魔塔编写报告

一.游戏介绍

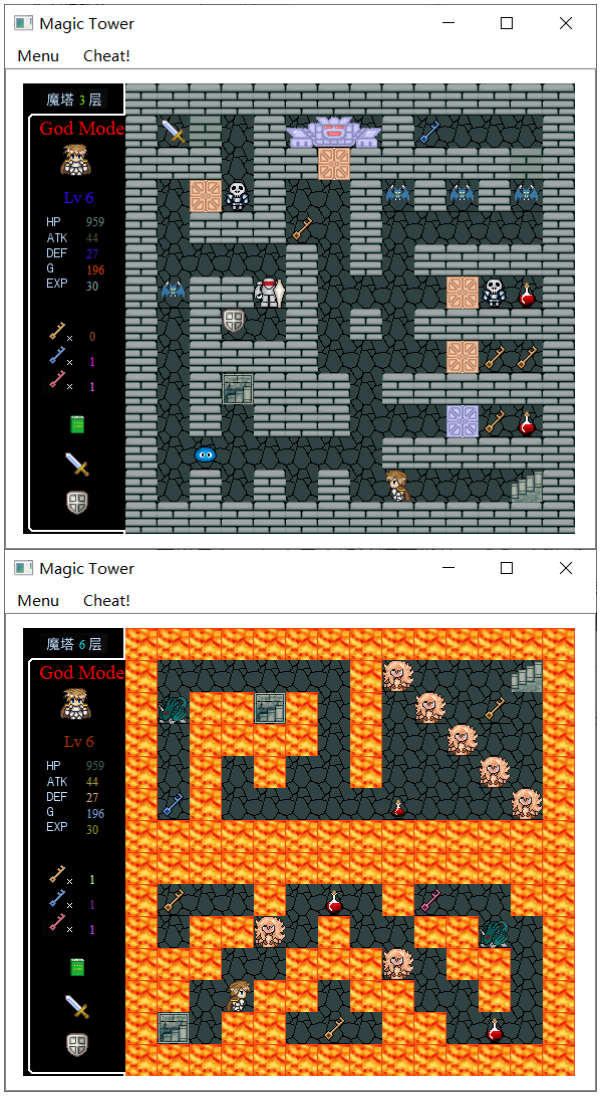
魔塔是在96年由日本设计师开发的一个flash游戏，代表着难度与经典。故事背景大约是勇者救公主但是打到一周目结局时，公主只是一个假人，魔王的数据高不可攀，勇者重回第一层，怪物属性翻44倍，勇者轮回的成长，魔王的数据永远高不可攀只能用魔法炸弹炸死，无限无尽的rpg探险游戏。

这里实现的魔塔是有着程序员风格、有美好结局但也有一定难度的魔塔翻版游戏。

玩家拥有hp、attack、金钱等属性，物品栏有武器、三种颜色的钥匙，能够通过打怪获得经验和金币，需要有一定的路线选择策略、适当的升级探索来继续进行，否则会因为hp过低或者没有对应的钥匙而完全卡关。

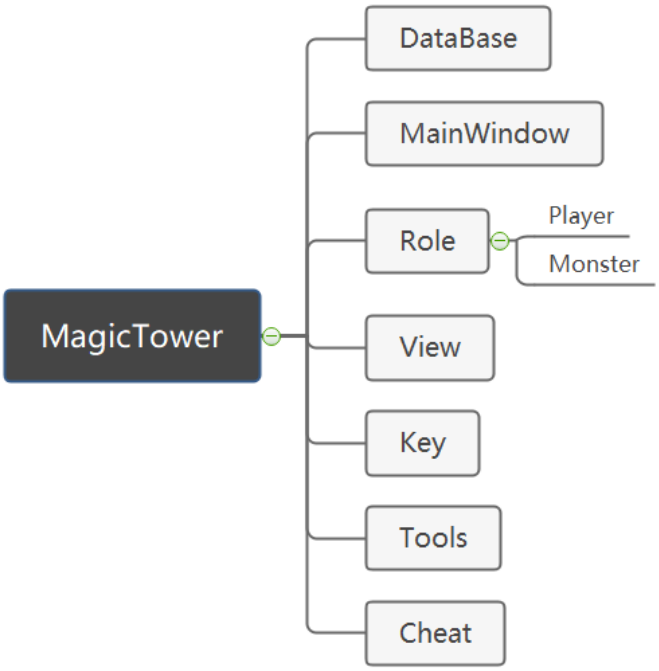
游戏asset一部分来自unity asset store，一部分来自于网络图片，bgm来自于网易云。

游戏效果：



二.程序结构

1.各种类



*DataBase*功能:

1.建立和data.sql的连接，载入地图数据、player数据、monster数据、Tools数据，keys数据

2.与保存游戏加载游戏相关的功能。

```
class DataBase {
public:
    void Connect(QString dbpath);
    void SaveMap(int num);
    void LoadMap(int num);//还有一些列一样形式的load和save函数，不一一列出
private:
    QString layer(int num);
    QSqlDatabase db;//用于读取和保存的sql
};
```

*MainWindow*功能:

1.初始化一系列的QAction，并且将QAction和对应的槽函数相连

2.创建各种画面，包括菜单、主角属性、战斗场面

3.添加各种物品的图片

4.实现了一系列的槽函数，包括画出整个场景、新建保存加载游戏、战斗、商店、查看怪物册

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    void Welcome();
    void Clear();
    void CreateActions();//多个create函数，包括menu、fight、inform

    void AddMapItem(int x, int y, int num);//多个add：有picture、player、text
    QString GetExpectedDamage(int num);

public slots:
```

```
    void slotDrawScene();//多个slot函数：slotNewGame、slotSaveGame、slotLoadGame、
slotMovePlayerItem、slotShop、slotBook、slotFight、slotEvent
private:
    View *view;
    QGraphicsScene *scene;
    QGraphicsPixmapItem *playerItem;
    QAction *clear;
    QAction *newGame;//一些列QAction，不列出
};
```

*View*功能:

1.处理按键的事件，分类如下：

a. 主要事件：移动player

b. 商店事件：选择要买的商品

c. 欢迎事件： 游戏开始时的选项

2.根据按键事件emit信号，给MainWindow处理

```
class View : public QGraphicsView {
    Q_OBJECT
signals://传给MainWindow的信号
    void move(int x, int y);
    void events(QString str);
    void fight(int num);
    void change();
    void quit();
public:
    explicit View(QWidget *parent = 0);
    ~View();
    QString GetStatus() { return status; }
    void SetStatus(QString status) { this->status = status; }
    void keyMain(QKeyEvent *event);//多种key：keyShop、keyWelcome、keySelect、
keyFight
    void action();
protected:
    void keyPressEvent(QKeyEvent *event);// Rewrite the virtual function
keyPressEvent
private:
    int access(int x, int y);
    QString status;
    int next_step;
};
```

Role: **1.维护角色属性、得到和设置属性的接口函数**

Player: **1.继承自Role类，增加选择性别、升级的功能，增加了朝向、坐标等属性和属性的设置和获得的函数**

Keys: **1.维护角色的三种钥匙数量**

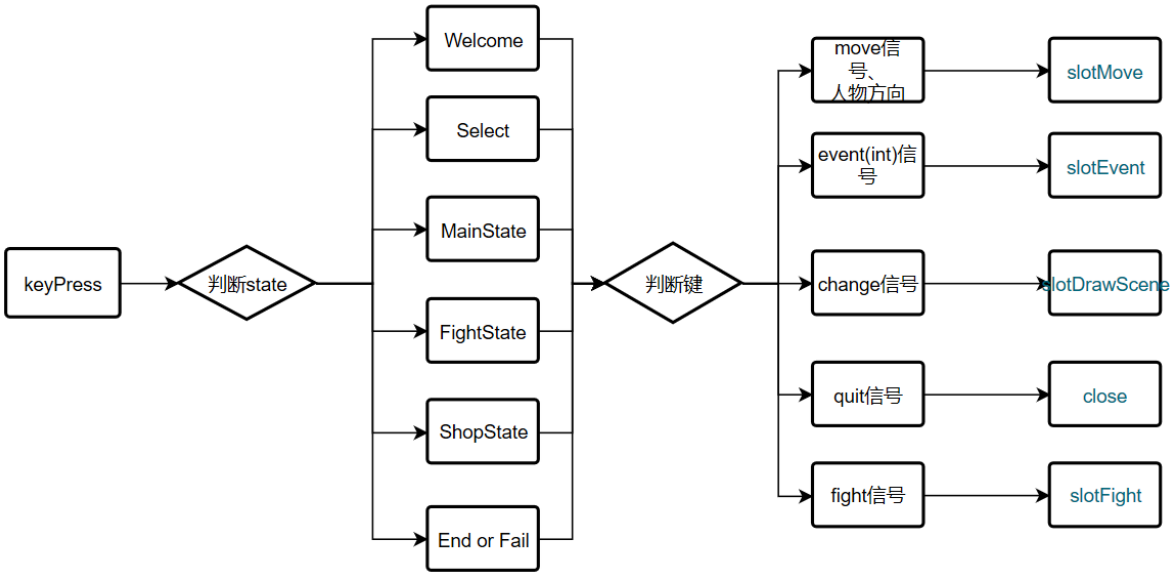
Cheat：**1.实现所有作弊行为相关的slot函数**

2.*Main*中定义的全局变量：

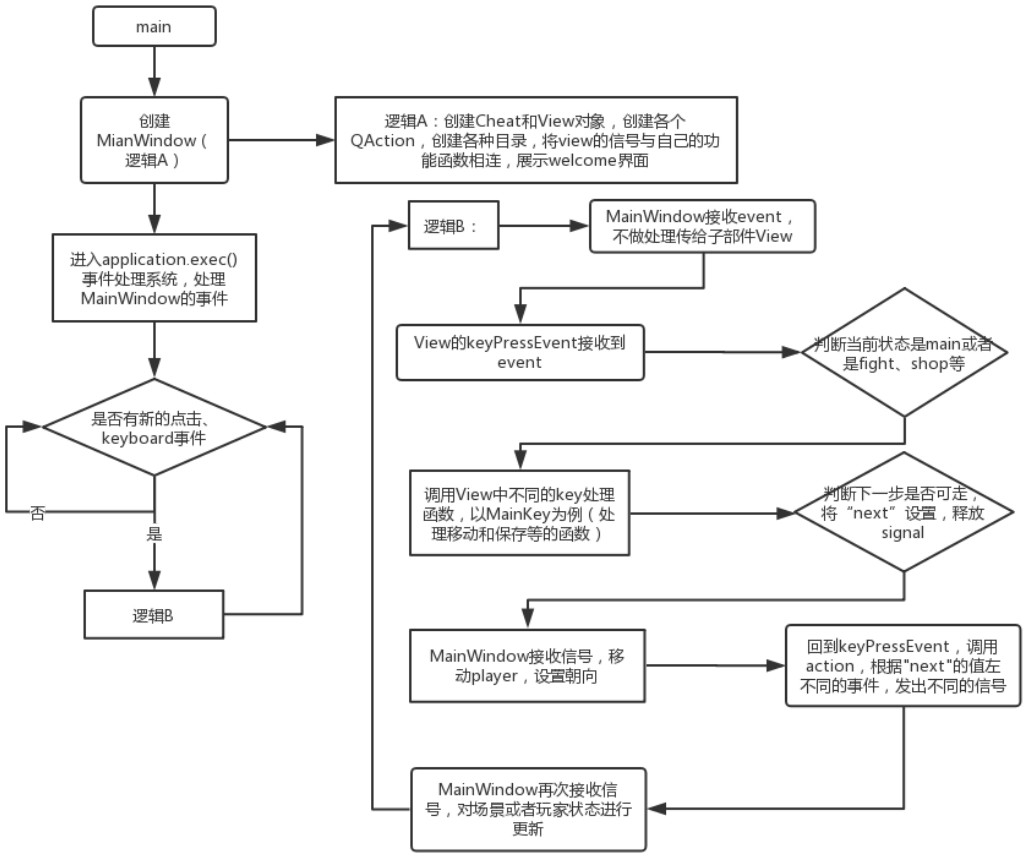
```
DataBase db;
Player player;
Monster monsters[13];
Tools tools;
Keys keys;
int map[14][14][10];//最后一维是层数，前两位是坐标，值是物品的种类
```

三.执行逻辑

槽、信号传递设置：



执行逻辑：



整个执行逻辑主要是View和MainWindow的交叉运行，通过QT的信号槽机制来进行信号的交互，View处理的是视图上的按键操作，MainWindow是对试图上的场景、人物等进行计算和图像的更新，其他类在运行期间几乎没用逻辑上的作用，只做结构上的作用。

以移动后遇到怪物进行fight为例：

```
//首先View的keyPressEvent接收来自MainWindow父部件到event事件：
void View::keyPressEvent(QKeyEvent * event) {
    if (status == "main") {         // Read key and move player
        keyMain(event);
        action();         // Different actions according to Nextstep's num
    }else....

//这时的status应该是main，跳转到View中的keyMian函数
void View::keyMain(QKeyEvent * event) {
    switch (event->key()) {
        case Qt::Key_Up:         // Move up
            if (access(player.GetPosx(), player.GetPosy()-1))
                emit move(0, -1);
            player.SetToward(1);
            break;
        case:...
//假设当前是向上按键，通过判断下一步是否可走，设置下一步的状态"next"
int View::access(int x, int y) {
    int tmp = map[x][y][player.GetFloor()];
    next_step = tmp;
    if (tmp == 0 ||...) {
        map[x][y][player.GetFloor()] = 0;
```

```
        return 1;
    } else
        return 0;
}
//回到keyMain中，emit move的信号，然后MainWindow接收
connect(view, SIGNAL(move(int, int)), this, SLOT(slotMovePlayerItem(int, int)));
//slotMovePlayerItem中更新player的位置和当前player位置的地图
//返回View的keyPressEvent执行action函数
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;
        case...
        case 23:
            // Fight with normalmonster
            SetStatus("fight");
            emit fight(next_step);
            break;
//action根据之前在access函数中设置的"next"值来判断遇到了什么物品，如果遇到了23（普通怪物）,emit fight的信号，该信号再次由MainWindow接收，并且画出战斗场景和做出主角属性的改变，结束按键事件的所有工作。
```

四.场景数据设置

sql中创建的表：

table	attribute	功能
tools	id|book|shield|sword	记录player的物品栏
map	id|layer0~layer9	记录从第0层到第9层的地图
keys	id|red|blue|yellow	记录player的钥匙持有数量
monster	id|hp|attack|defend|money|exp|miss|crit	各种怪物的属性和id号
player	id|hp|attack等|posx|floor|toward|sex	player的属性、位置、朝向等

创建表的sql语句（以monster为例）：

```
CREATE TABLE IF NOT EXISTS `monster` (
  `id` integer PRIMARY KEY AUTOINCREMENT,
  `hp` int NOT NULL,
  `attack` int NOT NULL,
  `defend` int NOT NULL,
  `money` int NOT NULL,
  `exp` int NOT NULL,
  `miss` int NOT NULL,
  `crit` int NOT NULL
);
```

值设置语句（monster为例）：

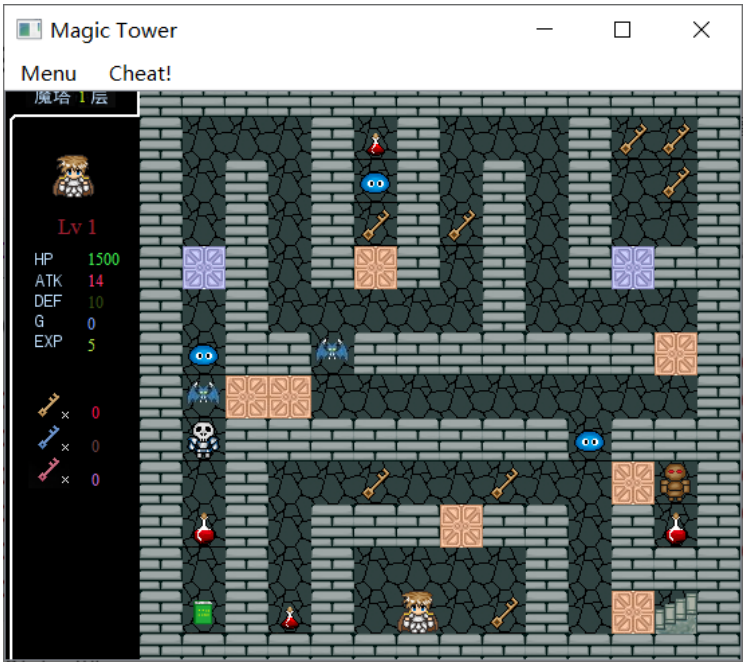
```
INSERT INTO `monster` (`hp`, `attack`, `defend`, `money`, `exp`, `miss`, `crit`,
`id`) VALUES
(50, 12, 9, 3, 1, 5, 95, 11),
(60, 14, 10, 6, 3, 5, 94, 12),
(45, 20, 11, 10, 5, 7, 93, 13),
(150, 17, 7, 15, 7, 9, 92, 14),
(95, 35, 9, 20, 8, 9, 91, 15),
(50, 16, 35, 26, 10, 0, 90, 16),
(60, 17, 10, 15, 7, 45, 93, 17),
(100, 25, 20, 25, 15, 15, 92, 18),
(100, 30, 0, 0, 0, 20, 99, 19),
(100, 40, 13, 20, 20, 8, 50, 20),
(175, 45, 25, 40, 25, 5, 95, 21),
(150, 50, 30, 50, 25, 10, 90, 22),
(300, 60, 35, 0, 0, 20, 80, 23);
```

地图的保存：

因为所有的物品的item都是互异的，所以保存的是item的id号，每一层是14*14的int类型值，如下展示了第0层的物品数据：

```
INSERT INTO `map` (`id`, `layer0`, `layer1`, `layer2`, `layer3`, `layer4`,
`layer5`, `layer6`, `layer7`, `layer8`, `layer9`) VALUES
(1,
 '02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 00 00 00 02 29 02 00 00 00 02 06
06 02 02 00 02 00 02 11 02 00 02 00 02 00 06 02 02 00 02 00 02 06 02 06 02 00 02
00 00 02 02 04 02 00 02 03 02 00 02 00 02 04 02 02 02 00 02 00 00 00 00 00 02 00
00 00 00 02 02 11 02 02 12 02 02 02 02 02 02 02 03 02 02 12 03 03 00 00 00 00 00
00 00 00 00 02 02 13 02 02 02 02 02 02 02 02 11 02 02 02 02 00 02 00 00 06 00 00
06 00 00 03 14 02 02 30 02 00 02 02 02 03 02 02 00 02 30 02 02 00 02 00 02 00 00
00 00 02 00 02 02 02 02 35 02 29 02 00 00 00 06 02 00 03 80 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02', "......", "......", ......
```

第0层示例：



关于Qt链接sql的操作参考此blog：https://blog.csdn.net/qq_43333395/article/details/91126477

五. 设计的特点

1.使用sql：

因为设计的魔塔共有十层，每层共有14*14个地图块，共有1440个地图信息，对于人工设置来说数据量很大，并且在维护时数据也难以控制，使用sql能减少数据量带来的问题，sql对同类数据设置很简单，不需要对同类对象进行派生，并且在这个游戏中同类物品的行为一样，只是各种值不一样派生显得和无意义。

在实现SaveGame和LoadGame的时候，如果使用文件存储，会造成大量格式问题，并且读取和保存操作涉及果陀文件指针的移动，使得代码量增大。

能使鲁棒性增加，QSqlDatabase拥有自动检测语法功能，sql设置时可以设置primary key的性质，保证了数据设计时的正确性，减少编写过程的数据调整。

2.信号机制：

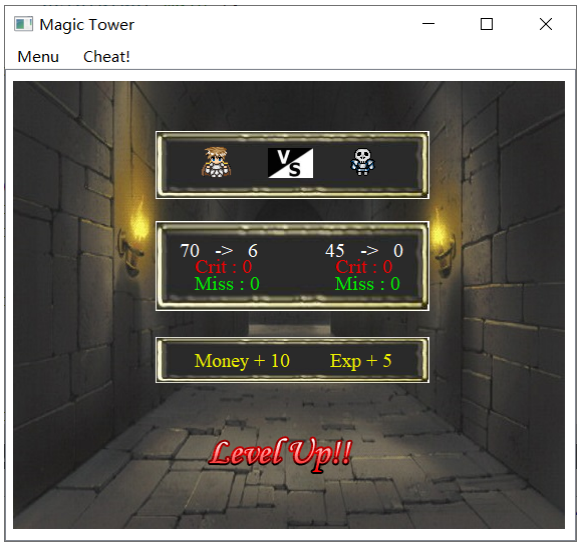
在运行过程中，会有多种View和MainWindow交互调用的情况，大量使用qt的信号槽机制，实现不同信号对应同一种调用方案，根据状态值不同而做出不同反应的效果，减少了重复代码的实现。

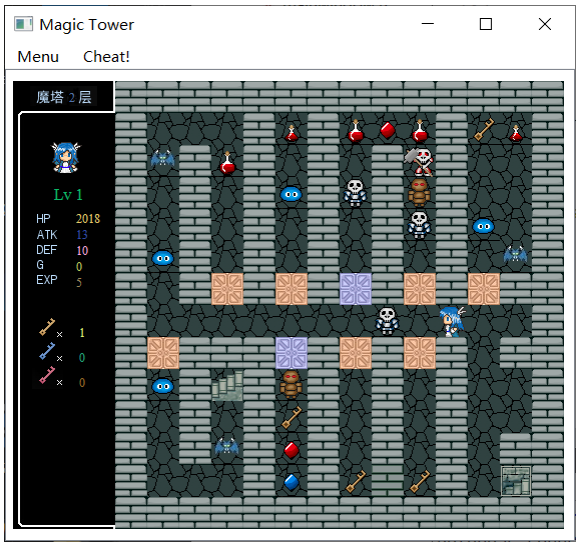
3.物品种类的设计：

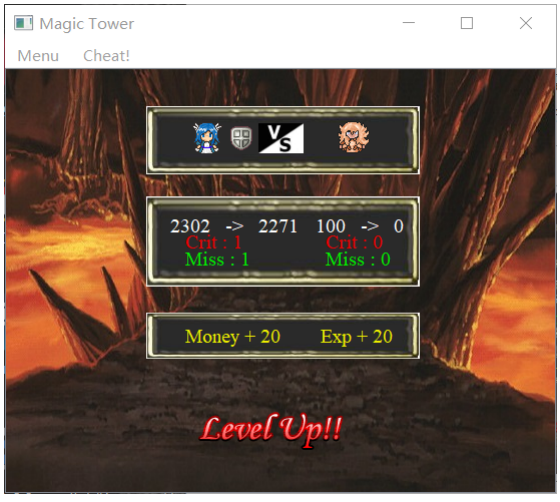
设计了尽量与原作相同的怪物、道具，多达13种怪物和几种种不同类型，要素过多。

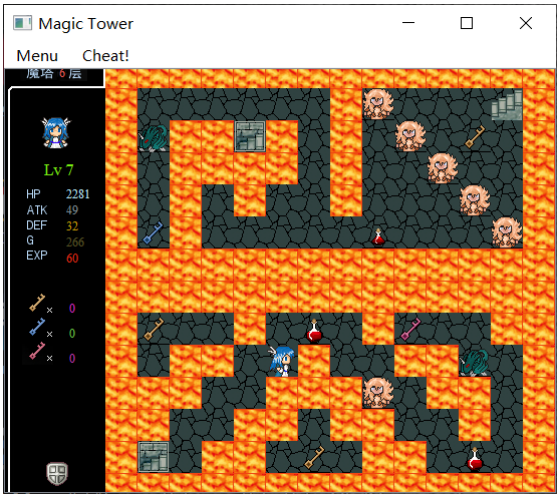
4.游戏体验：

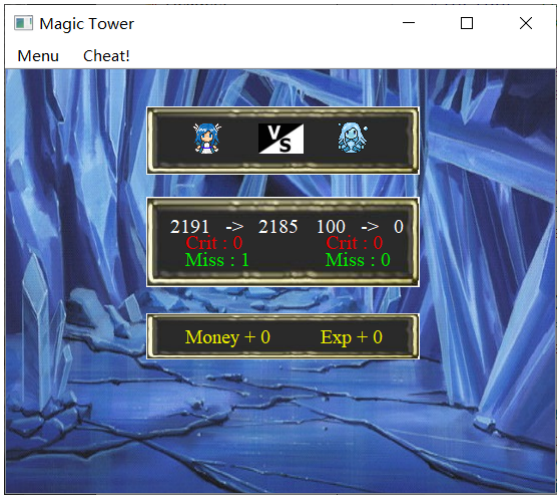
游戏画面丰富，人物属性的text会随机显示颜色，多个地图的tile根据层数不同发生改变，在不同层次的时候，战斗场景也会有不同的表现，并且配备了背景音乐。



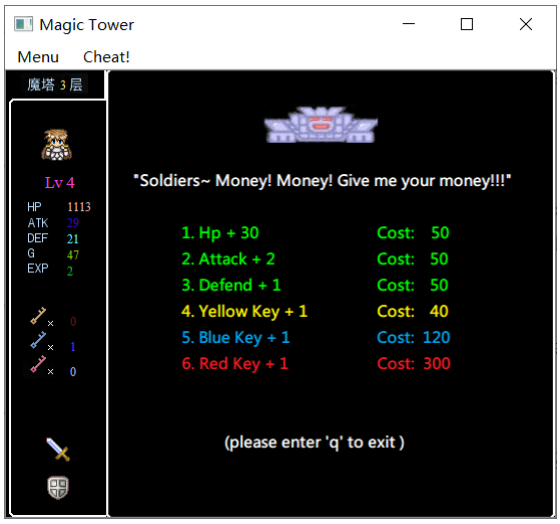


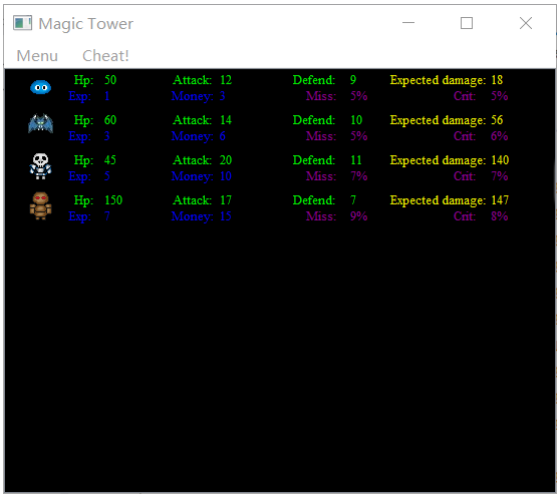




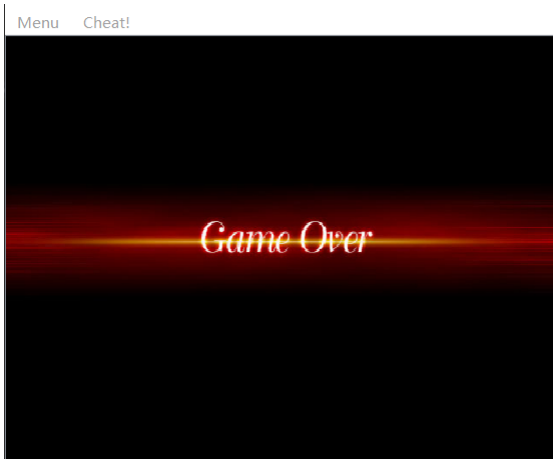


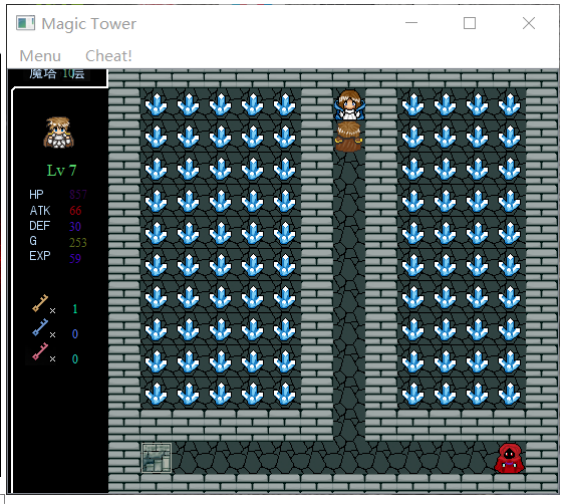
可以与多种物品进行互动，与怪物对战有战斗场面，可以读取手册，与npc对话，在商店购买物品。

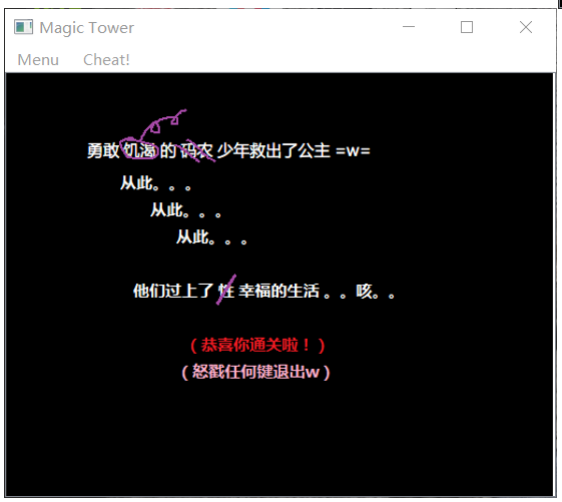




策略设置：游戏有一定的难度，需要进行路线的选择和金钱使用的策略，否则玩家会死亡或者进入死胡同。在上下楼梯时，会因为不同方向的进入而到达楼层不同的地方，到达看似不能到达的地方。并且会有隐藏房间，即可遇到假墙壁并且突破。正常的游戏模式大概需要半小时的流程来进行。







六.遇到的问题

1.音乐的播放

循环导致段错误，多次添加也会导致段错误，增加了music_thread线程来避免。

2.exec逻辑、信号机制

qt的exe逻辑是自动运行widget的函数循环执行，不需要在代码中显式的调用，而是通过事件的驱动进行。

关于信号的传递，是父部件自动传递给子部件，也是Qt自动的事件机制，不需要显示调用，在理解时有困难。

3.sql语法

AUTOINCREMENT不兼容，在sql sever中主键的设置是AUTO_INCREMENT，用于保证table中的主键（id）是递增的，适合于编程过程中的种类判断，但是在c++中AUTO_INCREMENT是invalid syntax，在不同环境下语法有不同。