Project Proposal

Space Shooter Game

Group Members

Abdullah	24k-0859 (Group Leader)
Hammad	24k-544
Abdul Majid	24k-0895.

1. Introduction

- **Background**: A 2D arcade-style game developed in C++ using Raylib, demonstrating OOP principles (inheritance, polymorphism, encapsulation) through interactive gameplay. The player controls a spaceship to combat polymorphic enemies with increasing difficulty.
- **Problem Statement**: Students often struggle to apply OOP concepts in real-world projects. This game bridges the gap by implementing:

➢ **Polymorphism**: Enemy hierarchy (`EnemyLevel1`, `EnemyLevel2`, `EnemyLevel3`).
➢ **Encapsulation**: Private members (e.g., `Player::health`) with public methods.
➢ **Composition**: `Player` aggregates `Bullet` objects.

- **Objectives**:

➢ Develop a modular game with scalable OOP architecture.
➢ Implement dynamic collision, scoring, and UI systems.
➢ Showcase resource management (textures, sounds) via RAII.

2. Scope of the Project

- **Inclusions**:

➢ **Player System**:
  - Movement (WASD/arrows), shooting (spacebar), health management.
  - Explosion animations on death (sprite sheets).

➢ **Enemy System**:
  - Polymorphic enemies with tiered health/damage:
    - `EnemyLevel1`: 1 HP, 10 damage.
    - `EnemyLevel2`: 2 HP, 20 damage.

        - `EnemyLevel3`: 3 HP, 30 damage.
    - Score-based spawning logic (e.g., `EnemyLevel2` at `score ≥ 100`).
- ➢ **Combat & Collision**:
    - AABB collision (`CheckCollisionRecs`).
    - Bullet-enemy/player-enemy interactions.
- ➢ **UI & Game States**:
    - Start menu, rules screen, win/loss conditions (`current_screen` states).
    - Health bar, score display, and button interactions (`Button` class).
- ➢ **Technical Features**:
    - 60 FPS game loop with scrolling backgrounds.
    - Sound effects (shooting, explosions) via Raylib audio.

• **Exclusions**:

- ➢ Multiplayer or online features.
- ➢ Advanced AI (e.g., pathfinding).
- ➢ Save/load functionality.

3. Project Description

• **Overview**:The game is structured around three core OOP pillars:

1. **Inheritance**: `Enemy` base class → `EnemyLevel1/2/3` derivatives.
2. **Polymorphism**: Virtual methods (`GetDamageOnCollision()`, `GetScoreValue()`).
3. **Encapsulation**: Private data (e.g., `Bullet::active`) with public interfaces.

• **Technical Requirements**:

- ➢ **Compiler**: C++17 (for RAII destructors).
- ➢ **Library**: Raylib 4.5 (graphics, audio, input).
- ➢ **IDE**: Visual Studio Code (Windows).
- ➢ **Assets**: PNG sprites (player, enemies), WAV/MP3 sounds.

• **Project Phases**:

- ➢ **Research**:
    - Study Raylib's API for texture/audio management.
- ➢ **Planning**:
    - Class diagrams (UML) for `Player`, `Enemy`, `Bullet`.
- ➢ **Implementation**:
    - **Sprint 1**: Player movement + shooting.
    - **Sprint 2**: Enemy polymorphism + collision.
    - **Sprint 3**: UI (buttons, health/score).

- ➢ **Testing**:
  - Unit tests for collision logic.
  - Playtesting for difficulty balancing.

4. Methodology

- **Approach**: The development of the Space Shooter Game will follow an iterative and collaborative methodology, ensuring efficient progress and adaptability. The approach includes:

Iterative Development

- ➢ **Short Sprints**: The project will be divided into 1-3 week sprints, each focusing on a core feature:
  - **Sprint 1**: Player movement, shooting, and basic collision.
  - **Sprint 2**: Enemy class hierarchy (polymorphism) and spawning logic.
  - **Sprint 3**: UI (menus, score/health display) and sound effects.
  - **Sprint 4**: Playtesting, bug fixes, and polish.
- ➢ **Continuous Testing**: Each sprint includes unit tests (e.g., collision checks) and peer reviews to validate functionality before integration.

- **Team Responsibilities**:
- ➢ **Abdullah** - Enemy Class, Ship collision.
- ➢ **Hammad**: Player Class, driver program.
- ➢ **Abdul Majid**: Bullet Class, scoring, collision logic.

5. Expected Outcomes

- **Deliverables**:

- ➢ **Executable Game**:
  - Win/loss states, dynamic difficulty scaling.
- ➢ **Source Code**:
  - Fully documented C++ classes (Doxygen-style comments).
- ➢ **Report**:
  - Explanation of OOP patterns (e.g., polymorphism in `Enemy`).
- ➢ **User Guide**:

    - Controls: WASD (movement), spacebar (shoot).
    - Rules: Avoid enemies, score 500 to win.

• **Relevance**: This project strengthens understanding of core ICT and programming concepts:

➢ **OOP Concepts**: Class design, inheritance, polymorphism.
➢ **Game Dev Skills**: Real-time rendering, collision detection, state management.

6. Resources Needed

• **Software**:

➢ Microsoft Visual Studio or VS Code.
➢ Raylib C++ Library.
➢ PNG asset editor (e.g., Paint.NET, GIMP)

• **Other Resources**:

➢ Raylib API reference.
➢ C++ RAII/design pattern guides.