Game Documentation

# Overview

This code provides the implementation of a simple space shooter game using the Pygame library. The player controls a spaceship and must shoot down enemy spaceships while avoiding their projectiles. The objective is to survive as long as possible and achieve a high score.

# Game Components

The game consists of the following components:

1. Spaceship: The player-controlled spaceship that can move left, right, up, and down on the game screen. It can shoot projectiles to destroy enemy spaceships.
2. Enemy Spaceships: The enemy spaceships that move across the game screen from the top. They shoot projectiles at the player’s spaceship and must be destroyed to earn points.
3. Projectiles: Both the player and enemy spaceships can shoot projectiles. The player’s projectiles destroy enemy spaceships, while the enemy’s projectiles can destroy the player’s spaceship.
4. Power-ups: Occasionally, power-ups in the form of shields appear on the game screen. When collected by the player’s spaceship, they provide temporary invincibility from enemy projectiles.
5. Scores: The game keeps track of the player’s current score and highest score achieved.
6. Lives: The player starts with a certain number of lives, and one life is lost when the player’s spaceship collides with an enemy projectile or enemy spaceship.

# Functions and Usage

The code includes the following functions:

1. `initialize\_game`: Initializes the Pygame library and sets up the game window.
2. `load\_images`: Loads the images for the player’s spaceship, enemy spaceships, projectiles, and power-ups from image files.
3. `initialize\_sprites`: Creates sprite objects for the player’s spaceship, enemy spaceships, projectiles, and power-ups.
4. `handle\_events`: Handles user input events, such as pressing arrow keys to move the player’s spaceship or pressing the spacebar to shoot projectiles.
5. `move\_spaceship`: Updates the position of the player’s spaceship based on user input.
6. `shoot\_projectile`: Creates a new projectile object and adds it to the corresponding sprite group.
7. `move\_enemy\_spaceships`: Updates the positions of the enemy spaceships as they move across the game screen.
8. `shoot\_enemy\_projectile`: Randomly selects an enemy spaceship and creates a new enemy projectile object, which is added to the corresponding sprite group.
9. `show\_pause\_screen`: Pauses the game and displays a pause screen. It waits for user input to resume the game or quit.
10. `check\_collisions`: Checks for collisions between sprites and performs the appropriate actions, such as destroying enemy spaceships or deducting lives.
11. `update\_scores`: Updates the current score and highest score based on the number of enemy spaceships destroyed.
12. `update\_lives`: Updates the number of lives remaining based on collisions between the player’s spaceship and enemy projectiles or enemy spaceships.
13. `update\_sprites`: Updates the positions and states of all sprites, including the player’s spaceship, enemy spaceships, projectiles, and power-ups.
14. `draw\_sprites`: Draws all sprites on the game screen.
15. `draw\_scores`: Draws the current score and highest score on the game screen.
16. `draw\_lives`: Draws the number of lives remaining on the game screen.
17. `draw\_shield`: Draws the shield power-up indicator on the game screen.
18. `game\_loop`: The main game loop that handles user input, updates the game state, and renders the game screen.
19. `start\_game`: Starts the game by initializing variables, displaying the start screen, and entering the game loop.

# Game Flow

1. The game starts by calling the `start\_game` function.

2. The start screen is displayed, and the player can choose to start a new game or load a saved game.

3. If a new game is started, the game variables and sprite groups are reset.

4. The game loop is entered, which continuously updates the game state, handles user input, and renders the game screen.

5. During the game loop, collisions are checked, scores and lives are updated, and sprites are updated and drawn on the screen.

6. The game loop continues until the player loses all lives or quits the game.

7. If the game ends, the game over screen is displayed, the highest score is updated if necessary, and the player can choose to restart or quit.

8. If the player chooses to restart, the game variables and sprite groups are reset, and the game loop starts again.

9. If the player chooses to quit, the game exits.

# Conclusion

The provided code offers a foundation for a simple space shooter game using Pygame. By understanding the game components, functions, and game flow, you can further customize and expand the game to add more features, levels, or challenges.