Magical Arena Game

Overview

This project implements a simple "Magical Arena" game where two players fight until one of them dies. Each player is characterized by health, strength, and attack attributes. Players take turns attacking and defending until one player's health reaches zero.

Project Structure

The project contains the following key classes:

- Player: Represents a player in the game with attributes for health, strength, and attack.
- Dice: Represents a six-sided dice used for generating random attack and defense values.
- Arena: Represents the arena where players fight. It manages the fight logic and determines the winner.
- Game: The main class to run the game.
- GameTest: Contains unit tests for the game logic.

Setup

Prerequisites

- Java Development Kit (JDK) 8 or later
- Maven (for running tests)

Running the Game

1. Clone the repository or download the source code.
2. Navigate to the src/main/java directory in your terminal.
3. Compile the Java files:

```
javac *.java
```

4. Run the main class to start the game:

```
java Game
```

This will initiate a fight between two predefined players, "Player A" and "Player B", and print the fight progress and the winner to the console.

Running the Tests

- Make sure you have Maven installed. You can check by running:

```
mvn -version
```

- Navigate to the root directory of the project.
- Run the tests using Maven:

```
mvn test
```

1. This will run the unit tests defined in GameTest.java and output the results to the console.

Project Notes

- The Dice class is used to simulate dice rolls for both attack and defense. It generates random numbers between 1 and 6.
- The Arena class contains the main logic for the fight. It manages the turns and health reductions based on the dice rolls and player attributes.
- The GameTest class contains unit tests to verify the functionality of the Player, Dice, and Arena classes.

Design Considerations

- Simple Design: The code is designed to be simple and easy to understand. Each class has a clear responsibility.
- Readability: The code is well-organized with clear naming conventions.
- Modelling: The objects and classes used in the code are well-designed and appropriate for the problem at hand.
- Maintainability: The code is easy to maintain and modify. Potential areas of concern or technical debt are minimal.
- Testability: Comprehensive unit tests are provided to ensure the code works as expected. The code has a high degree of test coverage.

Future Improvements

- Add more players and allow dynamic player creation.
- Implement additional game rules or power-ups.
- Create a user interface for a more interactive experience.
- Expand unit tests to cover more edge cases and scenarios.

License