

**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 | cycleke |
| 学号 | xxxxxxxxxx |
| 班号 | xxxxxxx |
| 电子邮件 | [cycleke@gmail.com](mailto:cycleke@gmail.com) |
| 手机号码 | xxxxxxxxxxx |

**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> 3](#_Toc29325503)

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

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc29325505)

[3.1.3.2 Implement ConcreteVerticesGraph 4](#_Toc29325506)

[3.1.4 Problem 3: Implement generic Graph<L> 5](#_Toc29325507)

[3.1.4.1 Make the implementations generic 5](#_Toc29325508)

[3.1.4.2 Implement Graph.empty() 5](#_Toc29325509)

[3.1.5 Problem 4: Poetic walks 5](#_Toc29325510)

[3.1.5.1 Test GraphPoet 5](#_Toc29325511)

[3.1.5.2 Implement GraphPoet 5](#_Toc29325512)

[3.1.5.3 Graph poetry slam 6](#_Toc29325513)

[3.1.6 Before you’re done 6](#_Toc29325514)

[3.2 Re-implement the Social Network in Lab1 7](#_Toc29325515)

[3.2.1 FriendshipGraph类 7](#_Toc29325516)

[3.2.2 Person类 8](#_Toc29325517)

[3.2.3 客户端main() 8](#_Toc29325518)

[3.2.4 测试用例 9](#_Toc29325519)

[3.2.5 提交至Git仓库 9](#_Toc29325520)

[3.3 Playing Chess 10](#_Toc29325521)

[3.3.1 ADT设计/实现方案 10](#_Toc29325522)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 14](#_Toc29325523)

[3.3.3 ADT和主程序的测试方案 19](#_Toc29325524)

[4 实验进度记录 19](#_Toc29325525)

[5 实验过程中遇到的困难与解决途径 20](#_Toc29325526)

[6 实验过程中收获的经验、教训、感想 20](#_Toc29325527)

[6.1 实验过程中收获的经验和教训 20](#_Toc29325528)

[6.2 针对以下方面的感受 20](#_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(abstractionfunction)
* 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

## 实验环境配置

此次实验的实验环境与Lab1的实验环境大体相似，不过需要安装EclEmma插件。可以在 Marketplace 中找到找到 EclEmma 并下载。



图 2‑1 在Eclipse Marketplace中安装EclEmma

下图是使用 EclEmma 测试代码覆盖率。

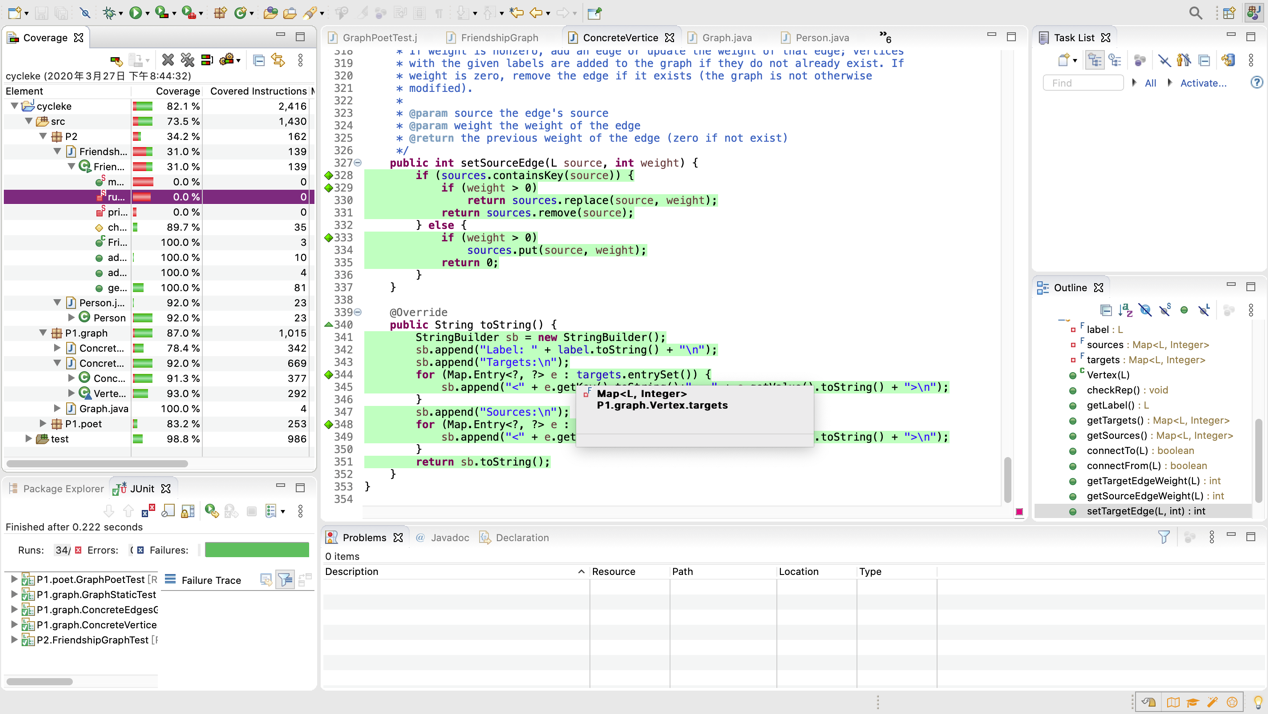


图 2‑2使用 EclEmma 测试代码覆盖率

## GitHub Lab2仓库的URL地址

https://github.com/ComputerScienceHIT/Lab2-xxxxxxxxxx

# 实验过程

## Poetic Walks

本实验的目的是练习设计、测试以及编写ADT，而且ADT的规约已经给出。要实现的ADT就是一个存储图的数据结构，已经有两种实现方式被给出了：用点建图和用边建图。

实验的后半部分是利用已经实现的ADT实现对一个根据语料库写诗的程序。

### Get the code and prepare Git repository

通过命令git clone [git@github.com:rainywang/Spring2020\_HITCS\_SC\_Lab2.git](mailto:git@github.com:rainywang/Spring2020_HITCS_SC_Lab2.git) 获取代码，之后将下载的目录中P1的代码复制到自己的Lab2目录中。通过 git init指令新建工作区，使用git commit提交更改，然后使用git remote add origin [git@github.com:ComputerScienceHIT/Lab2-xxxxxxxxxx.git](mailto:git@github.com:ComputerScienceHIT/Lab2-1180100217.git) 命令与远程仓库关联。

之后可以使用git push与git pull指令与远程仓库交互。

### Problem 1: Test Graph <String>

根据Graph的规约，我们需要对于Graph中的每个公开方法编写测试样例。如对于Graph#set(Object, Object, int)方法，我的测试测试策略为：区分边在原图中是否存在；设置的边值为0还是正数。进而有以下代码：



图 3‑1Graph#set(Object, Object, int)测试代码

其它函数的测试策略如下：

* 对于Graph#add(Object)函数，测试策略为：添加的点是否在图中。
* 对于Graph#remove(Object)函数，测试策略为：删除的点是否在图中；删除的点是否有边。
* 对于Graph#vertices()函数，测试策略为：图是否为空。
* 对于Graph#sources(Object)函数，测试策略为：target是否有边；target是否在图中。
* 对于Graph#target(Object)函数，测试策略为：source是否有边；source是否在图中。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

首先需要实现一个immutable辅助类Edge，其包含三个field：起点source、终点target和边权weight。对于表示不变量的约束是权重必须为正数和起点和终点不能相同。我为它实现了getter函数，重载equals和toString。

之后来实现mutable类ConcreteEdgeGraph，其包含两个field：点集vertices和边列表edges。

对于表示不变量的约束是满足Edge的不变量约束且边的起点和终点均在点集中。

对ConcreteEdgesGraph的保护是将vertices和edges有关的变量都设置为private final类型，防止外部直接修改。

需要实现的函数有add、set、remove、vertices、sources和targets。此外还需要重载toString函数。

add函数通过向点集vertices中加入点，之后checkRep，返回Set.add函数的返回值。

set函数首先判断边权是否非负，之后在edges中查找是否有起终点符合条件的边。如果有，则纪录旧边权，更改或删除对应的边；如果没有，则添加（边权为正数时）一条新的边。由于可能修改了图，所以同样的，我们需要调用checkRep函数，最后返回旧边权（如果图中原本没有对应的边，则返回0）。

remove函数则是首先在点集vertices中删除对应的点（如果点不在vertices中，则直接返回假值），之后将包含要删除的点的边从图中删除，之后checkRep，返回真值。

vertices函数返回点集vertices的一个复制。此函数不用checkRep是因为它没要修改图。而返回复制是为了防止通过非开放的API来修改图。

sources和targets函数的实现类似，查找target（source）属性与查找的点相同的边，将它们置于一个Map中，之后返回。

#### Implement ConcreteVerticesGraph

首先需要实现一个mutable辅助类Vertices，其包含三个field：点对应的属性label，起始边集source和终点边集target。对于表示不变量的约束是属性不是null和所有的边权为正数。我为它实现了各个getters和setters函数，重载toString函数。

之后来实现mutable类ConcreteVerticesGraph，其包含一个field：点集vertices。

对于表示不变量的约束是满足Vertices的不变量约束且一条边在起点和终点中的表述应当相同。

对ConcreteVerticesGraph的保护是将vertices有关的变量都设置为private final类型，防止外部直接修改。

需要实现的函数有add、set、remove、vertices、sources和targets。此外还需要重载toString函数。

add函数首先在点集vertices中搜索是否存在。如果存在，则返回假值；如果不存在则添加新点，之后checkRep，返回真值。

set函数首先判断边权是否非负，之后在source点的边中查找是否有终点符合条件的边。如果有，则纪录旧边权，更改或删除对应的边；如果没有，则添加（边权为正数时）一条新的边。同理于target点。由于可能修改了图，所以同样的，我们需要调用checkRep函数，最后返回旧边权（如果图中原本没有对应的边，则返回0）。

remove函数则是首先在点集vertices中删除对应的点（如果点不在vertices中，则直接返回假值），之后将包含要删除的点的边从其它的点中删除，之后checkRep，返回真值。

vertices函数返回点集vertices的label的一个复制。此函数不用checkRep是因为它没要修改图。而返回复制是为了防止通过非开放的API来修改图。

sources和targets函数的实现类似，直接返回target（source）的sources（targets）。

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

#### Make the implementations generic

此任务要求将Graph改造成支持泛型。

不需要进行任何额外操作，将Graph中的String改成泛型L就能支持泛型了。

#### Implement Graph.empty()

此任务要求生成一个空图实例。

这是一个静态方法，我选择返回了一个ConcreteVerticesGraph的实例。

### Problem 4: Poetic walks

#### Test GraphPoet

本题要求编写GraphPoet的测试，测试策略为：两个词是否存在于图中；图中两个词间是否存在一条长度为2的路径。

#### Implement GraphPoet

实现的GraphPoet包含两个field：一个Graph<String>类型的graph和一个List来存储原本诗中的各个词。

对于表示不变量的约束是graph合法和单词表中无空白字符或回车符。

对GraphPoet的保护是将graph和wordList变量都设置为private final类型，防止外部直接修改。

GraphPoet的构造函数旨在构建单词图。在构建图时，遍历每个单词，将它与它后面的单词作为一条边的两个顶点加入单词图，利用Graph中的set()直接添加。

而poem函数是根据要修改的诗中前后两个词来找到中间词，对每两个词的中间节点在图中进行寻找，如果找到了且权重最高则插入。

#### Graph poetry slam

本题是用自己的语料库和诗生成一首新的诗，调用GraphPoet的poet函数即可实现。

### Before you’re done

使用git add将文件添加到工作区，用git commit提交更改。由于之前已经将本地仓库已经和GitHub远程仓库关联，使用git push即可将本地代码提交到GitHub。

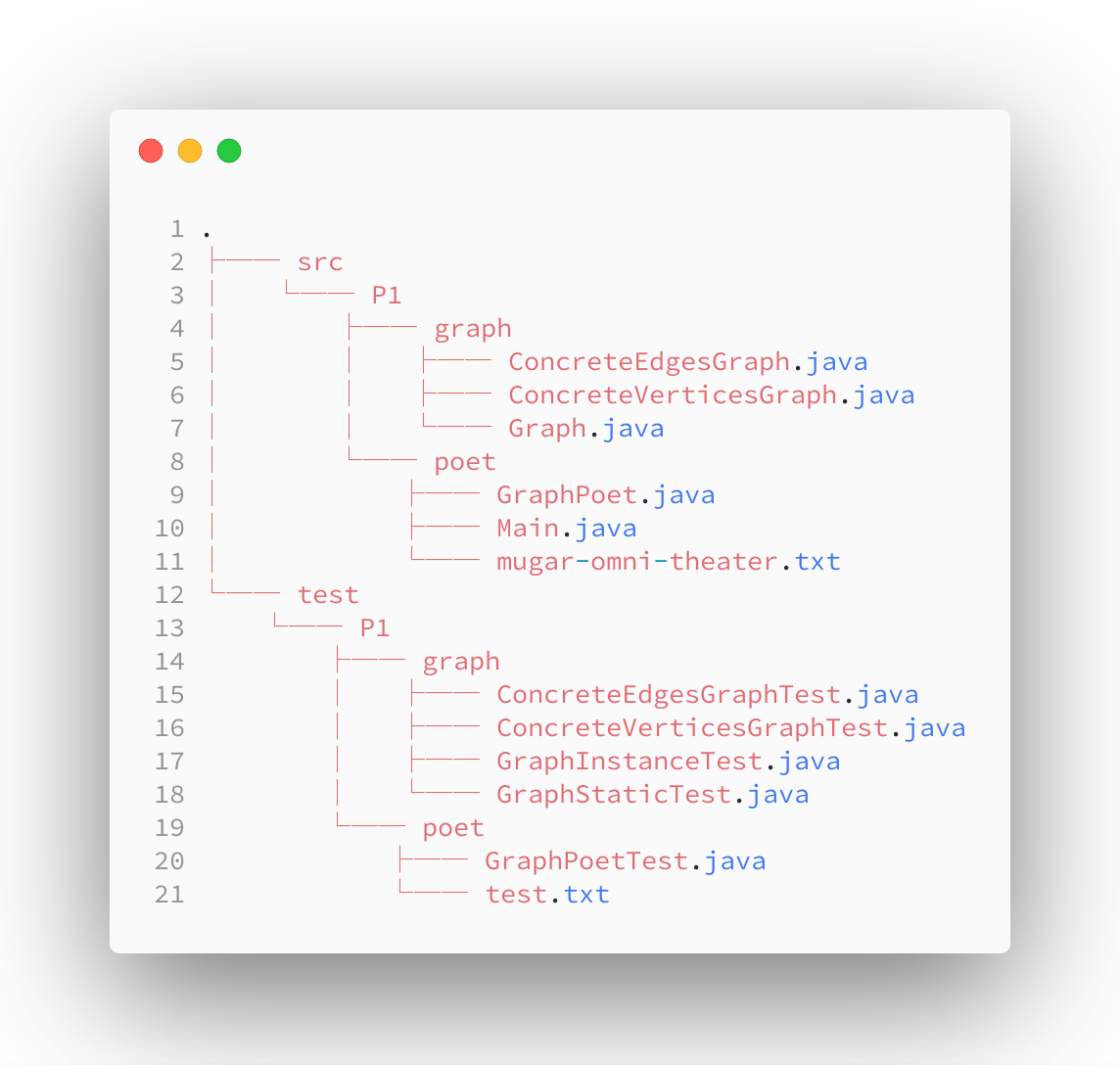


图 3‑2Lab1项目目录结构树状示意图

## Re-implement the Social Network in Lab1

此任务目的是使用此实验第一部分中已经实现的Graph ADT重写将实验一中的Social Network中的社交网络。通过已经实现的Graph，我们降低在实现社交网络时的难度，可以充分利用ADT，降低代码各个部分的耦合度。

### FriendshipGraph类

FriendshipGraph类继承了ConcreteVerticesGraph<Person>类，通过充分利用ConcreteVerticesGraph<Person>已经实现的ADT，降低了图的边和点增删的实现复杂度。

对于表示不变量的约束是图合法且边的边权均为1。

对于addVertex和addEdge可以利用ConcreteVerticesGraph<Person>的add和set函数，而getDistance仍使用BFS算法。

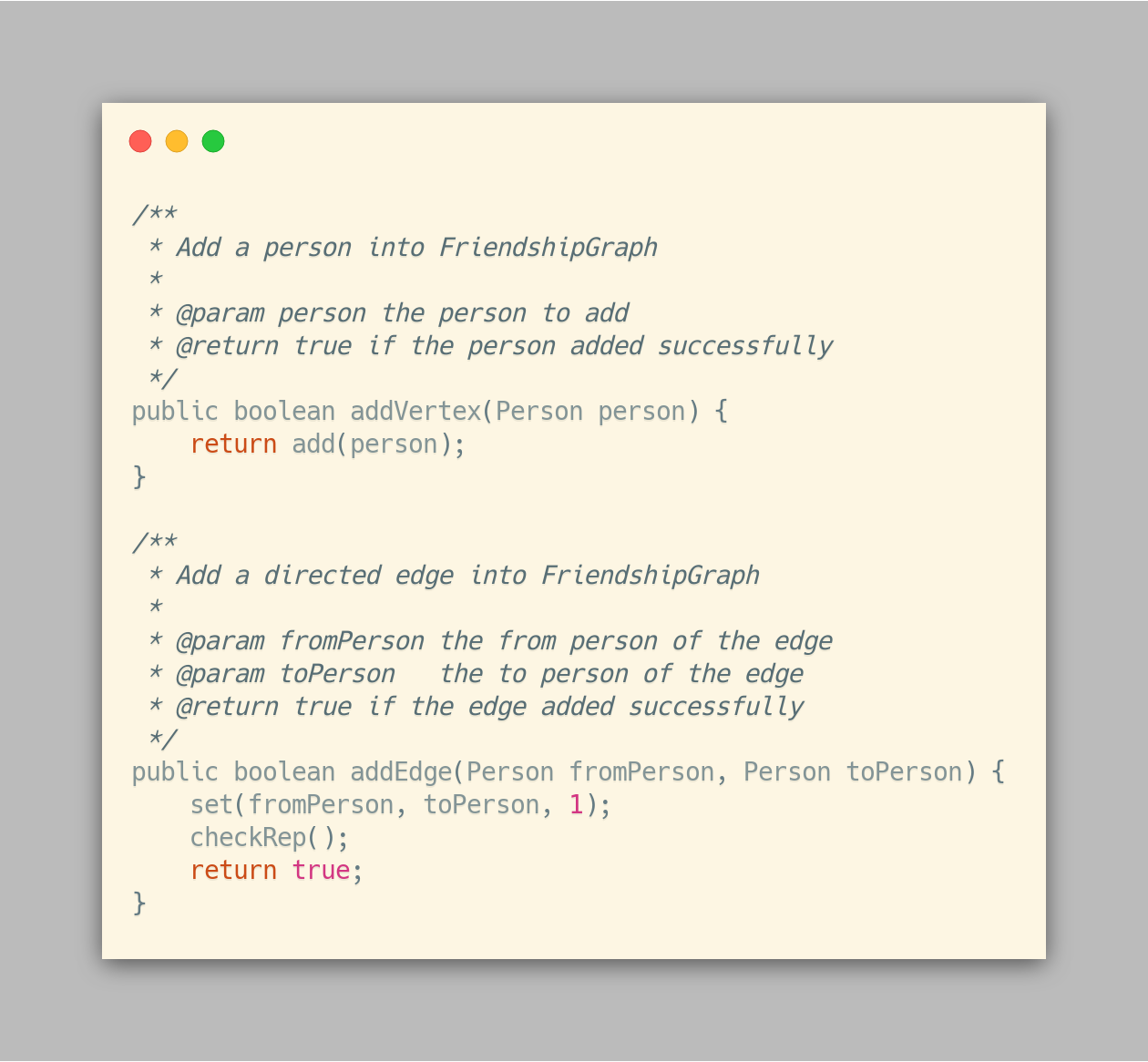


图 3‑3addVertex和addEdge的实现

### Person类

此任务只是重构了FriendshipGraph类的实现，所以无需更改Person类的实现。

### 客户端main()

由于FriendshipGraph类的addVertex、addEdge和getDistance规约没有变化，所以命令行客户端则可以使用原本的实现方法。



图 3‑4客户端的运行演示

### 测试用例

由于整个FriendshipGraph的规约没有变化，所以测试用例直接使用Lab1的测试用例。

### 提交至Git仓库

使用git add将文件添加到工作区，用git commit提交更改。由于之前已经将本地仓库已经和GitHub远程仓库关联，使用git push即可将本地代码提交到GitHub。

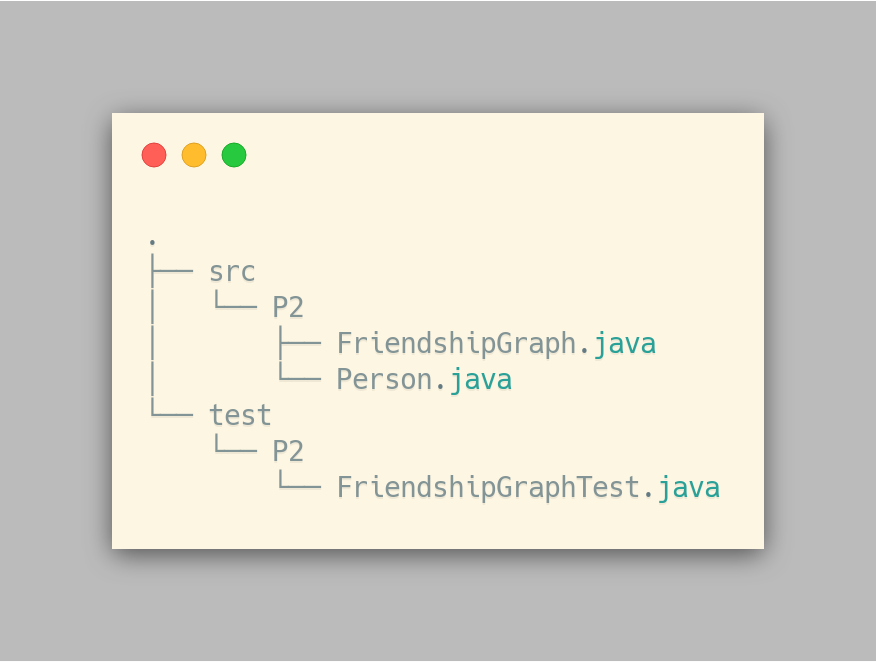


图 3‑5Lab2项目目录结构树状示意图

## Playing Chess

### ADT设计/实现方案

在此次实验中，我设计了三个子包：base，用于声明一些基类供其它的类继承；chess，用于实现国际象棋相关的类；go，用于实现围棋相关的类。

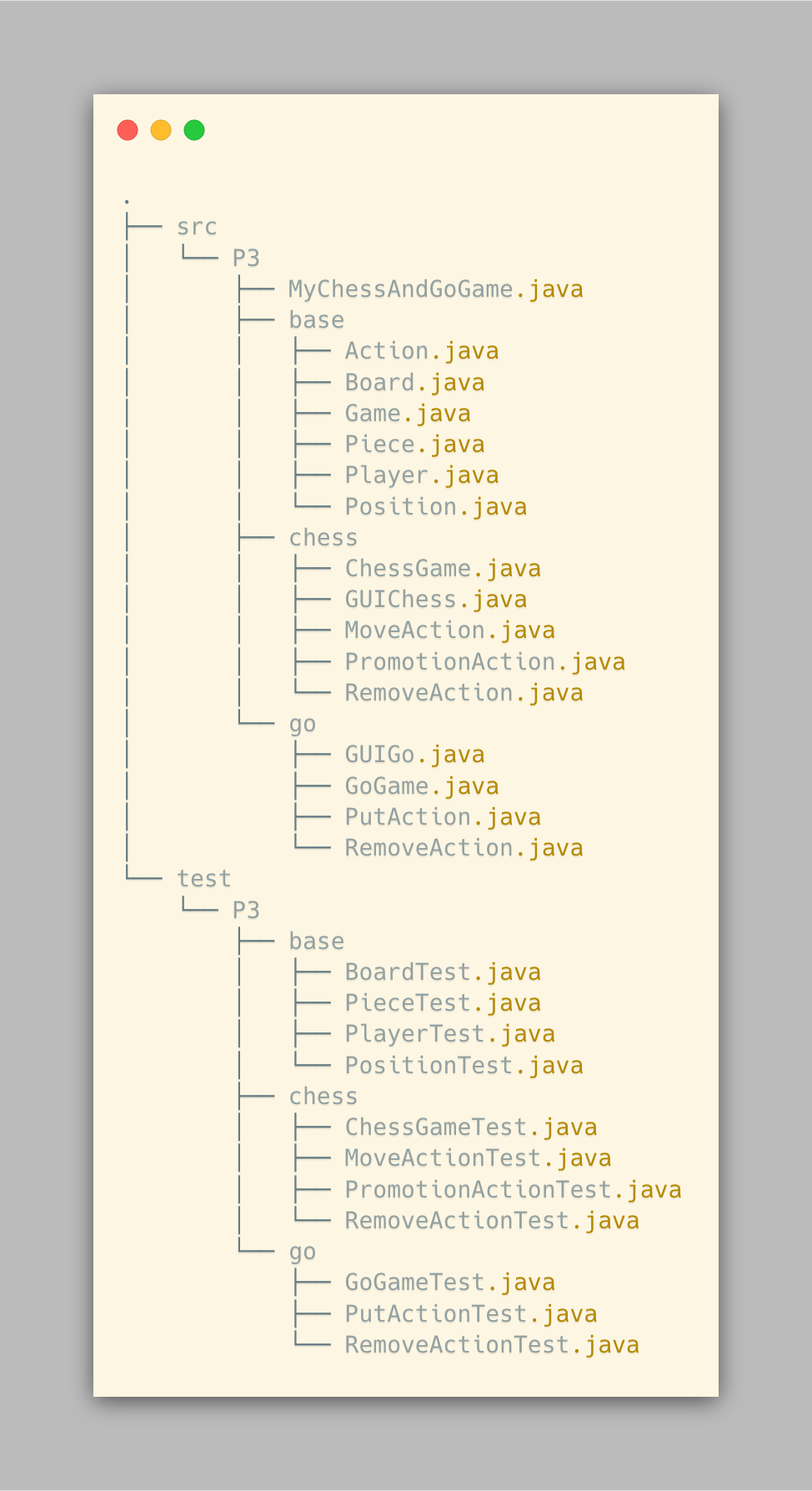


图 3‑6Lab3项目目录结构树状示意图

根据题目要求，我在base包中实现了Game、Player、Board、Piece、Position和Action类；在chess包中实现了自己的各种Actions、对应的Game类ChessGame以及GUI客户端GUIChess；在go包中实现了自己的各种Actions、对应的Game类GoGame以及GUI客户端GUIGo。

Action是一个存储一个放子、提子等动作的接口，用于抽象。它拥有方法doAction来将表示的动作作用于一个玩家上。

Position类用于存储棋子的位置，包含两个int用于存储一个二维坐标，是一个immutable类。

Piece类用于存储棋子，包含一个String类表示棋子的名字和一个Position类用于存储对应的位置，为一个mutable类。

Player类是一个玩家的抽象，用于存储一个玩家的名字和拥有的棋子，开放了接口用于查看名字和修改拥有的棋子。

Board类用来存储一个棋盘的大小。

Game抽象类用来表示一场游戏，包含两个玩家、一个棋盘以及比赛的历史动作。开放了一系列接口来修改和查询。为了减少实现难度，此类的表示不变量十分简单，如棋子都在棋盘上和棋子不能重叠，而没有对于游戏规则约束。

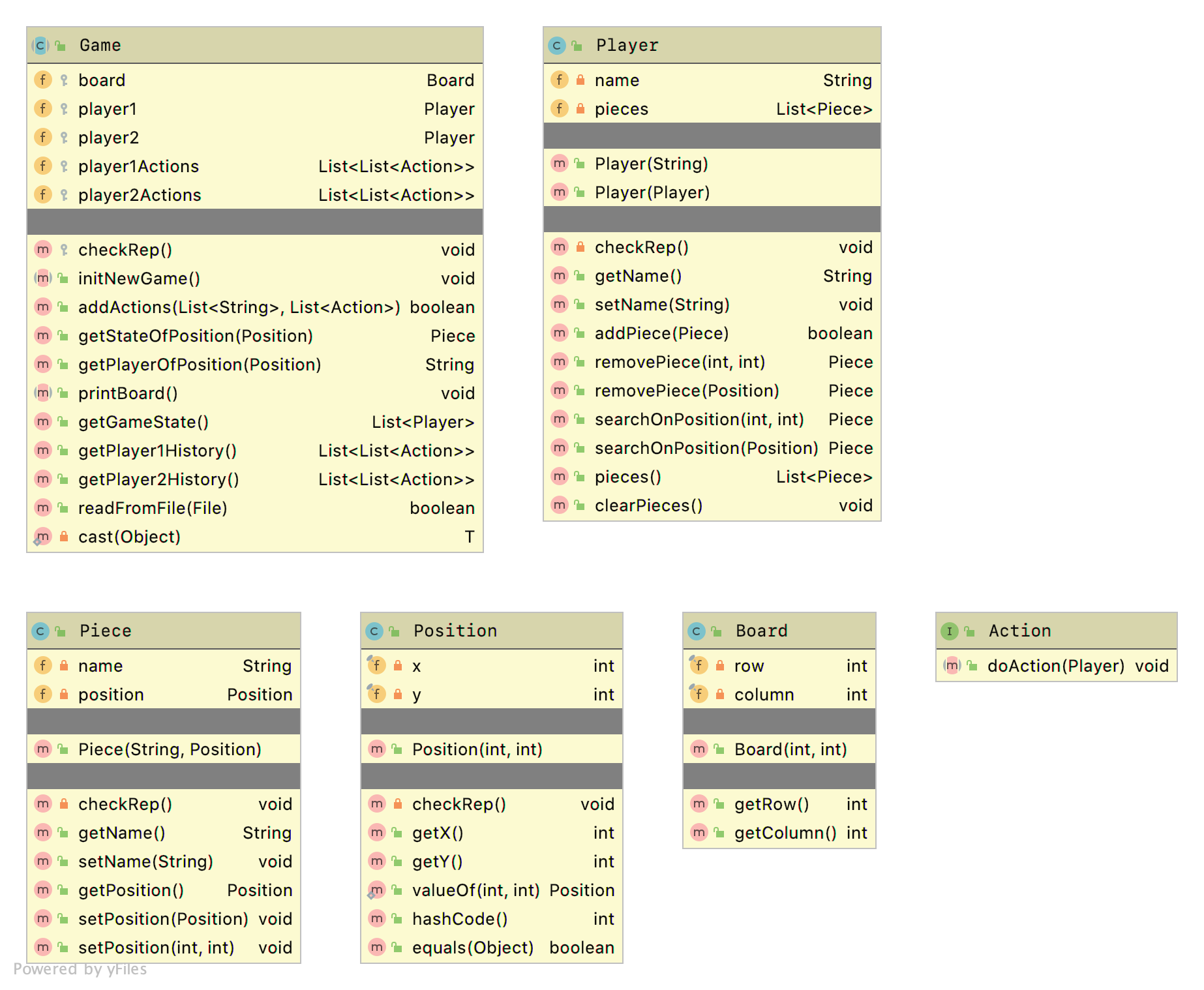


图 3‑7base包的各个类

对于chess包，ChessGame类继承Game类，特别地用于表示一局国际象棋游戏。而MoveAction，PromitionAction和RemoveAction类实现了国际象棋的移动，兵的变升和吃子。GUIChess则是ChessGame的GUI客户端。

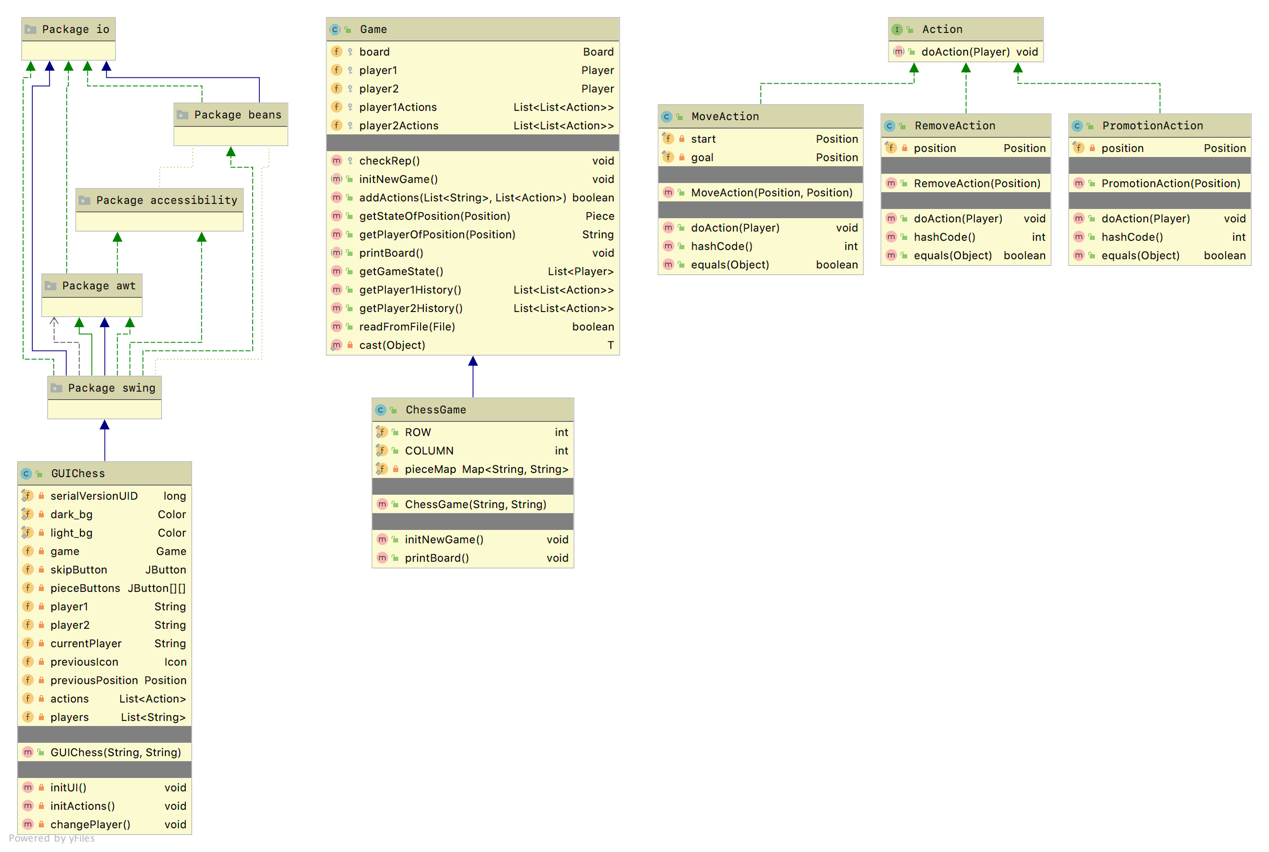


图 3‑8chess包的UML图

对于go包，GoGame类继承Game类，特别地用于表示一局围棋游戏。而PutAction和RemoveAction类实现了围棋的放子和提子。GUIGo则是ChessGo的GUI客户端。

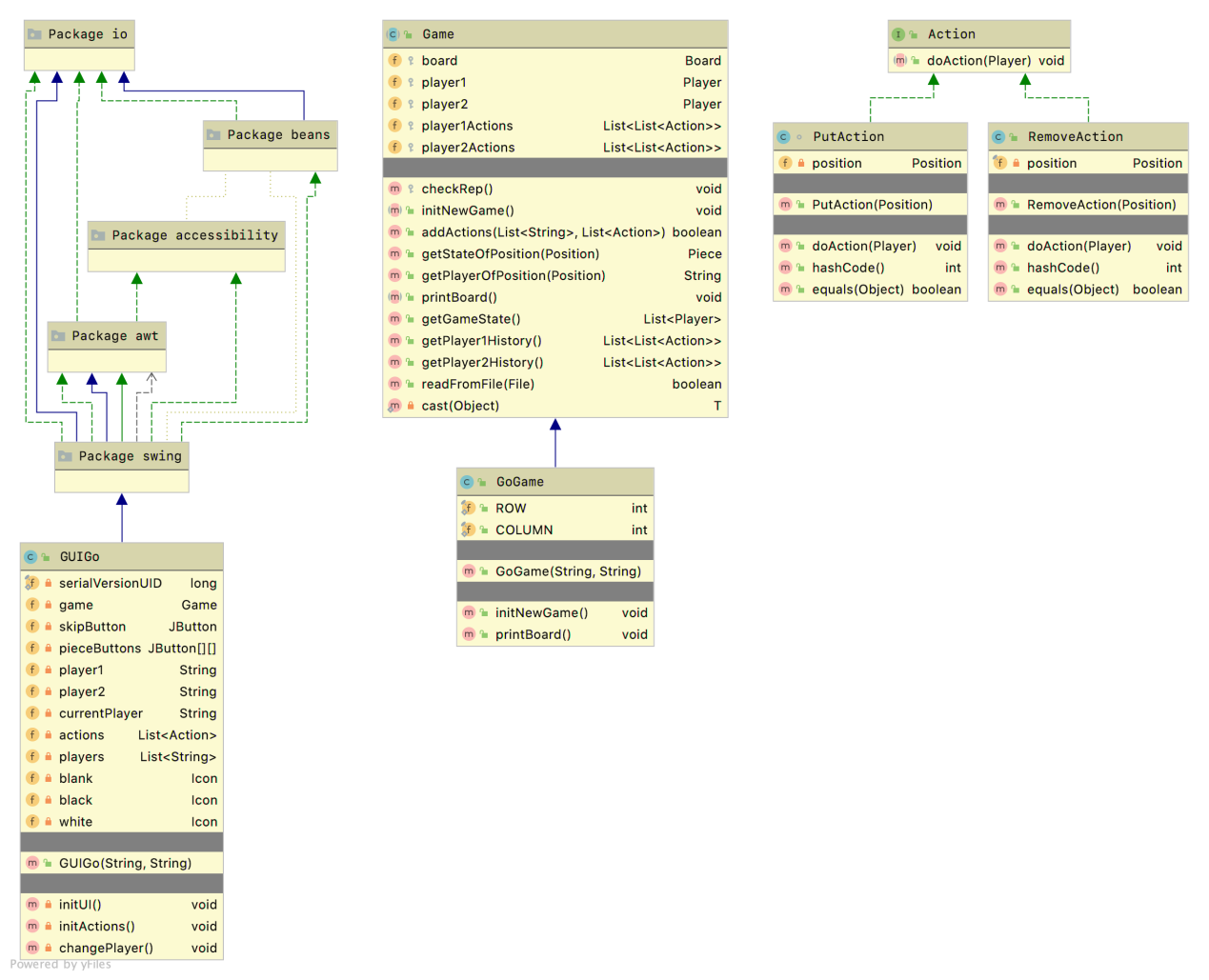


图 3‑9go包的UML图

### 主程序MyChessAndGoGame设计/实现方案

辅之以执行过程的截图，介绍主程序的设计和实现方案，特别是如何将用户在命令行输入的指令映射到各ADT的具体方法的执行。

MyChessAndGoGame使用GUI，GUI操作更加简单方便。

GUI的实现使用的是Swing库，通过按钮来实现各个动作，与Game后端进行交互。MyChessAndGoGame程序使用前面实现的GUIChess和GUIGo类，来实现国际象棋和围棋游戏。

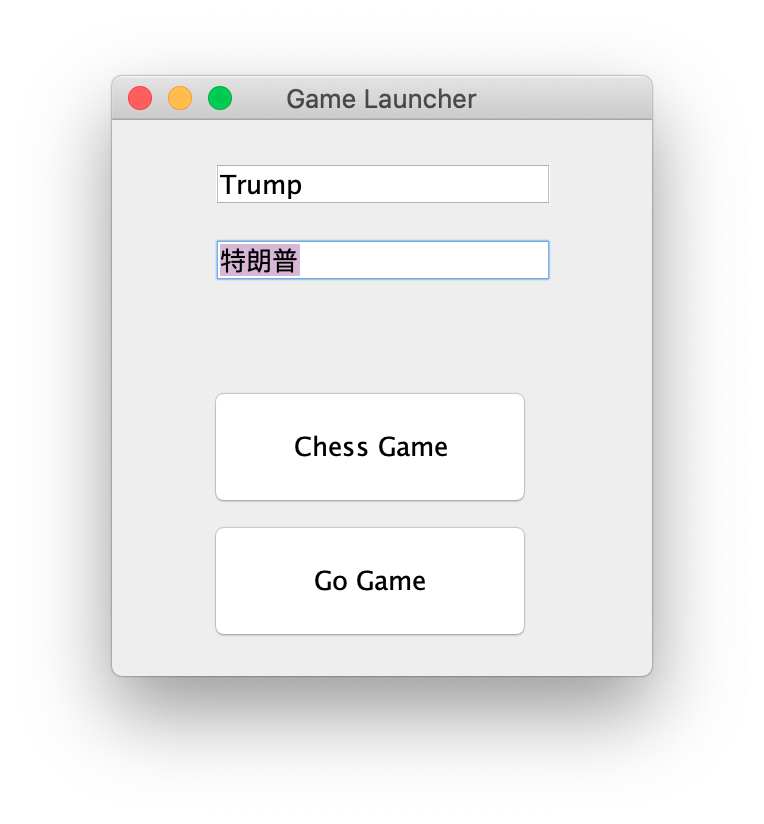


图 3‑10MyChessAndGoGame运行截图

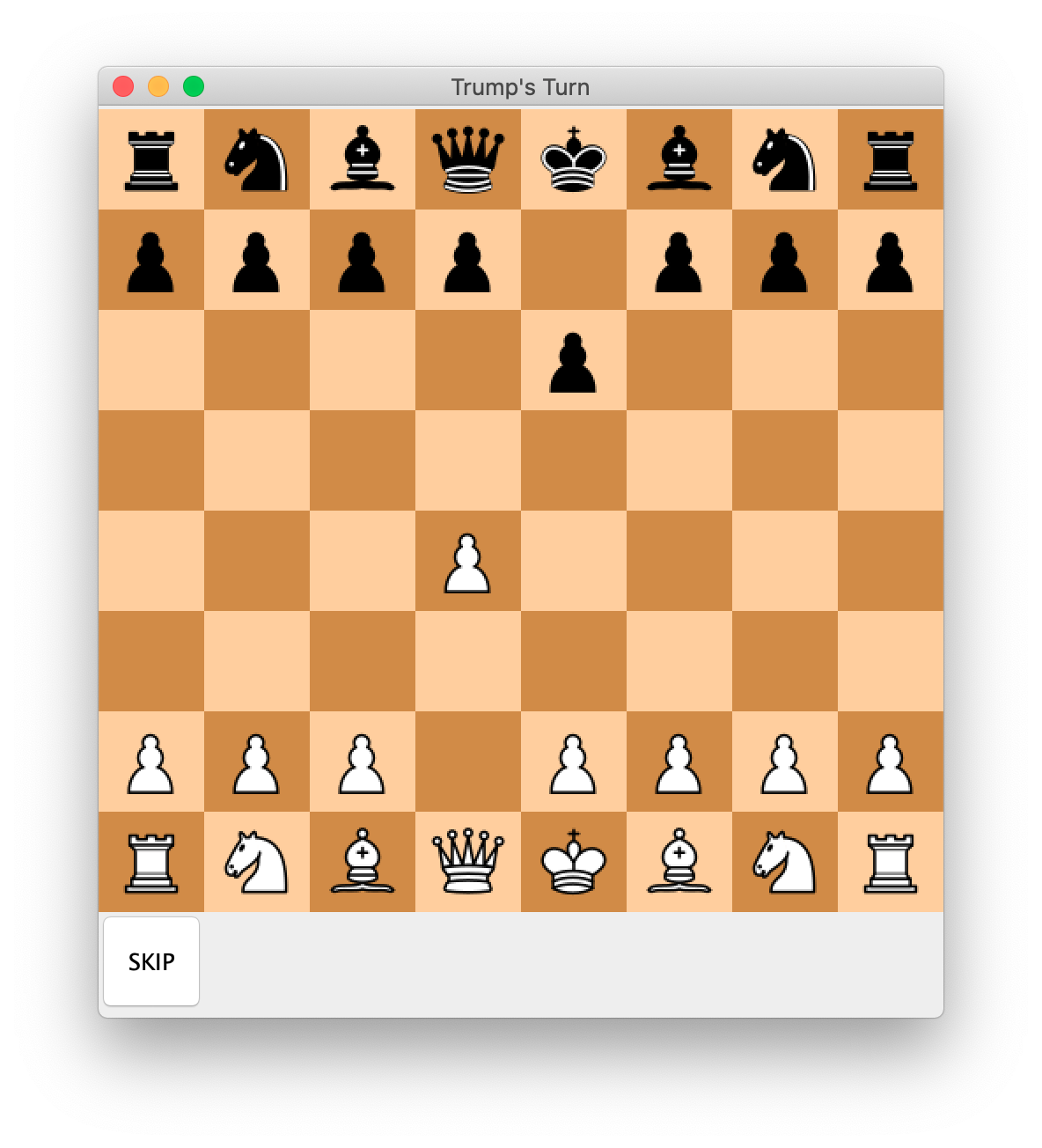


图 3‑11国际象棋运行截图

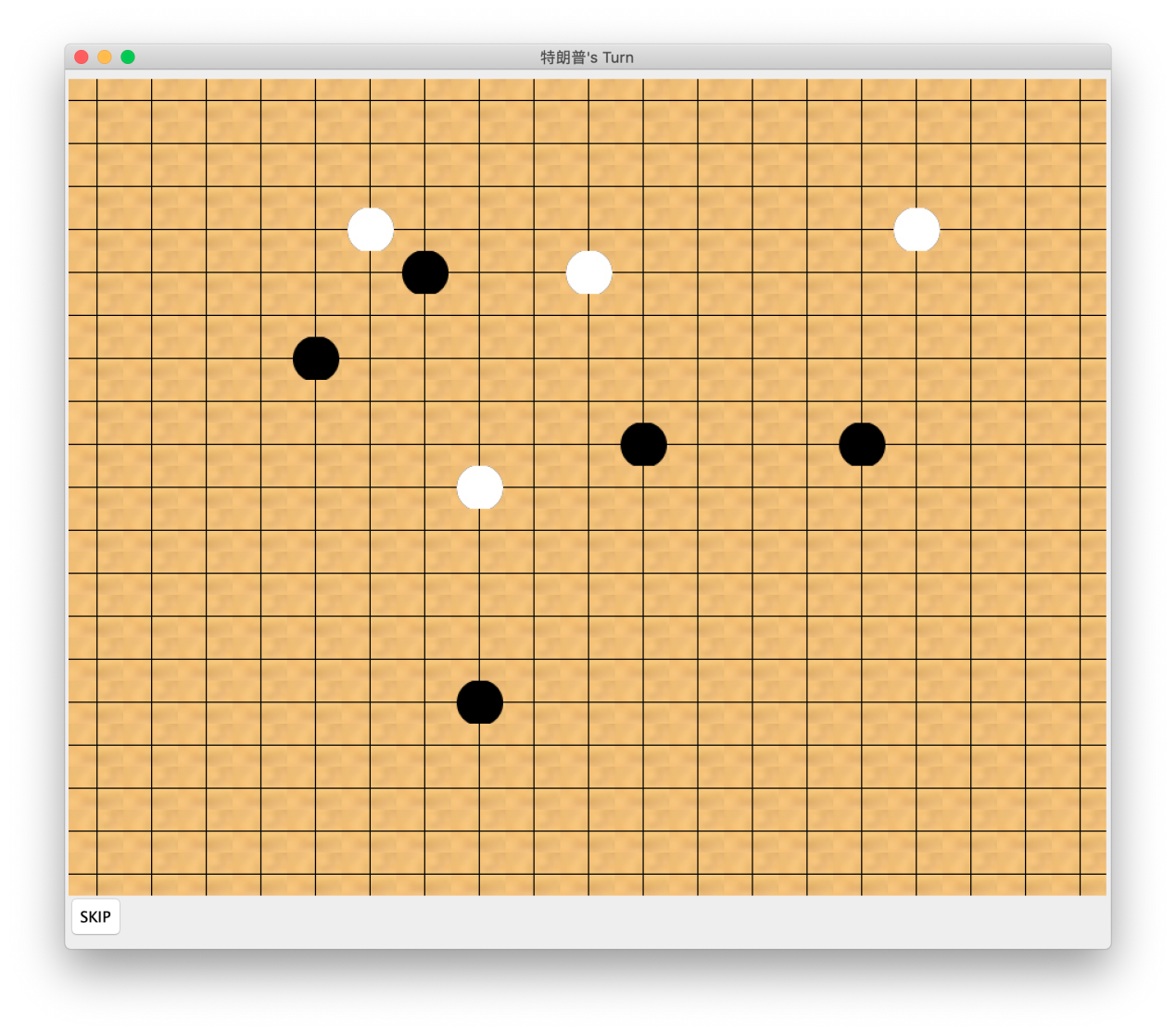


图 3‑12围棋运行截图

同时程序会在命令行中将当前游戏局面输出，以纪录游戏的历史。

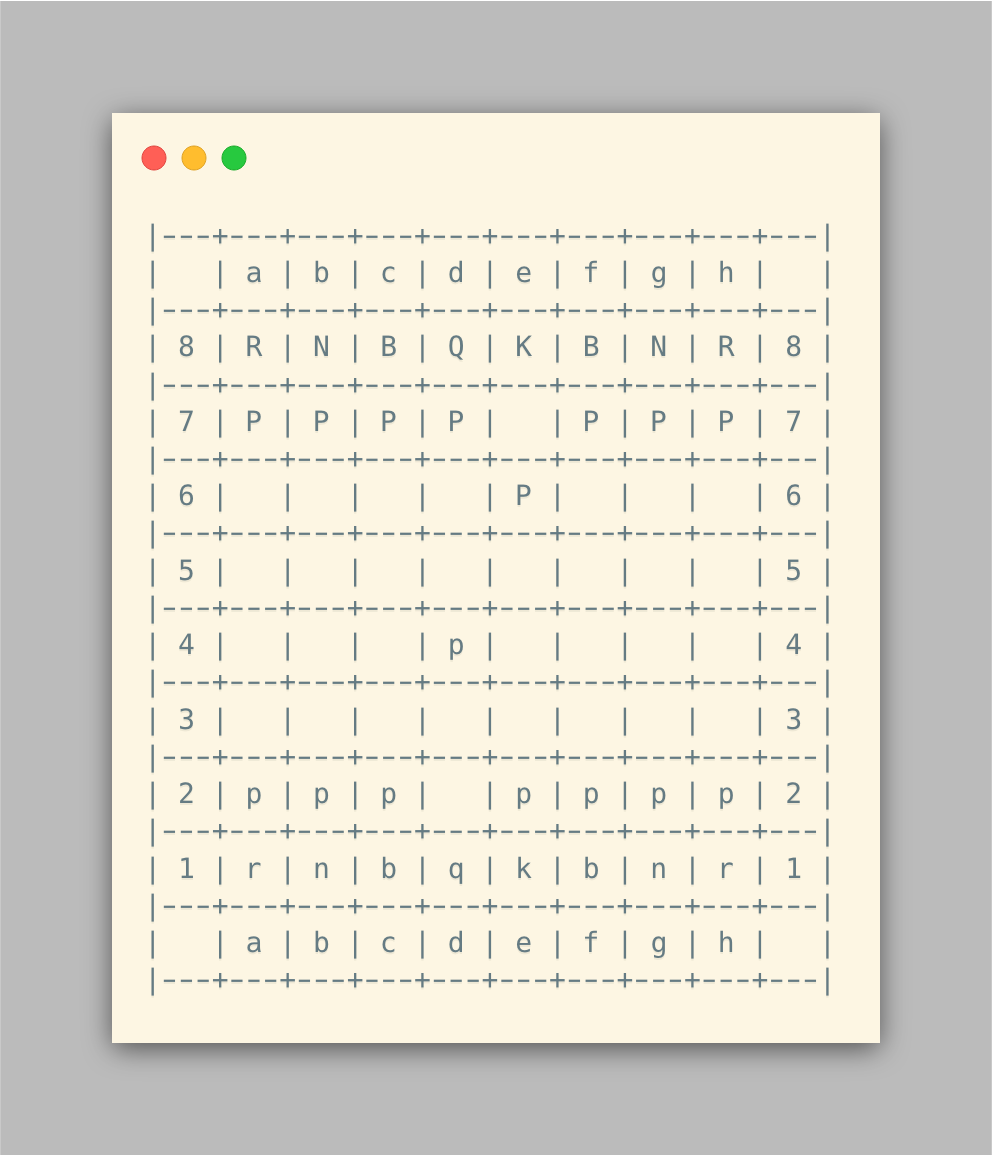


图 3‑13国际象棋终端输出

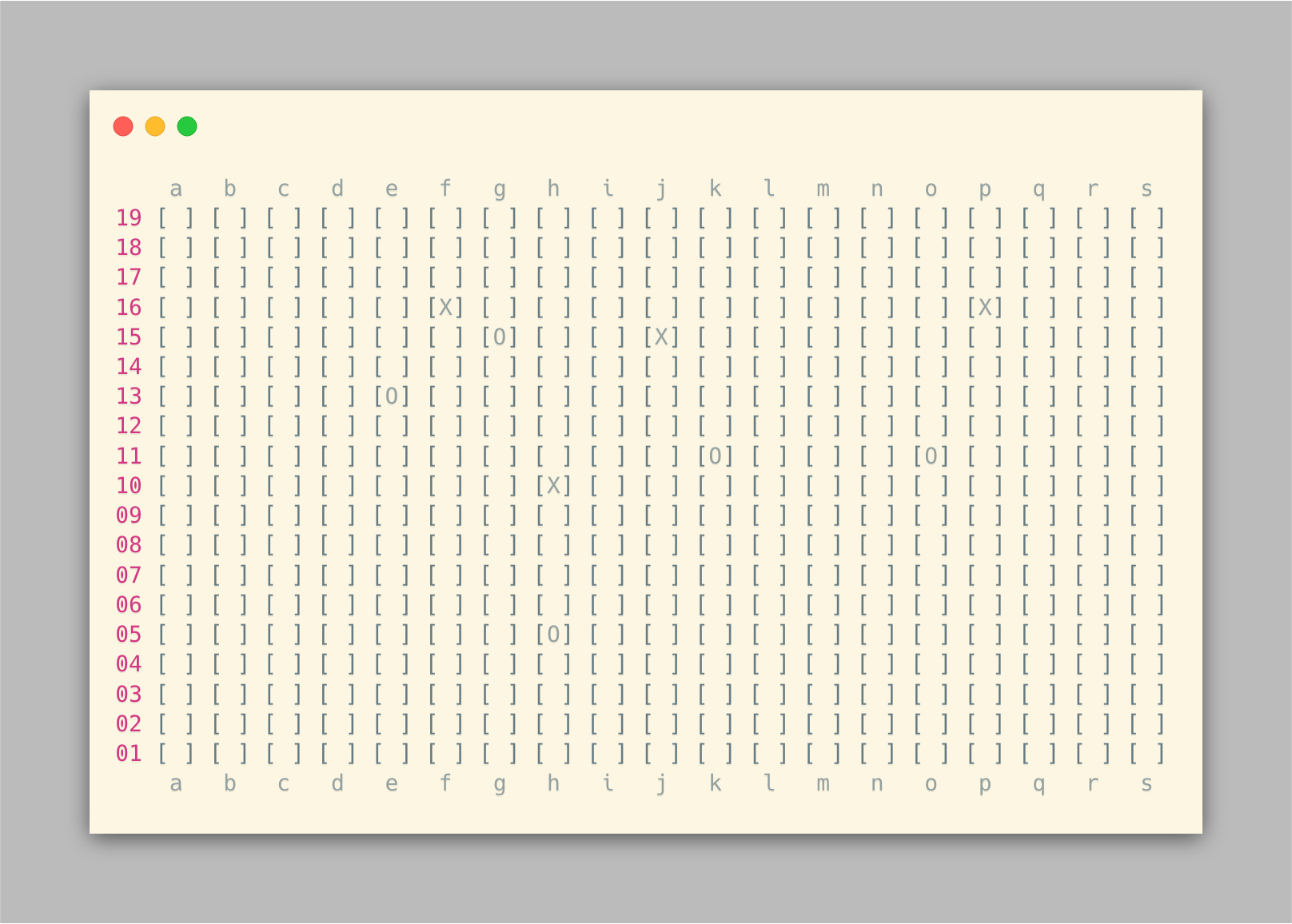


图 3‑14围棋终端输出

### ADT和主程序的测试方案

对于Action类，其测试策略合法动作和非法动作，具体实现根据。

对于Board类，测试其各个getter。

对于Piece类，测试其各个getter和setter。

对于Position类，测试各个其getter和重载的hashCode和equals。

对于Player类，其addPiece方法测试策略为原本是否拥有的棋子；removePiece方法测试策略为为删除的位置是否有棋子；searchOnPosition测试策略为搜索的位置是否有棋子；pieces测试策略为是否拥有棋子。

对于Game类及其子类，其addActions测试策略为players和actions是否等长，动作涉及一个还是两个玩家，动作列表是否为空；getStateOfPosition测试策略为对应位置有无棋子，对应位置的棋子为玩家一还是玩家二；getPlayerOfPosition测试策略为对应位置有无棋子，对应位置的棋子为玩家一还是玩家二；getPlayer1History和getPlayer2History的测试策略为对应玩家有无动作；readFromFile的测试策略为有无对应文件。

对于GUI客户端，此次实验中未使用自动化测试，而是使用个人手动测试。

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.23 | 18:00-19:30 | 完成P1 Problem1 | 完成 |
| 3.24 | 20:00-21:30 | 完成P1 Problem2 | 未完成，存在Bug |
| 3.25 | 18:00-19:30 | 调试P1 Problem2 | 完成 |
| 3.25 | 20:00-21:30 | 完成P1 Problem3 | 完成 |
| 3.26 | 18:00-19:30 | 完成P1 Problem4 | 完成 |
| 3.26 | 22:00-23:00 | 完成P2 | 完成 |
| 3.31 | 20:00-22:00 | 完成P3 base | 完成 |
| 4.4 | 20:00-22:30 | 完成P3 chess | 完成 |
| 4.5 | 20:00-22:30 | 完成P3 go | 完成 |
| 4.10 | 20:00-23:30 | 完成P3收尾 | 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 |
| FriendshipGraph无法继承ConcreteVerticesGraph<Person>类 | 通过编译器提示，发现是构造函数未添加public关键字 |
| P3的图形化 | 通过查阅网上资料学习 |

# 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

## 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？

ADT的抽象程度更高，可以降低代码的耦合度。在使用ADT进行编程时，前期的规划会花费更多的时间，但是编程时的难度会下降，而且更利于复用。而直接面向应用场景编程前期的编写会比较快，但是当程序较大时不利于维护。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛类的编程可以更加灵活，使代码的复用更加容易。如果不使用泛类，则会增加很多重复的代码。

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？

优势是这样的方法可以纯粹的为规约编写测试用例，可以覆盖绝大部份情况，同时之后编写代码时的目的性可以更强。这种测试方法的确很优秀，但是有时部分特殊情况会在编写时才会想到，所以它对于测试策略的要求较高。所以我比较喜欢之后再编写一些白盒测试。

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？

减少重复劳动，使开发更加简单。

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？

比较适应，这个任务的逻辑比较简单，所以对于这个场景还容易适应。相比与P1，P3的的难度要高很多，同时自由度更高。在实现P3前我更需要对于代码结构有更好的规划。前期我多次重构了代码的结构，在固定代码结构后编写就比较容易。

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？

这些工作可以方便程序员快速了解ADT的作用和需要注意的地方，方便之后维护和拓展。在以后的编程中，除了一些十分简单的辅助类，我愿意坚持这样做。

1. 关于本实验的工作量、难度、deadline。

本实验的主要难度在P3，因为需要自己设计代码结构。其它部分难度正常。实验总体工作量适中，deadline比较合适。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

软件构造对工程的开发很有用，可以让我获得更宏观的编程思路。暂无建议。