**迭代计划（简化版）**

　　　　　　　　　　　　　制定日期：

|  |  |  |  |
| --- | --- | --- | --- |
| 组号 | 15 | 项目名称 | MY PC Logo |
| 迭代名称 | 界面原型迭代 | 计划起止日期 | 2020.9.14 – 2020.10.5 |
| 任务、进度安排和人员分配：   |  |  |  |  | | --- | --- | --- | --- | | No | 任务 | 起止日期 | 人员 | | 1 | 讨论制定迭代计划 | 2020.9.14 – 2020.9.14 | 所有人 | | 2 | 调研市面上已有产品，学习已有经验，并思考可以创新的方面 | 2020.9.15 – 2020.9.16 | 所有人 | | 3 | 初步选择语言、工具与框架，初步设定需求，基于此了解项目所需技术 | 2020.9.17 – 2020.9.18 | 所有人 | | 4 | 拆分所需样例界面，进行初步设计，奠定主题样式 | 2020.9.19 – 2020.9.20 | 吴侃真、邹立凯 | | 5 | 完成《vision》文档1-3部分 | 2020.9.21 – 2020.9.22 | 邹立凯 | | 6 | 完成《vision》文档4-5部分 | 2020.9.21 – 2020.9.22 | 吴侃真 | | 7 | 完成《vision》文档6-8部分 | 2020.9.21 – 2020.9.22 | 毛彦凯 | | 8 | 完成《vision》文档9-10部分 | 2020.9.21 – 2020.9.22 | 王执 | | 9 | 整合《vision》文档，修改 | 2020.9.23 – 2020.9.24 | 所有人 | | 10 | 学习qml，用于绘制界面原型 | 2020.9.24 – 2020.9.25 | 王执、邹立凯、吴侃真 | | 11 | 与老师讨论后调整项目方向，讨论修改项目设计 | 2020.9.28 – 2020.9.29 | 所有人 | | 12 | 第二次修改《vision》文档 | 2020.9.30 – 2020.9.30 | 毛彦凯 | | 13 | 用adobe illustrator绘制主界面原型的初稿 | 2020.9.30 – 2020.9.30 | 邹立凯 | | 14 | 讨论、提出修改主界面初稿的意见 | 2020.10.1 – 2020.10.1 | 所有人 | | 15 | 编写《软件需求规约》文档，设计use case模型 | 2020.10.1 – 2020.10.2 | 毛彦凯 | | 16 | 修改主界面，绘制登录、设置、双人绘图等其他界面 | 2020.10.3 – 2020.10.4 | 邹立凯 | | 17 | 使用React完成前端的基本布局 | 2020.10.4 – 2020.10.5 | 吴侃真 | | 18 | 完成侧边栏与文件视图的编写 | 2020.10.6 – 2020.10.7 | 吴侃真 | | 19 | 内部评审，对第一次迭代进行总结 | 2020.10.6 – 2020.10.7 | 所有人 | | 20 | 编写迭代评估报告 | 2020.10.8 – 2020.10.9 | 毛彦凯 | | | | |
| 预期成果：  文档：   1. 界面原型迭代计划 2. Vision文档 3. 软件需求规约 4. 界面原型迭代评估报告 5. User-case模型   模型：   1. Adobe illustrator绘制的界面原型 | | | |
| 主要的风险和应对方案：   1. 技术风险  * 美观、有良好交互性的UI绘制及turtle绘制图形的展现 * 低延迟的联机通信 * 类似IDE的语法检查与智能提示   应对方案：可参考市面上同类产品与相关设计书籍教程对UI进行设计；深入学习网络IO与并发的各种知识，尝试不同方案选取最优；实时智能语法可以参考编译原理中学到的知识。解耦各个功能模块，即使某个技术无法实现，也不会影响到主体功能的使用。   1. 架构风险  * 搭建低延迟，高并发的联机架构 * 具有高绘图性能、执行稳定的单机架构   应对方案：针对联机架构，抛弃笨重的Spring后端，使用Go语言搭建轻量且具有高并发性能的服务器；针对单机架构，深入了解不同系统的绘图API，以获得比封装函数更好的性能。   1. 进度风险  * 其余课程难度大、抢占时间，可能影响本课程项目进度 * 开发过程中遇到难题使得进度推迟，打乱预订计划   应对方案：对整个项目需要完成的任务进行归类，设定优先级，先执行优先级高的任务，在时间非常紧张的情况下只能舍弃优先级低的任务，如进阶需求、详细文档等。   1. 需求风险  * 项目中期需求变更 * 进阶需求实现难度较大 * 由于项目经验不足，随着项目的推进才发现组内预先设定的需求不合理，需要修改   应对方案：进行良好的架构设计，尽量提高代码的可读性、可维护性，使得在需求发生变动时，能够较为快速地在原有项目基础上进行修改。   1. 集成风险  * 单机联机模块对接可能产生意想不到的问题   应对方案：对模块解耦时设计良好接口，并严格遵循接口约定不轻易改变。准备多套集成方案，确保一种方案有良好的模块兼容性。 | | | |
|  | | | |