上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

2019 年～2020 年春季学期

《程序设计》期末大作业

项目名称：Py（π）酷跑

小组名称：无魏三国

小组成员：

俞梦婕 518120910060

刘子欣 518120910055

孙美淇 518120910057

目录

1.项目摘要

本项目设计了一款跑酷类游戏，玩家通过空格等按键操纵一只会奔跑跳跃的小尼玛跨过一系列障碍物，或获取一些加分道具等。游戏采取积分制，跨过障碍物、获取道具都会有相应的加分，玩家应尽可能准确地操纵尼玛跑酷，以获取高分。

2.选题原因

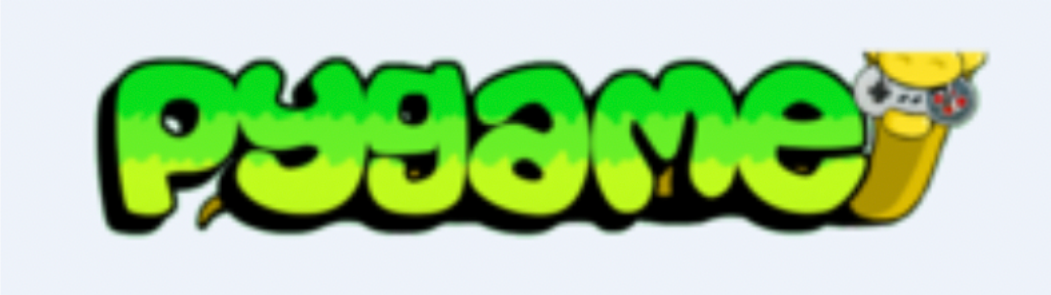
跑酷是时下风靡全球的时尚极限运动，以日常生活的环境为运动场所，依靠自身的体能，快速、有效、可靠地驾驭任何已知与未知环境的运动艺术。而对于很多由于种种限制不能进行跑酷运动的人来说，跑酷游戏就是一个很好的选择。

在 Apple store 上，以地铁跑酷为代表的一些跑酷游戏能达到近 2 亿下载量，而随便点开一个跑酷游戏，都会看到以千万计的下载量。在其中，有很多游戏就是用 python 语言编写的。在学习了 python 之后，本小组也尝试着自己编写代码，做了一个跑酷游戏。

在本游戏中，我们减弱了跑酷游戏的刺激性，是游戏变得更温和。同时，我们也尝试着添加一些益智元素，以娱乐、放松的方式进行学习。除此之外，小组还在背景中加入了交大具有代表性的元素。

3.使用工具

3.1 Pygame



Pygame 包含图像、声音，建立在 SDL 基础上，允许实时电子游戏研发而无需被低级语言（如机器语言和汇编语言）束缚。基于这样一个设想，所有需要的游戏功能和理念都（主要是图像方面）都完全简化为游戏逻辑本身，所有的资源结构都可以由高级语言提供，如 Python。

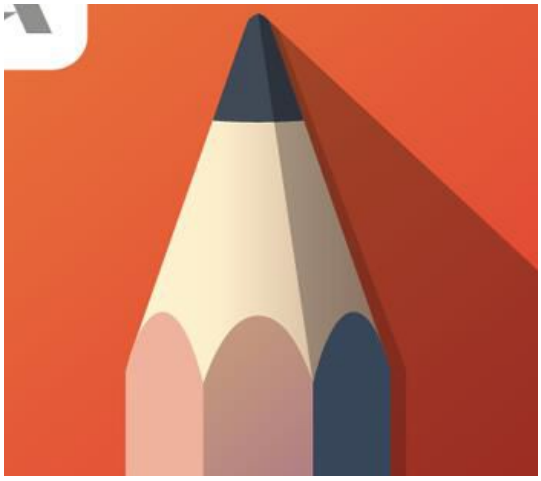
Pygame 是被设计用来写游戏的 python 模块集合，是在优秀的 SDL 库之上开发的功能性包。使用 python 可以导入 pygame 来开发具有全部特性的游戏和多媒体软件，Pygame 是极度轻便的并且可以运行在几乎所有的平台和操作系统上。Pygame 包已经被下载过成千上万次，并且也被访问过成千上万次。

3.2 Photoshop

Adobe Photoshop，简称“PS”，是由 Adobe Systems 开发和发行的图像处理软件。

Photoshop 主要处理以像素所构成的数字图像。使用其众多的编修与绘图工具，可以有效地进行图片编辑工作。PS 有很多功能，在图像、图形、文字、视频、出版等各方面都有涉及。

3.3 Autodesk SketchBook



SketchBook Pro 是 Autodesk 公司开发针对 Honeycomb 的应用，是一款功能强大的绘图工具，能够帮助使用者制作出专业水平的素描和绘画作品，让其在行动装置上绘制出令人难以置信的内容。Sketchbook 拥有专业级别的绘图工具带有极其丰富的触笔和各色工具，比如铅笔、喷咀、油划笔、原子笔、甚至填充效果等等。

4.代码内容

4.1 基本参数

0：42*55
1：27*55
2：42*55
3：42*55
4：40*56
5：35*55
6：38*55

7：41*56
8：40*55
9：41*55
pose 1：70*94
pose 2：71*95
pose 3：71*94

4.2 代码编写

4.2.1 导入 pygame

```
import pygame
from pygame.locals import *
```

首先，下载 pygame 包，并将其导入 python 程序中。
然后，导入 pygame 中的 locals 模块，该模块中包含了 pygame 定义的各种常量。

4.2.2 设置基本的内容

```
SCREENWIDTH = 800  # 窗口宽度
SCREENHEIGHT = 300  # 窗口高度
FPS = 50  # 更新画面的时间
```

在导入 pygame 之后，对窗口的宽度、高度及更新画面的时间进行赋值。

```
from itertools import cycle
```

从 python 的内建模块 itertools 中导入 cycle，将传入的序列无限重复下去。

4.2.3 定义类

```
class MyMap():

    def __init__(self, x, y):
        self.bg = pygame.image.load("image/mm2.png").convert()
        self.x = x
        self.y = y

    def map_rolling(self):
        if self.x < -790:
            self.x = 800
        else:
            self.x -= 5

    def map_update(self):
        SCREEN.blit(self.bg, (self.x, self.y))
```

这里定义了一个滚动地图类。用__init__()方法导入地图图片，用 map_rolling()方法滚动地图。如果 self.x 小于-790，则说明地图已滚动完成，则给地图重新定义一个新的坐标点，如果地图没有滚动完成，那么使其向左移动 5 个像素。map_update()方法则利用 blit 方法，将 selt.bg 放在 SCREEN 对象上，位置的坐标由 self.x 和 self.y 确定。

```
class Nima():
    def __init__(self):
        # 初始化角色矩形
        self.rect = pygame.Rect(0, 0, 0, 0)
        self.jumpState = False
        self.jumpHeight = 130  # 跳跃高度
        self.lowest_y = 160  # 角色最低坐标
        self.jumpValue = 0  # 高度增加量
        self.nimaIndex = 0
        self.nimaIndexGen = cycle([0, 1, 2])
        # 加载角色
        self.nima_img = (
            pygame.image.load("image/man_obst/pose1.png").convert_alpha(),
            pygame.image.load("image/man_obst/pose2.png").convert_alpha(),
            pygame.image.load("image/man_obst/pose3.png").convert_alpha(),
        )
        self.jump_audio = pygame.mixer.Sound('audio/jump.wav')
        self.rect.size = self.nima_img[0].get_size()
        self.x = 50;  # 角色的X坐标
        self.y = self.lowest_y;  # 角色的Y坐标
        self.rect.topleft = (self.x, self.y)
```

在这里定义了一个人物类。用__init__()方法定义尼玛的初始状态，包括跳跃状态、高度、坐标等。其中用 rect 储存矩形对象，rect.size 储存矩形大小，self.rect.topleft 则储存了左上角的显示位置，以此类推。然后导入尼玛图片，循环为动图。

接下来在 self.jump_audio 中储存小尼玛跳动的声音，并确定了尼玛的位置。

```
def jump(self):
    self.jumpState = True

# 角色移动
def move(self):
    if self.jumpState:  # 判断是否跳起
        if self.rect.y >= self.lowest_y:  # 若角色位于地面
            self.jumpValue = -5  # 以5个像素值向上移动
        if self.rect.y <= self.lowest_y - self.jumpHeight:  # 若角色到达顶部
            self.jumpValue = 5  # 以5个像素值向下移动
        self.rect.y += self.jumpValue
        if self.rect.y >= self.lowest_y:  # 如果角色回到地面
            self.jumpState = False  # 关闭跳跃状态
```

随即定义两个方法 jump()和 move()。Jump()用以设定起跳状态，而 move()则控制尼玛的移动。在人物跳起时，如果其站在地上，则上移 5 个像素；如果其已经到达了最高点，则下移 5 个像素，从而得出尼玛现在的位置。如果其回到地面，则表示起跳完毕。

```
def draw_nima(self):
    # 定义角色行走动图
    nimaIndex = next(self.nimaIndexGen)
    SCREEN.blit(self.nima_img[nimaIndex],
                (self.x, self.rect.y))
```

接下来，定义函数 draw_nima ()，导入小尼玛动图，将其绘制其中。

```
import random

class Obstacle():
    score = 1
    def __init__(self):
        self.rect = pygame.Rect(0, 0, 0, 0)
        # 加载障碍物图片
        self.num0 = pygame.image.load("image/man_obst/0.png").convert_alpha()
        self.num1 = pygame.image.load("image/man_obst/1.png").convert_alpha()
        self.num2 = pygame.image.load("image/man_obst/2.png").convert_alpha()
        self.num3 = pygame.image.load("image/man_obst/3.png").convert_alpha()
        self.num4 = pygame.image.load("image/man_obst/4.png").convert_alpha()
        self.num5 = pygame.image.load("image/man_obst/5.png").convert_alpha()
        self.num6 = pygame.image.load("image/man_obst/6.png").convert_alpha()
        self.num7 = pygame.image.load("image/man_obst/7.png").convert_alpha()
        self.num8 = pygame.image.load("image/man_obst/8.png").convert_alpha()
        self.num9 = pygame.image.load("image/man_obst/8.png").convert_alpha()
        # 加载分数图片
        self.numbers = (pygame.image.load('image/0.png').convert_alpha(),
                        pygame.image.load('image/1.png').convert_alpha(),
                        pygame.image.load('image/2.png').convert_alpha(),
                        pygame.image.load('image/3.png').convert_alpha(),
                        pygame.image.load('image/4.png').convert_alpha(),
                        pygame.image.load('image/5.png').convert_alpha(),
                        pygame.image.load('image/6.png').convert_alpha(),
                        pygame.image.load('image/7.png').convert_alpha(),
                        pygame.image.load('image/8.png').convert_alpha(),
                        pygame.image.load('image/9.png').convert_alpha())
        # 加载加分音效
        self.score_audio = pygame.mixer.Sound('audio/score.wav')  # 加分
```

然后定义障碍物类。这里先导入 random，为接下来的随机数做准备。用 score 储存障碍物的分数。定义__init__()方法，初始化障碍物矩形，然后用 self.numx 导入障碍物图片，用 self.numbers 导入分数图片，并用 self.score_audio 储存加分音效。

```
r = random.randint(0, 9)
if r == 0:
    self.image = self.num0
elif r == 1:
    self.image = self.num1
elif r == 2:
    self.image = self.num2
elif r == 3:
    self.image = self.num3
elif r == 4:
    self.image = self.num4
elif r == 5:
    self.image = self.num5
elif r == 6:
    self.image = self.num6
elif r == 7:
    self.image = self.num7
elif r == 8:
    self.image = self.num8
else:
    self.image = self.num9
```

用 random.randint(0,9)获取 0 到 8 的整数随机数，储存在 r 中，用以调用随机障碍物。

```
    #设置障碍物
     self.rect.size = self.image.get_size()
     self.width, self.height = self.rect.size
     self.x = 800;
     self.y = 260 - (self.height / 2)
     self.rect.center = (self.x, self.y)

 # 障碍物移动
 def obstacle_move(self):
     self.rect.x -= 5

 # 绘制障碍物
 def draw_obstacle(self):
     SCREEN.blit(self.image, (self.rect.x, self.rect.y))
```

用 self.rect.size 设置障碍物矩形，用 self.width 和 self.height 储存障碍物的宽和高，并进行绘制，用 obstacle_move()和 draw_obstacle()移动障碍物、绘制障碍物，原理与尼玛的相同。

```
# 获取分数
def getScore(self):
    self.score
    tmp = self.score;
    if tmp == 1:
        self.score_audio.play()  # 播放加分音乐
    self.score = 0;
    return tmp;

# 显示分数
def showScore(self, score):
    """在窗体顶部中间的位置显示分数"""
    self.scoreDigits = [int(x) for x in list(str(score))]
    totalWidth = 0
    for digit in self.scoreDigits:
        totalWidth += self.numbers[digit].get_width()
    Xoffset = (SCREENWIDTH - totalWidth) / 2
    for digit in self.scoreDigits:
        SCREEN.blit(self.numbers[digit], (Xoffset, SCREENHEIGHT * 0.1))
        Xoffset += self.numbers[digit].get_width()
```

定义 getScore()以获取分数，并在加分时播放加分音乐。定义 showScore()显示分数。Self.scoreDigits 为分数的列表，totalWidth 则用以储存要显示的所有数字的宽度。将 Xoffset 设置为横坐标，随着数字增加，Xoffset 的位置也随之改变。

```
# 游戏结束
def game_over():
    bump_audio = pygame.mixer.Sound('audio/bump.wav')
    bump_audio.play()
    screen_w = pygame.display.Info().current_w
    screen_h = pygame.display.Info().current_h
    over_img = pygame.image.load('image/gameover.png').convert_alpha()
    SCREEN.blit(over_img, ((screen_w - over_img.get_width()) / 2,
                                    (screen_h - over_img.get_height()) / 2))
```

定义 game_over()方法以结束游戏并播放结束游戏的音效。用 screen_w 和 screen_h 储存窗口的宽和高，用 over_img 储存游戏结束的图片，并用 SCREEN.blit()将其绘制在窗口中央。

4.2.4 脚本编写

```
def mainGame():
    score = 0
    over = False
    global SCREEN, FPSCLOCK, word
    pygame.init()

    FPSCLOCK = pygame.time.Clock() #循环时长
    SCREEN = pygame.display.set_mode((SCREENWIDTH, SCREENHEIGHT))
    pygame.display.set_caption('Py(π)酷跑') # 设置窗口标题
    # 设置游戏图标
    icon = pygame.image.load('image/pi.jpeg')
    pygame.display.set_icon(icon)
    # 创建地图
    bg1 = MyMap(0, 0)
    bg2 = MyMap(800, 0)
    # 创建尼玛
    nima = Nima()
    addObstacleTimer = 0  # 添加障碍物的时间
    list = []  # 障碍物列表
```

定义 mainGame()函数，设置初始得分。用 FPSCLOCK 控制循环的时间间隔，用 SCREEN 创建窗口以便于交互。接下来设置一系列的初始值，窗口标题、地图、尼玛及障碍物对象等。

```
while True:
    for event in pygame.event.get():

        if event.type == QUIT:
            exit()  # 关闭窗口
        if event.type == KEYDOWN and event.key == K_SPACE:
            if nima.rect.y >= nima.lowest_y:
                nima.jump()
                nima.jump_audio.play()
            if over == True:
                mainGame()
```

如果玩家单击了关闭窗口即 event.type==QUIT，则游戏退出；如果玩家按了空格键，则尼玛起跳并播放起跳音效。如果游戏结束，则重新调用 mainGame()方法，可再次启动游戏。

```
if over == False:

    bg1.map_update()
    bg1.map_rolling()
    bg2.map_update()
    bg2.map_rolling()
    nima.move()
    nima.draw_nima()
    # 计算障碍物间隔时间
    if addObstacleTimer >= 1300:
        r = random.randint(0, 100)
        if r > 40:
            obstacle = Obstacle()
            list.append(obstacle)
        addObstacleTimer = 0
    for i in range(len(list)):
        list[i].obstacle_move()
        list[i].draw_obstacle()
        # 判断尼玛与障碍物是否碰撞
        if pygame.sprite.collide_rect(nima, list[i]):
            over = True
            game_over()
        else:
            # 判断尼玛是否跳过障碍物
            if (list[i].rect.x + list[i].rect.width) < nima.rect.x:
                score += list[i].getScore()
        list[i].showScore(score)
```

如果游戏仍在运行，则更新并移动地图、尼玛。用 addObstacleTimer 判断障碍物间隔时间，并在障碍物出现之后重置。随后循环遍历障碍物，如果尼玛与障碍物相撞，则调用 game_over()方法，游戏结束；如果未相撞，则加分并显示分数。

```
        addObstacleTimer += 20
        pygame.display.update()  # 更新整个窗口
        FPSCLOCK.tick(FPS)


if __name__ == '__main__':
    mainGame()
```

用 addObstacleTimer 增加障碍物时间，调用 pygame.display.update()以更新整个窗口，用 FPSCLOCK.tick(FPS)设定循环时间。

脚本编写完毕。

5.项目优化

5.1 加入背景音乐

在设置游戏图标之后，用 pygame.mixer.music.load()导入背景音乐，并用 pygame.mixer.music.play()播放。

```
def mainGame():
    score = 0
    over = False
    global SCREEN, FPSCLOCK, word
    pygame.init()

    FPSCLOCK = pygame.time.Clock() #循环时长
    SCREEN = pygame.display.set_mode((SCREENWIDTH, SCREENHEIGHT))
    pygame.display.set_caption('Py(π)酷跑') # 设置窗口标题
    # 设置游戏图标
    icon = pygame.image.load('image/pi.jpeg')
    pygame.display.set_icon(icon)
    # 设置游戏背景音乐
    pygame.mixer.music.load("audio/bgm.mp3")
    pygame.mixer.music.play()
    # 创建地图
    bg1 = MyMap(0, 0)
    bg2 = MyMap(800, 0)
    # 创建尼玛
    nima = Nima()
    addObstacleTimer = 0  # 添加障碍物的时间
    list = []  # 障碍物列表
```

5.2 设置障碍物出现顺序

```
import pygame
from pygame.locals import *
import math
```

在导入 pygame 的同时导入 math，为下一步做准备。

```
#定义π中数字出现的顺序
pi1 = str(math.pi)
pi = pi1[0] + pi1[2:100]
def de(a):
    return a[1:]
```

在定义角色行走动图之后定义数字顺序，使障碍物按照 pi 的数字顺序出现。

```
if pi[0] == "0":  # 如果当前π的第一个数字为0显示障碍物数字0
    self.image = self.num0
elif pi[0] == "1":
    self.image = self.num1
elif pi[0] == "2":
    self.image = self.num2
elif pi[0] == "3":
    self.image = self.num3
elif pi[0] == "4":
    self.image = self.num4
elif pi[0] == "5":
    self.image = self.num5
elif pi[0] == "6":
    self.image = self.num6
elif pi[0] == "7":
    self.image = self.num7
elif pi[0] == "8":
    self.image = self.num8
elif pi[0] == "9":
    self.image = self.num9
pi = de(pi) #障碍物变化完毕，删去当前π的第一个数字，推进至下一个障碍物顺序
```

5.3 调整难度

```
import time

time1 = time.time()
```

```
time2 = time.time()-time1
if time2 > 30:
    addObstacleTimer += 25  # 增加障碍物时间
if time2 <= 30:
    addObstacleTimer += 15
print(time2)
pygame.display.update()  # 更新整个窗口
FPSCLOCK.tick(FPS)  # 循环应该多长时间运行一次
```

代码最前面加入 import time，更新窗口前则加入对增加时间的判断。如果玩家技巧性好，则障碍物出现频率加快，如果玩家不熟悉游戏，则障碍物出现频率增加得慢。

5.4 展望

1）额外加分：设置加分道具，在游戏界面的上半部分随机出现加分道具，当角色的横坐标等于道具横坐标且角色纵坐标大于等于道具纵坐标时，玩家将额外得到一分。

2）真人跑酷：通过电脑外置摄像头拍摄玩家人像，并通过对玩家人像轮廓的识别，实时抠像并显示在电脑屏幕背景上，作为游戏的角色，达到虚拟与现实结合的效果。

3）攻击技能：如果出现的数字不是按照π的顺序出现的话，则需要玩家对其进行攻击。

4）优化算法：优化判定角色与障碍物之间碰撞的算法，使得判断更加精确，避免因图片 pgn 的留白而导致判断范围扩大的问题。

6.总结

Pi 酷跑是一个很可爱的跑酷类游戏，而制作它的本项目则是一个学习 python 及 pygame 的很好的机会。在该项目中，我们不仅将自己在课堂上所学习的知识运用了进来，还学习了 pygame 的各种方法，加深了对编程的了解，更将努力转化为了成果。无论以后是否从事相关工作，python 对我们都具有十分重要的意义。