1 系统需求分析

当可怕的僵尸入侵的时候，谁能想到，弱小的植物能承担起保卫家园的责任？也许你早听 说过那些没有脑子的僵尸席卷整个街区、吃掉一户一户屋主人的脑子的故事。但这次，你终于 要真正面对入侵自己家园的僵尸了。有僵尸在你的草坪上！

草坪是植物们的领地。面对疯狂践踏的入侵者，植物们不惜以自己的身躯阻挡僵尸前进 的道路。可是僵尸的利齿不仅可以用来吃掉脑子，还可以用来吃掉植物。面对挡路的一切东西， 僵尸们选择：吃掉它们！

用什么来保卫你自己的家园和脑子呢？我们只有勇气，和自己手中的那把小铲子。

试设计游戏，并实现以下功能：

- 拥有图形界面，如游戏主界面和准备界面
- 能完成游戏操作，如放置植物，删除植物，生成僵尸，攻击等操作
- 正确且合理的胜负判断
- 实现卡牌冷却效果
- 两个场景的选择，晴天和黑夜
- 背景音效的实现，如碰撞，放置植物以及收集阳光等
- 正确显示阳光槽，植物卡牌槽等部件
- 实现九种植物和七种僵尸
- 实现调试功能，按下一些按键可以执行一些特殊操作，比如快速产生僵尸和增加阳光
- 实现代码层级的一些基本要求，如面向对象的继承派生多态等机制，文件操作（读取 媒体资源文件，读取记录用户名和游戏最长时间的文本文件），一些基本数据结构的使用 （如 list,map,set）等

2 总体设计

植物大战僵尸是一款游戏非常经典的益智类游戏，画风友好，玩法多样，整体逻辑比较 清晰。本次我选择使用 c++的 GUI 库 Qt 来写复现游戏。总体上来说完成了游戏加载界面，菜单选择界面，主游戏逻辑和交互界面和游戏的基本功能。由于手头的资源有限（比如植物 和僵尸的贴图）以及时间等方面的限制，本游戏实现了了七种僵尸和九种植物，以及白天和 黑夜两个场景模式，具有了一定的可玩性。

首先谈一下我在初期设计的主要想法。我认为，同为程序，不同种类的程序的开发思路 是截然不同的。比如设计一个数据管理系统和游戏的思路是截然不同的。我认为游戏的最 大特点在于它是一种交互性极强的产品形式，而且一般工程量较大，需要兼顾内在逻辑和 外在用户交互界面，并且由于和用户的交互性极强，操作多样化，非常适合面向对象的开发 思想。因此本次自选题我选择了实现植物大战僵尸这一经典的游戏。

基于自己的水平考虑，我选择了非常常规的经典面向对象的思路，定义良好的复用性强 的基类，如果想要添加新的对象和功能，只需要从基类派生并将新类加入生成器。在逻辑和 UI 界面的关系选择上，让每个游戏对象负责处理自己的逻辑和动画，而不是将逻辑和画面 完全分开，每回合逻辑执行完成后渲染画面。因为前者虽然效率低了一些，但是总体上说更 符合人的思路，对前期的架构设计要求也没有那么高，可以后期逐步完善，拓展性更强一些。

而在要求上，我完成了必须要求

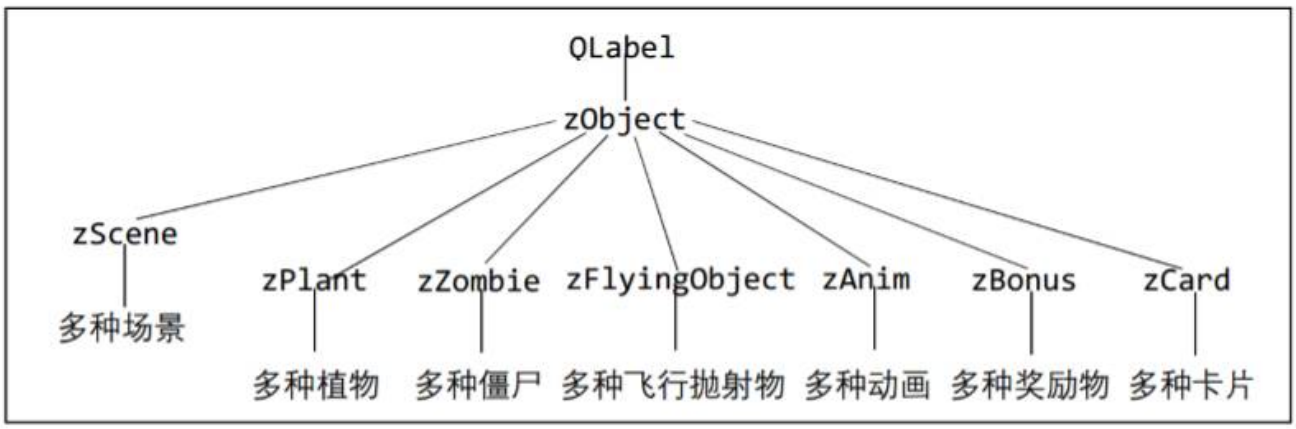
- 函数重载
- 类的继承和抽象基类
- 动态分配 内存和文件操作
- 使用了 list 和 set 等数据结构

总体上来说，需要实现的模块和功能有：

- **渲染场景**：包括加载欢迎界面和游戏主界面
- **多种植物**：游戏的主要实体对象之一
- **多种僵尸**：游戏的主要实体对象之一
- **多种飞行物**：如豌豆、孢子等，也是游戏的重要对象
- **多种动画**：在游戏中需要根据实物的状态来更新各种动画
- **奖励物**：阳光等
- **多种卡片**：主要指的是种植植物是选择的卡片，需要完成冷却功能

考虑到他们都具有相同的一些特性（例如都是实体对象，都有贴图，位置，自己的主逻辑函 数函数），而且为了方便主逻辑中队每个对象的遍历（如更新状态，删除对象），因此我定义了 zObjcet 类作为共同的基类，由 zObject 类派生出其余的类别。而 zObject 类本身由 Qt 的基本 类别 QLabel 派生，因为 QLabel 本身属于最简单的基本对象，而且具有贴图，播放动图，设置 大小等基本功能，非常适合派生出其他对象并在此基础上增加功能。

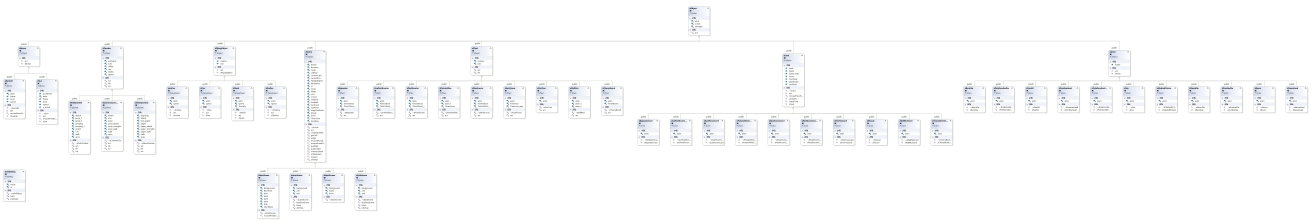
游戏的类大致结构如下图（具体结构见下方 UML 图），接下来将详细介绍程序的内容。



3 详细设计

3.1 基本对象 —— 架构、类的派生关系

本游戏的 UML（缩略图）如下：



本程序一共定义了 52 个类，除了一个继承自 Qt 最基本空间 QDialog 的 mianDialog 类用于 生成游戏基本图形界面框，其余所有类都继承自 zObject 类，而 zObjcet 类自身继承自 Qt 自带 基本类型 QLabel。

按照功能，zObject 类直接派生出七个类，分别对应上文所说的七种实体类别和功能，七个 大类分别再派生出具体的小类别，如 zPlant 类下派生出 9 种具体的植物类，而七种基类不参与 具体的对象实例化，这样的优点是逻辑清楚，方便管理。下面我将主要介绍基类和其派生出的 七个基本大类。

3.1.1 基类：zObject

其代码实现如下：

```
class zObject : public QLabel
{
    Q_OBJECT
public:
    explicit zObject(QWidget* parent = 0);
    bool alive = true;
    virtual void act()=0;
    int strength = 1;
    zScene* scene;
};
```

基类 zObject 自派生 QLabel，因为 QLabel 非常合适定义基本实体：QLabel 的 SetMovie 配合 QMovie 是显示游戏动画（素材为 gif 格式）的便捷利器；而且 QLabel 也没有什么 多余的属性与方法，基本来自 QWidget，只是一个普通的窗口组件。

bool alive，表示它是否活着。本属性不一定表示通俗意义的“活着”，比如一个动 画播放完了，我们就可以把 alive 置成 false，等着逻辑来把它删掉，可以用将本属性当作 标记，交给逻辑中专门的死亡处理机制，从外部释放掉内存空间，并把对象删除。

virtual void act()=0。这个虚函数就是游戏内所有对象的核心逻辑函数。主逻辑的重要功 能就是调用场上一切“活着”的对象的 act()来构成整体逻辑。

int strength 表示生命值。只有植物和僵尸有这样的属性，但是还是写进了接口。其余对 象默认置 1 即可。

zScene* scene 提供了一个指向自己所处的 zScene 的指针，便于 act()逻辑来访 问一些全局的对象。zScene 是游戏场景类，也由 zObject 派生。由于 parent 指针指向的 是 QWidget 而不是 zScene，此处 parent 指针没法优雅的访问 zScene 的一些特殊属性。parent 强制转化为 scene 以后就没有了这个问题。

3.1.2 场景管理类：zScene

zScene 是负责管理其他类的类，在它的属性里面有包含其余类对象的容器 QList，而其 余类也有指向所在场景的指针。ZScene 派生出四个具体类：zStartScene, zStartScreen, zLawnScene, zDarkScene，分别对应开始界面，开始选关界面，白天关卡界面，和夜晚关卡 界面。

3.1.3 主要实体类：zPlant 类和 zZombie 类

这两个类具有很多相似的之处，因此放在一起介绍。zPlant 和 zZombie 是两种基本的对象， 负责派生出各种丰富多彩的植物和僵尸。其共同点是都有 virtual void hit(int damage, bool silence = true)方法，供其余对象调用，对它产生伤害。植物有表示它所处网格位置的 int raw, column，僵尸有表示它在某行上的一维位置的 double xpos，另外僵尸还有诸多特殊状态都写在 了僵尸基类里面，比如冰冻。

3.1.4 飞行抛射类：zFlyingObjcet 类

飞行抛射物类，其特点是不断被发射和碰撞判定，代表物有豌豆、火球、孢子（小蘑菇 发射的）等。其 act()较为复杂，且有对僵尸 hit()方法的直接调用。目前碰撞判定采取的是 纯一维逻辑，只判定同行上的碰撞。这里对于原游戏做了一些简化，未加入杨桃这种可以向 五个方向发射小星星的机制。

3.1.5 动画类：zAnim 类

zAnim 动画类较为简单，只负责播放动画，放完就自动销毁，没有任何逻辑处理内容。zAnim 的用途十分广泛，任何逻辑执行时都可能抛出一个或多个动画，用来可视化的展现逻辑的效果。 而动画本身用快速更新贴图和一些 gif 实现。

3.1.6 奖励物：zBonus 类

特点是响应用户的鼠标点击，目前主要是阳光，可以很容易的拓展出金币等奖励物对象。

3.1.7 卡牌类：zCard 类：

放在游戏画面最左侧的植物卡槽。包括铲子也是一种卡牌。具有冷却时间、判定阳光消 耗等功能。并且由拖动放置的效果。

3.2 多媒体 —— 丰富多彩的效果实现

多媒体的实现方式主要有两方面：声音 QSound 与画面 QMovie。 QSound::play(char* path)很方便的实现了异步音效播放，只需要把 wav 格式的音效加入 qrc 资源文件，非常方便易用。由于 wav 格式的体积过大，我没有加入背景音乐播放的功能， 只实现了碰撞，射击等游戏内音效。

QMovie 需要挂载在 QLabel 及其派生类的对象下才能播放，而由于 QObject 类派生自 生自 QLable,因此很容易实现播放动画。在表现僵尸被冰冻减速的时候，还使用了 QMovie::setSpeed(50)使动画的播放速度减半，直观的呈现冰冻。

卡牌的冷却效果仅仅是使用了两层半透明 QWidget，这部分逻辑写在 zCard 的 act 里面。游戏开始的时候那个转场效果转变也是半透明 QWidget 制作的。

总体上说该游戏界面比较美观，很多细节模仿原版的设计，素材也来自原游戏。

3.3 游戏核心——逻辑与交互

由上述所说，本游戏内采取了独立个体的模式，即每一个对象内可以实现自己逻辑功能（act 函数），包括碰撞判定，切换状态、播放动画、产生新对象加入 scene 等等。zScene 下面包含 每种对象的容器。既然要多态，为什么不给所有对象都弄一个容器，干脆多态到底呢？分开容 器实际上是为了让碰撞判定效率更高，比如说飞行抛射物只用关心僵尸，在进行碰撞判定的时 候只需要扫描 zZombie 的容器 Zombies 即可，像动画之类的就可以完全跳过。但是这种分离 是需要适度的。比如目前飞行抛射物对植物的碰撞判定只有火炬树桩，单独建立一个容器显然是不经济的。

在划分实体的大类时，一个重要的划分依据就是碰撞判定的模式。比如飞行抛射物对僵尸 有碰撞判定（击中）、僵尸对植物有碰撞判定（啃咬）。而动画不参与碰撞判定，如上所说，判定 的功能由对象自身的 act()函数完成，动画类只负责贴图。这样一定程度上做到了动画和逻辑分离的模式。

游戏的主定时器定义在各种 scene 下，每 20ms 进行一次逻辑运算和画面更新。

代码如下：

```
void zLawnScene::onTimer()
{
    this->removeDeath();
    this->act();
    this->SunFront->setText(QString::number(this->sunPoint));
    this->createZombie();
    if (qrand() % 521 < 1)
    {
        zBonus* sun = new zSunFall(this);
        Bonuses.append(sun);
    }
    this->exit->raise();
    this->judge();
}
```

其中有一些细节操作，比如更新阳光槽的显示数值、把 UI 中的 exit 按钮提升到上面来、 随机掉落阳光.....忽略以上细节部分，先关注核心逻辑部分：

removeDeath()：死亡判定函数，负责扫描各个容器，把上一回合 alive 被置为 false 的对象释放并删除，防止内存泄漏。

act()逻辑，负责调用所有对象的 act()执行各自的逻辑。

createZombie()僵尸产生器，负责产生当前回合的僵尸。

judge()失败判定，判断玩家是否失败。

以上就是主定时器的实现，可以看到由于各种复杂的操作都被封装到了各个类自己的 act() 函数中，所以非常简单。这一点体现了封装的思想。

有关交互，Qt 有专门的窗口事件响应器和信号-槽机制对交互做出反应，然后直接把产生的 影响作用于对象，供下一回合的 act()调用来更新状态，相当于异步接受操作。但是由于主定时 器的更新时间只有 20ms，和用户交互起来非常流畅。

而各种僵尸和植物的一些具体操作函数，主要是一些一维碰撞判定，移动位置，切换 QMovie， 发射 zFlyingObject 对象，发射 zAnim 等，属于细节问题，在此不多赘述。

4 系统调试

在游戏的调试过程中，主要发现了以下两个问题：

- 最初我在将背景音乐播放加入了游戏中。但是随之而来诞生了一个问题： 由于背景音乐文件本身是 wav 格式，体积较大，加入 qrc 资源文件后会导致编译时内存不足而 失败。这一点可以通过加载外部文件解决。更大的一个问题是由于本游戏是单线程的，所有模 块运行在一个线程下，而 qt 的音乐播放机制不是很高效，会占用较多的资源，加入背景音乐后 会导致主游戏界面出现卡顿。因此我取消了背景音乐的播放功能。这一点可以考虑采用多线程 来解决，将音乐播放等多媒体操作单独写在主逻辑之外的线程，但由于当时已经属于开发后期， 重新开发多线程的架构成本太高，因此就没有具体实现。由此可见，在开始时的思考和架构的搭建非常重要。

- 内存泄漏的问题。在完成了游戏功能的实现后。我开始了游戏的测试。我在游戏中留了便 于调试的后台——快捷键即可快速产生大量僵尸或者植物等。当我手动产生了大量僵尸和植物 后我打开任务管理器监视内存和 CPU 的使用情况，却发现程序的内存使用率正在以肉眼可见的 美妙 MB 级别的量级攀升，而且没有下降的势头。很明显这里发生了内存泄漏的问题。

考虑到由于僵尸和植物都是一次性产生的，因此导致内存持续上升的应该是各种 zFlyingObject 对象，每发射一枚豌豆都会使用内存，而这些空间没有被正确释放。

而在查看代码后，我发现了问题代码：

```
while(p < FlyingObjects.count())
{
    if(!(FlyingObjects[p]->alive))
    {
        delete(FlyingObjects[p]);
        FlyingObjects.removeAt(p);
    }
    else
    {
        p++;
    }
}
```

一开始我在此处忘记在此处写上 delete 语句，导致被删除的指针指向的内存空间没有被释 放，内存占用不断飙升。

这一个小 bug 很快被解决了，但是我还是不放心，决定查一下程序中的内存泄漏问题， 结果发现我的程序中到处都是内存泄露，只是没有这个明显而已。最后我花了半天的时间通读 代码，修改了七八出没有正确释放内存的错误。 最后我总结了防止内存泄露的两个办法：

- 任何指针在其内存空间被释放之前，都要确保其能被访问到。
  我现在养成了把指针写在对象的属性里的习惯，拒绝在对象某个方法里新定义指针。 因为一旦方法结束了，在方法里定义的指针变量的生存期也就过了，指针变量消失了然而 它所指向的内存空间还在，这段内存空间无法访问又占用着资源，这就是内存泄露。
- 在对象的析构函数中，释放所有对象中出现过的指针的内存空间。
  全部！一个也不能少！这个时候手动写析构函数非常重要！而删除指针的时候涉及到野指针问题，delete 完以后就立刻把指针赋为 nullptr 即可。

按照这两个原则，我对大部分代码都进行了改写。最后的一次测试，我布好阵型，关掉 声音，让游戏自己运行了一晚上。晚上睡觉之前程序占内存 43.3MB，早上起来以后一看还 是 43.3MB。在如此长时间的压力下还能保持稳定，我相信内存泄露问题已经被我消灭了。

以上两个问题只是 debug 中一个有代表性的小环节，还有若干次性能优化、压力 测试，就不再赘述了。通过这样一个较大型项目的编写，我深刻感到写出高效，优美的代码是 一件需要非常高的技巧和反复锻炼的过程。而完成代码后并不是万事大吉，多种方式的测试对于成熟产品的开发具有十分重要的意义。

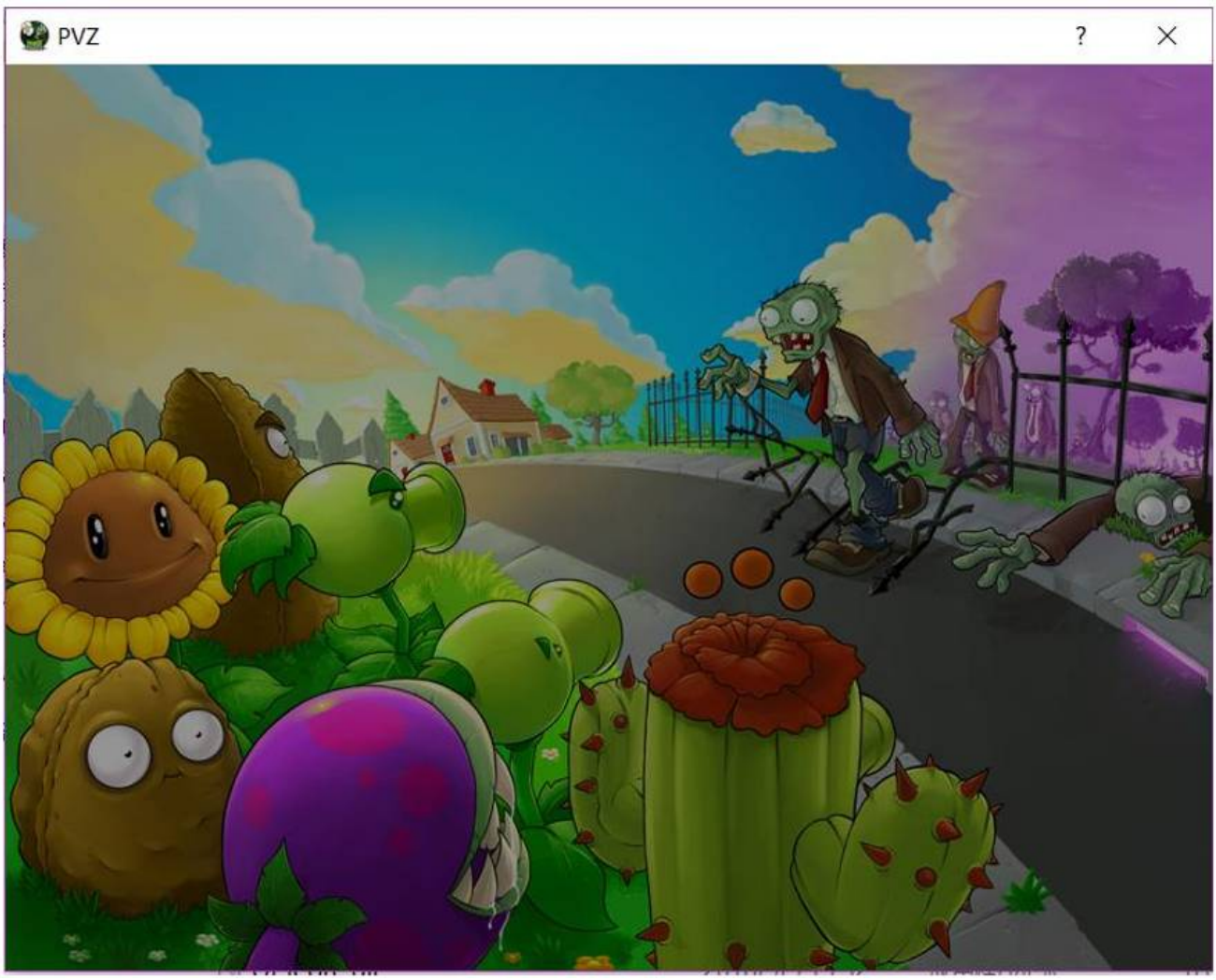
5 测试结果与分析

作为游戏的测试，就是一遍一遍的运行+花式玩耍。这一点我在后期进行了较多的测试， 例如长时间运行，同时产生多个对象等。经过测试和修改，程序已经能够在比较极端的局面 下教稳定运行。以下是运行界面的一些效果：

为了方便测试，我定义了一些快捷键，用于参考：

- 数字键盘 1：产生普通僵尸
- 数字键盘 2：产生旗子僵尸
- 数字键盘 3：产生路障僵尸
- 数字键盘 4：产生铁桶僵尸
- 数字键盘 5：产生铁门僵尸
- 数字键盘 6：产生撑杆僵尸
- 数字键盘 7：产生报纸僵尸
- 数字键盘 8：增加 100 点阳光
- 数字键盘 9：跳过准备阶段，进入僵尸全面进攻状态

加载界面



选择场景界面，有两个场景可供选择，左下角显示从 user.txt 读取的用户名和最好时间。

游戏界面 白天 多种僵尸和植物


PVZ
50
50
100
50
175
200
175
150


PVZ
850
50
100
50
175
200
175
150


PVZ
450
50
100
50
175
200
175
150


PVZ
1000
50
100
50
175
200
175
150
啵!!


PVZ
1000
50
100
50
175
200
175
150

游戏界面 黑夜


PVZ
50
0
50
100
50
175
200
175
150


PVZ
50
0
100
50
175
200
175
150
50


PVZ
200
0
50
100
50
175
200
175
150

僵尸潮测试高负荷下运行

失败界面

高负荷运行时僵尸潮内存和 **CPU** 占用

PVZ (32 位)	7.0%	117.3 MB	0 MB/秒	0 Mbps

6 总结

总体上说这次大作业给我带来了很大的收获和挑战，我在上面投入了非常大的精力，前前后后大约写了五周左右，也对于程序开发设计有了全新的认识。

在选题目时，我一开始的想法就是做游戏，而查询资料后选择了据传闻是“最好的 c++ GUI开发库”Qt。通过一段时间的自学后，我选择了最开始的题目：“小游戏 2048”.这个游戏非常简单，也很容易上手，结果导致我在三天内就把它完成了，整个项目也只有大约 1000 行左右的的代码。处于希望搞点大事情的心态，我选择放弃了这个项目，目标转向了现在这个较复杂的游戏植物大战僵尸。

Qt 库本身就是一个非常精致漂亮的 C++产品，通过学习 Qt 的使用，我对于 C++里面的一些机制例如重载，流等有了更深刻的认识，也学习了全新而威力巨大的信号-槽，事件等机制。而使用 Qt 开发这个游戏的过程中，更是一遍一遍地虐自己的过程。从开始的毫无思路，到四处找代码学习，再到写出来的代码一团糟乱跑不起来，再到能跑起来后各种奇形怪状的 bug，这个过程让我痛并快乐着。当项目成功运行的一刹那，喜悦和激动让我感到极大的幸福和满足感，之前的连续五六个小时的修改，debug,推倒重来带来的疲倦和挫折感一扫而空。我想这就是这样一段开发经历里给我带来的最大精神收获。

而回到知识能力层面，通过这样的一个项目，我的读代码，写代码，debug，测试能力都得到了极大的提升，也提高了我的自己解决问题的能力，比如搜索，和同学讨论，看书等。通过这样一个项目的锻炼，我明白了 C++是一个威力巨大的武器，其中的多样的机制例如重载，继承，派生和多态的用处极大，熟练使用这些机制能够很大程度上影响开发的效率，提升代码的质量。比如再开发完游戏后，我将第一个大作业成绩管理系统也移植到了 qt 平台

上，实现了 GUI 界面，而由于这个项目的开发，我只用了一天的时间就重构了项目。