**软件工程综合实验**

**基于Unity引擎的沙盒模拟经营游戏开发**

**设计报告**

**项目负责人：陈佳皓 报告（主）撰写：丁一萌**

**1.引言**

**1.1标识**

文档名：基于Unity引擎的沙盒模拟经营游戏开发-设计报告

项目名称：基于Unity引擎的沙盒模拟经营游戏开发

**1.2系统概述**

本项目基于Unity引擎开发一个2D沙盒模拟经营游戏，玩家采用上帝视角、在一个二维方块划分的地图上对村庄建设进行操控，主要发展种植业以及相关配套产业让村庄不断富裕。基础游戏循环在于种植->收获->出售产品/建造。可扩展功能包括随机灾害等事件对玩家进行干扰等。

本设计报告覆盖范围为游戏的核心功能模块设计和小部分扩展功能。

**1.3文档概述**

文档包含以下内容章节：

-引言（本章节）：标识，系统概述，文档概述，引用文件，关于撰写

-总体设计：设计目标，系统架构和运行环境，架构模式，模块划分，模块间接口设计

-详细设计：各模块详细设计，其他设计（如美术）

-附录：设计相关图示

**1.4引用文件**

《基于Unity引擎的沙盒模拟经营游戏开发-需求分析报告》

**1.5 关于撰写**

本报告第三部分：详细设计，其中具体设计工作有分工，故分工撰写：

* 陈佳皓：3.1，3.4，3.7.1，3.7.2，3.8
* 李宇清：3.2
* 刘瑜亮：3.9，3.7.3
* 丁一萌：3.3，3.5，3.6，3.10，3.11

格式修订和其他部分由主撰写人负责。

**2.总体设计**

**2.1设计目标**

设计的关键目标在于完成一个模块划分清晰且易于扩展的游戏Demo，以便于合作开发核心功能和进行后续扩展。

**2.2系统架构和运行环境**

软件架构：简单的单机客户端架构

Unity版本：2022.3 LTS（//TODO：？）

目标平台：Windows PC

**2.3架构模式**

项目采用单例模式进行架构，适应本项目规模（较小）且兼顾开发组人力限制（合作开发，易于上手）

**2.4模块划分**

- 地图系统：进行地图生成和管理

- 人力系统：实现小人接收/执行玩家指令并与其他部分交互完成指令的逻辑

- 物品系统：进行物品种类、基本属性和同类别物品共同逻辑的管理

- 建筑系统：进行建筑类型、基本属性以及同类别建筑共同逻辑的管理，以及进行人造结构的建设和拆除工作的指派

- 作物系统：进行作物的种类、基本属性以及同类别作物共同逻辑的管理

- 实体管理系统：物品/建筑/作物实体个体的管理

- 交易系统

- 用户界面系统：实现用户通过UI界面的交互，包括查看信息和发布指令

- 数据存储系统

- 时间模块：用于控制游戏的宏观时间流逝。提供倍速以及游戏时间暂停等功能。

**2.5模块间接口设计**

后端全局管理类关键模块之间通过单例访问点交互，综合利用组件化架构对象，实现层级结构。对于受保护的private数据，提供对外的封装函数获取其具体数值。尽可能保证模块内功能集中，从而减少逻辑运算时模块间的调用，增加聚合度，减少耦合度。

**3.详细设计**

**3.1地图系统**

**3.1.1模块概述**

地图设计为由若干等大小正方形地块拼接而成的矩形，具有固定算法，每次运行游戏时由算法自动生成其地形。

**3.1.2类/对象设计**

地图数据存储的核心为MapData类与LandformRange类。前者会存储该地格坐标以及若干交互信息，如信号属性，用以记录其是否可发生某种类型交互，譬如水地上无法建造，岩石地上无法种植。亦会存储若干成员变量，增加玩家交互丰富性，如维护地块的通行参数，影响小人的行进速度，维护地块的肥沃度，影响作物的生长速度等。后者是组成地形生成算法中地形值-地形映射表的主要类，用以构成地形自动生成中的重要环节。

全张地图的数据集中储存在mapDatas[,]数组中，与地格坐标一一对应，便于通过坐标进行快速访问。

**3.1.3关键算法/流程**

生成地形的主要算法为柏林噪声算法，以期生成相对接近现实的连绵起伏的地形。首先会依据坐标采用算法为每个地块生成各自的地形值，之后将其规范化至[0,1]的取值范围，提前划分好各地形与取值范围之间的映射表，譬如将地形值视作海拔，取值低处设定为水洼与泥土，取值高处设定为岩石地，并根据映射表为地块附着特定参数与资产等，从而实现地形的自动算法生成。

**3.1.4Unity相关设计**

在地形的贴图表示上，由于涉及了大量重复贴图，并且排布上整齐排列，故可以采用Unity的Tilemap系统，结合逻辑数据进行渲染，可以显著节约性能。

**3.2人力系统**

**3.2.1模块概述**

该模块负责管理游戏中的可操控小人pawn，为他们的任务执行提供相应的运行处理函数供玩家操控时调用。

**3.2.2类/对象设计**

**pawn类设计：**

|  |  |
| --- | --- |
| id | 唯一标识小人，生成时在原有基础上+1 |
| isontask | 标志小人是否在工作中，对taskmanager中的任务分配工作起指导作用 |
| handlingTask | pawn目前正在做的任务 |
| movespeed | 移动速度 |
| workspeed | 工作速度 |
| capacity | 运载容量 |
| instantCapacity | 当前剩余的运载容量 |
| materialId | 运载的材料ID |
| materialAmount | 运载的材料数量 |
| materialType | 运载的材料类型 |
| handlingTool | 小人所手持的工具 |
| PawntaskList | 小人自己的任务列表，用于支持小人的寻路任务以及批量种植任务。在后续的拓展中，也可以作为玩家给小人分配的任务队列使用 |

**controller属性设计：（该部分主要用于控制小人移动）**

|  |  |
| --- | --- |
| IsMoving | 是否移动 |
| isSegMoving | 是否进行分段移动（A\*寻路算法使用） |
| pawn | 关联的pawn |
| targetPosition | 移动最终目标地点 |
| moveFinished | 移动是否结束 |
| targetCellPos | 目标格子位置 |
| fromCellPos | 起始地点格子位置 |
| currentTargetCellPos | 当前目标格子位置（A\*寻路使用） |
| landTilemap | 地块点阵图（A\*寻路使用） |

**3.2.3关键数据结构**

Pawn以Pawntasklist维护每个小人单独的任务列表，在A\*寻路将总体移动目标拆分成多段移动任务时以及统一携带种子后分别发布种植任务进行多地块种植时发挥作用

**3.2.4关键算法/流程**

在处理种植、建造等不同任务时统一调用Handletask函数对当前小人的handlingtask进行处理，该处理函数会查表taskHandler和buildTaskHandler从而调用不同的任务处理函数尽心任务运行。在任务运行结束后执行resolvetask对一些数据进行重置以及部分任务的末尾处理。

在具体到某种类型的任务时，基本采用协程方式管理程序执行时机，有效避免了采用sleep会导致游戏整体等待的问题。

对任务的具体处理采用层层递进的方式，以实现移动任务为基石，加之装载卸载物品任务的实现，基础运输以及运输到蓝图两种运输形式以分别调用两次移动任务及装载卸载任务得以实现。在更高层次的建造任务中，通过蓝图接口不断获取需求，采用栈式结构创建并执行一系列运输任务，当蓝图需求填满后执行建造。

移动部分采用A\*寻路算法，维护一个是否可通行的地块点阵，依据是否可通行情况给出一系列拆分移动任务，从而实现对于不可移动区域的规避。

**3.3物品系统**

**3.3.1模块概述**

该模块以种类为单位管理物品，负责完成多个同种物品实例共同信息的存储，并接受来自其他系统的关于某种物品种类特性查询。

**3.3.2类/对象设计**

一个物品种类的数据存储于一个Material/Tool/Resource对象内（均为Item子类）

|  |  |
| --- | --- |
| 类名 | 属性 |
| Item | ID，名称，细分类和材质 |
| Material：Item | 可种植性 |
| （扩展）Tool：Item | 最大耐久，提供的强化属性及其强化值 |
| （扩展）Resource：Item | 最大耐久 |

**3.3.3关键数据结构**

Dictionary<ItemType, List<Item>> itemLists，存储所有物品信息

Dictionary<int,Item> itemDict = new Dictionary<int, Item>()，物品种类id向Item对象的映射，负责提供向itemList的快速查询，查询需持ID。

**3.3.4类方法接口**

Material类向外提供关于其是否为种子的接口，以及若为种子提供其可种植的作物种类ID的接口。

预期设计：使用一个专属于Material类的属性can\_plant\_crop（可种植性）记载其可种植的作物种类ID，若该Material不可种植则设置为一个在正常Crop种类Id之外的特殊值。此规则应当被类的方法封装：

|  |  |
| --- | --- |
| 接口 | 功能 |
| Material::IsSeed() | 返回是否为某种作物的种子 |
| Material::GetPlantCropId() | 返回该种子型物品可种植的作物种类的ID，若该物品不可被种植则返回-1表示非法值 |

**3.4建筑系统**

**3.4.1模块概述**

玩家与地格产生交互的主要方式是指定小人在地格上建筑，且建筑按照用途分类，建成之后可以实现改变地格属性，与Pawn交互等功能。

**3.4.2类/对象设计**

建筑的逻辑类为Building类，记录建筑的若干数据，建筑的实例类为BuildingInstance类，作为ItemInstance的子类由ItemInstanceManager统一管理。

**3.4.3关键算法/流程**

在玩家指令在某地格建筑后，会先在此地产生一个蓝图实例PrintInstance，并发布任务至TaskManager中的全局任务列表，随后TaskManager会在每一帧寻找空闲的Pawn并将此任务指派给它，Pawn会收到该建筑所需的材料列表并去地图上寻找相应材料，将其运输到蓝图处，直至材料筹备完全后开始建筑，并在建筑完毕时将PrintInstance转化为对应的BuildingInstance

**3.5作物系统**

**3.5.1模块概述**

该模块以种类为单位管理作物，负责完成多个同种作物实例共同信息的存储，并接受来自其他系统的关于某种作物种类特性查询，并对某种。查询需持ID。

**3.5.2类/对象设计**

（1）Crop类

一个作物种类的特性数据存储在一个Crop类对象中，包括：

- ID，名称

- 其种子的物品种类ID

- 总生命周期

- 湿度/光照/土壤肥度的标准值以及生长速度受其影响的模式（作用见（3.5.4））

（2）Factor类

Factor类用于抽象规范化作物的生长耗时和生长效率计算。

同子类的Factor具有相同的计算公式，同时保存自身独立的计算参数，

**3.5.3关键功能逻辑**

我们将作物实体生长消耗的总游戏时长的扰动因素换成两部分：

①种植时刻的农田的环境因子（fertility, humidity, light）与植物需求的适配度。

（引见3.5.2湿度/光照/土壤肥度的标准值以及受其影响的模式）

- 标准值对应标准生长速度

- 存在离标准值差距越小越优的模式和实际值与生长速度成正比的模式

- 各因素的对生长速度倍率的影响按乘算结合

②生长过程中可能会受到的全局灾害和增益，这些灾害和增益对不同种类的作物可能具有不同影响

那么我们将①绑定到作物实体生长所需总生长值realLifeTime上进行实现，将②绑定到实体单个固定帧生长值的增值growthPerFrame上进行实现。

（注：作物实体将相应地设置属性real\_lifetime作为其独立信息；并且设置属性growth\_per\_frame作为共有信息的缓存）

作物系统将负责使用相应的公式计算这两个值；前者Factor写死（植物特性），后者Factor可变（环境变化、全局buff等）

对于前者，作物系统提供

- 一个接口，输入作物类型id和环境数据，返回相应的realLifeTime

由于后者可变，作物系统需要提供：

- 一个接口供正在生长的作物实体获取当前的本种类作物的growthPerFrame。

- 各种可以扰动的环境因子（pest disaster, global buff等）修改growthPerFrame的接口。

**3.5.4关键数据结构与相应接口**

（1）List<Crop> cropList

可序列化的数据结构，保存所有作物种类对应的Crop对象，是（2）（3）中数据结构的初始化基础。不对外开放。

（2）为完成3.5.3中的功能，作物系统应在运行时维持以下数据结构：

// For growth per frame

Dictionary<int, LinearFactor> pestDisasterEnvFactorDict：

Dictionary<int, float> growthPerFrameDict

// …(可扩展)

// For real lifetime

Dictionary<int, EnvFactor> fertilityFactorDict

Dictionary<int, EnvFactor> humidityFactorDict

Dictionary<int, EnvFactor> lightFactorDict

// …(可扩展)

对外提供的接口见3.5.3

（3）Dictionary<int, Crop> cropDict

按作物种类ID快速查询作物种类信息的基础，相应接口为：

**public** Crop GetCrop(int id)

（4）Sprite[] CropSprites

按作物种类ID快速进行贴图查询的基础，贴图排列顺序如：

Sprite0（crop\_id=0，stage=0）

Sprite1（crop\_id=0，stage=1）

Sprite2（crop\_id=0，stage=2）

Sprite3（crop\_id=0，stage=3）

Sprite4（crop\_id=1，stage=0）

Sprite5（crop\_id=1，stage=1）

…

作物在生长过程中常根据实时状况切换贴图，故作物系统提供按作物种类Id和生长阶段编号快速查询贴图数据的功能，相应接口为：

**public** Sprite GetSprite(int crop\_id, int growth\_stage)

**3.5.6 Unity相关设计**

Crop类继承自Unity的ScriptableObject类，可以作为ScriptableObject以.asset形式文件静态储存或加载

**3.6实体管理系统**

**3.6.1模块概述**

该模块负责管理物品/建筑/作物的实体，基于避免冗余信息存储和提取物品/建筑/作物实体共性的思路进行设计，将各种实体可以产生的交互封装实现为对外的接口。

**3.6.2类/对象设计**

本模块的对象包含的信息主要为单个实体所独立具有的信息，同时兼顾实体状态的更新效率，在对高频使用的共同数字进行缓存

1. ItemInstance及其子类

一个ItemInstance实例代表一个物品/建筑/作物实体，具有属性包括：

* **实体ID**，实体类型，实体所属物品/建筑/作物种类ID
* 实体所在的地图格在整个地图中的坐标（整数）
* **用于将实体显示到UI中进行UI交互的GameObject**

ItemInstance具有以下子类型：

|  |  |
| --- | --- |
| 子类型名 | 特有属性 |
| ToolInstance | 当前耐久度 |
| MaterialInstance | 数量 |
| CropInstance | 当前生长值，所需生长总值，每固定帧生长值（缓存），收获物列表（缓存） |
| BuildingInstance | 当前耐久度，建筑材料列表（缓存），\*内容物 |
| PrintInstance | 蓝图所需的各项建筑材料id及其对应的<已收集数量，所需总数量> |
| ResourceInstance | 当前耐久度 |

（2）InsLabel及其子类

物品类型实体被存储或被小人携带时的特殊形态，保存实体独有的信息以及其类型信息，以便于发挥其在该形态下的功能，并为再次实体化提供信息。

一个InsLabel实例具有属性包括：

* 原实体类型，原实体，所属物品/建筑/作物种类ID
* 该标签的可合并性

子类型MaterialLabel 具有属性包括：数量

（扩展）子类型ToolLabel 具有属性包括：当前耐久度

**3.6.3 关键功能-生成实体**

生成各类型实体需要完成的工作有

- 共同工作：分配实体ID，设置其对应的GameObejct，将实体加入实体管理系统的全部实体列表，将实体设置到地图系统对应的地块中

- 各类型实体专有工作

|  |  |
| --- | --- |
| 实体类型名 | 销毁时专有工作 |
| ToolInstance | 设置当前耐久度为该种类最大耐久 |
| MaterialInstance | 设置物品数量 |
| CropInstance | 设置作物所需生长总值，初始化当前生长值为0，初始化每固定帧生长值的缓存，初始化掉收获物列表的缓存 |
| BuildingInstance | 设置当前耐久度，初始化建筑材料列表的缓存，初始化内容物为空 |
| PrintInstance | 初始化蓝图所需的各项建筑材料id及其对应的<已收集数量，所需总数量>列表，其中已收集数量初始化为0. |
| ResourceInstance | 设置当前耐久度 |

- 此外，各类型实体初始化时，所需的物品/建筑/作物种类的信息分别来源于物品系统/建筑系统/作物系统

故设计以下调用结构：

*单一SpawnItem函数*

*（完成不需物品/建筑/作物种类的信息的共同工作）*

*→多个Make<实体类型名>函数*

*（完成各类型实体专有工作）*

*→单一InitInstance函数*

*（完成需要物品/建筑/作物种类的信息的Gameobject初始化）*

**3.6.4 关键功能-销毁实体**

与SpawnItem做类似的考虑，其中各类型实体专有工作如下：

|  |  |
| --- | --- |
| 实体类型名 | 销毁时专有工作 |
| CropInstance | 可能掉落（部分）收获物 |
| BuildingInstance | 可能掉落（部分）已用于建造的建筑材料，并需要处理可能存在于建筑内的内容物 |
| PrintInstance | 可能掉落（部分）已收集的建筑材料 |
| ResourceInstance | 可能掉落（部分）物品 |

且游戏交互中存在不同类型的引发销毁的操作。

故我们设计枚举类型DestroyMode配合以下调用结构：

*单一DestroyItem函数*

*（完成共同工作）*

*→多个Clear<实体类型名>函数*

*（完成各类型实体专有工作）*

*→单一DetroyInstance函数*

*（完成销毁GameObject的共同工作）*

**3.6.4作物实体特性-作物生长**

设置函数UpdateAllCropInstance，并在继承自MonoBehaviour的函数FixUpdate中调用，从而完成作物生长机制的实际执行。

每固定帧使所有未成熟的作物实体的生长值增加。增加值为：

*crop\_instance.growth\_per\_frame \* 时间系统的timeScale*

并计算生长阶段。检查其贴图是否符合生长阶段，如不符合更新贴图。

（扩展：可类似地设计并实现建筑实体的耐久度消耗）

**3.6.5 实体交互接口**

|  |  |
| --- | --- |
| 类型名 | 特殊接口 |
| ToolInstance | 获取强化项 |
| \*ToolLabel | （装配到小人身上后）发生磨损 |
| MaterialInstance | 修改和获取数量 |
| \*MaterialLabel | 修改和获取数量，查询是否为种子并被种植 |
| CropInstance | 被收获或被毁坏 |
| PrintInstance | 获取材料需求，添加材料到其已准备的材料中 |
| ResourceInstance | 被采集 |

**3.6.6 Unity相关设计**

为了满足UI上的功能，我们做关于Unity的特殊设计，如

- 为Item设置类方法GetText和SetText，用于获取和修改其GameObject的文本组件子对象，使Material的数量显示能随其改变而变化。（并供后续其他使用）

- 将每个实体拥有的GameObject的name属性设置为该实体的ID，使UI的鼠标点击操作选中的GameObject能够反向识别到拥有它的实体

**3.6.7（扩展）其他-实体ID管理**

当前实体总规模较小，ID可直接增量不重复分配；为维持更长时间/更大规模的游戏运行，可考虑设计ID管理器来在生成/销毁实体时进行实体ID的分派和回收。

应当为此留下GetNewId和RecycleId的接口，若将来完成ID管理器，可以将这两个接口的实现方式更改为调用一个实体管理系统的私有ID管理器。

**3.7交易系统**

**3.7.1模块概述**

主要设计一种渠道使得玩家可以以多余材料换取通用货币再换取其它材料，从而实现部分的资源再分配。

**3.7.2关键数据结构**

tradergoods以int数值对记录当前商人商品列表的物品id与存货，商人商品可在特定时间点进行刷新。 而itemHistory与itemWeight分别记录了物品刷新历史与物品刷新概率的权重，通过算法设计尽可能减少商品的重复刷新。

**3.7.3关键算法/流程**

TraderManager 在关键流程中，采用单例模式确保全局唯一实例；每日通过概率递增机制刷新商人，防止其长期缺席；商品生成采用加权随机抽取算法，确保多样性和稀有度控制，并避免重复出现；同时记录商品历史，为后续权重调整和数据分析提供支持。商品生成后自动更新 UI，降低维护成本。整体方案兼顾随机性与玩家体验，易于拓展。

**3.8用户界面设计**

**3.8.1模块概述**

主要控制与用户发生交互的图形化界面的排布和显示。

**3.8.2关键数据结构**

selectedObjectPanel用以显示当前选中物品的详细信息

buildingMenuBars用以唤出指定种类建筑菜单

buildingMenuSquares用以展示建筑细则

instructMenuSquares用以展示可进行操作细则

traderMenuPanel与DebugMenuPanel用以在特定情况下展示内容，由UIManager管理显示与隐藏状态。

**3.8.3关键算法/流程**

ScrollContentController脚本会捕捉用户鼠标滚轮操作，并根据滚轮上下相应对地图内容进行缩放。MouseInteractManager会以多态状态机记录用户鼠标目前操作状态，选中对象等细则，并支持用户与对象之间的交互。

**3.9数据存储设计**

**3.9.1 模块概述**

数据存储模块（SLManager）旨在实现游戏运行过程中动态数据的持久化保存与读取，确保游戏状态在玩家退出或中断后能够被准确恢复。模块主要负责存储和恢复如物品实例（Tool、Material、Crop、Building、Resource 等）和作物管理等核心运行数据，采用 JSON 序列化方式写入本地文件系统，兼顾跨平台的安全性与可读性。

**3.9.2 类/对象设计**

本模块核心类为 SLManager，采用单例模式管理，在游戏启动时初始化并常驻内存。其内部设计多个用于序列化的数据结构，用于描述各类实例的状态信息。以物品实例的保存为例，ItemInstanceData 类结构包含了所有类型实例所需存储的字段，包括但不限于：位置、耐久度、数量、生长状态、建材清单等。

此外，自定义辅助结构如 SerializableVector3Int 用于解决 Vector3Int 在 JSON 序列化中的兼容性问题，PrintMaterialProgress 用于描述打印类建筑的建材收集状态。

**3.9.3 关键数据结构**

以 ItemInstanceData 为例，其主要结构如下：

public class ItemInstanceData {

public int id;

public string type; // 实例类型，如 "Tool", "Crop" 等

public int item\_id;

public SerializableVector3Int position;

public int durability;

public int amount;

public float growth, real\_lifetime, growth\_per\_frame;

public List<KeyValuePair<int, int>> harvest\_list, material\_list;

public int content\_id;

public List<PrintMaterialProgress> print\_material\_list;

public int resource\_durability;

}

每个字段对应不同类型实例的状态信息，模块通过字段组合的方式支持多类型数据统一管理。

**3.9.4 关键算法/流程**

模块工作流程以数据的保存（Save）与读取（Load） 为核心，具体如下：

（1）保存流程：

遍历如 ItemInstanceManager 等运行时管理器，提取所有实体数据；根据实体类型构造对应的 ItemInstanceData，提取其关键字段（如作物保存生长状态、建筑保存内容物等）。然后使用 Newtonsoft.Json 将数据序列化为 JSON 字符串。最后写入至 Application.persistentDataPath 路径下的 .json 文件中，确保数据在设备本地的持久性。

（2）加载流程：

检查对应 .json 文件是否存在；若存在，则读取文件内容，反序列化为对应的数据对象列表；根据数据中的 type 字段判断实体类型并初始化正确的实例；最后将还原出的实体加入管理器中并恢复运行状态（如生长数值、耐久度等）；同时对索引、状态缓存等结构进行同步处理，确保游戏逻辑完整运行。

**3.9.5测试机制：**

内置测试函数 TestSaveAndLoadItemInstanceManagerData() 实现保存-清空-读取的闭环验证；通过 Debug.Log 输出详细验证信息，确保各类数据在存储还原中无误。

**3.9.6 Unity相关设计**

在 Unity 相关设计方面，SLManager 模块首先采用了单例持久化的策略，其继承自 MonoBehaviour，并通过在 Awake() 函数中调用 DontDestroyOnLoad()，确保其在不同场景之间持续存在。此外，为实现与其他系统的数据联动，SLManager 会引用各类数据管理器的 Instance，在保存操作中提取运行数据，在加载操作中重建游戏状态，从而实现数据的完整流转。同时，考虑到跨平台运行的兼容性，模块统一采用 Application.persistentDataPath 作为存储路径，以确保在 Windows、Mac 等主流平台上的文件访问稳定可靠。最后，为提高调试效率，系统还设计了快捷键测试机制，例如通过 F4 触发保存、F5 触发加载，并结合调试日志输出，实现数据存储过程的高效验证与排错。

**3.10时间模块设计**

维护属于全局的TimeManager.Instance.time\_scale作为全局游戏时间倍速参考值，并建立季节和年份计算逻辑，为后续扩展如按天计算的灾害和季节性特殊情况做准备。

**3.11美术设计**

游戏美术资源统一采用像素风格的贴图，部分UI使用简单图形，计划来源为免费资源和组员绘制。

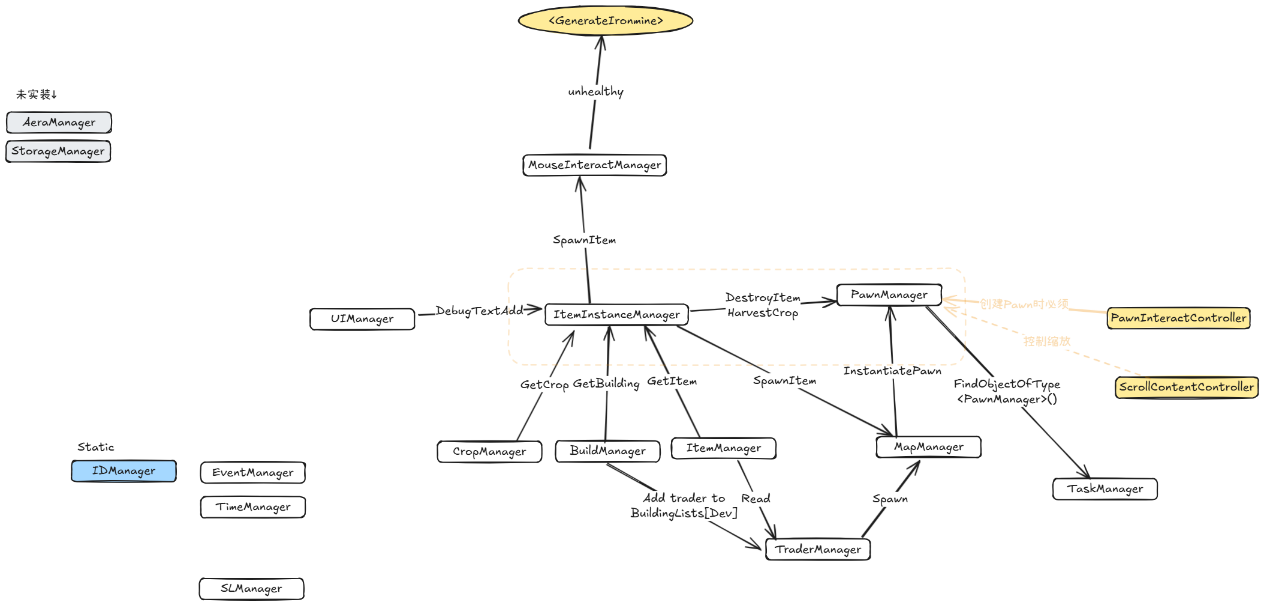
贴图应被导入到Unity项目的Asset/Sprite中

* Filter Mode同一设置为Point（no filter）
* Pixel Per Unit默认设置为16，大小对应大约一个地块

（本项目核心需求不在于美术，且美术资源容易后续更换和扩展，故不做过多设计限制。）

**4.附录**

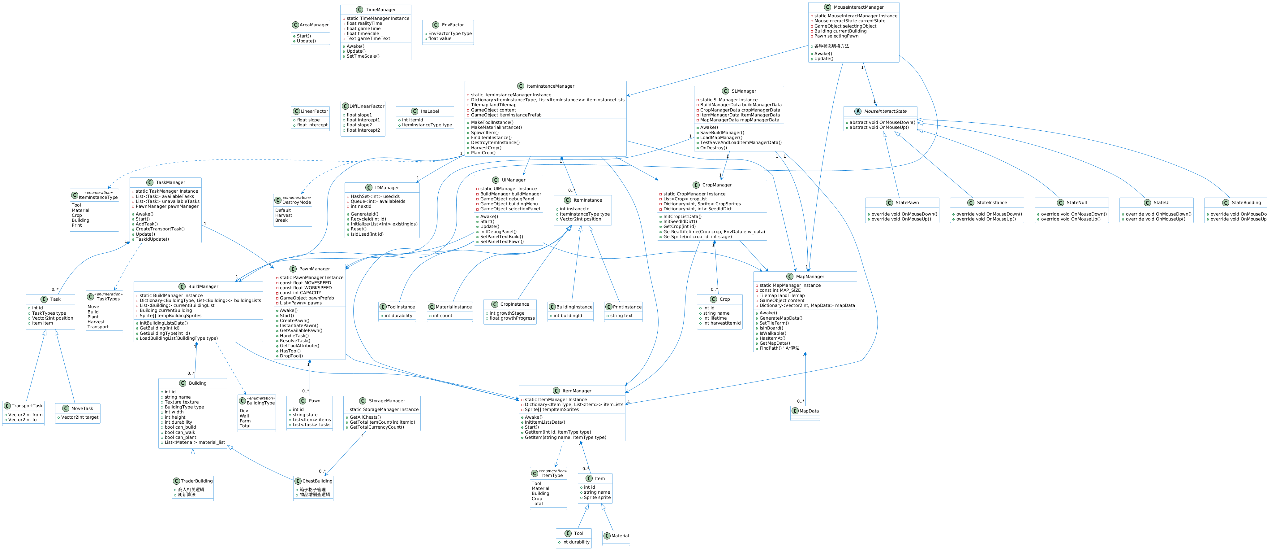
**4.1系统Manager间初始化依赖关系图（绘制：陈佳皓）**



网址：

<https://excalidraw.com/#json=ovaVn7L40RCJBOnHOpLZP,rL6ID-poF4APPF8JjKcGrA>

**4.2类UML图（绘制：刘瑜亮）**



源代码网址：

<https://github.com/david200404/SEProject_lyl/blob/main/Module.uml>