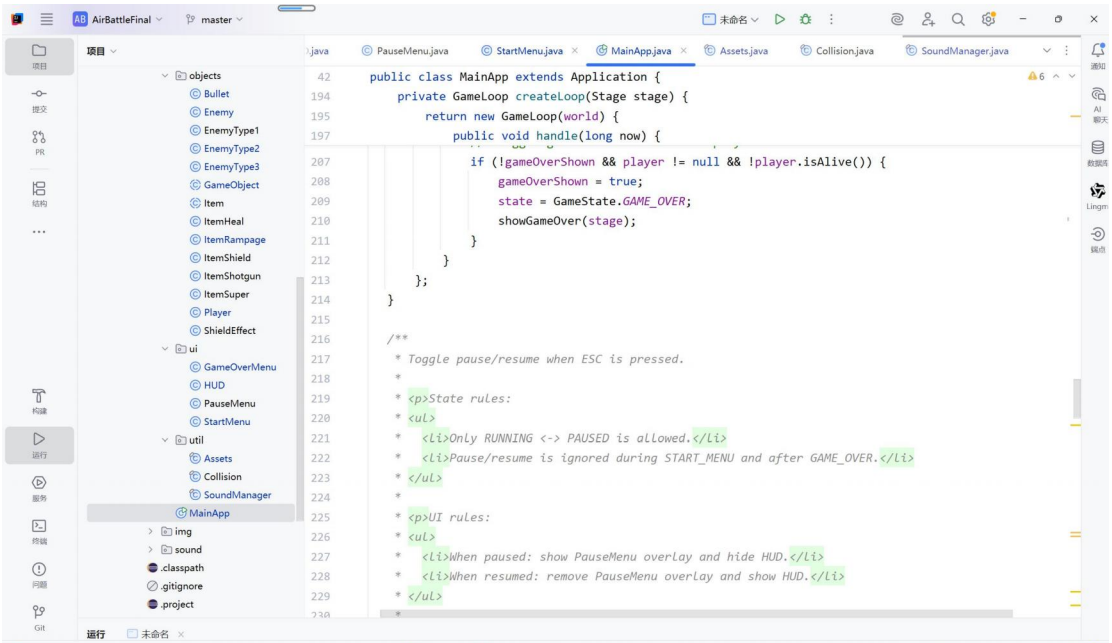
```
public class MainApp extends Application {
    public void start(Stage stage) {
        stage.setScene(scene);
        stage.setTitle("AirBattle");
        stage.setResizable(false);

        // Disable JavaFX default fullscreen exit behavior.
        // ESC is reserved for pause/resume, so we block the default ESC-to-exit-fullscreen shortcut.
        stage.setFullScreenExitHint("");
        stage.setFullScreenExitKeyCombination(KeyCombination.NO_MATCH);

        stage.show();

        input = new InputHandler(scene);

        pauseMenu = new PauseMenu(
                () -> {
                    root.getChildren().remove(pauseMenu);
                    if (hud != null) hud.setVisible(true);
                    loop.startRunning();
                    state = GameState.RUNNING;
                },
                stage::close
        );

        startMenu = new StartMenu(
                () -> {
                    root.getChildren().remove(startMenu);
```

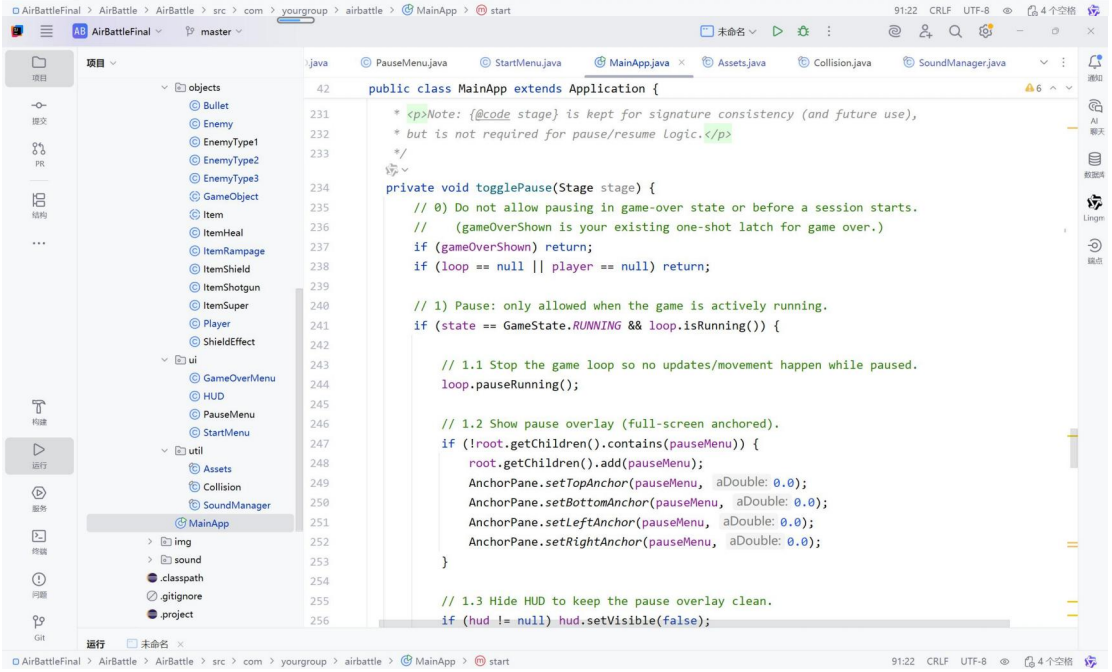


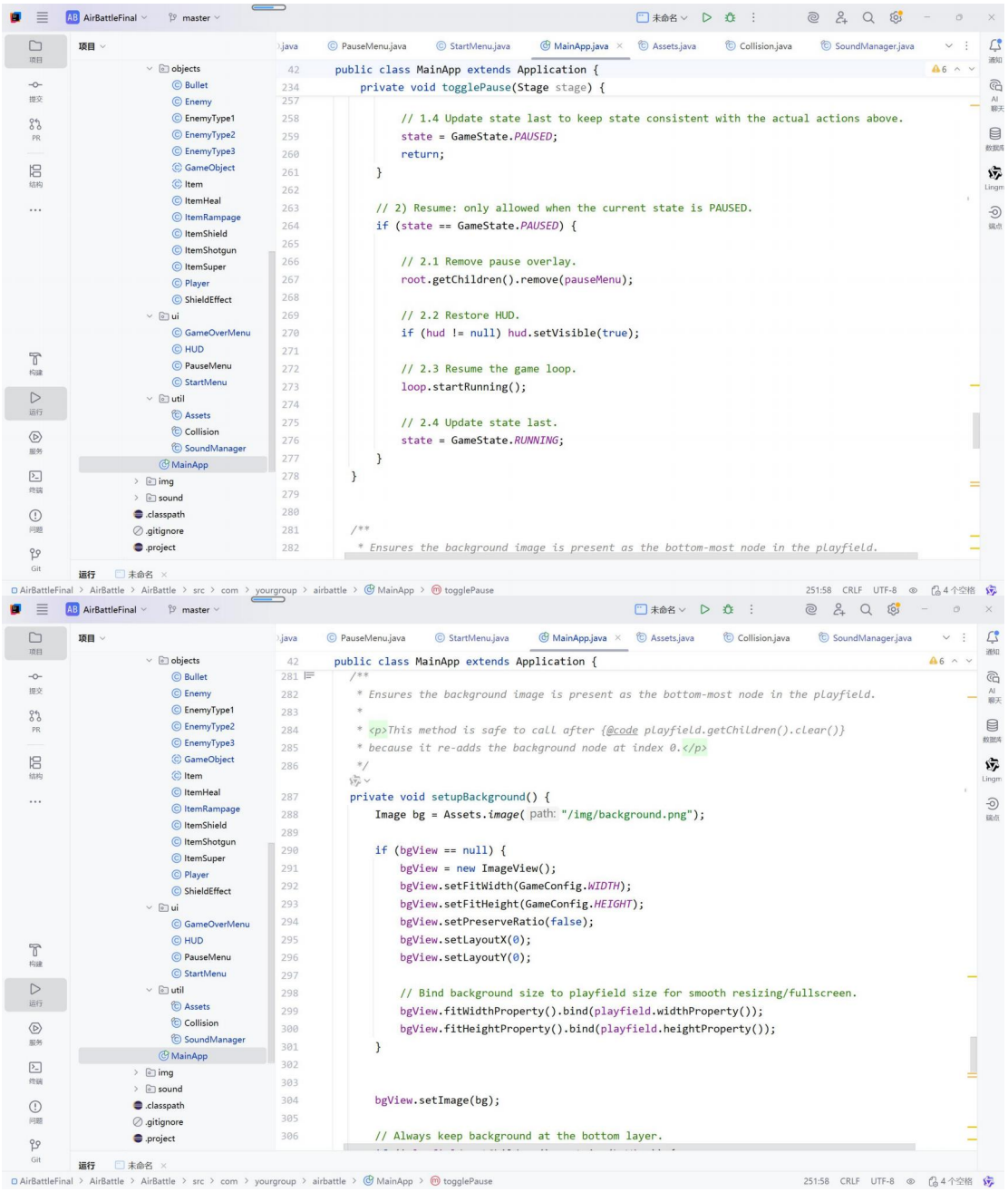
```
public class MainApp extends Application {
    public void start(Stage stage) {
                () -> {
                    root.getChildren().remove(startMenu);
                    initNewGame(stage);
                    state = GameState.RUNNING;
                },
                stage::close
        );

        // Show start menu on launch.
        root.getChildren().add(startMenu);
        AnchorPane.setTopAnchor(startMenu, aDouble: 0.0);
        AnchorPane.setBottomAnchor(startMenu, aDouble: 0.0);
        AnchorPane.setLeftAnchor(startMenu, aDouble: 0.0);
        AnchorPane.setRightAnchor(startMenu, aDouble: 0.0);

        // Play start/menu BGM.
        SoundManager.playStartBgm();

        // Global key bindings (window-level actions):
        // - ESC: toggle pause/resume during gameplay
        // - P  : toggle fullscreen/windowed mode
        scene.addEventHandler(KeyEvent.KEY_PRESSED, KeyEvent e -> {
            if (e.getCode() == KeyCode.ESCAPE) {
                // Only allow pause toggling during active gameplay states.
                if (state == GameState.RUNNING || state == GameState.PAUSED) {
                    togglePause(stage);
```

```
public class MainApp extends Application {
    public void start(Stage stage) {
        scene.addEventHandler(KeyEvent.KEY_PRESSED, KeyEvent e -> {
                }
            } else if (e.getCode() == KeyCode.P) {
                // Fullscreen toggle is a window-level concern, so it is handled here (not in InputHandl
                stage.setFullScreen(!stage.isFullScreen());
            }
        });
    }

    /**
     * Starts a brand-new game session:
     * <ul>
     *   <li>Clears the playfield and restores background</li>
     *   <li>Rebuilds world and loop</li>
     *   <li>Spawns the player</li>
     *   <li>Creates and attaches the HUD overlay</li>
     * </ul>
     */
    private void initNewGame(Stage stage) {
        if (input != null) input.clear();
        // Reset visuals.
        playfield.getChildren().clear();
        setupBackground();

        // Reset loop/world.
```


AirBattleFinal
master
未命名
项目
PauseMenu.java
StartMenu.java
MainApp.java
Assets.java
Collision.java
SoundManager.java
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
Assets
Collision
SoundManager
MainApp
img
sound
.classpath
.gitignore
.project
public class MainApp extends Application {
private void initNewGame(Stage stage) {
if (loop != null) {
loop.pauseRunning();
}
world = new GameWorld(playfield);
loop = createLoop(stage);
// Spawn player (Player uses a spawn callback to create bullets/effects).
player = new Player(
x: 430, y: 480,
Assets.image(path: "/img/player.png"),
input,
world.getBulletSprite(),
Assets.image(path: "/img/SuperBullet.png"),
Assets.image(path: "/img/shield.png"),
world::spawn
);
world.spawn(player);
// Reset HUD.
if (hud != null) root.getChildren().remove(hud);
hud = new HUD();
root.getChildren().add(hud);
state = GameState.RUNNING;
gameOverShown = false;
运行
未命名
AirBattleFinal › AirBattle › AirBattle › src › com › yourgroup › airbattle › MainApp › start
91:22 CRLF UTF-8 4 个空格
// Play gameplay BGM.
SoundManager.playFightBgm();
loop.startRunning();
}
/**
* Creates the main game loop wrapper that also synchronizes HUD and triggers game over.
*
* @param stage primary stage used for exit/restart callbacks
* @return a configured {@link GameLoop} instance
*/
private GameLoop createLoop(Stage stage) {
return new GameLoop(world) {
@Override
public void handle(long now) {
super.handle(now);
// Sync HUD with current session state.
if (player != null && hud != null) {
hud.setHp(player.getHp());
hud.setScore(world.getScore());
}
// Trigger game-over once when the player dies


AirBattleFinal
master
PauseMenu.java
StartMenu.java
MainApp.java
Assets.java
Collision.java
SoundManager.java
public class MainApp extends Application {
private GameLoop createLoop(Stage stage) {
return new GameLoop(world) {
public void handle(long now) {
if (!gameOverShown && player != null && !player.isAlive()) {
gameOverShown = true;
state = GameState.GAME_OVER;
showGameOver(stage);
}
}
};
}
/**
* Toggle pause/resume when ESC is pressed.
*
* State rules:
*
* Only RUNNING <-> PAUSED is allowed.
* Pause/resume is ignored during START_MENU and after GAME_OVER.
*
*
* UI rules:
*
* When paused: show PauseMenu overlay and hide HUD.
* When resumed: remove PauseMenu overlay and show HUD.
*
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
Assets
Collision
SoundManager
MainApp
img
sound
.classpath
.gitignore
.project
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > MainApp > start
91:22 CRLF UTF-8 4 个空格

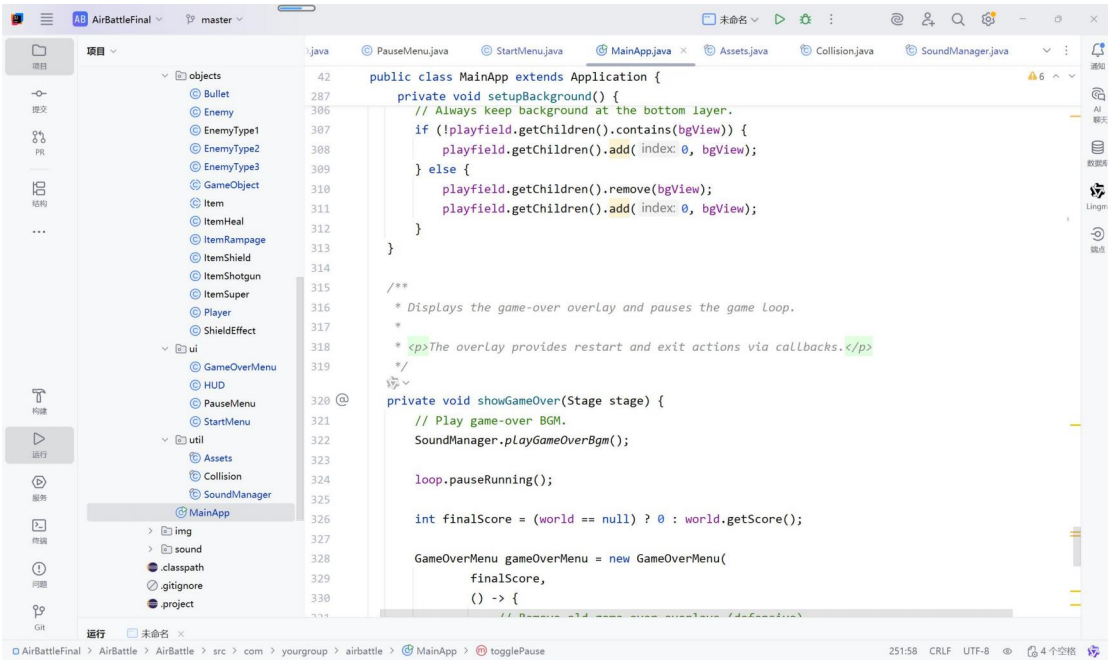

AirBattleFinal
master
PauseMenu.java
StartMenu.java
MainApp.java
Assets.java
Collision.java
SoundManager.java
public class MainApp extends Application {
* Note: {@code stage} is kept for signature consistency (and future use),
* but is not required for pause/resume logic.
*/
private void togglePause(Stage stage) {
// 0) Do not allow pausing in game-over state or before a session starts.
// (gameOverShown is your existing one-shot latch for game over.)
if (gameOverShown) return;
if (loop == null || player == null) return;
// 1) Pause: only allowed when the game is actively running.
if (state == GameState.RUNNING && loop.isRunning()) {
// 1.1 Stop the game loop so no updates/movement happen while paused.
loop.pauseRunning();
// 1.2 Show pause overlay (full-screen anchored).
if (!root.getChildren().contains(pauseMenu)) {
root.getChildren().add(pauseMenu);
AnchorPane.setTopAnchor(pauseMenu, aDouble: 0.0);
AnchorPane.setBottomAnchor(pauseMenu, aDouble: 0.0);
AnchorPane.setLeftAnchor(pauseMenu, aDouble: 0.0);
AnchorPane.setRightAnchor(pauseMenu, aDouble: 0.0);
}
// 1.3 Hide HUD to keep the pause overlay clean.
if (hud != null) hud.setVisible(false);
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
Assets
Collision
SoundManager
MainApp
img
sound
.classpath
.gitignore
.project
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > MainApp > start
91:22 CRLF UTF-8 4 个空格

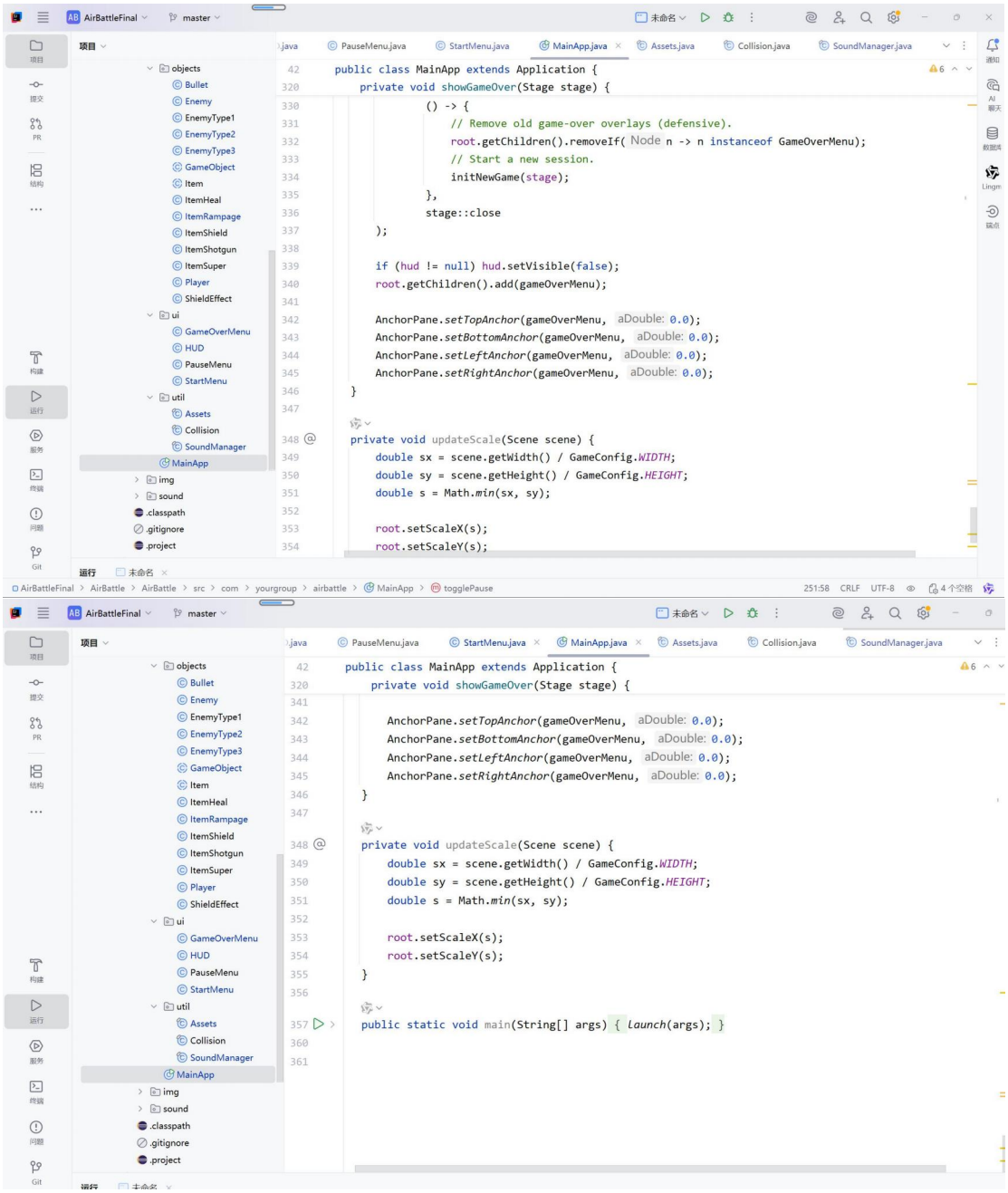


```
public class MainApp extends Application {
    private void togglePause(Stage stage) {

        // 1.4 Update state last to keep state consistent with the actual actions above.
        state = GameState.PAUSED;
        return;
    }

    // 2) Resume: only allowed when the current state is PAUSED.
    if (state == GameState.PAUSED) {

        // 2.1 Remove pause overlay.
        root.getChildren().remove(pauseMenu);

        // 2.2 Restore HUD.
        if (hud != null) hud.setVisible(true);

        // 2.3 Resume the game loop.
        loop.startRunning();

        // 2.4 Update state last.
        state = GameState.RUNNING;
    }
}


/**
 * Ensures the background image is present as the bottom-most node in the playfield.
```



```
public class MainApp extends Application {
    /**
     * Ensures the background image is present as the bottom-most node in the playfield.
     *
     * <p>This method is safe to call after {@code playfield.getChildren().clear()}
     * because it re-adds the background node at index 0.</p>
     */
    private void setupBackground() {
        Image bg = Assets.image( path: "/img/background.png");

        if (bgView == null) {
            bgView = new ImageView();
            bgView.setFitWidth(GameConfig.WIDTH);
            bgView.setFitHeight(GameConfig.HEIGHT);
            bgView.setPreserveRatio(false);
            bgView.setLayoutX(0);
            bgView.setLayoutY(0);

            // Bind background size to playfield size for smooth resizing/fullscreen.
            bgView.fitWidthProperty().bind(playfield.widthProperty());
            bgView.fitHeightProperty().bind(playfield.heightProperty());
        }


        bgView.setImage(bg);

        // Always keep background at the bottom layer.
```


AirBattleFinal
master
未命名
PauseMenu.java
StartMenu.java
MainApp.java
Assets.java
Collision.java
SoundManager.java
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
Assets
Collision
SoundManager
MainApp
img
sound
.classpath
.gitignore
.project
public class MainApp extends Application {
private void setupBackground() {
// Always keep background at the bottom layer.
if (!playfield.getChildren().contains(bgView)) {
playfield.getChildren().add(index: 0, bgView);
} else {
playfield.getChildren().remove(bgView);
playfield.getChildren().add(index: 0, bgView);
}
}
/**
* Displays the game-over overlay and pauses the game loop.
*
* The overlay provides restart and exit actions via callbacks.
*/
private void showGameOver(Stage stage) {
// Play game-over BGM.
SoundManager.playGameOverBgm();
loop.pauseRunning();
int finalScore = (world == null) ? 0 : world.getScore();
GameOverMenu gameOverMenu = new GameOverMenu(
finalScore,
() -> {
运行
未命名
AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > MainApp > togglePause
251:58
CRLF
UTF-8
4 个空格


AirBattleFinal
master
MainApp.java
PauseMenu.java
StartMenu.java
Assets.java
Collision.java
SoundManager.java
objects
Bullet
Enemy
EnemyType1
EnemyType2
EnemyType3
GameObject
Item
ItemHeal
ItemRampage
ItemShield
ItemShotgun
ItemSuper
Player
ShieldEffect
ui
GameOverMenu
HUD
PauseMenu
StartMenu
util
Assets
Collision
SoundManager
MainApp
img
sound
.classpath
.gitignore
.project
public class MainApp extends Application {
private void showGameOver(Stage stage) {
() -> {
// Remove old game-over overlays (defensive).
root.getChildren().removeIf(Node n -> n instanceof GameOverMenu);
// Start a new session.
initNewGame(stage);
},
stage::close
);
if (hud != null) hud.setVisible(false);
root.getChildren().add(gameOverMenu);
AnchorPane.setTopAnchor(gameOverMenu, aDouble: 0.0);
AnchorPane.setBottomAnchor(gameOverMenu, aDouble: 0.0);
AnchorPane.setLeftAnchor(gameOverMenu, aDouble: 0.0);
AnchorPane.setRightAnchor(gameOverMenu, aDouble: 0.0);
}
private void updateScale(Scene scene) {
double sx = scene.getWidth() / GameConfig.WIDTH;
double sy = scene.getHeight() / GameConfig.HEIGHT;
double s = Math.min(sx, sy);
root.setScaleX(s);
root.setScaleY(s);
AirBattle
src
com
yourgroup
airbattle
MainApp
togglePause
251:58
CRLF
UTF-8
4 个空格
public static void main(String[] args) { launch(args); }

APPENDIX 1

MARKING RUBRICS

Component Title	**Task(s)**				**Percentage (%)**	**100**			
Criteria	**Score and Descriptors**					**Weight (%)**	**Marks**		
	Excellent (5)	**Good (4)**	**Average (3)**	**Need Improvement (2)**	**Poor (1)**				
OOP Implementation	**OOP principles are expertly applied, with a well-structured and maintainable codebase. Inheritance, polymorphism, and encapsulation are used effectively.**	**OOP principles are correctly applied, with a mostly well-structured and maintainable codebase. Inheritance, polymorphism, and encapsulation are used appropriately.**	**OOP principles are applied to some extent, with a partially well-structured codebase. Inheritance, polymorphism, and encapsulation are used but with issues.**	**OOP principles are minimally applied, with a disorganized and hard-to-maintain codebase. Inheritance, polymorphism, and encapsulation are lacking.**	**OOP principles are not applied, leading to an unstructured and unmaintainable codebase. No use of inheritance, polymorphism, or encapsulation.**	**30**			
Functionality & Completeness	**The game functions perfectly as intended, with all core mechanics implemented and no issues.**	**The game functions well, with minor issues or improvements needed in some core mechanics.**	**The game functions satisfactorily but has notable issues or missing core mechanics.**	**The game has significant functionality issues, with major missing core mechanics.**	**The game is non-functional or barely functional, with no core mechanics implemented.**	**20**			
Code Organization & Readability	**The code is exceptionally well-organized, easy to follow, and consistently styled.**	**The code is well-organized, generally easy to follow, and mostly consistently styled.**	**The code is somewhat organized, moderately easy to follow, and inconsistently styled.**	**The code is disorganized, hard to follow, and inconsistently styled.**	**The code is extremely disorganized, virtually impossible to follow, and un-styled.**	**10**			
Comments and Documentation	**Thorough comments and documentation, explaining the purpose and usage of each class and method in detail. Clear instructions for playing the game.**	**Good comments and documentation, providing adequate explanations and instructions.**	**Basic comments and documentation, lacking some details and instructions.**	**Minimal comments and documentation, missing essential explanations and instructions.**	**No or negligible comments and documentation, leaving the code and game unexplained.**	**20**			
Concept	**The game concept is innovative and engaging, demonstrating creativity and originality.**	**The game concept is interesting and creative, showcasing an engaging idea.**	**The game concept is somewhat engaging, but lacks significant innovation or creativity.**	**The game concept is average and not particularly engaging or creative.**	**The game concept lacks innovation, creativity, and engagement.**	**10**			
Individual Contribution	**Exceptional individual contributions, demonstrating high-quality work and a strong sense of responsibility.**	**Good individual contributions, demonstrating solid effort and contributions from all team members.**	**Adequate individual contributions, demonstrating satisfactory effort and contributions from most team members.**	**Limited individual contributions, indicating minimal effort or inconsistent contributions.**	**Very poor individual contributions, indicating no or negligible effort and minimal or no contributions.**	**10**	**Student ID:**	**Student ID:**	**Student ID:**
					TOTAL	**100**			

Note to students: Please include the marking rubric when submitting your coursework.