高级语言程序设计

实验报告

南开大学 计算机大类
姓名 陈兴浩
学号 2310986
班级 计算机学院 1-3 班
2024 年 5 月 13 日

高级语言程序设计实验报告 目录

高级语言程序设计大作业实验报告

一. 作业题目

Fight For Your Girl---The Combination of Minecraft and PVZ

二. 开发软件

Qt 5.12.2 及其集成开发环境（IDE）Qt Creator 13.0.0（Community）。

三. 课题要求

1) 面向对象。
2) 单元测试。
3) 模型部分。
4) 验证。

四. 项目概述

该项目是一款小游戏项目，游戏背景是勇士的女孩被困在了传送阵的最深处，难以脱身。勇士对女孩真诚的爱/友谊（本项目有两个性别的角色）促使勇士努力救出她。勇士踏上了探险的道路，道路上有众多怪物，能力各不相同。但也有许多机遇和宝藏，获得他们可以提高勇士自身的能力和资源数量。更有一些被困在传送阵的前任冒险者们，为勇士提供建议，交换物资。

这条路注定是一条很困难的路，伴随着动感的音乐，勇士将共计闯过 10 个秘境，穿越 9 个传送阵，辗转来回，打败怪物，提升自己，获得钥匙，打开沿路废弃的古门。更有神秘莫测的战斗判定与结算方式。一旦血量归零，将会有很恐怖

的事情发生!

但鉴于每个勇士的能力各不相同，策略也有高有低，当卡在半路，除了重新开始，项目还提供了 Cheat 玩法，如果你觉得没必要老老实实走完全图（其实是有必要的），随意或有限制地使用 Cheat 玩法，也许可以帮助你更好地救出你的女孩。

这款游戏融合了 Minecraft 的游戏元素及材质包优化元素和 PVZ 的游戏元素，对于喜欢这两款游戏的玩家，享受这款游戏应当是自然的。其他玩家也可以在游戏过程中找到自己的乐趣。路途中情况频发，勇士需要谨慎应对!!!

这个项目使用了多个 cpp 文件，还包含了 qrc、sql 文件，文件之间模块化的编程提高了重用性和可维护性，使用了类的结构，使代码更能体现面向对象编程的特点，运用了 Qt 的信号槽等特性，在不同函数的联系之间给出了符合 Qt 的解决方案。具体的实现将在后文给出。

五. 主要流程

1．文件结构

这个项目包含了以下文件:

（1）**pro 文件**，主要配置了一个 Qt 项目的构建过程。它指定了项目使用的 Qt 模块，定义了目标可执行文件的名称，添加了源文件、头文件和资源文件，并启用了 Qt 废弃警告。这样，该项目就可以正确地编译和构建，并生成一个名为 "Fight" 的可执行文件，其中包含了指定的源代码、头文件和资源。

（2）**头文件**，分别是:

cheat.h：这段代码定义了一个名为 Cheat 的类，它继承自 QObject。该类声明了一系列信号和槽函数，用于处理作弊按钮的点击事件。

database.h：这段代码定义了一个名为 DataBase 的类，用于处理游戏数据的存储和加载。该类包含了一系列公有成员函数，用于连接数据库、保存和加载地图、道具、钥匙、玩家信息以及怪物信息等。

keys.h：这段代码定义了一个名为 Keys 的类，用于管理游戏中的红色、蓝色和黄色钥匙的数量。

mainwindow.h：这段代码定义了一个名为 MainWindow 的类，它是一个继承自 QMainWindow 的窗口类，用于显示游戏的主界面。其中包含各个游戏过程中画面的绘制、窗口标题的创建、槽函数和各类行为变量。

monster.h：这段代码定义了一个名为 Monster 的类，它继承自 Role 类，用于表示游戏中的怪物角色。这个类并没有添加额外的成员变量或函数，只是简单地继承了 Role 类的所有成员和方法。由于没有额外的成员或函数，所以这段代码主要是为了声明 Monster 类，以便在其他地方可以使用这个类。

player.h：

这段代码定义了一个名为 Player 的类，它继承自 Role 类，表示游戏中的玩家角色。Player 类包括用于获取玩家的等级、位置坐标、所在楼层、朝向、性别和需求的函数、设置属性函数、选择角色的函数、升级函数和表示玩家的等级、位置坐标、所在楼层、朝向、角色种类、需求的变量，如等级、位置、朝向等属性。

tools.h：定义了一些书、剑、盾牌等道具的函数和变量。

view.h：这段代码使用了 Q_OBJECT 宏，以支持 Qt 的信号和槽机制。类中声明了一些信号，这些信号用于在特定情况下发出通知，以便其他对象可

以捕获并执行相应的操作。类中声明了一些公共函数，包括用于处理键盘事件的函数，还有一个用于执行动作的函数。同时，提供了一些公共函数用于获取和设置类的状态。以及处理键盘事件的函数。

role.h：这段代码定义了一个名为 Role 的类。该类表示游戏中的角色。类中声明了一些公共函数，包括构造函数 Role、析构函数 ~Role，以及用于获取角色属性的函数。同时，提供了一些用于设置角色属性的函数。私有部分包含了角色的属性。

（3）**源文件**，包括:

cheat.cpp:实现了作弊的一些功能，并发送信号到其他部分，配合作弊功能实现和提示。

database.cpp: 实现了对游戏数据的存取操作，将游戏状态保存到数据库中，并从数据库中加载游戏状态，从而实现了游戏数据的持久化存储。

main.cpp:主函数，创建了 Qt 对象，显示主窗口并播放了背景音乐、连接了数据库，是项目开始执行的部分。

player.cpp:实现了角色的选择和数据的初始化和变化赋值。

mainwindow.cpp:重要部分，实现了各个场景的构建，判定每个场景在何时构建、如何构建、构建什么。与 view.cpp 联系紧密。

view.cpp:重要部分，实现了键盘活动转化为程序活动的过程，确定走过格子应当唤起的函数和走过之后图片的处理，判定键盘活动在何时唤起、如何唤起、唤起什么。与 mainwindow.cpp 联系紧密。

（4）**资源文件**:

主要是一些音乐、数据库和图片，用于场景的构建，有 qrc 文件进

行包含和管理。

2．代码具体分析

1、 各个头文件：

具体功能就如同前文所述，主要是定义了需要用到的类、变量和函数，包含了一些重要的头文件，不再过多赘述。仅说明一个实现细节：

`#ifdef` 和 `#endif` 是 C/C++ 中的预处理器指令，用于条件编译。它们的作用是根据条件选择性地编译代码块。 `#ifdef` 指令用于检查某个标识符是否已经被定义。如果该标识符已经被定义，则执行后面的代码块，否则忽略后面的代码块。`#ifdef` 的语法是 `#ifdef 标识符`。`#endif` 指令用于结束条件编译块。它通常与 `#ifdef`、`#ifndef` 或 `#if` 配对使用，用于标记条件编译的结束。`#endif` 没有参数。这两个指令通常配合使用，形成一个条件编译的代码块。在这个代码块中，根据某个标识符是否被定义，选择性地编译特定的代码。这在处理跨平台代码、调试信息开关等方面非常有用。

2、 main 文件：

```
1. QApplication a(argc, argv);
```

使用了 Qt 库创建了一个应用程序对象 QApplication，并传入命令行参数 argc 和 argv。

```
1.  QMediaPlaylist *playlist = new QMediaPlaylist();
2.      playlist->addMedia(QUrl("qrc:/sound/The music for the game.wav"));
3.      playlist->setPlaybackMode(QMediaPlaylist::Loop);
4.      QMediaPlayer *bgm = new QMediaPlayer();
5.      bgm->setPlaylist(playlist);bgm->play();
```

创建了一个 QMediaPlaylist 对象 playlist，并向其中添加了游戏背景音乐文件的路径，然后设置了循环播放模式。接着创建了一个 QMediaPlayer 对象 bgm，将 playlist 设置为其播放列表，并调用 play() 函数开始播放背景音乐。

```
1.  db.Connect(a.applicationDirPath()  + "/data.sql");
```

连接数据库。applicationDirPath 函数用于匹配不同计算机的前置绝对路径。

之后创建了窗口并显示，循环执行。

接下来，考虑到不同部分的功能关联，我将不会完全按照文件的顺序讲解代码，而是按照代码的内在逻辑来分析代码。

3、 实现过程分析

（1）Mainwindow.cpp

CreateAction 函数：

```
1.  saveGame = new QAction(tr("存档"), this);
```

这个语句创建了一个 QAction 对象，并使用 tr（）函数给出了其显示的名字。

```
1.      connect(cheats, SIGNAL(change()), this, SLOT(slotDrawSc
    ene()));
2.      connect(newGame, SIGNAL(triggered()), this, SLOT(slotNe
    wGame()));
```

这些语句通过 Qt 的信号与槽机制，将一个动作（Action）的触发信号 triggered() 与另一个对象（在这里是 cheats 对象）的特定槽函数 slotCheatMode() 关联起来。其他语句同。这个函数还设置了功能的快捷键，语法比较简单，在此不作叙述。

CreateMenus 函数：

```
1. QMenu *fileMenu = menuBar()->addMenu(tr(" 目录 "));
2. fileMenu->addAction(newGame);
3. fileMenu->addAction(saveGame);
4. fileMenu->addAction(loadGame);
5. fileMenu->addSeparator();    //分割线
6. fileMenu->addAction(exitGame);
```

这些语句就是创建了 Menu 菜单并创建了下属的按钮。Cheat 菜单同此，不再赘述。

之后，创建了程序的界面，并进行了渲染等初始化操作。

```
    connect(view, SIGNAL(change()), this, SLOT(slotDrawScene()));
    connect(view, SIGNAL(fight(int)), this, SLOT(slotFight(int)));
    connect(view, SIGNAL(events(QString)), this, SLOT(slotEvent(QString
)));
    connect(view, SIGNAL(move(int, int)), this, SLOT(slotMovePlayerItem
(int, int));
```

前文提到过，mainwindow 文件和 view 文件是紧密相连的，原因就是键盘的操作活动和图形界面的更新活动是密不可分的，所以这些语句就将 mainwindow 和 view 的文件紧密联系到一起。

接下来分析各个槽函数：

重要槽函数：slotEvent

```
1. if (str == "new") slotNewGame();
2. else if (str == "boy") player.ChooseKind(1);
3. else if (str == "girl") player.ChooseKind(0);
4. else if (str == "save") slotSaveGame();
5. else if (str == "load") slotLoadGame();
6. else if (str == "book") slotBook();
7. else if (str == "shop") slotShop();
8. else if (str == "quit") close();
```

```
9. else if (str == "Ideas") slotIdeas();
10.
11.else {
12.    Clear();
13.    AddPictureItem(-102, 0, "background");
14.    if (str == "ending")
    // 结局的确定
15.        str += QString::number(player.GetSex(), kDecimal);

16.    AddPictureItem(-102, 0, str);
17.}
```

这段代码是一个逻辑分支，根据传入的字符串 str 的不同值执行不同的操作。具体来说：
如果 str 的值是 "new"，则调用 slotNewGame() 函数，开始一个新游戏。
如果 str 的值是 "boy"，则调用 player.ChooseKind(1) 函数，选择男性角色。
如果 str 的值是 "girl"，则调用 player.ChooseKind(0) 函数，选择女性角色。
如果 str 的值是 "save"，则调用 slotSaveGame() 函数，保存游戏。
如果 str 的值是 "load"，则调用 slotLoadGame() 函数，加载游戏。
如果 str 的值是 "book"，则调用 slotBook() 函数，显示怪物图鉴。
如果 str 的值是 "shop"，则调用 slotShop() 函数，进入商店。
如果 str 的值是 "quit"，则关闭当前窗口。
如果 str 的值是 "Ideas"，则调用 slotIdeas() 函数，显示游戏提示。
最后，如果 str 的值不是上述任何一个情况，则执行 else 分支。这个分支通常用于处理一些特殊情况，可能会清除当前的游戏界面，然后根据 str 的值加载不同的图片或展示不同的内容。例如，如果 str 是 "ending"，可能会根据玩家选择的性别加载不同的结局图片。
之所以说这个槽函数重要，是因为它还指导了其他一系列槽函数的执行时机，同时与 view 文件联系，接受 view 文件传来的 status 信号。
Slot 函数所构建的逻辑基本简单相同，不过多分析，下面分析一些复杂重要的成员函数。

```
1. if (str == "new") slotNewGame();
2. else if (str == "boy") player.ChooseKind(1);
3. else if (str == "girl") player.ChooseKind(0);
4. else if (str == "save") slotSaveGame();
5. else if (str == "load") slotLoadGame();
6. else if (str == "book") slotBook();
7. else if (str == "shop") slotShop();
8. else if (str == "quit") close();
9. else if (str == "Ideas") slotIdeas();
10.
11.else {
12.    Clear();
13.    AddPictureItem(-102, 0, "background");
14.    if (str == "ending")
15.        str += QString::number(player.GetSex(), kDecimal);
```

```
    AddPictureItem(-102, 0, str);
}
```

这段代码用于清空场景中的所有图形项。
具体来说，这个函数执行了以下操作：
首先，通过 scene->items() 获取场景中的所有图形项，并将它们存储在一个 QList 中。
然后，进入一个 while 循环，只要 QList 不为空，就执行以下操作：
从 QList 中取出第一个图形项（使用 listItem.at(0) 获取第一个元素）。
使用 scene->removeItem() 将该图形项从场景中移除。
使用 listItem.removeAt(0) 将该图形项从 QList 中移除，以确保不断迭代并移除列表中的第一个元素。
当 QList 中所有的图形项都被移除后，while 循环结束，函数执行完成，场景中的所有图形项都被成功清空。这段代码的目的是确保在需要清空场景时，能够彻底清除所有的图形项，以便重新绘制新的图形内容或执行其他操作。

```
void MainWindow::slotSaveGame() {            // 保存数据
    db.SaveMap(1);
    db.SaveTools(1);
    db.SaveKeys(1);
    db.SavePlayer();
}

void MainWindow::slotLoadGame() {            // 加载数据
    db.LoadMap(1);
    db.LoadTools(1);
    db.LoadKeys(1);
    db.LoadPlayer();
    db.LoadMonsters();
    view->SetStatus("main");
    slotDrawScene();
}
```

slotSaveGame() 函数：
调用了数据库对象 db 的 SaveMap()、SaveTools()、SaveKeys() 和 SavePlayer() 成员函数，分别保存地图数据、工具数据、钥匙数据和玩家数据到数据库中。
slotLoadGame() 函数：
调用了数据库对象 db 的 LoadMap()、LoadTools()、LoadKeys()、LoadPlayer() 和 LoadMonsters() 成员函数，分别从数据库中加载地图数据、工具数据、钥匙数据、玩家数据和怪物数据。
然后，通过 view->SetStatus("main") 将视图的状态设置为 "main"，这可能是指主游戏界面。
最后，调用了 slotDrawScene() 函数，重新绘制游戏场景，以确保加载的数据能够正确显示在界面上。
这两个函数一起实现了游戏的存档和加载功能，允许玩家在游戏中保存当前进度，并在以后重新加载游戏时恢复到之前保存的状态。
其他重要的函数：

往界面上加入元素的函数：

```
void MainWindow::AddTextItem(int x, int y, QString str, int size, QColor color) {
    QFont font("Times", size);
    QGraphicsTextItem *item = new QGraphicsTextItem(str);
    item->setFont(font);
    item->setDefaultTextColor(color);
    scene->addItem(item);
    item->setPos(x-kOffsetX,y-kOffsetY);
}
```

以加入文字的函数为例：

1、根据传入的参数，创建了一个 QFont 对象 font，用于设置文本的字体和大小。

2、创建了一个 QGraphicsTextItem 对象 item，并将要显示的文本内容 str 传入构造函数中，以创建一个文本项。

3、使用 setFont() 函数将 font 设置为文本项的字体。

4、使用 setDefaultTextColor() 函数将指定的颜色 color 设置为文本项的默认文本颜色。

5、将文本项添加到场景中，使用 scene->addItem(item) 实现。

6、使用 setPos() 函数设置文本项在场景中的位置，位置由传入的参数 x 和 y 确定，同时通过 kOffsetX 和 kOffsetY 进行微调。

这样，调用这个函数就可以在游戏场景中添加指定位置、指定大小、指定颜色的文本元素，用于显示游戏中的文字信息，比如玩家状态、提示信息等。

SlotFight 函数：

其中使用循环模拟了每轮的伤害，具体代码如下：

```
random = qrand() % 99 + 1;
        damage = qMax(player.GetAttack() - monsters[num].GetDefend(), 1);
        damage += qFloor(damage*0.5*(random >= player.GetCrit()));
        monsterHp -= damage * (random > monsters[num].GetMiss());
        if (random <= monsters[num].GetMiss())
            playerMiss++;
        if (random >= player.GetCrit())
            monsterCrit++;
        if (monsterHp <= 0) {
            monsterHp = 0;
            break;
        }
        random = qrand() % 99 + 1;
        damage = qMax(monsters[num].GetAttack() - player.GetDefend(), 1);
     damage += qFloor(damage*0.5*(random >= monsters[num].GetCrit()));
```

```
15.         playerHp -= mode * damage * (random > player.GetMis
    s());
16.         if (random <= player.GetMiss())
17.             monsterMiss++;
18.         if (random >= monsters[num].GetCrit())
19.             playerCrit++;
20.         if (playerHp <= 0) {
21.             playerHp = 0;
22.             break;
23.         }
24.         time++;
25.     }
```

random = qrand() % 99 + 1;: 生成一个 1 到 100 之间的随机数, 存储在变量 random 中。
damage = qMax(player.GetAttack() - monsters[num].GetDefend(), 1);: 计算玩家对怪物造成的伤害，至少为 1。这里使用了 qMax() 函数来确保伤害值不会小于 1。
damage += qFloor(damage*0.5*(random >= player.GetCrit()));: 检查是否暴击。如果 random 大于等于玩家的暴击率，伤害增加 50%。这里使用了 qFloor() 函数将计算结果向下取整。
monsterHp -= damage * (random > monsters[num].GetMiss());: 计算怪物受到的伤害。如果 random 大于怪物的闪避率，怪物受到伤害，否则不受伤害。
if (random <= monsters[num].GetMiss()): 如果 random 小于等于怪物的闪避率，表示怪物闪避了玩家的攻击。
if (random >= player.GetCrit()): 如果 random 大于等于玩家的暴击率，表示玩家发动了暴击。
if (monsterHp <= 0): 如果怪物的血量小于等于 0，结束循环，战斗结束。
然后，进行怪物攻击玩家的过程，逻辑与上述过程类似。
time++: 每进行一轮战斗，时间加 1。
通过这一过程，做到了模拟怪物和勇士相互攻击的相对实际和公平的过程。
该模块的重要函数讲到这里，接下来我们来关注 view 文件与 mainwindow 文件的紧密联系!
（2）view 文件

```
1.  void View::keyPressEvent(QKeyEvent * event) {          // 选
    择目前键盘活动状态
2.      if (status == "main") {
3.          keyMain(event);
4.          action();
5.      } else if (status == "tip" || status == "book" || statu
    s == "help") {
6.          SetStatus("main");
7.          emit change();
8.      } else if (status == "ending" || status == "fail")
9.          emit quit();
10.     else if (status == "fight")
```

```
        keyFight();
    else if (status == "shop")
        keyShop(event);
    else if (status == "welcome")
        keyWelcome(event);
    else if (status == "select")
        keySelect(event);
    else if(status == "Ideas")
        keyIdea(event);
}
```

同样的，这个文件也具有一个至关重要的函数，我们来讲讲它为什么重要，需要注意的是，这个文件与上一个所讲述的文件具有密不可分的联系。

（1） 本代码运用传入的 status 字符串判断接下来将运用哪种键盘活动判定方式。

```
view->GetStatus() != "main"
```

（2） mainwindow 中可以通过这个语句获取 status 状态

（3） 与其他 key 状态函数紧密相连。

至于键位的实现则大同小异，下面选取一个来具体讲解。

```
void View::keyMain(QKeyEvent * event) {
    switch (event->key()) {
        case Qt::Key_Up:
            if (access(player.GetPosx(), player.GetPosy()-1
))
                emit move(0, -1);
            player.SetToward(1);
            break;
        case Qt::Key_Down:
            if (access(player.GetPosx(), player.GetPosy()+1
))
                emit move(0, +1);
            player.SetToward(3);
            break;
        case Qt::Key_Left:
            if (access(player.GetPosx()-1, player.GetPosy()
))
                emit move(-1, 0);
            player.SetToward(2);
            break;
        case Qt::Key_Right:
            if (access(player.GetPosx()+1, player.GetPosy()
))
                emit move(+1, 0);
            player.SetToward(4);
            break;
        case Qt::Key_B:
```

```
            if (tools.GetBook()) {
                SetStatus("book");
                emit events("book");
            }
            break;
        case Qt::Key_S:
            emit events("save");
            break;
        case Qt::Key_L:
            emit events("load");
            break;
        case Qt::Key_Q:
            emit events("quit");
            break;
        case Qt::Key_H:
            SetStatus("help");
            emit events("help");
            break;
        case Qt::Key_4:
            SetStatus("Ideas");
            emit events("Ideas");
            break;
        default:
            break;
    }
```

（1） emit events 函数将信号传递到 mainwindow 文件，配合场景的变化与更改。这些信号的接收者会连接到相应的槽函数，以便在接收到信号时执行特定的操作。例如，当 "B" 键被按下时，发出 events("book") 信号，接收者可以连接到这个信号，执行打开书籍界面的操作。当使用 Qt 中的信号与槽机制时，通过 emit 关键字可以发出信号，以通知连接到该信号的槽函数执行相应的操作。

（2） 根据按下的按钮不同，用 setStatus 函数重置当前键盘状况关键词，用于传递。接下来介绍定义的一个公用三维数组 map[x][y][z]，x，y 代指坐标，z 代指当前的层数（这是因为这个游戏有十层，为了避免用 10 个数组存储，故有此策略），这个位置对应的值是这个地图的特性，即本项目用这个特征值代表这个格子上对应的情况，是怪物、资源还是空地？由这个决定。而初始的值存储在文件 data.sql 中对应不同数值，程序的判断由下述函数执行。

```
void View::action() {
    switch (next_step) {
        case 3:
            keys.SetYellow(qMax(keys.GetYellow()-1, 0));
            break;
        case 4:
            keys.SetBlue(qMax(keys.GetBlue()-1, 0));
            break;
```

```
9.          case 5:
10.             keys.SetRed(qMax(keys.GetRed()-1, 0));
11.             break;
12.         case 6:
13.             keys.SetYellow(keys.GetYellow()+1);
14.             break;
15.         case 7:
16.             keys.SetBlue(keys.GetBlue()+1);
17.             break;
18.         case 8:
19.             keys.SetRed(keys.GetRed()+1);
20.             break;
21.         case 10:
22.             SetStatus("tip");
23.             emit events("tip1");
24.             break;
25.         case 11:
26.         case 12:
27.         case 13:
28.         case 14:
29.         case 15:
30.         case 16:
31.         case 17:
32.         case 18:
33.         case 19:
34.         case 20:
35.         case 21:
36.         case 22:
37.         case 23:
38.             SetStatus("fight");
39.             emit fight(next_step);
40.             break;
41.         case 24:
42.
43.             player.SetHp(player.GetHp()/3);
44.             break;
45.         case 25:
46.
47.             player.SetDefend(player.GetDefend()/3);
48.             break;
49.         case 26:
50.
51.             player.SetHarm(player.GetAttack()/3);
52.             break;
```

```
53.         case 27:
54.
55.             SetStatus("tip");
56.             emit events("tip2");
57.             break;
58.         case 28:
59.             player.SetCoin(player.GetMoney()+15);
60.             break;
61.         case 29:
62.             player.SetHp(player.GetHp()+18);
63.             break;
64.         case 30:
65.             player.SetHp(player.GetHp()+30);
66.             break;
67.         case 31:
68.             tools.SetSword(1);
69.             player.SetHarm(player.GetAttack()+10);
70.             break;
71.         case 32:
72.             player.SetHarm(player.GetAttack()+3);
73.             break;
74.         case 33:
75.             tools.SetShield(1);
76.             player.SetDefend(player.GetDefend()+8);
77.             break;
78.         case 34:
79.             player.SetDefend(player.GetDefend()+2);
80.             break;
81.         case 35:
82.             tools.SetBook(1);
83.             break;
84.         case 51:
85.
86.             SetStatus("shop");
87.             emit events("shop");
88.             break;
89.         case 80:
90.
91.             player.SetPart(player.GetFloor()+1);
92.             break;
93.         case 90:
94.             player.SetPart(player.GetFloor()-1);
95.             break;
96.         case 99:
```

```
97.             SetStatus("ending");
98.             emit events("ending");
99.             break;
100.        }
101.        if (status == "main")
102.            emit change();
103.    }
```

通过这样的判断，执行不同的函数。其中 11-22 没有任何内容的原因是，这些格子上同 23 一样，都是怪物，而 23 调用的函数传递的 next_step 值可以明确具体的怪物类型，故不填写任何内容，一直执行到 23，完全可以胜任，避免了代码重复和冗长

```
int View::access(int x, int y) {
    int tmp = map[x][y][player.GetFloor()];
    next_step = tmp;
    if (tmp == 0 || (tmp > 5 && tmp < 10) || (tmp > 10 && tmp < 27) ||
(tmp > 27 && tmp < 35) || tmp == 59 ||
        (tmp == 3 && keys.GetYellow() > 0) || (tmp == 4 && keys.GetBlue
() > 0) || (tmp == 5 && keys.GetRed() > 0)) {
        map[x][y][player.GetFloor()] = 0;
        return 1;
    } else
        return 0;
}
```

这个代码的逻辑不复杂，就是判断是否可以直接通过，如果不行，则更新之后通过，如果不可以，传递 0 信号，拒绝通过。

如果在 main 状态下，调用了这个连接函数：

connect(view, SIGNAL(change()), this, SLOT(slotDrawScene()));

即进行 slotDrawScene 的执行，构建新的秘境的画面状态，意思是通过了传送门，即进入了新的秘境。

（3） 其他文件里的重要函数或简单概述

下面是 cheat.cpp

```
1. void Cheat::slotCheatShield() {
2.     tools.SetShield(1 - tools.GetShield());
3.     if (tools.GetShield())
4.         player.SetDefend(player.GetDefend() + 8);
5.     else
6.         player.SetDefend(player.GetDefend() - 8);
7.     emit change();
8. }
```

这段函数实现了按钮按下后，数值的改变和画面的重绘，具体的内容上面都已经叙述，不再多说。其余函数大同小异，也不再赘述。

但关于绘制的过程，有一点可以一提：

```
AddTextItem(-50, 5, QString::number(player.GetFloor()+1, kDecimal), 8,
QColor(qrand()%256,qrand()%256,qrand()%256));
AddTextItem(-65, 100, "Lv "+QString::number(player.GetLevel() +1, kDeci
mal), 10, QColor(qrand()%256,qrand()%256,qrand()%256));
```

使用 qrand 函数，每次绘制时颜色都会有更改。

player.cpp

```
1. void Player::ChooseKind(int kind) {
2.     if (kind == 1) {
3.         SetHp(150);
4.         SetHarm(14);
5.         SetMiss(8);
6.         SetCrit(93);
7.     } else {
8.         SetHp(250);
9.         SetHarm(13);
10.        SetMiss(18);
11.        SetCrit(99);
12.    }
13.    SetDefend(10);
14.    SetCoin(10);
15.    SetExp(0);
16.    this->level = 0;
17.    this->posx = 6;
18.    this->posy = 12;
19.    this->floor = 0;
20.    this->toward = 3;
21.    this->kind = kind;
22.    this->need = 5;
23.}//初始数值
24.
25.void Player::LevelUp() {
26.    level += 1;
27.    SetHp(GetHp() + 10);
28.    SetHarm(GetAttack() + 1);
29.    SetDefend(GetDefend() + 1);
30.    need = qFloor(need*1.5) + 1;
31.}//升级后的函数
```

就是简单的逻辑判断和赋值，不再多说。

database.cpp

有关地图的显示：

```
1. QString DataBase::layer(int num) {
2.     QString str = " ";
3.     for (int i = 0; i < 14; i++)
4.         for (int j = 0; j < 14; j++) {
```

```
            if (map[j][i][num] < 10)
                str += "0";
            str += QString::number(map[j][i][num], kDecimal);
            str += " ";
        }
    return str;
}
```

这段代码用于生成并返回一个字符串表示的地图层数据。
函数逻辑如下：
初始化一个空字符串 str，用于存储地图层数据。
使用两个嵌套的循环遍历地图的每个位置。
对于每个位置，检查地图上的值是否小于 10，如果是，则在字符串中添加一个额外的 "0"，以确保每个数值都是两位数。
将当前地图位置的数值转换为字符串，并添加到 str 中。
在每个数值后面添加一个空格，以便在字符串中对地图数据进行分隔。
返回包含地图层数据的字符串 str。
这个函数的作用是将地图的二维数组表示转换为一个字符串，其中每个数值表示一个地图元素，通过添加额外的 "0" 和空格，确保了地图数据的格式化显示。

数据库的连接：

```
void DataBase::Connect(QString dbpath) {
    db.setDatabaseName(dbpath);//连接到数据库
    if (!db.open()) {
        QMessageBox msgBox;
        msgBox.setIcon(QMessageBox::Critical);
        msgBox.setText(db.lastError().text());//获取并返回错误信息
        msgBox.setStandardButtons(QMessageBox::Ok);
        msgBox.exec();//如果创建不成功，提示用户
    }

    QFile file(":/database/data.sql");          // 打开数据文件
    if(!file.exists()){
        qDebug() << "Database init file not exist, expected: " << file.fileName();
    }

    QSqlQuery query(db);//创建一个对象

    if (file.open(QIODevice::ReadOnly | QIODevice::Text))//文件只读打开
    {
```

```
        QStringList scriptQueries = QTextStream(&file).readAll().split(';');
        foreach (QString queryTxt, scriptQueries) {
            if (queryTxt.trimmed().isEmpty()) {
                continue;
            }
            if (!query.exec(queryTxt))
            {
                qFatal("One of the query failed to execute. Error detail: %s", query.lastError().text().toStdString().c_str());
            }
            query.finish();
        }
    }
    file.close();
    qDebug( "Database connected." );
}
```

这段代码用于连接到数据库并执行数据库初始化操作。
具体逻辑如下：
设置数据库的名称为传入的 dbpath，这个路径指向了数据库文件。
尝试打开数据库，如果打开失败，则创建一个严重错误的消息框，显示数据库打开失败的错误信息，并提示用户。
检查数据库初始化文件是否存在，如果不存在，则输出一条调试信息。
创建一个 QSqlQuery 对象，用于执行数据库查询。
打开数据文件并以只读文本模式打开，然后读取其中的所有内容，并按照分号 ; 将其分割成一个字符串列表，每个元素都是一个 SQL 查询语句。
遍历分割后的字符串列表，对每个查询语句执行查询操作。
如果执行查询失败，则以严重错误的方式终止程序，并输出错误详情。
完成所有查询操作后，关闭数据文件，并输出一条消息表示数据库已连接。
这段代码的作用是连接到数据库，并执行数据库初始化操作，将数据库文件中的 SQL 查询语句逐个执行，从而初始化数据库。

除此之外 主体函数基本都是：存档、读取存档
存档函数如下（选取一个为例）：

```
void DataBase::SaveMap(int num) {          // 保存 下同
    QSqlQuery query(db);
    QString str = "UPDATE map SET layer0=?, layer1=?, layer2=?, layer3=?, layer4=?, layer5=?, layer6=?, layer7=?, layer8=?, layer9=? WHERE id=" + QString::number(num+1, kDecimal);
    query.prepare(str);
    query.bindValue(0, layer(0));
    query.bindValue(1, layer(1));
```

```
    query.bindValue(2, layer(2));
    query.bindValue(3, layer(3));
    query.bindValue(4, layer(4));
    query.bindValue(5, layer(5));
    query.bindValue(6, layer(6));
    query.bindValue(7, layer(7));
    query.bindValue(8, layer(8));
    query.bindValue(9, layer(9));
    query.exec();
    query.clear();
}
```

创建了一个对象用于查询，之后使用 SQL 语句 prepare 留作后续程序使用，而 bindvalue 则使用这一个地址在不同的层级存储当前的数据，做到数据保存的目的。最终数据更新以语句 query.exec();执行，这是因为这个句子通过 prepare 的地址更新地图数据。UPDATE 语句是 SQL 语句。

读取存档：

```
void DataBase::LoadMap(int num) {
    QSqlQuery query(db);
    QString str = "SELECT layer0, layer1, layer2, layer3, layer4, layer5, layer6, layer7, layer8, layer9 FROM map WHERE id = " + QString::number(num+1, kDecimal);
    query.exec(str);//查询
    query.next();
    for (int i = 0; i < kMaxFloor; i++) {
        QString floors = query.value(i).toString();
        floors = floors.simplified();
        int tmp = 0;
        while (floors.length() > 3) {
            map[tmp%kMapLen][tmp/kMapLen][i] = floors.left(2).toInt();
            floors.remove(0,3);
            tmp++;
        }
        map[kMapLen-1][kMapLen-1][i] = floors.toInt();
    }
    query.clear();//释放
}
```

逻辑同上，就是通过 SQL 对象查询第二条记录（第一条记录是全新地图的模板），然后按层数遍历到最终的所有地图，通过对字符串的处理，进行地图的数据构建，最后释放数据。（这就是为什么要把一位数的特征值加 0 变成两位数并格式化显示）

（4）qrc 内含有数据库的具体代码、图片和音乐资源供程序调用和显示。

六. 单元测试

1. 数据库连接和数据读取/保存功能

测试目的： 验证数据库连接功能是否正常，以及数据读取/保存功能的准确性和稳定性。

测试方法： 执行数据库连接操作，读取/保存预先准备好的测试数据，并验证结果是否符合预期。

测试步骤：

使用正确的数据库路径执行数据库连接操作。
试玩一轮游戏，玩到一半执行存档功能，退出游戏。
重新进入游戏，执行数据读取操作，加载预先存储的地图数据。
执行数据保存操作，保存当前地图数据。
验证保存后的地图数据与原始数据一致。

测试结果：

数据库连接成功，并出现连接成功提示（qDebug）。
地图数据成功加载，地图显示正常，与预期一致。
地图数据成功保存，保存后的地图数据与原始数据一致，无异常情况发生。

测试结论：数据库连接和数据读取/保存功能均通过测试，符合预期要求，无异常情况发生。

2. 界面交互功能测试

测试目的： 验证界面交互功能是否正常，包括按钮点击、键盘输入等用户操作。

测试方法： 执行界面上的各种交互操作，包括点击按钮、输入特定按键等，验证界面的响应是否符合预期。

测试步骤：

点击“新的游戏”按钮，验证是否触发了新游戏的开始。
点击“存档”按钮，验证是否成功保存当前游戏状态。
点击“重载游戏存档”按钮，验证是否成功加载之前保存的游戏状态。
点击“退出游戏”按钮，验证是否成功退出游戏。
模拟键盘输入，按下方向键，验证角色是否能够移动。
模拟键盘输入，按下特定键，验证是否触发了对应的功能，如打开书籍、打开商店等。

测试结果：

点击各个按钮时，界面的响应与预期一致，功能正常触发。

键盘输入时，角色能够根据按键移动，操作流畅。

特定按键触发的功能正常执行，如打开书籍、打开商店等。

测试结论： 界面交互功能测试通过，各个按钮和键盘输入功能正常，符合预期要求，用户操作流畅，无异常情况发生。

3. 作弊功能测试

测试目的： 验证游戏中的作弊功能是否正常，包括增加属性、获取特定道具等。

测试方法： 执行作弊功能的操作，验证游戏中的角色属性、道具数量等是否按预期增加。

测试步骤：

触发作弊功能，例如血量+100。
触发作弊功能，例如攻击+10。
触发作弊功能，例如防御+10。
触发作弊功能，例如金币+50。
触发作弊功能，例如等级+1。
触发作弊功能，例如获取黄色钥匙+1。
触发作弊功能，例如获取蓝色钥匙+1。
触发作弊功能，例如获取红色钥匙+1。
触发作弊功能，例如打开怪物图鉴。
触发作弊功能，例如获得剑道具。
触发作弊功能，例如获得盾牌道具。
触发作弊功能，例如开启无敌模式。

测试结果：

角色血量成功增加 100 点。
角色攻击力成功增加 10 点。
角色防御力成功增加 10 点。
角色金币数量成功增加 50 个。
角色等级成功提升 1 级。
角色获得了 1 个黄色钥匙。
角色获得了 1 个蓝色钥匙。
角色获得了 1 个红色钥匙。
成功打开了怪物图鉴。
角色获得了剑道具。
角色获得了盾牌道具。
成功开启了无敌模式。

测试结论：游戏中的作弊功能测试通过，各个作弊功能能够正常触发，并且游戏中的角色属性、道具数量等按预期增加，符合预期要求。

4. 文本、图片增添功能测试

测试目的： 验证游戏中的文本和图片功能是否正常，包括添加文本元素和图片元素到场景中。

测试方法： 执行添加文本和图片元素的操作，验证文本和图片是否正确显示在游戏场景中。

测试步骤：

添加文本元素到场景中，检查文本是否正确显示。
添加图片元素到场景中，检查图片是否正确显示。

测试结果：

文本元素成功添加到场景中，并且文本内容正确显示。
图片元素成功添加到场景中，并且图片正确显示。

测试结论： 游戏中的文本和图片功能测试通过，能够正常添加文本和图片元素到场景中，并且显示正确，符合预期要求。

5. 角色状态更新单元测试

测试目的：确定角色能够在特定操作下做出相应的状态更新，包括朝向、血量、钥匙数量等各大特性。

测试方法：执行角色的各个方向移动，与怪物战斗，比较战斗前后各项数值的变化。失去钥匙，观察钥匙数量是否发生变化。

测试结果：所有元素均正常更新，符合预期要求。

七. 总结和收获

1．总结部分

这个项目在制作过程中遇到了很多困难，但也正是这些困难让我对计算机项目的设计过程有了直观清晰的了解。相比于之前编写的代码来说，这次的代码更长、文件更多、更有 C++面向对象的思想，也有模块化编程的思想，需要我考虑文件之间的引用关系和变量的取名关系，也让我在写代码、调试、运行的过程中对项目编写的大体流程有了直观的看法。图形化的编程也是之前从未接触到的，这更接近于我日常生活中使用的软件，也让我知道计算机的学习之路任重而道远。

与此同时，我也更加明确了网络资源的使用方式，在编写过程中，Bilibili、CSDN、Github 等平台都给我提供了大量可以学习的项目和资源，也有很多与比我水平更高的编写者的指导与鼓励，受益匪浅。

这个程序已经可供游客游玩，但出于时间和能力原因，还有很多我想做的内容没有付诸实践，还有一些疑问没有得到很好的解决，现列举如下，既是寻求有缘看到的大佬的帮助，也是鼓舞自己持续进步：游戏账号密码的联网登录模式还未实现，想要做一个游戏，我认为最好是随时随地登录账号就能玩，而不是在固定机器上的固定存档；作弊功能的数量和权限限制还没实现，防止控制不住自己破坏自己的游戏体验；如何绑定在不同情况下实现不同作用的相同快捷键而不引起冲突；如何增加丰富的动态反馈动画等等。

2．收获部分

1、 学会了如何使用 Github 和 Git 等现代工具上传、下载项目和代码，以及更新维护自己的项目。比起用网站上传输，明显用 Git 上传更方便、限制更小。在 Github 上也有很多优秀的开源项目，让我学习到了很多东西。

2、 进一步熟悉了类的思想和使用，也熟悉了 C++特性，如继承和多态、函数和方法等，进一步地，也熟悉了模块化编程、面向对象编程的编程思想，为以后编写大型项目积攒了经验。

3、 了解了 qmake 等模块的使用条件，学会下载开发工具和配置开发环境，明确配制过程中遇到的问题，以后针对性地配置自己的电脑文件格式，更好地契合开发环境。

4、 明确了图形化编程过程中图形的制作和限制，初步探索了如何设计适宜好看的界面，契合使用者的主观感受。

5、 学会了调试分析的基本过程和基本手段。在多次的调试过程和遇到的困难，我明确了如何确定问题和寻求帮助，才能更好更快地解决问题。

6、 在阅读他人代码的过程中，我学会了如何高效地把握住他人代码的框架和思想，然后再深入阅读，理解我没接触过的语法和编写实现方式。

7、 在有疑问的过程中，我更好地明确了如何应用现代 AI 工具帮助我解决问题，如何明确地发出指令并收获答案；也明确了更有效的网络检索方式，加入了一些社群，为日后的学习提供了更多的可能性。

8、 学会了编写过程中较为良好的风格和变量取名方式，为协作编码提供了基础。

9、 熟悉了 Qt 框架，Qt 具有丰富的功能和组件，如界面设计、信号与槽机制、图形绘制等。编写 Qt 项目帮助我熟悉了 Qt 框架的使用，提升开发效率。