Project Title: Interactive Fighter Battle Game

Objective:
To create an interactive fighter battle game using JavaScript, HTML, and CSS. The project will teach students how to apply object-oriented programming (OOP) concepts alongside DOM manipulation, animations, and event handling. The game will feature two characters, or "fighters," who engage in a turn-based battle with visual feedback for their actions.

Project Description:
The Fighter Battle Game will transition the provided script into a fully interactive web application. Players will see two fighters visually represented on the screen. Each fighter will have distinct images for different states (idle, attacking, and taking damage). The game will dynamically update the screen as the fighters attack each other, displaying health changes and animations. The match ends when one fighter's health reaches zero, declaring the winner.

To make the game more personalized and engaging, students will first take pictures of themselves on a white background, remove the background, and use these images as their fighter characters. The characters will be categorized into one of five predefined categories, each with unique abilities that influence gameplay.

Character Categories and Special Abilities:

1. **Warrior:** Has a chance to deal double damage during an attack.
2. **Mage:** Can heal themselves for an amount equal to their attack damage.
3. **Demon:** Has a chance to make the opponent skip a turn.
4. **Archer:** Gains a chance to perform a second attack immediately after the first.
5. **Tank:** Reduces incoming damage by 50%.

Learning Outcomes:

1. Understand how to integrate OOP principles with DOM manipulation.
2. Learn to implement animations using CSS.
3. Use JavaScript event listeners for interactivity.
4. Practice dynamically updating HTML elements.
5. Develop debugging and testing skills in a browser environment.

Steps to Complete the Project:

1. Prepare Character Images

- Take pictures of yourself or a friend in a neutral pose against a white background.
- Use a background removal tool (e.g., Remove.bg or Photoshop) to isolate the character.
- Save images for three states as gifs for animations: idle, attacking, and damaged.

2. Categorize Characters

- Decide which category each character will belong to (e.g., Warrior, Mage, etc.).
- Document their special abilities and how they will affect gameplay.

3. Set Up the Project Structure

- Create a new project folder.
- Inside the folder, create the following files:
    - `index.html` : For the webpage structure.
    - `styles.css` : For styling the game interface.
    - `logic.js` : For the game classes logic.
    - `script.js` : For the game execution logic and DOM manipulation.

4. Design the HTML Structure

- Design a layout that includes:
    - A container to house the game elements.
    - Two fighter areas, each displaying a fighter's image and health.
    - A "Start Match" button to initiate the game.
- Consider positioning the fighters to face each other on the screen.

5. Style the Game Using CSS

- Use CSS to:
    - Position the fighters and the game elements visually.
    - Style the health display for each fighter.
    - Prepare for animations by defining transitions for movement and state changes.

6. Extend the Fighter Class

- Create a **`Fighter`** class with:
    - Properties for name, health, attacks, and a renderer to handle DOM updates.
    - A method **`getHit(power)`** to decrease health.

  - A method `attack(opponent)` to select a random attack and damage the opponent.
  - A method `isAlive()` to check if the fighter is still alive.

7. Define the Attack Class

- Create an `Attack` class with:
  - Properties for id, name, power, and optional side effects.
  - Methods to manage and display attack properties if needed.

8. Implement the Rendering Class

- Create a `Rendering` class to:
  - Update the DOM elements, including fighter health and state images.
  - Handle transitions between idle, attack, and damage states using CSS classes or image changes.
  - Include methods such as `renderIdle()`, `renderAttack()`, and `renderDamage()`.

9. Create the FighterElement Class

- Create a `FighterElement` class to:
  - Link the HTML elements (fighter container, image, health display) to the JavaScript objects.
  - Ensure each fighter's DOM elements are updated in sync with their properties.

10. Implement Special Abilities

- Enhance fighter subclasses (e.g., `Warrior`, `Mage`) to:
  - Override methods like `attack(opponent)` or `getHit(power)` to implement unique abilities.
  - Example: The **Mage** class can include healing logic in the `attack()` method.

11. Build the Referee Class

- Create a `Referee` class to:
  - Manage the game loop and alternate turns between fighters.
  - Use `setInterval` to automate the match.
  - Declare a winner when one fighter's health reaches zero.

12. Add Animations

- Define CSS animations for:
  - Fighter movements during attacks.
  - State changes (e.g., switching to attack or damage poses).
- Trigger animations during the game using JavaScript.

13. Test and Debug

- Test the game to ensure:
  - Health updates correctly.
  - Animations trigger at the right times.
  - The game loop runs and stops as expected.
- Debug any issues with the game logic, DOM updates, or animations.

14. Optional Enhancements

- Add a character selection screen where players can choose their fighter.
- Include sound effects for attacks and damage.
- Display attack names and damage amounts on the screen.
- Add a "Restart Match" button to replay the game.

Submission Guidelines:

- Submit a zipped folder containing all project files.
- Ensure the game is playable by opening the `index.html` file in a browser.
- Provide a short README file explaining your implementation and any additional features.

Evaluation Criteria:

1. Correct implementation of OOP principles.
2. Proper use of DOM manipulation and animations.
3. Functionality and interactivity of the game.
4. Clean and readable code structure.
5. Creativity in optional enhancements.