



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:

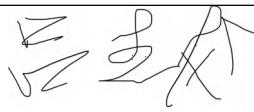
Own Work Declaration

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

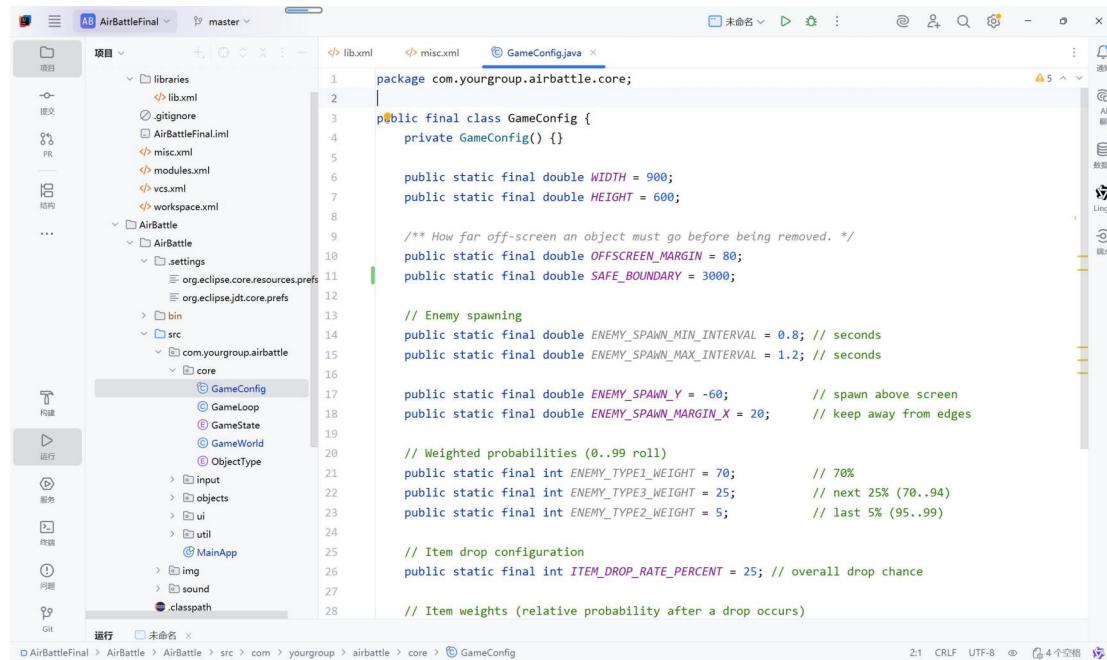
XIAMEN UNIVERSITY MALAYSIA

Content Page

AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\core	2
\GameConfig.java	2
\GameLoop.java	3
\GameState.java	4
\GameWorld.java	5
\ObjectType.java	14
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\input	15
\InputHandler.java	15
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\objects	18
\Bullet.java	18
\Enemy.java	20
\EnemyType1.java	23
\EnemyType2.java	25
\EnemyType3.java	27
\GameObject.java	30
\Item.java	35
\ItemHeal.java	37
\ItemRampage.java	38
\ItemShield.java	39
\ItemShotgun.java	40
\ItemSuper.java	41
\Player.java	42
\ShieldEffect.java	49
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\ui	51
\GameOverMenu.java	51
\HUD.java	54
\PauseMenu.java	58
\StartMenu.java	61
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\util	64
\Assets.java	64
\Collision.java	66
\SoundManager.java	67
AirBattleFinal\AirBattle\AirBattle\src\com\yourgroup\airbattle\MainApp.java	70

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

\GameConfig.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

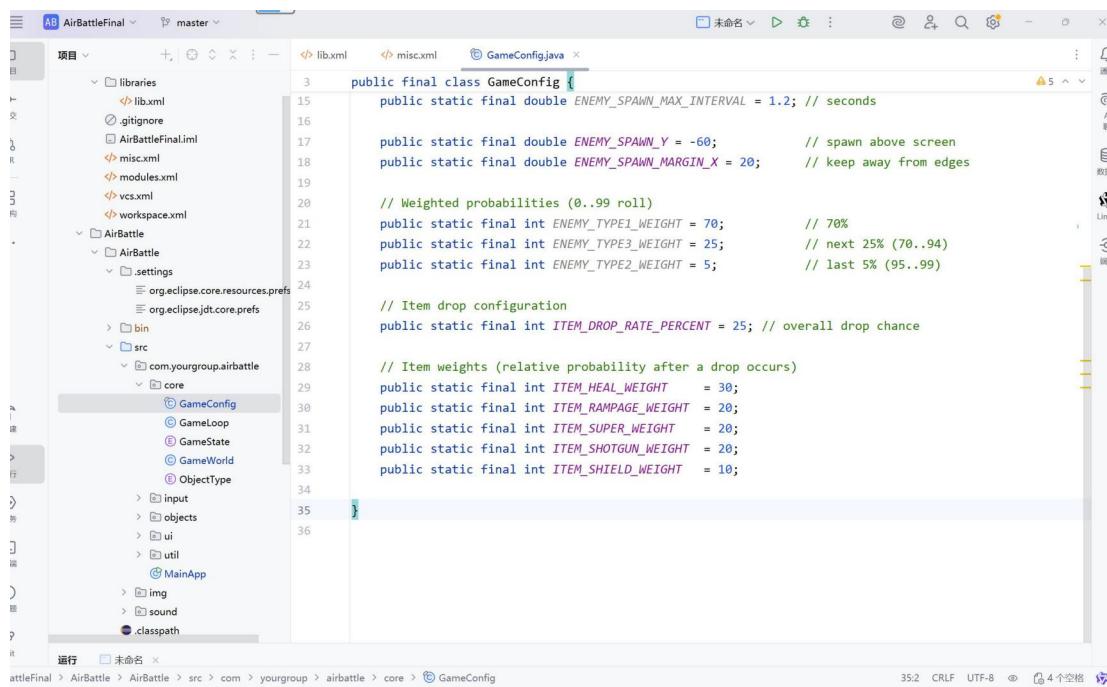
    public static final double ENEMY_SPAWN_Y = -60;           // spawn above screen
    public static final double ENEMY_SPAWN_MARGIN_X = 20;      // keep away from edges

    // 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)
}

```



```

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

```

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 class GameLoop extends AnimationTimer {
    public GameLoop(GameWorld world) { this.world = world; }

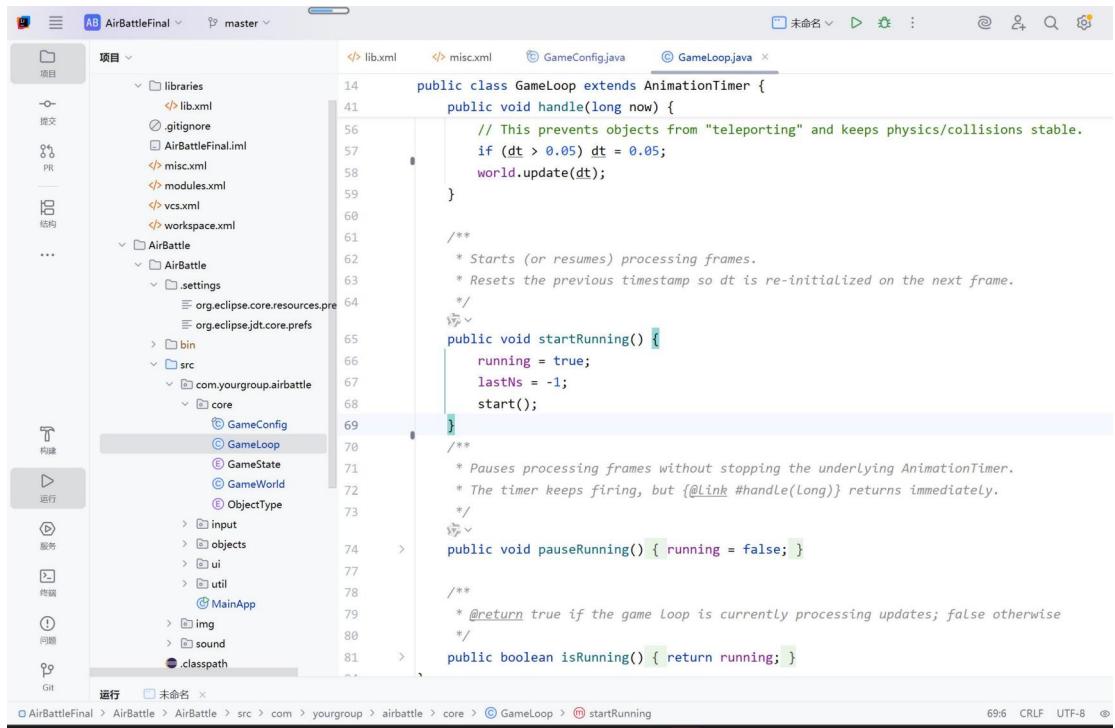
    /**
     * Called by JavaFX every frame while the AnimationTimer is started.
     * Computes delta time and updates the game world when the loop is running.
     *
     * @param now current timestamp in nanoseconds provided by JavaFX
     */
    @Override
    public void handle(long now) {
        // 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.
    }
}

```



The screenshot shows the Eclipse IDE interface with the GameLoop.java file open in the editor. The code implements an AnimationTimer for game logic. It includes methods for handling time steps, starting, pausing, and checking if the loop is running.

```

public class GameLoop extends AnimationTimer {
    public void handle(long now) {
        // This prevents objects from "teleporting" and keeps physics/collisions stable.
        if (dt > 0.05) dt = 0.05;
        world.update(dt);
    }

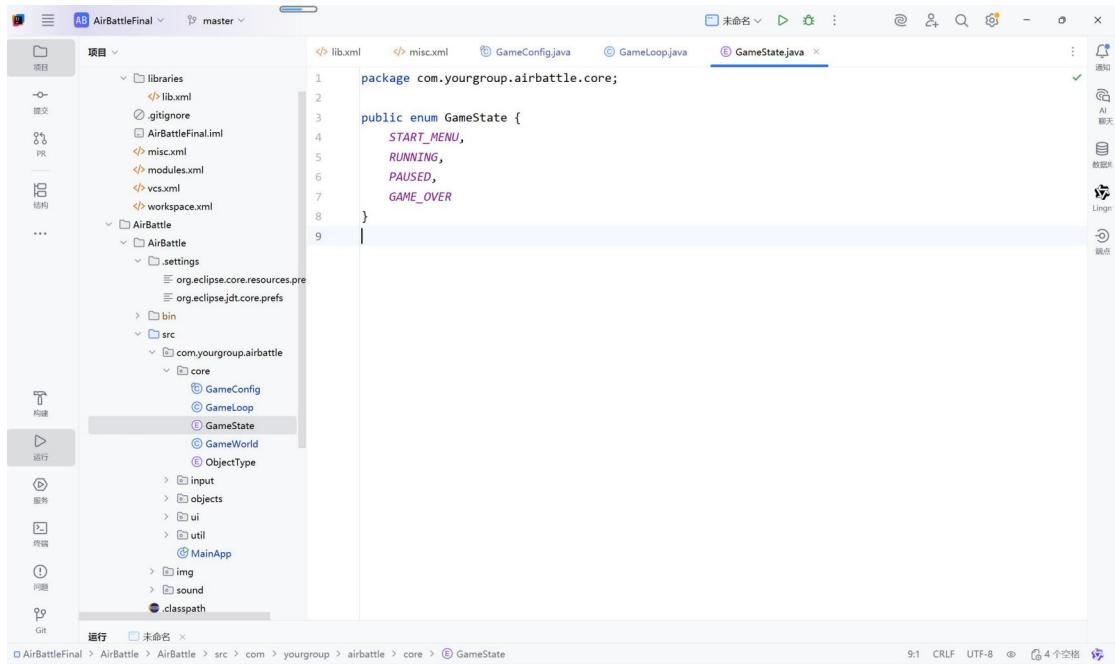
    /**
     * Starts (or resumes) processing frames.
     * Resets the previous timestamp so dt is re-initialized on the next frame.
     */
    public void startRunning() {
        running = true;
        lastNs = -1;
        start();
    }

    /**
     * Pauses processing frames without stopping the underlying AnimationTimer.
     * The timer keeps firing, but {@link #handle(Long)} returns immediately.
     */
    public void pauseRunning() { running = false; }

    /**
     * @return true if the game loop is currently processing updates; false otherwise
     */
    public boolean isRunning() { return running; }
}

```

\GameState.java



The screenshot shows the Eclipse IDE interface with the GameState.java file open in the editor. The code defines a public enum for game states, including START_MENU, RUNNING, PAUSED, and GAME_OVER.

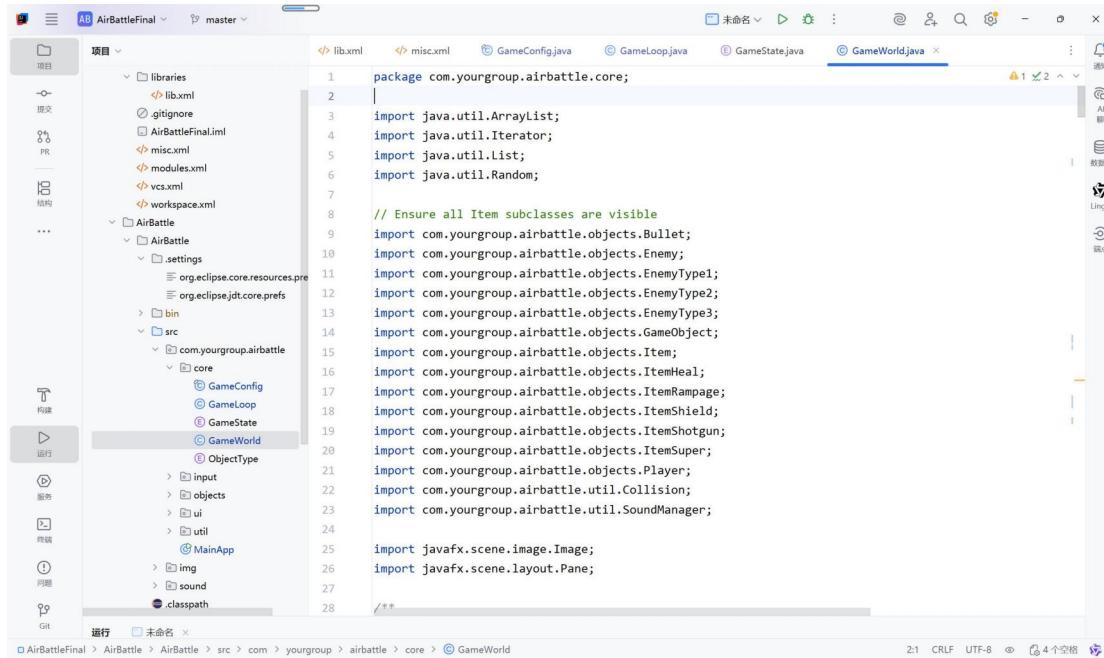
```

package com.yourgroup.airbattle.core;

public enum GameState {
    START_MENU,
    RUNNING,
    PAUSED,
    GAME_OVER
}

```

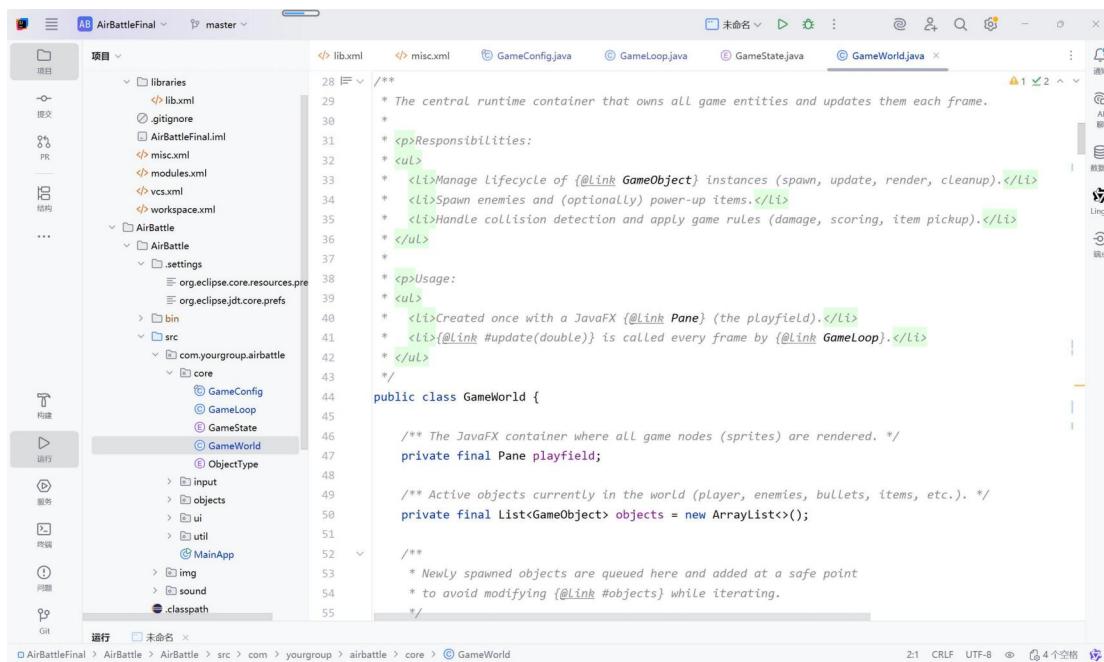
\GameWorld.java



```

1 package com.yourgroup.airbattle.core;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.List;
6 import java.util.Random;
7
8 // Ensure all Item subclasses are visible
9 import com.yourgroup.airbattle.objects.Bullet;
10 import com.yourgroup.airbattle.objects.Enemy;
11 import com.yourgroup.airbattle.objects.EnemyType1;
12 import com.yourgroup.airbattle.objects.EnemyType2;
13 import com.yourgroup.airbattle.objects.EnemyType3;
14 import com.yourgroup.airbattle.objects.GameObject;
15 import com.yourgroup.airbattle.objects.Item;
16 import com.yourgroup.airbattle.objects.ItemHeal;
17 import com.yourgroup.airbattle.objects.ItemRampage;
18 import com.yourgroup.airbattle.objects.ItemShield;
19 import com.yourgroup.airbattle.objects.ItemShotgun;
20 import com.yourgroup.airbattle.objects.ItemSuper;
21 import com.yourgroup.airbattle.objects.Player;
22 import com.yourgroup.airbattle.util.Collision;
23 import com.yourgroup.airbattle.util.SoundManager;
24
25 import javafx.scene.image.Image;
26 import javafx.scene.layout.Pane;
27
28 /**
29 * The central runtime container that owns all game entities and updates them each frame.
30 *
31 * <p>Responsibilities:</p>
32 * <ul>
33 *   <li>Manage lifecycle of {@link GameObject} instances (spawn, update, render, cleanup).</li>
34 *   <li>Spawn enemies and (optionally) power-up items.</li>
35 *   <li>Handle collision detection and apply game rules (damage, scoring, item pickup).</li>
36 * </ul>
37 *
38 * <p>Usage:</p>
39 * <ul>
40 *   <li>Created once with a JavaFX {@link Pane} (the playfield).</li>
41 *   <li>{@Link #update(double)} is called every frame by {@link GameLoop}.</li>
42 * </ul>
43 */
44 public class GameWorld {
45
46     /** The JavaFX container where all game nodes (sprites) are rendered. */
47     private final Pane playfield;
48
49     /** Active objects currently in the world (player, enemies, bullets, items, etc.). */
50     private final List<GameObject> objects = new ArrayList<>();
51
52     /**
53      * Newly spawned objects are queued here and added at a safe point
54      * to avoid modifying {@Link #objects} while iterating.
55     */
56 }

```

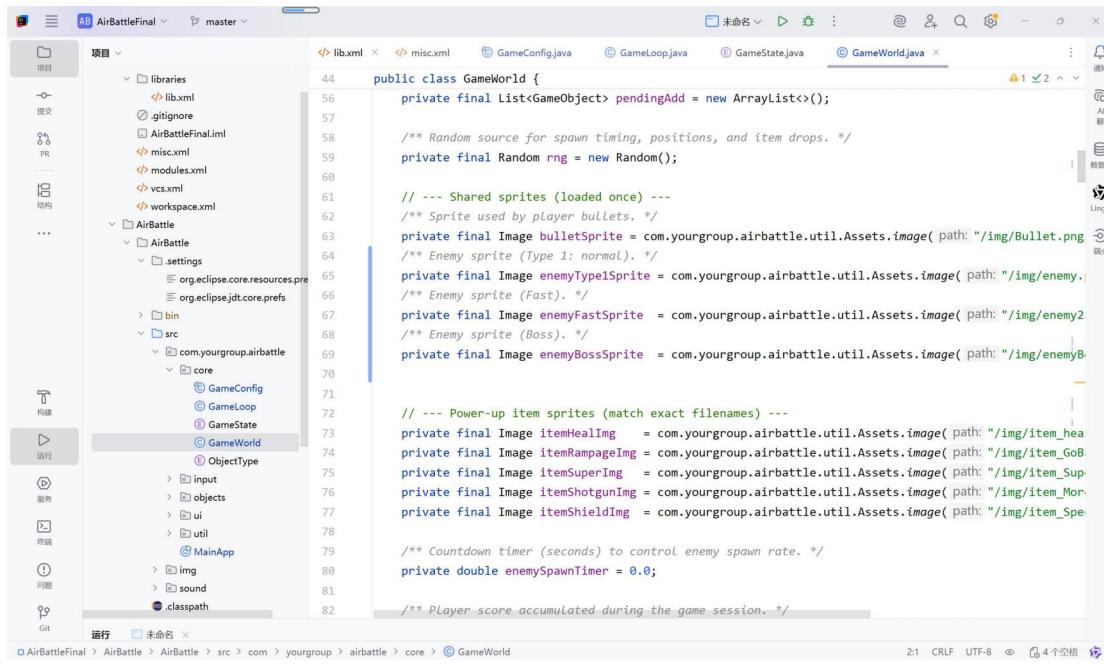


```

1 package com.yourgroup.airbattle.core;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.List;
6 import java.util.Random;
7
8 // Ensure all Item subclasses are visible
9 import com.yourgroup.airbattle.objects.Bullet;
10 import com.yourgroup.airbattle.objects.Enemy;
11 import com.yourgroup.airbattle.objects.EnemyType1;
12 import com.yourgroup.airbattle.objects.EnemyType2;
13 import com.yourgroup.airbattle.objects.EnemyType3;
14 import com.yourgroup.airbattle.objects.GameObject;
15 import com.yourgroup.airbattle.objects.Item;
16 import com.yourgroup.airbattle.objects.ItemHeal;
17 import com.yourgroup.airbattle.objects.ItemRampage;
18 import com.yourgroup.airbattle.objects.ItemShield;
19 import com.yourgroup.airbattle.objects.ItemShotgun;
20 import com.yourgroup.airbattle.objects.ItemSuper;
21 import com.yourgroup.airbattle.objects.Player;
22 import com.yourgroup.airbattle.util.Collision;
23 import com.yourgroup.airbattle.util.SoundManager;
24
25 import javafx.scene.image.Image;
26 import javafx.scene.layout.Pane;
27
28 /**
29 * The central runtime container that owns all game entities and updates them each frame.
30 *
31 * <p>Responsibilities:</p>
32 * <ul>
33 *   <li>Manage lifecycle of {@link GameObject} instances (spawn, update, render, cleanup).</li>
34 *   <li>Spawn enemies and (optionally) power-up items.</li>
35 *   <li>Handle collision detection and apply game rules (damage, scoring, item pickup).</li>
36 * </ul>
37 *
38 * <p>Usage:</p>
39 * <ul>
40 *   <li>Created once with a JavaFX {@link Pane} (the playfield).</li>
41 *   <li>{@Link #update(double)} is called every frame by {@link GameLoop}.</li>
42 * </ul>
43 */
44 public class GameWorld {
45
46     /** The JavaFX container where all game nodes (sprites) are rendered. */
47     private final Pane playfield;
48
49     /** Active objects currently in the world (player, enemies, bullets, items, etc.). */
50     private final List<GameObject> objects = new ArrayList<>();
51
52     /**
53      * Newly spawned objects are queued here and added at a safe point
54      * to avoid modifying {@Link #objects} while iterating.
55     */
56 }

```

XIAMEN UNIVERSITY MALAYSIA



```

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();

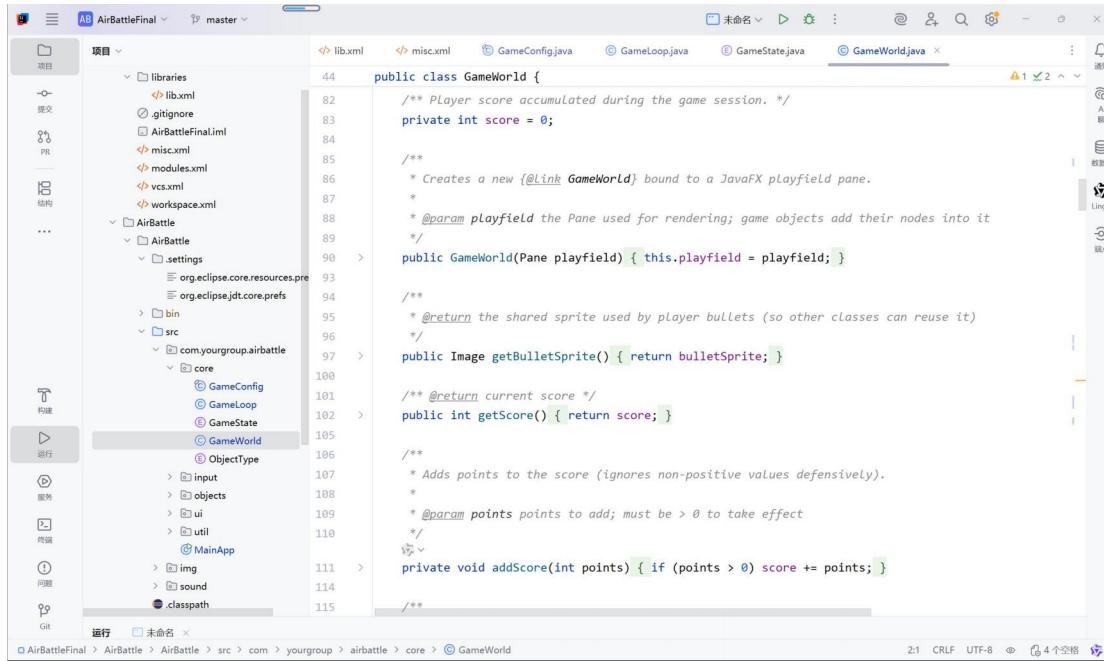
    // --- Shared sprites (loaded once) ---
    /** Sprite used by player bullets. */
    private final Image bulletSprite = com.yourgroup.airbattle.util.Assets.image( path: "/img/Bullet.png" );
    /** Enemy sprite (Type 1: normal). */
    private final Image enemyType1Sprite = com.yourgroup.airbattle.util.Assets.image( path: "/img/enemy1" );
    /** 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/enemy3" );

    // --- Power-up item sprites (match exact filenames) ---
    private final Image itemHealingImg = com.yourgroup.airbattle.util.Assets.image( path: "/img/item_heal" );
    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_Super" );
    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. */
}

```



```

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; }
}

```

XIAMEN UNIVERSITY MALAYSIA

The screenshot displays two instances of the Eclipse IDE interface, each showing the same Java file, `GameWorld.java`, from a project named `AirBattleFinal`.

Left Editor Content:

```
public class GameWorld {  
    /**  
     * 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  
     */
```

Right Editor Content:

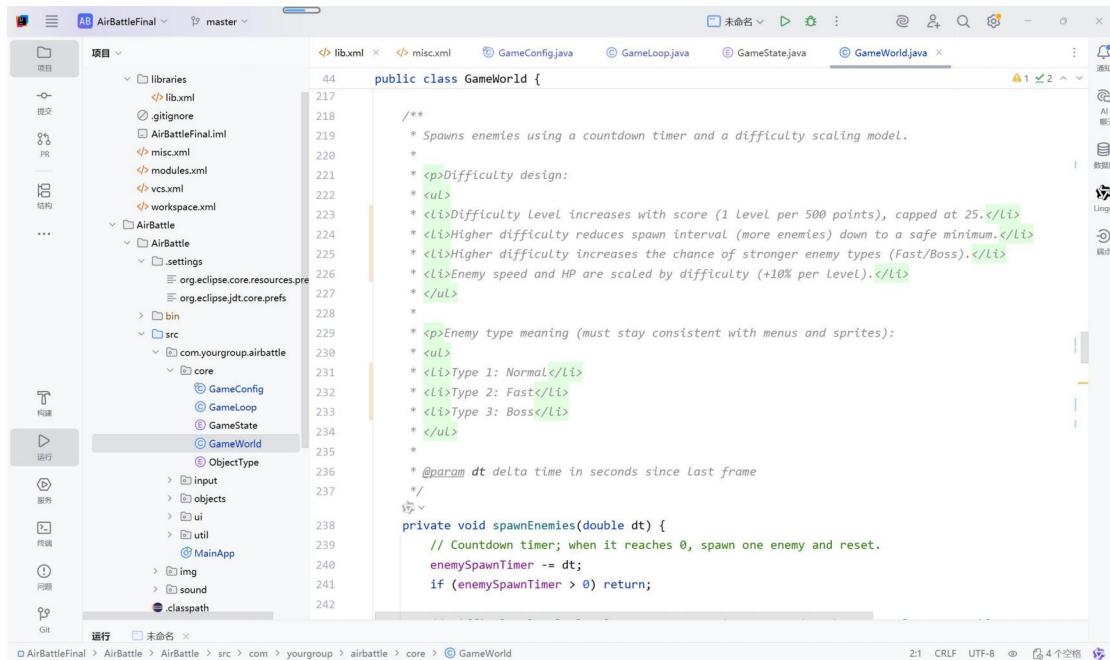
```
public class GameWorld {  
    * @param dt delta time in seconds since last frame  
    */  
    public void update(double dt) {  
        spawnEnemies(dt);  
  
        // Safely add newly spawned objects and their JavaFX nodes  
        if (!pendingAdd.isEmpty()) {  
            for (GameObject 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();  
    }
```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side screenshots of the Eclipse IDE interface, both displaying the same Java file: GameWorld.java. The file is located in the 'core' package under 'com.yourgroup.airbattle'. The code implements a GameWorld class with methods for updating game objects and spawning items based on weights defined in GameConfig.

```
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.  
     *  
     * <p>Drop chance and item weights are configured in (@Link GameConfig) so gameplay balance  
     * can be tuned without changing core game logic.</p>  
    */  
  
    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  
  
        private void trySpawnItem(double x, double y) {  
            + 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));  
            }  
        }  
    }  
}
```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows an IDE interface with the following details:

- Project Explorer:** Shows the project structure under "AirBattleFinal". The "src" folder contains packages like "com.yourgroup.airbattle.core" which include classes "GameConfig", "GameLoop", "GameState", "GameWorld", and "ObjectType".
- Code Editor:** The main editor window displays the "GameWorld.java" file. The code is annotated with JavaDoc comments explaining the difficulty scaling model and enemy types.
- Toolbars and Menus:** Standard IDE toolbars and menus are visible along the top and right side.
- Status Bar:** The bottom status bar shows the file path "AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > core > GameWorld.java", the line count "242", and encoding information "2:1 CRLF UTF-8".

```
public class GameWorld {  
    /**  
     * Spawns enemies using a countdown timer and a difficulty scaling model.  
     * <p>Difficulty design:  
     * <ul>  
     * <li>Difficulty Level increases with score (1 Level per 500 points), capped at 25.</li>  
     * <li>Higher difficulty reduces spawn interval (more enemies) down to a safe minimum.</li>  
     * <li>Higher difficulty increases the chance of stronger enemy types (Fast/Boss).</li>  
     * </ul>  
     * <p>Enemy type meaning (must stay consistent with menus and sprites):  
     * <ul>  
     * <li>Type 1: Normal</li>  
     * <li>Type 2: Fast</li>  
     * <li>Type 3: Boss</li>  
     * </ul>  
     * @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;  
    }  
}
```

XIAMEN UNIVERSITY MALAYSIA

Two screenshots of an IDE showing Java code for a game world class.

Screenshot 1:

```

public class GameWorld {
    private void spawnEnemies(double dt) {
        // 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)
    }
}

```

Screenshot 2:

```

public class GameWorld {
    private void spawnEnemies(double dt) {
        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();
    }
}

```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side screenshots of the Eclipse IDE interface, both displaying the same Java file: GameWorld.java. The left window shows the code with several annotations (like `private`, `public`, and `/* */`) and some comments. The right window shows the same code with additional annotations and comments, including a detailed description of the `handleCollisionsOptimized` method.

```
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.  
     */  
    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;  
                case ENEMY:  
                    enemies.add(o);  
                    break;  
                case POWERUP:  
                    powerups.add(o);  
                    break;  
            }  
        }  
  
        // Sort objects by type (bullet, enemy, powerup)  
        Collections.sort(bullets, new Comparator<GameObject>() {  
            public int compare(GameObject o1, GameObject o2) {  
                if (o1.getType() == BULLET_PLAYER && o2.getType() == ENEMY) return -1;  
                if (o1.getType() == ENEMY && o2.getType() == BULLET_PLAYER) return 1;  
                if (o1.getType() == BULLET_PLAYER && o2.getType() == POWERUP) return -1;  
                if (o1.getType() == POWERUP && o2.getType() == BULLET_PLAYER) return 1;  
                if (o1.getType() == ENEMY && o2.getType() == POWERUP) return -1;  
                if (o1.getType() == POWERUP && o2.getType() == ENEMY) return 1;  
                return 0;  
            }  
        });  
  
        // Resolve collisions between bullet and enemy  
        for (GameObject bullet : bullets) {  
            for (GameObject enemy : enemies) {  
                if (bullet.intersects(enemy)) {  
                    enemy.takeDamage();  
                    if (enemy.isDead()) {  
                        enemy.setAlive(false);  
                        enemy.setHealth(0);  
                        enemy.setScore(-1);  
                    }  
                    bullet.setAlive(false);  
                    bullet.setHealth(0);  
                }  
            }  
        }  
  
        // Resolve collisions between player and enemy  
        for (GameObject player : players) {  
            for (GameObject enemy : enemies) {  
                if (player.intersects(enemy)) {  
                    player.takeDamage();  
                    if (player.isDead()) {  
                        player.setAlive(false);  
                        player.setHealth(0);  
                        player.setScore(-1);  
                    }  
                    enemy.setAlive(false);  
                    enemy.setHealth(0);  
                }  
            }  
        }  
  
        // Resolve collisions between player and powerup  
        for (GameObject player : players) {  
            for (GameObject powerup : powerups) {  
                if (player.intersects(powerup)) {  
                    player.applyEffect(powerup);  
                    powerup.setAlive(false);  
                    powerup.setHealth(0);  
                }  
            }  
        }  
    }  
}
```

XIAMEN UNIVERSITY MALAYSIA

GameWorld.java (Top)

```

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;
            }
        }
    }
}

```

GameWorld.java (Bottom)

```

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;
        }
    }

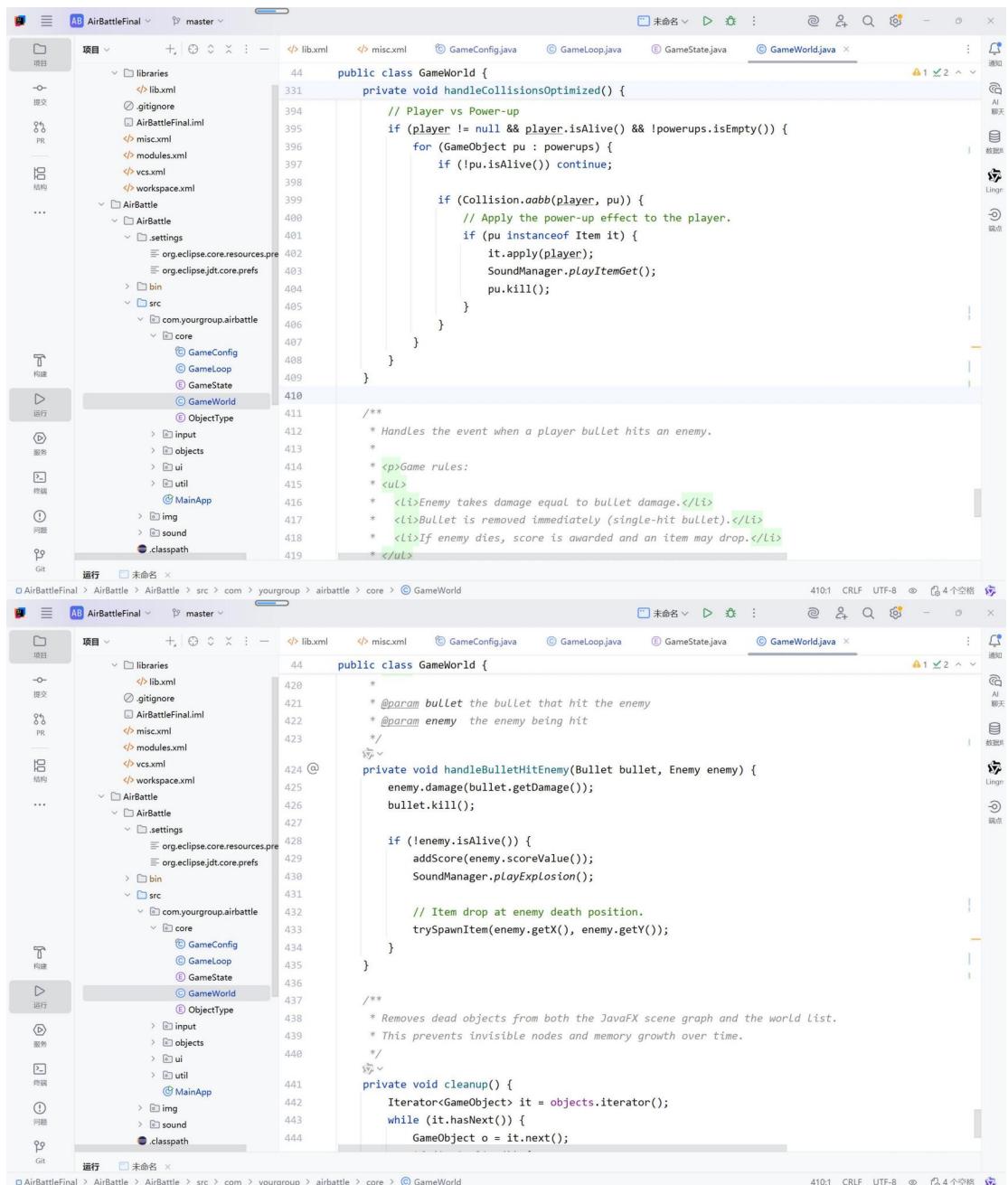
    / Player vs Enemy
    if (player != null & player.isAlive()) {
        for (GameObject enemyObj : enemies) {
            if (!enemyObj.isAlive()) continue;

            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
}

```

XIAMEN UNIVERSITY MALAYSIA



The image shows two side-by-side screenshots of the Eclipse IDE interface, both displaying the same Java file: GameWorld.java. The file is part of a project named 'AirBattleFinal'.

Project Structure:

- libraries
- lib.xml
- .gitignore
- AirBattleFinal.lml
- misc.xml
- modules.xml
- vcs.xml
- workspace.xml
- AirBattle
- src
- com.yourgroup.airbattle
- core
- GameConfig
- GameLoop
- GameState
- GameWorld
- ObjectType
- input
- objects
- ui
- util
- MainApp
- img
- sound
- classpath

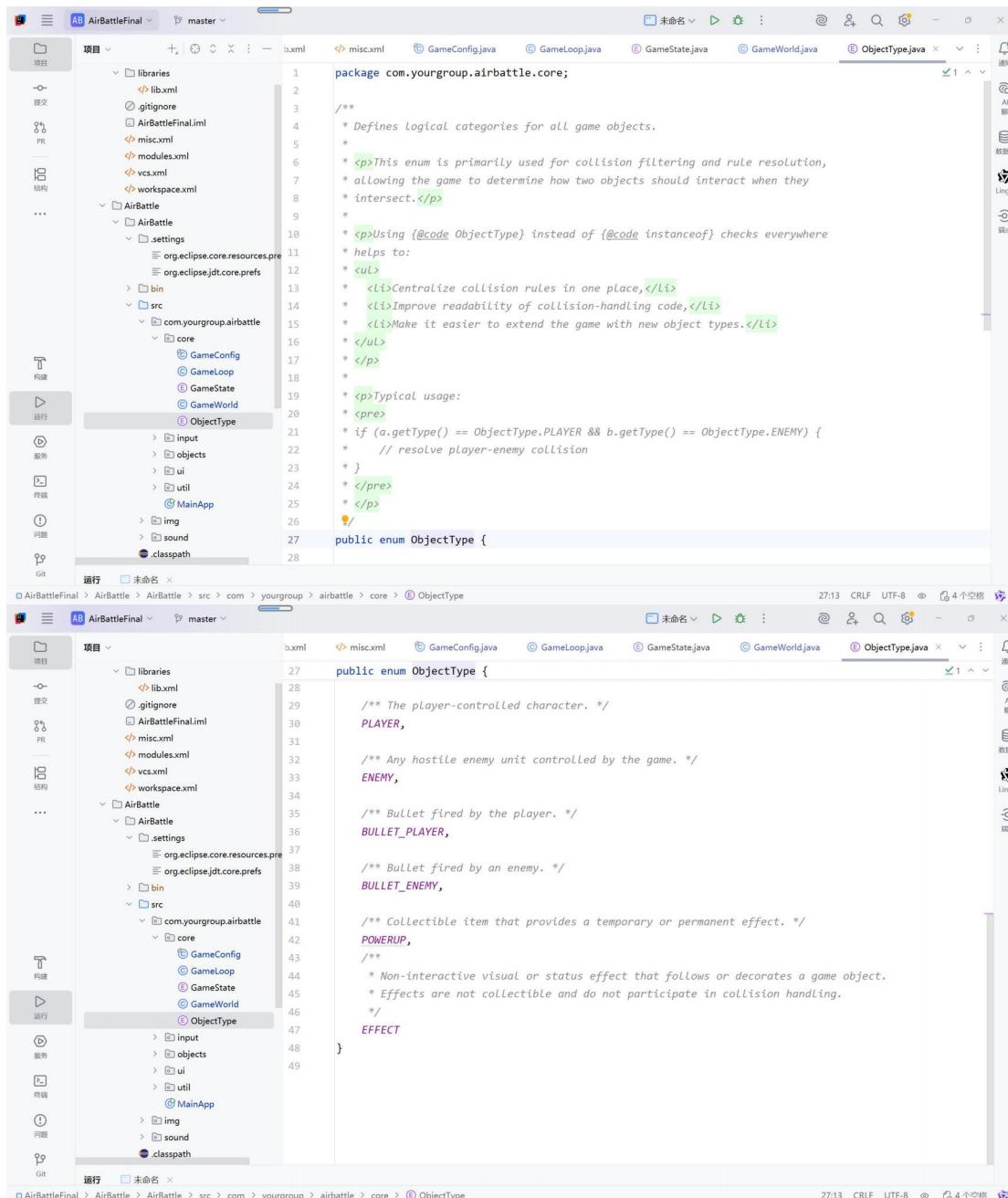
Code Snippet (Top Window):

```
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>  
         */  
    }  
}
```

Code Snippet (Bottom Window):

```
public class GameWorld {  
    *  
    * @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();  
                if (!o.isAlive())  
                    it.remove();  
            }  
        }  
}
```

\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:</p>
 * <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>Typical usage:</p>
 * <pre>
 * if (a.getType() == ObjectType.PLAYER && b.getType() == ObjectType.ENEMY) {
 *     // resolve player-enemy collision
 * }
 * </pre>
 * </p>
 */

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,

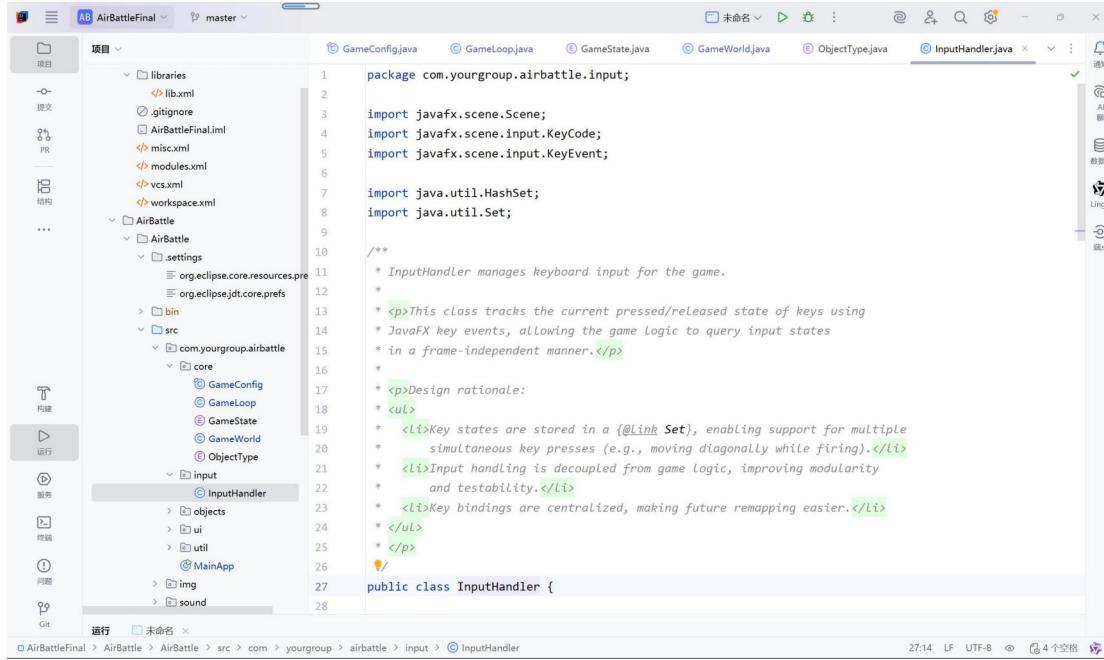
    /**
     * Collectible item that provides a temporary or permanent effect.
     */
    POWERUP,

    /**
     * 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



The screenshot shows a Java development environment with the following details:

- Project Structure:** The left sidebar shows a tree view of the project structure. It includes a 'libraries' folder containing 'lib.xml' and '.gitignore', an 'AirBattleFinal.iml' file, and 'misc.xml', 'modules.xml', 'vcs.xml', and 'workspace.xml' files. The 'src' folder contains 'AirBattle' and 'com.yourgroup.airbattle'. Inside 'com.yourgroup.airbattle' are 'core', 'input', and 'objects' packages. 'input' contains the 'InputHandler' class, which is selected in the code editor.
- Code Editor:** The right pane displays the 'InputHandler.java' code. The code is annotated with Javadoc comments explaining its purpose and design rationale.
- Toolbars and Status Bar:** The top bar has tabs for 'GameConfig.java', 'GameLoop.java', 'GameState.java', 'GameWorld.java', 'ObjectType.java', and 'InputHandler.java'. The status bar at the bottom shows the file path 'AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > input > InputHandler', the time '27:14', and encoding 'UTF-8'.

```

package com.yourgroup.airbattle.input;

import javafx.scene.Scene;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;

import java.util.HashSet;
import java.util.Set;

/**
 * InputHandler manages keyboard input for the game.
 *
 * <p>This class tracks the current pressed/released state of keys using JavaFX key events, allowing the game logic to query input states in a frame-independent manner.</p>
 *
 * <p><b>Design rationale:</b></p>
 * <ul>
 *   <li>Key states are stored in a {@link Set}, enabling support for multiple simultaneous key presses (e.g., moving diagonally while firing).</li>
 *   <li>Input handling is decoupled from game logic, improving modularity and testability.</li>
 *   <li>Key bindings are centralized, making future remapping easier.</li>
 * </ul>
 * </p>
 */
public class InputHandler {

```

XIAMEN UNIVERSITY MALAYSIA

```

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
     */
    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); }

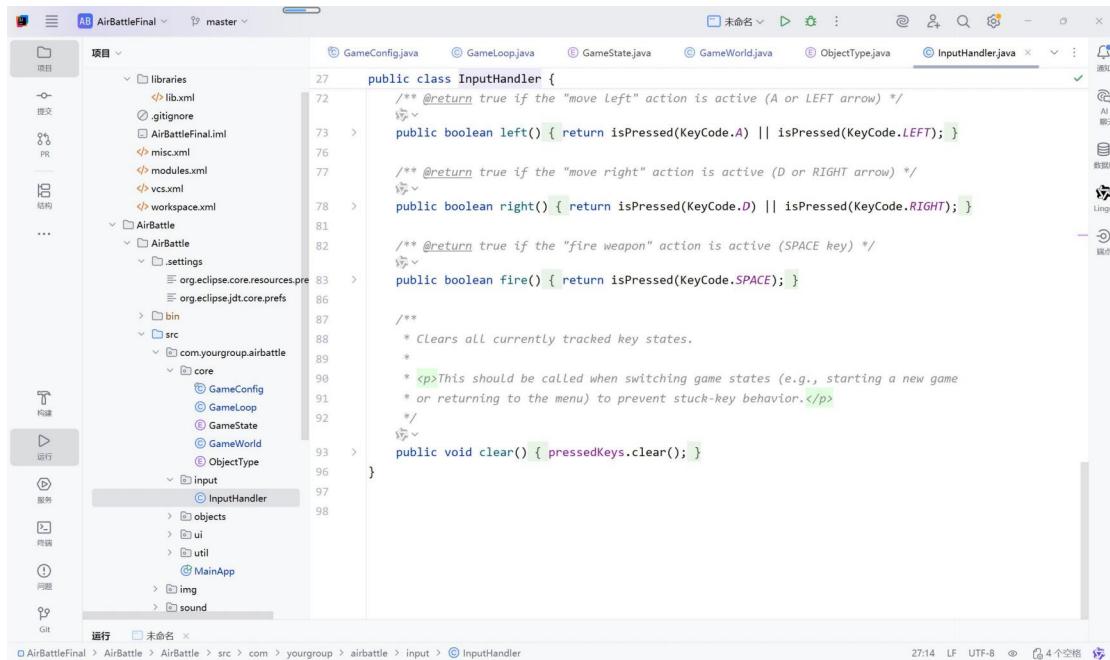
    /**
     * @return true if the "move left" action is active (A or LEFT arrow)
     */
    public boolean left() { return isPressed(KeyCode.A) || isPressed(KeyCode.LEFT); }

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

    /**
     * Clears all currently tracked key states.
     *
     * <p>This should be called when switching game states (e.g., starting a new game)</p>
     */
    public void clear() { pressedKeys.clear(); }
}

```

XIAMEN UNIVERSITY MALAYSIA

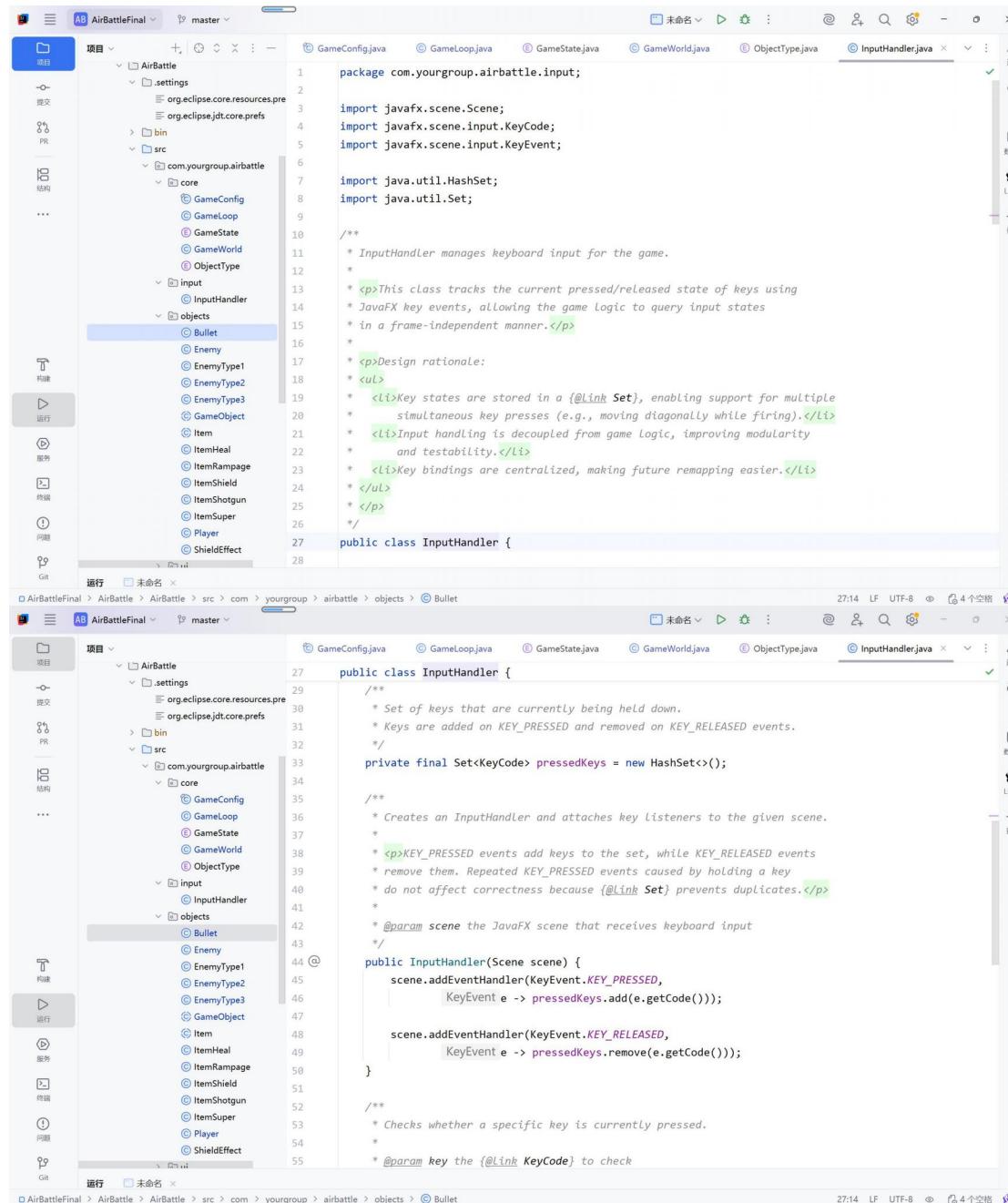


The screenshot shows a Java code editor within an IDE. The project structure on the left includes 'libraries' (lib.xml, .gitignore, AirBattleFinal.iml, misc.xml, modules.xml, vcs.xml, workspace.xml), 'AirBattle' (bin, src containing com.yourgroup.airbattle, core, objects, ui, util, MainApp), and 'img' and 'sound' folders. The current file is 'InputHandler.java' under 'src/com.yourgroup.airbattle/input'. The code implements an InputHandler interface with methods for handling key presses:

```
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.  
     *  
     * <p>This should be called when switching game states (e.g., starting a new game  
     * or returning to the menu) to prevent stuck-key behavior.</p>  
     */  
    public void clear() { pressedKeys.clear(); }  
}
```

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

\Bullet.java



```

package com.yourgroup.airbattle.input;

import javafx.scene.Scene;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;

import java.util.HashSet;
import java.util.Set;

/**
 * InputHandler manages keyboard input for the game.
 *
 * <p>This class tracks the current pressed/released state of keys using JavaFX key events, allowing the game logic to query input states in a frame-independent manner.</p>
 *
 * <p>Design rationale:<br/>
 * <ul>
 *   <li>Key states are stored in a {@link Set}, enabling support for multiple simultaneous key presses (e.g., moving diagonally while firing).</li>
 *   <li>Input handling is decoupled from game logic, improving modularity and testability.</li>
 *   <li>Key bindings are centralized, making future remapping easier.</li>
 * </ul>
 * </p>
 */
public class InputHandler {

```



```

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
     */

```

XIAMEN UNIVERSITY MALAYSIA

```

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.
     *
     * <p>This should be called when switching game states (e.g., starting a new game
     * or returning to the menu) to prevent stuck-key behavior.</p>
     */
    public void clear() { pressedKeys.clear(); }
}

```

\Enemy.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:</p>
 * <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.
     */
}

```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side screenshots of the Eclipse IDE interface, both displaying the same Java code for the `Enemy` class.

Project Structure:

- Left window: Project name: AirBattleFinal, master branch. Project structure: AirBattle > settings > org.eclipse.core.resources.pre, org.eclipse.jdt.core.prefs; bin; src > com.yourgroup.airbattle > core, input, objects. The `objects` folder contains: GameConfig, GameLoop, GameState, GameWorld, ObjectType, Bullet, Enemy, EnemyType1, EnemyType2, EnemyType3, GameObject, Item, ItemHeal, ItemRampage, ItemShield, ItemShotgun, ItemSuper, Player, ShieldEffect.
- Right window: Project name: AirBattleFinal, master branch. Project structure: AirBattle > src > com > yourgroup > airbattle > objects > Enemy. The `Enemy` class is selected.

Code View:

```

public class Enemy extends GameObject {
    /**
     * <p>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).</p>
     */
    protected double patrolLimitX = com.yourgroup.airbattle.core.GameConfig.WIDTH;

    /**
     * Updates the horizontal patrol limit for this enemy.
     *
     * <p>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.</p>
     *
     * @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.
    }
}

```

Code View (Bottom Window):

```

public class Enemy extends GameObject {
    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.
 *
 * <p>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.</p>
 *
 * @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.
}

```

XIAMEN UNIVERSITY MALAYSIA

```

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();
        }
    }

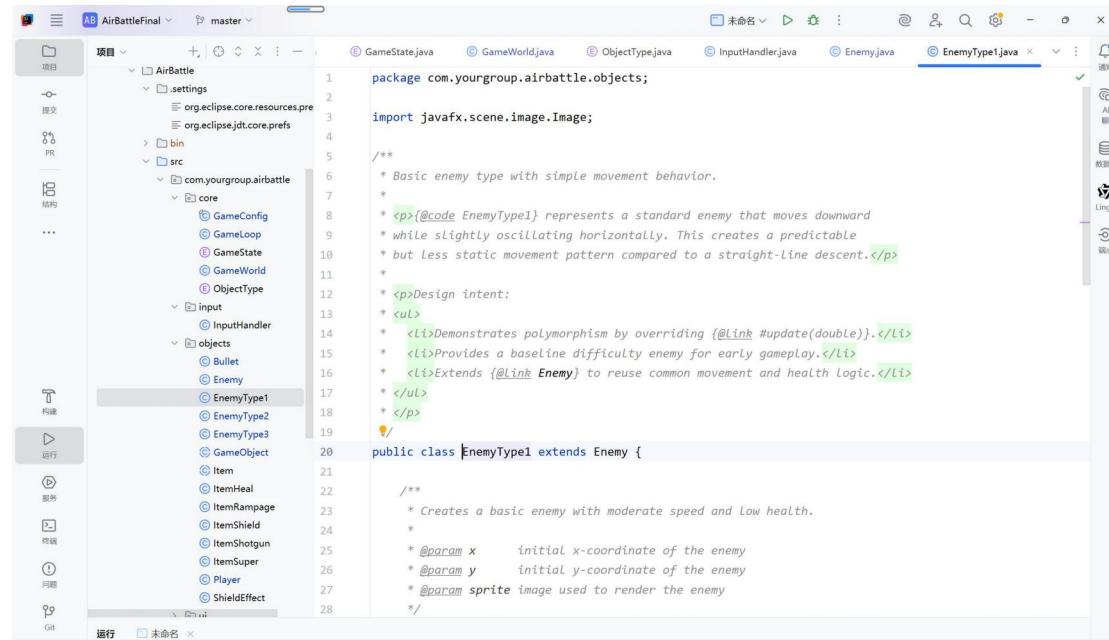
    /**
     * @return current health points of the enemy
     */
    public int getHP() { return hp; }

    /**
     * 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;

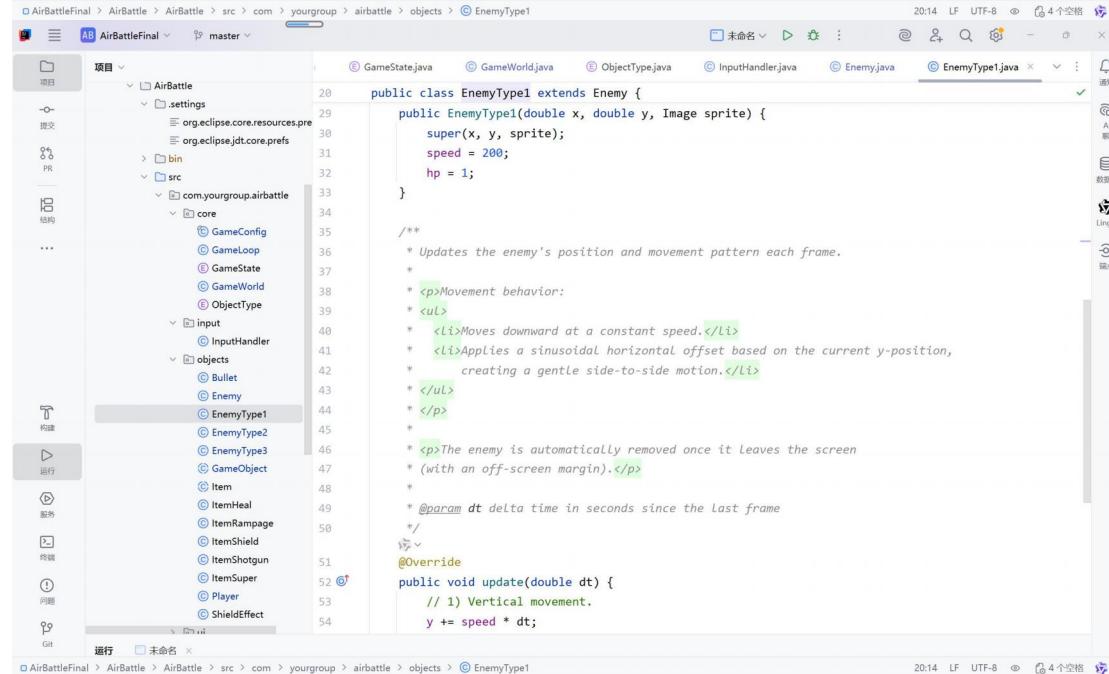
import javafx.scene.image.Image;

/**
 * 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:</p>
 * <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
     */

```

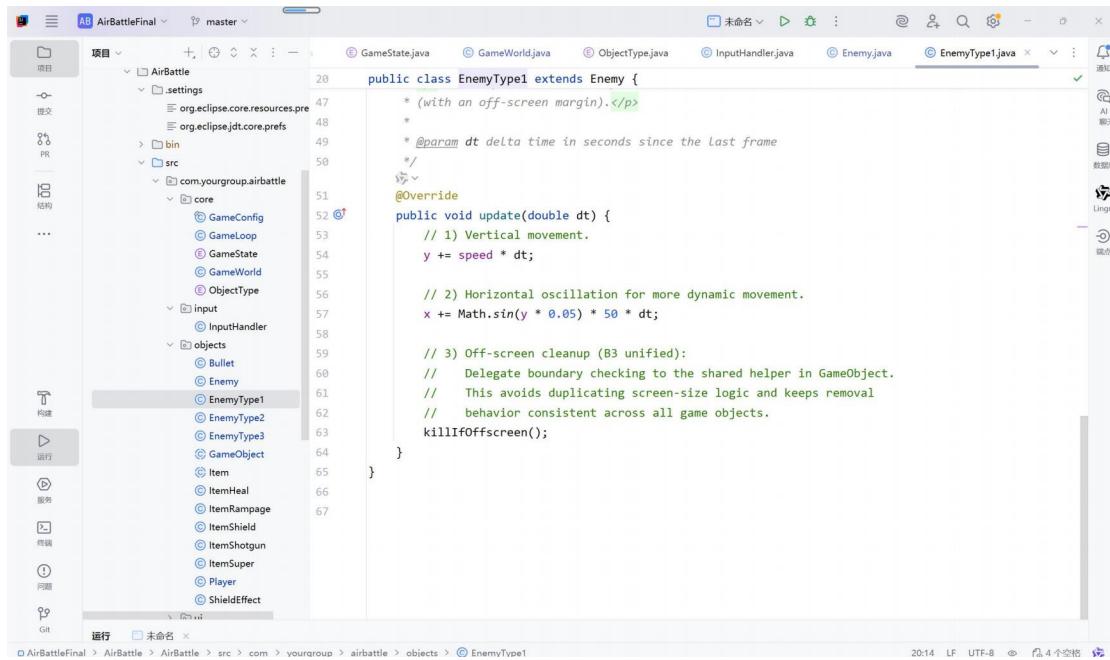
```

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:</p>
     * <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;
    }
}

```

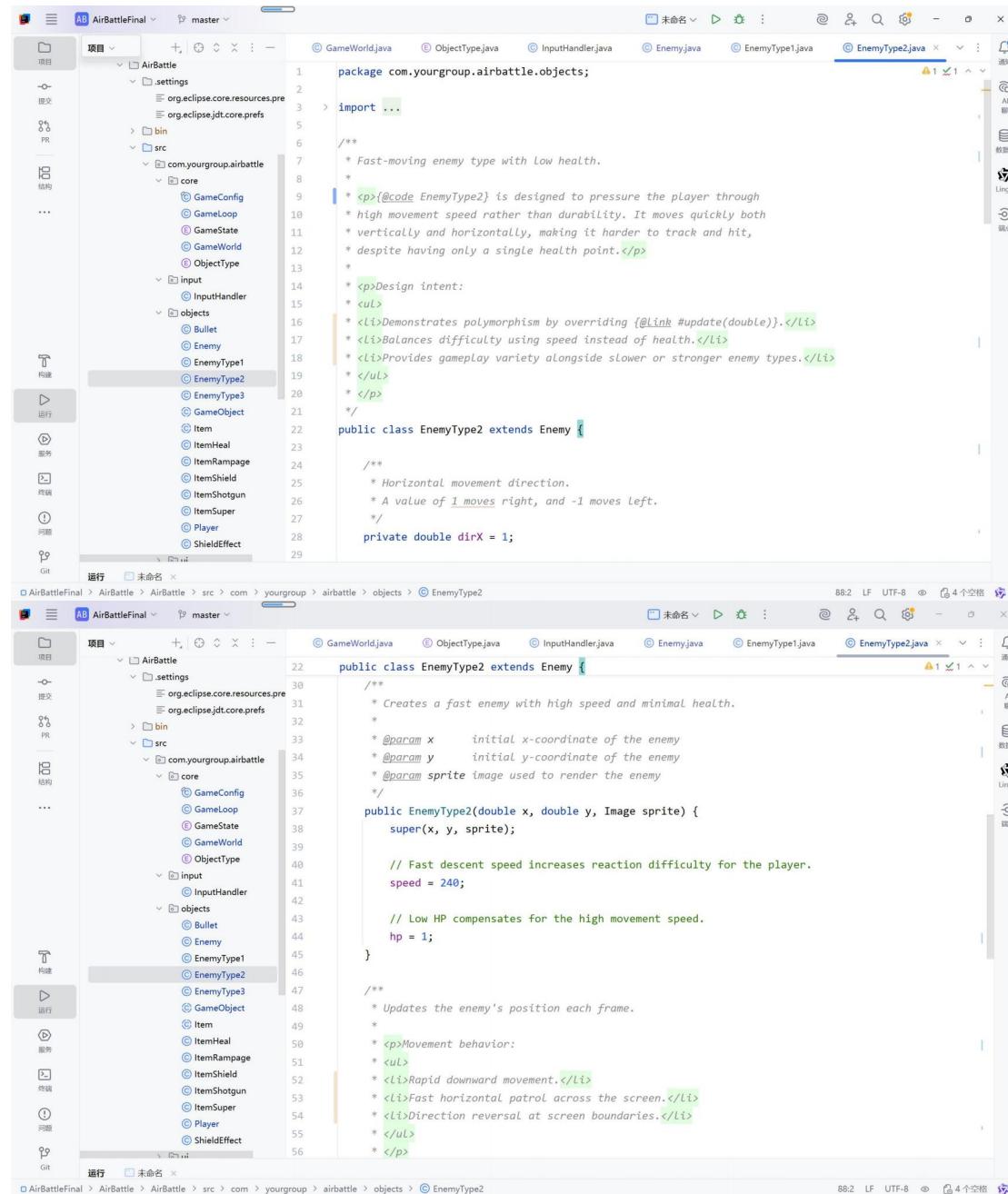
XIAMEN UNIVERSITY MALAYSIA



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "AirBattle". The "src" folder contains packages like "com.yourgroup.airbattle.core", "com.yourgroup.airbattle.input", and "com.yourgroup.airbattle.objects". Inside "objects", the file "EnemyType1.java" is selected.
- Code Editor:** Displays the content of "EnemyType1.java". The code defines a class "EnemyType1" that extends "Enemy". It includes comments explaining movement logic: vertical movement (y += speed * dt), horizontal oscillation (x += Math.sin(y * 0.05) * 50 * dt), and off-screen cleanup (killIfOffscreen). The code editor has syntax highlighting and line numbers.
- Toolbars and Status Bar:** Standard Eclipse toolbars are visible at the top. The status bar at the bottom right shows the time as 20:14, file encoding as UTF-8, and a message about 4 blank lines.

\EnemyType2.java



The image shows two screenshots of an IDE interface, likely Eclipse, displaying Java code for an enemy type. Both screenshots show the same file, `EnemyType2.java`, with minor differences in content.

```

1 package com.yourgroup.airbattle.objects;
2
3 import ...
4
5 /**
6  * Fast-moving enemy type with low health.
7  *
8  * <p>{@code EnemyType2} is designed to pressure the player through
9  * high movement speed rather than durability. It moves quickly both
10 * vertically and horizontally, making it harder to track and hit,
11 * despite having only a single health point.</p>
12 *
13 * <p>Design intent:</p>
14 * <ul>
15 * <li>Demonstrates polymorphism by overriding {@Link #update(double)}.</li>
16 * <li>Balances difficulty using speed instead of health.</li>
17 * <li>Provides gameplay variety alongside slower or stronger enemy types.</li>
18 * </ul>
19 * </p>
20 */
21
22 public class EnemyType2 extends Enemy {
23
24     /**
25      * Horizontal movement direction.
26      * A value of 1 moves right, and -1 moves left.
27      */
28     private double dirX = 1;
29
30
31     /**
32      * Creates a fast enemy with high speed and minimal health.
33      *
34      * @param x initial x-coordinate of the enemy
35      * @param y initial y-coordinate of the enemy
36      * @param sprite image used to render the enemy
37      */
38
39     public EnemyType2(double x, double y, Image sprite) {
40         super(x, y, sprite);
41
42         // Fast descent speed increases reaction difficulty for the player.
43         speed = 240;
44
45         // Low HP compensates for the high movement speed.
46         hp = 1;
47     }
48
49
50     /**
51      * Updates the enemy's position each frame.
52      *
53      * <p>Movement behavior:</p>
54      * <ul>
55      * <li>Rapid downward movement.</li>
56      * <li>Fast horizontal patrol across the screen.</li>
57      * <li>Direction reversal at screen boundaries.</li>
58      * </ul>
59      * </p>
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
249
249
250
251
252
253
254
255
255
256
257
257
258
259
259
260
261
262
263
264
265
265
266
267
267
268
269
269
270
271
272
273
274
275
275
276
277
277
278
279
279
280
281
282
283
284
285
285
286
287
287
288
289
289
290
291
292
293
294
295
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
307
308
309
309
310
311
311
312
313
313
314
315
315
316
317
317
318
319
319
320
321
321
322
323
323
324
325
325
326
327
327
328
329
329
330
331
331
332
333
333
334
335
335
336
337
337
338
339
339
340
341
341
342
343
343
344
345
345
346
347
347
348
349
349
350
351
351
352
353
353
354
355
355
356
357
357
358
359
359
360
361
361
362
363
363
364
365
365
366
367
367
368
369
369
370
371
371
372
373
373
374
375
375
376
377
377
378
379
379
380
381
381
382
383
383
384
385
385
386
387
387
388
389
389
390
391
391
392
393
393
394
395
395
396
397
397
398
399
399
400
401
401
402
403
403
404
405
405
406
407
407
408
409
409
410
411
411
412
413
413
414
415
415
416
417
417
418
419
419
420
421
421
422
423
423
424
425
425
426
427
427
428
429
429
430
431
431
432
433
433
434
435
435
436
437
437
438
439
439
440
441
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
451
452
453
453
454
455
455
456
457
457
458
459
459
460
461
461
462
463
463
464
465
465
466
467
467
468
469
469
470
471
471
472
473
473
474
475
475
476
477
477
478
479
479
480
481
481
482
483
483
484
485
485
486
487
487
488
489
489
490
491
491
492
493
493
494
495
495
496
497
497
498
499
499
500
501
501
502
503
503
504
505
505
506
507
507
508
509
509
510
511
511
512
513
513
514
515
515
516
517
517
518
519
519
520
521
521
522
523
523
524
525
525
526
527
527
528
529
529
530
531
531
532
533
533
534
535
535
536
537
537
538
539
539
540
541
541
542
543
543
544
545
545
546
547
547
548
549
549
550
551
551
552
553
553
554
555
555
556
557
557
558
559
559
560
561
561
562
563
563
564
565
565
566
567
567
568
569
569
570
571
571
572
573
573
574
575
575
576
577
577
578
579
579
580
581
581
582
583
583
584
585
585
586
587
587
588
589
589
590
591
591
592
593
593
594
595
595
596
597
597
598
599
599
600
601
601
602
603
603
604
605
605
606
607
607
608
609
609
610
611
611
612
613
613
614
615
615
616
617
617
618
619
619
620
621
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
151
```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side Eclipse IDE windows displaying the same Java code for `EnemyType2.java`. The code is part of a game project named `AirBattleFinal`.

```
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;

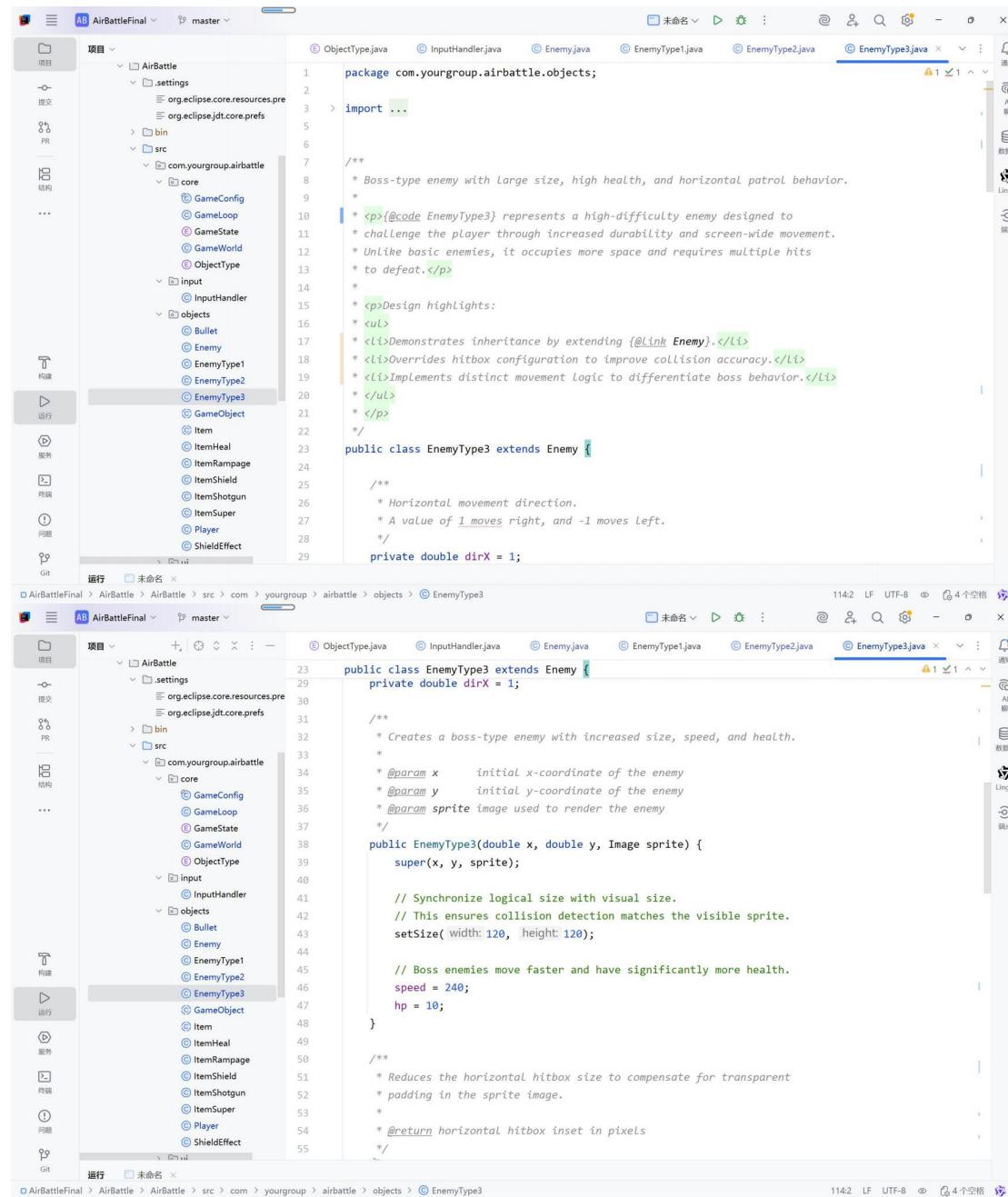
        // 2) Fast horizontal movement.
        x += dirX * 260 * dt;

        // 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();
    }
}
```

The code implements movement logic for the `EnemyType2` class, extending the `Enemy` class. It includes vertical movement, fast horizontal movement, and logic to reverse direction when reaching the screen edges. It also handles off-screen cleanup by delegating boundary checking to a shared helper in `GameObject`.

\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:</p>
 * <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 {
    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
     */
}

```

XIAMEN UNIVERSITY MALAYSIA

```

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:</p>
     * <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>
    */
}

```



```

public class EnemyType3 extends Enemy {
    * <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;

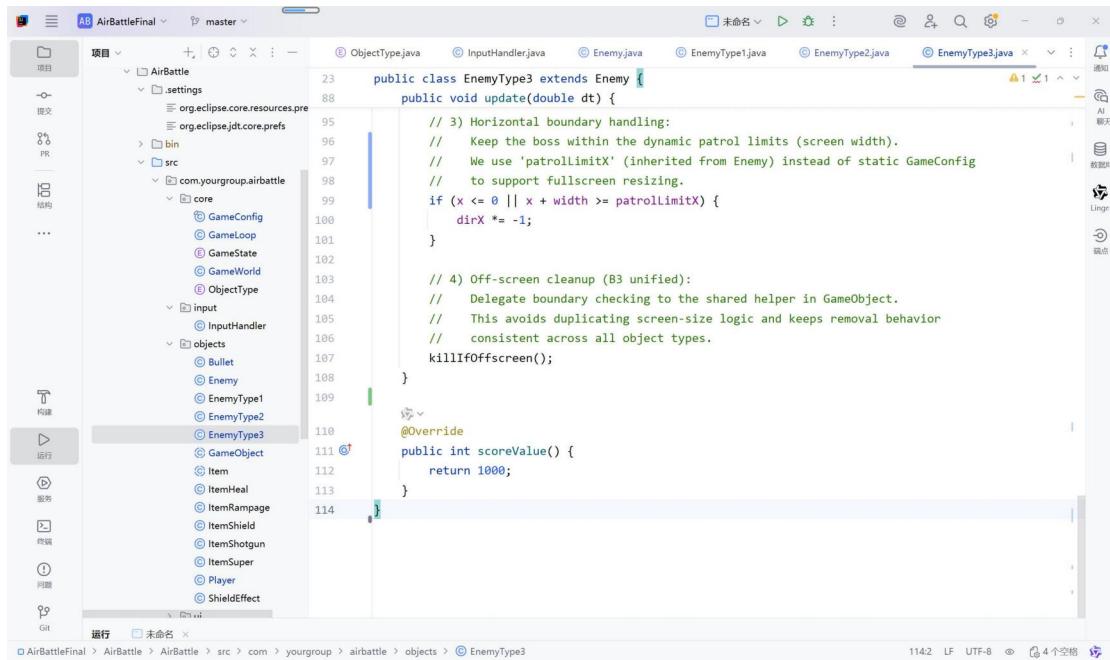
        // 2) Horizontal patrol movement.
        x += dirX * 260 * dt;

        // 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();
    }
}

```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows a Java code editor within an IDE. The project structure on the left includes a 'src' folder containing 'com.yourgroup.airbattle' and 'objects' subfolders, which further contain 'EnemyType3.java'. The code itself is for the `EnemyType3` class, which extends `Enemy`. It includes logic for horizontal boundary handling, off-screen cleanup, and overriding the `scoreValue` method to return 1000.

```
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;
    }
}
```

\GameObject.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:</p>
 */
public abstract class GameObject {
    /**
     * Rendering is separated from Logic: {@link #update(double)} modifies state,
     * while {@link #render()} applies state to the JavaFX node.</li>
     * Hitbox inset hooks allow sprites with transparent padding to have more accurate collisions.</li>
     */
    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();
    }
}

```

XIAMEN UNIVERSITY MALAYSIA

```

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()}.
     *
     * @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.
     *
     */
}

```



```

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

    /**
     * Updates the object's internal logic for one frame.
     *
     * Subclasses should update position, timers, AI, cooldowns, etc. here.
     * Rendering should be handled separately via {@link #render()}.
     *
     * @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.
     *
     * By default, this simply relocates the view. Subclasses may override
     * if they need additional visual updates (e.g., rotation, animation).
     */
    public void render() { relocateView(); }

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

```

XIAMEN UNIVERSITY MALAYSIA

```
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}.
     */
}
```

```
public abstract class GameObject {
    * Sets the render size of this object and synchronizes it with the {@link ImageView}.
    *
    * <p>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.</p>
    *
    * | new width in pixels
    * | 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.
     *
     * <p>Default is 0 (hitbox matches sprite bounds). Subclasses may override
     * to shrink the hitbox if the sprite contains transparent padding.</p>
     *
     * @return horizontal inset in pixels
     */
}
```

XIAMEN UNIVERSITY MALAYSIA

```

public abstract class GameObject {
    /**
     * Vertical inset (in pixels) applied to the hitbox on both top and bottom sides.
     *
     * @return vertical inset in pixels
     */
    protected double hitboxInsetY() { return 0; }

    /**
     * Left boundary of the hitbox in world coordinates
     */
    public double hitLeft() { return x + hitboxInsetX(); }

    /**
     * Right boundary of the hitbox in world coordinates
     */
    public double hitRight() { return x + width - hitboxInsetX(); }

    /**
     * Top boundary of the hitbox in world coordinates
     */
    public double hitTop() { return y + hitboxInsetY(); }

    /**
     * Bottom boundary of the hitbox in world coordinates
     */
    public double hitBottom() { return y + height - hitboxInsetY(); }

    // ===== Legacy bounds (sprite bounds) =====

    /**
     * Left boundary of the sprite bounds in world coordinates
     */
    public double left() { return x; }

    /**
     * Right boundary of the sprite bounds in world coordinates
     */
    public double right() { return x + width; }

    /**
     * Top boundary of the sprite bounds in world coordinates
     */
    public double top() { return y; }

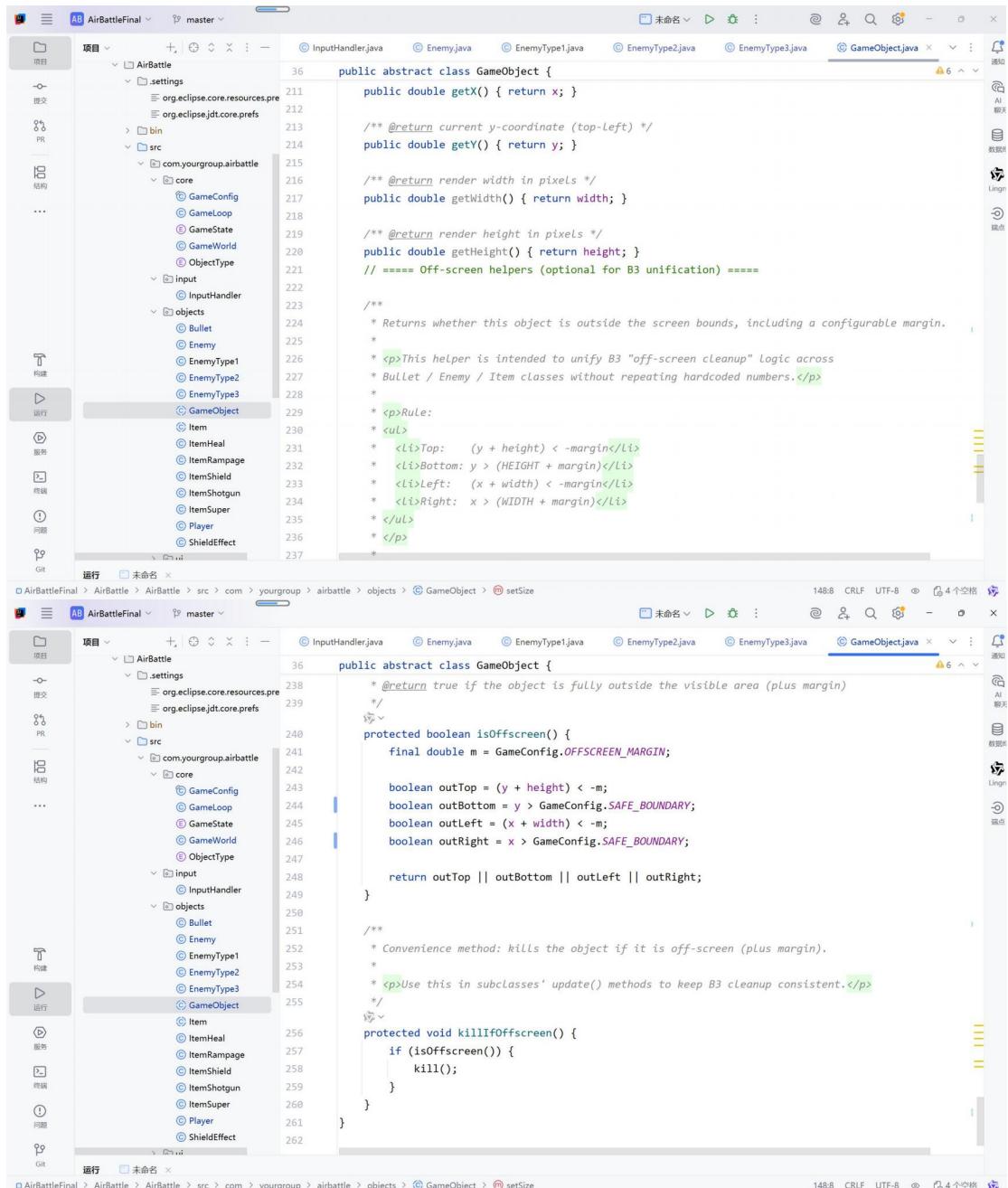
    /**
     * Bottom boundary of the sprite bounds in world coordinates
     */
    public double bottom() { return y + height; }

    /**
     * Current x-coordinate (top-left)
     */
    public double x() { return x; }
}

```

The screenshot displays two instances of the Eclipse IDE interface, each showing the same Java code for the `GameObject` class. The code defines several methods for calculating hitbox boundaries and sprite bounds in world coordinates. The top instance shows the original code, while the bottom instance shows the code with additional methods for `left()`, `right()`, `top()`, and `bottom()`. The code is annotated with Javadoc-style comments describing the purpose of each method.

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "AirBattle". The "src" folder contains packages like "com.yourgroup.airbattle" which further contain "core", "input", and "objects" packages. Within "objects", there is a "GameObject" class.
- Code Editor:** Displays the source code for the `GameObject` class. The code defines an abstract class with methods for getting coordinates, width, height, and checking if it's off-screen. It also includes a convenience method to kill the object if it's off-screen.
- Toolbars and Status Bar:** Standard Eclipse toolbars and status bar at the bottom indicating file count (4个空格).

```

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.
     *
     * <p>This helper is intended to unify B3 "off-screen cleanup" logic across
     * Bullet / Enemy / Item classes without repeating hardcoded numbers.</p>
     *
     * <p><b>Rule:</b>
     * <ul>
     *   <li>Top: (y + height) < -margin</li>
     *   <li>Bottom: y > (HEIGHT + margin)</li>
     *   <li>Left: (x + width) < -margin</li>
     *   <li>Right: x > (WIDTH + margin)</li>
     * </ul>
     * </p>
     */
}

```



```

public abstract class GameObject {
    * @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).
     *
     * <p>Use this in subclasses' update() methods to keep B3 cleanup consistent.</p>
     */
    protected void killIfOffscreen() {
        if (isOffscreen()) {
            kill();
        }
    }
}

```

\Item.java

```

1 package com.yourgroup.airbattle.objects;
2
3 import com.yourgroup.airbattle.core.ObjectType;
4 import javafx.scene.image.Image;
5
6 /**
7  * Abstract base class for all collectible power-up items.
8  *
9  * <p>{@code Item} represents any object that the player can collect to
10 * gain a beneficial effect, such as restoring health or enhancing abilities.
11 * Concrete subclasses define the specific effect applied to the player.</p>
12 *
13 * <p>Shared behavior:</p>
14 * <ul>
15 *   <li>Falls downward at a constant speed.</li>
16 *   <li>Is removed automatically when it Leaves the screen.</li>
17 *   <li>Uses a unified {@link ObjectType#POWERUP} category for collision handling.</li>
18 * </ul>
19 * </p>
20 *
21 * <p>Design intent:</p>
22 * <ul>
23 *   <li>Encapsulates item behavior in a single hierarchy.</li>
24 *   <li>Demonstrates abstraction through the {@link #apply(Player)} method.</li>
25 *   <li>Prevents conditional logic inside {@code Player} by delegating effects
26 *       to individual item subclasses.</li>
27 * </ul>
28 * </p>

```

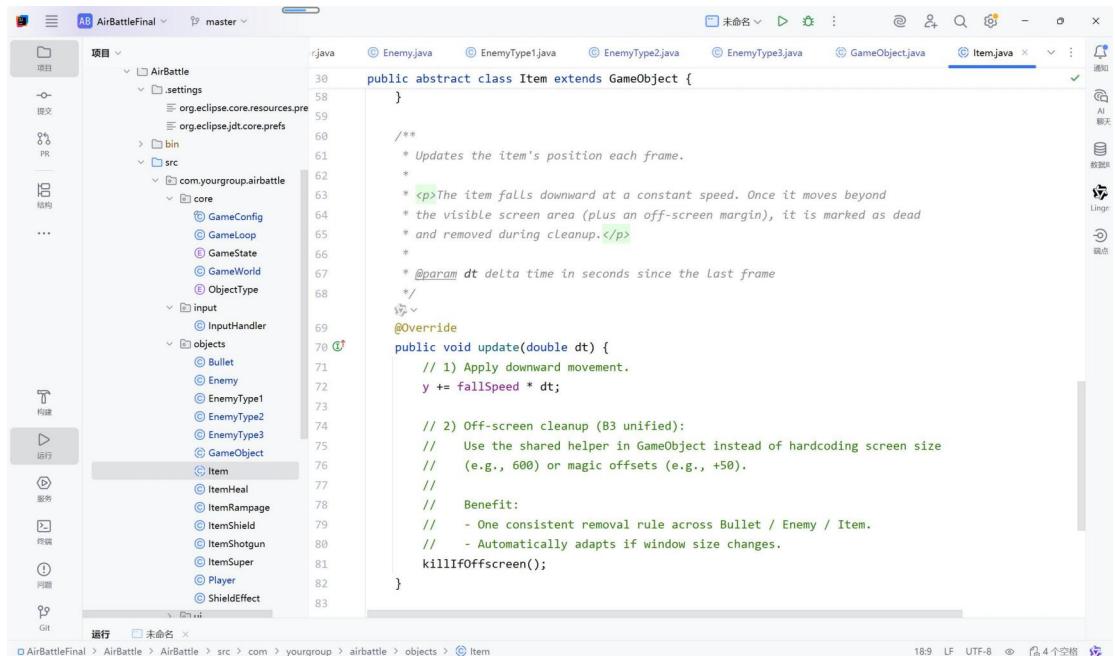


```

1 /*
2  * Vertical falling speed in pixels per second.
3  * Subclasses may adjust this value to change item behavior.
4  */
5 protected double fallSpeed = 80;
6
7 /**
8  * Creates a new collectible item.
9  *
10 * @param x      initial x-coordinate of the item
11 * @param y      initial y-coordinate of the item
12 * @param width  render width of the item
13 * @param height render height of the item
14 * @param sprite image used to render the item
15 */
16 public Item(double x, double y, double width, double height, Image sprite) {
17     super(x, y, width, height);
18 }
19
20 /**
21  * @return the logical object type used for collision filtering
22  */
23 @Override
24 public ObjectType getType() {
25     // Items are treated as POWERUPS for unified collision handling.
26     return ObjectType.POWERUP;
27 }

```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows the Eclipse IDE interface with the Item.java file open in the editor. The code implements the Item class, which extends GameObject. It includes methods for updating the item's position and applying effects to a player.

```
public abstract class Item extends GameObject {
    }

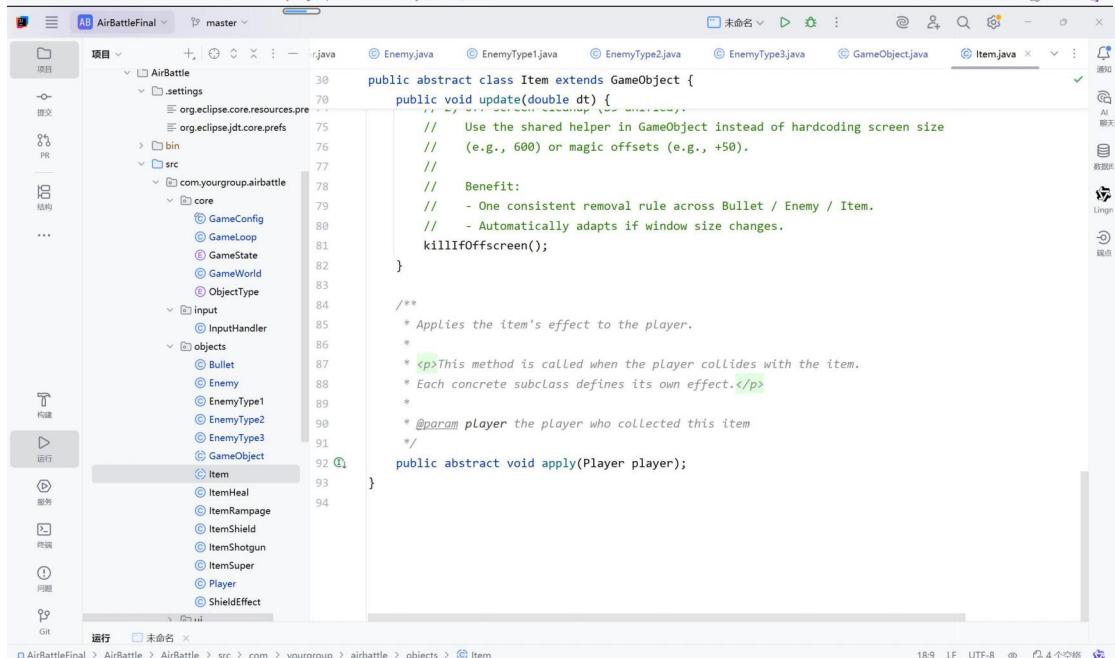
    /**
     * Updates the item's position each frame.
     *
     * <p>The item falls downward at a constant speed. Once it moves beyond
     * the visible screen area (plus an off-screen margin), it is marked as dead
     * and removed during cleanup.</p>
     *
     * @param dt delta time in seconds since 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();
    }

    /**
     * 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);
}
```

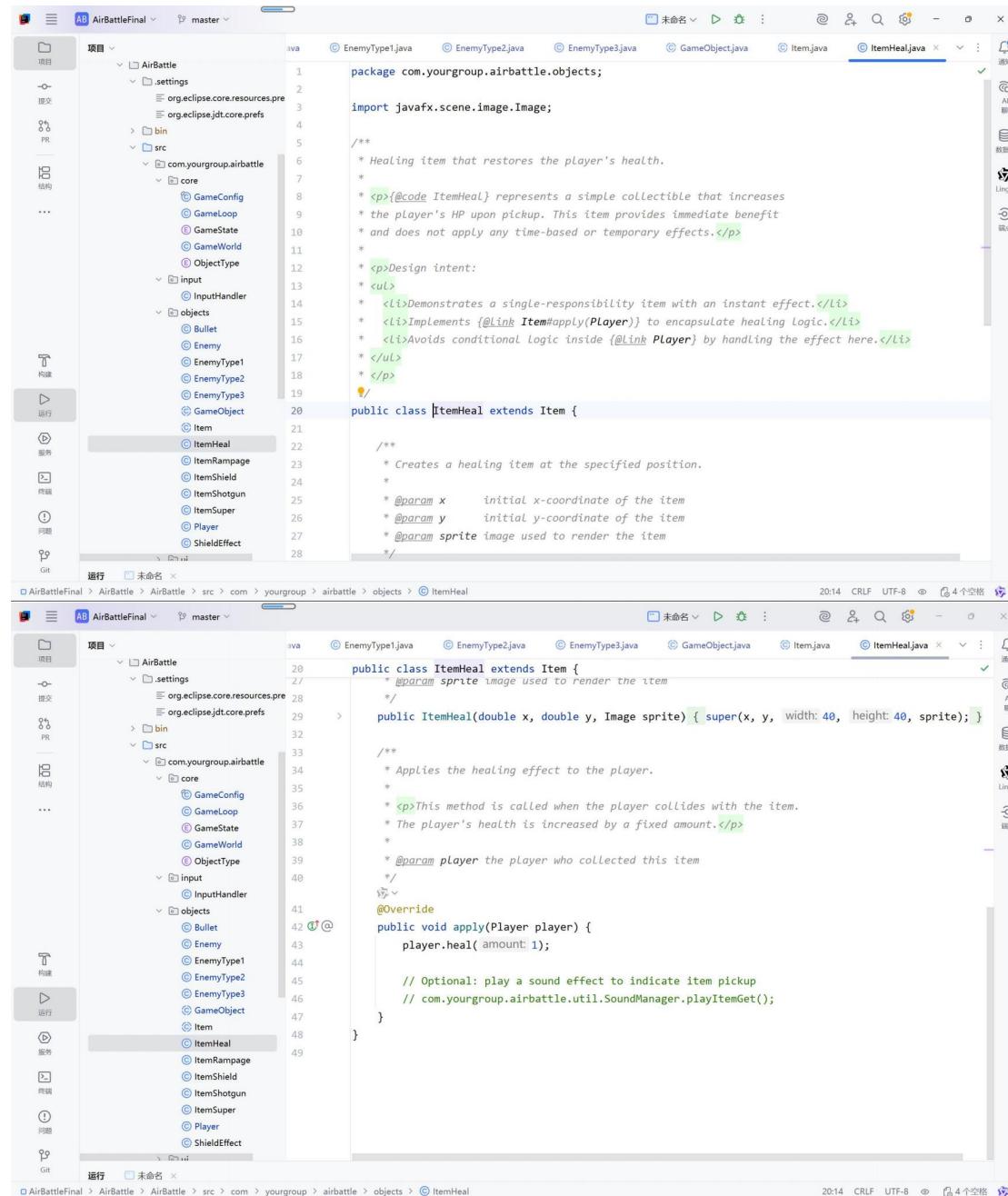
The code editor shows the Item.java file with line numbers 30 to 83. The file is part of the AirBattleFinal project, located in the src/com/yourgroup/airbattle/objects directory. The code is annotated with Javadoc comments and includes annotations like @Override and @param.



The second screenshot shows the same Eclipse IDE interface with the Item.java file open. The code is identical to the first screenshot, but the cursor is positioned on the line starting with 'public abstract void apply(Player player);'.

```
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:</p>
 * <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
     */
}

```



```

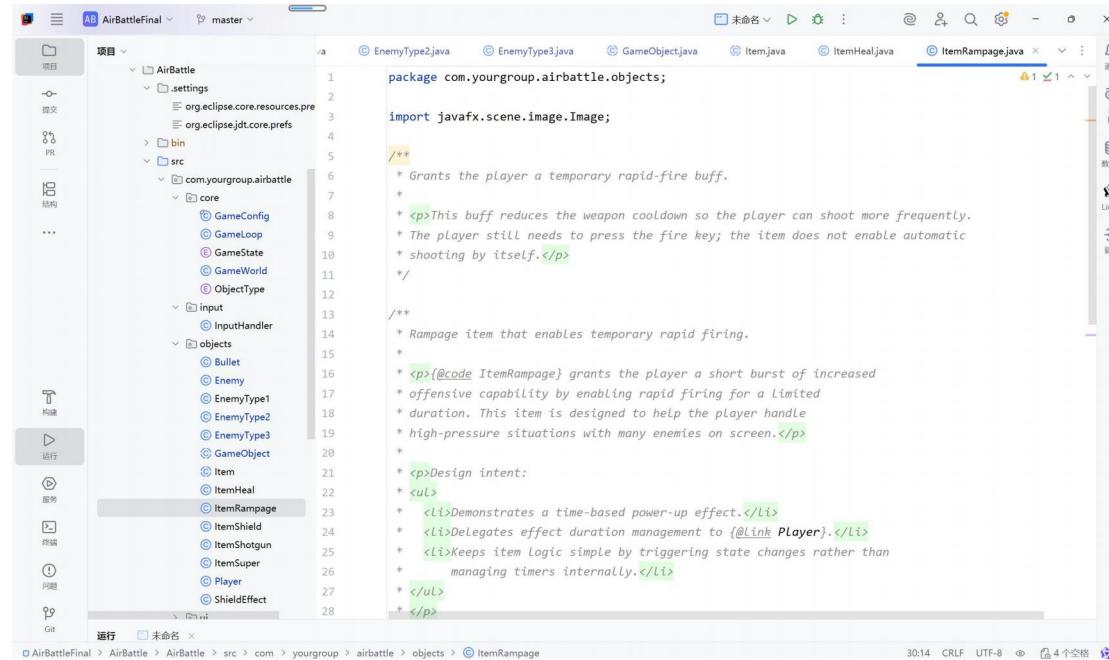
public class ItemHeal extends Item {
    /**
     * @param sprite image used to render the item
     */
    public ItemHeal(double x, double y, Image sprite) { super(x, y, width: 40, height: 40, sprite); }

    /**
     * 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



```

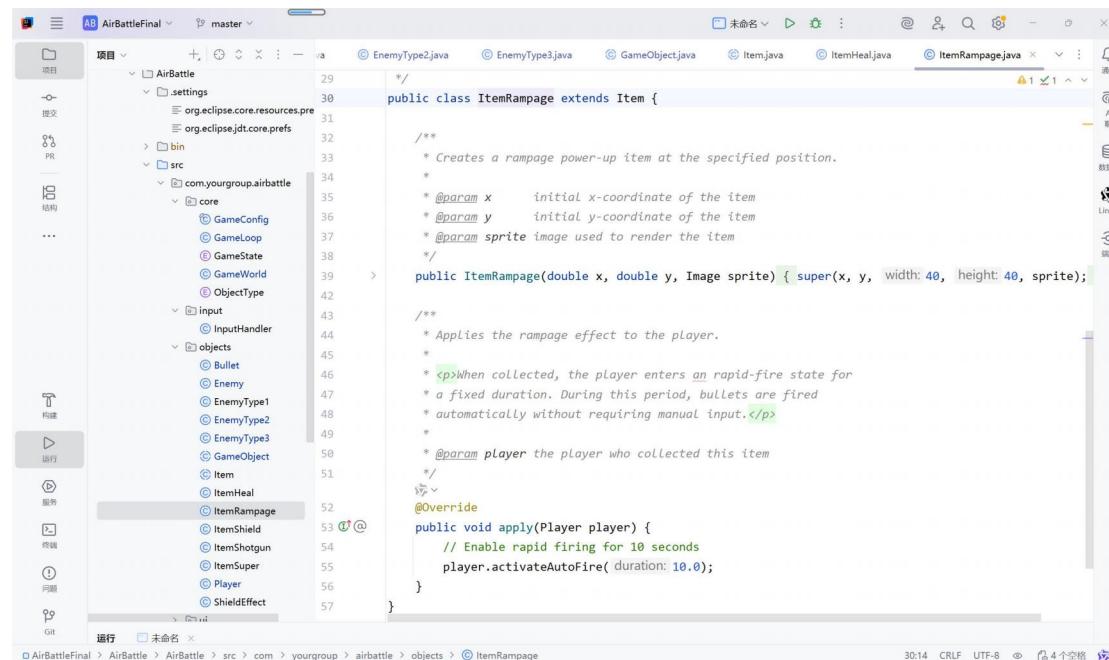
package com.yourgroup.airbattle.objects;

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:</p>
 * <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>
 */

```



```

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>
 *
 * 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>
 */
public class ItemRampage extends Item {

```



```

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); }

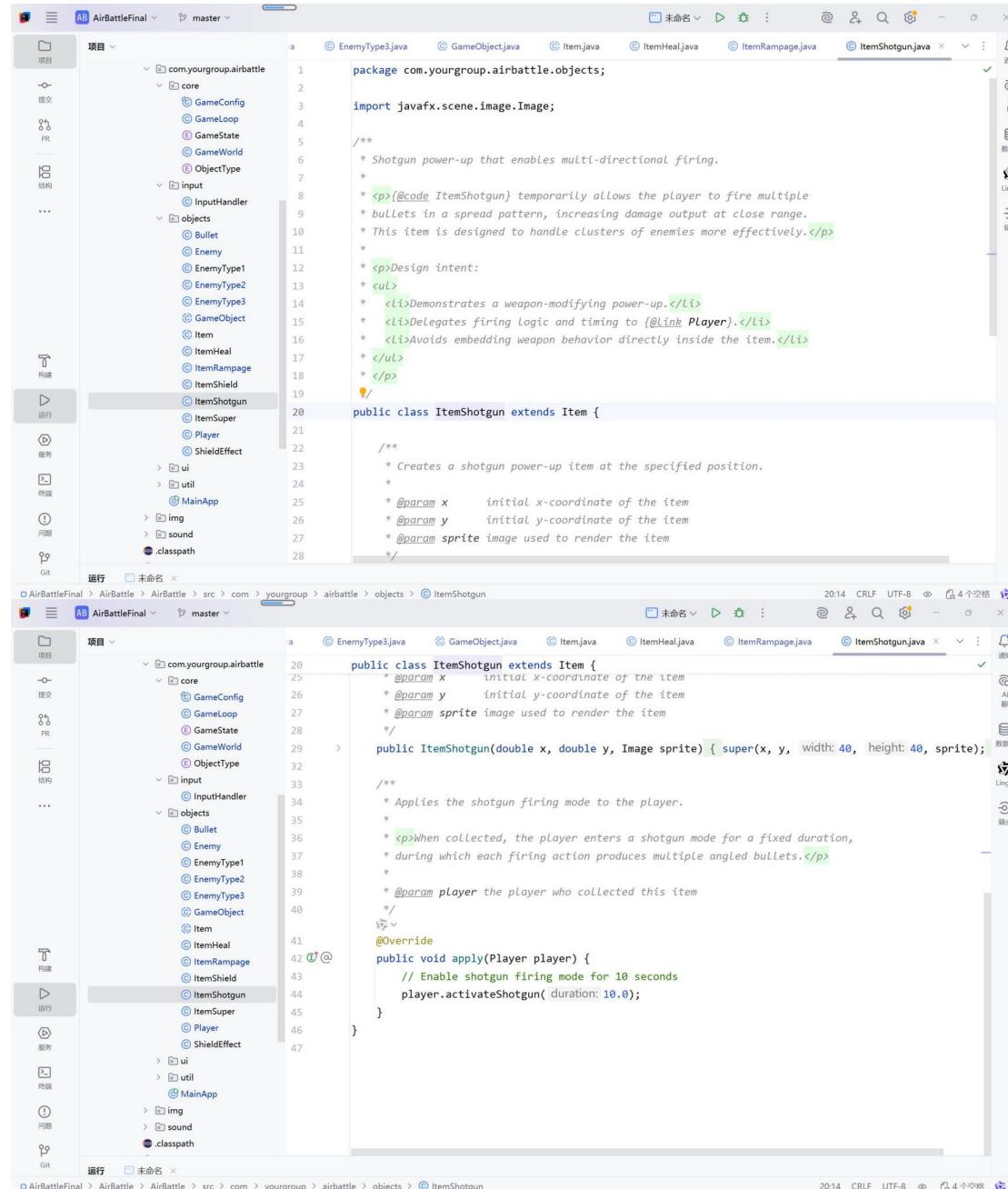
    /**
     * Applies the rampage effect to the player.
     *
     * <p>When collected, the player enters an rapid-fire state for  

     * 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



```

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:</p>
 * <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

```

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:</p>
 * <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
     */
}

```



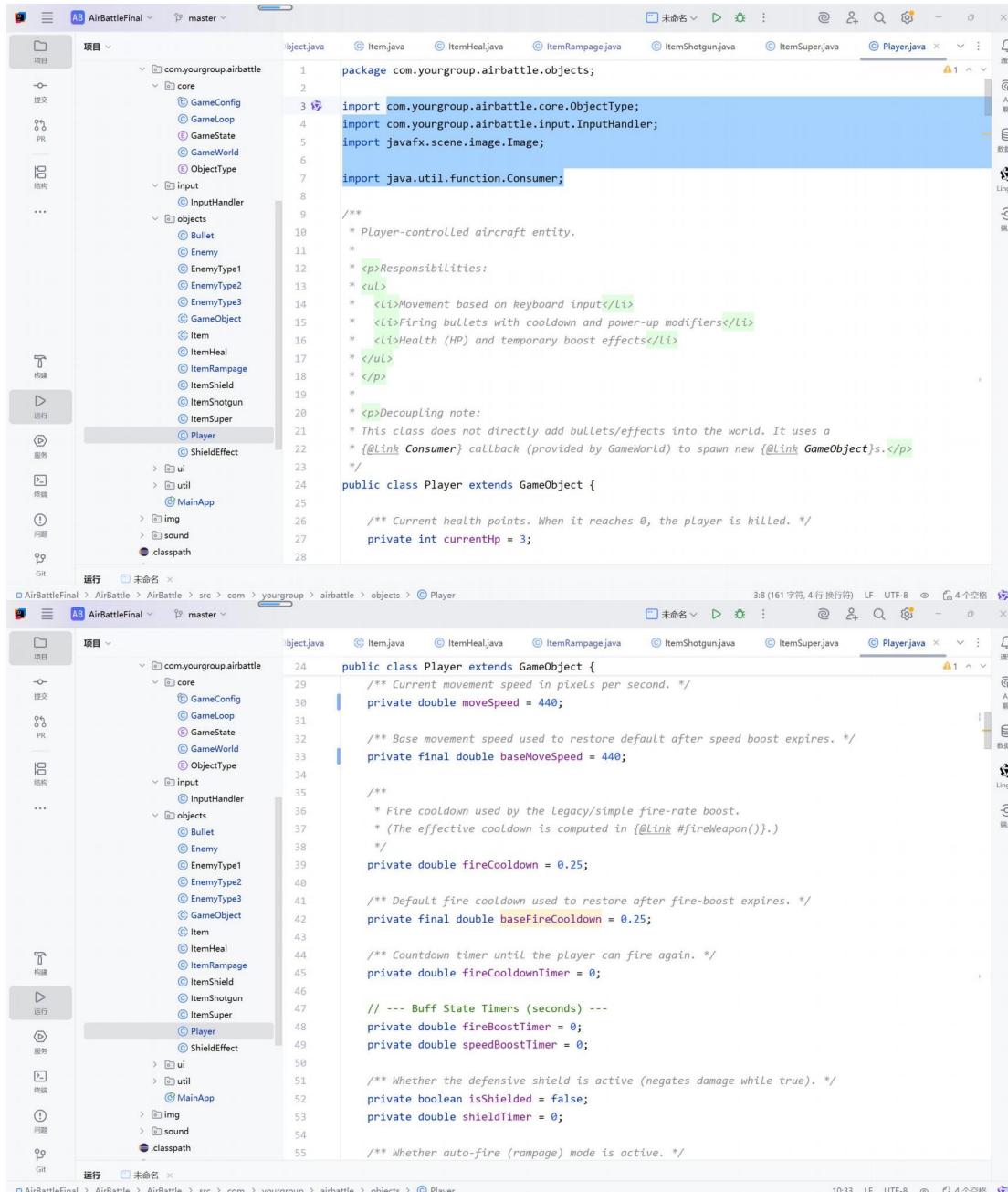
```

public class ItemSuper extends Item {
    /**
     * initial y-coordinate of the item
     * 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



The image shows two identical Java code snippets for the `Player` class, displayed side-by-side in a code editor. The code is part of a project named `AirBattleFinal` under the package `com.yourgroup.airbattle.objects`. The code defines a class `Player` that extends `GameObject`. It includes imports for `ObjectType`, `InputHandler`, and `Consumer`. The class has a private variable `currentHp` set to 3. The code is annotated with Javadoc comments describing the player's responsibilities and decoupling from the world.

```

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:</p>
 * <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>Decoupling note:</p>
 * 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;
}

```

XIAMEN UNIVERSITY MALAYSIA

```

public class Player extends GameObject {
    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, width: 40, height: 40, sprite);
        this.input = input;
    }
}

```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side Java code editors from an IDE. Both editors are displaying the same file, `Player.java`, located at `com.yourgroup.airbattle.objects.Player`.

Editor 1 (Top): Shows the `update()` method. The code is annotated with Javadoc comments and includes several overridden methods (`getObjectType`, `hitboxInsetX`, `hitboxInsetY`). The code is as follows:

```
public class Player extends GameObject {
    ...
    public Player(double x, double y) {
        ...
    }
    ...
    @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; }

    /**
     * @param dt delta time in seconds
     */
    @Override
    protected double hitboxInsetY() { return 6; }

    ...
}
```

Editor 2 (Bottom): Shows the `update()` method. The code is annotated with Javadoc comments and includes several overridden methods (`getObjectType`, `hitboxInsetX`, `hitboxInsetY`). The code is as follows:

```
public class Player extends GameObject {
    ...
    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;
        }

        // Expire auto-fire
    }
}
```

XIAMEN UNIVERSITY MALAYSIA

```

public class Player extends GameObject {
    public void update(double dt) {
        // Expire auto-tire
        if (isRapidFire) {
            autoFocusTimer -= 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) ---
    }
}

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:</p>
         * <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 <a href="#bullet">Bullet</a> objects via <a href="#spawner">#spawner</a>.</li>
         * </ul>
         */
        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);
        }
    }
}

```

XIAMEN UNIVERSITY MALAYSIA

```

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 + 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 + width / 2 - bulletWidth / 2, y: y - 20,
                bulletWidth, bulletHeight, sprite, damage, speedX: -100);
            spawner.accept(left);
            Bullet right = new Bullet(x + width / 2 - bulletWidth / 2, y: y - 20,
                bulletWidth, bulletHeight, sprite, damage, speedX: 100);
            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;
        }

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

```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side Java code editors. Both editors have a similar interface with a left sidebar containing project navigation, a central code editor area, and a right sidebar with various icons.

Top Editor (Player.java):

```
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.
     */
}
```

Bottom Editor (update() method):

```
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.
 */
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));
    }
}
```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side Java code editors. Both editors have a top navigation bar with tabs for '未命名' (Untitled), 'Player.java', and 'Object.java'. The left editor's title bar says 'AirBattleFinal' and the right one says 'AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > objects > Player > update'. The left editor displays the 'Player.java' code, which extends 'GameObject'. It contains methods for activating different shield modes: 'activateShield', 'activateAutoFire', 'activateSuperBullet', and 'activateShotgun'. The right editor displays the 'Object.java' code, which is a base class for various game objects like 'Player', 'Enemy', 'Item', etc. It includes a constructor, a 'superObject' field, and methods for setting and getting shield status ('isShielded' and 'getHp'). Both editors show line numbers on the left and have toolbars and status bars at the bottom.

```
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).
     */
}
```

```
public class Object extends GameObject {
    protected boolean isSuperBullet = false;
    protected double superBulletTimer = 0;

    /**
     * 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

```

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
     */
}

```



```

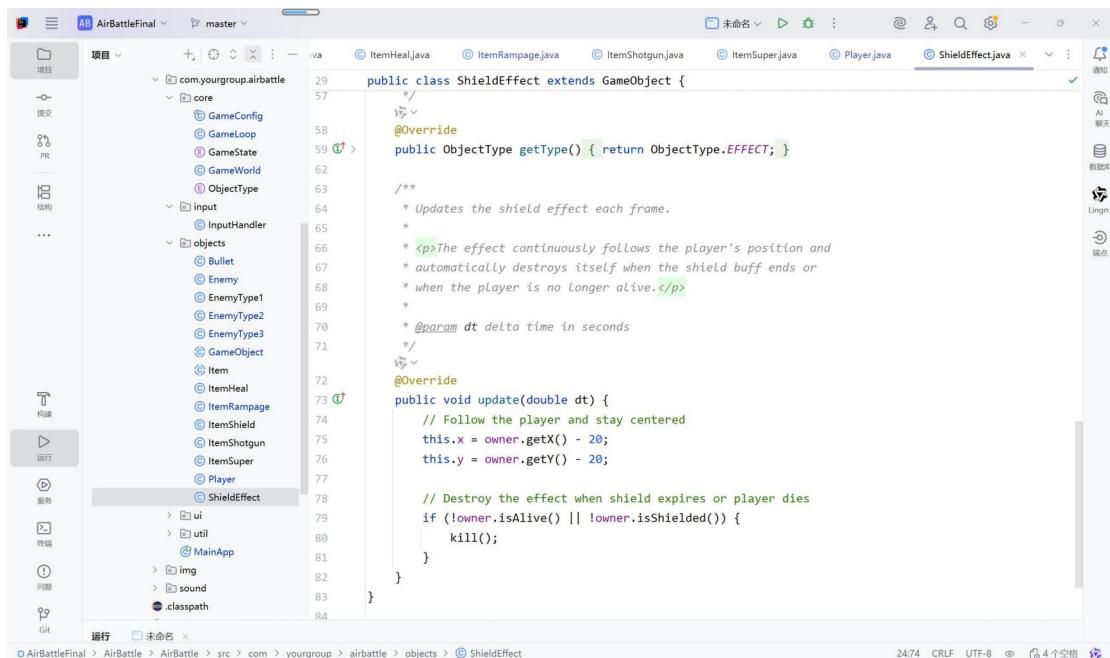
package com.yourgroup.airbattle.objects;

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:</p>
 * <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>Design intent:</p>
 * <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>
 */

```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows a Java code editor within an IDE. The project structure on the left includes packages like com.yourgroup.airbattle, core, input, objects, and others. The current file is ShieldEffect.java, which extends GameObject. The code implements a shield effect that follows the player's position and destroys itself when the shield buff ends or the player dies. It uses annotations like @Override and @param.

```
public class ShieldEffect extends GameObject {
    /**
     * Updates the shield effect each frame.
     *
     * <p>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.</p>
     *
     * @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();
        }
    }
}
```

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

\GameOverMenu.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:</p>
 * <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 {

```



```

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;

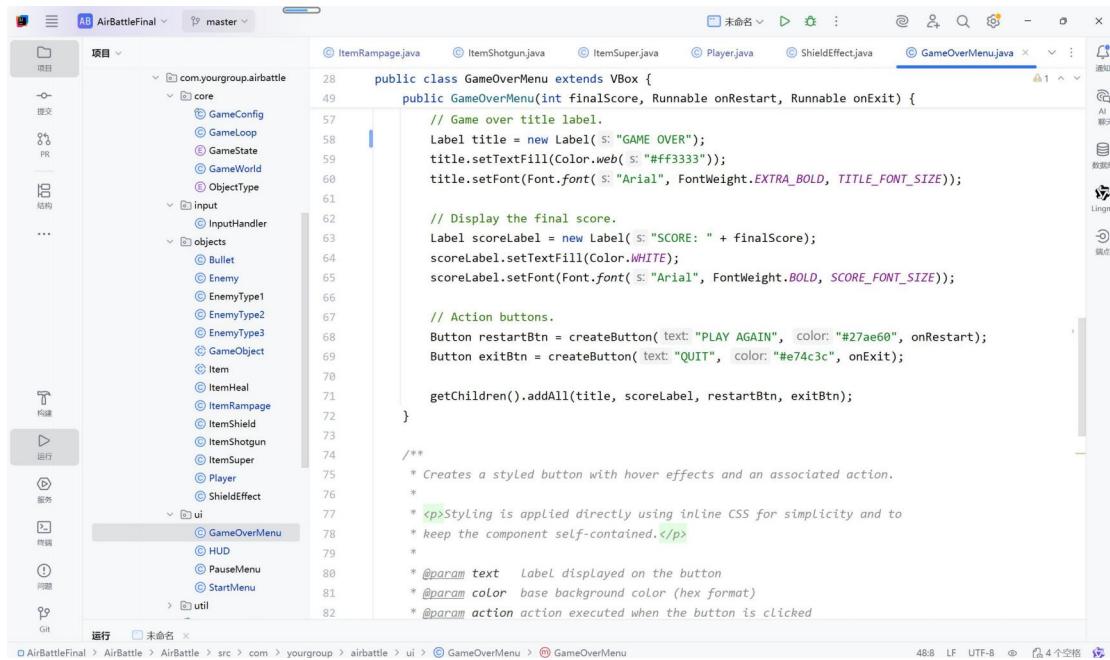
    private static final double BUTTON_WIDTH = 220;
    private static final double BUTTON_HEIGHT = 50;

    /**
     * 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);");
    }
}

```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows a Java IDE interface with the following details:

- Project Explorer:** Shows the project structure under "AirBattleFinal". The "GameOverMenu.java" file is selected.
- Code Editor:** Displays the content of GameOverMenu.java. The code is a Java class extending VBox, defining a game-over menu with a title label, a score label, and action buttons for restart and quit.
- Toolbars and Menus:** Standard IDE toolbars and menus are visible at the top.
- Status Bar:** Shows file paths, encoding (UTF-8), and other system information.

```
public class GameOverMenu extends VBox {
    public GameOverMenu(int finalScore, Runnable onRestart, Runnable onExit) {
        // Game over title label.
        Label title = new Label("GAME OVER");
        title.setTextFill(Color.web("#ff3333"));
        title.setFont(Font.font("Arial", FontWeight.EXTRA_BOLD, TITLE_FONT_SIZE));

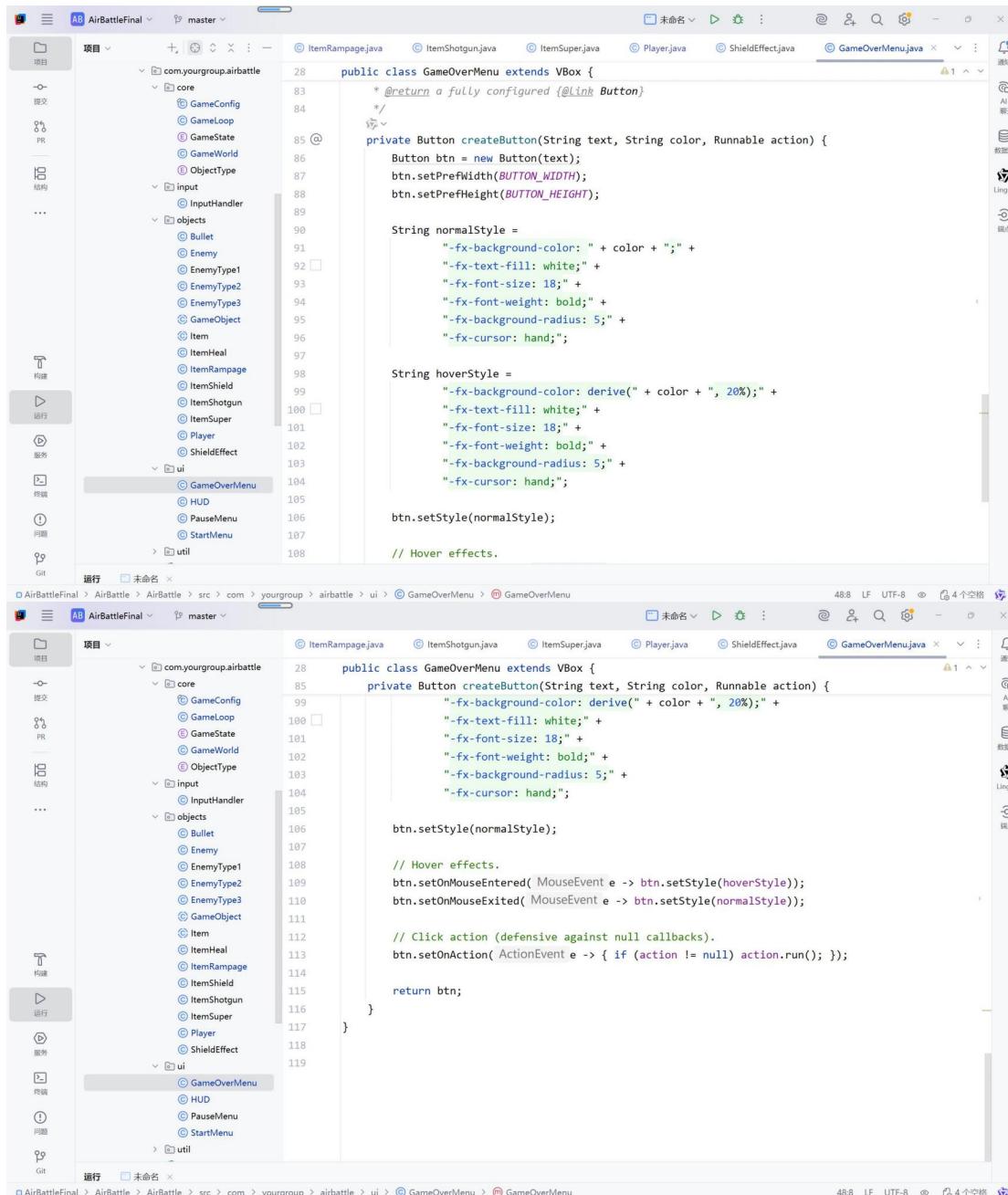
        // Display the final score.
        Label scoreLabel = new Label("SCORE: " + finalScore);
        scoreLabel.setTextFill(Color.WHITE);
        scoreLabel.setFont(Font.font("Arial", FontWeight.BOLD, SCORE_FONT_SIZE));

        // Action buttons.
        Button restartBtn = createButton("PLAY AGAIN", "#27ae60", onRestart);
        Button exitBtn = createButton("QUIT", "#e74c3c", onExit);

        getChildren().addAll(title, scoreLabel, restartBtn, exitBtn);
    }

    /**
     * Creates a styled button with hover effects and an associated action.
     *
     * <p>Styling is applied directly using inline CSS for simplicity and to
     * keep the component self-contained.</p>
     *
     * @param text Label displayed on the button
     * @param color base background color (hex format)
     * @param action action executed when the button is clicked
     */
}
```

XIAMEN UNIVERSITY MALAYSIA

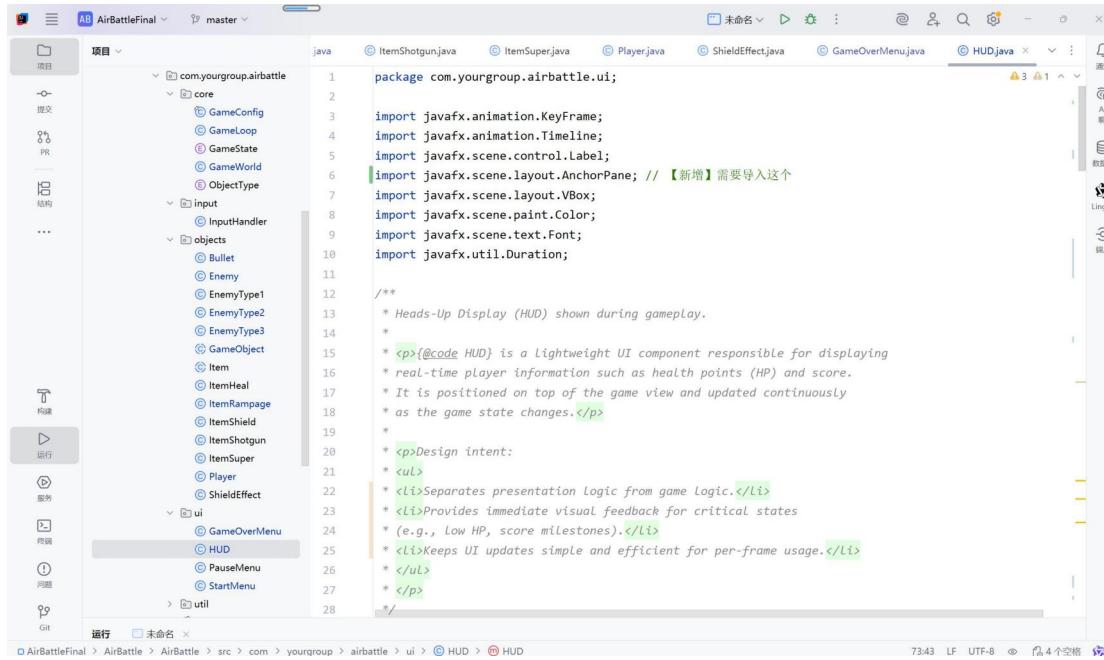


The image shows two side-by-side Java code editors, likely from an IDE like IntelliJ IDEA. Both editors are displaying the same file, `GameOverMenu.java`, which is a subclass of `VBox`.

```
public class GameOverMenu extends VBox {  
    * @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;" +  
            "-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));  
  
        // Click action (defensive against null callbacks).  
        btn.setOnAction( ActionEvent e -> { if (action != null) action.run(); } );  
    }  
}
```

The code implements a button creation logic with styles for normal and hover states. It also handles mouse events to change the button's style when the mouse enters or exits it. The click action is handled through a defensive null check.

\HUD.java



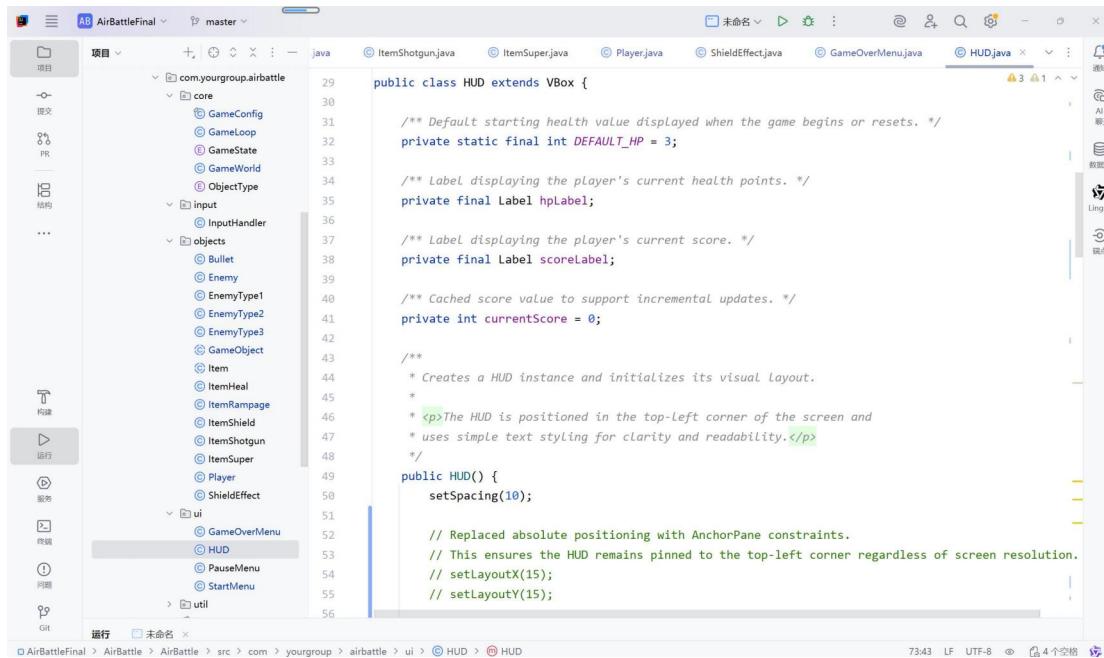
```

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:</p>
 * <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>
 */

```



```

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);
    }
}

```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side Java code editors, likely from an IDE like IntelliJ IDEA. Both editors are displaying the same file, `HUD.java`, which is part of the `com.yourgroup.airbattle` package. The code implements a `VBox` for the User Interface (UI) of a game.

```
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(this, 20.0);
        AnchorPane.setLeftAnchor(this, 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("HP: " + DEFAULT_HP);
        scoreLabel = new Label("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>

```

The second editor shows a continuation of the code, where the `setHp` method is implemented. It uses CSS styles to change the text color based on the player's current health points (hp).

```
        * <li>Green: safe</li>
        * <li>Orange: warning</li>
        * <li>Red: critical</li>
        * </ul>
        *

        * @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.
     */

```

XIAMEN UNIVERSITY MALAYSIA

The screenshot shows the IntelliJ IDEA interface with the project 'AirBattleFinal' open. The left sidebar displays the project structure, showing packages like com.yourgroup.airbattle.core, com.yourgroup.airbattle.objects, and com.yourgroup.airbattle.ui. The right pane shows the code editor for 'HUD.java'. The code implements a `VBox` for the User Interface (UI) and handles score management. The code is annotated with Javadoc comments and includes methods for setting and adding scores.

```
public class HUD extends VBox {
    ...
    public void setScore(int score) {
        ...
    }
    ...
    public void addScore(int points) {
        setScore(this.currentScore + points);
    }
}
```

XIAMEN UNIVERSITY MALAYSIA

The image shows two side-by-side Java code editors, likely from an IDE like IntelliJ IDEA. Both editors display the same file, `HUD.java`, which is part of a project structure under `com.yourgroup.airbattle.ui.HUD`.

Editor 1 (Left):

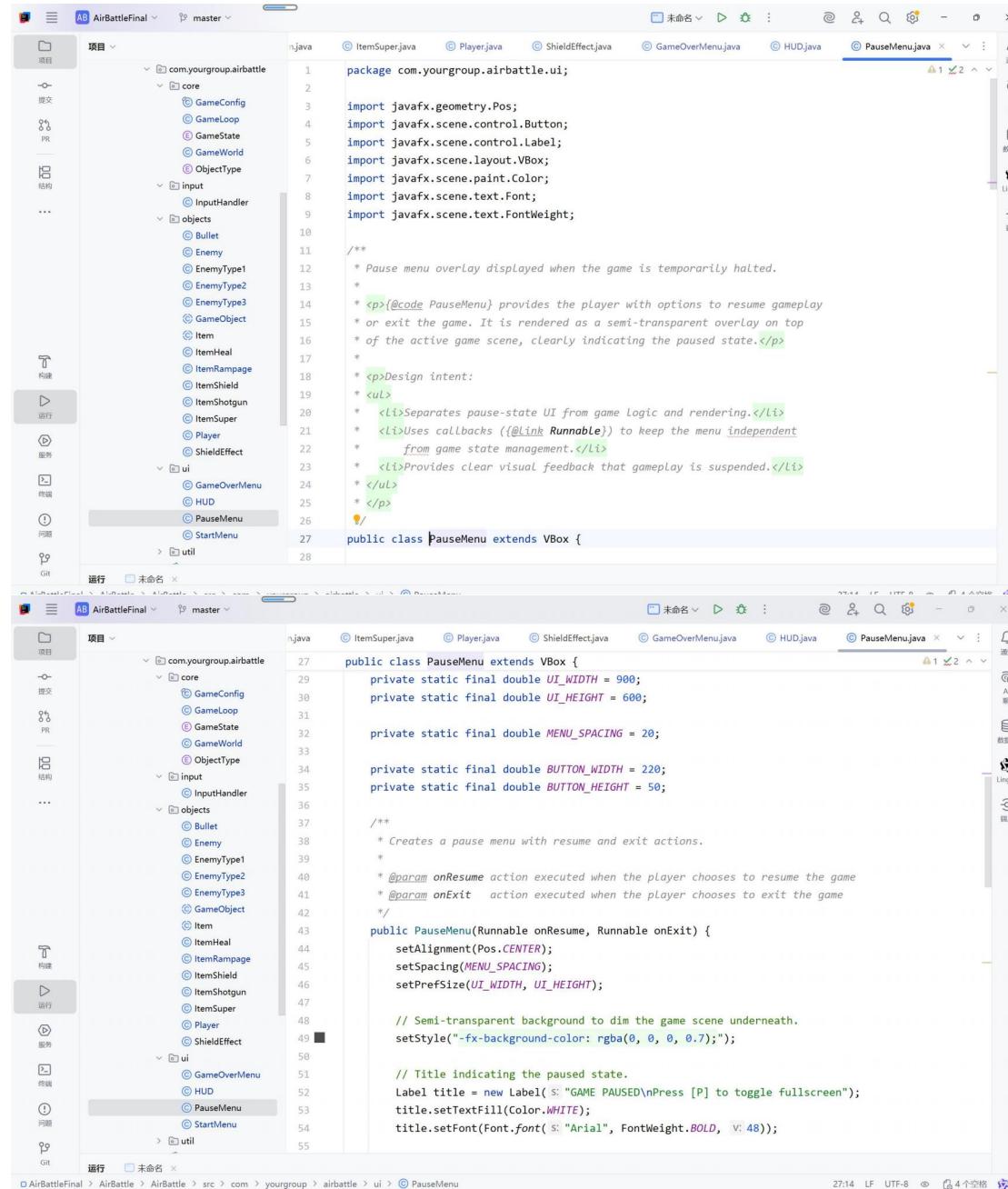
```
public class HUD extends VBox {  
    public int getScore() { return currentScore; }  
  
    /**  
     * Resets the HUD to its initial state.  
     *  
     * <p>Used when restarting the game.</p>  
     */  
    public void reset() {  
        setHp(DEFAULT_HP);  
        setScore(0);  
    }  
  
    /**  
     * 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 -> {  
                scoreLabel.setScaleX(1.0);  
            })  
        ).play();  
    }  
}
```

Editor 2 (Right):

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

The main difference between the two versions is in line 159 of the `animateScoreLabel` method. In the left editor, the code is `scoreLabel.setScaleY(1.2);`. In the right editor, it is changed to `scoreLabel.setScaleY(1.0);` and also includes a call to `scoreLabel.setTextFill(Color.WHITE);` before the final `scoreLabel.setScaleX(1.0);` call.

\PauseMenu.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:<br/>
 * <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 {

```



```

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;

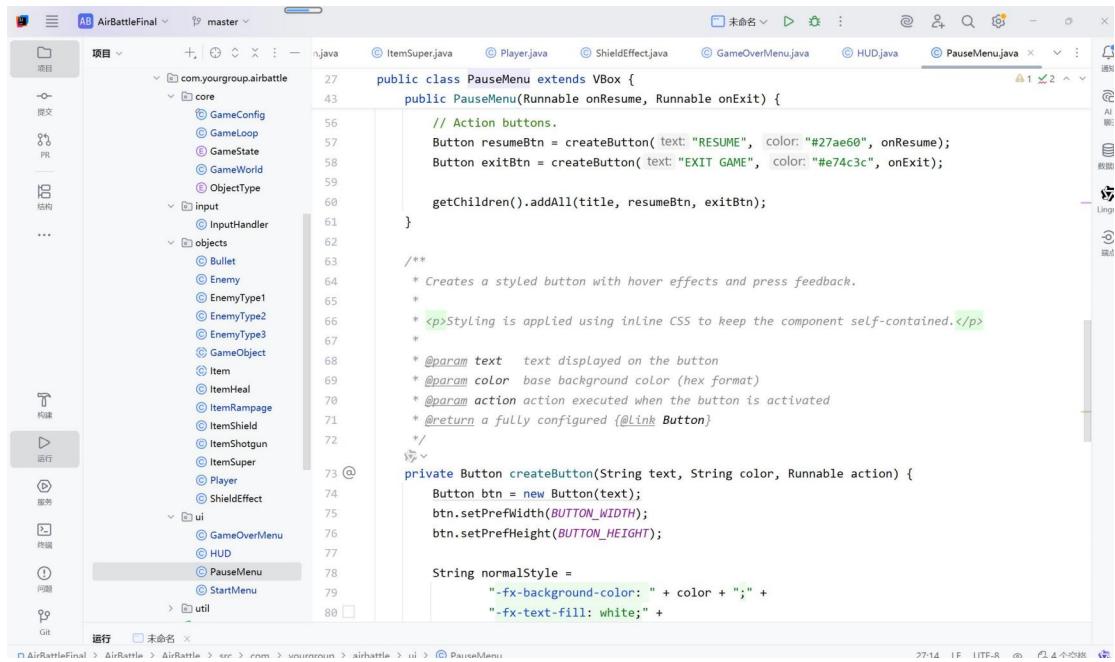
    /**
     * Creates a pause menu with resume and exit actions.
     *
     * @param onResume action executed when the player chooses to resume the game
     * @param onExit   action executed when the player chooses to exit the game
     */
    public PauseMenu(Runnable onResume, Runnable onExit) {
        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("GAME PAUSED\nPress [P] to toggle fullscreen");
        title.setTextFill(Color.WHITE);
        title.setFont(Font.font("Arial", FontWeight.BOLD, 48));
    }
}

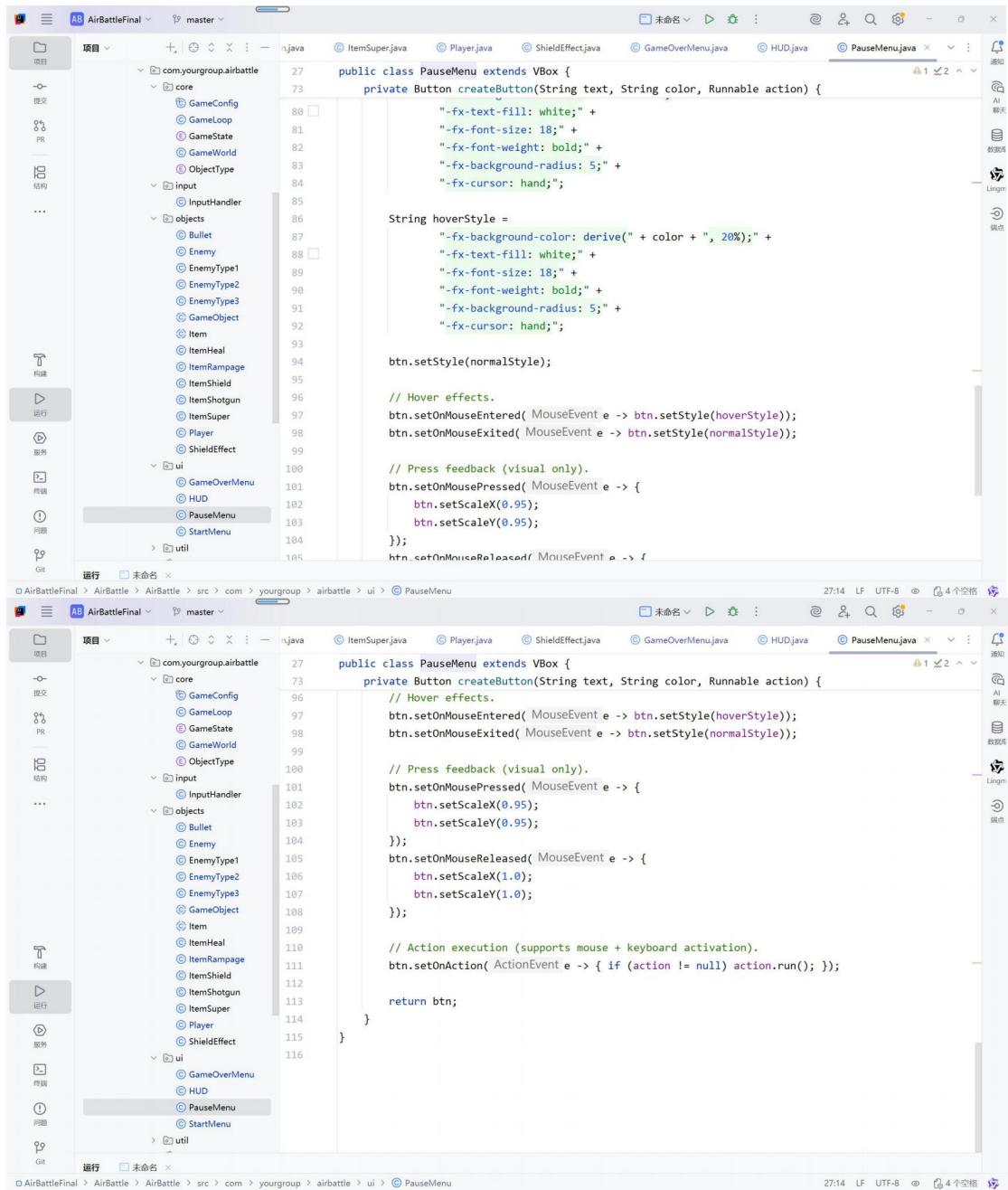
```

XIAMEN UNIVERSITY MALAYSIA



```
public class PauseMenu extends VBox {  
    public PauseMenu(Runnable onResume, Runnable onExit) {  
        // Action buttons.  
        Button resumeBtn = createButton("RESUME", "#27ae60", onResume);  
        Button exitBtn = createButton("EXIT GAME", "#e74c3c", onExit);  
  
        getChildren().addAll(title, resumeBtn, exitBtn);  
    }  
  
    /**  
     * Creates a styled button with hover effects and press feedback.  
     *  
     * <p>Styling is applied using inline CSS to keep the component self-contained.</p>  
     *  
     * @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;" +  
    }  
}
```

XIAMEN UNIVERSITY MALAYSIA



The image shows two side-by-side Java code editors, likely from an IDE like IntelliJ IDEA. Both editors display the same code for a class named `PauseMenu`. The code is a JavaFX-style class that extends `VBox` and contains methods for creating buttons and handling mouse events. The code is identical in both instances.

```
public class PauseMenu extends VBox {
    private Button createButton(String text, String color, Runnable action) {
        String normalStyle =
            "-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

```

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:</p>
 * <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.
     */
}

```



```

public class StartMenu extends VBox {
    * @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);

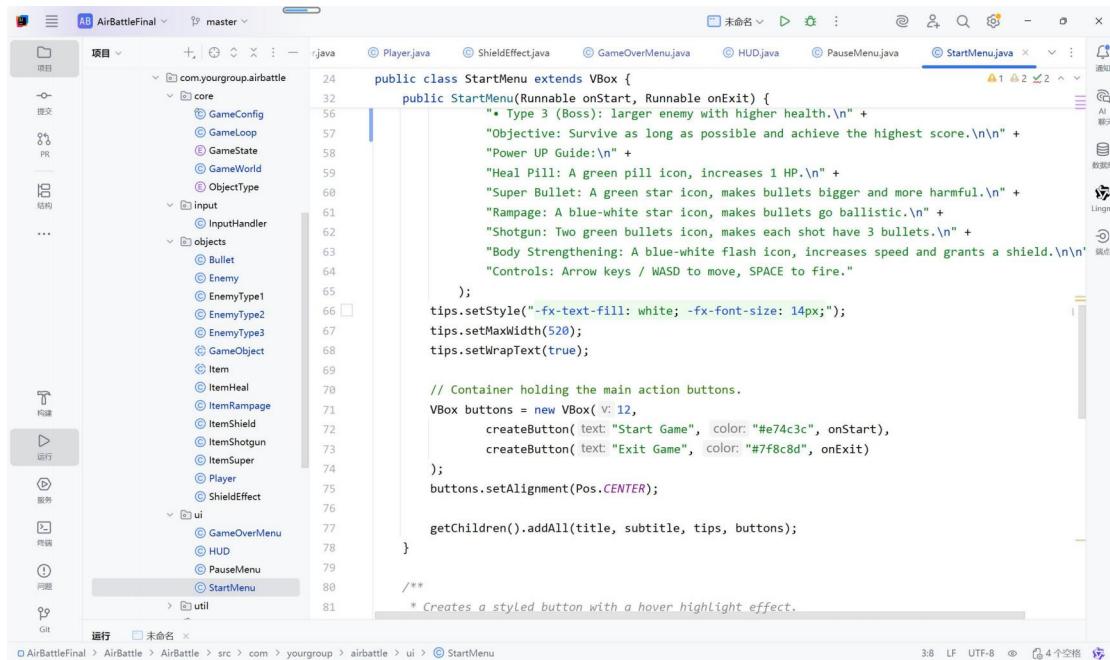
        // Background styling for the start menu screen.
        setStyle(""-fx-background-color: #0f3460;
                 -fx-padding: 40;
                """);

        // Main game title.
        Label title = new Label("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("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(
            "ENEMY GUIDE:\n" +
            "* Type 1 (Normal): standard enemy with basic movement.\n" +
            "* Type 2 (Fast): fast enemy that is harder to track\n"
        );
    }
}

```



The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `StartMenu.java` file, which extends `VBox`. The code defines a `StartMenu` class with methods for handling game startup and exit. It includes comments explaining various game mechanics like enemy types, power-ups, and controls.

```

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 bullet 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.
 */

```

XIAMEN UNIVERSITY MALAYSIA

```

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);

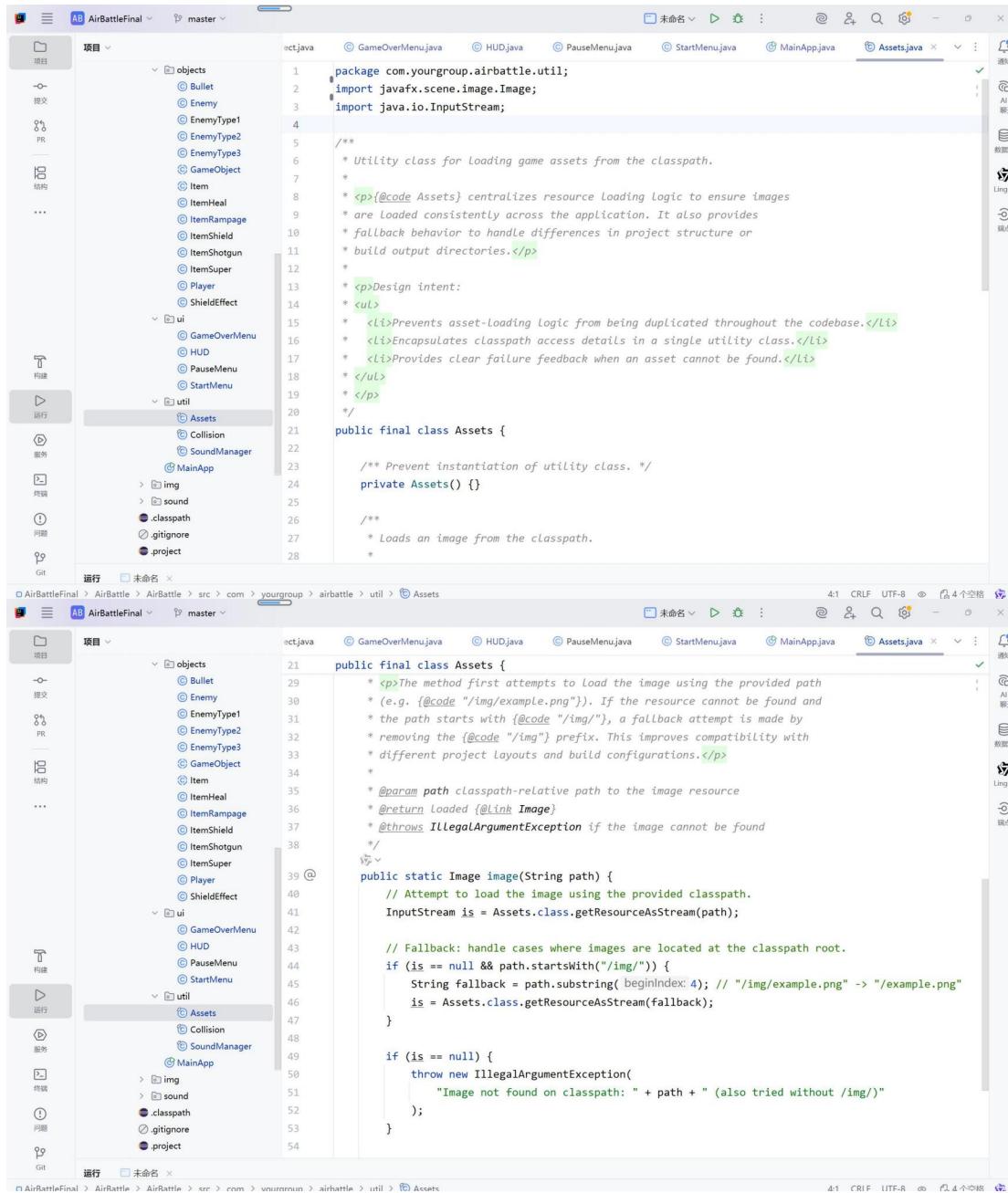
        String normalStyle =
            "-fx-background-color: " + color + ";" +
            "-fx-text-fill: white;" +
            "-fx-font-size: 16px;" +
            "-fx-font-weight: bold;" +
            "-fx-background-radius: 6;" +
            "-fx-cursor: hand;";

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

```

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

\Assets.java

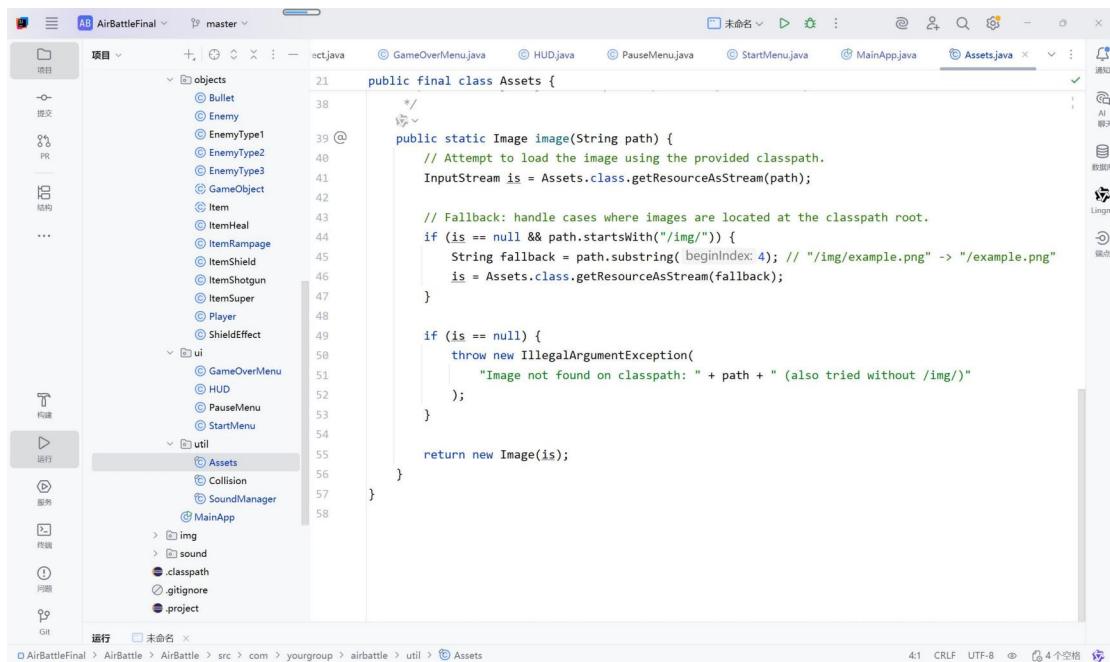


```

1 package com.yourgroup.airbattle.util;
2 import javafx.scene.image.Image;
3 import java.io.InputStream;
4
5 /**
6  * Utility class for Loading game assets from the classpath.
7  *
8  * <p>{@code Assets} centralizes resource Loading logic to ensure images
9  * are Loaded consistently across the application. It also provides
10 * fallback behavior to handle differences in project structure or
11 * build output directories.</p>
12 *
13 * <p>Design intent:</p>
14 * <ul>
15 *   <li>Prevents asset-loading logic from being duplicated throughout the codebase.</li>
16 *   <li>Encapsulates classpath access details in a single utility class.</li>
17 *   <li>Provides clear failure feedback when an asset cannot be found.</li>
18 * </ul>
19 * </p>
20 */
21 public final class Assets {
22
23     /** Prevent instantiation of utility class. */
24     private Assets() {}
25
26     /**
27      * Loads an image from the classpath.
28      */
29
30     /**
31      * The method first attempts to Load the image using the provided path
32      * (e.g. {@code "/img/example.png"}). If the resource cannot be found and
33      * the path starts with {@code "/img/"}, a fallback attempt is made by
34      * removing the {@code "/img/"} prefix. This improves compatibility with
35      * different project layouts and build configurations.</p>
36
37     * @param path classpath-relative path to the image resource
38     * @return Loaded {@Link Image}
39     * @throws IllegalArgumentException if the image cannot be found
40     */
41     public static Image image(String path) {
42         // Attempt to load the image using the provided classpath.
43         InputStream is = Assets.class.getResourceAsStream(path);
44
45         // Fallback: handle cases where images are located at the classpath root.
46         if (is == null && path.startsWith("/img/")) {
47             String fallback = path.substring(beginIndex: 4); // "/img/example.png" -> "example.png"
48             is = Assets.class.getResourceAsStream(fallback);
49         }
50
51         if (is == null) {
52             throw new IllegalArgumentException(
53                 "Image not found on classpath: " + path + " (also tried without /img/)"
54             );
55         }
56     }
57 }

```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows a Java code editor within an IDE. The project navigation bar at the top indicates the current file is Assets.java. The code itself is a static method named 'image' that takes a string path as an argument. It attempts to load the image from the classpath using `Assets.class.getResourceAsStream(path)`. If the path starts with '/img/' or '/sound/' and the file is not found, it uses a fallback path starting with '/img/' or '/sound/'. If the provided path is null, it throws an `IllegalArgumentException` with a descriptive message. Finally, it returns a new `Image` object.

```
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);
}
```

\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:</p>
 * <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>
    */
}

```



```

public final class 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

```

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:</p>
 * <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;
}

public final class SoundManager {
    /** 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");
        }
    }
}

```

XIAMEN UNIVERSITY MALAYSIA

```

public final class SoundManager {
    ...
    public final void playItemGet() {
        ...
        itemGetSound = Load( path: "/sound/GetItem.mp3" );
        ...
    }
}

private static void playStartBgm() {
    playBgmInternal(startBgm);
}

```

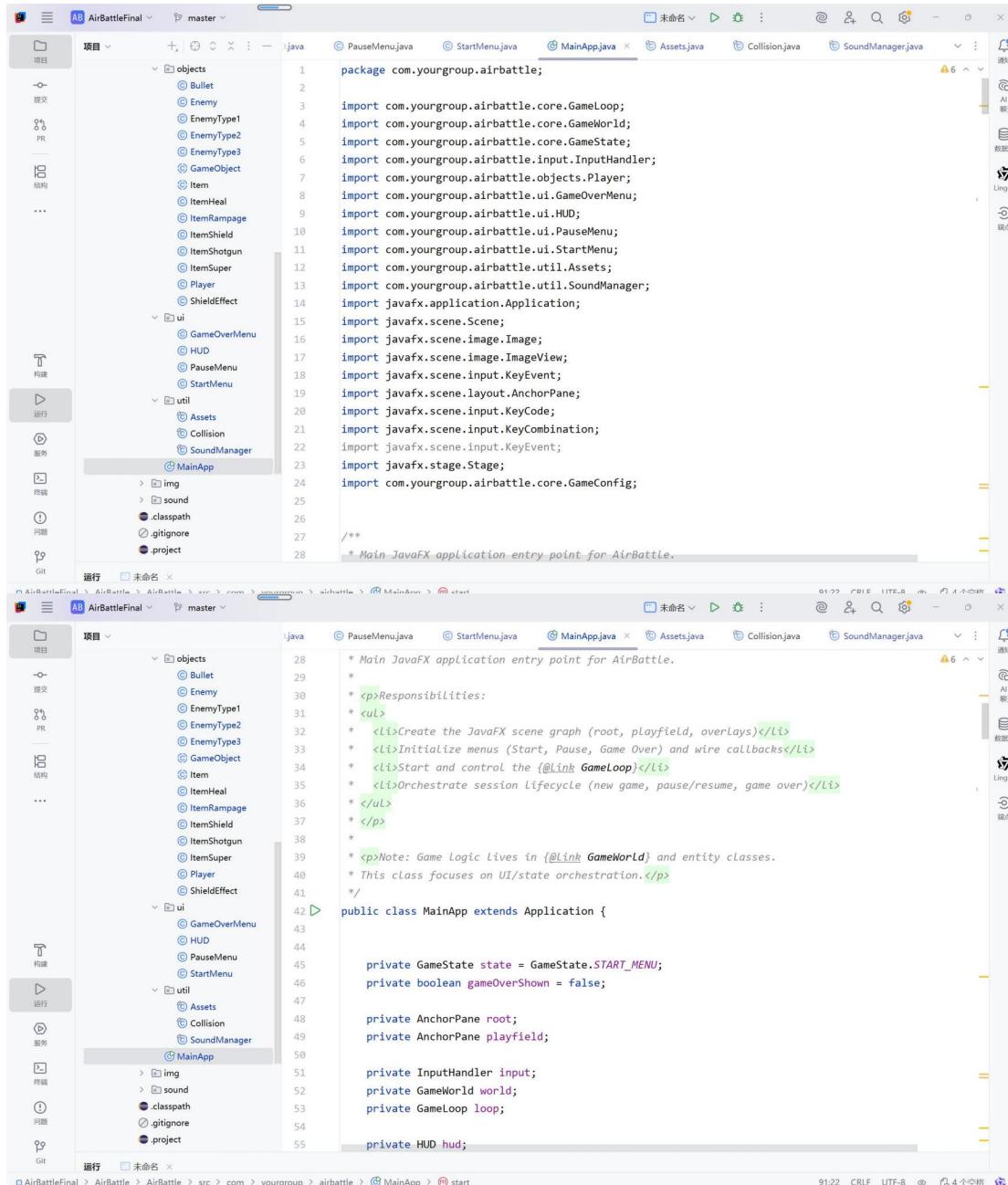
XIAMEN UNIVERSITY MALAYSIA

```
public final class SoundManager {
    /**
     * 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.
*
* Responsibilities:
*   - Create the JavaFX scene graph (root, playfield, overlays)
*   - Initialize menus (Start, Pause, Game Over) and wire callbacks
*   - Start and control the GameLoop
*   - Orchestrates session lifecycle (new game, pause/resume, game over)
*
* Note: Game Logic Lives in GameWorld and entity classes.
* This class focuses on UI/state orchestration.
*/

```

```

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;
}

```

XIAMEN UNIVERSITY MALAYSIA

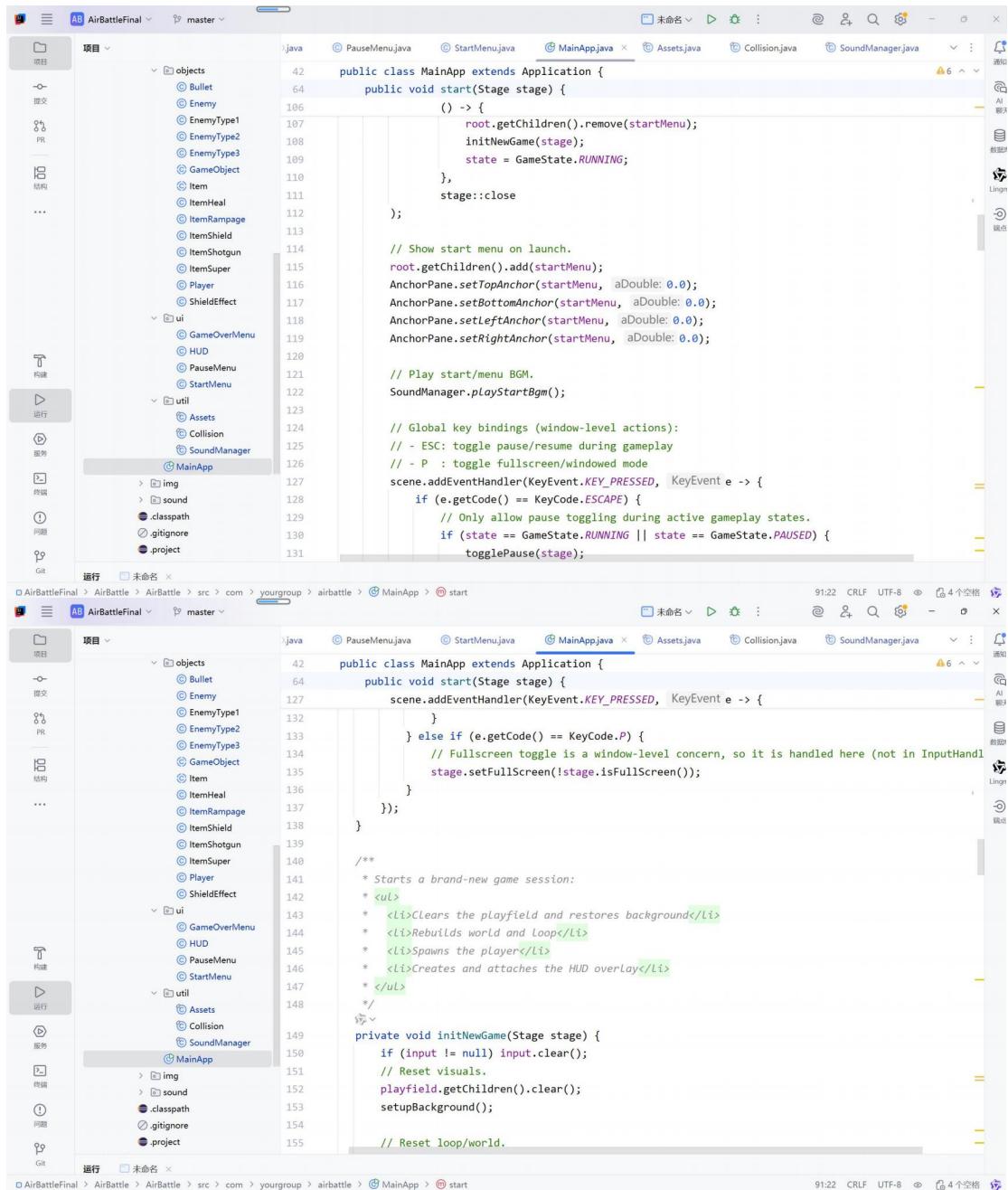
```
public class MainApp extends Application {
    private Player player;
    private PauseMenu pauseMenu;
    private StartMenu startMenu;
    private ImageView bgView;

    @Override
    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();
    }
}
```

XIAMEN UNIVERSITY MALAYSIA



```
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);
            }
        } else if (e.getCode() == KeyCode.P) {
            // Fullscreen toggle is a window-level concern, so it is handled here (not in InputHandler)
            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.
}
```

XIAMEN UNIVERSITY MALAYSIA

```

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;
    }
}

/**
 * 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
    };
}

```

XIAMEN UNIVERSITY MALAYSIA

```

    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.
         *
         * <p>State rules:
         * <ul>
         *   <li>Only RUNNING -> PAUSED is allowed.</li>
         *   <li>Pause/resume is ignored during START_MENU and after GAME_OVER.</li>
         * </ul>
         *
         * <p>UI rules:
         * <ul>
         *   <li>When paused: show PauseMenu overlay and hide HUD.</li>
         *   <li>When resumed: remove PauseMenu overlay and show HUD.</li>
         * </ul>
         */
    }
}

```



```

    public class MainApp extends Application {
        * <p>Note: {@code stage} is kept for signature consistency (and future use),  

        * but is not required for pause/resume logic.</p>
        */

        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);
            }
        }
}

```

XIAMEN UNIVERSITY MALAYSIA

```

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.
     */
    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);

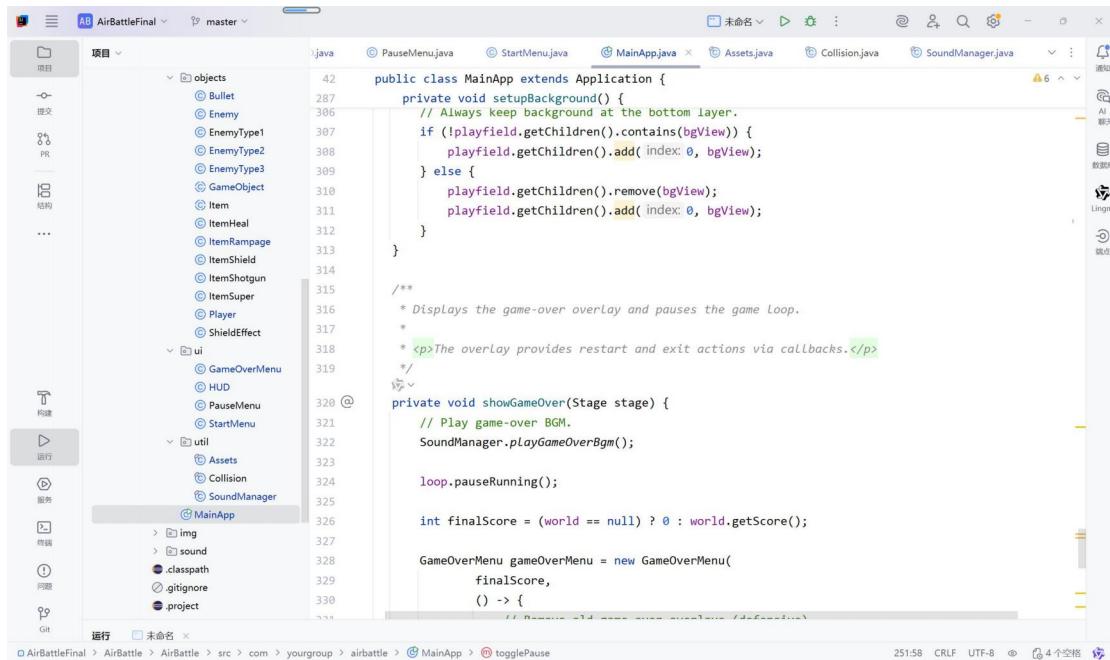
            // Bind background size to playfield size for smooth resizing/fullscreen.
            bgView.fitWidthProperty().bind(playfield.widthProperty());
            bgView.fitHeightProperty().bind(playfield.heightProperty());
        }

        bgView.setImage(bg);

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

```

XIAMEN UNIVERSITY MALAYSIA



The screenshot shows a Java code editor within an IDE. The project structure on the left includes packages like objects, ui, and util, along with files such as Assets.java, Collision.java, SoundManager.java, and MainApp.java. The MainApp.java file is open, displaying code related to game setup and background handling. The code uses annotations like @Override and @FXML. The status bar at the bottom shows the file path as AirBattleFinal > AirBattle > AirBattle > src > com > yourgroup > airbattle > MainApp.java, and the current time as 251:58.

```
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.
     *
     * <p>The overlay provides restart and exit actions via callbacks.</p>
     */
    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,
            () -> {
                ...
            }
        );
    }
}
```

XIAMEN UNIVERSITY MALAYSIA

```
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);
}

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.