

华中科技大学

《软件工程》项目报告

题目: 基于视错觉机制的推箱子小游戏

课程名称: 软件工程
专业班级: ACM1901
组 名: WLZL
同组成员: U201914996 李 涵
U201914901 刘安珉
U201914965 卫云泽
U201914956 郑 欣
指导教师: 万胜刚
报告日期: 2021.12.31

计算机科学与技术学院

任 务 书

1 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。

2 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

1 问题定义	1
1.1 项目背景与意义	1
1.1.1 项目需求 Need	1
1.1.2 实现方法 Approach	1
1.1.3 项目益处 Benefit	1
1.1.4 竞争对手 Competitors	2
1.1.5 推广途径 Delivery	2
1.2 项目基本目标	3
1.3 可行性分析	3
1.4 人员管理和项目进度管理	4
1.4.1 人员管理	4
1.4.2 项目进度管理	5
2 需求分析	7
2.1 需求分析概述	7
2.2 UML 相关需求分析图	7
2.2.1 E-R 图	7
2.2.2 状态转换图	8
2.2.3 用例图	8
2.3 原型系统设计	8
2.3.1 系统阐述	9
2.3.2 游戏核心	10
2.3.3 关卡选择	11
2.3.4 关卡设计	11
3 概要设计和详细设计	12
3.1 系统结构	12

3.1.1 游戏主体	12
3.1.2 关卡编辑器	12
3.2 场景管理	14
3.3 关卡管理	16
3.3.1 物体	16
3.3.2 视图	18
3.3.3 地图	20
3.4 场景物体控制	22
3.4.1 摄影机控制	22
3.4.2 动画控制	24
3.5 关卡加载与保存	25
3.6 关卡编辑器	25
3.6.1 物块创建与删除	25
3.6.2 鼠标点击事件状态机	26
3.6.3 UI	26
4 实现与测试	27
4.1 实现环境与代码管理	27
4.2 关键函数说明	28
4.2.1 状态机模板	28
4.2.2 场景管理	29
4.2.3 关卡管理	29
4.2.4 场景物体控制	31
4.2.5 关卡编辑器	32
4.3 测试计划、测试用例与结果分析	34
4.3.1 常用测试方法说明	34
4.3.2 视错觉逻辑测试	35
4.3.3 移动逻辑测试	37

4.3.4 状态机逻辑测试	39
4.3.5 关卡编辑器测试	39
5 前端实现与测试	41
5.1 网站架构	41
5.2 各网页的说明和特性	41
5.2.1 主页	41
5.2.2 归档	44
5.2.3 关于	44
5.2.4 落地页	45
5.2.5 其他	45
5.3 前端系统的测试	45
6 用户反馈	47
6.1 内测用户反馈情况总结	47
6.2 改进措施	48
7 总结与体会	49
7.1 李涵	49
7.2 刘安珉	50
7.3 卫云泽	52
7.4 郑欣	53

1 问题定义

1.1 项目背景与意义

1.1.1 项目需求 Need

在大学本科生群体中存在一个有趣的现象：每到期末考试月，同学们的社交平台中便流传着一些名为“绝杀考试救命资料”的网址。它们实际上是一些益智放松的小型网页游戏，如模拟投篮、消砖块等。由于他们规则简单、趣味性强、没有分数上限，总能使很大一部分同学在复习的间隙“上头”其中。显然，在当下紧张的大学生活背景下，尤其是期末考试月，同学们紧张的课业工作与休闲娱乐的需求之间存在矛盾，需要一种有效的娱乐途径。

然而，繁重的课业期间过长时间的“摸鱼”必然会导致事后的懊悔。原因在于如果游戏带来的仅是放松，则会让追求效率的我们感到浪费了时间。所以，我们对娱乐的需求不止于“有效”，而应是“高效”，即娱乐活动应对我们的思维拓展有正向作用，而非像短视频一样仅仅带来被动的享受。另外，当下的大学生整体处在一个“内卷”的环境中。为了取得更高的成绩、最大程度降低“错误率”，我们总是倾向于学习和展示既有的知识和成果，这种现象严重阻碍了创造性思维的发展。在放松期间能让思维驰骋，脱离思维定势，成为当下越来越大的需求。从更大范围来看，不仅大学生，庞大的工作群体也有相同的“高效”娱乐需要。

1.1.2 实现方法 Approach

游戏机制上，将《纪念碑谷》中的视错觉机制与推箱子游戏的机制融合，赋予推箱子游戏创新的玩法。

开发途径上，基于 Unity 平台，结合 Visual Studio 集成环境使用 C# 语言进行游戏软件的开发与 UI 的设计；使用 Visual Studio Code 的 drawio 插件及其他工具完成贴图和界面的绘制；使用 Hexo 完成游戏官网的搭建、配置、个性化，以及 SEO 优化的实现。

1.1.3 项目益处 Benefit

首先，Parabox 小游戏结合了推箱子的益智性和纪念碑谷的思维发散性，在通过闯关机制带来娱乐性的同时，也通过激发空间想象力帮助突破定式思维，从而培养创造力。进一步地，玩家可以通过关卡设计模式自定义自己的游戏场景，并通过相互交流，

使有趣的创意碰撞出更绚丽的火花。

另外，多人合作机制鼓励玩家协作解决问题，有利于增进人际关系，加强合作意识与协同能力。

1.1.4 竞争对手 Competitors

各种益智类小游戏是我们的竞争者。在市场上，同样采用视错觉机制的游戏包括《纪念碑谷》、《无尽长廊》等。《纪念碑谷》(Monument Valley) 是 Ustwo Games 出品的一款移动端游戏，它用有趣的空间错位感交织出了清新唯美的迷宫世界，玩家需要引领公主艾达在这座神奇的迷宫中寻找正确的道路，抵达终点，带来一次神奇建筑与奇妙几何体相结合的梦幻探险。游戏于 2014 年 4 月上线，迅速在全球 50 多个国家的 App Store 付费排行榜中排名第一，并且获得了 9.2 分的高分评价。其特点为，用了大量复杂的地形来力图达到多样化的体验设计，剧情前后相关性较强，游戏角色多样化。满足的是一些愿意专门花时间投入该游戏，并愿意沉入其中以获得对艺术和视觉体验极致的享受，游戏过程短但沉浸、情感体验充分，延伸思考性强。另外，还有 1.1.1 中提到的网页小游戏如《篮球》，玩家通过点击屏幕将篮球抛向筐筐，进球则获得相应分数，得分无上限。

相比竞争对手，我们的优势可以概括为两点。一方面，我们的游戏不采用无尽性，减弱了连续性，增强了扩展性。这样就便于我们利用碎片化时间，既能享受到视错觉带来的惊艳，又不至于沉迷其中，影响工作和学习；另一方面，我们的游戏支持自定义关卡，具有游戏社区，这就比单纯的有限关卡游戏包含更丰富的内容的更大的潜力。利用人们使用自己的智慧创造题目的荣誉感，使用户发掘自己的潜能。并通过社区交流，充分调用集体智慧。

另外，《纪念碑谷》作为视错觉游戏的代表作，只有 iOS 版本。本游戏首先基于桌面端开发，而后向安卓、iOS 端扩展，将提供更丰富的平台支持。

1.1.5 推广途径 Delivery

推广渠道

1. 利用 Gitee pages 服务发布游戏官网，提供游戏的下载渠道和游戏攻略。并在后续更新中开放游戏社区平台，供玩家分享自制地图和交流攻略；
2. 通过 QQ、微信等社交软件进行推广，如微信朋友圈；
3. 将游戏进行更加完善的封装后，上线游戏平台，如 Steam 等；

4. 通过互联网资讯平台、音频软件推荐、车站、电影院、商店等渠道发布广告，进行更大范围的商业推广。

促销策略 保持关卡更新的频率。每隔一段时间发布一定数量的关卡并上线首充优惠等新活动。

价值主张 产品的核心价值主张是“为休闲插上想象的翅膀”，卖点在于高频低时长的机制、高度 DIY 的特性和社交与互动的属性。

1.2 项目基本目标

结合《纪念碑谷》的视错觉机制和推箱子的游戏机制，开发出一款桌面端的创新型推箱子小游戏。游戏的主要有以下 3 种游玩模式：

1. 单人游戏模式。游戏中会设置一些单人关卡，供单人进行闯关式游戏。在这些关卡中玩家控制唯一的玩家方块，利用视错觉寻找合适视角，将若干个箱子推至目的地，最后自己移动到目标位置，即视为通关；
2. 多人游戏模式。在该模式中有多个玩家方块，每个方块由一个玩家控制。玩家之间需要相互合作，协调好顺序和策略，以在配合之下将箱子推至目的地；
3. 关卡编辑模式。玩家可以自由地放置箱子方块、障碍物方块和玩家方块并指定目标位置，来构建自己的视错觉推箱子关卡。

在项目过程中，我们基于自己的分工使用 1.1.2 中所述的平台进行开发，并利用 Gitee 进行代码和任务进度的管理，以保证项目目标的实现。

1.3 可行性分析

从技术的角度来看，本游戏的两个主体机制——视错觉和推箱子，在市场上都分别有成熟的游戏成品。其中，视错觉机制的实现主要是利用 3D 和 2D 视角的结合。本身在 3D 视角下并不连通的两个地面方块，在某 2D 视角中就可能连接在一起，玩家和箱子从而就能在此视角下从一个地面方块移动到另一个。利用多个 2D 视角，玩家就能完成在 3D 中看似不可能的移动。在开发途径上，Unity 提供了一套完整的软件开发方案，方便我们在较短的时间内开发出界面美观的游戏。游戏的实现主要通过 C# 语言完成。作为一种成熟的开发语言，C# 在网络上有完善的语法支持，以及许多与游戏开发相关

的特性解读。我们可以不必从零开始实现动画细节，而只需关注动画逻辑和游戏机制本身。

确认可实现性后，使用 COCOMO 模型对工作量进行定量评估。记以人月（PM）为单位的工作量为 E ，开发时间为 D 月，项目的代码行数估计值为 L （千行）。对于该项目的组织型软件，有模型公式如下：

$$\begin{cases} E = 2.4 \times L^{1.05} \\ D = 2.5 \times E^{0.38} \end{cases} \quad (1.1)$$

经小组的讨论与综合评估，取 $L = 5$ ，解得 $E \approx 13.01$, $D \approx 6.63$ 。所以具体参与软件开发的人数 $N = E/D \approx 2$ 。值得注意的是，上面计算出的 E 和 D 只作为中间结果，要对工作量进行更准确的评估需要采用中间 COCOMO 模型，加入 EAF 调节因子（使模型考虑到我们项目的特殊性和简易性）。基于这是一个课程项目而非商业项目的事 实，对各个调节因子取值得到 $EAF = \prod_{i=1}^{15} F_i \approx 0.26$ 。这样可得 $E' = E \times EAF \approx 1.95$ ，即需要接近两个月的开发时间，可以在软件工程结课验收前完成。所以，该项目具有很高的技术可行性。

从市场的角度来看，正如需求分析中所述，当今市面上对高质量娱乐的需求正在逐渐增大，而同时兼具娱乐性和创造激发性的游戏却屈指可数，故本游戏有着很大的市场空间。

从经济的角度来看，小组成员使用各自的计算机设备即可完成主要的开发工作；利用 Gitee 线上平台和学校的公共环境，团队可以方便地以线上线下结合的方式开展交流讨论以协同工作进度。所以项目的初步开发不存在大额的成本，具备经济上的可行性。

1.4 人员管理和项目进度管理

1.4.1 人员管理

基于项目团队的成员情况，采用开放与同步相结合的组织范型。在项目的分工上按照问题自然划分，不设置专门的组长进行任务分配；由于团队人数少，不存在积极性问题，在项目的进行上可以充分发挥各自的创造性，过程中采用民主的形式讨论交流。4位小组成员的具体分工如下：

李涵 负责项目全过程的文档整理，参与软件的测试。包括立项和需求分析阶段 Wiki 文档的撰写及示意图的绘制、项目执行阶段成员的进度统计和燃尽图的绘制。撰写了本报告的问题定义、需求分析（1、2 章）和测试方法部分。

刘安珉 参与项目的需求分析和软件测试，负责项目的推广。包括制作项目官方网站以提供下载渠道和游戏教程，以及游戏 Logo 设计。撰写了本报告网站的设计实现与测试部分（第 5 章）。

卫云泽 负责具体的任务安排（如甘特图中的任务分配）和游戏关卡设计器部分的实现与测试，并参与关卡的设计以及用户反馈的收集与整理。撰写了本报告 3、4 章的关卡设计器部分以及用户反馈。

郑欣 负责软件开发任务的梳理、游戏本体核心交互和 UI 机制的设计、实现与测试，并参与关卡的设计。撰写了本报告除关卡设计器外游戏主体的设计实现与测试部分（3、4 章）。

1.4.2 项目进度管理

总体上，通过甘特图完成项目初期具体任务的分派与规划；通过燃尽图在项目的全过程中对进度进行跟进和分析。

项目整体分为立项与设计、游戏主体开发、关卡设计器开发、关卡设计和测试发行 5 个部分。经过讨论和规划，将这 5 个主要阶段进一步细化成了一个个单独可执行的任务并进行了分派。具体分派情况见项目 Wiki 中的甘特图，其整体缩略图如图 1.1 所示。



小组各成员按照甘特图中所分派的具体任务完成项目，过程中定期进行进度统计，并通过燃尽图来判断当前进度是超前还是落后于预期，实时地调整执行计划。例如，在项目过程中的 11 月 26 日，小组的燃尽图如图 1.2 所示。可以看到当前总体上正按照进度计划正常进行，实际进度略微落后于计划进度，应在后几天之内投入时间，完成这一阶段各自的任务。

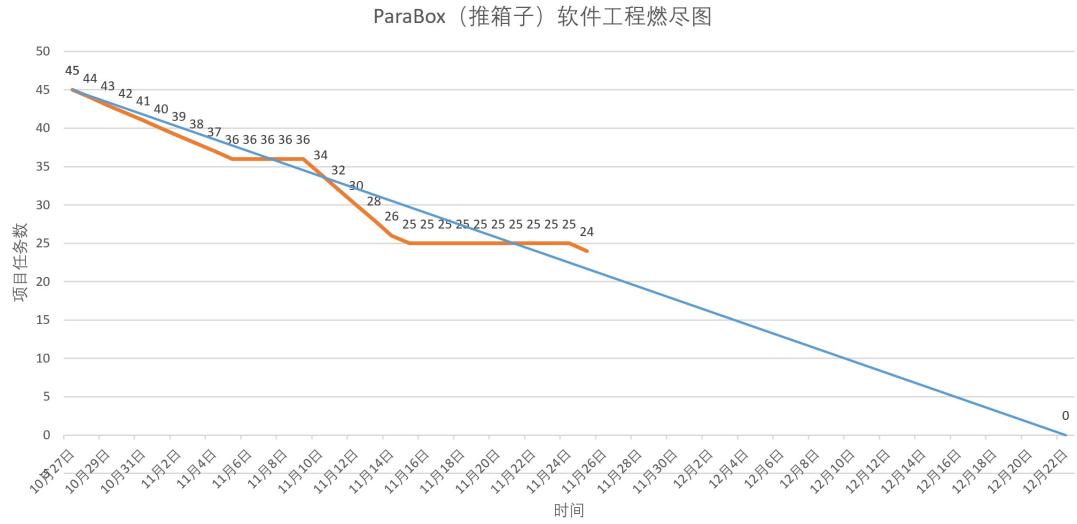


图 1.2: 11 月 26 日的燃尽图

2 需求分析

2.1 需求分析概述

需求分析过程主要分为需求设计、需求跟踪和需求变更三个过程。

进行需求设计时，首先有必要对于所获取的需求进行自然语言的描述，然后进一步地基于 UML 图形机制进行用例分析。在快节奏的生活中，人们往往疲于应付学业和工作上的任务，按部就班地完成分配任务而无自己的想法与创造性，越来越多的人倾向于让短视频侵占他们的休闲时间，无数个被动接受的场景使得人们需要一种能激发创造力，兼顾有趣和休闲娱乐的小游戏，让他们可以亲眼看见并让自己的想象力可视化。与此同时，中小学生在学习过程中也时常需要空间想象力和创新能力，此时一款更受年轻人群体喜爱的游戏形式可以同时满足孩子们的需要和兴趣。随着生活方式和生活水平的提高，不同年龄段对于游戏的需求也不一致，在需求的层面上有分散和片面性；但是对于游戏而言，更多地是关注游戏带来精神上的放松、刺激和满足感。所以，游戏应当具备基本的单人闯关模块，来增加游戏的趣味性；具有多人合作关卡来满足小团体的娱乐需求；能够自定义关卡来为想象力提供变成现实的途径。

对于我们所进行的课程项目而言，需求变更主要来自于开发过程中的反馈。在需求跟踪过程中持续获取外部意见，结合小组成员的权衡后形成游戏机制的修改意见。支持需求变更的系统应能追踪的重要依赖关系，所以在需求和系统的设计中要建立相关项之间的可追踪性。我们小组在进行进度规划、绘制甘特图时，梳理了各个任务之间的相互依赖关系，并将重要的前后关系在甘特图中用箭头标出。当发生需求变更时，可以依照关系链对需要做出修改的模块进行梳理。

2.2 UML 相关需求分析图

2.2.1 E-R 图

在游玩关卡时，对游戏整体进行抽象，得到 E-R 图如图 2.1 所示。

游戏的每一关由一张地图组成。地图中包含玩家、箱子、障碍物和机关。在单人模式中有一个玩家实体；在合作模式中有多个。玩家、箱子和机关都是相同形状的正方体，用不同颜色进行区分。障碍物不能移动，玩家可操作玩家方块推动箱子，通过 2D/3D 的转换以及从不同视角观察地图，找到通向目标的方法。机关的类型是指该机关时箱子的目标还是玩家方块的出口。当玩家将所有箱子推到目标，并自己移动到出口时，其转换为通关状态并进入选关界面。

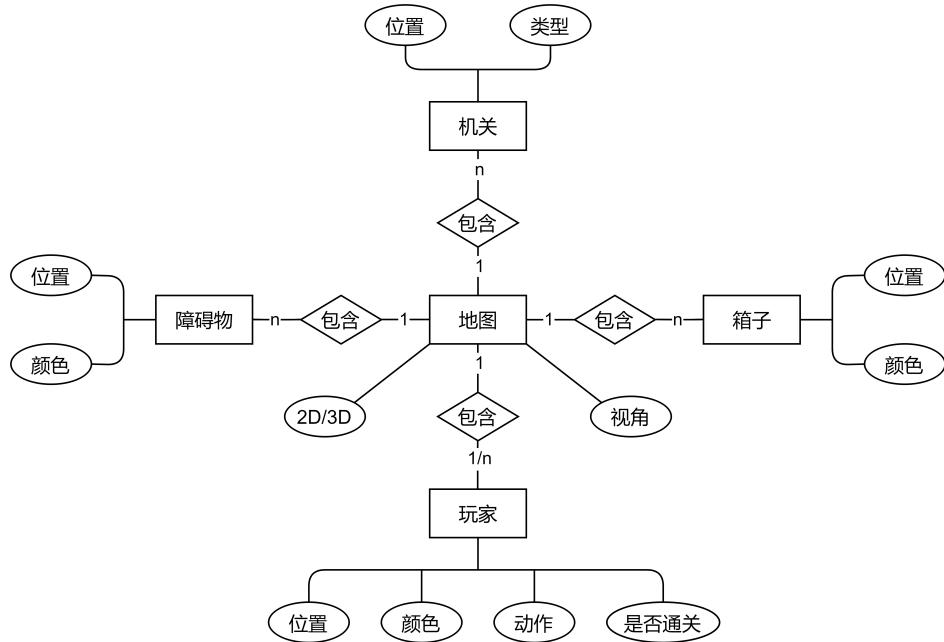


图 2.1: Parabox 游戏关卡 E-R 图

2.2.2 状态转换图

图 2.2 展示了游戏的状态转换流程：游戏整体包括两种模式——关卡编辑模式和游玩模式。游玩模式又分为单人模式和合作模式。游戏启动后，由原始状态转换到游戏启动状态进行模式的选择。选择模式后进入游戏状态，当满足通关条件后回到关卡选择状态以供玩家继续选关。玩家也可以在任意状态按下 Esc 键返回上一状态（在游戏启动状态返回便会退出游戏）。

2.2.3 用例图

明确了游戏的对象和流程后，通过用例图对用户的需求和在游戏中的操作进行分析，如图 2.3 所示。在游戏模式中玩家可以移动玩家方块和进行视角旋转（包括 3D/2D 转换）；在关卡编辑模式中，玩家可以放置障碍块、箱子以及玩家方块（主体）并设置包含目标位置和出口的机关。

2.3 原型系统设计

游戏的用例和基本流程明确后，我们设计了一个小型的原型系统，其主体是一个实验性的关卡，具备上面 UML 模型中游戏模式的主要元素。游戏核心模块 Demo 见代码仓库 /Demo 文件夹。

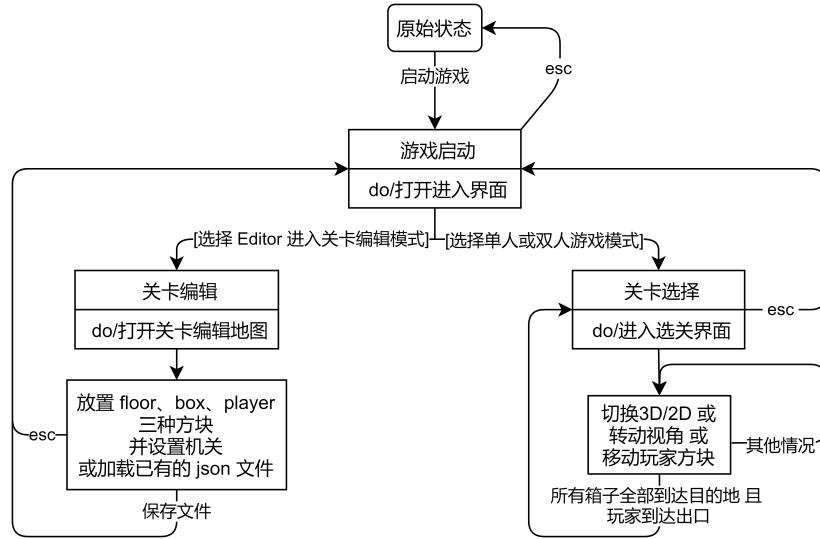


图 2.2: Parabox 状态转换图

2.3.1 系统阐述

针对需求，我们计划开发一款益智类游戏，该游戏利用视错觉，可以充分锻炼玩家的空间想象力和创造力。由于不论是手机还是电脑本质上还是一个二维平面，因此对于一个 3D 游戏，可以通过调整不同的视角，使得游戏地图会发生一些变化，用户所看见的就是实际上地图所表示的。

从游戏本身的逻辑上，我们可以看出该游戏是能充分锻炼玩家的思维和空间能力的，用户在游戏中是寻找满足感的，但也要给一定的难度，挫败用户的好胜心理，不同的地形，不同的任务，给了游戏难度的设置很大的自由度。

从情节策划方面，游戏的目的是为了消遣时间和进行游戏交流，应该加强对于用户与用户之间交流；达到友好的互助；我们可以设置即时聊天频道以及双人合作游戏。游戏的目的是寻找满足感，应该为用户设置特殊身份，特殊标志满足幻想，每通过一关会奖励一个勋章碎片，达到一定数量后可以点亮勋章，给用户设置特殊身份。并且此游戏承载一定的故事情节比如说用户是兔子，在一过关中寻找丢失的大萝卜，这些故事都是大众熟知的，让用户有身历其境的感觉。

在玩法选择方面，简单、有趣、团队合作、容易获得满足感，可以有效的支配游戏时间并能在游戏间获得快乐；因此，该产品将会设置关卡自定义模式，将地图设计交给用户，并且开放社区，鼓励用户分享关卡，让更多的人，享受设计的乐趣。

在艺术价值方面，该游戏一定会满足用户的体验感，游戏画面、技能、特效、任务；挖掘游戏的艺术价值，将游戏的场面设计的简洁大方优雅端庄；还可以在游戏环境增加曝光量，让用户获得更多的声望、装备、金钱、经验、道具等。

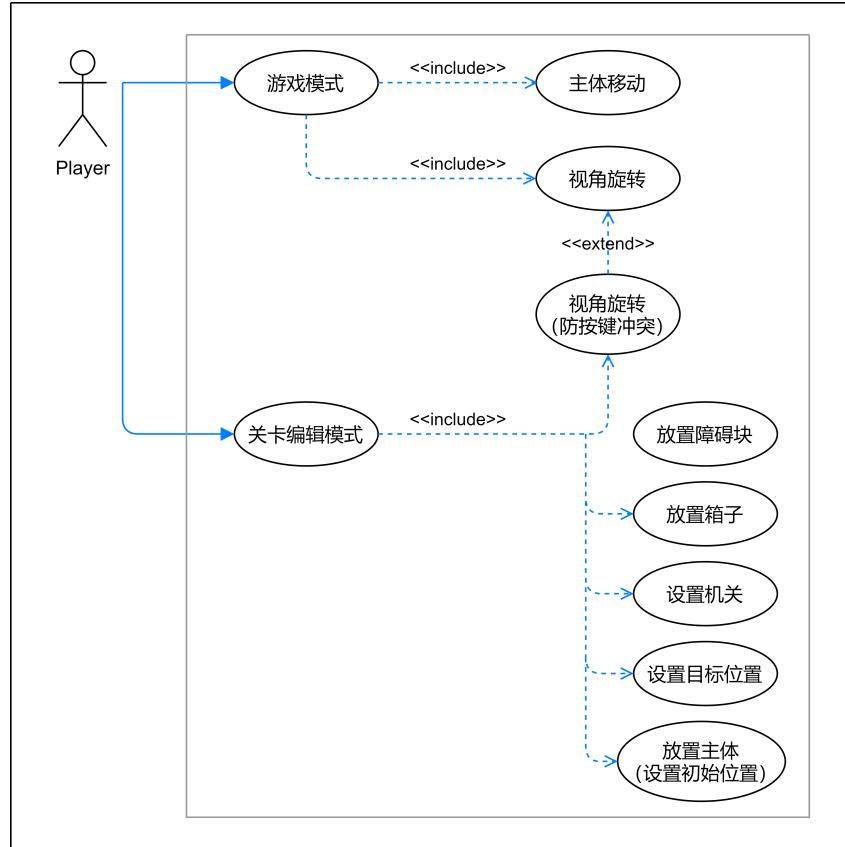


图 2.3: Parabox 游戏模式用例图

2.3.2 游戏核心

推箱子游戏界面 Demo 如图 2.4 所示，游戏玩家使用键盘或鼠标进行游戏，移动小球推动箱子到达目的地，方可过关。通过对游戏视角的变换，由视错觉原理会产生新的“可通过路面”，玩家需要开动脑筋，在不同视角下的可通过路面上移动小球，推动箱子到达看似不可达的目的地。

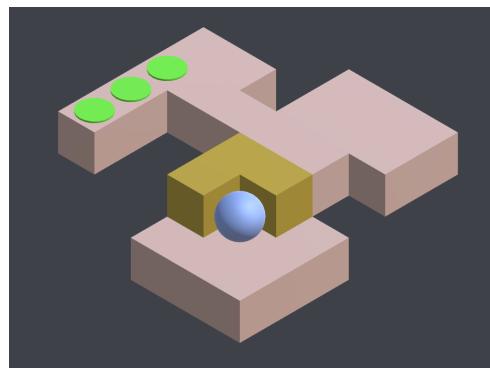


图 2.4: 游戏界面演示图

2.3.3 关卡选择

游戏的关卡选择界面也创新性地采用了游戏的视错觉机制，如图 2.5 所示。游戏主界面为第 0 关，选择任何一个箱子推到对应绿点之上即为开启对应关卡，进入关卡时，主界面地图逐渐下沉，相对应关卡的地图逐渐浮现。

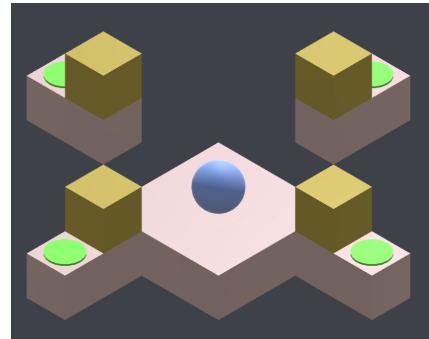


图 2.5: 关卡选择界面

2.3.4 关卡设计

游戏支持玩家自定义关卡，因此我们会为游戏配备关卡设计器，使用拖拽元素形成地图的方式实现关卡的设计与构建。如图 2.6 所示，玩家可在左侧元素选择栏中选择需要的元素，拖拽到右侧的虚线处，通过元素与虚线窗格的吸附实现关卡设计。设计过程中同样可转换视角进行视错觉的构思与构建。

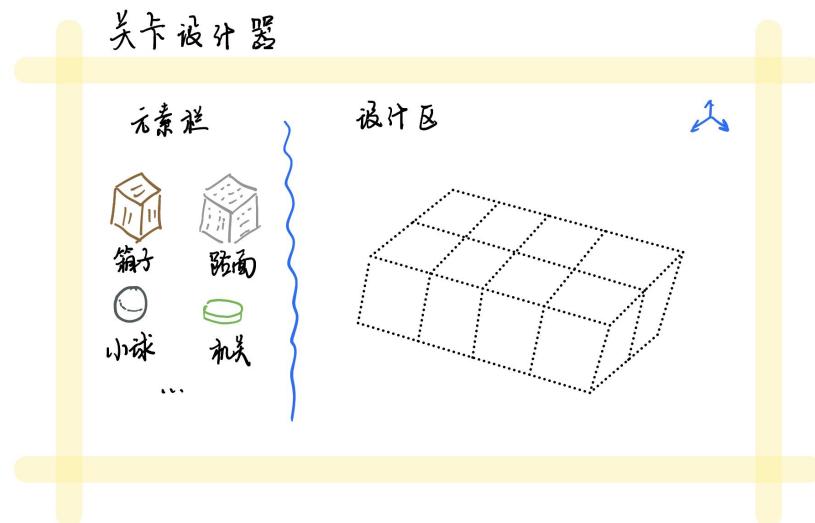


图 2.6: 关卡设计界面示意图

3 概要设计和详细设计

3.1 系统结构

游戏分为游戏主体和关卡编辑器两个部分，下面分别介绍这两个部分的主要结构。

3.1.1 游戏主体

游戏主体需要给玩家提供关卡选择和关卡游玩的功能，主要分为以下几个部分：

1. 场景管理器：是一个有限状态机，用于控制关卡、玩家和摄影机的状态，例如玩家的移动、摄影机的旋转与关卡的切换；
2. 地图：场景管理器中用于储存当前关卡地图的状态，提供给场景管理器移动玩家与视角切换的接口；
3. 物体：分为实体和触发器两大类，其中实体可分为箱子、玩家和路面，触发器可分为效果和目标。一个关卡由若干物体所组成；
4. 视角：分为透视图和正交视图两大类，正交视图又可以分为顶点视图、边视图和俯视图三类。用于根据实体信息计算得到边的信息，地图中玩家与箱子仅可在边上进行移动；
5. 摄影机控制：用于将场景管理器中当前视图展示在游戏界面，并为关卡编辑器提供接口；
6. 动画控制：控制玩家或箱子移动和高亮（当踩在目标点上）的动画；
7. 关卡加载与保存：从 json 文件中读取关卡信息，并为关卡编辑器提供接口。

系统整体结构如图 3.1。

3.1.2 关卡编辑器

关卡编辑器是供开发者或资深游戏玩家可视化创建或编辑关卡的游戏组件，流程设计如下：

1. 射线检测：基于 Unity 射线检测判断鼠标所处位置；
2. 鼠标事件处理：对鼠标特定动作（如左键、长按、右键等）事件进行响应；

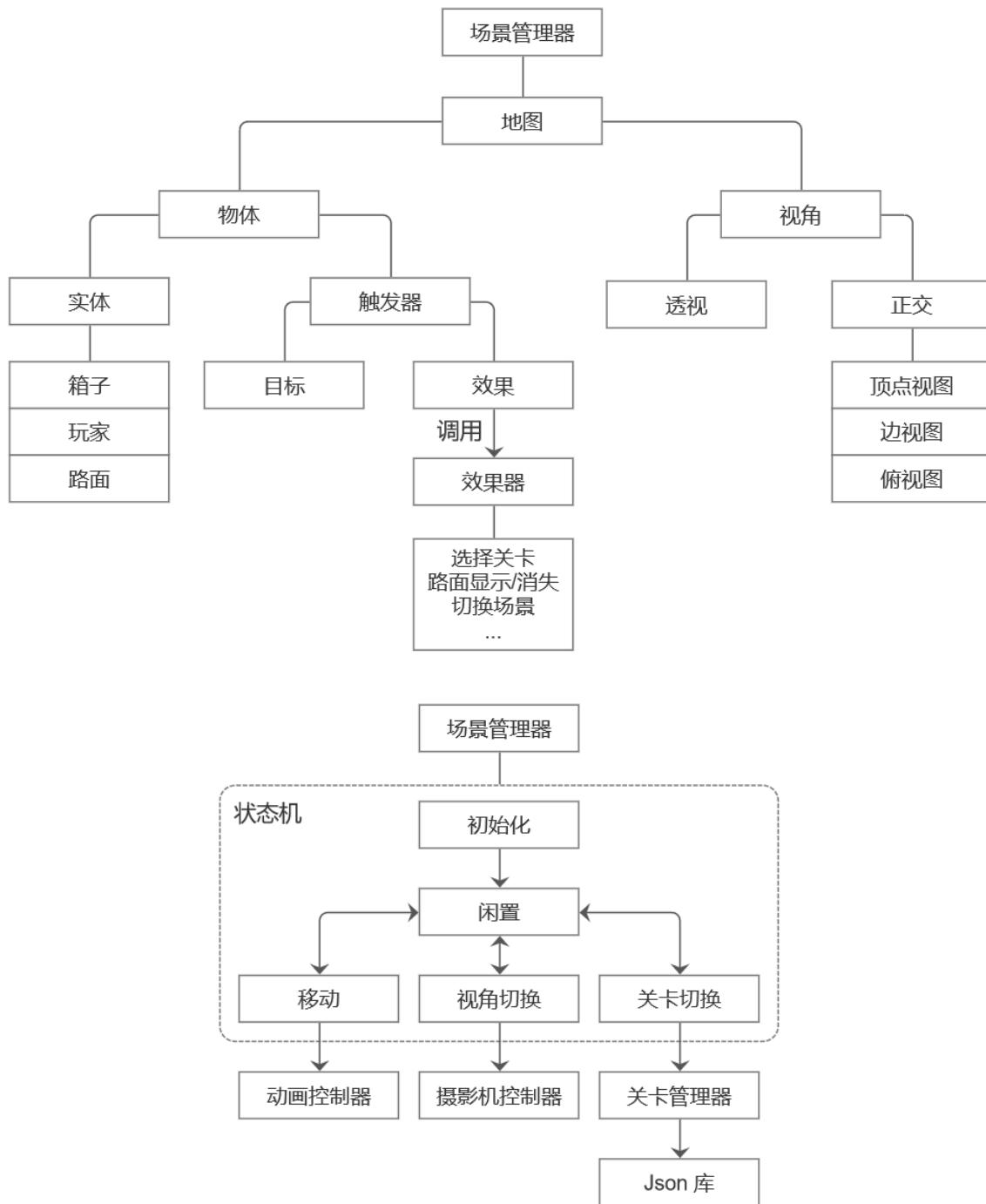


图 3.1: 系统结构图

3. 物块生成、视角转换、物块删除等功能；
4. UI：点击 UI 中相应物料图标可切换当前物块材料；关卡编辑器支持对已建成的关卡的保存和加载。

3.2 场景管理

从这一节开始自顶向下地讲解游戏主体部分的设计与原理。

场景管理器是在游戏主体部分中一直运行的一个状态机，采用**单例模式**设计。它分为 5 种状态：初始化状态、闲置状态、移动状态、视角切换状态、关卡切换状态，其 UML 类图如图 3.2。

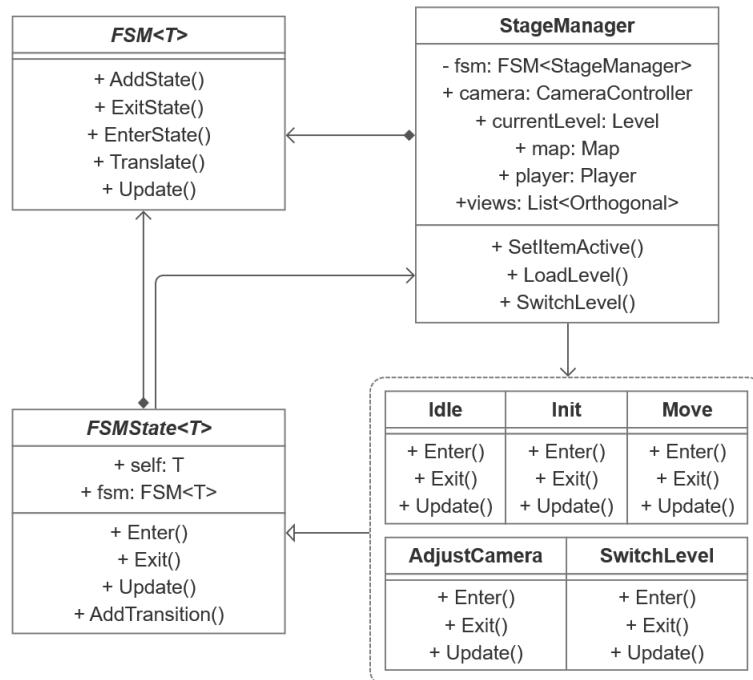


图 3.2: 场景管理器 UML 类图

下面依次介绍这些状态的功能以及转移条件。

初始化状态 (Init) 加载选关界面，初始化摄影机，并直接转移到闲置状态。

闲置状态 (Idle) 闲置状态即玩家没有移动且键盘与鼠标没有任何操作时的状态。该状态有以下几种转移：

1. 按下鼠标：转移到视角切换状态；

2. 按下方向键：转移到玩家移动状态。这里无论移动是否合法都会进行转移；
3. 按下“R”键或 Esc 键转移到关卡切换状态。若按下“R”键则重新加载当前关卡；按下 Esc 键且当前不在主界面则返回上一关，否则退出游戏。

除此之外，在该状态内按下空格可以切换玩家（如果有多个玩家），滚动鼠标滚轮可以调整摄影机与玩家之间的距离。

移动状态 (Move) 移动状态即计算玩家和箱子移动的结果并进行移动的状态，其执行流程与转移条件如下：

1. 进入移动状态时需要根据闲置状态传入的参数计算出玩家和箱子移动的结果，若无法移动则返回闲置状态；
2. 移动状态中若没有移动结束则更新移动物体动画控制器的状态。若检测到按下方向键，则将对应方向放入缓冲区；
3. 若移动结束则退出移动状态。退出移动状态时检测当前缓冲区内是否有按键，如果有则重新回到该方向的移动状态，否则进入闲置状态。退出时还需触发地图中的 Trigger，并判断是否通关，如果通关则进入关卡切换状态切换至下一关。

视角切换状态 (AdjustCamera) 视角切换状态用于鼠标点击时切换透视或正交状态，或鼠标拖动时改变摄影机方向。其执行流程与转移条件如下：

1. 当进入视角切换状态时记录摄影机原来的位置和鼠标坐标，并将摄影机控制器切换为透视状态；
2. 状态中根据鼠标移动的坐标更新摄影机的方向和位置；
3. 鼠标松开时退出状态。若鼠标按下时间较短或移动距离较短则改变透视正交状态，否则根据当前状态为正交还是透视决定是否吸附到地图中距离最近的视角。

关卡切换状态 (SwitchLevel) 关卡切换状态用于进行关卡的加载与切换，其内部是一个子状态机，分为原关卡中的物体消失和加载新关卡两个状态。其执行流程如下：

1. 子状态机进入状态 1，将原关卡中的所有物体缩小，直到消失时进入状态 2；
2. 状态 2 开始时加载新关卡，且一开始所有物体的大小都设置为 0。之后初始化摄影机并让所有物体逐渐出现，动画结束时子状态机退出状态 2，主状态机进入闲置状态。

状态机完整结构如图 3.3。

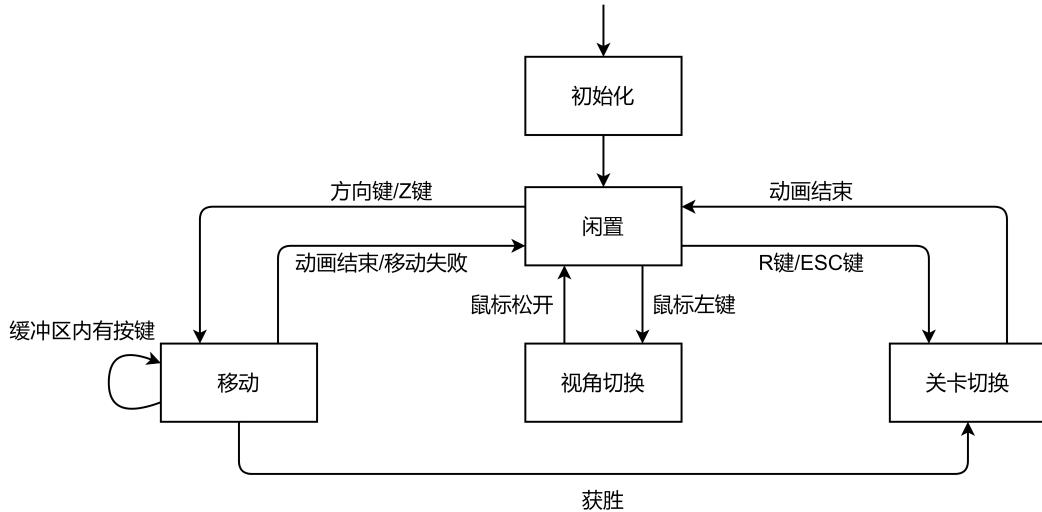


图 3.3: 场景管理器状态机

3.3 关卡管理

关卡管理器分为物体、视角和地图三个部分，是整个游戏的核心逻辑，它记录了关卡的视角信息、所有物体的位置与玩家移动信息的历史记录。

3.3.1 物体

物体类的 UML 图如图 3.4。

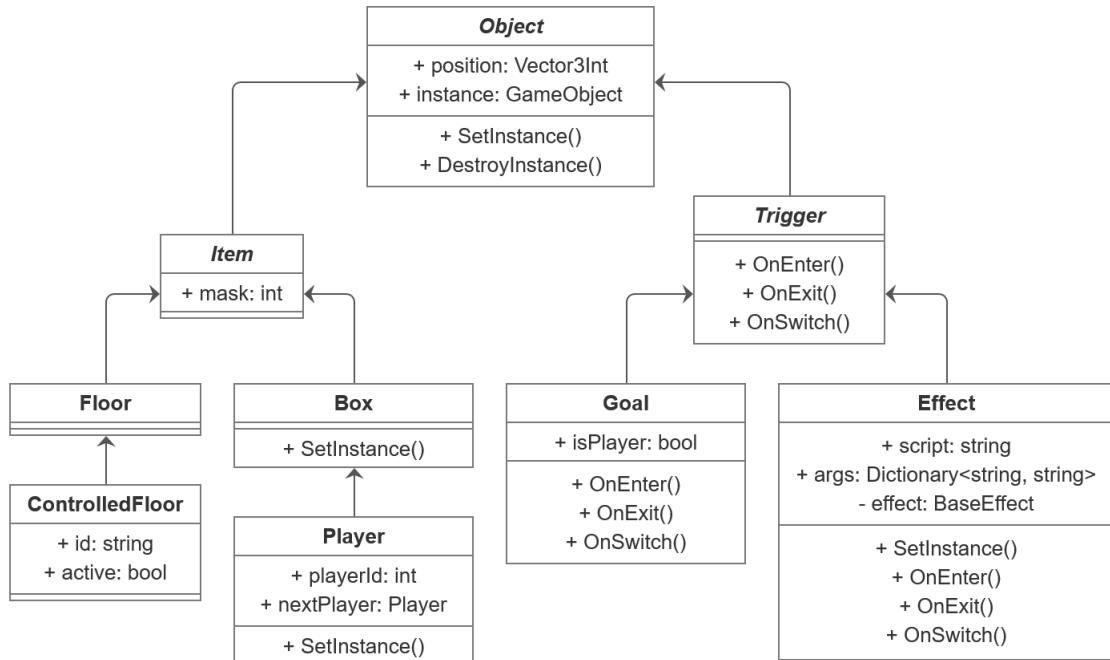


图 3.4: 物体的 UML 类图

Object 类 所有物体的基类, position 表示在地图上的位置, instance 表示该物体在游戏场景中对应的实体。两个成员方法分别表示创建实体和销毁实体。

Item 类 所有实体（即有碰撞体积）的基类, 继承自 Object。两个相邻的实体间只有 mask 的与不为 0 才能够通过。Item 类有 Floor 和 Box 两个子类, 其中 Box 需要重写 SetInstance 函数以挂载动画控制器。Player 类继承自 Box 表示玩家类, nextPlayer 表示切换玩家时下一个玩家的引用。ControlledFloor 继承自 Floor 表示受机关控制的路面, active 表示路面一开始是否可见。

Trigger 类 所有触发器（不可碰撞）的基类, 同样继承自 Object。当箱子（玩家）进入触发器时调用 OnEnter 方法, 由一个箱子换成另一个箱子时调用 OnSwitch, 箱子移出触发器时调用 OnExit。触发器类有 Goal 和 Effect 两个子类。Goal 类表示箱子或玩家的目标, 若 isPlayer 为真表示是玩家的目标。Effect 类支持自定义三个触发函数, script 用于动态加载的挂有 BaseEffect 脚本的 GameObject, effect 存放该物体上的脚本, args 为传入 effect 脚本的参数。脚本加载可以分为以下几步:

1. 从 Assets/Resources 中加载名为 script 的预制体;
2. 从预制体中获取 BaseEffect 类型的 Component 并赋值给 effect;
3. 在机关触发事件时执行 effect 中相应的事件。

目前已经制作的机关分为以下几种:

1. ButtonO: 机关上有物体时显示指定路面, 物体移走时路面消失。该机关的 OnEnter 动作为显示路面, OnExit 动作为隐藏路面, 可调用 StageManager.SetItemActive 函数实现;
2. ButtonX: 物体移入时切换指定路面显示或消失的状态。可在 OnEnter 事件中通过 StageManager.SetItemActive 实现;
3. ShowInfo: 机关上有物体时显示提示信息。在 OnEnter 事件中若传入的参数为玩家则调用 InfoController 以显示提示信息, 在 OnExit 事件中让提示信息消失;
4. ShowTitle: 机关上有物体时显示 Logo 和游戏玩法。实现与 ShowInfo 类似;
5. SceneSwitcher: 场景切换, 用于切换到关卡编辑器等。在 OnEnter 中调用 SceneManager.LoadScene 即可;
6. LevelSelector: 关卡切换, 在关卡选择界面中使用。在 OnEnter 中调用 StageManager.SwitchLevel 即可。

3.3.2 视图

视图类的主要功能是给地图建边。所有视角的基类是 BaseView，其成员仅包含建图函数 Build。其类图见图 3.5。

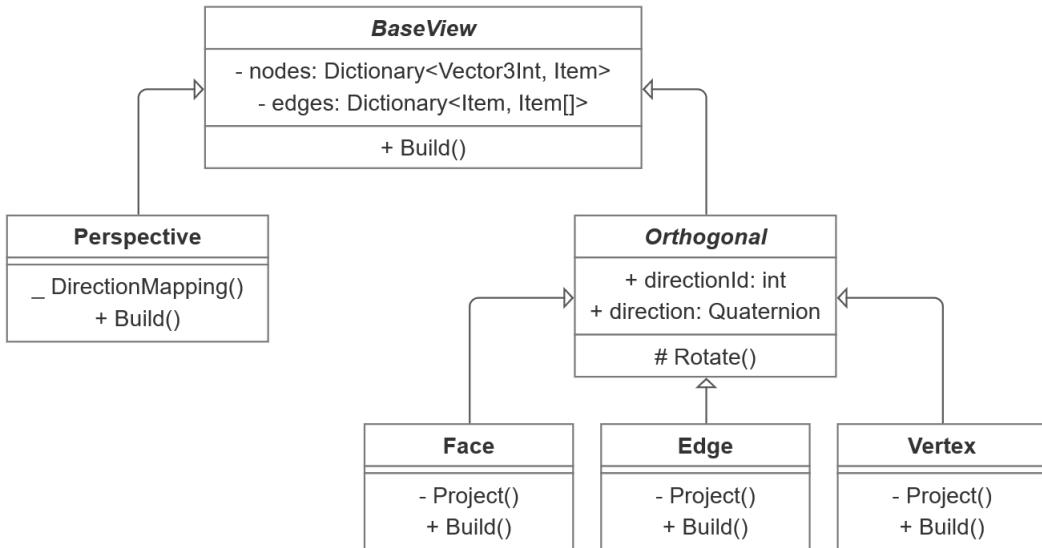


图 3.5: 视图的 UML 类图

BaseView 类有 **Perspective** 和 **Orthogonal** 两个子类。**Perspective** 类表示透视视角，即不允许任何视错觉，仅在三维空间中的相邻方块之间建边，实现起来比较简单。**Orthogonal** 类表示正交视角，分为顶点视图、边视图、俯视图三种视图，每种视图都需要计算不同的视错觉。

考虑玩家需要从 s 点移动到 t 点，此时摄影机的方向向量为 \mathbf{d} ，视图矩阵为 \mathbf{R} 。由于玩家观测不到 $s \cdot \mathbf{d}$ 与 $t \cdot \mathbf{d}$ 的不同， s 点与 t 点相邻当且仅当 \mathbf{Rs} 和 \mathbf{Rt} 在 xOz 平面上相邻。用 $\mathbf{P}(\mathbf{v})$ 表示摄影机看到的所有与点 \mathbf{v} 重合的点集，即 $\mathbf{P}(\mathbf{v}) = \{\mathbf{v}' \mid \mathbf{R}\mathbf{v}' = \mathbf{R}\mathbf{v}\}$ ，则 t 还需要满足 $t \cdot \mathbf{d}$ 是所有 $\mathbf{P}(t)$ 中最大的，这是因为不能允许物体在运动过程中主动被看起来在下面的方块遮挡。

除此之外，还需要满足 s 到 t 运动的路径中没有其他方块遮挡。遮挡判断的条件比较复杂，下面将分别说明三种视图的遮挡条件。

Face 类 即俯视图。我们需要让玩家站在箱子上时可以回到地面或在箱子上走动，在地面上时可以推动箱子，因此被遮挡的条件为 s 为玩家且 t 为箱子。

Edge 类 即边视图。不妨设此时摄影机的方向向量为 $\mathbf{d} = (0, -1, 1)$ ，相机的坐标系下 \mathbf{v} 的坐标为 $\mathbf{R}\mathbf{v} = (x, y - z, y + z)$ （如果是其他方向可以临时将所有物体绕原点进行旋

转，完成建图之后再旋转回去），则被遮挡的条件为

1. 移动方向为 z 轴正方向（若向负方向移动可以交换起点和终点的位置）： $y_t \leq y_s$ 且在集合 $\mathbf{P}(\mathbf{v} + (0, 0, 1))$ 与 $\mathbf{P}(\mathbf{v} + (0, 0, 2))$ 中均不存在满足 $y_t < y \leq y_s + 1$ 的物体。如图 3.6，图中两个灰色方块表示不能相互到达的起点和终点，黄色方块表示遮挡；

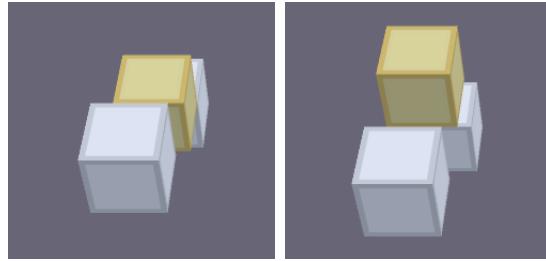


图 3.6: Edge 视图 z 正方向上的遮挡条件

2. 移动方向为 x 轴正方向：在集合 $\mathbf{P}(\mathbf{v} + (1, 0, 0))$ 中不存在满足 $y_t < y \leq y_s$ 的物体，且在 $\mathbf{P}(\mathbf{v} + (1, 0, 1))$ 中不存在满足 $y_t + 1 < y$ 的物体。如图 3.7；

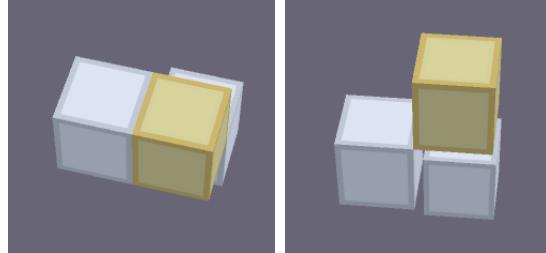


图 3.7: Edge 视图 x 正方向上的遮挡条件

Vertex 类 即顶点视图。不妨设此时摄影机的方向向量为 $\mathbf{d} = (1, -1, 1)$ ，相机的坐标系下 \mathbf{v} 的坐标为 $\mathbf{R}\mathbf{v} = (y + x, y - x - z, y + z)$ ，则被遮挡的条件为

1. 移动方向为 z 轴正方向，有以下 3 个条件：
 - (a) 在集合 $\mathbf{P}(\mathbf{v} + (1, 0, 1))$ 不存在满足 $y_t < y \leq y_s$ 的物体；
 - (b) 在集合 $\mathbf{P}(\mathbf{v} + (1, 0, 2))$ 不存在满足 $y_t + 1 < y \leq y_s + 1$ 的物体；
 - (c) 在集合 $\mathbf{P}(\mathbf{v} + (1, 0, 0))$ 不存在满足 $y_t < y \leq y_s$ 的物体。

如图 3.8；

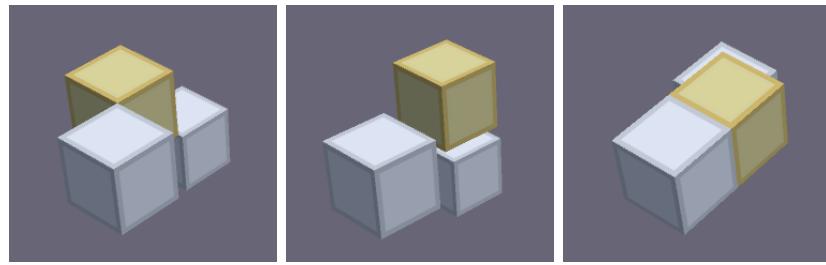


图 3.8: Vertex 视图 z 正方向（右上）的遮挡条件

2. 移动方向为 x 轴正方向，与 z 轴的情况对称，具体如下：

- 在集合 $\mathbf{P}(\mathbf{v} + (1, 0, 1))$ 不存在满足 $y_t < y \leq y_s$ 的物体；
- 在集合 $\mathbf{P}(\mathbf{v} + (2, 0, 1))$ 不存在满足 $y_t + 1 < y \leq y_s + 1$ 的物体；
- 在集合 $\mathbf{P}(\mathbf{v} + (1, 0, 0))$ 不存在满足 $y_t < y \leq y_s$ 的物体。

综上，现在我们得到了 s 可以向 t 连边的三个条件：

- $\delta = \mathbf{R}(s - t) \Rightarrow |x_\delta| + |z_\delta| = 1$;
- $t \cdot d = \max\{t' \cdot d \mid \mathbf{R}(t' - t) = \mathbf{0}\}$;
- s 到 t 的路径不能被遮挡。

3.3.3 地图

地图中储存了关卡中的所有物体与视图，其数据结构声明如下。

```

public class Map {
    public readonly Dictionary<Vector3Int, Item> items;           // 所有实体
    public readonly Dictionary<Vector3Int, Trigger> triggers;        // 所有触发器

    public readonly List<Player> players;                          // 所有玩家
    public readonly Dictionary<Vector3Int, Goal> goals;            // 所有目标点
    public System.Action triggerAction;                            // 触发器的动作
    public Views.BaseView view;                                    // 当前视角

    private readonly Dictionary<Item, Item[]> edges;             // 边
    private readonly Stack<MoveData> moveHistory;                 // 历史记录
}

```

上面结构中各成员的功能如下：

1. **items** 用哈希表储存地图中的所有实体。键为实体的坐标，值为实体的引用。可以保证每个键对应的实体唯一，因为不可能存在两个实体重叠；
2. **triggers** 与 items 类似，用哈希表储存所有触发器；
3. **players** 用数组存放所有玩家，按 playerId 升序排列；
4. **goals** 用哈希表储存所有终点的坐标。在场景管理器退出 Move 状态时检测是否所有目标上都有玩家或箱子，如果是则通关并进入关卡切换状态；
5. **triggerAction** 存放所有触发 trigger 后产生的动作。退出 Move 状态时执行所有动作并清空动作；
6. **view** 记录关卡当前的视角，每次更改的时候均需要调用 Build 函数重新建边；
7. **edges** 用一个哈希表作为邻接表，边的两个顶点均为实体，两个实体之间有变表示箱子或玩家可以从边的起点上方一格到终点上方一格；
8. **moveHistory** 用一个栈记录移动的历史记录以供撤销。MoveData 是一个 Item 到 (Vector3Int, Vector3Int) 对的映射，依次表示物体、起点坐标和终点坐标。

该类中最重要的函数是 TryMove 函数，表示尝试将给定玩家向给定方向移动一格，并推动该方向上的所有箱子。若移动成功返回移动结果，否则返回 null。该函数的算法流程如下：

1. 对于待移动物体 s ，找到 s 下方的物体 $s - \mathbf{k}$ ($\mathbf{k} = (0, -1, 0)$) 对应方向上的边 $e = (s - \mathbf{k}, t)$ 。如果边不存在则移动失败，直接返回 null；
2. 计算 s 上方的箱子高度。初始化 $h = 0$ ，如果 $s + h \mathbf{k}$ 上存在物体且不是路面则令 $h \leftarrow h + 1$ 。
3. 将 s 与所有 s 上方的玩家或箱子 $s + i \mathbf{k}$ ($0 \leq i < h$) 移动到 $t + (i + 1) \mathbf{k}$ ；
4. 回到第 1 步，移动 s 前方的物体 $s + d$ 。重复该操作直到 $s + d$ 上不存在物体；
5. 枚举最后得到的所有转移。如果目标位置上已经存在物体且不是箱子或玩家，则移动失败。

3.4 场景物体控制

3.4.1 摄影机控制

摄影机控制类包含控制摄影机的位置、方向、投影矩阵，并给视角切换状态提供控制摄影机方向的接口，在场景中挂载在 Main Camera 上。

控制摄影机的位置和方向 为了让摄影机一直跟随玩家移动，可把摄影机的位置设置为

$$\mathbf{p} = \mathbf{o} - d \mathbf{r} \quad (3.1)$$

其中 \mathbf{o} 表示玩家的坐标， \mathbf{r} 表示摄影机的方向向量， d 表示摄影机到玩家的距离，可以由鼠标滚轮调节。这样虽然可以保证玩家在摄影机的中心，但是当玩家速度变化过大时摄影机的移动会显得很突兀，因此考虑对摄影机的移动增加缓冲。

我采用一种迭代的方式更新摄影机坐标，即如果摄影机的目标位置为 \mathbf{e} ，则每一帧将摄影机当前坐标 \mathbf{p} 更新为

$$\mathbf{p}' = \mathbf{p} + \alpha (\mathbf{e} - \mathbf{p}) \quad (0 < \alpha \leq 1) \quad (3.2)$$

假设两帧时间间隔为 Δt ，时刻 0 时摄影机坐标为 \mathbf{s} ，计算可得时刻 T 时坐标为

$$\mathbf{p}(T) = \mathbf{s} + (1 - (1 - \alpha)^{T/\Delta t}) (\mathbf{e} - \mathbf{s}) \quad (3.3)$$

要使 \mathbf{p} 在时刻 T 与 \mathbf{e} 的相对坐标差小于 ε ，只需 $(1 - \alpha)^{T/\Delta t} < \varepsilon$ ，化简可得

$$\alpha < 1 - \varepsilon^{\Delta t/T} \approx -\frac{\ln \varepsilon}{T} \cdot \Delta t \quad (3.4)$$

取 $\varepsilon = 10^{-2}$ ， $T = 0.2$ s， Δt 由 Unity 中的 Time.deltaTime 获取，可以获得较好的效果。

控制摄影机角度的方式与控制位置相似，在此不作赘述。

投影矩阵 为了在显示器上展现三维的物体，渲染器需要把三维的点投影到一个二维平面上，进行这种投影变换需要投影矩阵。一般来说游戏引擎会自带透视和正交两种投影矩阵，但游戏为了实现透视与正交之间的切换需要构造其中间状态的投影矩阵。

假设摄影机位于原点，朝向 z 轴正半轴，我们现在要让观察空间 $z = d$ 平面处投影的大小不变（即玩家大小不变），方便起见我们先考虑 yOz 平面，如图 3.9.

令 $k = \tan x\alpha \cdot \cot \alpha$ ，可以计算出焦点 A 横坐标为

$$x_A = \left(\frac{\tan \alpha}{\tan x\alpha} - 1 \right) d = (k^{-1} - 1) d \quad (3.5)$$

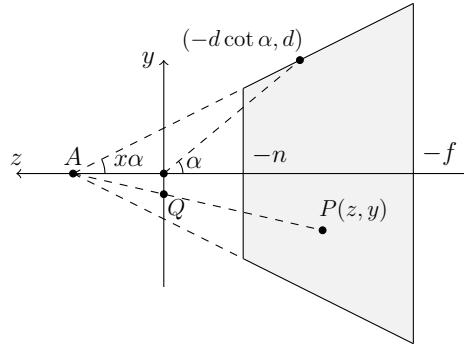


图 3.9: 视景体侧视图

其中 x ($0 \leq x \leq 1$) 表示透视线率, x 越小越接近正交的效果。对于点 $P(z, y)$, 我们求出 PA 与 y 轴的交点纵坐标 y_Q 与 $x_A \cot x\alpha$ 的比值可得 y 在投影屏面的映射为

$$y' = \frac{y_Q}{x_A \cot x\alpha} = \frac{\cot \alpha}{-kz + (1-k)d} \cdot y \quad (3.6)$$

假设屏幕的长宽比为 r , 同理也可得到 x 的投影为

$$x' = \frac{r \cot \alpha}{-kz + (1-k)d} \cdot x \quad (3.7)$$

接下来需要将 z ($-f \leq z \leq -n$) 映射到区间 $[-1, 1]$ 。令

$$\begin{cases} z' = \frac{Az + B}{-kz + (1-k)d} \\ z' |_{z=-n} = -1 \\ z' |_{z=-f} = 1 \end{cases} \quad (3.8)$$

这里分母是为了与 x', y' 统一, 可以解得

$$\begin{cases} A = \frac{k(n+f) + 2(1-k)d}{n-f} \\ B = \frac{2knf + (1-k)(n+f)d}{n-f} \end{cases} \quad (3.9)$$

为解决分母的非线性问题, 又引入 w 分量用于归一化, 实际得到的坐标都应该归一化使得 $w = 1$ 。由于上式中 x', y', z' 的分母都一致, 可以取 $w' = -kz + (1-k)d$ 。最终得到的投影矩阵为

$$M_{\text{proj}} = \begin{pmatrix} r \cot \alpha & 0 & 0 & 0 \\ 0 & \cot \alpha & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -k & (1-k)d \end{pmatrix} \quad (3.10)$$

3.4.2 动画控制

动画控制类主要用于控制物体的移动，挂载在每个可移动物体上。场景管理器进入 Move 状态时调用待移动物体的准备函数，在移动过程中调用移动函数，退出移动时调用退出函数。

这里移动物体不能简单地对其坐标线性插值来实现，否则很可能会出现遮挡错误的问题，因为物体移动前后可能不在同一平面上。一种解决思路是让物体一部分留在起点，另一部分从终点出现，类似于一个传送门，如图 3.10。这样虽然物体在三维空间上是分裂的，但在正交视图中是连在一起的。具体实现流程如下：

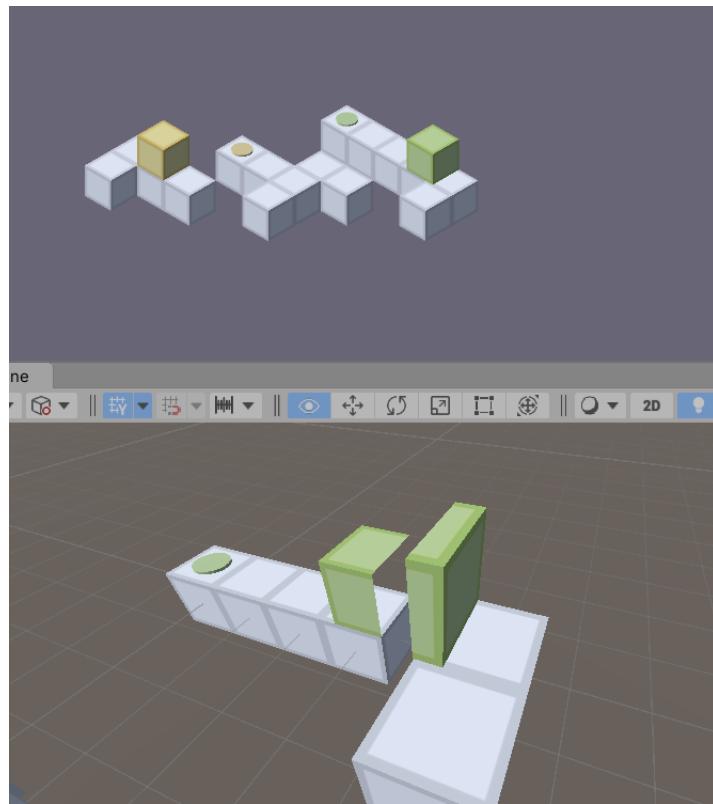


图 3.10: 移动动画实现原理（上：游戏界面；下：编辑器界面）

1. Prepare 函数隐藏待移动的对象并创建两个挂有 CopiedController 类的子对象 A, B。A 的坐标为起点，B 的坐标为终点前一格；
2. Move 函数通过改变两个子对象的 shader 属性（这里的裁剪效果是片段着色器实现的），隐藏物体 A 和 B 超出所在方格的部分；
3. Exit 函数删除两个子对象并显示父对象。

3.5 关卡加载与保存

这一部分主要是实现了对关卡数据的序列化与反序列化。关卡数据是用一个类似文件夹的结构储存的，提供了读取某个文件夹内所有指定类型的物体的功能。例如关卡中所有物体放在“Objects”内，视图放在“Views”内。除此之外还可以用于存放一些关卡信息，例如关卡基本信息、下一关的关卡名称和退出时返回到的关卡名称。通过这种方式储存关卡信息可以仅通过编辑 json 文件就能修改整个游戏流程，包括选关界面和游戏界面，便于制作关卡。

储存关卡所用的结构框架如下：

```

internal class JsonData {
    public Dictionary<string, JsonData> nodes;
    public Dictionary<string, List<object>> data;
}

public class Level {
    public string name { get; private set; }
    private JsonData objs;

    public void Add(string path, object obj); // 添加物品
    public List<T> FindAll<T>(string path); // 递归查找所有物品
    public List<T> Find<T>(string path); // 查找所有物品
    public T FindOne<T>(string path); // 查找一个物品
    public stringToJson(); // 序列化
    public static Level FromJson(string json); // 反序列化
}

```

JsonData 类是一个树状的数据结构，其中 nodes 记录了当前节点的所有子节点。data 用一个哈希表记录了当前结点保存的所有信息，键的类型为字符串，表示数据的类型；值为一个 object 的列表，存放了对应类型的所有数据。

Level 类表示关卡数据，其中 name 记录了关卡的名字，objs 存放了关卡的所有数据信息。该类实现了 ToJson 与 FromJson 函数用于序列化与反序列化，同时实现了 Find 和 Add 函数分别用于查找和修改关卡信息。

3.6 关卡编辑器

3.6.1 物块创建与删除

在关卡编辑器中，鼠标点击实现物块的创建与删除。物块创建的关键就是获取鼠标点击的位置，并通过该位置获取待生成物块的规格化坐标。鼠标点击的位置可通过射

线检测方法获得，由于游戏中实体物块（如地面、玩家和箱子）不可重叠，且物块必须位于整点处，故需要将射线检测获得的碰撞点取整处理。但实际碰撞点往往是边界坐标，且无法仅通过碰撞点获得期望的物块生成方向，可以在射线检测中获取射线的反方向向量 deltaRay，deltaRay 的模取较小值，计算物块坐标时将碰撞点坐标 target 加上 deltaRay，获得碰撞点向物块的期望生成方向偏移一个极小值的坐标点，对该坐标点进行规格化和取整处理，即可获得物块的生成坐标 position，坐标计算公式如下：

$$\text{position} = \left\lfloor \frac{\text{target} + \text{deltaRay}}{\text{scale}} \right\rfloor \cdot \text{scale} + \frac{1}{2} \text{scale} \quad (3.11)$$

物块的删除逻辑较为简单，只需要获取射线碰撞体，判断该物体是创建的实体物块即可删除。

3.6.2 鼠标点击事件状态机

由于鼠标按键存在操作上的复用，如单击左键创建物块，长按左键并拖动鼠标实现视角转换，双击左键或单击右键实现物块删除。鼠标左键的事件状态转换图如图 3.11 所示。

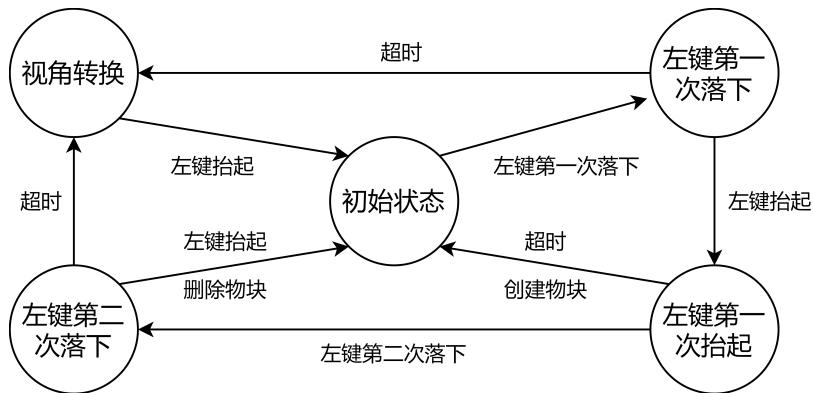


图 3.11: 鼠标左键事件状态转换图

3.6.3 UI

关卡编辑器 UI 用于实现物料的选择，保存与加载及返回主界面等用户交互。UI 界面的操作需要与物块创建与删除、视角转换等逻辑进行隔离，即游戏对 UI 事件的响应优先级应高于其他事件。

4 实现与测试

4.1 实现环境与代码管理

开发所用的软硬件环境见表 4.1。

CPU	Intel i7-1165G7 @ 2.80 GHz
RAM	32.0 GB
OS	Windows 10 (x64)
Platform	Unity 2021.2.7f1c1

表 4.1: 实验开发环境

代码版本管理的签入记录见图 4.1。

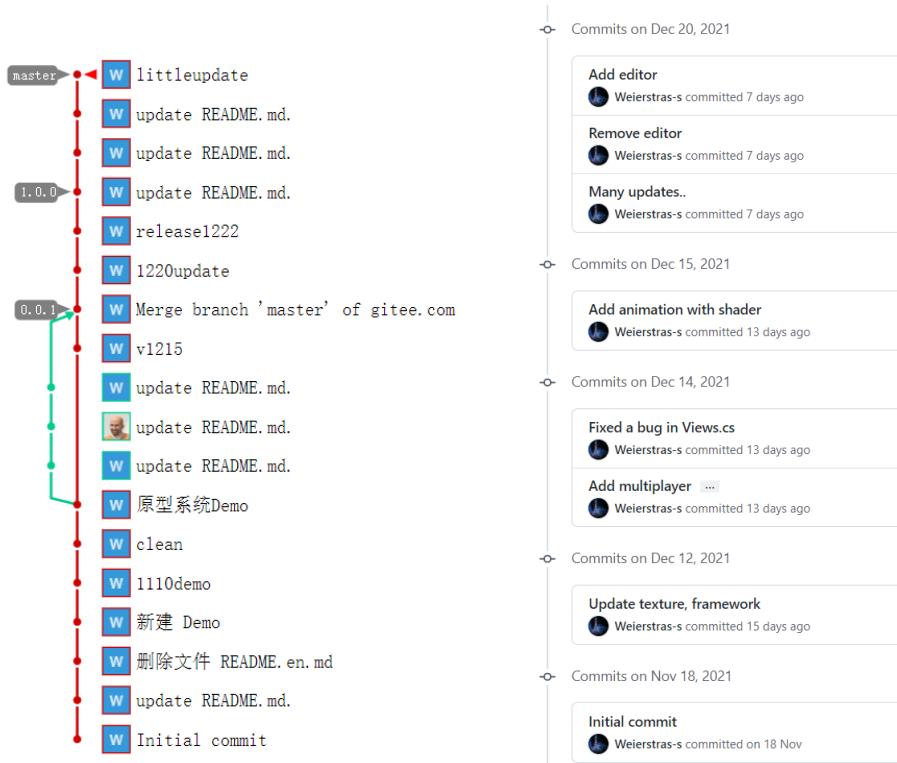


图 4.1: 代码版本管理签入记录

4.2 关键函数说明

4.2.1 状态机模板

```

namespace FSM {
    public delegate bool FSMTransition(ref object enter, ref object exit);
    public class FSMState<T> {
        public T self;
        public FSM<T> fsm;
        public virtual void Enter(object param);
        public virtual void Exit(object param);
        public virtual void Update();
        public void AddTransition<S>(FSMTransition cond);
    }

    public class FSM<T> {
        public void AddState<S>();
        public void ExitState(object exit);
        public void EnterState<S>(object enter);
        public void Translate<S>(object enter, object exit);
        public void Update();
    }
}

```

1. FSM: T 为受 FSM 控制的类, 使用时需在 T 中声明该 FSM 作为私有变量, 其成员函数功能如下:
 - (a) AddState: 向状态机中添加一个状态 S;
 - (b) ExitState: 退出当前状态, exit 为传给退出状态的参数;
 - (c) EnterState: 进入状态 S, enter 为传给进入状态的参数;
 - (d) Translate: 退出当前状态并进入状态 S, 即先后执行 ExitState 和 EnterState;
 - (e) Update: 状态机的更新函数, 在 T 的 Update 函数中执行, 每帧执行一次。
2. FSMState: 状态机的每个状态均需要继承该类并重写 Enter, Exit 和 Update 函数, 其成员函数功能如下:
 - (a) Enter: 进入该状态时执行, 在 FSM.EnterState 中调用;
 - (b) Exit: 退出该状态时执行, 在 FSM.ExitState 中调用;
 - (c) Update: 状态激活时每帧执行一次, 在 FSM.Update 中调用;
 - (d) AddTransition: 给当前状态增加转移, 当 cond 函数返回值为真的时候转移至状态 S;

3. FSMTransition: 状态机的转移, 委托返回值为是否需要转移, enter 和 exit 分别为转移条件成立时转移的退出和进入参数。

4.2.2 场景管理

```

public class StageManager : MonoBehaviour {
    public static StageManager manager { get; private set; }
    private static FSM<StageManager> fsm; // 状态机
    public new CameraController camera; // 摄影机控制器

    public Level currentLevel; // 当前关卡
    public Map map; // 地图
    public Player player; // 当前玩家
    public List<Orthogonal> views; // 所有可用视角
    public Dictionary<string, ControlledFloor> controlledItems; // 受控物块

    private GameObject GetPrefab(BaseObj item);
    public void LoadLevel(Level level); // 加载关卡
    public void SwitchLevel(Level level); // 切换关卡
    public void SetItemActive(string id, bool isActive); // 设置机关是否被激活
}

```

1. LoadLevel: 加载给定关卡。根据关卡数据修改 map 等成员的数据, 并调用 GetPrefab 函数加载关卡中所有物体对应的实体。由状态机 SwitchLevel 状态的第二阶段调用。
2. SwitchLevel: 切换至给定关卡。需要触发所有机关的退出动作并将 fsm 转移到 SwitchLevel 状态, 传入待转移的关卡数据。
3. SetItemActive: 设置机关是否被激活, id 为每个受控路面对应的唯一标识 (对应 controlledItems 的键值), isActive 表示是否激活。该函数由机关触发时调用。

4.2.3 关卡管理

视图

```

public class BaseView {
    public Dictionary<Vector3Int, Item> nodes;
    public Dictionary<Item, Item[]> edges;
    protected virtual void Build(); // 建图并存入 edges
}

```

```

public class Perspective : BaseView {
    public static int DirectionMapping(int dir, Quaternion cam);
    protected override void Build();
}

public class Orthogonal : BaseView {
    public int directionId = 0;
    public virtual Quaternion direction; // 摄影机方向
    protected void Rotate(int dir); // 坐标变换
}

```

1. BaseView: 所有视角的基类。Build 函数用于根据 nodes 的值创建 edges。
2. Perspective: 透视类, 继承自 BaseView。DirectionMapping 用于根据当前摄影机的方向确定按键映射。
3. Orthogonal: 正交类, 继承自 BaseView。各函数功能如下:
 - (a) direction: 获取当前方向下摄影机的角度, 由 AdjustCamera 状态结束时调用, 用于摄影机吸附效果。
 - (b) Rotate: 用于将所有物体旋转至标准方向, 便于 Build 函数处理。在 Build 函数中开始和结束时调用。

地图

```

public class Map {
    public bool IsWin(); // 判断当前状态是否获胜
    public void Build(); // 建图

    public MoveData TryMove(Item player, int dir); // 尝试移动玩家
    private void Move(MoveData data, bool rollback); // 根据移动信息移动玩家
}

```

1. IsWin: 判断是否获胜。由 Move 状态结束时调用。
2. Build: 建图。当修改变量 view 或触发与地图状态相关的机关时调用。
3. TryMove: 尝试将 player 向 dir 方向移动一格, 若移动成功返回 MoveData, 否则返回 null。在 Move 状态开始时调用。
4. Move: 根据 MoveData 移动物体。在 TryMove 函数成功后调用。

4.2.4 场景物体控制

摄影机控制

```

public class CameraController : MonoBehaviour {
    public class ProjectionMatrix {
        // 计算投影矩阵 x: 透视率
        public static Matrix4x4 Get(float x, Rect rect, float camDist);
    }

    private Smoothing<float> dist;           // 到玩家的距离
    public Smoothing<Vector3> center;         // 摄像机中心
    public Smoothing<Quaternion> rotation;     // 摄像机角度
    public Smoothing<float> persp;            // 透视率

    public void Init();                         // 初始化相机
    public void SetView(BaseView view, Quaternion? def); // 根据当前地图视角设置方向
    public BaseView GetTargetView(bool isPersp); // 选取与相机当前方向最接近的目标视角
    public void UpdateDist(float delta);        // 通过鼠标滚轮调节距离
}

```

1. Init: 初始化相机。包括将中心设置为玩家坐标，旋转角设置为默认视图，并清除一切动画状态。由状态机 SwitchLevel 状态的第二阶段调用。
2. SetView: 根据 view 和 def 的值设定摄影机的方向。如果 view 不是透视状态则根据 view.direction 设置方向，否则根据默认值 def 设置。由状态机 AdjustCamera 状态结束时调用。
3. GetTargetView: 选取与相机当前方向最接近的目标视角，用于实现摄影机吸附最近的正交视角的效果。同样由 AdjustCamera 状态结束时调用。
4. UpdateDist: 通过鼠标滚轮调节距离，delta 表示鼠标滚轮的位移。由 Idle 状态 Update 时调用。

动画控制

```

public class AnimController : MonoBehaviour {
    public class CopiedController : MonoBehaviour {
        public void SetDirection(Vector3 dir);
        public void SetHeight(float x);
        public void SetAlpha(float x);
    }
}

```

```

private readonly List<CopiedController> copies; // 复制体列表

private void SetDirection(Vector3 dir);           // 设置裁剪方向
private void SetHeight(float x);                  // 设置裁剪高度
private void SetAlpha(float x);                   // 设置高亮

public CopiedController Copy();                  // 复制实体
public void Prepare(Vector3 dest, bool rollback); // 初始化移动
public void Move(float x);                      // 移动
public void Exit();                            // 结束移动
}

```

1. SetDirection: 设置 shader 中的 Direction 属性, 即物体裁剪的方向。
2. SetHeight: 设置 shader 中的 Height 属性, 即裁剪时保留的高度, 取值为 $[-1, 1]$ 。
3. SetAlpha: 设置 shader 中的 Alpha 属性, 即上层贴图的透明图, 可实现高亮效果。
4. Copy: 将自己复制到一个 CopiedController 中并存入 copies。
5. Prepare: 初始化移动。调用 Copy 函数复制两个实体, 并将自己隐藏。
6. Move: 移动物体, x 为移动的时间。调用 SetDirection 和 SetHeight 实现图 3.10 所示效果。
7. Exit: 结束移动动画。调用 Destroy 函数删除两个复制体并显示自身。

4.2.5 关卡编辑器

射线检测

```

Ray ray;
RaycastHit hit;
bool isHit;
Vector3 deltaRay; //射线反射方向
private void GetRay() {
    // 射线检测
    ray = Camera.main.ScreenPointToRay(Input.mousePosition); // 屏幕坐标转射线
    isHit = Physics.Raycast((Ray)ray, out hit);             // 发出射线检测到了碰撞
    // 模很小的射线反射方向向量
    deltaRay.x = (float)((float)Math.Sign(ray.direction.x) * -0.01);
    deltaRay.y = (float)((float)Math.Sign(ray.direction.y) * -0.01);
    deltaRay.z = (float)((float)Math.Sign(ray.direction.z) * -0.01);
}

```

GetRay 函数实现射线检测和碰撞点处理。调用 Camera.main.ScreenPointToRay 方法将鼠标所处位置屏幕坐标转为摄像机发出的射线；调用 Physics.Raycast 方法判断该射线是否与碰撞体产生碰撞，并将碰撞点信息存入 hit 变量中。deltaRay 变量是根据射线方向构造的反方向向量，该向量的模取很小的值，用于后续物块生成位置的判断。

鼠标事件状态机

```
public class CreateCube : MonoBehaviour {
    private void MouseStateManager();
    private void FixedUpdate();
}
```

1. MouseStateManager：用于判断转换条件
2. FixedUpdate：实现了状态转换之间的计时功能

物块创建与删除

```
public class CreateCube : MonoBehaviour {
    public void ChangeToBox();
    public void ChangeToFloor();
    public void ChangeToPlayer();
    private void CubeCreator();
    private void CubeDesstroyer();
}
```

1. ChangeToBox、ChangeToFloor、ChangeToPlayer 等：游戏中存在箱子、地板、玩家等多种游戏元素，点击 UI 相应按钮，调用 ChangeToBox、ChangeToFloor、ChangeToPlayer 等函数实现实体物块元素类型切换。
2. CubeCreator：实现物块的创建，物块的位置根据式 3.11 计算得到。
3. CubeDesstroyer：实现物块的删除，判断射线检测中射线的碰撞对象是否为预制块的标签，若是则使用 Destroy 方法消除该块。判断的目的是避免消除地面。

关卡保存与加载

```
public class CreateCube : MonoBehaviour {
    private Dictionary<Vector3Int, BaseObj> items;
```

```

private Dictionary<Vector3Int, GameObject> gameObjects;
private Level currLevel_l;
private string currLevel_s;
public void SaveToJson();
public void LoadFromJson();
}

```

1. SaveToJson：关卡编辑中实时创建的物块存放于 items 和 gameObjects 中。点击 UI 中的 SAVE 按钮，调用 SaveToJson 方法，首先将 items 转换为关卡 Level 类，再调用 Level 类中的 ToJson 方法转换为字符串类型的 Json 格式文件输出。
2. LoadFromJson：点击 UI 中的 LOAD 按钮实现在关卡编辑器中加载关卡。加载关卡之前首先需要销毁 gameObjects 字典中保存的当前视图下的已有物块，再使用 Level 类的 FromJson 方法从 Json 格式字符串中读取关卡信息，生成读取的关卡中的物块。

4.3 测试计划、测试用例与结果分析

4.3.1 常用测试方法说明

软件测试指的是一种用来促进鉴定软件的正确性、完整性、安全性和质量的过程。常用的软件测试方法包括黑盒测试和白盒测试。

顾名思义，黑盒测试并不关注软件的内部实现，目的是测试软件是否满足功能需求。它从使用者的角度，从用户体验的各个方面检测软件是否满足需求，如是否有遗漏功能、是否存在界面出错或性能问题等。常见的黑盒测试技术包括等价类划分法、边界值分析法和对比测试法。等价类划分法将程序的输入数据集合按照输入条件划分为若干个等价类，每个等价类相对于输入条件表示为一组有效或无效的输入，然后为每一等价类设计一个测试用例；边界值分析法是对等价分类技术的补充，在一个等价类中，选择等价类边界上的值进行测试；对比测试法是指最终不止产生一个软件版本，而是多个版本并行测试并比较结果，如大多软件都包含的测试版和发行版。

与黑盒测试相对，白盒测试根据程序内部的控制结构设计测试用例，保证：模块中的每一独立的路径、判断的每一分支和循环的边界和一般条件都至少执行一次，并且所有内部数据结构的有效性都得到验证。常见的白盒测试技术包括逻辑覆盖法、路径测试法。这些技术都聚焦于软件的内部实现，目标是验证软件内部模块结构的实现效果符合预期。

4.3.2 视错觉逻辑测试

该模块是游戏的核心逻辑，需要根据视角判断所有物体之间的连通性，设计目标是建出来的图应符合直觉，即看上去能到达的地方就有边相连。该模块的测试是对 Views.* 类的单元测试，采用白盒测试中的逻辑覆盖法，需要分别对每个视角构造出可达和不可达的样例，通过 Debug.DrawLine 函数在编辑界面可视化地将图绘制出来。所选取的测试用例需要全面覆盖图 3.6、图 3.7 和图 3.8 所示的所有情况。

俯视图 该视图下的测试用例与测试结果如图 4.2。对该测试用例的分析如下：

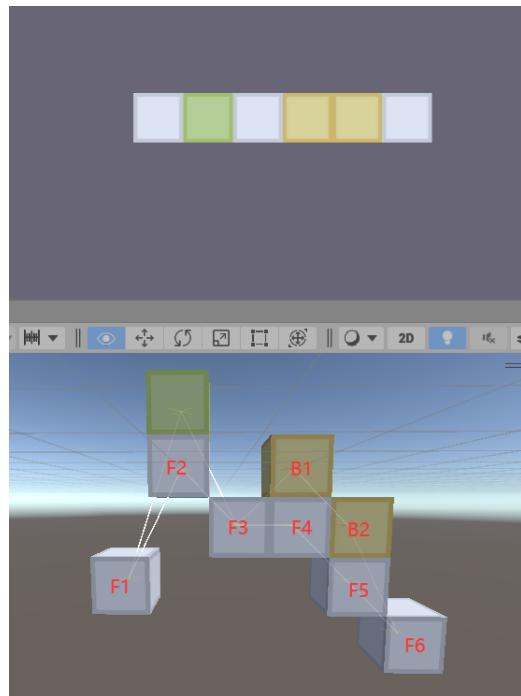


图 4.2: 俯视图测试

1. 尽管路面 F1 至 F6 不在同一平面上，但由于俯视图中 6 块路面在同一水平线上，因此可以相互到达。可以看到下面的编辑界面中相邻路块之间有白色的边相连；
2. 在俯视图下箱子 B1 和 B2 是相邻的，并且由于从箱子上可以回到路面上，因此 B1 与 B2 之间有边，从 B1 到 F3、从 B2 到 F6 有边；
3. 从路面上不能主动到箱子上，而是优先推动箱子，因此没有从 F3 到 B1、从 F6 到 B2 的边。

边视图 为全面覆盖边视图中两个移动方向的各种情形，我对该视图构造了两个测试用例，分别用于测试有遮挡和没有遮挡时建图是否符合要求，如图 4.3。对两个测试用例

的分析如下：

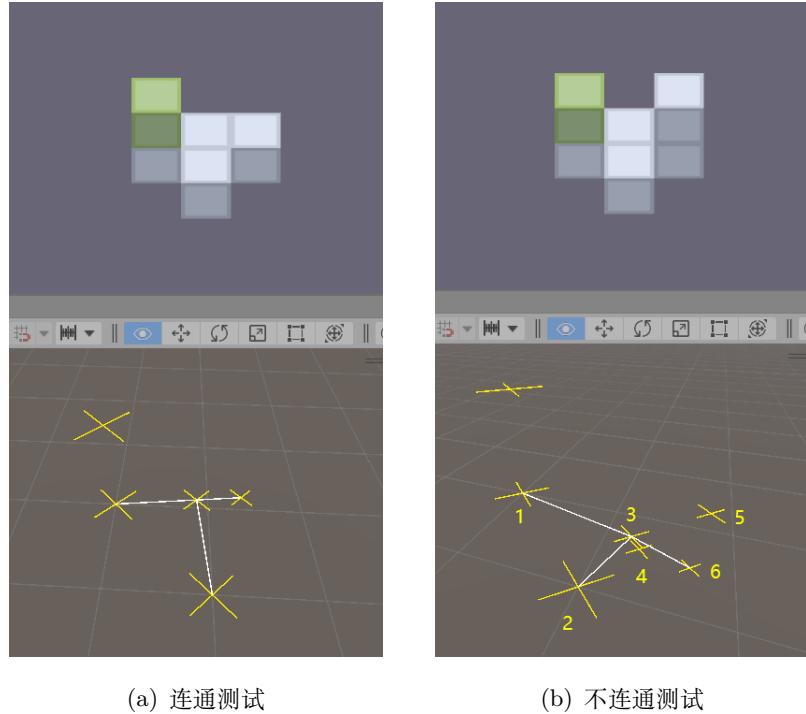


图 4.3: 边视图测试

1. 在 (a) 图中，所有地面之间可以相互到达，因此在编辑界面中可以看出来摄影机视角下相邻的黄色标记之间均有边相连；
2. 在 (b) 图中，1 号和 3 号、2 号和 3 号路面之间仍可以相互到达，但由于 4 号路面被 3 号遮挡，因此 2 号与 4 号路面之间没有边。
3. 由于 6 号路面被 5 号遮挡，故 3 号路面无法直接到 6 号。图中所示的白边表示从 6 号路面可以到 3 号，但由于不可能到达 6 号位置，因此这条边其实是无效的。

顶点视图 该视图下的测试用例的构造方式与边视图类似，但由于测试条件较多构造起来比较复杂。如图 4.4。对三个测试用例的分析如下：

1. 在 (a) 图中，所有地面之间可以相互到达，因此编辑界面中相邻点之间均有边；
2. 在 (b) 图中，由于 1 号路面被 2 号路面遮挡，因此仅 2 号与 3 号路面之间有边，1 号路面没有与 3 号路面相连；
3. 3 号路面的右表面遮挡了 3 与 4 间的路径，因此 3 号与 4 号路面之间没有边；

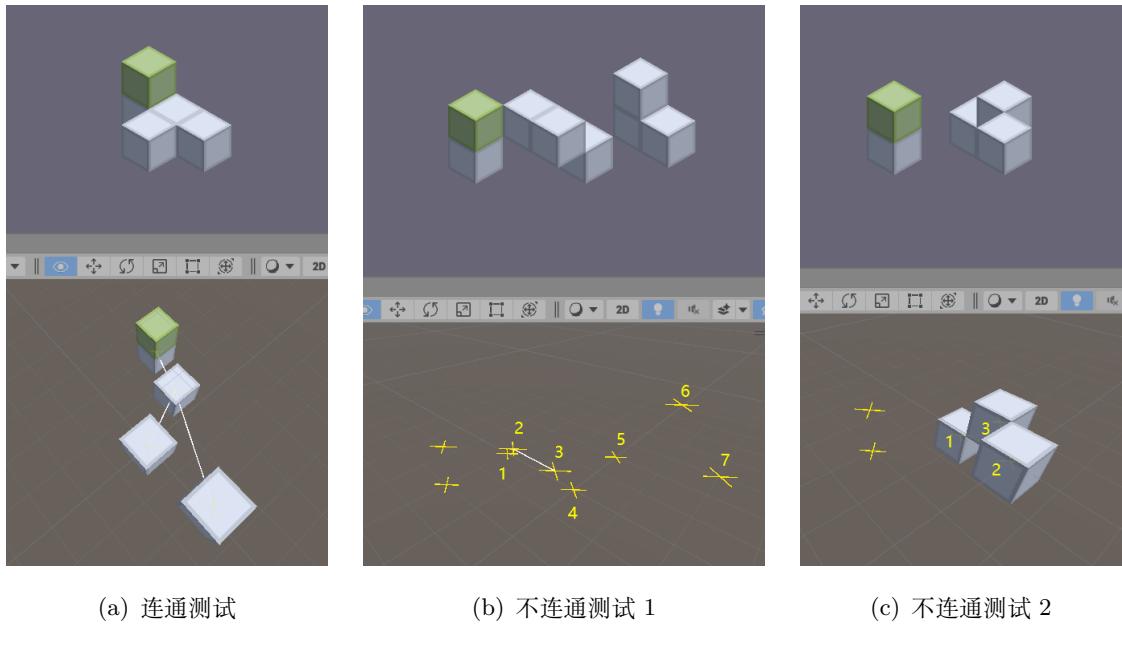


图 4.4: 顶点视图测试

4. 在摄影机的视角中 6 号路面在 5 号正上方，因此 5 与 7 之间没有边；
5. 在 (c) 图中，3 号路面的做表面遮挡了 1 与 2 间的路径，因此 1 与 2 之间没有边；

4.3.3 移动逻辑测试

移动功能是 Map 类的核心功能之一，包括物体在视错觉逻辑下的移动和箱子的推动。该模块需要集成在视错觉逻辑模块上进行测试，同样采用逻辑覆盖法进行测试。

移动功能 移动功能的测试用例与结果如图 4.5。测试过程与分析如下：

1. 测试过程：在 Start 函数中设置视角为 Vextex(0) 并调用 Move(0) 三次后调用 Move(3)，表示向右上方走 3 步后向右下方走 1 步；
2. 测试结果：可以观察到玩家从左上角方块移动到了右下角，说明移动功能正确。

推动功能 推动功能的测试用例需要覆盖以下几条路径：

1. 正常推动箱子，包括移动路径上有多个连续的箱子和箱子有多层；
2. 推动箱子失败，包括箱子被推到路面边缘和有物块挡住了箱子的路线；

构造的测试用例如图 4.6，测试过程与结果分析如下：

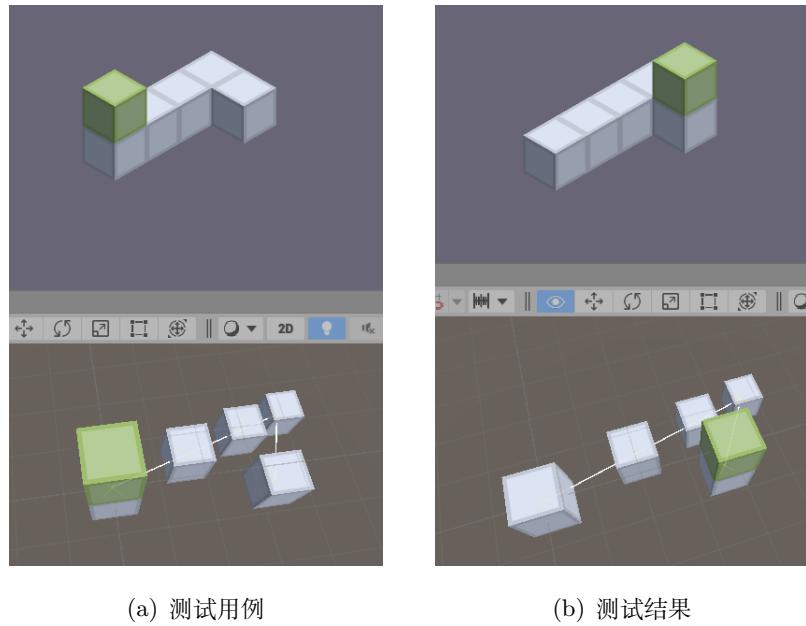


图 4.5: 移动功能测试

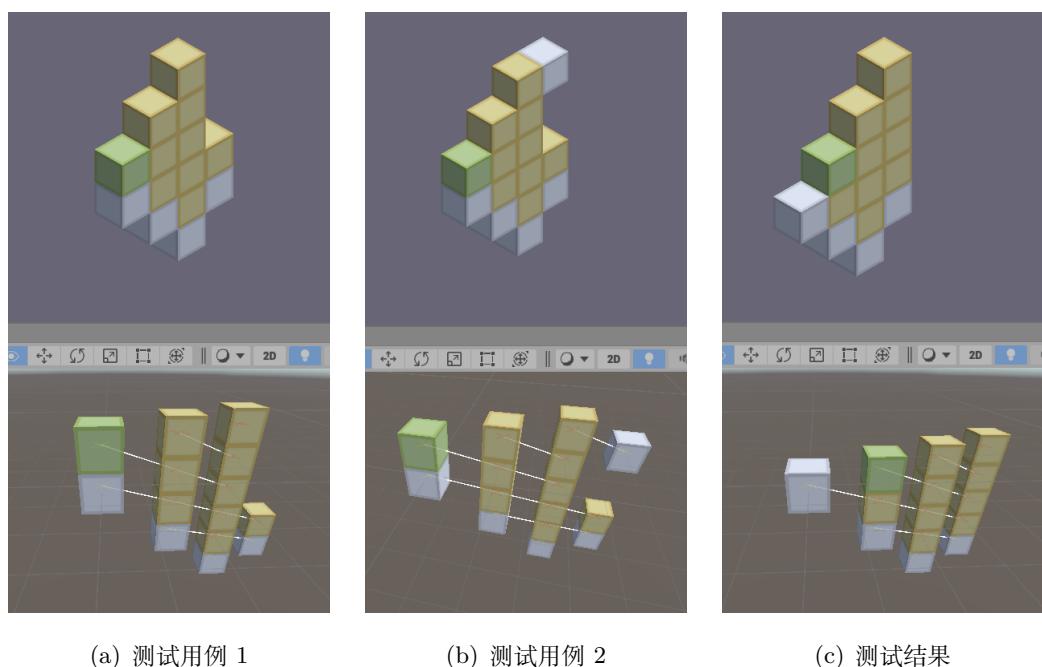


图 4.6: 推动箱子功能测试

1. 测试过程：在测试用例 1 中调用 Move(0) 两次，其中第一次移动对应第一条路经，第二次移动对应第二条路经的前半部分。在测试用例 2 中调用 Move(0) 一次，对应第二条路经的后半部分；
2. 测试结果：测试用例 1 的结果如图 (c)，玩家仅前进一格，与预期一致。测试用例 2 中玩家与箱子位置均没有发生变化，表示推动失败。因此推动部分逻辑正确。

4.3.4 状态机逻辑测试

状态机模块需要支持动画的展示、对玩家鼠标或键盘的输入给出响应。对该模块的测试采用路径测试法，几条基本的测试路径如表 4.2。

ID	测试内容
1	拖动鼠标旋转视角 → 连续按下两次方向键
2	按下方向键不放 → 通关
3	按下方向键 → 按下方向键但移动失败

表 4.2: 状态机测试路径

每当状态机进入或退出状态时在控制台中输出状态机当前的状态，得到测试结果如表 4.3。

ID	测试结果
1	Init → Idle → AdjustCamera → Idle → Move → Move → Idle
2	Init → Idle → Move → Move → Move → SwitchLevel → Idle
3	Init → Idle → Move → Idle → Move → Idle

表 4.3: 状态机测试结果

状态机运行结果与预期一致，功能正确。

4.3.5 关卡编辑器测试

物块选择、创建与删除 点击物品选择栏的对应物块选择物块类型，根据关卡设计需求使用鼠标左键点击关卡编辑区域，可生成对应类型的物块；长按地图并拖动鼠标可转换视角，不会生成物块；点击 UI 栏只会获得 UI 响应，不会产生视角的转换与物块的生

成。使用鼠标左键双击或鼠标右键单击已生成物块可以消除该物块。关卡编辑器的测试示例如图 4.7 所示。在测试中，物块的选择逻辑有效，物块生成与删除功能正常。

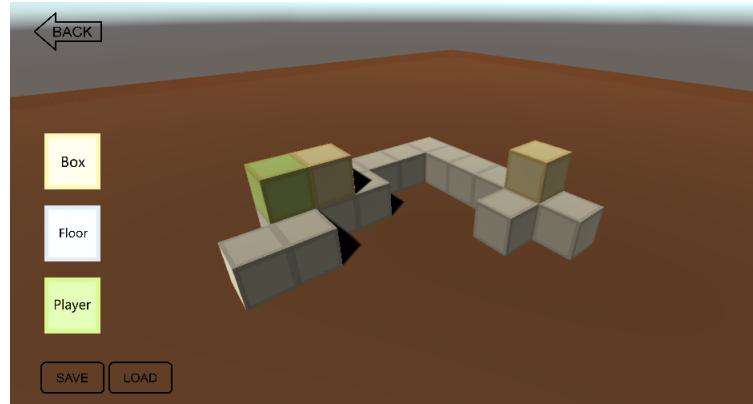


图 4.7: 关卡编辑器功能测试

关卡的保存与加载 完成关卡编辑后，可点击 SAVE 按钮实现关卡信息转换为 JSON 格式信息的输出，图 4.7 对应的 JSON 文件（部分）如图 4.8 所示。若点击 LOAD 按键，可清除当前编辑区域内容，并在编辑区自动生成读取的 JSON 文件中的关卡对应地图信息。

```

2   "Box": [
3     {
4       "mask": -1,
5       "position": "-1,1,-3"
6     },
7     {
8       "mask": -1,
9       "position": "2,1,1"
10    }
11  ],
12  "Player": [
13    {
14      "playerId": 0,
15      "mask": -1,
16      "position": "-1,1,-4"
17    }
18  ],
19  "Floor": [
20    {
21      "mask": -1,
22      "position": "-2,0,-1"
23    }
],

```

图 4.8: 关卡保存功能测试

5 前端实现与测试

5.1 网站架构

网站架构图如图 5.1 所示。在开发的过程中，我们希望用户能够被落地页吸引过来并且由此点击进入主页，找到我们游戏发布的官网，并最终在官网里面畅游。对于官网，我选择的这几个页面都是有用处的，归档页可以快速查看我们博客（通常是新手教程和版本更新）的发布时间和简单摘要，落地页是为了保证双向联通性，关于页是为了说明游戏的发布团队以及该团队最近是否有其他项目正在开发或者已经开发，最后的社区页是待实现的功能，也是今后要着重完善的地方。

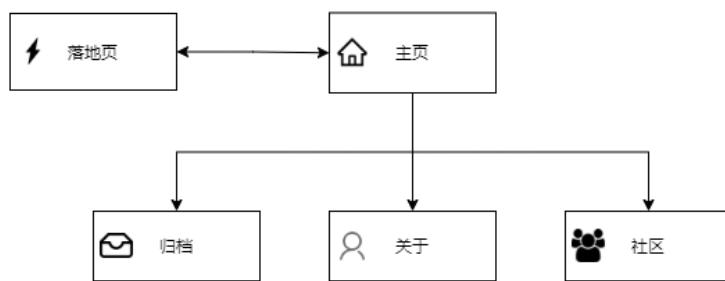


图 5.1: 前端系统架构图

5.2 各网页的说明和特性

5.2.1 主页

首先给出主页以及其他各个页面的细节部分。当用户进入网页时，会根据对方的网速和信号状况动态加载页面，并且为了不让用户在等待中感到枯燥，这里加入了预加载页面，如图 5.2 所示。这是一个吃糖豆的动态效果图，用纯 css 动画实现。具体使用关键帧来实现的。如图 5.3 所示。

而主页部分最为吸睛的是 banner 图部分，这里我将 banner 图改为只有一张，但以后如果有重磅消息更新，完全可以放在轮播图上进行轮播，对于现在的状态，呈现给我们的就是一张我们游戏的高清大图，为了防止背景颜色和首页的字体颜色重合（模糊不清），我对背景图片上蒙上了一层灰色蒙版，用于区分。这些都是细节，也是一个前端系统应当做到的为用户着想。主页图如图 5.4。

点击开始阅读时，会自动向下翻页，到达博客发布处，而点击现在下载，即可直接下载我们游戏的 exe 文件，点击即可运行玩耍，不需要任何依赖，这也是我做出的为用户着想的一个体现。同时也可从图 5.4 中看到，下方有一个 github 图标，这里链接的

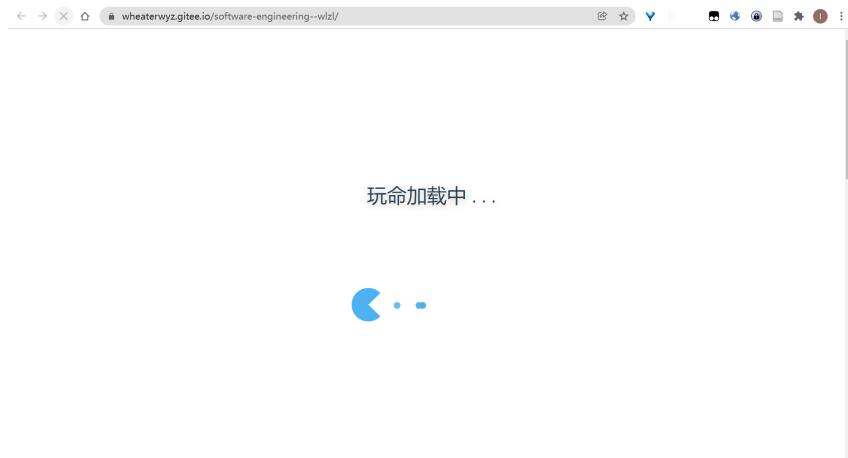


图 5.2: 预加载页面

```
@-webkit-keyframes rotate_pacman_half_up {0% {transform: rotate(270deg)}50% {transform: rotate(1turn)}to {transform: rotate(270deg)}}

@keyframes rotate_pacman_half_up {0% {transform: rotate(270deg)}50% {transform: rotate(1turn)}to {transform: rotate(270deg)}}

@-webkit-keyframes rotate_pacman_half_down {0% {transform: rotate(90deg)}50% {transform: rotate(0deg)}to {transform: rotate(90deg)}}

@keyframes rotate_pacman_half_down {0% {transform: rotate(90deg)}50% {transform: rotate(0deg)}to {transform: rotate(90deg)}}

@-webkit-keyframes pacman-balls {75% {opacity: .7}to {transform: translate(-100px, -6.25px)}}

@keyframes pacman-balls {75% {opacity: .7}to {transform: translate(-100px, -6.25px)}}
```

图 5.3: 预加载代码实现

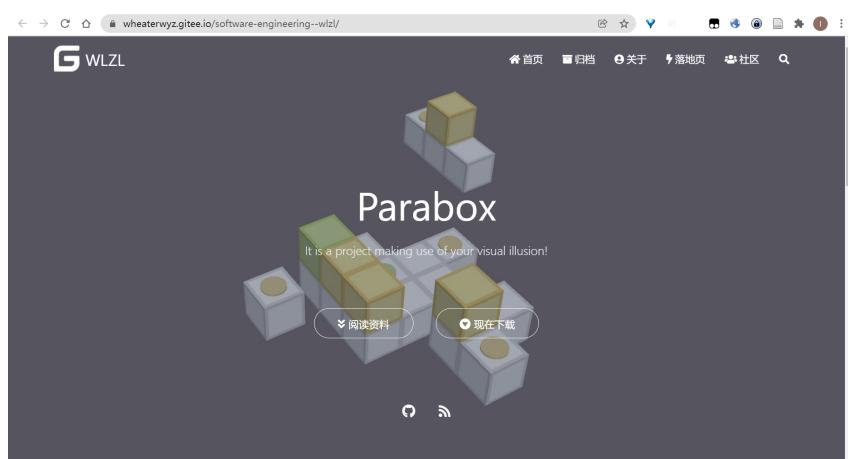


图 5.4: 首页轮播图

其实是我们的 gitee 仓库，另一个是 rss 订阅，这是真实有用的。我写出这个功能是为了之后如果我们发布了新的博客到此官网上，用 rss 订阅订阅了本网站的人会受到一条消息推送，使得玩家可以非常快速的得知最新消息和最快的下载新的版本。

为了使大家快速了解我们的产品游戏，我在首页设置了游戏背景一栏说明开发此游戏的初衷，和首页秀这一栏，用一个短小精悍的视频简洁有力的说明了本游戏的玩法和通关效果。如图 5.5 所示。可以将图 5.5 与图 5.4 对比，可以发现此时页眉出现了一条紫色向蓝色渐变的横幅，上面清晰的标注了我们团队的名称，以及各个子页面的图标和超链接，同时也要注意到，最右端有一个搜索图标，此图标也是真实有效的，点击之后搜索的效果如图 5.6 所示。

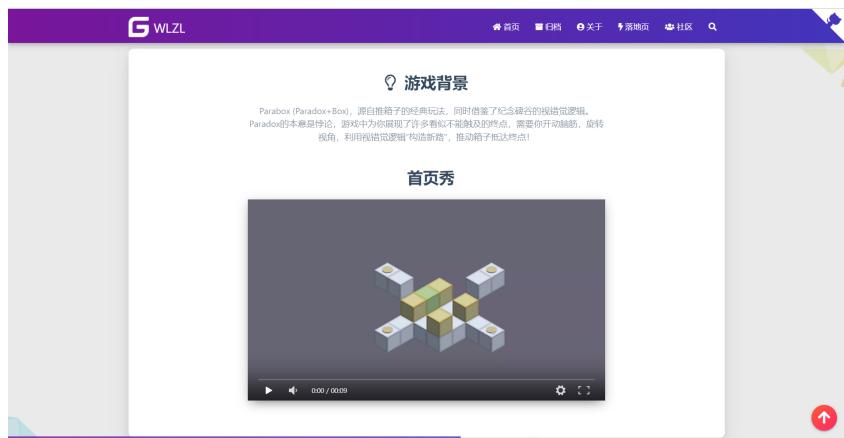


图 5.5: 游戏背景和首页秀

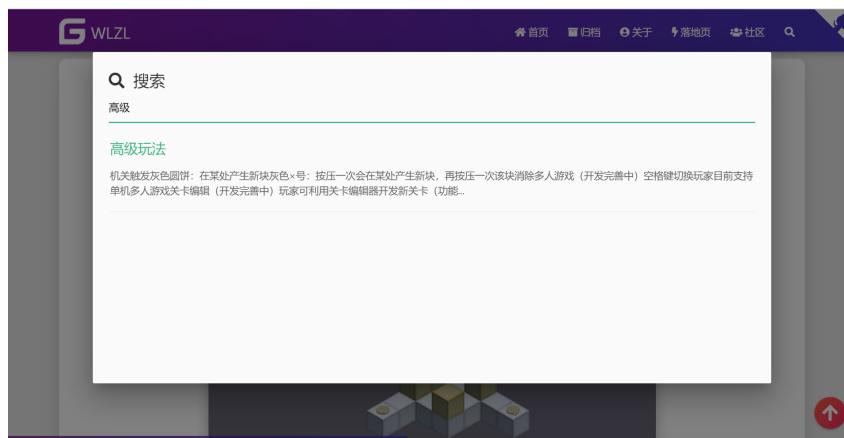


图 5.6: 关键词搜索功能

再往下滚动，是博客的卡片展示，这里卡片的背景图片是我们游戏的关卡图，并且在我们已经发布的两个博客中，卡片图片显示的都是最经典最让我们自豪的关卡设计图，此处也暗藏彩蛋，即当玩家在游戏中不知道如何通关时，如果他能再回头进入我们的官网，并看到这两幅图片，那么就会为他提供提示，告诉他如何通过这一关。

5.2.2 归档

归档界面主要展示的是我们的博客文章的发布日期和我们团队的发布动态。进入归档页面的一开始是仿照 github 动态方格所建立的一个动态图，方格代表着天数，颜色越绿表示在那一天我们发布的博客越多。这就可视化了我们团队对于游戏的关注度和发布动态，在向用户传达信息的同时鞭策自己开发。接着是以时间线的形式呈现我们在哪一年哪一天发布了什么文章，如图 5.7 所示。



图 5.7: 动态时间线展示

5.2.3 关于

在关于页面我们以动态的形式展现出我们团队的介绍，如图 5.8 所示。这里图片最底部紫色的线表示网页查看进度。越往下滑动，线会越长，当在页面最底部时，该线将占满一整条网页宽度的一条线。

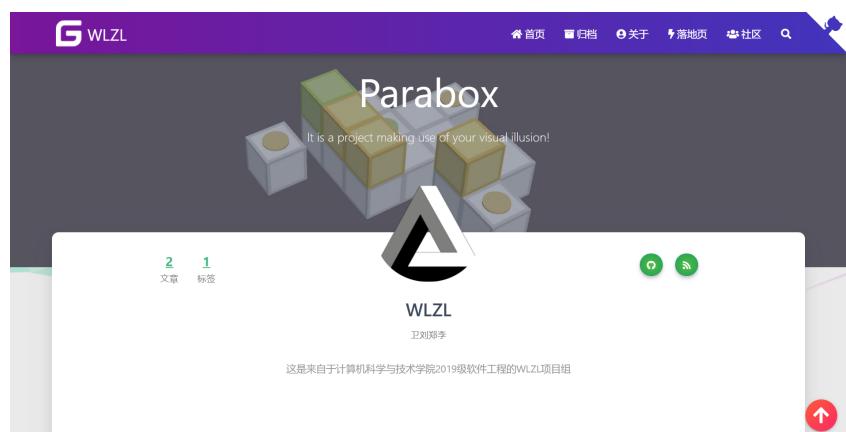


图 5.8: 关于页面

5.2.4 落地页

落地页，顾名思义就是当访客访问时第一个见到的页面，在营销领域中，落地页通常是一个区别于你的官网或其他页面的独立页面。落地页的目的十分的单纯，那就是营销转化。落地页是实现你内容营销策略的承载工具。简单来说，落地页是帮助你将访客转化为顾客的高效工具。通过使用落地页，你可以促成一笔交易，传播促销活动，提供信息并获取访客联系方式。我们产品的落地页如图 5.9 所示，可以看出其风格是与官网截然不同的，这也正是我故意设计的目的，多样化的对我们产品进行展示，从而始终带给浏览者新的体验。



图 5.9: 落地页展示

我们的落地页就只有长长的一个页面，有些类似于苹果官网上的发布页面，主要希望能让用户一览到底，并且如果有兴趣，可以继续深入我们的官网，因此该页面的所有链接都是跳往之前的官网的。这里不再具体展示图片，有兴趣可以点击该链接查看。

5.2.5 其他

除了搭建精美的页面之外，我还做了 SEO 优化。并向 Google 和 Baidu 两家搜索引擎提交了站点地图文件。如图 5.10 所示。

5.3 前端系统的测试

主要是对访问的测试，我们可以从图 5.11 所示，看出我们网站的访问量，这也侧面反映了我们游戏产品的欢迎程度。

这里的总访问量标准是用户请求网页的次数，而总访问人数指的是 IP 总数量，这里利用的是“计蒜客”API 接口，进行统计，可以看出在没有经过宣传，只是在院内传播的情况下，能有如此访问量也是相当不错的。



图 5.10: 谷歌站点抓取



图 5.11: 站点统计

由于搜索引擎更新较慢，以及没有投入经济做 SEO 的进一步优化，因此，站点仍然尚未收录进搜索引擎中，而且排名也较为靠后，目前来说，还是靠 gitee 本身自己的 SEO 来支持搜索，如图 5.12 所示，也将会在后续从网页本身的关键词，标题，描述等信息中加大信息量，从而提高 SEO 排名。



图 5.12: 谷歌搜索测试

可以说，虽然我们主打的是客户端的游戏，但是我们的产品发布页也不输于其他以 web 作为主体开发的软件工程项目，其中蕴含着大量的细节和彩蛋，等待用户发掘和探索，而我也在网页中留下了很多接口用来日后的开发完善。

6 用户反馈

6.1 内测用户反馈情况总结

在游戏程序正式发行之前，我们邀请了部分同学成为我们的内测用户，开展了内部测试。考虑到游戏的主要受众即为年轻人群体，在同学群体中开展内测较为有效。内测获得了数十条有效反馈，部分反馈的截图如图 6.1 所示。



图 6.1: 内测用户反馈截图 (部分)

对于网站部分，用户同样给予了反馈，首先是我队友，在我搭建的过程中，将每一步的变化都展现了出现，他们惊讶于一个静态网络能够从一个纯白背景到动态化加载，响应式网页的实现，也惊叹于网站本身优雅而又不显得过于花哨，既能言简意赅的表达网站本身的用意，又能适当的添加些色彩不至于网站显得枯燥。我们对收集到的反馈加以整理，获得如下反馈结果：

1. 优点：

- (a) 基于视错觉逻辑的游戏创意很好；
- (b) 关卡融入了解谜元素，富有挑战性；
- (c) 游戏关卡支持自定义，可玩性强；
- (d) 游戏画面简约耐看，物块移动十分顺手，过度动画效果好。

2. 不足：

- (a) 游戏难度跨越性较大，引导性欠佳，对于类似游戏接触较少的玩家而言较难上手；

- (b) 部分关卡的部分视角存在可能的视错觉歧义，一定程度影响了游戏体验；
- (c) UI 不完善，美工欠佳；
- (d) 缺乏对不同视角状态的提示，侧视角下物块移动方向不直观。

6.2 改进措施

根据 6.1 节的用户反馈情况，结合可操作性、耗时性等方面因素，对游戏进行如下改进：

1. 增加引导关卡：增加 4 关引导关卡，分别对游戏视错觉概念、基于视错觉的游戏套路与技巧进行分解引导，让游戏玩家在接触挑战性较强的关卡之前能够充分了解游戏特性；
2. 锁定部分关卡的部分视角：对于可能引起视错觉歧义的关卡的部分视角进行锁定处理，变相引导玩家使用正确的视角进行游戏。
3. 统一斜视角状态的 WSAD 操作逻辑：原游戏的操作逻辑并不会随视角的转换而改变，可能会出现视角转换 180 度之后操作逻辑相反的情况，影响了游戏体验。改进后的操作逻辑与用户视角保持一致，操作逻辑更符合直觉认知。

7 总结与体会

7.1 李涵

总结 在本次软件工程项目中，我负责项目全过程的文档整理工作和进度跟踪，并参与了软件模块整理与测试。

体会 在初期撰写立项和需求分析的 Wiki 文档时，我总感到仅使用自然语言对立项和需求分析的思路表达带来了许多不便。经过课堂上老师的讲解和线上题目的练习，我认识到这一过程的文档有许多方便的范式和模型来帮助我们梳理思路。例如在需求分析中，使用各种 UML 图可以从各个角度将系统的设计思想表达得更为完善。另一方面，绘制这些模型图的过程也引导我对系统一些重要的细节问题进行思考，如流程图就引导我梳理完善的游戏闭环过程。通过阅读课本，我掌握了更多需求设计过程中可以用到的规范化工具。这些已经成熟理论化的手段将软件的需求分析过程进行了有效的划分，并帮助我更方便、更规范化地与队友设计出了最终效果更好的需求模型。

由于我们团队采用了充分民主化的管理模式，并没有一个像经理或老师一样的角色对我们的任务进度进行强有力的监督，有效的进度划分与跟进手段就在我们的项目过程中起到了很大的作用。首先，我与队友一起，按照老师的要求对我们的项目任务进行细化并制作甘特图。这一过程不仅起到了进度安排的作用，还促使我们进一步细化了分工，并将项目的过程梳理地更加明确。在项目过程中维护的燃尽图帮助我将各个成员具体而琐碎的进度条目转化为直观的进度状态。根据实际曲线与理想曲线的相对关系还能清晰地看出当前进度是处于超前还是落后状态。在项目进行的同时还有繁重学业工作的情况下为我们的协同带来了很大的方便。

本次软件工程的项目过程还让我意识到了软件测试的重要性，也理解了课堂上有关软件测试“专业的事要交给专业的人来做”这句话。在关卡设计出后，最初的版本经过多次多角度的黑盒测试总能发现一些细节的问题，我们需要对问题进行整理，并结合白盒测试对产生的原因进行分析，开销远大于最初的设想。这也让我对容错性的概念产生了更多的思考。一方面，在结构的设计上要考虑到测试时更改的场景，具体手段就是充分地将软件模块化，并将模块之间充分地解耦合。另一方面，在项目的进度安排上也要留下容错空间，以在一个环节耗时超过预期时仍能游刃有余。

软件工程涉及软件开发过程的各个环节，这一项目对我而言不止于专业知识，在整理规划和协调沟通等方面也给我带来了全面的历练。在今后，我仍将在一个个实际项目之中提升自我。软件工程的学习与实践，才刚刚开始。

7.2 刘安珉

总结 在产品从提出构想，调研，设计，编程，测试，发布这一系列过程阶段中，我于前期担任“产品经理”，开发过程中我又化身“程序员开发者”来为本次项目添砖添瓦。

在产品前期，我主动参与了市场调研，首先了解了当下社会存在的一些用户需求，从中小学生需要创新力和空间想象力和游戏本身的娱乐属性出发来试图挖掘出其中的市场需求，即人们需要一款能够带来空间想象力和灵活思维的创新型游戏。其次从一个产品经理的身份为我们这款游戏（产品）寻找一个好的市场定位。我广泛查阅了相关的资料，最终从前瞻产业研究院的《2021 年中国游戏行业市场规模及细分市场分析》的分析报告中，看到当下游戏市场的前景和未来，因此确信开发这样一款游戏的可行性。

而在确定了我们要开发游戏的原型后，我又参与了产品的阐述部分，即该游戏的各种优点以及之后可以优化改进的方向。同时我又将本产品与市场上出现的竞品比如《纪念碑谷》进行了对比，突出我们游戏推崇的是极致简洁，主打的是利用碎片化时间和烧脑的解决方案，并以社区化（待完善的功能）和高度自定义化作为我们产品的特色，通过立足这几点来为我们产品打开市场。

接着我还提出了产品推广计划。做到真正理论上产品的完备性。我在总结上述的特色后提出我们要“为您的休闲插上想象的翅膀”这一价值主张，并定下了卖点是 DIY，高频低时长，社交与互动。同时进一步确定了产品的定价，产品的渠道，产品的促销策略，产品上市推广的风险和对策，甚至包括产品的预算。

前期这一系列的步骤里面，我尽量利用软工课上学到的知识和之前学过管理学和产品经理课程相关知识，最大限度地模拟真实地产品发布环境，并最终形成了《新产品开发报告》融合进了 gitee 仓库里的 wiki 中，形成了需求工程整个篇章和原型系统部分的一些内容。

而在产品开发的过程中，我主动学习 Unity 的使用和 C# 语言的编写，试图编写主界面场景，但在开发后期由于临时收到任务要搭建一个产品发布网站，再加上为了保证游戏项目风格的统一性，我自己设计的主界面场景相对于游戏本身的风格特色有些突兀，因此我从一个“C# 开发者”转变成一个“前端工程师”。主要使用语言栈也从 C# 变成了前端三件套（HTML、CSS、JS）。

在开发产品发布页的过程中，我也进行了设计，并最终加以实现。总的来说是一个主页，一个归档页，一个落地页以及一个介绍页，最后还有未完成的社区页（由于 gitee pages 是静态的，因此暂不支持社区等具有注册功能的动态功能）。我最终的网站是一个响应式网站，会根据设备的不同呈现出不同的布局，便于保持各类用户设备访问的流畅性和优美性。并且为了方便统计信息和检查游戏流行度，在页脚处统计了网站的访问量和访问人数。真正做到了一个产品发布页应该做到的和未曾做到的事情。此外，我还进行了 SEO 优化，已经向 Google 和 Baidu 两家搜索引擎提交了站点信息。

网页开发完毕后，我又完成了游戏的教程和封装以及发行版的发行，发表两篇教程博客在产品发布页，并把队友的项目文件封装成一个 exe，使得用户可以做到真正的下载即玩。而对于发行版的相关规定，我学习参照了语义化版本管理，将发行版本确立为 1.0.0，代表着项目最终的第一代截止与完成。

体会 本次项目我最大的体会是感受到了一个产品从提出构想，调研，设计，编程，测试，发布这一系列过程中的辛苦，细致谨慎和担忧与喜悦，经过这一系列的开发，我真正体会到了产品的完成是有多么不容易。

在项目进行过程中我遇到了很多困难，包括如何进行市场调研，如何搜寻市场数据和有关资料，如何从 0 开始 C# 编程，如何使用 Unity，如何开发网站，如何实现响应式和改版个性化 hexo 网站等，熬了很多个夜，也解决了很多个 Bug，学到了很多的东西。

首先，在团队协作完成项目过程中，我学会了合作开发项目的一些方法，体会到了团队协作中工作分配，开发流程，开发架构等一系列的问题和知识，从课堂上和搜索引擎中也学到很多的知识，如何能够更加高效的快速敏捷开发一个完整的产品也是我今后要继续学习的一个领域。

其次，在开发学习过程中，由于我在此之前从来没有接触过游戏相关的任何技术，这次项目也是从 0 开始，我看完了 C# 的一本书，用 C# 来编写一些 Script，也在 Unity 官网上学习到了 Unity 的使用方法，从能成功跑通一个现有项目开始到自己能开发出一个完整的场景。我遇到了很多 bug，也体会到了 Unity 新旧版本不兼容的问题。这也让我学到许多。

接着，对于网页开发，由于我之前有 web 开发相关经验，又在搭建 gitee pages 时借助了 hexo 框架，我深知一个好的框架和好的工具能够节省很多时间。然而我在网页开发最大的工作量是 hexo 框架的修改和定制化，而非 hexo 框架的搭建和环境的配置，如果进入我的页面，其实很难看出来这是 hexo 搭建的，因为我在此基础上魔改了很多东西，最终形成有我们产品特色的一套网站体系。

最后，我学到了除了编程代码之外的一些东西，比如进度计划表示的图，系统结构图，类图等等一些可视化协助软件开发的图表。还有如何将软件拆分成合理的模块和字体，使得多人合作更加顺畅和合理。这些方面的技巧会在我之后的学习生活中具有重要的意义和作用。

7.3 卫云泽

总结 在项目中，我参与了产品的策划、项目进度安排等内容；在产品开发阶段，我主要负责关卡编辑器模块与关卡信息存取逻辑的编写；在产品的测试发行阶段我参与了关卡的设计、测试、反馈，并基于团队工作形成 PPT，参与了课堂的项目汇报展示。

体会 本次软件工程课程项目是我本人的一次非常珍贵的团队协作开发软件的项目经历。从项目的立项、策划，到软件的编写、测试，再到最后的软件发布的整个过程中，我逐渐认识到软件工程的复杂性与重要性，也深刻意识到团队协作的软件开发与个人软件开发的异同。

在项目立项阶段，我提出从游戏类和非游戏类两方面出发，以同学日常需求为初衷开展头脑风暴，在与项目组成员的激烈讨论中我们确定了两个候选项目课题：非游戏类的二手市场交易平台和游戏类的基于视错觉的推箱子小游戏。结合万老师的意见与创新性，我们最终决定以推箱子小游戏作为我们的软件工程课题。

考虑到我们的创意是视错觉逻辑，而视错觉逻辑的经典游戏便是纪念碑谷。结合软件工程项目实际市场调研的缺乏，我主张从“学习”出发，与团队成员研究了纪念碑谷游戏的视错觉关键逻辑，形成了对游戏项目实现方案的雏形；同时基于 NABCD 模型给予了我们的游戏项目一个初步的定位。

好的课题与创意需要实实在在的落实，而如何与团队成员开展密切协作正是软件工程课程学习的关键。在结束了分工与工作量拆分之后，为了更好地掌控项目进度，我了解了项目组各位成员的意愿，形成了项目甘特图，为项目的开展安排了细致的“时间坐标”。

在产品开发阶段，我遇到的第一个难题是开发工具与语言的不熟悉——由于先前缺乏游戏开发相关经验，Unity 和 C# 语言对我而言是全新的事物，在产品开发前期的学习时间成本较高，进度较为缓慢。而到了产品开发的中期，由于基本掌握了 Unity 的基础用法和 C# 语言的特性，游戏开发速度得到了显著提升，此时面临的第二个问题是与项目组成员的协作问题。通过和项目组相关部分同学的交流，我们约定了命名空间、统一了函数接口等规范，在项目开发中也开展了多次联合调试，项目整体进度获得了飞跃。

在产品开发的最后阶段，团队成员多次相约共同开发，面对面的交流提高了程序开发的效率，“开会”一词也从“形式”转变为“刚需”，我也充分认识到了“目标明确”是高效会议的前提。

在软件测试与发行阶段，我参与了软件的评估和对用户反馈结果的分析，并基于用户反馈内容和自身使用体验提出了较有建设性的改进建议，如增加引导关卡等；同时我

也对软件整体细节进行了一定把控，如游戏初始界面、游戏提示等。为了让游戏玩家能够更快入门，我将游戏的基础玩法和高级玩法分解录制为 GIF 动画，由项目组成员发布在游戏官网上，作为游戏的新手教程。

在制作软件发布 PPT、准备软件发布报告时，我回顾了这两个月团队成员的协作与努力。尽管我们的协作模式是稚嫩的，成员之间的协作并没有达到非常熟练而默契的程度，有时也会因观点的分歧而发生“争执”，但每位同学都是努力的，目标是统一而一致的，也正是这一股子劲儿让我们最终找到了团队协作的感觉，享受了一把“众人拾柴火焰高”的乐趣。诚然，我们并没有实现最初设计的全部游戏功能，多人游戏及游戏社区等功能也处于开发阶段，但我们最终形成的发行版本已经具备了游戏的完整功能，可以带来较好的游戏体验，我们视自己的产品如自己的孩子一般珍视它，也迫不及待地想要宣传这款游戏，让更多的人能够玩到我们的创意。

我相信，这次软件工程课程项目只是我们团队开发经历的一个开端，在课程中我学会的软件开发流程和团队协作技巧会让我受益良久。

7.4 郑欣

总结 本次软件工程项目中我担任的是游戏的策划和主程序，负责了游戏核心逻辑的编写与测试和部分关卡的设计，共编写代码约 2400 行。在结课时，软件的功能基本完善，已经能够进行完整的游戏流程，并由于其优秀的可玩性与创意在玩家之间广受好评。不足之处是由于时间关系，引导关卡的设计略有欠缺，并且由于团队中没有美术，游戏画面较为朴素。

体会 这个游戏源于我在大一时的一个 Idea，但当时受限于编程能力与图形学等方面的知识无法实现。本次的项目让我感受到了自己在三年内水平的提高，同时也感受到了与同组同学合作共同完成一个项目的喜悦。

本次项目中我在技术上遇到过许多困难，其中最大的挑战就是玩家移动的动画。一开始当玩家的起点和终点不在同一平面上时，我是将玩家移动到较高处。这样在绝大多数情况下动画都不会出现问题，但有一种情况遮挡仍然会出现错误。仔细分析后我发现这种情况下如果还是将玩家当成一个整体来移动不可能达到想要的效果。因此我想到可以将玩家切成两半，让玩家只在自己所在的两个方格内运动，多余的部分直接在渲染器中裁剪掉。这种效果需要采用 shader 实现。

由于之前没有接触过 shader 且从零开始学习 shader 的成本太高，我决定先在网上搜索类似的效果。经过搜索我发现 Fenglele Fans 的博客中实现了类似的裁剪效果，但是没有添加光照。因此我又从另一篇博客中找到了有基本光照功能的代码进行了合并从而满足了制作动画的基本需求。后续添加高亮效果同样需要用到 shader，但由于已经有

了编写 shader 的基础因此实现起来比较轻松。

实现正交与透视之间的切换对我来说同样是一个挑战。我一开始想到的方案是用透视和正交两个摄影机，当需要从透视切换到正交时可以将透视摄影机拉远并放大，然后切换到正交相机。但这种实现方法不仅代码量很大，而且在切换摄影机过程中会产生一个细微的突变，因此最后我决定用一个摄影机通过修改投影矩阵实现这个效果。

投影矩阵的推导对我来说是比较困难的，主要是因为需要自己找到一个合适的参数使得视野看上去在匀速变化。一开始我尝试过使用已有的透视投影矩阵和正交投影矩阵进行插值，但是这样做有两个缺点：一是透视率变化不是线性的，在接近正交的时候视野的变化率十分大；而是无法统一两个视野的大小，在正交的状态下使用相同的参数显示的物体会略大于透视。后来我想到可以对透视矩阵稍作修改，让投影中心不在相机上而在相机后方的某个位置，让视角匀速变化，最终实现了透视与正交之间的平滑过渡。

除此之外，本次项目还有一个难点是实现视错觉逻辑。投影在摄影机上的点必须相邻这个必要条件比较容易想到，但难点是除了这个条件还有很多特判。并且由于视错觉是一个很抽象的东西，没有一般的公式可以推导，因此这一块在实际开发的过程中出了很多 Bug，有一些无法实现或有歧义的情况还需要在关卡设计中做一些约定。尽管最后这个问题还没有一个尽善尽美的答案，但经过玩家测试在现有的关卡中这一部分还没有出现任何问题。

总的来说，通过这次项目开发我体会了一个小型独立游戏开发的基本流程与其面临的挑战，收获颇丰。但由于开课时间较晚并且跟很多其他课程的课设和考试时间冲突，实际有效的时间只有开始的一个多星期和结课前几天。同时，由于同组成员没有 Unity 开发经验，上手这个项目花费了较长的时间。因此最后的软件还有一些原本想要实现但由于时间关系没有完成或舍弃了的功能，例如关卡编辑器、设置界面和联机功能。不过总体来说能在有限的开发时间内达到现在的这个效果让我感到十分满意。