项 目 报 告

成员：王子睿 胡俞中 段林沛

2025 年 01 月 12 日

目 录

1 简介

在项目根目录下输入 make 或者 python src/main.py 可以运行游戏。

本项目为 SI100B 课程的最终项目，项目文件 已上传到了 GitHub。我们制作了一款类银河战士恶魔城游戏，玩家陷入梦境之中，手握一把丝线之剑，可在平台间跳跃与怪物搏斗。

战斗采用即时制，玩家通过键盘控制来进行移动和攻击，躲避怪物攻击的同时对其造成伤害。玩家击败怪物之后可以获得灵魂货币，到商人 NPC 处可以用灵魂货币购买升级，击败怪物一定次数之后可以获得胜利。

程序的入口是 main.py ，其中的 Game 类负责游戏的整体逻辑。游戏的各个部分被分割到不同的模块。

- settings.py ： 存储各种可调参数
- level.py ： 地图的控制
- tile.py ： 地图中的障碍物
- player.py ： 玩家的控制
- weapon.py ： 玩家的三种武器
- npc.py ： NPC 的控制
- enemy.py ： 敌人的控制
- menu.py ： 游戏的开始与结束画面
- app_data.py ： 定义了表示当前窗口的枚举类型 AppForm 及程序运行状态变量
- scrollbar.py ： 定义用于实现音量控制的滚动条功能
- speech_recog.py ： 完成用户交互的语音识别功能，可实现语音操控菜单
- keys.py ： 处理玩家输入
- utils.py ： 一些工具函数
- debug.py ： 一些调试用的函数

2 项目内容

2.1 场景

2.1.1 大型场景

游戏的场景是一个大型地图，地图中有两位 NPC ，若干会复活的敌人与一个 BOSS。

图 2.1 地图

整个地图由 `level_dream.py` 中的 `LevelDream` 类控制，它继承自 `level.py` 中定义的 `Level` 类。地图中的障碍物、敌人、角色与 `NPC` 都是在这里被创建实例，并在游戏循环中被更新。

游戏中的图像可能会重叠，为了让它们被正确显示我们定义了 `VisibleGroups` 类。先对可见图像按照 `y` 坐标排序，然后按照“物品”、“敌人与 NPC”、“主角”的先后顺序绘制。

2.1.2 交互对象

地图中的障碍物在 `tile.py` 中定义。每个障碍物都有自己的碰撞箱，玩家会与之发生碰撞。

图 2.2 地图中的障碍物 1

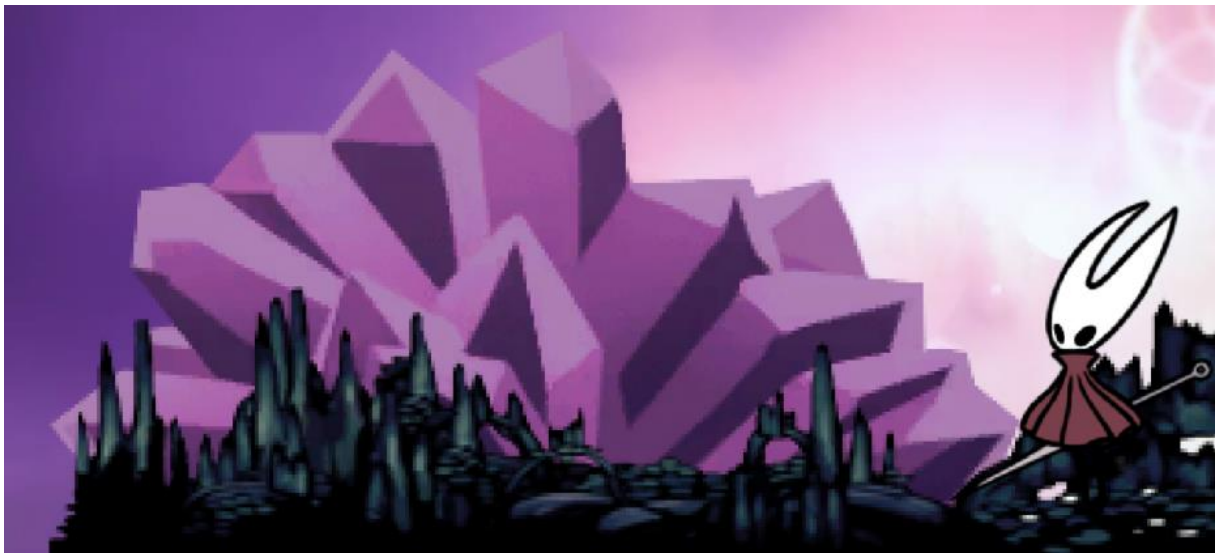


图 2.3 地图中的障碍物 2

2.2 角色

2.2.1 主角

属性 玩家拥有生命值与魔力值，被敌人打到会损失生命，释放魔法会消耗魔力。

移动 玩家可以按下 <A> 和 <D> 来左右移动，按下 <Space> 跳跃，按下 <Shift> 冲刺，按住<Shift> 疾跑。为了让玩家的移动更加自然，我们在玩家的跳跃过程中引入了重力加速度，并在检测到玩家落地的时候应用摩擦力。

攻击 玩家可以按下 <J> 造成普通攻击，按下 <H> 丢出武器，进行远程攻击，按下 <U> 挥舞丝线，释放魔法。

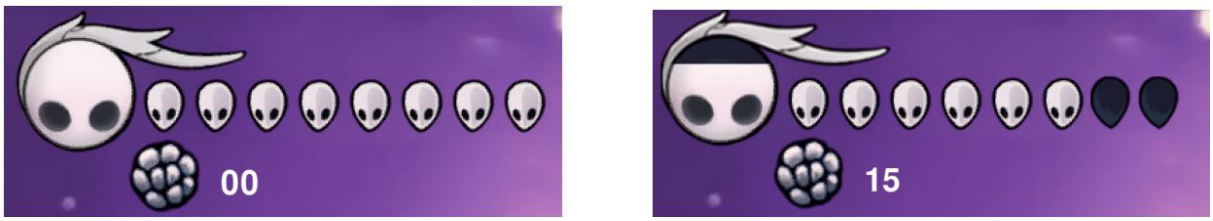


图 2.4 玩家属性显示

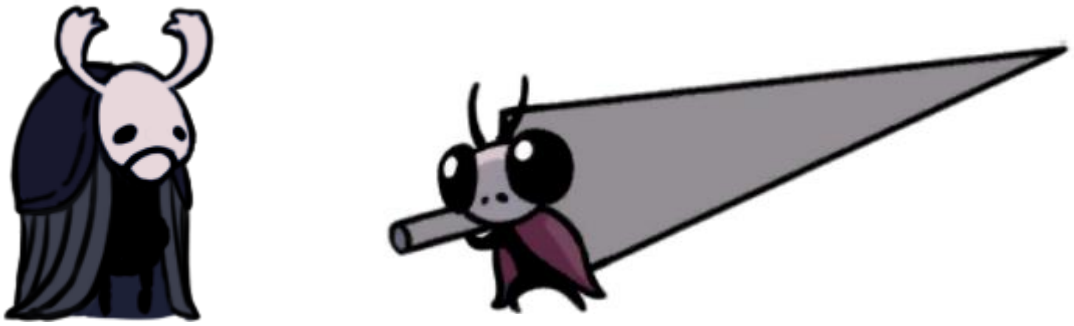
2.2.2 NPC

游戏中有两位 NPC，一个是向导，另一个是铁匠，他们都继承自 npc.py 中定义的 NPC 类。向导会在游戏开始时向玩家介绍游戏的基本操作，铁匠可以升级玩家的武器。

当玩家接近 NPC 的时候按下 <R> 可以开始对话，如果需要回复可以按下 </> 后输入内容，按下 <Enter> 发送。玩家打开与铁匠的对话后按下 <B> 可以升级武器（消耗 10 单位货币）。

为了不漏掉玩家在输入对话的时候按下的每一个按键，我们在 keys.py 中通过处理游戏中获取到的每一个 pygame.event.Event 事件来管理输入。

具体的对话内容我们调用了 python-openai 库的 API，其中可能会有一些不可打印字符，我们在 utils.py 使用正则来“标准化”了一遍返回的字符串。



(a) 向导　　(b) 铁匠

图 2.5 NPC

2.2.3 敌人

地图中存在可以复活三次的敌人。敌人在每一帧都有一定概率转向，在距离初始位置太远时会强制转向。敌人每次被击败都会展示死亡动画，一段时间后复活。

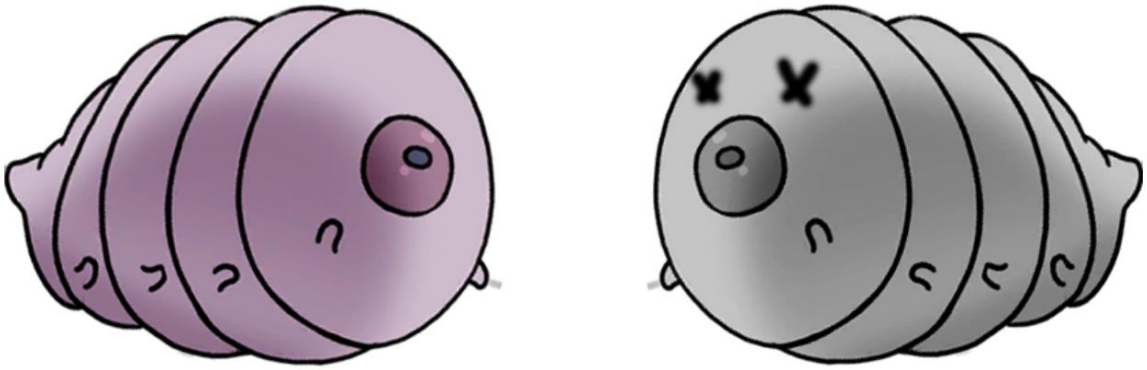
初始时候玩家需要三次普通攻击才能杀死一个敌人，但是在升级两次之后就只需要一次攻击。每杀死一次敌人都会获得 5 单位货币。

(a)移动　　(b)死亡

图 2.6 普通敌人

2.2.4 BOSS

在地图的右上方，经过一段台阶之后玩家可以遇到最终 BOSS。它具有更高的攻击力与血量，初始状态下玩家需要三十次普通攻击才能击败它。



(a) 移动　　(b) 死亡

图 2.7 BOSS

2.3 游戏机制

核心机制 玩家可以自由探索地图，与 NPC 对话，与敌人战斗。游戏的胜利条件是击败最终 BOSS，这对于初始状态下的玩家可能有点难度，但是玩家可以升级武器来增

强实力。

碰撞系统 游戏每一帧中玩家都会调用 move() 方法，其中会先进行 x 方向的移动，完成碰撞检测，再处理 y 方向的移动，最后第二次碰撞检测。每次碰撞检测玩家会与可碰撞精灵组的每一个元素进行碰撞检测。由于碰撞箱都是矩形，两两碰撞检测可以 O(1) 完成，在目前的数据规模下并不会造成卡顿。

资源系统 玩家通过击败敌人获得灵魂货币，在铁匠处升级武器可以消耗货币。

图 2.8 货币

2.4 游戏性

2.4.1 菜单

菜单 UI 界面（花纹、游戏名、菜单项）通过 photoshop 制作完成，以此呈现协调一致的显示效果，如图 2.6 所示。

图 2.9 菜单窗口

主菜单项由 START,SETTING,QUIT 三项组成：

- 单击 START 菜单项可进入游戏窗口
- 单击 SETTING 菜单项可显示音量调节滚动条，可对背景音乐进行音量调节

图 2. 10 单击 setting 显示滚动条

- 单击 QUIT 即可退出游戏。

将鼠标置于三个按钮上会使菜单项变粗并发出声音，涉及了碰撞的判断，在代码部分会详细说明。此外，经过研究我们使用了 LLM——SPEECH RECOGNITION 功能，实现可以通过语音输入来控制菜单，当用户按下空格后，通过语音下达“开始”、“音量大/小一点”、“静音”、“退出”等指令来操控游戏。将会在“3.1 菜单的语音操控”部分详细说明。

下面罗列程序中各个文件的相应作用：

(1) menu.py 提供了菜单窗口的所有功能，其中定义了 MenuItem 类和 MenuForm 类。

- MenuItem 继承了 pygame 的精灵类（pygame.sprite.Sprite），将每个菜单项视为游戏中的一个精灵角色，以便于判断鼠标是否与此发生碰撞。在鼠标移动事件 MouseMotion 时检测是否发生“碰撞”（即用户鼠标是否碰到按钮）。碰到时触发简短的提示单，并在 Click 时触发执行的相应功能。
- MenuForm 模仿了.Net 平台的 WinForm 窗口，定义了窗口大小，背景图，精灵组（sprite.Group），并提供了 load 和 refresh 成员函数，实现窗口加载的初始化设置，以及窗口的刷新。同时对鼠标 Down，Up，Click，Motion 事件提供了响应函数。其中 Click 函数中通过鼠标与菜单项（精灵）的碰撞检测来判断单击了哪个菜单。

(2) scrollbar.py 实现了滚动条调节音量的功能，其中定义了 Bar 类和 Cursor 类和 ScrollBar 类。其实 Bar 和 Cursor 均继承了 pygame 的精灵类，分别提供了标尺和游标的显示。ScrollBar 类包含了一个 Bar 和一个 Cursor 对象，同时响应了鼠标的 Down，Up，Click，Motion 事件，实现了对背景音量的控制。

(3) speech_recog.py 通过使用 speech_recognition 调用 google 的语音识别 LLM，实现了语音操控菜单的功能，并通过 pyttsx3 包实现了语音提示，使得游戏操作界面更加智能。

2.4.2 BGM

游戏在开始菜单，游戏结束有不同的背景音乐。我们使用了 pygame.mixer.music 来播放音乐，可以通过菜单中的按钮来调节音量。

3 创意

3.1 菜单的语音操控

为更好地增加用户体验，我们使用了 Python 的 speech_recognition 库调用 google 的 LLM 实现操控指令的语音识别功能。目前主要的中文语音指令包括:“开始”、“声音大一点”、“声音小一点”、“静音”、“退出”。以及相应的英文指令:“start”,“up”,“down”,“mute”,“exit”。目前对中文识别率比较高，究其原因是我们在调用 recognize_google 函数时，指定了 language='zh-CN'。我们也尝试做两次识别（第一次中文，第二次英文，即 language='en-US'），但发现这样程序响应略显得慢（部分原因可能与访问 google 网络有关，但 LLM 识别次数越多，占用时间也会越长）。图 3.1 所示是语音识别操控程序的效果:

图 3. 1 菜单指令的语音识别

鉴于调用 google LLM 进行语音识别的实时性无法满足操控游戏的速度要求，所以我们在游戏过程中并没有使用语音控制，而用在了实时性要求相对较低的菜单界面控制。我们也会对提高大语言模型的计算速度进行进一步的研究。

3.2 从零开始的游戏

本项目没有使用助教提供的模板，整个代码的框架都是自己设计实现的。虽然这可能给我们带来了一些不必要的麻烦，但是也让我们更加了解了整个游戏的实现细节。而且能够实现更不一样的游戏机制。

3.3 调试模式

本项目在开始的时候就为了调试设置了 debug.py 文件，其中的 display() 函数在有些时候能够比 print() 更优雅的显示变量的值，而的 FreeCamera 类可以代替玩家测试地图。另外本项目在运行的时候可以传入 --DEBUG 参数，开启调试模式并显示碰撞箱。

3.4 杂项

- 为了让玩家的移动更加自然，我们在玩家的跳跃过程中引入了重力加速度，并在检测到玩家落地的时候应用摩擦力。
- 为了让游戏在不同分辨率下自然显示，游戏中几乎每一个类都有 scale 参数。