EWNJavaSystem 开发手册

安徽大学计算机博弈实验室

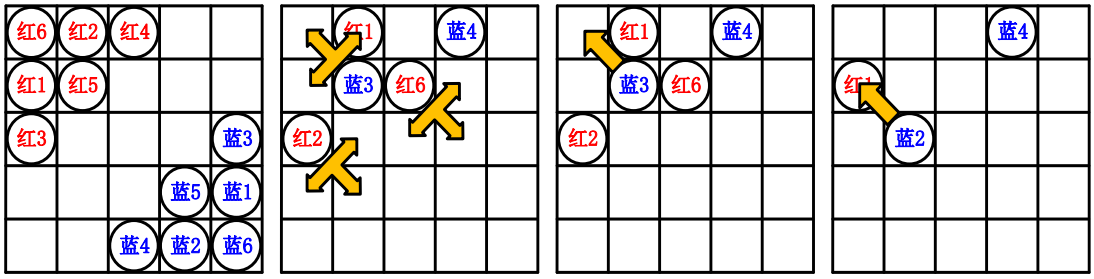
2019 年 3 月

目录

1 爱恩斯坦棋介绍

爱恩斯坦棋，德语为 EinStein Würfelt Nicht!，简称 EWN，是德国的教授 Ingo Althöfer 为纪念爱因斯坦相对论诞生 100 周年而发明的游戏，于 2005 年德国爱因斯坦年官方展览。爱恩斯坦棋的德语直译为“爱因斯坦不投骰子!”，它的发明来源于爱因斯坦说过的一句话：“我相信上帝不玩骰子”。

爱恩斯坦棋的棋盘由5×5大小的正方形棋位组成，且对弈方分别标记为红方和蓝方，红蓝双方分别有标记为 1~6 号数字的棋子。双方走子前需要对棋盘布局，布局位置如图 1.1(a)所示，红方的棋子可在棋盘左上角的六个棋位随意放置，蓝方的棋子在右下角的六个棋位随意放置。



(a) 初始布局　(b) 合法移动方向　(c) 到达角部位置　(d) 吃掉全部棋子

图 1.1 爱恩斯坦棋规则

完成棋盘布局后，双方轮流行棋，先行方即可以是红方也可以是蓝方。行棋分为两个步骤：投骰子和走动棋子。

首先，行棋方投掷骰子，根据骰子点数来确定本次行棋可走动的棋子。如果棋盘上存在编号与骰子点数相同的棋子，则只能走动该棋子，否则可在大于或小于骰子点数并且接近该点数的棋子中选择一个走动。例如在图 1.1(b)中，假设投掷到的骰子点数为 3，蓝方只能走动蓝 3 棋子，红方可走动红 2 和红 6 中的任意一个棋子。

其次，行棋方走动指定的棋子。允许的移动方向如图 1.1(b)所示，红方棋子可向右方、下方、右下方走动，蓝方棋子可向左方、上方、左上方走动。每次行棋只能走动一格棋位，且不允许跳格。如果目标棋位上存在棋子，则该棋子被移出棋盘（即该棋子被吃掉），既可吃己方棋子也可吃对方棋子。

对弈获胜的方式有两条，满足任意一条即可获胜：第一，己方的棋子率先走

动到对方角部棋位，如图 1.1(c)所示，蓝 3 走动到左上方顶点位置后获胜；第二，己方率先吃掉对方全部棋子，如图 1.1(d)所示，蓝 2 吃掉红 1 后获胜。

中国大学生计算机博弈大赛对比赛时间和先后手进行了限制。甲乙双方共对阵 7 局，第 1、4、5 局甲方先手，第 2、3、6、7 局乙方先手，先胜 4 局者为胜（此为国赛规则，并非校赛规则）。每局每方限时 4 分钟，超时判负。

2 环境配置与系统运行

2.1 硬件环境

EWNJavaSyetem 可在个人电脑上进行开发，其硬件配置需求如表 2.1 所示。

表 2.1 硬件运行环境

硬件需求	推荐配置
CPU	Intel® Core™ i3 及以上 CPU
内存大小	4GB 及以上
硬盘大小	10GB 以上
显示器分辨率	建议 1920*1080

2.2 软件环境

操作系统：Microsoft Windows 10、8、7 64 位

编程语言：Java8

开发环境：Apache Maven 3.6.0

开发工具（IDE）：Eclipse Photon Release (4.8.0)

2.3 项目导入

打开 Eclipse IDE 编程工具，点击工具栏中的“文件”→“导入”按钮，会弹出“导入”窗口。在该窗口中选择“Maven”→“Existing Maven Projects”选项（最新的 Eclipse 默认安装了 Apache Maven 工具，如果没有“Maven”选项，则需要手动安装 Maven 或下载最新版 Eclipse），并单击“下一步”，此时会弹出“Import Maven Projects”窗口，如图 2.1 所示。之后，单击“Browse…”按钮，选择“EWNJavaSystem”文件夹，并点击“完成”按钮即可导入项目。导入后的项目文件可在“包资源管理器”中查看，如图 2.2 所示。

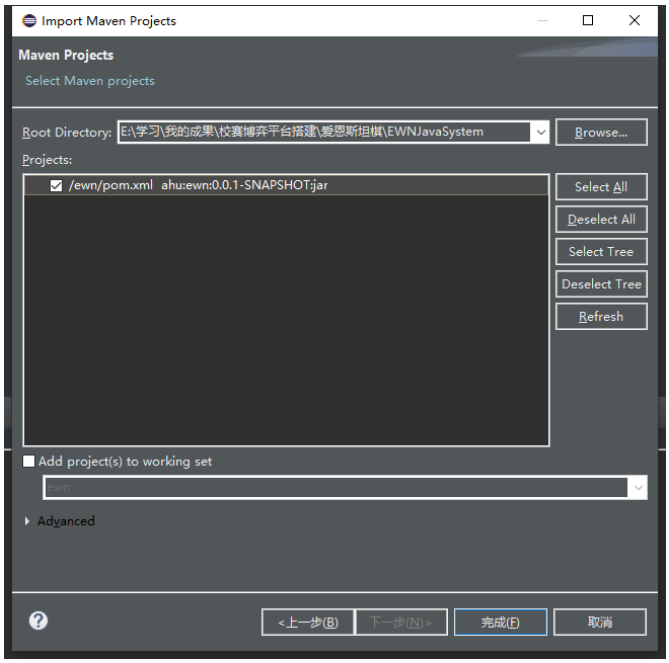


图 2.1 Maven 项目导入

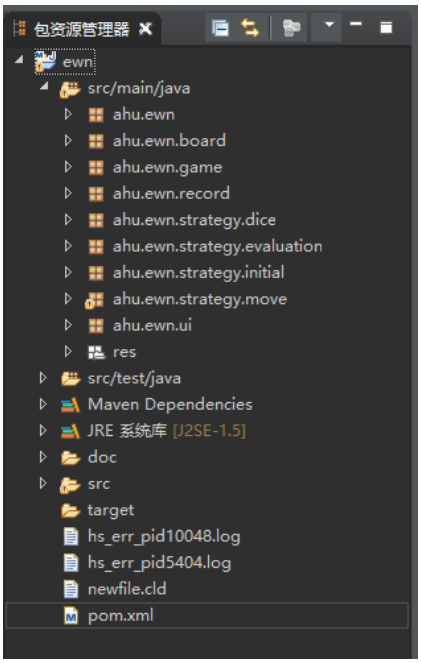


图 2.2 项目文件

2.4 系统运行

本系统的执行入口位于 ahu.ewn 包的 App 类，运行 App 类后即可显示系统界面，如图 2.3 所示。

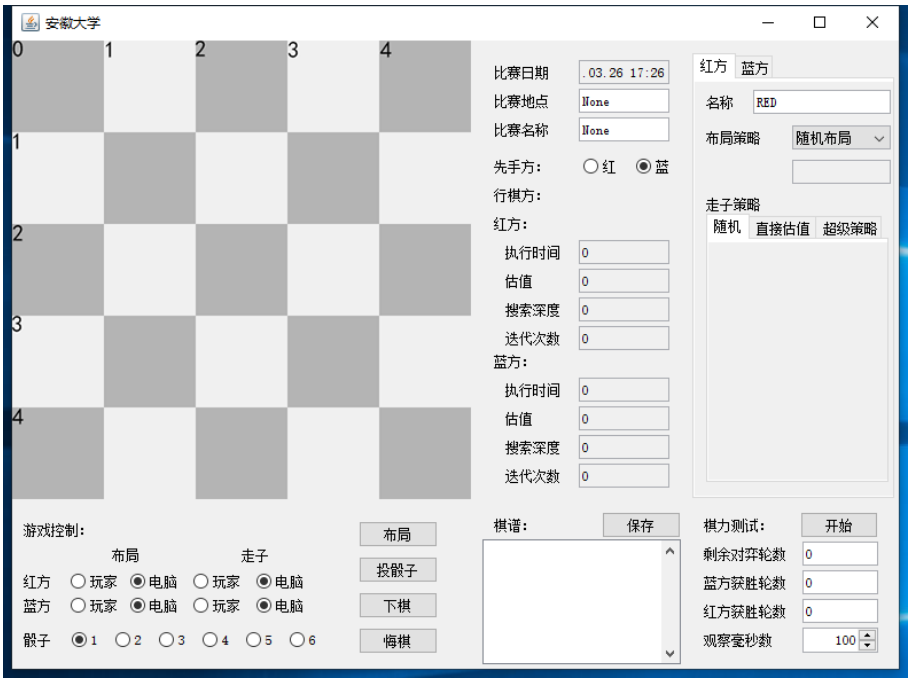


图 2.3 EWNJavaSystem 主界面

3 系统使用说明

如图 3.1 所示，系统界面分为六个功能面板：棋盘显示面板、游戏控制面板、数据显示面板、棋谱面板、策略配置面板和自动对弈面板。

图 3.1 功能面板分布

3.1 棋盘显示面板

棋盘显示面板负责棋盘的显示和玩家下棋功能。由于爱恩斯坦棋分为布局和行棋两个阶段，游戏开始后，该面板会显示初始的棋盘布局情况，布局结束后显示当前的对弈棋盘。

如果用户在游戏控制面板中设置了“玩家布局”或“玩家走子”，那么用户可以通过拖动棋子实现布局或走子。在“玩家布局”中，用户可以通过鼠标“拖动”一方的所有棋子，并且被拖动的棋子可以覆盖目标位置上的棋子，以实现恢复残局功能（如果对弈中途系统意外崩溃，可以恢复棋局）。在“玩家走子”中，用户只能拖动符合规则的棋子，并且棋子只能拖动到符合规则的棋位上。例如在图 3.1 中，如果蓝方的骰子点数为 2，那么用户只能拖动蓝 2 棋子到左、上、左上方三个棋位上，蓝 4 和蓝 5 无法拖动。

3.2 游戏控制面板

游戏控制面板负责控制对弈的进程，设置红蓝方“玩家、电脑”走子，以及骰子点数。

该面板右排第一个按钮为开始/重置按钮。游戏未开始时显示“布局”字样，点击按钮后进入布局阶段，按钮显示“确认”字样，此时可以进行布局。布局完成后点击“确认”进入行棋阶段，按钮显示“重置”字样。点击“重置”时直接结束当前游戏，开始新游戏的布局阶段。第二个按钮为投骰子按钮，点击后可以随机生成一个骰子点数。第三个按钮为下棋按钮，如果是“电脑下棋”，则点击下棋按钮后系统就会根据指定的策略行棋，如果是“玩家下棋”，用户可以拖动符合规则的棋子，确认无误后点击下棋按钮即可行棋。第四个按钮为悔棋按钮，点击一次悔一步棋，直至悔至初始布局棋盘。

游戏控制面板的左上方包含红蓝双方的人/机布局和人/机走子按钮组，用于切换人-人、人-机、机-机对弈。左下方包含骰子点数按钮组，可以直接指定一个骰子点数。

3.3 数据显示面板

该面板用于显示对弈的数据，能显示红蓝双方的走子策略数据（执行时间、估值、搜索深度、迭代次数，详见 MoveStrategy 类的 API 文档）、比赛信息（时间、地点、名称）、先手方和行棋方。

3.4 棋谱面板

棋谱面板用于显示和保存棋谱，即对弈过程，棋谱格式参考 2018 国赛棋谱标准（http://computergames.caai.cn/info/news180708.html）。棋谱中的比赛时间由系统自动设置，比赛地点和名称在数据显示面板中手动输入。

3.5 策略配置面板

策略配置面板用于配置红蓝双方的队名、布局策略和走子策略。

系统提供两种布局策略：随机布局和固定布局。随机布局是指六个棋子随机放置。固定布局中，用户需要手动输入 6 个数字 1~6，并点击确认按钮。系统会按从上往下，从左往右的顺序摆放棋子。例如：如果红蓝双方都输入：123456，则生成的初始布局如图 3.2 所示。

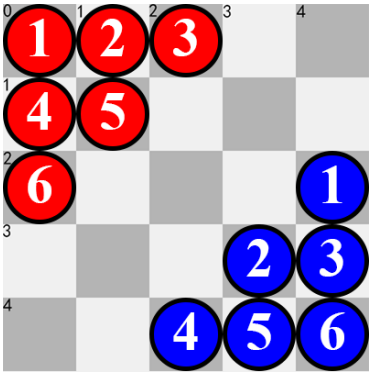


图 3.2 固定布局策略的摆放顺序

系统提供三种走子策略：随机走子、直接估值和自定义策略，对应的界面如图 3.3 所示。随机策略：在所有合法走法中随机选择一个走法进行行棋。直接估值：模拟所有的合法走法进行行棋，对行棋后得到的新棋盘进行估值（给棋盘打分），最终选择估值最高的走法。估值方法有两种：随机估值和超级估值，超级估值是系统为用户提供的接口，用户可以自己设计估值函数（输入是一张棋盘，输出是一个数字，详见 ahu.ewn.strategy.evaluation.MySuperEvaluate 类）。

系统提供的第三种超级策略是自定义的策略接口，用户可以自己设计算法，从合法走法中选择合适的走法（详见 ahu.ewn.strategy.move.MySuperMove 类）。

（a）随机策略　　（b）直接估值策略　　（c）自定义策略

图 3.3 走子策略

3.6 自动对弈面板

自动对弈面板可以实现机机对弈的自动化，可用于测试红蓝双方的胜率。如果想使用此功能，用户需在“剩余对弈轮数”中输入自动对弈的轮数，并将红蓝双方的“获胜轮数”清零（输入“0”）。点击“开始”按钮后，系统会设置红蓝方为“电脑布局”和“电脑下棋”，并自动地布局→下棋→布局→…，统计胜负情况。自动对弈期间会自动切换先行方，可以随时点击“暂停”按钮暂停下棋。

自动对弈面板中的观察毫秒数用于控制自动对弈的速度，即两次行棋会间隔一段时间，毫秒数设置范围：[0~20000]。

注：自动对弈期间不建议使用其他面板，可能会出错。

3.7 对弈操作过程

一局对弈的完整操作过程如下：

1. 输入比赛地点、名称；
2. 输入红蓝方队名，选择布局策略和走子策略；
3. 选择先手方；
4. 设置红蓝双方是“玩家”还是“电脑”来布局/下棋；
5. 点击“布局”按钮，按 3.1 节所述并完成布局操作；
6. 点击“确认”按钮，进入行棋阶段；
7. 点击“投骰子”按钮或手动输入骰子点数；
8. 如果“电脑”行棋，直接点击“下棋”按钮，完成行棋动作；如果“玩家”行棋，按规则拖动棋子，确认无误后点击“下棋”按钮，完成行棋动作；
9. 返回第 7 步，直至游戏结束。

行棋过程中，可以点击“悔棋”按钮悔一步棋，可以随时点击“重置”按钮重置游戏，也可以随时调整双方的策略，随时设置“玩家”、“电脑”下棋。

自动对弈的完整操作过程如下：

1. 输入比赛地点、名称；
2. 输入红蓝方队名，选择布局策略和走子策略；

3. 输入“剩余对弈轮数”；
4. 点击“开始”按钮。

4 系统设计与开发

4.1 棋盘表示

棋盘表示是指棋盘局面在计算机中的表示和存储方法。常用的表示方法有数组和比特棋盘表示法。数组表示法是将棋盘以数组的形式存储于计算机中，数组的索引对应棋盘的棋位，值代表棋位上的棋子编号。本文采用数组表示法表示棋盘，于 ahu.ewn.board.ChessBoard 类中。

首先，对棋子进行编码。爱恩斯坦棋棋盘中一个棋位有 13 种状态，分别是存在红方的 1~6 号棋子、存在蓝方的 1~6 号棋子，以及不存在棋子。因此本文用 13 个常量对棋子进行编码，对应关系如表 4.1 所示，个位表示棋子的编号，十位表示棋子颜色，1 为蓝色，2 为红色，数字 0 表示无棋子。

表 4.1 棋子编码

棋子	无	蓝1	蓝2	蓝3	蓝4	蓝5	蓝6	红1	红2	红3	红4	红5	红6
编码	00	11	12	13	14	15	16	21	22	23	24	25	26

其次，对棋盘进行编码。棋盘采用5×5大小的数组来表示，数组的行索引和列索引分别对应棋盘从上至下、从左至右的行号和列号，数组的内容为棋子的编号。以图 3.2 为例，棋盘的数组形式如下：

{{21,22,23,00,00},
{24,25,00,00,00},
{26,00,00,00,11},
{00,00,00,12,13},
{00,00,14,15,16}}

4.2 走法表示

本系统的走法定义于 ahu.ewn.game.Move 类中。走法由棋子 Piece 和移动方

向 MoveDirection 组成。为了统一移动方向，本系统将移动方向分为：左方（Left），右方（Right），前方（Forward）。移动方向是以棋子为本体，以敌方的阵营为指向而定的，如图 4.1 所示。

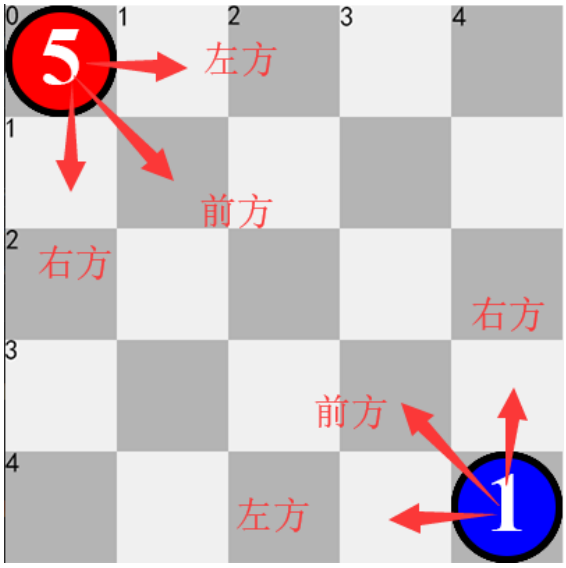


图 4.2 移动方向的定义

4.3 估值函数接口

自定义估值策略位于 ahu.ewn.strategy.evaluation.MySuperEvaluate 类中，用于可以自己实现以下方法。实现后，可以在界面中的“直接估值”里的“超级估值”使用该估值。

```
public double getValue(ChessBoard board, PieceType type) {
        //------------------------------------------------------------
        //在这里完成你的评估函数，计算并返回棋盘对玩家 type 的价值，值越大越对玩家 type 有利
        //例如：
        //PieceType winner = board.getWinner();
        //if(winner == type) return 1;   //我方获胜
        //else if(winner != PieceType.NULL) return -1;   //敌方获胜
        //else return 0; //没有人获胜
        //------------------------------------------------------------

        //默认返回值为 0，评估函数设计完成之后，可以将下面一行删除
        return 0;
}
```

4.4 走子策略接口

系统提供了一个更直接的策略接口：ahu.ewn.strategy.move.MySuperMove 类。该类包含一个关键的抽象方法 getMove()。输入参数为当前的游戏状态和骰子点数，输出为合法的走法。用户可以参考以下代码，在此处完成自己的策略。代码实现后，可以在界面中的“超级策略”里使用该策略。

```
public Move getMove(GameState gameState, byte dice) {
        // TODO 自动生成的方法存根
        //获取当前的棋盘
        ChessBoard board = gameState.getCurrentBoard();
        //获取当前的玩家颜色
        PieceType player = gameState.getCurrentPlayer().getTurn();
        //通过走法产生器得到符合规则的走法
        Map<Move, ChessBoard> legalMoves = MoveGenerator.getLegalMovesByDice(board, player, dice);

        //-----------------------------------------------
        //在这里完成你的策略，从 legalMoves 中选择一个 Move，可以参考 StaticEvaluationMove
        //例如
        //Move bestMove = ... ;
        //
        //如果希望界面能显示策略的价值、搜索深度、迭代次数等信息，就必须在这里对 value、maxDepth、visitNum 赋值
        //this.value = 0;
        //this.maxDepth = 0;
        //this.visitNum = 0;
        //
        //return bestMove;
        //-----------------------------------------------

        //系统默认输出随机的走法，策略编码完成后可以将这一行删除
        return new RandomMove().getMove(gameState, dice);
    }
```

4.5 代码结构说明

该系统包含的包和描述如表 4.2 所示。

表 4.2 包名及描述

名称	描述
ahu.ewn	主程序，ahu 表示安徽大学，ewn 表示爱恩斯坦棋
ahu.ewn.board	用于描述棋盘的相关类：棋盘、棋子、颜色等
ahu.ewn.game	用于描述游戏的相关类：玩家、走法、游戏状态等
ahu.ewn.record	用于定义棋谱
ahu.ewn.strategy	策略的基类和各种具体实现：骰子策略（dice）、估值函数（evaluation）、布局策略（initial）、走子策略（move）
ahu.ewn.ui	用于生成界面（无需看懂）

对弈环境和下棋策略的定义是博弈系统的核心数据结构。该系统的核心类结构图如图 4.3 所示。其中，“GameState”类表示一场对局，包括当前棋盘“currentBoard”、对弈双方“Player”和棋谱“Record”等属性，以及下棋“step()”和悔棋“pushback()”等控制对弈推演过程的核心方法。

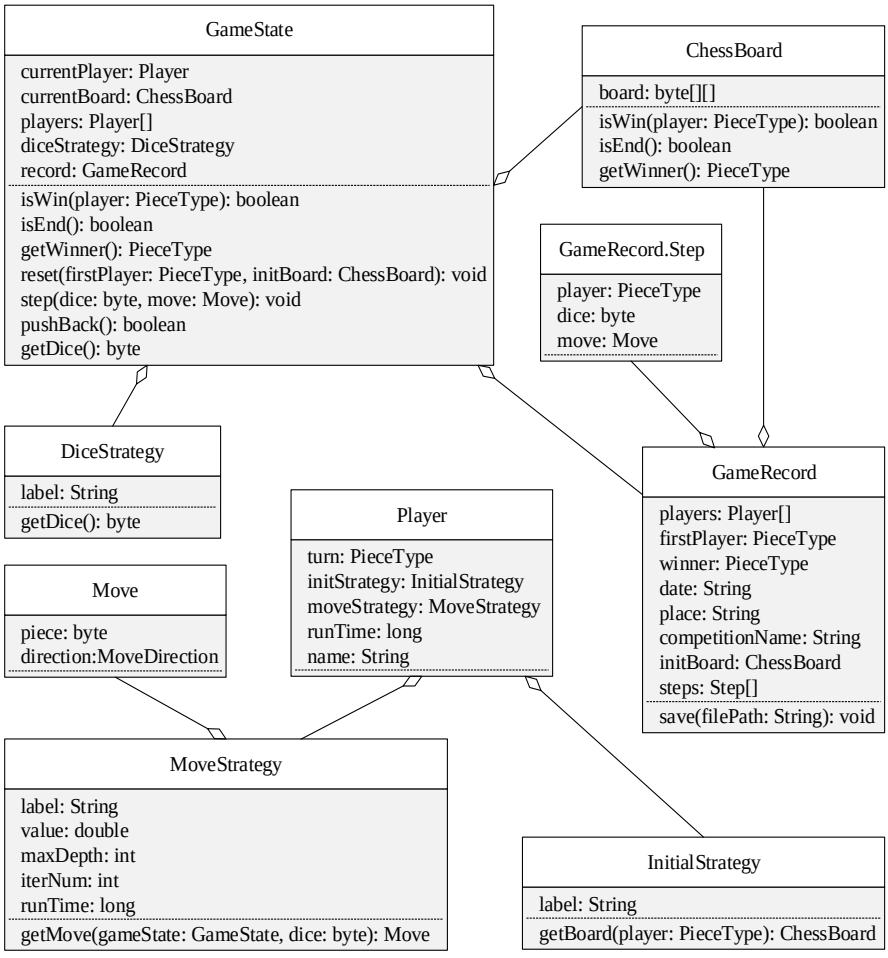


图 4.3 EWNGameSystem 核心类图

4.6 API 文档说明

本系统提供完整的类属性和方法说明文档（ahu.ewn.ui 除外），用户可依次打开“EWNJavaSystem\ewn\doc”文件夹，双击“index”文件，即可在浏览器中查阅文档。也可以直接阅读源代码，将鼠标移动到函数上即可显示函数说明。

4.7 棋力测试示例

策略编写好后，一般都会测试策略的胜率，下面给出胜率测试代码示例，该代码位于 ahu.ewn.Test 类中：

```
public class Test {

    public static void main(String[] args) {
        // 对弈轮数
        int gameNum = 10000;
        // 蓝方获胜轮数
        int blueWinNum = 0;
        // 红获胜轮数
        int redWinNum = 0;
        // 先手方，即蓝方先下棋
        PieceType firstPlayer = PieceType.BLUE;
        // 指定蓝方的布局策略和下棋策略
        Player bluePlayer = new Player(PieceType.BLUE, new StaticInitial(), new MySuperMove());
        // 指定红方的布局策略和下棋策略
        Player redPlayer = new Player(PieceType.RED, new StaticInitial(), new StaticEvaluationMove(new MySuperEvaluate()));
        // 定义一局游戏，并设置玩家为上面定义的玩家
        GameState game = new GameState();
        game.setPlayer(bluePlayer);
        game.setPlayer(redPlayer);
        // 对弈轮数的迭代
        for(int cnt = 1; cnt <= gameNum; cnt++) {
            // 重置游戏，生成初始布局
            game.reset(firstPlayer);
            // 红蓝双方轮流行棋，直至游戏结束
            while(game.isEnd()==false) {
                // 随机生成一个骰子点数
                byte dice = game.getDice();
                // 行棋方的走子策略产生一个合法走法
```

```
                    Move move = game.getCurrentPlayer().getMoveStrategy().getMove(game, dice);
                    // 执行合法走法，即下一步棋
                    game.step(dice, move);
                }
                // 统计获胜方
                if(game.getWinner()==PieceType.BLUE) blueWinNum += 1;
                else redWinNum += 1;
                // 切换先手方
                firstPlayer = firstPlayer==PieceType.BLUE? PieceType.RED: PieceType.BLUE;
            }// for(int cnt = 1; cnt <= gameNum; cnt++)
            // 打印结果
            System.out.println("blue vs red: "+blueWinNum+" vs "+redWinNum);
        }
}
```