第18课：游戏结束

步骤目标

本文将实现以下功能：

1. 判别游戏是否结束。
2. 执行游戏结束的动作。
3. 显示“游戏结束”图片和“按 s 字母键开始”图片。如图1所示。
4. 重新开始游戏。

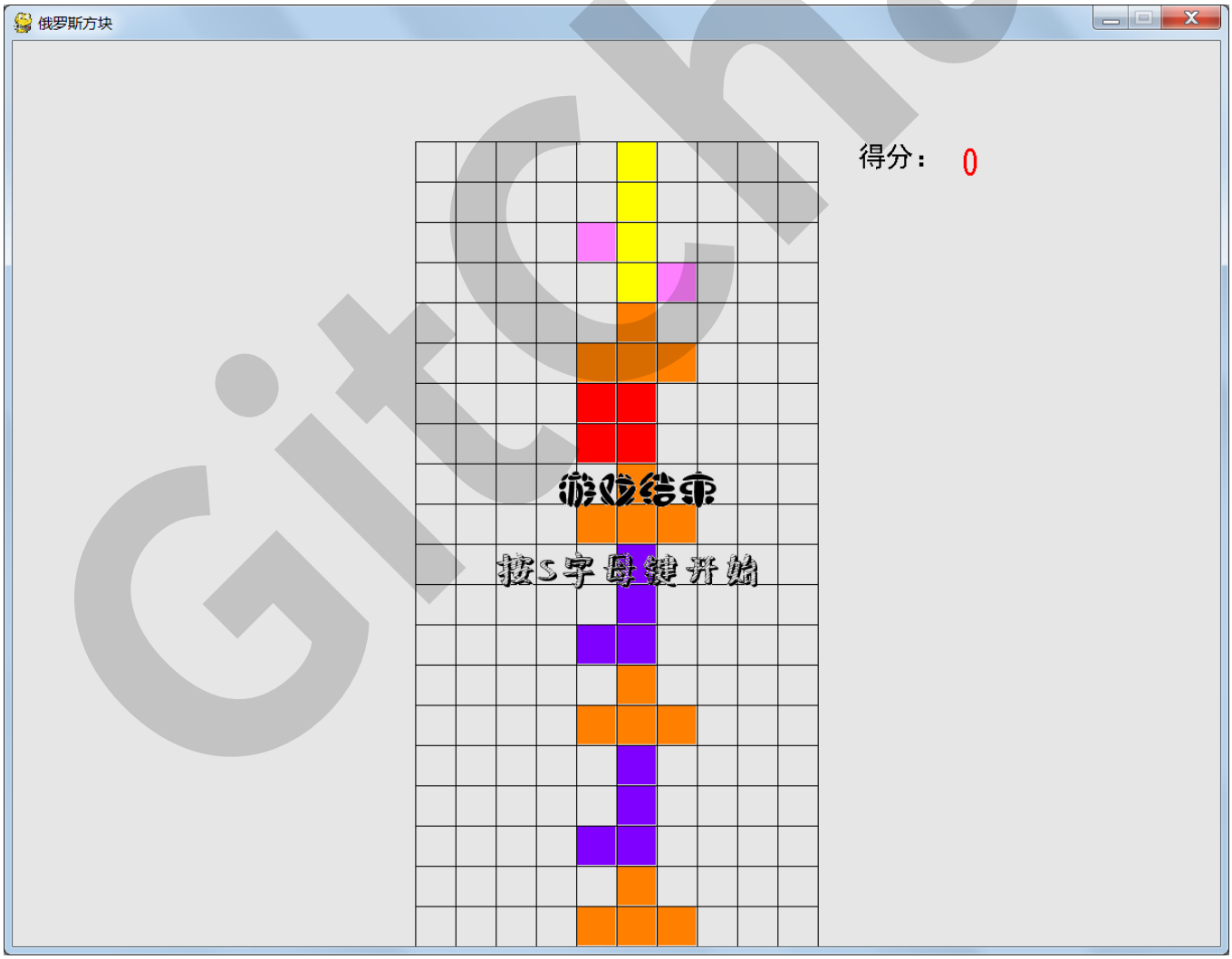

图1 游戏结束

判别游戏结束

我们要在方块触底的时候判别游戏是否结束。方块触底之际，它同时碰到游戏区域的顶部，那么意味着游戏结束。

第二个判别游戏结束的时机是新方块刚在顶部出现的时候。如果这一时候新方块撞到了墙体，则视为游戏结束。对于第二个时机，我们留到下一篇来处理。否则，本文拥有太多内容。

前面所开发的程序代码中，触底的处理流程是：

1. 方块对象调用 `can_move_down` 方法检测能否向下移动。如果不能，则视为触底，把方块对象的 `is_on_bottom` 属性设为 True。
2. 程序主循环内，检测到 `is_on_bottom` 属性值为 True，则执行把方块砌入墙体、消行计分和生成新方块三项任务，如以下代码所示。

```
#游戏主循环
while True:
    #方块触底的话
    if game_state.piece and game_state.piece.is_on_bottom:
        game_state.wall.add_to_wall(game_state.piece)
        game_state.add_score(game_state.wall.eliminate_lines())
        game_state.piece = Piece(random.choice(PIECE_TYPES), screen, game_state.wall)
    ……
```

我们能猜到，判别游戏结束成为第四项任务。

先不急着在主循环内添加完成第四项任务的代码。我们先把完成上述三项任务的代码移到 GameState 类，用 touch_bottom() 方法来封装它们。这样，主循环内的代码变成：

```
#游戏主循环
while True:
    #方块触底的话
    if game_state.piece and game_state.piece.is_on_bottom:
        game_state.touch_bottom()
    ......
```

GameState 类的 `touch_bottom()` 方法的初始版本是：

```
def touch_bottom(self):
        self.wall.add_to_wall(self.piece)
        self.add_score(self.wall.eliminate_lines())
        self.piece = Piece(random.choice(PIECE_TYPES), self.screen, sel
f.wall)
```

下面，我们在 `touch_bottom()` 方法内扩充代码，变成如代码1所示的代码。`touch_bottom` 方法在 gamestate.py 文件内定义。

```
# Tetris/gamestate.py
46  def touch_bottom(self):
47      self.wall.add_to_wall(self.piece)
48      self.add_score(self.wall.eliminate_lines())
49      # print(game_state.game_score)
50      for c in range(COLUMN_NUM):
51          if self.wall.is_wall(0, c):
52              # game_area.draw_gameover()    #在这里绘制文字是不起作用的。
必须放到填充窗口背景之后。
53              self.stopped = True
54              break
55      if not self.stopped:
56          self.piece = Piece(random.choice(PIECE_TYPES), self.screen,
self.wall)
57      else:
58          self.stop_timer()
```

代码1 触底处理方法 `touch_bottom`（左侧数字是文件内的代码行号）

对于代码1中的代码，简要说明如下：

1. 第50~54行，检测了墙体是否碰到游戏区域顶部。这是通过检查顶部第一行有没有砖块来得出结论的。如果是，那么令 stopped 属性为 True，表示游戏结束，进入停止状态。
2. 第56行，是游戏未进入停止状态的情形下生成新方块。也就是上面讲到的第三项任务。
3. 第58行，正是游戏结束要采取的措施，即取消定时器。这样就不会发出定时事件。

GameState 类的 `stop_timer` 方法的定义如下所示。

```
def stop_timer(self):
    pygame.time.set_timer(pygame.USEREVENT, 0)   #传入0表示清除定时器
```

GameState 类的 `touch_bottom` 方法完成了两件与游戏结束相关的事。一是判别游戏结束与否。二是游戏结束后的处理，包括令 stopped 属性为 True 和取消定时器。

显示游戏结束

显示的效果参见图1。游戏区域中央显示了两副图片，即“游戏结束”图片和“按 s 字母键开始”图片。这两幅图片是利用在线字体转换器网站编辑而成的。你也可以访问文末给出的链接，下载我制作好的图片。

显示图片的做法在前面两篇文章中已经描述过。我们要做的是：

1. 把“游戏结束”图片文件放到 images 子文件夹内。“按 s 字母键开始”图片文件已经有了。

2. 在 gameresource.py 文件定义的 GameResource 类内，为“游戏结束”图片定义一个属性，名叫 `gameover_img`。在构造函数内，把 `gameover_img` 设值为 None。

3. GameResource 类内，定义一个 `load_gameover_img` 方法，代码如下：

```
1.  def load_gameover_img(self):
2.      if not self.gameover_img:
3.          self.gameover_img = pygame.image.load(self.img_path + "game-ove
    r.png").convert_alpha()
4.
5.          return self.gameover_img
```

4. GameDisplay 类内，`draw_game_area` 方法修改为代码2所列代码。第32、33行是新增代码，作用是绘制“游戏结束”图片。

```
1.  # Tetris/gamedisplay.py
2.  19  @staticmethod
3.  20  def draw_game_area(screen, game_state, game_resource):
4.  21      '''绘制游戏区域'''
5.  22      for r in range(21):
6.  23          pygame.draw.line(screen, EDGE_COLOR, (GAME_AREA_LEFT, GAME_
    AREA_TOP + r * CELL_WIDTH),
7.  24                           (GAME_AREA_LEFT + GAME_AREA_WIDTH, GAME_ARE
    A_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_state.wall)
30      GameDisplay.draw_score(screen, game_state.game_score)
31      if game_state.stopped:
32          if game_state.session_count > 0:
33              GameDisplay.draw_game_over(screen, game_resource)
34          GameDisplay.draw_start_prompt(screen, game_resource)
35      if game_state.paused:
36          GameDisplay.draw_pause_prompt(screen, game_resource)
```

代码2 `draw_game_area` 方法新增绘制“游戏结束”图片

第32行中的 `game_state.session_count` 是什么呢？是用来记住游戏玩了几轮的属性。游戏程序刚启动的时候，`session_count` 属性值设为0。这是 GameState 类的构造函数内完成的。接下来，每一次开始新游戏，`session_count` 增1。这是在 GameState 类的 `start_game()` 方法内完成的。

为什么要用到 `session_count` 属性呢？原因在于，程序需要区分玩家是不是玩第一轮。玩第一轮的话，开始游戏之前不能显示“游戏结束”，尽管游戏处于停止状态，stopped 属性为 True。而玩以后轮次的话，要显示“游戏结束”。第一轮开始前，`session_count` 属性值为0。以后每一轮游戏结束，`session_count` 属性值大于0。这样程序能区分开玩家是不是玩第一轮。

开始新游戏

一轮游戏结束后，玩家按下 s 字母键开始新一轮游戏。这与开始第一轮游戏有点区别：要把上一轮遗留的痕迹清除掉。要清除的痕迹包括：

- 定时器间隔时间。变为初始值1000ms。
- stopped 属性，变为 False。
- 墙体要清空。
- paused 属性，变为 False。

- 游戏得分。变为0。

此外，记录玩了几轮的 `session_count` 要增1。

GameState 类的 `start_game` 方法完成以上功能，见代码3。第35~38行代码是本步骤新增的代码。

```
# Tetris/gamestate.py
30  def start_game(self):
31      self.stopped = False
32      self.set_timer(TIMER_INTERVAL)
33      self.timer_interval = TIMER_INTERVAL
34      self.piece = Piece(random.choice(PIECE_TYPES), self.screen, self.wall)
35      self.session_count += 1
36      self.wall.clear()
37      self.game_score = 0
38      self.paused = False
```

代码3 `start_game` 方法

第36行代码中调用的 clear() 方法的作用是把墙体矩阵清空，代码如下。

```
def clear(self):
    for r in range(LINE_NUM):
        for c in range(COLUMN_NUM):
            self.area[r][c] = WALL_BLANK_LABEL
```

小结

本文实现了方块触顶时结束游戏的功能。做法是：

1. 在方块触底之际检测方块有没有触顶。如果顶部第一行有墙，则视为触顶。
2. 在 GameState 类内添加 stopped 属性，值为 True 表明游戏结束，值为 False 表明游戏未结束。触顶的时候，stopped 属性值设为 True。
3. 在 `draw_game_area` 函数内，检测 stopped 属性值，若为 True，则显示“游戏结束”。要注意一个细节，就是玩第一轮之前，我们不能显示“游戏结束”。为此，在 GameState 类内添加 `session_count` 属性，初值为0，以后每玩一轮就增1。`session_count` 属性值

大于0，且 stopped 属性值为 True 时，才显示“游戏结束”。

4.游戏结束后，玩家重新开始一轮时，程序要把上一轮遗留的痕迹清除掉。

完成本步骤全部功能的代码可从以下链接浏览或下载。

- Github

事实上，我们还留了一点尾巴。这就是我们没有考虑新方块撞墙（指方块与墙体重叠，而触底是指方块下面挨着墙体）的情形。这也应该视为游戏结束。我们留到下一篇来解决这一问题，以及向左、向右移动或翻转时撞墙的问题。