AngryBirds

AngryBirds is a 2D physics-based game inspired by the iconic *Angry Birds*. Players use their slingshot skills to launch birds and destroy the mischievous pigs hiding behind structures. The game challenges your aim and strategy as you tackle each level with unique bird abilities, randomized levels, and structures that require careful planning to overcome.

Backstory

Once upon a time, a peaceful bird kingdom was disrupted by cunning pigs who stole their precious eggs. Driven by anger and a quest for justice, the birds decided to fight back! Armed with unique abilities and a trusty slingshot, the birds take on the pigs, destroying their defenses one structure at a time. Will you help the birds reclaim their eggs and restore peace to their kingdom?

Technologies Used

- **LibGDX**: For rendering and game framework.
- **Box2D**: To simulate realistic physics and collisions.
- **Java**: Core language for development.

How to Run

1. **Dependencies**:
   Ensure you have the following installed:
   - **Java Development Kit (JDK) 21** or later.
   - **LibGDX library** and **JUnit testing framework**.
2. **Building the Project**:
   - Clone the repository and open it in an IDE like IntelliJ IDEA or Eclipse.
   - Use **Gradle** to build the project.
     `./gradlew build`
   - Run the main class to start the game.

Gameplay Overview

- **Enter the Level**: Start the level with a slingshot loaded with birds.
- **Launch Birds**: Drag and aim to set the trajectory, then release to launch birds.
- **Destroy Structures**: Strategically target structures made of wood, glass, or other materials.
- **Defeat Pigs**: Reduce the pigs' health through collisions and eliminate them to clear the level.
- **Retry or Progress**: If pigs survive, retry the level; otherwise, advance to the next one.
- **Special Abilities**: Activate unique bird abilities with the spacebar.
- **Save and Load Game**: Progress can be saved and loaded to resume later.

Key Features

1. **Level Design**: Handcrafted levels with increasing complexity.
2. **Randomized Levels**: Option to generate random levels for a fresh challenge.
3. **Realistic Physics**: Structures respond dynamically to collisions based on material and force.
4. **Unique Bird Abilities**: Each bird has special skills activated by pressing the spacebar.
5. **Health System**: Structures and pigs have health points that reduce upon impact.
6. **Save/Load Game**: Continue your adventure from where you left off.
7. **Retry and Next Level**: Flexible options to retry a failed level or move forward after success.

Project Structure

- **Screens Package**:
  Manages various screens, including menus, settings, and the play screen.
  - **PlayScreen**: Displays the current level and gameplay interface.
- **Sprites Package**:
  Core gameplay elements like birds, pigs, and structural components. These use Box2D physics to handle interactions and collisions.
- **LevelCreator**:
  Handles level design, rendering components, and generating randomized levels.

JUnit Testing

SlingTest

1. `testSingletonInstance`: Verifies the singleton behavior of the `Slingshot` class.
2. `testStartDrag`: Tests the drag functionality of the slingshot.
3. `testRelease`: Validates the release mechanism and position reset.
4. `testDispose`: Ensures the `dispose` method executes without errors.

BirdTest

1. `testInitialization`: Tests the initialization of the `Bird` object.
2. `testSpecialFeature`: Verifies the activation and usage of the special feature.
3. `testGetHealth`: Validates the default health value.
4. `testDestroySelf`: Verifies the destruction behavior.
5. `testGetMass`: Checks the mass of the bird body.
6. `testGetDensity`: Validates the density of the bird body.

BoxTest

1. `testInitialization`: Tests the initialization of the `Box` object.
2. `testGetHealth`: Validates the default health value.
3. `testDestroySelf`: Verifies the destruction behavior.
4. `testGetMass`: Checks the mass of the box body.
5. `testGetDensity`: Validates the density of the box body.

PigTest

1. `testInitialization`: Tests the initialization of the `Pig` object.
2. `testGetHealth`: Validates the default health value.
3. `testDestroySelf`: Verifies the destruction behavior.

4. `testGetMass`: Checks the mass of the pig body.
5. `testGetDensity`: Validates the density of the pig body.

Design Patterns

1. Singleton Pattern

- **Class Involved**: `Slingshot`
- **Description**: The **Singleton Pattern** is used to ensure that only one instance of the `Slingshot` class exists throughout the lifetime of the game. This is important because we want to maintain a single point of interaction for the slingshot mechanism during the gameplay.

  The singleton pattern ensures that the `Slingshot` instance is created lazily and only once. Subsequent calls to the `getInstance()` method will return the same instance.

- **Benefits**:
  - Ensures global access to the slingshot instance.
  - Prevents multiple instances from being created, thus saving resources.

2. Factory Method Pattern

- **Classes Involved**: `Pig`, `Box`, `Bird`
- **Description**: The **Factory Method Pattern** is used to create objects without specifying the exact class of the object that will be created. This pattern allows for more flexibility, as subclasses can modify or extend the creation process.

  In this project, the creation of different game entities (such as `Pig`, `Box`, `Bird`) is done via constructor methods, which are treated as a form of the factory method. This allows you to extend the system with new types of entities easily, without modifying existing code.

- **Benefits**:
  - Enables creating new types of entities in a flexible manner.
  - Decouples the object creation process from the main gameplay logic, promoting clean and maintainable code.
  - Facilitates easy extension for new types of entities in the future (e.g., different types of pigs or birds).

Screenshots

(Include screenshots of gameplay and menu screens here.)

Future Improvements

- **Additional Levels**: Introduce more levels with diverse challenges.
- **Star Rating System**: Earn stars based on your score or performance.
- **Power-Ups**: Add wildcards, a power sling, and other features to improve trajectory and gameplay.
- **Advanced Graphics**: Enhance visuals for a more immersive experience.

Credits

- **LibGDX Documentation**: For invaluable guidance on game development.
- **Box2D Tutorials**: For teaching physics-based interactions.
- **Brent Aureli's Tutorials**: Inspiration and practical insights into using LibGDX.

Help the birds reclaim their kingdom! ""