

**问题求解实战结题报告**

**计算机学院**

**2025年6月15日**

**项目名称：幻影迷宫**

**学号：2023141460164**

**姓名：刘畅**

**指导教师：张轶**

**目录**

[一、 游戏概述 5](#_Toc201171657)

[1. 游戏介绍及特色 5](#_Toc201171658)

[2. 开发环境 6](#_Toc201171659)

[3. 游戏总体工作量 6](#_Toc201171660)

[4. 总体流程 7](#_Toc201171661)

[二、 游戏内容 8](#_Toc201171662)

[1. 开始界面 8](#_Toc201171663)

[2. 说明界面 9](#_Toc201171664)

[3. 进入游戏 9](#_Toc201171665)

[4. 暂停界面 11](#_Toc201171666)

[5. 失败界面 11](#_Toc201171667)

[6. 胜利界面 12](#_Toc201171668)

[7. 主角 12](#_Toc201171669)

[8. 技能 13](#_Toc201171670)

[a. 烟圈 14](#_Toc201171671)

[b. 爆炸锐克 14](#_Toc201171672)

[c. 恢复 15](#_Toc201171673)

[9. 敌人/敌对单位 15](#_Toc201171674)

[a. 普通耄耋（普通猫） 15](#_Toc201171675)

[b. 爆炸耄耋（爆炸猫） 16](#_Toc201171676)

[c. 史莱姆 17](#_Toc201171677)

[d. 炮台 18](#_Toc201171678)

[e. 大耄耋（大猫） 18](#_Toc201171679)

[10. 道具 19](#_Toc201171680)

[a. 电池 19](#_Toc201171681)

[b. 尼古丁 19](#_Toc201171682)

[c. 随机礼物 19](#_Toc201171683)

[d. 毒药（负面道具） 19](#_Toc201171684)

[11. 门 20](#_Toc201171685)

[12. 普通关卡的生成 20](#_Toc201171686)

[13. Boss关卡的生成 21](#_Toc201171687)

[三、 代码结构与核心逻辑 23](#_Toc201171688)

[1. 文件结构 23](#_Toc201171689)

[2. 代码逻辑设计 23](#_Toc201171690)

[a. 场景的设计 23](#_Toc201171691)

[b. 敌人的设计 24](#_Toc201171692)

[c. 门的设计 26](#_Toc201171693)

[d. 效果的设计 27](#_Toc201171694)

[e. 道具的设计 28](#_Toc201171695)

[f. 其他底层代码 29](#_Toc201171696)

# 游戏概述

## 游戏介绍及特色

《幻影迷宫》（Phantom Maze）是一款2D俯视视角的动作冒险闯关游戏，融合了战斗、解谜与地图探索元素。玩家将在游戏中扮演一只灵巧的蝙蝠，穿梭于六大风格各异的迷宫关卡，解锁出口、收集资源、击败敌人，最终挑战强大的Boss。

每个关卡地图由墙壁、地面、道具与机关等元素构成，玩家需要灵活躲避敌人（如奶龙、鬼魂），利用声波攻击清除障碍，拾取血包恢复生命，通过收集金币强化攻击力或购买大招次数。有些关卡还设置了钥匙机制，玩家需先解锁大门才能进入Boss房。Boss战中，敌人拥有火球轰炸、召唤鬼魂等多种攻击方式，玩家需抓住攻击空隙释放声波或八方向大招精准反击，策略性十足。

本游戏拥有以下几大特色：

1. **多样化关卡设计**：六大普通迷宫加一场Boss战，每关结构、敌人刷新与道具分布均不同，提供持续的新鲜感与挑战；
2. **丰富的成长系统**：金币可用于购买强化（如攻击距离或技能使用次数），蝙蝠道具可提升声波范围，玩家可自由规划成长路线；
3. **真实战斗体验**：怪物AI具备巡逻与追踪机制，Boss具备多阶段攻击形态，爆炸、冷却、击退等机制均细致设计，提升沉浸与观赏感；
4. **完整UI与交互设计**：包括主菜单、暂停、胜利/失败界面、技能释放提示等，均配有独立UI界面与音效；
5. **简洁流畅的操作体验**：支持WASD或方向键移动、空格释放声波、J键释放大招、数字键购买强化，部分功能支持鼠标点击，适合不同玩家习惯；
6. **结局分明，挑战十足**：击败Boss即可通关，反之生命值归零将重新开始，鼓励反复尝试与策略规划；
7. **暗色美术与背景音乐风格**：整体画风偏神秘压抑，配合富有节奏感的音乐增强氛围，营造沉浸式探险体验。

本项目专为喜欢解谜、探索与战斗融合玩法的玩家设计，关卡难度阶梯式提升，兼顾新手上手体验与高阶挑战深度，是一款富有挑战性和趣味性的原创2D迷宫闯关游戏。

## 开发环境

系统：windows 11

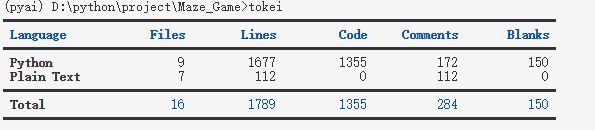
使用软件：

1. Visual Studio Code（开发工具）
2. Adobe Photoshop （素材制作）
3. Python 3.12
4. AI图像生成工具 （素材制作）

编程语言：Python

使用模块：pyzero、pygame等

## 游戏总体工作量



本项目代码部分共包含 16 个文件，其中 Python 源代码文件 9 个，地图数据（Plain Text 格式）文件 7 个。根据代码统计工具 tokei 的结果，项目整体共 **1789 行文本**，其中：

* 1. **核心功能代码**占比最大，约为 **1355 行有效 Python 代码**，涵盖了主界面逻辑、地图读取、角色与敌人行为、碰撞检测、技能实现、UI控制、音效管理、关卡机制等模块，体现了较为完整的 2D 闯关类游戏架构；
  2. **注释部分**达 284 行，表明项目在开发过程中注重代码可读性与后期维护，遵循良好的编程规范；
  3. **空行 150 行**，进一步体现出良好的代码结构与分块设计；
  4. **地图设计部分**通过 7 个文本文件记录了各关卡的字符矩阵结构，便于地图的可视化编辑与程序读取，虽无代码行，但包含完整逻辑元素（如墙体、陷阱、钥匙、出口等）；

游戏中的图像素材量达到了63个，音乐素材量达到了13个。

此外，我在开发过程中使用了git作为版本控制工具，并且托管在Github上。



## 总体流程

玩家打开游戏后，首先进入由美观的背景图片构成的开始界面，屏幕上显示"开始游戏"按钮。玩家可以点击按钮或按特定按键进入游戏介绍界面，该界面详细说明游戏操作和目标。

完成介绍后，游戏加载预设地图，玩家将控制一只蝙蝠角色在迷宫中探索。迷宫中分布着墙壁、道具，以及包括奶龙(MilkDragon)在内的敌人。玩家需要使用方向键或WASD进行移动，通过空格键释放声波攻击来击退敌人。

在地图探索过程中，玩家可能会遇到各种挑战，包括躲避鬼魂(Ghost)的追击。当玩家到达特定位置时，可能会触发Boss战，玩家需要利用声波攻击和机智的走位战胜Boss。Boss会使用多种攻击方式，包括发射火球等。

游戏过程中，玩家可以按下暂停键暂停游戏。如果角色生命值归零，游戏会显示失败界面，玩家可以选择重新开始游戏，重新挑战。

成功通过所有关卡后，玩家将迎来胜利界面，完成整个游戏流程。

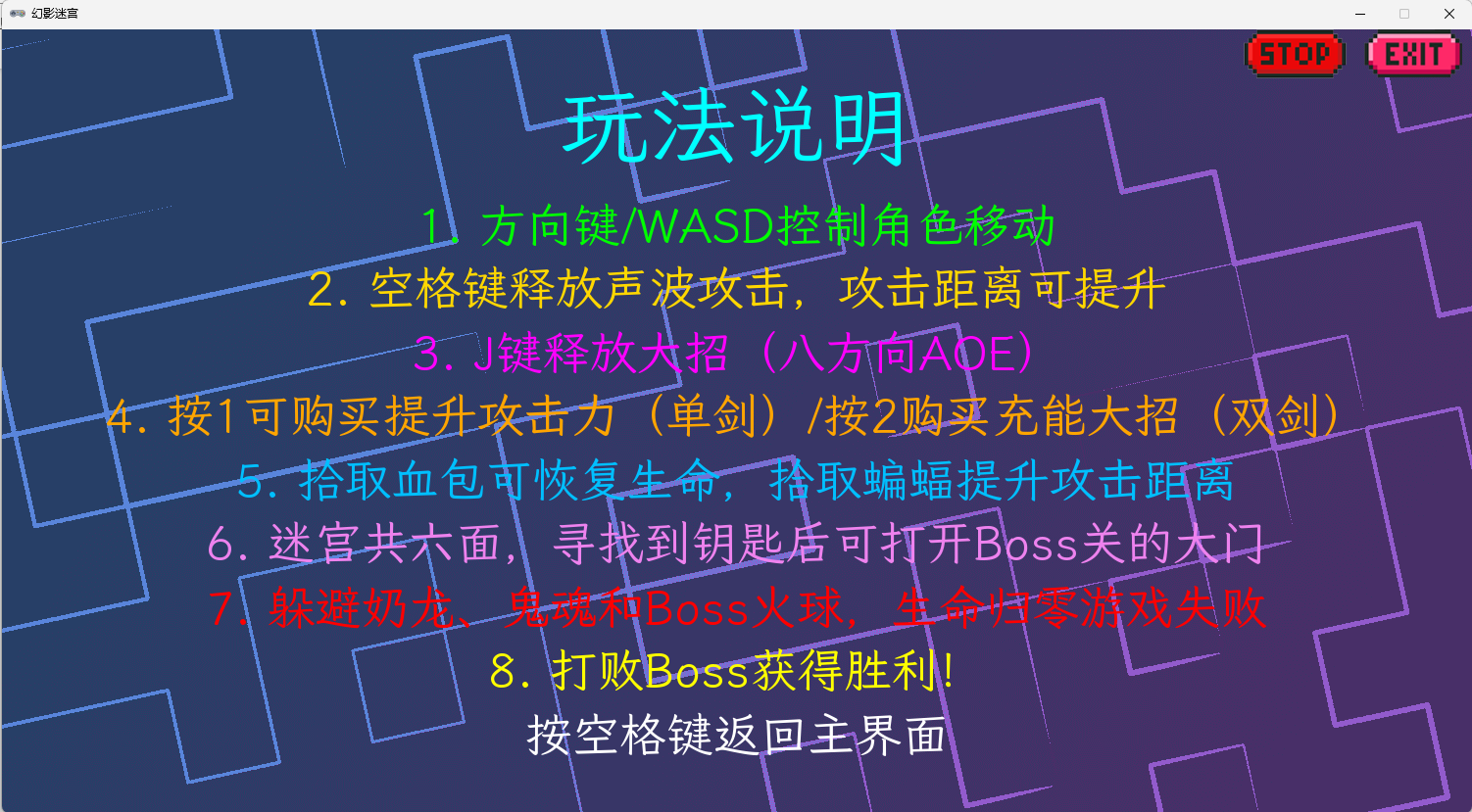
# 游戏内容

## 开始界面



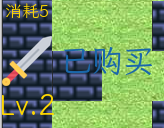
这是进入游戏后的开始界面。

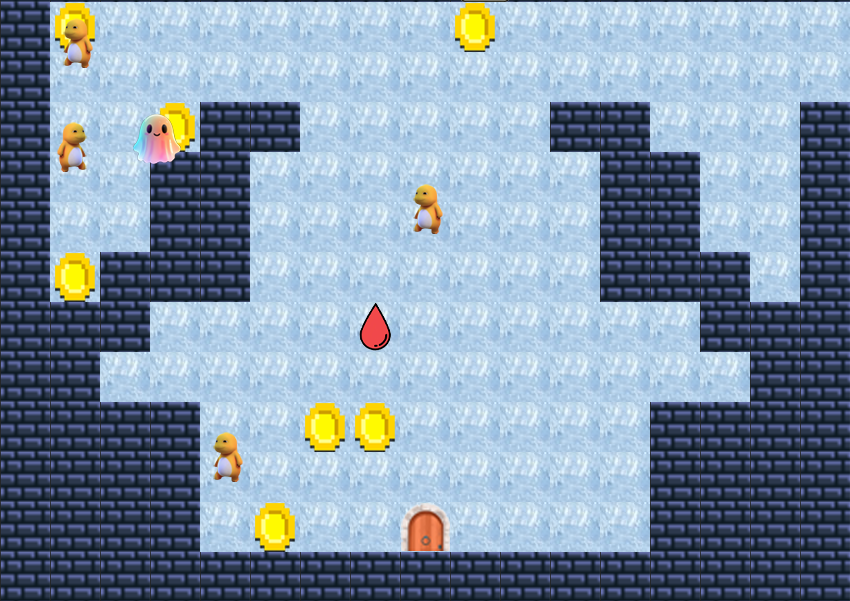
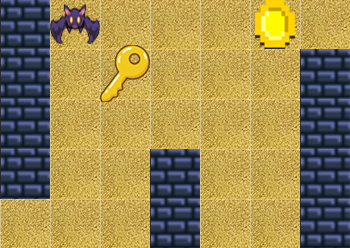
## 说明界面



## 游戏界面



* 在游戏中央区域，是一个由蓝色砖墙和绿色草地构成的迷宫地图，玩家将在其中探索。
* 迷宫中分布有多个金色硬币，玩家可以收集以获得积分或奖励。
* 敌人分布在地图的多个位置，形象为奶龙，会阻碍玩家前进。
* 地图中还存在一个彩色的特殊角色，是特殊敌人监管者。
* 地图中央位置有一个红色水滴形道具，为血包。
* 迷宫的右侧和底部设有门状结构，为当前关卡的传送点。
* 在游戏左上角，会显示当前生命值，即蝙蝠数量。
* 在游戏的顶部中央，有6个标签标识为“Side 1”至“Side 6”，代表多重地图/维度切换功能。
* 游戏右上角包含两个按钮：“STOP”（停止游戏）和“EXIT”（退出游戏）。
* 在游戏左侧状态栏显示“消耗: 5”、“Lv.1”（当前等级）、“消耗: 5”以及“技能”选项，下面显示数值“0”，代表技能数量。
* 游戏左下角会显示“金币 x 0”，表示当前玩家收集的金币数量。
* 购买技能/升级会出现：
* Boss关卡：
* 钥匙以及特殊的Boss关卡传送点：



## 暂停界面

当按下“STOP”时，游戏进入暂停状态，所有角色停止更新。按下空格恢复。



## 失败界面

当生命值为0时，游戏判定为失败。提示按下空格重新开始。



## 胜利界面

打败boss后胜利：



## 主角

主角由三部分构成：头部、身体、香烟。

其中头部有两种状态，分别是正常、哭泣。当受到攻击时，会显示一段时间的哭泣效果：



身体也有两种状态，分别是停止、奔跑。并且考虑了方向问题，当玩家向左运动时，循环播放向左奔跑的动态帧，向右同理。当玩家停止运动的时候切换到停止状态。



香烟也有两种状态，分别是显示与不显示。当使用部分技能时（比如烟圈攻击、恢复生命值），还会触发主角的抽烟效果，即将香烟切换为显示状态，同时这个也考虑了左右方向（主角向右则烟在右边，反之同理）。

最终组合如下：

## 技能

所有的技能如下：



每个技能都有自己独立的冷却时间，在图中以半透明灰色遮罩呈现。

### 烟圈

烟圈的图像如下：



这个技能用于攻击，消耗一格电量。烟圈具有扩散效果，即其会随着时间的流逝变大，朝着外部（主角正面方向）移动，当到达一定时间时消失。当烟圈与敌人碰撞时，会产生攻击与击退效果。

其攻击、击退效果的力度基本上与武器等级呈现log函数关系。其冷却时间最短。

### 爆炸锐克



当使用这个技能的时候，会沿着顺时针方向陆续发出8根香烟。每根香烟都有独立的碰撞箱，会运动一定时间后消失。当与敌人发生碰撞时，就会赋予敌人攻击与击退效果。

这个技能的香烟运动速度、击退效果、伤害与武器等级呈现正相关的关系。

### 恢复

当使用这个技能时，会立即恢复自身10滴血，并消耗1格电量。并且主角会产生一个抽烟的动画：



这个技能的冷却时间最长。

## 敌人/敌对单位

值得注意的是，所有的敌对单位都被限制在了一个关卡运动的范围内。它们无法移动到下一关。

并且为了增强可玩性，敌对单位的碰撞箱都相对其实际大小有所缩小。

### 普通耄耋（普通猫）

其原始素材与游戏展示如下：

在开发过程中，GIF图像是被转成了PNG格式再应用的。

实际上，这里还对其做了细节优化，在其未接触玩家时，只会播放前半截动画（较为温和），当碰撞上玩家时，才会播放后半截图像（动作幅度更大）。

其运动逻辑如下：找到本身与玩家之间的方向向量，然后将这个向量乘以运动速度，作用与自身。但是直接这样做的话会产生一个问题，就是当当前关卡具有很多这种敌人时，会让很多重叠在一起，很混乱。因此，我为这种敌人的运动方向添加了垂直分量，会沿着垂直于到玩家向量的方向移动。这样就会产生**螺旋着接近玩家的效果**（并且左旋或者右旋也是随机决定的）。即使关卡中有许多敌人，也不至于太过于重叠。

这个敌对单位的移速是最快的。

当其接触到玩家时，会触发攻击效果，一次攻击产生10点伤害，并且会产生一个攻击冷却。当敌对单位处于攻击冷却中时，将不会移动。待攻击冷却结束后，继续朝着玩家移动并且恢复攻击能力。

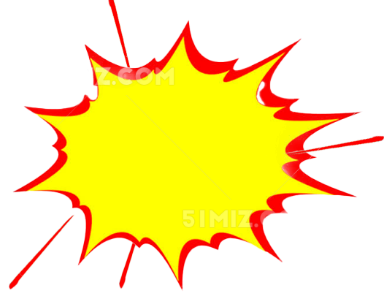
### 爆炸耄耋（爆炸猫）

其原始素材如下：



我使用pr软件为其增加了类似蜗牛运动的动画效果，即沿水平方向重复伸缩。由于这个素材具有方向性，因此，在其向右移动时，会对其素材进行一个水平翻转操作。

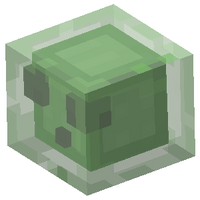
这个敌对单位的设计我参考了我的世界中的苦力怕。其运动逻辑很相似，其缓慢地朝着玩家移动，就像蜗牛一样。当接触到玩家时，会立即死亡并且触发一个爆炸效果：



这个爆炸效果会在敌对单位的中心产生，并且有扩大效果，会随着时间的流逝快速扩大，并且在很短时间内消失。当玩家与这个爆炸效果接触时，会产生大量伤害（50点生命值）。

### 史莱姆

其原始素材如下：



这个敌对单位同样来自于我的世界，我使用pr软件为其增加了类似史莱姆运动时一弹一弹的效果，即沿着水平、竖直两个方向同时产生简谐运动的拉伸效果，并且相位刚好相反：



同样，这个敌对单位也会缓慢朝着玩家移动，当其碰撞到玩家时，会对玩家造成少量伤害（5点），并且对其产生一个击退效果（模拟其将玩家弹开的效果），玩家会持续受到一个沿着史莱姆到玩家方向上的一个击退效果。

### 炮台

其原始素材如下（包含两幅图像）：

炮台的位置是固定的，无法移动，但其会旋转，并且始终呈现炮口对准玩家的姿态。当攻击冷却结束后，会朝着玩家射出一发子弹并且再次进入攻击冷却状态。这枚子弹会在炮台的中心位置产生，沿着射出时玩家所在的位置直线移动，在运动一定时间后消失。如果玩家没有及时离开，就会被子弹击中，产生普通攻击伤害（10点生命值）。

### 大耄耋（大猫）



大耄耋基本上就是普通耄耋的加强版：生命值增大，体积增大，移速增快。其出现在Boss关中。

## 道具

道具是随机出现在关卡中的，当玩家与其碰撞，就会产生特定的效果。

### 电池



电池用来给武器（锐克）充电，每次可以增加一格电量，如果充满了就不会继续增加了。

### 尼古丁



这个道具能增加武器（锐克）的等级，每次会将其升级一个等级。锐克的等级越高，其攻击力越强。当这个道具被使用时，会在屏幕左下角提示玩家“你的锐克升级了”

### 随机礼物



随机礼物会随机选择一个正面效果赋予玩家。随机选择的范围参见下文“门”章节。

### 毒药（负面道具）



当玩家与毒药发生碰撞时，会被赋予一个持续数秒中毒效果，生命值持续下降。

负面道具的意义何在呢？我当然知道玩家不会主动去拾起这个负面道具，但是当关卡中敌人众多时，只有这个区域没有敌人时，就会给玩家一个二选一的思考题：到底是硬刚敌人，还是中毒逃跑。这种设计让游戏更加有趣。

## 门

其共有6种造型，每次生成时会随机选取一个：

门用于隔离不同关卡，并且在玩法上经过了精心设计。每个门都带有一个正面buff，在每次进入下一个关卡时可以进行二选一。其效果包含：

1. 恢复生命值
2. 增加玩家移动速度
3. 恢复锐克电量
4. 增加最大生命值
5. 缓慢恢复生命值
6. 升级锐克等级
7. 增加锐克最大电量

当玩家进入了一扇门后，与之对应的另一扇门也会被同步禁用，实现二选一的效果。并且当门被禁用后，其字体颜色会变成灰色。

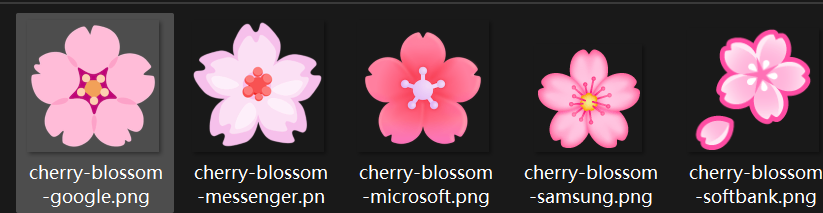
## 普通关卡的生成

当当前关卡数不是10的倍数时，就会生成普通关卡。其背景图片会从背景图片库中的8个图片中随机选择一张。

对于某些小怪，其生成的数量与当前关卡的序数呈现正相关（log曲线）的关系，当前关卡序数却高，生成的小怪越多。

对于场景中的道具，其会采取抽签的形式（1/2的概率）决定是否生成，如果生成，又会随机地决定其在于关卡中的位置。

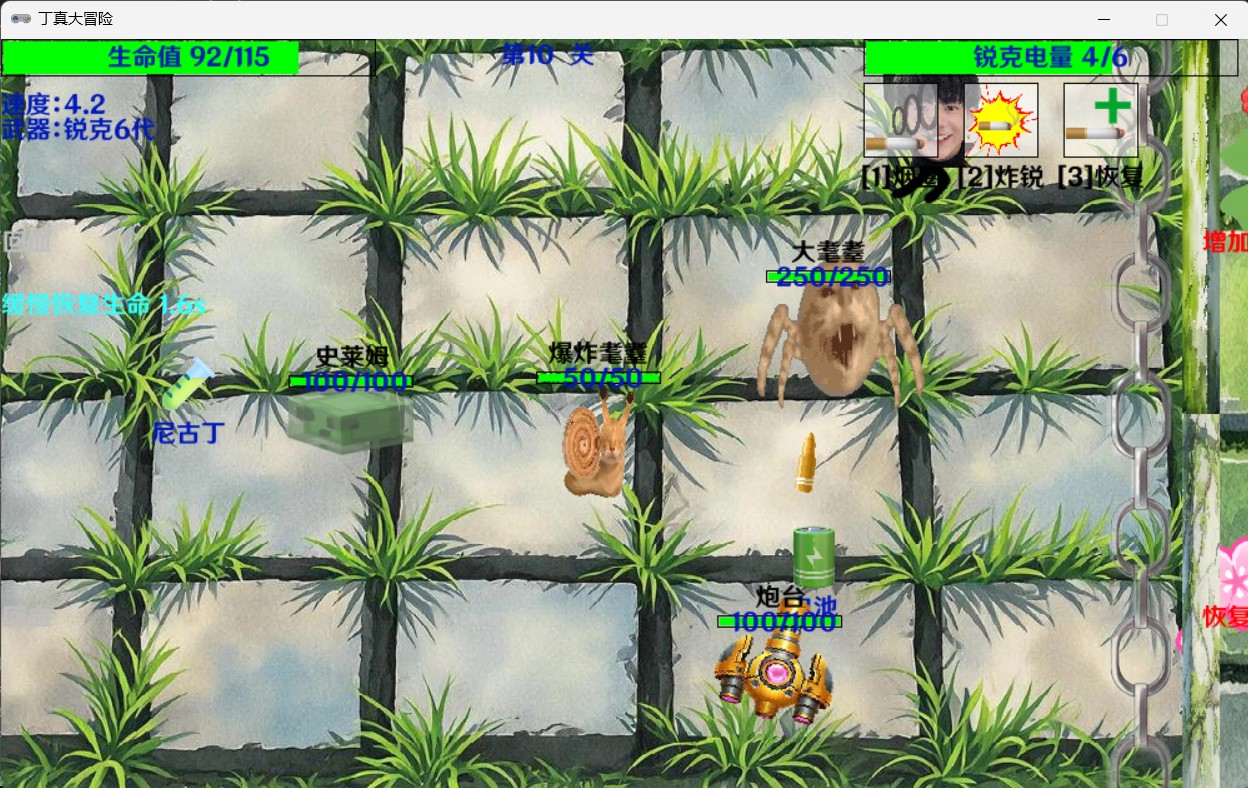
然后还有场景中的背景元素（共有26种），会随机地生成10个在关卡的上半部分，10个在关卡的下半部分，营造更加生动的效果：



## Boss关卡的生成

每当关卡数来到10的倍数时，就会生成一个Boss关卡，其特点如下：

该关卡有一个主要敌人（比如大耄耋，其移速更快、生命值更高、体积更大），并且有一些小怪。如果主要敌人没有被打败，那么这个关卡就无法通过（右侧有一个铁链会组织玩家的通行）。



当玩家击败了主要敌人后，右侧的锁链会消失，并且产生一个爆炸效果（这个爆炸效果也会有一个扩大然后消失的效果，但不会对玩家造成伤害，单纯是视觉上的效果），然后玩家可以进入下一关：



在设计过程中，我还遇到了一个问题：如果玩家的锐克电量不足以击败主要敌人，就会造成卡关。于是，我还设计了一个保底机制：在玩家电量耗尽且关卡中没有“电池道具”时，会随机刷新一个“电池”道具，这样就不会造成玩家的卡关问题。

# 代码结构与核心逻辑

## 文件结构

为了进一步提高开发效率，我将不同类型的代码放在了不同的文件中，其主要分布如下：

1. main.py 用于pyzero的游戏接口，包括绘图、事件更新、按键检测等接口。
2. elements.py 这是游戏的主要工作，包括了主角及其数据结构、敌人及其数据结构、门、场景等主要元素
3. mapping.py 用于数学映射工作：包括各种平衡性，如敌人生成数量与关卡序数的关系、武器攻击力度与武器等级的关系等等。
4. utils.py 工具类，主要是pyzero的Actor对象
5. assets.py 存储共享数据结构，主要用于在不同文件之间共享屏幕尺寸、帧间隔时间等信息。

## 代码逻辑设计

在程序实现过程中大量应用设计模式（继承与组合）了以提升开发效率。在该程序中，我一共自定义了48个类，其中有40个类都是继承子类。

其中最深的继承达到了3层，例如：

Actor -> EnhancedActor(增加了可见性、可移动性属性) -> Attack（实现基本的攻击方法） -> SmokeAttack（对应烟雾攻击技能）

### 场景的设计

场景是一系列对象的组合，包括

1. 门
2. 普通元素
3. 场景动态元素（敌对单位）
4. 道具
5. 主角发出的攻击
6. 敌人发出的攻击
7. 高优先级元素（渲染在最顶层）

这么细是为了改善游戏的性能，因为大量的碰撞检测是非常消耗算力资源的。在每次更新时(update)，就只用判断如下元素之间的碰撞关系：

1. 判定主角与门的碰撞关系
2. 主角与敌对单位的碰撞关系
3. 主角与敌人发出的攻击的碰撞关系
4. 主角发出的攻击与敌人的碰撞关系
5. 主角与道具的碰撞关系

而不是所有元素的相互碰撞关系，大幅减小了性能开销。

### 敌人的设计

我将敌人的行为与敌人的素材解耦了，这是为了实现更灵活的代码复用，比如将耄耋的逻辑代码同时应用于普通耄耋与大耄耋。

敌人的逻辑行为的基类是EnemyData，其核心架构如下：

class EnemyData:

def \_init\_\_(self,……):

        完成资源初始化：生命值等各种属性

    def onDeath(self): #死亡

        pass # 被重载的方法，在死亡时执行的动作

    def attacked(self,damage): #被攻击

        被攻击的代码，判定是否死亡

    def tick(self): # 当场景update时触发

        实现朝着玩家移动相关的代码

def logic\_tick(self):

逻辑上能攻击的tick

    def draw(self):

        绘制相关元素

按照这种设计模式，史莱姆的设计就能大大简化：

class SlimeEnemy(EnemyData):

    def \_\_init\_\_(self,……):

        super().\_\_init\_\_(……)

        self.tips="史莱姆"

        self.moving\_speed=1.5

        scale\_without\_img(self.actor,0.5)

    @override

    def logic\_tick(self):

        if self.actor.colliderect(self.mainActor.actor):

            self.mainActor.be\_attacked(5) # 造成5点伤害

            RepelEffect(self.mainActor,(d\_x,d\_y),) #击退效果

            self.set\_cd() #冷却时间

### 门的设计

其基类的核心代码如下：

class Door:

    def \_\_init\_\_(self,bind:Actor,other=None):

        初始化相关属性

    def on\_enter(self,mainActor:MainActor):

        若玩家进入这个门，就禁用另一扇门（实现二选一）。

并且对玩家施加特定效果

    def onUse(self,mainActor:MainActor):

        pass #应当被重载的部分，实现特定效果

    def draw(self):

        绘制元素

有了这个基类，具体的门就很容易被设计了，例如这个是缓慢回血门的设计：

class RecoverRekePowerDoor(Door):

    def \_\_init\_\_(self,bind:Actor):

        super().\_\_init\_\_(bind)

        self.tips="缓慢回血"

    @override

    def onUse(self,mainActor:MainActor):

        SlowRecoverEffect(mainActor,)

可以看到，仅仅需要重载onUse函数就可以实现特定的效果。此处是将SlowRecoverEffect效果应用在了主角身上。

### 效果的设计

效果是针对时间上连续的任务而言的，比如缓慢回血、击退，其基类核心代码如下：

class Effect():

    def \_\_init\_\_(self,target,time):

        self.target=target # 目标

        self.time=time #剩余时间

        self.duration=time # 总时间

        effects.append(self) #添加到全局效果列表

        self.isShowUI=False #是否显示在屏幕上

        self.tips="default" # 名称

        self.attach\_on\_finish\_functions=[] # 效果完成的回调函数

        self.type=EffectType.NEUTRAL #效果类型 （对应显示不同颜色）

    def tick(self):

        更新剩余时间与是否完成判定

    def invoke(self):

        pass # 应该被重载的核心代码

    def on\_finish(self):

        pass #结束时被调用

在这种框架下，具体效果的实现就非常简单，例如缓慢回血效果：

class SlowRecoverEffect(Effect):

    def \_\_init\_\_(self,target,time=5\*1000,strength=5):

        super().\_\_init\_\_(target,time)

        self.strength=strength

        self.isShowUI=True

        self.tips="缓慢恢复生命"

        self.type=EffectType.POSITIVE

    def invoke(self):

        delta=self.strength\*assets.elapsed\_time\_frame/1000

        self.target.health=min(self.target.health+delta,self.target.max\_health)

只用计算每个调用的时刻应该恢复多少血量，然后将其赋给即可主角即可。

### 道具的设计

道具的基类主要代码如下：

class Tool(Actor):

    def \_\_init\_\_(self,image):

        ……

    @override

    def draw(self):

        绘制素材

    def onUse(self,mainActor:MainActor):

        pass #应当被重载的方法，实现特定的效果

道具“电池”就能很容易被实现：

class Battery(Tool):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("battery")

        self.tips="电池"

    @override

    def onUse(self,mainActor:MainActor):

        mainActor.reke\_power=

min(mainActor.reke\_max\_power,mainActor.reke\_power+1)

### 其他底层代码

由于pyzero能提供的接口很有限，许多效果实现不了。由于pyzero是建立在pygame的基础上的，因此可以扒出其pygame对象，并以此实现特定效果，这里举几个典型的：

1. 烟圈、爆炸的扩大效果就需要调用到pygame的代码（缩放）

actor.\_surf=pygame.transform.scale(actor.\_surf, (new\_width, new\_height))

actor.\_orig\_surf=pygame.transform.scale(actor.\_orig\_surf, (new\_width, new\_height))

1. 动画效果：

由于敌对单位都是由一系列动画帧组成的，因此我设计了一个可以播放循环图片序列的Actor子类：

class GifActor(EmptyActor):

    def \_\_init\_\_(self, gif\_path,size:Tuple[int,int]=None):

        super().\_\_init\_\_()

        self.frames = []

        self.current\_frame = 0

        self.animation\_speed = 0.1

        self.time\_since\_last\_frame = 0

        self.visible=True

        gif\_actors.append(self)

        self.flip\_state=False

        self.canMove=True

        self.frames=get\_images\_from\_folder(gif\_path)

        if not self.frames:

            raise ValueError(f"Failed to load GIF frames from {gif\_path}")

        self.\_surf = self.frames[0]

        self.width = self.\_surf.get\_width()

        self.height = self.\_surf.get\_height()

        self.anchor = (self.width/2, self.height/2)

        self.range=(0,len(self.frames))

        if size is not None:

            scale(self,size[0],size[1])

            tmp=[]

            for frame in self.frames:

                tmp.append(pygame.transform.scale(frame, size))

            self.frames=tmp

        self.frame\_flip\_state=[False]\*len(self.frames)

    def update(self, dt):

        if self.visible:

            self.time\_since\_last\_frame += dt

            if self.time\_since\_last\_frame >= self.animation\_speed:

                self.time\_since\_last\_frame = 0

                self.current\_frame +=1

                if self.current\_frame>=self.range[1]:

                    self.current\_frame=self.range[0]

                self.\_surf = self.frames[self.current\_frame]

        else:

            self.\_surf = placeholder\_image

        if xor(self.flip\_state,self.frame\_flip\_state[self.current\_frame]):

            self.frames[self.current\_frame]=flip\_pygame\_surface(self.frames[self.current\_frame])

            self.\_surf = self.frames[self.current\_frame]

            self.frame\_flip\_state[self.current\_frame]=not self.frame\_flip\_state[self.current\_frame]

    def draw(self):

        super().draw()