计算机学院 2021 级

研究与开发实践结题报告

计算机学院

2022 年 12 月 11 日

项目名：基于 PyGame 的地图闯关游戏

“Ori’s Adventure”

学号：2021141460286

姓名：罗玉婷

指导教师：张轶

目录

一、游戏概述

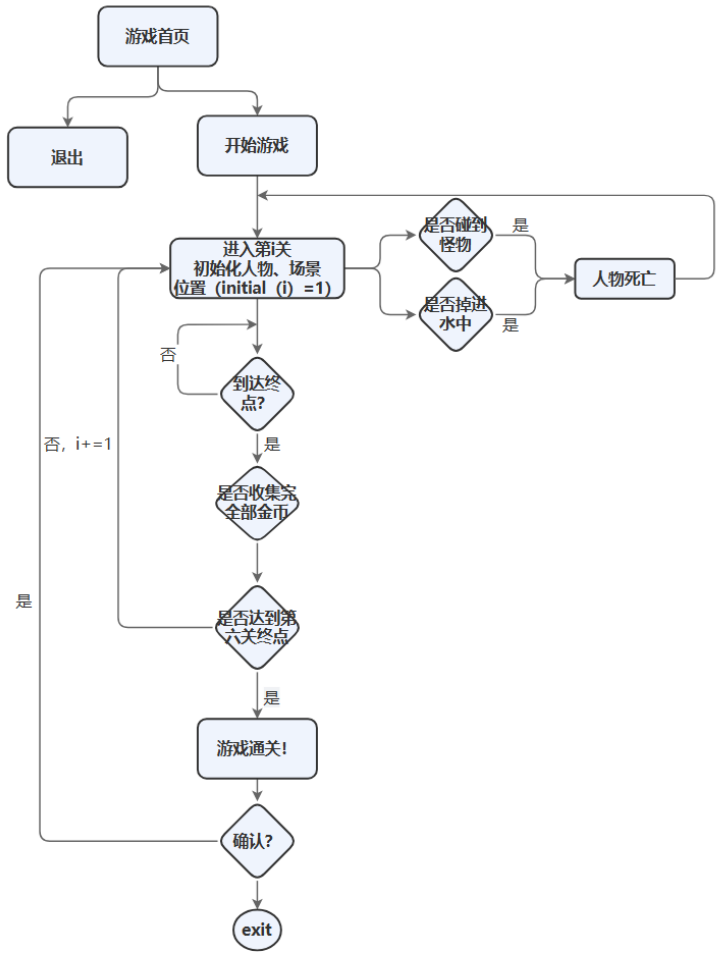
1.1 游戏介绍

本游戏“Ori’s Adventure”是一款集冒险、收集、闯关三种元素为一体的 2D 地图像素风游戏。在本游戏中，玩家需要通过控制主人公 ori 的跳跃与行走来躲避、穿越障碍并收集金币，最后达到终点才能通关进入下一关卡。游戏总设计为六关，地图设计及障碍元素由浅入深、由简到繁，随着探索的深入，会逐渐遭遇到河流、怪物、水、纵向移动的地砖等障碍。同时穿插了各类不同效果的音乐和音效，增加了游戏的趣味性。

1.2 开发环境

本游戏是在 win11 的操作系统上采用基于 pygame 库的海龟编辑器进行编码，代码语言类别 100%为 python。

1.3 游戏流程



二、游戏内容

2.1 游戏规则及操作方法

规则：

规则 1：玩家与河流、怪物的碰撞视为死亡，游戏结束，可选择初始化本关卡；

规则 2：玩家通关条件为收集完本关卡内所有金币且到达终点，两条件缺一不可；

规则 3：玩家可通过人物的跳跃等来规避障碍但最多只可进行一段跳；

规则 4：玩家可以判断地砖垂直、水平移动的时机进行行动从而穿越平台跨越；

操作方法：

1. 游戏界面内的提示按钮，诸如开始游戏、重玩本关、退出等，通过玩家鼠标事件点击进行操作；
2. 人物的控制：键盘上的←↑→键分别控制人物向左移动、向上跳跃以及向右移动，而且跳跃逻辑与移动逻辑是并行的，玩家可以同时按下移动键和跳跃键，从而实现某方向的横向跳跃。

2.2 游戏主体内容

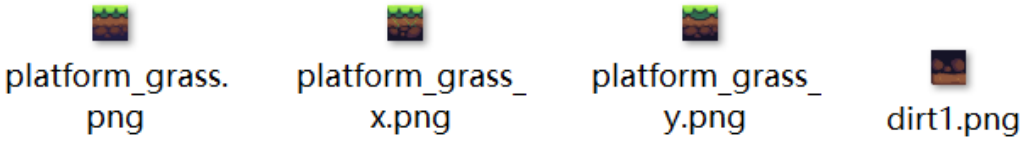
2.2.1 游戏首页

游戏首页包含两大操作：start（开始游戏，进入第一关）、exit（退出游戏），左上角窗口名称 ori’s adventure 标明本游戏名字，首页背景为橙黄色调的彩霞云朵

图案，使人心情上扬。同时操作 ui 与背景色调一致，避免视觉冲突

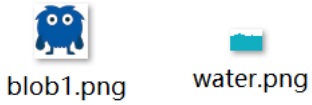
2.2.2 关卡介绍总览

- **地形介绍：**



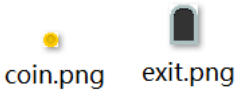
一共有三种草地类型，platform_grass 代表固定不动的草地，platform_grass_x 代表会左右来回移动的草地，platform_grass_y 代表会上下来回移动的草地；还有一种地形 dirt 代表地基与游戏窗口边界

- **障碍介绍**



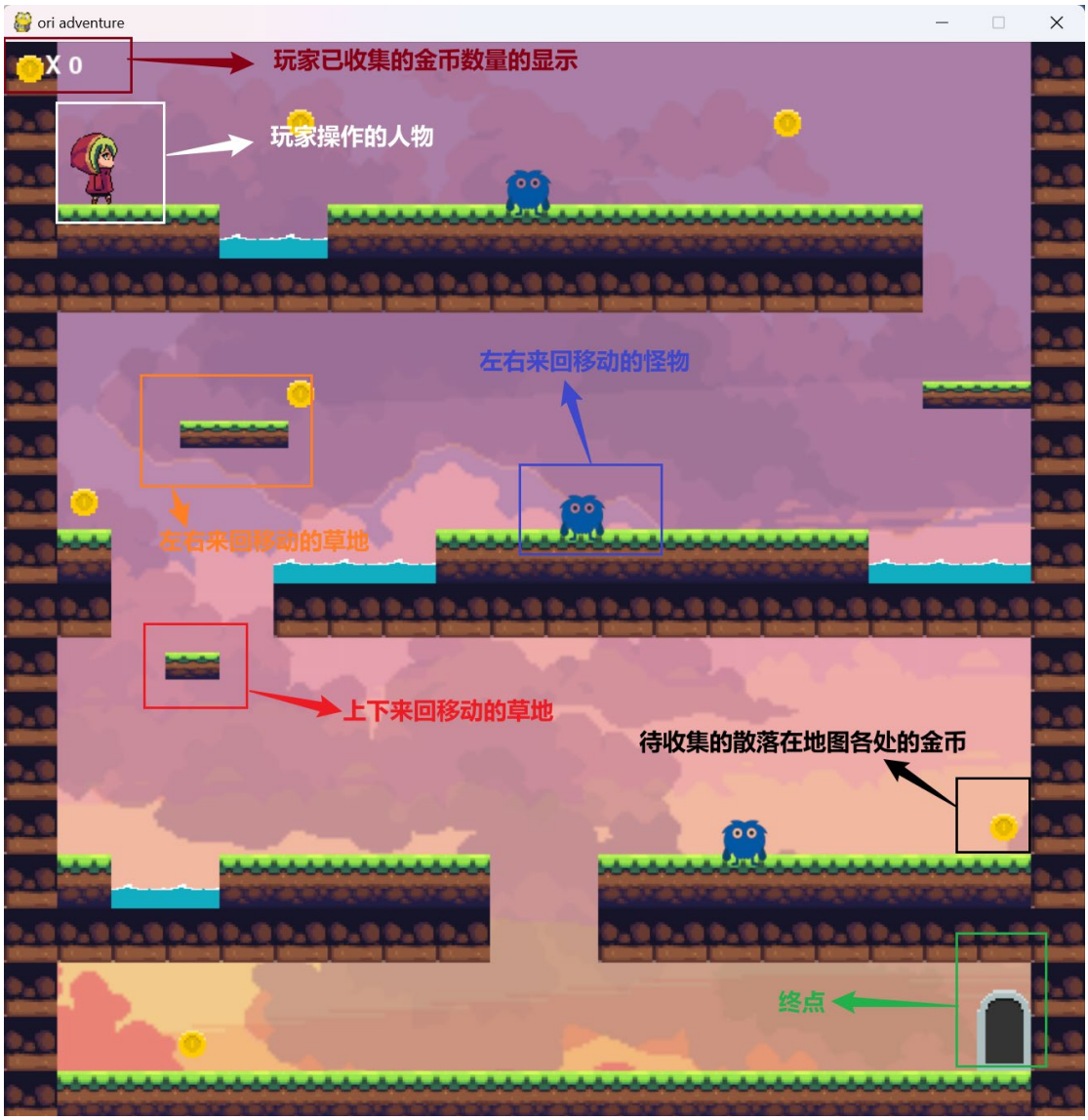
Blob 代表蓝色小怪物，他会自动的左右来回移动增加巡守范围，water 代表水流，人物不可进入

- **关卡要素**



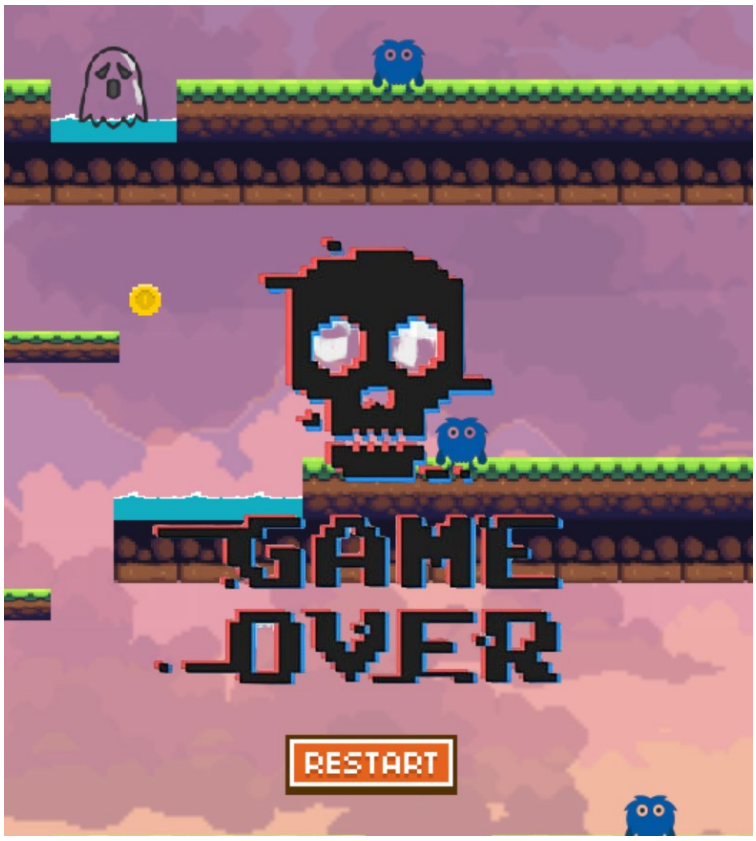
Coin 是人物必须要收集的金币，exit 是终点大门，人物收集完金币后终点才有效，否则为无效碰撞无法进入下一关

2.2.3 第一关

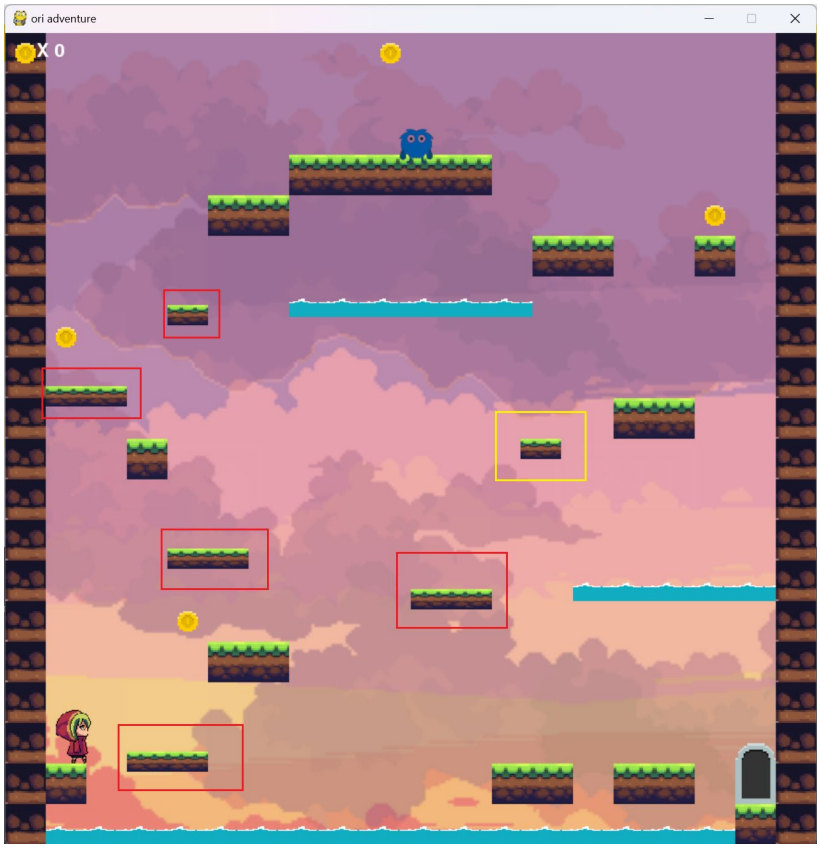


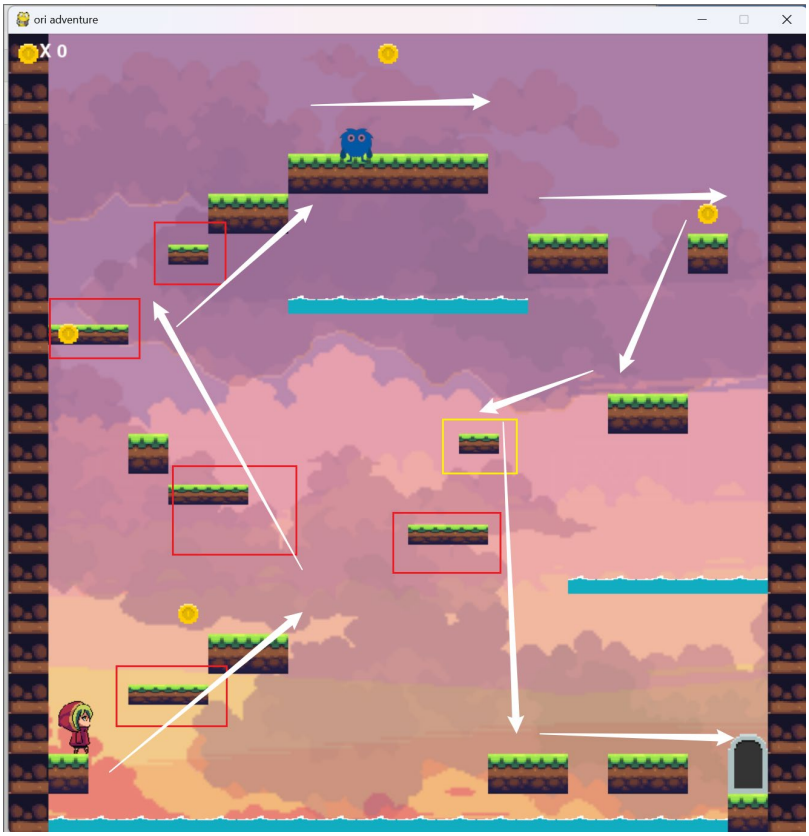
每一关人物的初始出生位置都不相同，玩家需要根据对自身位置的判断与终点的距离和关卡的设计，思考自己如何能合理拿到所有的金币进入下一关。在第一关中，地图的设计非常清晰明了，人物只需由上到下躲避怪物拿到所有金币即可进入终点。同时怪物的位置居于一长段草地的中间，给玩家的可移动范围留有非常充足的余地，比较容易通过规避。

当游戏人物死亡，游戏画面会立即终止，人物贴图变为幽灵同时从死亡的地方右下至上慢慢飘起最后停在屏幕上方，给游戏带来一些动态诙谐感，同时弹出游戏失败的弹窗，选择 restart 按钮即可重玩本关；当人物通关后则会直接跳转进入下一关



2.2.4 第二关

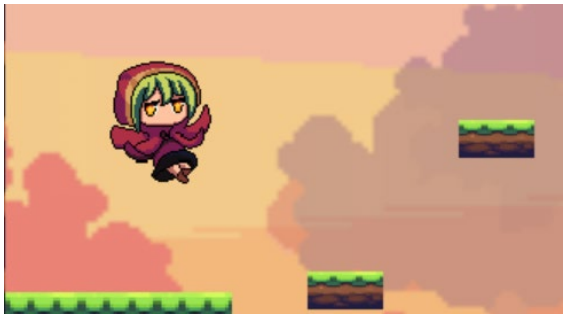




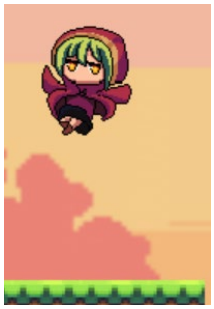
第二关我截取了两张图像，可以看到，人物的出生位置和终点都相较于第一关发生了较大的变化。通过对所框选区域的上下对比即可大致感受出移动草地的运动范围，红色框代表上下移动的草地，黄色框内代表左右移动的草地，因为，一些障碍是玩家跳跃无法到达的，因此玩家需要把握这些移动草地的运动节奏，判断自己跳跃和移动的关键时机从而到达新的平台

白色箭头指示了玩家通关关卡的大致运动方向。

人物的跳跃动效，当人物向右跳跃时，显示为



当人物向左跳跃时，显示为。人物的斗篷状态略有差距。



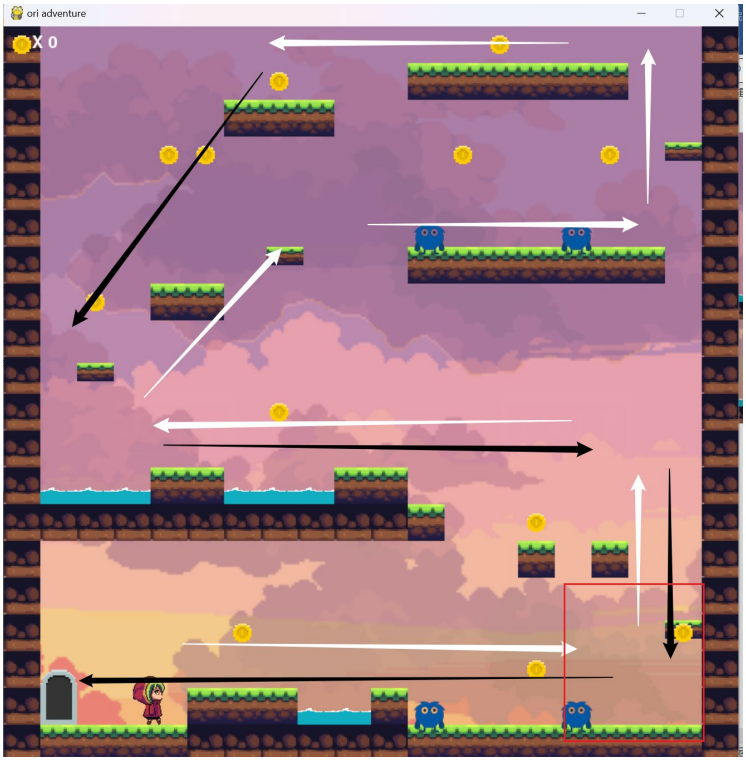
2.2.5 第三关

第三关的设计如上图所示，第三关侧重于考验玩家对跳跃时机的把握，难度较小。

2.2.6 第四关

第四关主要考验玩家对怪物的躲避操作，特别是界面下方的两个连续的怪物，他们同时在左右移动，玩家要把控好自己跳跃和移动的时机，难度比较大，非常容易死亡。同时界面上方有三个连续的躲避怪物且吃金币的操作，玩家要同时把控二者的节奏。

2.2.7 第五关



第五关比较创新的两个地方在于，（1）人物出生在终点的旁边，但这并不代表人物可以速通，由于吃完所有金币的条件，代表人物必须经历这些障碍**两次**，极大提高了人物的操作难度；（2）右下角的设计，人物要在规避怪物的同时跳上垂直运动的平台

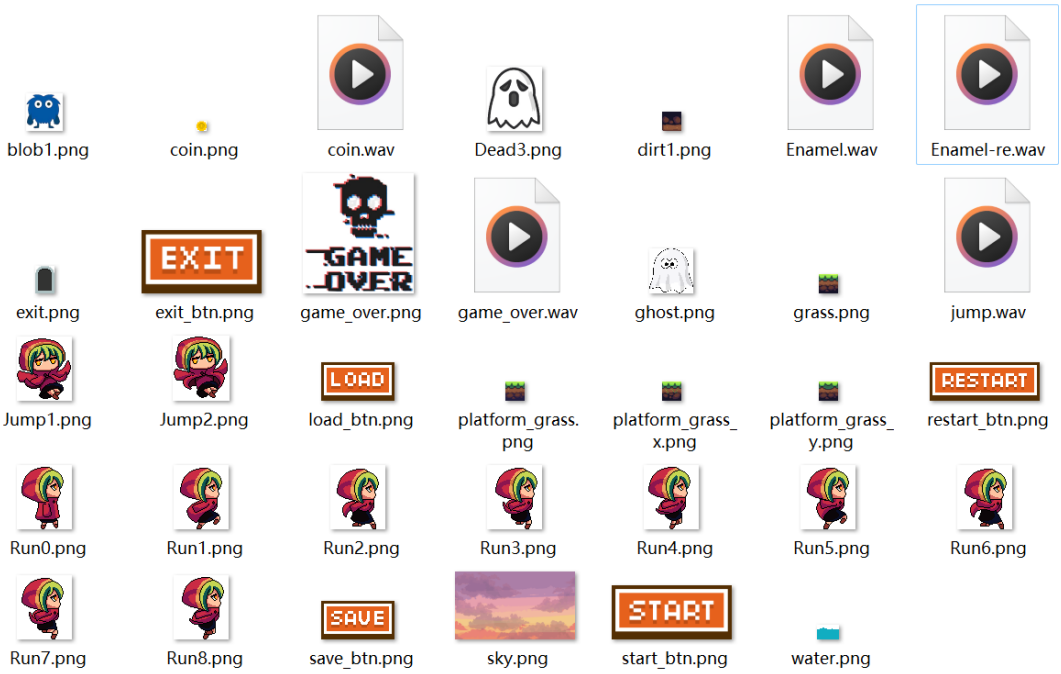
2.2.8 第六关

第六关是最后一关，难度较第五关并无太大提升，主要考验人物的跳跃和对怪物的规避，有些跳跃比较考验技巧。游戏设置了多次跳跃和多个移动平台。

上图是我游玩通关的场景，而后点击 restart 则返回第一关。游戏所有 内容游玩流程结束。

三、游戏核心逻辑

3.1 游戏素材选取



游戏的素材包括：

• 音乐与音效：

音效（金币碰撞 coin.wav |死亡 game_over.wav | 跳跃 jump.wav）、背景音乐（enamel-re.wav）

• 游戏按钮：

开始游戏（start）、退出游戏（exit）重玩本关（restart）

• 人物逻辑

人物的行走（run）、人物的跳跃（jump）

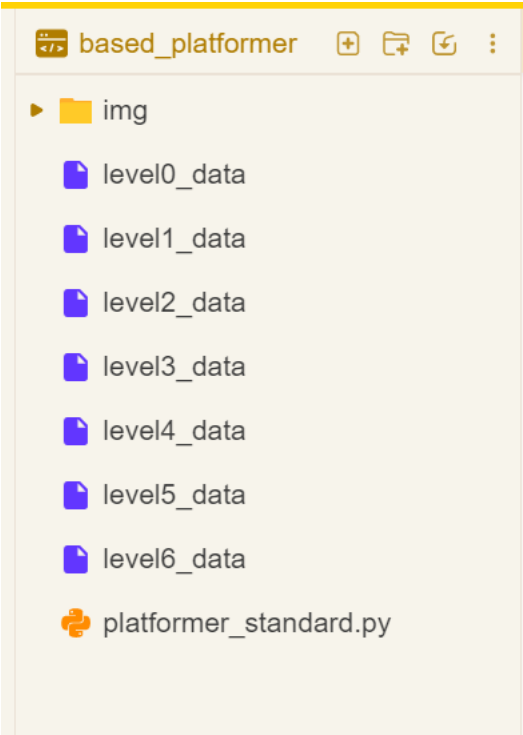
人物的运动素材初始只有向右方向的，代码处理中向左的逻辑采用了 pygame 自带的图像镜像反转的函数来获取。

• 地图部分

在前述地图介绍中已经有所介绍这里不再赘述。

此外，关于人物和地图的所有贴图都是以透明通道的 png 格式应用。原始素材来源于国外的开源游戏素材网站 craftpix，我下载之后通过图像处理软件 进行了切割和抠图处理得到最终的游戏素材。

3.2 项目结构总览



Img 文件夹内存储了所有的游戏素材
Level[i].data 是游戏关卡数据
Platformer_standard.py 是游戏主要工程文件

3.3 地图显示

• 地图的初始化绘制逻辑：

函数 draw_world()用于根据提供的 world_data 列表（二维数组）绘制游戏世界的图像。游戏世界的不同元素（如土块、草块、敌人、移动平台、水、金币、出口等）都有其对应的标识符，通过检查 world_data 中的值，选择合适的图像进行绘制。

对于 world_data 中的每个元素，
如果元素的值是 1，表示土块，使用 dirt_img 作为图像，
如果元素的值是 2，表示草块，使用 grass_img 作为图像，
如果元素的值是 3，表示敌人，使用 blob_img 作为图像，
如果元素的值是 4，表示横向移动平台，使用 platform_x_img 作为图像，
如果元素的值是 5，表示纵向移动平台，使用 platform_y_img 作为图像，
如果元素的值是 6，表示水，使用 water_img 作为图像，
如果元素的值是 7，表示金币，使用 coin_img 作为图像，
如果元素的值是 8，表示出口，使用 exit_img 作为图像，

```
def draw_world():
    for row in range(20):
        for col in range(20):
            if world_data[row][col] > 0:
                if world_data[row][col] == 1:
                    #dirt blocks
                    img = pygame.transform.scale(dirt_img, (tile_size, tile_size))
                    screen.blit(img, (col * tile_size, row * tile_size))
                if world_data[row][col] == 2:
                    #grass blocks
                    img = pygame.transform.scale(grass_img, (tile_size, tile_size))
                    screen.blit(img, (col * tile_size, row * tile_size))
                if world_data[row][col] == 3:
                    #enemy blocks
                    img = pygame.transform.scale(blob_img, (tile_size, int(tile_size * 0.75)))
                    screen.blit(img, (col * tile_size, row * tile_size + (tile_size * 0.25)))
                if world_data[row][col] == 4:
                    #horizontally moving platform
                    img = pygame.transform.scale(platform_x_img, (tile_size, tile_size // 2))
                    screen.blit(img, (col * tile_size, row * tile_size))
                if world_data[row][col] == 5:
                    #vertically moving platform
                    img = pygame.transform.scale(platform_y_img, (tile_size, tile_size // 2))
                    screen.blit(img, (col * tile_size, row * tile_size))
                if world_data[row][col] == 6:
                    #water
                    img = pygame.transform.scale(water_img, (tile_size, tile_size // 2))
                    screen.blit(img, (col * tile_size, row * tile_size + (tile_size // 2)))
                if world_data[row][col] == 7:
                    #coin
                    img = pygame.transform.scale(coin_img, (tile_size // 2, tile_size // 2))
                    screen.blit(img, (col * tile_size + (tile_size // 4), row * tile_size + (tile_size // 4)))
                if world_data[row][col] == 8:
                    #exit
                    img = pygame.transform.scale(exit_img, (tile_size, int(tile_size * 1.5)))
                    screen.blit(img, (col * tile_size, row * tile_size - (tile_size // 2)))
```

• 关卡的载入

World 类将游戏世界的图像和元素进行初始化和绘制，以便在主游戏循环中使用： 在初始化方法中，该类接受 data 参数（包含了表示地图布局的数据）。遍历数据，加载相应的图像，创建不同的游戏元素（如土块、草块、敌人、平台、水、金币和出口）及其位置

关卡列表 tile_list，用于存储游戏中所有的关卡信息。绘制方法 draw 中，遍历 tile_list，将每个关卡的图像绘制到屏幕上的相应位置。

```
class World():
    def __init__(self, data):
        self.tile_list = []

        # load images
        dirt_img = pygame.image.load('img/dirt1.png')
        grass_img = pygame.image.load('img/platform_grass.png')

        row_count = 0
        for row in data:
            col_count = 0
            for tile in row:
                if tile == 1:
                    img = pygame.transform.scale(
                        dirt_img, (tile_size, tile_size))
                    img_rect = img.get_rect()
                    img_rect.x = col_count * tile_size
                    img_rect.y = row_count * tile_size
                    tile = (img, img_rect)
                    self.tile_list.append(tile)
                if tile == 2:
                    img = pygame.transform.scale(
                        grass_img, (tile_size, tile_size))
                    img_rect = img.get_rect()
                    img_rect.x = col_count * tile_size
                    img_rect.y = row_count * tile_size
                    tile = (img, img_rect)
                    self.tile_list.append(tile)
                if tile == 3:
                    blob = Enemy(col_count * tile_size,
                                 row_count * tile_size + 15)
                    blob_group.add(blob)
                if tile == 4:
                    platform = Platform(
                        col_count * tile_size, row_count * tile_size, 1, 0)
                    platform_group.add(platform)
                if tile == 5:
                    platform = Platform(
                        col_count * tile_size, row_count * tile_size, 0, 1)
                    platform_group.add(platform)
                if tile == 6:
                    water = Water(col_count * tile_size,
                                  row_count * tile_size + (tile_size // 2))
                    water_group.add(water)
                if tile == 7:
                    coin = Coin(col_count * tile_size + (tile_size // 2),
                                row_count * tile_size + (tile_size // 2))
                    coin_group.add(coin)
                if tile == 8:
                    exit = Exit(col_count * tile_size, row_count *
                                tile_size - (tile_size // 2))
                    exit_group.add(exit)
                col_count += 1
            row_count += 1

    def draw(self):
        for tile in self.tile_list:
            screen.blit(tile[0], tile[1])
```

- **地图素材的初始化与更新**

  地图素材的每个类都继承了 pygame.sprite.Sprite 类，可以被 Pygame 的精灵

系统管理和绘制。类的实例在游戏中用于表示不同的游戏元素，并通过更新方法来处理其元素的状态和位置：

Enemy 类表示敌人（blob）。初始化方法 __init__ 用于设置敌人的初始位置和移动方向， update 方法用于更新敌人的位置和移动状态。

Platform 类表示平台。初始化方法 __init__ 用于设置平台的初始位置、水平和垂直移动速度，以及一个 update 方法用于更新平台的位置和移动状态。

Water 类表示水。初始化方法 __init__ 用于设置水的初始位置， update 方法用于更新水的位置。

Coin 类表示金币。初始化方法 __init__ 用于设置金币的初始位置，update 方法用于更新金币的位置。

Exit 类表示终点。初始化方法 __init__ 用于设置出口的初始位置， update 方法用于更新出口的位置。

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load('img/blob1.png')
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.move_direction = 1
        self.move_counter = 0

    def update(self):
        self.rect.x += self.move_direction
        self.move_counter += 1
        if abs(self.move_counter) > 50:
            self.move_direction *= -1
            self.move_counter *= -1
```

```
class Platform(pygame.sprite.Sprite):
    def __init__(self, x, y, move_x, move_y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/platform_grass.png')
        self.image = pygame.transform.scale(img, (tile_size, tile_size // 2))
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.move_counter = 0
        self.move_direction = 1
        self.move_x = move_x
        self.move_y = move_y

    def update(self):
        self.rect.x += self.move_direction * self.move_x
        self.rect.y += self.move_direction * self.move_y
        self.move_counter += 1
        if abs(self.move_counter) > 50:
            self.move_direction *= -1
            self.move_counter *= -1
```

```
class Water(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/water.png')
        self.image = pygame.transform.scale(img, (tile_size, tile_size // 2))
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y


class Coin(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/coin.png')
        self.image = pygame.transform.scale(
            img, (tile_size // 2, tile_size // 2))
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)


class Exit(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/exit.png')
        self.image = pygame.transform.scale(
            img, (tile_size, int(tile_size * 1.5)))
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
```

3.4 人物控制逻辑

在每个游戏帧中更新玩家的状态，根据输入进行移动和跳跃，处理动画效果，并模拟重力影响。

按下 K_SPACE 键，并且玩家没有跳跃且不在空中，播放跳跃音效，并设置垂直速度为 -15，表示向上跳跃，并将 self.jumped 状态设置为 True；如果释放 K_SPACE 键，将 self.jumped 设置为 False。

按下 K_LEFT 键，水平速度 dx 设置为 -5，方向 self.direction 设置为 -1，表示向左移动；

按下 K_RIGHT 键，水平速度 dx 设置为 5，方向 self.direction 设置为 1，表示向右移动。

添加重力。增加垂直速度，模拟重力效果，但速度受到限制，不会无限加速。

跳跃处理：如果玩家在空中，根据方向设置相应的跳跃图像。

```
    if game_over == 0:
        # get keypresses
        key = pygame.key.get_pressed()
        if key[pygame.K_SPACE] and self.jumped == False and self.in_air == False:
            jump_fx.play()
            self.vel_y = -15
            self.jumped = True
        if key[pygame.K_SPACE] == False:
            self.jumped = False
        if key[pygame.K_LEFT]:
            dx -= 5
            self.counter += 1
            self.direction = -1
        if key[pygame.K_RIGHT]:
            dx += 5
            self.counter += 1
            self.direction = 1
        if key[pygame.K_LEFT] == False and key[pygame.K_RIGHT] == False:
            self.counter = 0
            self.index = 0
            if self.direction == 1:
                self.image = self.images_right[self.index]
            if self.direction == -1:
                self.image = self.images_left[self.index]
########################jump_image#######################
        # handle animation
        if self.in_air:
            if self.direction == 1:
                self.image = self.jump_image_right
            elif self.direction == -1:
                self.image = self.jump_image_left
        else:
            if self.counter > walk_cooldown:
                self.counter = 0
                self.index += 1
                if self.index >= len(self.images_right):
                    self.index = 0
                if self.direction == 1:
                    self.image = self.images_right[self.index]
                if self.direction == -1:
                    self.image = self.images_left[self.index]
```

3.5 碰撞检测

碰撞检测：

采用 pygame 自带的 spritecollide 函数进行碰撞检测

与草坪、土地的碰撞检测：

分别检测玩家在 x 、y 方向上是否与砖块发生碰撞。如果是在 x 方向上发生碰撞，，将水平速度 dx 设置为 0，防止穿过砖块。如果是在 y 方向上，根据玩家是在跳跃还是下落中进行处理：

若玩家是在跳跃中（self.vel_y < 0），将垂直速度 dy 调整，使玩家落在砖块上方，并将垂直速度归零。

若玩家是在下落中，将垂直速度 dy 调整，使玩家停在砖块下方，并将垂直速度归零。同时将 self.in_air 设置为 False，表示不再处于空中状态。

与敌人（blob_group）碰撞检测：

如果玩家与敌人或水发生碰撞，游戏结束，并播放游戏结束音效。

与出口（exit_group）碰撞检测：

如果玩家与出口发生碰撞，且地图现存金币数量为 0，游戏胜利

```
# check for collision
self.in_air = True
for tile in world.tile_list:
    # check for collision in x direction
    if tile[1].colliderect(self.rect.x + dx, self.rect.y, self.width, self.height):
        dx = 0
    # check for collision in y direction
    if tile[1].colliderect(self.rect.x, self.rect.y + dy, self.width, self.height):
        # check if below the ground i.e. jumping
        if self.vel_y < 0:
            dy = tile[1].bottom - self.rect.top
            self.vel_y = 0
        # check if above the ground i.e. falling
        elif self.vel_y >= 0:
            dy = tile[1].top - self.rect.bottom
            self.vel_y = 0
            self.in_air = False

# check for collision with enemies
if pygame.sprite.spritecollide(self, blob_group, False):
    game_over = -1
    game_over_fx.play()

# check for collision with water
if pygame.sprite.spritecollide(self, water_group, False):
   game_over = -1
   game_over_fx.play()

# check for collision with exit
if pygame.sprite.spritecollide(self, exit_group, False) and len(coin_group)==1:
    game_over = 1

# check for collision with platforms
for platform in platform_group:
    # collision in the x direction
    if platform.rect.colliderect(self.rect.x + dx, self.rect.y, self.width, self.height):
        dx = 0
    # collision in the y direction
    if platform.rect.colliderect(self.rect.x, self.rect.y + dy, self.width, self.height):
        # check if below platform
        if abs((self.rect.top + dy) - platform.rect.bottom) < col_thresh:
            self.vel_y = 0
            dy = platform.rect.bottom - self.rect.top
        # check if above platform
```

四、技术难点及其解决

4.1 音效音乐的处理

在游戏开发过程中，音乐是一个很关键的部分，最开始我的背景音乐无法循环起来，关卡跳转或者重玩时背景音乐即暂停。最后我通过设置了一个背景音乐播放状态的 bool 变量来检测音乐的播放并在人物死亡，重启游戏时检测该变量的状态并作出相应改变来达到背景音乐循环播放且随着游戏暂停而暂停的效果。

```python
if main_menu == True:
    if playing_music == False:
        # 在点击游戏首页时开始播放音乐
        pygame.mixer.music.play(-1)  # -1 表示无限循环
        playing_music = True
```

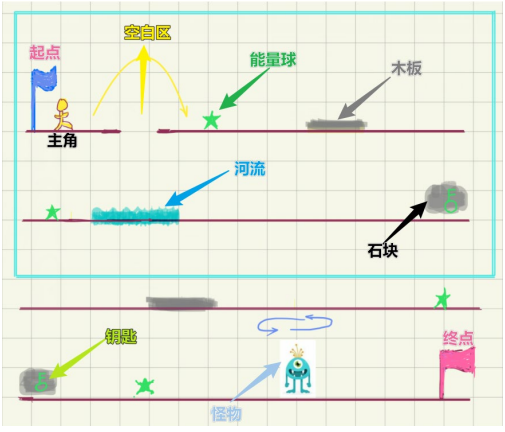
```python
# if player has died
if game_over == -1:
    pygame.mixer.music.stop()
    playing_music = False
    if restart_button.draw():
        world_data = []
        world = reset_level(level)
        game_over = 0
        score = 0
        pygame.mixer.music.play(-1)
        playing_music = True
```

4.2 各类碰撞检测

由于 pygame 自带的碰撞函数使检测贴图之间的碰撞，贴图是矩形的，然而实际素材效果是非线性的，因此在游玩的视觉效果上会呈现出未接触到则发生了碰撞效果的感觉，为了减少此方面的误差，我中途重新调整精细化了贴图的间距边缘大小，使其更贴近于素材实际边界。当然不可避免的是仍然有视觉误差，这是我以后可以改进的方面。

4.3 关卡的设计

本游戏我设计了六关，如何使关卡难度层层递进，关卡设计合理（能够成功通关，保证玩家无法卡 bug 达到速通效果）是我一直困扰的问题，我在关卡设计中事先在平板上做了草图来预先判断关卡的合理性以避免重复绘制关卡的冗余操作。而后在项目实际操作中，每一关我都进行了边操作边修改边调整的步骤，最终达到合理的效果，这部分调试我耗费了很长的时间



4.4 人物的控制

人物的控制是一大难点，人物的走动与人物连续运动图像之间的调用，人物运动方向的转换，人物跳跃（跳跃状态下无法再跳跃、跳跃重力的设置、跳跃姿态与行走状态的转化）是困扰我很久的一部分。经过不断修改调试以及分析和添加各种状态的方法，我才最终实现人物控制逻辑。

```
######################jump_image#######################
            # handle animation
            if self.in_air:
                if self.direction == 1:
                    self.image = self.jump_image_right
                elif self.direction == -1:
                    self.image = self.jump_image_left
            else:
                if self.counter > walk_cooldown:
                    self.counter = 0
                    self.index += 1
                    if self.index >= len(self.images_right):
                        self.index = 0
                    if self.direction == 1:
                        self.image = self.images_right[self.index]
                    if self.direction == -1:
                        self.image = self.images_left[self.index]
```

4.5 地图的绘制。

地图初始化的灵感来源于课堂上老师所教的利用二维数组的值来对应地图素材的变化。在本游戏中，我的游戏地图就是被切割为了 20*20 的二维数组方块，然后将地图素材与 1~8 数字进行一一对应，通过赋值二位数组达到了展现不同地图效果的目的。

```
    if world_data[row][col] == 1:
        #dirt blocks
        img = pygame.transform.scale(dirt_img, (tile_size, tile_size))
        screen.blit(img, (col * tile_size, row * tile_size))
    if world_data[row][col] == 2:
        #grass blocks
        img = pygame.transform.scale(grass_img, (tile_size, tile_size))
        screen.blit(img, (col * tile_size, row * tile_size))
    if world_data[row][col] == 3:
        #enemy blocks
        img = pygame.transform.scale(blob_img, (tile_size, int(tile_size * 0.75)))
        screen.blit(img, (col * tile_size, row * tile_size + (tile_size * 0.25)))
    if world_data[row][col] == 4:
        #horizontally moving platform
        img = pygame.transform.scale(platform_x_img, (tile_size, tile_size // 2))
        screen.blit(img, (col * tile_size, row * tile_size))
    if world_data[row][col] == 5:
        #vertically moving platform
        img = pygame.transform.scale(platform_y_img, (tile_size, tile_size // 2))
        screen.blit(img, (col * tile_size, row * tile_size))
    if world_data[row][col] == 6:
        #water
        img = pygame.transform.scale(water_img, (tile_size, tile_size // 2))
        screen.blit(img, (col * tile_size, row * tile_size + (tile_size // 2)))
    if world_data[row][col] == 7:
        #coin
        img = pygame.transform.scale(coin_img, (tile_size // 2, tile_size // 2))
        screen.blit(img, (col * tile_size + (tile_size // 4), row * tile_size + (tile_size // 4)))
    if world_data[row][col] == 8:
        #exit
        img = pygame.transform.scale(exit_img, (tile_size, int(tile_size * 1.5)))
        screen.blit(img, (col * tile_size, row * tile_size - (tile_size // 2)))
```

五、项目总结

在本学期的研究开发实践课上，跟着老师的学习，从最初的 C++中的 easyX 图形库的使用到最后使用 pygame 进行小游戏的便捷开发，我收获了很多关于游戏开发的知识。特别是碰撞逻辑、画面的移动与接洽、人物的控制与连续运动，多维地图的设计与实现。此前我在 Java 课程上了解过一些关于小游戏的制作，这个学期我又收获了许多关于如何使用 C++和 python 语言并利用集成的库和函数来进行游戏开发的知识，收益良多。

我学习了如何使用 Pygame 框架来创建游戏：了解了游戏主循环、事件处理、碰撞检测等 Pygame 的基本概念，并系统实践了项目结构和组织的架构。同时我还学到了如何加载图像、创建动画以及处理玩家输入，实现了角色的跳跃、行走和动画切换，为游戏增添了交互性；了解了如何在游戏中集成音效和背景音乐，为游戏增加了声音层次，提高了用户体验。通过创建多个关卡，学到了如何设计和加载不同的游戏关卡并如何更好地组织和管理游戏数据。

在这个小游戏项目不断调试和测试的过程中，我得到了很好的实践机会，帮助我应用了所学的编程知识，同时也是一个不断学习和完善的过程。希望以后我能够继续挑战自己，构建更复杂的游戏或项目，不断提高编程技能。