

# 项目报告

孟子越 苏道涵 漆之禹

2025 年 1 月 12 日

## 1 引言

本项目是 SI100B 课程的最终项目，项目文件在 [aMbZfcaYn/2024-SI00B-pygame](https://github.com/aMbZfcaYn/2024-SI00B-pygame)

这是一款平台跳跃游戏。玩家的初始场景罗列了游戏内存在的各种元素，并通过传送门进入每个关卡收集金币，获取装备。

游戏的主要内容是平台跳跃闯关，战斗方式有踩踏和子弹发射两种。

在关卡内玩家可以遇到售卖装备的 NPC 并解锁子弹发射的能力，NPC 的对话逻辑通过 openAI 实现，同时根据玩家收集金币数量的不同，店主的态度也会发生转变。

进入最后一道传送门后游戏通关（别忘了在通关界面给自己放放烟花哦~\*^\_^\*）

## 2 项目实施

### 2.1 Player 的移动逻辑（Core mechanic/Collision system/Main character）

右图中展示了 player 移动逻辑的相关变量，其中：

velocity\_x 和 velocity\_y 记录速度

moveSpeed 是玩家横向移动时的速度

jumpForce 是跳起后玩家获得的纵向速度

gravity 是在每个 tick 玩家损失的纵向速度，即重力加速度

而由于玩家只有在地面上才可以起跳，所以：

isGrounded 记录玩家是否着地，即能否起跳

通过在玩家脚底绘制一个小矩形（即文件中的 groundCheck）进行检测

若小矩形与障碍物重叠，则说明玩家着地

groundCheckDis 即为小矩形的高度（不宜太大，故设置为 1）

接下来需要进行玩家位置的更新，这实现在 update 与其他函数中

首先通过 update 进行输入和更新玩家速度，随后进行如下移动逻辑：

```
##move
moveX = moveY = True
xRect = self.rect.move(self.velocity_x, 0)
yRect = self.rect.move(0, self.velocity_y)

#普通墙体的互动
for obstacle in scene.walls:
    if obstacle.rect.colliderect(xRect): moveX = False
    if obstacle.rect.colliderect(yRect): moveY = False

#移动平台的互动
for machine in scene.machines:
    if machine.rect.colliderect(xRect): moveX = False
    if machine.rect.colliderect(yRect): moveY = False

if moveX: self.rect = self.rect.move(self.velocity_x, 0)
if moveY: self.rect = self.rect.move(0, self.velocity_y)

self.PositionFix(scene, moveY, yRect)
```

```
def PositionFix(self, scene, moveY, yRect):
    if not self.isGrounded and not moveY and self.velocity_y > 0:
        for obstacle in scene.obstacles:
            if obstacle.rect.colliderect(yRect):
                self.rect.bottom = obstacle.rect.top
                break

    if not self.headCollide and not moveY and self.velocity_y < 0:
        for obstacle in scene.obstacles:
            if obstacle.rect.colliderect(yRect):
                self.rect.top = obstacle.rect.bottom
                break
```

移动逻辑为计算 2 个新矩形：只在 x 方向移动和只在 y 方向移动后的矩形，哪个方向能动就往哪个方向

动，以此避免玩家与障碍物的重合（重合会导致玩家完全被卡住不能动）。

PositionFix 函数则用于解决另一问题，即在玩家下落到平台的过程中，如果玩家的 y 速度不能恰好让玩家的底部与平台上部重合，就会导致玩家在半空时，y 方向移动后矩形与地板重合，从而使得玩家悬空。PositionFix 通过判断上述情况，将玩家的底部直接更改到平台上部，从而解决上述问题。

玩家向上移动时也会有类似问题，通过相似的 headCollide 方法进行处理，因此 PositionFix 解决“悬空”和“头顶不到天花板”两个小问题。

综上实现了平台跳跃的基础移动逻辑。

## 2.2 动画系统

所有动画通过模拟一般动画的播放来实现，每种动画的每一帧按顺序用列表储存。每个动画有 3 个变量：animationTime、animIndex 和 animTimer，每经过 animationTime 次 tick 后，animIndex 都会+1 再对动画列表的长度取模，在绘制时只需要将这一对象的 image 设置为动画列表的第 animIndex 张图片，即可实现动画的效果。

### 2.2.1 Player 动画(Main character)

Player 有 4 个动画列表：静止、移动、跃起和下落。Player 类通过简易的状态机来实现动画。

```
class Player:
    #states
    self.states = ["Idle", "Move", "Jump", "Fall"]
    self.state = "Idle"
    #animation info
    self.animTimer = 0
    self.animIndex = 0
    self.currentAnim = None
    self.idleAnim = [pygame.transform.scale( pygame.image.load(rf"\assets_library\PlayerBasic\PlayerIdle.png"), (PlayerSettings.width, PlayerSettings.height) )]
    self.moveAnim = [pygame.transform.scale( pygame.image.load(rf"\assets_library\PlayerBasic\PlayerMove-{i}.png"), (PlayerSettings.width, PlayerSettings.height) ) for i in range(1,5)]
    self.jumpAnim = [pygame.transform.scale( pygame.image.load(rf"\assets_library\PlayerBasic\PlayerJump.png"), (PlayerSettings.width, PlayerSettings.height) )]
    self.fallAnim = [pygame.transform.scale( pygame.image.load(rf"\assets_library\PlayerBasic\PlayerFall.png"), (PlayerSettings.width, PlayerSettings.height) )]
```

Player 类中状态机的相关变量

Player 的状态取决于自身速度和是否着地

右图中的方法通过简单的逻辑判断角色当前的状态

当角色着地时 y 速度被设置为 0，所以可以不借助 groundCheck

currentAnim 是角色当前的动画列表

当状态切换时，currentAnim 变成新的动画列表

同时 Reset 方法将 animIndex 和 animTimer 重设为 0

综上即为 Player 的动画系统

```
def StateControl(self):
    if self.velocity_y != 0 :
        if self.velocity_y < 0:
            if self.state != "Jump":
                self.state = "Jump"
                self.currentAnim = self.jumpAnim
                self.Reset()
            else:
                if self.state != "Fall":
                    self.state = "Fall"
                    self.currentAnim = self.fallAnim
                    self.Reset()
        elif self.velocity_x != 0 :
            if self.state != "Move":
                self.Reset()
                self.state = "Move"
                self.currentAnim = self.moveAnim
            else:
                if self.state != "Idle":
                    self.Reset()
                    self.state = "Idle"
                    self.currentAnim = self.idleAnim
```

Player 状态的判定

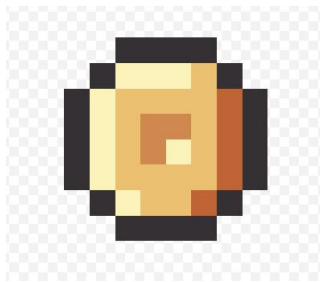
(Player 的实现较重要，故展示了部分代码)

### 2.2.2 动态场景动画(Scenes)

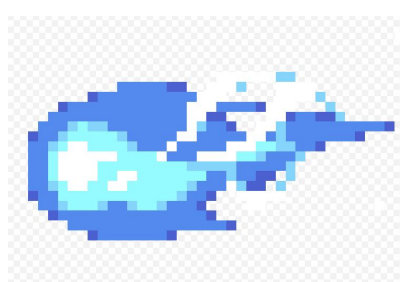
游戏内有 3 类动态场景：敌人(Enemy)、金币(Coin)和子弹(Bullet)



Enemy



Coin

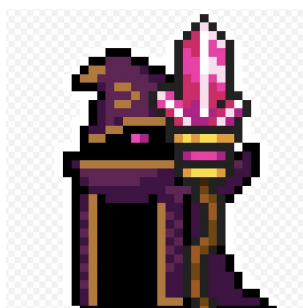
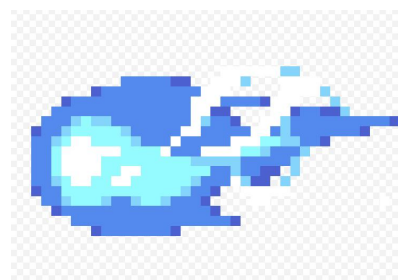


Bullet

他们有公共父类 Anim，在 Anim 的 update 方法里更新动画的相关变量，从而实现动画效果

## 2.3 交互场景(Interactive objects)

游戏内有如下 8 类可交互场景：墙体(Wall)、陷阱(Trap)、滑块(Machine)、金币(Coin)、敌人(Enemy)、子弹(Bullet)、NPC、传送门(Portal)。实现在 Player 的 Detect 方法和各个 class 的 update 方法中



### 2.3.1 墙体(Map.class Wall) (Collision system)

起简单阻碍玩家移动的作用

### 2.3.2 滑块(Map.class Machine) (Collision system)

当 Player 的 groundCheck 与滑块重合后，滑块启动

初始化时赋予滑块不同的方向（上下左右），滑块会实现相应不同的移动逻辑

滑块到达终点后会停顿 3 秒，随后以更快的速度返回起点  
当玩家站在滑块上时，会跟随滑块移动

### 2.3.3 陷阱(Map.class Trap)

玩家与陷阱重合，则玩家死亡，post 死亡事件

### 2.3.4 金币(Map.class Coin) (Resource system)

当玩家与金币重合时，player.coins 会增加金币的 value，同时在 UI 界面上实时更新

由于游戏内金币是一项较为重要的资源，我们不希望玩家进行“刷钱”的操作，所以我们借助玩家这一全局的对象，记录玩家获取过哪个场景的哪个金币，在玩家死亡或重新开始某一关时，不再渲染已经获得过的金币

### 2.3.5 敌人(Map.class Enemy) (Simple enemy)

敌人的交互有两种情况，当玩家与敌人重合时，首先判定玩家的 attackCheck 是否与敌人重合，如果重合说明玩家踩在敌人身上，攻击成功，删除这一 enemy；否则为敌人攻击玩家，玩家死亡，post 死亡事件

### 2.3.6 子弹(Map.class Bullet)

游戏中后期获得魔法能力，此时可以由 player 的 update 方法向当前场景加入子弹

子弹有 3 类简单逻辑：

达到最大飞行距离、与障碍物（包括墙体和滑块）重合、与敌人重合

前 2 类会进行自身的删除，而第 3 类将会删除敌人（为了让子弹玩起来更爽快，我们没有设置子弹碰到敌人后的自我删除）

### 2.3.7 NPC(Map.class NPC) (Friendly NPC)

NPC 魔法师是一个商人，帮助玩家解锁魔法能力

当玩家与 NPC 重合后，NPC 进入 awake 状态，此时 NPC 的 image 会多出对话框，提示玩家进行对话  
当 NPC awake 并且玩家按下 T 键时，post 进入商店的事件，并进入商店界面（下文进行介绍）

### 2.3.8 传送门(Map.class Portal)

借助场景切换逻辑，玩家与传送门重合后，post 进入下一场景的事件

## 2.4 大语言模型的嵌入 (Dialogue system/Decision system)

OpenAI 模型嵌在商店购物界面，这一界面的初始化会读取 player 信息，对于 player 不同的金币数，扮演不同的态度：

如果玩家的金币少于 80，AI 不会售出魔杖

如果玩家金币介于 80 到 100 之间，当老板听到关键词 ShanghaiTech 后，会同意玩家仅支付 80 金币买下魔杖

如果玩家金币多于 100，老板会热情招待，但不会降价

关于判定玩家是否成功购买，我们会在玩家退出商店后，内部询问 AI 是否卖出了魔杖（这一询问不会在 window 中显示），再根据结果对玩家进行更新（金币扣除以及是否获得魔法能力）

代码详见(Scene.class ShopScene)

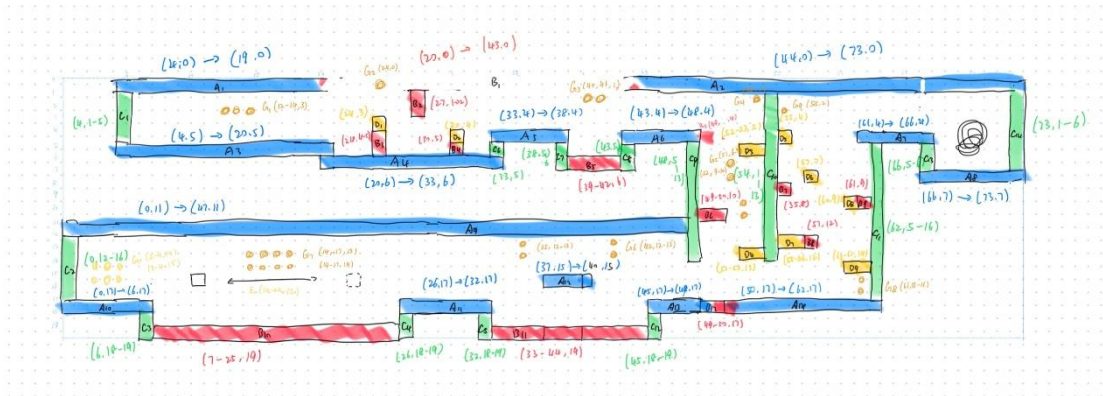


## 2.5 各类场景的搭建(Different scenes)

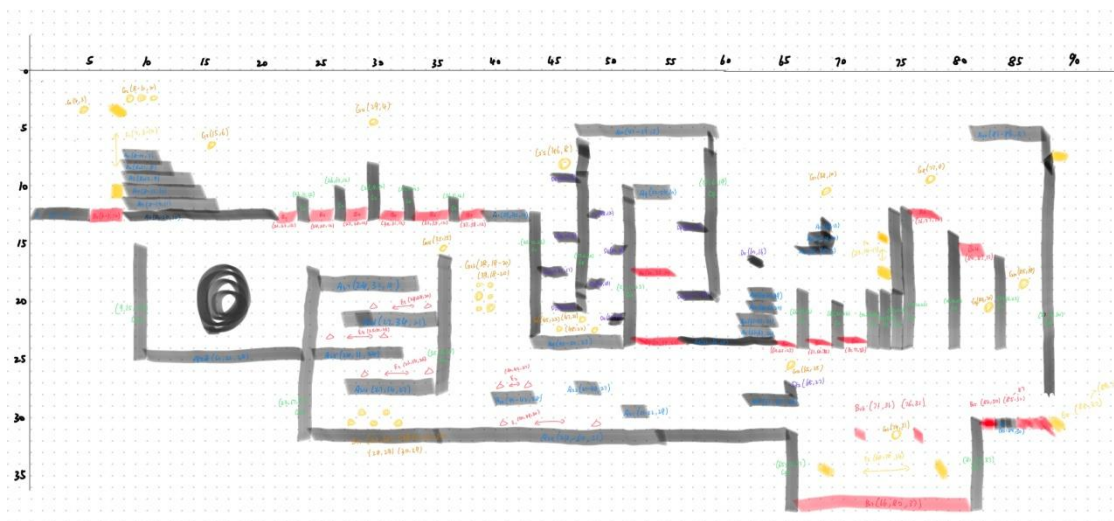
### 2.5.1 关卡设计

关卡绘制:

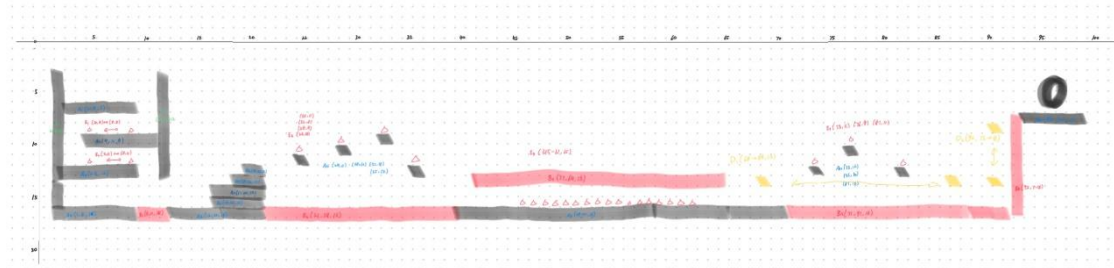
第一关:



第二关:



第三关:



关卡搭建:

将各个方块转化为坐标, 集合在 LevelMap.py 中, 再在 Map.py 搭建关卡, 最后在 Scene.py 整合为场景并进行绘制

### 2.5.2 非关卡界面绘制(Menu)

游戏有开始、商店、暂停、死亡、胜利、游戏内 UI 共 6 类界面，全部在 `Scene.py` 中实现

### 2.5.3 场景的切换

需要切换场景时（如玩家进入传送门/从死亡界面回到关卡），在相应的位置向全局 `post` 事件，再通过 `GameManager` 的事件队列和 `flush_scene` 方法转换场景。

## 2.6 BGM

只在商店界面进行了 BGM 的播放

# 3 创新

## 3.1 平台跳跃的机制

## 3.2 大语言模型实现商店

## 3.3 胜利界面的烟花庆祝

通过鼠标点击位置的追踪，在胜利界面实现了放烟花的小功能。