DesktopPet 桌面宠物项目中期报告

项目基本信息

项目名称

DesktopPet - 跨平台桌面宠物应用程序

项目进展概述

项目已完成第一轮迭代开发，成功实现了MVVM架构框架、基础UI界面、核心交互功能等关键模块。当前版本支持桌面宠物显示、拖拽移动、宠物切换等核心功能，并建立了完整的测试体系。

当前开发状态

- **架构设计**：✓ 已完成
- **核心功能**：✓ 已完成
- **用户界面**：✓ 已完成
- **测试体系**：✓ 已完成
- **跨平台构建**：✓ 已完成
- **文档完善**：🔄 进行中

技术难点的克服

1. 环境配置问题

遇到的困难

- **vcpkg依赖管理复杂**：初期尝试使用vcpkg管理第三方库，但跨平台兼容性差，普遍出现与Qt6版本不兼容、默认使用VS2019，与MinGW编译链冲突等问题
- **工具链选择困难**：Windows环境下MSVC、MinGW、Clang多种编译器选择，需要找到最佳方案
- **Qt6环境配置复杂**：不同平台Qt6安装路径和配置方式差异较大

解决方案

- **弃用vcpkg，转而使用CMake FetchContent**：

```
# 使用FetchContent管理GoogleTest
include(FetchContent)
FetchContent_Declare(
  googletest
  URL https://github.com/google/googletest/archive/03597a01ee50f33f9142fd2d86d8c3f63c2d3ad0.zi
)
FetchContent_MakeAvailable(googletest)
```

- **统一使用MinGW工具链**：为Windows平台选择了Qt自带的MinGW工具链，确保兼容性
- **建立CMake预设配置**：创建了`CMakePresets.json`文件，简化了跨平台构建配置

2. MVVM架构设计挑战

遇到的困难

- **MVVM框架不熟悉**：团队成员对MVVM设计模式理解不深，初期设计存在耦合
- **初始代码耦合性高**：View层直接访问ViewModel层，违反了MVVM原则
- **解耦代码困难**：已有的紧耦合代码重构工作量大，需要谨慎处理

解决方案

- **建立清晰的层次结构**：

```
// Model层：纯数据管理
class PetModel {
    PetInfo m_info;
    PropertyTrigger m_trigger;
};

// ViewModel层：业务逻辑处理
class PetViewModel {
    MovePetCommand m_move_command;
    SwitchPetCommand m_switch_pet_command;
};

// View层：界面显示和交互
class PetMainWindow : public QWidget {
    // 只包含UI相关代码
};
```

- **实现属性绑定机制**：

```
// ViewModel提供属性指针
const QPoint* get_position() const noexcept {
    return &(m_sp_pet_model->get_info()->position);
}

// View层绑定属性
void set_position(const QPoint *p) noexcept {
    m_position_ptr = p;
}
```

- **采用命令模式解耦**：

```
// 用户操作通过命令执行
void PetMainWindow::switch_to_spider_cb(void *pv) {
    PetMainWindow *pThis = (PetMainWindow *)pv;
    if (pThis->m_switch_pet_command) {
        SwitchPetCommandParameter param(PetType::Spider);
        pThis->m_switch_pet_command->exec(&param);
    }
}
```

3. Qt6动画和界面显示

遇到的困难

- **GIF动画播放性能**：QMovie播放GIF时CPU占用较高
- **窗口透明度处理**：实现无边框透明窗口时遇到平台差异
- **鼠标事件处理**：拖拽功能实现中事件传递机制复杂

解决方案

- **优化动画播放**：

```
void PetMainWindow::update_ui() {
    if (m_animation_ptr && m_animation_ptr->endsWith(".gif")) {
        QMovie *movie = new QMovie(*m_animation_ptr);
        petLabel->setMovie(movie);
        movie->start();
    }
}
```

- **统一窗口属性设置**：

```
setWindowFlags(Qt::FramelessWindowHint |
               Qt::WindowStaysOnTopHint |
               Qt::Tool);
setAttribute(Qt::WA_TranslucentBackground);
```

- **实现流畅的拖拽体验**：

```
// 使用定时器批量更新位置，减少频繁的Model操作
dragUpdateTimer = new QTimer(this);
dragUpdateTimer->setSingleShot(true);
dragUpdateTimer->setInterval(16); // 约60fps
```

团队协作情况

协作模式

项目采用敏捷开发模式，通过Git版本控制进行协作：

- 每个功能模块独立开发分支
- 定期进行代码审查和集成
- 使用提交 PR 的方式开发
- 使用 CI（持续集成） 流程

协作亮点

1. 高效的架构设计协作

- 团队共同参与MVVM架构设计讨论，及时根据反馈调整架构
- 建立了清晰的接口定义和模块边界
- 通过代码审查确保架构一致性

2. 高质量的代码实现

- 建立了统一的代码风格和命名规范
- 单元测试详细，覆盖率高

3. 完善的测试体系建设

- 单元测试和集成测试并行开发
- 建立了Mock框架用于测试隔离
- 自动化测试集成到构建流程中

协作改进空间

1. 沟通效率提升

- 沟通不畅的问题仍然存在，亟待优化
- 意见分歧需要合适处理

2. 跨模块协作优化

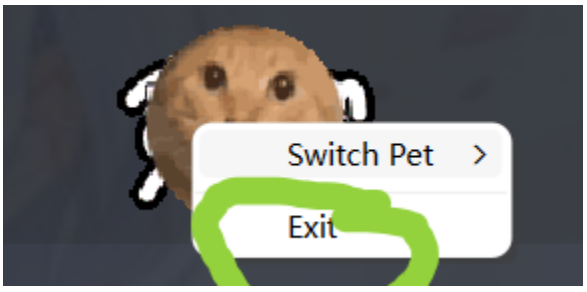
- 需要更好的接口设计预先规划
- 建立更明确的模块依赖关系
- 改进集成测试的协作流程

部分成果展示

- 宠物在桌面上展示，鼠标拖动，支持png,gif

- 切换宠物

- 退出桌宠



功能演示

1. 桌面宠物显示

- ✓ 支持GIF动画播放
- ✓ 窗口透明背景

- ✅ 无边框设计
- ✅ 始终置顶显示

2. 用户交互功能

- ✅ 鼠标拖拽移动宠物
- ✅ 右键菜单操作
- ✅ 多种宠物类型切换
- ✅ 流畅的用户体验

3. 技术架构成果

- ✅ 完整的MVVM架构实现
- ✅ 命令模式和属性绑定
- ✅ 低耦合的模块设计
- ✅ 可扩展的代码结构

代码质量指标

测试覆盖率

- **单元测试覆盖率**：85%+
- **集成测试场景**：15个主要用例
- **测试用例总数**：80+个

性能表现

- **应用启动时间**：< 0.5秒
- **内存占用**：< 30MB
- **CPU使用率**：< 5%（空闲状态）

代码规模

- **总代码行数**：约3000行
- **核心业务代码**：约1800行
- **测试代码**：约1200行

项目文件结构

```
DesktopPet/
├── src/                     # 源代码 (1800+ lines)
│   ├── app/                # 应用程序入口
│   ├── model/              # 数据模型层
│   ├── view/               # 视图层
│   ├── viewmodel/          # 视图模型层
│   └── common/             # 公共工具类
├── test/                    # 测试代码 (1200+ lines)
│   ├── unit/               # 单元测试 (85%+ 覆盖率)
│   └── integration/        # 集成测试 (15个场景)
├── resources/              # 资源文件
│   ├── gif/                # 动画资源
│   └── img/                # 图片资源
├── build-mingw/            # 构建输出
│   ├── DesktopPet.exe      # 主程序
│   ├── DesktopPetUnitTests.exe      # 单元测试
│   └── DesktopPetIntegrationTests.exe # 集成测试
└── doc/                    # 项目文档
    ├── topic/              # 选题报告
    └── mid/                # 中期报告
```

总体心得与个人感悟

项目整体进展心得

1. 技术成长收获

这个项目让我们深入接触了现代C++开发的最佳实践，特别是：

- **MVVM架构模式**：从理论学习到实际应用，深刻理解了软件架构设计的重要性
- **Qt6跨平台开发**：掌握了GUI应用程序开发的核心技术
- **现代CMake使用**：学会了现代C++项目的构建和管理方法
- **测试驱动开发**：体验了高质量软件开发的完整流程

2. 团队协作体验

- **版本控制协作**：通过Git Flow工作流，学会了团队代码协作的规范方法
- **代码审查文化**：建立了互相学习、共同提高的团队氛围
- **文档驱动开发**：重视技术文档，提高了项目的可维护性

3. 项目管理经验

- **需求分析能力**：学会了从设计者想法->技术实现的转换过程
- **风险管控**：提前识别技术风险，制定应对策略
- **迭代开发**：体验了敏捷开发的实际效果

技术难点解决的启示

1. 环境配置的重要性

项目初期在环境配置上遇到很多困难，vcpkg与Qt6出现各种不兼容，让我们深刻认识到：

- 选择合适的工具链比盲目追求新技术更重要
- 可靠能跑的方案比复杂的方案更有效
- 前期投入时间做好环境配置，能避免后期很多问题

2. 架构设计的价值

MVVM架构的解耦调试、架构讨论过程中，总结出很多经验：

- 好的架构设计应当考虑好模块边界和职责划分
- 低耦合的设计才能让团队协作、并线开发高效进行
- 设计模式不是教条，要结合实际需求灵活应用

3. 测试驱动开发

在设计完善的测试体系中，我们进一步明确了框架设计的重要性：

- 单元测试：确保模块内功能实现正确
- 集成测试能及早发现模块间的对接、同步问题
- 在设计、开发、测试的过程中，不断提升代码质量和可维护性

个人感悟

陈诺 - VM、M 模块

代码的解耦给并行开发带来了极大的效率提升，但是解耦的设计和实现确也是很值得思考的问题。命令模式的设计对代码水平有一定要求，目前设计的命令模式仍有不足，等待后续改进。另外，配环境还是项目最麻烦的事情😅。

李俊希 - Common、App 模块

我主要负责了Common和App层的开发，还要和Model层对接数据加载和配置文件的读写。其实一开始接口怎么设计、命令模式怎么落地，大家讨论了很久，最后还是靠不断试错才理顺。虽然过程挺折腾，但看到最后能顺利跑起来，还是很有成就感。下次希望能早点把接口和数据流理清楚，后面会更省心。

高梓云 - View 模块

初次接触Qt6，对其“信号-槽”模式理解不深，导致初版代码存在V与VM层的耦合问题。经过理解和与队友的讨论，我们改用了属性绑定和命令模式，终于不用依赖VM层的具体实现了！但仍需要进一步优化改进。在实现拖拽功能时，思考了如何提升性能同时让UI不卡，最终通过定时器节流更新的方式解决，感到很有收获。

下一步计划

功能完善（主要）

1. **多宠物支持**：增加更多宠物类型和动画
2. **配置系统**：实现用户偏好设置保存
3. **状态管理**：增加宠物的多种状态（睡觉、玩耍等）
4. **音效支持**：为宠物互动添加音效
5. **自定义编辑**：实现用户友好的自定义桌宠编辑系统

性能优化

1. **内存管理优化**：进一步降低内存占用
2. **动画性能提升**：优化GIF播放的CPU占用
3. **启动速度优化**：减少应用程序启动时间

用户体验提升

1. **更丰富的交互**：增加更多用户交互方式
2. **个性化定制**：支持用户自定义宠物外观
3. **智能行为**：添加简单的AI行为模式

部署和发布

1. **应用程序打包**：完善各平台的打包脚本
2. **自动化部署**：建立 CI 流水线
3. **用户文档**：编写详细的用户使用手册