元气骑士项目

一、项目介绍

使用qt库设计一个2D的地牢冒险游戏。
角色信息包含血条蓝条盾条，盾条在不受伤害一段时间后自动回复，蓝条血条不可回复。
角色攻击包含近战和远程，近战不会消耗蓝条，远程攻击消耗蓝条。
地图信息包含墙和砖块，砖块可被打破。
子弹分为敌方和己方，碰到墙和人就消失。
敌方分为三类，近战类，远程类，boss类，具有一定的ui，血量。

二、项目迭代

1. 第一轮迭代

- 人物移动

  实现键盘控制人物的移动，分别用wsad键对应角色的上下左右移动。

- 显示地图

  地图由若干个方块来构成其边框，人物初始生成在地图的中间，当角色碰到这些方块时会被阻挡不能继续前进。

三、技术难点

该项目要利用Qt库实现，需要先配置各种环境变量。由于刚开始接触MVVM框架与qt库，对于项目框架的具体理解还不深刻，花费了很多时间来熟悉框架。在项目编译的过程中也遇到了许多环境变量有关的问题，包含qt库的引用以及cmake相关的问题。MVVM框架依赖于数据绑定，而Qt的数据模型需要额外的工作来与MVVM的数据绑定机制相集成。熟悉QT库以及Cmake相关的测试就踩了很大的坑。

具体开发问题解决

人物移动命令设计

- 考虑到人物移动是使用WASD键进行移动，因此针对四个按键我们需要使用四个命令设计 `leftmove`，`rightmove`，`upmove`，`downmove`,需要进行四个命令的完善和设计非常的繁琐，加之我们对MVVM框架不大熟悉，测试过程也不大顺利
- 同时考虑移动的实现效果，如果将移动分割成四个方向，那么我们只能实现**“直线移动”**，不能够实现斜向移动（例如同时按下 `W`，`A` 键，控制人物进行向左上方移动）
- 因此我们考虑引入一个方向向量 `direction`,通过QT库中的 `keyboardevent` 将移动改动为向量信息，因此实现了斜向移动以及一个命令的实现

地图信息存储

- 对于地图的渲染方面我们考虑了两个方向
- 第一个方向是整体地图存储图片+分块渲染，具体而言就是将整个地图作为一个图片进行渲染，然后将整个图片分割成多个区域，同时判断人物所在的区域，根据人物在不同的区域渲染不同的部分图片，因此就是“背景固定+人物在区域内移动”
- 第二个方向是以人物为中心渲染地图，人物始终处于地图的最中心，同时渲染的单位从“整张地图”,变成了“单个方块”，因此我们需要存储角色所在的位置以及整体的地图分布（哪一个方块可以行动，哪一个方块是墙），根据人物的坐标进行限制
- 综上考虑我们选择了方案2的渲染方式，这样的渲染可以实现地图信息的动态修改，难点在于“像素单位渲染”和“方块图像渲染”之间的转化，这方面需要进行一定的分析

碰撞效果分析

- 使用上述的地图存储方法，我们就可以对于每一个块的属性进行存储，同时存储了人物的移动信息，判断到墙的时候阻止人物移动
- 由于碰触到了地图的边缘，加上人物的形象本身具有一定的像素大小，因此对于碰撞分析时不能简单的使用像素点进行分析

四、协作情况

- 袁承尧：创建框架，完成common和app层
- 周俊：完成model和viewmodel层
- 郑伟廷：完成view和window层

五、部分效果图

地图和人物的渲染

人物移动的转向示意

人物碰撞到墙体无法移动，且不会出现穿模状态

六、总体心得

本次项目我们使用qt库和MVVM框架来完成，整合Qt和MVVM框架需要大量时间来学习和理解，我们在熟悉框架时遇到了很大的困难，尤其是和qt有关的环境配置和cmake方法，要让程序跑起来就已经花费了大量的时间和精力。同时由于对MVVM框架的理解不深，在开发时走了很多弯路。但一旦熟悉了这两者的结合方式，开发过程会变得更加高效。使用Qt和MVVM框架开发的程序具有很强的可扩展性。Qt提供了丰富的工具和库，而MVVM框架的模块化特性使得新功能的添加和修改变得相对简单。因此我们能够很容易地添加新的功能。

在开发的初期由于对于MVVM框架不大理解，三个队员之间的工作极大的耦合，导致各个成员之间需要相互等待，一个成员的编写工作需要等待另一个成员的编写完代码进行测试，造成了极大的负担。最后通过老师的指导下，问题得到了一定的解决。

七、个人心得

袁承尧：这次实验我负责了common和app层，这部分的代码压力不大，于是，在和组员的商讨下，我开始做jenkins和CMake相关的任务，这部分相当复杂，我们采用Qt库，其中的库文件d3d12在编译的时候出现了找不到的问题，非常的麻烦，而且我们也不知道什么时候他被调用了。和老师交流后，我们采用Qt5进行尝试，重新撰写CMakeLists的过程让我更加了解了这个工具的用处。

周俊：Model负责存储应用程序的数据，ViewModel负责将Model中的数据与View进行绑定，以便在数据变化时更新视图。当数据源发生变化时，Model需要能够通知ViewModel和View，以便及时更新用户界面。ViewModel处理用户界面的命令，例如按键点击等操作，将其转换为对Model的相应操作。因此，这两个模块基本完成了程序的底层逻辑，维护了应用程序的数据模型。

郑伟廷：主要负责view层和window层的相关开发，使用Qt库开发的过程通过键盘和鼠标进行操控，因此需要熟悉Qt库相关的知识以及对于前面过程中model和viewmodel层中的命令完成和属性改变的通知的撰写，使用Qt库使用的渲染中遇到了Qt5和Qt6不兼容以及相关的其他问题，以及对于MVVM框架不了解，通过APP层进行测试的过程也造成了极大的困扰，在开发过程中逐渐理解了MVVM框架，熟悉了对于Qt库的使用，但对于后面的迭代还有很长的路要走。