Documentation: 2D Space Shooter Game using OOP Techniques

Problem A: Produce a sample game using OOP techniques.

Definition Statement

Summary:

The goal of this project is to develop a sample game using object-oriented programming (OOP) techniques. The game will be a 2D space shooter where the player controls a spaceship to shoot down enemies and earn points.

Complexity:

The complexity of this game is considered moderate. It involves managing player input, handling collisions, updating game entities, and displaying graphics and text.

Constraints:

The game will be developed using the Pygame library.

The game window will have a fixed size of 800x600 pixels.

The player's spaceship will move horizontally and shoot bullets vertically.

Enemy spaceships will appear from the top of the screen and move downwards.

The game will keep track of the player's score and display it on the screen.

Highscores will be stored in an SQLite database.

Intended Users:

The intended users for this game are gamers and developers who are interested in learning about game development using OOP principles in Python.

Required Interactivity:

The game will require user interaction through keyboard input. The player will be able to move the spaceship left and right using the arrow keys and shoot bullets using the spacebar. The game will respond to these inputs to update the game state and display the game entities on the screen.

Algorithm Designs

Pseudo Code:

Import necessary libraries/modules

Initialize game elements and variables

Set up the game window

Define colors

Define Database class

    Initialize database connection

    Create table if it doesn't exist

    Add player score to the table

    Update player score in the table

    Get top scores from the table

    Close the database connection

Connect to the database

Take input of the player's name

Set up input prompt and rendering

Input loop

    Handle events

        Check for quit event

Check for key presses (Enter, Backspace, and other characters)

Update the rendered input text

Clear the screen

Draw the input text

Update the display

Add the player's name to the database

Define Player class

Initialize player attributes

Move player left

Move player right

Shoot bullets

Update player bullets

Draw player on the screen

Define Bullet class

Initialize bullet attributes

Update bullet position

Draw bullet on the screen

Define Enemy class

Set maximum number of enemies

Initialize enemy attributes

Update enemy position

Draw enemy on the screen

Create new enemy

Define ScoreCounter class

Initialize score attribute

Increment score

    Draw score on the screen

Set up game loop
    Handle events
        Check for quit event
    Handle player movement
    Update player bullets
    Update enemy positions
    Perform collision detection
    Spawn new enemies
    Clear the screen
    Draw game elements
    Update the display

Display Game Over message
Wait before displaying high scores
Update player's score in the database
Retrieve top scores from the database
Display high scores
Wait before quitting the game
Close the database connection
Quit the game

Records of Review Discussions

First Review Discussion:

- Reviewed the initial code implementation for the Space Shooter game.
- Discussed the structure and organization of the code.
- Examined the implementation of player movement, shooting, enemy spawning, collision detection, and game over conditions.

Decisions Made:

- Agreed to keep the existing code structure as it follows object-oriented programming (OOP) principles.
- Decided to proceed with the current implementation and refine it further.

Second Review Discussion:

- Reviewed the integration of the SQLite database for storing player names and scores.
- Discussed the implementation of adding player names to the database and updating scores.

Decisions Made:

- Confirmed that the database integration was correctly implemented.
- Decided to proceed with the current implementation and ensure proper database interactions.

Third Review Discussion:

- Reviewed the user input handling for entering player names.
- Discussed potential edge cases and special characters in player names.

Decisions Made:

- Confirmed that the user input handling was implemented appropriately.
- Decided to proceed with the current implementation and address any edge cases during testing.

Fourth Review Discussion:

- Reviewed the collision detection logic for player bullets and enemies.

- Discussed the scoring mechanism and its increment upon collision.

Decisions Made:

- Confirmed that the collision detection and scoring mechanism were implemented correctly.
- Decided to proceed with the current implementation and ensure accurate score tracking.

Fifth Review Discussion:

- Reviewed the game over condition and its display on the screen.
- Discussed the delay before displaying high scores.

Decisions Made:

- Confirmed that the game over condition and display were implemented correctly.
- Decided to proceed with the current implementation and maintain the delay before displaying high scores.

Test Plan:

1. Test Player Movement:
    - Verify that the player can move left and right using the arrow keys.
    - Test the player's movement boundaries to ensure they cannot move off the screen.
2. Test Player Shooting:
    - Verify that the player can shoot bullets using the spacebar.
    - Test the maximum number of bullets the player can shoot at once.
    - Check that there is a delay between consecutive shots.
3. Test Enemy Spawning:
    - Ensure that enemies are spawning at regular intervals.
    - Verify that the maximum number of enemies on the screen is maintained.
    - Check that enemies are being reused when they go off the screen.
4. Test Collision Detection:
    - Test collisions between player bullets and enemies.
    - Verify that the score increments when a collision occurs.
    - Ensure that the collided bullet and enemy are removed from the game.
5. Test Game Over Condition:
    - Verify that the game ends when the player collides with an enemy.
    - Test the display of the "Game Over" message on the screen.

6. Test Database Integration:
    - Verify that the player's name is correctly stored in the database.
    - Test updating the player's score in the database.
    - Check that the top scores are retrieved correctly from the database.
7. Test User Input:
    - Test entering the player's name to ensure correct input handling.
    - Check for any potential edge cases, such as empty names or special characters.
8. Test Highscore Display:
    - Verify that the top scores are displayed correctly on the screen.
    - Test the layout and formatting of the highscore display.
9. Test Game Quitting:
    - Verify that the game closes properly when the window close button is clicked.
    - Ensure that the database connection is closed before quitting the game.

Test Logs

1. Test Case: Verify player movement
    - Expected Result: The player should be able to move left and right using the arrow keys.
    - Actual Result: Player movement is functioning correctly within the screen boundaries.
2. Test Case: Verify player shooting
    - Expected Result: The player should be able to shoot bullets using the spacebar.
    - Actual Result: Player shooting is functioning correctly, and bullets are generated as expected.
3. Test Case: Verify enemy spawning
    - Expected Result: Enemies should spawn at regular intervals.
    - Actual Result: Enemy spawning is working correctly, and the maximum number of enemies on the screen is maintained.
4. Test Case: Verify collision detection
    - Expected Result: Collisions between player bullets and enemies should be detected.
    - Actual Result: Collision detection is working accurately, and the score increments upon collision.
5. Test Case: Verify game over condition
    - Expected Result: The game should end when the player collides with an enemy.
    - Actual Result: The game ends correctly upon player-enemy collision, and the "Game Over" message is displayed.

6. Test Case: Verify database integration
    - Expected Result: Player name should be stored correctly in the database.
    - Actual Result: Player name is successfully stored in the database, and score updates are reflected.
7. Test Case: Verify user input handling
    - Expected Result: Player input for name should be handled correctly.
    - Actual Result: User input is properly handled, including potential edge cases.
8. Test Case: Verify highscore display
    - Expected Result: Top scores should be displayed accurately on the screen.
    - Actual Result: Highscores are correctly displayed with the expected layout and formatting.
9. Test Case: Verify game quitting
    - Expected Result: The game should close properly without any errors.
    - Actual Result: The game closes correctly when the window close button is clicked, and the database connection is closed.

Error Report

Issue: Player movement is not working properly.

Description: When pressing the arrow keys, the player does not move.

Fix: The issue was due to missing event handling for key presses in the game loop. By adding the appropriate event handling code for key presses, the player movement was fixed. The player can now move left and right using the arrow keys as intended.

Issue: Game crashes when player collides with an enemy.

Description: When the player collides with an enemy, the game crashes instead of displaying the "Game Over" message.

Fix: The issue was caused by missing logic to handle the collision between the player and enemy. By implementing the necessary collision detection and game over handling, the game now properly ends when a collision occurs. The "Game Over" message is displayed on the screen as expected.

Issue: Database connection not closing properly.

Description: After quitting the game, the database connection remains open.

Fix: The issue was due to missing code to close the database connection before quitting the game. By adding the appropriate code to close the database connection, the issue was resolved. The database connection now closes properly when the game is exited.

Issue: Incorrect retrieval of top scores from the database.

Description: The displayed top scores do not match the actual highest scores stored in the database.

Fix: The issue was caused by an incorrect SQL query used to retrieve the top scores. By modifying the SQL query to correctly order the scores and limit the results to the desired number, the top scores are now retrieved accurately from the database.

Optimization Logs

Improved Player Movement:

Implemented boundary checks to prevent the player from moving off the screen, resulting in smoother gameplay and preventing potential crashes.

Enhanced Enemy Spawning:

Optimized enemy spawning logic to ensure a consistent and controlled rate of enemy creation, providing a balanced gameplay experience.

Bullet Management Optimization:

Implemented a limit on the number of bullets the player can shoot simultaneously to prevent performance issues and maintain game responsiveness.

Collision Detection Efficiency:

Optimized collision detection algorithms between player bullets and enemies to improve performance and reduce processing overhead.

Database Interaction Optimization:

Implemented efficient database queries to store and retrieve player scores, reducing database access time and enhancing overall game performance.

Performance Improvements:

Employed various optimization techniques, such as minimizing unnecessary calculations and reducing redundant code, to improve the game's overall performance and responsiveness.

My Evaluation

The development of the Space Shooter game using OOP concepts has been successful. The program effectively implements player movement, shooting mechanics, enemy spawning, collision detection, score tracking, and database integration. The game provides an enjoyable user experience with smooth gameplay. The code demonstrates a clear understanding of object-oriented programming principles and follows good coding practices. Overall, the completed program meets the requirements of the problem statement and delivers a functional and entertaining game.

Documentation

As the sole developer of the Space Shooter game, I have demonstrated individual responsibility and effective self-management throughout the development process. I have taken ownership of the project, including designing, coding, and testing the game.

I have followed a structured approach by breaking down the development tasks into manageable components, such as player movement, shooting mechanics, enemy spawning, collision detection, and database integration. This allowed me to work on each feature independently and efficiently.

Additionally, I have effectively managed my time and resources by setting clear goals and prioritizing tasks. I have maintained a systematic workflow, starting with designing the game, writing the code using object-oriented programming concepts, and thoroughly testing each functionality to ensure its proper working.

Furthermore, I have demonstrated adaptability by incorporating user input through the player's name input feature. I have handled different events, including keyboard inputs, window close events, and collision detections, ensuring a seamless and user-friendly experience.

Throughout the development process, I have maintained clear and concise code documentation, making it easier for others to understand and collaborate on the project if needed. The provided code includes comments explaining the purpose and functionality of each class, method, and variable.

In conclusion, my individual responsibility and effective self-management are reflected in the systematic development approach, efficient use of time and resources, adaptability to user input, and clear code documentation exhibited in the Space Shooter game project.

Problem B: Produce a module that uses OOP techniques to access a database to create and update a high score table.

Definition Statement

Summary:

The goal is to develop a module that allows for the creation and updating of a high score table using object-oriented programming (OOP) techniques and database access. The module will facilitate the storage and retrieval of player scores, enabling the maintenance and display of top scores for a given game or application.

Complexity:

The complexity of this problem is considered moderate. It involves integrating OOP principles, database management, and user interaction to handle score data efficiently and provide an intuitive interface for users.

Constraints:

- The module must be developed using Python.
- The database system to be used should support SQL queries (e.g., SQLite).
- The module should conform to good programming practices and adhere to OOP concepts.
- The design should prioritize data security and avoid vulnerabilities such as SQL injection.

Intended Users:

The module is intended for game developers or software developers who want to incorporate a high score feature into their games or applications. It can be used by individuals or teams working on projects involving score tracking and leaderboard functionality.

Required Interactivity:

The module should provide the following interactive features:

- Initialize the high score table in the database.
- Add a player's score to the high score table.

- Update a player's score in the high score table.
- Retrieve and display the top scores from the high score table.
- Allow customization of the number of top scores to be displayed.
- Handle potential errors or exceptions related to database access and operations.

Algorithm Designs

Pseudo Code:

Define Database class:

    Initialize database connection

    Create high score table if it doesn't exist

    Add player score to the table

    Update player score in the table

    Get top scores from the table

    Close the database connection

Connect to the database

Take input of the player's name

Set up input prompt and rendering

Input loop:

    Handle events:

        Check for quit event

        Check for key presses (Enter, Backspace, and other characters)

    Update the rendered input text

    Clear the screen

    Draw the input text

    Update the display

Add the player's name to the database

Set up game loop:

    Handle events:

        Check for quit event

    Clear the screen

    Draw game elements

    Update the display

Display Game Over message

Wait before displaying high scores

Update player's score in the database

Retrieve top scores from the database

Display high scores

Wait before quitting the game

Close the database connection

Quit the game

Test Plan for High Score Table

1. **Test Name: Database Connection:**

   Objective: Ensure that the database connection is established successfully.

   Test Steps:

   - Connect to the database.
   - Check the status of the database connection.

   Expected Result: The database connection should be established without any errors.

   Actual Result: The database connection was established successfully without any errors.

2. **Test Name: Creating High Score Table:**

   Objective: Verify that the high score table is created in the database.

   Test Steps:

   - Create an instance of the Database class.
   - Check if the high score table exists in the database.

   Expected Result: The high score table should be created if it doesn't exist.

   Actual Result: The high score table was created in the database.

3. **Test Name: Adding Player Score**

   Objective: Test the functionality of adding a player's score to the high score table.

   Test Steps:

   - Create an instance of the Database class.
   - Add a player's score to the high score table.
   - Retrieve the player's score from the table.

   Expected Result: The player's score should be added to the high score table and retrieved successfully.

   Actual Result: The player's score was added to the high score table and retrieved successfully.

4. **Test Name: Updating Player Score**

   Objective: Verify that a player's score can be updated in the high score table.

   Test Steps:

   - Create an instance of the Database class.
   - Update a player's score in the high score table.
   - Retrieve the updated score from the table.

   Expected Result: The player's score should be updated in the high score table and retrieved successfully.

   Actual Result: The player's score was updated in the high score table and retrieved successfully.

5. **Test Name: Retrieving Top Scores**

   Objective: Test the functionality of retrieving the top scores from the high score table.

   Test Steps:

   - Create an instance of the Database class.
   - Retrieve the top scores from the high score table.

   Expected Result: The top scores should be retrieved successfully, and the result should match the expected number of scores.

   Actual Result: The top scores were retrieved successfully, and the result matched the expected number of scores.

6. **Test Name: Closing Database Connection**

   Objective: Ensure that the database connection is closed properly.

   Test Steps:

   - Create an instance of the Database class.
   - Close the database connection.

   Expected Result: The database connection should be closed without any errors.

   Actual Result: The database connection was closed properly without any errors.