**基于pygame的RPG游戏-魔塔（MagicTower）**

1. **项目环境(魔塔MagicTower)**

开发工具：Pycharm

Python版本：3.7.4

相关模块：[pygame](https://so.csdn.net/so/search?q=pygame&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_54556126/article/details/_blank)模块；以及一些python自带的模块。

环境搭建

安装Python并添加到环境变量，pip安装需要的相关模块即可。

**2. 项目介绍**

《魔塔》是一种策略类的固定数值[RPG](https://baike.baidu.com/item/RPG/24462?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AD%94%E5%A1%94/_blank)游戏。游戏需要动很多脑筋，任何一个轻率的选择都可能导致游戏的失败。魔塔游戏虽不大，但是制作精美，道具很多，而且难度不低，对智商是一次艰巨的考验。

虽然魔塔的界面很像是一般的地牢游戏，貌似随便的打打杀杀就可以过关，但事实上玩这个游戏需要动很多脑筋，任何一个轻率的选择都可能导致游戏的失败，该游戏有属性攻击、防御、生命、金币、经验。对怪物的伤害次数计算公式，是敌人的生命/（自己的攻击-敌人的防御）；而伤害的数目计算是怪物伤害次数（敌人的攻击-自己的防御）。

原版游戏魔塔有五十层，由于时间以及技术原因仅仅实现了前十层。

**3. 模块介绍**

**3.1 初始界面**

首先，我们打开原版游戏的开始界面，发现是这样的：



**图1 初始界面**

具体而言，我们的思路是先定义一个按钮类，来模拟原始游戏中的“开始游戏”，“游戏说明”和“离开游戏”这三个按键的功能：

代码：

'''按钮类'''

class Button(pygame.sprite.Sprite):

def \_\_init\_\_(self, text, fontpath, fontsize, position, color\_selected=(255, 0, 0), color\_default=(255, 255, 255)):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.text = text

self.color\_selected = color\_selected

self.color\_default = color\_default

self.font = pygame.font.Font(fontpath, fontsize)

self.font\_render = self.font.render(text, True, color\_default)

self.rect = self.font\_render.get\_rect()

self.rect.center = position

'''更新函数: 不断地更新检测鼠标是否在按钮上'''

def update(self):

mouse\_pos = pygame.mouse.get\_pos()

if self.rect.collidepoint(mouse\_pos):

self.font\_render = self.font.render(self.text, True, self.color\_selected)

else:

self.font\_render = self.font.render(self.text, True, self.color\_default)

'''绑定到屏幕上'''

def draw(self, screen):

screen.blit(self.font\_render, self.rect)

主要的原理就是不断检测鼠标是否在对应的按钮区域，如果在，则对按钮的颜色做出改变(这里是变成红色)，否则按钮使用默认的颜色(这里是白色)，以此来向玩家表明这是可点击的按钮。

然后，我们来实例化它三次来添加这三个按钮到游戏的开始界面中，其他额外的代码主要是显示游戏的标题等信息：

'''游戏开始界面'''

class StartGameInterface():

def \_\_init\_\_(self, cfg):

self.cfg = cfg

self.play\_btn = Button('开始游戏', cfg.FONTPATH\_CN, 50, (cfg.SCREENSIZE[0]//2, cfg.SCREENSIZE[1] - 400))

self.intro\_btn = Button('游戏说明', cfg.FONTPATH\_CN, 50, (cfg.SCREENSIZE[0]//2, cfg.SCREENSIZE[1] - 300))

self.quit\_btn = Button('离开游戏', cfg.FONTPATH\_CN, 50, (cfg.SCREENSIZE[0]//2, cfg.SCREENSIZE[1] - 200))

'''外部调用'''

def run(self, screen):

# 魔塔

font = pygame.font.Font(self.cfg.FONTPATH\_CN, 80)

font\_render\_cn = font.render('魔塔', True, (255, 255, 255))

rect\_cn = font\_render\_cn.get\_rect()

rect\_cn.center = self.cfg.SCREENSIZE[0] // 2, 200

# Magic Tower

font = pygame.font.Font(self.cfg.FONTPATH\_EN, 80)

font\_render\_en = font.render('Magic Tower', True, (255, 255, 255))

rect\_en = font\_render\_en.get\_rect()

rect\_en.center = self.cfg.SCREENSIZE[0] // 2, 350

# (Ver 1.12)

font = pygame.font.Font(self.cfg.FONTPATH\_CN, 40)

font\_render\_version = font.render('(Ver 1.12)', True, (255, 255, 255))

rect\_ver = font\_render\_version.get\_rect()

rect\_ver.center = self.cfg.SCREENSIZE[0] // 2, 400

clock = pygame.time.Clock()

while True:

screen.fill((0, 0, 0))

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit(0)

elif event.type == pygame.MOUSEBUTTONDOWN:

if event.button == 1:

mouse\_pos = pygame.mouse.get\_pos()

if self.play\_btn.rect.collidepoint(mouse\_pos):

return True

elif self.quit\_btn.rect.collidepoint(mouse\_pos):

pygame.quit()

sys.exit(0)

elif self.intro\_btn.rect.collidepoint(mouse\_pos):

self.showgameintro(screen)

for btn in [self.intro\_btn, self.play\_btn, self.quit\_btn]:

btn.update()

btn.draw(screen)

for fr, rect in zip([font\_render\_cn, font\_render\_en, font\_render\_version], [rect\_cn, rect\_en, rect\_ver]):

screen.blit(fr, rect)

pygame.display.flip()

clock.tick(self.cfg.FPS)

'''显示游戏简介'''

def showgameintro(self, screen):

font = pygame.font.Font(self.cfg.FONTPATH\_CN, 20)

font\_renders = [

font.render('魔塔小游戏.', True, (255, 255, 255)),

font.render('游戏背景故事为公主被大魔王抓走, 需要勇士前往魔塔将其救出.', True, (255, 255, 255)),

]

rects = [fr.get\_rect() for fr in font\_renders]

for idx, rect in enumerate(rects):

rect.center = self.cfg.SCREENSIZE[0] // 2, 50 \* idx + 100

clock = pygame.time.Clock()

while True:

screen.fill((0, 0, 0))

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit(0)

elif event.type == pygame.MOUSEBUTTONDOWN:

if event.button == 1:

mouse\_pos = pygame.mouse.get\_pos()

if self.play\_btn.rect.collidepoint(mouse\_pos):

return True

elif self.quit\_btn.rect.collidepoint(mouse\_pos):

pygame.quit()

sys.exit(0)

elif self.intro\_btn.rect.collidepoint(mouse\_pos):

return

for btn in [self.intro\_btn, self.play\_btn, self.quit\_btn]:

btn.update()

btn.draw(screen)

for fr, rect in zip(font\_renders, rects):

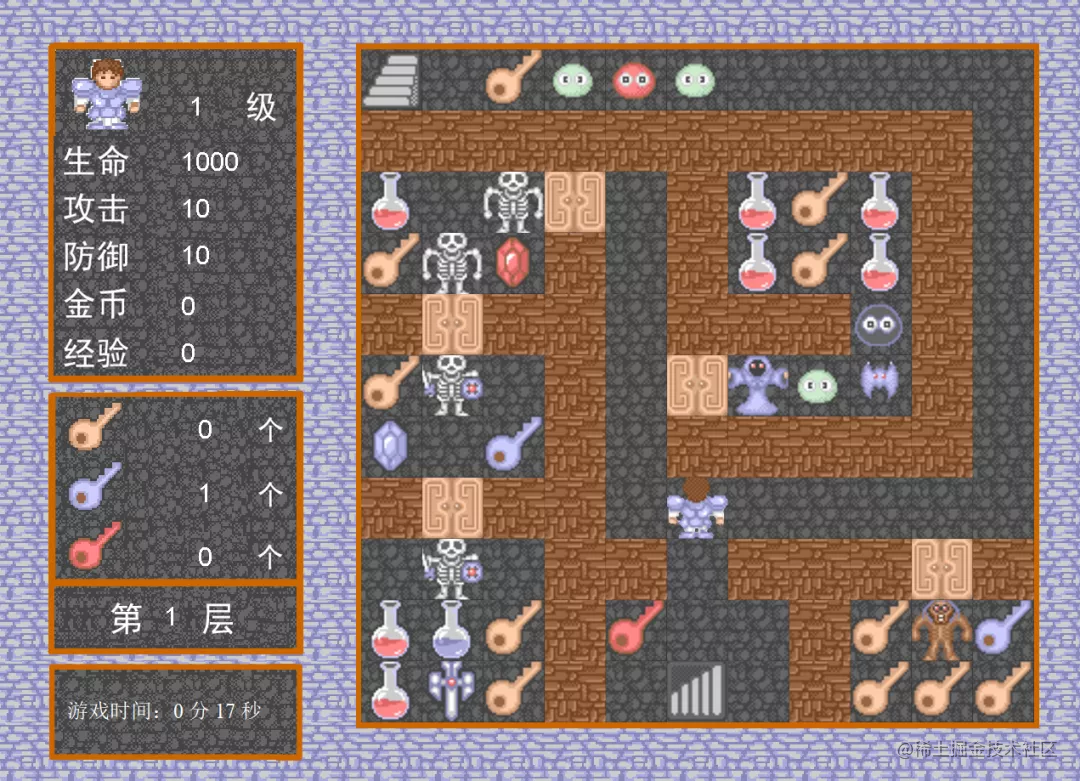
screen.blit(fr, rect)

pygame.display.flip()

clock.tick(self.cfg.FPS)

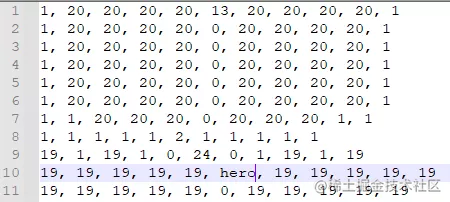
**3.2 游戏界面**

**3.2.1 游戏元素**



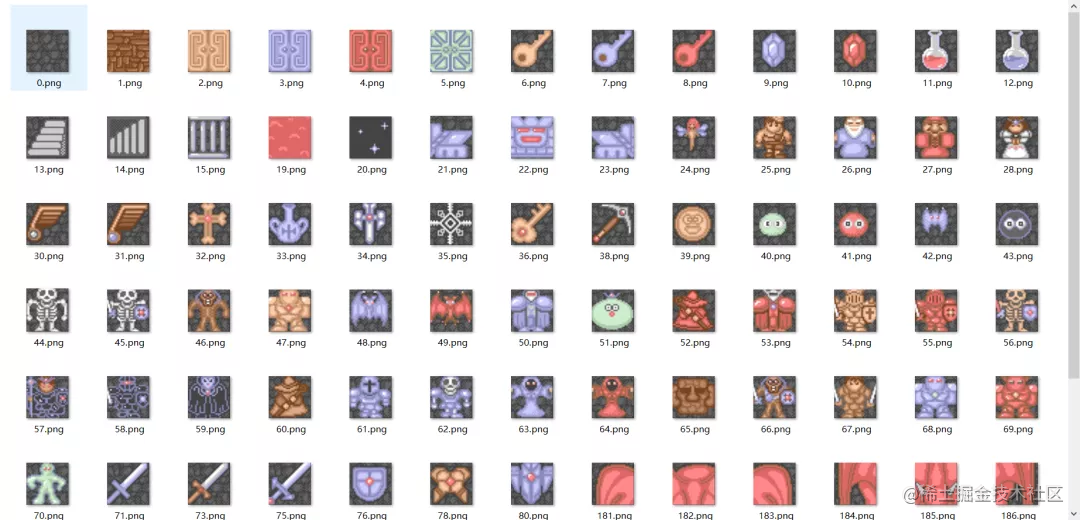
**图2 游戏画面图**

要实现图2这样的效果我们需要先在文本文件里定义游戏地图的样子，类似图3所示这样子，其中每个数字代表一种游戏元素：



**图3 定义元素图**

其中游戏中的图片素材我式收集到了网上找到的别人整理好的游戏素材，当然由于时间原因并没有全部用到：



**图4 游戏元素图**

于是，我们可以写一个游戏地图文件的解析类，就像这样：

'''游戏地图解析类'''

class MapParser():

def \_\_init\_\_(self, blocksize, filepath, element\_images, offset=(0, 0), \*\*kwargs):

self.count = 0

self.switch\_times = 15

self.image\_pointer = 0

self.offset = offset

self.blocksize = blocksize

self.element\_images = element\_images

self.map\_matrix = self.parse(filepath)

'''解析'''

def parse(self, filepath):

map\_matrix = []

with open(filepath, 'r') as fp:

for line in fp.readlines():

line = line.strip()

if not line: continue

map\_matrix.append([c.strip() for c in line.split(',')])

return map\_matrix

'''将游戏地图画到屏幕上'''

def draw(self, screen):

self.count += 1

if self.count == self.switch\_times:

self.count = 0

self.image\_pointer = int(not self.image\_pointer)

for row\_idx, row in enumerate(self.map\_matrix):

for col\_idx, elem in enumerate(row):

position = col\_idx \* self.blocksize + self.offset[0], row\_idx \* self.blocksize + self.offset[1]

if elem+'.png' in self.element\_images:

image = self.element\_images[elem+'.png'][self.image\_pointer]

image = pygame.transform.scale(image, (self.blocksize, self.blocksize))

screen.blit(image, position)

**3.2.2 勇士类**

首先，我们来定义一下我们的英雄勇士类，左边原来有文字显示勇士当前的状态：

'''定义我们的主角勇士'''

class Hero(pygame.sprite.Sprite):

def \_\_init\_\_(self, imagepaths, blocksize, position, fontpath=None):

pygame.sprite.Sprite.\_\_init\_\_(self)

# 设置基础属性

self.font = pygame.font.Font(fontpath, 40)

# 加载对应的图片

self.images = {}

for key, value in imagepaths.items():

self.images[key] = pygame.transform.scale(pygame.image.load(value), (blocksize, blocksize))

self.image = self.images['down']

self.rect = self.image.get\_rect()

self.rect.left, self.rect.top = position

# 设置等级等信息

self.level = 1

self.life\_value = 1000

self.attack\_power = 10

self.defense\_power = 10

self.num\_coins = 0

self.experience = 0

self.num\_yellow\_keys = 0

self.num\_purple\_keys = 0

self.num\_red\_keys = 0

'''将勇士绑定到屏幕上'''

def draw(self, screen):

screen.blit(self.image, self.rect)

font\_renders = [

self.font.render(str(self.level), True, (255, 255, 255)),

self.font.render(str(self.life\_value), True, (255, 255, 255)),

self.font.render(str(self.attack\_power), True, (255, 255, 255)),

self.font.render(str(self.defense\_power), True, (255, 255, 255)),

self.font.render(str(self.num\_coins), True, (255, 255, 255)),

self.font.render(str(self.experience), True, (255, 255, 255)),

self.font.render(str(self.num\_yellow\_keys), True, (255, 255, 255)),

self.font.render(str(self.num\_purple\_keys), True, (255, 255, 255)),

self.font.render(str(self.num\_red\_keys), True, (255, 255, 255)),

]

rects = [fr.get\_rect() for fr in font\_renders]

rects[0].topleft = (160, 80)

for idx in range(1, 6):

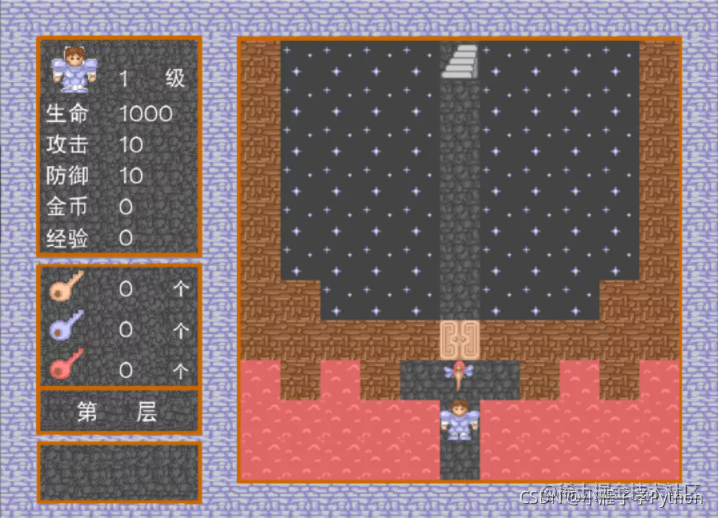
rects[idx].topleft = 160, 127 + 42 \* (idx - 1)

for idx in range(6, 9):

rects[idx].topleft = 160, 364 + 55 \* (idx - 6)

for fr, rect in zip(font\_renders, rects):

screen.blit(fr, rect)



**图5 勇士类**

**3.2.3 移动设计**

完成了勇士类最基础的定义，接下来我们就该让他动起来，具体而言，我们先实现一个勇士行动的类函数：

'''行动'''

def move(self, direction):

assert direction in self.images

self.image = self.images[direction]

move\_vector = {

'left': (-self.blocksize, 0),

'right': (self.blocksize, 0),

'up': (0, -self.blocksize),

'down': (0, self.blocksize),

}[direction]

self.rect.left += move\_vector[0]

self.rect.top += move\_vector[1]

然后写个按键检测，并根据玩家按下的键值来决定勇士的行动方向：

key\_pressed = pygame.key.get\_pressed()

if key\_pressed[pygame.K\_w] or key\_pressed[pygame.K\_UP]:

self.hero.move('up')

elif key\_pressed[pygame.K\_s] or key\_pressed[pygame.K\_DOWN]:

self.hero.move('down')

elif key\_pressed[pygame.K\_a] or key\_pressed[pygame.K\_LEFT]:

self.hero.move('left')

elif key\_pressed[pygame.K\_d] or key\_pressed[pygame.K\_RIGHT]:

self.hero.move('right')

# 行动冷却

self.move\_cooling\_count = 0

self.move\_cooling\_time = 5

self.freeze\_move\_flag = False

# 在冷却中的时候进行计数：

if self.freeze\_move\_flag:

self.move\_cooling\_count += 1

if self.move\_cooling\_count > self.move\_cooling\_time:

self.move\_cooling\_count = 0

self.freeze\_move\_flag = False

'''行动'''

def move(self, direction):

if self.freeze\_move\_flag: return

assert direction in self.images

self.image = self.images[direction]

move\_vector = {

'left': (-self.blocksize, 0),

'right': (self.blocksize, 0),

'up': (0, -self.blocksize),

'down': (0, self.blocksize),

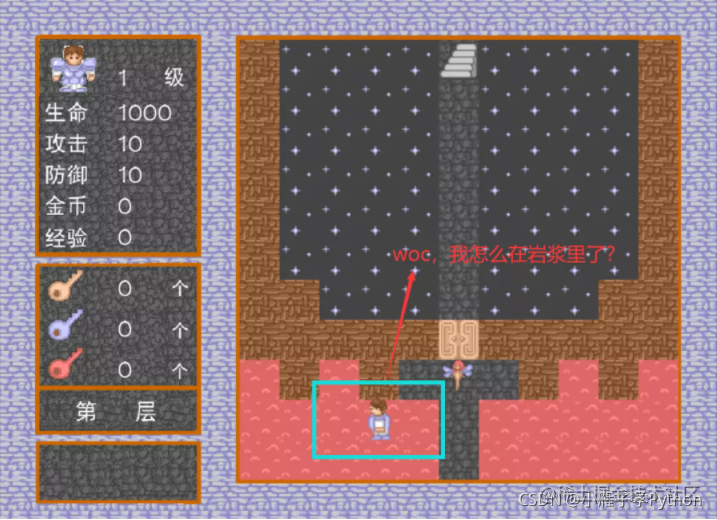
}[direction]

self.rect.left += move\_vector[0]

self.rect.top += move\_vector[1]

self.freeze\_move\_flag = True

最严重的问题，那就是行动会不合法，比如勇士会出现在这样的位置如图6：



**图6 位置不合法**

因此，我们需要再添加额外的移动是否合法的判断：

'''行动'''

def move(self, direction, map\_parser):

if self.freeze\_move\_flag: return

assert direction in self.images

self.image = self.images[direction]

move\_vector = {'left': (-1, 0), 'right': (1, 0), 'up': (0, -1), 'down': (0, 1)}[direction]

block\_position = self.block\_position[0] + move\_vector[0], self.block\_position[1] + move\_vector[1]

if block\_position[0] >= 0 and block\_position[0] < map\_parser.map\_size[1] and \

block\_position[1] >= 0 and block\_position[1] < map\_parser.map\_size[0]:

if map\_parser.map\_matrix[block\_position[1]][block\_position[0]] in ['0']:

self.block\_position = block\_position

elif map\_parser.map\_matrix[block\_position[1]][block\_position[0]] in ['24']:

self.dealcollideevent(

elem=map\_parser.map\_matrix[block\_position[1]][block\_position[0]],

block\_position=block\_position,

map\_parser=map\_parser,

)

self.rect.left, self.rect.top = self.block\_position[0] \* self.blocksize + self.offset[0], self.block\_position[1] \* self.blocksize + self.offset[1]

self.freeze\_move\_flag = True

**3.2.4 事件触发**

我们主要实现一些基础的功能，比如一些简单事件的触发，包括遇到门，捡到钥匙等等：

'''处理撞击事件'''

def dealcollideevent(self, elem, block\_position, map\_parser):

# 遇到不同颜色的门, 有钥匙则打开, 否则无法前进

if elem in ['2', '3', '4']:

flag = False

if elem == '2' and self.num\_yellow\_keys > 0:

self.num\_yellow\_keys -= 1

flag = True

elif elem == '3' and self.num\_purple\_keys > 0:

self.num\_purple\_keys -= 1

flag = True

elif elem == '4' and self.num\_red\_keys > 0:

self.num\_red\_keys -= 1

flag = True

if flag: map\_parser.map\_matrix[block\_position[1]][block\_position[0]] = '0'

return flag

# 捡到不同颜色的钥匙

elif elem in ['6', '7', '8']:

if elem == '6': self.num\_yellow\_keys += 1

elif elem == '7': self.num\_purple\_keys += 1

elif elem == '8': self.num\_red\_keys += 1

map\_parser.map\_matrix[block\_position[1]][block\_position[0]] = '0'

return True

# 捡到宝石

elif elem in ['9', '10']:

if elem == '9': self.defense\_power += 3

elif elem == '10': self.attack\_power += 3

map\_parser.map\_matrix[block\_position[1]][block\_position[0]] = '0'

return True

# 遇到仙女, 进行对话, 并左移一格

elif elem in ['24']:

map\_parser.map\_matrix[block\_position[1]][block\_position[0] - 1] = elem

map\_parser.map\_matrix[block\_position[1]][block\_position[0]] = '0'

return False

最后，我们来实现一下勇士上下楼梯时切换当前游戏地图的效果，这咋听起来似乎有点难办，但其实不然，只需要将发生上下楼梯事件的命令返回到游戏主循环：

# 上下楼梯

elif elem in ['13', '14']:

if elem == '13': events = ['upstairs']

elif elem == '14': events = ['downstairs']

return True, events

'''行动'''

def move(self, direction, map\_parser):

# 判断是否冷冻行动

if self.freeze\_move\_flag: return

assert direction in self.images

self.image = self.images[direction]

# 移动勇士

move\_vector = {'left': (-1, 0), 'right': (1, 0), 'up': (0, -1), 'down': (0, 1)}[direction]

block\_position = self.block\_position[0] + move\_vector[0], self.block\_position[1] + move\_vector[1]

# 判断该移动是否合法, 并触发对应的事件

events = []

if block\_position[0] >= 0 and block\_position[0] < map\_parser.map\_size[1] and \

block\_position[1] >= 0 and block\_position[1] < map\_parser.map\_size[0]:

# --合法移动

if map\_parser.map\_matrix[block\_position[1]][block\_position[0]] in ['0']:

self.block\_position = block\_position

# --触发事件

elif map\_parser.map\_matrix[block\_position[1]][block\_position[0]] in ['2', '3', '4', '6', '7', '8', '9', '10', '13', '14', '24']:

flag, events = self.dealcollideevent(

elem=map\_parser.map\_matrix[block\_position[1]][block\_position[0]],

block\_position=block\_position,

map\_parser=map\_parser,

)

if flag: self.block\_position = block\_position

# 重新设置勇士位置

self.rect.left, self.rect.top = self.block\_position[0] \* self.blocksize + self.offset[0], self.block\_position[1] \* self.blocksize + self.offset[1]

# 冷冻行动

self.freeze\_move\_flag = True

# 返回需要在主循环里触发的事件

return events

然后在主循环中进行响应即可：

# --触发游戏事件

for event in move\_events:

if event == 'upstairs':

self.map\_level\_pointer += 1

self.loadmap()

elif event == 'downstairs':

self.map\_level\_pointer -= 1

self.loadmap()

以上就是主要的一些流程以及代码。

**3 总结**

总体来说，这段代码实现了一个经典的RPG游戏-魔塔（MagicTower），虽然由于时间仅实现了10层。这只是一个简化版的游戏框架，需要完事其世界观以及后续的内容，要构建一个完整的游戏，还需要进一步扩展和完善，包括图形界面美化、用户提示、规则判定等方面的功能。比如经典魔塔有5种：50层魔塔（N.W、Oz & Kenichi做）、21层/[24层魔塔](https://baike.baidu.com/item/24%E5%B1%82%E9%AD%94%E5%A1%94?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AD%94%E5%A1%94/_blank)（胖老鼠做）、[新新魔塔](https://baike.baidu.com/item/%E6%96%B0%E6%96%B0%E9%AD%94%E5%A1%94?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AD%94%E5%A1%94/_blank)（cos180做）、[魔塔2006](https://baike.baidu.com/item/%E9%AD%94%E5%A1%942006?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AD%94%E5%A1%94/_blank)（Oksh做）、[魔塔2000](https://baike.baidu.com/item/%E9%AD%94%E5%A1%942000?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AD%94%E5%A1%94/_blank)（ANGELA做），其中，具有里程碑意义的魔塔有三种：50层魔塔、21/24层魔塔、新新魔塔。综上所述，在设计这款游戏的过程中遇到了许多困难，经过大量时间的学习以及资料查找完成了简易版的魔塔demo，很有成就感，对研究生生涯的学习有很大的帮助，代码能力也有所提升。