JavaA_project经验指南

JAVA Suki swing Obsolete JAVAFX Advanced

（关于英文文档我自己都看不太明白，而且英语水平完全依赖于喂给GPT的中文水平，就不给大家写英文了）

前言：

这是一份个人向的javaA project指南，结合我个人写这个项目的经验+当sa时候的经验，整体内容偏介绍引入和经验分享，仅供参考。（如果大家对具体技术板块有疑惑，可以给我提issue或者私聊我，我后续会一直完善并维护这个仓库🏝️）

（目前的经验来源于2023fall及以前学长的祖传经验，后续javaA会不会改革还不确定，但是我尽量用白话把许多经验向的内容给大家找补一下信息差，会带有一点主观性🙌）

这里推荐另一份由froster神写的博客指南，偏技术性，讲了很多相对更先进的内容🤩

java祖传的project框架——MVC（Model-View-Controller）

这个框架本身在工业界使用的就比较多，比如java平台的Spring就是一个非常强大的MVC架构（这里安利一下陶老师开的计算机系统设计基础课，俗称java2，算是计系里比较好水的选修学分，同时还能学不少java深层次的内容（2024Spring除外😤，关于我在一门叫java2的课程上写cpp架构的Qt和学画图，这就是深大软院的实力🤫）），很多web和桌面应用也是采用MVC模式。这个架构本身并不复杂，再复杂一点估计也没法拿来在通识课上用了。

组件

当然，这是一个偏经验向的指南，我三言两句肯定是讲不明白MVC的，仅仅作为对小白选手的一种引入（虽然MVC这个概念其实蛮虚的🧐）

Model（模型）

模型组件负责处理应用程序的数据逻辑。（俗称后端）它通常负责从数据库获取数据、进行数据的处理和存储，以及封装业务逻辑。模型是独立于用户界面的，因此改变应用程序的工作流程不需要改变模型。后端逻辑的通常是debug的灾区（以2023fall的斗兽棋为例：Model要负责棋子的移动逻辑，棋子互吃逻辑，特殊地形逻辑。2024Spring《2048》的后端就得负责格子的移动逻辑，合并逻辑（和前几学期对比来看，这个算很简单的了👍））

功能

- **数据访问**：模型组件包含从数据库中检索和存储数据的代码。（javaA一般可能不涉及这个）
- **数据处理**：执行数据验证、计算和逻辑判断。（核心）
- **状态维护**：存储和管理应用的状态。

View（视图）

视图负责展示数据或者应用程序的输出（俗称前端👀）。它从模型接收数据，并将这些数据转换成适合用户阅读和交互的格式。视图只负责显示数据，不执行任何业务逻辑。（简单来说，前端只负责和用户的交互，而且也只体现在视图上，具体功能的正确与否还要看后端的实现逻辑有没有bug）

功能

- **数据显示**：将数据以图形界面形式展示给用户。
- **用户交互**：接受用户的输入（如按钮点击、文本输入等）并将这些输入发送到控制器。

这里插一个题外话，有些班是讲了swing的，有些班可能没讲（比如我学的时候swing就没讲过），也有可能学swing的时间点已经比较靠后了，学完了再来写前端不太现实。而且swing本身学习成本不高（早该被淘汰的老古董了😈），如果大家时间成本比较高，想冲击一个更高的上限，非常建议大家去学一下javaFx（小邱神给大家做了保姆级教程就在隔壁🥰）整体模式还是MVC没变，但是库里自带的组件都非常优雅（至少比swing这种一眼看上去就是10几年前的产物更让人舒适👍）

Controller（控制器/连接器）

控制器是一种模型和视图之间的中介，处理用户输入（对javaA来说最重要的事件处理其实就是鼠标的点击操作），调用模型进行数据处理（调用后端），然后选择视图进行数据展示（更新前端）。

功能

- **输入处理**：解析用户的输入。
- **业务逻辑调用**：根据用户的输入调用模型的适当功能。
- **视图选择**：选择用于显示数据的适当视图和更新视图。（这里是另一个比较难debug的点，比如想要做一些延时动画或者读档回退功能，在合适的时间点更新视图完全是不一样的效果，会产生非常多奇怪的bug🧐）

一般正常组队情况是一个人写Model，一个人写View，大多数情况下Controller还是由写Model的同学来完成，但是推荐两个人都得懂这部分，不然debug完全依赖于某一个人会非常痛苦。（还可能产生不愉快😒）

个人经验方面：

关于组队：

从我见过的血与泪，仇与恨，~~欺诈与无耻~~来看，如果两个相识并且互相信任的人组队，不太可能存在贡献比分成问题。首先，贡献比是很难通过coding量来量化的，coding难度和debug难度都没法具体衡量。

因此我给到的参考是：

- 一个人负责Model部分，一个人负责View部分，controller两个人都得懂。（按照以往的经验，后端部分的难度是肯定大于前端的，但是2024Spring的《2048》也许是个例外，最后卷的部分很有可能集中在前端上）
- 如果按照以上进行分工，就不必过多纠结贡献比问题，只要定期能完成对应的任务即可。（这个话题其实不太好开口，特别是对于朋友而言）

- 我不提倡两个人里面有一个人能力非常强，一路领跑，结果最后分数没拿满再要求调整贡献比。（除非另一个人心甘情愿被带🤫）
- 关于合作，这里强推大家使用git来合并代码（个人认为最简单的bonus之一，学习成本其实并不高👌），2023fall的时候就有小组前端和后端的代码进度没法统一，单纯靠相互发文件夹来合成代码，风险其实很高（这一点也不优雅😎）对于志向在计系的同学请一定要学会，目前如果受时间限制的话，不必要去学git的命令行，但是会用基础的功能即可（短期为了project的话可以选择成本更低的GitHub+github Desktop）（既然你都有GitHub账号了，那不妨申请个campus认证，万一恰好就发现了有copilot这么个东西呢🤫）
- 极端情况：对于随机分配的组队，到了15周队友才第一次见面，或者整个project期间联系不上队友，向老师反映情况（之前遇到这种情况真的让老师很头痛，很大概率这组会延迟答辩，分数打折扣➕折磨老师）
- （来自yeetone哥哥的忠告🤌）两个人里面至少要有一个核心头脑，能把MVC的每个部分都掌握，统领大局

关于项目进度：

据我的观察，一般会有三分之一的小组整个五一假期都在卷project，当然大多数人其实都是五一之后才开始着手项目，时间上来说完全足够（除非想要冲击16周的大课展示+bonus全满）

据2023fall的统计经验，80%的组基本上在12～13周还在学习阶段，进展大概只有30%

但是14周一过大部分组的进度都在60%往上了（我猜这里一定也有很多故事吧...为了计系课熬夜coding的米娜桑😮‍💨）

快慢与否与目标期望挂钩，project本来就是一个锻炼大家短期内爆发性学习的机会。

正常速度推进，大概会在13周结束的时候完成项目的基础操作流程（可以不包括存档和读档），无论bonus做的多或少，请至少留出一个周来作为改进和debug的容错，以及把尽可能的bug都排查一遍（大概率你debug的时间比你有效coding的时间更长）

关于15周/16周，我后续会提，不过建议大家都先把15周答辩看作最终时间💨

关于最后得分：

- 得分是基础分（80）+bonus（30/20），15周和16周的答辩分数上限不一样，但是对最终成绩影响并不大。16周答辩能拿满分和15周没有本质区别。但是冲击一下16周的大课展示其实是一个非常好的表现机会，对于申请SA（如果政策能回调😭），找学术导师等，都是对自己coding能力的有力证明。
- 基础分80:只要是正常的跟进项目进度，基础功能都是很容易拿到的分数，这80分就是对应这个游戏的最基础功能。而且SA在答辩的时候基本上都慈悲为怀，不会可以为难大家或者给大家挑刺。
- bonus：据我观察，其实bonus能拿满的小组也没有那么多，大多数小组都会挑一些简单的bonus来实现。这里我推荐几个比较简单的bonus——git，UI美术（音效，图片替换），背景音乐。除此之外就是“回溯功能”，一般能做出来回溯功能，大概率存档功能也没问题，虽然难度比前几个略高，但是性价比也不错。除此之外，bonus的给分比较灵活，鼓励大家自由发挥（比如写个笨蛋AI什么的🤓——说句实在话，其实现场答辩的时候根本没法去验证你写的AI能聪明到什么程度，这里比较考验说话的艺术👀）
- 对于javaA这门课来说，大部分同学其实作业➕平时分都能拿满，功利一点来看的话project其实对最后的成绩影响不大（前提是基础功能全部通过），溢出的bonus不能直接填补到总评里。如果只是想javaA这门课拿高分的话，把时间拿去复习期末的八股知识比卷bonus其实更有性价比（如果想锻炼自己的coding ability➕想看看自己能否适应计系的课业压力，我建议大家至少做到不留遗憾，毕竟计系很可能一学期要做2个甚至3个比javaA更难的project😈）

- 答辩的时候，哪怕控制台报错了，直接假装没看见（这里比较考验你的心理素质🤫，也考验SA的心理素质🙈，我一般都假装没看见，防止大家都尴尬。如果SA比较严苛，控制台报错是会扣分的）其次，答辩的时候就是直接对着打分表一点点验证，做了什么就展示什么，不会刻意去为难大家的代码（我反正是不会刻意去把别人的程序跑爆红的，毕竟课程project基本上都是XX山居多，健壮性...🤫）记得可以给自己做了哪些bonus列个表，防止遗漏👍

提问板块

（本来是没打算写技术向的指南的）从2024Spring开始累积吧，问的比较多的问题我放在这里，后续更新根据互助群和issue汇总👀

1.序列化/反序列化

对于第一次接触的朋友，不用太纠结这两个定义的细节，把语境放到这个project本身，你核心要做的其实就两件事：

- 把棋盘信息按照某种特定的形式或规则写入到文件里（将程序中的对象和数据结构转换成一种格式）
- 这里的文件格式可以选择txt（大多数人会选择这个）或者json（这里推荐Gson 或 Jackson这样的现成库，可以节省很多工作量）
- 反序列化是序列化的逆过程，它将序列化的数据格式转换回原来的数据结构或对象。
- 存档功能其实就是序列化当前棋盘的所有属性，写入到文件里。
- 读档功能就是反序列化文件，把它重新还原回游戏的状态。

读档和存档是基本分的部分，我印象中分值占比不小，而且序列化做的好基本上可以顺带把“回溯”功能一并做了（bonus）回溯功能在打分的时候分为两种，只回溯一步和回溯到开始（分值不同），这里差异就在于你序列化的逻辑是只保存上一步的棋盘状态还是把所有的操作步骤全部保留。

既然谈序列化了，那我提一下serializeable这个接口吧（不知道大伙用不用得上）

```
import java.io.Serializable;
```

这个接口可以让类直接序列化，标记一个类的对象可以被 ObjectOutputStream 进行序列化，即可以被转换为字节流，这样就可以通过文件、数据库或网络等方式持久保存或传输。例如：我有一个Person类，调用这个接口后，所有Person的实例对象都可以被直接序列化。（可能这个才是真正意义上的序列化，project的要求估计只是让各位学会文件的读写就行了👀）

我帮你们答了🤫

javaA的project~~不查重！不查重！不查重！~~被查重的比例非常低，但是最后依旧会要求上传源代码（不提倡当代码之父/母到处去“借鉴”👀不然答辩的时候问你某个具体的功能如何实现，别让善良的老师和SA们尴尬）

AI工具的话，不会限制大家使用，但也别全依靠这个（不然真的没办法自己debug），我个人推荐cursor和copilot（~~不知道这算不算带坏学弟学妹~~不管了我也是学长教的😈），前者可以帮你debug，后者可以帮你减少很多重复性的coding（让推倒重来的沉没成本变低，不至于心态溃散）

project的bug就尽量少向老师或者SA发难了，每个人的写法都不一样，bug也千奇百怪（还不如让GPT给你找bug，说不定还能学不少细节内容）

Other

也许，我说也许……

你会在未来的一个月体会到连续几天通宵coding的爽感，或者说写project会让你有种油然的正反馈感，也或者说“痛并快乐着”🥳如果你对这样的生活怀有着一丝憧憬（非XXXm👐）那么很好，赛博民工的标准属性已经练成，你也许会喜欢计系生活🤞（劝人学计，今日功德跌停🤓）

私聊debug问问题或者未来对计系有憧憬，可以直接QQ上找我玩，联系方式在GitHub个人主页上

（不知道为啥某些学弟学妹会对我有奇怪的印象😭，再也不玩抽象了，我真的不二次元也不现充，或者说🤔有人觉得我外在很虚浮？whatever，以上内容都是发自内心的大白话经验，我也不在乎“提问的智慧”，大家有什么问题直接找我就好，我尽我所能给大家解答，真诚希望大家能享受这次coding🌈

祝愿大家project渡劫顺利👊

要上了吗...🙌

会赢的米娜桑😎