**期末作业**

付文君 2021141090183

# 一、CMMI 成熟度模型简述

CMMI（Capability Maturity Model Integration）是一个用于评估与改进软件开发过程的成熟度模型，由美国卡内基·梅隆大学的软件工程研究所（SEI）开发，主要用于提升组织在开发、获取和维护软件系统方面的能力。

CMMI 的成熟度分为 **五个等级**，每一级代表了组织在过程管理方面的能力：

| **等级** | **名称** | **描述** |
| --- | --- | --- |
| Level 1 | 初始级（Initial） | 缺乏稳定过程，依赖个人英雄主义，结果不可预测 |
| Level 2 | 可管理级（Managed） | 建立基本的项目管理制度，可控制项目进度与质量 |
| Level 3 | 已定义级（Defined） | 建立组织级标准过程，项目遵循组织级流程执行 |
| Level 4 | 量化管理级（Quantitatively Managed） | 使用量化指标来监控、控制软件过程 |
| Level 5 | 优化级（Optimizing） | 持续改进过程，使用反馈和创新机制优化软件质量与效率 |

CMMI 提供了过程改进的指导原则，有助于团队从“无序混乱”走向“有计划、有度量、有反馈”的可持续开发。

# 二、当前项目的软件过程成熟度评估

该项目重点聚焦于 Unity引擎下使用C#实现战斗系统功能，涵盖以下内容：

1. 脚本控制root motion根运动使玩家移动，使用混合树完成人物待机、奔跑、快跑的平滑切换

2. 脚本控制摄像机平滑移动，使用过Cinemachine制作第三人称控制器

3. 使用InputSystem接收玩家输入，并令其管理脚本继承单例模式

4. 使用事件委托、泛型创建事件管理器，对游戏中触发的事件如翻墙、生成伤害、触发处决等进行管理

5. 创建对象池，通过对象池对角色的音效进行控制，如脚步声、攻击音效、格挡音效和受伤音效

6. 设计战斗系统，创建Scriptable Object存储角色招式，并使用List存储组合技及敌人格挡招式

这些技术细节体现出我在**战斗系统设计、动画控制、事件驱动架构、资源优化** 等方面有较好的落地能力。

我们可以从以下几个关键维度来评估该项目在 CMMI 中所处的等级：

| **维度** | **描述** | **评估** |
| --- | --- | --- |
| 过程文档 | 是否有统一流程、文档标准 | ❌ 缺乏过程文档，开发流程依赖个人经验 |
| 项目管理 | 是否有需求管理、版本控制、任务分解等 | ⚠️ 开发任务未提及计划管理流程 |
| 过程复用 | 是否有统一的模板/模块复用 | ✅ 存在事件管理器、对象池等模块复用 |
| 质量控制 | 是否有测试、错误跟踪与修复机制 | ❌ 未提及测试计划或缺陷管理工具 |
| 数据度量 | 是否采集关键指标用于分析 | ❌ 无性能统计、输入延迟、帧率等数据指标采集 |
| 持续改进 | 是否定期回顾与优化系统架构 | ⚠️ 体现出技术成长，但未形成组织化优化过程 |

### ✳️ 结论：****该项目符合 CMMI Level 2 的特征****

项目具备一定的管理流程（使用单例、对象池、事件系统等组件化思想），但整体缺乏统一的标准、质量保证机制、组织级过程复用与量化分析，处于 **“可管理级”**。

# 三、基于当前成熟度的过程改进计划

该阶段的核心是：**组织建立标准化的软件过程，所有项目遵循这些流程实施开发。**

### 1. ****开发流程标准化****

| **改进项** | **建议** |
| --- | --- |
| 制定开发流程 | 建立从需求、设计、开发、测试到发布的流程图与说明文档 |
| 任务拆解模板 | 使用看板工具（如 Trello、Jira）将开发任务进行模块化拆分 |
| PR 审查流程 | 建立统一的代码提交、评审与合并流程（Git Flow） |

### 2. ****文档制度建设****

| **改进项** | **建议** |
| --- | --- |
| 设计文档 | 编写系统架构说明、模块职责划分、数据流图等 |
| 接口说明 | 所有公共组件如事件系统、对象池需有接口文档 |
| 使用手册 | 编写开发者指南，说明战斗系统如何接入、测试、扩展 |

### 3. ****代码与模块复用标准****

| **改进项** | **建议** |
| --- | --- |
| 模块封装 | 抽象事件系统、输入系统、对象池为独立组件，提供统一 API |
| 命名规范 | 建立强制的命名风格文档，统一类名、字段、变量、注释 |
| 代码模板 | 提供常用类模板，如 MonoBehaviour 基类、事件监听模板等 |

### 4. ****质量管理机制****

| **改进项** | **建议** |
| --- | --- |
| 单元测试 | 针对招式组合、输入检测等核心逻辑编写测试用例 |
| 自动化测试 | 可使用 Unity Test Framework 做基础验证 |
| Bug 记录与回溯 | 使用 Excel 或 issue 系统记录问题、复现步骤与处理人 |
| 性能分析 | 集成 Unity Profiler 采集帧率、内存、GC 等性能指标 |

### 5. ****量化指标采集（向 Level 4 靠拢）****

| **指标** | **说明** |
| --- | --- |
| 输入响应时间 | 玩家按键到动作反馈所耗时间 |
| FPS | 平均帧率、最低帧率 |
| GC Alloc | 每次战斗产生的 GC 分配大小 |
| 模块复用率 | 项目中通用组件的复用频率（如事件系统） |

## 四、过程改进计划时间表（示例）

| **周期** | **任务** | **负责人** | **输出物** |
| --- | --- | --- | --- |
| 第1周 | 梳理开发流程，制定流程图 | 技术负责人 | 研发流程文档 |
| 第2周 | 编写模块文档（事件系统、对象池） | 各模块作者 | Markdown 格式接口说明 |
| 第3周 | 引入 PR 代码审查制度，统一提交规范 | 项目经理 | Git PR 流程说明 |
| 第4-5周 | 针对核心逻辑编写测试用例 | QA + 开发 | 单元测试文件 |
| 第6周 | 开展第一次代码回顾会议 | 全员 | 回顾报告 |