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，此次设计的游戏为小兔（BUNNY）以及超级玛丽等等游戏的组合体，实现功能强大的游戏。

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[见附 1]

对于类似超级马里奥兄弟的基于平台开发的游戏，首先应该确定如何对这个游戏的各个角色以及角色的属性、小兔的行进方向、游戏地图的生成等进行详细的规划。
对游戏的总体功能有如下的几个基本规划：

①游戏需要一个玩家，玩家能实现跳跃、加分、升级、发射、打死怪物、重生等功能；

②游戏中需要由多个固定地图的关卡，而且难度一次升级，每次关卡有不同的游戏场景、遇到不同的怪物、通关方式都是以达到某个距离或者碰到某个旗帜为准。

③游戏中需要有金币、津贴（如捡到后会加速、会加命、保存游戏进度、过关奖励、能发炮弹，而且要能够有足够隐藏和让玩家意想不到）、有背景贴图、不同的怪物、而且有不同的死亡方式（碰到怪物、火焰、隐藏机关或者子弹时）等。

④游戏地图应该有仔细的端详，设计应该巧妙，能够保证玩家每次能找到顺利的方式通关，而不会出现半途中无法通关的现象。

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 pygame as pg
import random as rd
from pygame.locals import *
from settings import *
from sprites import *
from os import path
```

游戏程序分为三大模块：主程序（main.py）、游戏精灵（sprites.py）和游戏配置（settings.py）。
对于每一个模块的规划如下：
①主程序（main.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），对游戏进行整体控制。
游戏类中有以下方法：

初始化游戏的构造方法（__init__），游戏数据的导入（load_data），新游戏的开始（new），游戏的进行（run），游戏状态的更新（update），游戏控制事件（events），

游戏画面的绘制（draw）等等。

A. 构造方法（__init__）

该部分用于初始化游戏的全部基本属性。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()
```

游戏最重要的是设置屏幕宽度和高度，然后设置标题属性，则需要在游戏配置中添加 WIDTH、HEIGHT 以及 TITLE 的值，本游戏中设置为 1280、720，标题设置为“我的游戏”。

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。该游戏中定义了一个 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）。

游戏还需要加载字体，pygame 提供了 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。

B. 数据的导入（load_data）

游戏数据的导入包括定义的放置图片的文件夹以及声音的文件夹等。这个游戏中有一个图片文件夹（img）和一个声音文件夹（sound），导入时采用相对地址，即：

```
self.dir = path.dirname(__file__)
```

来获取放置当前游戏主文件 main.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。

在此游戏中，由于所有的 sprite 都已聚合在一张图片中，因此导入图片时直接采用 path.join(sprite_dir, SPRITESHEET)，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’。

C. 新游戏的开始（new）

在每一关开始的时候，需要重置一些参数，以及对角色的重新初始化，每一关开始的时候，分数要清零，并同时将最高分数存放在一个文件夹中，然后初始化 self.all_sprites，以及一些附属的类：self.platforms（游戏平台）、self.powerups（游戏津贴）、self.mobs（游戏敌人）等。对音乐也需要初始化。

D. 游戏的进行（run）

一个新的游戏开始时，需要播放背景音乐（bgm.ogg），以及游戏结束的时候需要让背景音乐渐渐消失：pygame.mixer.music.fadeout()。

游戏需要用 while 循环来控制游戏的开始和结束，while 的参数 self.playing 则作为游戏的状态参数，当玩家被敌人攻击或者掉入深坑后结束游戏，同时设置 self.playing

= False，来实现对游戏的控制。对整个框架的浓缩，代码如下：

```
while self.playing:
    self.clock.tick(FPS)
    self.events()
    self.update()
    self.draw()
```

E.游戏状态的更新（update）

游戏是以帧数进行的，因此该部分控制每一帧显示在屏幕上的各个平台以及玩家的状态，游戏玩家在玩游戏的时候，玩家的位置会随着帧数的改变出现在屏幕上的不同位置，玩家从左向右到达整个窗口的 1/2 时，整个窗口的所有物体需要向左移动，以实现类似摄像机的功能，同时到达游戏窗口最左端的所有物体已无法在屏幕上呈现，但是仍然在游戏的内存中存在，因此需要将这一部分物体删除。

玩家的 rect 的最顶端坐标大于游戏屏幕的 HEIGHT 参数后，即玩家已经掉到超过屏幕最下面的部分，因此可以判断玩家已经丢掉一条命，需要播放 die.wav，结束背景音乐，以及设置游戏运行的参数 self.playing = False，重新开始游戏。

F.游戏控制事件（events）

游戏中会接收各种键盘按键，因此需要用这一部分来接收玩家的按键参数，并转化为对游戏对象的控制，包括按下按键（KEY_DOWN），抬起按键（KEY_UP），退出游戏（QUIT）。

G.游戏画面的绘制（draw）

该部分代码实现将游戏中玩家获得的分数、玩家的关卡、玩家的剩余时间等实时显示在屏幕的最上端，以及对游戏开始结束画面的文字绘制，从而实现向玩家传达游戏中的积分奖励信息。

实现以上所有方法的编制之后，需要对 game 类进行调用，以开启该游戏，代码如下：

```
g = Game()
g.show_start_screen()
while g.running:
    g.new()
    g.show_go_screen()
pg.quit()
```

②游戏精灵（sprites.py）

该部分主要配置游戏中可能出现的所有精灵（sprites），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 main.py 中的 game 类传入 sprites.py 从而实现游戏程序设计中很多代码的简化。

游戏中定义了玩家类（Player）、平台类（Platform）、敌人类（Mob）、津贴类（Pow）。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（settings.py）

游戏配置中分为五大部分参数：游戏屏幕（screen），字体（font），游戏的物理属性（速度，水平加速度，重力加速度，垂直向上的初速度，摩擦系数），游戏的随机平台列表，RGB 颜色（元组类型），导入的文件名（在此处为图片文件 spritesheet_jumper.png 和保存最高分的文件 highscore.txt）。

配置游戏的物理属性需要对物理属性进行赋值，定义摩擦系数 PLAYER_FRICTION = 10，然后将 settings.py 作为模块导入主函数中。

根据摩擦力公式、速度公式以及位移公式：

$$F_f = Kv, v = at, x = v_0 t + \frac{1}{2}at^2$$

可以在游戏的 sprites.py 中添加以下三行：

```
self.a.x += self.vel.x * PLAYER_FRICTION
self.vel += self.a
self.pos += self.vel + 0.5 * self.a
```

由于该三行公式并不能保证玩家在有一定水平初速度之后释放手中的所有按键一段时间后速度 self.vel = 0，因此还需要添加以下判断条件：

```
if abs(self.vel.x) < 0.1:
    self.vel.x = 0
```

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，可以选择是否对游戏的地图进行固定，并设置多个关卡供闯关。

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

```
import pygame as pg
import random as rd
from pygame.locals import *
from settings import *
from sprites import *
from os import path
from maps import *

class Game:
    def __init__(self):
#游戏初始化各种数据
        pg.init()
        pg.mixer.init()
```

```
        self.screen = pg.display.set_mode((WIDTH, HEIGHT))
        pg.display.set_caption(TITLE)
        self.clock = pg.time.Clock()
        self.running = True
        self.lives = 5
        self.c_score = 0
        self.b_score = 0
        self.s_score = 0
        self.g_score = 0
        self.game_over = False
        self.font_name = pg.font.match_font(FONT)
        self.load_data()

    def load_data(self):
        # 加载数据
        self.dir = path.dirname(__file__)
        sprite_dir = path.join(self.dir, 'Spritesheets')

        #导入图片地址
        self.img_dir = path.join(self.dir, 'img')

        #导入云彩图片
        self.cloud_images = []
        for i in range(1, 4):
        self.cloud_images.append(pg.image.load(path.join(self.img_dir,
        "cloud{}.png".format(i))).convert())
        #导入分数
        with open(path.join(self.dir, HS_FILE), 'w') as f:
        try:
        self.highscore = int(f.read())
        except:
        self.highscore = 0
        #统一图片路径
        self.spritesheet = Spritesheet(path.join(sprite_dir, SPRITESHEET))
        self.snd_dir = path.join(self.dir, 'sound')

        #声音路径
        self.jump_sound = pg.mixer.Sound(path.join(self.snd_dir, 'jump1.wav'))
        self.die_sound1 = pg.mixer.Sound(path.join(self.snd_dir, "die.wav"))
        self.boost_sound = pg.mixer.Sound(path.join(self.snd_dir, 'speedup.wav'))
        self.coin_sound = pg.mixer.Sound(path.join(self.snd_dir, 'coin.wav'))
        self.up_sound = pg.mixer.Sound(path.join(self.snd_dir, 'up.wav'))

        #导入 HUD(屏幕最上方显示命、显示硬币数和胡萝卜数)
        hud = path.join(self.img_dir, 'HUD')
        self.show_carrot = pg.image.load(path.join(hud, 'carrots.png')).convert()
        self.show_carrot.set_colorkey(BLACK)
        self.show_carrot_rect = self.show_carrot.get_rect()
        self.show_carrot = pg.transform.scale(self.show_carrot, (30, 30))
        self.show_carrot_rect.x = 0.2*WIDTH + 15
        self.show_carrot_rect.y = 0.06*HEIGHT

        self.show_b_coin = pg.image.load(path.join(hud, 'coin_bronze.png')).convert()
        self.show_b_coin.set_colorkey(BLACK)
        self.show_b_coin_rect = self.show_carrot.get_rect()
        self.show_b_coin = pg.transform.scale(self.show_b_coin, (30, 30))
        self.show_b_coin_rect.x = 0.4*WIDTH + 15
        self.show_b_coin_rect.y = 0.06*HEIGHT

        self.show_s_coin = pg.image.load(path.join(hud, 'coin_silver.png')).convert()
        self.show_s_coin.set_colorkey(BLACK)
        self.show_s_coin_rect = self.show_carrot.get_rect()
        self.show_s_coin = pg.transform.scale(self.show_s_coin, (30, 30))
        self.show_s_coin_rect.x = 0.6*WIDTH + 15
        self.show_s_coin_rect.y = 0.06*HEIGHT
        self.show_g_coin = pg.image.load(path.join(hud, 'coin_gold.png')).convert()
        self.show_g_coin.set_colorkey(BLACK)
        self.show_g_coin_rect = self.show_carrot.get_rect()
        self.show_g_coin = pg.transform.scale(self.show_g_coin, (30, 30))
```

```
        self.show_g_coin_rect.x = 0.8*WIDTH + 15
        self.show_g_coin_rect.y = 0.06*HEIGHT

        self.show_life = pg.image.load(path.join(hud, 'lifes.png')).convert()
        self.show_life.set_colorkey(BLACK)
        self.show_life_rect = self.show_carrot.get_rect()
        self.show_life = pg.transform.scale(self.show_life, (26, 35))
        def set_title_image(self):
        self.title_image = pg.image.load(path.join(self.img_dir, "logo.png")).convert()
        self.title_image.set_colorkey(BLACK)
        self.title_rect = self.title_image.get_rect()
        self.title_rect.center = WIDTH/2+300, HEIGHT/2+50
        self.t_w, self.t_h = self.title_rect.width//2, self.title_rect.height//2
        self.title_image = pg.transform.scale(self.title_image,(self.t_w, self.t_h))
        self.screen.blit(self.title_image, self.title_rect)

        def draw_lives(self, surf, x, y, lives, img):
        for i in range(lives):
        img_rect = img.get_rect()
        img_rect.x = x + 30*i
        img_rect.y = y
        surf.blit(img, img_rect)

    def new(self):
        #开始一个新的游戏

        #创建 sprite 组
        self.score = 0 #初始化分数

        # 游戏道具的分数
        self.carrot_num = 0
        self.bcoin_num = 0
        self.gcoin_num = 0
        self.scoin_num = 0

        self.world = 1 #设立初始关卡
        self.all_sprites = pg.sprite.LayeredUpdates()
        self.platforms = pg.sprite.Group()
        self.powerups = pg.sprite.Group()
        self.mobs = pg.sprite.Group()
        self.clouds = pg.sprite.Group()

        self.carrots = pg.sprite.Group()
        self.bronzes = pg.sprite.Group()
        self.silvers = pg.sprite.Group()
        self.golds = pg.sprite.Group()

        self.player = Player(self)
        for plat in PLATFORM_LIST: #获取平台列表中的平台
        Platform(self, *plat)
        self.mob_timer = 0
        pg.mixer.music.load(path.join(self.snd_dir, 'bgm.ogg'))
        for i in range(5):
        c = Cloud(self)
        c.rect.x -= WIDTH
        self.run()

    def run(self):
        #游戏循环
        pg.mixer.music.play(loops=-1)
        self.playing = True
        while self.playing:
        self.clock.tick(FPS)     #控制游戏帧数
        self.events()            #游戏中会碰到的事件
        self.update()            #更新游戏状态
        self.draw()              #绘制游戏画面
        pg.mixer.music.fadeout(500) #游戏结束音乐逐渐消失（500 毫秒）

    def update(self):
```

```
#游戏循环 - 更新游戏状态
self.all_sprites.update()

if self.lives > 5:
     self.lives = 5

# 产生一个敌人
now = pg.time.get_ticks()
if now - self.mob_timer > 5000 + rd.choice([-1000, -500, 0, 500, 1000]):
     self.mob_timer = now
     Mob(self)

# 检查玩家是否碰到平台（只在掉落的时候）
if self.player.vel.y > 0:
     hits = pg.sprite.spritecollide(self.player, self.platforms, False)
     if hits:
          lowest = hits[0]
          for hit in hits:
               if hit.rect.bottom > lowest.rect.bottom:
                    lowest = hit #检测撞击到的平台哪个最低
               if self.player.pos.y < lowest.rect.centery:
                    self.player.pos.y = lowest.rect.top
                    self.player.vel.y = 0
                    self.player.jumping = False
               self.player.walking = True
#宽度到达 width/2 时屏幕整体左移动
if self.player.pos.x > WIDTH / 2:
     if rd.randrange(100) < 1:
          Cloud(self)

     self.player.pos.x += -max(abs(self.player.vel.x),2)
     for plat in self.platforms: #平台跟着移动
          plat.rect.x += -max(abs(self.player.vel.x),2)
          if plat.rect.right < 0: #平台移出左屏幕就 kill
               plat.kill()

     for powerup in self.powerups: #津贴跟着移动
          powerup.rect.x += -max(abs(self.player.vel.x),2)
          if powerup.rect.right < 0:
               powerup.kill()
               self.score += 10

     for carrot in self.carrots: #萝卜跟着移动
          carrot.rect.x += -max(abs(self.player.vel.x),2)
          if carrot.rect.right < 0:
               carrot.kill()
     for bronze in self.bronzes: #铜跟着移动
          bronze.rect.x += -max(abs(self.player.vel.x),2)
          if bronze.rect.right < 0:
               bronze.kill()
     for silver in self.silvers: #银跟着移动
          silver.rect.x += -max(abs(self.player.vel.x),2)
          if silver.rect.right < 0:
               silver.kill()
     for gold in self.golds: #金跟着移动
          gold.rect.x += -max(abs(self.player.vel.x),2)
          if gold.rect.right < 0:
               gold.kill()
     for mob in self.mobs:
          mob.rect.x += -max(abs(self.player.vel.x),2)
          if mob.rect.right < 0: #敌人移出左屏幕就 kill
               mob.kill()

     r = rd.choice([2, 2.5, 3, 3.5])
     for cloud in self.clouds: #云彩的移动
          cloud.rect.x += -max(abs(self.player.vel.x / r),2)
          if cloud.rect.right < 0:
               cloud.kill()
#玩家 die
```

```
        if self.player.rect.bottom > HEIGHT and self.player.rect.bottom < HEIGHT + 30:
            pg.mixer.music.stop()
            self.die_sound1.play()
        if self.player.rect.top > HEIGHT:
            self.playing = False
            self.lives -= 1
        #彻底退出游戏
        if self.lives == 0:
            self.playing == False
            self.game_over = True
        #玩家捡到游戏津贴
        pow_hits = pg.sprite.spritecollide(self.player, self.powerups, True)
        for pow in pow_hits:
            if pow.type == 'boost':
                self.score += 500
                self.player.jumping = False
        carrot_hits = pg.sprite.spritecollide(self.player, self.carrots, True)
        for carrot in carrot_hits:
            if carrot.type == 'carrot':
                self.up_sound.play()
                self.score += 1000
                self.c_score += 1
                self.lives += 1
        bronze_hits = pg.sprite.spritecollide(self.player, self.bronzes, True)
        for bronze in bronze_hits:
            if bronze.type == 'bronze':
                self.coin_sound.play()
                self.score += 100
                self.b_score += 1
        silver_hits = pg.sprite.spritecollide(self.player, self.silvers, True)
        for silver in silver_hits:
            if silver.type == 'silver':
                self.coin_sound.play()
                self.score += 300
                self.s_score += 1
        gold_hits = pg.sprite.spritecollide(self.player, self.golds, True)
        for gold in gold_hits:
            if gold.type == 'gold':
                self.coin_sound.play()
                self.score += 500
                self.g_score += 1
                self.lives = 5
        #玩家碰到敌人
        mob_hits = pg.sprite.spritecollide(self.player, self.mobs, False,
        pg.sprite.collide_mask)
        if mob_hits:
            self.die_sound1.play()
            pg.mixer.music.stop()
            self.playing = False
            self.lives -= 1

        #随机产生平台
        while len(self.platforms) < 18:
            p = Platform2(self, rd.randrange(1280, 1880, 200),
                          rd.randrange(int(HEIGHT/4), HEIGHT - 120, 180))
            hits = pg.sprite.spritecollide(p, self.platforms, False)
            if not hits:
                self.platforms.add(p)
                self.all_sprites.add(p)

    def events(self):
        #游戏循环 - 游戏中会碰到的事件
        for event in pg.event.get():
        #检查是否要关闭窗口
            if event.type == QUIT:
                if self.playing:
                    self.playing = False
                self.running = False
            if event.type == KEYDOWN:
```

```
                if event.key == K_SPACE or event.key == K_UP or event.key == K_w:
                    self.player.jump()
            if event.type == KEYUP:
                if event.key == K_SPACE or event.key == K_UP or event.key == K_w:
                    self.player.jump_cut()

    def draw(self):
    #游戏循环 - 绘制游戏画面
        self.screen.fill(SKYBLUE)
        self.all_sprites.draw(self.screen)
        self.draw_text_up("BUNNY", int(HEIGHT/20), WHITE, 0.06*WIDTH, 0.06*HEIGHT)
        self.draw_text_up("%06d"%self.score, int(HEIGHT/20), WHITE, 0.06*WIDTH,
        (0.06+1/20)*HEIGHT)
        self.screen.blit(self.show_carrot, self.show_carrot_rect)
        self.draw_text_up("%03d"%self.c_score, int(HEIGHT/20), WHITE, 0.2*WIDTH,
        (0.06+1/20)*HEIGHT)
        self.screen.blit(self.show_b_coin, self.show_b_coin_rect)
        self.draw_text_up("%03d"%self.b_score, int(HEIGHT/20), WHITE, 0.4*WIDTH,
        (0.06+1/20)*HEIGHT)
        self.screen.blit(self.show_s_coin, self.show_s_coin_rect)
        self.draw_text_up("%03d"%self.s_score, int(HEIGHT/20), WHITE, 0.6*WIDTH,
        (0.06+1/20)*HEIGHT)
        self.screen.blit(self.show_g_coin, self.show_g_coin_rect)
        self.draw_text_up("%03d"%self.g_score, int(HEIGHT/20), WHITE, 0.8*WIDTH,
        (0.06+1/20)*HEIGHT)
        self.draw_lives(self.screen, WIDTH - 150, 10, self.lives, self.show_life)
        pg.display.flip()

    def draw_text_up(self, text, size= int(HEIGHT/20), color=WHITE, x=WIDTH/2, y = 15):
        #文本显示（上方）
        font = pg.font.Font(self.font_name, size)
        text_surface = font.render(text, True, color)
        text_rect = text_surface.get_rect()
        text_rect.x = x
        text_rect.y = y
        self.screen.blit(text_surface, text_rect)

    def draw_text(self, text, size= int(HEIGHT/20), color=WHITE, x=WIDTH/2, y = 15):
        #文本显示
        font = pg.font.Font(self.font_name, size)
        text_surface = font.render(text, True, color)
        text_rect = text_surface.get_rect()
        text_rect.midtop = (x, y)
        self.screen.blit(text_surface, text_rect)

    def wait_for_key(self):
        waiting = True
        while waiting:
            self.clock.tick(FPS)
            for event in pg.event.get():
                if event.type == QUIT:
                    waiting = False
                    self.running = False
                if event.type == KEYUP:
                    if event.key == K_KP_ENTER or event.key == K_SPACE or K_RETURN:
                    waiting = False

    def show_start_screen(self):
        #开始游戏画面
        self.screen.fill(BGCOLOR)
        self.set_title_image()
        self.draw_text("Arrows to move, Space to jump", 22, WHITE, WIDTH / 2, HEIGHT / 2+100)
        self.draw_text("Press a key to play", 22, WHITE, WIDTH / 2, HEIGHT * 3 / 4)
        self.draw_text("High Score: " + str(int(self.highscore)), 22, WHITE, WIDTH / 2, 15)
        pg.display.flip()
        self.wait_for_key()

    def show_go_screen(self):
    #继续游戏画面
```

```
        if not self.running:
            return
        self.draw_text("GAME OVER", 100, WHITE, WIDTH / 2, HEIGHT / 4)
        self.draw_text("Score: " + str(int(self.score)), 22, WHITE, WIDTH / 2, HEIGHT / 2)
        self.draw_text("Carrot: " + str(int(self.c_score)), 22, WHITE, WIDTH / 2, HEIGHT / 2 +
        65)
        self.draw_text("Bronze: " + str(int(self.b_score)), 22, WHITE, WIDTH / 2, HEIGHT / 2 +
        90)
        self.draw_text("Silver: " + str(int(self.s_score)), 22, WHITE, WIDTH / 2, HEIGHT / 2 +
        115)
        self.draw_text("Gold: " + str(int(self.g_score)), 22, WHITE, WIDTH / 2, HEIGHT / 2 + 140)
        self.draw_text("Press a key to play again", 44, WHITE, WIDTH / 2, HEIGHT * 3 / 4)
        if self.score > self.highscore:
            self.highscore = self.score
            self.draw_text("NEW HIGH SCORE!", 22, WHITE, WIDTH / 2, HEIGHT / 2 + 30)
            with open(path.join(self.dir, HS_FILE), 'w') as f:
                f.write(str(int(self.score)))
        else:
            self.draw_text("High Score: " + str(int(self.highscore)), 22, WHITE, WIDTH / 2,
            HEIGHT / 2 + 30)
        pg.display.flip()
        self.wait_for_key()

    def game_over_screen(self):
        if not self.running:
            return
        self.draw_text("GAME OVER", 100, WHITE, WIDTH / 2, HEIGHT / 4)
        self.draw_text("Score: " + str(int(self.score)), 44, WHITE, WIDTH / 2, HEIGHT / 2)
        self.draw_text("Press a key to return to the start screen", 44, WHITE, WIDTH / 2, HEIGHT
        * 3 / 4)
        if self.score > self.highscore:
            self.highscore = self.score
            self.draw_text("High Score: " + str(int(self.highscore)), 22, WHITE, WIDTH / 2,
            HEIGHT/ 2 + 40)
            with open(path.join(self.dir, HS_FILE), 'w') as f:
                f.write(str(int(self.score)))
        else:
            self.draw_text("High Score: " + str(int(self.highscore)), 22, WHITE, WIDTH / 2,
            HEIGHT / 2 + 40)
        pg.display.flip()
        self.wait_for_key()
        self.game_over = False
        self.lives = 5

g = Game()
g.show_start_screen()
while g.running:
g.new() #新游戏
if g.playing == False and g.game_over == False:
g.show_go_screen()#继续游戏屏幕
if g.playing == False and g.game_over == True:
g.game_over_screen()

pg.quit()
```

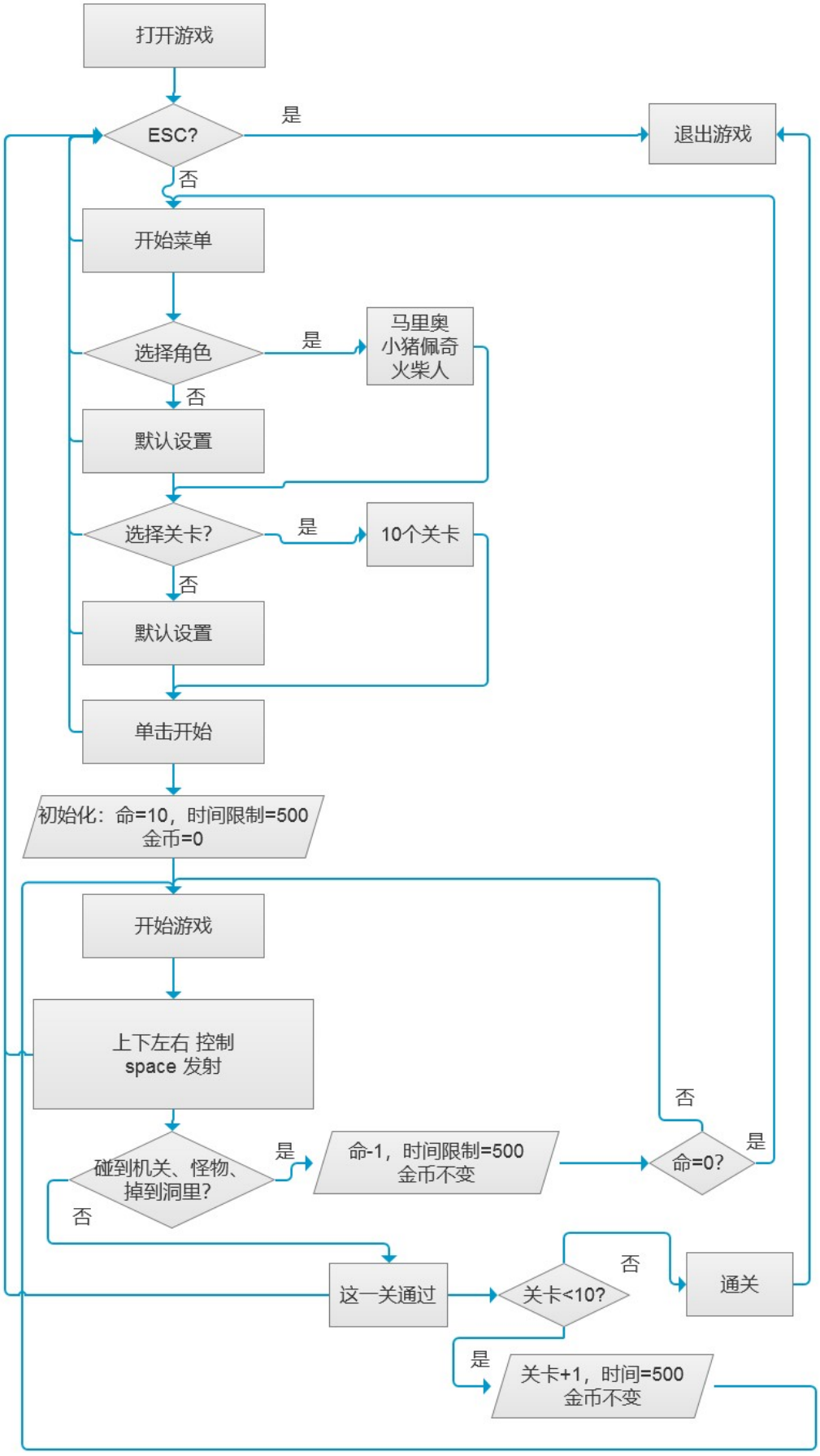
五、运行结果

运行的时候出现一个新的窗口，该窗口即为游戏主画面，画面中初始化一只兔子，该兔子能够在玩家的键盘控制下完成上下左右移动控制，并能完成不同关卡，并能实现最高分的保存。

六、结果分析

游戏的画面模仿部分超级玛丽的经典画面，只不过在贴图和角色上做了变换，添加了一些新的部分，基本上能够符合游戏的要求，游戏设计不管是在想象力上、还是在审美上都能够脍炙人口，实现在创造的虚拟世界中让玩家能够尽情享受游戏的乐趣。

附一：游戏大致程序框图


打开游戏
ESC?
是
退出游戏
否
开始菜单
选择角色
是
马里奥
小猪佩奇
火柴人
否
默认设置
选择关卡?
是
10个关卡
否
默认设置
单击开始
初始化：命=10，时间限制=500
金币=0
开始游戏
上下左右 控制
space 发射
碰到机关、怪物、
掉到洞里?
是
命-1，时间限制=500
金币不变
命=0?
是
否
否
这一关通过
关卡<10?
否
通关
是
关卡+1，时间=500
金币不变