***井字棋产品***

**测试报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 日 期 | 版 本 | 说明 | 作 者 |
| 2020.4.3 | 1.0 |  | 曹修远 |
| 2020.4.4 | 1.1 | 添加了自己负责的内容 | 卓永康 |
| 2020.4.4 | 1.2 | 添加了自己负责的内容 | 蒋先强 |
|  |  |  |  |

**目 录**

[1引言 3](#_Toc35958945)

[1.1编写目的 3](#_Toc35958946)

[1.2背景 3](#_Toc35958947)

[1.3定义 3](#_Toc35958948)

[1.4参考资料 3](#_Toc35958949)

[2测试概要 3](#_Toc35958950)

[2.1动态测试内容和方法 3](#_Toc35958951)

[2.2静态测试内容和方法 4](#_Toc35958954)

[3 测试环境 4](#_Toc35958957)

[4测试总结 5](#_Toc35958958)

[4.1动态测试情况说明 5](#_Toc35958959)

[4.2缺陷情况说明 5](#_Toc35958960)

[4.3静态扫描情况说明 5](#_Toc35958961)

[4.4建议 5](#_Toc35958962)

[4.5评价 6](#_Toc35958963)

# 1引言

## 1.1编写目的

## 本测试报告为井字棋游戏功能测试报告，目的在于总结测试阶段的测试以及分析测试结果，描述系统是否符合需求，并依据结果对该产品做出评价和建议。作为测试质量参考文档提供给用户、测试人员、开发人员、项目管理者、其他质量管理人员和需要阅读本报告的高层经理阅读。

## 1.2背景

a.被测试软件名称：井字棋游戏

b.任务提出者：朱群涛、卓永康、徐成、陶永新、曹修远、蒋先强

开发者：朱群涛、卓永康、徐成、陶永新、曹修远、蒋先强

目标用户：任何对该游戏感兴趣的用户

本软件在下列环境中均通过测试：

1、Java version 11.0.5

Windows 10 18363

处理器:intel i7-9750h

内存：16Gb

IDE:IDEA 2019.3.3

2、Java version 10.0.2

Windows 7

处理器：intel i5-6300hq

内存：8Gb

3、Java version 12.0.1

Windows 10 19041

处理器：i5-7300hq

内存：16Gb

IDE:IDEA 2019.3.4

4、Java version 11.0.5

Windows 10

处理器：i5-7300hq

内存：8Gb

IDE:IDEA 2019.3.3

5、Java version 11.0.5

Windows 10

处理器：i5-6200u

内存：8Gb

需要指出的是，由于使用了最低版本为10.0.2的Java版本，若使用更早版本的Java可能对测试结果产生无法预测的影响。同时，在本软件中，操作系统以及硬件对于测试的影响较小。受限于条件，未测试软件在Mac OS中的运行情况。如有需要，请自行测试。

## 1.3定义

棋盘：指一个3x3大小的棋盘，棋子用x和o表示，空白的棋盘位使用“ • ”表示。

下棋：对于用户来说，指输入有效合法下棋坐标并点击回车键确认的过程。

对于计算机来说，指程序根据预置算法，对用户的下棋进行相应的回应。

获胜方：参照游戏规则，当有一方任意三个标记形成一条直线，则为获胜。

合法输入：指从键盘输入的棋盘坐标或“exit”命令，且坐标在指定范围内。

注：第一版中，合法输入的形式为“r,c”，且必须为英文逗号。第二版中另行规定。

PVP: 指玩家对战玩家（Player versus player）

PVE: 指玩家对战环境（Player VS Environment）

## 1.4参考资料

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 标题 | 文件名称 | 发表日期 | 资料来源 |
| 1 | 井字棋的定义以及规则 | 百度百科-井字棋 | N/A | https://baike.baidu.com/item/井字棋/4467444?fr=aladdin |

# 2测试概要

## 2.1动态测试内容和方法

### 2.1.1 测试内容

第一版：

|  |  |  |
| --- | --- | --- |
| 测试  设计 | 测试内容 | 对应测试脚本 |
| 测试1 | 测试接受从键盘输入 | GetInputTest.Java |
| 测试2 | 用户输入下棋坐标后，能否作出正确响应，更新棋盘 | GetMoveTest.Java |
| 测试3 | 电脑分析计算自己下一步的赢棋步骤是否正确 | TestToWinTest.java |
| 测试4 | 测试是否能检测出来自键盘的不合法输入并给出相应提示 | CheckInputTest.Java |
| 测试5 | 用户下完棋后电脑下棋的落子位置是一定顺序的 | GetComMoveTest.Java |
| 测试6 | 测试计算某下棋序列中各棋子个数是否正确 | TestWayTest.java |
| 测试7 | 测试此次落子后是否有人获胜 | TestWinTest.java |
| 测试8 | 测试棋盘打印是否符合要求 | PrintMatTest.java |
| 测试9 | 测试是否能正确返回指定序列中各棋子个数 | ChessBoardTestWayTest.Java |
| 测试10 | 测试能否正确判定是否获胜 | ChessBoardTestAnyOneWinTest.Java |
| 测试11 | 测试能否正确返回包含某棋子的所有获胜序列 | ChessBoardGetMyWinListTest.Java |

第二版：

|  |  |  |
| --- | --- | --- |
| 测试  设计 | 测试内容 | 对应测试脚本 |
| 测试1 | 测试是否实现选择模式与退出功能 | GameSystemTest.Java |
| 测试2 | 测试是否能正确返回指定序列中各棋子个数 | ChessBoardTestWayTest.Java |
| 测试3 | 测试能否正确判定是否获胜 | ChessBoardTestAnyOneWinTest.Java |
| 测试4 | 测试能否正确返回包含某棋子的所有获胜序列 | ChessBoardGetMyWinListTest.Java |

### 2.1.2 测试方法

*在此说明针对不同测试项所使用的测试方法（包括黑盒测试方法和白盒测试方法）。*

第一版：

|  |  |  |
| --- | --- | --- |
| 测试  设计 | 测试内容 | 对应测试方法 |
| 测试1 | 测试接受从键盘输入 | 等价类划分、逻辑覆盖 |
| 测试2 | 用户输入下棋坐标后，能否作出正确响应，更新棋盘 | 等价类划分、判定覆盖 |
| 测试3 | 电脑分析计算自己下一步的赢棋步骤是否正确 | 等价类划分 |
| 测试4 | 测试是否能检测出来自键盘的不合法输入并给出相应提示 | 判定条件覆盖 |
| 测试5 | 用户下完棋后电脑下棋的落子位置是一定顺序的 |  |
| 测试6 | 测试计算某下棋序列中各棋子个数是否正确 | 等价类划分 |
| 测试7 | 测试此次落子后是否有人获胜 | 等价类划分 |
| 测试8 | 测试棋盘打印是否符合要求 | 等价类划分，判定覆盖 |
| 测试9 | 测试是否能正确返回指定序列中各棋子个数 | 等价类划分 |
| 测试10 | 测试能否正确判定是否获胜 | 等价类划分 |
| 测试11 | 测试能否正确返回包含某棋子的所有获胜序列 | 完全覆盖 |

第二版：

|  |  |  |
| --- | --- | --- |
| 测试  设计 | 测试内容 | 对应测试方法 |
| 测试1 | 测试是否实现选择模式与退出功能 | 基本路径测试、语句覆盖 |
| 测试2 | 测试是否能正确返回指定序列中各棋子个数 | 等价类划分 |
| 测试3 | 测试能否正确判定是否获胜 | 等价类划分 |
| 测试4 | 测试能否正确返回包含某棋子的所有获胜序列 | 完全覆盖 |

## 2.2静态测试内容和方法

### 2.2.1 测试内容和方法

使用IDEA中的阿里巴巴编码规约（Alibaba Java Coding Guidelines）进行了代码静态扫描

### 2.2.2 对规范的理解

*在此选择Java开发规范中的若干项（2~3项即可），围绕当前开发的产品，举例说明自己对该规范的理解。如果实在找不到例子，也可以结合自己以往的开发经验，对照谈谈自己对该规范文档中部分规范的理解。*

*【对规范的理解】范例如下：*

*《阿里巴巴Java开发手册》中指出：【推荐】如果模块、接口、类、方法使用了设计模式，在命名时体现出具体模式。 说明：将设计模式体现在名字中，有利于阅读者快速理解架构设计理念。*

*根据我的实践体会举例如下：在代码“public class OrderFactory;”中，类名称的命名就体现了工厂模式的使用。*

1、本次开发中，体会有很多，但是体会最深的是，阿里巴巴开发规范禁止在程序中使用“魔法值”，也就是未经定义的常量直接出现在程序中。很多时候，作为条件判断或者for语句的条件，我们经常在条件中直接写上数字。看似简便，但是却会带来很多问题。比如在GameSystem类里，一开始我直接使用了数字作为chooseMode方法的判断条件。后来才写了FLAG将他们储存起来。直接使用常量，看似简便，却会带来很大的问题。会大大的降低代码的可读性，给合作的开发人员、测试人员带来很多问题。这个规范在《Clean Code》和《重构》里都提及过，甚至在所有涉及代码规范的文章和书籍里都有类似的。

SICP里传达过这样一个观点——“代码首先是给人读的，只是恰好可以执行！”，虽然有些极端，但可以看出可读性的重要性。

另外一点就是注释的规范性，比如，开发规范中禁止行尾注释，类里必须要添加作者信息，类注释必须使用JavaDoc形式等等。

还有一个我平时一直都忽略的细节。先看这句判断代码

**if** (contentList.**length** >= 1 && contentList[0].equals(**"exit"**))

这句在扫描时报出了一个Critical级别的错误：Object的equals方法容易抛空指针异常，应使用常量或确定有值的对象来调用equals。

这是我以前从未注意到的，所以代码应该改为如下形式：

**if** (contentList.**length** >= 1 && **"exit"**.equals(contentList[0]))

--曹修远

2、本次开发令我印象最深的代码规范是在 if/else/for/while/do 语句中必须使用大括号。即使只有一行代码，避免采用单行的编码方式：if (condition) statements;平时习惯于不加大括号，比如此次的TestToWin.java中有

**if**(numX == 2)   
 **return true**;

就很不符合规范。因为这种代码会让读代码的人不容易发现，所以以后要多加注意才行。

--卓永康

3、本次开发，我发现了许多我平时都不太去注意注意的代码规范问题。比如说在平时敲代码过程中，我都习惯在行尾注释，但这在阿里巴巴开发规范中是禁止的，还有对于常量、方法、类等一系列命名，我都不够规范，比如此次的PrintMat.java中for(int i = 0; i < MAXIM; i++)，我一开始是直接用的数字3代替的行列最大维度，这就没有很好的可读性，自己或许清楚其意，但自己写的代码不仅仅是给自己看的。

--蒋先强

4.《手册》中指出 常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。类内共享常量：直接在类内部 private static final 定义。包内共享常量：即在当前包下单独的 constant 目录下。比如在此次开发过程中的第一版重构版版代码ChessBoard中类内共享常量就定义如下

**private static final int**[][] ***WIN\_LIST*** ={{1,2,3},{4,5,6},{7,8,9},{1,4,7},{2,5,8},{3,6,9},{1,5,9},{3,5,7}};

同时包内共享常量统一保存在常量类Constans中：

**public final class** Constants {

**public static final int *CHESSMAN\_X*** = 1;

**public static final int *CHESSMAN\_O*** = 2;

**public static final int *CHESSMAN\_BLANK*** = 0;

}

通过这种方式对常量进行命名和定义，能够提高一定的代码可读性，同时可以减少魔法值的使用。 --朱群涛

# 3 测试环境

本软件在下列环境中均通过测试：

1、Java version 11.0.5

Windows 10 18363

处理器:intel i7-9750h

内存：16Gb

IDE:IDEA 2019.3.3

2、Java version 10.0.2

Windows 7

处理器：intel i5-6300hq

内存：8Gb

3、Java version 12.0.1

Windows 10 19041

处理器：i5-7300hq

内存：16Gb

IDE:IDEA 2019.3.4

4、Java version 11.0.5

Windows 10

处理器：i5-7300hq

内存：8Gb

IDE:IDEA 2019.3.3

5、Java version 11.0.5

Windows 10

处理器：i5-6200u

内存：8Gb

需要指出的是，由于使用了最低版本为10.0.2的Java版本，若使用更早版本的Java可能对测试结果产生无法预测的影响。同时，在本软件中，操作系统以及硬件对于测试的影响较小。受限于条件，未测试软件在Mac OS中的运行情况。如有需要，请自行测试。

本次测试设计，仅使用了Junit4版本，无法预计在其他Junit版本中的运行情况。

# 4测试总结

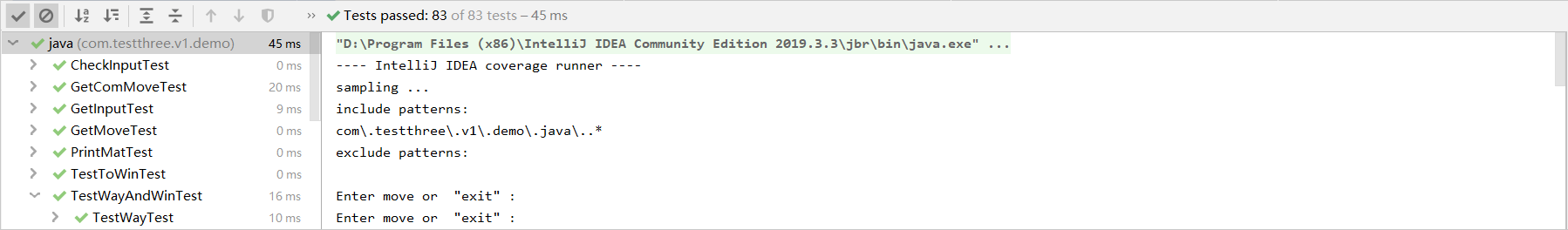
## 4.1动态测试情况说明

*本节主要关注测试脚本的运行情况。*

*通过使用相关覆盖指标来说明（例如，测试对功能点的覆盖，测试对某些风险点的覆盖等），说明测试设计是否全面、合理。建议给出必要的运行界面截图来体现得到的结论。*

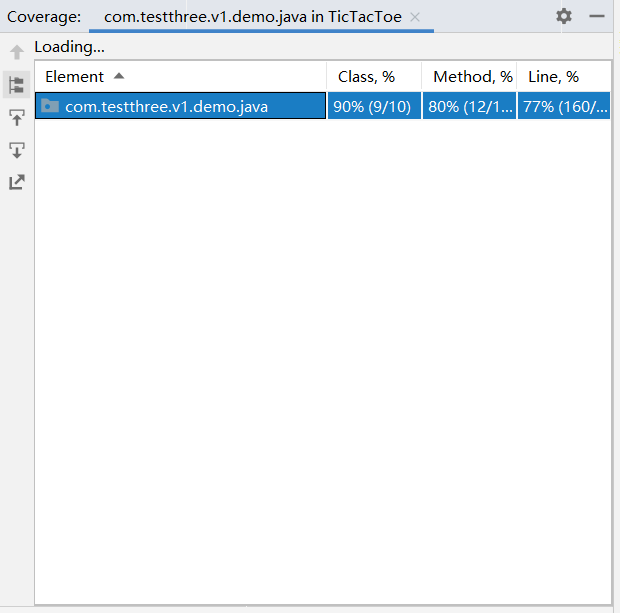
*陈述经测试证实了的本软件的能力。如果所进行的测试是为了验证一项或几项特定性能要求的实现，应提供这方面的测试结果与要求之间的比较，并确定测试环境与实际运行环境之间可能存在的差异 对能力的测试所带来的影响。*

1、版本1动态测试情况



版本1测试用例运行情况

版本1共有83个测试用例，运行全部通过。

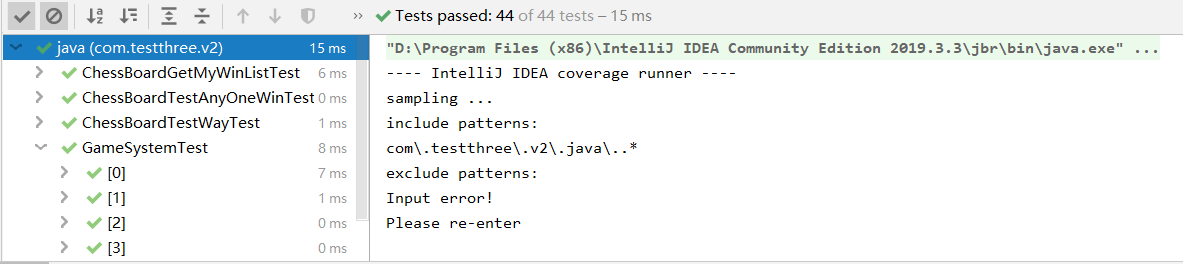


版本1测试用例覆盖情况

就覆盖情况来看，测试覆盖了90%的类和80%的方法。唯一一个没有被覆盖的类是Main类。Main类主要创建游戏对象，不需要进行测试。

这里需要指出的是控制游戏逻辑的GameDriver类中的gameDriver方法用循环控制整体游戏逻辑，暂未想到测试方法，所以没有针对该方法的测试。

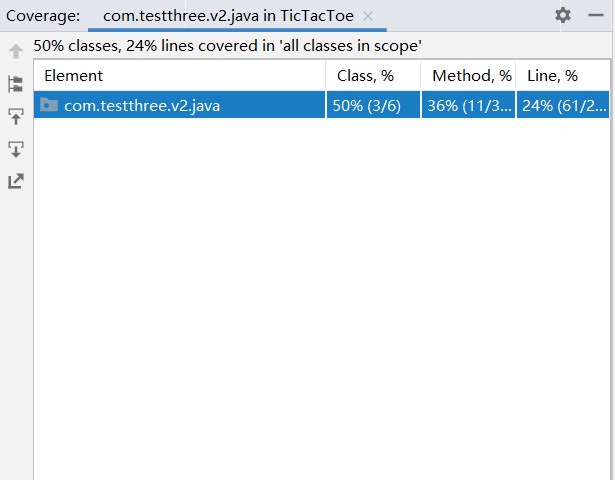
2、版本2动态测试情况：



版本2测试用例运行情况

版本2在版本1重构的基础上进行开发，所以版本2的测试仅包含了Refactored版本的部分测试以及新增的功能的测试。

版本2共有44个测试用例，运行全部通过。



版本2测试用例覆盖情况

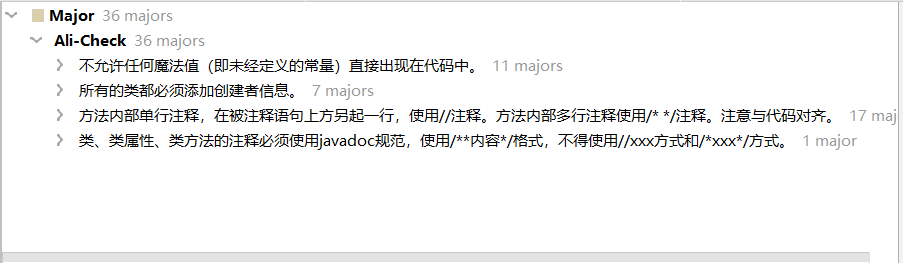
## 4.2缺陷情况说明

*通过分析测试用例的执行结果，以及对发现的缺陷进行分析，从功能的角度说明被测产品的质量如何，是否符合用户需求，至少是否符合需求规格说明的描述。*

*陈述经测试证实的软件缺陷和限制，说明每项缺陷和限制对软件性能的影响，并说明全部测得的性能缺陷的累积影响和总影响。*

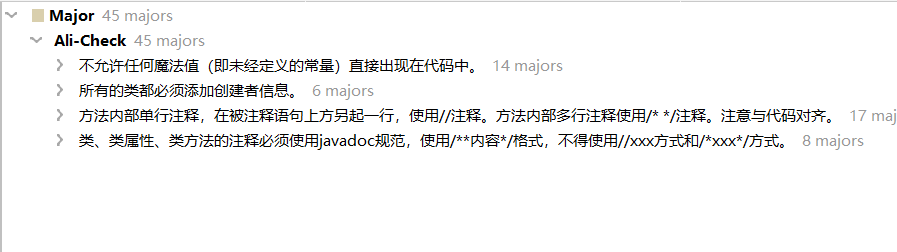
## 4.3静态扫描情况说明

1、版本1的demo版本：



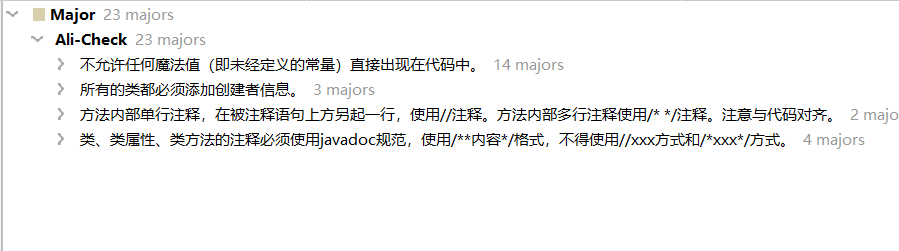
截至编写测试报告时，有36个Major级别的规范问题。没有Blocker级别和Critical级别的问题。问题主要集中在魔法值，注释不规范和没有创建者信息上。

2、版本1的重构(Refactored)版本：



截至编写测试报告时，有45个Major级别的规范问题。没有Blocker级别和Critical级别的问题。问题主要集中在魔法值，注释不规范和没有创建者信息上。

3、版本2：



截至编写测试报告时，有23个Major级别的规范问题。没有Blocker级别和Critical级别的问题。问题主要集中在魔法值，注释不规范和没有创建者信息上。

最后，总体来看，问题都是Major级别的不规范，没有出现Blocker级别和Critical级别，这一点还是值得肯定的。其次，问题集中在魔法值，注释规范等我们平常容易忽略的方面，对我们未来的开发给出了很好的建议。

## 4.4建议

*对每项缺陷提出改进建议，如：*

1. 各项修改可采用的修改方法；
2. 各项修改的紧迫程度；
3. 各项修改预计的工作量；
4. 各项修改的负责人。

## 4.5评价

*说明该项软件的开发是否已达到预定目标，能否交付使用。*