东南大学国家示范性软件学院
College of Software Engineering
Southeast University

软件测试基础与实践

实验报告

实验名称：黑盒测试实验二

实验地点：计算机楼 268

实验日期：2019 年 11 月 20 日

学生姓名：姜子玥

学生学号：71117201

东南大学 软件学院 制

一、实验目的

（1） 能根据待测软件的特点，选择合适的方法对软件进行黑盒测试(功能测试)；

（2） 了解随机测试，巩固白盒测试和黑盒测试方法；

（3） 了解 JUnit 测试开发框架及其应用；

（4） 能对一些特定的程序进行蜕变测试。

二、实验环境

硬件环境：PC 机一台

软件环境：Java 编程环境　　C/C++编程环境待测程序：RectManager 和Sin.exe

实验指导书和待测程序可从课程主页下载：

https://github.com/npubird/softwaretesting

三、实验内容

（一）实验一：随机测试 VS 黑盒测试 VS 白盒测试

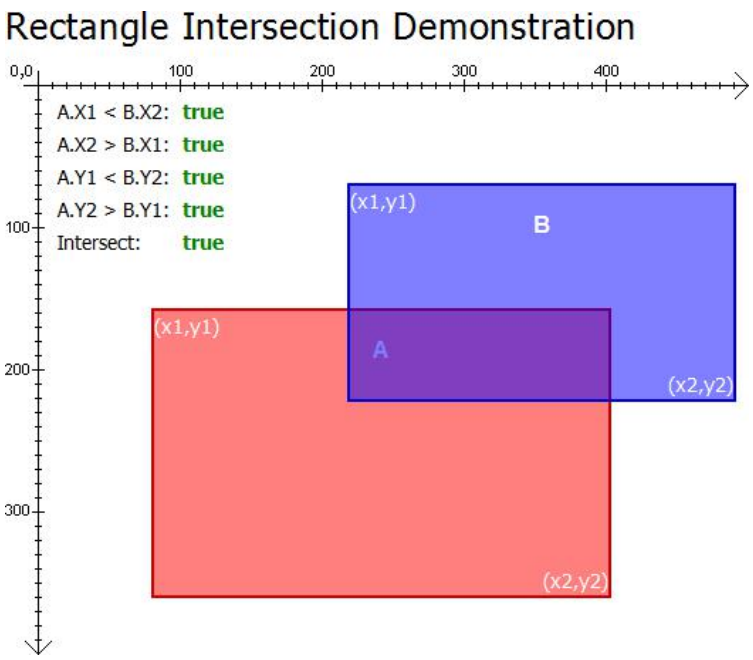
实验背景：在游戏引擎开发中，检测物体碰撞是一项重要的基础功能，比如 DOTA 和王者荣耀等游戏中的各种华丽大招的伤害波及范围计算等。

为简单起见，我们这里只考虑二维平面空间的情况，并用 RectManager 程序判断平面上任意两矩形的相交关系

（A:不相交，B:相交：B1:相交为一个区域，B12:包含，B13:完全重合，B2:交点为 1 个点，B3:交点为 1 条线段），如果相交，则同时给出相交部分的面积。

这里的二维平面限定为 iphone4 屏幕(640*960 分辨率)，且所有矩形的边都与坐标轴平行。计算机图形学中，通常用左上角和右下角的坐标来表示一个矩形。

任意两个矩形的关系可借用这个工具来辅助分析：http://silentmatt.com/rectangle-intersection/坐 标系请参照下图：



实验内容：

（1）请编写一简单程序，随机生成两个矩形的数据作为测试用例，请用这些测试用例对 RectManager 进行测试。

（2）请用黑盒测试方法设计相应的测试用例来测试程序；

（3）请分析 RectManger 的实现源代码，利用基本路径测试方法对程序进行测试

（4）在上述实验的基础上分析三种测试方法发现缺陷的能力上有和区别

实验过程：

（1）编写的程序源代码如下：（详见附件中程序）

```
import java.util.Random;

public class RectManager {
    float area;
    int nFlag;

    public RectManager() {
    }
```

```
    public static void get_sample_test(){
        //用以计数不同情况的数据
        int count_noxiangjiao = 0;
        int count_xiangjiao = 0;
        int count_baohan = 0;
        int count_chonghe = 0;
        int count_dot = 0;
        int count_line = 0;
        int count_error = 0;

        //500000 个测试用例
        int total = 500000;
        for(int i=0; i<total;i++){
            Rect A = new Rect();
            Rect B = new Rect();
            //随机产生数值
            A.left = new Random().nextInt(641);
            A.top = new Random().nextInt(961);
            A.right = new Random().nextInt(641);
            A.bottom = new Random().nextInt(961);
            //随机产生 B 的数据
            B.left = new Random().nextInt(641);
            B.top = new Random().nextInt(961);
            B.right = new Random().nextInt(641);
            B.bottom = new Random().nextInt(961);
            if (A.left >= 0 && A.right < 960 && A.top >= 0 && A.bottom < 640 &&
A.right >= A.left && A.bottom >= A.top) {
                if (B.left >= 0 && B.right < 960 && B.top >= 0 && B.bottom < 640 &&
B.right >= B.left && B.bottom >= B.top) {
                    RectManager test = new RectManager();
                    test.solve(A, B);
                    if (test.nFlag == 0) {
                        count_noxiangjiao++;
                    } else {
                        if (test.nFlag == 1) {
                            count_xiangjiao++;
                        } else if (test.nFlag == 2) {
                            count_baohan++;
                        } else if (test.nFlag == 3) {
                            count_chonghe++;
                        } else if (test.nFlag == 4) {
                            count_dot++;
                        } else if (test.nFlag == 5) {
```

```
                                count_line++;
                            }
                        }
                    } else {
                        count_error++;
                    }
                } else {
                    count_error++;
                }
            }

            //统计命中率
            System.out.println("矩形不相交概率："+(double)count_noxiangjiao/total);
            System.out.println("矩形相交于一个区域概率："+(double)count_xiangjiao/total);
            System.out.println("矩形相交于一个区域且为包含关系概率：
"+(double)count_baohan/total);
            System.out.println("矩形相交于一个区域且正好重合概率：
"+(double)count_chonghe/total);
            System.out.println("矩形相交于一个区域且交点为1个点概率：
"+(double)count_dot/total);
            System.out.println("矩形相交于一个区域且交点为1条线段概率：
"+(double)count_line/total);
            System.out.println("测试用例不合法的概率："+(double)count_error/total);
        }
        public static void main(String[] args) {
            get_sample_test();
        }

        private void solve(Rect A, Rect B) {
            int nMaxLeft;
            if (A.left >= B.left) {
                nMaxLeft = A.left;
            } else {
                nMaxLeft = B.left;
            }

            int nMaxTop;
            if (A.top >= B.top) {
                nMaxTop = A.top;
            } else {
                nMaxTop = B.top;
            }

            int nMinRight;
```

```
            if (A.right <= B.right) {
                nMinRight = A.right;
            } else {
                nMinRight = B.right;
            }

            int nMinBottom;
            if (A.bottom <= B.bottom) {
                nMinBottom = A.bottom;
            } else {
                nMinBottom = B.bottom;
            }

            if (nMaxLeft <= nMinRight && nMaxTop <= nMinBottom) {
                this.nFlag = 1;
                this.area = (float)((nMinRight - nMaxLeft + 1) * (nMinBottom - nMaxTop + 1));
                if (B.left == A.left && B.right == A.right && B.top == A.top && B.bottom ==
A.bottom) {
                    this.nFlag = 3;
                } else if (nMaxLeft == A.left && nMinRight == A.right && nMaxTop == A.top
&& nMinBottom == A.bottom || nMaxLeft == B.left && nMinRight == B.right && nMaxTop
== B.top && nMinBottom == B.bottom) {
                    this.nFlag = 2;
                } else if (nMaxLeft == nMinRight && nMaxTop == nMinBottom) {
                    this.nFlag = 4;
                } else if (nMaxLeft == nMinRight && nMaxTop < nMinBottom || nMaxLeft <
nMinRight && nMaxTop == nMinBottom) {
                    this.nFlag = 5;
                }
            } else {
                this.nFlag = 0;
            }
        }
}
```

测试运行结果截图如下：

（2）对该程序进行**黑盒测试**过程如下：

a) 等价类划分

序号	类型	等价类
1	无效等价类	输入含非数字类型数据
2	无效等价类	输入含非整型数据
3	无效等价类	输入含负数
4	无效等价类	输入含超出范围的数据
5	无效等价类	矩形 A 或 B 的 left 大于 right
6	无效等价类	矩形 A 或 B 的 top 大于 bottom
7	有效等价类	矩形不相交
8	有效等价类	矩形相交于一个区域
9	有效等价类	矩形相交于一个区域且为包含关系
10	有效等价类	矩形互相重合
11	有效等价类	矩形相交于一个点
12	有效等价类	矩形相交于一个区域且交点为 1 条线

设计测试用例如下：

序号	输入	预期输出	实际输出	覆盖等价类序号
1	A[a,1,15,1] B[1,2,5,c]	Input error	InputMismatchException	1
2	A[1.5,2,3,4] B[0.5,6,3,8]	Input error	InputMismatchException	2
3	A[-1,2,5,2] B[66,4,-9,5]	Input error	Input error in Rectangle A	3
4	A[655,2,1,1] B[1,1000,2,2]	Input error	Input error in Rectangle A	4
5	A[2,1,1,2] B[1,1,2,2]	Input error	Input error in Rectangle A	5
6	A[1,2,2,1] B[1,1,2,2]	Input error	Input error in Rectangle A	6
7	A[1,1,2,2] B[3,3,5,5]	矩形不相交	矩形不相交	7
8	A[1,1,3,3] B[2,2,5,5]	矩形相交于一个区域，相交面积：4.0	矩形相交于一个区域，相交面积：4.0	8
9	A[1,1,5,4] B[2,2,3,3]	矩形相交于一个区域且为包含关系，相交面积：4.0	矩形相交于一个区域且为包含关系，相交面积：4.0	9
10	A[1,1,2,2] B[1,1,2,2]	矩形相交于一个区域且为正好重合，相交面积：4.0	矩形相交于一个区域且为正好重合，相交面积：4.0	10
11	A[1,1,2,2] B[2,2,5,5]	矩形相交于一个区域且交点为 1 个点！ 相交面积：1.0	矩形相交于一个区域且交点为 1 个点！ 相交面积：1.0	11
11	A[1,1,2,2] B[2,1,3,3]	矩形相交于一个区域且交点为 1 条线段！ 相交面积：2.0	矩形相交于一个区域且交点为 1 条线段！ 相交面积：2.0	12

b)边界值分析

边界条件	参数	Min-1	Min	Min+1	Max-1	Max	Max+1
1	A.left	-1	0	1	958	959	960
2	A.top	-1	0	1	638	639	640
3	A.right	-1	0	1	958	959	960
4	A.bottom	-1	0	1	638	639	640
5	B.left	-1	0	1	958	959	960
6	B.top	-1	0	1	638	639	640
7	B.right	-1	0	1	958	959	960
8	B.bottom	-1	0	1	638	639	640

边界值分析测试用例如下：

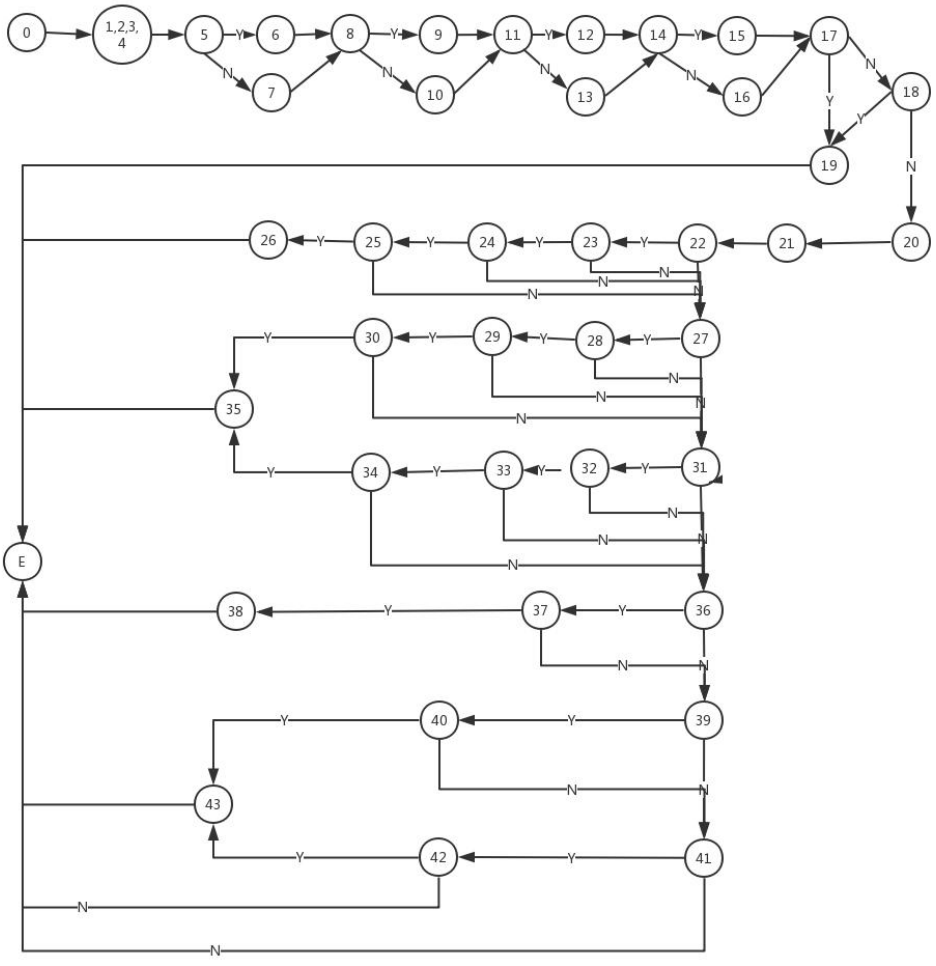
序号	输入	期望输出	实际输出	边界条件
1	A[-1,1,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	1
2	A[0,1,2,2] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：4.0	矩形相交于一个区域且为包含关系！ 相交面积：4.0	
3	A[1,1,2,2] B[1,1,2,2]	矩形相交于一个区域且正好重合！ 相交面积：4.0	矩形相交于一个区域且正好重合！ 相交面积：4.0	
4	A[958,1,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
5	A[959,1,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
6	A[960,1,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
7	A[1,-1,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
8	A[1,0,2,2] B[1,1,2,2]	矩形相交于一个区域且	矩形相交于一个区域且	

		为包含关系！ 相交面积：4.0	为包含关系！ 相交面积：4.0	2
9	A[1,1,2,2] B[1,1,2,2]	矩形相交于一个区域且正好重合！ 相交面积：4.0	矩形相交于一个区域且正好重合！ 相交面积：4.0	
10	A[1,638,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
11	A[1,639,2,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
12	A[1,640,2,2]B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
13	A[1,1,-1,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	3
14	A[1,1,0,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
15	A[1,1,1,2] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：2.0	矩形相交于一个区域且为包含关系！ 相交面积：2.0	
16	A[1,1,958,2] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：4.0	矩形相交于一个区域且为包含关系！ 相交面积：4.0	
17	A[1,1,959,2] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：4.0	矩形相交于一个区域且为包含关系！ 相交面积：4.0	
18	A[1,1,960,2] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
19	A[1,1,2,-1] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	4
20	A[1,1,2,0] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
21	A[1,1,2,1] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：2.0	矩形相交于一个区域且为包含关系！ 相交面积：2.0	
22	A[1,1,2,638] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：4.0	矩形相交于一个区域且为包含关系！ 相交面积：4.0	

23	A[1,1,2,639] B[1,1,2,2]	矩形相交于一个区域且为包含关系！ 相交面积：4.0	矩形相交于一个区域且为包含关系！ 相交面积：4.0	
24	A[1,1,2,640] B[1,1,2,2]	Input error in Rectangle A	Input error in Rectangle A	
.....对 B 的边界检测同 A				

（3）**白盒测试**

程序流图如下：



0：开始
1：int MaxLeft=0;
2:int nMaxTop=0;
3:int nMinRight=0;
4:int nMinBottom=0;
5: if(A.left>=B.left){
6:nMaxLeft=A.left;
7:nMaxLeft=B.left;
8:if(A.top>=B.top){
9:nMaxTop=A.top
10:nMaxTop=B.top
11:if(A.right<=B.right){
12:nMinRight=A.right
13:nMinRight=B.right
14:if(A.bottom<=B.bottom)
15:nMinBottom=A.bottom
16:nMinBottom=B.bottom
17:if((nMaxLeft>nMinRight
18:nMaxTop>nMinBottom
19:nFlag=0;
20:nFlag=1
21:area=(float)((nMinRight-nMaxLeft+1)*(nMinBottom-nMaxTop+1));
22:if(B.left==A.left
23:&&B.right==A.right
24:&&B.top==A.top
25:&&B.bottom==A.bottom)
26:nFlag=3;
27:else if(nMaxLeft==A.left
28:&&nMinRight==A.right
29:&&nMaxTop==A.top
30:&&nMinBottom==A.bottom
31:nMaxLeft==B.left
32:nMinRight==B.right
33:nMaxTop==B.top
34:nMinBottom==B.bottom
35:nFlag=2;
36:else if(nMaxLeft==nMinRight
37:nMaxTop==nMinBottom
38:nFlag=4;
39:nMaxLeft==nMinRight
40:nMaxTop<nMinBottom
41:nMaxLeft<nMinRight
42:nMaxTop==nMinBottom
43:nFlag=5;
E:出口

计算得程序流图中环复杂度为 25

设计**基本路径**如下：

编号	基本路径
1	0-1,2,3,4-5-6-8-9-11-12-14-15-17-19-E
2	0-1,2,3,4-5-7-8-9-11-12-14-15-17-19-E
3	0-1,2,3,4-5-6-8-10-11-12-14-15-17-19-E
4	0-1,2,3,4-5-6-8-9-11-13-14-15-17-19-E
5	0-1,2,3,4-5-6-8-9-11-12-14-16-17-19-E
6	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-19-E
7	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-24-25-26-E
8	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-29-30-35-E
9	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-27-28-29-30-35-E
10	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-24-27-28-29-30-35-E
11	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-24-25-27-28-29-30-35-E
12	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-32-33-34-35-E
13	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-31-32-33-34-35-E
14	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-29-31-32-33-34-35-E
15	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-29-30-31-32-33-34-35-E
16	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-37-38-E
17	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-32-36-37-38-E
18	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-32-33-36-37-38-E
19	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-32-33-34-36-37-38-E
20	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-43-E
21	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-37-39-40-43-E
22	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-41-E
23	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-41-E
24	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-41-42-E
25	0-1,2,3,4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-41-42-43-E

（4）经过上述三种不同的测试方式，我发现三种方法缺陷能力有以下差别：

随机测试：很难产生特殊情况的测试用例，如边界值情况。例如本次实验中我进行了多次随机测试但没有一次产生出两个区域正好重合的情况。

黑盒测试：使用等价类划分可以避免大量的重复测试用例，边界值分析可以考虑到特殊情况出现时的测试，有较强的发现缺陷能力，但唯一的缺陷就是需要耗费大量的设计测试用例时间

白盒测试：代码覆盖率很高，但边界值、非法输入的反面测试用例很难得到，同时根据基本路径设计测试用例耗费时间会远远多于黑盒测试设计测试用例。

总之，随机测试简单但很容易忽略边界值的特殊情况‘黑河测试可以很好的测试大多数可能性；白盒测试代码覆盖率高，但缺少反面测试、设计测试实例耗时多。

（二）实验二：蜕变测试问题

实验背景：在很多软件测试活动中，人们发现给出一个测试用例的期望输出是一件很困难的事情，在一些情况下，人们甚至无法给出测试用例的预期输出。这种测试预测输出无法获得的问题，就称为测试预言问题(Test Orcale Problem)。为解决 Test Orcale 问题，1998 年 T.Y. Chen 提出的蜕变测试的概念。

游戏引擎中需要高效的计算，所以其中的数学函数都需要重新进行快速实现，不能调用系统的自带数学函数。如下的程序片段是一个 sin(x)函数的实现代码示例：

```
//always wrap input angle to -PI..PI
if (x < -3.14159265)
    x += 6.28318531;
else
if (x >  3.14159265)
    x -= 6.28318531;

//compute sine
if (x < 0)
    sin = 1.27323954 * x + .405284735 * x * x;
else
    sin = 1.27323954 * x - 0.405284735 * x * x;
```

Sin.exe（该执行程序见课程主页）是一个用某种数值计算方法求解数学函数f(x)=sin(x)的程序。请设计测试用例，实现对该程序的黑盒测试。

实验要求：（1）给出每个测试用例设计的理由

（2）这个实验中展示的测试用例输出值无法预测的情形还可能出现在哪些实际应用中。

实验过程：

蜕变关系 MR 如下所示：

R1: sin(x) = -sin(-x)

R2: sin(x) = sin(180-x)

R3: sin(x) = sin(x+360i*k)， k 为整数

R4: $\sin^2(x)+\sin^2(90-x)=1$

(1)测试用例生成：

序号	原先测试用例	后续测试用例	预期结果	实际结果	设计理由
1	30		0.5	0.5	一般测试用例
2		-30	-0.5	-0.523599	sin(x) = -sin(-x)
3		150	0.5	0.50001	sin(x) = sin(Pi-x)
4		390	0.5	0.5	sin(x) = sin(x+2Pi*k)， k 为整数
5		60	0.8660	0.866025	$\sin^2(x)+\sin^2((Pi/2)-x)=1$

（2）如实验中展示的测试用例输出值无法预测的情形还可能出现在对桥梁应力测试、ex 函数测试、cos（x）函数测试等实际应用中。

三、实验思考

（1）通过测试，是否发现程序中存在的缺陷？

通过测试，能够发现存在有一些缺陷，例如实验一种输入不符合要求时，应该提醒哪一个输入不符合要求，而不是出错

（2）经过一系列的实验，请谈谈你所感受的软件测试中存在哪些挑战性的难题。

经过实验，发现软件测试中存在以下难题：

对于黑盒测试：需要充分考虑所有可能的等价类并进行合乎逻辑的划分，这需要较高的测试技巧；采用边界值测试时需要先确定参数。

对于白盒测试：测试的前提是通读理解代码并画出程序流图或流程图，这对测试人员的代码理解能力要求较高，同时耗费大量的测试时间在绘制图，并根据图来设计测试用例上。

（3）随机测试中，对很多同学的随机数生成范围来说，矩形不相交的统计概率不是 0.5，你的实验是否存在这种现象？

我的实验中存在上述现象。

（4）蜕变关系是否容易发现？蜕变测试的可适用场景有哪些？如果是 tan(x), log2(x)等函数的实现，其蜕变测试怎么做呢？需要寻找什么蜕变关系

- 单就实验二来说，蜕变关系还是较容易发现的，因为 sin（x）函数是我所比较熟悉的函数，很容易就找出了它的蜕变关系。
- 蜕变测试可适用的场景是程序的执行结果不能预知的现象，依据蜕变关系来生成新的测试用例，通过验证蜕变关系是否被保持来决定测试是否通过。
- 对于 tan(x)蜕变关系是：tan(x) = 1/tan(90-x)　tan(x) = tan(x+360*k),k 为整数

  tan(x) = tan(x+(2*k+1)*180)，k 为整数
- 对于 $\log_2(x)$的蜕变关系是 $\log_2(x*y) = \log_2(x) + \log_2(y)$　　$\log_2(x/y) = \log_2(x) - \log_2(y)$