# 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 screenshot of a computer

AI-generated content may be incorrect.

- 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.