Nebula Guardian

Project Architecture Overview

The project is structured into several core components, each responsible for different aspects of the game's functionality. These components interact to provide a cohesive gaming experience, managing everything from rendering to user input and game logic.

Dependencies and Configuration Files :

- Makefile: Contains compilation instructions for running the application.
- .gitignore: Specifies files and directories to be ignored by Git.

Utilities Files :

- converter.c: An additional script for converting .ppm images into a custom .565 (rgb565) format used for displaying graphics on the parlcd screen.
- utils.cpp: Contains the Image struct and loadImg function, which loads custom .565 images for further use in the application.

Main Files :

- main.cpp: Launches the application.
- scene_controller.cpp: Manages the main game loop and transitions between different game states.

MicroZed Board Controller Files :

- parlcd_controller.cpp: Handles the initialization and memory mapping for the parallel LCD display.
- diod_controller.cpp: Manages the control of the LED line on the device through direct memory access.
- knobs_controller.cpp: Handles interaction with the knobs and button presses on the device, including reading the current state and temporarily disabling inputs.
- render_controller.cpp: Responsible for managing the rendering of various graphical elements on the LCD screen. It handles the initialization of the display, loading of images, and rendering of text and graphics.

Entities Files :

- entity.cpp: Represents a generic game entity with position and size attributes, and it interfaces with a RenderController for rendering purposes.
- player.cpp: Represents the player's ship in the game, inheriting from the Entity class. It manages the position, rendering, movement, health, and appearance of the player's ship.
- bullet.cpp: Represents a bullet entity in the game, inheriting from the Entity class. It manages the position, rendering, and state of the bullet.
- asteroid.cpp: Represents an asteroid entity in the game, inheriting from the Entity class. It manages the position, rendering, and state of the asteroid.

Entity Controller Files :

- asteroid_controller.cpp: Manages the generation, rendering, and state of asteroids in the game.
- bullet_controller.cpp: Manages the creation, rendering, and state of bullets in the game.
- collision_controller.cpp: Responsible for detecting and handling collisions between game entities such as bullets, asteroids, and the player. Also updates the game score when collisions are detected.

Scene Files :

- scene.cpp: Serves as a base class for different game scenes, managing their rendering and user input.
- main_menu.cpp: Represents the main menu scene of the game, inheriting from the Scene class and managing the display and user interactions specific to the main menu.
- level.cpp: Represents the main gameplay scene of the game, inheriting from the Scene class. It manages the player's interactions, rendering of the game elements, and updates during gameplay.
- gameover_menu.cpp: Represents the game over scene, inheriting from the Scene class. It displays the final score and provides an option to return to the main menu.
- settings_menu.cpp: Represents the settings scene, inheriting from the Scene class. It allows the player to select different skins for their ship and then return to the main menu.

Summary

This architecture divides the game into manageable, modular components, facilitating easy maintenance, testing, and expansion. Each component has a clear responsibility, contributing to the overall functionality and user experience of the game.