

**2020年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 余涛 |
| 学号 | 1180300829 |
| 班号 | 1803008 |
| 电子邮件 | [1063695334@qq.com](mailto:1063695334@qq.com) |
| 手机号码 | 15586430583 |

**目录**

[1 实验目标概述 1](#_Toc29325498)

[2 实验环境配置 1](#_Toc29325499)

[3 实验过程 2](#_Toc29325500)

[3.1 Poetic Walks 2](#_Toc29325501)

[3.1.1 Get the code and prepare Git repository 2](#_Toc29325502)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc29325503)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc29325504)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc29325505)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc29325506)

[3.1.4 Problem 3: Implement generic Graph<L> 10](#_Toc29325507)

[3.1.4.1 Make the implementations generic 10](#_Toc29325508)

[3.1.4.2 Implement Graph.empty() 11](#_Toc29325509)

[3.1.5 Problem 4: Poetic walks 12](#_Toc29325510)

[3.1.5.1 Test GraphPoet 12](#_Toc29325511)

[3.1.5.2 Implement GraphPoet 13](#_Toc29325512)

[3.1.5.3 Graph poetry slam 14](#_Toc29325513)

[3.1.6 Before you’re done 15](#_Toc29325514)

[3.2 Re-implement the Social Network in Lab1 16](#_Toc29325515)

[3.2.1 FriendshipGraph类 16](#_Toc29325516)

[3.2.2 Person类 18](#_Toc29325517)

[3.2.3 客户端main() 18](#_Toc29325518)

[3.2.4 测试用例 19](#_Toc29325519)

[3.2.5 提交至Git仓库 20](#_Toc29325520)

[3.3 Playing Chess 21](#_Toc29325521)

[3.3.1 ADT设计/实现方案 21](#_Toc29325522)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 26](#_Toc29325523)

[3.3.3 ADT和主程序的测试方案 37](#_Toc29325524)

[4 实验进度记录 44](#_Toc29325525)

[5 实验过程中遇到的困难与解决途径 45](#_Toc29325526)

[6 实验过程中收获的经验、教训、感想 45](#_Toc29325527)

[6.1 实验过程中收获的经验和教训 45](#_Toc29325528)

[6.2 针对以下方面的感受 45](#_Toc29325529)

# 实验目标概述

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现ADT。具体来说：

针对给定的应用问题，从问题描述中识别所需的ADT；

设计ADT 规约（pre-condition、post-condition）并评估规约的质量；

根据ADT 的规约设计测试用例；

ADT的泛型化；

根据规约设计ADT 的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）

使用OOP 实现ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；

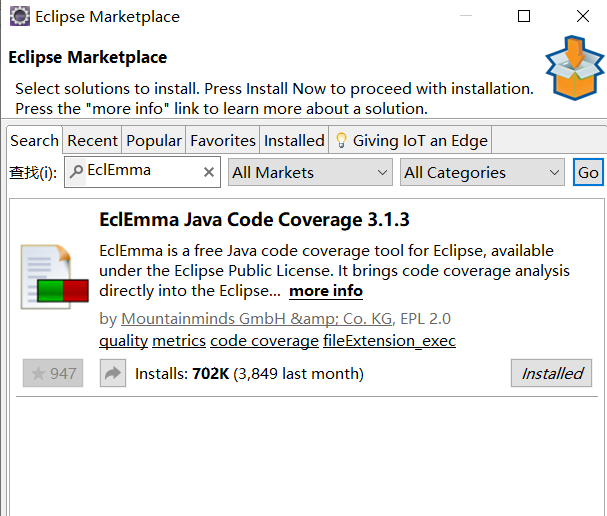
测试ADT 的实现并评估测试的覆盖度；

使用ADT 及其实现，为应用问题开发程序；

在测试代码中，能够写出testing strategy 并据此设计测试用例。

# 实验环境配置

本次实验需要在eclipse中配置EclEmma，经过查询了解到可以直接在eclipse中的市场中安装，在市场中搜索如图：



然后一步一步安装即可。

安装过程中没有出现异常。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/Lab2-1180300829>

# 实验过程

## Poetic Walks

本次实验给出了一个图接口，要求我们建立一个边图类、一个点图类分别继承自这个图接口，并且在里面实现一系列方法，并且实现抽象数据型，并用这个图的抽象数据型完成poet的工作。主要目的是练习ADT的规约设计和ADT的实现。

### Get the code and prepare Git repository

先得到网址

https://github.com/rainywang/Spring2020\_HITCS\_SC\_Lab2/tree/master/P1

然后在gitbash直接执行

git clone <https://github.com/rainywang/Spring2020_HITCS_SC_Lab2.git>下载工程文件

然后打开自己本次实验github库的网址，在F盘中执行：

git clone https://github.com/ComputerScienceHIT/Lab2-1180300829.git

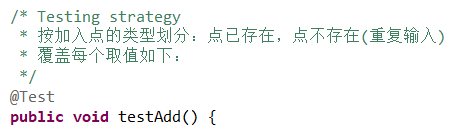
本地仓库建好

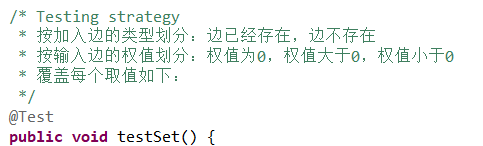


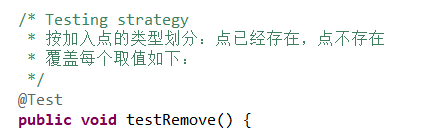
### Problem 1: Test Graph <String>

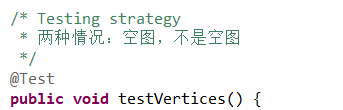
思路：这是针对Graph<String>设计相应的测试策略，主要针对里面的每个方法进行等价类划分的测试。

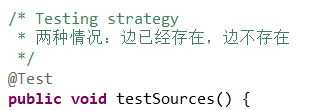
过程：如下图所示，为所有方法的测试策略：

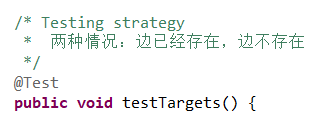












结果：在下面的junit总体中看。

检查覆盖率：



### Problem 2: Implement Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

#### Implement ConcreteEdgesGraph

**3.1.3.1.1 实现EDGE类：**

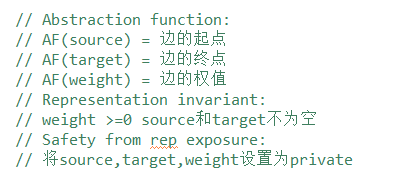
(1).EDGE中的字段应该包括边的长度，边的起点和终点，所以定义私有类型的这些变量如下图：



(2).在EDGE中需要实现的方法如下图所示：

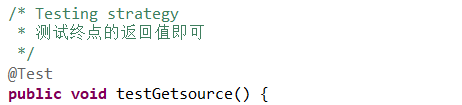
|  |  |
| --- | --- |
| Edge | 初始化构造方法，初始化新边的两个点和边的权值 |
| checkRep | 检查表示不变性,边不为空且权值大于等于0 |
| getsource | 返回边的一个点source |
| gettarget | 返回边的另外点target |
| getweight | 返回边的weight |
| toString | 返回一条边的字符串，形式为“起点->终点权重为xx” |

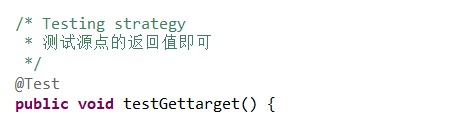
(3).AF,RI和Safety from rep exposure如下图：

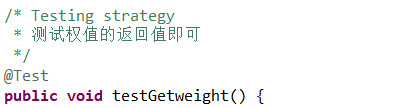


(4).测试策略：









**3.1.3.1.2 实现ConcreteEdgesGraph类：**

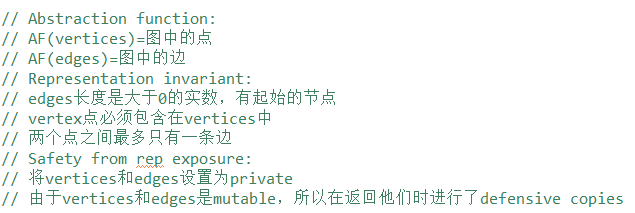
(1).ConcreteEdgesGraph的字段中应该包括顶点set表和边list表，定义私有类型的表如下图所示：



(2).在ConcreteEdgesGraph中需要实现的方法如下图所示：

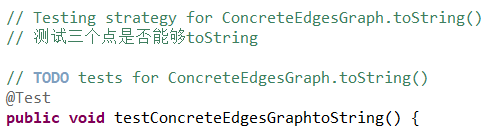
|  |  |
| --- | --- |
| ConcreteEdgesGraph | 构造方法 |
| checkRep | 检查表示不变性，edges长度是大于0的实数，有起始的节点 |
| add | 顶点不为空时，添加一个顶点进入点表中 |
| set | 输入source，target，weight，分别为边的起点、终点和权值。若权值为负，返回-1。若权值为正且新边已经存在，则除去原边并添加新边。若权值为正且新边不存在，则直接添加新边。若权值为0且新边已经存在，则出去原边。只要改变了原边权值，都返回原边权值，没有权值则返回0 |
| remove | 除去某个点及与它相邻的所有边。只需要遍历edges，寻找是否有边的起点或者终点为该点，直接删去即可，使用迭代器实现。 |
| vertices | 返回所有的点集 |
| sources | 输入一个终点，返回与它相连的所有边和起点构成的Map |
| targets | 输入一个起点，返回与它相连的所有边和终点构成为的Map |
| toString | 将整个图中所有点的指向转化为一条字符串输出 |

(3).AF,RI和Safety from rep exposure如下图：

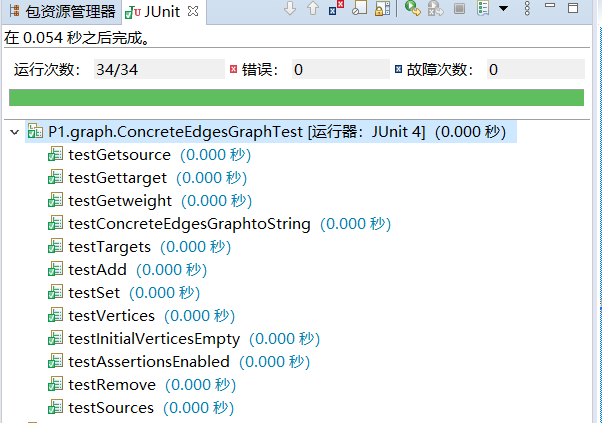


(4).测试策略：

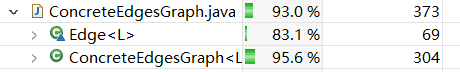
继承Graph的测试策略，只需要增加toString的测试即可：



测试结果：



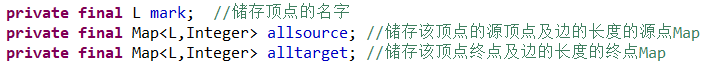
检查覆盖率



#### Implement ConcreteVerticesGraph

**3.1.3.2.1 实现Vertices类：**

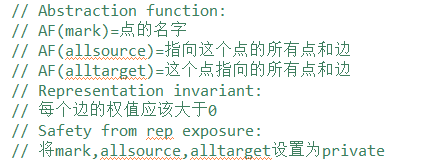
(1).Vertices的字段中应当包括点的名字，点的源点表Map，点的终点表Map，定义私有类型的表如下图所示：



(2).在Vertices需要实现的方法如下图所示：

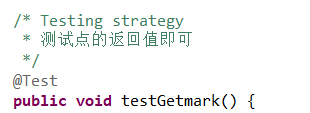
|  |  |
| --- | --- |
| Vertex | 初始化构造方法，用点的名字创建 |
| checkRep | 检查表示不变性，每个边的权值应该大于0 |
| getmark | 返回点的名字mark |
| getsource | 返回能到达该点的所有点和边构成的Map |
| gettarget | 返回某个点能到达的所有点和边构成的Map |
| addsource | 在源点Map中加入某源点，若weight不为0，则将其加入source中(若源点已存在，则更新其weight并返回原weight，不存在则直接构建新点并返回0)。若weight为0，则移除源点(不存在返回0，存在返回原weight) |
| addtarget | 在终点Map中加入某终点，若weight不为0，则将其加入target中(若终点已存在，则更新其weight并返回原weight，不存在则直接构建新点并返回0)。若weight为0，则移除终点(不存在返回0，存在返回原weight) |
| removesource | 在源点表中删除某起始点，并返回旧的边长 |
| removetarget | 在终点表中删除某终点，并返回旧的边长 |
| toString | 得到一个点的字符串表示 |

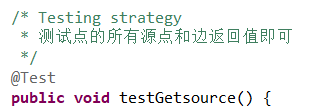
(3).AF,RI和Safety from rep exposure如下图：

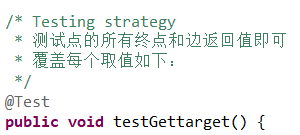


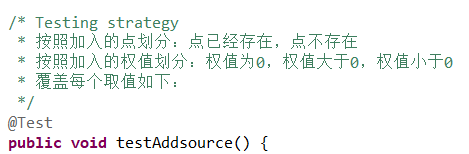
(4).测试策略：

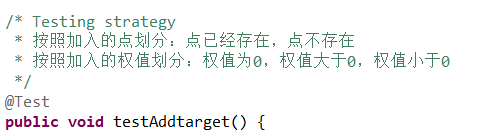


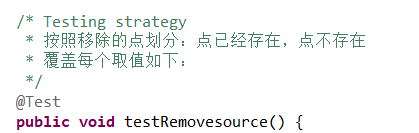


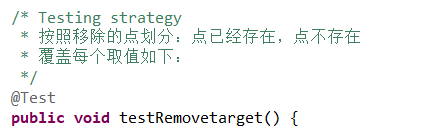












**3.1.3.2.2 实现ConcreteVerticesGraph类：**

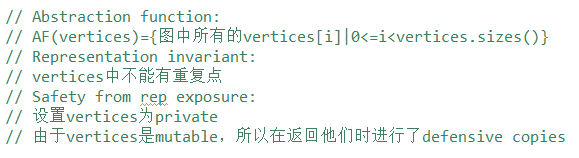
(1).ConcreteVerticesGraph的字段为Vertex构成的List，定义私有类型的表如下图所示：



(2).在ConcreteVerticesGraph需要实现的方法如下图所示：

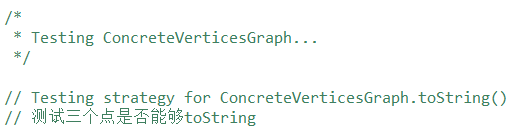
|  |  |
| --- | --- |
| ConcreteVerticesGraph | 构造方法 |
| checkRep | 检查表示不变性，vertices中不能有重复点 |
| add | 顶点不为空时，添加一个顶点进入点表中 |
| set | 输入source，target，weight，分别为边的起点、终点和权值。若权值为负，返回-1。若权值为正且新边已经存在，则除去原边并添加新边。若权值为正且新边不存在，则直接添加新边。若权值为0且新边已经存在，则出去原边。只要改变了原边权值，都返回原边权值，没有权值则返回0 |
| remove | 除去某个点及与它相邻的所有边。只需要遍历vertices，寻找是否有与待删除点相同的名字的点直接删去即可，如果名字不相同，则在该点的源点表和终点表中寻找删去即可，使用迭代器实现。 |
| vertices | 返回所有的点集 |
| sources | 输入一个终点，返回与它相连的所有边和起点构成的Map |
| targets | 输入一个起点，返回与它相连的所有边和终点构成为的Map |
| toString | 将整个图中所有点的指向转化为一条字符串输出 |

(3).AF,RI和Safety from rep exposure如下图：

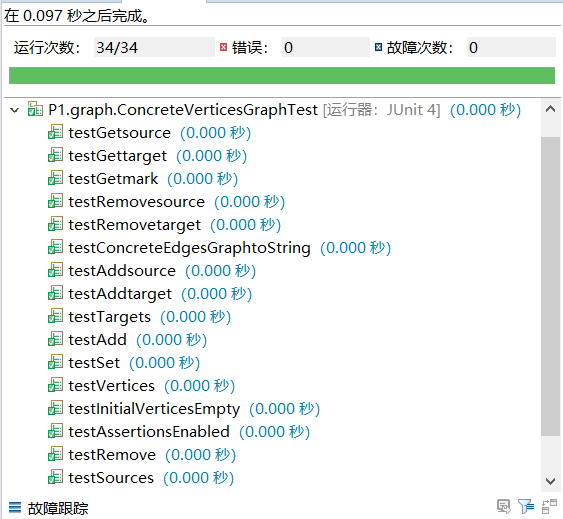


(4).测试策略：

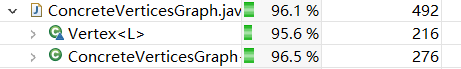
继承Graph的测试策略，只需要增加toString的测试即可：



测试结果：



检查覆盖率：

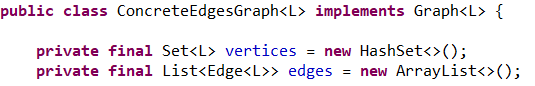


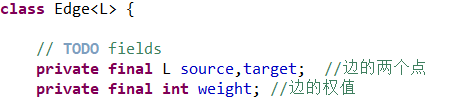
### Problem 3: Implement generic Graph<L>：

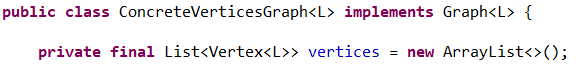
#### Make the implementations generic

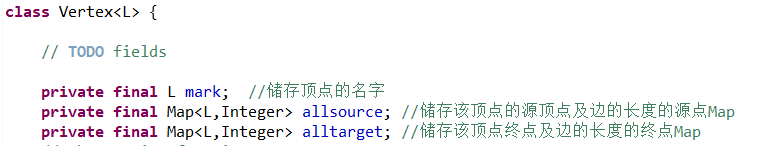
使用泛型实现即可：

在修改过程中通过eclipse提示的错误修改即可完成：





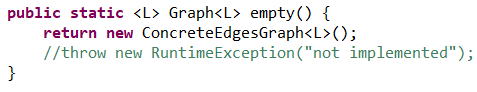




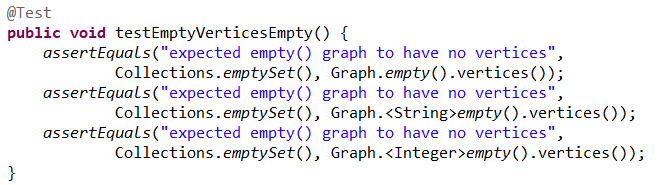
方法的修改同理

#### Implement Graph.empty()

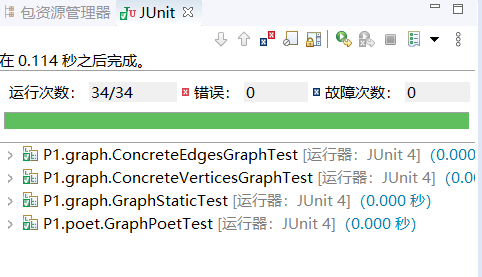
只需调用一个具体的实现：

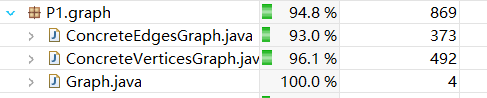


补充测试策略：



检查整体测试及测试覆盖率：



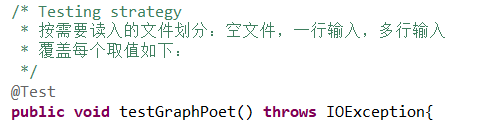


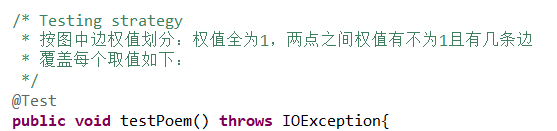
### Problem 4: Poetic walks

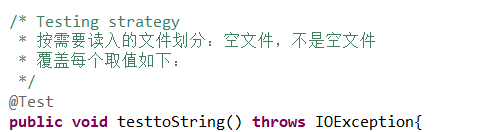
任务要求我们实现一个类，利用之前实现的图结构，能够将语料库转化为该种图结构，并且在图中搜索，完成对输入的诗句的句子进行扩充。

#### Test GraphPoet

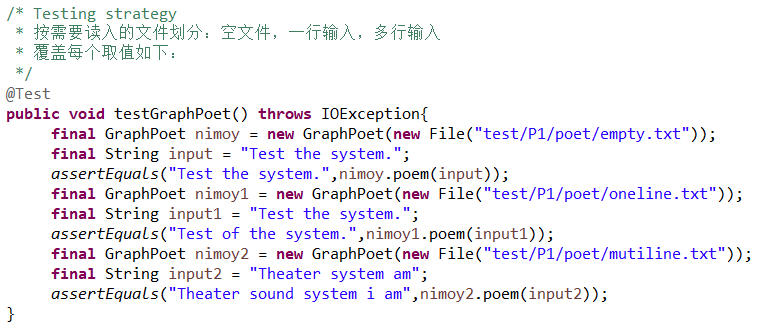
测试策略：

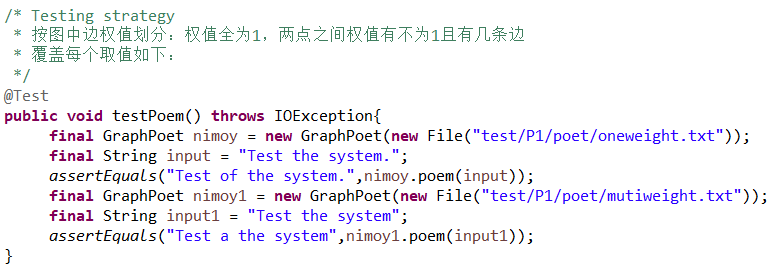


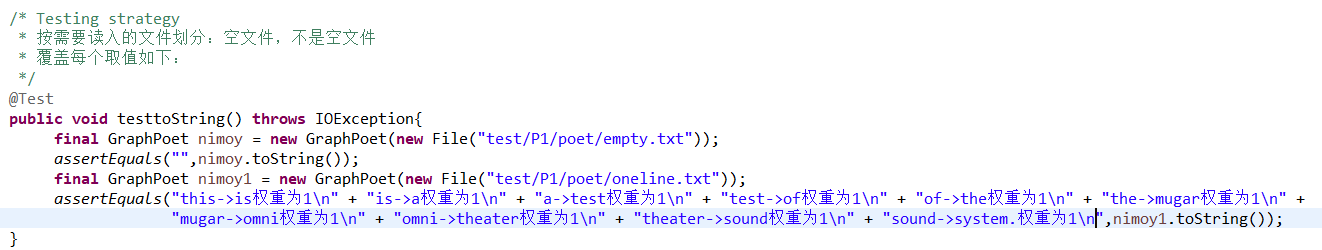




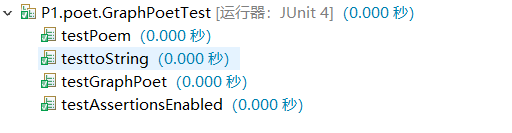
具体实现读入一系列满足要求的文件：







测试结果为：



测试覆盖率为：

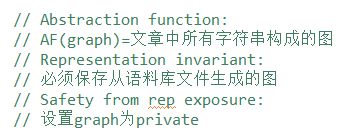


#### Implement GraphPoet

(1). GraphPoet的字段为String构成的Graph，定义私有类型的表如下图所示：



(2). AF,RI和Safety from rep exposure如下图：

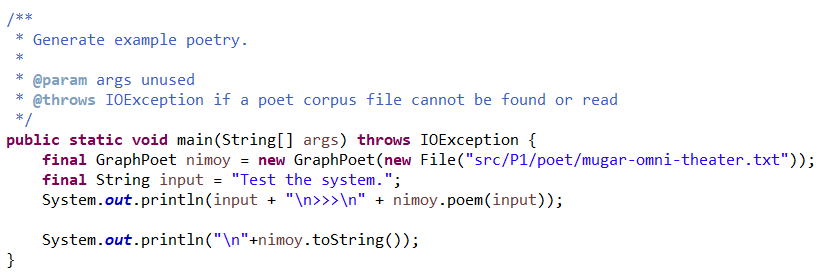


(3).在GraphPoet中需要实现的方法如下图所示：

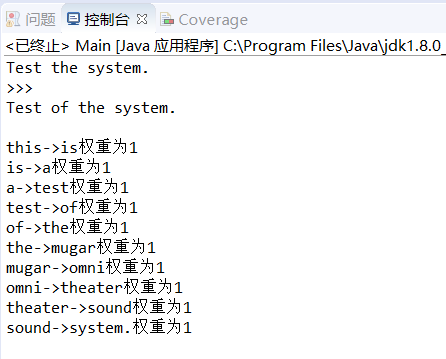
|  |  |
| --- | --- |
| GraphPoet | 输入文件的路径，一行一行读入，储存在List中，然后每次取相邻的元素，在图中添加新的边 |
| checkRep | 检查不变性，必须保存从语料库文件生成的图 |
| poem | 输入需要进行扩充的字符串，声明声明一个StringBuilder保存，每次读取一个词，当前词作为source，下一个词作为target，然后在garph中寻找source的终点表中是否有与target的源点表中相同的元素，并且找到权值最大的和的点加入source和target之间，具体实现为：  返回扩充后的字符串 |
| toString | 调用ConcreteEdgesGraph中的toString方法，将整个图中所有点的指向转化为一条字符串输出 |

#### Graph poetry slam

在提供代码的基础上增加一个toString的输出。



运行main函数如下输出：



### Before you’re done

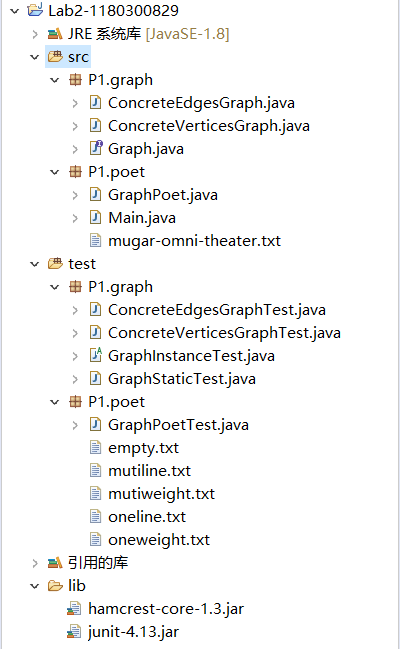
如何通过Git提交当前版本到GitHub上你的Lab2仓库。







在这里给出你的项目的目录结构树状示意图。



## Re-implement the Social Network in Lab1

这次实验要求我们基于Poetic Walks中定义的Graph<L>及其两种实现（本人使用的是ConcreteVerticesGraph<L>），实现Lab1中Social NetWorek中的各种功能，并且尽可能复用ConcreteVerticesGraph<L>中已经实现的方法，然后运行提供的main()和执行Lab1中的Junit测试用例，使之正常运行。

### FriendshipGraph类

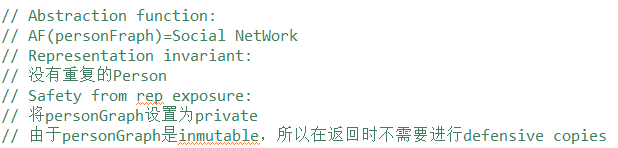
(1).FriendshipGraph的字段为Person构成的ConcreteEdgesGraph，定义私有类型的表如下图所示：



(2).在FriendshipGraph需要实现的方法如下图所示：

|  |  |
| --- | --- |
| FriendshipGraph | 构造方法 |
| addVertex | 在图中增加新Person，只需要调用ConcreteEdgesGraph中的add即可： |
| addEdge | 为某个人增加朋友,a为这个人，b为增加的朋友,直接调用ConcreteEdgesGraph中的set即可： |
| getallprople | 直接返回即可： |
| getDistance | 得到两个人之间的最短距离。与Lab1类似，稍加修改即可：  实现此功能我们需要了解先广方式求最短路径的方法。首先我们需要知道若两个Person对象为同一个，则返回0。然后定义一个Map集合theway和一个Person队列myqueue，队列myqueue用来储存广搜的遍历结果，theway的Map集合用来储存广搜的所有元素及他们与第一个元素的距离。具体实现方法为首先将第一个元素c1入队，并且把第一个元素c1和下标0入集合，当队列非空时，弹出队首元素top，并且得到top在集合theway中的下标distance，然后定义一个Map为friend，执行personGraph.targets(top)得到队首元素的所有朋友的Map，然后定义Set为allfriend，对刚才的friend执行keySet()得到allfriend，只要allfriend中的元素与c2不同，就把这些元素全部入队，并且把这些元素及下标distance+1放入集合。只要队列非空，继续执行以上步骤，直到找到某个元素与c2相同并且返回这个元素在集合theway中的下标。如果直到队列为空还没找到c2，则返回-1： |
| main | 复制Lab1的即可 |

(3).AF,RI和Safety from rep exposure如下图：



### Person类

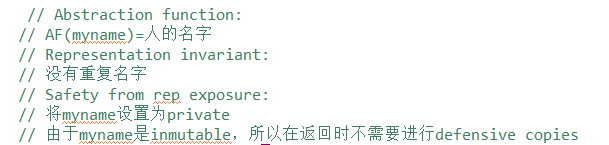
(1).FriendshipGraph的字段为Person构成的ConcreteEdgesGraph，定义私有类型的表如下图所示：



(2).在FriendshipGraph需要实现的方法如下图所示：

|  |  |
| --- | --- |
| Person | 没有重复名字则加入，构造方法 |
| getmyname | 返回本人名字，直接返回即可 |

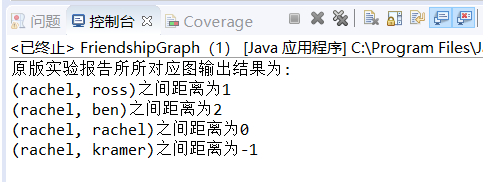
(3).AF,RI和Safety from rep exposure如下图：



### 客户端main()

按照Lab1中的main复制过来即可：

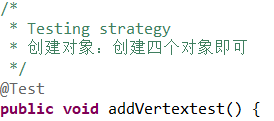
运行后如下：

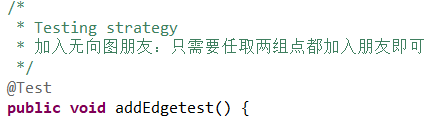


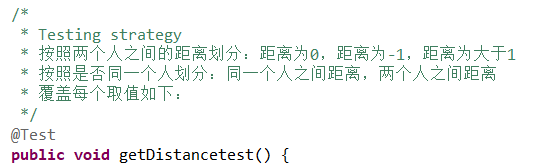
### 测试用例

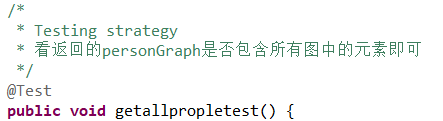
测试策略：

与Lab1的测试策略相同：

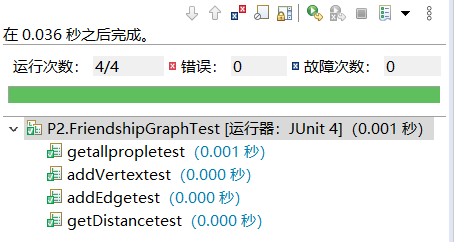








测试结果：



检查覆盖率：(需要提前注释main()函数)：



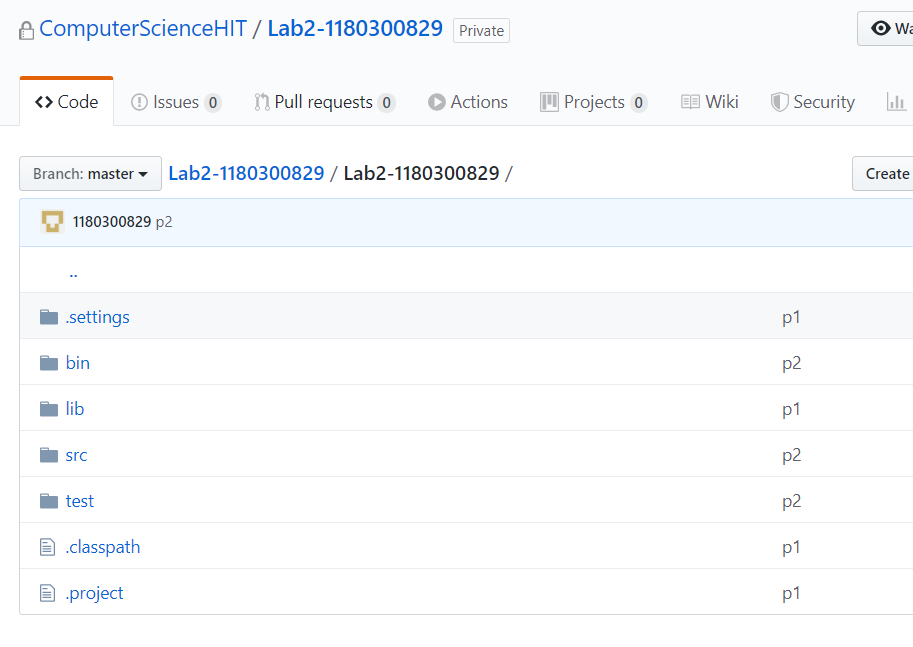
### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab2仓库。

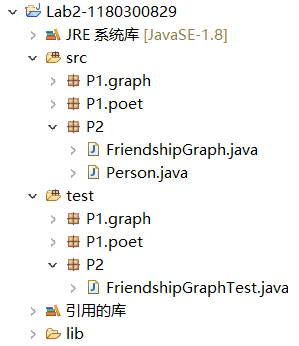








在这里给出你的项目的目录结构树状示意图。



## Playing Chess

### ADT设计/实现方案

**3.3.1.1 Position类**

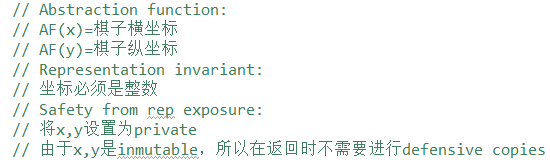
(1).Position类的字段为int的横坐标x和int的纵坐标y，定义私有类型的表如下图所示：



(2).在Position类中需要实现的方法如下图所示：

|  |  |
| --- | --- |
| Position | 构造方法 |
| getx | 返回当前点的横坐标 |
| gety | 返回当前点的纵坐标 |
| equals | 重写euqals，判断两个Position是否相等 |

(3).AF,RI和Safety from rep exposure如下图：



**3.3.1.2 Player类**

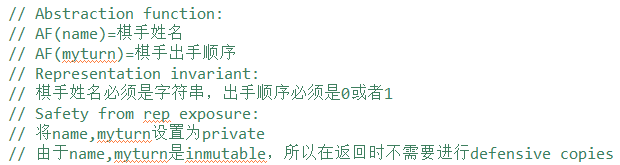
(1) .Player类的字段为String的棋手姓名x和Integet的棋手出手顺序myturn，定义私有类型的表如下图所示：



(2).在Player类中需要实现的方法如下图所示：

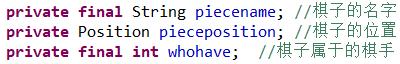
|  |  |
| --- | --- |
| setmyturn | 设置棋手出手顺序 |
| setname | 设置棋手名字 |
| getmyturn | 返回棋手出手顺序 |
| getname | 返回棋手名字 |

(3).AF,RI和Safety from rep exposure如下图：



**3.3.1.3 Piece类**

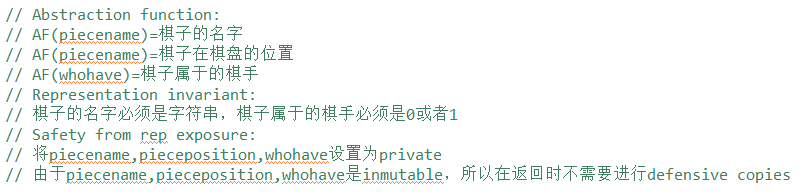
(1) .Piece类的字段为String的棋子的名字piecename，Position的棋子位置pieceposition和int的棋子属于的棋手whohave，定义私有类型的表如下图所示：



(2).在Piece类中需要实现的方法如下图所示：

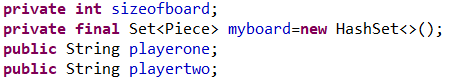
|  |  |
| --- | --- |
| Piece | 初始化构造方法，设置棋子的名字和棋子的拥有者 |
| getwhohave | 返回棋子的所有者 |
| getpiecename | 返回棋子的名字 |
| getpieceposition | 返回棋子的位置 |
| createnewposition | 重新设计新的棋子位置 |

(3).AF,RI和Safety from rep exposure如下图：



**3.3.1.4 Board类**

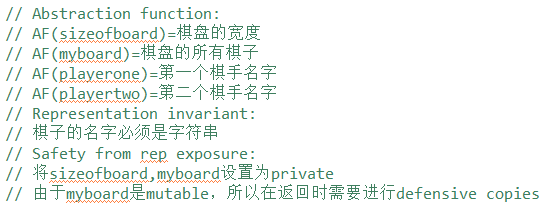
(1) . Board类的字段为int的棋盘宽度sizeofboard，Piece类型的Set表棋盘的所有棋子myboard，String的第一个棋手名字playerone和String的第二个棋手名字playertwo，定义类型的表如下图所示：



(2).在Board类中需要实现的方法如下图所示：

|  |  |
| --- | --- |
| Board | 初始化构造方法，设置棋盘大小 |
| getmyboard | 返回myboard |
| whethervalidposition | 判断某位置P是否越界 |
| whethervalidplayer | 得到描述某个位置的状态的字符串 |
| countplayerpiece | 统计两个棋手含有的棋子数量 |
| whetherempty | 检查某个位置是否是空闲 |
| putpiece | 放置某颗棋子在棋盘上，放置成功返回true ，指定位置超出棋盘范围、指定位置已有棋子返回false |
| removepiece | 移除某位置的棋子(提子)，移除成功返回true ，指定位置超出棋盘范围、所提棋子不是对方棋子、初始位置尚无可移动的棋子返回false |
| movepiece | 移动棋子，移动成功返回true，p1位置超出棋盘范围、p2位置超出棋盘范围、目的地已有其他棋子、始位置尚无可移动的棋子、两个位置相同、初始位置的棋子并非该棋手所有返回false |
| eatpiece | 吃子，吃子成功返回true，p1位置超出棋盘范围、p2位置超出棋盘范围、两个位置相同、第二个位置上的棋子不是对方棋子、第一个位置上的棋子不是自己棋子、第一个位置上无棋子、第二位置上无棋子返回false |

(3).AF,RI和Safety from rep exposure如下图：



**3.3.1.5 Action类**

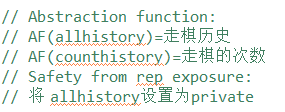
(1) .Action类的字段为String类型的AllayList表走棋历史allhistory和 int的走棋的次数counthistory，定义私有类型的表如下图所示：



(2).在Action类中需要实现的方法如下图所示：

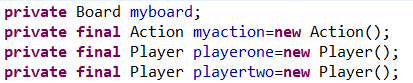
|  |  |
| --- | --- |
| whethervalidplayer | 得到描述某个位置的状态的字符串 |
| countplayerpiece | 统计两个棋手含有的棋子数量 |
| printhistory | 输出记录操作过程的数组 |
| putpiece | 放置成功返回true ，指定位置超出棋盘范围、指定位置已有棋子、该棋子并非属于该棋手，所指定的棋子已经在棋盘上返回false |
| removepiece | 移除某位置的棋子(提子)，移除成功返回true ，指定位置超出棋盘范围、所提棋子不是对方棋子、初始位置尚无可移动的棋子返回false |
| movepiece | 移动棋子，移动成功返回true，p1位置超出棋盘范围、p2位置超出棋盘范围、目的地已有其他棋子、始位置尚无可移动的棋子、两个位置相同、初始位置的棋子并非该棋手所有返回false |
| eatpiece | 吃子，吃子成功返回true，p1位置超出棋盘范围、p2位置超出棋盘范围、两个位置相同、第二个位置上的棋子不是对方棋子、第一个位置上的棋子不是自己棋子、第一个位置上无棋子、第二位置上无棋子返回false |

(3).AF,RI和Safety from rep exposure如下图：



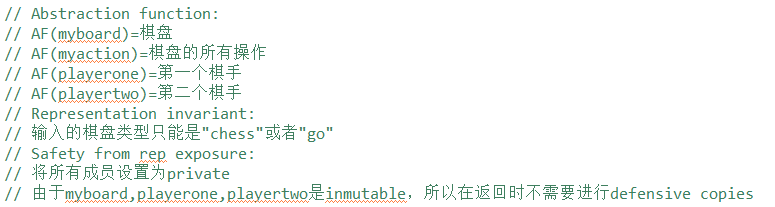
**3.3.1.6 Game类**

(1) .Game类的字段为Board的棋盘myboard，Action的棋盘的所有操作myaction ，Player的第一个棋手playerone和Player的第二个棋手playertwo，定义私有类型的表如下图所示：



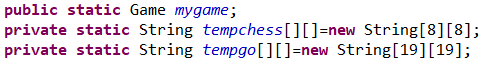
(2).在Game类中需要实现的方法如下图所示：

|  |  |
| --- | --- |
| Game | 构造方法，初始化棋盘，输入chess初始化国际象棋，将所有棋子放在棋盘上，输入go初始化围棋，棋盘上没有棋子 |
| setplayernames | 初始化棋手 |
| getmyboard | 返回棋盘 |
| getplayerone | 返回棋手A |
| getplayertwo | 返回棋手B |
| getplayer | 得到当前回合的棋手 |
| whethervalidplayer | 得到描述某个位置的状态的字符串 |
| countplayerpiece | 统计两个棋手含有的棋子数量 |
| printhistory | 输出记录操作过程的数组 |
| putpiece | 放置成功返回true ，指定位置超出棋盘范围、指定位置已有棋子、该棋子并非属于该棋手，所指定的棋子已经在棋盘上返回false |
| removepiece | 移除某位置的棋子(提子)，移除成功返回true ，指定位置超出棋盘范围、所提棋子不是对方棋子、初始位置尚无可移动的棋子返回false |
| movepiece | 移动棋子，移动成功返回true，p1位置超出棋盘范围、p2位置超出棋盘范围、目的地已有其他棋子、始位置尚无可移动的棋子、两个位置相同、初始位置的棋子并非该棋手所有返回false |
| eatpiece | 吃子，吃子成功返回true，p1位置超出棋盘范围、p2位置超出棋盘范围、两个位置相同、第二个位置上的棋子不是对方棋子、第一个位置上的棋子不是自己棋子、第一个位置上无棋子、第二位置上无棋子返回false |

(3).AF,RI和Safety from rep exposure如下图： 

### 主程序MyChessAndGoGame设计/实现方案

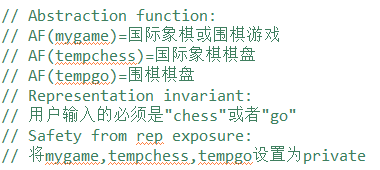
(1) . MyChessAndGoGame类的字段为Game的国际象棋或围棋游戏mygame，String二维数组的国际象棋棋盘tempchess 和String二维数组的围棋棋盘tempgo，定义私有类型的表如下图所示：



(2).在MyChessAndGoGame类中需要实现的方法如下图所示：

|  |  |
| --- | --- |
| empty | 初始化棋盘的棋子全为o |
| chessmenu | 国际象棋菜单 |
| gomenu | 围棋菜单 |
| printchess | 打印国际象棋的棋盘 |
| printgo | 打印围棋的棋盘 |
| main | 主程序 |
| chessgame | 国际象棋比赛的界面和操作 |
| gogame | 围棋比赛的界面和操作 |

(3).AF,RI和Safety from rep exposure如下图：



(4).具体命令行输入的指令映射的实现方案：

**在main方法中：**

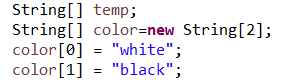
定义了一个Scanner变量scanner用来储存用户的输入，执行while循环直到用户输入chess或者go的字符串，从而建立新的Game类mygame。然后依次读入两位棋手的名字，mygame执行setplayernames方法储存名字，然后根据用户的输入，分别对两种比赛宣布开始，分别调用chessgame方法或者gogame方法：



**在chessgame方法中：**

定义了一个Scanner变量scanner用来储存用户的输入，定义一个标志变量flag，当标志变量为true时不断重复执行while让用户输入，每次循环输出菜单chessmenu()，然后执行switch根据用户的输入选择方法进行调用。在坐标的处理中，使用split来进行字符串的切分。在每一个case中执行case对应的ADT中的方法即可。定义了一个顺序变量turn，棋手每次执行成功自己的操作后进行即可完成turn不断在0和1之间转换，从而对两位棋手进行交替操作。

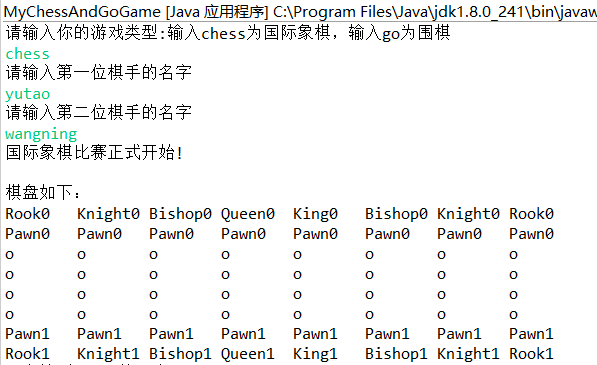
**在gogame方法中：**

和chessgame类似，但在开始需要进行对棋子的初始化，因为围棋在Game类中是没有初始化棋子的，即可。然后同chessgame，定义了一个Scanner变量scanner用来储存用户的输入，定义一个标志变量flag，当标志变量为true时不断重复执行while让用户输入，每次循环输出菜单chessmenu()，然后执行switch根据用户的输入选择方法进行调用。在坐标的处理中，使用split来进行字符串的切分。在每一个case中执行case对应的ADT中的方法即可。定义了一个顺序变量turn，棋手每次执行成功自己的操作后进行即可完成turn不断在0和1之间转换，从而对两位棋手进行交替操作。

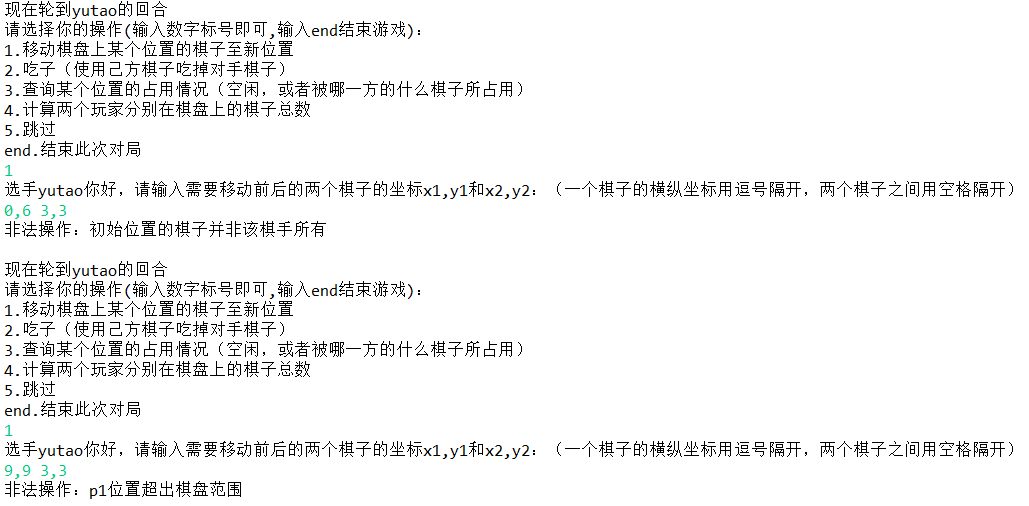
(5). 执行过程的截图：

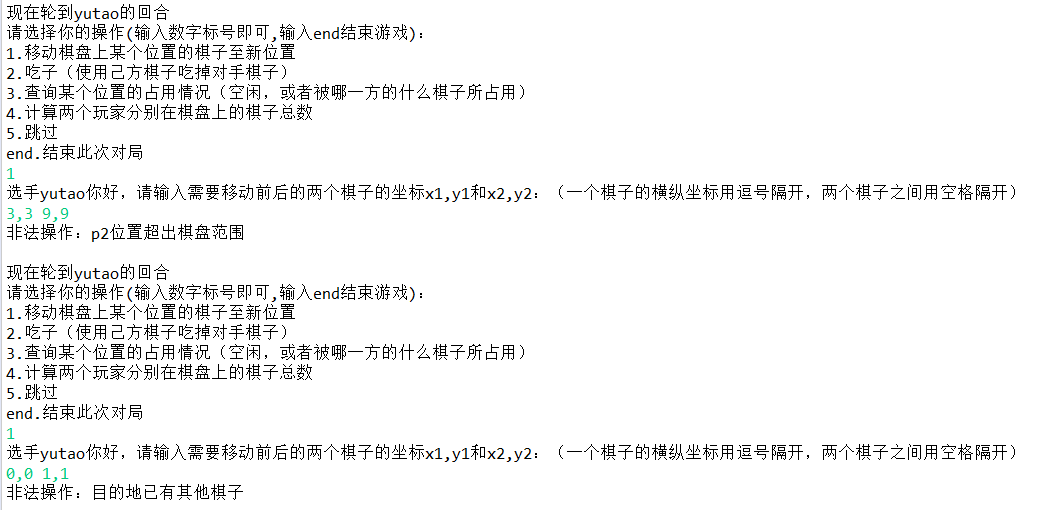
**对于象棋：**

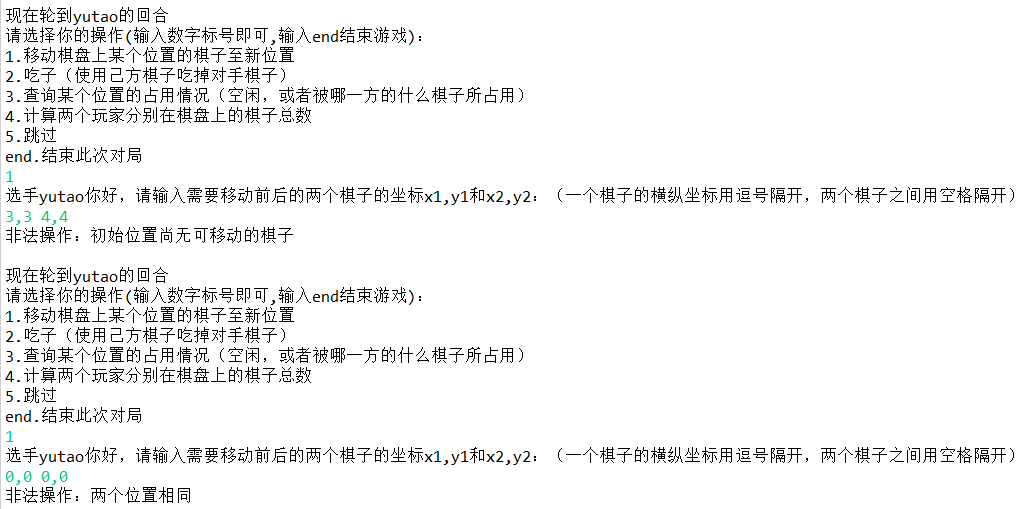
初始化操作：

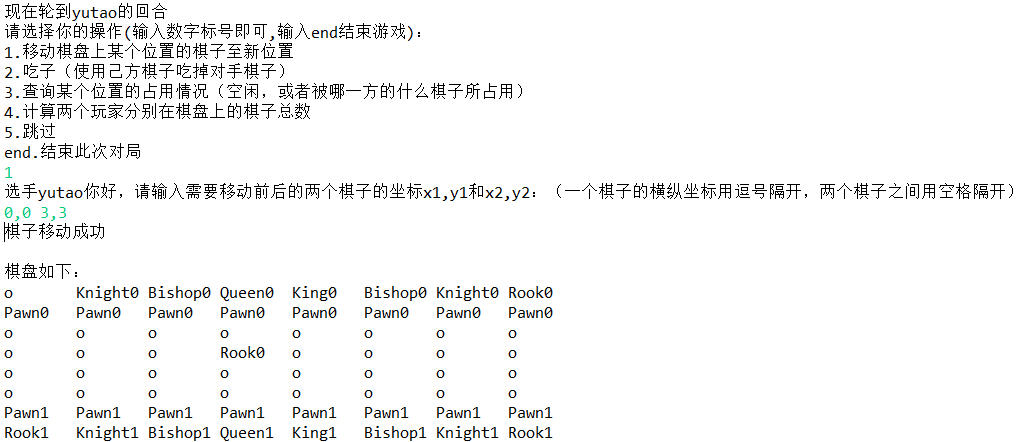


1. 移动棋盘上某个位置的棋子至新位置:

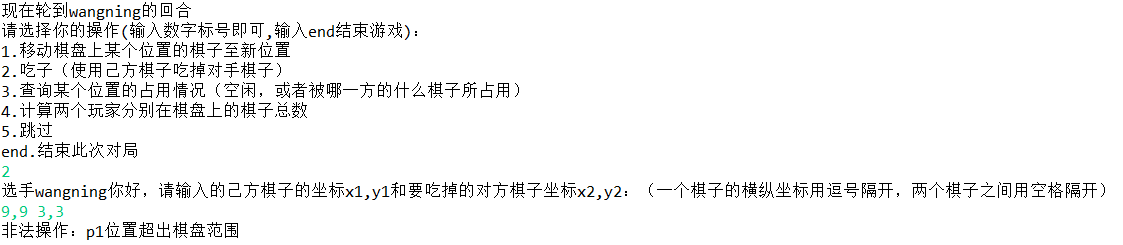


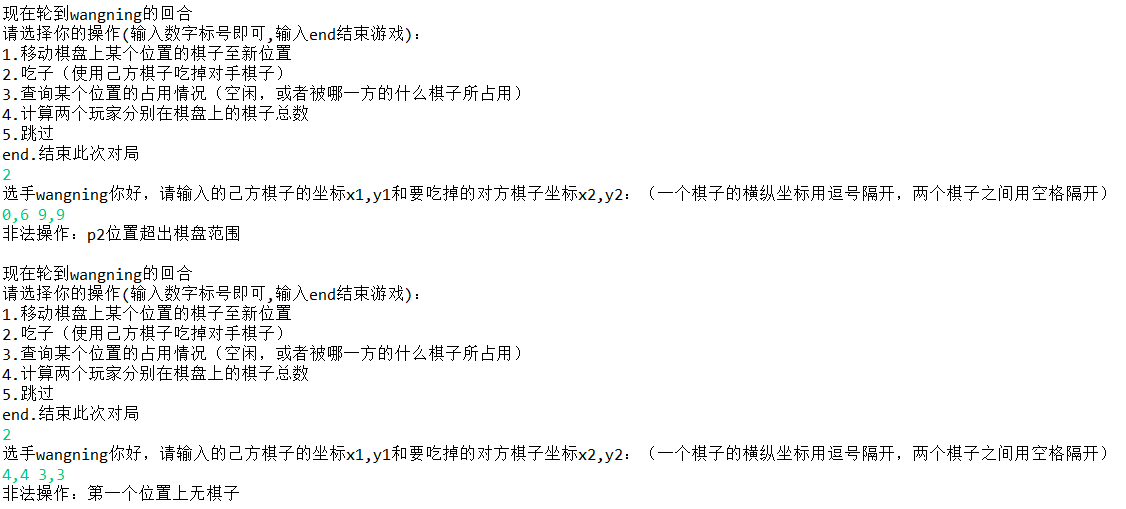


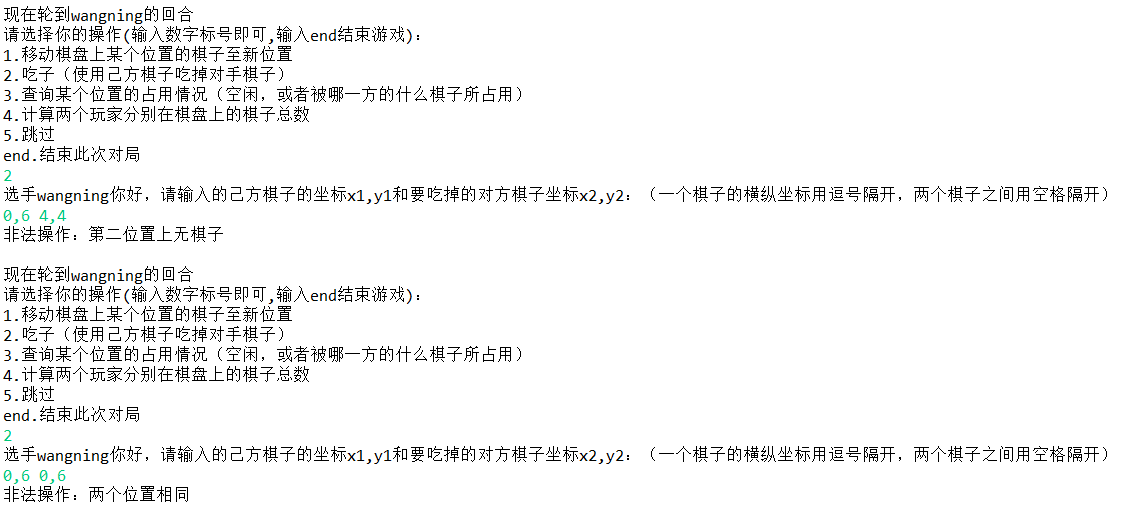


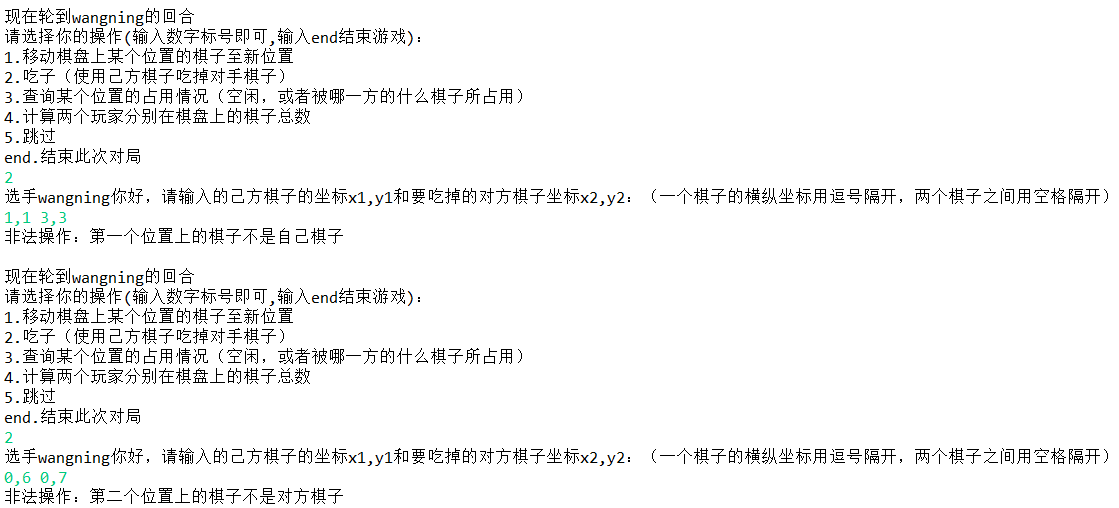


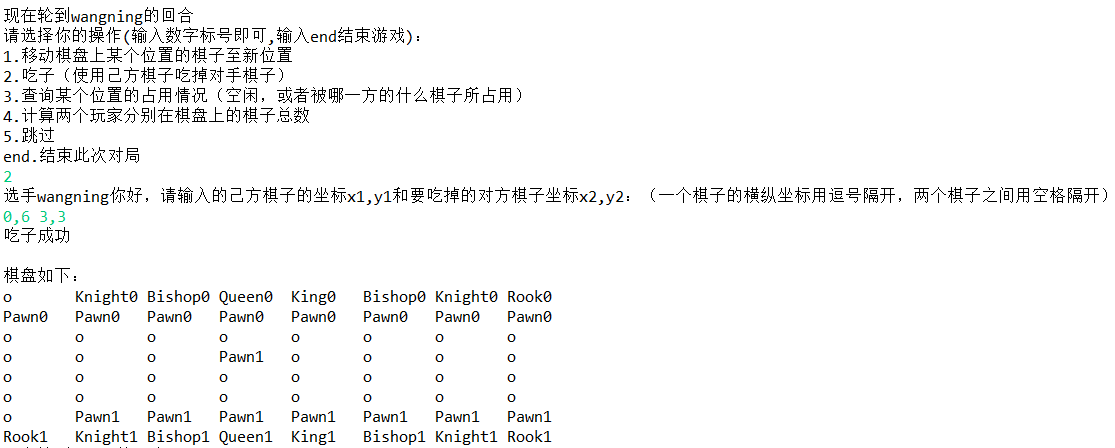
2.吃子（使用己方棋子吃掉对手棋子）:



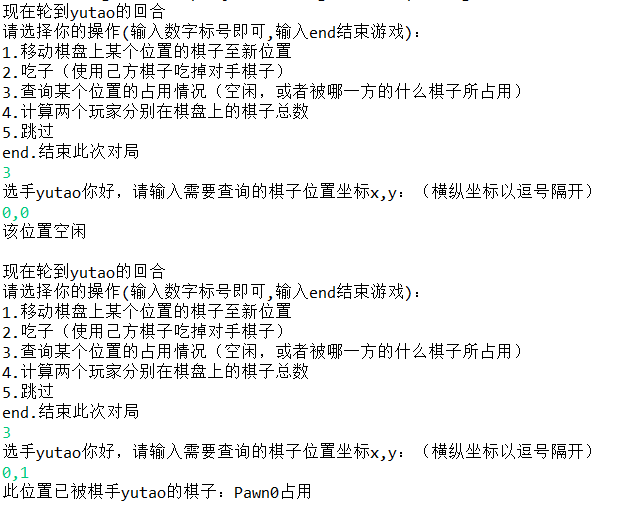


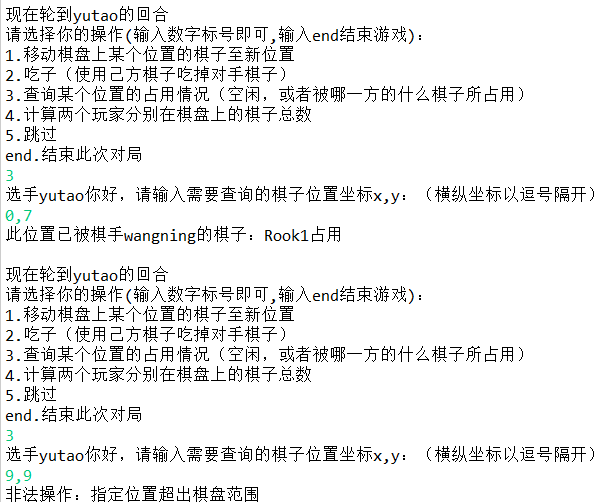




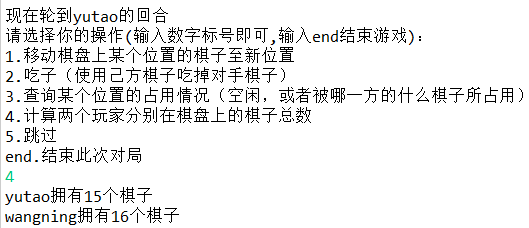


3.查询某个位置的占用情况（空闲，或者被哪一方的什么棋子所占用）:

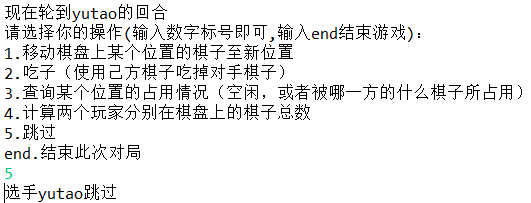




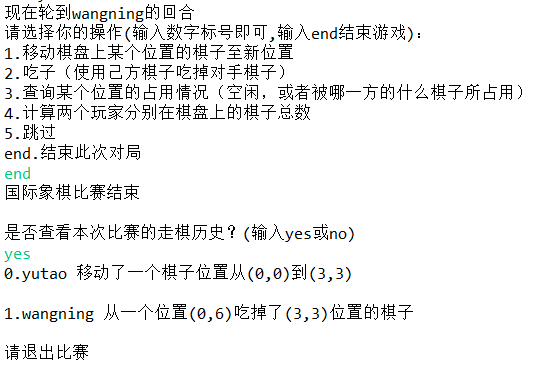
4.计算两个玩家分别在棋盘上的棋子总数:



5.跳过

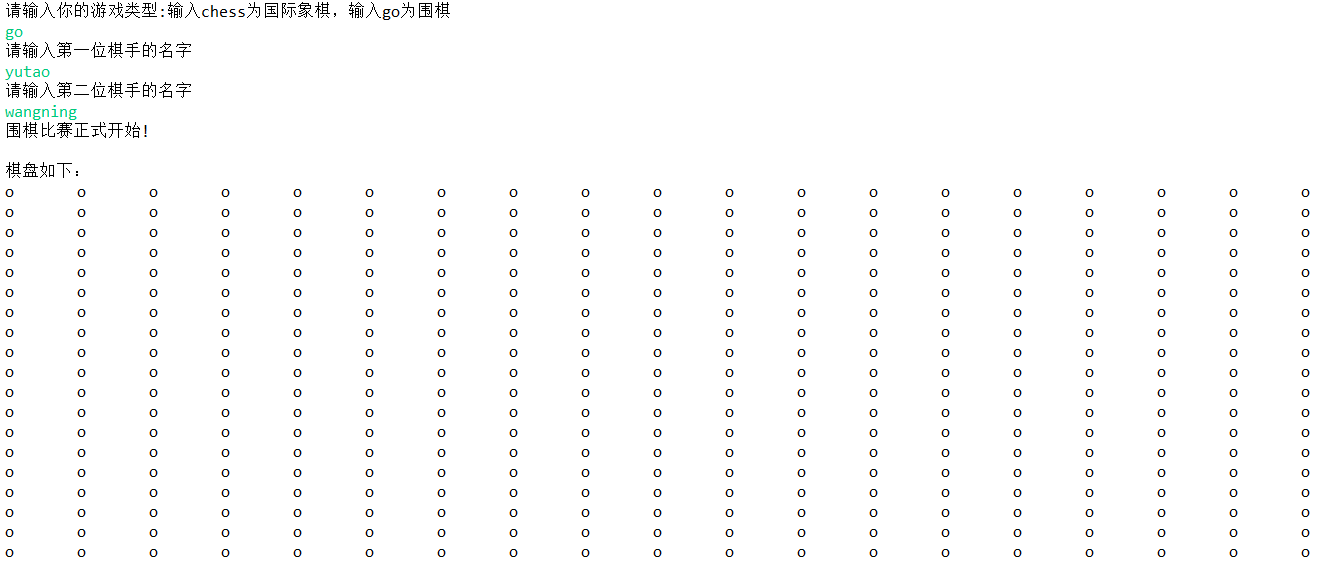


end.结束此次对局:



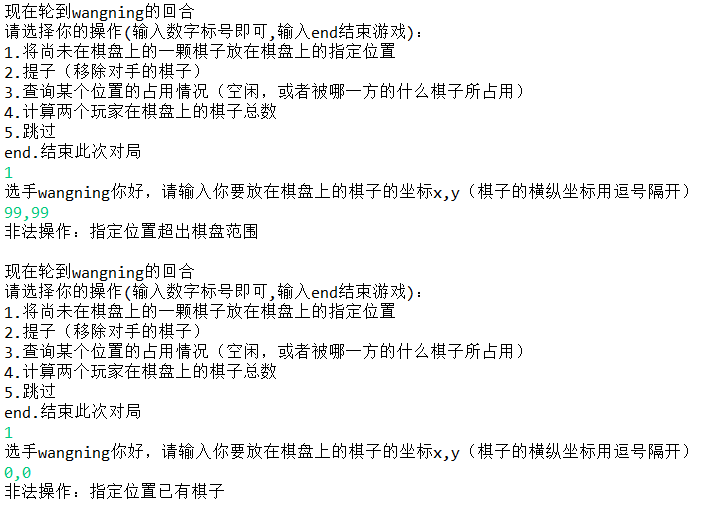
**对于围棋：**

初始化操作：



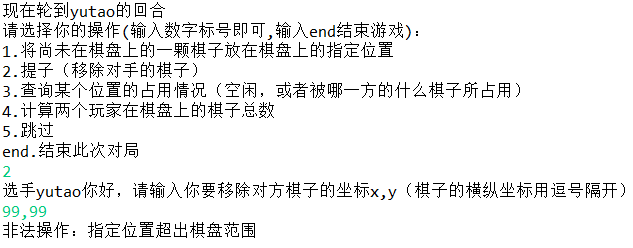
1.将尚未在棋盘上的一颗棋子放在棋盘上的指定位置：

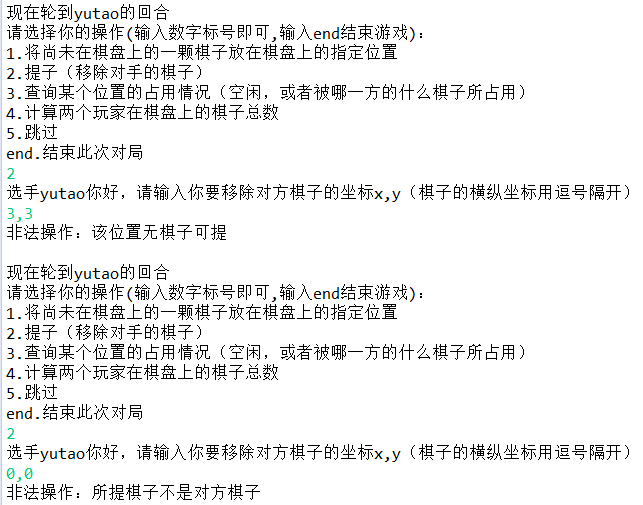


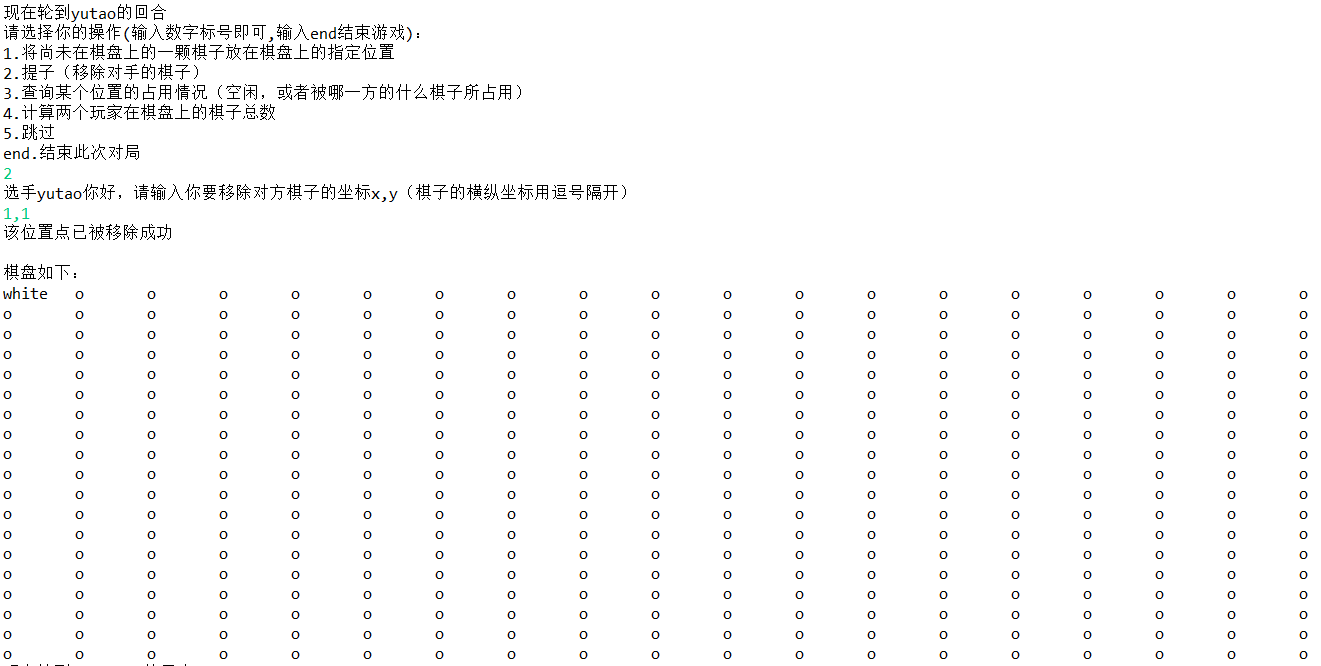




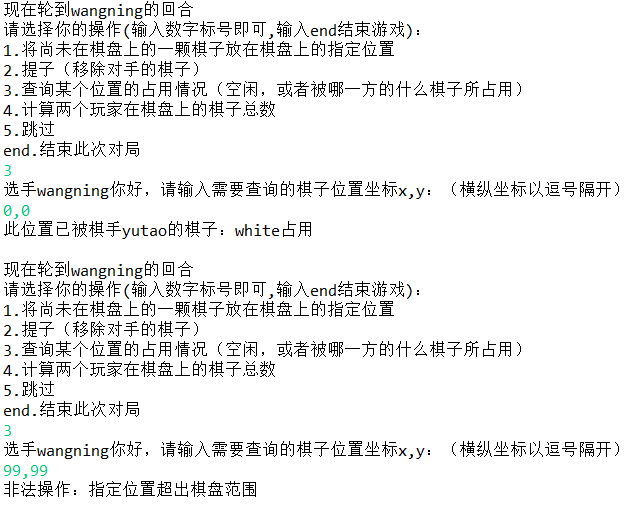
2.提子（移除对手的棋子）：

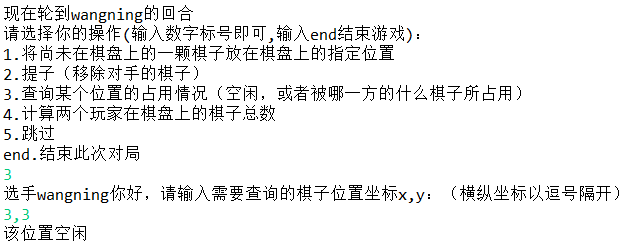




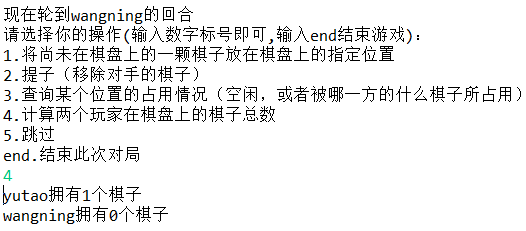


3.查询某个位置的占用情况（空闲，或者被哪一方的什么棋子所占用）：

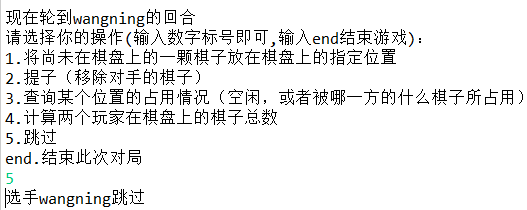




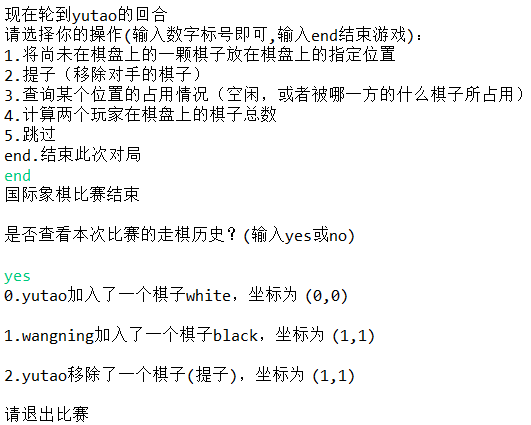
4.计算两个玩家在棋盘上的棋子总数：



5.跳过



end.结束此次对局：

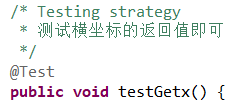


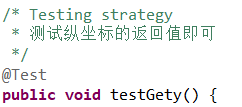
### ADT和主程序的测试方案

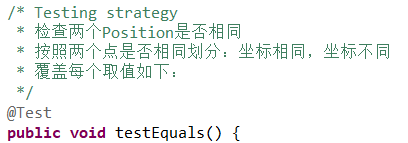
主程序使用main手动测试：针对各种情况在上面的截图中都给出了结果。

对于其他ADT：

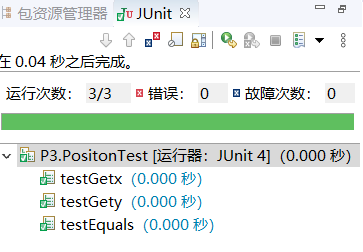
**PositonTest：**







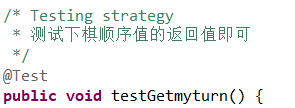
测试结果：

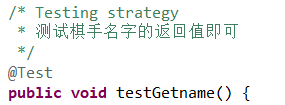


覆盖率：

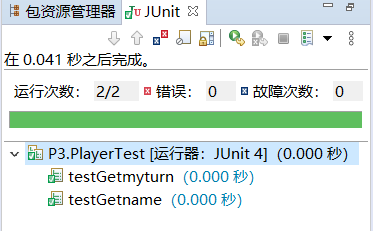


**PlayerTest：**





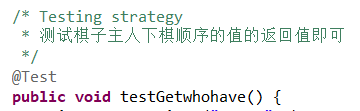
测试结果：

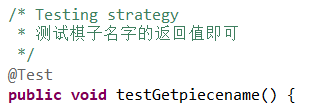


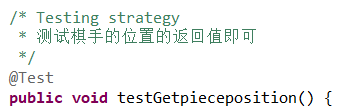
覆盖率：



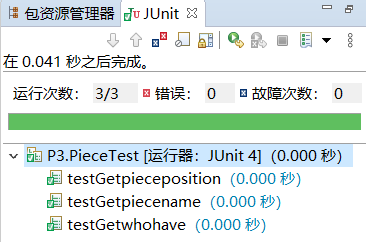
**PieceTest：**







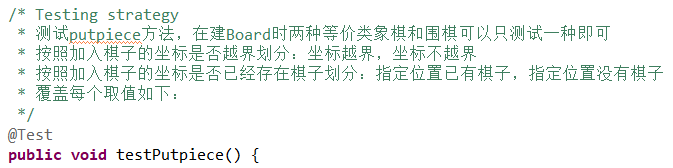
测试结果：

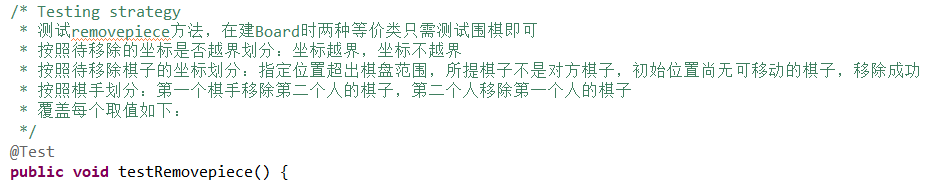


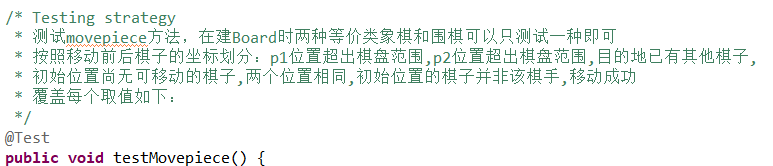
覆盖率：

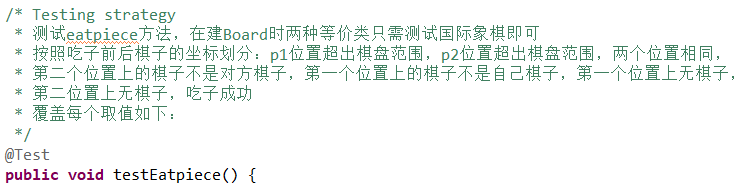


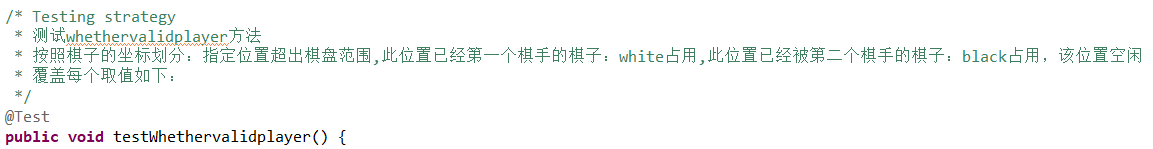
**BoardTest：**

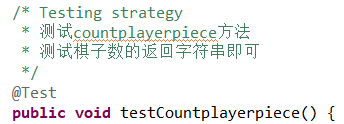




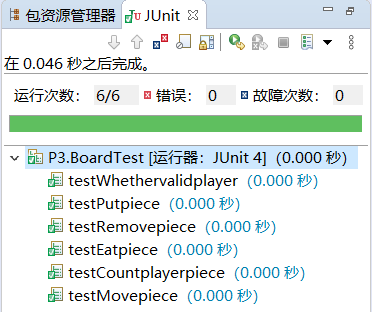








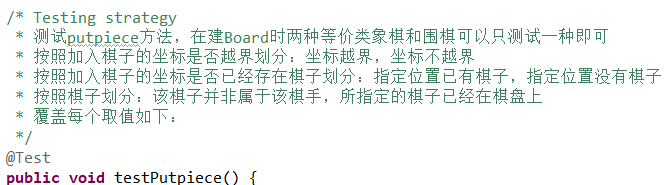
测试结果：

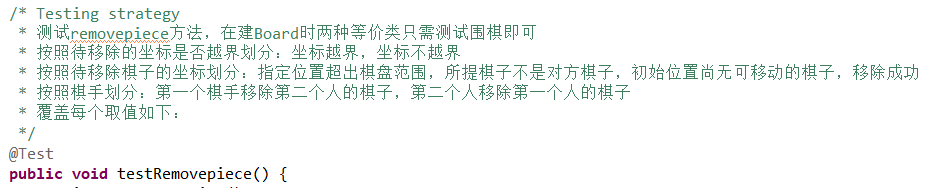


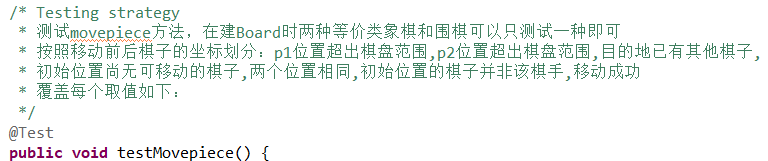
覆盖率：

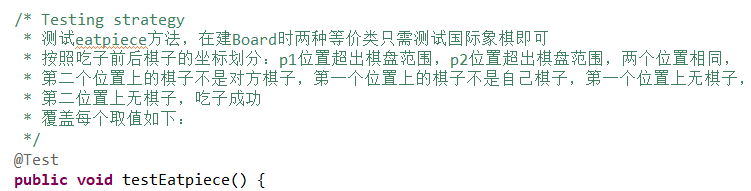


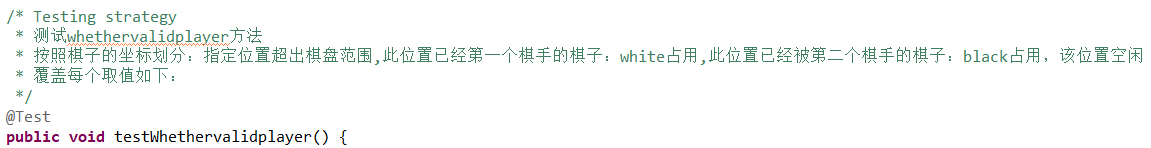
**AcitionTest：**

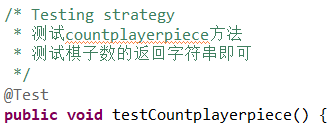


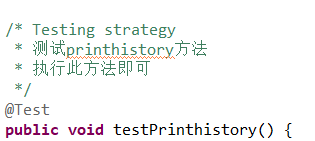




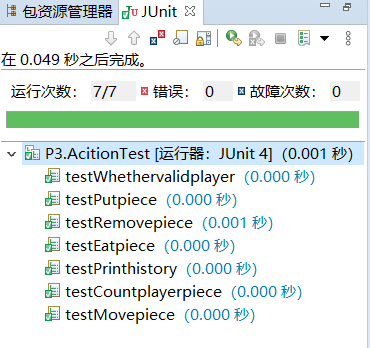








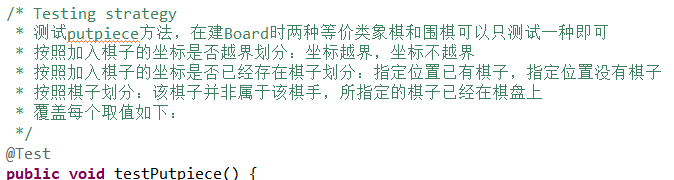
测试结果：

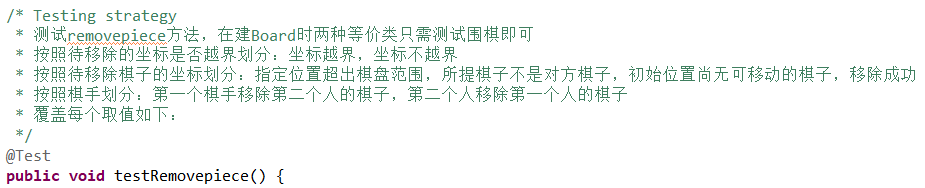


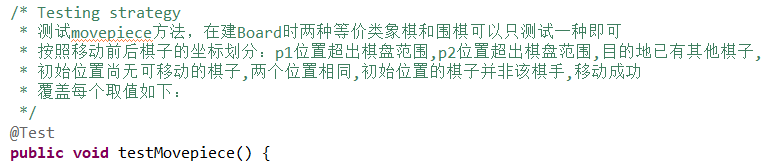
覆盖率：

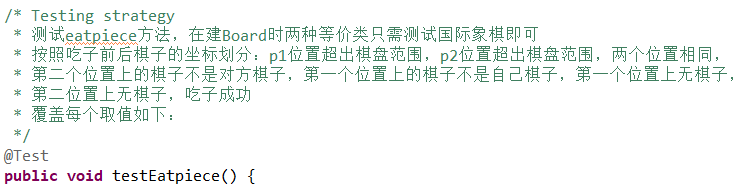


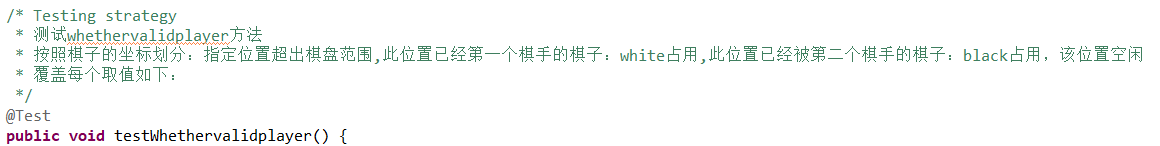
**GameTest：**

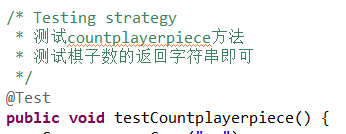


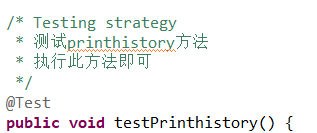


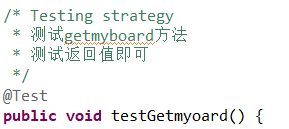


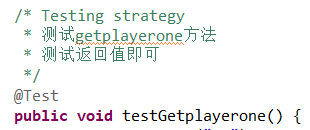


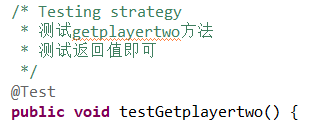












测试结果：



覆盖率：



# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.20 | 18:00-22:00 | P1中边图类 | 未完成 |
| 3.21 | 16:00-22:30 | 完成P1边图类 | 完成 |
| 3.22 | 19:00-23:00 | P1点图类 | 未完成 |
| 3.23 | 18:00-22:30 | 完成P1点图类 | 完成 |
| 3.24 | 18:00-22:00 | 完成P1中的诗集类 | 未完成 |
| 3.25 | 17:50-22:20 | P1诗集类 | 完成 |
| 3.26 | 19:00-22:30 | P1中点图和边图的test | 完成 |
| 3.27 | 20:00-22:40 | 完成P1除了测试外的报告 | 完成 |
| 3.28 | 14:00-20:30 | P1的所有测试 | 完成 |
| 3.29 | 13:00-21:00 | 完成P2及测试 | 完成 |
| 4.1 | 18:00-21:30 | 完成P3中Position，Player，Piece | 完成 |
| 4.3 | 18:00-22:30 | 完成P3中Board | 未完成 |
| 4.4 | 19:00-21:30 | 完成P3中Board | 完成 |
| 4.4 | 19:00-22:10 | 完成P3中Action及Game | 完成 |
| 4.6 | 19:00-21:30 | 完成P3中的主类 | 完成 |
| 4.7 | 19:00-22:30 | 完成所有P3的测试 | 未完成 |
| 4.8 | 14:00-22:30 | 完成所有P3的测试 | 完成 |
| 4.9 | 15:00-20:30 | 完成P2和P3部分的报告 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 刚开始写P1时还没有讲解AF,RI,Safety from rep exposure，不知道有什么用 | 后来上课明白了进行了补充 |
| 不理解泛型的意思 | 查找资料和后来上课明白了 |
| 不是太理解继承的关系 | csdn求助，后来课堂上也进行了讲解 |
| P3因为没有给出代码框架，自行设计各个类遇到很多困难 | 考虑每一个类应该实现的方法和储存的内容，最后完成了代码 |
| 在写P3的test时总是考虑掉情况 | 慢慢分析等价类，终于完成了test的覆盖 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

在自行设计多种类来实现功能的情况下，自己设计的很多类之间有很多重复和矛盾的部分，很多关系弄不清楚，代码很冗杂，测试起来也较麻烦。对于java还是不够熟练，需要多加练习

## 针对以下方面的感受

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？

答：面向对象能够每次完成对某个对象的编程，与面向过程的编程的思路完全不一样。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

答：泛型能够适应更多的变化，更加灵活。

1. 在给出ADT的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？

答：能够保证代码的正确性，及时修改。不适应。

1. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？

答：可以提高代码的利用率，减少重复。

1. P3要求你从0开始设计ADT并使用它们完成一个具体应用，你是否已适应从具体应用场景到ADT的“抽象映射”？相比起P1给出了ADT非常明确的rep和方法、ADT之间的逻辑关系，P3要求你自主设计这些内容，你的感受如何？

答：最开始一头雾水，后来通过慢慢分析对每个类进行了实现，但是还是很多方法用的很繁琐，并且很多类很重复，一些功能没有用到。

1. 为ADT撰写specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？

答：防止内部变量被外部修改。很繁琐，写但了更好。

1. 关于本实验的工作量、难度、deadline。

答：工作量很大，难度不太大，主要写注释占用太多时间，写测试也用很多时间，报告的内容太多，需要截图太多。deadline很合适。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

希望有更多的中文注释，英文看着很难受。