实验三报告

# 单元测试

1. 测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | AircraftWar1 | | | |
| 待测试类及方法 | HeroAircraft.decreaseHp(int) | | | |
| 测试类及方法 | HeroAircraftTest.decreaseHp(int) | | | |
| 前提条件（如有） | 通过HeroAircraftTest类的beforeAll()方法获取英雄机单例，保存在静态成员变量中 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际结果 | 测试结果 |
| 测试英雄机的生命值减少方法 | 用正常值、负值、超范围值当作输入，测试方法是否正确的修改了英雄机生命值 | 不出现生命值为负的情况；当传入负值时不改变英雄机生命值 | 所有参数都能让断言评估为正确，即符合期望结果 | 通过 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | AircraftWar2 | | | |
| 待测试类及方法 | HeroAircraft.crash(int) | | | |
| 测试类及方法 | HeroAircraftTest.crash(int) | | | |
| 前提条件（如有） | 通过HeroAircraftTest类的beforeAll()方法获取英雄机单例并创建敌机工厂，保存在静态成员变量中；通过HeroAircraftTest类的setUp()方法设置英雄机的位置、通过工厂创建敌机和子弹并设置其宽高 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际结果 | 测试结果 |
| 测试英雄机的碰撞方法 | 用与英雄机的位置重合或距离较近的敌机和子弹，检测crash方法是否返回true | 英雄碰撞敌机和子弹后，crash方法返回true | 所有参数都能让假设成立，即符合期望结果 | 通过 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | AircraftWar3 | | | |
| 待测试类及方法 | EliteEnemy.forward() | | | |
| 测试类及方法 | EliteEnemyTest.forward() | | | |
| 前提条件（如有） | 通过EliteEnemyTest类的beforeAll()方法创建敌机工厂，保存在静态成员变量中；通过EliteEnemyTest类的setUp()方法通过工厂创建敌机 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际结果 | 测试结果 |
| 测试精英敌机的移动方法 | 多次调用移动方法，检测精英敌机是否是向屏幕下方移动的、精英敌机的横向移动是否会超出屏幕 | 多次调用移动方法后，精英敌机的Y坐标变大，X坐标不会超出屏幕范围 | 多次调用移动方法后，精英敌机的Y坐标变大，X坐标不会超出屏幕范围，符合期望 | 通过 |

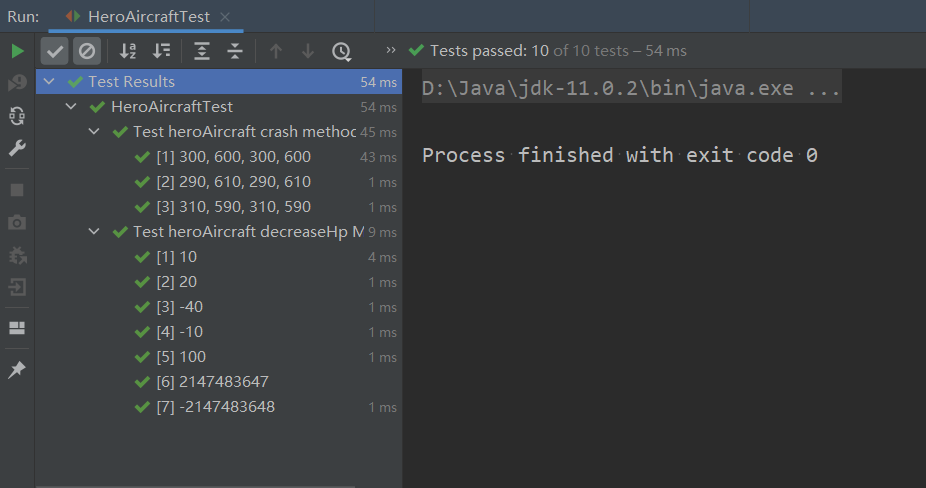
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | AircraftWar4 | | | |
| 待测试类及方法 | EliteEnemy.shoot() | | | |
| 测试类及方法 | EliteEnemyTest.shoot() | | | |
| 前提条件（如有） | 通过EliteEnemyTest类的beforeAll()方法创建敌机工厂，保存在静态成员变量中；通过EliteEnemyTest类的setUp()方法通过工厂创建敌机 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际结果 | 测试结果 |
| 测试精英敌机的射击方法 | 调用射击方法，检测精英敌机是否可以射出子弹以及子弹的方向是否是向屏幕下方移动 | 调用射击方法产生的子弹集合中有至少一个子弹，且子弹的移动方向是y轴正方式 | 所有断言评估为正确，即符合期望结果 | 通过 |

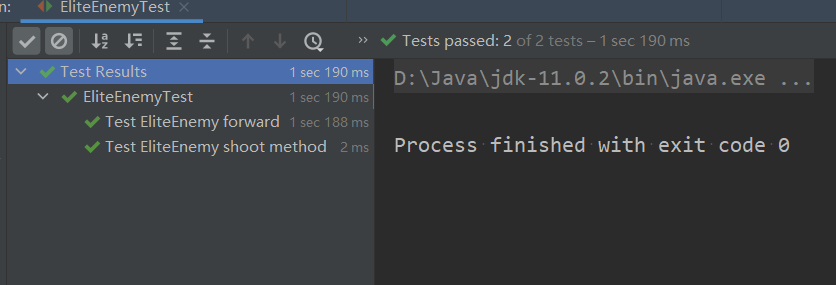
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | AircraftWar5 | | | |
| 待测试类及方法 | BloodProp.forwar() | | | |
| 测试类及方法 | BloodPropTest.forwar() | | | |
| 前提条件（如有） | 通过BloodPropTest类的beforeAll()方法创建道具工厂，保存在静态成员变量中；通过BloodPropTest类的setUp()方法通过工厂创建道具 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际结果 | 测试结果 |
| 测试道具的移动方法 | 调用移动方法，检测道具是否是向屏幕下方垂直移动 | 调用移动方法后，Y坐标变大，X坐标不变 | 所有断言评估为正确，即符合期望结果 | 通过 |

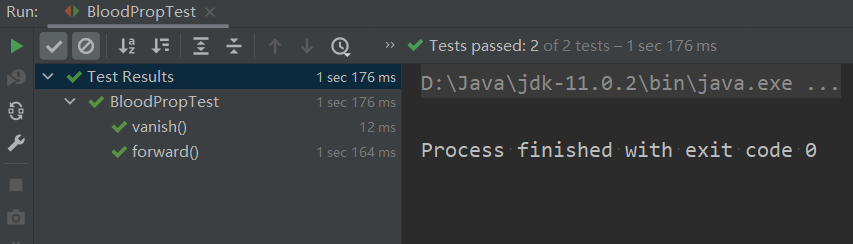
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | AircraftWar6 | | | |
| 待测试类及方法 | BloodProp.forwar() | | | |
| 测试类及方法 | BloodPropTest.forwar() | | | |
| 前提条件（如有） | 通过BloodPropTest类的beforeAll()方法创建道具工厂，保存在静态成员变量中；通过BloodPropTest类的setUp()方法通过工厂创建道具 | | | |
| 用例描述 | 测试步骤 | 期望结果 | 实际结果 | 测试结果 |
| 测试道具的消失方法 | 调用消失方法，检测道具是否被判定为失效 | 调用消失方法后，道具标识是否消失的字段在逻辑上表明道具已经失效 | 所有断言评估为正确，即符合期望结果 | 通过 |

1. JUnit单元测试结果

*请把JUnit每个测试类（包含多个方法）的运行结果截图。*





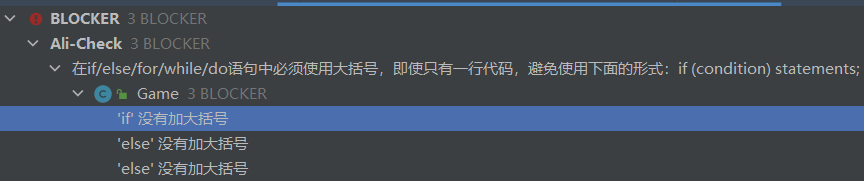


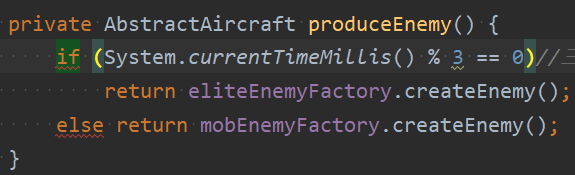
# 编码规约

*列举2个实验中使用到的阿里编码规约插件的例子，截图或文字描述插件扫描出的代码问题以及你是如何解决的。*

1. 程序结构中单条语句的大括号问题

*问题举例：*





*问题分析：*

不加大括号的书写容易因缩进等问题造成可读性降低

如:

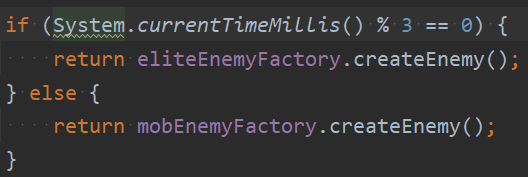
if (condition)

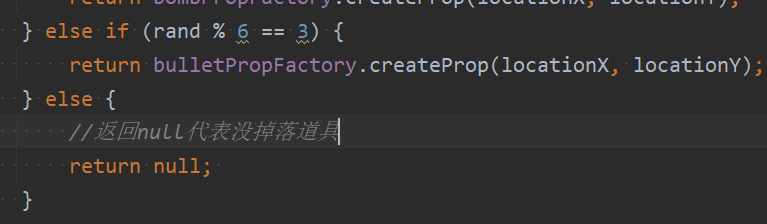
statement1;

statement2;

这样写看起来好像当condition成立时statement1和 statement2才执行，但实际上，statement2一定会执行

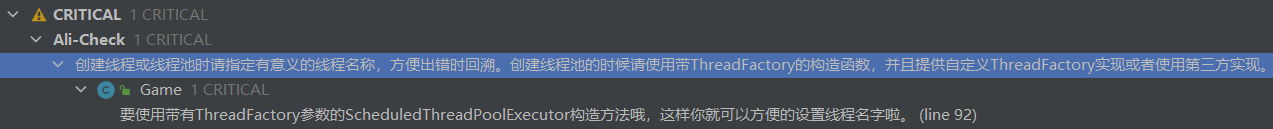
*问题解决：*给单条语句加上大括号

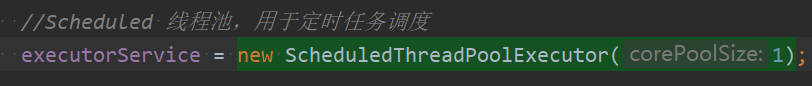




1. 线程的命名问题

*问题举例：*

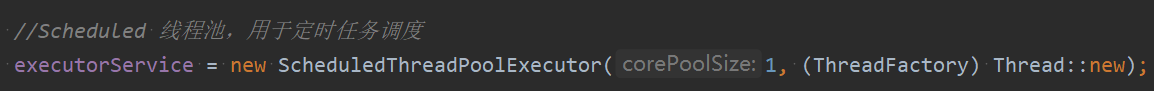




*问题分析：*

如果用户创建的线程使用默认名字，可读性就会大大降低，无法通过线程的名字判断线程相关执行单元的工作，增大了调度的难度，让程序难以维护

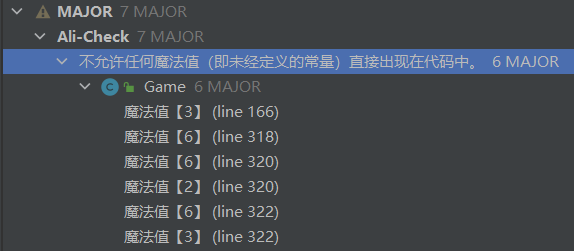
*问题解决：*使用ThreadFactory创建线程池中的线程

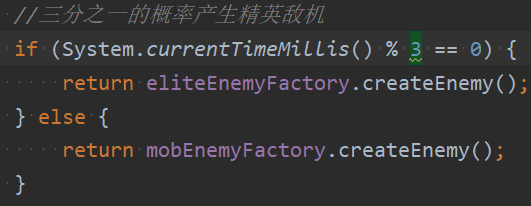


这里使用构造器引用，让线程池通过工厂、工厂通过Thread类的构造方法，创建线程（强转是为了匿名内部类对应的接口以及Thread类的构造器）

1. 未经定义的常量问题

*问题举例：*





*问题分析：*字面值直接出现在程序中，可能会降低程序可读性，影响程序执行逻辑的展现（甚至有时0和null等字面值也是如此）

*问题解决：*在合适的位置定义要使用的常量，并使用

