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 机房

实验日期：2019 年 11 月 20 日

学生姓名：唐珞鑫

学生学号：71117106

东南大学软件学院制

一、实验目的

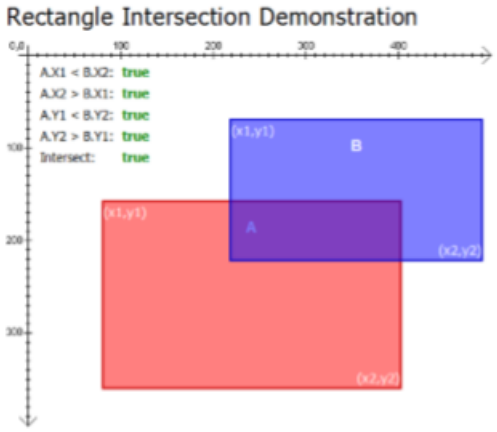
（1）能根据待测软件的特点，选择合适的方法对软件进行黑盒测试(功能测试)；
（2）了解随机测试，巩固白盒测试和黑盒测试方法；
（3）了解 JUnit 测试开发框架及其应用；
（4）能对一些特定的程序进行蜕变测试。

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

二、实验内容

实验 1：随机测试 VS 黑盒测试 VS 白盒测试

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：



(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。

要求：

a)编写程序，实现用随机函数生成大量测试用例（10 万-100 万个），对上述问题进行随机测试。

b)注意随机测试用例产生的范围应比屏幕范围稍微大一点。屏幕范围：x 取值范围[0-639]，y 取值范围[0-959];

c)在测试用例生成程序中，同时调用 RectManager 中的方法直接驱动测试自动执行。

d)对随机测试结果进行统计，分析随机测试用例对两矩形相交的各种关系的覆盖情况（统计上的命中概率）。

e)给出源代码和测试运行结果。熟悉 JUnit 的同学，可用 JUnit 实现上述随机测试。

(2)请用黑盒测试方法，设计相应的测试用例来测试程序（可参考并重用实验四中设计的测试用例）；提示：程序运行命令行：java-jarRectManager.jar

(3)请分析 RectManager 的实现源代码，利用基本路径测试方法对程序进行白盒测试；只要求针对 solve()方法进行测试（只给出基本路径，不用具体设计测试用例）。

(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何差别。

(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。

测试用例数量：100 万个

测试用例范围：根据题目要求， 测试用例产生的范围应该比屏幕范围稍微大一点，得出用例产生范围为：x[-3~642],y[-3~962]

源代码：

```
int m=0;
int n=0;
int k=0,a=0,b=0,c=0,d=0,e=0,f=0,g=0,h=0;
 for(int i=0;i<1000000;i++) {
  Random random=new Random();

  m=random.nextInt(646)-3;
  n=random.nextInt(646)-3;
  System.out.println(m);
  System.out.println(n);
  if(m<n) {
      A.left = m;
          A.right = n;
  }else {
      A.left = n;
          A.right = m;
  }
```

```
m=random.nextInt(966)-3;
n=random.nextInt(966)-3;
System.out.println(m);
System.out.println(n);
if(m<n) {
    A.top = m;
        A.bottom = n;
}else {
    A.top = n;
        A.bottom = m;
}

m=random.nextInt(646)-3;
n=random.nextInt(646)-3;
System.out.println(m);
System.out.println(n);
if(m<n) {
    B.left = m;
        B.right = n;
}else {
    B.left = n;
        B.right = m;
}

m=random.nextInt(966)-3;
n=random.nextInt(966)-3;
System.out.println(m);
System.out.println(n);
if(m<n) {
    B.top = m;
        B.bottom = n;
}else {
    B.top = n;
        B.bottom = m;
}
if((!(A.left>=0 && A.right<640)|| !(A.top>=0 && A.bottom<960)
              ||!(A.right>=A.left) || !(A.bottom>=A.top))&&(!(B.left>=0
&& B.right<640)|| !(B.top>=0 && B.bottom<960)
          ||!(B.right>=B.left) || !(B.bottom>=B.top))) {
  System.out.println("Input error in Rectangle A");
  System.out.println("Input error in Rectangle B");
  k++;
  }
 if (!(A.left>=0 && A.right<640)|| !(A.top>=0 && A.bottom<960)
```

```
        ||!(A.right>=A.left) || !(A.bottom>=A.top)){
        System.out.println("Input error in Rectangle A");

    }
else if (!(B.left>=0 && B.right<640)|| !(B.top>=0 && B.bottom<960)
        ||!(B.right>=B.left) || !(B.bottom>=B.top)){
        System.out.println("Input error in Rectangle B");
        b++;
    }

    if (!(B.left>=0 && B.right<640)|| !(B.top>=0 && B.bottom<960)
            ||!(B.right>=B.left) || !(B.bottom>=B.top)){
            System.out.println("Input error in Rectangle B");

        }
    else if (!(A.left>=0 && A.right<640)|| !(A.top>=0 && A.bottom<960)
                ||!(A.right>=A.left) || !(A.bottom>=A.top)){
                System.out.println("Input error in Rectangle A");
            a++;
            }

    RectManager test= new RectManager();
    test.solve(A, B);

    if (test.nFlag==0){
        System.out.println("矩形不相交！");
        c++;
    }else if (test.nFlag==1){
        System.out.println("矩形相交于一个区域！");
        d++;
    }else if (test.nFlag==2){
        System.out.println("矩形相交于一个区域且为包含关系！");
        e++;
    }else if (test.nFlag==3){
        System.out.println("矩形相交于一个区域且正好重合！");
        f++;
    }else if (test.nFlag==4){
        System.out.println("矩形相交于一个区域且交点为1个点！");
        g++;
    }else if (test.nFlag==5){
        System.out.println("矩形相交于一个区域且交点为1条线段！");
        h++;
    }
    System.out.println("相交面积："+test.area);
```

```
    }

    System.out.println(k+"A:"+a+"B:"+b+"矩形不相交"+c+"矩形相交于一个区域！"+d
+"矩形相交于一个区域且为包含关系！"+e+"矩形相交于一个区域且正好重合！"+f
+"矩形相交于一个区域且交点为1个点！"+g+"矩形相交于一个区域且交点为1条线段！"+h);
```

测试运行结果：

实验结果	出现次数	所占比例
Input error in Rectangle A	29578	2.9578%
Input error in Rectangle B	29681	2.9681%
Input error in Rectangle A,B	923	0.0923%
矩形不相交	555017	55.5017%
矩形相交于一个区域	385836	38.5836%
矩形相交于一个区域且为包含关系	55856	5.5856%
矩形相交于一个区域且正好重合	0	0%
矩形相交于一个区域且交点为1个点	6	0.0006%
矩形相交于一个区域且交点为1条线段	3160	0.3160%

(2)请用黑盒测试方法，设计相应的测试用例来测试程序（可参考并重用实验四中设计的测试用例）；提示：程序运行命令行：java-jarRectManager.jar

根据等价类划分，可划分为：

输入数据	有效等价类	无效等价类
两个四边形的坐标	1.输入的数据在坐标范围内且矩形 A 或 B 的 left 不大于 right 矩形 A 或 B 的 top 不大于 bottom	2.矩形 A 或 B 的 left 或者 right 在[0,639]之外 3. 矩形 A 或 B 的 top 或者 bottom 在[0,959]之外 4. 矩形 A 或 B 的 left 大于 right 5. 矩形 A 或 B 的 top 大于 bottom
两个四边形重合情况	6.矩形不相交 7.矩形相交于一个区域 8.矩形相交于一个区域且为包含关系 9.矩形相交于一个区域且正好重合 10. 矩形相交于一个区域且交点为 1 个点 8. 矩形相交于一个区域且交点为 1 条线段	

根据边界值分析：

1.1固定其他数据的left边界条件

边界条件	Left	Right	Top	Bottom
1	0	[0,639]	[0,959]	[0,959]
2	639	[0,639]	[0,959]	[0,959]

1.2固定其他数据的right边界条件

边界条件	Left	Right	Top	Bottom
3	[0,639]	0	[0,959]	[0,959]
4	[0,639]	639	[0,959]	[0,959]

1.3固定其他数据的top边界条件

边界条件	Left	Right	Top	Bottom
5	[0,639]	[0,639]	0	[0,959]
6	[0,639]	[0,639]	959	[0,959]

1.4固定其他数据的bottom边界条件

边界条件	Left	Right	Top	Bottom
7	[0,639]	[0,639]	[0,959]	0
8	[0,639]	[0,639]	[0,959]	959

根据等价类划分和边界条件分析，设计测试用例：

序号	测试用例		覆盖条件		期望输出	实际输出
	A	B	边界条件	等价类		
01	-1,1,2,2	1,1,2,2	1	2	Input error in Rectangle A	Input error in Rectangle A
02	0,1,2,2	1,1,2,2		1,8	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
03	1,1,2,2	1,1,2,2		1,9	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
04	638,1,2,2	1,1,2,2	2	4	Input error in Rectangle A	Input error in Rectangle A
05	639,1,2,2	1,1,2,2		4	Input error in Rectangle A	Input error in Rectangle A
06	640,1,2,2	1,1,2,2		2,4	Input error in Rectangle A	Input error in Rectangle A
07	1,1,-1,2	1,1,2,2	3	2,4	Input error in Rectangle A	Input error in Rectangle A
08	1,1,0,2	1,1,2,2		4	Input error in Rectangle A	Input error in Rectangle A

09	1,1,1,2	1,1,2,2		1,8	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
10	1,1,638,2	1,1,2,2	4	1,8	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
11	1,1,639,2	1,1,2,2		1,8	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
12	1,1,640,2	1,1,2,2		2	Input error in Rectangle A	Input error in Rectangle A
13	1,-1,2,2	2,2,3,3	5	3	Input error in Rectangle A	Input error in Rectangle A
14	1,0,2,2	2,2,3,3		1,10	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
15	1,1,2,2	2,2,3,3		1,10	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
16	1,958,2,2	2,2,3,3	6	5	Input error in Rectangle A	Input error in Rectangle A
17	1,959,2,2	2,2,3,3		5	Input error in Rectangle A	Input error in Rectangle A
18	1,960,2,2	2,2,3,3		3,5	Input error in Rectangle A	Input error in Rectangle A
19	1,1,2,-1	2,2,3,3	7	3,5	Input error in Rectangle A	Input error in Rectangle A
20	1,1,2,0	2,2,3,3		5	Input error in Rectangle A	Input error in Rectangle A
21	1,1,2,1	2,2,3,3		1,6	矩形不相交!	矩形不相交!
22	1,1,2,958	2,2,3,3	8	1,8	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
23	1,1,2,959	2,2,3,3		1,8	矩形相交于一个区域且交点为1条线	矩形相交于一个区域且交点为1条线段!

					段！ 相交面积：2.0	相交面积：2.0
24	1,1,2,960	2,2,3,3		3	Input error in Rectangle A	Input error in Rectangle A
25	1,1,3,3	2,2,4,4	-	7	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

(3)请分析 RectManager 的实现源代码，利用基本路径测试方法对程序进行白盒测试；只要求针对solve()方法进行测试（只给出基本路径，不用具体设计测试用例）。

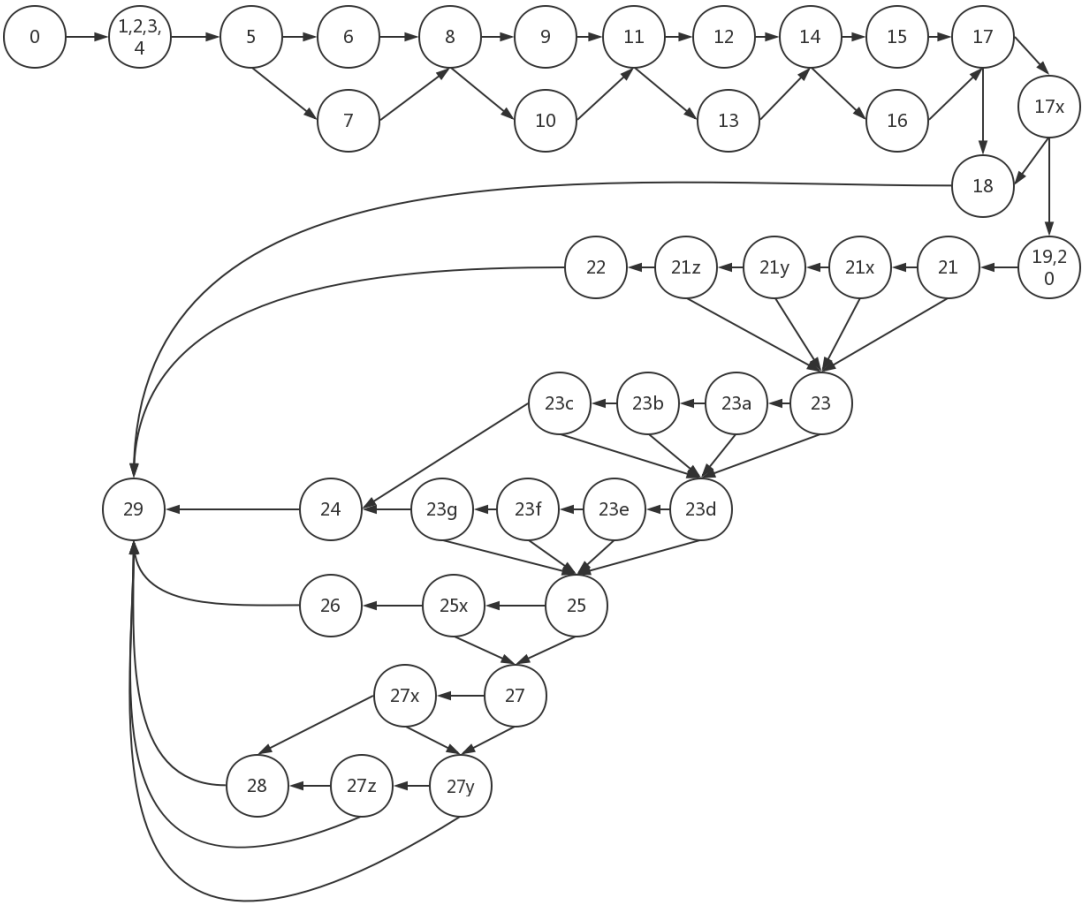
对程序进行标号：

```
    private void solve(Rect A, Rect B){
1               int nMaxLeft = 0;
2               int nMaxTop = 0;
3               int nMinRight = 0;
4               int nMinBottom = 0;

                // Get the max left.
5               if (A.left >= B.left)
                {
6                   nMaxLeft = A.left;
                }
                else
                {
7                   nMaxLeft = B.left;
                }

                // Get the max top.
8               if (A.top >= B.top)
                {
9                   nMaxTop = A.top;
                }
                else
                {
10                  nMaxTop = B.top;
                }

                // Get the min right.
11              if (A.right <= B.right)
                {
12                  nMinRight = A.right;
                }
```

```
            else
            {
13              nMinRight = B.right;
            }

            // Get the min bottom.
14          if (A.bottom <= B.bottom)
            {
15              nMinBottom = A.bottom;
            }
            else
            {
16              nMinBottom = B.bottom;
            }

        // Judge whether intersects.
17          if ((nMaxLeft > nMinRight) || (nMaxTop > nMinBottom))
            {//不相交
18           nFlag=0;

            }
            else
            {//相交
             //B1:相交为一个区域
19           nFlag = 1;
20           area = (nMinRight - nMaxLeft   + 1 ) * (nMinBottom - nMaxTop + 1);

21           if ((B.left==A.left) && (B.right==A.right) && (B.top==A.top) &&
(B.bottom==A.bottom)){
                 //B13:完全重合
22               nFlag = 3;
             }
23           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))){
                 //B12:包含
24               nFlag = 2;
             }
25           else if ((nMaxLeft==nMinRight) && (nMaxTop == nMinBottom)){
                 //B2:交点为1个点
26               nFlag = 4;
             }
```

```
27              else if (((nMaxLeft==nMinRight) && (nMaxTop < nMinBottom))
                      || ((nMaxLeft<nMinRight) && (nMaxTop == nMinBottom))){
                    //B3:交点为 1 条线段
28                  nFlag = 5;
                }
               }

            }
```

画出程序流图



V(G)=25

基本路径集合：

P1:0-1-2-3-4-5-6-8-9-11-12-14-15-18-29

P2:0-1-2-3-4-5-6-7-8-9-11-12-14-15-17-18-29

P3:0-1-2-3-4-5-7-8-10-11-12-14-15-17-18-29
P4:0-1-2-3-4-5-7-8-10-11-13-14-15-17-18-29
P5:0-1-2-3-4-5-7-8-10-11-13-14-16-17-18-29
P6:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-18-29
P7:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-21x-21y-21z-22-29
P8:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23a-23b-23c-24-29
P9:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-21x-23-23a-23b-23c-24-29
P10:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-21x-21y-23-23a-23b-23c-24-29
P11:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-21x-21y-21z-23-23a-23b-23c-24-29
P12:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-23f-23q-24-29
P13:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23a-23d-23e-23f-23q-24-29
P14:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23a-23b-23d-23e-23f-23q-24-29
P15:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23a-23b-23c-23d-23e-23f-23q-24-29
P16:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-25-25x-26-29
P17:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-25-25x-26-29
P18:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-23f-25-25x-26-29
P19:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-23f-23q-25-25x-26-29
P20:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-25-27-27x-28-29
P21:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-25-27-27x-28-29
P22:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-23f-25-27-27x-28-29
P23:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-23f-23g-25-27-27x-28-29
P23:0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-23e-23f-23g-25-27-27x-28-29
P24: 0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-25-27-27y-27z-28-29
P25: 0-1-2-3-4-5-7-8-10-11-13-14-16-17-17x-19-20-21-23-23d-25-27-27x-27y-27z-28-29

(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何差别。

随机性测试在产生了大量测试用例的情况下，仍然有特殊情况并没有覆盖到，很多测试是重复和冗余的。不过随机测试可以不需要人为设计测试用例，节省了人力，但是测试效果没办法得到保证。

黑盒测试在实现了较为全面的测试的同时，时间需求并不是很高，并且人为设计的测试用例更加有针对性，边界值的分析针对了最容易出现问题的地方，更容易测试出系统中的缺陷。

白盒测试相比于黑盒测试关注了内部代码，对于路径的覆盖让测试更为全面准确，代码的覆盖率高，不过也需要耗费更多的时间。

在实际运用中需要权衡考虑测试效果和时间成本，也许多种测试方法相结合能达到更好的效果。

实验 2：蜕变测试问题

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。

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。

MR：

sin（x）= - sin（-x）

sin（x）= sin（x + k*PI）

sin（x）= sin（PI - x）

测试用例生成：

序号	原先测试用例	后续测试用例	期望结果	实际结果	
01	30		0.5	0.5	理由
02		-30	-0.5	-0.523599	sin（x）= - sin（-x）
03		390	0.5	0.5	sin（x）= sin（x + 2*PI）
04		150	0.5	0.500001	sin（x）= sin（PI - x）
05		-150	0.5	-2.61799	sin（x）= sin（x -1*PI）
06		-330	0.5	-5.75959	sin（x）= sin（x -2*PI）

三、实验思考

(1)实验一中对于非法测试没有保护，属于程序存在一些缺陷。实验二对于负数会产生错误。
(2)软件测试需要对于各种方法有清晰的认识和了解，不能对于方法产生混淆，同时对于方法使用的熟练度有着要求，面对庞杂的测试工作，必须要熟练才能不忙乱。软件测试同样考验着耐心，必须要沉住气面对工作，冷静地去全方位的考虑问题。
(3)实验数据统计概率为 0.55，基本在 0.5 水平上，偏离 0.5 可能的原因也许是测试用例产生数量不够大，或是随机数生成范围设置过大，导致过多的数据落在“输入错误“ ”的区间里。
(4)蜕变关系容易发现，可应用于各种为办法预测输出结果的场景，如果是 tan(x)等函数，蜕变用例可根据相关的公式推算出来。