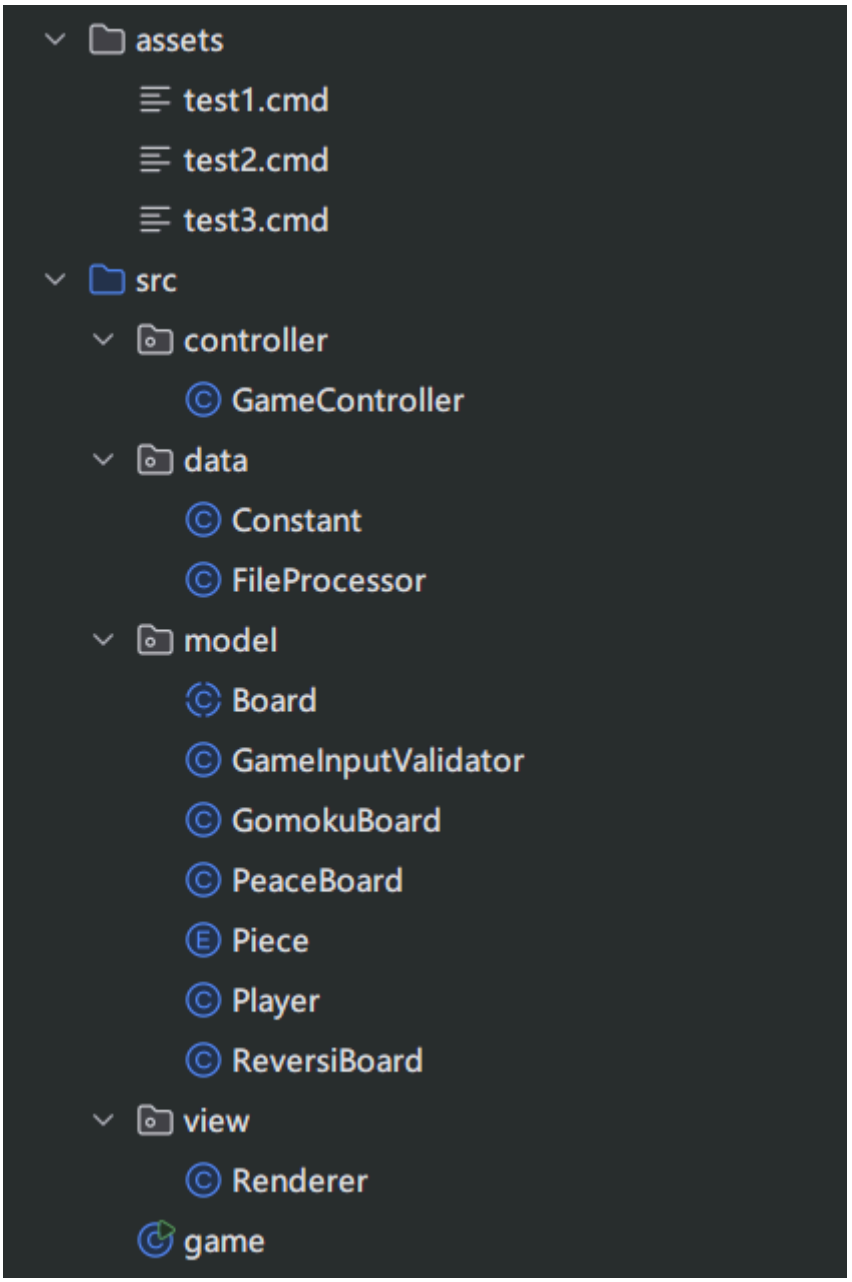
俞楚凡 OOP Lab6

Lab6是一次极大的变动，彻底重构了原有代码，降低了耦合性，增强了模块化性，可拓展性，提升了代码质量。

之所以被叫做lab5refactor是因为本次lab是在原本对于lab5的重构之上更改的。

代码结构



项目采用了**MVC**设计模式，即Model - View - Controller。其中，controller部分包含了各种方法的调度控制模式，`GameController`是一个游戏实例的各种数据更新，逻辑判断的调度。`data`是持久层，包含了常量类`Constant`和读取外部回放文件的`FileProcessor`。本次demo模式的三个回放文件存放在资源文件夹`assets`下。`model`为业务层，存放各种业务逻辑相关的类。其中，`Board`为抽象的棋盘类，`PeaceBoard`,`ReversiBoard`和`GomokuBoard`继承了`Board`类。`GameInputValidator`提供了输入合法性判断的逻辑，`Piece`是棋子类型的枚举类，`Player`是玩家类。`view`是视图层，负责画面的渲染。`game`是主类。

项目做到了充分的解耦，各层之间没有耦合，模块化，参数化管理，极少的全局变量单例。

下面简述一下这次Lab的要求的实现思路。

首先，在抽象类`Board`中添加了棋盘长宽，统一了获得棋盘长宽的接口`getRow()`和`getCol()`，减少了魔法数字的出现。但是由于棋盘大小的不同，在渲染层`Renderer`中需要特判`Gomoku`模式的棋盘。

障碍的添加非常简单。由于其不影响原来获胜条件的判定，直接在枚举类`Piece`中添加两种障碍即可。

炸弹只需要在落下的时候判定一下是否还有剩余炸弹数量以及该位置是否是对方的子即可。是的话可以直接替换。

关于**demo**模式

playback在`GameInputValidator`中添加了检验。本次依旧没有添加命令行模式，基本上是通过if else的多分支结构来依次检验命令合法性。合法的命令返回一个唯一标识ID和参数signal，对于落子和炸弹，signal是输入本身。对于playback，参数是空格后的文件名。

程序将在`../assets`下查找指定文件并执行。效果与从键盘读入一样。在`scanner`中，由于`scanner.close()`会直接将标准输入流关闭，所以不能只使用唯一的scanner单例来控制输入流的切换。我的实现是分别定义了标准输入流`scannerSystemIn`和文件输入流`scannerFile`，并且使用一个统一的`scanner`，通过令`scanner = scannerSystemIn`和`scanner = scannerFile`来做切换。效果良好。

延迟一秒使用了`Thread.Sleep(1000)`。由于程序执行本身还需要时间，所以实际每一步需要的时间会大于一秒。呈现出来的效果会比视频慢一些。