**Compliance Document for Frogger Refactor Coursework**

**Task 1: Running the Application**

**Overview**

The application has been successfully configured and compiled using Gradle. All source files have been structured correctly, and compilation errors have been resolved.

**Key Evidence**

1. **Gradle Configuration:**
   * gradlew, gradlew.bat, and gradle-wrapper.properties files are present.
   * distributionUrl points to a valid Gradle version (gradle-8.10-bin.zip).
2. **Source File Organization:**
   * Files are organized into appropriate packages under src/main/java/com/frogger.
3. **Build and Run Verification:**
   * The project builds successfully without errors.
   * The game launches correctly using the Main.java class.

**Task 2: Creating Tests for Key Features**

**Overview**

Comprehensive unit tests have been created for the key features of the game, focusing on the Player, Obstacle, and GameModel classes.

**Key Evidence**

1. **Test Cases for Player Class:**
   * Validates movement (moveUp, moveDown, moveLeft, moveRight).
   * Tests boundary conditions and life management (loseLife, gainLife).
2. **Test Cases for Obstacle Class:**
   * Ensures obstacles move correctly and detect collisions with the player.
3. **Test Cases for GameModel Class:**
   * Validates level progression, score updates, and game-over conditions.
   * Ensures power-ups are spawned and collected correctly.

**Example Evidence of Test-Driven Development**

* **Failing Test (Initial):** Created a test to validate power-up collection, which initially failed due to a missing implementation.
* **Implementation and Passing Test:** Implemented the collidesWith method in the PowerUp class. The test passed after implementation.

**Task 3: High- and Low-Level Refactoring**

**High-Level Refactoring**

1. **Code Organization:**
   * Restructured into meaningful packages: controller, model, and view.
   * Adheres to the MVC architecture.
2. **Design Patterns Applied:**
   * Factory pattern for spawning power-ups.
   * Separation of concerns for improved maintainability.
3. **SOLID Principles:**
   * Single Responsibility: Each class handles a specific responsibility (e.g., Player, Obstacle).
   * Open-Closed: New features like power-ups are added without modifying existing code.

**Low-Level Refactoring**

1. **Code Smell Elimination:**
   * Removed hardcoded values; introduced constants (e.g., grid dimensions).
   * Simplified loops and conditions.
2. **Coding Conventions:**
   * Followed Java naming conventions and proper formatting.
   * Added comments and Javadoc for major classes and methods.
3. **Test Refactoring:**
   * Ensured tests are modular and independent.
   * Removed redundant test setups.

**Task 4: Additional Features**

**Feature Added: Power-Ups**

1. **Description:**
   * Introduced collectible power-ups that grant extra lives.
   * Power-ups are randomly spawned and appear on the grid.
2. **Implementation:**
   * Created a PowerUp class with position and collision detection logic.
   * Modified GameModel to handle power-up spawning and collection.
   * Updated GameView to display power-up locations.
3. **Justification:**
   * Adds complexity and replayability to the game.
   * Aligns with the classic Frogger gameplay style.

**Evidence:**

* **Code Snippet:**

if (powerUp != null && powerUp.collidesWith(player)) {

player.gainLife();

System.out.println("Power-Up Collected! Extra life gained.");

powerUp = null;

}

* **Screenshot:** Displays player collecting a power-up and gaining an extra life.

**Task 5: Development Tools Usage**

**Gradle**

* Successfully configured to build the project.
* Ensures dependency management and compatibility.

**Git**

* Repository structured with feature branches.
* Meaningful commit messages, e.g., Added power-up feature.
* Evidence of merging and resolving conflicts.

**Task 6: Documentation**

**README File**

1. **Project Overview:**
   * Brief description of the game and its objectives.
   * Instructions on how to play and controls.
2. **Developer Notes:**
   * High-level architecture description.
   * Steps to build and run the project using Gradle.
3. **Credits:**
   * Acknowledges contributors and references.

**Javadoc Comments**

* Added to all major classes and methods.
* Example:

/\*\*

\* Represents the player character in the Frogger game.

\* Handles movement and life management.

\*/

public class Player {

// Class implementation

}

**Summary**

The project complies with the coursework requirements. It demonstrates effective use of testing, refactoring, and feature implementation, ensuring maintainability and quality.