第12课：垒墙

步骤目标

本文要实现的目标是，方块落到游戏区域底部后，垒成一堵墙。前一文所完成的代码版本在运行的时候，方块落到底部便消失了，这看起来有些怪异。我们暂时不考虑俄罗斯方块游戏的消行功能，以后会实现消行功能。

鉴于此，本文将完成以下两个任务：

1. 给出记住墙体的做法；
2. 实现把落到底部方块垒成墙体功能。如图1所示，游戏区域底部出现了一堵墙。

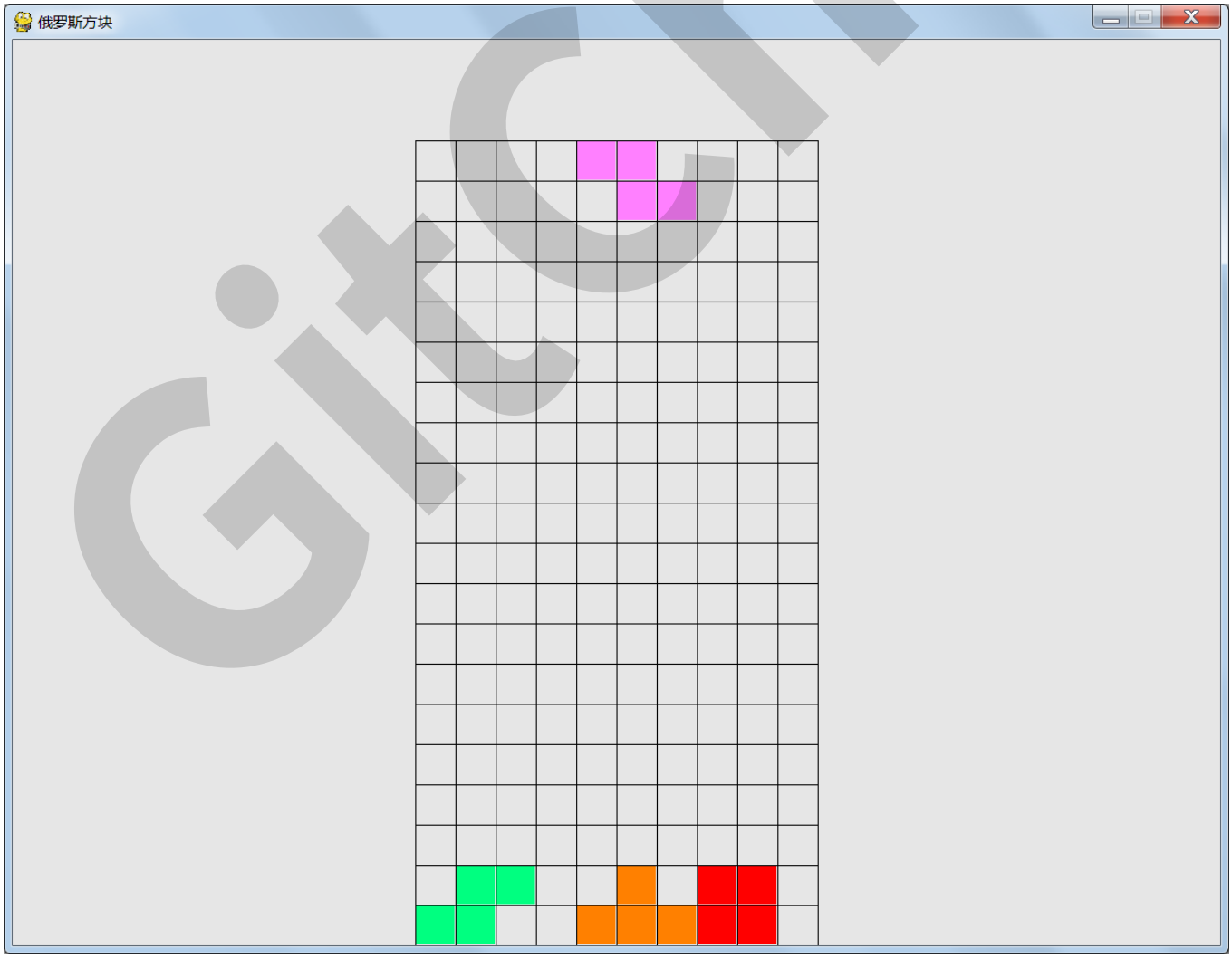

图1 方块落到底部，垒成一堵墙

记住墙体的做法

墙体的表示

游戏区域是一个 20X10 的网格矩阵。通过记住哪些单元格内“有砖块”，就能记住墙体。如何来记住单元格 “有砖块”呢？我们的做法是：

1. 构造一个 20X10 的二维字符矩阵 wall。矩阵元素 `wall[r][c]` 对应游戏区域第 r+1 行第 c+1 列单元格。
2. 如果单元格内没有砖块，则单元格的值是 `-`，即中划线字符。
3. 如果单元格内有砖块，则单元格的值是代表方块类型的字母，即以下字母中的一种：

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

假若某个单元格的值是 S ，意味着 S 型方块落在该位置后成为墙体的一部分。

图2是墙体矩阵示意图。左侧子图是墙体。顶部的Z型方块还没有落到底部，不成为墙体的组成部分。底部两行中，第20行第1，2列的元素值是’S’ ，意味着这里落有一个S型方块。第20行第5列的元素值是’T’ ，说明这里垒了一个T型方块。

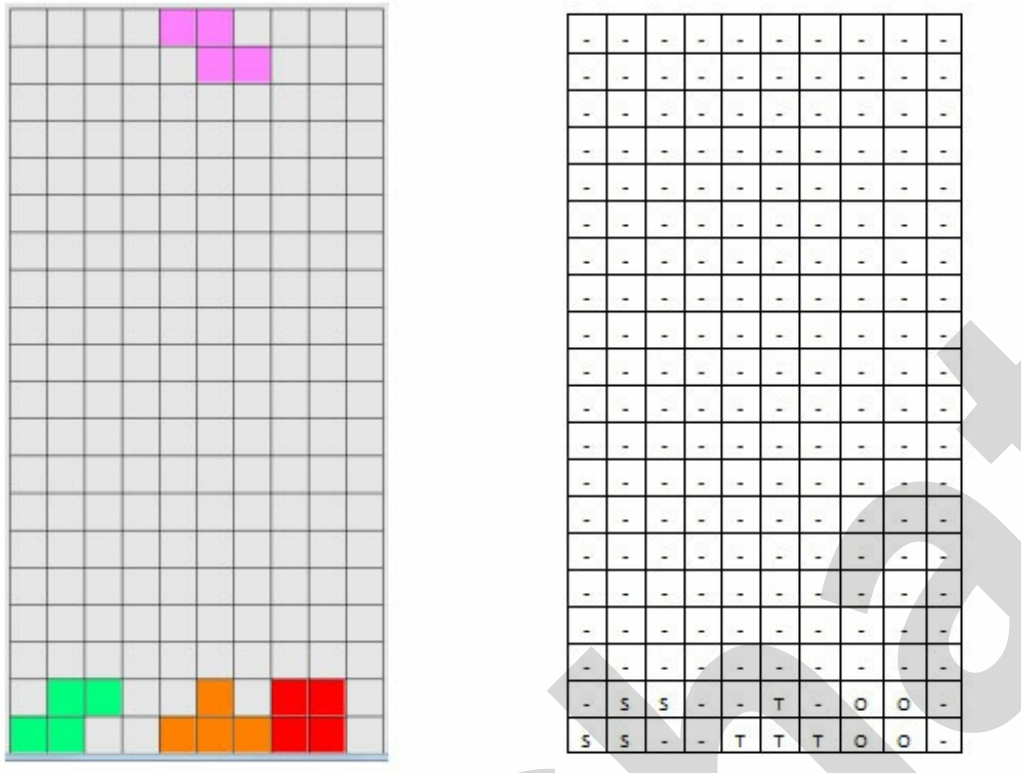

图2 墙体矩阵（左侧子图是墙体示例，右侧子图是对应的墙体矩阵，重点看底部两行的对应。）

初始化游戏区域

我们为墙体创建一个类，名字是 GameWall。这个类的构造函数定义如代码1所示。

```
# TetrisGame/gamewall.py
8   class GameWall():
9       '''游戏区墙体类。记住落到底部的方块组成的“墙”。'''
10      def __init__(self, screen):
11          '''游戏开始时，游戏区20*10个格子被'-'符号填充。“墙”是空的。'''
12          self.screen = screen
13          self.area = [ ]
14          line = [WALL_BLANK_LABEL] * COLUMN_NUM
15          for i in range(LINE_NUM):
16              self.area.append(line[:])
17
18      def print(self):
19          '''打印20*10的二维矩阵self.area的元素值。用于调试。'''
20          print(len(self.area), "rows", len(self.area[0]), "colums")
21          for line in self.area:
22              print(line)
```

代码1 GameWall 类

GameWall 类的 area 属性存储墙体矩阵。第15、16行把所有矩阵元素初始化为 `-`。这里，line 是10个 `-` 元素组成的列表。值得强调的是，第16行中 `line[:]` 里头的 `[:]` 不能少！

我们来看看 line 列表是怎么生成的。这是第 14 行干的活。其中，`WALL_BLANK_LABEL` 是在 `settings.py` 文件内定义的常量，值为表示墙体为空的字符 `-`。而 `LINE_NUM`，`COLUMN_NUM` 常量的值是20、10，即行数和列数。第14行把 line 赋值为 `-`*10，我们知道结果是10个 `-` 组成的列表。

实现垒墙功能

垒墙的时机

什么时候垒墙呢？答案是方块触底的时候。记得吗，我们在程序主循环内检测了方块触底与否。当前方块的 `is_on_bottom` 属性为 True 时，表明方块触底了。此时，要调用垒墙方法。相关代码如代码2所示。

```
# TetrisGame/main.py
14  def main():
…       ……  #与前一版本相同，故省略。
25      random.seed(int(time.time()))     #产生不同的随机序列
26      piece = Piece(random.choice(PIECE_TYPES), screen)
27      game_wall = GameWall(screen)
28      #游戏主循环
29      while True:
30          #方块触底的话
31          if piece.is_on_bottom:
32              game_wall.add_to_wall(piece)
33              piece = Piece(random.choice(PIECE_TYPES), screen)
…           ……  #与前一版本相同，故省略。
```

代码2 方块触底时调用垒墙方法

代码2中的各行代码说明如下：

1. 第27行代码生成了墙体对象 `game_wall`。它内部的墙体矩阵被初始化成空（也即没有任

何砖块）。
2. 第31行代码检测当前方块是否触底了。
3. 触底的话，第32行调用 `game_wall` 对象的垒墙方法 `add_to_wall`，当前方块 piece 作为参数。这行代码把落到底部的方块加入墙体。

垒墙函数 `add_to_wall`

垒墙方法 `add_to_wall` 要做的事情就是，把落到游戏区底部的方块砌到墙体矩阵内。也就是，把组成方块的四个小块砌到墙体矩阵内。

垒墙方法 `add_to_wall` 是 gamewall.py 文件内定义的 GameWall 类的方法。该方法定义如代码3所示。

```
# TetrisGame/gamewall.py
31  def add_to_wall(self, piece):
32      '''把方块piece砌到墙体内'''
33      shape_turn = PIECES[piece.shape][piece.turn_times]
34      for r in range(len(shape_turn)):
35          for c in range(len(shape_turn[0])):
36              if shape_turn[r][c] == 'O':
37                  self.set_cell((piece.x + c, piece.y + r),
piece.shape)
```

代码3 GameWall 类的 `add_to_wall` 方法（左侧数字是文件内代码行号）

它以方块对象作为参数。该方块已经落到游戏区域底部。下面对 `add_to_wall` 方法内部的代码稍作解释。

1. 第33行是得到方块的姿态矩阵。
2. 第34、35行是遍历姿态矩阵内的元素。
3. 第36、37行的作用是，对于每一个方块内的小块，调用 `set_cell` 方法把它砌到墙体内。注意，第一个参数是一个元组，格式是：`（列号，行号）`，对应于：`（横坐标，纵坐标）`。

GameWall 类的 `set_cell` 方法的定义如下。该方法的 position 参数指出墙体矩阵内的坐标；`shape_label` 参数是方块的类型记号，比如 S、L、J 等字母。

```
    def set_cell(self, position, shape_label):
        '''把第r行c列的格子打上方块记号（如S, L, ...），因为该格子被此方块占据。
```

```
    '''
            c, r = position
            self.area[r][c] = shape_label
```

绘制墙体

绘制墙体的时机

墙体可以看作游戏区域的一部分，尽管墙体会变化。因此，在 `draw_game_area()` 内调用绘制墙体的方法为好。

绘制墙体的方法

绘制墙体的方法是 GameWall 类的 paint() 方法。它的实现思路是扫描20X10墙体矩阵，遇到“有砖块”的单元格，就据砖块类型对应的颜色填充该单元格。

绘制墙体与绘制方块有相似之处：两者都调用“填充单元格”的函数 `draw_cell`。那么，是不是要让 GameWall 类的 paint() 方法调用 Piece 类内 `draw_cell()` 方法呢？答案是：不。我们不应当让 GameWall 类依赖 Piece 类——墙体没必要拥有关于方块的知识。

为避免重复书写代码，我们把“填充单元格”的代码提炼成工具类 GameDisplay 的方法。GameDispaly 这个工具类的定义如代码4所示。它的第一个方法是 `draw_cell()`，完成的是填充单元格的功能。它的第二个方法 `draw_game_area()` 是从 main.py 文件迁移过来的（这样使得 main.py 更加简洁），完成的是绘制网格线和墙体的功能。第29行代码调用绘制墙体的方法 `draw_wall()`。

```
# TetrisGame/gamedisplay.py
5   from settings import *
6   import pygame
7
8
9   class GameDisplay():
10      @staticmethod
11      def draw_cell(screen, x, y, color):
12          '''第y行x列的格子里填充color颜色。一种方块对应一种颜色。'''
13          cell_position = (x * CELL_WIDTH + GAME_AREA_LEFT + 1,
14                           y * CELL_WIDTH + GAME_AREA_TOP + 1)
15          cell_width_height = (CELL_WIDTH - 2, CELL_WIDTH - 2)
16          cell_rect = pygame.Rect(cell_position, cell_width_height)
17          pygame.draw.rect(screen, color, cell_rect)
```

```
18
19      @staticmethod
20      def draw_game_area(screen, game_wall):
21          '''绘制游戏区域'''
22          for r in range(21):
23              pygame.draw.line(screen, EDGE_COLOR, (GAME_AREA_LEFT, G
AME_AREA_TOP + r * CELL_WIDTH),
24                               (GAME_AREA_LEFT + GAME_AREA_WIDTH, GAME
_AREA_TOP + r * CELL_WIDTH))
25          for c in range(11):
26              pygame.draw.line(screen, EDGE_COLOR, (GAME_AREA_LEFT +
c * CELL_WIDTH, GAME_AREA_TOP),
27                               (GAME_AREA_LEFT + c * CELL_WIDTH, GAME_
AREA_TOP + GAME_AREA_HEIGHT))
28
29          GameDisplay.draw_wall(game_wall)
```

代码4 GameDispaly 工具类

GameWall类的draw_wall()方法定义如下。这些代码位于gamedisplay.py的第31~37行。

```
@staticmethod
def draw_wall(game_wall):
    '''绘制墙体'''
    for r in range(LINE_NUM):
        for c in range(COLUMN_NUM):
            if game_wall.area[r][c] != WALL_BLANK_LABEL:
                GameDisplay.draw_cell(game_wall.screen, c, r,
PIECE_COLORS[game_wall.area[r][c]])
```

> 知识点：类定义内部，方法名之前的 `@staticmethod` 表明这一方法是一个静态方法。所谓静态方法，是指没有访问对象的属性的方法。静态方法本质上跟函数类似。只不过调用的写法是“类名.方法()”，比如 `GameDisplay.draw_cell()`，或 `GameDisplay.draw_wall()`。

有了 GameDisplay 类的 `draw_cell()` 方法，Piece 类的 `draw_cell()` 方法变更为：

```
    def draw_cell(self, x, y):
        GameDisplay.draw_cell(self.screen, x, y, PIECE_COLORS[self.shap
e])
```

小结

本文实现了垒墙功能。当下落的方块触底的时候，方块嵌入游戏区域的墙体中。在游戏主循环中，我们检测当前方块是否触底，触底的话将把当前方块加入墙体，然后生成新方块。

我们用 20x10 的墙体矩阵来表示墙体，元素值是字符。我们定义了 GameWall 类来存储和管理墙体。

绘制墙体就是对不为空的单元格进行绘制，即填充颜色。我们定义了 GameDisplay 类。这个类包含绘制单元格，绘制墙体和绘制游戏区域的方法。绘制游戏区域的方法调用绘制墙体的方法。而程序主循环内调用绘制游戏区域的方法。这意味着，每执行主循环一次，墙体就会被刷新一次。

完成垒墙功能的完整代码可从以下链接下载，以供参考。

- Github

当前程序版本中，方块会重叠，如图3所示。这一问题将在下一篇得以解决。

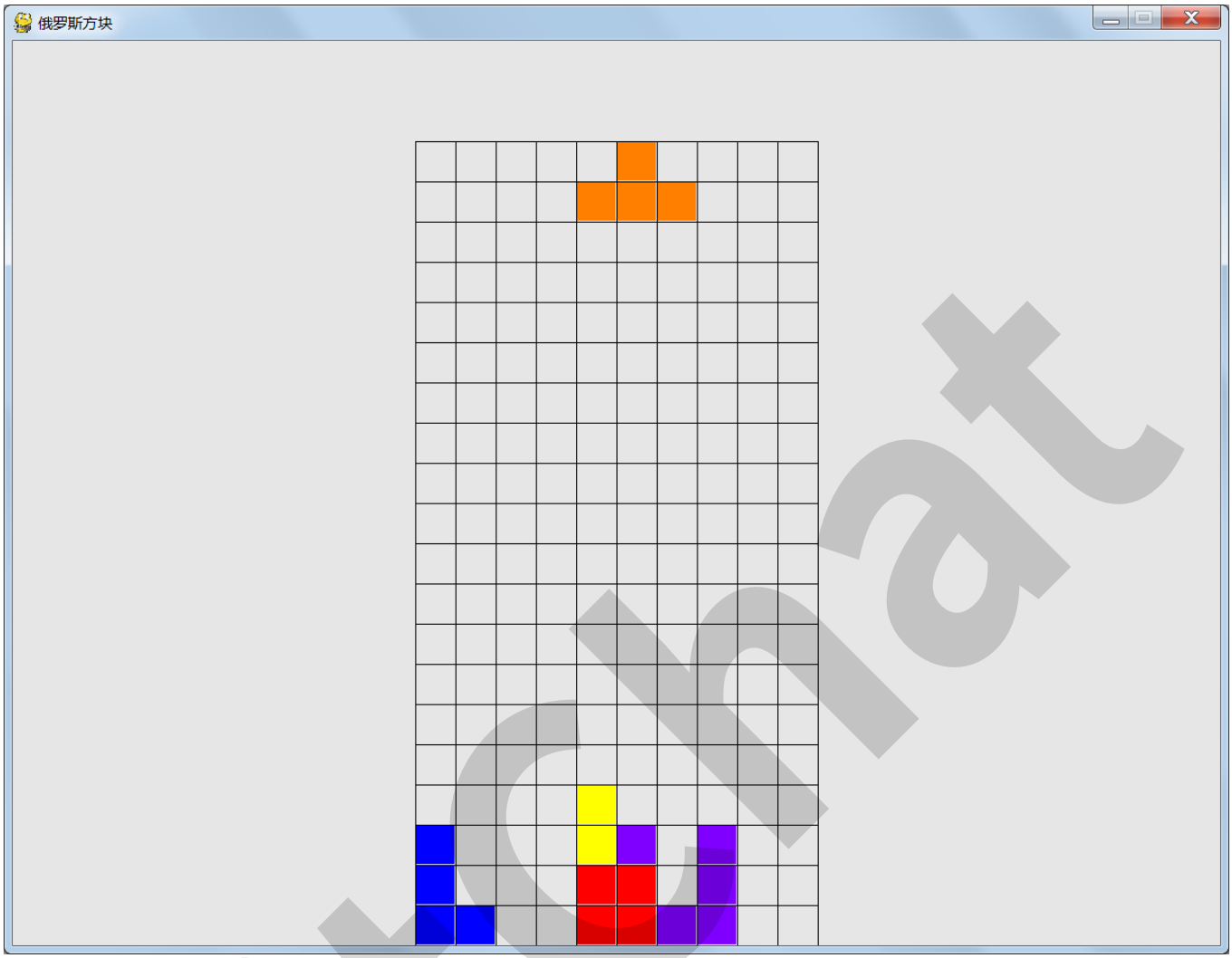

图3 方块重叠在底部几行