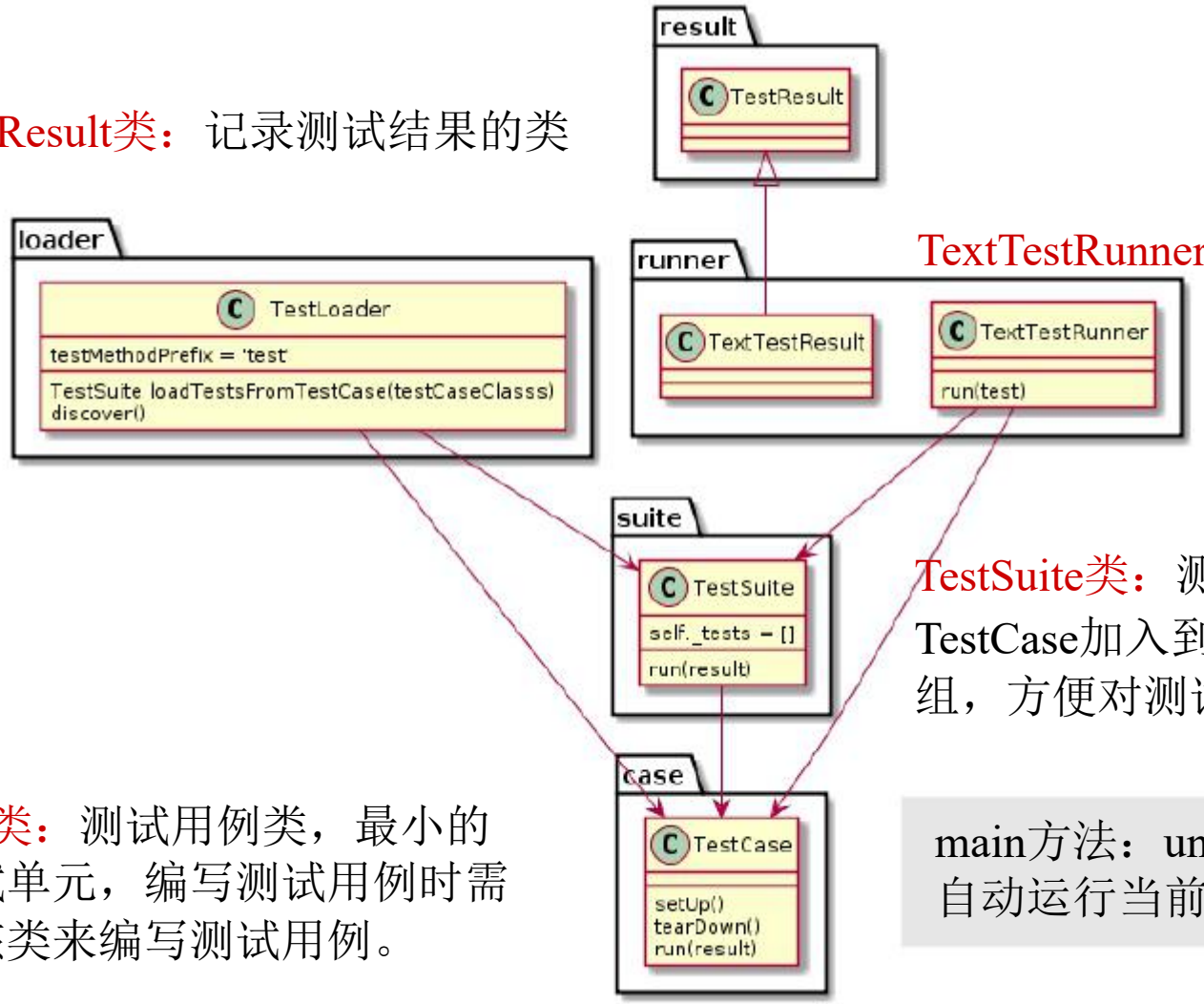
# 实验二：单元测试

## Python单元测试工具

# Python单元测试之unittest

TestResult类：记录测试结果的类

TestLoader类：加载测试的类，通常无需手动创建。




TextTestRunner类：执行测试用例

TestSuite类：测试套件类，可以将TestCase加入到TestSuite中进行分组，方便对测试进行管理。

TestCase类：测试用例类，最小的完整测试单元，编写测试用例时需要继承该类来编写测试用例。

main方法：unittest.main()可以自动运行当前文件中的所有测试

# Python单元测试之unittest

测试过程中需要通过属性断言对结果进行判断，以验证结果是否满足需求。

- TestCase类提供了多种强大的断言方法，如asserTrue, assertFalse, assertEqual, assertNotEqual, assertIs等。

  参考文档见 https://docs.python.org/3/library/unittest.html

- 这些断言方法可以在断言的同时加上一个message参数，这样可以使断言的意义明确而且方便维护，在测试失败时抛出可读的信息。

# Python单元测试之unittest

① import unittest

② 定义一个继承自unittest.TestCase的测试用例类

③ 定义setUp和tearDown，在每个测试用例前后做一些辅助工作

④ 定义测试用例，名字以test开头

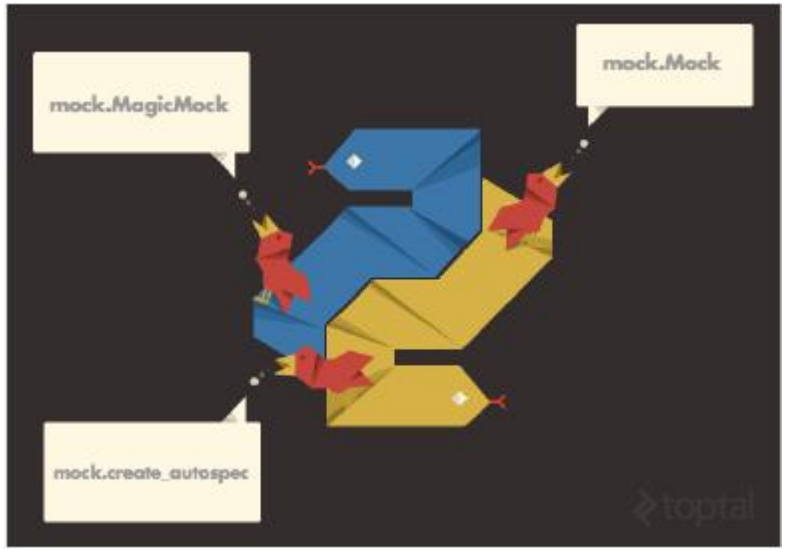
⑤ 一个测试用例应该只测试一个方面，测试目的和测试内容应很明确。主要是调用assertEqual、assertRaises等断言方法判断程序执行结果和预期值是否相符

⑥ 调用unittest.main()启动测试

⑦ 如果测试未通过，会输出相应的错误提示；如果测试全部通过则显示ok，添加-v参数显示详细信息。

# Python单元测试之mock

Python 3.3开始内置了Mock工具包，可以使用mock对象替代掉指定的Python对象，以达到模拟对象的行为。




- Mock类：用于创建mock对象，当访问mock对象的某个属性时，mock对象会自动创建该属性。
- MagicMock类：Mock对象的子类，预先定义了操作符（如__lt__, __len__）。
- patch装饰器：可以将其作用在测试方法上，限定在当前测试方法中使用mock来替换真实对象。

# Python单元测试之mock

属性断言：mock对象提供了一系列断言方法，可以在使用属性断言时判断程序对mock对象的调用是否符合预期。

- assert_called_with， assert_called_once_with， assert_any_call， assert_has_calls
- https://docs.python.org/3/library/unittest.mock.html#the-mock-class

行为控制：通常程序需要从依赖对象的方法上取得返回值，mock对象也提供了一些途径对返回值进行控制。

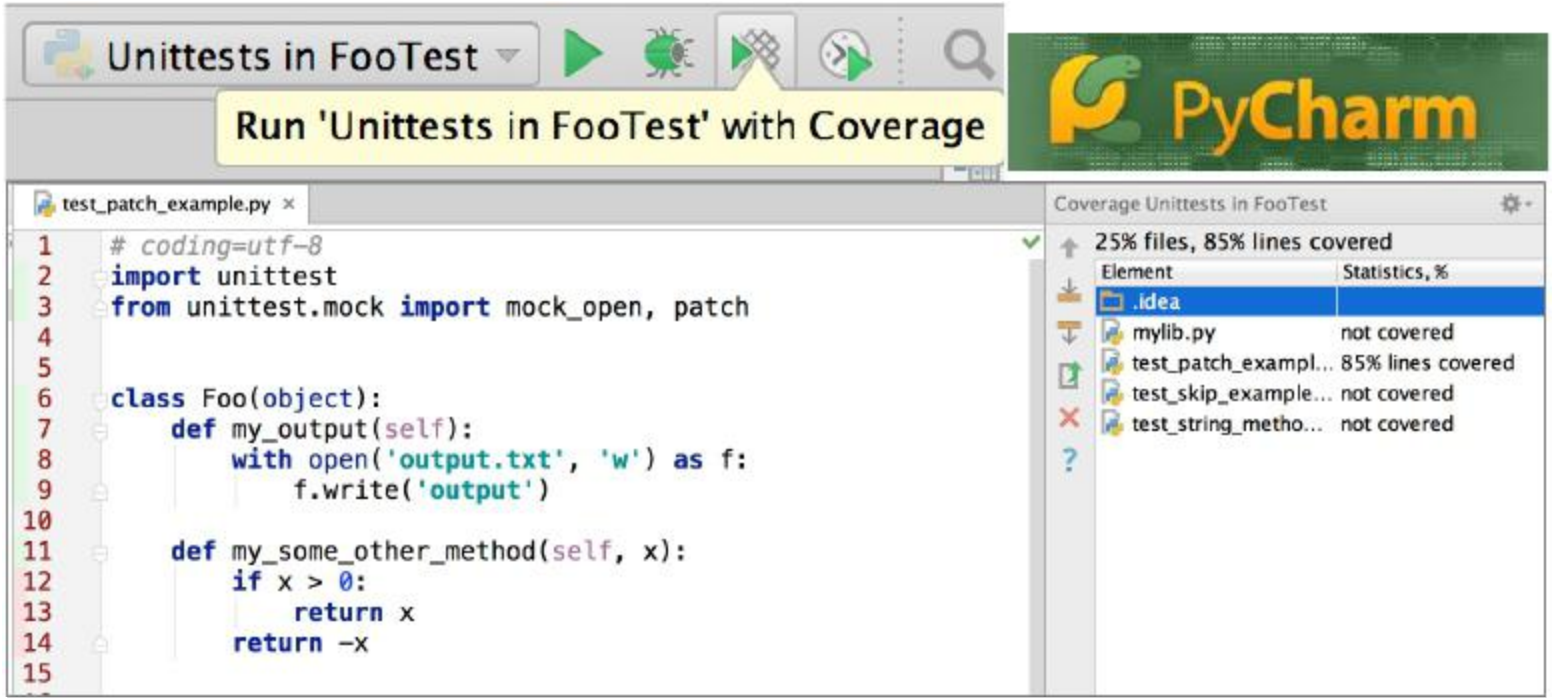
- return_value: 固定返回值
- side_effects: 返回值的序列或自定义方法

# Python单元测试之覆盖分析

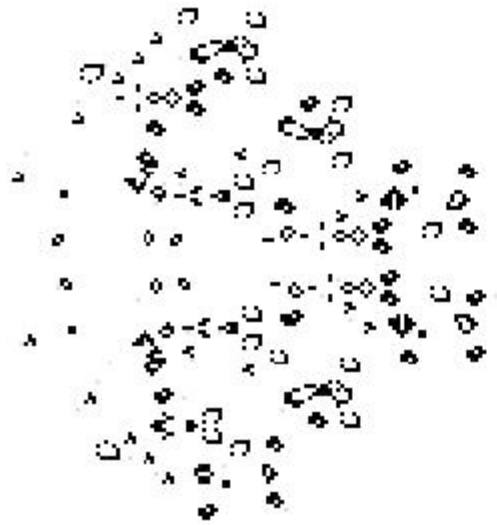
coverage.py是一个用来统计python程序代码覆盖率的工具，它使用起来非常简单，并且支持最终生成界面友好的html报告。

安装 ▶▶ pip install -U coverage.py

使用 ▶▶



# 案例：生命游戏

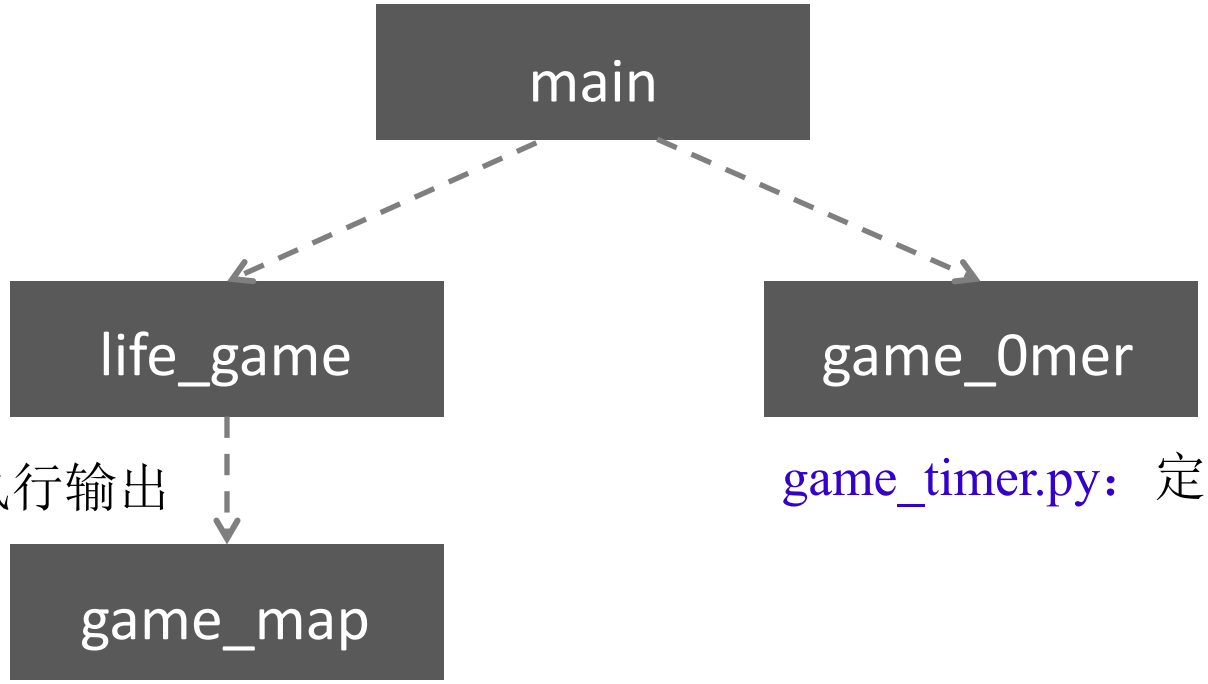


Python单元测试

- unittest
- mock
- coverage.py

# 案例：生命游戏

main.py：生命游戏的主程序，用户使用的入口




life_game.py：游戏的执行输出

game_timer.py：定时器，定时触发

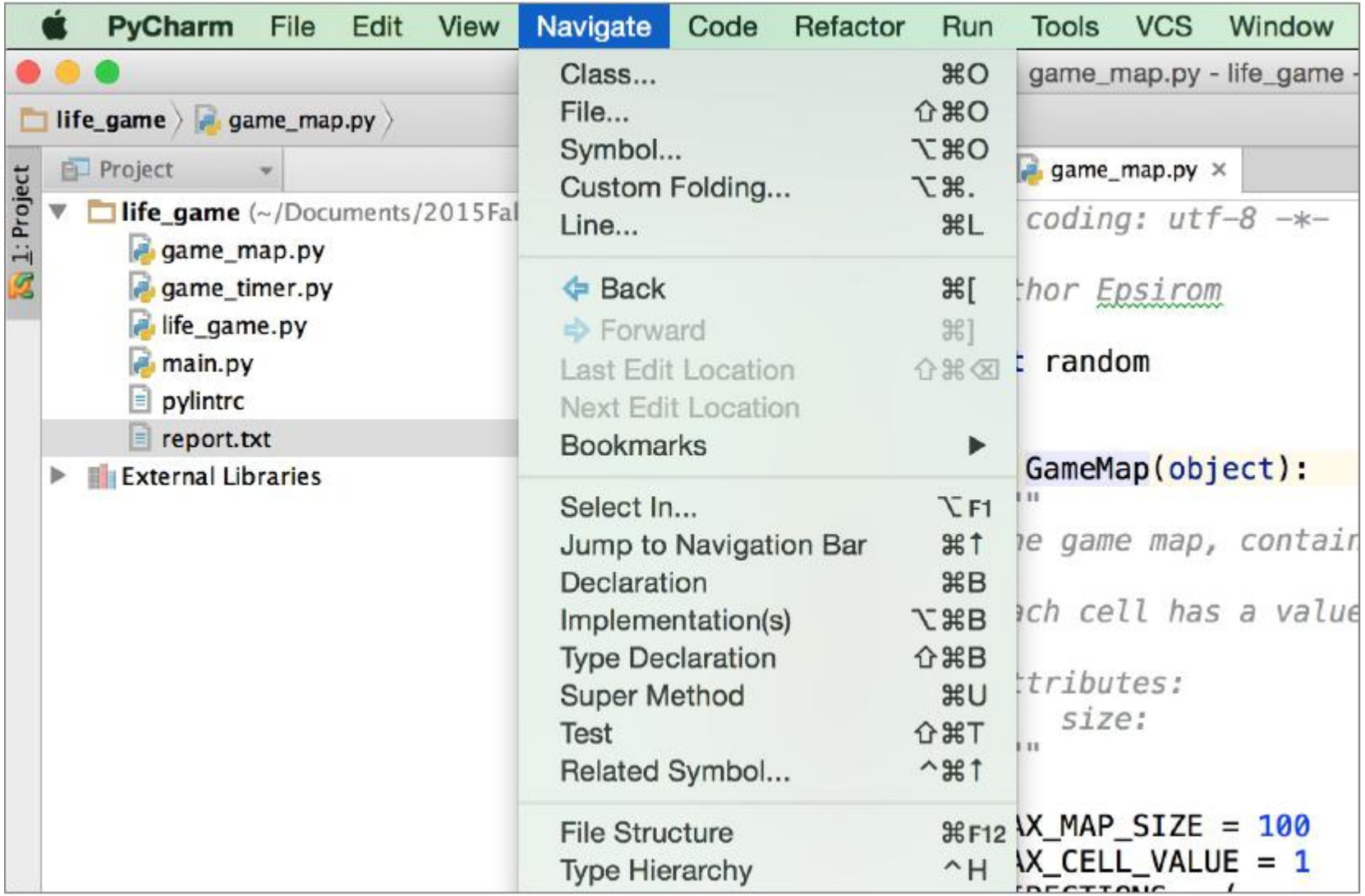
game_map.py：生命游戏地图，包含了所需的底层操作

# 案例：生命游戏

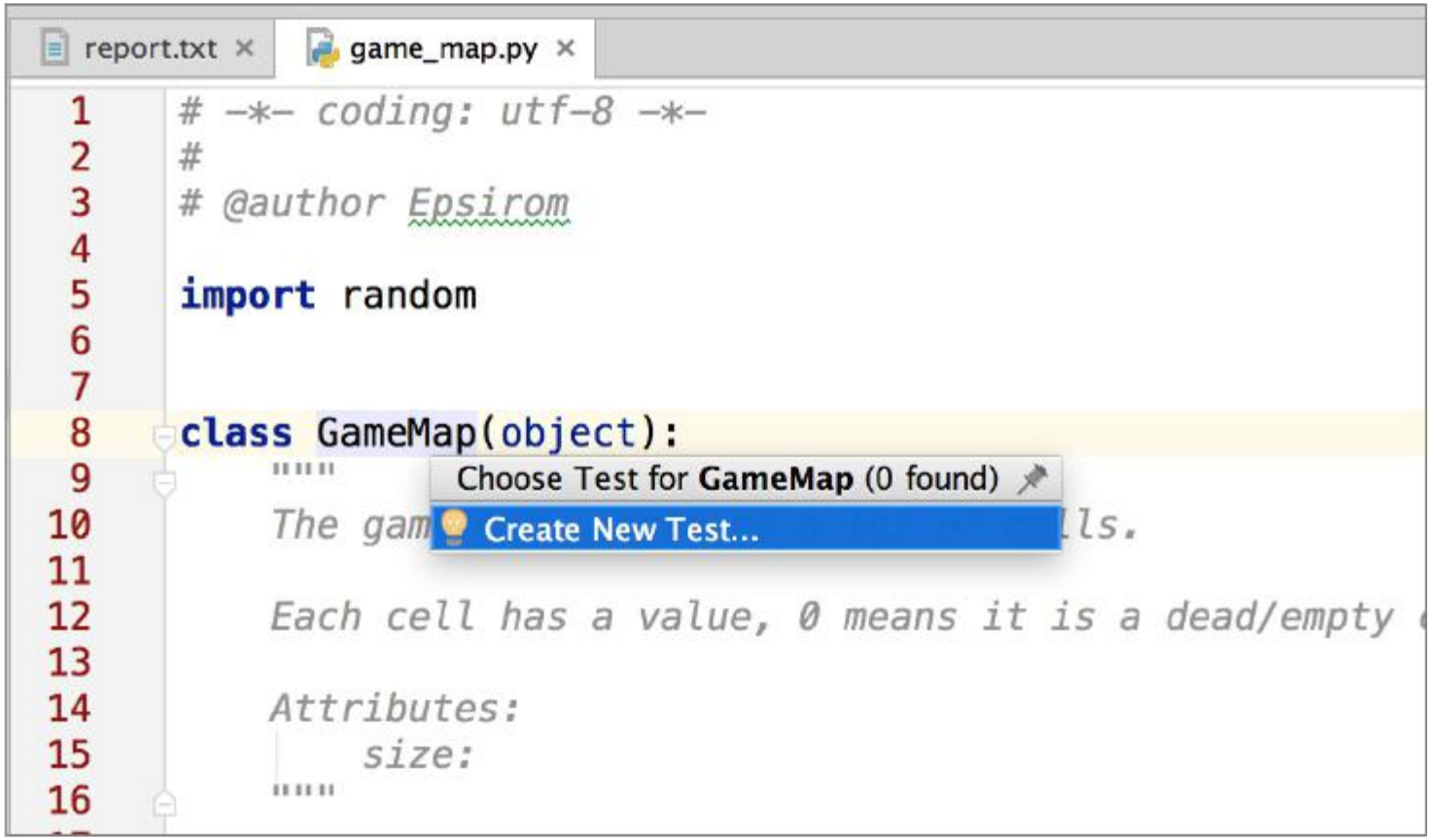
game_map

- rows，cols：表示地图的行数和列数
- reset：以一定的概率设置地图的每个格子的状态
- get/set：获取、设置地图的某个格子的状态
- get_neighbor_count：获取一个格子的邻居数量
- get_neighbor_count_map：获取每个格子的邻居数量
- set_map：设置地图
- print_map：打印地图

创建测试


PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window
life_game game_map.py
Project
life_game (~/Documents/2015Fal
game_map.py
game_timer.py
life_game.py
main.py
pylintrc
report.txt
External Libraries
1: Project
Class... ⌘O
File... ⇧⌘O
Symbol... ⌥⌘O
Custom Folding... ⌥⌘.
Line... ⌘L
Back ⌘[
Forward ⌘]
Last Edit Location ⇧⌘⌫
Next Edit Location
Bookmarks
Select In... ⌥F1
Jump to Navigation Bar ⌘↑
Declaration ⌘B
Implementation(s) ⌥⌘B
Type Declaration ⇧⌘B
Super Method ⌘U
Test ⇧⌘T
Related Symbol... ^⌘↑
File Structure ⌘F12
Type Hierarchy ^H
game_map.py - life_game
game_map.py
coding: utf-8 -*-
thor Epsirom
random
GameMap(object):
e game map, contain
ch cell has a value
tributes:
size:
X_MAP_SIZE = 100
X_CELL_VALUE = 1

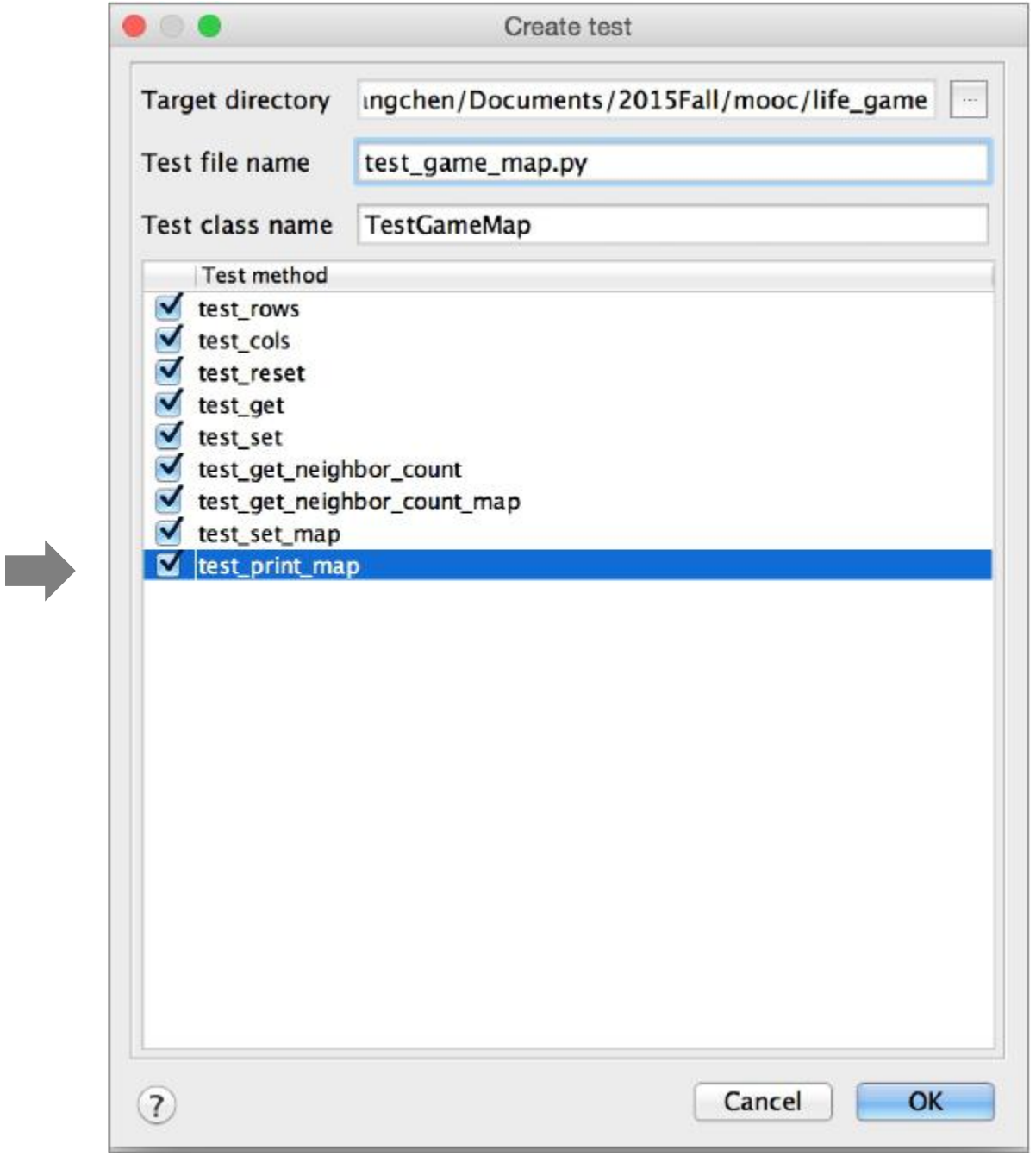
创建测试


report.txt
game_map.py
1 # -*- coding: utf-8 -*-
2 #
3 # @author Epsirom
4
5 import random
6
7
8 class GameMap(object):
9 """
10 The gam ls.
11
12 Each cell has a value, 0 means it is a dead/empty
13
14 Attributes:
15 size:
16 """
Choose Test for GameMap (0 found)
Create New Test...

创建测试


Create test
Target directory
ıngchen/Documents/2015Fall/mooc/life_game
Test file name
test_gameMap.py
Test class name
TestGameMap
Test method
test_rows
test_cols
test_reset
test_get
test_set
test_get_neighbor_count
test_get_neighbor_count_map
test_set_map
test_print_map
Cancel
OK

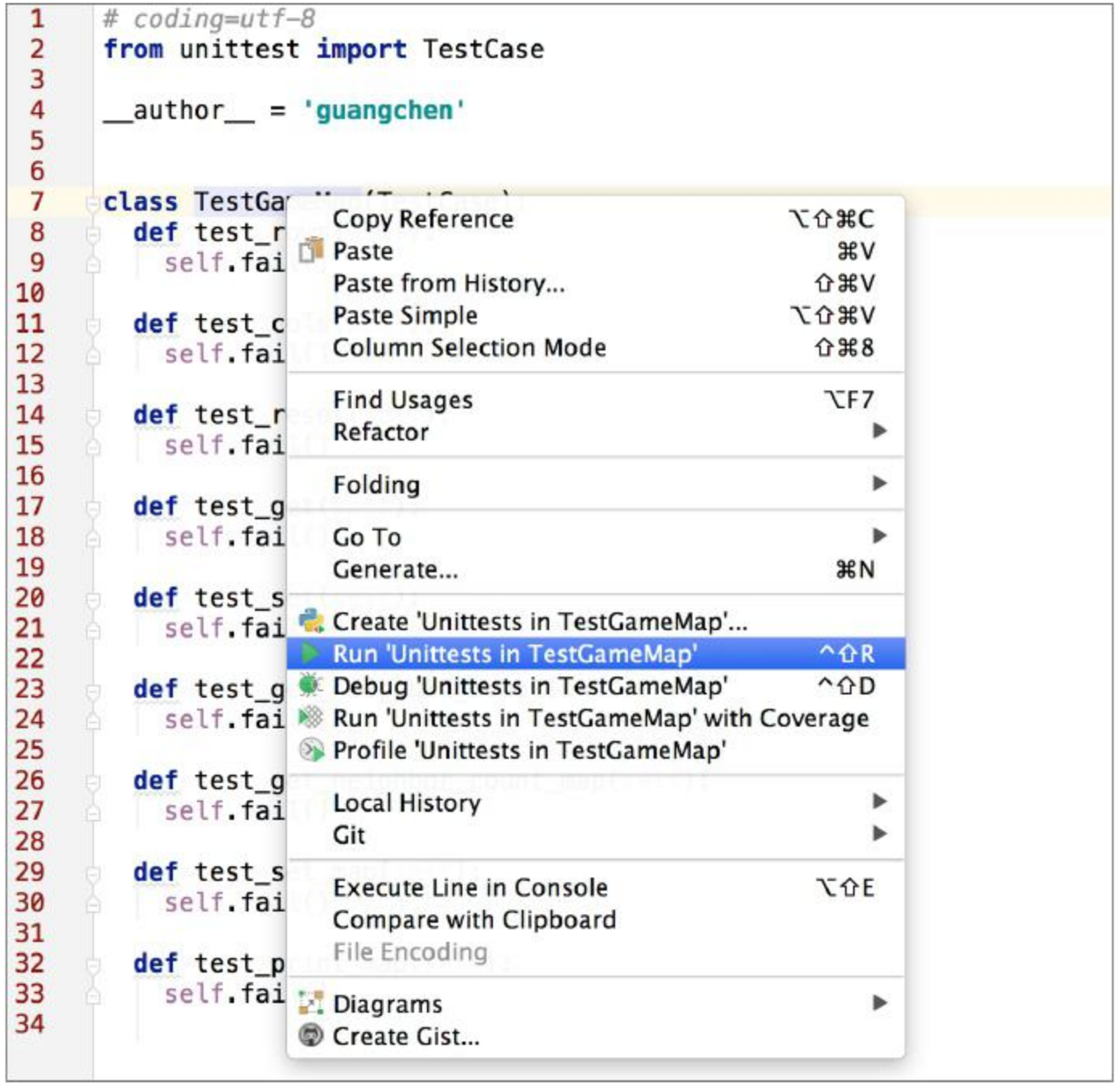
创建测试

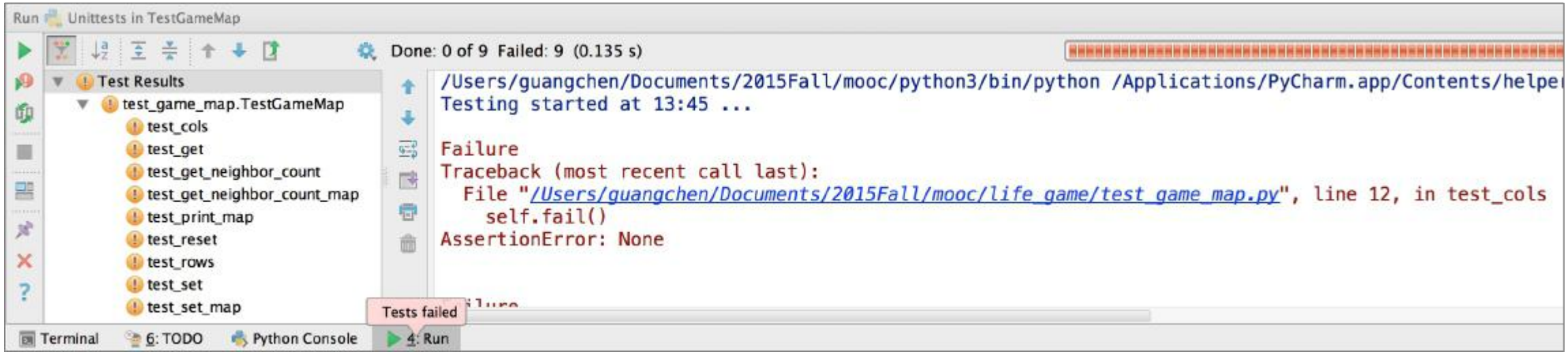

Create test
Target directory
ıngchen/Documents/2015Fall/mooc/life_game
Test file name
test_game_map.py
Test class name
TestGameMap
Test method
test_rows
test_cols
test_reset
test_get
test_set
test_get_neighbor_count
test_get_neighbor_count_map
test_set_map
test_print_map
Cancel
OK

创建测试

```python
# coding=utf-8
from unittest import TestCase

__author__ = 'guangchen'


class TestGameMap(TestCase):
    def test_rows(self):
        self.fail()

    def test_cols(self):
        self.fail()

    def test_reset(self):
        self.fail()

    def test_get(self):
        self.fail()

    def test_set(self):
        self.fail()

    def test_get_neighbor_count(self):
        self.fail()

    def test_get_neighbor_count_map(self):
        self.fail()

    def test_set_map(self):
        self.fail()

    def test_print_map(self):
        self.fail()

```

运行测试


1 # coding=utf-8
2 from unittest import TestCase
3
4 __author__ = 'guangchen'
5
6
7 class TestGa
8 def test_r
9 self.fai
10
11 def test_c
12 self.fai
13
14 def test_r
15 self.fai
16
17 def test_g
18 self.fai
19
20 def test_s
21 self.fai
22
23 def test_g
24 self.fai
25
26 def test_g
27 self.fai
28
29 def test_s
30 self.fai
31
32 def test_p
33 self.fai
34
Copy Reference ⌥⇧⌘C
Paste ⌘V
Paste from History... ⇧⌘V
Paste Simple ⌥⇧⌘V
Column Selection Mode ⇧⌘8
Find Usages ⌥F7
Refactor
Folding
Go To
Generate... ⌘N
Create 'Unittests in TestGameMap'...
Run 'Unittests in TestGameMap' ^⇧R
Debug 'Unittests in TestGameMap' ^⇧D
Run 'Unittests in TestGameMap' with Coverage
Profile 'Unittests in TestGameMap'
Local History
Git
Execute Line in Console ⌥⇧E
Compare with Clipboard
File Encoding
Diagrams
Create Gist...




TestCase.fail() 无条件使当前测试失败

# 案例：生命游戏

- setUp方法：创建每个测试方法都需要的公共对象
- tearDown方法：销毁公共对象（如果需要的话），如数据库断开连接等

创建测试fixture

这里只需要setUp方法，并在其中创建一个GameMap待测对象

```
class TestGameMap(TestCase):
    def setUp(self):
        self.game_map = GameMap(4, 3)
```

# 案例：生命游戏

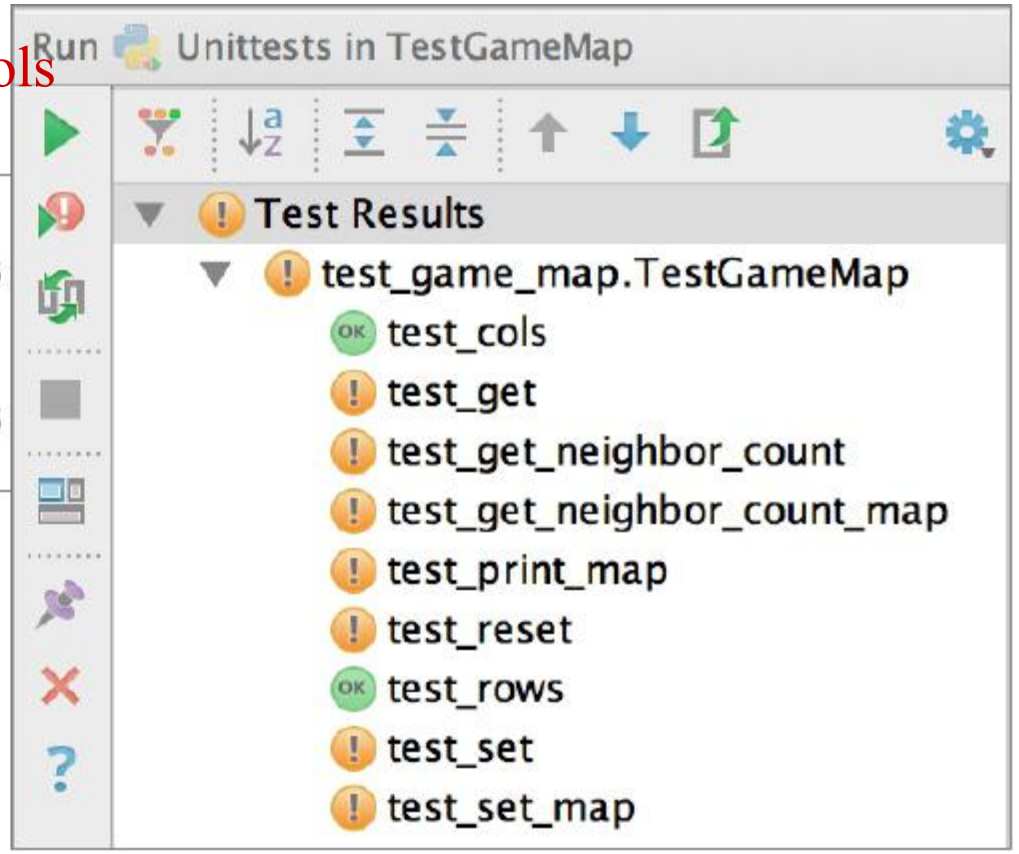
属性测试 ➡ 测试 rows 和 cols

```
def test_rows(self):
    self.assertEqual(4, self.game_map.rows, "Should get correct rows")

def test_cols(self):
    self.assertEqual(3, self.game_map.cols, "Should get correct cols")
```

# 案例：生命游戏

属性测试 → 测试 rows 和 cols

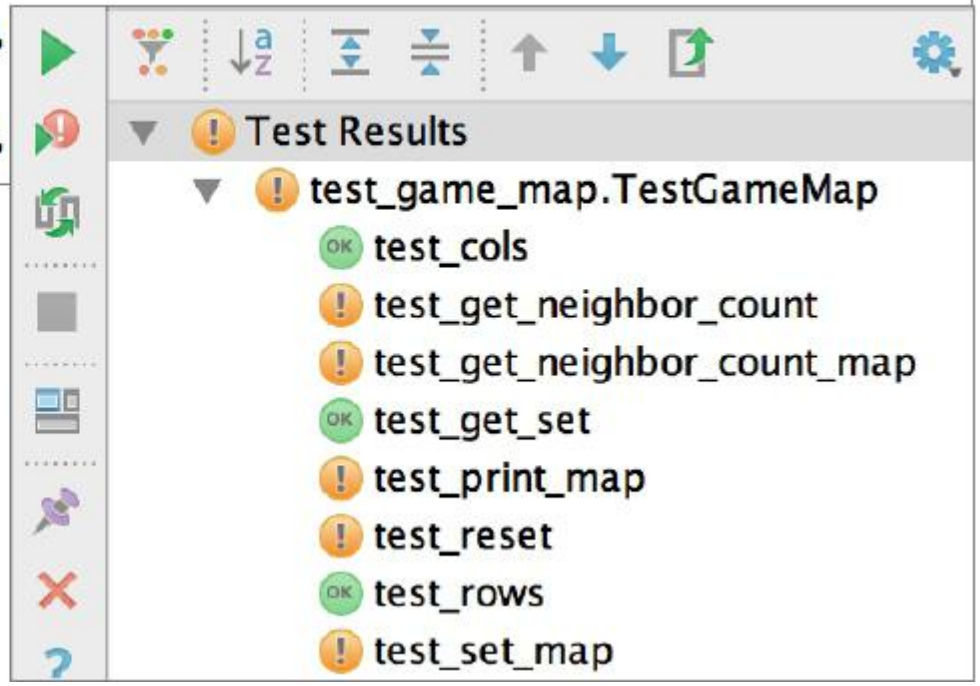
```
def test_rows(self):
    self.assertEqual(4, self.game_map.rows

def test_cols(self):
    self.assertEqual(3, self.game_map.cols
```



# 案例：生命游戏

方法测试 ➡ get/set：两个方法相互联系，合并为一个测试

```
def test_get_set(self):
    self.assertEqual(0, self.game_map.get(0,
    self.game_map.set(0, 0, 1)
    self.assertEqual(1, self.game_map.get(0,
```



# 案例：生命游戏

方法测试 ➡ reset：依赖概率，需要进行mock

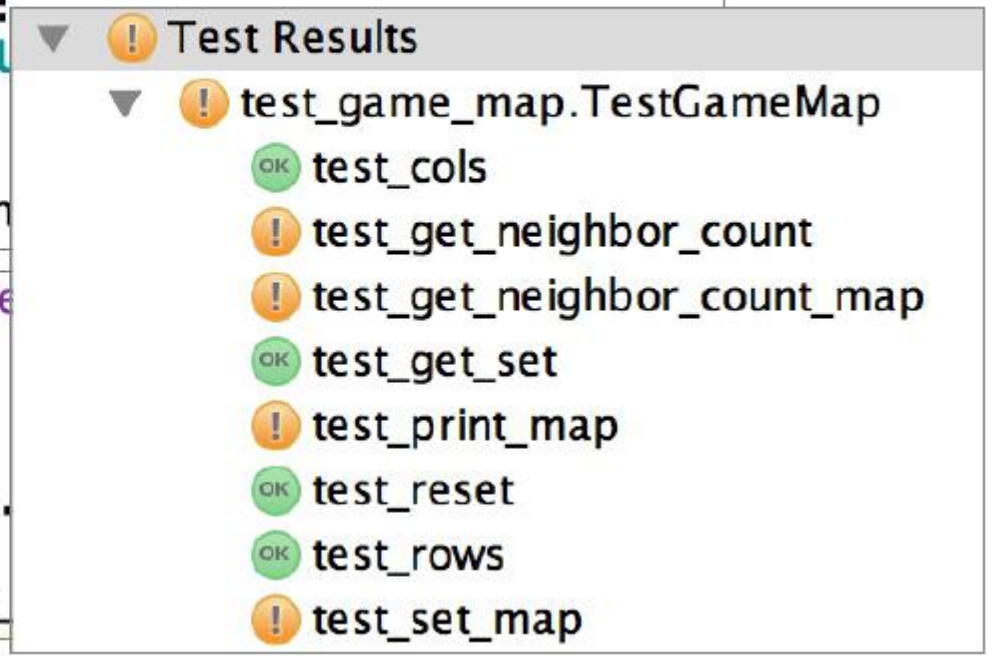
```
def reset(self, possibility=0.5):
    """Reset the map with random data."""
    if not isinstance(possibility, float):
        raise TypeError("possibility should be float")
    for row in self.cells:
        for col_num in range(self.cols):
            row[col_num] = 1 if random.random() < possibility else 0
```

# 案例：生命游戏

方法测试 ➡ reset：依赖概率，需要进行mock

```
def reset(self, possibility=0.5):
    """Reset the map with random data."""
    if not isinstance(possibility, float):
        raise TypeError("possibility shoul
    for row in self.cells:
        for col_num in range(self.cols):
            row[col_num] = 1 if random.ran
```

```
@patch('random.random', new=Mock(side_effe
def test_reset(self):
    self.game_map.reset()
    for i in range(0, 4):
        self.assertEqual(1, self.game_map.
        for j in range(1, 3):
            self.assertEqual(0, self.game_
```



# 方法测试

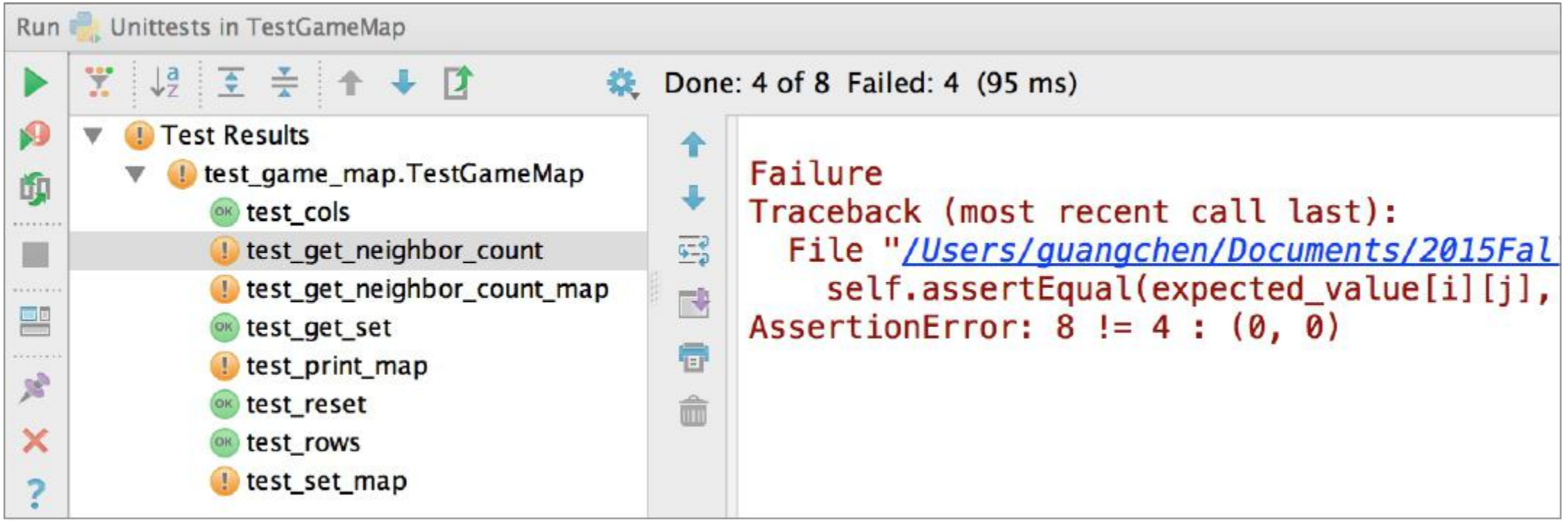
get_neighbor_count

```python
def get_neighbor_count(self, row, col):
    """Get count of neighbors in specific cell.

    Args:
        row: row number
        col: column number

    Returns:
        Count of live neighbor cells
    """
    if not isinstance(row, int):
        raise TypeError("row should be int")
    if not isinstance(col, int):
        raise TypeError("col should be int")
    assert 0 <= row < self.rows
    assert 0 <= col < self.cols
    count = 0
    for d in self.DIRECTIONS:
        d_row = row + d[0]
        d_col = col + d[1]
        if d_row >= self.rows:
            d_row -= self.rows
        if d_col >= self.cols:
            d_col -= self.cols
        count += self.cells[d_row][d_col]
    return count
```

# 方法测试

get_neighbor_count

```
def test_get_neighbor_count(self):
    expected_value = [[8] * 3] * 4
    self.game_map.cells = [[1] * 3] * 4
    for i in range(0, 4):
        for j in range(0, 3):
            self.assertEqual(expected_value[i][j], (self.game_map.get_neighbor_count(i, j)), '(%d, %d)' % (i, j))
```



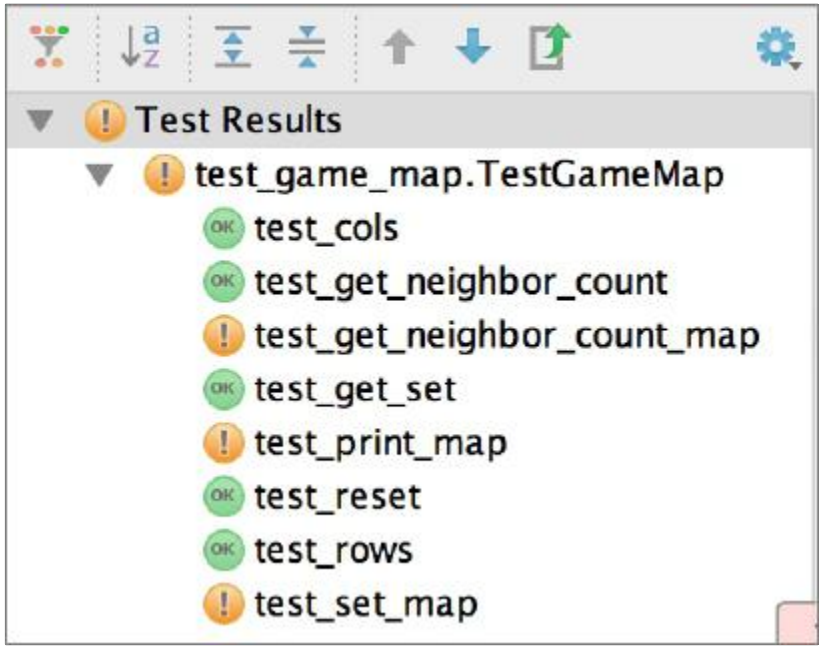
# 方法测试

get_neighbor_count

```
count = 0
for d in self.DIRECTIONS:
    d_row = row + d[0]
    d_col = col + d[1]
    if d_row >= self.rows:
        d_row -= self.rows
    if d_col >= self.cols:
        d_col -= self.cols
    count += self.cells[d_row][d_col]
return count
```

```
DIRECTIONS = (
    (0, 1, ),
    (0, -1, ),
    (1, 0, ),
    (-1, 0, )
)
```

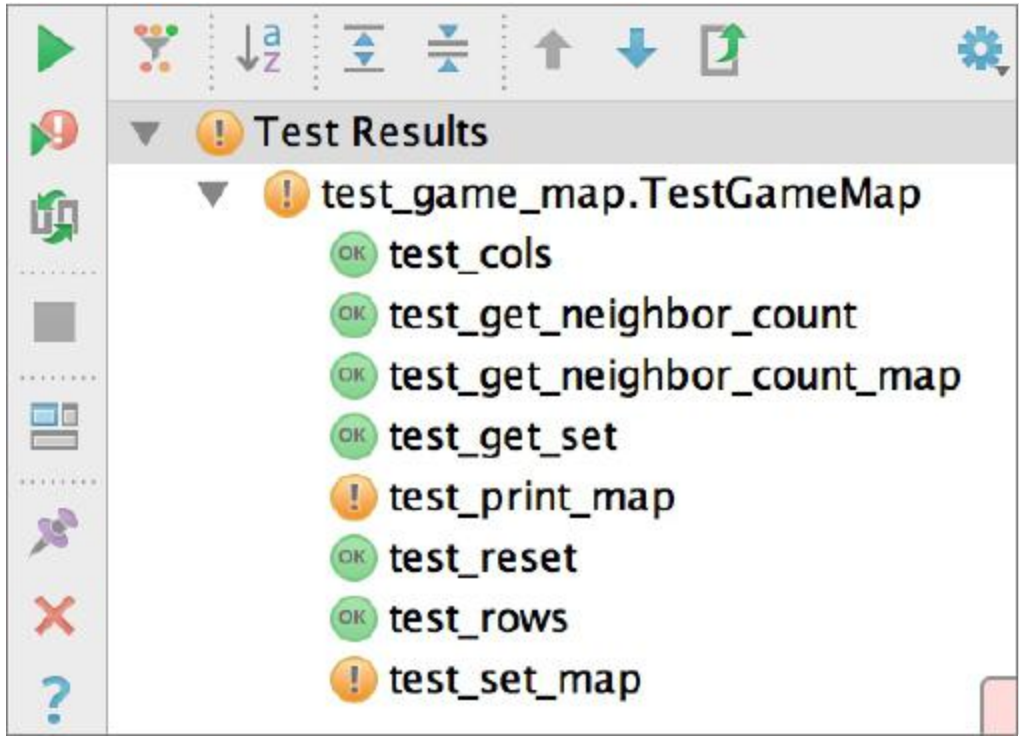
```
DIRECTIONS = (
    (0, 1, ),
    (0, -1, ),
    (1, 0, ),
    (-1, 0, ),
    (1, 1),
    (1, -1),
    (-1, 1),
    (-1, -1)
)
```



## 方法测试

get_neighbor_count_map：依赖 get_neighbor_count，测试时对依赖方法进行mock，保持测试的独立性。

```
@patch('game_map.GameMap.get_neighbor_count', new=Mock(return_value=8))
def test_get_neighbor_count_map(self):
    expected_value = [[8] * 3] * 4
    self.assertEqual(expected_value, self.game_map.get_neighbor_count_map())
```



# 案例：生命游戏

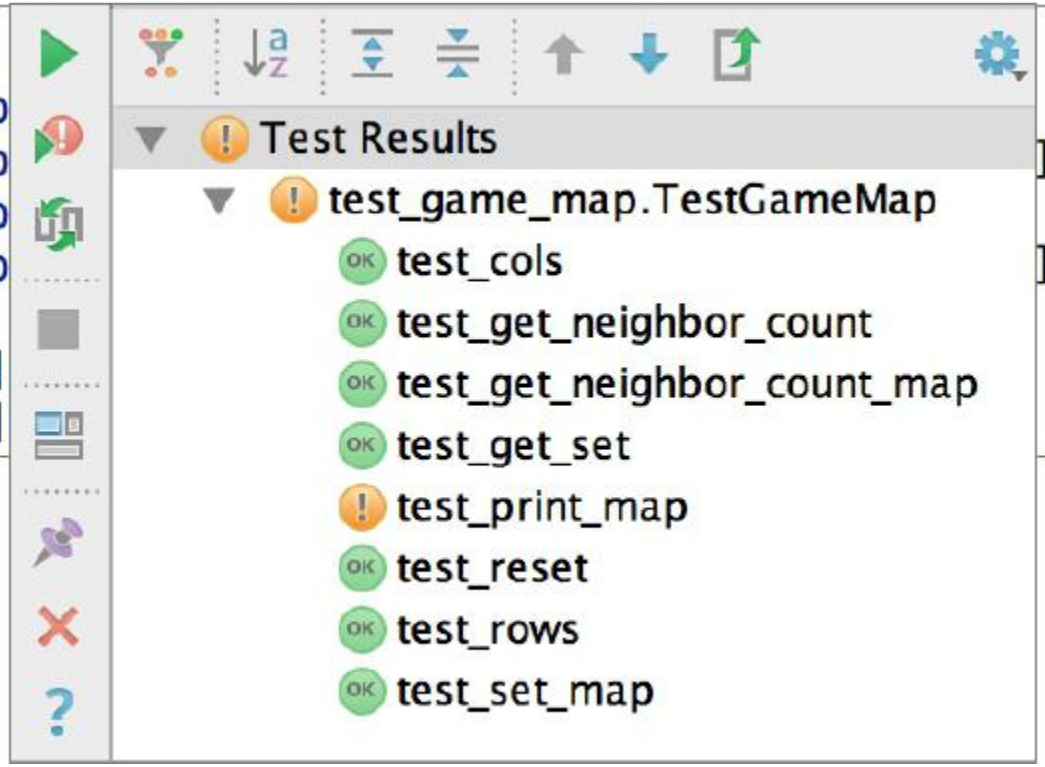
方法测试 ➡ set_map

```
def set_map(self, new_map):
    if not isinstance(new_map, list):
        raise TypeError("new_map should be list")
    assert len(new_map) == self.rows
    for row in new_map:
        if not isinstance(row, list):
            raise TypeError("rows in new_map should be list")
        assert len(row) == self.cols
        for cell in row:
            if not isinstance(cell, int):
                raise TypeError("cells in new_map should be int")
            assert 0 <= cell <= self.MAX_CELL_VALUE
    self.cells = new_map
```

# 案例：生命游戏

方法测试 ➡ set_map

```
def test_set_map(self):
    self.assertRaises(TypeErro
    self.assertRaises(Assertio
    self.assertRaises(TypeErro
    self.assertRaises(Assertio

    self.game_map.set_map([[1]
    self.assertEqual([[1] * 3]
```



# 案例：生命游戏

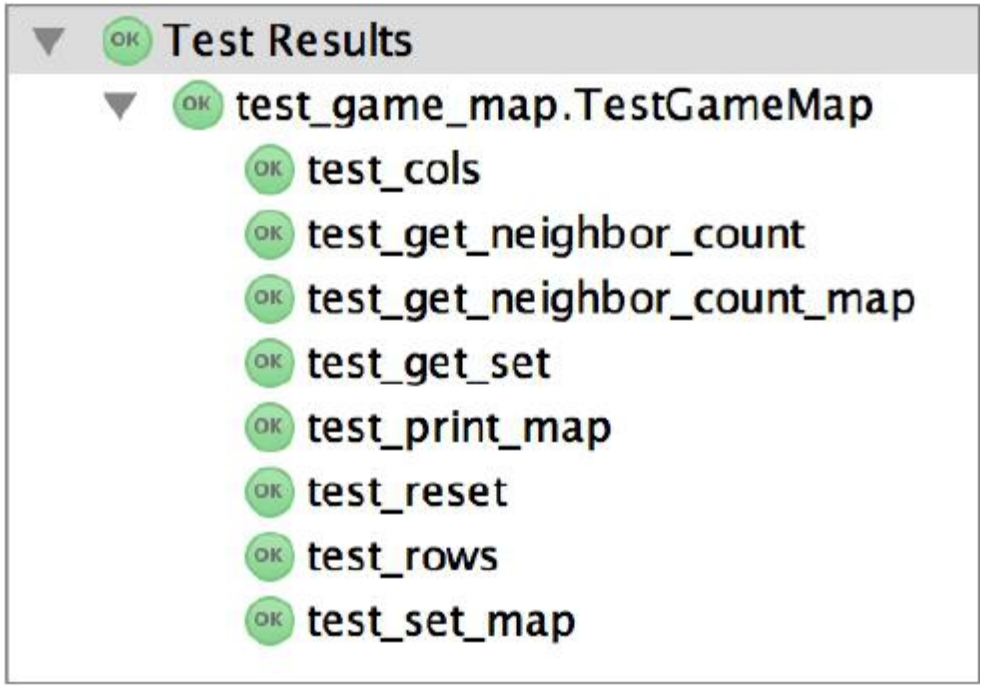
方法测试 ➡ print_map

```
def print_map(self, cell_maps=None, sep=' '):
    if not cell_maps:
        cell_maps = ['0', '1']
    if not isinstance(cell_maps, list) and not isinstance(cell_maps, dict):
        raise TypeError("cell_maps should be list or dict")
    if not isinstance(sep, str):
        raise TypeError("sep should be string")
    for row in self.cells:
        print(sep.join([cell_maps[cell] for cell in row]))
```

# 案例：生命游戏

方法测试 ➡ print_map

```
def test_print_map(self):
    self.game_map.cells = [
        [0, 1, 1],
        [0, 0, 1],
        [1, 1, 1],
        [0, 0, 0]
    ]
    with patch('builtins.print') as mock:
        self.game_map.print_map()
        mock.assert_has_calls([
            call('0 1 1'),
            call('0 0 1'),
            call('1 1 1'),
            call('0 0 0'),
        ])
```



覆盖率分析


```
33          if not isinstance(rows, int):
34              raise TypeError("rows should be int")
35          if not isinstance(cols, int):
36              raise TypeError("cols should be int")
37          assert 0 < rows <= self.MAX_MAP_SIZE
```

```
54          if not isinstance(possibility, float):
55              raise TypeError("possibility should be float")
56          for row in self.cells:
```

```
62          if not isinstance(row, int):
63              raise TypeError("row should be int")
64          if not isinstance(col, int):
65              raise TypeError("col should be int")
66          assert 0 <= row < self.rows
```

```
72          if not isinstance(row, int):
73              raise TypeError("row should be int")
74          if not isinstance(col, int):
75              raise TypeError("col should be int")
76          if not isinstance(val, int):
77              raise TypeError("val should be int")
```