Underground 游戏设计分报告

姓名：付星赫

学号：3170104650

指导老师：袁昕

1. 分工任务及解决方案

1.1 分工任务

本人负责的分工任务如下：

（1） 持续集成工具 AppVeyor 的配置

（2） 小地图部件功能的实现

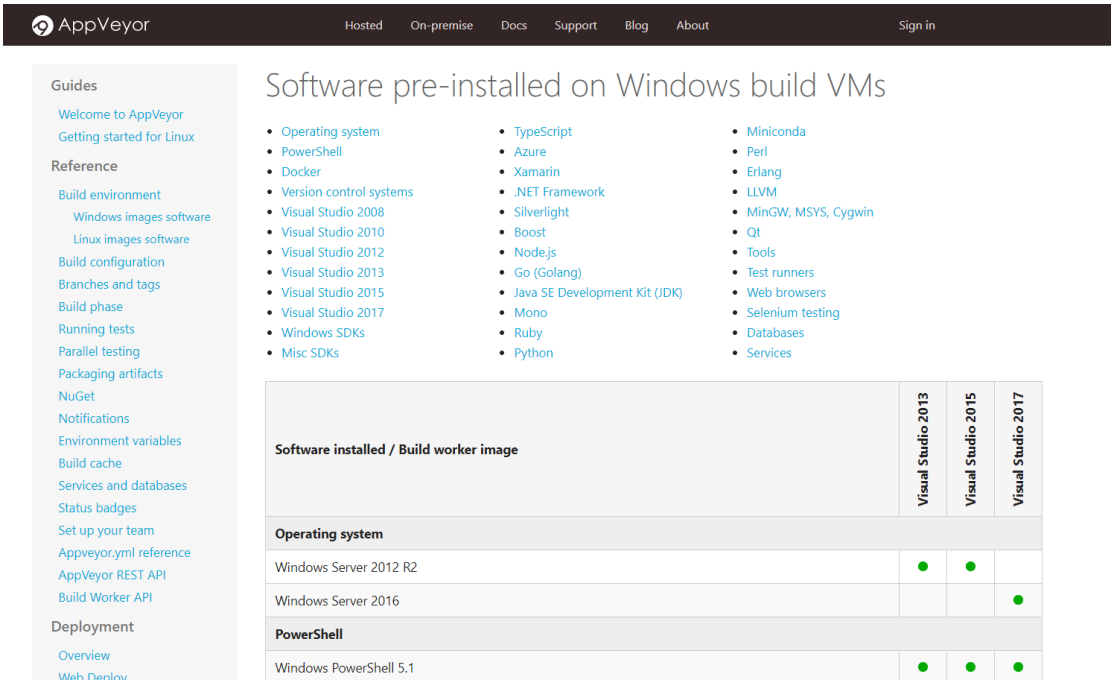
（3） MVVM 框架的设计和应用

1.2 解决方案和设计思路

（1）持续集成工具的配置

为了对项目进行版本控制，方便合作和沟通，我们在 GitHub 上建立了代码仓库。为了更方便地获得新的提交后的 build 结果，确保合并分支请求的正确性，我们还需要一款持续集成工具。考虑到快速便捷的需求，我们选用了一款在 GitHub 上对开源项目免费的托管持续集成工具 AppVeyor。

AppVeyor 的配置相对比较简便，通过查阅官方文档， 可以方便地获取其在不同系统虚拟机上所提供支持的软件版本。我们的项目是在 Windows 平台进行开发，使用 VS2017 和 Qt 5.13.0，AppVeyor 提供对这些软件相应版本的支持（AppVeyor 在近期支持了 Qt 5.13.0）。



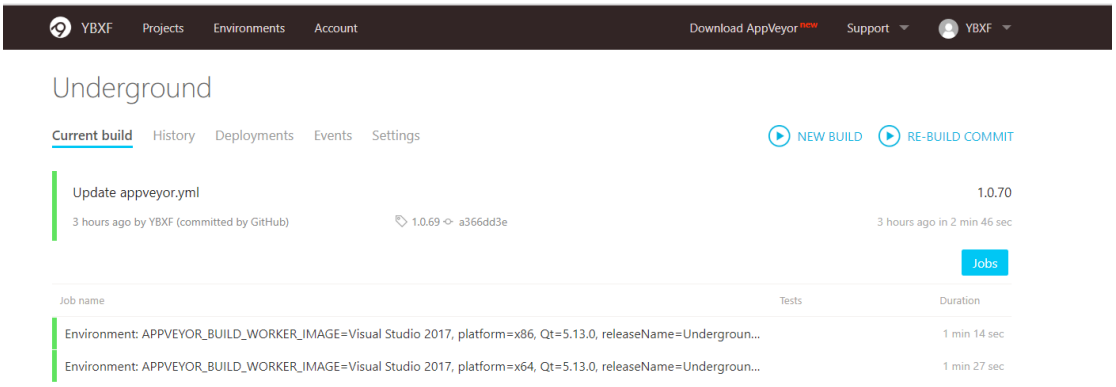
在新建持续化集成项目后，我们既可以在 Setting 中对环境进行配置，也可以在分支中添加.yml 配置文件进行配置，Setting 中也可导出当前配置的.yml 文件，这极大地方便了我

们对.yml 文件的创建和修改。考虑到采用脚本的灵活性，我们在项目中加入.yml 文件实现持续化集成环境的配置。下面给出.yml 文件的细节。

```
version:
1.0.{build}
    image: Visual Studio 2017
    environment:
      matrix:
      - APPVEYOR_BUILD_WORKER_IMAGE: Visual Studio 2017
        platform: x86
        Qt: 5.13.0
        releaseName: Underground_qt512_vs2017_x86
      - APPVEYOR_BUILD_WORKER_IMAGE: Visual Studio 2017
        platform: x64
        Qt: 5.13.0
        releaseName: Underground_qt512_vs2017_x64
    build_script:
    - cmd: >-
        if "%APPVEYOR_BUILD_WORKER_IMAGE%"=="Visual Studio 2017" set
    msvc=msvc2017
        if "%APPVEYOR_BUILD_WORKER_IMAGE%"=="Visual Studio 2017" set
    vs=C:\Program Files (x86)\Microsoft Visual
    Studio\2017\Community\VC\Auxiliary\Build
        if "%platform%"=="x86" set QTDIR=C:\Qt\%qt%\%msvc%
        if "%platform%"=="x64" set QTDIR=C:\Qt\%qt%\%msvc%_64
        set PATH=%PATH%;%QTDIR%\bin;
        if "%platform%"=="x86" set vcvarsall=%vs%\vcvarsall.bat
        if "%platform%"=="x64" set vcvarsall=%vs%\vcvarsall.bat
        if "%platform%"=="x86" call "%vcvarsall%" x86
        if "%platform%"=="x64" call "%vcvarsall%" x64
        qmake C:\projects\underground\maze_mvvm\maze_mvvm.pro
        nmake
    after_build:
    - cmd: >-
        if "%APPVEYOR_REPO_TAG%"=="true" windeployqt
    release\maze_mvvm.exe --qmldir %QTDIR%\qml
    artifacts:
        path: release
        name: $(releaseName)
    deploy:
        provider: GitHub
        auth_token: $(GITHUB_OAUTH_TOKEN)
```

可以看到，在以上文件中我们通过 environment 的 matrix 指定了所使用的软件版本、

平台和项目发行名称，项目将会在 x64 和 x86 两种平台上进行 build。在 build_script 部分，我们使用 cmd 命令实现了对自定义项目的 build，并使用了 qmake 和 nmake 两种工具。经过我们的测试和实际使用，持续集成工具能够正常运行并能打包发行可用的版本，这在很大程度上简化了我们的合并过程。



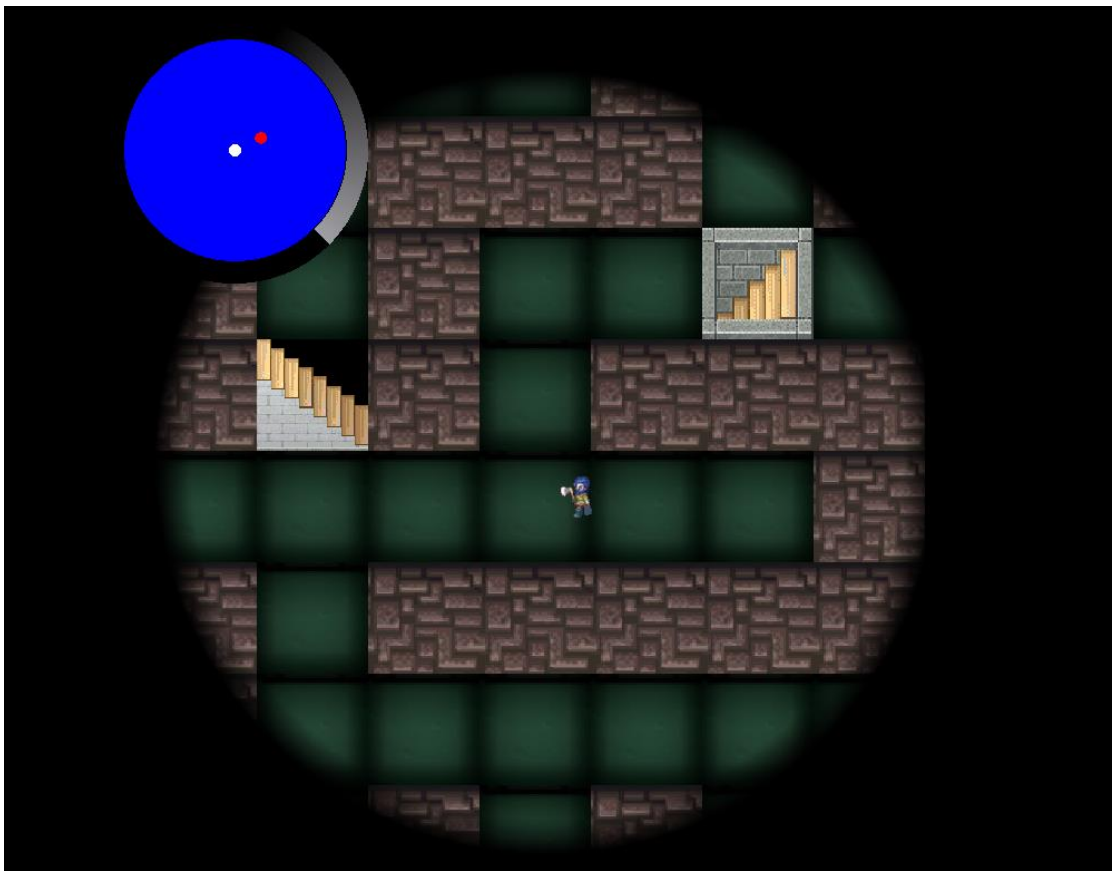
（2）基于迷宫坐标的小地图部件

小地图部件所要实现的效果是以图像的形式显示出终点到当前人物所在位置的相对方位和距离。在 Qt 绘制过程中最重要的是给出两点在迷宫中的坐标显示出以人物为原点的坐标系中终点的位置。关键的绘制函数 paintEvent()如下所示:

```
void smallMap::paintEvent(QPaintEvent *event){
    QPixmap pixmap(size());
    //QPainter painter(&pixmap);
    QPainter painter(this);

    QTransform transform;
    //updatePos();
    painter.setRenderHint(QPainter::SmoothPixmapTransform);
    painter.save();
    painter.setPen(Qt::NoPen);
    painter.setBrush(Qt::ConicalGradientPattern);
    //QRadialGradient radialGradient(10 + 200 , 10 + 200, 150, 150,
110);
    //radialGradient.setColorAt(0, Qt::gray);
    //radialGradient.setColorAt(0.2, Qt::black);
    //radialGradient.setColorAt(0.8, Qt::gray);
    QConicalGradient conicalGradient(150, 150, -45);
    conicalGradient.setColorAt(0, Qt::gray);
    conicalGradient.setColorAt(0.3, Qt::black);
```

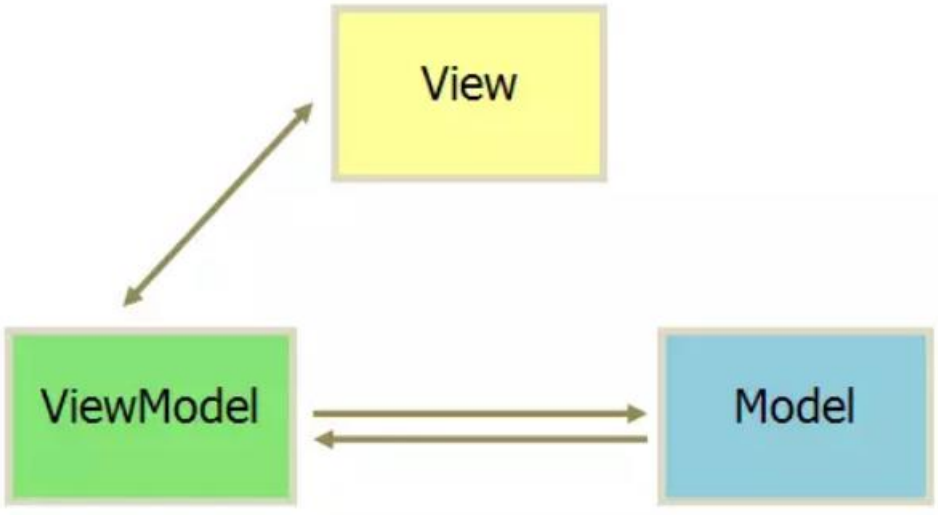
```
    //conicalGradient.setColorAt(1.0, Qt::yellow);
    painter.setBrush(QBrush(conicalGradient));
    painter.drawEllipse(40, 40, 240, 240);
    painter.save();
    painter.setPen(Qt::black);
    painter.setBrush(QBrush(Qt::blue, Qt::SolidPattern));
    painter.drawEllipse(50 + 10, 50 + 10, 200, 200);

    painter.setPen(Qt::white);
    painter.setBrush(Qt::white);
    painter.drawEllipse(155, 155, 10, 10);

    double_t angle, dis;
    int x = (*des_x + 1) * BLOCK_SIZE;
    int y = (*des_y + 1) * BLOCK_SIZE;
    dis = sqrt((double_t)(x - man->get_x()) * (x - man->get_x()) +
(y - man->get_y()) * (y - man->get_y()));
    angle = asin((double_t)(y - man->get_y()) / dis) * 180.0 /
3.1415926;
    if(x - man->get_x() < 0) angle = 180 - angle;
    transform.rotate(-angle);
    painter.setWindow(-160, 160, 320, -320);
    painter.setTransform(transform, false);

    painter.setPen(Qt::red);
    painter.setBrush(Qt::red);
    int cx;
    cx = (dis / 10) < 100 ? dis / 10 : 100;
    painter.drawEllipse(cx - 5, -5, 10, 10);
}
```

计算相对位置时需要注意的是 Qt 中坐标系变换的 rotate()函数是将坐标系顺时针旋转，因此相对位置为了与视觉一致需要将计算出来的方位角关于 x 轴对称反转，在调用 Qt 的 rotate()函数时坐标系旋转角度为-angle。由于小地图大小有限，距离需要根据 10：1 的比例尺进行表示，超出小地图范围的距离长度将会使终点被显示在小地图边缘。在进行绘制之前，我们先调用 setWindow()函数将坐标系转化成以小地图圆心为原点的坐标系，再根据计算出来的角度值对坐标系进行旋转变换，最后用根据比例尺换算得到的距离在变换后的坐标系 x 轴上绘制出终点所在位置即可。效果如下(左上为小地图，白点为人物，红点为终点):



（3）MVVM 框架在迷宫游戏中的应用

为了使迷宫游戏各个模块之间能够解耦，使整个项目能够便于维护和分工合作，我们舍弃了耦合的架构，采用 MVVM 的数据绑定模式来进行开发。为了做到这一点，本人设计了通知、命令、参数的基类，并使用 smart pointer 进行数据绑定。

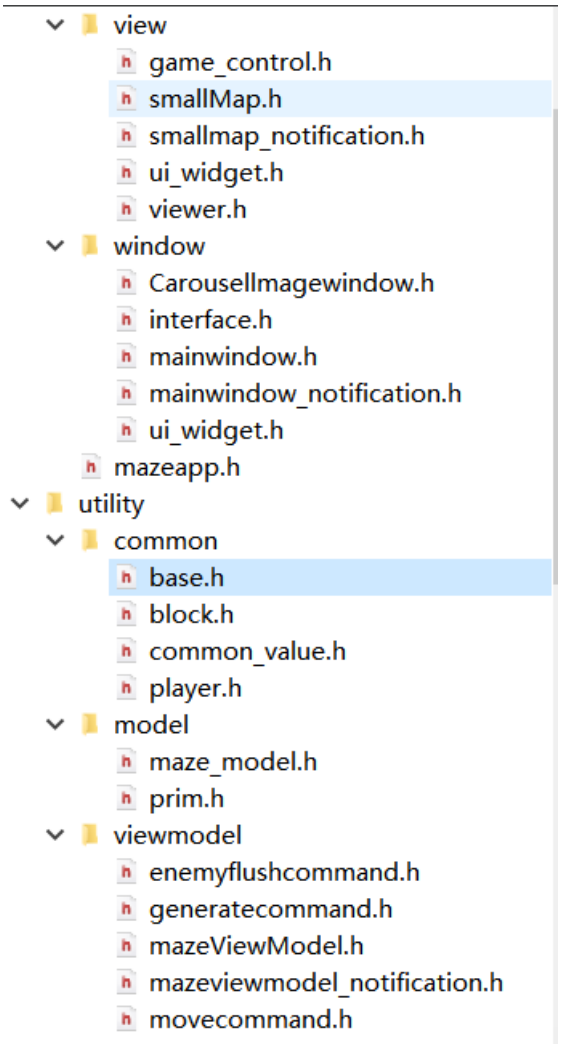


几个基类的定义如下:

```
class Parameters
{
public:
    Parameters(){}
};

class DirectionParameters: public Parameters{
private:
    std::string dir;
public:
    DirectionParameters(std::string Dir):dir(Dir){}
    std::string get_Dir(){
        return dir;
    }
};

class Command
{
protected:
    std::shared_ptr<Parameters> params;
public:
    Command(){}
    virtual ~Command(){}
    void set_parameters(std::shared_ptr<Parameters> parameters){
        params = parameters;
    }
    virtual void exec() = 0;
};

class Notification
{
public:
    Notification(){}
    virtual ~Notification(){}
    virtual void exec() = 0;
};
```

其中 Command 负责从 View 层到 ViewModel 层的关联，通过 Command，View 层向 ViewModel 层传递所需的参数，ViewModel 层根据参数执行子类实现的 exec()函数，直接调用绑定的 Model 层的函数进行数据修改和更新。Model 层每次进行数据更新后，通过 Notification 的 exec()函数对 ViewModel 层绑定的数据进行更新，而 ViewModel 层数据更新后通过 Notification 来进行 View 层的刷新，更新 View 层显示的内容，exec()的具体实现由

View 层和 ViewModel 层的子类负责。



通过文件架构可以看到，ViewModel 层实现了多种 Command 子类，包含敌人刷新、人物移动等。在 Command 子类的具体实现上，我们绑定了相应的的 ViewModel，并调用 ViewModel 中实现的相应 exec 函数，Command 与 ViewModel 相关。View 层只需要调用 smart pointer 指向的 Command 对象的虚函数 exec()并提供相应函数就可以完成对数据的修改，并不需要包含 ViewModel 和 Model 层的头文件，也不需要知道任何相关实现细节。Notification 通知子类则由表层负责实现，但同样解耦，仅用基类的 smart pointer 进行绑定。

```
#include"enemyflushcommand.h"
#include"mazeViewModel.h"

EnemyFlushCommand::EnemyFlushCommand(mazeViewModel
*mvm):maze_viewmodel(mvm){}

void EnemyFlushCommand::exec(){
    maze_viewmodel->exec_enemy_flush_command();
```

```
}

void EnemyFlushCommand::setViewModel(mazeViewModel* VM){
    maze_viewmodel = VM;
}
```

App 层的数据绑定如下所示。View 层所需要的数据是由 smart pointer 指向的通用类型或 Common 层定义类的对象，Command 和 Notification 的绑定也也是由基类的 smart pointer 完成，View 层和 ViewModel，Model 层得以分离。利用 smart pointer 对对象的管理也更加便捷，不需要考虑内存泄漏的问题，降低了开发过程中存在的代码风险。

```
mazeApp::mazeApp(int argc, char* argv[]):QApplication(argc, argv),
    MW(new MainWindow), mazeM(new maze_model), mazeVM(new
mazeViewModel)
{
    mazeVM->bind(mazeM);

mazeM->set_update_display_data_notification(mazeVM->get_update_disp
lay_data_notification());

mazeVM->set_update_view_notification(MW->get_view_notification());

MW->get_viewer()->get_control().set_move_command(mazeVM->get_move_c
ommand());

MW->get_viewer()->get_control().set_generate_command(mazeVM->get_ge
nerate_command());

MW->get_viewer()->get_control().set_mons_flush_command(mazeVM->get_
enemy_flush_command());
    MW->get_viewer()->get_control().bindDesc(mazeVM->get_des_x(),
mazeVM->get_des_y());
    MW->get_viewer()->get_control().bindMaze(mazeVM->get_maze());
    MW->get_viewer()->get_control().bindPlayer(mazeVM->get_man());

MW->get_viewer()->get_control().bindMonster(mazeVM->get_mons());

MW->get_viewer()->get_control().bindSize(mazeVM->get_row_size(),
mazeVM->get_col_size());
    MW->get_smallMap()->bindDesc(mazeVM->get_des_x(),
mazeVM->get_des_y());
    MW->get_smallMap()->bindPlayer(mazeVM->get_man());
}
```

2. 心得体会

(1) 课程中，我对 GitHub 等代码仓库的版本控制有了更深刻的理解，对多人合作开发中的版本迭代和分支合并也思考了很多，同时还掌握了 AppVeyor 等持续化集成工具的使用方法并深刻地感受到了这一系列工具对团队开发过程带来的巨大帮助。

(2) 开发一款图形界面应用离不开图形库，我们小组在开发过程中选择了 Qt。尽管这并不一定是最好用的图形库，但借助 Qt 我们了解了图形界面开发中所需要考虑的种种情形，并把脑海中的设想在电脑上显示了出来，这一过程并不简单但充满乐趣。

(3) MVVM 是贯穿课程的部分，既可以说它是一种技术框架，也可以说它是一种模块化开发中诞生的理念。开始时我们对它的理解不够深刻，但在学习和使用的过程中，它解答了我以前关于协作开发的一些困惑，我逐渐明白通过解耦我们可以专注于个人的工作而无需过多地了解其他人的实现细节，因而在开发中也可不必束手束脚。在个人的开发中，我认为采用解耦的方式也可以让项目变得更加脉络清晰。

3. 改进意见

(1) 关于 c++图形界面的实现虽然有很多工具可以选择，但个人感觉课程中如果能够有更加详细的介绍会帮助同学们快速地理解和掌握这些工具，并能选用合适的工具来进行开发，这可以大大减少同学们在这方面耗费的时间。

(2) 如果能够有事先准备好的参考选题和部分需求可以让整个课程的目标看起来更加明确也更容易上手，课程的成果也会比较容易评判。