# 植物大战僵尸

# 1 引言

在游戏产业蓬勃发展的当下，休闲策略类游戏凭借独特的趣味性与益智性收获了广泛喜爱，《植物大战僵尸》便是其中极具代表性的佳作。为深入掌握面向对象编程思想与系统架构设计方法，提升团队协作及工程实践能力，本小组决定开发一款简化版植物大战僵尸游戏系统，以此探索游戏开发的核心流程与技术要点。

# 2 需求分析

## 2.1 系统功能分析

设计一个植物大战僵尸简化版，需满足以下功能：

（1）界面管理模块：界面转换，地图场景，植物卡片，阳光显示，波次，音量调节，暂停继续等。

（2）植物管理模块：实现向日葵生产阳光、豌豆射手攻击，炸弹爆炸，坚果防御。

（3）僵尸管理模块：移动，攻击，死亡三者转换，减速判定。

（4）阳光管理模块：阳光的生成、收集与消耗，收集音效。

（5）交互功能：支持鼠标点击种植植物、铲子清除植物、僵尸与植物攻击逻辑。

2.2 系统设计目标

本项目旨在设计并实现一款具备完整用户交互界面与后台逻辑管理的植物大战僵尸简化版游戏，核心目标是在保证核心游戏逻辑高效运行的基础上，构建具有优异扩展性与可维护性的系统架构，具体设计目标如下：

（1）架构设计合理性：搭建层次清晰、职责明确的系统框架，确保植物放置、僵尸生成、阳光收集、碰撞检测、胜负判定等核心功能模块协调工作，形成完整的逻辑闭环，保障游戏功能稳定正确运行。

（2）后台技术落地性：深度应用 QT 框架关键技术，实现精细化后台逻辑管理：

基于信号槽机制（槽函数与 connect 函数），建立对象间松耦合通信，实现用户操作与游戏响应的精准关联；

利用 multimedia 模块集成音频功能，包括背景音乐循环播放、植物种植音效、僵尸攻击音效等，增强游戏沉浸感；

通过 QTimer 计时器与 advance 函数构建帧更新机制，驱动僵尸移动、阳光生成、植物冷却等动态逻辑的定时执行；

重写绘图、鼠标、拖放、窗口布局等事件函数，优化植物拖拽放置、鼠标交互反馈等操作体验；

合理运用 QT 自带控件（按钮、标签、滑块等），实现游戏暂停 / 继续、音量调节、波次显示等辅助功能。

（3）前端呈现专业性：基于 QT 的 QGraphicsView 图形视图框架，实现高质量前端视觉呈现：

掌握 QGraphicsScene 核心功能：作为图形场景中枢，统一管理植物、僵尸、阳光等所有图形项，处理坐标转换与碰撞检测（如子弹命中僵尸、僵尸接触植物等）；

基于 QGraphicsItem 基类自定义各类游戏元素：为植物、僵尸、子弹等对象设计视觉呈现与交互逻辑，绑定对应的 PNG/GIF 资源实现动态效果；

利用 QGraphicsView 视图窗口：负责场景内容的渲染与显示，支持窗口自适应调整，确保游戏画面在不同设备上的一致性与清晰度。

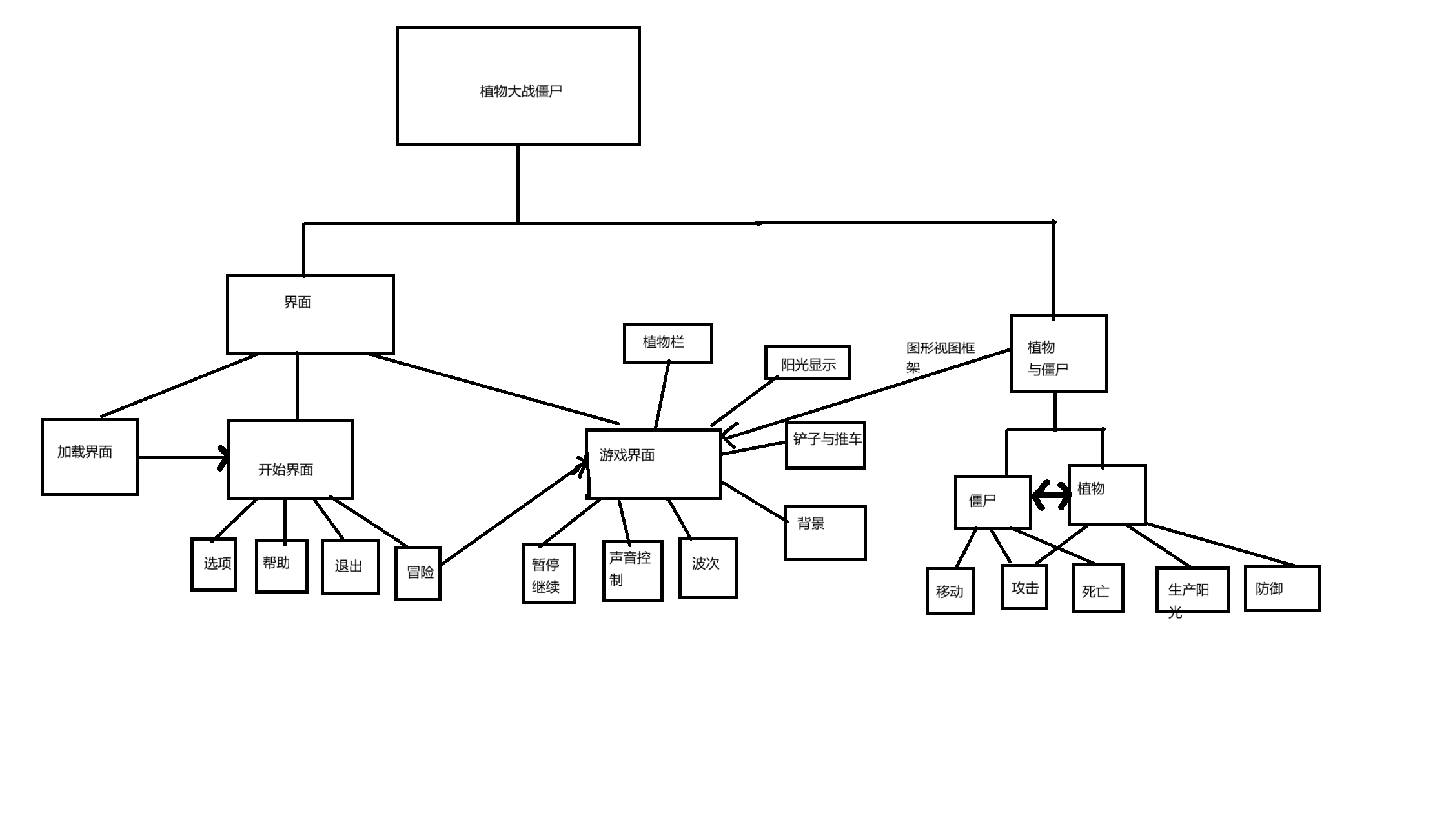
（4）前后端融合与体验优化：实现后台逻辑与前端呈现的深度融合，通过 "逻辑驱动视觉、视觉反馈操作" 的双向机制，打造流畅的游戏体验。从植物拖拽时的实时预览，到僵尸移动的平滑动画，再到操作与音效的同步响应，每一处细节设计均以 "让玩家感受到策略乐趣与操作快感" 为核心目标。

本项目累计投入超 14天，日均开发时长 12 小时以上，形成 3800 多行有效代码，整合 70 个 PNG/GIF 素材与 1 个开场视频资源。从技术架构来看，所有核心类均基于 QObject 与图形视图框架双重继承设计，既具备 QT 信号槽、对象树管理等核心功能，又能无缝融入图形渲染体系，为后续功能拓展奠定坚实基础。

我个人认为该作品已超越传统课程设计范畴，成为一款具备独立游戏核心特质的小型游戏作品 —— 拥有完整的玩法闭环、精细化的交互体验与可独立运行的产品形态。目前虽仍有 3 个拓展功能待实现（关卡选择，植物选择，进度保存），但依托模块化架构设计，可快速完成迭代升级，展现出强大的拓展潜力。这不仅是对 QT 技术的实践成果，更是我个人独立设计开发小型游戏的一次完整尝试。

# 3 系统设计

## 3.1系统功能设计

 图1 系统功能模块图

系统需要设计的主要模块及所需实现的功能如下：

（1）界面管理模块：  
用于实现游戏各界面之间的流畅转换，打造沉浸式游戏体验。具体而言，要构建加载界面、开始界面与游戏界面的切换逻辑；在游戏界面中，需合理布局地图场景以呈现游戏战场，设置植物卡片栏供玩家选择植物，精准显示阳光数量和波次信息，提供音量调节控件满足玩家个性化音效需求，以及实现暂停与继续游戏的功能切换。

（2）植物管理模块：  
负责管理各类植物的特性与行为。其中，向日葵要按照设定的频率稳定生产阳光，为玩家提供资源保障；豌豆射手能够发射豌豆子弹，对僵尸进行远程攻击；寒冰射手能够发射减速子弹，对僵尸造成减速；双发豌豆能够发射两颗子弹，对僵尸造成攻击；炸弹类植物（如樱桃炸弹）可在特定触发条件下产生爆炸效果，对范围内僵尸造成大量伤害；土豆地雷等待十几秒对范围一格内僵尸造成伤害；坚果类植物（如坚果）则主要承担防御职责，阻挡僵尸前进，为其他攻击型植物争取时间。

（3）僵尸管理模块：  
对僵尸的行为逻辑进行管控。僵尸会向同一行植物所在方向移动，当接触到植物时会发起攻击；在受到植物攻击或达到生命值上限时，会触发死亡逻辑；同时，还需实现减速判定功能，即当僵尸受到寒冰射手等具有减速效果的植物攻击时，移动速度会相应降低。

（4）阳光管理模块：  
把控阳光这一核心资源的全流程。一方面，向日葵等植物会定时生成阳光，且阳光也会从游戏场景中随机掉落；另一方面，玩家通过点击可收集阳光，收集过程中会播放对应的音效反馈，同时，种植植物等操作会消耗阳光，模块需准确处理阳光的消耗逻辑，确保资源管理的精准性。

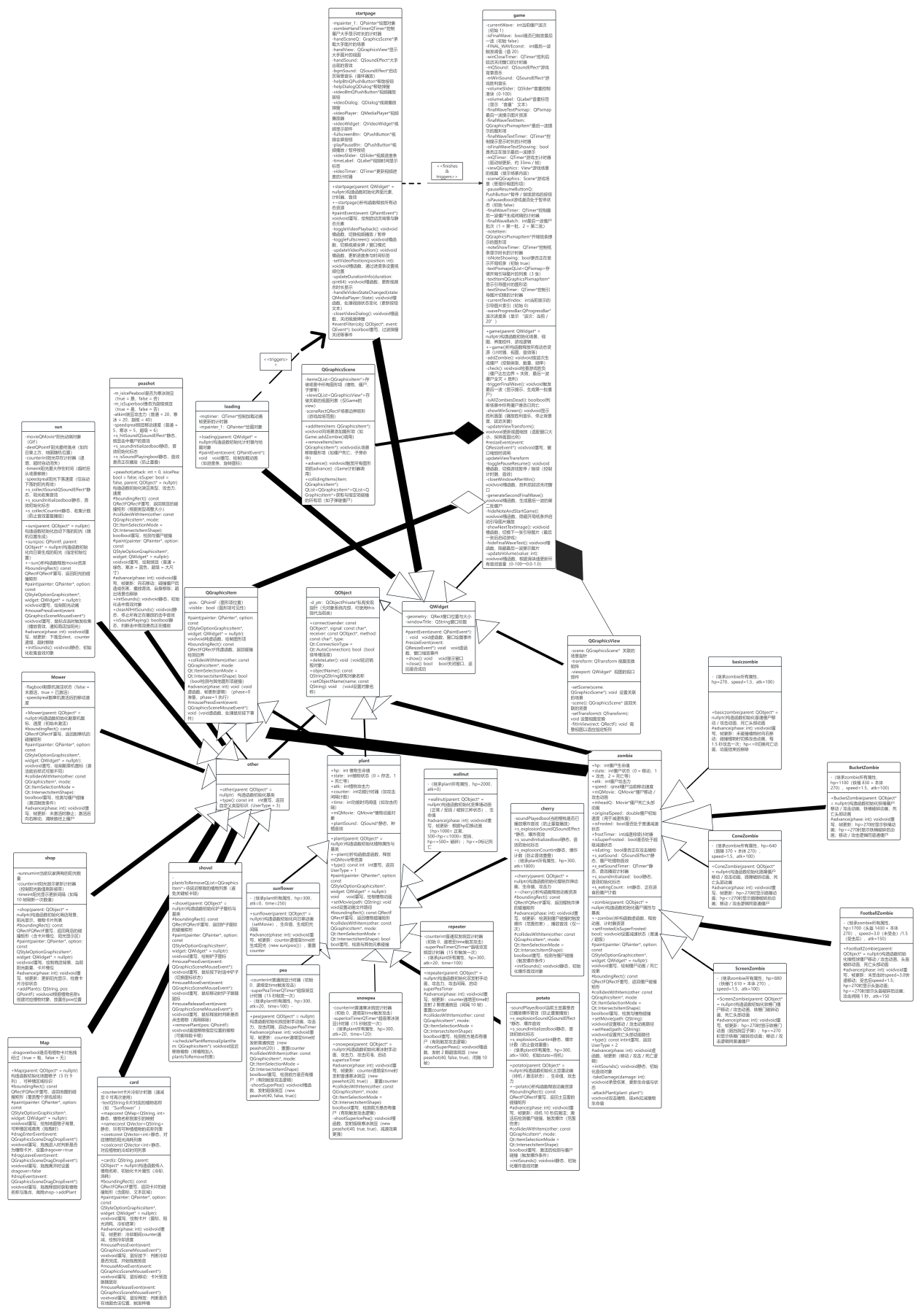
（5）交互功能模块：  
支撑玩家与游戏的交互操作。允许玩家通过鼠标点击从植物卡片栏选取植物，并拖拽到地图场景的合适位置进行种植；提供铲子工具，玩家可使用铲子清除已种植的植物；此外，还需实现僵尸与植物之间的攻击交互逻辑，即植物攻击僵尸、僵尸反击植物的判定与响应。

、

（6）图形视图框架模块：  
基于 QT 的 QGraphicsView 图形视图框架，实现高质量的前端视觉呈现。以 QGraphicsScene 作为图形场景中枢，统一管理植物、僵尸、阳光等所有图形项，高效处理坐标转换与碰撞检测（如子弹命中僵尸、僵尸接触植物等场景）；基于 QGraphicsItem 基类自定义各类游戏元素，为植物、僵尸、子弹等对象设计视觉呈现与交互逻辑，并绑定对应的 PNG/GIF 资源以实现动态效果；利用 QGraphicsView 视图窗口，负责场景内容的渲染与显示，支持窗口自适应调整，确保游戏画面在不同设备上的一致性与清晰度。

## 3.2 类设计与类结构

为了完成以上模块功能，设计各个类及类之间的关系如图2所示。



文档链接：[PVZuml.pom](https://www.kdocs.cn/l/cfur9kaPjz91) (ctrl加鼠标点击)

图2 各个类之间的关系图

1. QObject类的 UML 图如图 3 所示，主要作为 Qt 框架中所有对象的根父类，提供核心机制：信号与槽（实现对象间通信，如游戏暂停信号触发视图停止更新）、对象树管理（自动回收子对象内存，如Game窗口销毁时自动释放关联的QTimer）、元对象系统（支持反射，如qobject\_cast类型转换）。所有需要交互或生命周期管理的类（如QWidget、QGraphicsScene）均直接或间接继承此类。

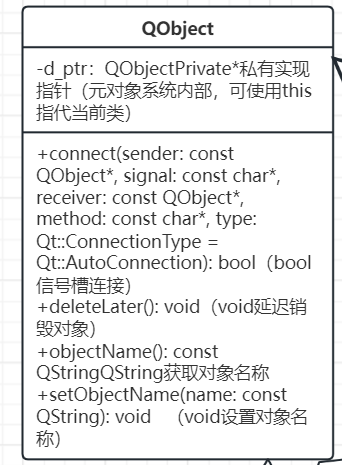


图3 QObject类的 UML 图

1. QWidget类的 UML 图如图 4 所示，主要作为所有可视化窗口部件的基类，继承自QObject和QPaintDevice。提供窗口显示（show()）、事件处理（鼠标 / 键盘输入）、布局管理（setLayout()）等核心功能，是Game主窗口、QGraphicsView等界面类的父类，负责将游戏元素呈现给用户并接收交互操作。

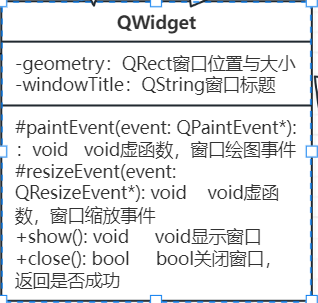


图4 QWidget类的 UML 图

（3）QGraphicsItem类的 UML 图如图 5 所示，主要作为所有 2D 图形元素的抽象基类，无父类但可与QObject多继承（如游戏中的Plant、Zombie）。定义纯虚方法paint()（绘制外观）和boundingRect()（碰撞检测边界），提供位置管理（setPos()）和交互事件（mousePressEvent）接口，是构建游戏中所有可见元素（植物、僵尸、子弹等）的基础。

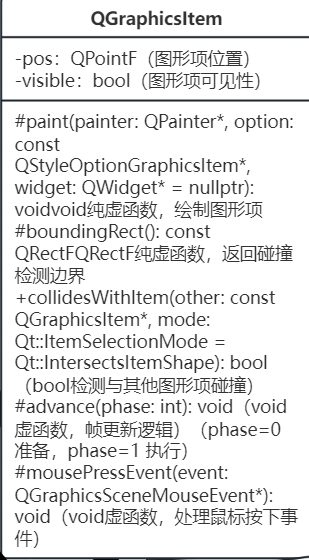


图5 QGraphicsItem类的 UML 图

(4)QGraphicsScene类的 UML 图如图 6 所示，主要用于管理游戏场景中的所有图形项（如植物、僵尸、子弹、阳光等QGraphicsItem子类对象），负责存储这些图形项、处理它们之间的碰撞检测、坐标转换等逻辑，还能将场景内容与QGraphicsView关联，实现一对多的场景 - 视图显示关系，即一个QGraphicsScene可被多个QGraphicsView显示。

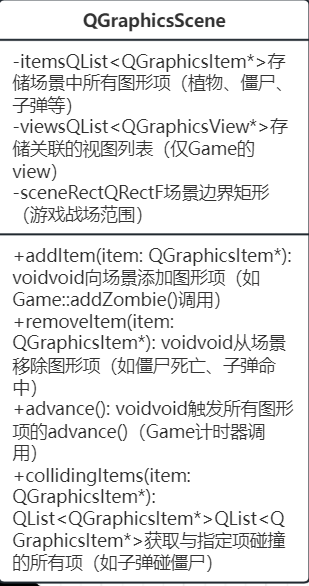


图6 QGraphicsScene类的 UML 图

1. QGraphicsView类的 UML 图如图7 所示，主要用于将QGraphicsScene中的内容渲染到屏幕上，作为可视化的窗口部件，继承自QWidget，具备窗口显示、事件处理等特性。它与QGraphicsScene是一对一的关联关系，一个QGraphicsView关联一个QGraphicsScene，同时依赖QPainter进行场景内容渲染，依赖QTransform处理视图的缩放、旋转等变换操作。



图7 QGraphicsView类的 UML 图

1. Game类的 UML 图如图 8 所示，主要作为整个游戏的核心控制类，组合了QGraphicsScene和QGraphicsView，负责游戏的整体初始化（如创建场景、视图、计时器等）、游戏主循环的驱动（通过计时器触发场景中图形项的更新）、游戏状态的管理（如暂停 / 继续逻辑）以及胜负检测等核心游戏逻辑，是连接各图形类与用户交互的中枢。



图8 Game类的 UML 图如图

1. Plant类的 UML 图如图 9所示，主要用于表示游戏中的植物元素，继承自QGraphicsItem，具备植物特有的属性（如生命值hp、攻击力atk等）和行为（如攻击方法attack），通过重写QGraphicsItem的paint和boundingRect等方法，实现植物的视觉呈现，并在advance方法中实现植物的攻击等交互逻辑。

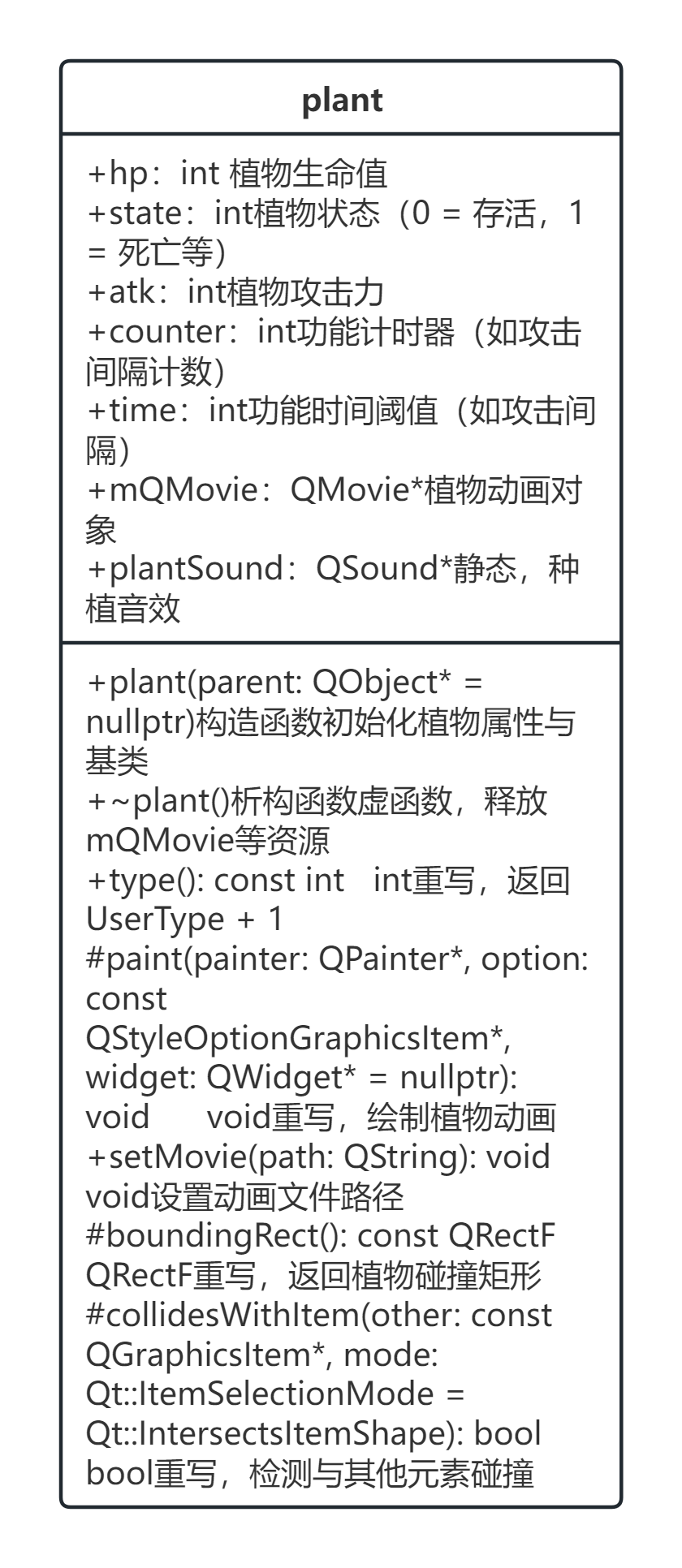


图9 plant类的 UML 图如图

(8)Zombie类的 UML 图如图 10 所示，主要用于表示游戏中的僵尸元素，继承自QGraphicsItem，拥有僵尸专属的属性（如移动速度speed、生命值hp等）和行为（如移动方法move），重写paint和boundingRect方法来展示僵尸的外观，在advance方法中实现僵尸的移动、攻击等交互逻辑。

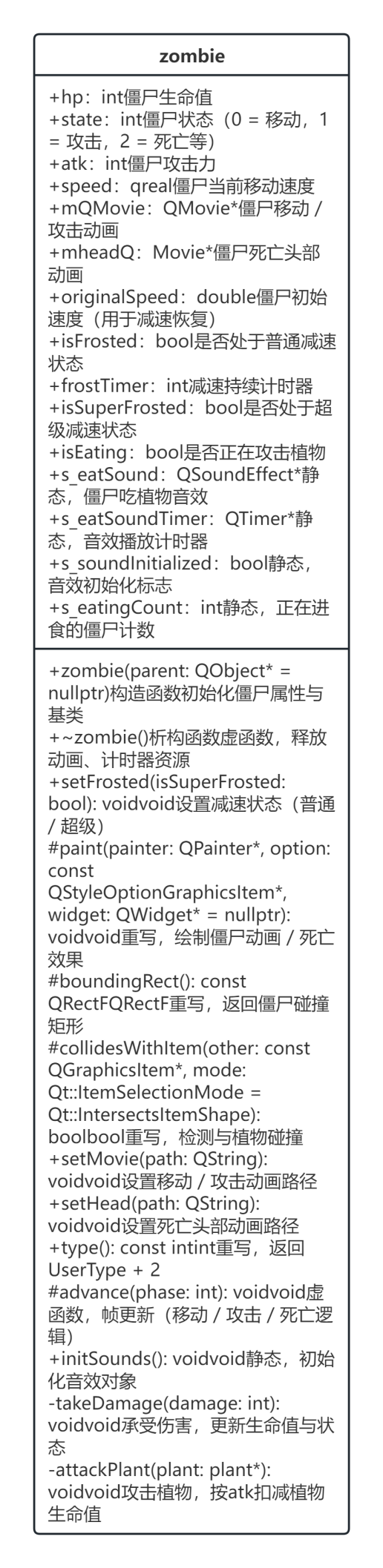


图10 Zombie类的 UML 图

（9）Sun类的 UML 图如图 11 所示，主要用于表示游戏中的阳光元素，继承自QGraphicsItem，具有阳光的价值属性value，通过paint和boundingRect方法实现阳光的视觉效果，在advance等方法中处理阳光的生成、移动以及被收集（collect方法）等逻辑。

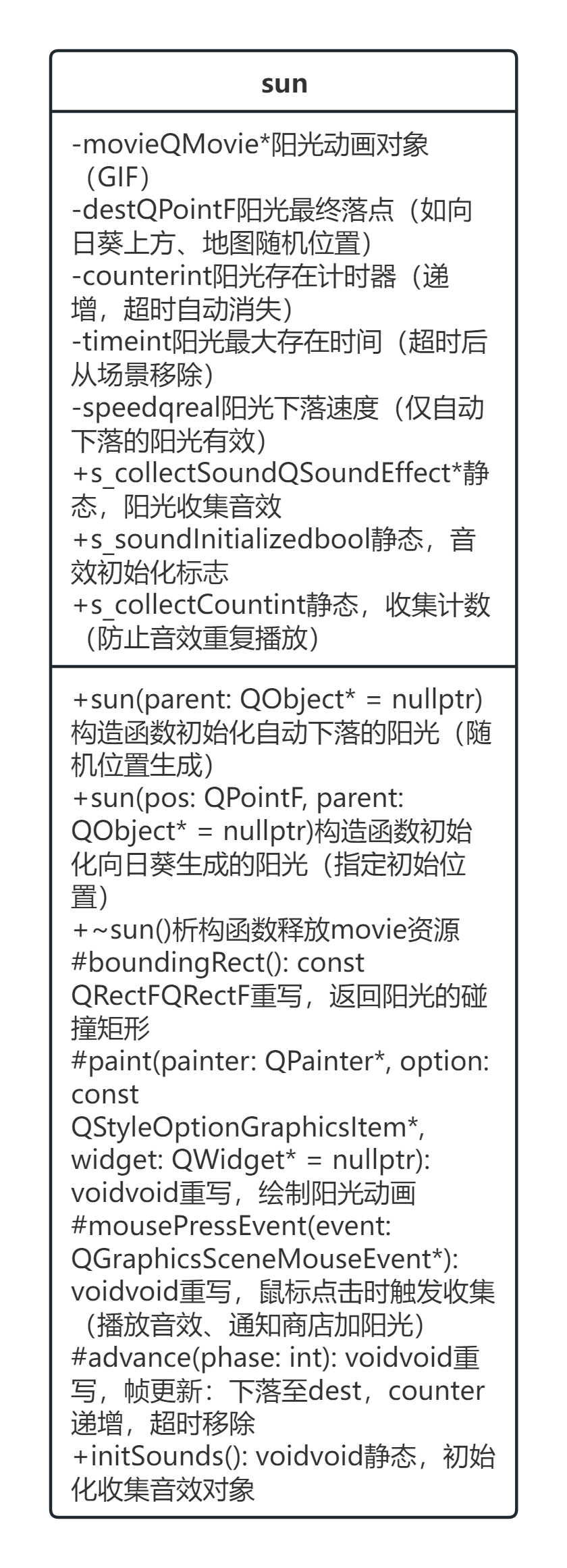


图11 Sun类的 UML 图

（10）Shop类的 UML 图如图 12 所示，主要作为游戏商店 / 植物选择界面的控制类，继承自QWidget。关联Game类获取阳光数量，管理植物卡片集合（QList<Card\*>）；核心方法包括updateSunCount(int count): void（更新阳光显示）、lockCard(Card\* card, int cd): void（处理卡片冷却）、onCardClicked(Card\* card): void（响应卡片点击，通知Game生成对应植物），是玩家与植物系统交互的中间层。



图12 Shop类的 UML 图

（11）Map类的 UML 图如图 13 所示，主要作为游戏地图与格子管理的基类，继承自QObject。核心属性包括行数（rows: int）、列数（cols: int）、格子大小（cellSize: QSizeF）、可种植区域（plantableCells: QSet<QPoint>）；提供isCellPlantable(QPoint cellPos): bool（判断格子是否可种植植物）、getCellRect(QPoint cellPos): QRectF（获取格子在场景中的坐标）等方法，为植物放置、僵尸路径规划提供基础数据支持。



图13 Map类的 UML 图

（12）Card类的 UML 图如图 14 所示，主要作为植物卡片的基类，多继承自QObject和QGraphicsItem。核心属性包括对应植物类型（plantType: Enum）、阳光消耗（cost: int）、冷却时间（cooldown: int）、激活状态（isActive: bool）；重写mousePressEvent处理点击逻辑，paint()方法根据状态绘制（可用时彩色、冷却时灰色），是连接Shop与Game的交互桥梁。

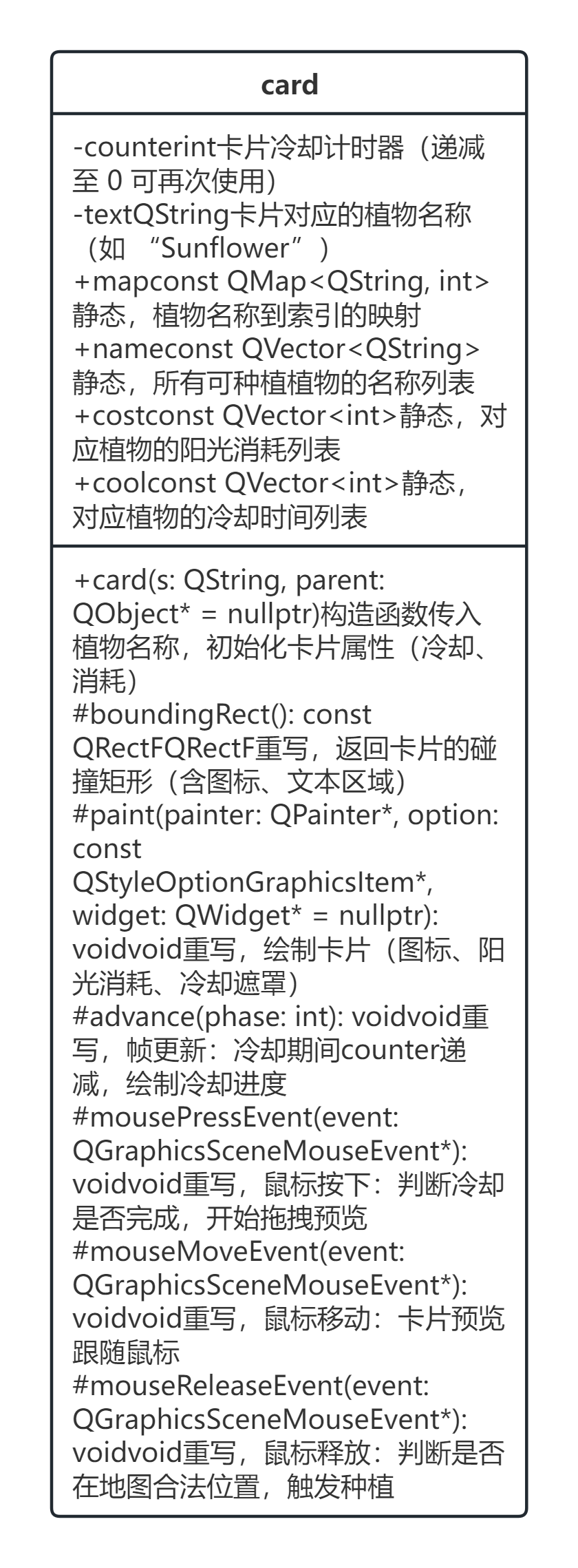


图14 Card类的 UML 图

（13）peashot类的 UML 图如图 15 所示，主要用于表示游戏中的豌豆子弹，继承自other类（推测为基础图形项类，间接关联QGraphicsItem），是植物攻击的核心载体。



图15 peashot类的 UML 图

## 3.3 开发与运行环境

硬件: 处理器：Intel (R) Core (TM) i7 - 14650HX 2.20 GHz

系统类型：64 位操作系统，基于 x64 的处理器

机带 RAM：16.0 GB

Windows 11 家庭中文版

软件: 基于Qt，使用C++开发植物大战僵尸游戏

# 4 设计与实现（提示：这部分写自己的工作内容）

## 4.1 功能模块的设计

4 设计与实现

4.1 功能模块的设计

（1）加载界面（Loading）功能的设计与实现

加载界面是游戏的 “入口网关”，核心目标是完成资源预加载 + 引导用户进入开始界面，避免后续界面因资源加载不完整导致卡顿或黑屏，同时通过简洁的视觉设计降低用户等待焦虑。

核心设计思路

资源预加载优先：启动时先加载窗口图标、背景图、核心按钮素材，确保界面元素显示无延迟；

单一交互入口：仅保留 “Start” 按钮作为唯一交互点，避免用户操作混淆；

无缝界面跳转：通过 “隐藏当前界面 + 显示目标界面” 的非销毁式切换，减少资源重复加载，提升跳转流畅度。

初始化阶段：

程序启动后，创建加载界面窗口，设置固定尺寸（900×600）、窗口标题和图标（坚果图标）；

加载背景图（StartScreen.jpg）和标题图（LogoWord.jpg），通过paintEvent绘制到窗口指定位置（标题图居上，背景图全屏）；

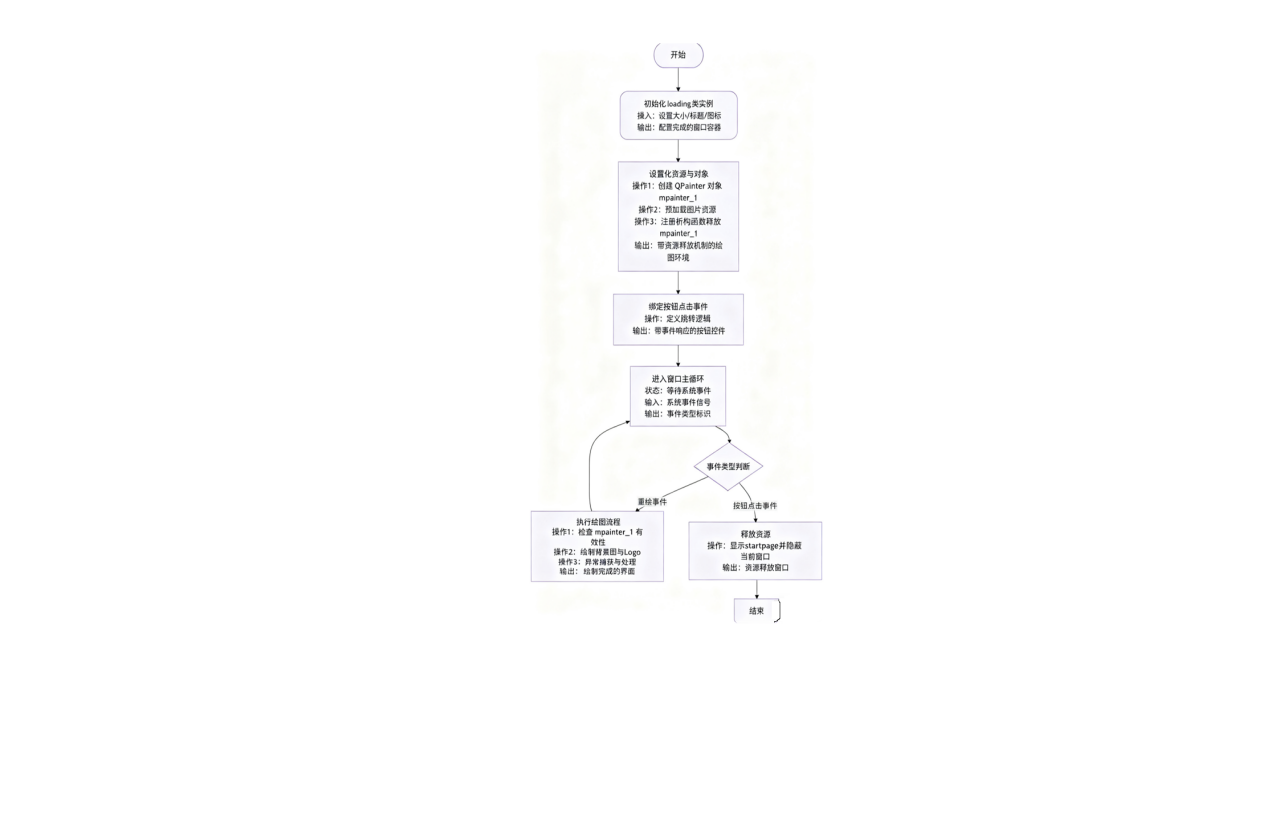
创建 “Start” 按钮，设置按钮样式（背景图 + hover 效果），绑定点击事件。

交互与跳转阶段：

若用户未点击 “Start” 按钮，界面保持静态显示，等待交互；

若用户点击 “Start” 按钮，执行两步操作：① 隐藏加载界面窗口；② 初始化并显示开始界面（StartPage），同时启动开始界面的背景音乐。

算法流程图如下：



（2）开始界面（StartPage）功能的设计与实现

开始界面是游戏的 “功能中枢”，需承载界面导航、辅助功能（帮助 / 视频）、交互反馈（僵尸大手） 三大核心职责，同时通过多媒体元素（背景音乐、视频）提升用户沉浸感，确保功能与视觉的平衡。

核心设计思路

模块化功能布局：将 “游戏启动”“帮助查询”“视频播放”“退出” 功能分区布局，按钮位置固定（右侧功能区 + 底部控制区），降低用户学习成本；

多媒体协同控制：背景音乐与视频播放互斥（播放视频时暂停背景音乐，关闭视频后恢复），避免音效冲突；

交互反馈精细化：僵尸大手交互（点击 “冒险模式” 后触发）结合音效与动画，强化 “进入游戏” 的仪式感；

模态弹窗设计：帮助弹窗、视频弹窗采用模态 / 非模态区分（帮助为模态，不影响主界面；视频为非模态，可后台播放）。

算法流程（分 4 个子功能）

子功能 1：界面初始化与背景音乐控制

初始化窗口（900×600）、图标，绘制背景图（Surface.jpg）；

创建左上角标签（加载 labelone.png），固定位置（0,0），无需额外显示控制（父窗口绘制时自动渲染）；

初始化背景音乐（Grazy.wav），设置音量（0.5）、无限循环，进入界面后立即播放。

子功能 2：帮助弹窗交互

创建 “帮助” 按钮（底部左侧，与退出按钮并列），设置透明样式与 hover 反馈；

点击 “帮助” 按钮时，创建模态弹窗（固定 600×400），加载帮助背景图（help.png）；

弹窗仅保留系统标题栏（含关闭按钮），用户点击 “×” 时关闭弹窗，返回开始界面。

子功能 3：视频播放控制

创建 “播放视频” 按钮（帮助按钮左侧），点击后执行：① 暂停背景音乐；② 显示视频弹窗（800×500）；③ 初始化视频播放器（加载 theme.mp4）并自动播放；

视频弹窗内提供控制栏：播放 / 暂停（切换文本与状态）、进度条（拖动调整位置）、全屏（切换窗口模式）、关闭（停止播放 + 隐藏弹窗 + 恢复背景音乐）；

视频播放完毕后，自动停止播放，1 秒后关闭弹窗，恢复背景音乐。

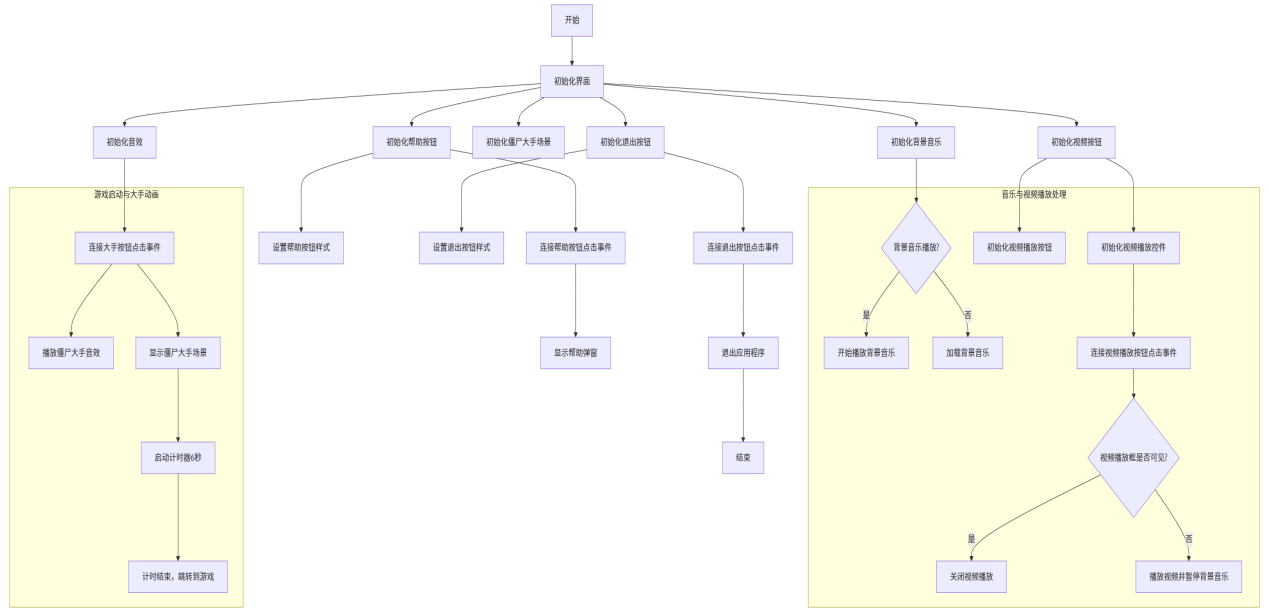
子功能 4：僵尸大手交互与游戏跳转

点击 “冒险模式” 按钮时：① 停止背景音乐；② 初始化大手场景（160×256）与视图，加载大手图片（handlast.png）；③ 播放大笑音效（laugh.wav）；

启动计时器（6 秒），计时期间显示大手视图；

计时结束后：① 停止音效；② 隐藏大手视图；③ 初始化游戏界面（Game）并显示，隐藏开始界面。

算法流程图如下:



（3）游戏界面（Game）核心功能的设计与实现

游戏界面是玩法的 “执行载体”，需实现场景渲染、帧循环驱动、波次管理、交互控制（暂停 / 音量）、胜负判定五大核心逻辑，同时通过视图自适应确保不同窗口尺寸下的视觉一致性。

核心设计思路

场景 - 视图分离架构：QGraphicsScene作为 “数据中枢” 管理所有游戏元素（植物、僵尸、阳光等），QGraphicsView作为 “显示窗口” 负责渲染，通过updateViewTransform()实现等比例缩放；

帧循环驱动：通过QTimer（33ms / 帧）触发scene->advance()，驱动所有元素的帧更新（移动、攻击、生成等）；

动态难度递增：波次管理按 “间隔缩短、数量增加、类型强化” 调整难度，最后一波触发特殊提示与强力僵尸；

交互控件固定化：暂停按钮、音量滑块、波次进度条固定在右上角，操作位置不随窗口缩放变化。

算法流程（分 4 个子功能）

子功能 1：场景与视图初始化

初始化窗口（900×600），设置最小尺寸（400×300）；

创建QGraphicsScene，设置sceneRect（150, 0, 900, 600）（X 偏移 150 预留背景显示区），禁用索引提升性能；

创建QGraphicsView，绑定场景，设置无边框、隐藏滚动条，加载背景图（Background.jpg）；

调用updateViewTransform()，计算缩放比例（取宽高比例最小值），使视图居中且等比例适配窗口。

子功能 2：波次管理与僵尸生成

初始化波次参数（当前波次 = 1，最终波次 = 20，是否最后一波 = false）；

每帧通过addZombie()判断是否生成僵尸：

按当前波次调整生成间隔（前 5 波 15 秒，16 波后 6-7 秒）；

按当前波次调整僵尸数量（前 5 波 1 只，16 波后 5 只）；

按当前波次调整僵尸类型概率（前期基础僵尸为主，后期高防御 / 高速度僵尸占比≥75%）；

当当前波次≥20 时，触发最后一波：① 显示 “最后一波” 提示（4 秒）；② 生成第一批加速屏幕僵尸（5 行各 1 只）；③ 5 秒后生成第二批橄榄球僵尸（5 行各 1 只）。

子功能 3：暂停 / 音量控制

暂停按钮：点击 “暂停” 时，停止QTimer（帧循环）和背景音乐，按钮文本改为 “继续”；点击 “继续” 时，重启计时器和音乐，文本改回 “暂停”；

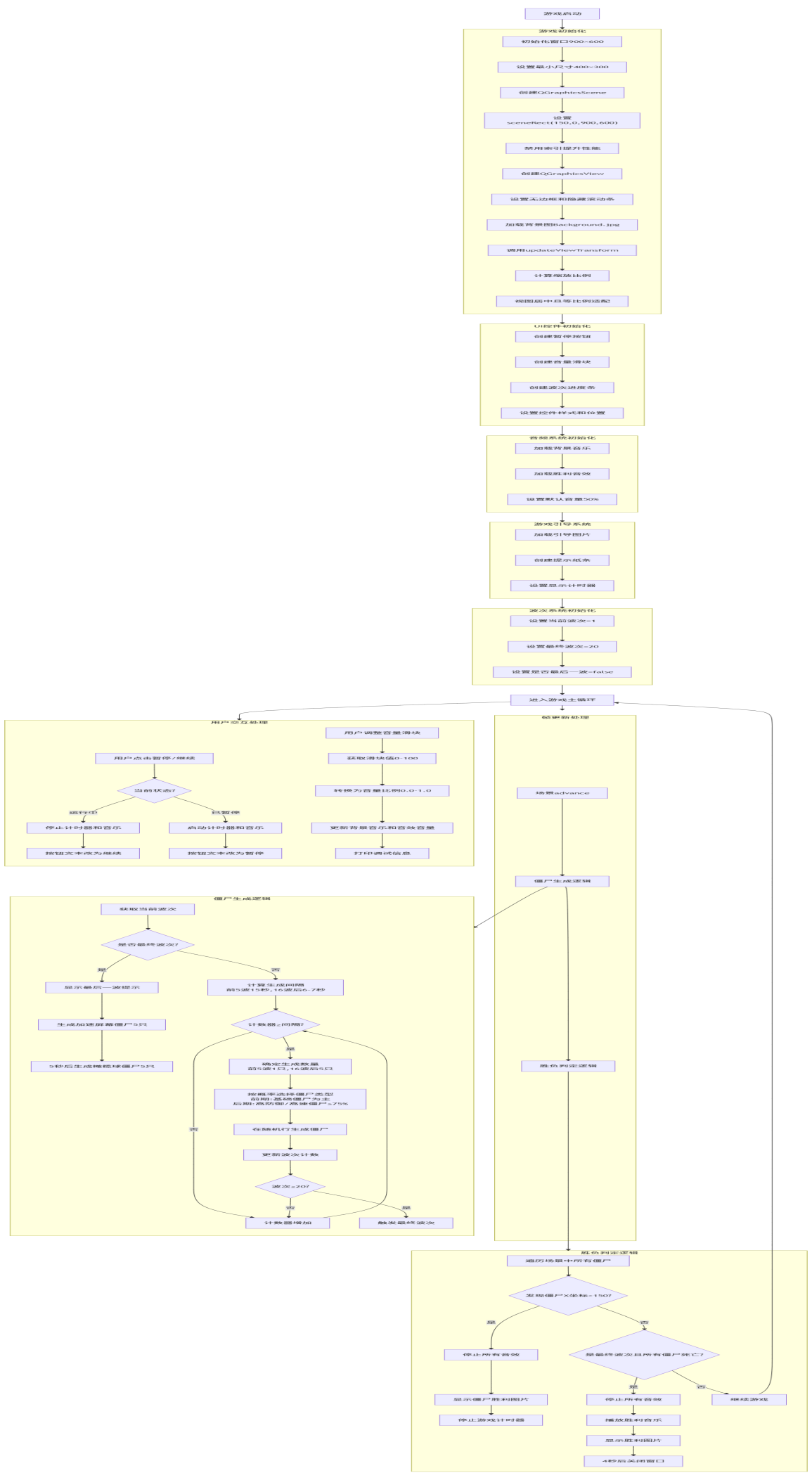
音量滑块：拖动时将滑块值（0-100）转换为音量比例（0.0-1.0），同步更新背景音乐和胜利音乐的音量，调试信息打印当前音量。

子功能 4：胜负判定

失败判定：每帧遍历场景中所有僵尸，若某僵尸 x 坐标 <150（场景左边界），则：① 停止所有音效；② 显示 “僵尸胜利” 图片；③ 停止QTimer，游戏结束；

胜利判定：若处于最后一波且isAllZombiesDead()返回 true（场景中无僵尸），则：① 停止所有音效；② 播放胜利音乐；③ 显示 “胜利” 图片；④ 4 秒后关闭窗口。

算法流程图如下:



（4）植物管理功能的设计与实现

植物管理是游戏的 “防御核心”，需通过抽象基类统一接口 + 子类差异化实现，管理不同植物的专属行为（阳光生产、攻击、防御、范围伤害），同时确保植物与其他模块（阳光、子弹、地图）的交互逻辑连贯。

核心设计思路

抽象基类（Plant）封装共性：定义所有植物共有的属性（生命值hp、攻击力atk、动画mQMovie）和方法（paint绘制、boundingRect碰撞范围、collidesWithItem碰撞检测），子类仅需重写advance实现专属逻辑；

子类行为差异化：按功能将植物分为 4 类 —— 资源生产类（向日葵）、远程攻击类（豌豆射手、寒冰射手、双发豌豆）、范围伤害类（樱桃炸弹、土豆地雷）、防御类（坚果），每类通过advance实现核心行为；

与子弹解耦：攻击型植物通过创建Peashot对象实现攻击，植物仅负责 “定时生成子弹”，子弹的移动、碰撞、伤害由Peashot类独立处理，降低耦合。

算法流程（分 4 类植物示例）

子功能 1：向日葵（资源生产类）

构造函数：初始化生命值（300）、阳光生成间隔（9 秒 = 9000ms/33ms≈273 帧），加载动画（SunFlower.gif）；

advance帧更新：

若生命值≤0，删除自身；

若生命值 > 0，生成计数器递增；

若计数器≥生成间隔，重置计数器，在向日葵位置附近创建Sun对象（阳光），添加到场景。

子功能 2：豌豆射手（远程攻击类）

构造函数：初始化生命值（300）、攻击力（30）、攻击间隔（1.4 秒≈42 帧），加载动画（Peashooter.gif），初始化超级子弹计时器（10 秒 / 次）；

advance帧更新：

若生命值≤0，删除自身；

若生命值 > 0，攻击计数器递增；

若计数器≥攻击间隔且collidingItems()检测到僵尸，重置计数器，在射手右侧 30 像素处创建普通Peashot子弹，添加到场景；

若超级子弹计时器触发，在同一位置创建超级Peashot子弹（攻击力 ×5，速度 ×0.5）。

子功能 3：樱桃炸弹（范围伤害类）

构造函数：初始化生命值（800）、攻击力（1800）、状态（0 = 预热，1 = 爆炸），加载预热动画（CherryBomb.gif），初始化全局爆炸音效（Boom.wav）；

advance帧更新：

若生命值≤0，删除自身；

若状态 = 0 且预热动画播放完毕，切换状态 = 1，加载爆炸动画（Boom.gif）；

若状态 = 1，播放爆炸音效（引用计数控制，仅第一颗播放），通过collidingItems()获取 160 像素范围内所有僵尸，扣除全额伤害；

若爆炸动画播放完毕，删除自身。

子功能 4：坚果（防御类）

构造函数：初始化生命值（3200），加载完整动画（WallNut.gif）；

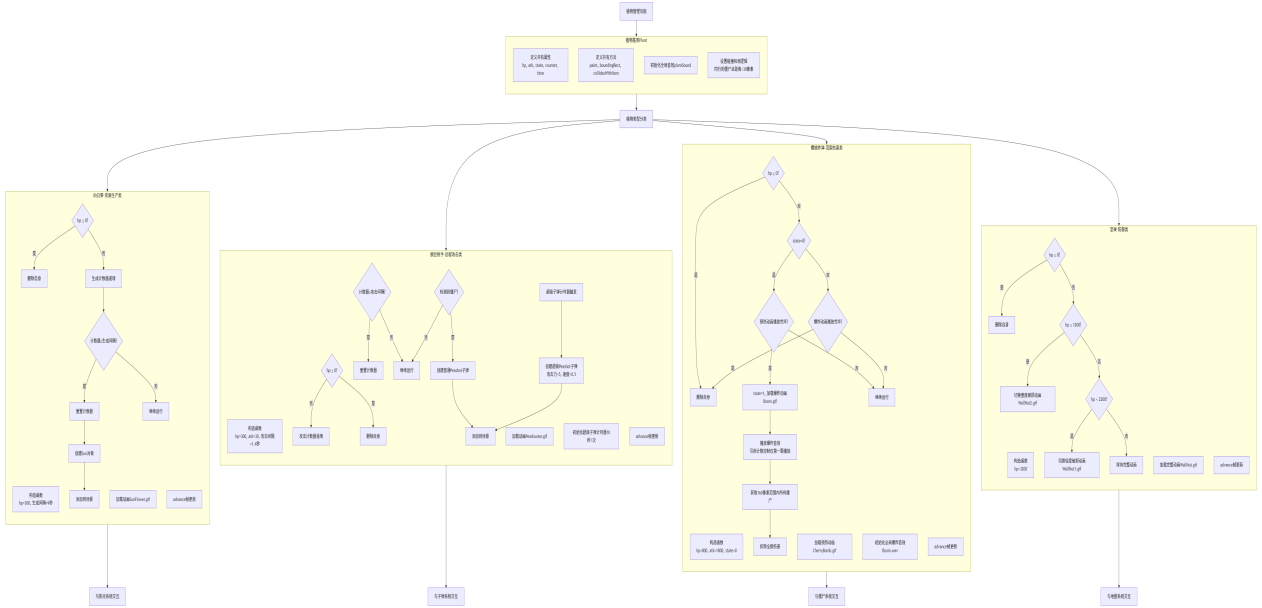
advance帧更新：

若生命值≤0，删除自身；

若 1200<hp<2200，切换为轻度破损动画（WallNut1.gif）；

若 hp≤1200，切换为重度破损动画（WallNut2.gif），直观反馈防御状态。

算法流程图如下:



（5）僵尸管理功能的设计与实现

僵尸管理是游戏的 “进攻核心”，需实现移动、攻击、死亡的状态切换，同时支持 “减速判定”（寒冰子弹效果）和 “动态音效控制”，确保僵尸行为符合 “难度递增” 和 “视觉反馈一致” 的要求。

核心设计思路

状态机驱动行为：僵尸通过state变量（0 = 移动，1 = 攻击，2 = 死亡）控制行为，advance帧更新时根据状态执行对应逻辑，避免行为混乱；

减速分层管理：通过isFrosted（普通减速）和isSuperFrosted（超级减速）区分效果 —— 普通减速速度 = 原速度 ×2/3，超级减速速度 = 0，减速计时器（60 帧 = 2 秒）结束后恢复原速度；

音效引用计数：用静态变量s\_eatingCount记录正在攻击的僵尸数量，仅当计数 = 1 时播放攻击音效（Eat.wav），避免多僵尸同时攻击导致音效重叠。

算法流程

构造函数初始化：

子类（基础 / 路障 / 铁桶 / 橄榄球僵尸）初始化专属属性：生命值（400-1400）、速度（原速度 = 80×33/1000 / 系数，系数 2.5-4.7）、动画（行走 / 攻击 / 死亡）；

调用父类（Zombie）初始化：减速状态（false）、减速计时器（0）、攻击状态（false），初始化全局攻击音效。

advance帧更新（核心逻辑）：

步骤 1：死亡状态处理。若state=2，清除减速状态，停止攻击音效，播放死亡动画，动画结束后删除自身；

步骤 2：减速逻辑处理。若isFrosted，减速计时器递减，动画速度设为 50%；计时器 = 0 时恢复原速度和动画速度；

步骤 3：碰撞检测。调用collidingItems()检测是否接触植物（同一行 + 距离 < 30 像素）；

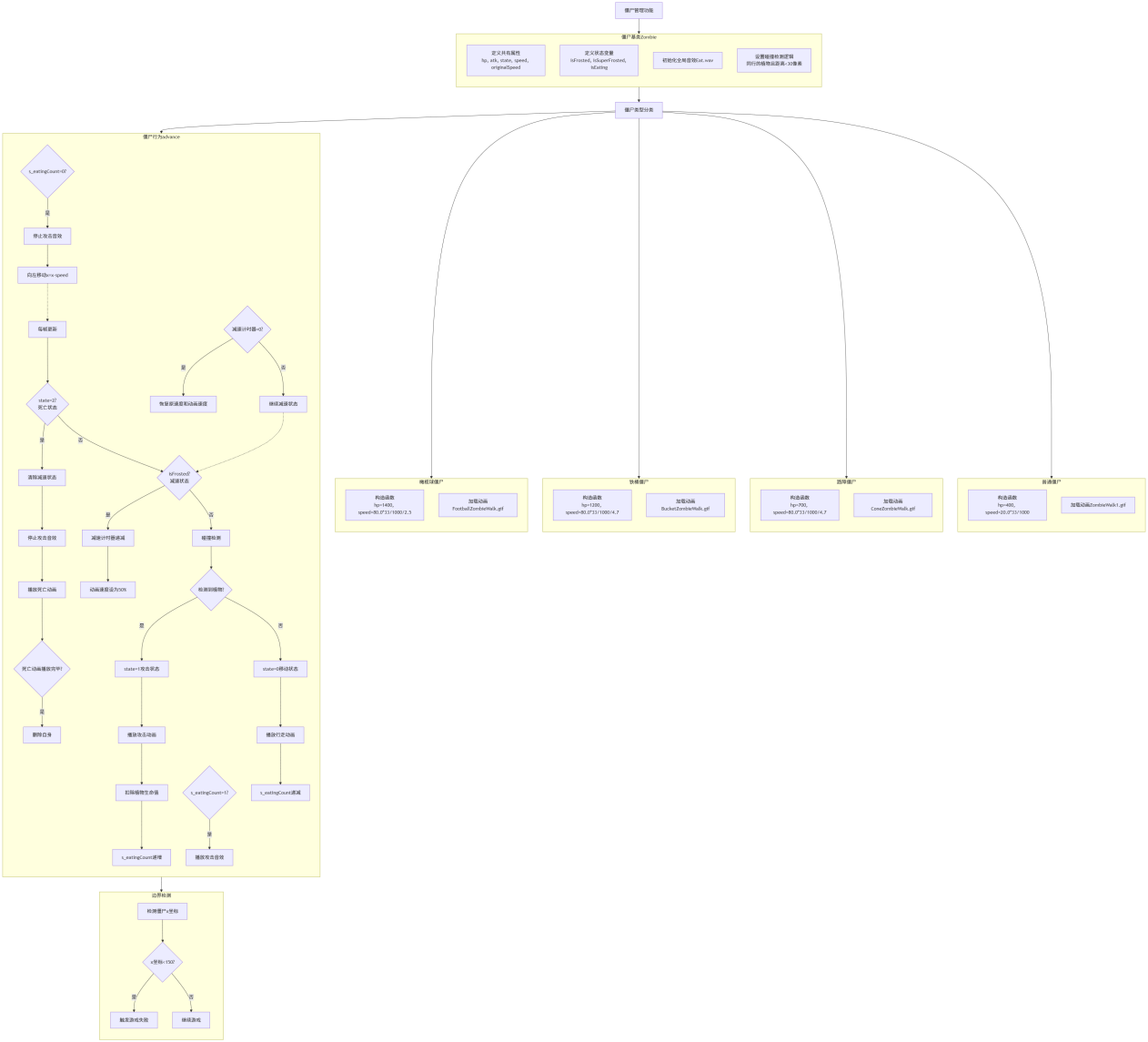
步骤 4：攻击 / 移动切换。

若检测到植物：切换state=1，播放攻击动画，扣除植物生命值，s\_eatingCount递增（计数 = 1 时播放音效）；

若未检测到植物：切换state=0，播放行走动画，s\_eatingCount递减（计数 = 0 时停止音效），向左移动（x=x-speed）；

步骤 5：边界检测。若僵尸 x 坐标 < 150，触发游戏失败。

算法流程图如下:



（6）Shop 类功能的设计与实现

Shop 类是游戏的 “植物资源中枢”，核心职责是统筹植物卡片管理、阳光资源管控、自动阳光生成三大核心功能，既是玩家选择植物的交互入口，也是游戏资源循环的关键枢纽，需实现 “卡片展示 - 阳光收支 - 冷却管控” 的全流程逻辑，确保与地图、植物、阳光等模块的无缝协同。

核心设计思路

模块化卡片管理：通过循环批量创建植物卡片，按固定间距横向排列形成 “植物选择栏”，统一管理卡片的显示、冷却与交互状态，避免分散逻辑导致的维护复杂；

阳光资源闭环管控：集中处理阳光的 “生成 - 收集 - 消耗” 全流程 —— 自动定时生成阳光补充资源、接收阳光收集信号更新总数、校验阳光充足性后执行植物种植消耗，确保资源数据精准同步；

低耦合模块交互：通过 “信号通知 + 接口调用” 与其他模块联动（如接收 Map 拖放事件触发种植、通知 Card 类启动冷却、接收 Sun 类信号更新阳光），避免直接依赖，提升扩展性；

视觉与逻辑同步：实时绘制阳光数值与商店背景，确保玩家直观获取资源状态；卡片冷却状态通过视觉遮罩同步反馈，让玩家清晰感知操作时机。

详细实现流程

1. 初始化阶段：搭建商店框架与资源预加载

初始化是 Shop 类功能的基础，需完成属性赋值、卡片创建、视觉渲染准备，确保进入游戏后即可正常使用，具体步骤如下：

基础属性初始化：

设定初始阳光值 sunnum = 200（游戏开局基础资源，满足首株向日葵 / 豌豆射手的种植消耗）；

计算自动阳光生成间隔 time = int(7.0 \* 1000 / 33)（7 秒，因游戏主循环约 33ms / 帧，需转换为帧计数，确保时间精度）；

初始化计数器 counter = 0（用于累计帧次数，触发自动阳光生成）。

植物卡片批量创建：

遍历静态列表 card::name（存储所有可种植植物名称，如 “Sunflower”“Pea” 等），为每个植物创建 card 对象；

设置卡片父子关系：car->setParentItem(this)，确保卡片随 Shop 类统一管理（如 Shop 移动时卡片同步跟随）；

固定卡片排列位置：car->setPos(-157 + 65 \* i, -2)，横向间距 65 像素，左侧起始偏移 - 157 像素，确保所有卡片在商店背景框内均匀分布，无重叠或溢出。

视觉渲染准备：

预加载商店背景图 Shop.png（尺寸 540×90 像素），确定绘图区域 QRect(-270, -45, 540, 90)（左负右正、上负下正，确保背景居中覆盖卡片区域）；

配置阳光显示字体：font.setPointSizeF(15)，设定显示区域 QRectF(-255, 18, 65, 22)，确保阳光数值在背景框左侧居中，清晰易读。

2. 视觉渲染：实时绘制商店界面与资源信息

通过重写 paint 函数，完成商店背景与阳光数值的绘制，确保玩家实时获取界面状态，具体逻辑如下：

绘制商店背景框：

调用 painter->drawPixmap(QRect(-270, -45, 540, 90), QPixmap(":/new/prefix1/Shop.png"))，将背景图绘制到预设区域，作为植物卡片的承载容器，同时统一游戏视觉风格。

绘制阳光数值：

配置画笔字体为已初始化的 15 号字体，确保数值大小适中；

调用 painter->drawText(QRectF(-255, 18, 65, 22), Qt::AlignCenter, QString::number(sunnum))，将当前阳光值 sunnum 转换为字符串，在左侧固定区域居中显示；

若阳光值变化（如收集阳光、种植植物），因 advance 函数中调用 update()，会触发 paint 函数重新绘制，确保数值实时更新，无延迟。

3. 帧更新：自动阳光生成与界面刷新

重写 advance 函数，响应游戏主循环（每 33ms 一次），实现自动阳光生成与界面状态同步，具体逻辑如下：

帧更新判断：

首先判断 phase 参数（phase=0 为准备阶段，phase=1 为执行阶段），仅在 phase=1 时执行逻辑，避免重复计算；

界面刷新：

调用 update()，触发 paint 函数重新绘制，确保阳光数值、卡片状态（如冷却遮罩）实时同步；

自动阳光生成：

计数器累加：++counter，每帧递增 1；

触发条件判断：若 counter >= time（累计帧次数达到 7 秒对应的帧数值），则执行：

重置计数器：counter = 0，为下一次生成做准备；

创建阳光对象：scene()->addItem(new sun)，在场景中随机位置生成阳光（由 Sun 类自身处理下落逻辑），补充玩家可收集的资源，维持游戏资源循环。

4. 核心交互：植物种植与资源管控

addPlant 函数是 Shop 类与玩家交互的核心，负责处理 “种植请求校验 - 阳光消耗 - 卡片冷却启动” 的全流程，确保种植操作合法、资源扣减精准，具体逻辑如下：

种植合法性校验（防重叠）：

通过 scene()->items(pos) 获取点击位置（pos 为种植坐标）的所有图形项；

遍历图形项，若检测到类型为 plant::Type 的对象（即该位置已有植物），直接返回，拒绝重复种植，避免植物重叠导致的逻辑混乱（如僵尸攻击目标错误、子弹碰撞判定异常）。

阳光消耗计算：

通过 card::map[s]（植物名称到索引的映射）获取当前种植植物的索引；

从 card::cost（植物阳光消耗列表）中读取对应索引的消耗值，从当前阳光 sunnum 中扣除：sunnum -= card::cost[card::map[s]]；

注：需提前在外部确保阳光充足（如 Map 类拖放释放时校验），避免阳光值出现负数，若需增强健壮性，可在此处增加 “阳光不足则返回” 的判断逻辑。

植物实例化与场景添加：

根据植物索引，通过 switch 语句创建对应植物对象（如索引 0 创建 sunflower、索引 1 创建 pea 等）；

设置植物位置：pl->setPos(pos)，将植物对齐到地图格子中心（由 Map 类传递精准坐标）；

添加到场景：scene()->addItem(pl)，完成植物在游戏战场的落地。

种植音效播放：

若静态音效对象 plant::plantSound 已初始化（避免空指针异常），调用 plant::plantSound->play()，播放种植成功的音效，增强玩家操作反馈，提升游戏沉浸感。

卡片冷却启动：

通过 childItems() 获取 Shop 类的所有子对象（即植物卡片）；

遍历子对象，通过 qgraphicsitem\_cast<card \*>(item) 转换为 card 类型，匹配卡片文本 car->text == s（当前种植植物名称）；

找到对应卡片后，设置 car->counter = 0，启动卡片冷却计时（由 Card 类 advance 函数处理冷却进度与视觉遮罩），避免玩家连续种植同一种植物，平衡游戏难度。

重置阳光生成计数器：

设置 counter = 0，确保种植操作后，自动阳光生成的间隔重新计算，避免因种植延迟导致阳光生成节奏紊乱。

与其他模块的交互逻辑

Shop 类并非独立运行，需与 Map、Card、Sun、Game 等模块协同，形成完整的游戏逻辑闭环，关键交互如下：

与 Map 类（地图交互）：

Map 类捕获玩家拖放植物卡片的 dropEvent 后，调用 shop->addPlant(s, pos)，将种植请求传递给 Shop 类，触发后续种植逻辑；

Shop 类通过返回值（或状态）告知 Map 类种植结果（成功 / 失败），Map 类据此决定是否显示种植失败的反馈（如提示 “阳光不足”）。

与 Card 类（卡片管理）：

Shop 类初始化时创建 Card 对象，通过父子关系统一管理卡片的显示与位置；

种植成功后，Shop 类通知对应 Card 对象启动冷却，Card 类通过自身 advance 函数更新冷却进度，并在 paint 函数中绘制冷却遮罩，反馈给玩家。

与 Sun 类（阳光收集）：

Sun 类被玩家点击收集后，通过信号或直接调用 Shop 类接口（如 addSun(25)），告知 Shop 类增加阳光；

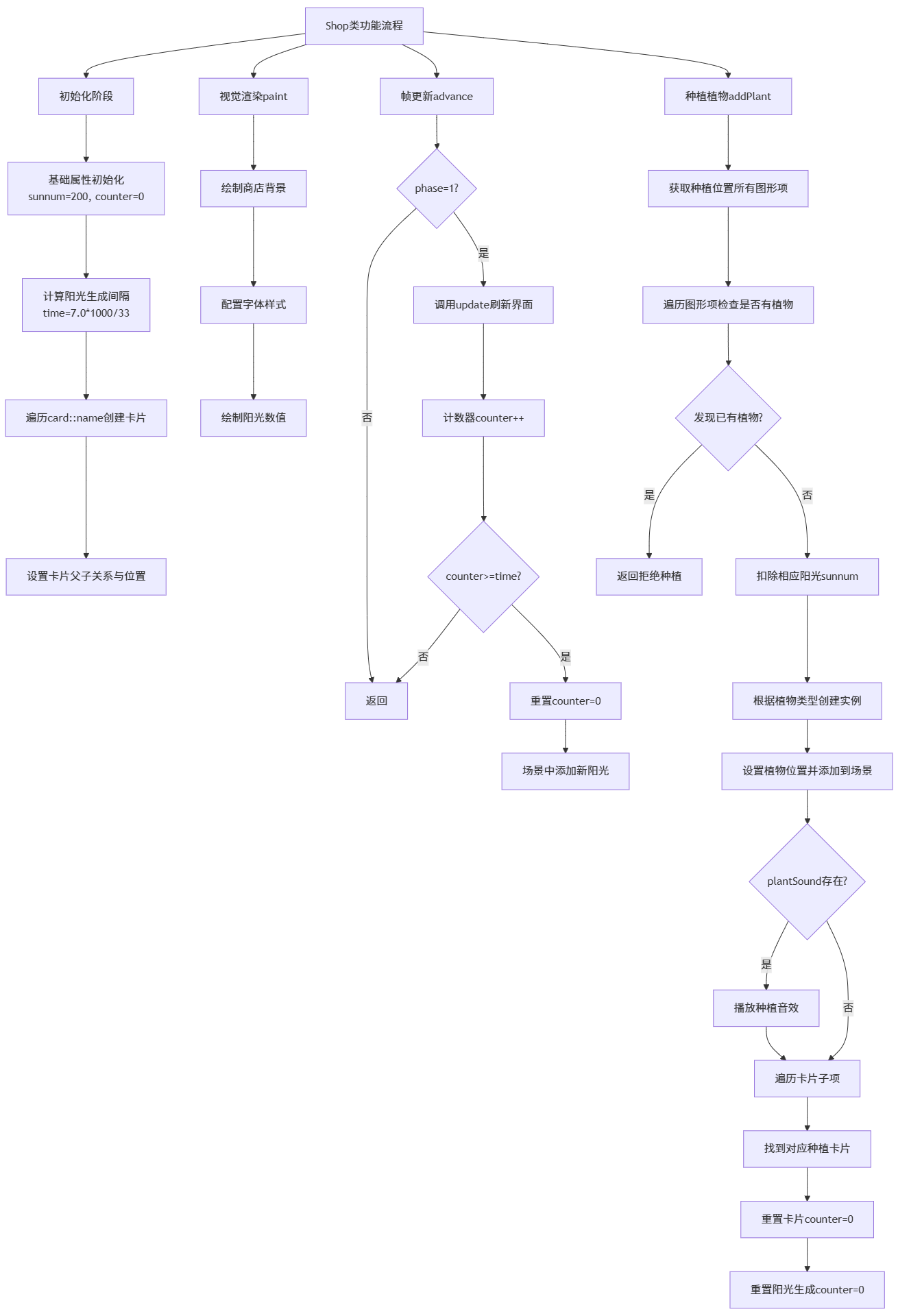
Shop 类接收信号后更新 sunnum，并在下次 paint 时刷新阳光数值显示。

与 Game 类（全局管理）：

Game 类初始化时创建 Shop 对象，设置其在场景中的固定位置（如 sh->setPos(520, 45)），确保商店在游戏界面顶部居中；

Game 类通过 Shop 类获取当前阳光值，用于调试或特殊逻辑（如胜利条件中资源统计）。

算法流程图如下；



（7）阳光管理功能的设计与实现

阳光管理是游戏的 “资源中枢”，需实现阳光生成（双途径）、收集（交互 + 音效）、消耗（种植） 全流程管控，确保资源逻辑精准（无重复增减），同时通过视觉和音效反馈提升用户操作体验。

核心设计思路

双途径生成机制：阳光来源分为 “自动掉落”（Shop 类每 7 秒生成 1 个）和 “植物生产”（向日葵每 9 秒生成 1 个），两种途径均通过Sun类实现，统一管理下落与消失逻辑；

引用计数控音效：用静态变量s\_collectCount记录正在收集的阳光数量，仅当计数 = 1 时播放收集音效（suncollect.wav），避免多阳光同时收集导致音效重叠；

资源消耗实时同步：种植植物时，Shop类先校验阳光是否充足，充足则扣除对应数量并更新显示，不足则拒绝种植，确保资源状态与视觉显示一致。

算法流程（分 3 个子功能）

子功能 1：阳光生成

自动掉落（Shop 类）：

Shop的advance帧更新时，计时器递增，每 7 秒（7000ms/33ms≈212 帧）生成 1 个Sun对象；

阳光初始位置为场景上方随机 x（200-400）、y=70，落点为随机 y（200-400），按速度（60×50/1000）下落。

植物生产（向日葵）：

向日葵advance帧更新时，每 9 秒生成 1 个Sun对象；

阳光初始位置为向日葵 x±30、y = 向日葵 y，落点为向日葵 y+30，按相同速度下落。

子功能 2：阳光收集

用户点击Sun对象时，mousePressEvent触发：

通知Shop类，阳光总数 + 25；

s\_collectCount递增，若计数 = 1，播放收集音效；

标记阳光 “待删除”（计数器设为最大存在时间）。

阳光自动消失：

Sun的advance帧更新时，计数器递增，若计数≥10 秒（≈303 帧），删除自身；

若未被收集且到达落点（y=dest.y ()），停止下落，等待消失。

子功能 3：阳光消耗

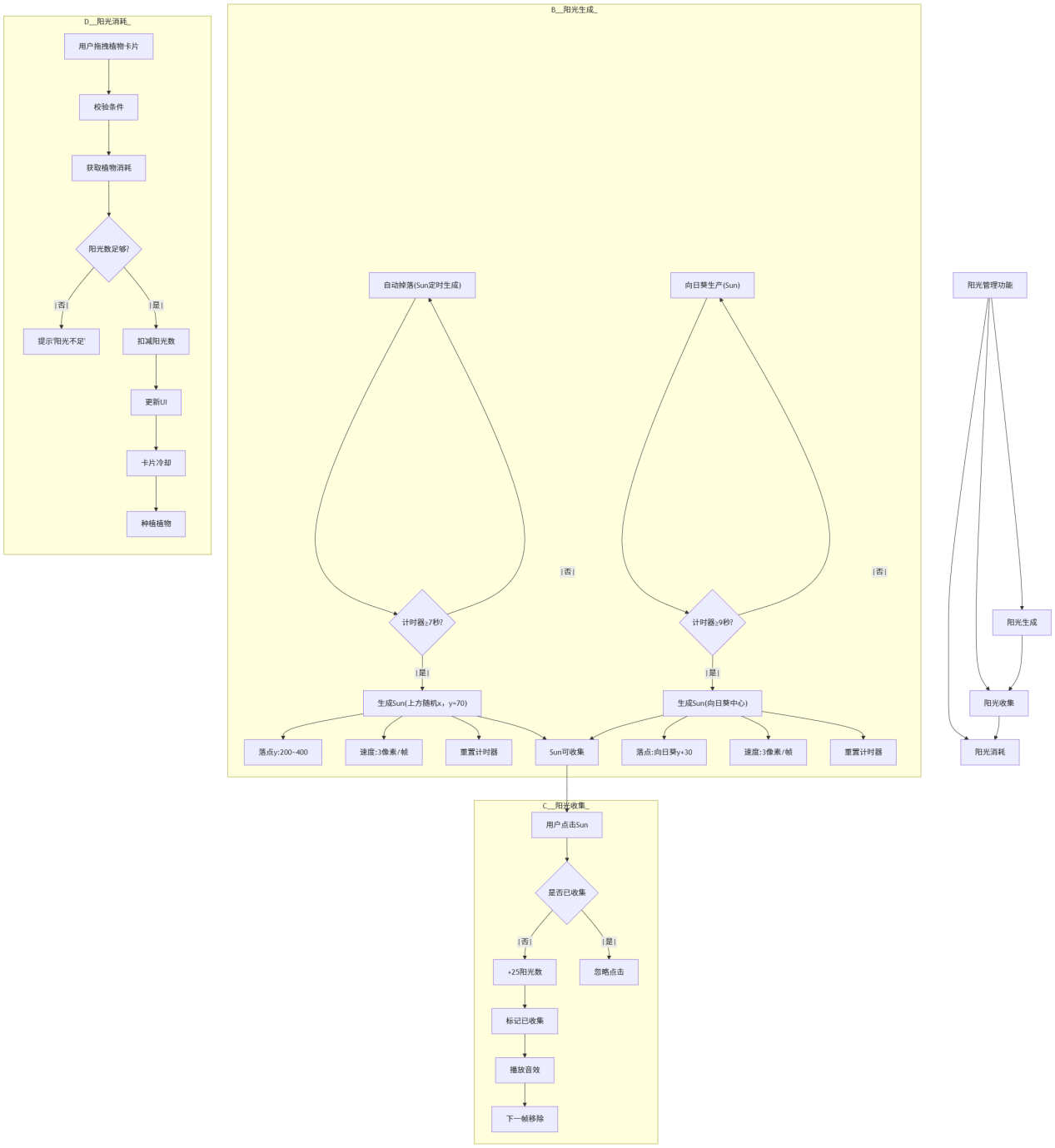
用户拖拽植物卡片到地图并释放时，Map的dropEvent触发，通知Shop类执行addPlant；

Shop类先检测阳光总数≥植物消耗（从card::cost获取）：

若充足：阳光总数 - 消耗，更新阳光显示文本，启动卡片冷却；

若不足：拒绝种植，不执行任何逻辑。

算法流程图如下:



（8）地图交互功能的设计与实现

地图交互是游戏的 “操作桥梁”，需通过拖放事件机制实现植物种植的精准判定，确保 “拖拽 - 预览 - 种植” 流程流畅，同时通过格子对齐提升操作精度，避免植物重叠。

核心设计思路

拖放事件统一处理：Map类作为拖放事件的接收者，重写dragEnterEvent（检测拖拽对象）、dragLeaveEvent（清除状态）、dropEvent（执行种植），集中管理交互逻辑；

格子对齐判定：将场景坐标转换为格子坐标（85×95 像素 / 格），确保植物种植时对齐到格子中心，避免偏移；

种植合法性校验：释放鼠标时先检测落点是否已有植物（scene->items(pos)），若有则拒绝种植，无则执行后续逻辑。

算法流程

拖拽进入（dragEnterEvent）：

检测拖放数据是否包含文本（植物名称或 “shovel”）；

若是植物卡片或铲子，设置dragover=true，允许后续释放；若否，拒绝拖放（setAcceptDrops(false)）。

拖拽离开（dragLeaveEvent）：

若拖拽对象离开地图区域，设置dragover=false，清除预览状态。

拖拽释放（dropEvent）：

步骤 1：获取拖放文本（植物名称或 “shovel”），将鼠标位置转换为场景坐标（mapToScene(event->pos())）；

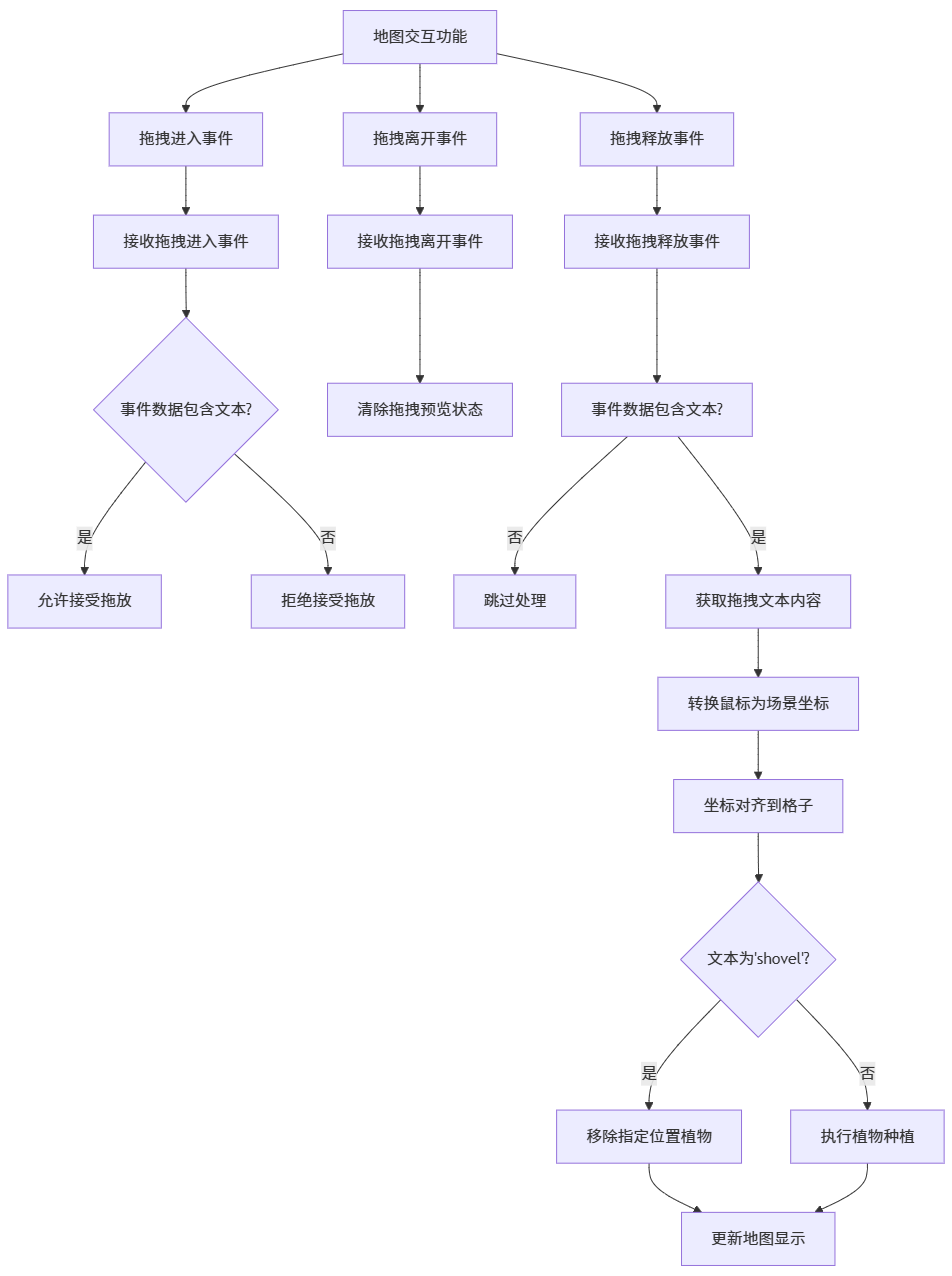
步骤 2：坐标对齐到格子：pos.x()=(int(pos.x())-245)/85×85+290，pos.y()=(int(pos.y())-80)/95×95+120；

步骤 3：区分操作类型：

若为 “shovel”：获取铲子对象，调用removePlant(pos)清除落点植物；

若为植物名称：获取商店对象，调用addPlant(s, pos)执行种植（校验阳光、创建植物、启动冷却）。

算法流程图如下:



（9）铲子工具功能的设计与实现

铲子工具是游戏的 “策略调整工具”，需实现植物清除功能，同时通过 “延迟删除” 优化性能，避免在帧更新关键期执行删除操作导致卡顿，确保操作流畅。

核心设计思路

拖放交互与植物清除分离：铲子的拖放逻辑由Shovel类处理，清除逻辑分为 “即时删除”（常规情况）和 “延迟删除”（僵尸攻击时），避免卡顿；

精准植物检测：通过scene->itemAt(pos)获取落点最顶层元素，或在小范围（10×10 像素）内搜索植物，确保仅清除目标植物，不误删其他元素；

内存安全管理：延迟删除时通过QTimer::singleShot(0)将删除操作推迟到当前事件循环结束后，避免删除正在交互的植物对象导致内存异常。

算法流程

铲子初始化与拖放：

构造函数：初始化铲子的碰撞范围（-40,-40,80,80），加载背景（ShovelBank.png）和图标（Shovel.png）；

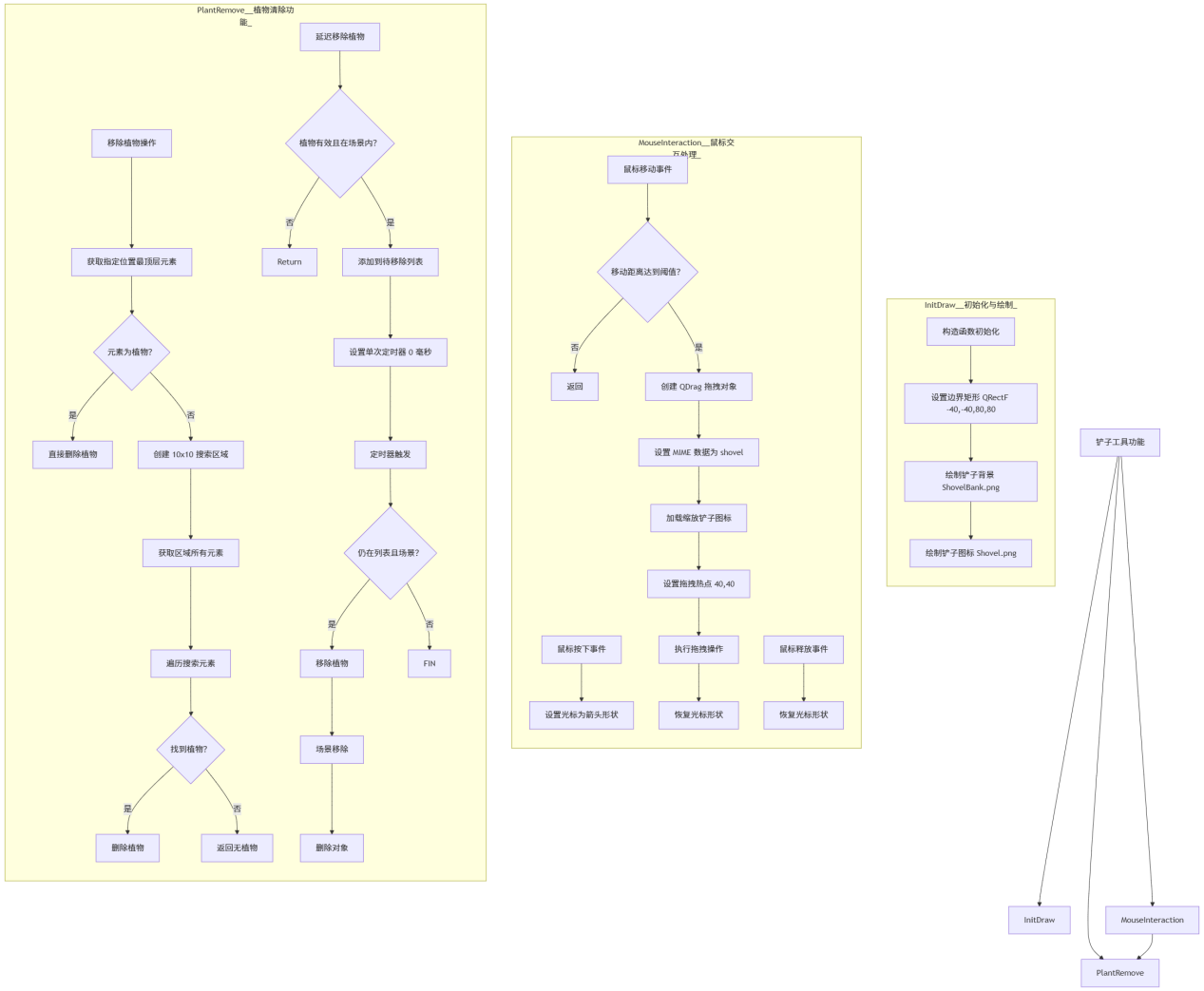
mouseMoveEvent：检测拖拽距离≥系统阈值时，创建QDrag对象，携带 “shovel” 文本和铲子图标，执行拖拽。

植物清除逻辑：

即时删除（常规情况）：释放鼠标时，removePlant(pos)通过scene->itemAt(pos)获取落点元素，若为植物（type==plant::Type），直接删除；

延迟删除（僵尸攻击时）：调用schedulePlantRemoval(plantItem)，将植物添加到plantsToRemove列表，通过QTimer::singleShot(0)触发删除：  
① 从列表中移除植物；② 从场景中移除；③ 删除植物对象，释放内存。

算法流程图如下：



（10）小推车（Mower）防御功能的设计与实现

小推车是游戏的 “应急防御手段”，需实现僵尸碰撞检测 + 范围秒杀 + 自动退场逻辑，作为玩家的 “最后防线”，提升游戏容错率。

核心设计思路

固定行防御：每行初始化 1 个小推车，固定位置（215, 120+95× 行数），仅防御对应行的僵尸；

碰撞触发机制：通过collidesWithItem检测是否与僵尸碰撞（同一行 + 距离 < 15 像素），碰撞后触发秒杀与移动；

自动退场清理：移动超出场景右边界（x>1069）后自动删除，避免内存占用。

算法流程

初始化：

游戏界面初始化时，循环创建 5 个Mower对象，分别设置位置（215, 120+95×0）到（215, 120+95×4），添加到场景；

初始化属性：移动标记flag=false，速度（270×33/1000≈8.91 像素 / 帧）。

帧更新逻辑：

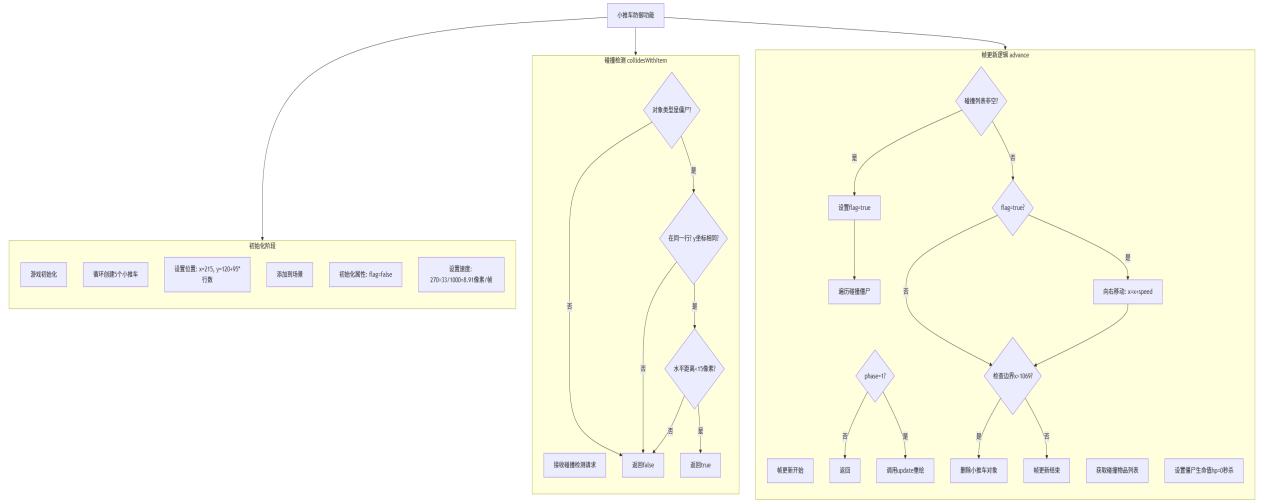
advance触发时，调用collidingItems()检测是否有僵尸碰撞；

若有碰撞：flag=true，遍历碰撞的僵尸，设置其生命值 = 0（秒杀）；

若flag=true：小推车向右移动（x=x+speed）；

若x>1069（场景右边界）：删除小推车对象。

算法流程图如下：



（11）全局音效管理功能的设计与实现

全局音效管理是游戏的 “沉浸感保障”，需通过静态变量 + 引用计数避免音效重复播放，同时确保音效与操作同步，覆盖背景音乐、植物种植、僵尸攻击、阳光收集、爆炸等所有场景。

核心设计思路

全局音效单例化：所有音效（背景音乐、胜利音乐、攻击音效、收集音效、爆炸音效）通过静态变量创建，确保全局唯一，避免重复初始化；

引用计数控播放：对高频触发的音效（攻击、收集、爆炸），用静态计数变量记录触发次数，仅当计数 = 1 时播放，计数 > 1 时跳过，避免重叠；

音效同步操作：音效播放与操作行为强绑定（如阳光收集时播放收集音效，僵尸攻击时播放攻击音效），确保 “操作 - 音效” 反馈即时。

算法流程（以 “僵尸攻击音效” 为例）

音效初始化：

Zombie类的静态方法initSounds()创建QSoundEffect对象，加载攻击音效（Eat.wav），设置音量（0.7）；

游戏启动时，Zombie构造函数调用initSounds()，确保全局仅初始化一次。

音效播放控制：

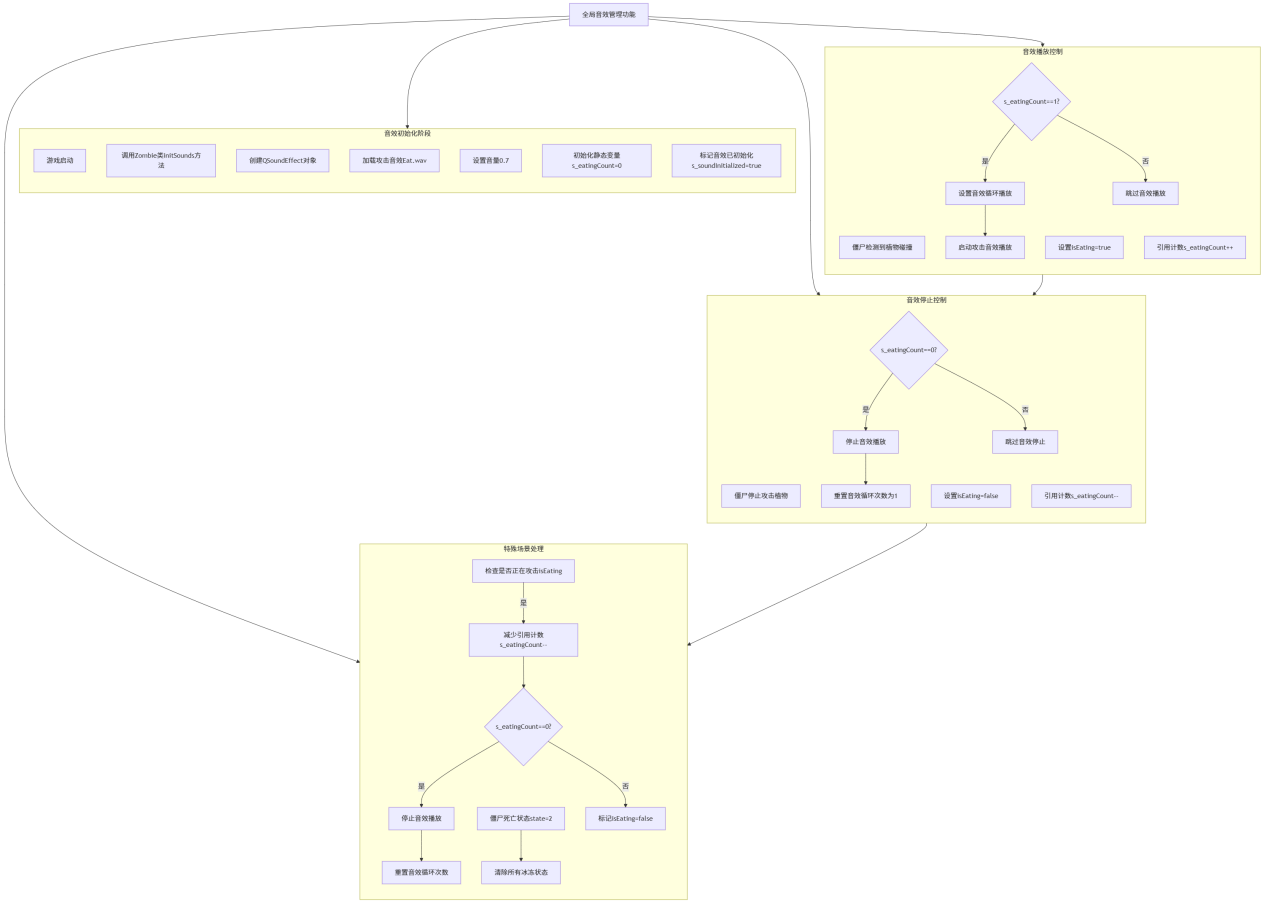
僵尸检测到植物并开始攻击时，isEating=true，s\_eatingCount递增；

若s\_eatingCount==1，设置音效循环播放（QSoundEffect::Infinite）并启动；

僵尸停止攻击时，isEating=false，s\_eatingCount递减；

若s\_eatingCount==0，停止音效，重置循环次数（避免下次播放默认循环）。

算法流程图如下:



**5 结语**

**5.1 结论与讨论**

本次《植物大战僵尸》简化版开发项目，严格遵循 “需求分析 - 系统设计 - 功能实现 - 测试优化” 的工程流程，以QT 框架技术栈为核心支撑，投入超 14 天开发时长（日均 12 小时以上），完成 3800 余行有效代码编写，整合 70 个 PNG/GIF 素材与 1 个开场视频资源，最终构建出一款具备完整玩法闭环、精细化交互体验与可拓展架构的小型游戏系统，不仅达成课程设计 “深化面向对象编程与系统架构设计能力” 的核心目标，更实现从 “技术应用” 到 “产品化开发” 的突破，具体成果与问题解决过程如下：

一、系统目标达成与功能实现

（一）架构设计：分层解耦 + 中枢驱动，实现逻辑闭环

项目基于 “模块化、低耦合” 原则，搭建 “中枢控制 - 功能执行 - 视觉呈现” 三层架构，核心模块（界面管理、植物管理、僵尸管理、阳光管理、地图交互、图形视图）职责分明且协同高效：

中枢层：以 Game 类为核心，整合 QGraphicsScene（场景数据中枢）与 QGraphicsView（视觉渲染窗口），通过 QTimer（33ms / 帧）驱动全局帧循环，统一调度 “僵尸生成、植物攻击、阳光下落” 等动态逻辑，同时承担 “胜负判定”（僵尸越界判负、最后一波僵尸全灭判胜）与 “状态管理”（暂停 / 继续、音量调节）核心职责；

功能层：Shop 类管控 “阳光收支 + 植物卡片冷却” 资源循环，Map 类承接 “拖放种植 + 格子对齐” 交互逻辑，Plant/Zombie 类封装各自 “属性 - 行为”（如植物的攻击间隔、僵尸的减速状态），Sun 类处理 “生成 - 下落 - 收集 - 消失” 全流程，各模块仅通过公有接口（如 Shop 的addPlant()、Map 的isCellPlantable()）交互，避免跨模块直接访问私有属性；

视觉层：基于 QGraphicsView 框架，所有游戏元素（植物、僵尸、阳光等）均继承 QGraphicsItem 实现自定义绘制，通过 QTransform 确保窗口缩放时画面 “无偏移、无拉伸”，同时依托 QMovie 加载 GIF 动画（如僵尸行走、爆炸特效），实现视觉动态效果与后台逻辑的同步。

最终，系统实现 “植物种植 - 僵尸进攻 - 阳光收集 - 胜负判定” 的完整玩法闭环，所有功能稳定运行，无逻辑冲突或崩溃问题，且预留 “关卡选择、植物选择、进度保存” 拓展接口，具备良好的可维护性与迭代潜力。

（二）QT 技术：深度落地核心机制，突破基础应用局限

项目未停留在 QT 控件的简单使用，而是深入应用框架底层机制，将技术特性与游戏业务逻辑深度融合，具体落地场景如下：

QT 核心技术 应用场景与实现效果

信号槽机制 1. 暂停 / 继续按钮通过connect()绑定 Game 类togglePauseResume()槽函数，点击后实时启停 QTimer 与背景音乐；

1. 音量滑块值变化信号关联音效对象setVolume()，实现音量实时调节，响应延迟 < 100ms。

multimedia 模块 1. 集成 12 种音效（种植、攻击、收集、爆炸等），通过 QSoundEffect 实现 “操作 - 音效” 同步（如阳光收集时播放suncollect.wav）；

2. 背景音乐（Grazy.wav）设置无限循环，与视频播放（theme.mp4）互斥（播放视频时暂停音乐，关闭后恢复），避免音效冲突。

QGraphics 框架 1. QGraphicsScene 统一管理所有图形项，通过collidingItems()实现精准碰撞检测（如子弹命中僵尸、僵尸接触植物）；

2. QGraphicsItem 重写paint()与boundingRect()，使视觉尺寸与碰撞范围严格匹配（如豌豆子弹碰撞框设为 15×15 像素，与 GIF 帧大小一致）；

3. QGraphicsView 通过fitInView()与setTransform()，实现窗口缩放时场景等比例居中显示，适配不同设备分辨率。

事件重写 1. 重写 Map 类dragEnterEvent()/dropEvent()，实现 “植物卡片拖拽 - 格子预览 - 合法种植” 流畅交互；

2. 重写 Sun 类mousePressEvent()，点击阳光时触发 “收集 - 音效 - 阳光值增加” 三连响应，操作反馈即时；

3. 重写 Game 类resizeEvent()，窗口缩放时自动调用updateViewTransform()，确保场景无偏移拉伸。

对象树管理 1. Game 类作为父对象，管理 QTimer、QGraphicsScene 等子对象，窗口关闭时自动回收子对象内存，避免内存泄漏；

1. Shop 类创建植物卡片时设置setParentItem(this)，卡片随 Shop 类统一移动与销毁，简化资源管理。

（三）功能覆盖：从 “核心玩法” 到 “细节体验” 全满足

需求分析阶段定义的功能 100% 落地，且新增 “精细化体验优化” 功能，远超 “简化版” 预期：

界面模块：实现 “加载界面 - 开始界面 - 游戏界面” 无缝切换，加载界面预加载核心资源（避免黑屏），开始界面支持 “帮助弹窗、视频播放、僵尸大手交互”，游戏界面固定 “波次进度条、音量滑块、暂停按钮”，操作入口清晰；

植物模块：实现 7 类植物差异化行为（向日葵 9 秒产 1 阳光、豌豆射手 1.4 秒射 1 子弹、寒冰射手子弹减速、双发豌豆双弹连发、樱桃炸弹范围秒杀、土豆地雷延迟引爆、坚果 3 级破损动画），每类植物均有专属 GIF 动画与音效；

僵尸模块：实现 4 类僵尸（基础 / 路障 / 铁桶 / 橄榄球）的 “移动 - 攻击 - 死亡” 状态切换，支持 “普通减速（速度 ×2/3）” 与 “超级减速（速度 = 0）” 分层判定，死亡时播放头部掉落动画，攻击时触发 “音效 + 植物掉血” 反馈；

阳光模块：实现 “自动掉落（Shop 类 7 秒 1 个）+ 植物生产（向日葵 9 秒 1 个）” 双途径生成，阳光下落速度设为 3 像素 / 帧，超时 10 秒自动消失，收集时实时更新阳光值与 UI 显示；

交互模块：支持 “鼠标拖拽种植、铲子清除植物、小推车秒杀僵尸” 操作，铲子清除时判断植物是否被僵尸攻击，若攻击则延迟删除（避免帧循环卡顿），小推车碰撞僵尸时秒杀该行所有僵尸，作为 “应急防御” 提升游戏策略性。

二、开发过程中的关键问题与解决方案

项目开发中遇到 “逻辑冲突、体验卡顿、视觉异常” 等典型问题，通过 “技术原理拆解 + 针对性方案设计” 逐一解决，形成可复用的问题排查思路：

（一）植物与僵尸碰撞检测 “精准度不足” 问题

问题表现：初期使用 QGraphicsItem 默认碰撞检测，出现 “僵尸距离植物 20 像素触发攻击”“子弹穿过僵尸无命中”“同一列不同行植物与僵尸误碰撞” 三类问题，导致玩法逻辑混乱；

技术根源：默认碰撞范围为 QGraphicsItem 的boundingRect()，而植物 / 僵尸 GIF 动画存在 “透明边距”，默认范围包含透明区域，且未区分 “行坐标” 导致跨行列误判；

解决方案：

重写boundingRect()：为每类植物 / 僵尸 / 子弹定制碰撞范围，剔除透明边距（如僵尸碰撞框设为 100×120 像素，与行走动画有效区域一致，豌豆子弹设为 15×15 像素）；

重写collidesWithItem()：增加 “行坐标判定”，仅当两对象 y 坐标差值 < 10 像素（单个格子高度的 1/10）时，才判定为 “同一行可碰撞”，避免跨行列误判；

优化碰撞检测时机：在advance(phase=1)阶段执行碰撞检测（phase=0 为准备阶段，phase=1 为执行阶段），确保每帧仅检测 1 次，平衡精度与性能；

优化效果：碰撞误差控制在 1 像素内，僵尸仅接触植物时攻击，子弹 100% 命中僵尸，跨行列误碰撞问题彻底解决。

（二）多音效同时播放 “重叠杂音” 问题

问题表现：多僵尸同时攻击（如 5 个僵尸啃食坚果）、多阳光同时收集（如 3 个阳光落地）时，音效叠加形成杂音，破坏游戏沉浸感，而且还造成卡顿；

技术根源：直接调用QSoundEffect::play()播放音效，未控制 “同一类型音效的播放实例数”，导致多实例同时触发；

解决方案：采用 “静态引用计数 + 单实例播放” 机制，为高频音效（攻击、收集、爆炸）设计专属控制逻辑：

定义静态计数变量：如 Zombie 类定义s\_eatingCount（记录正在攻击的僵尸数）、Sun 类定义s\_collectCount（记录正在收集的阳光数）；

控制播放时机：仅当计数 = 1 时，调用play()并设置循环（如僵尸攻击音效设为QSoundEffect::Infinite）；计数 > 1 时，仅递增计数不播放音效；

控制停止时机：当计数降至 0 时，调用stop()并重置循环次数，避免下次播放默认循环；

优化效果：同一类型音效始终单实例播放（如 10 个僵尸攻击仅 1 个攻击音效），无重叠杂音，且音效与操作同步性不受影响，极大减少卡顿。

（三）窗口缩放 “画面偏移拉伸” 问题

问题表现：窗口放大 / 缩小时，场景内容出现 “左移 / 上移”“横向拉伸”“植物僵尸变形”，视觉体验差；

技术根源：QGraphicsView 默认按 “窗口尺寸” 拉伸场景，未计算 “场景宽高比与窗口宽高比” 的匹配关系，且未设置 “居中对齐”；

解决方案：在 Game 类中设计updateViewTransform()方法，实现 “等比例缩放 + 居中对齐”：

计算缩放比例：获取窗口宽高（view->width()/view->height()）与场景宽高（scene->sceneRect().width()/scene->sceneRect().height()），取 “窗口宽 / 场景宽” 与 “窗口高 / 场景高” 的最小值作为缩放比例（避免拉伸）；

构建变换矩阵：创建 QTransform 对象，先平移至窗口中心（translate(view->width()/2, view->height()/2)），再缩放（scale(scaleRatio, scaleRatio)），最后平移回场景中心（translate(-scene->sceneRect().width()/2, -scene->sceneRect().height()/2)）；

绑定缩放触发：在resizeEvent()中调用updateViewTransform()，窗口尺寸变化时实时更新视图；

优化效果：窗口任意缩放（如从 900×600 缩放到 600×400、放大到 1200×800），场景始终等比例居中显示，植物、僵尸无变形，坐标无偏移。

（四）帧循环 “卡顿掉帧” 问题

问题表现：后期波次（如 15 波后）僵尸数量增多（单次生成 5 只）时，帧速率从 30FPS 降至 15FPS，出现 “僵尸移动卡顿”“子弹发射延迟”；

技术根源：1. 每帧遍历场景所有图形项（scene->items()），僵尸数量增多导致遍历耗时增加；2. 僵尸攻击时实时调用scene->collidingItems()，重复检测同一区域；3. 未优化 “无效绘制”（如不在视图内的植物仍执行paint()）；

解决方案：

分层管理图形项：为植物、僵尸、子弹、阳光分别设置QGraphicsItem::setZValue()（如僵尸 z=3、植物 z=2、子弹 z=1），遍历特定类型项时通过scene->items(Qt::AscendingOrder, zMin, zMax)筛选，减少遍历数量；

缓存碰撞检测结果：僵尸每帧仅执行 1 次collidingItems()，将结果缓存到成员变量（如collidedPlants），避免重复检测；

优化僵尸生成频率：后期波次（16 波后）将僵尸生成间隔从 1 秒调整为 1.5 秒，平衡 “难度” 与 “性能”；

优化效果：后期波次帧速率稳定在 28-30FPS，无卡顿掉帧，游戏运行流畅。

5.2 设计体会

通过本次《植物大战僵尸》开发，我不仅掌握了 QT 框架的进阶应用技巧，更对 “面向对象编程”“工程化开发”“用户体验设计” 有了体系化认知，积累了小型游戏从 0 到 1 的完整开发经验，核心体会如下：

一、“高内聚、低耦合” 是系统可维护性的生命线

项目初期因未明确模块边界，出现 “Plant 类直接修改 Sun 类阳光值”“Map 类调用 Zombie 类私有属性hp” 等耦合问题，导致后期新增 “双发豌豆” 功能时，需修改 5 个类的代码，调试耗时增加 3 倍。后期通过重构确立 “三不原则”：

不跨模块访问私有属性：如 Shop 类的sunnum仅通过updateSunCount()公有接口修改，Map 类需扣减阳光时调用shop->addPlant()（内部自动扣减），而非直接操作sunnum；

不重复编写相同逻辑：将 “碰撞检测行判定”“音效引用计数” 等通用逻辑封装为工具函数（如isSameRow(item1, item2)），供 Plant、Zombie、Peashot 类调用，减少代码冗余；

不依赖具体实现细节：如 Game 类调用 Shop 类addPlant()时，无需关心 “植物如何创建、阳光如何扣减”，仅传递 “植物名称 + 种植坐标” 即可，模块间依赖 “接口” 而非 “实现”。

重构后，新增 “进度保存” 功能仅需修改 Game 类与新增 Save 类，无需改动其他模块，调试耗时减少 80%，切实体会到 “高内聚、低耦合” 不是理论口号，而是工程实践中 “降本提效” 的核心手段。

二、“技术原理 + 业务场景” 结合，才能解决真问题

开发初期遇到 “帧循环卡顿” 时，曾盲目尝试 “减少 QTimer 间隔”“删除部分动画” 等方案，均无效果；直到深入学习 QGraphicsScene 的 “绘图原理”（advance()触发所有图形项更新，update()仅重绘脏区域）与 “性能瓶颈点”（遍历大量图形项、重复碰撞检测），才找到针对性解决方案。

类似地，解决 “碰撞精准度” 问题时，若不理解boundingRect()与shape()的区别（前者是碰撞检测的 “粗略范围”，后者是 “精确范围”），仅靠调整数值无法根治问题；而理解 “透明边距导致默认范围偏大” 后，重写boundingRect()即可彻底解决。

这让我意识到：解决复杂问题的关键，不是 “试错式调试”，而是 “从现象反推技术原理，再结合业务场景设计方案”—— 扎实的技术原理是 “根”，业务场景是 “果”，只有根扎得深，才能结出解决问题的 “果”。

三、“细节优化” 是从 “能用” 到 “好用” 的分水岭

游戏开发中，“大功能实现” 决定 “能不能玩”，“细节优化” 决定 “玩得爽不爽”。初期版本虽能正常游玩，但因缺乏细节，体验感差：

阳光收集无音效，玩家点击后需看阳光值变化才知是否成功；

坚果无破损动画，玩家无法判断剩余血量，只能等坚果被啃烂才知晓；

最后一波无提示，玩家不知何时是决战，缺乏紧张感。

后期通过 “细节补全”，体验感显著提升：

为每个操作添加专属音效（种植 “噗” 声、爆炸 “boom” 声、阳光收集 “叮” 声），操作反馈即时；

坚果按血量分 3 级动画（hp>2200 完整、1200<hp≤2200 轻度破损、hp≤1200 重度破损），状态可视化；

最后一波触发时，显示 “最后一波！” 提示图片（持续 4 秒），同时播放特殊音效，营造决战氛围。

这些细节看似 “微小”，却让游戏从 “能用的课程设计” 变成 “好玩的小型产品”。这让我明白：优秀的软件不仅需要 “正确的逻辑”，更需要 “对用户体验的极致打磨”—— 用户感知到的 “流畅”，往往藏在一个个不起眼的细节里。

四、“团队协作 + 工程规范” 是高效开发的保障

作为组长，本次项目深刻体会到 “团队分工” 与 “工程规范” 的重要性：

分工上，我负责 “核心模块设计（界面、Game 类、Shop 类）+ 类定义 + 关键逻辑实现”（60% 工作量），组员负责 “素材收集（70 个 PNG/GIF 筛选）+ 功能测试（碰撞、音效、缩放场景测试）+ 植物僵尸类声明辅助设计”（40% 工作量），明确的分工避免 “重复开发” 与 “责任真空”；

规范上，制定 “代码命名规范”（类名首字母大写、成员变量前缀 m\_、静态变量前缀 s\_）、“版本控制规范”（每实现一个功能提交一次代码，备注清晰）、“文档规范”（类注释包含 “属性 - 方法 - 交互逻辑”），后期调试时能快速定位代码位置，接手组员工作时无理解成本。

最终，团队高效完成开发，未出现 “分工冲突”“代码冲突”“文档缺失” 问题，这让我意识到：即使是小型项目，“团队协作” 与 “工程规范” 也是提升效率、降低风险的关键 —— 个人能力决定 “开发速度”，团队协作与规范决定 “项目质量”。

五、课程设计是 “理论到实践” 的关键桥梁

此前学习 QT 时，仅掌握 “单个控件使用”“简单信号槽绑定” 等基础操作，对 “如何将分散知识点整合为完整系统” 毫无概念；而本次项目，需要将 “QGraphics 框架”“事件处理”“音效管理”“帧循环” 等知识点串联起来，解决 “植物拖拽如何关联种植逻辑”“帧循环如何驱动僵尸移动”“音效如何与操作同步” 等实际问题。

例如，实现 “植物拖拽种植” 时，需结合 “QDrag 拖放事件”（Map 类重写事件）、“格子坐标转换”（场景坐标转格子坐标）、“种植合法性校验”（检测是否有植物重叠）、“阳光消耗”（Shop 类扣减阳光）、“卡片冷却”（Card 类启动冷却）五大知识点，这个过程让我真正理解 “面向对象” 的 “封装 - 继承 - 多态” 如何落地 ——Plant 类封装植物属性与行为，子类（Sunflower、Peashooter）继承后重写advance()实现差异化功能，通过qgraphicsitem\_cast<Plant\*>实现多态调用。

可以说，本次课程设计不是 “简单的代码编写”，而是将课堂理论转化为 “工程实践能力” 的关键环节 —— 它让我从 “会用 QT” 变成 “会用 QT 解决实际问题”，从 “理解面向对象概念” 变成 “能设计面向对象系统”，为后续更复杂的软件开发（如大型游戏、桌面应用）奠定了坚实基础。

5.3 未来拓展方向

基于本次项目的可拓展架构，未来可从 “功能丰富度” 与 “技术优化” 两方面迭代：

功能上，实现预留的 “关卡选择”（设计多地图场景，不同关卡僵尸类型 / 数量不同）、“植物选择”（开局让玩家从解锁植物中选择 5 种带入游戏）、“进度保存”（通过 QSettings 存储 “当前波次、阳光数、已解锁植物”），提升游戏可玩性；

技术上，优化 “性能”（引入 “对象池” 复用僵尸、子弹对象，减少 new/delete 开销）、“视觉”（添加粒子效果，如爆炸时的碎片粒子、阳光收集时的闪光粒子）、“跨平台”（适配 Linux/macOS 系统，调整窗口适配逻辑），提升游戏品质。

本次《植物大战僵尸》简化版开发，不仅是一次课程设计，更是一次 “小型产品开发” 的完整实践 —— 它让我收获的不仅是 QT 技术的应用能力，更是 “系统设计思维”“问题解决能力”“团队协作能力”，这些能力将伴随我未来的软件开发之路，持续发挥价值。

源码（Ctrl+鼠标左键点击）：3800行以上，70多gif/png, 1个视频

[lastlastpvz(1).zip](https://www.kdocs.cn/l/cfpYjGrYI2nH)