Design and Implementation of "Puff Puff Planet" A Lightweight, Retro Game Using SFML

School of Mathematical and Computational Sciences
Yachay Tech University
daniel.troya@yachaytech.edu.ec
Daniel Troya Atancuri

***Abstract*—This project presents Puff Puff Planet, a game developed in C++ to make programming education fun and accessible for children. Built with SFML for vibrant graphics and smooth gameplay, the game combines physics-based mechanics with engaging challenges. The current prototype demonstrates stable performance and intuitive controls, highlighting C++'s versatility in creating immersive experiences for young players.**

***Index Terms*—SFML Framework, Game Development, C++, Lightweight Gaming, Advanced Programming**

I. Introduction

The growing demand for video games has led developers to explore new approaches. However, modern games increasingly require high performance hardware, creating accessibility barriers. This project addresses these challenges by developing **Puff Puff Planet**, a lightweight and retro style game.

The game is built using SFML (Simple and Fast Multimedia Library), a cross-platform framework that provides an easy-to-use API for handling graphics, sound, and player inputs. Designed to be simple and accessible, it features clear game mechanics and a simple interface made for young players, offering a fun experience that works on most computers.

II. Conceptual Background

A. SFML

Simple and Fast Multi-media Library (SFML) is a cross-platform framework that provides an easy-to-use API for handling graphics, audio, and input. Designed for simplicity and accessibility, it features intuitive mechanics and a user-friendly interface tailored for young players, ensuring an engaging experience without demanding hardware resources.

B. Sprites

Is a two-dimensional bitmap image that is integrated into a larger scene. Sprites are commonly used to represent characters, objects, or other visual elements in 2D games, with the characteristics:

- They are **independent graphical objects** that can move separately from the background
- Can be **animated** by displaying a sequence of images called frames (more frames result in clean animation)
- Often organized in **sprite sheets** (collections of multiple sprites in a single image file)

C. ChatGPT Image Generator

ChatGPT is an AI capable of generating images. In this project, we used this tool to create all the images for the game. We simply described the story and provided prompts to generate the visuals. It is a powerful tool because the images were tailored to our specific requirements.

III. Problem Statement

The video game industry has experienced exponential growth in graphical fidelity and system requirements, leaving users with older or low hardware unable to enjoy modern titles. While high performance games dominate the market, there remains significant demand for lightweight, accessible alternatives that prioritize fun gameplay. In this project, we approached development with a layer of sarcastic humor, aiming to shift the player's perception of "What kind of game am I playing?"

Some disadvantages in current game development trends include:

- **Hardware Limitations:** Many modern games rely on engines with excessive overhead, excluding players with modest or older PCs.
- **Design Complexity:** Games targeting younger or casual audiences often complicate simple mechanics with unnecessary features, reducing overall accessibility.
- **Lack of Open Source:** Most commercial games are not open source, which is problematic. Players often think of ways the game could be improved, but without access to the source code, they can't contribute new features or fixes.

IV. Solution Statement

Puff Puff Planet addresses these challenges through the following design principles:

- **Efficient Performance:** The game is built with SFML, ensuring low resource consumption while maintaining engaging gameplay.
- **Simple Gameplay:** Mechanics are inspired by classic retro games, combining nostalgic fun with modern sarcastic humor.

- **Accessible Design:** The interface is intuitive and friendly, suitable for players of all ages (+16) and hardware capabilities.
- **Open Source Philosophy:** The project is open source, allowing players to contribute to the development process, propose new features, improve gameplay mechanics, and even form a community to drive future updates, adding more functionalities and complexity without losing the essence of fun.

This project demonstrates how lightweight frameworks like SFML can bridge the gap between accessibility and entertainment in modern game development.

V. Scope

This project begins as a simple spaceship game. At the moment, its implementation is basic in terms of complexity and functionality. However, with proper documentation, it can be scaled in the future. The main goal is to capture the user's attention and make the game memorable. Some specific objectives include:

A. Objectives

- Create a humorous game experience using sarcastic humor, encouraging users to replay and enjoy the experience more than once.
- Design a simple and intuitive interface to minimize stress. Players shouldn't need a manual to understand what each button does. This project is intended to be "open and play", no rare or confusing controls.
- Implement background music to enhance immersion and capture the player's attention, making the gameplay experience more engaging.

B. Technical Implementation

Some key technical aspects considered in this project are:

- The sprites will use vibrant and emotive colors, making the game visually friendly and enjoyable for the eyes.
- Modular programming will be applied to maintain a clean structure, allowing each game component to be developed and improved independently.
- The code will be thoroughly commented to facilitate fast and effective debugging in case any issues arise.
- The game screen dimensions are set to a width of 1200 and a height of 1000. This resolution provides enough space for smooth spaceship movement and ensures all characters are displayed clearly and in high quality.

VI. Libraries

As you can see, the main library used for the development of this game was the Simple and Fast Multimedia Library (SFML). From this main library, we specifically used the following module:

A. Graphics.hpp

This module is essential for rendering the game. It allows us to add and manage text, manipulate sprites, handle window operations, and more. It is a powerful tool because it provides access to classes such as:

- **sf::CircleShape:** Used in early development to test rendering and movement of basic shapes on the screen.
- **sf::RenderTexture:** This class allows us to render textures off-screen before displaying them. It was used to compose and display game elements like characters and objects efficiently.
- **sf::RenderWindow:** The core window class that handles rendering everything to the screen. It allows us to set the frame rate limit (144 FPS), detect whether the window is open or closed, and manage input events through functions like `pollEvent()`, which listens for player actions and interactions.
- **sf::Sprite:** This class is essential for rendering images in the game. Sprites represent visual objects, and this class provides a variety of useful methods:
  - **Set Properties:** You can configure properties like color, texture, scale, rotation, and position to customize each sprite.
  - **Get Properties:** Functions such as `getPosition()`, `getGlobalBounds()`, and `getLocalBounds()` help retrieve spatial data, which is crucial for gameplay mechanics like collision detection or ensuring sprites remain on screen.
- **sf::Text:** This class is used to render text on screen. Just like sprites, you can set and get various properties (font, size, position, color), giving you full control over how textual elements are displayed in the game.

B. System.hpp

This library provides essential system-level functionalities, such as vector math, time handling, and utility functions. In our game, we use elements like:

- **sf::Vector2f:** Used to manage positions and movement directions in classes like `Bullet`, `Enemy`, and `Player`.
- **Integration with other SFML modules:** `System.hpp` is indirectly required for `Graphics`, `Window`, and `Audio` functionalities.

It is a foundational library that supports the internal behavior and interaction of game entities.

C. Window.hpp

This library is responsible for handling real-time user input, such as keyboard and window events. It plays a central role in our main game loop:

- **sf::Event:** Used inside the `Game` class (specifically within the render window) to detect actions like key presses or window closure.
- **pollEvent():** A method tied to the main window object that continuously checks for user input and allows responsive gameplay.

Without this library, interaction between the player and the game would not be possible.

D. Audio.hpp

This library is designed for managing game audio, including background music and sound effects. In our project:

- Audio functionality is included via `<SFML/Audio.hpp>`, allowing the potential to integrate sound effects (shooting, collisions, or background music).
- Although present, full implementation of audio elements is not completed yet. However, the library is prepared for future expansions in sound design.

Audio elements will greatly enhance immersion and player experience once implemented.

All of this can help us to create a video game engine, which is responsible for graphical rendering, managing game physics, sound, etc. Today there exist different game engines, such as:

- Unity (C#)
- CryEngine (C++)
- Unreal Engine (C++)

and many others. Each engine has its own specialized properties for different applications.

VII. Game

A. Resume

Puff Puff Planet is a 2D space shooter packed with psychedelic visuals, pizza-powered weapons, and intergalactic chaos. You play as Mr. Pacheco, a space explorer caught in the cosmic munchies while trying to escape the Intergalactic Police.

VIII. Characters

Figure 1. Mr. Pacheco — An astrophysics engineer motivated by science and space.

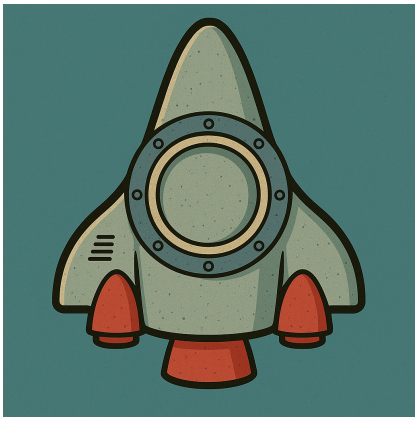

Figure 2. The Mozzarelladora — Mr. Pacheco's loyal spaceship, fueled by pizza and dreams.

Figure 3. Intergalactic Police — The cosmic authority chasing Mr. Pacheco across the stars.

Figure 4. It's just a pizza... or is it also a bullet?

A. Game History

Mr. Pacheco, an astrophysics engineer and passionate lover of Lonchys, was selected for the interplanetary mission "Found New Planet", aimed at exploring unknown worlds and retrieving valuable resources to bring back to Earth.

After years of drifting through the cosmic void, Mr. Pacheco landed on a strange and lush planet called Cannabora. A world teeming with psychedelic flora and fauna, full of vibrant plants and floating creatures that looked straight out of a dream... or a rave.

He collected several botanical samples and brought them aboard his ship. But while he slept, the plants began releasing a mysterious green smoke that spread through the ship's ventilation system. When Mr. Pacheco woke up, something felt off. The controls seemed to float, the colors were brighter, and an uncontrollable laughter pulsed through his body. His perception of space and time had shifted... dramatically.

But things got worse: the plants he had taken were sacred part of an ancient medicinal ritual protected by the laws of Cannabora. Now, the Intergalactic Police are after him for cosmic contraband.

Disoriented and hit with a galactic-level case of the munchies, Mr. Pacheco retrofitted his ship's defense system to fire hot pizzas, the only "ammunition" left on board after weeks of travel. Now, he must cross the galaxy, dodging patrols, space cops, and cosmic traps to make it back home... all while battling laughter, hunger, and that damn "autopilot" button.

B. Game Features

In the game, you control the main character, "Mr. Pacheco", who arrives on his spaceship. Your goal is to eliminate as many Galactic Police units as possible and earn points. The more enemies you destroy, the more points you earn — aim high and become the ultimate galactic shooter!

Currently, features like different types of enemies, bosses, and dynamic soundtracks are not yet available, but they are planned for future development.

The core idea is to survive and shoot as many enemies as possible to increase your score. Remember: you have a health bar, and every time a police ship collides with you or reaches your side of the screen, you lose points.

C. Controls

Thanks to the SFML library, we implemented two control modes for moving the ship, allowing players to adapt to different gameplay styles:

- **Normal Mode:** Use the arrow keys to move.
  - Up: ↑
  - Down: ↓
  - Left: ←
  - Right: →
- **Personalized Mode:** Use the WASD keys to move.
  - Up: W
  - Down: S
  - Left: A
  - Right: D

For shooting, there's only one key: **Space Bar**. We avoided using the mouse's left click, since players might accidentally fire a pizza when clicking elsewhere.

IX. Game Development

A. main.cpp

This script simply defines an object of the `Game` class and calls its `run()` function to start the game.

B. Game.h

This file contains the definition of the `Game` class, where all the functions used in the game are declared. It also defines the types of objects, such as texts, bullets, and enemies, and how they are managed.

Bullets and enemies are handled using vectors of pointers, which allows us to reference each one by its memory address. This approach avoids redundant copying or reloading, reducing memory overhead and improving performance. Additionally, all necessary variables are initialized here.

There are three main functions defined in this class:

- `run()` – Calls the two main game loop functions.
- `update()` – Calls all the update-related functions.
- `render()` – Calls all the rendering-related functions.

C. Game.cpp

This file implements all the functions declared in `Game.h`. It contains the core game loop, including event handling, player and enemy updates, collision detection, enemy spawning, and rendering of all game elements. It ensures the game flow runs continuously and smoothly during gameplay.

D. Player.h

Defines the `Player` class, which is responsible for managing the player's behavior. This includes movement, shooting logic, and access to the player's bullets. It also stores the sprite, texture, and a vector of bullets for rendering and interaction with enemies.

E. Player.cpp

Implements the methods declared in `Player.h`. It handles the player's movement based on keyboard input, firing bullets when specific keys are pressed, and updating the bullets. It also ensures bullets are removed when they go off-screen to avoid memory issues.

F. Bullet.h

Defines the `Bullet` class, which represents the projectiles fired by the player. It includes attributes like position, speed, texture, and the methods required to update the bullet's state. It also provides collision boundaries for hit detection.

G. Bullet.cpp

Implements the logic of the `Bullet` class. It manages how bullets move upwards each frame, checks if they leave the screen, and provides functionality to retrieve their bounds for collision checking with enemies.

H. Enemy.h

Defines the `Enemy` class, which models the enemy entities falling from the top of the screen. It includes data members for position, sprite, texture, and methods to update their position and access their collision bounds.

I. Enemy.cpp

Implements the behavior of enemies during gameplay. Each enemy moves downwards automatically, is removed if it goes off-screen, and can be destroyed upon collision with a bullet. The file ensures that enemy updates are handled efficiently.

X. Preliminary Results

XI. Conclusions

- Video game design must follow a structured and modular approach to prevent logic errors, improve maintainability, and efficiently identify and resolve issues during development. A well-organized codebase makes testing and future expansion much easier.
- Libraries like SFML play a crucial role in game development. They simplify complex tasks such as graphics rendering, input handling, and resource management, allowing developers to focus on the game logic rather than low-level implementations.
- Using object-oriented programming and separating responsibilities between classes (e.g., player, enemies, bullets) improves clarity, scalability, and debugging. This separation of concerns is essential for building more complex game systems.

References

[1] SFML Development Team, "SFML Documentation 3.0.0," https://www.sfml-dev.org/documentation/3.0.0/topics.html, 2023.

[2] R. Pupius, *SFML Game Development By Example*. Packt Publishing Ltd, 2015.

[3] M. G. Milchev, *SFML Essentials*. Packt Publishing, 2015.