

**《软件工程》项目报告**

**题目： 鲁米娜：逐光之旅**

**课程名称： 软件工程**

**专业班级： BD2202**

**组 名： 啊不是我吗？**

**同组成员： 学号： U202215601**

**姓名： 江 南**

**指导教师： 冯 琪**

**报告日期： 2024.12.16**

**计算机科学与技术学院**

# 任 务 书

**一 总体要求**

1. 综合运用软件工程的思想，协同完成一个软件项目的开发，掌软件工程相关的技术和方法；

2. 组成小组进行选题，通过调研完成项目的需求分析，并详细说明小组成员的分工、项目的时间管理等方面。

3. 根据需求分析进行总体设计、详细设计、编码与测试等。

**二 基本内容**

根据给出的题目任选一题，自行组队，设计与开发中软件过程必须包括：

**1. 问题概述、需求分析：**正确使用相关工具和方法说明所开发软件的问题定义和需求分析，比如NABCD模型，Microsoft Visio，StarUML等工具 (20%)；

**2. 原型系统设计、概要设计、详细设计**：主要说明所开发软件的架构、数据结构及主要算法设计，比如墨刀等工具（35%）；

**3. 编码与测试**：编码规范，运用码云等平台进行版本管理，设计测试计划和测试用例（30%）；

**4．功能创新**：与众不同、特别吸引用户的创新（10%）；

**5. 用户反馈**：包括用户的使用记录，照片，视频等（5%）。

**目 录**

**[任 务 书](#_Toc168272642)** [I](#_Toc168272642)

**[1 问题定义](#_Toc971537393)** [1](#_Toc971537393)

[1.1项目背景与意义 1](#_Toc1310796010)

[1.1.1 项目背景 1](#_Toc1661289144)

[1.1.2 项目意义 1](#_Toc1851748561)

[1.2 项目基本目标 1](#_Toc1005052403)

[1.3 可行性分析 2](#_Toc1956853566)

[1.4人员管理和项目进度管理 3](#_Toc125829957)

[1.4.1 人员管理 3](#_Toc1700178651)

[1.4.2 项目进度管理 3](#_Toc485180375)

**[2 需求分析](#_Toc431154966)** [4](#_Toc431154966)

[2.1 需求分析概述 4](#_Toc811688584)

[2.2 UML相关需求分析图 6](#_Toc1233905544)

[2.3 原型系统设计 7](#_Toc898929)

**[3 概要设计和详细设计](#_Toc75914174)** [10](#_Toc75914174)

[3.1 系统结构与类图说明 10](#_Toc284236100)

[3.2 关键数据说明 13](#_Toc1152501772)

[3.3 数据管理说明 27](#_Toc1942269711)

[3.3.1 数据管理 27](#_Toc1975598377)

[3.3.2 垃圾回收 28](#_Toc1637255972)

**[4 实现与测试](#_Toc1653152393)** [29](#_Toc1653152393)

[4.1实现环境与代码管理 29](#_Toc388844265)

[4.2 关键函数说明 29](#_Toc512824034)

[4.3 软件测试与测试计划 34](#_Toc1181664027)

[4.3.1 游戏软件测试方法 34](#_Toc298534333)

[4.3.2 测试计划 35](#_Toc944735339)

[4.4 结果分析 35](#_Toc1820240302)

[4.4.3 开发者测试（冒烟测试、UI测试、功能性测试）分析 35](#_Toc1874204199)

[4.4.2 玩家测试（性能测试、玩法测试）分析 38](#_Toc459838397)

**[5 总结](#_Toc1857776473)** [39](#_Toc1857776473)

[5.1 用户反馈 39](#_Toc1384437978)

[5.1.1 用户反馈 39](#_Toc263781001)

[5.2.2 后续改进方向 39](#_Toc961036399)

[5.2 全文总结 40](#_Toc914248906)

**[6 体会](#_Toc535868857)** [41](#_Toc535868857)

[6.1 技术层面 41](#_Toc1948947728)

[6.2 软件开发层面 41](#_Toc396396805)

# **1** 问题定义

## 1.1项目背景与意义

### 1.1.1 项目背景

由于时间上的重叠性，本项目软件在作为课程项目的同时，作为参赛作品参与TapTap「聚光灯」21天GameJam开发比赛。项目基于满足课程要求和赛题要求的前提下，完成了本款2D横版条约游戏——《鲁米娜：逐光之旅》的Demo开发与上线。

### 1.1.2 项目意义

本款游戏的核心概念是「Light」。我希望通过这款游戏，展现本人自疫情以来对生活、社会以及后疫情时代的思考，并以此让更多人能在娱乐的同时，从中获得感悟与激励。同时，基于本项目软件是一款游戏软件，我同时想传达如此一个理念：游戏仅是游戏，这即是游戏最重要的意义。

## 1.2 项目基本目标

游戏核心玩法设计为2D横版平台跳跃类游戏 + 碎片化剧情。

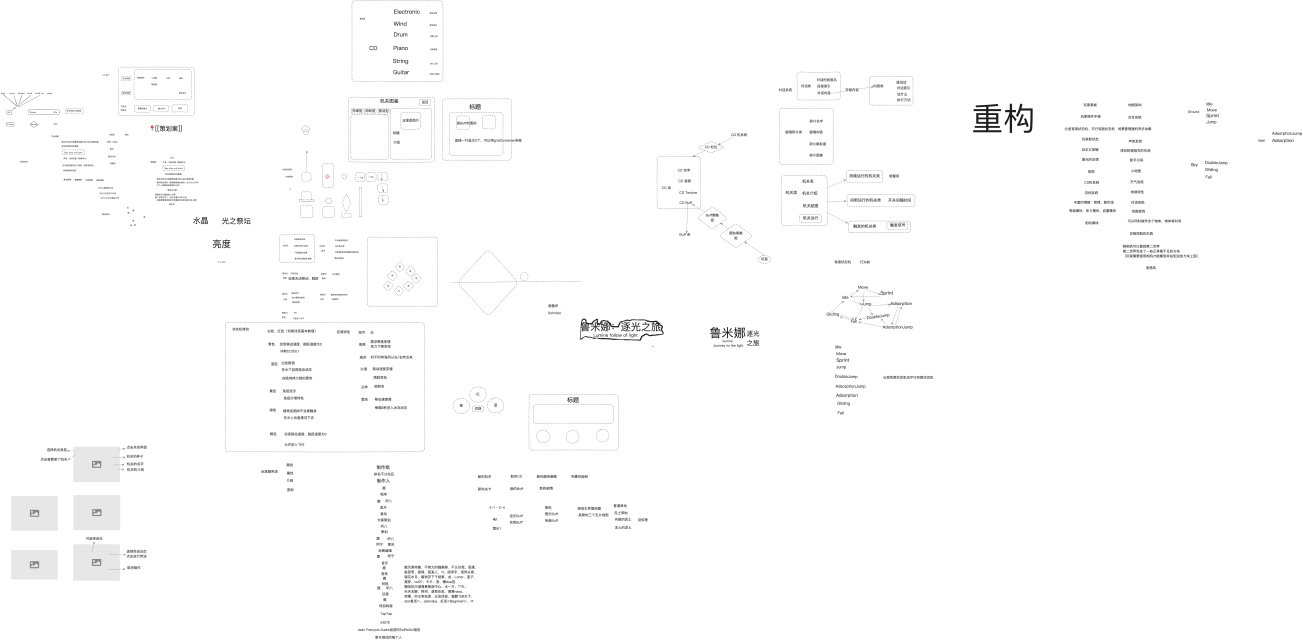


图 1-1 初步头脑风暴（较为混乱的笔记）

本次开发主体目标是🌧完成Demo版本的开发与上线。经过构思与设计，Demo版本应实现完成以下功能设计开发：

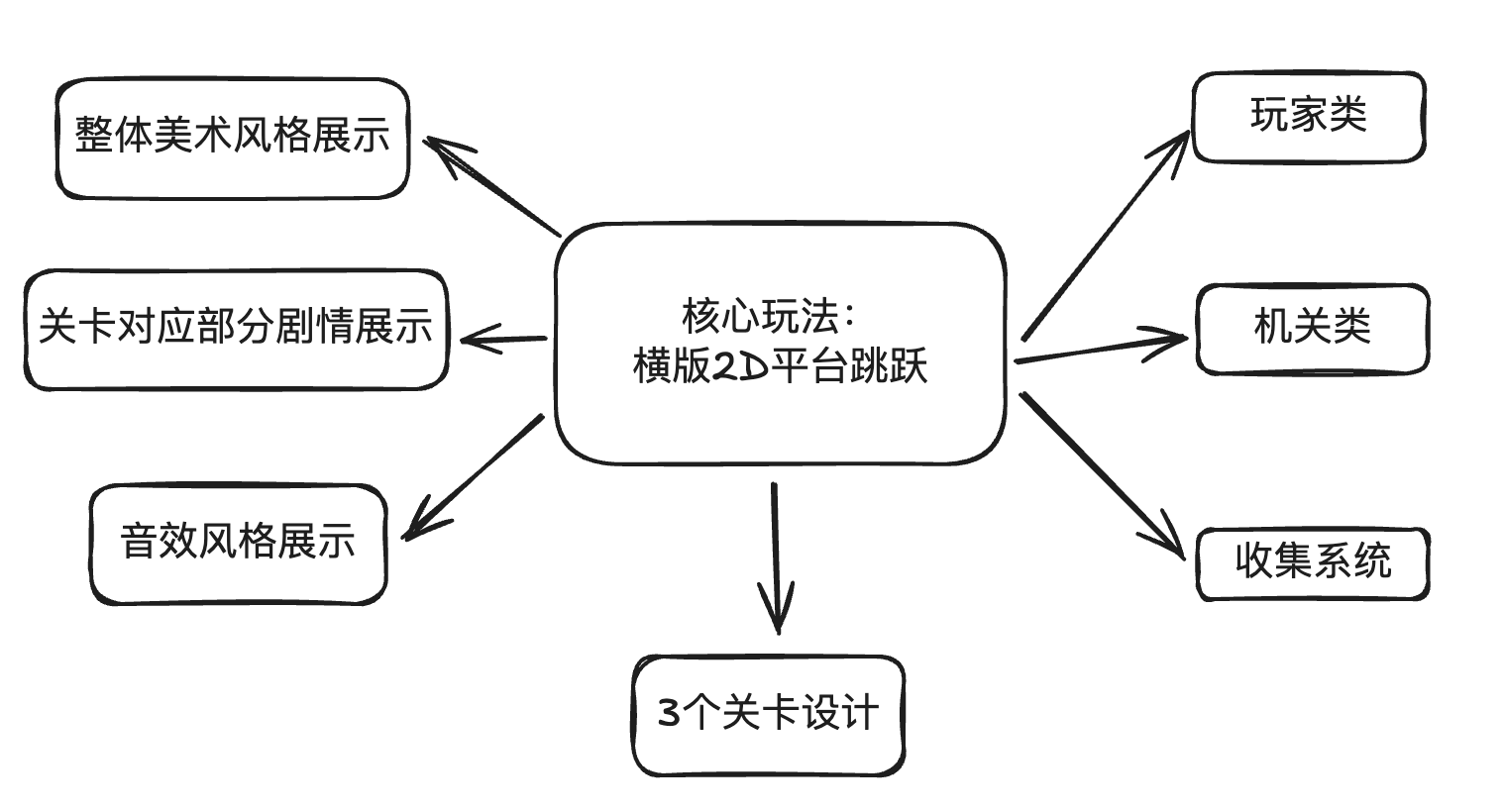


图 1-2 Demo目标

此处（图1-2）并未列出诸如相机类、地图编辑器等通用型类的实现，而主要展示关键需要完成的目标。

## 1.3 可行性分析

通过以上对项目背景、项目意义和项目基本目标的考察研究，并结合自身能力和项目开发平台等因素进行综合考量，现从技术、经济可行性等方面分析项目可行性：

1、技术可行性

本次项目依托Godot游戏引擎进行开发工作，使用gds和内嵌C#进行开发。Godot引擎内置的管道，以及本人之前自己开发的双网格系统等插件，使得本项目的开发在技术上具有可行性。

2、经济可行性

作为技术人员，游戏软件开发上线过程中的主要开销在于美术、音效、文案素材和平台上线费。在此非常感谢一位外校朋友协助完成了游戏美术资源的绘制，音效、文案以及Shader由本人独立制作，并选择了无上线费用的TapTap平台进行上线，在经济上满足了项目的可行性。

综上所述：本项目具有极高可行性，可以在21天内完成3次版本的迭代开发与测试。

## 1.4人员管理和项目进度管理

**1.4.1 人员管理**

本项目的策划、程序开发、音效职位均由本人承担。

在此感谢一位美院朋友协助完成美术绘制。由于是私人朋友且非本校人员，此处不提及姓名。

**1.4.2 项目进度管理**

项目具体开发进度如下：

表 1-1 项目开发进度管理

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 任务名称 | 开始时间 | 截止期限 | 持续时间 | 完成度 |
| 0.0.1（玩家类&&部分UI） | 2024/10/8 | 2024/10/12 | 4 | 100% |
| 0.0.2（机关类&&部分UI） | 2024/10/12 | 2024/10/15 | 3 | 100% |
| 0.0.3（UI && Shader） | 2024/10/15 | 2024/10/18 | 3 | 100% |
| 0.1.0（Map1&&NPC类） | 2024/10/18 | 2024/10/21 | 3 | 100% |
| 0.1.1（Map2&&对话系统&&音效） | 2024/10/21 | 2024/10/25 | 4 | 100% |
| 0.1.2（Map3&&CD类&&文案） | 2024/10/25 | 2024/10/27 | 2 | 100% |
| 1.0.0（BUG修复） | 2024/10/27 | 2024/10/28 | 1 | 100% |

由于是独立开发，且版本迭代速度快，任务量大且种类繁多、难以归类，甘特图反而不如常规表格直观清晰，故并未采用甘特图的方式进行项目管理，而是采用表格+Trello任务面板的方式进行项目进度管理。

# **2** 需求分析

## 2.1 需求分析概述

以下对项目使用 传统NABCD 模型进行说明。

**·N（Need）：解决用户的什么需求?**

该游戏的核心目标是解决玩家对富有深度、寓意性的游戏体验的需求。用户不仅想要娱乐，也希望在游戏中获得情感上的共鸣和哲理性的启发。游戏的主题「Light」体现了对后疫情时代生活、社会的思考，满足了玩家在娱乐中反思生活和自我成长的需求。

**·A（Approach）：有什么方法来解决用户关注的问题？**

通过设计2D横版条约游戏，玩家在游戏中不仅体验娱乐性，还能通过核心概念“Light”触及更深层的思考。游戏机制、视觉效果和剧情设计都围绕“光”这一主题展开，期望通过游戏内的交互与故事情节传达更深刻的社会与情感层面的信息。

**·B（Benefit）：给用户带来什么好处？**

作为一款2D横版游戏，提供了丰富的娱乐体验，玩家可以在游戏过程中获得乐趣。玩家在游戏过程中能够感受到深刻的哲理思考，尤其是与“Light”相关的主题，能够在娱乐的同时反思自我和社会。通过后疫情时代的背景和思考，帮助玩家与游戏角色建立情感联系，提升玩家的情感投入。

**·C（Competitors）：有无类似的竞争者，他们的产品怎么样？**

在市场上，有一些同类型的游戏也以哲理和情感为核心，例如《Limbo》、《The End is Neigh》、《Ori》等。这些游戏不仅注重玩法设计，还在视觉和剧情方面有独特的艺术表现，能够引发玩家深刻的情感共鸣，堪称“第九艺术”。

相比起其他软件产品，游戏的竞争力并不完全取决于“创新”，而更依赖自身的特色。故作为一个游戏开发者，我并不将其视为竞争者，而视为学习借鉴的对象。

**·D（Delivery）：如何推广？**

作为TapTap「聚光灯」21天GameJam比赛的参赛作品，可以通过TapTap平台进行宣传与推广，利用平台的用户流量和社区互动提高曝光度。通过微博、微信、B站等社交平台进行宣传，发布游戏的开发进度、游戏理念以及试玩视频，吸引潜在玩家的关注。与游戏评测博主、YouTube或B站的UP主合作，进行试玩视频和评测，借助玩家口碑传播吸引更多关注。通过游戏开发者与玩家的互动，加入用户反馈环节，持续优化游戏体验，增加玩家的粘性和参与度。

从以上分析其实可以看出，传统的NABCD模型并不非常适用于本项目的需求分析（基于本项目为一人开发+游戏类型软件）。因此，尝试用另一种NABCD模型从开发者方面来进行一次需求分析：

**·N (Need，需求)**

·课程项目需求：作为课程项目的一个组成部分，必须满足课程规定的开发要求。

·比赛需求：项目需要在TapTap「聚光灯」21天GameJam比赛中符合赛题的要求。

·功能需求：开发一个2D横版条约游戏《鲁米娜：逐光之旅》Demo，展示“Light”这一核心概念。

**·A (Approach，方法)**

·跨项目开发：该项目在课程要求和比赛要求之间进行平衡，同时满足教学目标和比赛标准。

·主题表现：游戏的核心概念围绕“Light”展开，反映了后疫情时代的思考，结合情感和哲理，带有一定的文化与思想深度。

·游戏设计：开发了2D横版条约类型的游戏，突出了游戏的“光”主题，通过视觉、剧情、关卡设计等方面体现这一概念。

**·B (Benefits，效益)**

·思想传播：通过游戏表达自疫情以来的对生活、社会以及后疫情时代的思考，期望玩家在享受游戏娱乐的同时获得启发和激励。

·技术提升：完成这样一个跨课程和竞赛要求的项目，有助于提升技术水平，特别是在游戏设计、程序开发以及项目的实际运作方面。

·比赛成果：如果项目成功参加比赛，并获得认可，可以为开发者带来奖项以及未来更多项目实习的机会。

**·C (Constraints，约束)**

·时间约束：项目需要在21天的短时间内完成，同时满足课程要求和比赛要求。时间的紧迫可能会影响开发过程中的细节和功能的完善。

·技术约束：在短时间内开发一个符合赛题的游戏Demo，需要限制某些复杂的技术实现或功能扩展。

**·D (Dangers，风险)**

·时间管理风险：由于项目同时承担课程和比赛任务，可能出现时间分配不均，导致项目进度滞后或质量不达标。

·开发质量风险：由于是Demo版本，开发可能面临游戏功能、画面、体验等方面的不完善，可能无法完全展示游戏的潜力。

·情感表达风险：游戏的哲理和思想表达如果过于隐晦或难以理解，可能无法有效传达给玩家，影响项目的预期效果。

## 2.2 UML相关需求分析图

UML是一种标准化的图形化语言，用于描述、设计和文档化软件系统的结构和行为。它通过不同类型的图来展示系统的静态结构和动态行为，帮助开发人员、设计人员、分析人员和其他相关人员更好地理解系统的设计和运行方式。

UML的主要图类型包括类图、对象图、用例图等分析图。类图会在3.1节进行详细阐述，此处我们只展示2D横版平台跳跃最核心的UML分析图（图2-1）。

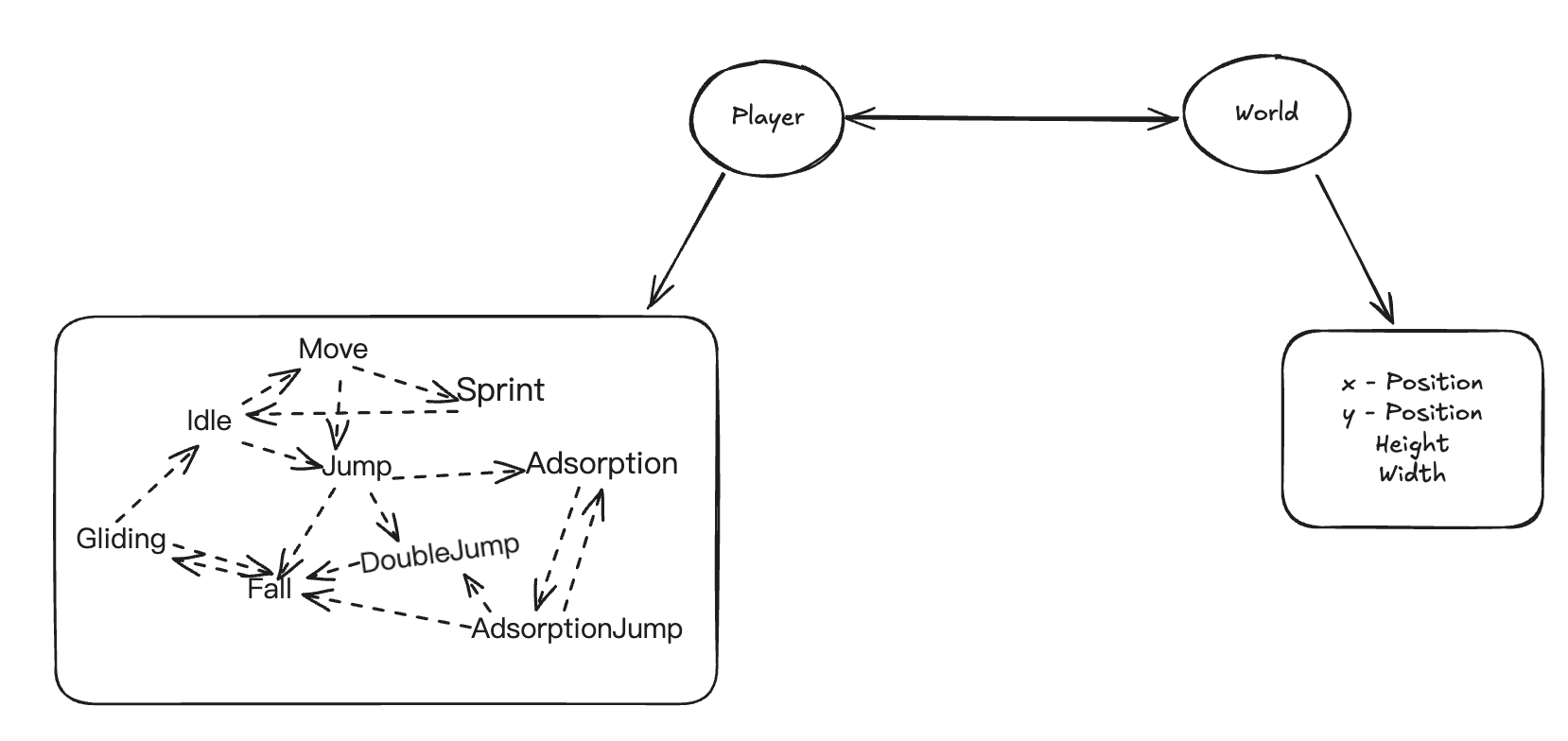


图 2-1 UML分析图

## 2.3 原型系统设计

同样地，由于单人独立开发不存在交接问题，可以采用快速轻量化项目管理与开发方式，故本项目并未使用墨刀等软件进行原型系统设计，而是直接使用Excalidraw结合Godot进行原型系统设计。（注明：与Unity引擎不同，Godot引擎极其轻量化，好比Visual Studio与VScode之间的关系与区别。在Godot中搭建UI并不比使用墨刀等软件复杂，故选择此种方式进行设计）

（1）主页面UI设计

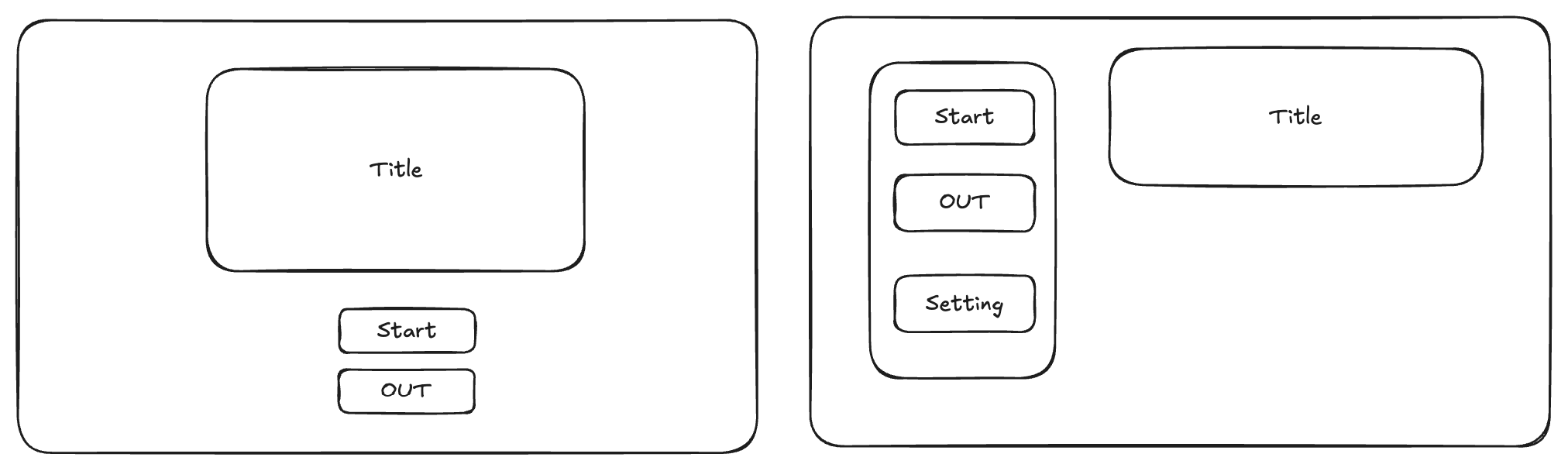


图 2-2 两版主页面UI设计

设计了两版主页面UI。主页面应当显示游戏标题，能连接到游戏主体（World Scene）、设置界面、并能正常退出游戏。最终考虑视觉效果，采用右边版本。

（2）各类UI设计

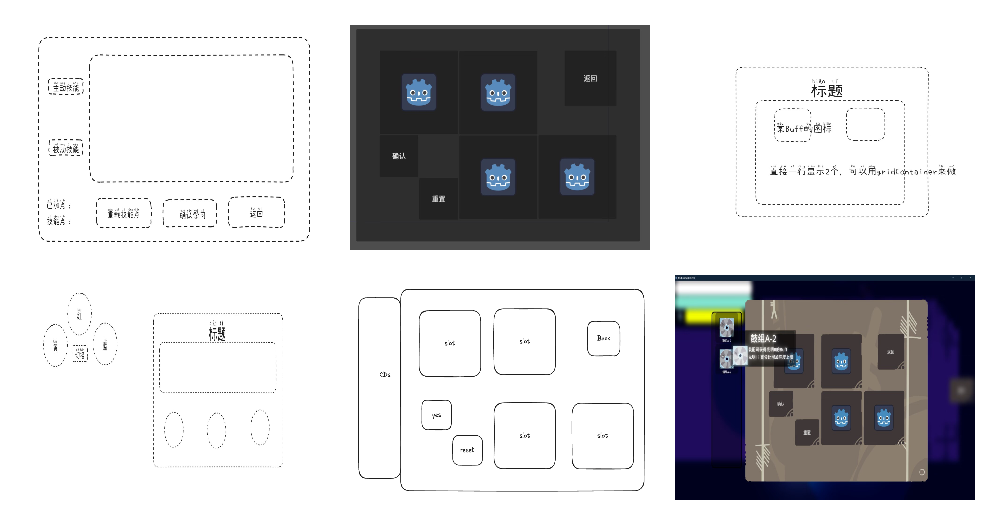


图 2-3 部分UI原型机设计

由于UI界面过多，会在第3章进行详细说明，故不在此进行一一展示。每一个UI均应实现各自对应功能和界面连接（与World Scene连接、与其他UI界面直接的彼此连接）。

（3）玩家类

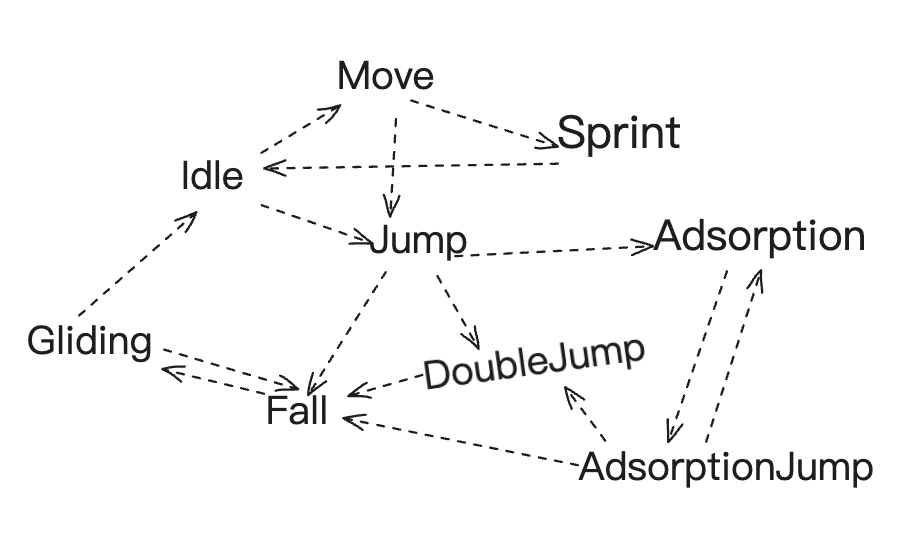


图 2-4 玩家状态设计

玩家的原型机设计主要要完成玩家状态的设计，完成玩家在站立、移动、跳跃之间转换模式的明确。

（4）机关类&&CD类&&剧情碎片类

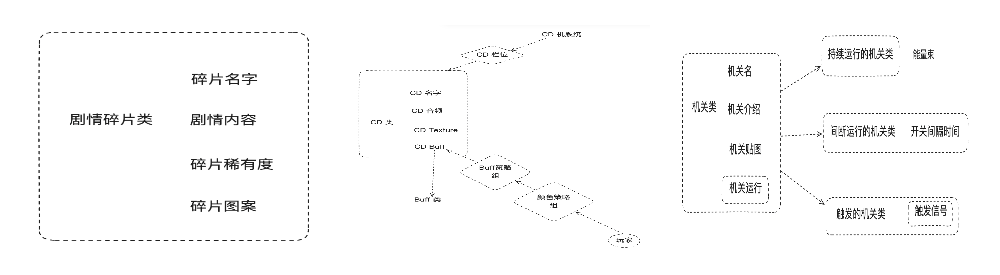


图 2-5 机关类&&CD类&&剧情类原型系设计

这些类在原型设计上不必进行过多可视化，直观呈现上与UML关系图相近，主要起到提醒开发人员类的设计作用，对玩家不透明。

（5）其它子系统

诸如GameWorld、GameEngine以及模块组件等，属于技术实现层面上的底层与优化问题，与用户需求以及前期需求分析并不相干，故不在此处进行赘述。

# **3** 概要设计和详细设计

## 3.1 系统结构与类图说明

系统结构较为复杂，不可简单以前后端来划分。以下进行一一拆解说明，并且秉持能用表格说明，不用语言赘述的原则。

一级整体模块结构如下：

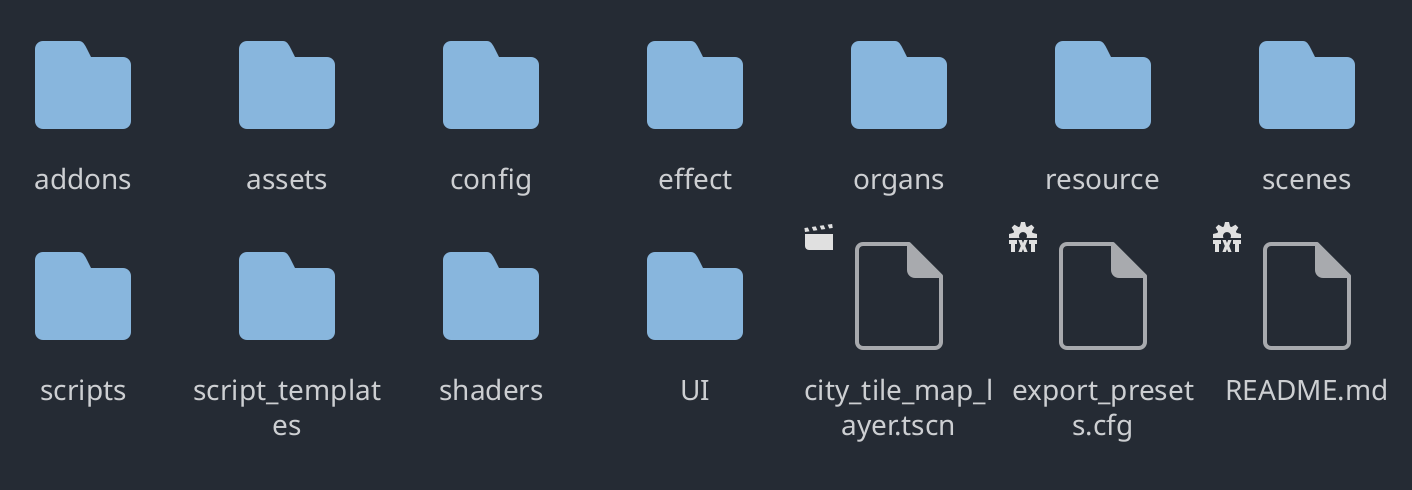


图 3-1 项目文件结构

表3-1 一级模块

|  |  |
| --- | --- |
| 模块 | 功能 |
| 资源类 | 管理游戏中各类美术、音乐资源以及资源回收 |
| script类 | 作用于整个Game的脚本管理 |
| UI类 | 各类UI上层模块，包括作用于全部UI的脚本管理 |
| Scene类 | 场景管理 |
| Buff类 | 玩家状态管理 |
| Organ类 | 包括但不限于机关等的Organ设计与管理 |

（1）Asset类（资源类）

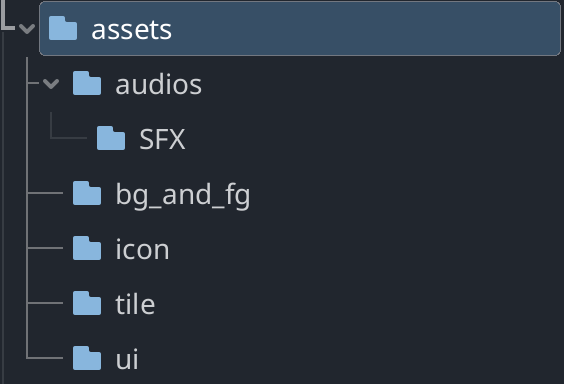


图 3-2 资源类结构

·Audio类：存储管理SFX和背景音资源文件。

·BG&FG类：存储管理全局拆分化场景美术资源文件。

·Icon类：存储管理游戏内所有Icon资源文件。

·Tile类：存储管理TileMap瓦片资源文件。

·UI类：存储管理UI相关美术资源文件。

Asset类主要负责存储管理全局非代码的资源文件。

（2）Script类

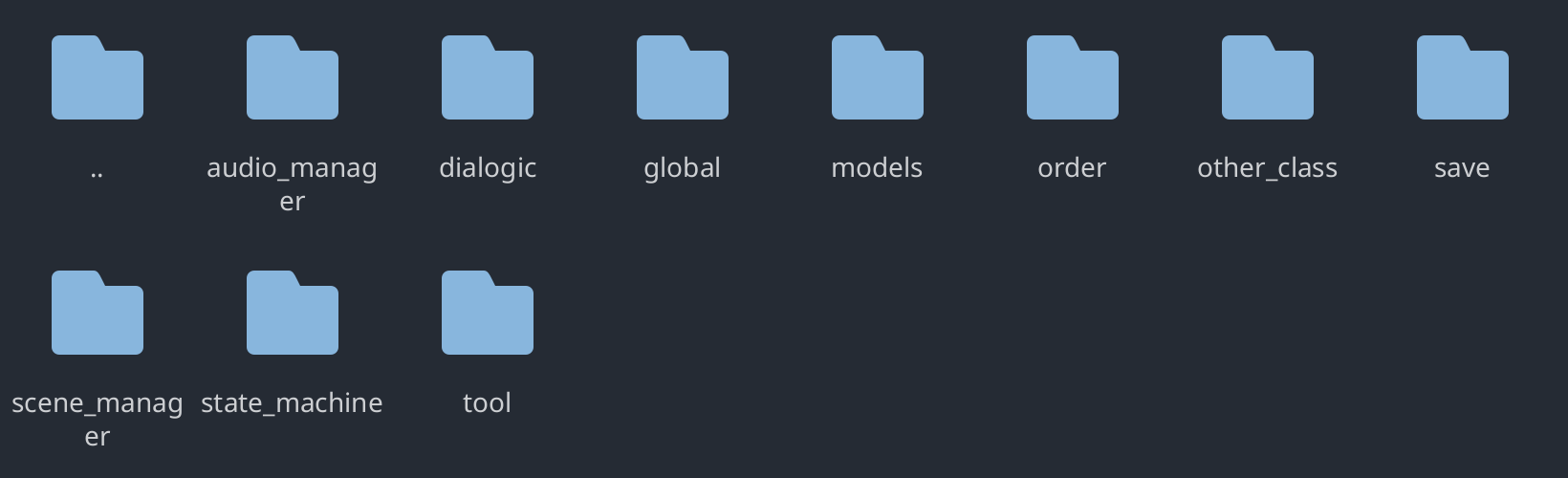


图 3-3 Script类结构

·AudioManager类：音频管理器。负责管理与播放音频/音效。

·Dialogic类：对话类。管理存储对话文本资源，实现对话框交互。

·Global类：全局数据与工具管理。包含Global数据脚本、GlobalValue数据脚本、UITool&&其它Tool脚本。

·Model类：作用于WorldScene的全局模块管理。包括生命模块，时间模块，天气模块，体力模块，等级模块。

·Order类：供给开发人员的测试调试工具类。包括代码控制脚本，外挂脚本等。

·Save类：存档系统与资源管理回收。为玩家提供存档系统并实现全局资源的管理回收，实现优化。

·SceneManager类：场景管理器。对游戏中全部（且大量）的场景进行管理。

·StateMachine类：状态管理器。其中实现state脚本模拟一个接口类，在sate\_matchine脚本中真正实现状态机管理控制。

·Tool类：供给开发人员的开发工具。功能是实现在Godot引擎TODO显示台中直观显示待开发功能和待修复BUG。

·其他类：存放其它Demo阶段不必单设文件目录的类。例如CD管理器、机关管理器等。

Script类主要存放作用于WorldScene或者全局的脚本，以及子模块管理器。

（3）UI类

由于Demo阶段设计到的UI较少，仅有22个，故不必做二级分类。

UI类实现了各个UI场景文件与挂载脚本的存储管理。

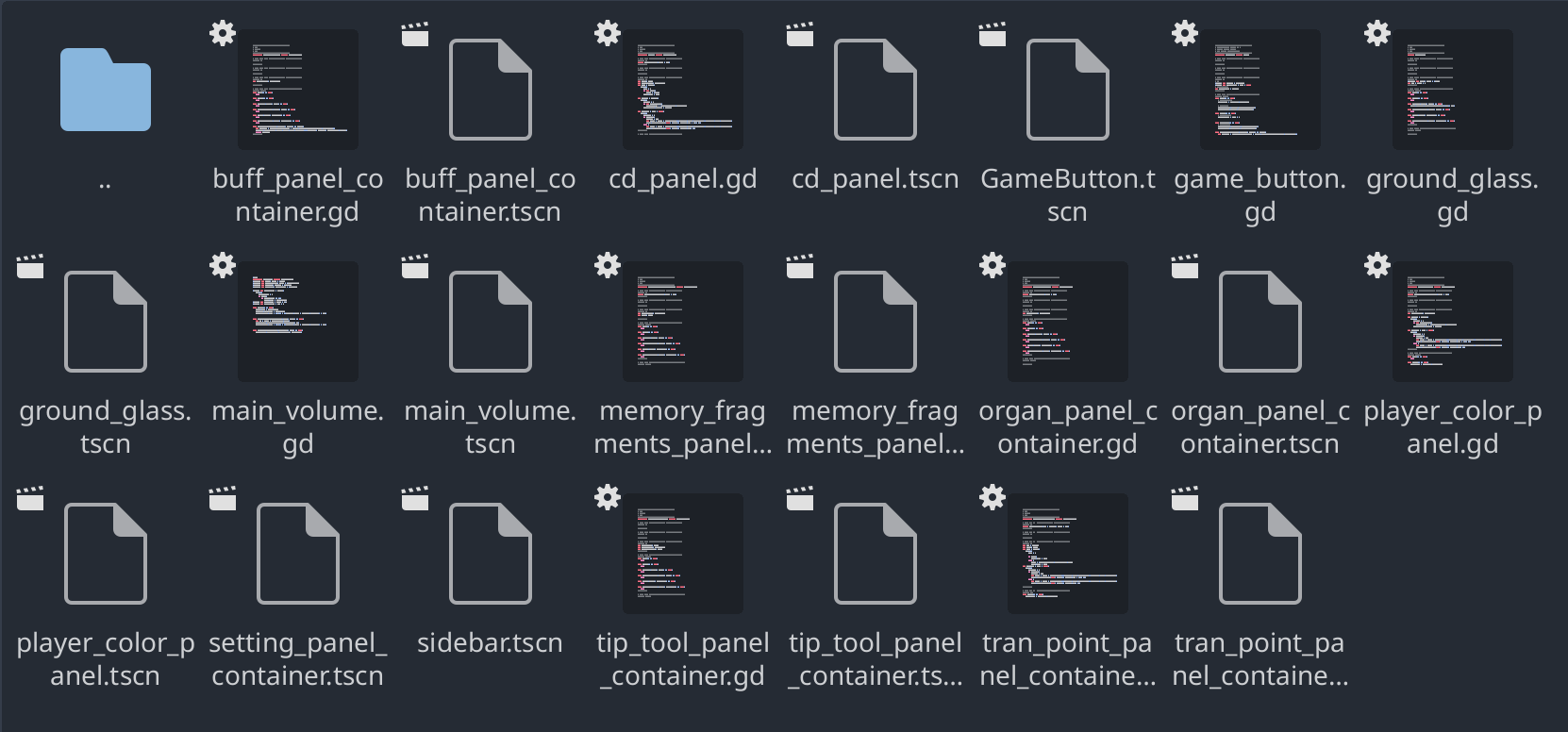


图 3-4 UI类下级结构

（4）Scene类

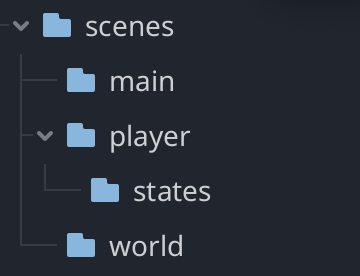


图 3-5 Scene类结构

·Main类：开始场景。

·World类：GameWorld。包括整个游戏TileMap场景以及存储管理。

·Player类：玩家。其中player.gd模拟一个接口类，在States中真正完成对玩家状态机的实现。

Scene类完成了对游戏Scene的管理，属于最不能丢失出错的类。

（5）BUFF类

同样基于Demo体量问题，并没有对BUFF类相关代码单设文件夹进行管理，而是拆分到各级目录中。BUFF类主要实现对玩家状态的管理，可以拆分成以下结构：

·PlayerBuff类：对由作用于Player上的Model（生命模块、体力模块、等级模块）造成的BUFF进行存储管理。其同样使用模拟接口类player.gd。

·WorldBuff类：对由作用于World上的Model（时间模块、天气模块）造成的BUFF进行存储管理，最终作用于Global（通过管道实现）和Player上（通过模拟接口类player.gd实现）。

注意，我们并不在BUFF类里完成对BUFF的实现，而是实现对其的定义管理，提供接口。下面的Organ类同理。

（6）Organ类（机关类）

与BUFF类一样，拆分到其它目录中，实现对各类机关（机关、CD）的存储管理。可以拆分成以下结构：

·Organ类：又可拆分为持续型机关、触发型机关、间发型机关三子类，对这三类机关进行整体的管理。Organ类与GlobalValue通过管线进行数据交换。

·CD类：对CD进行整体管理。CD是本游戏中单设的音效BUFF，根据不同的CD音效可造成不同的场景效果和BUFF效果。CD类与GlobalValue通过管线进行数据交换，通过模拟接口类player.gd实现与Player的数据交换。

## 3.2 关键数据说明

本项目涉及的数据定义过多，如果对每一项进行说明，则报告篇幅不合理且过于赘述。故不进行一一说明，仅对其中Global类、GlobalValue类、Player类，World场景节点结构进行展示。

在此对粘贴代码行为表示抱歉，但这已经是最好的展示方式。

（1）Global类

1. # TODO 全局 ===============>信 号<===============
2. #region 信号
3. signal war\_of\_pursuit
4. signal ready\_safe\_lumine
5. #endregion
6. # TODO 全局 ===============>常 量<===============
7. #region 常量
8. const TRAIL = preload("res://effect/trail.tscn")
9. const CD\_PROP = preload("res://resource/props/cd\_prop.tscn")
10. const MEMORY\_PROP = preload("res://resource/props/memory\_prop.tscn")
11. #endregion
12. # TODO 全局 ===============>变 量<===============
13. #region 变量
14. # 全局：定义全局场景过度的 shader 效果
15. var shader\_index = 1
16. # 全局：传送点保存
17. var spawn\_points : Dictionary = {"初始点" : Vector2(-360, 300)}
18. # 全局：玩家
19. var player : Player
20. # 全局：环境
21. var world\_environment : WorldEnvironment
22. # 全局：颜色
23. var global\_canvas\_modulate: CanvasModulate
24. # 全局：玩家拥有颜色
25. var has\_player\_colors : Array[PlayerColor] = [preload("res://resource/player\_colors/player\_color\_orange.tres")]
26. # 全局：玩家携带颜色
27. var player\_colors : Array[PlayerColor] = [preload("res://resource/player\_colors/player\_color\_white.tres"), preload("res://resource/player\_colors/player\_color\_red.tres"), preload("res://resource/player\_colors/player\_color\_cyan.tres")]
28. # 全局：玩家获得的回忆碎片
29. var memory\_fragments : Array[MemoryFragment] = []
30. # 全局：是否与红光对话了
31. var red\_dialogic\_enabled : bool = false
32. # 全局：计时器
33. var ending\_timer : Timer
34. # 全局：是否开始追逐战
35. var is\_war\_of\_pursuit : bool = false:
36. set(v):
37. is\_war\_of\_pursuit = v
38. if is\_war\_of\_pursuit:
39. war\_of\_pursuit.emit()
40. var evil\_lumine\_is\_relly\_attack : bool = false:
41. set(v):
42. evil\_lumine\_is\_relly\_attack = v
43. if evil\_lumine\_is\_relly\_attack:
44. ready\_safe\_lumine.emit()
45. # 全局：跳跃教程
46. var learning\_jumping : bool = false
47. # 全局：漂浮教程
48. var learning\_gliding : bool = false
49. # 全局：冲刺教程
50. var learning\_sprinting : bool = false
51. # 全局：是否加载存档
52. var is\_load\_save : bool = false
53. # 全局：是否加载指令
54. var is\_command : bool = false
55. # 全局：是否显示提示
56. var key\_tip\_enabled : bool = true
57. # 全局：相机模式
58. var camera\_mode\_enabled : bool = true
59. # 全局：是否全屏
60. var full\_screen\_enabled : bool = false
61. # 全局：是否开启辉光
62. var glow\_enabled : bool = true:
63. set(v):
64. glow\_enabled = v
65. if SceneManager.current\_scene\_index == 1:
66. await get\_tree().process\_frame
67. world\_environment.environment.glow\_enabled = glow\_enabled
68. # 全局：音量
69. var main\_volume : float = .6
70. var sfx\_volume : float = 1
71. var bgm\_volume : float = 1
72. var ambient\_volume : float = 1
73. # 全局：是否静音
74. var is\_main\_volume : bool = false
75. var is\_sfx\_volume : bool = false
76. var is\_bgm\_volume : bool = false
77. var is\_ambient\_volume : bool = false
78. #endregion

（2）GlobalValue类

1. # TODO 全局属性 ===============>信 号<===============
2. #region 信号
3. signal value\_update
4. signal buff\_update
5. #endregion
6. # TODO 全局属性 ===============>常 量<===============
7. #region 常量
8. # 全局属性：最大玩家等级
9. const MAX\_PLAYER\_LEVEL : int = 50
10. # 全局属性：最大技能等级
11. const MAX\_PLAYER\_SKILL\_LEVEL : int = 5
12. # 全局属性：每级 buff 提升的亮度比例
13. const HEALTH\_BUFF\_RADIO : float = .1
14. # 全局属性：每级 buff 提升的冲刺距离
15. const SPRINT\_BUFF\_RADIO : float = .1
16. # 全局属性：每级 buff 提升体力比例
17. const ABUNDANT\_BUFF\_RADIO : float = .1
18. # 全局属性：每级 buff 减少冲刺cd比例
19. const CONTINUE\_SPRINT\_BUFF\_RADIO : float = .15
20. # 全局属性：每级 buff 减少的下落速度
21. const FEATHER\_FALL\_BUFF\_RADIO : float = .15
22. # 全局属性：每级 buff 增加的强电比例
23. const RAY\_DESTROY\_BUFF\_RADIO : float = .4
24. #endregion
25. # TODO 全局属性 ===============>变 量<===============
26. #region 变量
27. # 定义全局重力
28. var gravity : Vector2 = Vector2(0, 9.8)
29. #region 技能树相关属性
30. # 全局属性：技能点
31. var skill\_point : int = 0:
32. set(v):
33. skill\_point = v
34. if skill\_point <= 0:
35. skill\_point = 0
36. # 全局属性：已使用技能点
37. var used\_skill\_point : int = 0:
38. set(v):
39. used\_skill\_point = v
40. # 全局属性：玩家等级
41. var player\_level : int = 1:
42. set(v):
43. player\_level = v
44. if player\_level >= MAX\_PLAYER\_LEVEL:
45. player\_level = MAX\_PLAYER\_LEVEL
46. # 全局属性：玩家亮度
47. var player\_luminance\_level : int = 0:
48. set(v):
49. player\_luminance\_level = v
50. if player\_luminance\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
51. player\_luminance\_level = MAX\_PLAYER\_SKILL\_LEVEL
52. value\_update.emit()
53. # 全局属性：玩家体力
54. var player\_strength\_level : int = 0:
55. set(v):
56. player\_strength\_level = v
57. if player\_strength\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
58. player\_strength\_level = MAX\_PLAYER\_SKILL\_LEVEL
59. value\_update.emit()
60. # 全局属性：玩家冲刺
61. var player\_sprint\_cd\_level : int = 0:
62. set(v):
63. player\_sprint\_cd\_level = v
64. if player\_sprint\_cd\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
65. player\_sprint\_cd\_level = MAX\_PLAYER\_SKILL\_LEVEL
66. value\_update.emit()
67. # 全局属性：玩家跳跃高度
68. var player\_jump\_height\_level : int = 0:
69. set(v):
70. player\_jump\_height\_level = v
71. if player\_jump\_height\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
72. player\_jump\_height\_level = MAX\_PLAYER\_SKILL\_LEVEL
73. value\_update.emit()
74. # 全局属性：玩家体力回复
75. var player\_strength\_recover\_level : int = 0:
76. set(v):
77. player\_strength\_recover\_level = v
78. if player\_strength\_recover\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
79. player\_strength\_recover\_level = MAX\_PLAYER\_SKILL\_LEVEL
80. value\_update.emit()
81. # 全局属性：玩家血量降低
82. var player\_health\_expend\_level : int = 0:
83. set(v):
84. player\_health\_expend\_level = v
85. if player\_health\_expend\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
86. player\_health\_expend\_level = MAX\_PLAYER\_SKILL\_LEVEL
87. value\_update.emit()
88. # 全局属性：玩家体力降低
89. var player\_strength\_expend\_level : int = 0:
90. set(v):
91. player\_strength\_expend\_level = v
92. if player\_strength\_expend\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
93. player\_strength\_expend\_level = MAX\_PLAYER\_SKILL\_LEVEL
94. value\_update.emit()
95. # 全局属性：玩家吸附下滑速度
96. var player\_adsorp\_down\_level : int = 0:
97. set(v):
98. player\_adsorp\_down\_level = v
99. if player\_adsorp\_down\_level >= MAX\_PLAYER\_SKILL\_LEVEL:
100. player\_adsorp\_down\_level = MAX\_PLAYER\_SKILL\_LEVEL
101. value\_update.emit()
102. # 全局属性：玩家空中转体
103. var can\_air\_twist : bool = false
104. # 全局属性：玩家二段跳
105. var can\_double\_jump : bool = false
106. # 全局属性：玩家吸附
107. var can\_adsorption\_jump : bool = false
108. # 全局属性：玩家破坏激光
109. var can\_ray\_destory : bool = false
110. # 全局属性：玩家交互激光
111. var can\_ray\_interact : bool = false
112. #endregion
113. # 全局属性：亮度提升Buff
114. var health\_buff\_level : int = 0:
115. set(v):
116. health\_buff\_level = v
117. buff\_update.emit()
118. # 全局属性：冲刺距离Buff
119. var sprint\_buff\_level : int = 0:
120. set(v):
121. sprint\_buff\_level = v
122. buff\_update.emit()
123. # 全局属性：充沛Buff
124. var abundant\_buff\_level : int = 0:
125. set(v):
126. abundant\_buff\_level = v
127. buff\_update.emit()
128. # 全局属性：连冲Buff
129. var continue\_sprint\_buff\_level : int = 0:
130. set(v):
131. continue\_sprint\_buff\_level = v
132. buff\_update.emit()
133. # 全局属性：羽落Buff
134. var feather\_fall\_buff\_level : int = 0:
135. set(v):
136. feather\_fall\_buff\_level = v
137. buff\_update.emit()
138. # 全局属性：强电buff
139. var ray\_destroy\_time\_level : int = 0:
140. set(v):
141. ray\_destroy\_time\_level = v
142. buff\_update.emit()
143. #endregion

（3）模拟接口类Player.gd

1. # TODO 玩家 ===============>信 号<===============
2. #region 信号
3. #endregion
4. # TODO 玩家 ===============>常 量<===============
5. #region 常量
6. #endregion
7. # TODO 玩家 ===============>变 量<===============
8. #region 变量
9. # 玩家的激光射线
10. @onready var ray\_cast\_2d: RayCast2D = $RayCast2D
11. # 玩家的激光贴图
12. @onready var ray\_line\_2d: Line2D = $RayLine2D
13. # 玩家的灯光
14. @onready var point\_light\_2d: PointLight2D = $Light/PointLight2D
15. # 玩家的生命模块
16. @onready var health\_model: HealthModel = $HealthModel
17. # 玩家的体力模块
18. @onready var strength\_model: StrengthModel = $StrengthModel
19. # 玩家的经验模块
20. @onready var level\_model: LevelModel = $LevelModel
21. # 玩家的碰撞体
22. @onready var collision\_shape\_2d: CollisionShape2D = $CollisionShape2D
23. # 玩家的图像
24. @onready var deer\_polygon: DeerPolygon = $DeerPolygon
25. # 玩家的速度
26. @export var speed : float = 350
27. # 玩家的跳跃速度
28. var jump\_speed : float
29. @export var max\_jump\_speed : float = 500:
30. set(v):
31. max\_jump\_speed = v
32. \_on\_global\_value\_value\_update()
33. # 玩家的冲刺速度
34. @export var sprint\_speed : float = 1200
35. # 玩家吸附时下落速度
36. @export var adsop\_down\_speed : Vector2 = Vector2(0., 2.4)
37. # 玩家存档点
38. @export var spawn\_pos : Marker2D
39. # 玩家的方向
40. var dir : float
41. # 玩家最后的方向
42. var last\_dir : float
43. # 玩家跳跃次数
44. var jump\_count : int = 2
45. # 玩家是否跳跃
46. var is\_jump : bool = false
47. # 玩家是否滑翔
48. var is\_gliding : bool = false
49. # 玩家是否吸附
50. var is\_adsorption : bool = false
51. # 玩家是否死亡
52. var is\_dead : bool = false:
53. set(v):
54. is\_dead = v
55. collision\_shape\_2d.set\_deferred("disabled", is\_dead)
56. # 玩家是否冲刺
57. var is\_sprint : bool = false
58. # 玩家是否能冲刺
59. var can\_sprint : bool = true:
60. set(v):
61. can\_sprint = v
62. if not can\_sprint:
63. sprint\_cd = max\_sprint\_cd
64. # 玩家是否受伤
65. var is\_hurt : bool = false
66. # 玩家受伤方向
67. var hurt\_dir : Vector2 = Vector2.ZERO
68. # 玩家冲刺CD
69. var max\_sprint\_cd : float = 3:
70. set(v):
71. max\_sprint\_cd = v
72. if max\_sprint\_cd <= .1:
73. max\_sprint\_cd = .1
74. # 玩家当前冲刺CD
75. var sprint\_cd : float
76. # 玩家残影出现的CD
77. var trail\_time : float = .06
78. # 玩家是否能回复光源
79. var can\_heal\_light : bool = false
80. # 玩家是否处于紫光范围
81. var is\_in\_purple\_light : bool = false
82. # 玩家是否在交互
83. var interact : bool = false
84. # 玩家是否飞行
85. var is\_fly : bool = false
86. # 玩家重力
87. var gravity : Vector2
88. # 玩家漂浮倍率
89. var gliding\_gravity\_level : float = 4
90. # 玩家生命消耗速度
91. var health\_expend\_speed : float = .4
92. # 玩家体力消耗速度
93. var strength\_expend\_speed : float = 1
94. # 玩家是否在雪山范围
95. var not\_in\_snowy\_mountain : bool = true
96. # 玩家重置
97. var is\_back\_spawn\_pos : bool = false
98. # 玩家重置需要的时间
99. var back\_cd : float = 3:
100. set(v):
101. back\_cd = v
102. if back\_cd <= 0:
103. # 使用 Tween 实现将玩家传送回存档点
104. var tween : Tween = create\_tween().set\_ease(Tween.EASE\_IN\_OUT).set\_trans(Tween.TRANS\_BACK).set\_parallel()
105. tween.tween\_property(self, "scale", Vector2.ZERO, 0.75)
106. tween.chain().tween\_property(self, "global\_position", spawn\_pos.global\_position, 0.75)
107. tween.chain().tween\_property(self, "scale", Vector2.ONE, 0.75)
108. await tween.finished
109. back\_cd = 3
110. # 玩家激光
111. var ray\_shot : bool = false:
112. set(v):
113. ray\_shot = v
114. if not ray\_body:
115. ray\_destory\_time = 5
116. # 玩家激光摧毁方块速度
117. var max\_ray\_destroy\_time : float = 3
118. var ray\_destory\_time : float = 3:
119. set(v):
120. ray\_destory\_time = v
121. if ray\_destory\_time <= 0:
122. ray\_destory\_time = max\_ray\_destroy\_time \* \
123. (1 - GlobalValue.ray\_destroy\_time\_level \* GlobalValue.RAY\_DESTROY\_BUFF\_RADIO)
124. if ray\_body:
125. ray\_body.queue\_free()
126. ray\_body = null
127. # 玩家的激光碰到的机关
128. var ray\_body : CharacterBody2D:
129. set(v):
130. ray\_body = v
131. if ray\_body == null:
132. gravity = GlobalValue.gravity
133. # 玩家的 Buff 组
134. var buffs : Array[BaseBuff]
135. # 玩家的 颜色策略组
136. var player\_color : PlayerColor:
137. set(v):
138. player\_color = v
139. if player\_color != null:
140. player\_color.player\_color\_effect()
141. # 玩家在加buff前最大生命值
142. var max\_health : float
143. # 玩家在加buff前最大体力
144. var max\_strength : float
145. # 玩家的默认冲刺 cd
146. var normal\_sprint\_cd : float = 3.:
147. set(v):
148. normal\_sprint\_cd = v
149. \_on\_global\_value\_value\_update()
150. # 玩家钩索
151. var is\_joit : bool = false:
152. set(v):
153. is\_joit = v
154. # 玩家是否是轻的
155. var is\_light : bool
156. # 玩家的额外重力
157. var ex\_gravity : Vector2 = Vector2.ZERO
158. # 玩家在水晶
159. var in\_the\_crystal : bool = false
160. # 玩家无法跳跃
161. var cant\_jump : bool = false
162. #endregion

（4）WorldScene下节点结构

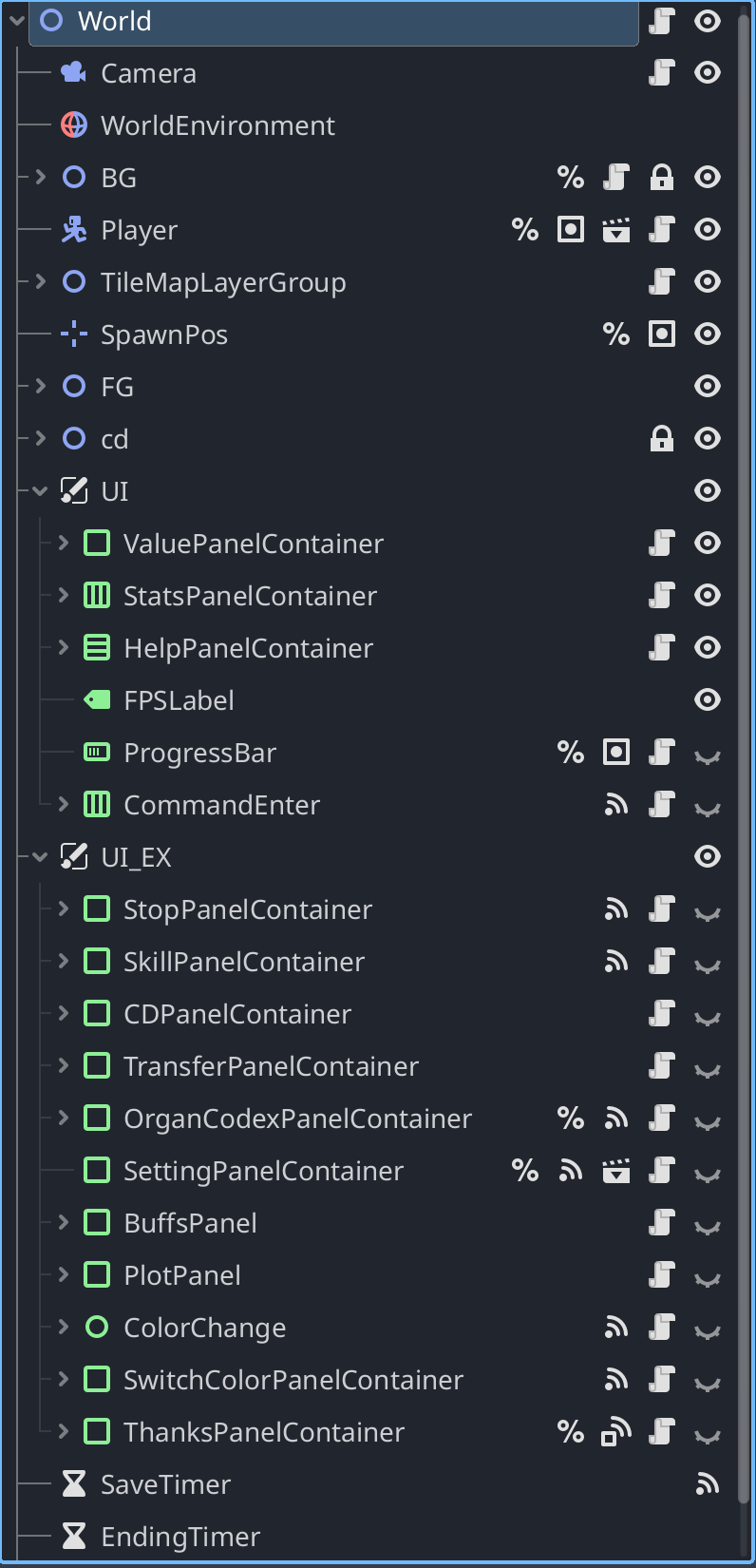


图 3-6 World节点结构

## 3.3 数据管理说明

### 3.3.1 数据管理

游戏软件中的数据管理主要涉及如何高效地存储、读取和处理各种游戏数据。数据管理的核心目标是确保游戏数据能够被有效地存储和快速地加载，同时避免内存泄漏和冗余数据的产生。常见的游戏数据管理包括资源管理、场景管理、状态管理等，市面上的游戏引擎（如UE、Unity、Godot等）本身也都实现了高效的资源加载与管理，确保游戏在不同平台上的兼容性与性能。

**·资源管理**

游戏中的资源包括图形、音频、动画、模型、纹理等。为了使游戏运行高效，游戏开发者通常会使用资源池来管理这些资源。资源池是一种缓存机制，当游戏需要使用某一资源时，首先从资源池中查找，如果没有则加载资源并存储在池中，避免重复加载。通过这种方式，可以避免频繁的磁盘IO操作，提高游戏的运行效率。

但是，本游戏考虑体量问题，并没有采用资源池这一解决方案，而是采用了资源压缩的优化方法。通过对资源文件进行压缩，不仅能节省存储空间，还能减少加载时间。

**·场景管理**

游戏中的每一关卡或场景都包含了大量的对象、数据和互动信息。场景管理的任务是确保游戏世界的各个部分能够按需加载，并在不再需要时及时卸载。为了实现高效的场景切换，很多游戏引擎采用了分区加载（LOD）技术，即根据玩家的位置和视角来动态加载和卸载游戏中的资源。例如，玩家靠近某个区域时，相关的纹理和模型会被加载进内存，远离时则会被卸载。

但是需要提到的是，Godot引擎并没有采用这一技术，需要开发者自己实现。本项目中考虑到体量问题，并没有采用LOD技术，而是采用了同步加载。

**·状态管理**

游戏的状态管理涉及到玩家的进度、游戏的各种设定、物体的状态等。通过状态机来管理不同的游戏状态（如菜单状态、暂停状态、游戏进行状态等）。

### 3.3.2 垃圾回收

垃圾回收（GC）是指自动回收不再使用的内存资源，从而避免内存泄漏的技术。在游戏开发中，垃圾回收不仅需要处理传统的内存管理问题，还需要在高性能要求的环境下优化垃圾回收的效率，以避免影响游戏的流畅性。

垃圾回收机制的基本原理是：游戏中的对象通常由程序通过堆内存分配来创建，随着游戏的进行，一些对象可能不再需要使用。垃圾回收的任务就是自动检测这些不再使用的对象，并释放其占用的内存。

常见的垃圾回收策略有两种：引用计数法和标记-清除法。本项目采用标记-清除法。

·标记-清除法：通过“标记”可达对象（即当前还有其他对象引用的对象），然后“清除”那些无法访问的对象。该方法可以解决引用计数的循环引用问题，但会引入停顿问题。

同时，在GC优化上，本项目采用延迟垃圾回收的方案。

·延迟垃圾回收：在游戏的非关键时刻（如菜单界面或加载界面），游戏引擎可以延迟进行垃圾回收，避免影响实时游戏过程中的性能。

# **4** 实现与测试

## 4.1实现环境与代码管理

实验环境说明如下：

表 4-1 实现环境与托管平台说明

|  |  |  |  |
| --- | --- | --- | --- |
| 托管平台 | 引擎 | 语言 | 硬件环境 |
| GitHub | Godot4.3 | GDS && C# | 无限制 |

由于是独立开发，不存在代码贡献问题。

## 4.2 关键函数说明

与3.2节同理，由于系统较为复杂，代码量较为庞大，并非报告篇幅可以详细展现，并且必须结合代码说明，故此处我们以模拟接口类player.gd代码为例展示一下系统中模块间的调用逻辑。

在此仍旧对粘贴代码的行为表示抱歉。

1. # TODO 玩家 ===============>虚方法<===============
2. #region 常用的虚方法
3. func \_ready() -> void:
4. Global.player = self
5. health\_model.health\_update.connect(\_on\_health\_model\_health\_update)
6. health\_model.is\_dead.connect(\_on\_health\_model\_is\_dead)
7. if not Global.is\_load\_save:
8. level\_model.reset\_level()
9. ray\_line\_2d.add\_point(ray\_cast\_2d.global\_position)
10. GlobalValue.value\_update.connect(\_on\_global\_value\_value\_update)
11. GlobalValue.buff\_update.connect(\_on\_global\_value\_buff\_update)
12. \_on\_global\_value\_value\_update()
13. gravity = GlobalValue.gravity
14. AudioManager.cd\_update.connect(\_on\_audio\_manager\_cd\_update)
15. player\_color = Global.player\_colors[0]
16. func \_process(\_delta: float) -> void:
17. dir = get\_input\_dir()
18. # 判断 dir 如果非零，记录最后方向
19. if not dir == 0:
20. last\_dir = dir
21. if Input.is\_action\_just\_pressed("action\_jump") and is\_on\_floor():
22. is\_jump = true
23. if Input.is\_action\_just\_pressed("action\_sprint") and is\_on\_floor() and can\_sprint:
24. can\_sprint = false
25. is\_sprint = true
26. if Input.is\_action\_just\_pressed("action\_interact"):
27. interact = true
28. elif Input.is\_action\_just\_released("action\_interact"):
29. interact = false
30. is\_adsorption = Input.is\_action\_pressed("action\_adsorption")
31. is\_gliding = Input.is\_action\_pressed("action\_jump")
32. is\_back\_spawn\_pos = Input.is\_action\_pressed("action\_back")
33. ray\_shot = Input.is\_action\_pressed("mouse\_left") and GlobalValue.can\_ray\_destory
34. is\_joit = Input.is\_action\_pressed("mouse\_right") and GlobalValue.can\_ray\_interact
35. func \_physics\_process(delta: float) -> void:
36. move\_and\_slide()
37. ray\_shoot(delta)
38. if in\_the\_crystal:
39. health\_model.health += delta \* 4
40. strength\_model.strength += delta \* 4
41. elif can\_heal\_light:
42. health\_model.health += delta \* 2
43. strength\_model.strength += delta \* 2
44. elif is\_in\_purple\_light :
45. health\_model.health -= delta
46. else :
47. # 血量减少
48. health\_model.health -= delta \* health\_expend\_speed
49. # 冲刺状态的 CD
50. if not can\_sprint:
51. sprint\_cd -= delta \* (1 + GlobalValue.continue\_sprint\_buff\_level \* GlobalValue.CONTINUE\_SPRINT\_BUFF\_RADIO)
52. if sprint\_cd <= 0:
53. can\_sprint = true
54. if absf(velocity.x) > 600:
55. trail\_time -= delta
56. if trail\_time <= 0:
57. trail\_time = .06
58. var trail = Global.TRAIL.instantiate()
59. get\_parent().add\_child(trail)
60. get\_parent().move\_child(trail, get\_index())
61. trail.texture = deer\_polygon.texture
62. trail.global\_position = global\_position
63. # 玩家经验提升
64. # WARNING 这里为了测试修改成了10
65. if velocity.length() > 0:
66. if level\_model.level == 50:
67. level\_model.e\_x\_p = level\_model.max\_exp
68. else :
69. level\_model.e\_x\_p += delta \* 3
70. # 玩家回城
71. if is\_back\_spawn\_pos and back\_cd > 0:
72. back\_cd -= delta
73. #endregion
74. # TODO 玩家 ===============>信号链接方法<===============
75. #region 信号链接方法
76. # TODO\_FUC 玩家：根据血量更新亮度
77. func \_on\_health\_model\_health\_update(health: float, \_max\_health: float) -> void:
78. # FIXME 后面记得将这个亮度值 1 修改成一个变量
79. point\_light\_2d.texture\_scale = (health / \_max\_health) \* 1
80. # TODO\_FUC 玩家：生命模块：死亡信号
81. func \_on\_health\_model\_is\_dead() -> void:
82. is\_dead = true
83. # TODO\_FUC 玩家：等级模块：升级
84. func \_on\_level\_model\_level\_up(level: int) -> void:
85. GlobalValue.skill\_point += 2
86. GlobalValue.player\_level = level
87. # TODO\_FUC 玩家：技能属性修改
88. func \_on\_global\_value\_value\_update() -> void:
89. health\_model.max\_health = 20 + GlobalValue.player\_luminance\_level \* 10
90. max\_health = health\_model.max\_health
91. strength\_model.max\_strength = 50 + GlobalValue.player\_strength\_level \* 15
92. max\_strength = strength\_model.max\_strength
93. max\_sprint\_cd = normal\_sprint\_cd - GlobalValue.player\_sprint\_cd\_level \* .4
94. jump\_speed = max\_jump\_speed + GlobalValue.player\_jump\_height\_level \* 20
95. strength\_model.strength\_recover\_speed = .8 + GlobalValue.player\_strength\_recover\_level \* .1
96. health\_expend\_speed = .4 - GlobalValue.player\_health\_expend\_level \* .04
97. strength\_expend\_speed = 1 - GlobalValue.player\_strength\_expend\_level \* .1
98. adsop\_down\_speed = Vector2(0., 1.6 - GlobalValue.player\_adsorp\_down\_level \* .14)
99. # TODO\_FUC 玩家：CD 更改信号方法
100. func \_on\_audio\_manager\_cd\_update(cd : Array[CD]) -> void:
101. buffs = []
102. for i in cd.size():
103. if cd[i] == null: continue
104. var \_cd : CD = cd[i]
105. buffs.append(\_cd.cd\_buff)
106. for i in buffs:
107. i.buff\_effect()
108. # TODO\_FUC 玩家：Buff更新信号方法
109. func \_on\_global\_value\_buff\_update() -> void:
110. health\_model.max\_health = max\_health
111. health\_model.max\_health \*= (1 + (GlobalValue.health\_buff\_level \* GlobalValue.HEALTH\_BUFF\_RADIO))
112. strength\_model.max\_strength = max\_strength
113. strength\_model.max\_strength \*= (1 + (GlobalValue.abundant\_buff\_level \* GlobalValue.ABUNDANT\_BUFF\_RADIO))
114. #endregion

我们有两种数据交换的方式：信号链接与自定义函数调用

**（1）信号链接**

1. health\_model.health\_update.connect(\_on\_health\_model\_health\_update)
2. health\_model.is\_dead.connect(\_on\_health\_model\_is\_dead)

如上，connect( )函数是Godot引擎提供的便捷管道链接方式，可以快速地将信号进行链接，实现数据在不同模块间的传输与交换。对于Model类、Global类、GlobalValue类均可以采用这种方式进行链接。

**（2）自定义函数调用**

参考上述代码中的信号链接方法。在Godot引擎自身提供的链接方法不足以支撑功能实现时，我们可以自定义信号链接方法，以此实现数据在不同模块间的传输与交换。

综合上述两种方法，便可以完成模拟接口类的定义与实现。

## 4.3 软件测试与测试计划

### 4.3.1 游戏软件测试方法

游戏软件的测试与传统软件的测试方法类似，但由于游戏的特性（，游戏测试往往更加复杂和多样。以下是几种常用的游戏软件测试方法：

**（1）功能性测试**

功能性测试主要验证游戏的各个功能模块是否按预期工作，例如角色控制、战斗系统、任务系统等。开发者需要确保游戏中的每个功能模块都能正常运行，并且不会出现错误或崩溃。与传统软件测试一样，功能测试应覆盖游戏的各个方面，包括游戏开始、暂停、结束、保存等。

**（2）性能测试**

游戏的性能测试至关重要，直接关系到玩家的体验。性能测试包括帧率测试、加载时间测试、内存消耗测试等。测试人员需要检查游戏在不同硬件配置下的表现，确保即使在高负载情况下，游戏仍能保持流畅运行，避免出现卡顿、掉帧等问题。

**（3）UI测试**

游戏的UI测试和传统软件的UI测试相似，主要关注游戏界面的可用性、可访问性、视觉设计等。UI测试需要确保菜单、按钮、图标等元素的布局和互动逻辑符合设计要求，并能在各种设备和分辨率下正常显示。

**（4）玩法测试**

玩法测试是指测试游戏的核心机制和玩法是否有趣、平衡。游戏的核心玩法设计（如战斗系统、任务系统、经济系统等）是否有足够的吸引力是游戏成功的关键。

**（5）冒烟测试**

冒烟测试是快速验证游戏中最基础的功能是否可用，确保游戏的基本运行不出错。测试人员通常会在游戏的初步版本中进行冒烟测试，检查是否能够启动游戏、进入主菜单、加载关卡等基本操作。

### 4.3.2 测试计划

本项目采用前述5种测试方法进行测试。其中：

·冒烟测试、UI测试、功能性测试由开发者（即本人）在开发过程中完成。

·性能测试、玩法测试由三次内测玩家反馈完成。

## 4.4 结果分析

### 4.4.3 开发者测试（冒烟测试、UI测试、功能性测试）分析

（1）UI测试&&功能性测试

各模块功能运行无误，各UI功能正常无误。但在长时间运行下，UI会有卡顿现象。

·原因：GDScript语言中Tween函数本身性质问题，与PHP运行编译慢是一样的道理。

·解决方案：已解决。减少Tween的使用即可。采用其它方式实现动画效果。

（2）冒烟测试

以下是冒烟测试部分结果图。



图 4-1 玩家、机关、Map正常

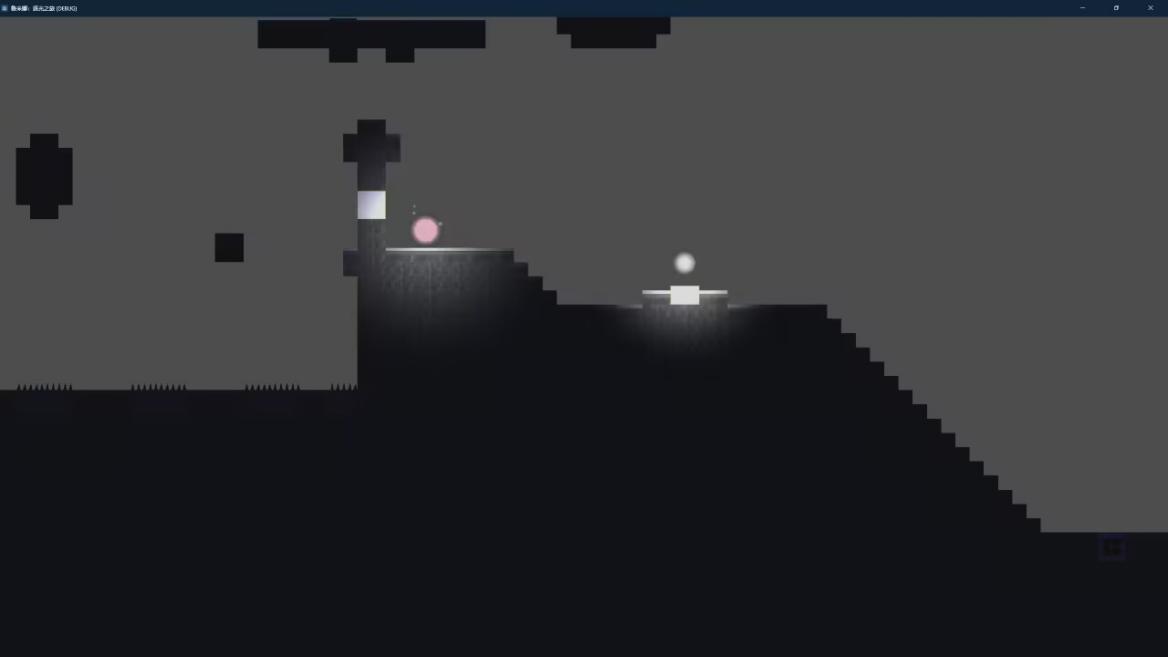


图 4-2 一级灯效正常



图4-3 背景轮换后运行正常

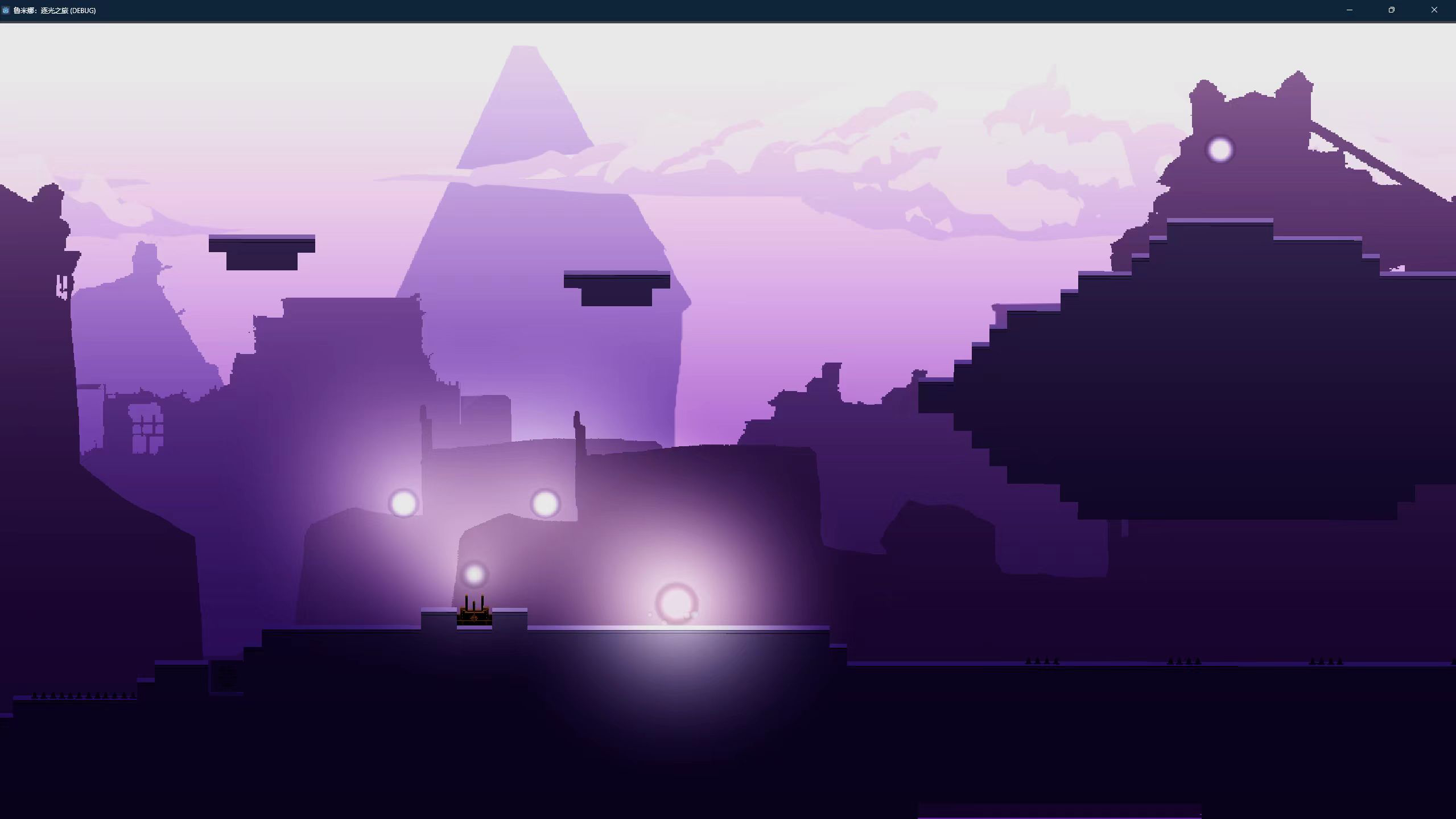


图 4-5 World Shader挂载后运行正常



图 4-6 UI&&UI Shader运行正常

### 4.4.2 玩家测试（性能测试、玩法测试）分析

表 1 玩家测试结果

|  |  |  |
| --- | --- | --- |
| **测试目标** | **问题反馈** | **是否修复** |
| **玩家手感&&机关合理性** |  |  |
|  | 漂浮距离过长、二段跳衔接卡顿 | Check. |
|  | 陷阱代价过小 | Check. |
| **不同设备上UI&&Shader运行测试** |  |  |
|  | 地图有时会帧数骤降，考虑是背景轮换问题 | Check. |
|  | UI有卡顿问题，考虑Tween的原因 | Check. |
| **Demo版本完整测试** |  |  |
|  | 切地图时会有帧数骤降，考虑Tile瓦片原因 | UNDONE |
|  | 关卡3Boss战设计难度过高 | Check. |
|  | Shader在Camera拉远时有消失的情况 | Check. |
|  | 引导不行 | UNDONE |

根据玩家反馈，可确定下一步正式版本的重构计划。会在第5章详细阐述。

# **5** 总结

## 5.1 用户反馈

### 5.1.1 用户反馈

根据玩家评测，目前Demo主要有以下几个问题：

·游戏引导不够明细。

·难度梯度上升太快。

·玩法有待加强。

·对硬件配置较差的机型负载较大。

### 5.2.2 后续改进方向



图 5-1 重构MindStorm

正式版本重构，除却完成先前订下的目标玩法实现之外，应当注意以下几个方面：

**（1）技术方面：**

·场景的LOD加载：期望解决机型负载问题。

·玩家状态机重写：对手感应当进一步改进。将有限状态机改为分层+平行有限状态机。

·玩家黑板：优化存储玩家状态数据，虽然仍感觉有屎山代码的可能性。

**（2）玩法方面**

·优化引导：至关重要。

·增加机制：比如色彩BUFF，相机相关操作。

## 5.2 全文总结

在本项目软件开发中，我承担了策划、程序、文案和音效等多项职责，制作了一款横版2D跳跃游戏Demo。

作为策划，我设计了游戏的基本玩法和关卡结构，确保游戏的挑战性与可玩性相平衡。

在程序方面，我负责了游戏的核心逻辑开发，包括角色控制、碰撞检测、物理引擎的实现和关卡切换等。并优化了游戏的性能，使其在不同设备上都能正常运行。

在文案方面，我编写了游戏中的基础剧情和角色设定，注重游戏故事的连贯性与趣味性，以吸引玩家的参与，给予玩家感悟与激励。

在音效方面，我自行制作并添加了背景音乐、跳跃音效、碰撞音效等，增强了游戏的沉浸感和互动性。

# **6** 体会

## 6.1 技术层面

本次游戏开发不是我第一次制作游戏，但是是我第一次使用Godot引擎进行游戏开发。在开发过程中，我深刻地感受到其与Unity引擎之间的差异性与共通性。

与Unity相比，Godot采用了非常模块化的设计。它的核心引擎（Godot Engine）与各类模块（如2D、3D、物理、音频等）之间分离，开发者可以根据需求启用和禁用不同模块，这使得它在资源使用上更加灵活和高效。

Godot的场景系统使用树形结构，几乎一切都是场景，这使得场景的重用和层次关系相比起Unity引擎，显得非常直观。Godot采用场景树结构，每个游戏对象都是场景的节点，开发者通过树形结构管理场景和资源。Godot的资源加载非常灵活，支持同步和异步加载，也支持热加载，适用于实时修改和调试。

但是Godot的渲染管线比较简单，采用基于节点的渲染架构。适合2D项目。而至于在3D项目上，我个人观点：Godot目前完全不具备任何制作3D游戏的能力，如果有人做出来了，那是个人能力。

## 6.2 软件开发层面

虽然我之前进行过很多白盒游戏、游戏和程序的开发，但之前我并没有系统地学习过专业的软件开发知识，对整个软件开发流程了解有限。经过这次的项目实践，我终于全面了解了软件开发的全流程。从需求分析、设计、编码到测试、部署，每一个环节都有其独特的挑战和需要注意的细节。

作为独立开发者，我不仅要负责程序开发，还要兼顾策划、文案、音效等各方面的内容，这让我体会到了开发中多任务处理的复杂性。同时，我也意识到独立开发和团队开发的本质区别。独立开发虽然能保证更大的自由度，但也容易出现资源分配不均、问题无法全面发现的状况；而团队开发则能够通过分工合作，充分利用各成员的专长，提升整体开发效率和产品质量，但同时，在原型系统确定等方面，又容易因为意见不一而拖慢进度。

总体而言，这次经验让我更全面地理解了软件开发的复杂性，也为未来的团队合作奠定了基础。