第7章 Minesweeper

前面几张已经介绍过Pygame模块的基本使用方法，本章将会把重点放在实现扫雷游戏的Python语法和算法。在第6章 Connect 4中，我们介绍了矩阵（二维数组）的结构和基本算法，在这章则会接触矩阵的基本搜素算法。 除此之外，本章将会介绍一些在应用方便的基础知识，比如应用广泛的JSON文件和正则表达式

我们在Minesweeper游戏将要涉及的几个功能如下:

- 元组
- Pygame 通过鼠标的人机交互
- 文件处理
- JSON文件
- 正则表达式 Regular Expression
- 二维数组的深度优先搜索

7.1 元组 Tuple

7.1.1 Tuple 基本语法

元组是一种常用的对象类型，很多高层编程语言都有它的身影。 在C++中他叫作pair或是set，在Java中它叫做Tuple。元组的作用其实很简单：表示一组**相关的**数据。 等等，表示一组数据？为什么这句话听起来与第6章学习的list如此相似？ 是的，Tuple这种数据类型的应用场景和方式与list的确有很多相似的地方。来看看语法：

```
fruit_tuple = ('apple', 'banana', 'orange') # a tuple of fruits
fruit_list = ['apple', 'banana', 'orange']  # a list of fruits
```

Python作为一种高级语言，一个元组所包含的元素类型可以不一样, 也可以有多种创建方法。 在创建元组的时候，Python允许我们使用()来表示这组数据的类型为元组，就像[]表示一组数据为列表一样。 与此同时，Python同样允许我们不写()，同样表示元组。 然而，作者极其不推荐这种写法，因为很多时候这种创建方式的表达意义并不明确，会带来很差的可读性。 要时刻记住，写代码的首要目的是让人能看懂，其次才是让计算机运行

```
tup1 = ('apple', 50, 'banana', 16.7)     # 包含不同类型的元素
tup2 = 'apple', 50, 'banana', 16.7       # 不是使用()创建
tup3 = ()   # 创建空元组
tup4 = ('delicious',)    # 创建只有一个元素的元组，有逗号
tup5 = ('delicious')     # 创建只有一个元素的元组，无逗号

print(tup1)
print(tup2)
print(tup3)
print(tup4)
print(tup5)
```

依次打印tup1-tup4, 看看这段代码的运行结果是什么。从运行结果可以看出，tup4和tup5的打印结果 有很大的区别。如果元素只有一个元素，请务必记住要在元素后加上都好',' ，否则Python解释器会把你的 意思理解为字符串，如同tup5的运行结果：

```
('apple', 50, 'banana', 16.7)
('apple', 50, 'banana', 16.7)
()
('delicious',)
delicious
```

在了解如何创建元组后，我们需要做的就是访问元组。 访问元组元素的方式与列表类似，通过objName[index]的方式来表示元组中的元素。

```
tup = ('apple', 'banana', 'orange', 'peach') # a tuple of fruits
print(tup)
print(tup[0])   # apple
print(tup[-2])  # orange
print(tup[1:])  # ('banana', 'orange', 'peach')

for fruit in tup:
    print(fruit)
```

运行结果如下：

```
('apple', 'banana', 'orange')
apple
orange
('banana', 'orange', 'peach')
apple
banana
orange
peach
```

除了简单的创建和访问之外，元组类还支持一些列的基础计算。 Python允许我们使用'+'操作符来合并两个元组，'*'操作符来复制元组的元素：

```
tup1 = (1, 'a', 2)
tup2 = ('b', 3, 'c')
tup3 = tup1 + tup2
print(tup3) # (1, 'a', 2, 'b', 3, 'c')

tup4 = ('abc',)*3
print(tup4) # ('abc', 'abc', 'abc')
```

元组最为Python的一个类，当然要有成员函数。 Python给元组类定义了五个最常用的成员函数：

- len(tuple)
- max(tuple)
- min(tuple)
- tuple(list)

```
tup1 = (1, 2, 3)
tup2 = (1, 2, 3)
list1 = [4, 5, 6]

print(len(tup1))    # 3
print(max(tup1))    # 3
print(min(tup1))    # 1
print(tuple(list1)) # (4, 5, 6)
```

7.1.2 Tuple与List的区别

1. 可更改与不可更改

通过第6章所学，我们知道如何对列表的元素进行更新。然而，同样的更改在元组是不可行的。 什么意思呢？这就是列表和元组的第一大区别。 在列表类型中，元素的值是可以被更改的；而在元组类型中，元素的之是不可更改的。 因此，我们可以使用元组最为字典的key值，而列表不行。 为了更加直观的体现，让我们用代码尝试一下吧:
列表：

```
fruits = ['apple', 'banana', 'orange', 'peach']
print(fruits)

fruits[1] = 'watermelon' # 更新fruits列表的第2位元素
print(fruits)
```

运行结果如下：

```
['apple', 'banana', 'orange', 'peach']
['apple', 'watermelon', 'orange', 'peach']
```

元组:

```
fruits = ('apple', 'banana', 'orange', 'peach')
fruits[1] = 'watermelon' # 尝试更新fruits元组的第2位元素
```

运行结果如下:

```
Traceback (most recent call last):
  File "try.py", line 4, in <module>
      fruits[1] = 'watermelon'
      TypeError: 'tuple' object does not support item assignment

      shell returned 1
```

很明显，Python解释器不允许我们这样做。它也给出了明显的提醒: TypeError: 'tuple' object does not support item assignment.

不光是元组的元素不可更改，它的长度同样也是不可更改的。 Tuple类甚至都没有给出增加/减少元素的函数:

```
# continues from the last code block
fruits.append('kiwi')
```

运行结果如下:

```
Traceback (most recent call last):
  File "try.py", line 3, in <module>
      fruits.append('kiwi')
      AttributeError: 'tuple' object has no attribute 'append'

      shell returned 1
```

2. 复制与引用

在第6章中，我们学习过函数传递值和传递引用的区别。 同样的思想也在列表与元组的区别中有所体现。 因为元组类型的不可更改性，当你使用tuple(tup_name)函数进行类型转换是，返回值是原元组的引用。 而列表具有可更改该性，list(list_name)的返回值是被复制产生的新列表。

```
fruit_tuple = ('apple', 'banana', 'orange') # a tuple of fruits
fruit_list = ['apple', 'banana', 'orange']  # a list of fruits

# test for tuple
tuple2 = tuple(fruit_tuple)
print(tuple2 is fruit_tuple)

# test for list
list2 = list(fruit_list)
print(list2 is fruit_list)
```

运行结果如下:

```
True
False
```

3. 空间占用

由于元组类型的不可能改性，Python解释器会为其分配更小的内存占用。 因此，当元组和列表包含通函的元素时，元组占用的空间会更小一些。

```
# continues from the last code block
print(fruit_tuple.__sizeof__())
print(fruit_list.__sizeof__())
```

运行结果如下:

```
48
64
```

综上所述，我们在编写程序时，根据需求来选择使用元组或者列表。 如果无需更改内容，或者需要使内容不可更改，我们会选择使用元组。 比如传递游戏屏幕上的坐标，使用一种颜色，等等。 而当我们需要对内容进行更改是，我们只能选择列表。 比如游戏棋盘。

7.2 Pygame 通过鼠标的人机交互

在上一章，我们学习了Pygame模块的基础功能，比如设置游戏屏幕，键盘交互，绘制图形等。 在这一章，因为扫雷不能用只用键盘来玩，我们将要学习如何通过鼠标点击与计算机交互。 在讨论通过鼠标的Pygame人机交互以前，我们先来认识一下常规鼠标的构造。

从图片可以看出，普通鼠标的正面（上面）有三个按键。从左到右，把它们标号为1,2,3。

键位	代号
左键	# 1
滚轮键	# 2
右键	# 3

为什么要定义这三个常用按键的代号呢？因为我们在Pygame中就是以这三个代号称呼他们的 在扫雷游戏中，我们需要用到鼠标的左键和右键，也就是1和3。在取得鼠标信息的过程中， 需要注意到的Pygame功能有2点:

- pygame.MOUSEBUTTONUP
- pygame.mouse.get_pos()

```
# 首先定义左键和右键，使得代码可读性更高
LEFT = 1
RIGHT = 3

# 引用第6章中获取状态的代码
for event in pygame.event.get():
    pos = pygame.mouse.get_pos()
    print(pos)  # 打印鼠标点击在游戏屏幕上的坐标
    if event.button == LEFT:
        print("Left Click")
    elif event.button == RIGHT:
        print("Right Click")
```

7.3 JSON

JSON全称叫做JavaScript Object Notation，是一种面对对象的通用表示方法。 JSON这种表示方法具备什么样的特性呢？

- 轻量级
  JSON文件本质是文本文件，占用极小的储存空间，读取时占用内存很小
- 易懂
  JSON文件格式可以很好的描述面对对象的程序设计理念，更加直观的说明程序设计API
- 独立于编程语言
  尽管JSON叫做JavaScript Object Notation，但其实并非只能用于JavaScript。 大多数流行的编程语言都有自己的JSON处理模块， 比如Python的json模块，C++的Json::Value库

由于这三个特性，JSON经常被用于软件的设置文件，比如笔者常用的Visual Studio Code和Microsoft Terminal;以及客户端与服务器之间的通讯，比如C++的jsoncpprpc库。

7.3.1 JSON语法

JSON文件有至少组key-value pair组成。 其中的key指一个对象的名字，它必须是字符串。 其中的value值对象的值，它可以使任何类型。 如果value是不同类型，我们直接写出它就行。 如果value是列表，我们需要把元素写在[]中，与Python一样。 如果value是一个对象，我们需要把它写在{}中。

如果JSON包含多组key-value pair的话，我们需要用','将他们分隔开。 最后一组key-value pair后面不能有','。下面会展示一个例子，使用JSON来描述一个课程。

```
// course.json
{
    "name": "Python Programming in Games",
    "id": 123456,
```

```
    "chapters": [
        // ...
        "chapter 6": "Connect 4",
        "chapter 7": "Minesweeper"
        // .. many more chapters
    ],
    "this chapter": {
        "number": 7,
        "name": "Minesweeper",
        "subsections": [
            "7.1": "Tuple",
            "7.2": "JSON",
            "7.3": "Regular Expression",
            "7.4": "DFS in 2D Array",
            "7.5": "Minesweeper Program"
        ]
    }
}
```

7.3.2 JSON在设置文件中的应用

在这个小结，我们将阅读Windows Terminal设置文件用的一个节选片段。 然后解释这个设置文件中每组key-value pair的意义。

```
{
    // Add custom color schemes to this array.
    // To learn more about color schemes, visit https://aka.ms/terminal-color-schemes
    "scheme":
    [
        {
            "name": "Breeze",
            "black": "#12181d",
            "red": "#eb5b5b",
            "green": "#bfeea4",
            "yellow": "#fc8162",
            "blue": "#6eb7eb",
            "purple": "#9b59b6",
            "cyan": "#1abc9c",
            "white": "#eff0f1",
            "brightBlack": "#d8e3e4",
            "brightRed": "#c0392b",
            "brightGreen": "#1cdc9a",
            "brightYellow": "#fdbc4b",
            "brightBlue": "#3daee9",
            "brightPurple": "#57b4df",
            "brightCyan": "#62dd69",
            "brightWhite": "#fcfcfc",
            "background": "#31363b",
            "foreground": "#eff0f1"
        }
```

```
    ],

    // Add custom keybindings to this array.
    // To unbind a key combination from your defaults.json, set the command
to "unbound".
    // To learn more about keybindings, visit https://aka.ms/terminal-
keybindings
    "keybindings":
    [
        // Copy and paste are bound to Ctrl+Shift+C and Ctrl+Shift+V in
your defaults.json.
        // These two lines additionally bind them to Ctrl+C and Ctrl+V.
        // To learn more about selection, visit https://aka.ms/terminal-
selection
        { "command": {"action": "copy", "singleLine": false }, "keys":
"ctrl+c" },
        { "command": "paste", "keys": "ctrl+v" },

        // Press Ctrl+Shift+F to open the search box
        { "command": "find", "keys": "ctrl+shift+f" },

        // Press ctrl+shift+. to open a new pane.
        // - "split": "auto" makes this pane open in the direction that
provides the most surface area.
        // - "splitMode": "duplicate" makes the new pane use the focused
pane's profile.
        // To learn more about panes, visit https://aka.ms/terminal-panes
        { "command": "splitPane", "keys": "ctrl+shift+." },
        { "command": "closePane", "keys": "ctrl+shift+,"},
        { "command": "newTab", "keys": "ctrl+shift+a" },
        { "command": "closeTab", "keys": "ctrl+shift+z"},
        { "command": "nextTab", "keys": "ctrl+tab" }
    ]
}
```

7.3.3 编写Minesweeper游戏的JSON设置文件

7.3.4 Python中的json模块

7.4 正则表达式 Regular Expression

7.5 二维数组的深度优先搜索

7.6 Minesweeper 游戏编写