使用Unittest模块进行的单元测试

1.简介

单元测试是软件工程项目开发的一个重要步骤。在程序开发过程中，一旦代码累加起来，对大量的代码进行调试和修改将是一件费时费力的事情。而单元测试可以对尽可能小的模块进行测试，而不必运行整个程序。它还能帮助我们很快准确的定位到问题的位置、出现问题的模块和单元。

单元测试是对程序中最小的可测试模块--函数或类来进行测试；被测试单元要求有输出，以便单元测试模块能够捕获返回值，并且与预期值进行比较，从而得出测试通过与否。

1. 事项

a.自从 Python 2.1 版本后，PyUnit成为 Python标准库的一部分，使用时只需import unittest

b.软件测试中最基本的组成单元式测试用例，PyUnit使用TestCase类来表示测试用例，并要求所有用于执行测试的类都必须从该类继承,即unittest.TestCase

c.动态方法，只编写一个测试类来完成对整个软件模块的测试，这样对象的初始化工作可以在setUp()方法中完成，而资源的释放则可以在tearDown()方法中完成

d.PyUnit模块中定义了一个名为main的全局方法，使用它可以很方便地将一个单元测试模块变成可以直接运行的测试脚本，main()方法使用TestLoader类来搜索所有包含在该模块中的测试方法，并自动执行它们。

1. 例子

综合2.中的各个事项，一个简要的单元框架测试代码模板如下

1.#main\_runner.py

**2.from** widget **import** Widget

**3.import** unittest

4.#执行测试的类

**5.class** WidgetTestCase(unittest.TestCase):

6.    **def** setUp(self):

7.        self.widget = Widget()

8.    **def** tearDown(self):

9.        self.widget.dispose()

10.        self.widget = None

11.    **def** testSize(self):

12.        self.assertEqual(self.widget.getSize(), (40, 40))

13.    **def** testResize(self):

14.        self.widget.resize(100, 100)

15.        self.assertEqual(self.widget.getSize(), (100, 100))

16.#测试

**17.if** \_\_name\_\_ == "\_\_main\_\_":

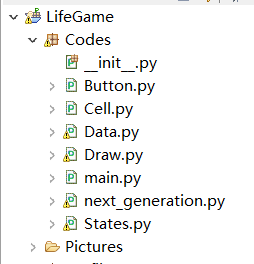
18.    unittest.main()

于是本工程的单元测试便按照这个模板来进行

1. 待测试分析

由下图见，各个文件中存在待测试方法的有Button、Cell、Draw、next\_generation、States

Data中是纯数据，main为主程序入口，不再进行测试



1. 测试用例
2. **Button类**

#-\*- coding:utf-8 –\*-

import pygame,unittest

import Codes.Data as Data

from Codes.Cell import Cell

from Codes.Button import Button

class ButtonTestCase(unittest.TestCase):

def setUp(self):

self.button = Button()

self.button.drawbutton(screen=pygame.display.set\_mode((Data.WIDTH \*Cell.size+Data.WIDTH\*2, Data.HEIGHT \* Cell.size)))

def tearDown(self):

self.button = None

def testdrawbutton1(self):

self.assertEqual(self.button.positionMove,((2+Data.WIDTH)\*Cell.size,Data.HEIGHT))

def testdrawbutton2(self):

self.assertEqual(self.button.positionStop,((2+Data.WIDTH)\*Cell.size,Data.HEIGHT\*3))

def testdrawbutton3(self):

self.assertEqual(self.button.positionReset,((2+Data.WIDTH)\*Cell.size,Data.HEIGHT\*5))

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**b.Cell类**

#-\*- coding:utf-8 –\*-

import unittest

from Codes.Cell import Cell

class CellTestCase(unittest.TestCase):

def setUp(self):

self.cell = Cell((500,1000))

def tearDown(self):

self.cell = None

#测试cell的成员属性

def testSize(self):

self.assertEqual(self.cell.size, 10)

def testImage(self):

self.assertEqual(self.cell.rect.topleft,(500,1000))

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**C.draw方法**

#-\*- coding:utf-8 –\*-

import unittest,pygame

import numpy as np

import Codes.Data as Data

from Codes.Cell import Cell

from Codes.Draw import draw

class DrawTestCase(unittest.TestCase):

def setUp(self):

self.screen = pygame.display.set\_mode((Data.WIDTH \*Cell.size+Data.WIDTH\*2, Data.HEIGHT \* Cell.size))

pygame.world=np.zeros((Data.HEIGHT,Data.WIDTH))

def tearDown(self):

self.screen = None

#测试draw方法

def testDraw(self):

draw(self.screen)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**d.next\_generation方法**

#-\*- coding:utf-8 –\*-

import unittest,pygame

import numpy as np

import Codes.Data as Data

from Codes.Cell import Cell

from Codes.next\_generation import next\_generation

class StatesTestCase(unittest.TestCase):

def setUp(self):

self.screen = pygame.display.set\_mode((Data.WIDTH \*Cell.size+Data.WIDTH\*2, Data.HEIGHT \* Cell.size))

pygame.world=np.zeros((Data.HEIGHT,Data.WIDTH))

def tearDown(self):

self.screen = None

def testNext(self):

next\_generation()

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**e.States状态方法**

#-\*- coding:utf-8 –\*-

import unittest,pygame

import Codes.Data as Data

import Codes.States as States

from Codes.Cell import Cell

class StatesTestCase(unittest.TestCase):

def setUp(self):

self.screen = pygame.display.set\_mode((Data.WIDTH \*Cell.size+Data.WIDTH\*2, Data.HEIGHT \* Cell.size))

def tearDown(self):

self.screen = None

#初始化

def testInit(self):

self.assertEqual(States.init(self.screen), 'Stop')

#停止时

def testStop(self):

self.assertEqual(States.stop(self.screen), 'Stop')

#演化时

def testMove(self):

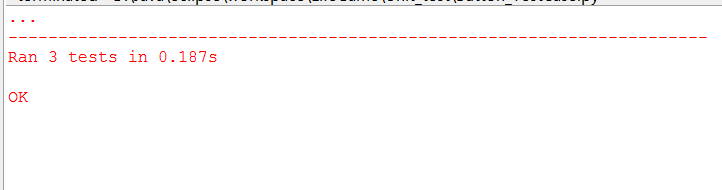
self.assertEqual(States.move(self.screen), 'Move')

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

1. 总结

单元测试最基本的原理就是比较预期结果是否与实际执行结果相同，如果相同则测试成功，否则测试失败。按照5.中的测试用例进行测试后，如果显示下图，就表明该类方法通过了测试。



补充：仅凭一个测试用例就判断方法通过测试还是不太够的，需要编写多个测试用例才能对某一功能进行比较完整的测试，这就构成了测试用例集。

由于本次项目的规模并不大，其中的方法也算不上复杂，所以没有编写太多的测试用例，就可以推断一个方法通过了测试。