Python 程序设计文档

郭竞兴

一、设计内容

C10 类：DIY 街机游戏；设计一款有丰富游戏体验的街机游戏，容易上手而且有一定趣味，此次设计的为从天而降下来的秤砣然后把水果（香蕉）砸扁，玩家用电脑控制水果来躲避其砸扁。

二、设计工具

基于 Python 的 Pygame 模块：利用 Pygame 框架编写程序；
Visual Studio Code 和 Python 3.7.0 的 IDLE 编辑器：编写程序代码；
游戏音效合成工具 - Bfxr：设计游戏音效，导出 wav 文件；
格式工厂：将仅支持部分平台的 mp3 文件转换成支持所有平台的 ogg 文件；
Adobe Photoshop CC 2017：对游戏的绘图进行修正和美化。

三、设计步骤

1. 设计程序框图

对于这样根据鼠标来操纵的游戏，首先应该确定如何对这个游戏的各个角色（该游戏中为两个角色：香蕉和 16 吨的秤砣）以及角色的属性、两个角色的行进方向等进行详细的规划。

对游戏的总体功能有如下的几个基本规划：

①游戏中有一个香蕉、还有一个 16 吨的秤砣，秤砣匀速从天而降，碰到玩家能实现游戏的结束，再次单击初始化所有参数；

②游戏中需要由多个关卡，通过每关只需要秤砣掉下 10 次而且香蕉不碰到即可，而且难度每一次升级，每次关卡秤砣掉落速度都加上初始数据-1。

③游戏中为了提高玩家的游戏体验，应该在撞击的时候添加声音，在游戏结束时播放游戏结束音乐，游戏全程有背景音乐。

2. 规划程序框架

游戏模块的导入，对于主程序需要导入的所有模块，代码如下：

```
import os, sys, pygame
from pygame.locals import *
import objects, config
```

游戏程序分为三大模块：主程序（squish.py）、游戏精灵（objects.py）和游戏配置（config.py）。

对于每一个模块的规划如下：

①主程序（squish.py）

该部分控制游戏的动作，游戏的初始化，游戏的继续等各种游戏控制。根据游戏的流程，需要建立一个游戏类（Game），以及一些状态类，对游戏进行整体控制。

状态类：

①class State

范型游戏状态类，可以处理时间并在给定的表面上显示自身。

由于是控制游戏状态，因此需要获取键盘的参数来控制游戏的进行和退出，因此需要一个方法 handle 传入 event 参数处理退出时间的默认事件处理需要获取到一些键盘参数 KEYDOWN 中的 K_ESCAPE 和关闭窗口参数 QUIT。

游戏刚开始需要设计一个背景色填充屏幕，用于第一次显示状态。使用背景色填充屏幕，将游戏默认的颜色改为现在的天蓝色(0, 155, 155)，更符合游戏当时的场景。

②class Level(State)

游戏等级。用于计算已经落下了多少秤砣，移动子图形以及其他和游戏逻辑相关的任务。游戏中每一关的初始化都靠该类来完成，包括对秤砣和香蕉的初始化，因此在该类中应该对秤砣掉下来的次数进行计数，计满了 10 个数之后自动过关，然后提高掉落速度让难度升级。通过 update 可以更新从前一帧开始的游戏的状态，如果香蕉碰到了秤砣，那么告诉游戏切换到 GameOver 状态，否则在秤砣落地时将其复位。如果在本馆内所有秤砣都落下了，则让游戏切换到 LevelCleared 状态。

Display 方法在第一次显示（只清空屏幕）后显示状态。与 firstDisplay 不同，这个方法使用 pygame.display.update 对 self.sprites.draw 提供的、需要更新的矩形列表进行更新。

③class Paused(State)

简单的暂停游戏状态，按下键盘上任意键或者点击鼠标都会结束这个状态。

该方法使得游戏进程暂停，因此需要接受来自键盘或鼠标的任意键。Handle 方法通过对 State 进行委托（一般处理退出事件）以及对按键和鼠标点击最、作为反应来处理事件。如果键被按下或者鼠标被点击，将 self.finished 设定为真。而 update 方法更新等级。如果按键被按下或者鼠标被点击（比如 self.finished 为真），那么告诉我们切换到下一个由 self.nextState()表示的状态（应该由子类实现）。最后通过 firstDisplay 方法暂停状态第一次出现时，绘制图像（如果有的话）并且生成文本。

④class Info(Paused)

简单的暂停状态，显示有关游戏的信息。在 Level 状态后显示（第一级）。

⑤class StartUp(Paused)

显示展示图片和欢迎信息的暂停状态。在 Info 状态后显示。

⑥class LevelCleared(Paused)

提示用户过关的暂停状态。在 next level 状态后显示。

⑦class GameOver(Paused)

提示用户输掉游戏的状态。在 first level 后显示。

⑧class Game——游戏主类

负责主事件循环的游戏对象，任务不包括在不同状态间切换。

Game 类中有以下方法：

初始化游戏的构造方法（__init__），以及游戏的进行（run）。

A. 构造方法（__init__）

该部分用于初始化游戏的全部基本属性。

游戏中需要获取代码的路径，以获取一些游戏和图像的目录，因此需要用到 os 模块中的 path 方法，获取当前代码路径：

```
path = os.path.abspath(args[0])
dir = os.path.split(path)[0]
```

移动那个目录（这样图片文件可以在随后打开）来获取放置当前游戏主文件 squish.py 的文件夹，此方法可适用于任何一个操作系统，在 windows 操作系统中，文件夹和子文件、子文件夹的分隔符为“\”，而在 Linux 操作系统中采用“/”分隔符，等等；通过 path.join(dir, “img”)可以将 dir 和 img 用相应的分隔符连接起来，返回一个字符串类型的文件地址。

```
os.chdir(dir)
```

然后需要用 self.state 参数设置无状态方式启动，将其设置为 None，在第一个事件循环迭代中移动到 StartUp，于是 self.nextState = StartUp()。

B. 游戏的运行（run）

游戏的运行这个方法动态设定变量。进行一些重要的初始化工作，并且进入主事件循环。初始化一些基本的参数，对于 pygame 包，需要初始化一个游戏需要进行初始化操作：

```
pygame.init()
```

如果需要加载游戏声音（背景音乐、音效）等，需要添加一行：

```
pygame.mixer.init()
```

pygame.display.set_mode(screen_size)传入的是元组参数，如果需要设置标题，就需要用到 pygame.display.set_caption()，在括号内配置字符串，标题设置为“Fruit Self Defense”。

游戏需要用到一个 while 主循环来控制游戏的进度，整个游戏的核心便是游戏的 while 循环，它通常可分为 4 个部分：

(1) 如果 nextState 被修改了，那么移动到新状态，并且显示它（第一次）

(2) 代理当前状态的事件处理

(3) 更新当前状态

(4) 显示当前状态

一般控制游戏的速度需要计算时间，则需要一个时间参数来控制游戏的快慢，pygame 提供了一个 clock 包，可以控制 clock 参数来设置游戏的速度进行。游戏中可以定义一个 self.clock 参数，即在此处用到 self.clock = pygame.time.Clock()，然后用 clock.tick(FPS)来控制游戏的速度（即游戏进行时的帧速率 FPS，此处将其设为 60，即一秒钟 60 帧）。

游戏如果还需要加载字体，那么 pygame 的优点便是提供了现成的 font 包，可以使用 pygame.font.match_font(FONT)将计算机中和 FONT 最匹配的字体导入，导入类型为 font 类，FONT 为字符串类型，此处设置为’arial’，可在配置文件中更改。然后运用向量截取的方式将图片中需要的那一部分截取下来。

对于声音文件夹的初始化，跟上面所提及的图片文件地址的初始化类似，只不过将文件夹名改成了’sound’。

②游戏物体（objects.py）

该部分主要配置游戏中可能出现的所有精灵（objects），所有的精灵都继承于 pygame 中的 pygame.sprite.Sprite 类，该类可实现游戏很多功能的简化以及易于理解。

游戏的参数操作实现了跨文件的简化，如将 squish.py 中的 game 类传入 objects.py 从而实现游戏程序设计中很多代码的简化。

游戏中定义了香蕉类（Banana）、秤砣类（Weight）。对于这些类的初始化，需要实现对 Sprite 类的继承，因此需要在__init__(self)方法中添加如下一行：

```
pygame.sprite.Sprite.__init__(self)
```

然后对类中添加其他的属性，如玩家的位置，玩家的 centerx、centery 坐标，玩家的 image 和对 image 设置的颜色键（set_colorkey）等等。

③游戏配置（config.py）

游戏配置中分为五大部分参数：图片加载（image），字体（font），游戏的物理属性（下降速度，香蕉速度，通过每关速度增加值，每关的秤砣数量，香蕉距边缘的值），游戏的随机平台列表，RGB 颜色（元组类型），导入的文件名（在此处为香蕉图片文件 banana_image 和秤砣图片文件 weight_image）。

配置游戏的物理属性需要对物理属性进行赋值，然后将 config.py 作为模块导入主函数中。根据 RGB 颜色的属性，通过控制元组(r, g, b)中的三个参数可以实现对颜色的控制，在游戏配置中定义了一些可能会经常用到的基本颜色，并用大写标注以示区别。

对平台的随机控制是在开始编辑游戏时有助于对游戏的理解而添加的额外部分，在游戏开发完成之后，并设置多个关卡的不同属性供闯关。

3. 编写程序

按照上述的基本步骤编写程序、搭好框架，在不够的地方补充，在有必要的地方加上注释，才能更通俗易懂，每一步都要按照思路进行。游戏代码应该有一定的健壮性和可移植性，不论是遇到电脑崩溃还是跨平台等各种因素，程序都可以保证游戏进度的保存和分数的保存。程序代码在第四部分。

4. 运行程序

正常运行每一关，即为初步设计成功，运行完成后，可以在相关平台（如 GitHub）上发布 1.0 版，然后根据玩家的评论和建议进行修改添加，推出不断升级的版本，在 GitHub 上发布时也要注意提供相应源文件的基本框架以及前几次游戏的备份，并在 GitHub 上发布每次游戏和上次游戏的差别，让玩家更能够理解，还要提供基本框架，为下次编写新的游戏程序节省时间。

四、设计程序

根据程序的规划设计，可以开始设计程序，程序的设计由三个部分组成，分别为 squish.py，objects.py 和 config.py，该三部分程序分别控制不同的功能，squish.py 为主要程序，单击该程序即可运行，然后是 objects.py，主要控制游戏的两个物体——秤砣和香蕉的初始化，最后是 config.py，配置游戏的基本参数，游戏的设计按照该三部分有组织的进行，在今后修改的时候只需要在相应部分添加即可。

第一部分代码：objects.py:

```
# coding=utf-8
import pygame, config, os
from random import randrange

# 这个模块包括 Squish 的游戏对象。

class SquishSprite(pygame.sprite.Sprite):
    """
    Squish 中所有子图形的范型超类。
    """
    def __init__(self, image):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image).convert()
        self.image.set_colorkey(config.white)
        self.rect = self.image.get_rect()
        #self.image = pygame.transform.scale(self.image,   (self.rect.width//2,
    self.rect.height//2))
        screen = pygame.display.get_surface()
        shrink = -config.margin * 2 # -60
```

```
        size = screen.get_rect(); # (0,0,800,600)
        self.area = screen.get_rect().inflate(shrink, shrink) #     (30,30,740,540)
        print(self.area)

class Weight(SquishSprite):
    """
    落下的秤砣。它使用了 SquishSprite 构造函数设置它的秤砣图像，并且会以给定的 速度作为构造
函数的参数来设置下落的速度。
    """
    def __init__(self, speed):
        SquishSprite.__init__(self, config.weight_image)
        self.speed = speed
        self.reset()

    def reset(self):
        """
        将秤砣移动到屏幕顶端（视线外），放置到任意水平位置上。
        """
        # random between (30,770)
        x = randrange(self.area.left, self.area.right)
        self.rect.midbottom = x, 0

    def update(self):
        """
        根据它的速度将秤砣垂直移动（下落）一段距离。并且根据它是否触及屏幕底端来    设置
    landed 属性。
        """
        self.rect.top += self.speed
        self.landed = self.rect.top >= self.area.bottom

class Banana(SquishSprite):
    """
    绝望的香蕉。它使用 SquishSprite 构造函数设置香蕉的图像，并且会停留在屏幕底 端。
    它的水平位置由当前的鼠标位置（有一定限制）决定
    """
    def __init__(self):
        SquishSprite.__init__(self, config.banana_image)
        self.rect.bottom = self.area.bottom
        # 在没有香蕉的部分进行填充。如果秤砣移动到这些区域，它不会被判定为碰撞（或 者说是将
    香蕉压扁）：
        self.pad_top = config.banana_pad_top
        self.pad_side = config.banana_pad_side

    def update(self):
        """
        将 Banana 中心点的横坐标设定为当前鼠标指针的横坐标，并且使用 rec 的 clamp    方法确保
    Banana 停留在所允许的范围内。
        """
        self.rect.centerx = pygame.mouse.get_pos()[0]
        self.rect = self.rect.clamp(self.area)

    def touches(self, other):
        """
        确定香蕉是否触碰到了另外的子图形（比如秤砣）。除了使用 rec 的 colliderect   方法外，
        首先要计算一个不包括香蕉图像顶端和侧边“空区域”的新矩形（使用 rect 的    inflate 方法
    对顶端和侧边进行填充）。
        """
        # 使用适当的填充缩小边界：
```

```
        bounds = self.rect.inflate(-self.pad_side, -self.pad_top)
        # 移动边界，将它们放置到 banana 的底部。
        bounds.bottom = self.rect.bottom
        # 检查边界是否和其他对象的 rect 交叉。
        return bounds.colliderect(other.rect)
```

第二部分代码：config.py：

```
banana_image = 'banana.png'
weight_image = 'weight.png'
splash_image = 'weight.png'

# 改变这些设置以影响一般的外观：
screen_size = 800, 600
background_color = 0, 155, 155
margin = 30
full_screen = 1
font_size = 48
white = 255, 255, 255

# 这些设置会影响游戏的表现行为：
drop_speed = 1
banana_speed = 10
speed_increase = 1
weights_per_level = 10
banana_pad_top = 40
banana_pad_side = 20

WHITE = 0, 0, 0
```

第三部分代码：squish.py：

```
# coding=utf-8
import os, sys, pygame
from pygame.locals import *
import objects, config

pygame.mixer.init()
start_sound = pygame.mixer.Sound("ue.wav")
game_over_sound = pygame.mixer.Sound("over.wav")
next_sound = pygame.mixer.Sound("next.wav")
# 这个模块包括 Squish 游戏的主要游戏逻辑。
class State:
    """
    范型游戏状态类，可以处理时间并在给定的表面上显示自身。
    """
    def handle(self, event):
        """
        只处理退出时间的默认事件处理。
        """
        if event.type == QUIT:
            sys.quit()
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                sys.quit()

    def firstDisplay(self, screen):
        """
```

```
        用于第一次显示状态。使用背景色填充屏幕。
        """
        screen.fill(config.background_color)
        # 记得要使用 flip,让更改可见
        pygame.display.flip()

    def display(self, screen):
        """
        用于在已经显示过一次状态后再次显示。默认的行为是什么都不做。
        """
        pass

class Level(State):
    """
    游戏等级。用于计算已经落下了多少秤砣，移动子图形以及其他和游戏逻辑相关的任务。
    """
    def __init__(self, number=1):
        self.number = number
        # 本关内还要落下多少秤砣?
        self.remaining = config.weights_per_level

        speed = config.drop_speed
        # 为每个大于 1 的等级都增加一个 speed_increase
        speed += (self.number - 1) * config.speed_increase
        # 创建秤砣和香蕉:
        self.weight = objects.Weight(speed)
        self.banana = objects.Banana()
        both = self.weight, self.banana # This could contain more sprites...
        self.sprites = pygame.sprite.RenderUpdates(both)
        game_over_sound.stop()
        start_sound.play()
        next_sound.stop()

    def update(self, game):
        "从前一帧更新游戏状态"
        # 更新所有子图形:
        self.sprites.update()
        # 如果香蕉碰到了秤砣，那么告诉游戏切换到 GameOver 状态:
        if self.banana.touches(self.weight):
            game.nextState = GameOver()
            game_over_sound.play()
            start_sound.stop()
            # 否则在秤砣落地时将其复位。
            # 如果在本馆内所有秤砣都落下了，则让游戏切换到 LevelCleared 状态:
        elif self.weight.landed:
            self.weight.reset()
            self.remaining -= 1
            if self.remaining == 0:
                game.nextState = LevelCleared(self.number)

    def display(self, screen):
        """
        在第一次显示（只清空屏幕）后显示状态。与 firstDisplay 不同，这个方法使用
        pygame.display.update 对
        self.sprites.draw 提供的、需要更新的矩形列表进行更新。
        """
        screen.fill(config.background_color)
        updates = self.sprites.draw(screen)
```

```
        pygame.display.update(updates)

class Paused(State):
    """
    简单的暂停游戏状态，按下键盘上任意键或者点击鼠标都会结束这个状态。
    """
    finished = 0 # 用户结束暂停了吗？
    image = None # 如果需要图片的话，将这个变量设置为文件名
    text = '' # 将它设定为一些提示性文本

    def handle(self, event):
        """
        通过对 State 进行委托（一般处理退出事件）以及对按键和鼠标点击最、作为反应来处理事件。
        如果键被按下或者鼠标被点击，将 self.finished 设定为真。
        """
        State.handle(self, event)
        if event.type in [MOUSEBUTTONDOWN, KEYDOWN]:
            self.finished = 1
            start_sound.stop()

    def update(self, game):
        """
        更新等级。如果按键被按下或者鼠标被点击（比如 self.finished 为真），那么告诉我们切换
    到下 一个由 self.nextState()表示的状态（应该由子类实现）
        """
        if self.finished:
            game.nextState = self.nextState()



    def firstDisplay(self, screen):
    """
    暂停状态第一次出现时，绘制图像（如果有的话）并且生成文本。
    """
    #首先，使用填充背景色的方式清空屏幕：
    screen.fill(config.background_color)

    # 使用默认的外观和指定的大小创建 Font 对象
    font = pygame.font.Font(None, config.font_size)

    # 获取 self.text 中的文本行,忽略开头和结尾的空行：
    lines = self.text.strip().splitlines()

    # 计算文本的高度(使用 font.get_linesize()）以获取每行文本的高度：
    height = len(lines) * font.get_linesize()

    # 计算文本的放置位置（屏幕中心）：
    center, top = screen.get_rect().center # 为 400,300
    top -= height // 2 # 260
    # 如果有图片要显示的话...
    if self.image:
    # 载入图片：
        image = pygame.image.load(self.image).convert()
        image.set_colorkey(config.white)
        # 获取它的 rect:
        r = image.get_rect() # 为 rect(0,0,166,132)
        # 将图像向下移动其高度的一半的距离：
        top += r.height // 2 # 326
```

```
            # 将图片放置在文本上方 20 像素处：
            r.midbottom = center, top - 20 # 400,306
            # 将图像移动到屏幕上：
            screen.blit(image, r)

        antialias = 1 # Smooth the text
        black = 0, 0, 0 # Render it as black

        # 生成所有行，从计算过的 top 开始，并且对于每一行向下移动 font.get_linesize()像素：
        for line in lines:
            text = font.render(line.strip(), antialias, black)
            r = text.get_rect() # 0,0,312,37
            r.midtop = center, top # 400,326
            screen.blit(text, r)
            top += font.get_linesize()

        # 显示所有更改：
        pygame.display.flip()

class Info(Paused):
    """
    简单的暂停状态，显示有关游戏的信息。在 Level 状态后显示（第一级）
    """
    nextState = Level
    text = '''''
    In this game you are a banana,
    trying to survive a course in
    self-defense against fruit, where the
    participants will "defend" themselves
    against you with a 16 ton weight.'''

class StartUp(Paused):
    """
    显示展示图片和欢迎信息的暂停状态。在 Info 状态后显示。
    """
    nextState = Info
    start_sound.play()
    image = config.splash_image
    text = '''''
    Welcome to Squish,
    the game of Fruit Self-Defense'''

class LevelCleared(Paused):
    """
    提示用户过关的暂停状态。在 next level 状态后显示。
    """
    def __init__(self, number):
        self.number = number
        self.text = '''''Level %i cleared
        Click to start next level''' % self.number
        start_sound.stop()
        next_sound.play()

    def nextState(self):
        return Level(self.number+1)

class GameOver(Paused):
    """
```

```
    提示用户输掉游戏的状态。在 first level 后显示。
    """
    nextState = Level
    text = '''''
    Game Over
    Click to Restart, Esc to Quit'''

class Game:
    """
    负责主事件循环的游戏对象，任务不包括在不同状态间切换。
    """
    def __init__(self, *args):
        # 获取游戏和图像放置的目录：:
        path = os.path.abspath(args[0]) # 当前代码文件路径
        dir = os.path.split(path)[0] # 代码目录
        # 移动那个目录（这样图片文件可以在随后打开）:
        os.chdir(dir) # cd 到代码目录
        # 无状态方式启动:
        self.state = None
        # 在第一个事件循环迭代中移动到 StartUp
        self.nextState = StartUp()

    def run(self):
        """
        这个方法动态设定变量。进行一些重要的初始化工作，并且进入主事件循环。
        """
        pygame.init() # 初始化所有 pygame 模块。
        pygame.mixer.init()
        # 决定以窗口模式还是全屏模式显示游戏
        flag = 0 # 默认窗口

        if config.full_screen:
            flag = FULLSCREEN # 全屏模式
        screen_size = config.screen_size
        screen = pygame.display.set_mode(screen_size)#, flag 非全屏模式

        pygame.display.set_caption('Fruit Self Defense')
        pygame.mouse.set_visible(False)

        # 主循环:
        while True:
            # (1) 如果 nextState 被修改了，那么移动到新状态，并且显示它（第一次）
            if self.state != self.nextState:
                self.state = self.nextState
                self.state.firstDisplay(screen)
            # (2)代理当前状态的事件处理:
            for event in pygame.event.get():
                self.state.handle(event)
            # (3) 更新当前状态:
            self.state.update(self)
            # (4) 显示当前状态:
            # time.sleep( 0.5 )
            self.state.display(screen)

if __name__ == '__main__':
# print(sys.argv)
game = Game(*sys.argv)
game.run()
```

五、运行结果

运行的时候出现一个新的窗口，该窗口即为游戏主画面，开始出现一个引导游戏的画面，游戏指示我们一些游戏的基本操作，按任意键或者点击鼠标即可开始游戏。

香蕉只能在固定的纵坐标下移动，横坐标值与鼠标横坐标值相等。只要秤砣和香蕉有重叠部分，即游戏结束。

六、结果分析

游戏中对速度的控制随着关卡数增加而增加，游戏容易上手，对游戏结束的标志也很简单，适合初学 pygame 者上手学习，游戏有很多开发空间，在游戏的开发过程中可以添加各种下降物，可以增加玩家的命等等。