Jumper 测试报告

version 1.0.0

修订历史

日期	版本	作者	描述
2015.8.21	1.0.0	郑越升、龙嘉伟、黄勇进、林倩文	

目录

1 测试计划

使用一个用例测试 Jumper 与其他 actor 的交互功能；使用另外一个用例测试 Jumper 对于边界问题的处理。

第一部分包括五项小测试,分别是 Jumper 对于石头、花朵、虫子、演员以及 Jumper 的交互情况。

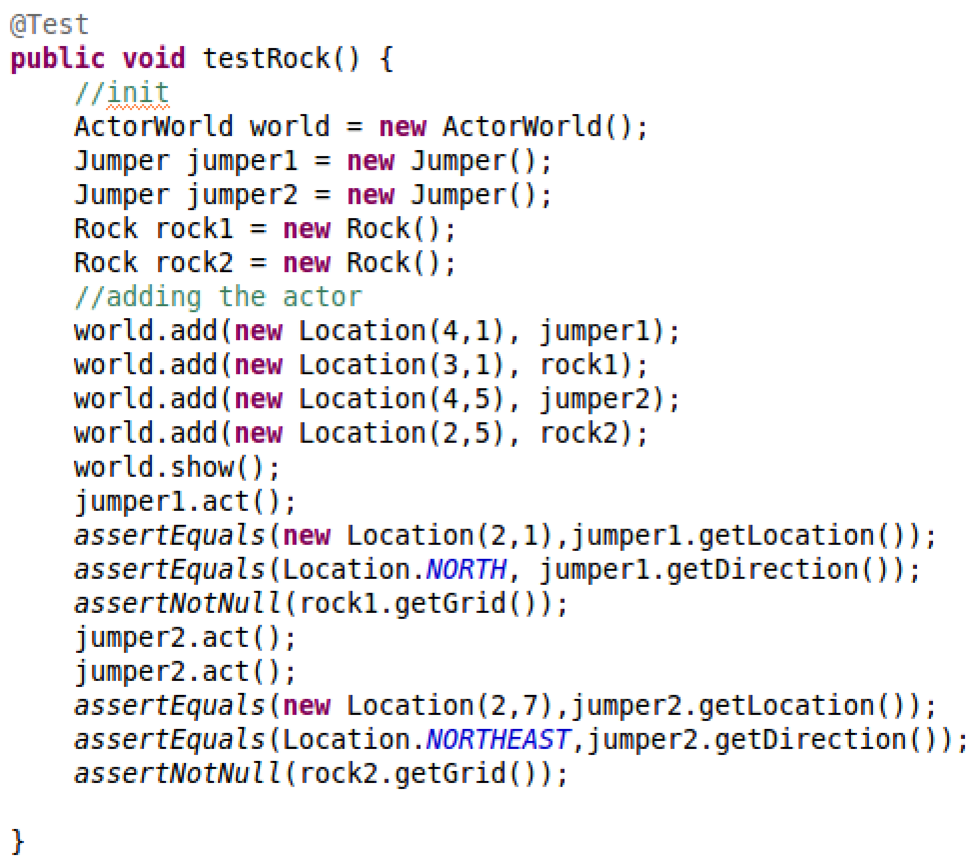
第二部分包括两项小测试，分别是 Jumper 面对边界以及距离边界距离为一个单位时的情况。

2 用例 1

测试目的：测试 Jumper 能否按照预期地与其他 actor 进行交互。

2.1 测试用例

用例 1：Jumper 与 Rock

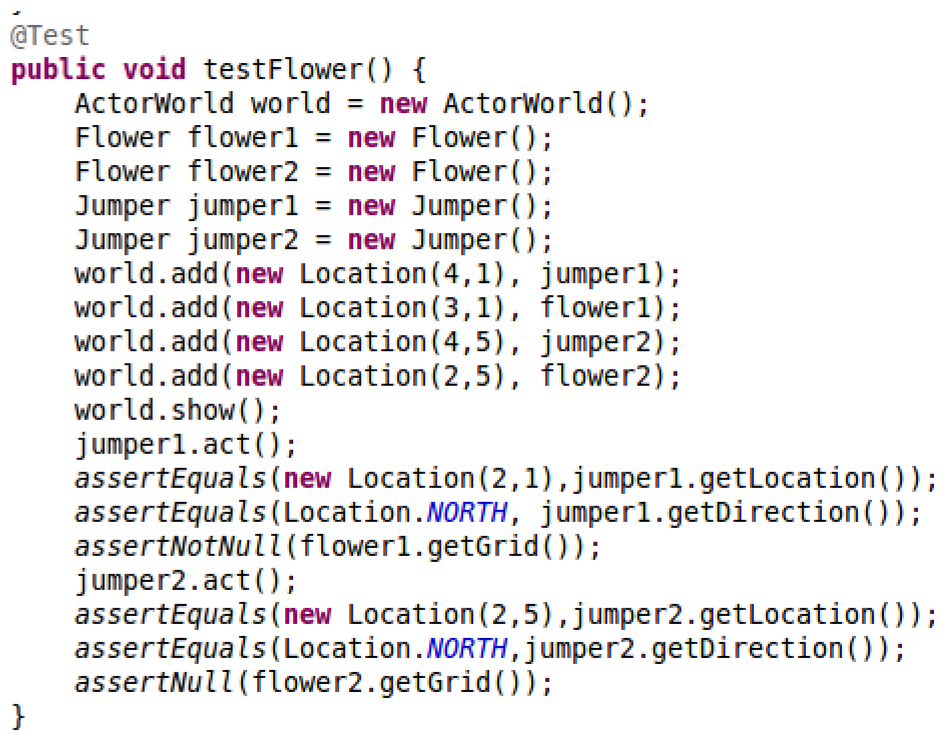


```
@Test
public void testRock() {
    //init
    ActorWorld world = new ActorWorld();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    Rock rock1 = new Rock();
    Rock rock2 = new Rock();
    //adding the actor
    world.add(new Location(4,1), jumper1);
    world.add(new Location(3,1), rock1);
    world.add(new Location(4,5), jumper2);
    world.add(new Location(2,5), rock2);
    world.show();
    jumper1.act();
    assertEquals(new Location(2,1),jumper1.getLocation());
    assertEquals(Location.NORTH, jumper1.getDirection());
    assertNotNull(rock1.getGrid());
    jumper2.act();
    jumper2.act();
    assertEquals(new Location(2,7),jumper2.getLocation());
    assertEquals(Location.NORTHEAST,jumper2.getDirection());
    assertNotNull(rock2.getGrid());

}
```

首先测试 Jumper 离石头一个单位时的情况，此时 Jumper 能够移动两个单位，跳过石头。对 Jumper 的位置和方向进行断言。然后测试 Jumper 离石头两个单位时的情况，同样是对 Jumper 的位置进行断言。这里还另外对 Rock 是否存在进行了断言，测试 Jumper 是否对 Rock 产生了其他影响。

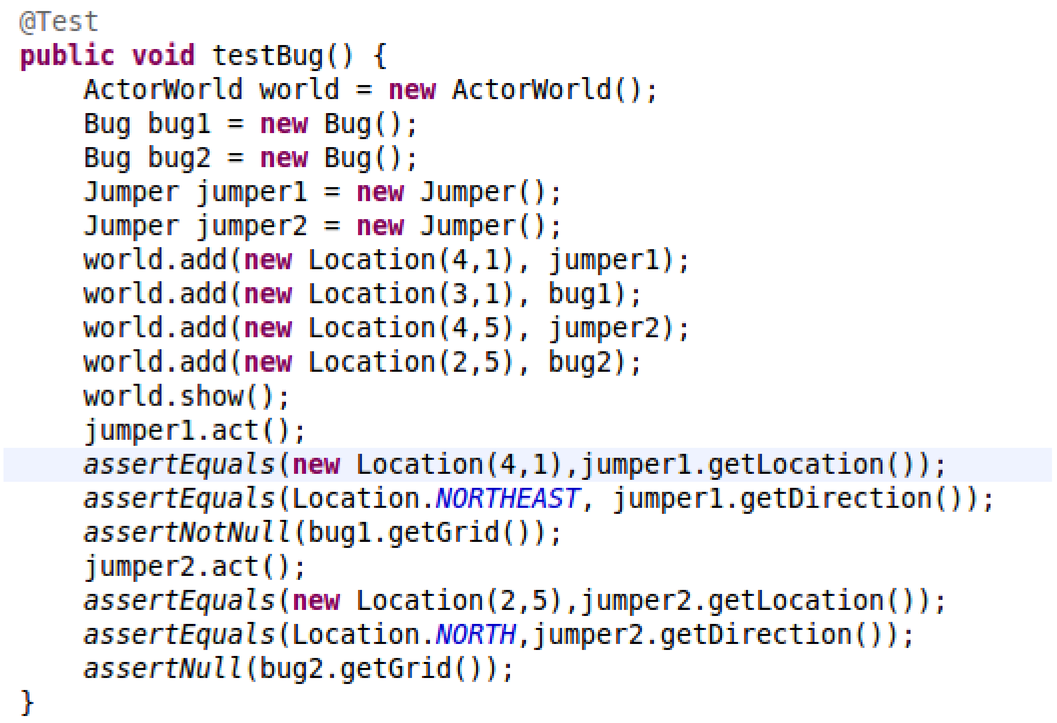
用例 2：Jumper 与 Flower



```java
@Test
public void testFlower() {
    ActorWorld world = new ActorWorld();
    Flower flower1 = new Flower();
    Flower flower2 = new Flower();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    world.add(new Location(4,1), jumper1);
    world.add(new Location(3,1), flower1);
    world.add(new Location(4,5), jumper2);
    world.add(new Location(2,5), flower2);
    world.show();
    jumper1.act();
    assertEquals(new Location(2,1),jumper1.getLocation());
    assertEquals(Location.NORTH, jumper1.getDirection());
    assertNotNull(flower1.getGrid());
    jumper2.act();
    assertEquals(new Location(2,5),jumper2.getLocation());
    assertEquals(Location.NORTH,jumper2.getDirection());
    assertNull(flower2.getGrid());
}
```

首先测试 Jumper 离花一个单位时的情况，然后测试离花两个单位的情况。因为 Jumper 能将花吃掉，所以第二种情况需要检查 Flower 是否已被移除。

用例 3：Jumper 与 Bug

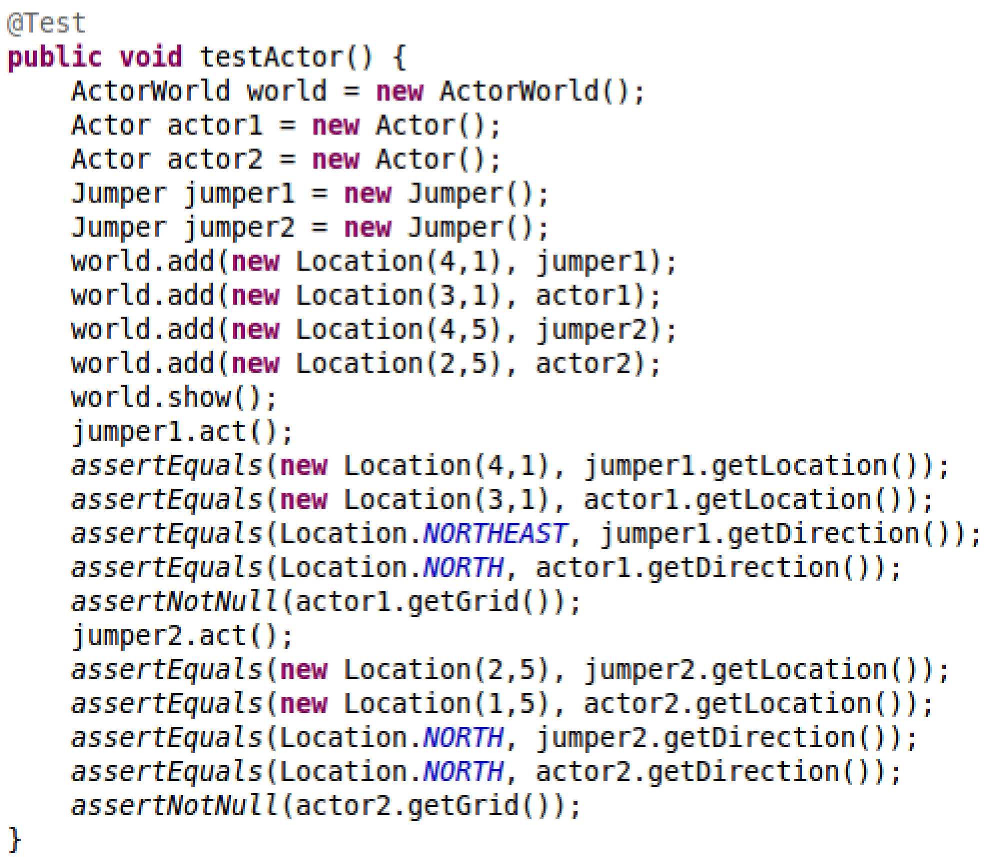


```java
@Test
public void testBug() {
    ActorWorld world = new ActorWorld();
    Bug bug1 = new Bug();
    Bug bug2 = new Bug();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    world.add(new Location(4,1), jumper1);
    world.add(new Location(3,1), bug1);
    world.add(new Location(4,5), jumper2);
    world.add(new Location(2,5), bug2);
    world.show();
    jumper1.act();
    assertEquals(new Location(4,1),jumper1.getLocation());
    assertEquals(Location.NORTHEAST, jumper1.getDirection());
    assertNotNull(bug1.getGrid());
    jumper2.act();
    assertEquals(new Location(2,5),jumper2.getLocation());
    assertEquals(Location.NORTH,jumper2.getDirection());
    assertNull(bug2.getGrid());
}
```

同样是距离为一个单位与两个单位的两种情况，由于虫子也会被 Jumper 吃掉，所以

同样要对虫子是否被移除进行断言。

用例 4：Jumper 与 Actor



```java
@Test
public void testActor() {
    ActorWorld world = new ActorWorld();
    Actor actor1 = new Actor();
    Actor actor2 = new Actor();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    world.add(new Location(4,1), jumper1);
    world.add(new Location(3,1), actor1);
    world.add(new Location(4,5), jumper2);
    world.add(new Location(2,5), actor2);
    world.show();
    jumper1.act();
    assertEquals(new Location(4,1), jumper1.getLocation());
    assertEquals(new Location(3,1), actor1.getLocation());
    assertEquals(Location.NORTHEAST, jumper1.getDirection());
    assertEquals(Location.NORTH, actor1.getDirection());
    assertNotNull(actor1.getGrid());
    jumper2.act();
    assertEquals(new Location(2,5), jumper2.getLocation());
    assertEquals(new Location(1,5), actor2.getLocation());
    assertEquals(Location.NORTH, jumper2.getDirection());
    assertEquals(Location.NORTH, actor2.getDirection());
    assertNotNull(actor2.getGrid());
}
```

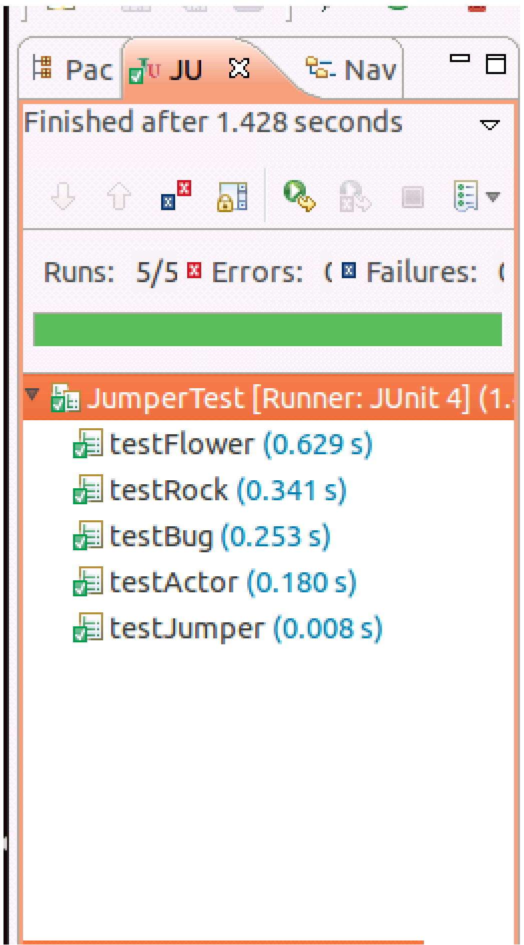
Jumper 不能直接跳过 Actor，所以距离为一的情况下，检查 Jumper 是否转向 。Jumper 会撞飞 Actor，所以距离为二的情况下，除了检查 Jumper 的位置与方向外，还要对 Actor 的位置与方向以及是否被移除进行检查。

用例 5：Jumper 与 Jumper

```java
@Test
public void testJumper() {
    ActorWorld world = new ActorWorld();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    Jumper jumper3 = new Jumper();
    Jumper jumper4 = new Jumper();
    world.add(new Location(4,1), jumper1);
    world.add(new Location(3,1), jumper2);
    world.add(new Location(4,5), jumper3);
    world.add(new Location(2,5), jumper4);
    jumper2.setDirection(Location.SOUTH);
    jumper4.setDirection(Location.SOUTH);
    jumper1.act();
    jumper2.act();
    jumper3.act();
    jumper4.act();
    assertEquals(new Location(4,1),jumper1.getLocation());
    assertEquals(Location.NORTHEAST, jumper1.getDirection());
    assertEquals(new Location(3,1),jumper2.getLocation());
    assertEquals(Location.SOUTHWEST, jumper2.getDirection());
    assertEquals(new Location(4,5), jumper3.getLocation());
    assertEquals(Location.NORTHEAST, jumper3.getDirection());
    assertEquals(new Location(2,5), jumper4.getLocation());
    assertEquals(Location.SOUTHWEST, jumper4.getDirection());
}
```

Jumper 同样不能直接跳过 Jumper，因此距离为一时会发生转向。检查两个 Jumper 的方向以及位置。

2.2 测试结果



```
Finished after 1.428 seconds

Runs:  5/5  Errors: (  Failures: (

JumperTest [Runner: JUnit 4] (1.
    testFlower (0.629 s)
    testRock (0.341 s)
    testBug (0.253 s)
    testActor (0.180 s)
    testJumper (0.008 s)
```

各个样例的检查均通过。

2.3 结果分析

测试结果与预期一致。

用例 1：距离石头一个单位时，Jumper 跳过两个单位；距离石头两个单位时，由于 Jumper 的原落点上有石头，Jumper 先顺时针旋转 45 度，然后再跳两个单位。

用例 2：距离为一时，直接跳过；距离为二时，Jumper 将花吃掉。

用例 3：距离为一时，不能直接跳过，顺时针旋转 45 度；距离为二时，Jumper 将虫子吃掉。

用例 4：距离为一时，不能直接跳过，转向；距离为二时，Jumper 将 Actor 撞飞，方向为 Jumper 的原方向。

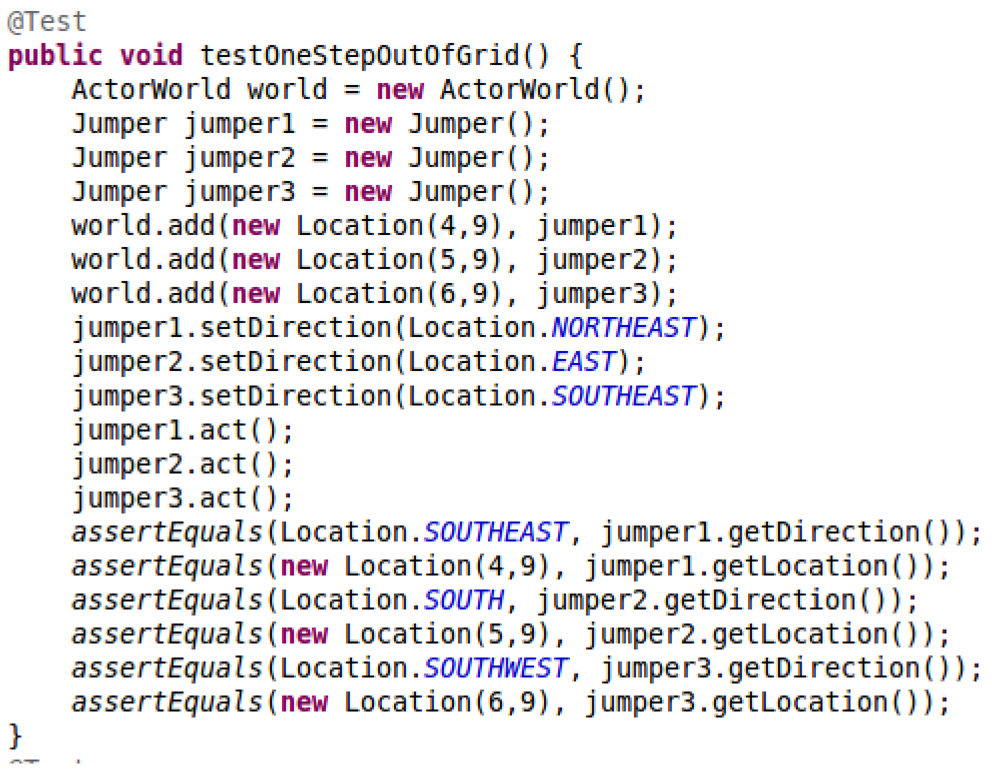
用例 5：距离为一和二时，两个 Jumper 都发生了顺时针 45 度的转向。

3 用例 2

测试目的：检查 Jumper 对于 Grid 边界的处理。

3.1 测试用例

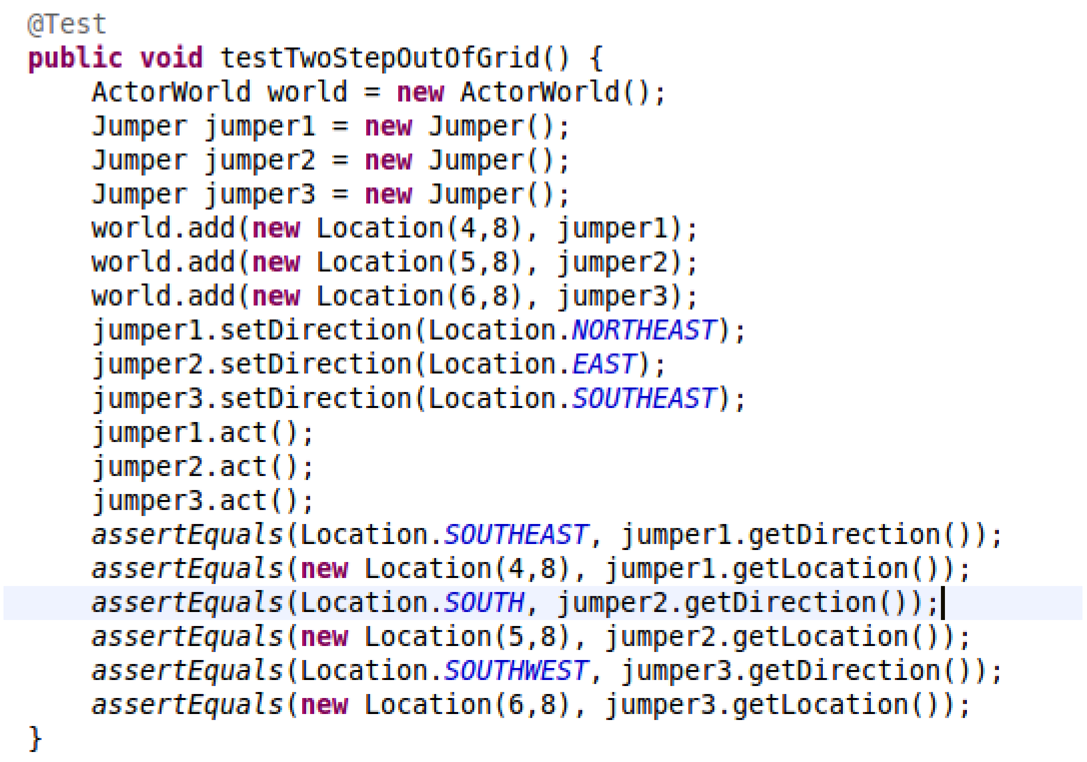
用例 1：当 Jumper 距离边界距离为 0 时的情况。



```
@Test
public void testOneStepOutOfGrid() {
    ActorWorld world = new ActorWorld();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    Jumper jumper3 = new Jumper();
    world.add(new Location(4,9), jumper1);
    world.add(new Location(5,9), jumper2);
    world.add(new Location(6,9), jumper3);
    jumper1.setDirection(Location.NORTHEAST);
    jumper2.setDirection(Location.EAST);
    jumper3.setDirection(Location.SOUTHEAST);
    jumper1.act();
    jumper2.act();
    jumper3.act();
    assertEquals(Location.SOUTHEAST, jumper1.getDirection());
    assertEquals(new Location(4,9), jumper1.getLocation());
    assertEquals(Location.SOUTH, jumper2.getDirection());
    assertEquals(new Location(5,9), jumper2.getLocation());
    assertEquals(Location.SOUTHWEST, jumper3.getDirection());
    assertEquals(new Location(6,9), jumper3.getLocation());
}
```

检查 act()后 Jumper 的方向与位置以检验是否跳出。

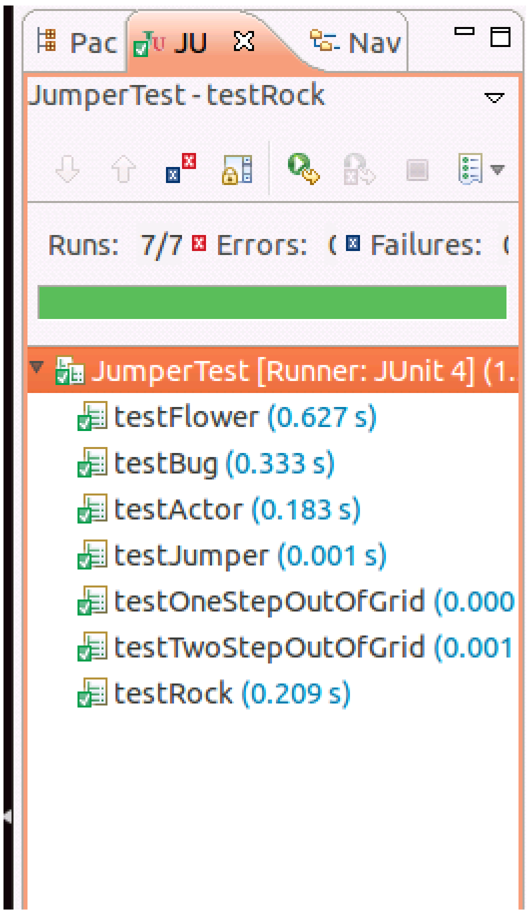
用例 2：当 Jumper 距离边界距离为 1 时的情况。



```java
@Test
public void testTwoStepOutOfGrid() {
    ActorWorld world = new ActorWorld();
    Jumper jumper1 = new Jumper();
    Jumper jumper2 = new Jumper();
    Jumper jumper3 = new Jumper();
    world.add(new Location(4,8), jumper1);
    world.add(new Location(5,8), jumper2);
    world.add(new Location(6,8), jumper3);
    jumper1.setDirection(Location.NORTHEAST);
    jumper2.setDirection(Location.EAST);
    jumper3.setDirection(Location.SOUTHEAST);
    jumper1.act();
    jumper2.act();
    jumper3.act();
    assertEquals(Location.SOUTHEAST, jumper1.getDirection());
    assertEquals(new Location(4,8), jumper1.getLocation());
    assertEquals(Location.SOUTH, jumper2.getDirection());
    assertEquals(new Location(5,8), jumper2.getLocation());
    assertEquals(Location.SOUTHWEST, jumper3.getDirection());
    assertEquals(new Location(6,8), jumper3.getLocation());
}
```

同样例 1。

3.2 测试结果



```
JumperTest - testRock

Runs: 7/7  Errors: (  Failures: (

JumperTest [Runner: JUnit 4] (1.
    testFlower (0.627 s)
    testBug (0.333 s)
    testActor (0.183 s)
    testJumper (0.001 s)
    testOneStepOutOfGrid (0.000
    testTwoStepOutOfGrid (0.001
    testRock (0.209 s)
```

3.3 结果分析

结果与预期一致。

在用例 1 和 2 中，Jumper 都会在面对边界时顺时针旋转 90 度调整，而不会跳出边界。