Component Code Report

Component: Additional Features for Tile Breaker Game

Overview

The tile-breaker game, developed in C++, features a paddle controlled by the user, a ball that bounces off surfaces, and a grid of tiles at the top of the screen. This report details the additional components added to enhance gameplay, which include lives management, score and level handling, and various power-up functionalities.

The game leverages the FsSimpleWindow library for rendering and the YsGlFontBitmap library for text rendering, providing an engaging user experience. The additional components are responsible for managing game difficulty, displaying lives, and applying temporary gameplay changes, such as increasing the paddle width or modifying the ball speed.

Features and Implementation

1. Lives Management

- **Description**: The player starts with a default of 2 lives. Each time the ball falls below the paddle, one life is deducted. When all lives are lost, the game ends with a "Game Over" message.
- **Implementation**: The check_ball_status() function reduces the lives variable each time the ball falls. If the lives count reaches zero, the end_game() function is called to terminate the game.

2. Score and Level Management

- **Description**: The game tracks the player's score, incrementing it for each tile hit. When the score reaches the total number of tiles in the current level, the game progresses to the next level, with an increase in ball speed and a decrease in paddle width.
- **Implementation**: The update_score_level() function updates the score and checks if the player has cleared all tiles. Upon clearing all tiles, the level_up() function is called to adjust game parameters and increase the level.

3. Power-Ups

- **Description**: The game includes several power-ups, such as extra lives, increased paddle width, and temporary changes in ball speed. These power-ups are intended to make the game more dynamic and engaging.
- **Implementation**: The apply_power_up() function handles different power-ups based on a string identifier. Each power-up modifies relevant variables, such as lives, paddle_width, or ball_speed. The check_power_up_timer() function ensures temporary power-ups revert after their duration ends.

4. Debugging and Testing Components

- **Debugging**: The debug_and_test_components() function allows for manual testing of each game component by printing the current state of lives, score, level, paddle width, and ball speed. This is useful for verifying the correctness of each component's behavior in various scenarios.
- **Testing**: Debugging involves incrementing the score, simulating ball falls, and applying power-ups to validate the logic. For example, the number of lives is checked after each ball fall, and the ball speed is verified to increase when an extra life is added.

Testing Methodology

1. Lives and Ball Speed Testing

- **Goal**: Verify that lives decrement correctly when the ball falls and that ball speed increases with additional lives.
- **Method**: Lives were tested by simulating ball falls using check_ball_status(true) and confirming the decrease in lives. The ball speed was verified to increase each time an extra life was applied using apply_power_up("extra_life").

2. Score and Level Management Testing

- **Goal**: Ensure that the score increments correctly when tiles are hit and that the level progresses after clearing all tiles.
- **Method**: The score was incremented by simulating multiple tile hits using update_score_level(total_tiles). The level-up mechanism was verified by checking changes in ball speed and paddle width.

3. Power-Up Testing

- **Goal**: Validate the correct application and expiration of power-ups, such as paddle width increase and ball speed modification.
- **Method**: Each power-up was applied, and its effect was observed in the terminal output. For example, apply_power_up("increase_paddle_width") was used to temporarily increase paddle width, and check_power_up_timer() ensured the effect reverted after 5 seconds.

4. Level-Up Mechanism Testing

- **Goal**: Ensure that the level-up mechanism functions as intended, with increased difficulty after each level.
- **Method**: The update_score_level() function was used to reach the required score for level-up. The terminal output was checked to verify changes in the level, ball speed, and paddle width.

Conclusion

The additional components of the tile-breaker game, including lives management, score and level handling, and power-ups, were successfully implemented and tested. The debugging

functionality integrated into the game provides real-time feedback for each component, ensuring correct functionality and allowing for easy testing.

In future iterations, the game could benefit from modularizing the code further, such as encapsulating power-up handling and lives management into dedicated classes to improve maintainability and scalability.