# Zomboid: Final Shelter - C++ Text-Based RPG

## 1. 项目简介

**Zomboid: Final Shelter** 是一款基于 C++ 开发的文本角色扮演游戏 (Text-Based RPG)。本项目旨在模拟一个末日丧尸环境下的生存挑战，玩家需扮演幸存者，在废弃的城市环境中探索、收集资源、对抗丧尸，并最终抵达目标安全点。游戏采用命令行界面，通过数字菜单进行交互，是实践 C++ 面向对象编程、内存管理、多文件项目组织等核心概念的学习项目。

## 2. 项目背景与目标

- **主题设定**: 丧尸末日 (Zombie Apocalypse), 生存恐怖 (Survival Horror)
    - **核心玩法**: 探索互联的房间，管理生命/耐力等核心属性，进行基于文本的战斗，收集物品以增强生存能力。
    - **项目目标**:
        - 实现一个功能完整的文本冒险游戏循环。
        - 应用 C++ 的核心特性，包括类、继承、虚函数 (接口)、指针和智能指针 ( `std::unique_ptr` )。
        - 构建清晰的多文件项目结构 ( `.hh` / `.cpp` )。
        - 练习面向对象设计原则。
        - 掌握基本的命令行编译流程 (g++)。

## 3. 核心特性

- **文本驱动交互**: 游戏进程完全通过控制台文本描述和玩家的菜单选择驱动。
- **多样的游戏元素**:
    - **环境探索**: 包含 6 个（3x2 布局）相互连接的房间，具有不同的描述和内容物。
    - **敌人种类**: 包含至少 4 种具有不同属性和行为模式的丧尸 (Walker, Runner, Bloater, Crawler)。
    - **物品系统**: 包含 7 种可收集物品，提供属性增益（武器、护甲）或恢复效果（消耗品）。
- **生存机制**: 玩家需关注生命值，并通过物品管理攻击和防御属性。
- **面向对象架构**:

    - 使用 `ICharacter` 接口规范角色行为。
    - `Player` 和 `Zombie` 类继承自 `ICharacter`。
    - `Room` 类封装地点信息和逻辑。
    - `Game` 类管理整体游戏流程和状态。
- C++ **特性应用**:
    - **指针**: 用于实现房间之间的连接 ( `Room* paths[]` ) 以及在房间中引用全局管理的僵尸对象 ( `Zombie*` )。
    - **智能指针**: 利用 `std::unique_ptr` 自动管理动态分配的资源（如 `Item`, `Zombie`, `Room` 对象本身），确保内存安全，避免内存泄漏。
    - **继承与多态**: 通过 `ICharacter` 接口和虚函数实现玩家与僵尸的统一处理。
    - **STL 容器**: 使用 `std::vector` 管理玩家背包、房间物品和僵尸列表；使用 `std::map` 管理菜单选项与动作的映射。
    - **枚举类**: 使用 `enum class ActionType` 定义清晰的动作类型。
- **模块化代码**: 源代码分离为头文件 ( `.hh` ) 和实现文件 ( `.cpp` )，存放于 `include/` 和 `src/` 目录。
- **清晰的胜利条件**: 到达指定的 5 号房间（废弃建筑大厅）。
- **用户友好菜单**: 基于数字的菜单系统引导玩家进行操作。
- **跨平台兼容性 (基础)**: 使用标准 C++ 和基本的控制台操作，具备在不同系统编译运行的潜力。

# 4. 技术栈

- **语言**: C++ (要求 C++11 或更高标准)
- **核心库**: Standard Template Library (STL), `<iostream>`, `<string>`, `<vector>`, `<memory>`, `<map>`, `<sstream>`, `<limits>`, `<algorithm>`, `<cctype>`, `<cstdlib>`
- **编译**: g++ (推荐) 或其他兼容 C++11 的编译器 (如 Clang, MSVC)

# 5. 安装与运行

## 5.1. 环境要求

- 支持 C++11 或更高标准的 C++ 编译器 (例如 g++ >= 4.8)。
- (可选) Git 客户端，用于克隆仓库。

## 5.2. 编译步骤 (使用 g++)

1. **克隆仓库**:

   ```
   git clone <your-repository-url>
   cd YourProjectFolder
   ```

   或者直接下载 ZIP 文件并解压。

2. **确认目录结构**:

   ```
   YourProjectFolder/
   ├── include/     (*.hh files)
   └── src/         (*.cpp files)
   ```

3. **进入源文件目录**:

   ```
   cd src
   ```

4. **执行编译**:

   ```
   g++ main.cpp Player.cpp Zombie.cpp Room.cpp Game.cpp Item.cpp Action.cpp -o zomboid_game -std=c++11 -I../include -Wall -Wextra -pedantic
   ```

   - **说明**:
     - `main.cpp ... Action.cpp`: 列出所有需要编译的 `.cpp` 文件。若 `Item.cpp` 或 `Action.cpp` 为空，可省略。
     - `-o zomboid_game`: 指定输出的可执行文件名为 `zomboid_game`。
     - `-std=c++11`: 强制使用 C++11 标准。
     - `-I../include`: 指定头文件的搜索路径。
     - `-Wall -Wextra -pedantic`: (推荐) 开启更多编译器警告，有助于提高代码质量。

## 5.3. 运行游戏

- Linux / macOS:

  ```
  ./zomboid_game
  ```

- Windows (MinGW/MSYS2/Cygwin):

```
./zomboid_game.exe
```

- Windows (CMD/PowerShell):

```
.\zomboid_game.exe
```

## 6. 代码结构详解

- `include/` **目录**:
  - `ICharacter.hh`: 角色接口类定义。
  - `Item.hh`: 物品 `struct` 定义。
  - `Action.hh`: `ActionType` 枚举和 `ActionInfo` 结构体定义。
  - `Player.hh`, `Zombie.hh`, `Room.hh`, `Game.hh`: 对应类的头文件，包含类声明和成员函数原型。
- `src/` **目录**:
  - `Player.cpp`, `Zombie.cpp`, `Room.cpp`, `Game.cpp`: 对应类的成员函数实现。
  - `Item.cpp`, `Action.cpp`: (可能为空) 对应结构体的实现（如果不在头文件中）。
  - `main.cpp`: 包含 `main` 函数，作为程序的入口点，负责创建 `Game` 对象并调用其 `run` 方法。

## 7. 当前实现状态 (Checklist)

- [x] **房间系统**: 6 个互联房间，布局 3x2。
- [x] **敌人系统**: 4 种不同类型丧尸 (Walker, Runner, Bloater, Crawler)。
- [x] **物品系统**: 7 种可收集物品 (武器、护甲、消耗品)。
- [x] **胜利条件**: 到达 5 号房间。
- [x] **交互系统**: 数字菜单驱动。
- [x] **内存管理**: 使用 `std::unique_ptr`。
- [x] **游戏循环**: 完整的主循环，包含胜负判断。
- [x] **项目结构**: 多文件 ( `.hh` / `.cpp` ) 结构。
- [x] Bonus - **清屏**: 已实现。