Introduction to Programming

Game Project

Comprehensive Project Documentation

1. Overview

This project is based on the 'Lost in Space' game framework. It involved implementing dynamic arrays with C++ vectors and using data structures to manage multiple power-ups. The aim was to evolve the simple original version into a more complete, modular, and interactive game. The project emphasizes structured programming, decomposition, collision detection, and power-up effects, all developed using the SplashKit library.

2. Task Requirements

The Game Project outlined the following requirements:
- Create a new game_data struct to manage the game state (player and power-ups).
- Implement vector<power_up_data> to store multiple power-ups dynamically.
- Relocate code into update_game and draw_game functions to simplify main.
- Add functionality for randomly spawning power-ups, updating their positions, and checking collisions.
- Apply appropriate effects (fuel replenishment, score increase, etc.) when player collects power-ups.
- Draw a functional HUD showing player level, position, and fuel percentage.
- Ensure proper modular decomposition into multiple header and source files.

3. Project Files

The project was decomposed into multiple modules for clarity and modularity:
- player.h → Player header file (defines player_data struct, player functions).
- player.cpp → Player implementation (movement, input handling, update logic).
- power_up.h → Power-up header file (defines power_up_data struct, power-up functions).
- power_up.cpp → Power-up implementation (creation, drawing, updating).
- lost_in_space.h → Lost in Space game header (game_data struct, declarations of game functions).
- lost_in_space.cpp → Lost in Space game implementation (update_game, draw_game, collisions, etc.).
- main.cpp → Main entry point, resource loading, and event loop.

4. Data Structures

The project employed the following core data structures:
- game_data struct: Stores the overall game state including the player and vector of power-ups.
- player_data struct: Holds player information such as sprite, ship kind, score/level, and fuel percentage.
- power_up_data struct: Holds information about each power-up, including type and sprite.

Enums used:
- ship_kind: AQUARII, GLIESE, PEGASI.
- power_up_kind: LIFE, FUEL, STAR, HEART.

5. Key Functionalities

5.1 Game Lifecycle

- main(): Entry point that opens the window, loads resources, creates a new game, and runs the event loop.
- load_resources(): Loads bitmaps and sounds required for the game.

5.2 Game Logic

- new_game(): Initializes the game with a new player.
- update_game(): Manages spawning of power-ups, updating player and power-up states, and collision checking.
- draw_game() and draw_hud(): Responsible for drawing the HUD, player, and all power-ups.

5.3 Power-Up System

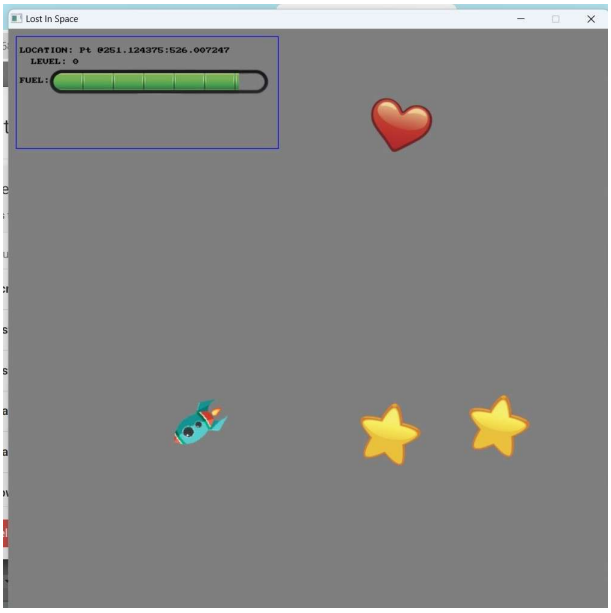
- add_power_up(): Randomly generates new power-ups within a specified range.
- check_collisions(): Detects player-power-up collisions and applies effects.
- apply_power_up(): Applies fuel replenishment or score increase and plays sound effects.
- remove_power_up(): Removes collected power-ups safely from the vector.

5.4 Player System

- new_player(): Creates the player with default sprite, fuel, and ship kind.
- update_player(): Updates player position, camera, and fuel consumption.
- handle_input(): Handles keyboard inputs for movement, rotation, and ship switching.

6. Demonstration

The demonstration consists of:
- Screenshot of the game window showing the player and multiple power-ups.



- A screencast (.mp4) demonstrating gameplay, movement, power-up spawning, collection, and HUD updates. The player can see fuel levels, level progression, and feedback sounds when power-ups are collected.

https://youtu.be/Tx1fT8I-ibs

7. Outcomes

From completing this project, the following outcomes were achieved:
- Successfully implemented dynamic arrays (vectors) for handling multiple power-ups.
- Applied modular decomposition for better program design.
- Implemented collision detection and response between player and power-ups.
- Created a functional HUD with live updates on player status.
- Strengthened understanding of SplashKit library for graphics, sound, and sprite management.
- Gained experience in iterative development: testing small changes, debugging, and integrating functionality.

8. Conclusion

This Game Project successfully the 'Lost in Space' project into a more complex and fully featured game. The task demonstrated proficiency in C++ programming concepts, dynamic arrays, modular design, and interactive graphics programming.

The project outcomes provided valuable hands-on experience in developing structured, data-driven game applications.