

|  |
| --- |
| **JAVA 程序设计报告** |
| ——Snooker游戏开发 |
|  |
| 本文简单介绍了斯诺克游戏的背景及模型抽象，并给出一些合理的假设，使开发一个真实感很强的斯诺克游戏成为可能。 |
|  |
| **Ricky** |
| **2010-6-2** |
|  |

**第一部分 简介**

1. 引入

Snooker，中文一般作：斯诺克，本意是“阻碍、障碍”，所以斯诺克台球有时也被称为障碍台球。它流行于英国、爱尔兰、加拿大、澳大利亚和印度等英联邦国家以及香港[[1]](#footnote-1)等地，是一种高雅的双人游戏。游戏规则是两人轮流用球杆击白色主球，通过主球按一定的次序撞击球桌上的其他球入网得分，最后得分高者胜。

1. 背景

斯诺克游戏的起源一般认为是在公元1875年，由驻扎在印度的一位英国军官内维尔·鲍斯·张伯伦（Neville Bowes Chamberlain）和他的战友们首先发明的。

其实在斯诺克产生之前，台球游戏早就存在，而且有多种玩法。其中，有一种叫做“黑球入袋”（Black Pool）的玩法，在维尔·鲍斯·张伯伦所在的军队中非常流行。这种玩法用1个白球，15个红球和1个黑球。有一天，维尔·鲍斯·张伯伦和他的战友们觉得“黑球入袋”的玩法太简单、乏味，便决定增加黄色、绿色、粉色三个彩球上去。不久，他们还是嫌球不够，再加上了棕色球和蓝色球。这样，便形成了至今已风行全球的22个球的斯诺克台球。

据说，斯诺克台球的命名也与维尔·鲍斯·张伯伦有关。有一次，维尔·鲍斯·张伯伦同一个伙伴在打这种由他们新发明的22个球的台球时，一个很容易的进球，对方没有打中。他便顺口戏谑对方是斯诺克（斯诺克是那时当地军事院里对一年级新生的流行称法）。他这么顺口一叫，提醒了大家，使大家意识到，对于这种新的台球玩法，大家都是新手，都是斯诺克。于是，斯诺克的叫法便开始流行并固定下来。

而将斯诺克带回英国的是英国人约翰·罗伯特(John Roberts)。1885年，当时的英国英式台球冠军约翰·罗伯特在印度旅行时见到了张伯伦并从他那里知道了斯诺克的这种新玩法。于是，罗伯特就把斯诺克台球带回到英格兰。但是，当时在英国传统的BILLIARDS台球（英式台球）占据着主导地位，被认为是正统的和科学的玩法。斯诺克台球一时只能是民间的一种娱乐方式，难于登上大雅之堂。 一直到了20世纪30年代，英式台球日渐衰落，许多名手才逐渐转向斯诺克台球。其间出现了斯诺克大明星乔·戴维斯(Joe Davis)，斯诺克台球才真正开始在英国流行。[[2]](#footnote-2)

与高尔夫差不多，斯诺克要求选手在任何级别的比赛上，在任何时候都有要具备高标准的礼仪举止。无论是在地方联赛上，还是在职业联赛中的高级别赛上，这一规定均适用。

1. 在比赛开始和结束时，要和对手握手。
2. 在比赛开始和结束时，要和裁判握手。
3. 无论何时，都要将自己所有的犯规行为时行申报。
4. 不能站在对手的击球线上。
5. 在比赛仍在进行时，不能随意对对手的运气进行评论。
6. 在比赛仍在进行时，不能对比赛形势表达不满。
7. 当你的对手在台面击球时，不能划火柴或点燃打火机，别外要避免咳嗽出声。
8. 在某盘或某场比赛中，当你击失一杆或你的对手仍在台面击球时，不能认输。等到你的对手击完球后，方可认输。
9. 不能与裁判或你的对手进行争论。[[3]](#footnote-3)

就以上的这些礼仪，足以体现斯诺克游戏的高雅。

在斯诺克游戏球的制作上，历史上曾使用过昂贵的象牙。二战以前改用较为轻便的微晶球，而现在的球全部是用高级合成树脂制造。

1. 逻辑抽象

整个游戏其实就是对球的位置、速度、加速度等实际的物理量进行计算机模拟处理。为了较高真实度地还原整个物理过程而又不至于算法太复杂以至于作者有限的能力无法完成游戏代码的编写工作，提出以下模型及假设：

1. 由于斯诺克游戏球的制作材料弹性系数很高，在对球与球的碰撞处理上取弹性系数[[4]](#footnote-4)为1，即完全弹性碰撞。对球与桌边的碰撞采用相同的处理方式。
2. 球的位置是在以桌面为基面的二维空间上的点（以球心计），球的速度是该二维空间内的矢量。
3. 不考虑球的绕球心的旋转运动。
4. 不考虑球的滑动滚动，即对球滚动的处理都认为是无滑滚动[[5]](#footnote-5)。
5. 处理碰撞时，不考虑切向的摩擦作用力。
6. 目标

通过对 java面向对象特性的系统地学习，运用多媒体处理、图形处理等相关技术，完成一个集竞技性、娱乐性、可移植性强的游戏。

视觉上：

java本身自带很多功能强大的图像图形处理的类，学会熟练地使用它们，并将其有效、合理地运用到游戏的开发中去。如多使用 png 图片等来获得透明的物体。运用双缓冲[[6]](#footnote-6)（Double-buffering）技术，来减少、消除游戏过程中的画面闪烁问题，以获得良好的动态感。另外采用多线程 (Silberschatz, Galvin, & Gagne, 2007)（Thread: A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.）技术来定时刷新画面。

听觉上：

使用java来播放游戏过程中产生的诸如球与球相碰、球与桌相碰的声音及游戏开始和游戏整个过程的背景音乐等。

**第二部分 实现**

1. 算法设计

作为一个球类游戏，其中最难的就是对于球的碰撞处理，包括球与球之间的碰撞和球与球桌之间的碰撞。

# 球与球的碰撞

基于之前的假设，我们可以认为每个台球就是平面上的一个圆，设两球的圆心分别为，球的半径为R，当两球相碰撞时必然有：

无论多少个球同时相碰，我们都可以分解为两两球碰撞来处理，所以这里只需考虑两个球相碰拉情况，当有三个或更多球相碰时，认为是1号球先与2号球相碰，再是1号球与3号球相碰，等等。在实际运行中，发现这种处理方式真实场景的还原度还可以，且由于每次只处理两个球的碰撞，大大减少了编程的难度。如图 1所示，设两两碰撞时，球1的位置为P­1，速度为，球2的位置为P2，速度为。连接两球的球心，设在上的投影为，的投影为。先考虑在一条直线上两物体发生完全弹性碰撞时的情况，由能量守恒，可以得：

S1

S2

V1

V2

P1

P2

图 1

运量守恒：

整理得：

即在碰撞方向上，两质量相等的球完全弹性碰撞后会交换速度。而本程序中不考虑球之间的切向作用力，所以在碰撞方向垂直的方向上，球的速度不变。由此可以推出两球碰撞后的速度公式：

式中的，矢量可以通过，与的单位矢量的点乘得到：

1. **球与桌的碰撞**

总体来讲，球与桌的碰撞可以为分两类：

1. 球与桌边碰撞
2. 球与桌角碰撞

对于第一种情形，很好处理，只需将相应方向上的球速反向即可。对于第二种情形，均认为桌角是由半径等同台球半径的圆弧构成。

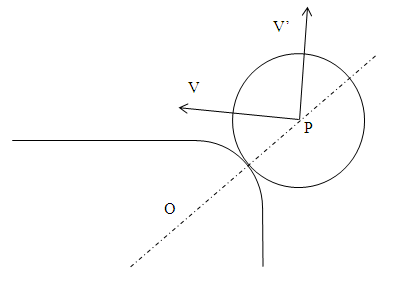


图 2

如图 2，设球心位置为P，球速为，桌角圆心位置为O，完全弹性碰撞后球速为，只需将原球速在垂直方向上作对称即可。

除了球的碰撞，另外一个比较难的问题是对玩家行为的判定，即游戏规则的实现。斯诺克游戏规则本身就相当复杂，游戏过程中会出现各种各样的情况。刚开局时，白球要首先碰到红球，如果没进，换人，如果进球了，作相应加分，然后将进了的彩球还原，并且下次必须首先碰到彩球。如果白球进了，对方加分，并换人……写个判断函数，人都搞晕。后来，一同学提示我可以用有限状态机[[7]](#footnote-7)来实现，于是就试了试，果然清晰些了。于是按大体类别，将玩家所处的状态分为四类：

1. **初始状态。即刚开始游戏或刚换到此玩家。**如果白球进了，换人，保持当前状态。如果先碰红球，且有进球，状态2 。其他，状态1。
2. **合法碰到一个红球状态。**如果白球进了，换人，状态1。如果先碰彩球，且有进球，状态3。
3. **合法碰到一个彩球状态。**如果白球进了，换人，状态1。如果先碰红球，且有进球，如果红球没了，状态4，其他，状态2。
4. **没有红球状态。**按照黄、绿、棕、蓝、粉、黑的顺序击球。
5. 类的设计

对于游戏可见的每个实体这里都要单独设计类，另外一些实际存在，对外有所表现但是没有实体的一些物理量也单独设计类，所以可以得到如下类的列表：

|  |  |
| --- | --- |
| **类名** | **方法和属性（部分）** |
| Speed | double vx 水平方向速度；double vy 竖直方向速度；  double v 总速度；  void add(Speed) 矢量加；void sub(Speed)矢量减；  void mul(double)标量乘；void div(double)标量除；  void speedUp(double)速度增加一个单位；  void speedDown(double)速度增加一个单位； |
| Ball | Speed speed球速；Point2D position球位置；int value球分值；  boolean ball\_in球是否进网；double E球的弹性碰撞系数；  double radius球的半径；  boolean isCollision(Ball)检查是否与球相碰；  void collision(Ball)执行碰撞，改变速度； |
| Stick | double angle 球杆角度 ；double force击球力度； Point mouse鼠标位置； double stick\_x,stick\_y 球杆所指的中心位置；  AffineTransform at球杆旋转的变换；boolean hidden隐藏球杆；  void setDirection()设置球杆方向； |
| Player | String name 玩家名；int Score玩家得分； |
| Table | int basex, basey // 桌子左上角的坐标  int width, height; // 桌子宽高  int off\_x, off\_y; // 桌子内边界宽度  int holeRadius = 14; // 球洞半径  Point2D[] c\_point; // 桌角中心位置  Point2D[] holes; // 球洞位置  Rectangle2D boundry; // 球桌边界  Area MainBallArea; // 球桌主球区  boolean isCollisition(Ball); // 球是否碰到桌  void collisition(Ball); // 执行碰撞，改变球带  boolean ballIn(Ball); // 判断球是否入网 |
| Powerbar | int force力量；int flag力量增加还是减少；boolean start开始蓄力；  double getPower() 返回力量大小；  void start() 开始蓄力；  void stop()停止蓄力； |
| Snooker（游戏主类） | Thread snooker; // 游戏动画线程  Image offscreen; // 双缓冲，减少闪烁  ArrayList<Ball> balls; // 台上的球的链表  boolean putWhiteBall; // 是否放置主球  Ball whiteball; // 主球  ArrayList<Ball> ballin = new ArrayList<Ball>();// 某一杆的进球个数  int redBalls; // 台面上的红球个数  int firstColli; // 主球第一次碰到的球  Stick stick; // 球杆  Player[] players; // 两个玩家  Table table; // 球桌  Powerbar powerbar; // 力量条  int turns; // 当前轮流次序  boolean gameOver; // 游戏结束  boolean moving; // 是否还有球移动  ArrayList<Ball> initBalls(); // 初始化球的位置  void check(); // 检查游戏规则，并作相应处理 |

1. 流程设计

游戏的整个流程图大致如下：

否

否

否

是

是

是

游戏初始化

启动动画线程

换人

加分

规则判断

击球

放置主球

犯规

结束

主球入网

游戏结束

换人

1. 问题描述

调BUG的过程是痛苦的，调试好BUG后的心情是愉悦的。整个开发过程中遇到过不少的BUG，而有些BUG硬是跟踪跟到死也没看出些明堂来。

首先一个，也是最严重的一个，同时也是调试后收获最大的一个，就是两球相碰后会出现粘在一起的情况。如下图：



碰撞判断核心代码如下：

for (int i = 0; i < balls.size(); i++) {

Ball ball1 = balls.get(i);

if (ball1.isIn())

continue;

for (int j = i + 1; j < balls.size(); j++) {

Ball ball2 = balls.get(j);

if (ball2.isIn())

continue;

if (ball1.isCollision(ball2)) {

ball1.collision2(ball2);

}

}

}

大致就是双重循环，将每一不重复对球都判断一次，如果相碰，就执行碰撞。按理说这样之后它们的速度不会再相对了，所以下次应该不会再碰。但是下次进来时，程序仍然认为它们是相碰的。原来相碰是按两球心距离小于球半径的两倍来判断的，而相碰前一帧画面中，球速度可能很大，两球心已经很近了（小于两倍半径），碰后球速可能变小，在一帧画面中，两球相互远离后，距离仍在两倍半径内。发现问题后，我将 isCollisition 函数中的判定条件改宽了，只要距离小于半径的两倍再加上一个常数就算它们撞了，但是没能实际解决问题，无论常数多大。一次偶然的情况下，突然想到，球是否相碰不仅与位置，应该还与速度方向有关。即如果两球距离小于两倍半径，且它们的速度在相互靠近的话，就算碰撞了，远离就不算。这样在原有的代码基础上加了几行用于判断速度方向的代码：

if (position.distance(ball.position) < 2 \* radius + 2) {

Speed t = new Speed(ball.position.getX() - this.position.getX(),

ball.position.getY() - this.position.getY());

double t1 = this.speed.dot(t); // 球1速度在球心连线上的投影

double t2 = ball.speed.dot(t); // 球2速度在球心连线上的投影

if (t1 - t2 > 0)

retn = true;

}

这次问题得到了解决！（略过其中的一个小手误）

用同样的思想，也解决了球与桌子碰撞的问题。之前，一旦球碰到桌子边就会来回不停地碰，产生的原因其实与球粘一起是一样的，加入速度考虑后，问题就没有了！

1. 可改善提高之处

限于本人的水平和时间的不足，本程序尚有很大的提高空间：

1. 游戏时没有添加任何按钮、菜单等，用户可操作性较差；
2. 游戏过程中提示信息过少，不明白玩家不知道怎么玩；
3. 击球时，没有瞄准线，玩家很难把握击球方向；
4. 游戏结束后，不能重新开始；
5. 没有练习模式；
6. 不能人机对战；
7. ……

1. 斯诺克\_百度百科。<http://baike.baidu.com/view/5025.htm> [↑](#footnote-ref-1)
2. 斯诺克发展历史。<http://www.bokee.net.to4.zju6.edu.cn/bloggermodule/blog_viewblog.do?id=535040> [↑](#footnote-ref-2)
3. soso 问问——斯诺克历史的英文介绍。<http://wenwen.soso.com/z/q160771941.htm> [↑](#footnote-ref-3)
4. 两个刚体碰撞前相对速度为，碰撞后的相对速度为，则弹性系数。满足。 [↑](#footnote-ref-4)
5. 即球绕球心旋转产生的球面线速度等于球心平移的速度。 [↑](#footnote-ref-5)
6. 是指利用CPU空闲时间将要显示的下一帧图像在后台绘制好，然后一次性输出到显示屏的技术。 [↑](#footnote-ref-6)
7. Finite State Machine。有限状态机(FSM)，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。 [↑](#footnote-ref-7)