《坦克大战》练习程序
指导教程

目录

1. pygame 基础

1.1. pygame 简介

Pygame 是一个专门用来开发游戏的 Python 模块，主要为开发、设计 2D 电子游戏而生，它是一个免费、开源的第三方软件包，支持多种操作系统，具有良好的跨平台性(比如 Windows、Linux、Mac 等)。Pygame 是在 SDL(Simple DirectMedia Layer，一套开源的跨平台多媒体开发库)基础上开发而来，其目的是取代 PySDL。

SDL 主要用于多媒体领域，比如开发游戏、模拟器、媒体播放器等。SDL 是一套开放源代码的跨平台多媒体开发库，使用 C 语言编写，它提供了多种控制图像、声音、输入/输出的函数。

Pygame 非常适合初学者学习，使用者可以把它当做进入游戏开发世界的“敲门砖”，通过对 Pygame 的学习，即能够练习 Python 编程同时也可以让使用者了解到更为广阔的世界。

1.2. pygame 的安装

最为轻便的一种安装方式，推荐大家使用。首先要确保电脑中已经安装了 Python (建议使用 3.7 以上版本)，然后打开 cmd 命令行工具，输入以下命令即可成功安装：

```
pip install pygame
```

上述安装方法同样适用于 Linux 和 Mac 操作系统。

1.3. pygame 极简教程

1.3.1. pygame 模块与对象简介

pygame 中有许多模块，每个模块对应不同的功能，常用模块如下表所示：

模块名	说明	模块名	说明
pygame.display	访问显示设备	pygame.locals	包含各种常量
pygame.draw	绘制形状、线和点	pygame.mixer	装载和播放声音
pygame.event	管理事件	pygame.mouse	使用鼠标

pygame.font	使用字体	pygame.music	控制流音频
pygame.image	加载图片	pygame.time	管理时间和帧信息
pygame.key	读取键盘按键	pygame.sprite	包含游戏基础对象

pygame 中的关键对象如下表所示

模块名	说明	模块名	说明
pygame.Color	表示颜色的对象	pygame.Rect	存放矩形区域坐标的对象
pygame.Surface	表示图像的对象	pygame.	

详细的模块及 API 介绍，请参见：https://www.pygame.org/docs/

1.3.2. pygame 游戏基本运作方式

每个 Pygame 游戏的核心都是一个主循环，将其称为“游戏循环”。这个循环一直在不断运行，一遍又一遍地执行游戏运行时要做的所有事情。每次循环显示一次游戏当前画面，称为帧。

Pygame 游戏程序的主循环中需要实现 3 个基础操作：

1)响应并处理用户事件（鼠标点击或键盘按下事件）

游戏在运行时需要响应和处理用户的相关操作，从而实现游戏与用户的交互，用户操作可能对应的就是键盘上的键被按下，或鼠标被点击等事件。

2)更新游戏对象位置或状态

如果飞机对象在空中飞行，受到重力作用，自身的位置需要改变。如果两个对象相互碰撞，

则需要爆炸或停止移动等。

3)游戏画面的渲染

游戏根据需要在屏幕上重新绘制所有游戏对象或者所有移动的对象，实现窗口显示内容的即时刷新。

1.3.3. 游戏的基本程序框架

pygame 开发游戏程序的基本程序框架比较简单，如下给出示例代码予以说明：

```
1.  import pygame
2.
3.  WIDTH = 480  # 游戏窗口的宽度
4.  HEIGHT = 360 # 游戏窗口的高度
5.  FPS = 30 # 帧率
6.  # 初始化 pygame 并创建窗口
7.  pygame.init()
8.  screen = pygame.display.set_mode((WIDTH, HEIGHT)) #screen 也用于后续示例代码中
9.  pygame.display.set_caption("游戏窗口的标题")#设置游戏窗口标题栏文字
10. # 游戏主循环
11. while True:
12. # 循环中要实现核心三件事
13. # 1、事件处理
14. # 2、更新状态
15. # 3、渲染显示
```

上述代码中主要包括两部分，第一部分是对 pygame 进行初始化并建立主窗口，第二部分是建立主循环。

第 1 行导入 pygame 库。

第 3-5 行设置好一些配置变量

第 7 行，pygame.init()是启动 pygame，并进行“初始化”操作。

第 8 行，按照在配置常量中设置的窗口大小，创建游戏屏幕对象并赋值给 screen。

第 11 行，主循环的开始处。循环中要实现上节中所述的 3 个基础处理。

注意，由于上述代码没有实现事件处理，因此执行后无法通过常规操作来关闭游戏窗口。

1.3.4. 图像的渲染、绘制

如下代码实现了窗口背景色的设置和屏幕的更新。

```
# 设置背景颜色
screen.fill((255, 255, 255))  # (255, 255, 255)为白色
# 更新屏幕
pygame.display.update()
```

游戏画面中显示的很多对象，通常都是直接使用图片，不需要程序进行绘制，这样做实现比较简便，而且显示内容的制作也不会受到程序的限制。图片都是以图像文件的方式单独存放，在需要使用时，通过 pygame.image.load()方法进行装载。

在 pygame 中，pygame.image.load()装载图像文件后，会生成一个 Surface 对象，这个对象可以通过 blit()方法绘制在屏幕上。示例代码如下：

```
pygameImage = pygame.image.load('pygame_tiny.png')
screen.blit(pygameImage,(50,50))  # 将图像绘制到屏幕上
pygame.display.update()
```

通过装载图像能生成一个 Surface 对象，也可以直接创建 Surface 对象，然后将该对象绘制到屏幕上。示例如下：

```
#创建一个 50*50 的图像,并优化显示
face = pygame.Surface((50,50), flags=pygame.HWSURFACE)
#填充红色
face.fill(color='red')
# 将绘制的图像添加到主屏幕上，(100,100)是位置坐标
screen.blit(face, (100, 100))
```

1.3.5. 文字绘制

pygame 支持使用 pygame.font 对象将文本打印到窗口上。要打印文本的话，首先需要创建一个字体对象，Font 的第一个参数为 None 是告诉 pygame 获得系统默认字体，也可以是具体的字体名称。Font 的第二个参数指明字体大小。如下所示：

```
myfont = pygame.font.Font(None,60)
```

创建了 Font 对象后，还需要使用这个对象，将要显示的文本进行渲染，转换成相应的图像对象后再进行绘制、显示，代码如下所示：

```
textImage = myfont.render("pygame", True, WHITE)
```

生成了 textImage 对象后，就可以使用 blit()方法来进行绘制了。上面代码中的 render 函数第一个参数是要绘制的文本，第二个参数是启用抗锯齿能力，第三个参数是设置文本的颜色。

下面的代码，将上面生成的文本图像 textImage 绘制到屏幕的坐标(10，100)处：

```
screen.blit(textImage, (10,100))
pygame.display.flip()
```

1.3.6. 时间控制

游戏循环的另一个重要方面是控制整个循环的运行速度。游戏中有个术语叫 FPS（Frames Per Second），它代表每秒帧数，也叫帧率。这意味着游戏循环每秒应发生多少次。这很重要，游戏不能运行的太快或太慢。另外，也不能在不同的计算机上呈现不同的运行速度。

注：在 Pygame 中时间以毫秒为单位(1 秒=1000 毫秒)，这样会使游戏的设计更为精细。

要实现时间控制，需要在进入主循环前创建一个 pygame.time.Clock 对象，然后在循环内调用该对象的 tick()方法即可。

程序的结构类似如下所示：

```
clock = pygame.time.Clock()
    while True:

        # 循环中的其他语句

        clock.tick(FPS)
```

tick()方法告诉 pygame 一秒循环多少次。如果设置 FPS 为 20，那么意味着我们希望游戏的每个循环持续 1/20(0.05)秒。如果循环代码（更新，绘图等）处理很快，只需要 0.03 秒，那么 pygame 将等待 0.02 秒。以上是计算机处理比较快的情况。如果电脑本身运行比较缓慢，导致一秒内未必能执行 20 次循环，那么 tick(20)就无法起作用了。

1.3.7. 事件处理

pygame.event 模块中的 get()方法可以从操作系统的消息队列中获取事件消息，并返回一个事件列表。对返回事件列表进行遍历，可检查预期事件是否存在，存在则表明事件发生了，那么就可以做相应处理。

如下为事件处理代码示例，通过点击窗口右上角按钮或者按下 ESCAPE 键可实现游戏的退出。

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    key = pygame.key.get_pressed()
    if key[pygame.K_ESCAPE]:
        pygame.quit()
        sys.exit()

    screen.fill((255, 255, 255))
    pygame.display.update()
```

Pygame 中事件类型与对应的属性如下表：

事件类型	属性
QUIT	none
ACTIVEEVENT	gain, state
KEYDOWN	key, mod, unicode, scancode
KEYUP	key, mod, unicode, scancode
MOUSEMOTION	pos, rel, buttons, touch
MOUSEBUTTONUP	pos, button, touch
MOUSEBUTTONDOWN	pos, button, touch
JOYAXISMOTION	joy (deprecated), instance_id, axis, value
JOYBALLMOTION	joy (deprecated), instance_id, ball, rel
JOYHATMOTION	joy (deprecated), instance_id, hat, value
JOYBUTTONUP	joy (deprecated), instance_id, button
JOYBUTTONDOWN	joy (deprecated), instance_id, button
VIDEORESIZE	size, w, h
VIDEOEXPOSE	none
USEREVENT	code

关于程序中所涉及到的事件原理，简要说明见附录一

1.3.8. 动画效果

移动效果

以图像为例，要实现移动，只要每次在屏幕上进行绘制时改变绘制坐标即可。图像的起点坐标可以通过 get_rect()方法获得。

get_rect()返回一个 Rect 对象表示该 Surface 对象的矩形区域。该矩形对象（Rect）有 left，top 属性表示起点坐标，width 和 height 属性为图像的尺寸。

如下代码片段，通过得到的 Rect 对象，进行起始位置的设置，然后再将该对象用做图像绘制时的定位参数，实现图像在画面上的横向移动。

```
pygameImage = pygame.image.load('pygame_tiny.png')
rect = pygameImage.get_rect()
rect.top = 100
clock = pygame.time.Clock()
while True:
    screen.fill((255, 255, 255))
    rect.left += 50
    if rect.left >= 360:
        rect.left = 0
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
    screen.blit(pygameImage,rect)
    pygame.display.update()
    clock.tick(30)
```

帧动画效果。

所谓帧动画，就是连续变换显示一系列不同的图，从而得到动画的效果。帧动画可以实现对象的动作效果，即使始终在一个位置不移动，但由于图片的轮动，目标对象还会表现为在原

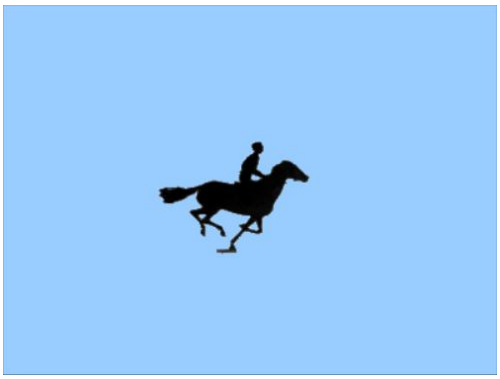
地不停地在做动作。

pygame 的精灵类支持实现这样的效果，具体实现间如下为示例：

```
import sys
import pygame

class RunningHorse(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__(self)
        self.image_list = []
        self.currentIndex = 0
        self.rect = None
        self.count = 0

    def load(self):
        # 载入组图
        for i in range(11):
            self.image_list.append(
                        pygame.image.load(f'horse ({i+1}).gif').convert_alpha())

    def update(self, left, top, rate=5):  # 刷新率
        self.image = self.image_list[self.currentIndex ]
        self.count += 1
        self.rect = self.image.get_rect()
        self.rect.left = left
        self.rect.top = top
        # 调整当前帧的下标
        if self.count % rate == 0:
            self.currentIndex += 1
            if self.currentIndex >= len(self.image_list):
                self.currentIndex = 0
            if self.count > len(self.image_list) * rate:
                self.count = 0


pygame.init()
screen = pygame.display.set_mode((400, 300))
pygame.display.set_caption("飞奔的骏马-帧动画")

clock = pygame.time.Clock()

# 创建精灵对象
runningHorse = RunningHorse()
```

```
runningHorse.load()
group = pygame.sprite.Group()
group.add(runningHorse)
while True:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    key = pygame.key.get_pressed()
    if key[pygame.K_ESCAPE]:
        pygame.quit()
        sys.exit()

    screen.fill((154, 205, 255))
    group.update(100, 100)
    group.draw(screen)
    pygame.display.update()
```

实现效果如下图：



2. 坦克大战游戏

2.1. 简要说明

最早的《坦克大战》是由 Namco 游戏公司开发的一款平面射击游戏，于 1985 年发售。游戏以坦克战斗及保卫基地为主题，属于策略型联机类。

本文以 Python+Pygame 为基础，通过开发一个简易版本的实验性小游戏，向读者介绍游戏基本功能的实现方法，并以此为手段来实践 Python 程序的制作过程。整个实践过程采用循序渐进的方式，分阶段介绍每项基础功能的实现思路和具体实现细节。

要完成整个游戏的制作，既需要注意到代码编写的细节之处，还需要对程序整体各方面有一定的规划，使程序整体逻辑协调，功能完善。

2.1.1. 目标游戏的基本设定

目标游戏主要的对抗双方分别是玩家坦克和电脑方坦克。

玩家可以通过键盘控制自己的坦克来摧毁电脑方的坦克，当把所有的电脑方坦克全部消灭即为胜利，反之自己被电脑方摧毁则失败。

游戏初始，电脑方坦克总数为 20 个，但画面中仅可同时出现 5 个(都可以自行设置)，玩家一个坦克，双方互相射击。电脑方坦克射击时，子弹对己方无效，只有打中玩家坦克，或者玩家坦克打中电脑方坦克，此时被击中的坦克才会出现爆炸效果并从画面中消失。

玩家有三次复活机会(可自行设置)，全部都被打中后游戏结束。

2.1.2. 基本构成元素

游戏启动会打开一个窗口，窗口中即为游戏画面，窗口右侧边有显示面板。游戏主要出现的元素有：玩家坦克、电脑方坦克、射出的子弹、画面中的障碍(墙)。

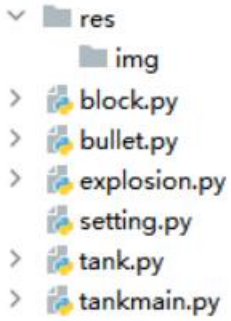
2.2. 项目分析

按照面向对象的设计思想对目标游戏程序进行分析，游戏中可以定义以下类：

- 程序主类：表示整个游戏程序。包括游戏启动、游戏结束的功能
- 坦克类(包括玩家和电脑方)：主要包括坦克的创建、显示、移动和射击功能。
  - ✓ 玩家坦克继承坦克类，有创建和与电脑方坦克碰撞的处理方法
  - ✓ 电脑方坦克继承坦克类，有创建、与玩家坦克碰撞的处理方法
- 子弹类：表示坦克射出的子弹。包括子弹创建、显示和移动的功能
- 障碍类：表示游戏画面中的障碍，如墙。有障碍的创建、显示功能。还有属性决定是否能被穿过
- 爆炸效果类：子弹击中坦克的效果。主要有显示爆炸效果的功能
- 侧边面板：游戏画面的右侧边沿用于显示信息的部分。主要是显示信息的功能

2.3. 程序的基本结构规划

根据上一节的分析，采用按照各类分别一个模块文件的策略，项目中的文件结构如下所示：



`res` 为资源文件目录，背景音效文件及图片文件保存在该目录下。其中图片文件存放在其下的子目录 `img` 中。

代码文件分别

- **`block.py`** 障碍类
- **`bullet.py`** 子弹类

- **explosion.py**　　爆炸效果类
- **setting.py**　　游戏的配置项
- **tank.py**　　坦克类及其子类
- **tankmain.py**　　游戏的主类

2.4. 游戏程序制作

为便于理解和制作，游戏采用分阶段逐步实现的方式。每一个阶段都有自己的目标，每个阶段都是下一阶段的前期准备。如下是各阶段的目标:

阶段 1(stage1)：搭建项目，创建基本程序结构。实现最初始的游戏主窗口及事件处理。

阶段 2(stage2)：玩家坦克的显示以及键盘控制下的转向与移动

阶段 3(stage3)：实现电脑方坦克的现实与移动

阶段 4(stage4)：为双方坦克增加子弹发射功能

阶段 5(stage5)：通过碰撞检测，实现子弹击中后的效果

阶段 6(stage6)：最后的完善。音效、玩家可复活、增加右边显示面板和游戏终结画面等

2.5. 阶段 1(stage1)搭建游戏基本框架

在 2.3 节所述内容的基础上，开始进入 stage1 阶段。本阶段是游戏初始阶段，目标是搭建游戏程序的基本框架，可以显示游戏窗口。

首先将窗口大小的配置写入 setting.py 文件。当前仅有两个配置项常量，窗口大小 SCREEN_SIZE 和背景色 BG_COLOR 两个元组变量

```
# 主窗口的长宽设置(宽，高)不包含右侧面板的宽度
SCREEN_SIZE = (800, 600)

BG_COLOR = (80, 150, 80)  # 背景色
```

主程序类负责显示和管理整个游戏的窗口，实现代码如下：

```
import sys
import pygame
from setting import *


class TankGame:
    def __init__(self):
        self.window = None

    # 游戏主程序
    def run(self):
        pygame.init()  # 初始化 pygame

        # 设置主窗口 Surface 对象的大小
        self.window = pygame.display.set_mode(SCREEN_SIZE)

        pygame.display.set_caption("坦克大战练习版")

        while True:
            # 给窗口设置填充色
            self.window.fill(BG_COLOR)
            # 获取事件,开始事件处理
            eventList = pygame.event.get()
            for event in eventList:
                # 判断是否点击了
                if event.type == pygame.QUIT:
                    self.exit()
```

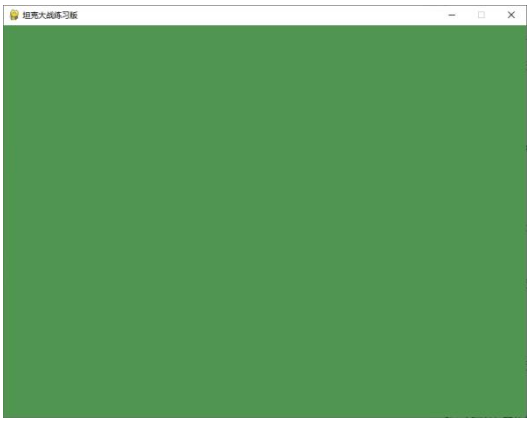
```
            pygame.display.update()

    # 结束游戏
    def exit(self):
        pygame.quit()  # 卸载 pygame 装载的全部模块
        sys.exit()  # 终止程序


if __name__ == '__main__':
    TankGame().run()
```

上述代码中，重点在 run()方法，主要实现了如下事项：

- 使用 pygame.init()完成对 pygame 的初始化
- 建立游戏主循环，并填充窗口，刷新屏幕显示
- 获取事件列表，并通过遍历该列表完成对目标事件的处理。目前仅实现了对退出事件的处理
- 实现了主程序类的 exit()方法，当要退出程序是，调用该方法

上述代码中，通过 from setting import * 导入了配置文件中的全部配置项，并在主程序文件中使用该配置项。后续代码依然采用此方式来引用相关配置项，因此关于配置项的设置与使用不再赘述。

本阶段实现了游戏窗口的现实，并通过点击窗口右上角的关闭按钮可结束程序运行。效果截图如下：



2.6. 阶段 2(stage2)玩家坦克的显示与移动

本阶段目标：在游戏窗口中现实玩家坦克，并实现坦克的上下左右移动。

2.6.1. 坦克的显示

1、实现思路

首先在 res\img 目录下，准备好坦克的图片。玩家坦克与电脑方坦克图片不同，且每个都有上、下、左、右四个方向，如下所示：

要实现坦克的显示，基本过程如下：

首先，通过装载坦克图片文件，生成 Surface 对象。

由于坦克有四个方向，因此需要生成四个方向的 Surface 对象。在程序中，可以通过字典结构，将代表四个方向的'U'、'D'、'L'、'R'作为 key，与对应的 Surface 对象相关联。

创建了坦克的 Surface 对象后，就可以通过主窗口 Surface 对象的 blit()方法，把坦克 Surface 对象画上去

最后，通过 display 模块对窗口中 Surface 对象进行刷新显示操作，就将坦克显示在了屏幕上。

基于上述分析，可以在坦克类中添加一个用于处理显示的方法，即 display()。

2、代码实现

基于上述分析，坦克类创建如下：

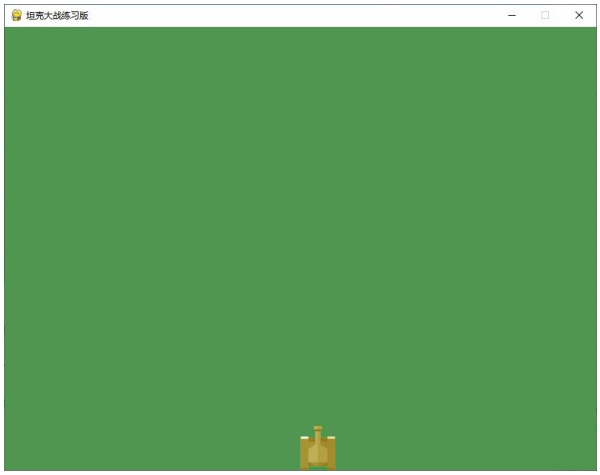
```
# 基础坦克类
class Tank:
    def __init__(self, left, top):
        """
        坦克对象
        :param left: 左边距
        :param top: 上边距
        """
        self.images = {  # 存放不同方向的图片,备选
            'U': pygame.image.load('res/img/tank_up.png'),
            'D': pygame.image.load('res/img/tank_down.png'),
            'L': pygame.image.load('res/img/tank_left.png'),
            'R': pygame.image.load('res/img/tank_right.png'),
        }
        # 坦克的初始方向
        self.direction = 'U'
        # 当前显示用图片
        self.image = self.images[self.direction]
        # 获取图片当前区域
        self.rect = self.image.get_rect()
        # 设置坦克的初始位置
        self.rect.left = left
        self.rect.top = top

    # 坦克显示方法
    def display(self, screen):
        # 根据当前方向取得要展示的 image 对象
        self.image = self.images[self.direction]
        screen.blit(self.image, self.rect)
```

有了坦克类后，还需要修改主程序类，通过创建坦克对象并使用对象的显示方法来显示坦克。主程序类代码修改片段如下：

```
    def __init__(self):
        ...略...
        self.playerTank = None   # 增加玩家坦克变量

    # 初始化玩家坦克的方法
    def initPlayerTank(self):
        self.playerTank = Tank(400, 540)
    # 玩家坦克的显示处理
    def displayPlayerTank(self):
        self.playerTank.display(self.window)
```

```
def run(self):
    ...略...

    self.initPlayerTank()  # 新增玩家坦克的初始化
    while True:
        ......
        self.displayPlayerTank()   # 新增代码
        # 获取事件,开始事件处理
        eventList = pygame.event.get()

    ...略...
```

3、屏幕效果



2.6.2. 坦克的移动

1、实现思路

实现坦克移动的基本思路：

首先，坦克在显示到屏幕上时，还需要提供显示的位置坐标。

主程序类的 run()方法中，通过一个无限循环结构，实现了对窗口画面的持续刷新，每次刷新，都需要将画面中的内容重新显示一遍。每次显示时，如果改变了坦克的显示位置坐标，那么，坦克再次显示时就会与上一次的位置不同。

通过持续地改变显示坐标和重新绘制，那么就可以实现屏幕上坦克移动的效果了。

基于上述的分析，可以考虑在坦克类的实现中 move()方法，每次调用该方法就可以实现

处理坦克显示坐标的改变，从而在屏幕上实现移动效果。

坦克类增加移动方法如下：

```
class Tank:

    ...已有代码略...

    # 移动方法
    def move(self):
        if self.direction == 'L':  # 朝向左移动
            if self.rect.left > 0:
                self.rect.left -= self.speed
        elif self.direction == 'R':  # 朝向右移动
            if self.rect.left + self.rect.width < SCREEN_SIZE[0]:  # 与屏幕宽度比较
                self.rect.left += self.speed
        elif self.direction == 'U':  # 朝向上移动
            if self.rect.top > 0:
                self.rect.top -= self.speed
        elif self.direction == 'D':  # 朝向下移动
            if self.rect.top + self.rect.height < SCREEN_SIZE[1]:  # 与屏幕宽度比较
                self.rect.top += self.speed

    ...已有代码略...
```

实现对玩家坦克的控制：

游戏中的两种坦克都可以采用上述思路来实现移动。但二者在游戏中的移动还是有区别的。玩家坦克的移动，需要由玩家进行控制，而电脑方坦克的移动是自动的。

要实现玩家坦克的人为控制，就需要程序能响应玩家的键盘操作，并根据玩家的按键来控制坦克的移动和方向。这需要解决两个问题：一是通过对应按键确定玩家坦克的移动方向；二是根据用户按键的行为决定玩家坦克的移动和停止。

对于控制移动方向的处理办法，就是在程序的事件处理中添加对方向键按下事件的捕获，然后改变玩家坦克的方向属性值即可。

对于实现移动及控制停、走要结合在一起来解决。首先，坦克类增加一个状态属性，以控制坦克对象当前是可移动状态还是停止状态。然后，在主程序类的坦克移动方法中，调用坦克

对象的 move()方法，实现其显示位置的改变，最终就可以实现坦克的移动显示效果。

2、代码实现

首先在主程序类的 run()方法中，修改事件处理代码如下：

```
def run(self):

    ...已有代码略...

    while True:
        ...已有代码略...
        # 获取事件,开始事件处理
        eventList = pygame.event.get()
        for event in eventList:
            # 判断按下的是否键
            if event.type == pygame.QUIT:
                self.exit()
            # 按键事件的处理
            if event.type == pygame.KEYDOWN:
                if self.playerTank:
                    # 按下方向键,改为移动状态
                    if event.key in (pygame.K_LEFT,
                                     pygame.K_RIGHT,
                                     pygame.K_UP,
                                     pygame.K_DOWN):  #
                        self.playerTank.stop = False  # 移动状态
                    if event.key == pygame.K_LEFT:
                        self.playerTank.direction = 'L'

                    elif event.key == pygame.K_RIGHT:
                        self.playerTank.direction = 'R'

                    elif event.key == pygame.K_UP:
                        self.playerTank.direction = 'U'

                    elif event.key == pygame.K_DOWN:
                        self.playerTank.direction = 'D'
            if event.type == pygame.KEYUP:
                # 抬起方向键,改为停止状态
                if self.playerTank and event.key in (pygame.K_LEFT,
                                                     pygame.K_RIGHT,
                                                     pygame.K_UP,
                                                     pygame.K_DOWN):
```

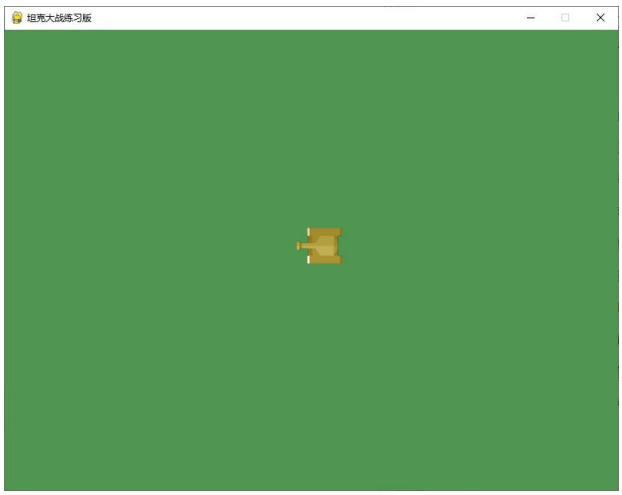
```
                self.playerTank.stop = True  # 停止状态
        pygame.display.update()
```

上述事件处理代码，仅是根据用户的按键来改变坦克的相应属性值，即 self.playerTank.direction 决定了坦克的方向；self.playerTank.stop 决定了坦克是走还是停。

坦克对象在显示的时候，根据状态属性的值来呈现相应的效果。效果的呈现，依赖于主程序类中 displayPlayerTank()对坦克对象显示和移动方法的调用。下面是对 displayPlayerTank 方法的修改，增加坦克对象 move()方法的调用语句。

```
    # 玩家坦克的显示处理
    def displayPlayerTank(self):
        self.playerTank.display(self.window)
        if self.playerTank and not self.playerTank.stop:  # 检查坦克对象是否为移动状态
            self.playerTank.move()  #通过调用 move 方法,实现移动
```

屏幕效果



3、需要改进的问题

虽然上述代码已经实现了玩家坦克的移动效果，由于计算机的处理速度非常快，所以，在实际运行后，存在坦克移动过快的问题。为了能够对移动速度进行控制，还需要通过 pygame.time 模块对游戏画面的刷新进行控制。

代码修改如下：

setting.py 中添加一个屏幕刷新率的静态配置项

```
    ...略...
FPS = 60  # 屏幕显示刷新率
```

主程序类的修改：

```
    def run(self):
        ...略...
        clock = pygame.time.Clock()  # 在主循环前创建 Clock 对象
        while True:
            ...略...
            pygame.display.update()
            clock.tick(FPS)  # 新增语句,通过 FPS 限制主循环的速度
```

有了上述的额外控制，玩家坦克的移动速度就变得可以接受了。

2.7. 阶段 3(stage3)电脑方坦克的显示与移动

在实现了玩家坦克的显示后，还需要在游戏画面中将电脑方的坦克显示出来。在坦克的显示和移动方面，电脑方坦克与玩家坦克的实现思路是一样的。

电脑方坦克的不同之处有两点。第一，要同时显示多个；第二，要能够自行移动。因此在实现时与玩家坦克略有不同。

电脑方坦克与玩家坦克功能虽有差别但基本上是一样的，因此，可以先定义一个基础的坦克类，通过继承的方式来派生玩家和电脑方坦克类。

电脑方坦克也和玩家坦克一样，需要在主程序类中增加初始化、显示的专门处理方法。

2.7.1. 坦克的显示

1、电脑方坦克的处理思路

电脑方坦克要同时显示多个，意味着在游戏时会同时有多个相应的坦克对象存在，并在显示、移动处理时，都要进行相应的操作。

电脑方坦克的运行无需人工干预，在本游戏中，采用随机控制的策略，这样也易于实现。随机控制包括创建时的朝向是随机的，在运行中也可以在一定距离后随机变换方向。为了控制电脑方坦克在移动了一定步数后开始变换方向，对象中需增加一个计步的属性变量 `step`，同时要有一个新的移动方法 `randMove()`，不仅可以移动还能实现变换方向，最后添加一个随机产生方向的方法 `randDirection()`。

为了管理电脑方坦克，可以采用列表进行存储。因此需要修改主程序类，添加相应的列表变量，即 `computerTankList`。

电脑方坦克也需要创建和显示操作，因此，主程序类中也要增加相应的创建和显示方法，即 `initComputerTanks()` 与 `displayComputerTanks()`。

2、具体实现

根据上述分析，在 setting.py 文件中，增加一个新配置项常量

```
STEP = 100  # 电脑方坦克换向的移动长度
```

首先在 tank.py 文件中增加一个 Tank 类的子类 EnemyTank 作为电脑方坦克类。根据上述分析，代码如下：

```
class EnemyTank(Tank):
    def __init__(self, left, top):
        super().__init__(left, top)
        self.images = {  # 存放不同方向的图片
            'U': pygame.image.load('img/entank_up.png'),
            'D': pygame.image.load('img/entank_down.png'),
            'L': pygame.image.load('img/entank_left.png'),
            'R': pygame.image.load('img/entank_right.png'),
        }
        # 初始方向
        self.direction = self.randDirection()
        # 设置转向计步变量
        self.step = STEP

    def randDirection(self):
        return ['U', 'D', 'L', 'R'][random.randint(0, 3)]

    def randMove(self):
        if self.step <= 0:
            self.direction = self.randDirection()
            # 变换方向后,重置计步数
            self.step = STEP
        else:
            self.move()
            self.step -= 1  # 每移动一次计步减少一次
```

准备好电脑方坦克类之后，需要继续对主游戏类进行修改以在游戏画面中实现电脑方坦克的显示。

修改内容有：

1)增加用于存储电脑方坦克的列表变量，computerTankLst

2)增加电脑方坦克初始化的方法 initComputerTanks()。其基本逻辑如下：

根据参数，循环创建 EnemyTank 对象，并 append()方法追加至 computerTankList 中。

在创建时，需传入左上偏移坐标值，上偏移固定，左偏移随机生成。

3)增加显示电脑方坦克的方法 displayComputerTanks()。电脑方坦克保存在computerTankList 列表中，该方法在实现时，需遍历 computerTankList 列表，对每一个电脑方坦克对象都调用 display()和 randMove()方法，完成移动与显示操作。

4)为了使代码更清晰，易于维护，将游戏主循环中的事件处理部分包装成一个方法eventProcess()

5)修改 run()方法，增加创建、显示电脑方坦克的语句

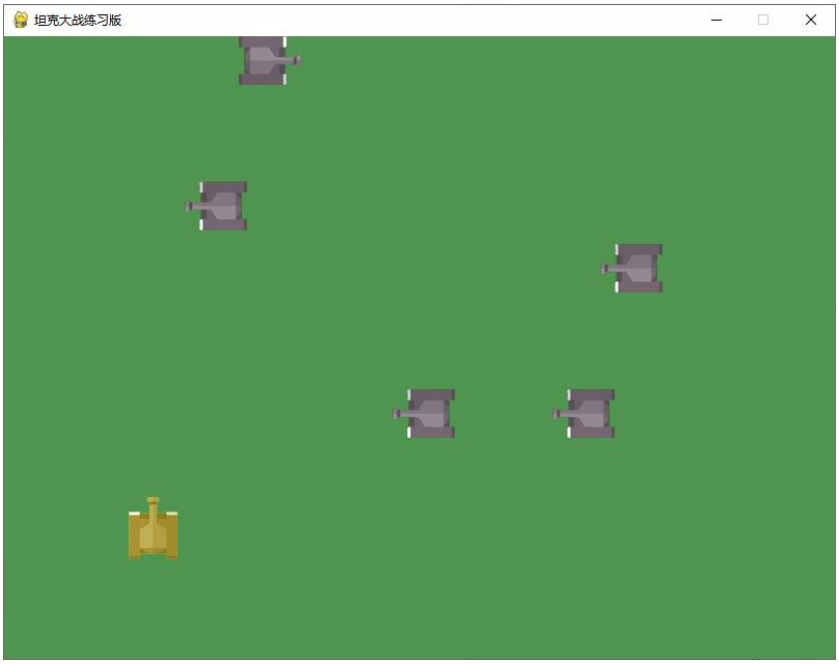
变更后的主程序类代码片段如下：

```
class TankGame:
    def __init__(self):
        ...略...
        self.computerTankList = []

    # 将事件处理的代码移至方法中
    def eventProcess(self):
        # 获取事件,开始事件处理
        eventList = pygame.event.get()
        for event in eventList:
            # 判断按下的是否键
            if event.type == pygame.QUIT:
                self.exit()
            # 按键事件的处理
            if event.type == pygame.KEYDOWN:
                if self.playerTank:
                    # 按下方向键,改为移动状态
                    if event.key in (pygame.K_LEFT,
                                     pygame.K_RIGHT,
                                     pygame.K_UP,
                                     pygame.K_DOWN):  #
                        self.playerTank.stop = False  # 移动状态
                    if event.key == pygame.K_LEFT:
                        self.playerTank.direction = 'L'

                    elif event.key == pygame.K_RIGHT:
                        self.playerTank.direction = 'R'

                    elif event.key == pygame.K_UP:
```

```
                    self.playerTank.direction = 'U'

                elif event.key == pygame.K_DOWN:
                    self.playerTank.direction = 'D'
        if event.type == pygame.KEYUP:
            # 抬起方向键,改为停止状态
            if self.playerTank and event.key in (pygame.K_LEFT,
                                                 pygame.K_RIGHT,
                                                 pygame.K_UP,
                                                 pygame.K_DOWN):
                self.playerTank.stop = True  # 停止状态

# 初始化电脑方坦克
def initComputerTanks(self, n):
    top = 10
    # 循环生成电脑方坦克
    for i in range(n):
        left = random.randint(0, 600)
        e = EnemyTank(left, top)
        self.computerTankList.append(e)

# 显示电脑方坦克
def displayComputerTanks(self):
    # 通过循环方式,逐个显示电脑方坦克
    for ct in self.computerTankList:
        ct.display(self.window)
        ct.randMove()

# 运行游戏
def run(self):
    ...已有代码略...
    self.initComputerTanks(5)  # 新增初始化电脑方坦克
    ...已有代码略...
    clock = pygame.time.Clock()
    while True:
        # 给窗口设置填充色
        self.window.fill(BG_COLOR)

        self.displayPlayerTank()
        self.displayComputerTanks()  # 显示电脑方坦克
        self.eventProcess()  # 变更为事件处理函数
        pygame.display.update()
        clock.tick(FPS)
```

3、屏幕效果

本阶段实现的效果如下，玩家可控制自己的坦克方向与移动，电脑方坦克随机在画面中自主移动。



2.8. 阶段 4(stage4)子弹的发射及数量控制

双方坦克都能够正常显示和移动后，本阶段的目标就是要实现射击功能。

2.8.1. 功能分析

根据游戏的设定，射击功能大致应该是这样的：

a、双方坦克都具有射击功能，但具体实现策略不同。

b、玩家坦克的射击由玩家控制，按空格键，就发射，不按就不发射。

c、玩家坦克的射击功能，有如下限制规则：

a）画面上最多同时显示三个射出的子弹

b）子弹移出画面，或击中目标消失后，才可继续进行射击

d、电脑方坦克的射击，也是采用随机动作的策略，可持续射击，但射击间隔时间由程序随机进行延迟。

e、为了方便实现，本阶段暂不考虑对射中目标后的效果处理，留待下阶段实现。

2.8.2. 发射功能的实现办法

1)基本规划

关于发射功能的实现，基本思路如下：

首先，坦克对象发射子弹，就是创建一个子弹对象，因此，坦克类要增加一个发射方法来实现子弹对象的创建。

子弹对象经发射方法创建后，不会马上消失，这时需要主程序类来负责保存。与处理电脑方坦克一样的策略，在主程序类中使用列表来保存子弹对象，玩家与电脑方射出的子弹需分开存放。

子弹对象产生后，还需要在游戏窗口中进行显示。子弹对象的显示，需要主程序类进行总控，因此主程序类中还要添加对子弹进行显示处理的新方法，玩家坦克和电脑方坦克的子弹，

因策略不同还需分不同方法进行处理。

在主程序类的总控下，会执行在窗口中显示子弹的操作，而在具体显示时，又是通过子弹类自己的显示方法来负责完成。同样，子弹的移动，也是通过子弹类自己的移动方法负责实现的。

综上所述，实现射击功能需要增加和修改的内容包括:

a、增加子弹类，表示射出的子弹。每次射击就是创建一个子弹对象。子弹对象具有显示与移动的方法。

b、坦克类要增加发射子弹的方法。通过该方法可以完成子弹对象的创建。

c、电脑方坦克增加自己的子弹发射方法，覆盖父类同名方法。

d、在主程序类中分别添加显示玩家坦克与电脑方坦克所发射子弹的方法

e、修改主程序类中主循环代码，添加调用显示子弹方法的语句

f、修改事件处理代码，增加玩家按下空格键实现射击动作的语句

2)子弹类的实现

下面，先来看一下子弹类的初始化都需要做哪些事情，准备哪些属性。

子弹在四个方向上的图片不同，需使用字典来存放装载子弹图片的 image 对象，key 为对应的方向字符。创建属性 images。

```
    images = {  # 存放不同方向的子弹图片
        'U': pygame.image.load('res/img/bullet-U.png'),
        'D': pygame.image.load('res/img/bullet-D.png'),
        'L': pygame.image.load('res/img/bullet-L.png'),
        'R': pygame.image.load('res/img/bullet-R.png'),
    }
```

子弹方向的属性，由该属性来决定显示的图片，初始化时，与坦克该子弹的方向一致

```
    self.direction = tank.direction
```

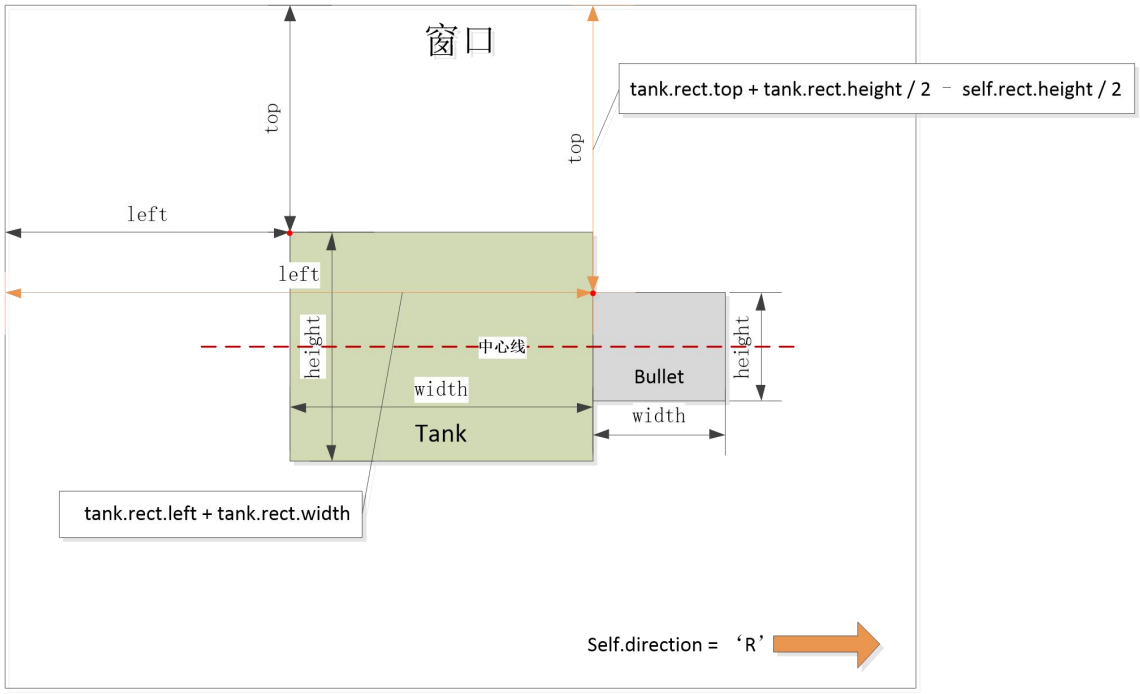
记录当前显示图片对象的属性，根据子弹方向属性，在子弹图像字典中提取。

```
        self.image = Bullet.images.get(self.direction)
```

记录子弹图像显示区域的属性，初始由图像对象获取

```
        self.rect = self.image.get_rect()
```

在子弹类进行初始化时，需要对子弹的显示位置进行设置，即子弹类属性 self.rect 的 left 与 top，该位置与发射子弹的坦克相关关联。因此，需要基于坦克对象的显示位置，根据当前的方向，进行初始位置的计算，计算逻辑如下图所示(以右侧方向为例)：



子弹类还需要对移动速度进行设置，还需要有一个属性来记录自己的状态是否有效，故添加如下两个属性。移动速度设置为坦克速度的 2 倍。

```
        self.speed = MOVE_SPEED * 2
        self.alive = True
```

综合如上分析，子弹类的初始化方法__init__()如下。子弹的发射与具体的某个坦克对象相关，因此，在初始化方法中增加了一个参数 tank，通过该参数将关联坦克对象传递进来：

```
    def __init__(self, tank):
        # 子弹的方向与坦克的一致
        self.direction = tank.direction
        # 设置子弹图片
        self.image = Bullet.images.get(self.direction)
        # 获取区域
```

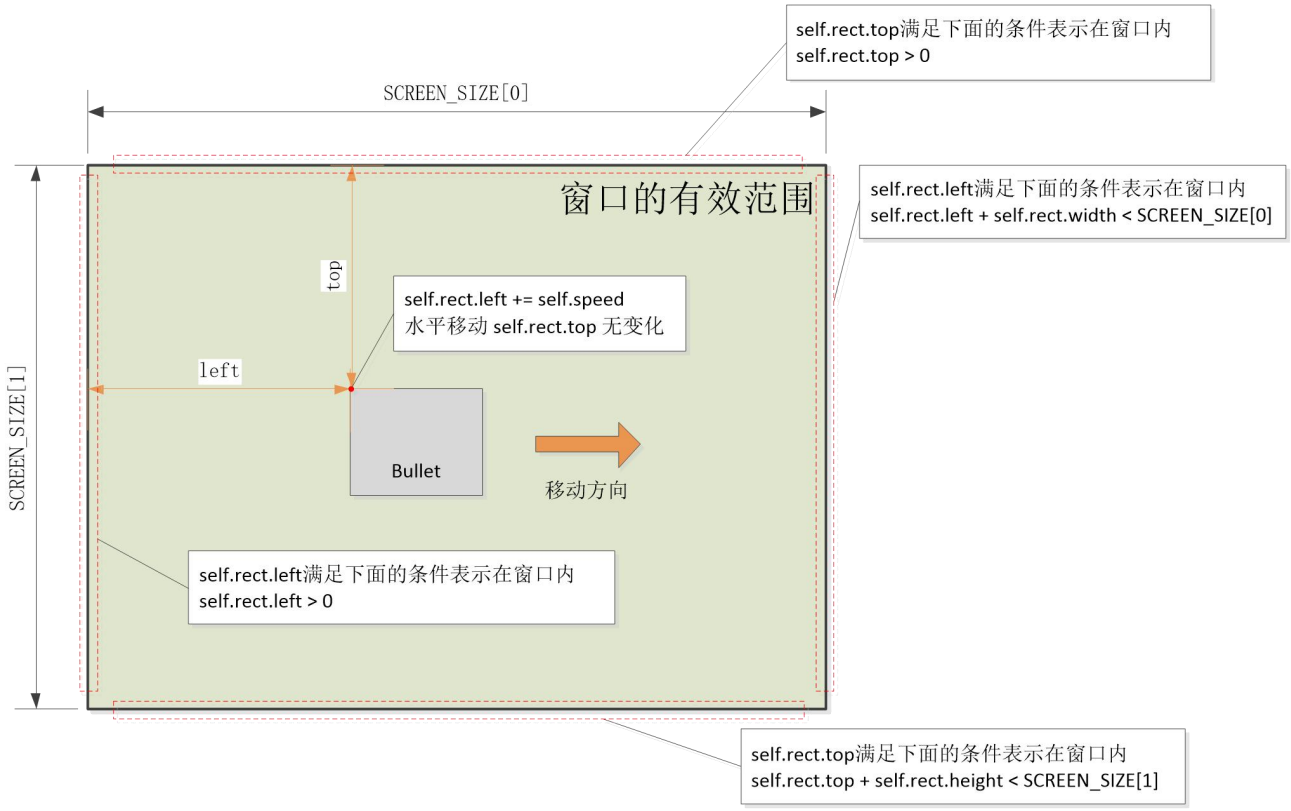
```
        self.rect = self.image.get_rect()
        # 子弹的初始位置与坦克的方向有关
        if self.direction == 'U':
            self.rect.left = tank.rect.left + tank.rect.width /2- self.rect.width/2
            self.rect.top = tank.rect.top - self.rect.height
        elif self.direction == 'D':
            self.rect.left = tank.rect.left + tank.rect.width /2- self.rect.width/2
            self.rect.top = tank.rect.top + tank.rect.height
        elif self.direction == 'L':
            self.rect.left = tank.rect.left - self.rect.width
            self.rect.top = tank.rect.top + + tank.rect.height/2 - self.rect.height/2
        elif self.direction == 'R':
            self.rect.left = tank.rect.left + tank.rect.width
            self.rect.top = tank.rect.top + tank.rect.height/2 - self.rect.height/2

        # 子弹的速度
        self.speed = MOVE_SPEED * 2
        # 表示子弹是否有效,初始为有效状态
        self.alive = True
```

接下来是子弹类的显示方法 `display()`。这个方法的逻辑比较简单，只是通过 `blit` 方法将自己添加到父 `Surface` 对象中，故代码如下：

```
    # 显示方法
    def display(self, screen):
        screen.blit(self.image, self.rect)
```

子弹类的移动也需要一个方法 `move()`来实现。其实现的基本逻辑，就是每次调用该方式时，根据子弹的初始方向，对子弹的显示起始位置进行修改，这样，当屏幕刷新显示子弹时，由于起始位置已经变化，因此再次显示在窗口中时位置也跟着发生变化，从而实现了“移动”的效果。

子弹类的 `move()`方法是通过修改子弹对象的显示起始位置来实现移动效果的。在改变位置后，还需要对位置的有效性进行检查，如果子弹移出了窗口的有效范围，也就无需显示了，此时需将子弹设置为无效状态。子弹位置与有效范围的检查示意如下图：



上图中揭示了子弹移动时，起始位置 self.rect.left、self.rect.top 的变化规律(仅右方向)。self.rect.left 的增加与减少，分别对应子弹的右移和左移，self.rect.top 的增加与减少，对应子弹的下移与上移。游戏中子弹的运动比较简单，射出后只会沿着初始方向单向移动，即仅为左、右、上、下四个方向，不会再有其他的变化。self.speed 可以控制移动的速度。

子弹要在窗口中显示出来，起始位置需要在窗口的有效范围内，即：

self.rect.left 值在 0 ~ SCREEN_SIZE[0]-self.rect.width 之间都是有效的。

self.rect.top 值在 0 ~ SCREEN_SIZE[1]- self.rect.height 之间都是有效的。

所以，当子弹的初始位置超出有效范围后，应将子弹设置为无效状态。

综上，子弹类的 move()方法的代码如下：

```
# 移动方法
def move(self):
    if self.direction == 'L':
        if self.rect.left >= 0:
```

```
                self.rect.left -= self.speed

        elif self.direction == 'R':
            if self.rect.left + self.rect.width < SCREEN_SIZE[0]:  # 与屏幕宽度比较
                self.rect.left += self.speed

        elif self.direction == 'U':
            if self.rect.top >= 0:
                self.rect.top -= self.speed

        elif self.direction == 'D':
            if self.rect.top + self.rect.height < SCREEN_SIZE[1]:  # 与屏幕宽度比较
                self.rect.top += self.speed
        # 此处判断返回要是不准确，可能导致个别子弹停在屏幕边上消不掉
        if self.rect.left < 0 or \
                self.rect.left + self.rect.width >= SCREEN_SIZE[0] or \
                self.rect.top < 0 or \
                self.rect.top + self.rect.height >= SCREEN_SIZE[1]:
            self.alive = False
```

至此，子弹类就准备好了。

3)坦克类的修改

坦克要实现子弹的发射，需要增加相应的发射方法 shot()。坦克类的 shot()方法主要就是实现子弹对象的创建，子弹对象被创建后，就可以用于在游戏窗口中进行显示。

坦克类的 shot()方法如下：

```
class Tank:
    ...略...
    # 射击方法
    def shot(self):
        return Bullet(self)
```

调用 Bullet()创建子弹对象时，将代表自身的 self 作为参数，这样就将新创建的子弹与坦克自身建立了关联。

电脑方坦克的射击是自动的，因此，其射击方法不使用父类的，需要重新定义。

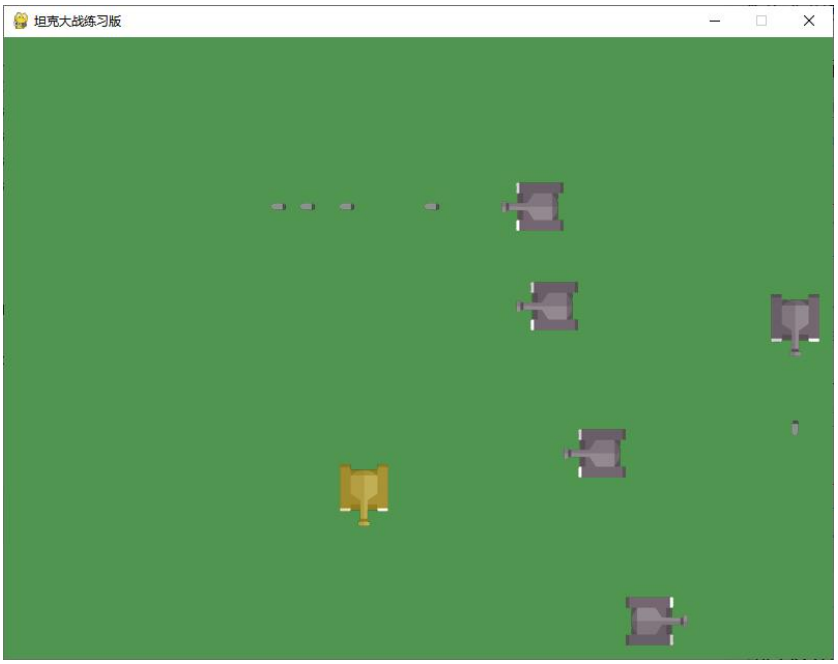
```
class EnemyTank(Tank):
    ...略...
    # 射击方法
    def shot(self):
```

```
        num = random.randint(1, 1000)
        if num < 10:
            return Bullet(self)
```

4)主程序的修改

当双方坦克发射子弹时，会创建响应的子弹类，主程序中将创建的子弹放入列表中，每次在刷新程序窗口时，将遍历这些列表，逐一将子弹显示在窗口中。

2.8.3. 实现效果



2.9. 阶段 5(stage5)碰撞检测

截至目前为止，已经实现了坦克的移动、射击功能，但是游戏还存在问题。这个问题表现为，一是双方坦克在相遇时会穿过对方，二是射出的子弹也会穿目标而过。本阶段的目标是在游戏中借助于 pygame 自有的碰撞检测功能，来实现游戏中各元素的碰撞检测，以解决坦克互相穿过的问题，同时实现子弹击中目标的处理。

2.9.1. 功能分析

游戏中要实现碰撞检测，为了简化实现，基本思路就是借助 pygame 中的精灵类(pygame.sprite.Sprite)来实现。pygame 的碰撞检测是建立在自有精灵类基础上的，即只对精灵类及其子类的对象有效。因此，利用类的继承性，将已有的坦克、子弹类也变更成精灵类的子类即可。

有了碰撞检测后，当两个坦克走到了一起即发生了碰撞，此时就应该停下来；当子弹与目标发生碰撞后，就应该做击中效果的处理。

下面分别就坦克的碰撞检测与处理以及子弹击中后的效果处理分别加以说明。

2.9.2. 坦克的碰撞检测

游戏当中实现了对坦克的碰撞检测，那么坦克的动态效果在实现上就要增加一些处理，这样在显示效果上就会更加符合自然逻辑。

改进后的坦克移动处理策略应该是这样的：

画面中双方坦克会来回移动，发生交汇碰撞是肯定的，坦克是实体，不应该像幽灵一样的彼此穿过。当坦克交汇时，程序可以利用碰撞检测感知两个坦克将要被画在同一个区域内，此时则应该让坦克停止移动，这样的话，就好像坦克碰上了障碍，画面上也就不会再出现相互穿过的景象了。

下面阐述一下在程序中如何使用 pygame 的碰撞检测。

首先创建一个用于碰撞检测的基类，该类继承了 pygame.sprite.Sprite，然后再以该类作为坦克和子弹类的共同父类即可。该类仅有一个 alive 属性，表示当前对象的生存状态。默认是生存有效状态。

```
# 为了实现碰撞检测，定义一个继承 Sprite 类的共同基类
# 这样 tank，bullet 等就都可以借助精灵类实现碰撞检测
class SpriteObject(pygame.sprite.Sprite):
    def __init__(self):
        # 调用父类 (Sprite) 的构造方法
        super().__init__(self)

        # 表示元素有效
        self.alive = True
```

有了上述的基类后，修改坦克类的定义如下：

```
# 基础坦克类
class Tank(SpriteObject):  # 该为继承自 SpriteObject 类
    def __init__(self, left, top):
        super(SpriteObject, self).__init__()
        ...已有代码略...
```

坦克类继承了 SpriteObject 后，还需要添加一个自定义的碰撞检测方法 checkCollide()以实现坦克的碰撞检查。该方法的策略就是，将待检测的目标对象列表作为参数传入，然后遍历这个列表，利用 pygame.sprite.collide_rect()方法逐个与自身进行检测，如果发生碰撞，那么就调用自己的 halt()方法停止移动。

```
    # 检测与其他坦克的碰撞
    def checkCollide(self, spriteObjectList):
        for obj in spriteObjectList:
            # 此处判断要排除掉 obj 对象可能为 None 的情况
            if obj and obj is not self and pygame.sprite.collide_rect(obj, self):
                self.halt()
```

如上述代码所示，坦克停止移动的方法是 halt()，也是本次需要添加的。

要将坦克的移动停下来，其实现策略比较简单，坦克对象能够记录自己上一次的绘制起始位置坐标，即 left 与 top 值，然后在需要停止操作时，只需将当前的绘制起始位置坐标还原为先前的值即可。因为起始坐标位置不动，那么在每次刷新画面重绘坦克时，它的位置就不会

改变，也就停了下来。

上述策略的具体实现需要对 tank 类增加两处内容以及修改一处：

1、增加两个属性，用来记录历史绘制起始坐标，分别为 lastLeft 与 lastTop

2、增加 halt()方法

3、修改 move()方法，在每次移动之前，即改变当前起始位置坐标前，将坐标值保存到上述新增的属性中。

代码片段如下：

```
class Tank(SpriteObject):
    def __init__(self, left, top):
        ...已有代码略...
        # 添加用于保存坦克历史起始位置的两个属性
        self.lastLeft = self.rect.left
        self.lastTop = self.rect.top

    # 移动方法的修改
    def move(self):
        self.lastLeft = self.rect.left  #  每次移动操作前,记录上一次的位置
        self.lastTop = self.rect.top
        ...已有代码略...

    # 增加的方法。
    # 碰上障碍时,通过该方法使坦克保持原地不动
    def halt(self):
        self.rect.left = self.lastLeft
        self.rect.top = self.lastTop

    ...已有代码略...
```

至此，坦克类对于碰撞检测功能的支持就算改好了。下面，还需要将这些修改应用到主程序中。

主程序类是整个游戏的总控程序，坦克对象的创建、显示、移动也都是在主程序中发起的，因此，程序的大流程中要添加碰撞检测，还需要对既有的玩家坦克和电脑方坦克的显示方法进

行修改，增加对坦克对象的碰撞检查方法的调用语句，这样才真正能够在程序运行中开启对碰撞的检测。

首先修改主程序类中玩家坦克的显示方法 displayPlayerTank()。该方法通过调用玩家坦克对象的对应方法，来实现显示、移动操作。现在，调价调用 checkCollide()方法，就可以实现玩家坦克对象与所有电脑方坦克对象的碰撞检测。添加的代码如下片段所示：

```
class TankGame:
    ...已有代码略...

    # 玩家坦克的显示处理
    def displayPlayerTank(self):
        ...已有代码略...
        # 添加的语句。检查玩家坦克与电脑方坦克的碰撞
        self.playerTank.checkCollide(self.computerTankList)

    ...已有代码略...
```

同样道理，电脑方坦克的显示方法也需要添加类似语句。不过电脑方坦克数量多于一个，因此即需要检测与玩家坦克的碰撞，也需要检查与其他电脑方坦克的碰撞。修改后的代码片段如下：

```
    # 电脑方坦克的显示处理
    def displayComputerTanks(self):
        # 通过循环方式,逐个处理电脑方坦克
        for ct in self.computerTankList[:]:
            ...已有代码略...
            # 添加碰撞检测的语句
            ct.checkCollide([self.playerTank])  # 检测与玩家坦克的碰撞
            ct.checkCollide(self.computerTankList)  # 检测与其他坦克的碰撞
```

以上就实现了玩家、电脑方坦克的碰撞检测功能。

2.9.3. 子弹的碰撞检测

坦克发生碰撞会停止移动，子弹与其他对象放生碰撞，就表示击中了目标，如果目标是敌方坦克，那么还需要进行击中效果的处理。

对于子弹发生碰撞的显示策略，应该是这样的：在击中目标后，按照正常逻辑子弹应该在

画面中消失，如果击中了敌方坦克，那么对方也应该在画面中消失，并且为了有更好的画面体验，还要显示爆炸的效果。游戏中，为了简化处理，电脑方坦克的子弹对自己一方的坦克不做碰撞处理。

上述的分析，阐述的是子弹发生碰撞后的效果，那么，对于实现碰撞检测，本质上应该是与坦克的实现策略是一致的。因此要对子弹类进行如下修改：

1、同坦克类，更改子弹 Bullet 类为 SpriteObject 的子类

2、为子弹类增加碰撞检测方法

```
class Bullet(SpriteObject):
    ...已有代码略...
    def __init__(self, tank):
        super(SpriteObject, self).__init__()
            ...已有代码略...

    ...已有代码略...

    def checkTankCollide(self, tankList):  # 检查子弹与坦克是否碰撞上
        for tank in tankList:
            # 此处判断要排除 tank 为 None 情况
            if tank and pygame.sprite.collide_rect(tank, self):
                tank.alive = False
                self.alive = False
```

子弹类的碰撞检测方法的实现逻辑与坦克类的不同，在检测到发生碰撞后，只需要将发生碰撞的坦克对象和子弹自己的状态改变一下，就是设置二者的 alive 属性为 False 即可。alive 属性变更为 False 以后，说明这个对象已经变成无效状态，那么在显示处理环节，就可以将其剔除掉，从而不再显示。

虽然玩家坦克和电脑方坦克发射的子弹是分开进行显示处理的，但处理策略和坦克的类似，需要主程序类中对应显示方法 displayBullets()和 displayComputerBullets()的配合才能在游戏中完整实现子弹碰撞后的效果处理。

子弹与目标发生碰撞后的，都被设置为无效状态，程序的处理策略就是在上述两个方法中，

在遍历处理每一个子弹对象时，需增加状态有效性检查的逻辑，以对不同状态的子弹对象进行分别处理。如果状态有效，则继续显示、移动、碰撞检测的处理；如果状态无效，则将其从保存子弹对象的列表中移除即可。

同样，坦克对象的显示处理方法，也需要继续修改，同子弹对象的处理类似，也增加对象状态性的检查逻辑。状态有效，继续正常处理，若状态无效，对于电脑方坦克，需要从坦克对象列表中移除对应对象，对于玩家坦克，需要将变量置为 None。

综上所述，对主程序类的修改包括：

1、displayBulltes()方法，增加碰撞检测

2、displayComputerBulltes()方法，增加碰撞检测

3、displayPlayerTank()方法，增加状态检测和设置玩家坦克属性为空

4、displayComputerTanks()方法，增加状态检测和电脑方坦克对象的移除

下面列出对主程序类修改的代码片段：

```
# 玩家坦克的显示处理
def displayPlayerTank(self):
    if self.playerTank and self.playerTank.alive:  # 增加有效状态检测
        self.playerTank.display(self.window)
        # 检查坦克对象是否为移动状态
        if self.playerTank and not self.playerTank.stop:
            self.playerTank.move()  # 通过调用 move 方法,实现移动
        # 检查玩家坦克与电脑方坦克的碰撞
        self.playerTank.checkCollide(self.computerTankList)
    elif self.playerTank:
        self.palyerTank = None


# 显示电脑方坦克
def displayComputerTanks(self):
    # 通过循环方式,逐个显示电脑方坦克
    for ct in self.computerTankList[:]:
        # 判断坦克状态是否有效
        if ct.alive:
            ct.display(self.window)
```

```
                ct.randMove()
                bb = ct.shot()
                if bb:
                    self.computerBulletList.append(bb)
                ct.checkCollide([self.playerTank])  # 检测与玩家坦克的碰撞
                ct.checkCollide(self.computerTankList)  # 检测与其他坦克的碰撞
            else:
                self.computerTankList.remove(ct)


    # 显示玩家子弹
    def displayBulltes(self):
        # 通过循环来显示子弹
        for bullet in self.bulletList[:]:
            if bullet.alive:
                bullet.display(self.window)
                bullet.move()
                bullet.checkTankCollide(self.computerTankList) # 增加子弹的碰撞检测
            else:   # 子弹无效则移除
                self.bulletList.remove(bullet)


    # 显示电脑方坦克子弹
    def displayComputerBulltes(self):
        for bullet in self.computerBulletList[:]:
            if bullet.alive:
                bullet.display(self.window)
                bullet.move()
                bullet.checkTankCollide([self.playerTank]) # 增加子弹的碰撞检测
            else:  # 子弹无效则移除
                self.computerBulletList.remove(bullet)
```

2.9.4. 进一步改进

当子弹击中坦克后，画面上如果能够显示一个动态的爆炸效果会使得游戏更生动，逼真。因此，下面来阐述一下如何在程序中添加爆炸效果。

要实现爆炸效果显示，要解决两个问题。

第一、实现爆炸的动画效果。这个可以通过迅速显示一组连续的爆炸效果图片来呈现。

要解决这个问题，需要新增一个爆炸类 Explosion，它用来保存连续爆炸图片，还具有

按顺序显示这些图片的方法。当需要显示爆炸效果时，就创建一个爆炸对象，通过对象的显示方法来实现效果显示。

第二、爆炸效果的显示操作应衔接在子弹与目标碰撞之后。

每当子弹击中坦克，需要呈现爆炸效果时，可以马上创建一个爆炸对象。爆炸对象创建后，可以保存到列表中，当主程序循环刷新画面时，增加对爆炸对象的显示处理，这样就可以在游戏画面中呈现爆炸效果了。

基于上述分析，可增加 Explosion 类如下，该类只有__init__()和 display()两个方法：

```
class Explosion:
    images = [
        pygame.image.load('res/img/explosion-1.png'),
        pygame.image.load('res/img/explosion-2.png'),
        pygame.image.load('res/img/explosion-3.png'),
        pygame.image.load('res/img/explosion-4.png'),
        pygame.image.load('res/img/explosion-5.png')
    ]

    def __init__(self, tank):
        self.frameIndex = 0  # 爆炸图片的序号,初始为 0。-1 表示状态无效
        self.image = Explosion.images[self.frameIndex]
        rect = self.image.get_rect()
        self.rect = tank.rect  # 通过坦克对象的 rect 属性获取起始显示位置
        # 爆炸图片与坦克图片大小不一样,因此起始点坐标要重新算。默认二者的中心点是一样的
        self.rect.left = tank.rect.left - (rect.width - tank.rect.width) / 2
        self.rect.top = tank.rect.top - (rect.height - tank.rect.height) / 2

    def display(self, screen):
        if self.frameIndex < len(self.images):
            screen.blit(self.image, self.rect)
            self.image = self.images[self.frameIndex]
            self.frameIndex += 1
        else:
            self.frameIndex = -1
```

有了 Explosion 类，下面就需要对 Bullet 类的 checkTankCollide()方法进行修改，当子弹与坦克发生碰撞时，需要添加创建爆炸类的语句。修改如下：

```
class Bullet(SpriteObject):

    ...已有代码略...

    def checkTankCollide(self, tankList):  # 检查子弹与坦克是否碰撞上
        result = []
        for tank in tankList:
            if pygame.sprite.collide_rect(tank, self):
                tank.alive = False
                self.alive = False
                result.append(Explosion(tank))
        return result
```

为了方便与主程序类的衔接，该方法会返回一个包含爆炸对象的列表。

下面是主程序类的修改。主要修改内容如下：

1、添加属性 explosionList，用于将新创建的爆炸对象都保存起来

2、添加一个方法 displayExplosion()，用于在程序的主循环中专门处理爆炸类的显示

```
    # 爆炸效果的显示处理
    def displayExplosion(self):
        for explosion in self.explosionList[:]:
            if explosion.frameIndex >= 0:
                explosion.display(self.window)
            else:
                self.explosionList.remove(explosion)
```

3、在主程序中增加上述方法的调用语句

在主程序类的 run()方法中，在循环中添加上述方法的调用，如下：

```
    # 运行游戏
    def run(self):
        pygame.init()  # 初始化 pygame

        ...已有代码略...

        while True:
            # 给窗口设置填充色
            self.window.fill(BG_COLOR)

            self.displayPlayerTank()
            self.displayComputerTanks()  # 显示电脑方坦克
            self.displayBulltes()  # 显示并移动子弹
```

```
            self.displayComputerBulltes()
            self.displayExplosion()  # 添加对处理爆炸显示方法的调用
            self.eventProcess() # 事件处理
            pygame.display.update()
            clock.tick(FPS)
```

4、修改主程序类中处理子弹显示的两个方法，displayBullets()和displayComputerBullets()，增加对爆炸类的处理。

```
    # 显示玩家子弹
    def displayBulltes(self):
        # 通过循环来显示子弹
        for bullet in self.bulletList[:]:
            if bullet.alive:
                bullet.display(self.window)
                bullet.move()
                elist = bullet.checkTankCollide(self.computerTankList) # 修改
                if len(elist)>0:  # 发生碰撞则会返回爆炸对象
                    self.explosionList.extend(elist)  # 添加爆炸对象
            else:   # 子弹无效则移除
                self.bulletList.remove(bullet)


    # 显示电脑方坦克子弹
    def displayComputerBulltes(self):
        for bullet in self.computerBulletList[:]:
            if bullet.alive:
                bullet.display(self.window)
                bullet.move()
                bbullet.checkTankCollide([self.playerTank]) # 增加子弹的碰撞检测
                elist = bullet.checkTankCollide([self.playerTank])
                if len(elist)>0: # 发生碰撞则会返回爆炸对象
                    self.explosionList.extend(elist)  # 添加爆炸对象
            else:  # 子弹无效则移除
                self.computerBulletList.remove(bullet)
```

重新启动程序，此时坦克双方子弹在打到对方坦克时，对方坦克就可以在消失的同时显示爆炸的动态效果了。

2.10. 阶段 6(stage6)最后的完善

到本阶段，游戏的基础性功能就算就绪了。为了使游戏的整体效果更好，增加易用性、可玩性，很多方面还可以进一步改进、完善。

2.10.1. 增加音效，如玩家坦克出现时的声音、子弹发射声音、爆炸声音

pygame 中有混合器模块，提供了对背景声音播放的支持。本游戏也利用该功能来实现自己的背景音效。

pygame 中有两种方式来播放声音，如下所示：

方法 1 通过创建 pygame.mixer.Sound 对象，然后利用该对象的 play()方法来播放。该方法可创多个 Sound 对象，各对象可同时播放。

```
pygame.mixer.init()
mymusic = pygame.mixer.Sound('res/start.wav')
mymusic.play()
```

方法 2 使用 pygame.mixer.music.load()方法装载声音文件，然后使用 music 模块中的 pygame.mixer.music.play()方法播放。如果多次 load()不同的声音文件，每次播放都是最新装载的那个文件。不支持同时播放。

```
pygame.mixer.init()
pygame.mixer.music.load('res/start.wav')
pygame.mixer.music.play()
```

上述两种方法都有一个前提，就是要先执行 pygame.mixer.init()，对 pygame 的混合器进行初始化，这个在使用中需注意。

本游戏比较简单，两种方式均可使用，虽然使用方法有差异，但效果基本一致。游戏中选择第一种方式，主要实现三种效果：玩家坦克出现时的效果音，玩家坦克发射子弹时的效果音以及爆炸时的声音。

玩家坦克出现以及玩家发射子弹的操作都在主程序类中，因此直接在该类中修改。主程序类增加两个属性，保存两种效果音的 Sound 对象，然后在初始化玩家坦克 initPlayerTank()

方法，和玩家发射子弹的事件处理代码处分别添加音效播放语句即可。

修改代码片段如下：

```
class TankGame:
    def __init__(self):

        ...已有代码略...

        self.playerBGM = None  # 添加的背景音效属性
        self.fireBGM = None

    # 运行游戏
    def run(self):
        pygame.init()  # 初始化 pygame
        pygame.mixer.init()

        self.playerBGM = pygame.mixer.Sound('res/start.wav')
        self.fireBGM = pygame.mixer.Sound('res/fire.wav')

        ...已有代码略...


    ...已有代码略...

    def eventProcess(self):

        ...已有代码略...

                    if PLAYER_SHOT_LIMIT < 0 or \ # PLAYER_SHOT_LIMIT 为负表示不限制
                            len(self.bulletList) < PLAYER_SHOT_LIMIT:
                        # 创建子弹
                        self.bulletList.append(self.playerTank.shot())
                        self.fireBGM.play()  # 添加播放背景音语句

        ...已有代码略...


    ...已有代码略...

    # 初始化玩家坦克的方法
    def initPlayerTank(self):
        self.playerTank = PlayerTank(400, 540)
        self.playerBGM.play()# 添加播放背景音语句
```

```
...已有代码略...
```

2.10.2. 玩家可以复活，电脑方坦克数量可以进行配置

目前游戏中电脑方的坦克数量是在代码中设定好的，如果要调整的话非常不方便。可以通过配置文件，在程序外部进行预先设定，这样程序启动时，就会根据设定来显示电脑方的坦克数。

再有，为了增加可玩性，还应该允许玩家可以有不止一条命，这样在坦克被对方击中后，还能够接着玩下去。玩家能够有多少条命，也可以通过配置文件来进行设置。

最后，有了上述配置后，程序本身要做适当修改，一是支持使用上述配置项，二是增加对玩家复活功能的支持。

要增加的配置项有：

```
# 画面上同时显示的电脑方坦克数
COMPUTER_TANKS = 5
# 电脑方总共的坦克数
TOTAL_COMPUTER_TANKS = 20
#  玩家坦克的生命数
PLAYER_LIFE = 3
```

游戏对于电脑方坦克数量的配置有两个常量，一个是 COMPUTER_TANKS，另一个是 TOTAL_COMPUTER_TANKS。基于这两个配置常量，电脑方坦克在游戏中的策略是这样的：

1、为了画面不过于拥挤，需要设置同时最多显示几个电脑方坦克。这个由 COMPUTER_TANKS 来设定

2、允许设置多于实际显示数量的电脑方坦克综述，这个由 TOTAL_COMPUTER_TANKS 来设置。

3、随着游戏的进行，电脑方坦克会减少，所以 TOTAL_COMPUTER_TANKS 仅是初始时的设置值，当前实际值还需要主程序类增加一个属性 computerTankCount 来进行记录

游戏启动后，电脑方坦克的初始化在主程序类中完成，即通过 initComputerTanks()。该方法的参数即为要创建的坦克数，现在这个方法的实参可以通过配置项来设置，因此调用语句需要进行修改。

有了对当前电脑方坦克数量的记录，那么，在程序运行过程中，这个计数应该与实际情况同步。当电脑方坦克被击中，这个计数需要同步减一操作。另外如果画面中同时显示的坦克数小于当前的总数，还需要重新创建一个新的电脑方坦克对象，以添加到 computerTankList 中，这样才可以维持画面上显示的坦克数。这个需要对电脑方坦克的显示方法 displayComputerTanks()进行修改。

综上，主程序的__init__()、run()和 displayComputerTanks()修改如下：

```
class TankGame:
    def __init__(self):

        ...已有代码略...

        # 默认电脑方坦克的初始总数
        self.computerTankCount = TOTAL_COMPUTER_TANKS

    ...已有代码略...

    # 运行游戏
    def run(self):

        ...已有代码略...
        self.initPlayerTank()
        # 修改初始化电脑方坦克
        self.initComputerTanks(COMPUTER_TANKS \
                               if self.computerTankCount > COMPUTER_TANKS \
                               else self.computerTankCount)
        clock = pygame.time.Clock()

        ...已有代码略...

        while True:
```

```
            ...已有代码略...


    ...已有代码略...

    def displayComputerTanks(self):
        # 通过循环方式,逐个显示电脑方坦克
        for ct in self.computerTankList[:]:
            # 判断坦克是否还活着
            if ct.alive:

                ...已有代码略...

            else:
                self.computerTankList.remove(ct)
                self.computerTankCount -= 1
                # 总数还多于可同时显示的数量
                if COMPUTER_TANKS <= self.computerTankCount:
                    self.initComputerTanks(1)  # 补充一个新坦克
```

玩家复活功能的规划：主程序类增加一个属性 playerLife 用来记录玩家的生命数，其初始值等于上述的配置项 PLAYER_LIFE。同样，这个属性值也要与游戏运行的实际情况相符，即，玩家坦克被消灭后，生命数需减一，同时记录玩家坦克对象的属性 playerTank 需置为 None。这个操作可以在玩家坦克的显示方法中 displayPlayerTank()实现。

playerLife 的值大于 0，玩家就可以进行复活操作。这需要通过修改事件处理的代码，让玩家来触发。现设定玩家按下 ESC 键时，如果 playerTank 为 None 值，且生命值 playerLife 大于 0，那么就可以再次创建一个玩家坦克对象，从而实现复活。

通过上面的分析，代码的修改还是在主程序对象中。涉及__init__()、displayPlayerTank()和 eventProcess()。

如下为修改的代码：

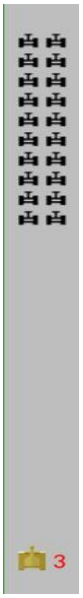
```
class TankGame:
    def __init__(self):

        ...已有代码略...
```

```
    # 默认玩家生命初始值
    self.playerLife = PLAYER_LIFE

...已有代码略...

# 玩家坦克的显示处理
def displayPlayerTank(self):
    if self.playerTank and self.playerTank.alive:

        ...已有代码略...

    elif self.playerTank:
        self.playerTank = None
        self.playerLife -= 1  # 添加,生命值减一操作

...已有代码略...

def eventProcess(self):
    # 获取事件,开始事件处理
    eventList = pygame.event.get()
    for event in eventList:
        # 判断按下的是否键
        if event.type == pygame.QUIT:
            self.exit()
        # 按键事件的处理
        if event.type == pygame.KEYDOWN:
            if self.playerTank:

                ...已有代码略...

            elif event.key == pygame.K_ESCAPE:  # 新增 ESC 按键事件的处理
                # 玩家生命值不为零
                if self.playerTank is None and self.playerLife > 0:
                    self.initPlayerTank()  # 初始化玩家

    ...已有代码略...
```

2.10.3. 增加右侧显示面板

经过上面的修改，现在游戏中可以有对于画面显示数量的电脑方坦克，玩家也有多条命。

为了能更好的将这些信息呈现在界面上，可以在右侧添加一个面板来显示玩家当前的生命数以及电脑方的总坦克数。

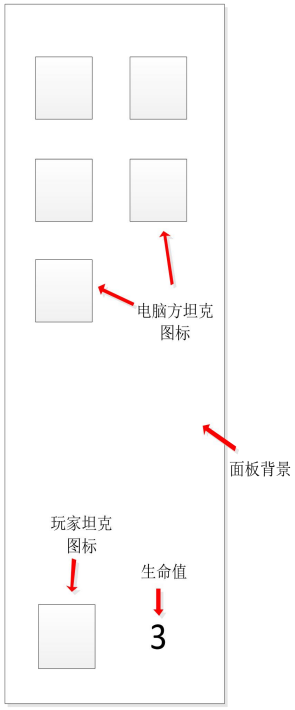


面板的设计如右图所示：

上面大部分面积使用坦克图标进行填充，每个图标对应一个坦克，以此显示电脑方坦克的数量。当电脑方坦克被击中时，面板上的图标也会相应减少。

下边有一个表示玩家的图标，以及一个表示剩余生命数的数字。

面板的显示的规划如左图所示，由面板、图标、文字等元素堆叠指定位置同时显示。



面板是一个 pygame.Surface 对象，根据配置中窗口大小、面板宽度的设置，计算该对象的初始位置以及面板的高度。

有了面板的尺寸，电脑方的每一个图标，都会通过计算得到各自的显示坐标位置。然后在按位置显示出来。图标间距是在代码中写死的。玩家图标类似处理类似。

玩家生命值是 pygame.font.Font 经渲染后的 Surface 对象，也是通过计算得到排放位置。

为了方便处理，在程序中将上述内容都封装在一个面板类中，统一由面板对象来处理面板的显示。实现细节见如下代码：

```
class Panel:
    # 右边条面板的宽度和高度
    PANELWIDTH, PANELHEIGHT = setting.PANELWIDTH, setting.SCREEN_SIZE[1]
    # 右边条面的左上角点
    PANELLEFT, PANELTOP, = setting.SCREEN_SIZE[0], 0

    # 设定显示位置
    def __init__(self, playerCount=setting.PLAYER_LIFE, totalTanks=setting.TOTAL_COMP
UTER_TANKS):
        # 准备控制面板的 Surface
```

```
        self.panel = pygame.Surface((Panel.PANELWIDTH, Panel.PANELHEIGHT), flags=pyga
me.HWSURFACE)
        # 填充方块显色
        self.panel.fill((190, 190, 190))
        self.panelpoint = (Panel.PANELLEFT, Panel.PANELTOP)
        # 最多显示二十个机器方坦克图标
        self.tank_iconimg = pygame.image.load('res/img/tank_icon.png')
        self.tankicon_point = (self.PANELLEFT + 10, 30)
        # 玩家的图标
        self.player_icon = pygame.image.load('res/img/player_icon.png')
        self.player_point = (Panel.PANELLEFT + 10, Panel.PANELHEIGHT - 50)
        self.icon_points = []
        self.font = pygame.font.Font('c:/Windows/Fonts/simhei.ttf', 20)
        self.player_count = playerCount
        self.player_count_text = self.font.render(str(self.player_count), True, (255,
 0, 0), (190, 190, 190))
        # 获得显示对象的方块(rect)区域坐标
        self.text_position = (Panel.PANELLEFT + 35, Panel.PANELHEIGHT - 45)
        self.reset(totalTanks)

    def display(self, screen):
        screen.blit(self.panel, self.panelpoint)
        for i in range(len(self.icon_points) // 2):
            screen.blit(self.tank_iconimg, self.icon_points[2 * i])
            screen.blit(self.tank_iconimg, self.icon_points[2 * i + 1])
        if len(self.icon_points) > 0 and len(self.icon_points) % 2:
            screen.blit(self.tank_iconimg, self.icon_points[len(self.icon_points)-1])
        screen.blit(self.player_icon, self.player_point)
        screen.blit(self.player_count_text, self.text_position)

    # 初始化坦克图标的定位坐标列表
    def reset(self, total):
        row = total // 2 + total % 2
        for i in range(row):
            # self.tankicon_point 是坦克图标初始显示位置的坐标元组
            self.icon_points.append((self.tankicon_point[0], self.tankicon_point[1] +
 20 * i))
            self.icon_points.append((self.tankicon_point[0] + 20, self.tankicon_point
[1] + 20 * i))
        if total % 2:
            self.icon_points.pop()

    # 减少坦克图标数
    def cutdown(self, n):
```

```
        for i in range(n):
            self.icon_points.pop()

    def setPlayer(self, n):
        self.player_count = n
        self.player_count_text = self.font.render(str(self.player_count), True, (255,
0, 0), (190, 190, 190))
```

有了面板类后，需要在主程序类中实例化面板对象，并添加到程序的整体绘制流程中，这样就可以把面板显示出来了。另外，面板中显示的内容与玩家生命值、电脑方坦克总数相关联，需要在主程序的运行过程中进行面板显示内容同步更新的操作。

下面阐述一下如何在主程序类中将面板衔接进来。

首先在主程序类中增加一个属性 panel，用来保存 Panel 对象。在 run()方法中创建 Panel 对象。由于添加了面板，所以，要把整个窗口的大小重做调整，以留出地方放面板。

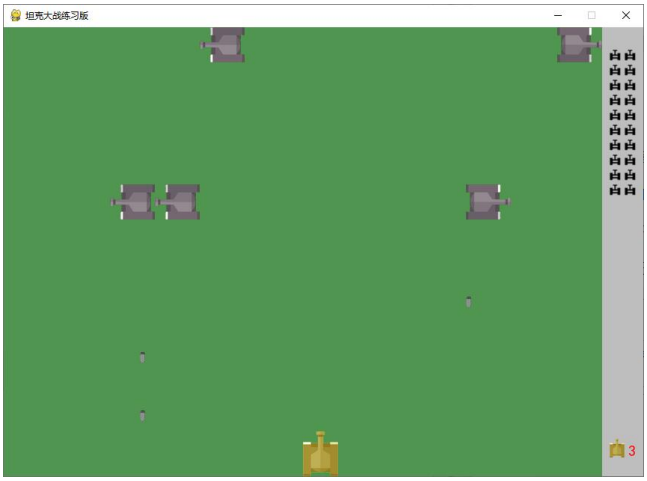
```
class TankGame:

    def __init__(self):
        self.panel = None

        ...已有代码略...

    # 运行游戏
    def run(self):

        ...已有代码略...
        self.fireBGM = pygame.mixer.Sound('res/fire.wav')

        # 设置主窗口 Surface 对象的大小
        self.window = pygame.display.set_mode((SCREEN_SIZE[0] + PANELWIDTH,
                                               SCREEN_SIZE[1]))
        self.panel = Panel() # 创建面板对象
        pygame.display.set_caption("坦克大战练习版")

        ...已有代码略...

        while True:

            ...已有代码略...
```

```
        self.displayExplosion()
        self.panel.display(self.window)  # 显示面板
        self.eventProcess()
        pygame.display.update()
        clock.tick(FPS)

...已有代码略...
```

以上代码就可以实现面板的显示了。不过内容同步还没实现，需要在主程序类的玩家坦克、电脑方坦克显示方法中增加同步内容的语句，修改后代码片段如下：

```
# 玩家坦克的显示处理
def displayPlayerTank(self):
    if self.playerTank and self.playerTank.alive:
        self.playerTank.display(self.window)
        if self.playerTank and not self.playerTank.stop:  # 检查移动状态
            self.playerTank.move()  # 通过调用 move 方法,实现移动
        self.playerTank.checkCollide(self.computerTankList) #检查与电脑方坦克的碰撞
    elif self.playerTank:
        self.playerTank = None
        self.playerLife -= 1
        self.panel.setPlayer(self.playerLife)

# 显示电脑方坦克
def displayComputerTanks(self):
    # 通过循环方式,逐个显示电脑方坦克
    for ct in self.computerTankList[:]:
        # 判断坦克是否还活着
        if ct.alive:
            ct.display(self.window)
            ct.randMove()
            bullet = ct.shot()
            if bullet:
                self.computerBulletList.append(bullet)
            ct.checkCollide([self.playerTank])  # 检测与玩家坦克的碰撞
            ct.checkCollide(self.computerTankList)  # 检测与其他坦克的碰撞
        else:
            self.computerTankList.remove(ct)
            self.panel.cutdown(1)
            self.computerTankCount -= 1
            if COMPUTER_TANKS <= self.computerTankCount:  # 总数多于可同时显示的数量
                self.initComputerTanks(1)  # 补充一个新坦克
```

至此，面板的添加修改完成。显示效果如下：



2.10.4. 游戏画面中可以添加一些障碍

在游戏中添加障碍，与在游戏中添加坦克、子弹的实现方法类似，不过障碍是静止的，因此无需去处理移动的问题。

障碍的显示和处理，主要考虑如下几个问题：

1）障碍的属性是怎样的。显示的位置、大小、样子是怎样的？

2）如果障碍存在，双方坦克在碰到障碍时是停止还是可以穿过？

3）双方坦克在射击时，障碍是否可以被击毁？

本练习中，为了简化处理，仅实现一种类型的障碍，即砖墙。大小为 60*60，在游戏中设定为可以被击毁，但需要多次射击，这个次数可以进行设定；另外，作为障碍物，双方坦克在碰到墙时，应停止前进，而不是穿过。以上就是游戏中对所增加的障碍的逻辑设定。

下面首先要定义障碍类 wall，代码如下：

```
class Wall:
    def __init__(self, left, top):
        self.image = pygame.image.load('res/img/wall.png')
        self.rect = self.image.get_rect()
        self.rect.left = left
        self.rect.top = top
        # 设置是否有效标志
```

```
        self.alive = True
        # 设置墙的生命值
        self.life = WALL_LIFE  # 来自 setting 的静态变量

    # 显示方法
    def display(self, screen):
        screen.blit(self.image, self.rect)
```

有了障碍类的定义，下面就可以在主程序类中添加墙初始化、显示墙的方法，代码如下：

```
    # 初始化墙。暂设定在画面水平方向居中显示五个
    def initWalls(self):
        for i in range(1, 6):
            wall = Block(130 * i, 240)
            self.wallList.append(wall)

    # 显示墙
    def displayWalls(self):
        for wall in self.wallList[:]:
            if wall.alive:
                wall.display(self.window)
            else:
                self.wallList.remove(wall)
```

添加好上述方法后，修改主程序如下：

```
    # 运行游戏
    def run(self):
        pygame.init()  # 初始化 pygame

        ...已有代码略...

        while True:
            # 给窗口设置填充色
            self.window.fill(BG_COLOR)
            self.displayWalls()  # 显示墙
            self.displayPlayerTank()
            self.displayComputerTanks()  # 显示电脑方坦克
            self.displayBulltes()  # 显示并移动子弹
            self.displayComputerBulltes()
            self.displayExplosion()  # 添加对处理爆炸显示方法的调用
            self.panel.display(self.window)  # 显示面板
            self.eventProcess() # 事件处理
            pygame.display.update()
            clock.tick(FPS)
```

至此，画面中墙的显示就有了。不过，要实现墙的阻挡效果，还有射击几次后墙可以消失，

还需要添加墙的碰撞检测。坦克类中有 checkCollide()方法来实现碰撞检测，对于墙的检测也同样可以使用。

```
    def displayPlayerTank(self):
        if self.playerTank and self.playerTank.alive:
            ...已有代码略...
            self.playerTank.checkCollide(self.wallList)  # 添加检测墙的碰撞
            self.playerTank.checkCollide(self.computerTankList)  # 检查电脑方坦克的碰撞
        elif self.playerTank:
            ...已有代码略...

    def displayComputerTanks(self):
        # 通过循环方式,逐个显示机方坦克
        for ct in self.computerTankList[:]:
            # 判断坦克是否还活着
            if ct.alive:
                ct.display(self.window)
                ct.randMove()
                ct.checkCollide(self.wallList)  # 添加检测墙的碰撞
                bullet = ct.shot()
                if bullet:
                    self.computerBulletList.append(bullet)
                ct.checkCollide([self.playerTank])  # 检测与玩家坦克的碰撞
                ct.checkCollide(self.computerTankList)  # 检测与其他坦克的碰撞
            else:
            ...已有代码略...
```

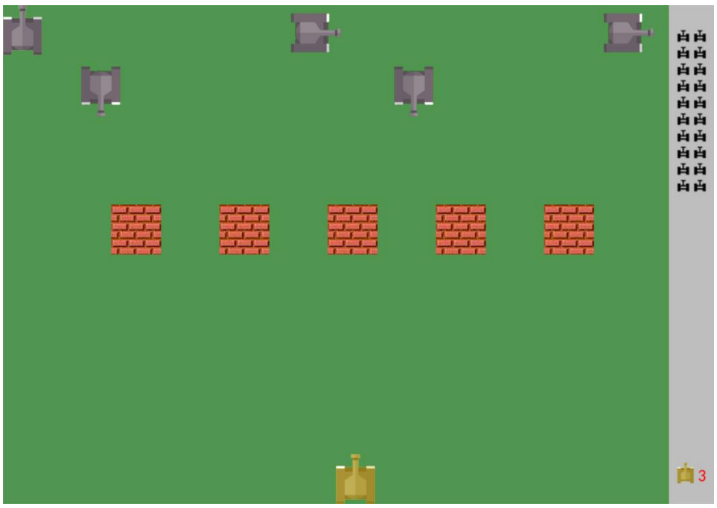
子弹打到障碍上时，也要有相应处理，因此，子弹类要增加与墙的碰撞检测方法。代码如下：

```
    def checkWallCollide(self, wallList):
        for wall in wallList:  # 遍历全部墙
            if pygame.sprite.collide_rect(wall, self):
                self.alive = False   # 发生碰撞,就设置子弹为无效状态
                wall.life -= 1  # 墙的生命值减一
                if wall.life <= 0:
                    wall.alive = False  # 墙的生命小于零时,也被设置为无效状态
```

子弹类增加了上述方法后，玩家坦克和电脑方坦克的子弹就都可以射击墙，并累积一定次数后，墙就被摧毁。要实现这个效果，还需要在主程序类中子弹显示处理方法内，添加子弹与墙的碰撞检测才行。修改的代码如下：

```
    # 显示玩家子弹
    def displayBulltes(self):
        # 通过循环来显示子弹
        for bullet in self.bulletList[:]:
            if bullet.alive:
                bullet.display(self.window)
                bullet.move()
                bullet.checkWallCollide(self.wallList)  # 添加与墙的碰撞检测
                elist = bullet.checkTankCollide(self.computerTankList)
                if len(elist) > 0:  # 有爆炸对象,就添加到主程序的列表中
                    self.explosionList.extend(elist)
            else:  # 子弹飞出窗口则移除
                self.bulletList.remove(bullet)

    # 显示机方坦克子弹
    def displayComputerBulltes(self):
        for bullet in self.computerBulletList[:]:
            if bullet.alive:
                bullet.display(self.window)
                bullet.move()
                bullet.checkWallCollide(self.wallList)  # 添加与墙的碰撞检测
                elist = bullet.checkTankCollide([self.playerTank])
                if len(elist) > 0:
                    self.explosionList.extend(elist)
            else:  # 子弹飞出窗口则移除
                self.computerBulletList.remove(bullet)
```

至此，障碍的添加就算全部完成了。显示效果如下：



2.10.5. 增加游戏终结的处理

当电脑方坦克总数被清零，或者玩家生命值被清零时，游戏就宣告结束了。此时，游戏画

面应该切换为游戏终结画面，并且提示玩家是重新开始游戏，还是退出游戏。玩家可以通过界面提示，进行按键选择操作。

游戏终结画面的显示与面板类似，但在显示终结画面时，要捕获用户按键输入，此时游戏已经结束，进入到终结画面的显示流程中，因此，主程序的流程会有一个调整。另外，如果玩家选择重新游戏，此时需要回到原来的主流程中，并且需要重置游戏为初识状态。

玩家失败是的终结画面如下：

游戏终结处理的基本策略如下：

1、主程序类中增加一个方法 gameover()，该方法会判断游戏是否符合结束条件，即本节开始所说的是否有一方被清零。如果条件不符合，返回 False，主流程继续，否则进入另一个新增方法中以显示游戏终结画面，等待用户的操作选择。

2、主程序类中再增加一个方法 restartGame()，该方法在 gameover()方法中被调用，有如下几个职责：

1)显示游戏终结画面，并提示玩家进行选择：退出还是重玩

2)启动另一个子循环流程进行画面显示与事件处理，等待用户输入

3)基于用户的输入返回 True 或 False

3、主程序类做代码重构，增加一个 init()方法，用于对游戏进行重置操作

按照如上思路，主程序类的 run()方法要修改，主循环添加判断语句，需要添加 gameover()、restartGame()、init()三个方法，修改__init__()方法。

首先进行重构，修改__init__()方法，添加 init()方法，代码片段如下：

```
class TankGame:

    def __init__(self):
        self.window = None
        self.playerBGM = None
        self.fireBGM = None
        self.init()

    def init(self):
        self.panel = None
        # 玩家坦克
        self.playerTank = None
        self.playerLife = PLAYER_LIFE
        # 电脑方坦克
        self.computerTankList = []
        # 默认电脑方坦克数量
        self.computerTankCount = TOTAL_COMPUTER_TANKS
        # 玩家发射的子弹对象列表
        self.bulletList = []
        # 电脑方坦克发射的子弹对象列表
        self.computerBulletList = []
        # 爆炸对象列表
        self.explosionList = []
        # 全部墙对象的列表
        self.wallList = []

    ...已有代码略...
```

下面是 run()方法的修改：

```
    def startGame(self):

        ...已有代码略...
```

```
        while True:

            ...已有代码略...

            self.panel.display(self.window)
            if self.gameover():  # 终结画面入口,若返回 True 则退出主流程,游戏结束
                break
            pygame.display.update()
            clock.tick(FPS)

        self.endGame()
```

新增 gameover()方法如下，首先进行游戏是否符合结束条件的判断，不符合，直接返回 False。符合了才继续执行进入终结画面。

```
    # 游戏结束的处理
    def gameover(self):
        if (self.playerTank is None and self.playerLife == 0) or \
                (len(self.computerTankList) == 0 and self.computerTankCount == 0):
            if self.restartGame():  # 用户在终结画面是否选择重玩
                self.init()
                self.panel = Panel()
                self.initComputerTanks(
                    COMPUTER_TANKS if self.computerTankCount > COMPUTER_TANKS
                    else self.computerTankCount)
                self.initWalls()
                self.initPlayerTank()
                return False
            return True
        return False
```

restartGame()方法要建立一个新的工作循环，与主程序流程一样，通过对窗口画面的绘制，将终结画面显示出来，同时启动事件处理，以检测用户的按键输入。

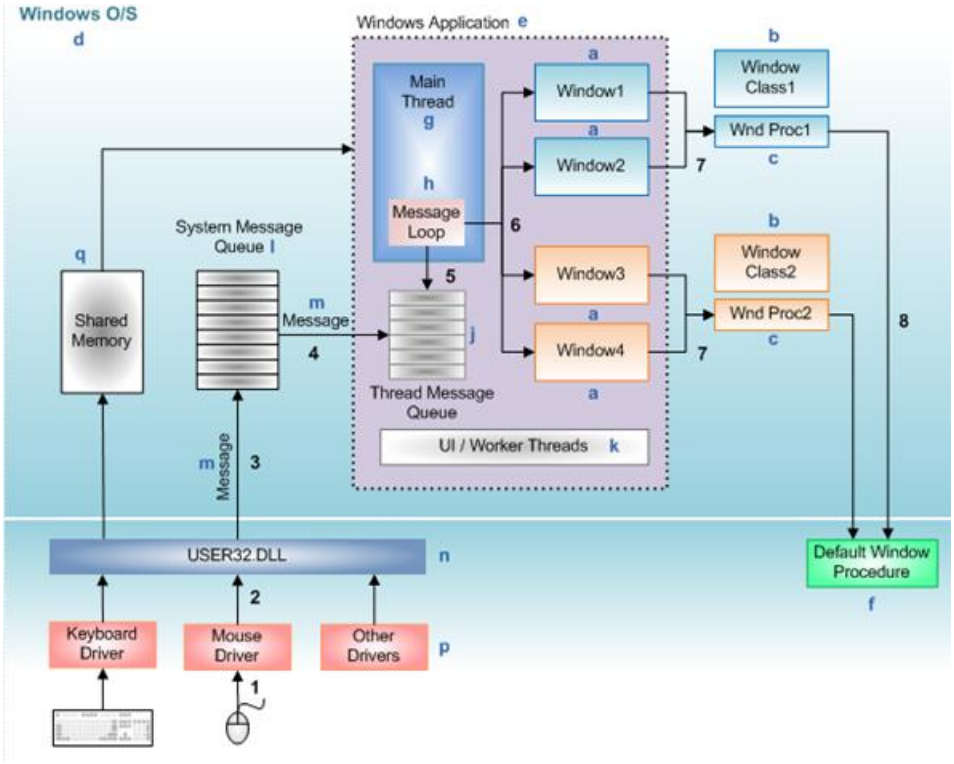
完整的代码如下：

```
    # 是否重启游戏的处理
    def restartGame(self):
        if self.playerLife == 0:
            over_image = pygame.image.load('res/img/gameover.gif')
        else:
            over_image = pygame.image.load('res/img/youwin.gif')
        over_image_rect = over_image.get_rect()
```

```
        over_image_rect.left = SCREEN_SIZE[0] / 2 - over_image_rect.width / 2
        over_image_rect.top = SCREEN_SIZE[1] / 2 - over_image_rect.height
        # 字体显示
        font = pygame.font.SysFont('SimHei', 25)
        text1_render = font.render('Press <R> to restart the game.',
                                   True, (85, 65, 0))
        text2_render = font.render('Press <Esc> to quit the game.',
                                   True, (85, 65, 0))
        text1_rect = text1_render.get_rect()
        text2_rect = text2_render.get_rect()
        text1_rect.left, text1_rect.top = (
            SCREEN_SIZE[0] / 2 - text1_rect.width / 2,
            over_image_rect.top + over_image_rect.height)
        text2_rect.left, text2_rect.top = (
            SCREEN_SIZE[0] / 2 - text2_rect.width / 2,
            text1_rect.top + 30)

        while True:
            self.window.fill(BG_COLOR)
            self.window.blits(((over_image, over_image_rect),
                               (text1_render, text1_rect),
                               (text2_render, text2_rect)))
            # 获取事件
            eventList = pygame.event.get()
            # 遍历获取的事件
            for event in eventList:
                # 判断按下的是否键
                if event.type == pygame.QUIT:
                    self.exit()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_ESCAPE:
                        return False
                    elif event.key == pygame.K_r:
                        return True
            pygame.display.update()
```

附录　Windows 操作系统消息机制极简说明

Windows 操作系统中，消息机制框架原理如下图所示：

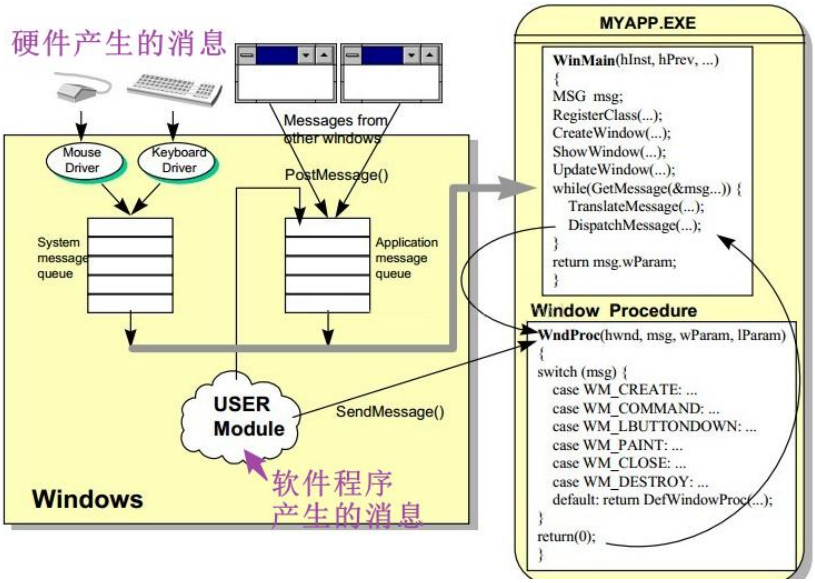


Windows 应用程序中的窗口和窗口类

Windows 窗口应用程序（e）具有一个主线程（g）、一个或多个窗口（a）和一个或多个子线程（k）[工作线程或 UI 线程]。

应用程序必须指定窗口类并向 Windows 系统（d）注册，然后才能创建窗口（a）并显示。窗口类是一种包含窗口属性的结构，例如窗口样式，图标，光标，背景颜色，菜单资源名称和窗口类名称等。注册窗口类会将窗口过程、类样式和其他类属性与类名相关联。

每个窗口类都有一个关联的窗口过程（c），该过程由应用程序中同一类的所有窗口（a）共享。窗口过程处理该类的所有窗口的消息。



窗口过程(DefWindowProc 也叫 DefaultWindowProcedure)..

窗口过程（c）是接收和处理发送到窗口（a）的所有消息的函数。每个窗口类（b）都有一个关联的窗口过程（c），并且使用该类创建的每个窗口（a）都使用相同的窗口过程来响应消息。系统通过将消息数据作为参数传递给过程，将消息发送到窗口过程。然后，窗口过程对消息执行适当的操作。它检查消息标识符，并在处理消息时使用消息参数指定的信息。

窗口过程通常不会忽略消息。如果它不处理消息，则必须将消息发送回系统以进行默认处理。窗口过程通过调用 DefWindowProc 函数（f）来执行此操作，该函数执行默认操作并返回消息结果。默认窗口过程函数 DefWindowProc 定义了所有窗口共享的某些基本行为。默认窗口过程为窗口提供最少的功能。

Windows 消息(Message)

Windows 消息(m)只不过是一组参数，例如窗口句柄，消息标识符和两个称为消息参数的值(WPARAM 和 LPARAM)。窗口句柄标识消息所针对的窗口。消息标识符是标识消息用途的常量。例如，消息标识符 WM_PAINT 告诉窗口过程窗口的工作区已更改，必须重新绘制。

有两种类型的 Windows 消息，如系统定义的消息和应用程序定义的消息。系统在与应用程序通信时发送或发布系统定义的消息。应用程序还可以发送或发布系统定义的消息。例如，

WM_PAINT 消息标识符用于系统定义的消息，该消息请求窗口绘制其内容。

应用程序可以创建由其自己的窗口使用的消息，或与其他进程中的窗口进行通信。如果应用程序创建自己的消息，则接收消息的窗口过程必须解释消息并提供适当的处理。

消息的流转

基于 DOS 的应用程序通过 C 运行时(Runtime)函数调用，从操作系统获取用户输入。例如，它调用 gets() C-Runtime 函数从键盘获取输入。但是，在 Windows 操作系统中，窗口应用程序不会通过显式函数调用来获取输入。相反，它们等待系统将输入传递给它们。这就是在 Windows 操作系统环境下，为什么窗口应用程序工作方式采用事件驱动的原因。

每当用户移动鼠标时，单击 [1] 键盘上的鼠标按钮或类型，鼠标的设备驱动程序 (p) 或键盘将详细信息切换 [2] 给 User32.dll (n)，后者又将输入转换为 Windows 消息，并将其放在系统消息队列 (1) 中[3]。

Windows 操作系统从系统消息队列中一次删除一条消息，检查它们以确定目标窗口，然后将它们发布 [4] 到创建目标窗口的线程的消息队列中。线程的消息队列接收线程创建的窗口的所有鼠标和键盘消息。线程从其队列中删除 [5] 消息，并指示系统将它们发送到 [6 和 7] 到相应的窗口过程进行处理。线程如何从线程的消息队列和进程中删除消息？现在让我们讨论一下。

消息循环

Windows 应用程序的入口点是 WinMain()函数，就像"C"语言程序中的 main()函数一样。WinMain()函数充当应用程序的主线程函数。WinMain()函数中的以下几行代码就建立起了消息循环。

```
int WINAPI WinMain (HINSTANCE hInst,
                    HINSTANCE hPrevInst,
                    char * cmdParam,
```

```
                    int cmdShow)
{
    while (GetMessage(&Msg, NULL, 0, 0))
    //消息的接收主要有 3 个函数：GetMessage、PeekMessage、WaitMessage。
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
}
```

上述循环中调用 GetMessage()函数检查消息队列中的 Windows 消息。如果找到有效消息，则会从其队列中删除消息。从队列中删除消息后，应用程序可以使用 DispatchMessage()函数指示系统将消息发送到窗口过程进行处理，如果是应用无法处理的消息就发送到 windows 系统让其调用 DefaultWindowProcedure()函数处理。如果消息队列中没有消息，则 GetMessage()调用将被阻止，直到有有效的消息进入消息队列。

在这两个调用之间，调用了 TranslateMessage()函数但不执行任何操作，而只是将虚拟关键消息转换为字符消息。如果注释此行，应用程序仍可正常工作。唯一的问题是，会产生与键盘相关的错误提示。