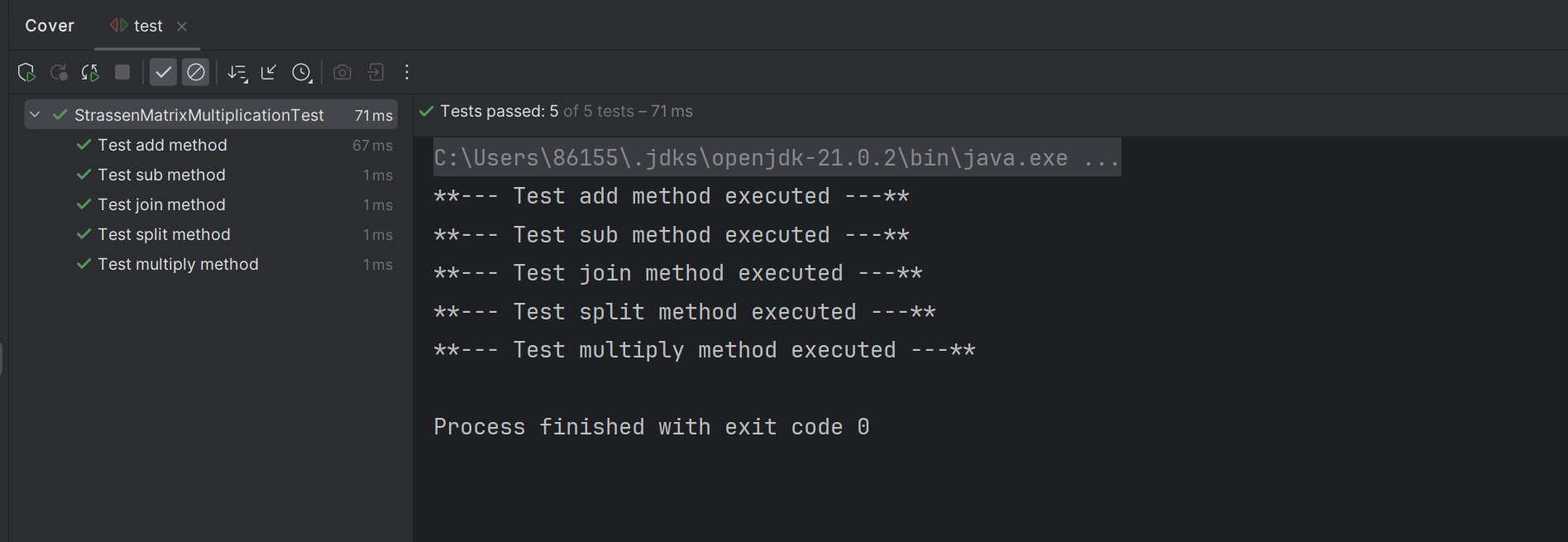
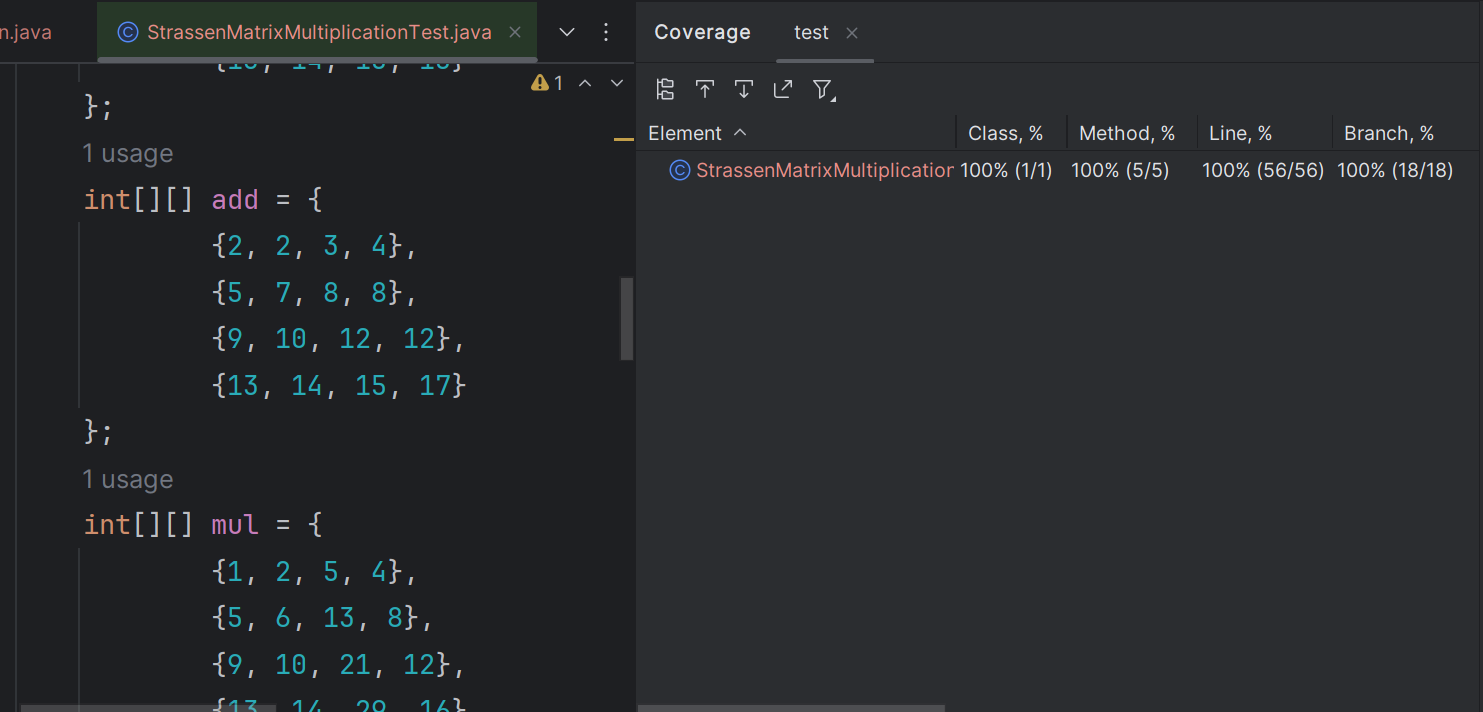
实验三报告

1. StrassenMatrixMultiplication类测试结果

*单元测试结果截图及代码覆盖率结果截图，单元测试代码无需提交。*

**

代码覆盖率

**

1. 英雄机类测试结果

*采用黑盒测试和白盒测试的常用方法，为飞机大战系统中的英雄机类设计单元测试用例。要求至少选择3个方法（包含其父类方法）作为单元测试的对象，编写测试用例并完成单元测试。截图JUnit单元测试的结果，单元测试代码随项目一起提交。*

2.1测试用例*（一个方法一个测试用例）*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | 01 | | | |
| 待测试类及方法 | HeroAircraft  .decreaseHp() | | | |
| 测试类及方法 | HeroAircraftTest.decreaseHp() | | | |
| 前提条件（如有） | 已经创建一个英雄机实例 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际输出 | 测试结果 |
| 对于不同的血量变化值，通过比较调用decreaseHp前后英雄机血量变化是否符合预期。 | 记录测试前血量，并对测试列表中血量一一输入，比较调用函数前后血量变化，即调用assertEquals(BeforeBlood-blood,AfterBlood);语句不报错，即调用前血量-血量=调用后血量，此时测试通过 | {2,5,100,-40,-1000}  三个正数，两个负数，其中-1000用于测试血量是否上溢，测试成功时，输出全部为true | 全部为true | 通过 1713597389252 |

    private int BeforeBlood;

    private int AfterBlood;

*//@Test*

    @ParameterizedTest

    @ValueSource(ints = {2,5,100,-40,-1000})

    void decreaseHp(int blood) {

        BeforeBlood=heroAircraft.getHp();

        int maxblood=heroAircraft.maxHp;

        heroAircraft.decreaseHp(blood);

        AfterBlood=heroAircraft.getHp();

        assertEquals(Math.min(BeforeBlood-blood,maxblood),AfterBlood);

    }

*用例编号：唯一标识测试用例的序号，一般是数字或模块名首字母大写+数字序号。*

*待测试类及方法：该用例所测试的类名和方法名*

*测试类及方法：相应的测试代码的类名和方法名*

*前提条件（如有）：执行该测试用例的前提条件，比如碰撞检测，需已创建英雄机和敌机（或道具）。*

*用例描述：用一句话简单总结该测试用例的用意和目的。*

*测试步骤：详细完整的操作过程描述。*

*期望结果：正常情况下的响应结果。*

*实际结果：程序通过测试步骤后实际的响应结果。*

*测试结果：通过或失败*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | 02 | | | |
| 待测试类及方法 | AbstractFlyingObject  .crash() | | | |
| 测试类及方法 | HeroAircraftTest.crash() | | | |
| 前提条件（如有） | 已经创建一个英雄机实例 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际输出 | 测试结果 |
| 测试边界情况以及正常情况下，位置重叠时是否撞击。  创建了一个子弹对象、一个mob敌机对象，以及一个英雄机对象，对其进行两两检测，其位置设定取决于碰撞框，对于每种组合，分别检测了能够碰撞和不能碰撞的两种边界情况。 | 在这里补充了一般飞行物的情况（子弹撞敌机）  使用一个因子对子弹对象的边界框高度进行修正，接着通过两物体的碰撞框高和宽计算两物体应保持的xy方向距离，接着对两种边界条件进行检测：碰撞：(x+distancex-1,y+distancey-1)；不碰撞：  (x+distancex-1,y+distancey)  并对A碰B和B碰A两种情况均进行了测试 | 前两次碰撞检测全部为ture，后两次检测全部为false | 前两次碰撞检测全部为ture，后两次检测全部为false | 通过 |

    private static Stream <List<AbstractFlyingObject > >objectProvider() {

        EnemyFactory factory1=new MobEnemyFactory();

        AbstractFlyingObject flyingObject1=factory1.createEnemy();

        BulletpropFactory factory2=new BulletpropFactory();

        AbstractFlyingObject flyingObject2=factory2.createprop();

        List<AbstractFlyingObject > enemyList1 = new ArrayList<>();

        enemyList1.add(flyingObject1);

        enemyList1.add(flyingObject2);

        List<AbstractFlyingObject > enemyList2 = new ArrayList<>();

        enemyList2.add(flyingObject1);

        enemyList2.add(heroAircraft);

        List<AbstractFlyingObject > enemyList3 = new ArrayList<>();

        enemyList3.add(flyingObject1);

        enemyList3.add(heroAircraft);

        return Stream.of(

                enemyList1, enemyList2, enemyList3

*//                new AbstractFlyingObject[]  { flyingObject1, heroAircraft},//英雄机和敌机*

*//                new AbstractFlyingObject[]  { flyingObject2, heroAircraft}//英雄机和子弹*

        );

    }

    @ParameterizedTest

    @MethodSource("objectProvider")

    void crash(List<AbstractFlyingObject> abstractFlyingObjects ){

        AbstractFlyingObject flyingObject1=abstractFlyingObjects.get(0);

        AbstractFlyingObject flyingObject2=abstractFlyingObjects.get(1);

*// 创建被测飞行物对象*

*//对方坐标、宽度、高度*

        int factor =  2 ; *//我方*

        int fFactor = flyingObject2 instanceof AbstractAircraft ? 2 : 1;*//对方*

        int x = flyingObject2.getLocationX();

        int y = flyingObject2.getLocationY();

        int fWidth = flyingObject2.getWidth();

        int fHeight = flyingObject2.getHeight();

        double distancex=(double)(fWidth+flyingObject1.getWidth())/2.0;

        double distancey=(double)(fHeight/fFactor+flyingObject1.getHeight()/factor)/2.0;

        flyingObject1.setLocation(x+distancex-1,y+distancey-1);

        System.out.printf("%f, %f%n",distancex,distancey);

        System.out.printf("%d, %d%n",x,y);

        System.out.printf("%d, %d%n", flyingObject1.getLocationX(),flyingObject1.getLocationY());

        assertTrue(flyingObject1.crash(flyingObject2));

        assertTrue(flyingObject2.crash(flyingObject1));

        flyingObject1.setLocation(x+distancex-1,y+distancey);

        System.out.printf("%f, %f%n",distancex,distancey);

        System.out.printf("%d, %d%n",x,y);

        System.out.printf("%d, %d%n", flyingObject1.getLocationX(), flyingObject1.getLocationY());

        assertFalse(flyingObject1.crash(flyingObject2));

        assertFalse(flyingObject2.crash(flyingObject1));

    }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | 03 | | | |
| 待测试类及方法 | HeroAirCraft.getinstance() | | | |
| 测试类及方法 | HeroAirCraftTest.getinstance() | | | |
| 前提条件（如有） | 无 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际输出 | 测试结果 |
| 判断英雄机是否实现多线程下的单例类。  多个线程同时调用getinstance方法创建英雄机对象，并将该对象存入哈希集合中，判断最终集合是否仅含有一个元素。 | 首先创建一个同步锁哈希表heroset，再声明一个thread类作为线程，并在其run方法里实现向heroset中添加一个新创建的英雄机对象。  使用一个最大容纳量为10的线程池，并运行20个线程，判断最终集合中元素数量是否仍旧为1. | Heroset.size()==1，即assert结果为True | assert结果为True | 通过 |

2.2 单元测试结果

全部通过！

