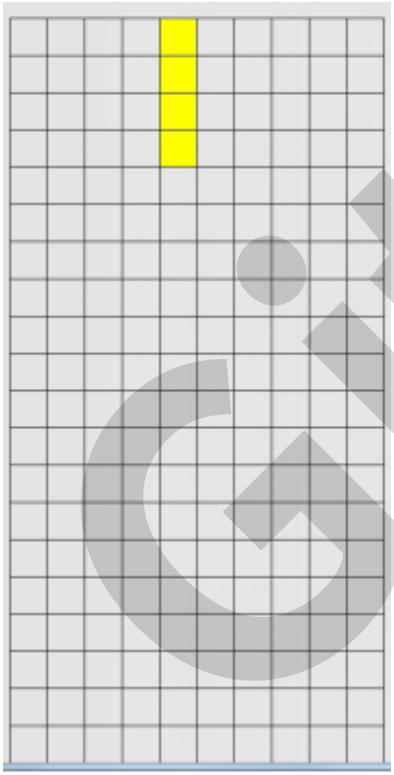
第10课：随机生成新方块

步骤目标

本文要实行的目标是：

1. 游戏界面上的当前方块触底时，生成一个新方块。
2. 生成的新方块的类型是随机的。
3. 不同类型的方块，颜色不同。

完成本步骤后，程序运行效果如图1所示。左侧子图中，当前方块是I型的。它触底后，程序生成一个新方块，这个新方块的类型是随机选择的。右侧子图中，随机选择得到T型方块。我们也可以看到，两种方块的颜色是不同的。



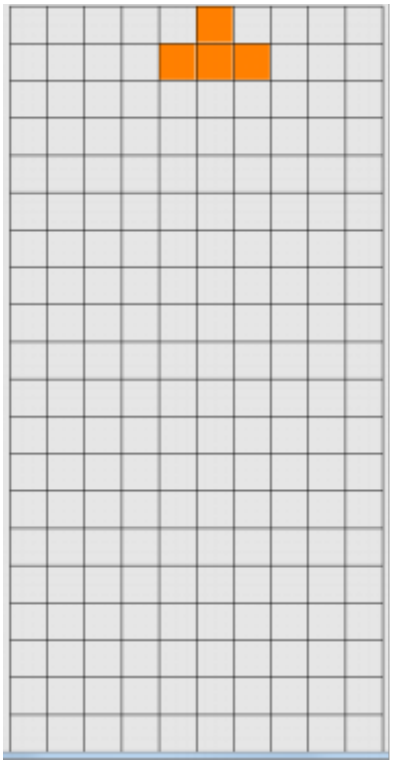


图1 当前方块触底时，程序生成一个新方块

我们要做的任务是：

1. 检测到方块触底。
2. 生成一个新方块，而且类型是随机选择的。
3. 根据方块的类型决定方块的颜色。

检测方块触底

只有向下移动方块才会造成触底，所以我们只要在向下移动之际检测方块是否触底就可以了。事实上，`move_down()` 方法内调用了 `can_move_down()` 方法，干的正是这个活。当 `can_move_down()` 方法返回 False 结论的时候，就是方块触底了。我们进一步要做的是，把“方块已经触底”记录下来。

为什么是记录“方块已经触底”，而不是直接生成新方块呢？嗯，在对象的方法内生成新对象，替换旧对象，不是推荐的做法。

为了记录“方块已经触底”，我们在 Piece 类内设置了一个属性，名字是 `is_on_bottom`。方块对象刚生成的时候，该属性值是 False，表示没有触底。一旦检测到方块触底了，就设置为

True。它的用法如代码1所示。

```
# TetrisGame/piece.py
10  class Piece():
11      def __init__(self, shape, screen):
12          self.x = 4
13          self.y = 0
14          self.shape = shape
15          self.turn_times = 0    #翻转了几次，决定显示的模样
16          self.screen = screen
17          self.is_on_bottom = False     #到达底部了吗？
…          ……    #此处与前一版本相同，故省略
44
45      def move_down(self):
46          '''方块向下移动1格。如果到达了底部，设置is_on_bottom属性为True.'''
47          if self.can_move_down():
48              self.y += 1
49          else:
50              self.is_on_bottom = True
```

代码1 `is_on_bottom` 属性记录“方块是否触底”

随机生成新方块

上一节讲到，一旦检测到方块触底，当前方块的 `is_on_bottom` 属性值就变为 True。那么，如何生成新方块呢？

答案是，在游戏主循环中查看当前方块的 `is_on_bottom` 属性值，如果发现值为 True，那么就生成新方块。

前一版本的 main.py 文件中，在 main 函数内、程序主循环之前生成了方块对象。所采用的的语句是：

```
piece = Piece('S', screen)
```

现在，我们改成代码2所示的代码。第19， 24， 25行正是修改后的代码。第19行代码把 piece 变量设为空值（ None ）。这样，在第24行的测试条件 not piece 将为真。于是，游戏开始时会生成新方块。

```
TetrisGame/main.py
10  def main():
…       ……  #与前一版本相同，省略。
19      piece = None
20
21      #游戏主循环
22      while True:
23          #生成方块对象
24          if not piece or piece.is_on_bottom:
25              piece = Piece('S', screen)
26
…           ……  #与前一版本相同，省略。
```

代码2 生成新对象

上述修改能够生成新方块，不过总是生成新的 S 型方块。

俄罗斯方块游戏进行期间，需要不断生成新方块，而且类型是随机选择的。下面让我们来实现随机选择新方块类型的功能。代码3所列代码实现了这一功能。

```
# TetrisGame/main.py
10  def main():
…       ……  #与前一版本相同，省略。
19      piece = None
20      random.seed(int(time.time()))     #产生不同的随机序列
21      #游戏主循环
22      while True:
23          #生成方块对象
24          if not piece or piece.is_on_bottom:
25              piece = Piece(random.choice(PIECE_TYPES), screen)
26
…           ……  #与前一版本相同，省略。
```

代码3 随机选择新方块类型

代码3的代码说明如下:

1. 第20行代码中，time.time() 是获得从1970年1月1日0点以来的秒数，返回值是一个浮点数。把返回的秒数值转换为整数后，用作 random 模块（文件开头要导入）的 seed 函数

的参数。seed 的中文含义是种子。调用 seed 函数的作用是设置随机数种子。这样，下次运行程序会产生与上次运行不同的随机数序列。也就是，前一次运行生成的方块序列与后一次运行生成的方块序列会不同。如果没有第20行代码，那么两次运行产生的方块序列就会相同。想通这一点，你需要知道随机数种子的作用。这里不做更多说明，你可以上网搜索资料学习了解。

2. 第25行代码中，`random.choice(PIECE_TYPES)` 是在 `PIECE_TYPES` 列表中随机选取一个元素。`PIECE_TYPES` 列表定义如下：

```
1.  PIECE_TYPES = ['S', 'Z', 'J', 'L', 'I', 'O', 'T']
```

该语句放在 settings.py 文件内。

方块类型不同，颜色不同

要做到方块类型不同，显示的颜色不同，需要：

1. 为每一种方块设定颜色值；
2. 建立类型与颜色值之间的对应关系；
3. 绘制方块的时候，根据类型选择颜色。

我们定义一个叫做 `PIECE_COLORS` 的字典来达成意图。该字典的定义放在 `settings.py` 文件内，内容如下：

```
1.  PIECE_COLORS = {
2.      'S': (0, 255, 128),
3.      'Z': (255, 128, 255),
4.      'J': (128, 0, 255),
5.      'L': (0, 0, 255),
6.      'I': (255, 255, 0),
7.      'O': (255, 0, 0),
8.      'T': (255, 128, 0)
9.  }
```

以 O 型方块为例，它的颜色值是(255, 0, 0)，也就是 RGB 值中，红色 R=255，绿色G=0，蓝色B=0，显示效果是红色。

接下来，修改绘制方块的代码，做到根据类型选择颜色。修改的地方是 Piece 类的 paint() 方

法调用的 `draw_cell()` 方法内。修改代码如下所示，其中改动后的内容为：`PIECE_COLORS[self.shape]`，其余地方不用改动。

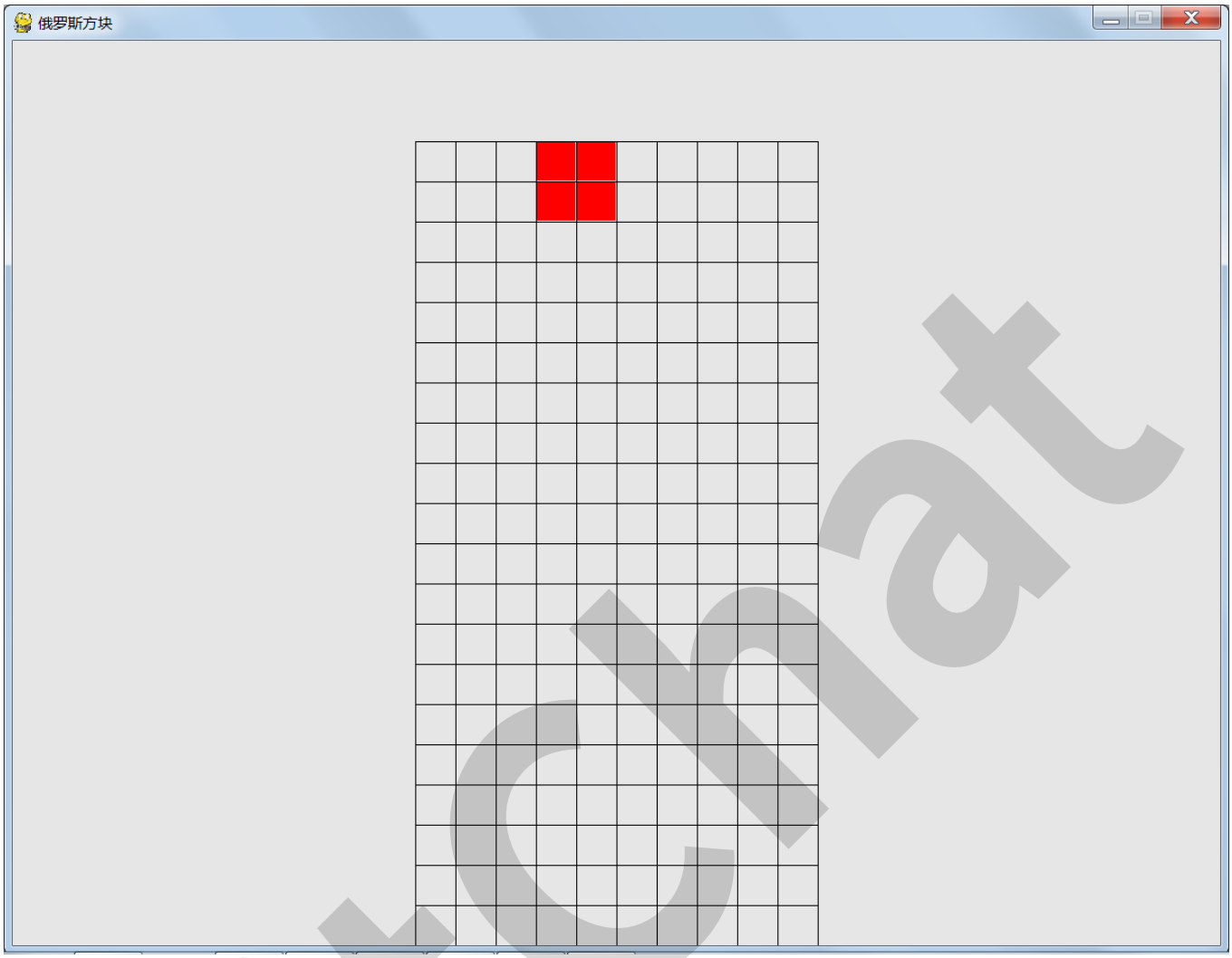
```
def draw_cell(self, x, y):
    cell_position = (x * CELL_WIDTH + GAME_AREA_LEFT + 1,
                     y * CELL_WIDTH + GAME_AREA_TOP + 1)
    cell_width_height = (CELL_WIDTH - 2, CELL_WIDTH - 2)
    cell_rect = Rect(cell_position, cell_width_height)
    pygame.draw.rect(self.screen, PIECE_COLORS[self.shape], cell_rect)
```

到此，本步骤完成了。

修订方块未居中问题

新方块生成后，我们希望它出现在顶部居中位置。但我们发现没有居中，如图2所示。图2中，O 型方块偏左了。

> **提示：**软件开发过程中，总是会遇到各种各样的问题。本节指出的新方块没有居中就是一个问题。我们要做的就是发现问题，纠正问题。不要期待一开始就编写出没问题的程序。



<

图2 新方块未居中显示

要修订问题，需要作出以下改动：

（1）piece.py 文件内，Piece 类的构造函数 `__init__` 内，把方块的定位点从（3, 0）改为（4, 0）。改动后的代码如下所示：

```
        def __init__ (self, shape, screen):
            self.x = 4
            self.y = 0
            self.shape = shape
            self.turn_times = 0    #翻转了几次，决定显示的模样
            self.screen = screen
            self.is_on_bottom = False     #到达底部了吗？
```

（2）settings.py 文件内，J 型、L 型和 T 型的方块的姿态矩阵由3x4变为3x3，删除的是空白列（即没有 O 字母的列）。请你对照下面的代码进行修订。

```
J_SHAPE_TEMPLATE = [['.O.',
                     '.O.',
                     'OO.'],
                    ['O..',
                     'OOO',
                     '...'],
                    ['OO.',
                     'O..',
                     'O..'],
                    ['OOO',
                     '..O',
                     '...']]

L_SHAPE_TEMPLATE = [['O..',
                     'O..',
                     'OO.'],
                    ['...',
                     'OOO',
                     'O..'],
                    ['OO.',
                     '.O.',
                     '.O.'],
                    ['..O',
                     'OOO',
                     '...']]

T_SHAPE_TEMPLATE = [['.O.',
                     'OOO',
                     '...'],
                    ['.O.',
                     '.OO',
                     '.O.'],
                    ['...',
                     'OOO',
                     '.O.'],
                    ['..O',
                     '.OO',
                        '..O']]
```

小结

本文实现了随机生成新方块功能。在上一方块触底的时候，将生成新的方块。我们使用 random 模块来随机生成新方块。此外，我们做到了不同类型的方块颜色不同，以及修订了新方块未居中显示的问题。

完成本文的全部功能，以及修订新方块未能居中显示的问题的代码可从 Github 上链接下载，用于比对。

- Github 地址

如果你能够了然于胸，那么你应该能够自如地修改代码。再仔细的实现步骤描述也比不上你自己的透彻理解。要加深理解，你就再多练习一次，加油！