

南开大学

本科生 C++ 图形化界面开发(设计)

中文题目: C++ Qt 复合式推箱子开发

版本: 2024/04/24

外文题目: Nankai University QT programme for CS

学号: 2311819

姓名: 王雨萌

年级: 2023 级

专业: 工科试验班 (信息科学与技术)

系别: 工科试验班 (信息科学与技术)

学院: 网络空间安全学院

完成日期: 2024 年 5 月

摘 要

本篇文章旨在介绍我的 Qt 项目开发的思考以及实现的过程，作者尽量把每一个细节说明清楚。

关键词：Qmake, 推箱子, C++ GUI, Qt creator6.2.1

Abstract

The article is aimed at introducing my journey to realize what I picture. It is advised that I will detail my work and wish you can grasp my thinking.

Key Words: Qmake, Push the Box, C++ GUI, Qt creator 6.2.1

目 录

摘要.....	I
Abstract.....	II
目录.....	III
第一章 课程要求.....	1
第二章 主要流程.....	2
第一节 整体流程.....	2
第二节 具体实现 1.0 - 经典模式实现	2
2.2.1 功能 1 - 地图的二维映射实现	2
2.2.2 功能 2 - 人物移动	4
2.2.3 功能 3 - 方向与人物转动效果	6
2.2.4 功能 4 - 逻辑碰撞函数	6
2.2.5 功能 5 - 胜利检测以及文件加载检测	9
第三节 具体实现 2.0 - 游戏进入界面设计	10
2.3.1 按钮 1 - 游戏进入主界面的按钮弹跳的设计	10
2.3.2 按钮 2 - 选择按钮跳转界面	12
2.3.3 界面 1 - 关卡的界面设计	13
第四节 具体实现 3.0 - 双人模式设计	14
2.4.1 功能 1 - 服务器通讯	14
2.4.2 功能 2 - 客户端连接与通讯	16
2.4.3 功能 3 - 网络游戏构建	17
第五节 单元检测.....	18
2.5.1 检测 1 - 观察地图容纳入链表容器的分割过程是否正确	18
2.5.2 检测 2 - 观察图片是否载入 GUI	19
附录.....	20

第一章 课程要求

摘要中包括了对我使用的 GUI 编程平台的介绍, 以及我的作业题目, 不再赘述。大作业代码提交到 github 或者 gitee (码云, 中国开源社区平台, 对等 github) www.gitee.com, 要求能够体现大作业完成过程中的不同版本迭代。在创建项目时, 点击页面顶部菜单栏账号左侧的加号 (位于页面右侧) 来新建仓库。创建仓库时, 应选择公开方式, C++ 语言。

1. 面向对象。采用 C++ 面向对象的编程思维和模式
2. 单元测试。因为采用了面向对象的编程方向, 所以我们可以对 Qt 项目分解成功能实现单元。
3. 模型部分。我们先采用先理论模型的构建, 然后完成模型的实现
4. 验证。最后对 Qt 项目进行功能验证。
5. 简洁易读的模板实现。使用常用宏包实现排版, 方便二次修改。

当前执行标准为《本科毕业论文 (设计) 指导手册 (2022 年 11 月)》和《本科生学位论文排版规范 (2023 修订) (版本 2.1.1) (计算机学院) (网络空间安全学院)》

第二章 主要流程

第一节 整体流程

我们实现的过程：地图的二维映射 - 人物移动逻辑实现 - 逻辑碰撞的实现 - 胜利检测/文件加载检测；

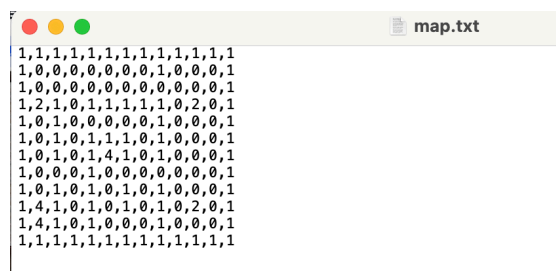
1. **地图二维映射**: 通过数字图像简单模拟地图，通过 List 链表容纳并且实现地图的映射。
2. **人物移动逻辑实现**: 这一部分主要是完成人物移动的边界设置、人物在指定的路径完成运动、人物移动的键盘事件的设置、人物移动-关于人物的身体转动的效果实现
3. **逻辑碰撞的实现**: 主要是通过坐标检测的方式完成人物与地图元素的交互逻辑的实现
4. **胜利检测/文件加载检测**: 比较简单，目的是为了正确加载地图文件以及实时更新并且检测是否完成了胜利条件。

第二节 具体实现 1.0 - 经典模式实现

下面的部分我将会按照 Qt 源代码以及我的编辑日志来详细说明功能实现的路径。

2.2.1 功能 1 - 地图的二维映射实现

我们首先做一个 txt 文件写入我们 map 抽象映射。如图片 2.1，并且储存在 qrc 资源文件布局。



```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,1,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,1
1,2,1,0,1,1,1,1,1,0,2,0,1
1,0,1,0,0,0,0,0,1,0,0,0,1
1,0,1,0,1,1,1,0,1,0,0,0,1
1,0,1,0,1,4,1,0,1,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,1
1,0,1,0,1,0,1,0,1,0,0,0,1
1,4,1,0,1,0,1,0,1,0,2,0,1
1,4,1,0,1,0,0,0,1,0,0,0,1
1,1,1,1,1,1,1,1,1,1,1,1,1
```

图 2.1 数字二维地图 - 映射

```

bool GameMap::initByFile(QString FileName)//这里做一个二维地图的基础映射 (0, 1)
{
    QFile file (FileName);
    if (! file .open(QIODevice::ReadOnly)) return false;

    //读取所有内容
    QByteArray arrAll = file .readAll(); //注意这是字节数组
    arrAll .replace("\r\n","\n");
    QList<QByteArray>linelist = arrAll .split ('\n'); //使用链表容器来容纳
    mRow = linelist .size (); //确定行数

    mPArr = new int*[mRow];

    for(int i = 0; i< mRow; i++)
    {
        QList<QByteArray> colList = linelist [i] .split (',' );
        mCol = colList .size (); //确定列
        mPArr[i] = new int[mCol]; //开辟列

        for(int j = 0; j < mCol; j++) mPArr[i][j] = colList [j] .toInt ();
    }
    return true;
}

```

至此，我们完成了对 map.txt 文件的读取、内容储存为链表容器的内容物，同时再次储存到 GameMap 类中的二维数组中。建立起抽象的二维映射地图。

```

void GameMap::Paint(QPainter *_p,QPoint _Pos)
{
    for(int i = 0; i<mRow;i++)
    {
        for(int j = 0; j<mCol;j++)
        {
            QString imgUrl;
            switch(mPArr[i][j])
            {
                case 0: imgUrl = ":/Ele/MapImg/road.png"; break;
                case 1: imgUrl = ":/Ele/MapImg/wall.png"; break;
                case 2: imgUrl = ":/Ele/MapImg/box.png"; break;
                case 4: imgUrl = ":/Ele/MapImg/point.png"; break; //目前还缺少这张图片
            }
        }
    }
}

```

```

        case 5:imgUrl = ":/Ele/MapImg/inpoint.png";break;//目前还缺少这张图片
    }

    QImage img(imgUrl);
    // qDebug()<<imgUrl<<" - "<<QRect(j*img.width(),i*img.height(),img.width(),img.height());
    _p->drawImage(QRect(_Pos.x()+j*img.width()*0.5,_Pos.y()+i*img.height()*0.5,img.width()
        *0.5,img.height()*0.5),img);//注意括号!
    // _p->drawImage(_Pos,img);
}
}
}

```

2.2.2 功能 2 - 人物移动

人物位置显示

```

Role::Role(QObject *parent)
    : QObject{parent}
{
    //初始化人物所在的位置
    mRow = 1;
    mCol = 1;
    mImg2 = QImage(":/Ele/MapImg/player-d.png");
    mImg = QImage(":/Ele/MapImg/player-t.png");
    mImg3 = QImage(":/Ele/MapImg/player-l.png");
    mImg4 = QImage(":/Ele/MapImg/player-r.png");
    //显示位置:
    mPaintPos = QPoint(mCol,mRow)*mImg.width()*0.5;
    //
}

```

Problem. 人物显示有几个难点:

1. 我的图片像素比较大，如果自己裁剪会掉画质并且模糊，如何通过代码处理。
2. 如何确定 QPainter 绘制图片的位置？
3. 如何实时更新人物的位置？（根据我当时测试时，键盘事件后需要刷新窗口才会更新）

Solution. 合适的解决方案:

- 我在 QAssistant 中查询找到了 [**drawImage 函数**], 具体可以通过控制第三个 (*mImg.width()*0.5*) 和第四个参数 (*mImg.height()*0.5*) 来控制图片

```
if(mode_num == 1)
_p->drawImage(QRect(+_pos.x()+mPaintPos.x(),_pos.y()+mPaintPos.y(),mImg.width()*0.5,mImg.height()*0.5)
,mImg);
```

- 我通过学习了解到 Qmake 中自带**向量类 QPoint**, QPoint 的运算法则和自由向量的运算法则相同。窗口左上角的向量映射坐标为 (0,0);

```
//位置向量
mPaintPos = QPoint(mCol,mRow)*mImg.width()*0.5;

//对于图片位置的向量描述
QRect(+_pos.x()+mPaintPos.x(),_pos.y()+mPaintPos.y(),mImg.width()*0.5,mImg.height()*0.5)
```

- 我们通过学习知道, Qt 内置 **QTimer** 类用于定时, 以及 **connect 函数** 接受信号并且反馈操作, 此外, 我还专门学习了 C++11 的新特性 - **lambda 表达式**, 因为没有相关的槽函数实现我需要的更新操作。

```
//设置时钟
mtimer = new QTimer(this);
mtimer->start(100);

connect(mtimer,&QTimer::timeout,[ this ]() {
    this->update();
}); //定时更新绘图
```

人物移动函数

```
void Role::move(int _dRow,int _dCol,int mode_num)
{
    mRow += _dRow;
    mCol += _dCol;

    if(mode_num == 1)
    {
        mPaintPos = QPoint(mCol,mRow)*mImg.width()*0.5;
    } else if(mode_num == 2)
```

```

{
    mPaintPos = QPoint(mCol,mRow)*mImg2.width()*0.5;
} else if (mode_num == 3)
{
    mPaintPos = QPoint(mCol,mRow)*mImg3.width()*0.5;
} else if (mode_num == 4)
{
    mPaintPos = QPoint(mCol,mRow)*mImg4.width()*0.5;
}
}

```

2.2.3 功能 3 - 方向与人物转动效果

```

void Role:: Paint_self (QPainter *_p,QPoint _pos,int mode_num)
{
    if(mode_num == 1)_p->drawImage(QRect(+_pos.x()+mPaintPos.x(),_pos.y()+mPaintPos.y(),mImg.width()
        *0.5,mImg.height()*0.5),mImg);
    if(mode_num == 2)_p->drawImage(QRect(+_pos.x()+mPaintPos.x(),_pos.y()+mPaintPos.y(),mImg2.width
        ()*0.5,mImg2.height()*0.5),mImg2);
    if(mode_num == 3)_p->drawImage(QRect(+_pos.x()+mPaintPos.x(),_pos.y()+mPaintPos.y(),mImg3.width
        ()*0.5,mImg3.height()*0.5),mImg3);
    if(mode_num == 4)_p->drawImage(QRect(+_pos.x()+mPaintPos.x(),_pos.y()+mPaintPos.y(),mImg2.width
        ()*0.5,mImg2.height()*0.5),mImg4);
}

```

我们在这里通过 modenum, 当键盘事件发生后, 我们就传入对应的 modenum 实现人物的朝向的改变。如上图片 2.2;

2.2.4 功能 4 - 逻辑碰撞函数

```

void Widget::keyPressEvent(QKeyEvent *event)
{
    switch(event->key())//设置人物的键盘事件 - 人物的上下左右移动
    {
        case Qt::Key_W:
        {
            Collision (-1,0,1) ;
            break;

```

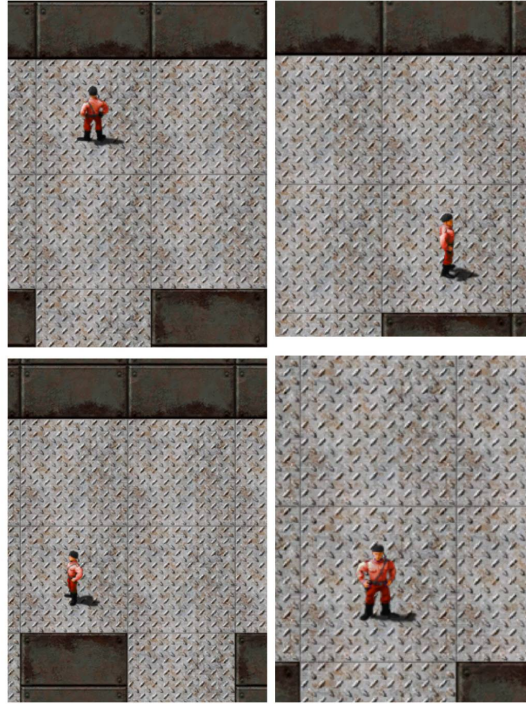


图 2.2 方向改变以及人物转动的效果

```
}  
case Qt::Key_S:  
{  
    Collision (1,0,2) ;  
    break;  
}  
case Qt::Key_A:  
{  
    Collision (0,-1,3) ;  
    break;  
}  
case Qt::Key_D:  
{  
    Collision (0,1,4) ;  
    break;  
}  
}  
  
//逻辑碰撞检测函数  
}  
  
void Widget:: Collision (int _dRow, int _dCol, int num)  
{
```

```

//判断位置
int newRow = mRole->mRow + _dRow;
int newCol = mRole->mCol + _dCol;

mode_num = num;

//通过改变地图元素 - 实现逻辑碰撞
if (mPMap->mPArr[newRow][newCol] == 1)
{
    return;
} else if (mPMap->mPArr[newRow][newCol] == 2) //2是箱子的代数
{
    if (mPMap->mPArr[newRow+_dRow][newCol+_dCol] == 0)
    {
        //改变地图元素
        mPMap->mPArr[newRow+_dRow][newCol+_dCol] = 2;
        mPMap->mPArr[newRow][newCol] = 0;
    } else if (mPMap->mPArr[newRow+_dRow][newCol+_dCol] == 4)
    {
        mPMap->mPArr[newRow+_dRow][newCol+_dCol] = 5;
        mPMap->mPArr[newRow][newCol] = 0;
    } else {
        return; //否则不能推动箱子
    }
} else if (mPMap->mPArr[newRow][newCol] == 5) {
    if (mPMap->mPArr[newRow+_dRow][newCol+_dCol] == 0)
    {
        //改变地图元素
        mPMap->mPArr[newRow+_dRow][newCol+_dCol] = 4;
        mPMap->mPArr[newRow][newCol] = 2;
    } else if (mPMap->mPArr[newRow+_dRow][newCol+_dCol] == 4) {
        mPMap->mPArr[newRow+_dRow][newCol+_dCol] = 5;
        mPMap->mPArr[newRow][newCol] = 4;
    } else {
        return;
    }
}

mRole->move(_dRow,_dCol,num);
qDebug()<<"角色将要前往"<<QPoint(newRow,newCol)<<mPMap->mPArr[4][3];

```

```
win()); //每次发生一次键盘事件，我们都去检测一次胜利条件
}
```

具体碰撞函数的解释直接参考上面代码段的注释。主要原理简介：通过改变地图元素位置实现逻辑上的人物推动箱子，箱子进入洞中。

2.2.5 功能 5 - 胜利检测以及文件加载检测

胜利检测

```
void Widget::win() //胜利检测函数：没有优化过，可能会出现突然卡住的情况：我们优化的考量是：
    开局就检测所有4，然后观察这些4是否exist;
{
    for(int i = 0; i < mPMap->mRow; i++)
    {
        for(int j = 0; j < mPMap->mCol; j++)
        {
            if(mPMap->mPArr[i][j] == 4)
            {
                return;
            }
        }
    }
    QMessageBox::warning(this, "胜利！", "哥们，您已经完成胜利☑");
}
```

文件加载检测

```
QString fileName = ":/Ele/map.txt";
mPMap->initByFile(fileName);
if(!mPMap->initByFile(fileName))
{
    QMessageBox::warning(this, "警告", "地图文件初始化失败"); //提醒用户文件配置出现问题
}
```

第三节 具体实现 2.0 - 游戏进入界面设计

2.3.1 按钮 1 - 游戏进入主界面的按钮弹跳的设计

- 这个我们封装了一个 `<myStartBtn.h>` 类来实现所有按钮的美术效果，具体实现的效果有：

1. 使得各种形状的按钮都可以完美插入界面中
2. 按钮的鼠标点击事件 - 按钮点击后效果
3. 按钮弹跳

按钮弹跳效果函数 这里面我们调用了 `<QPropertyAnimation>` 类，这个类可以实现 `QObject` 类的动画效果，尤其是对于图片，我们通过改变图片矩的初始位置和末尾位置，同时设置符合一般弹跳的弹跳曲线 - 抛物线。

显然，`zoom1` 和 `zoom2` 分别实现了按钮弹跳起，按钮弹跳回。

```
void MyStartBtn::zoom1()
{
    //创建对象
    QPropertyAnimation * animation = new QPropertyAnimation (this,"geometry");
    //设置动画间隔时间
    animation->setDuration(200);
    //起始位置
    animation->setStartValue (QRect(this->x(), this->y(), this->width(), this->height()));
    animation->setEndValue(QRect(this->x(), this->y()+10, this->width(), this->height()));
    //设置弹跳曲线
    animation->setEasingCurve(QEasingCurve::OutBounce);

    animation->start ();
}

void MyStartBtn::zoom2()
```

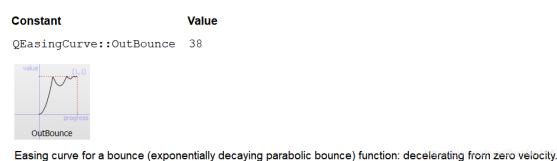


图 2.3 Qt Assistant - QEasingCurve::OutBounce

```

{
    //创建对象
    QPropertyAnimation * animation = new QPropertyAnimation (this,"geometry");
    //设置动画间隔时间
    animation->setDuration(200);
    //起始位置
    animation-> setStartValue (QRect(this->x(), this->y()+10, this->width(), this->height()));
    animation->setEndValue(QRect(this->x(), this->y(), this->width(), this->height()));
    //设置弹跳曲线
    animation->setEasingCurve(QEasingCurve::OutBounce);

    animation-> start ();
}

```

按钮的鼠标点击事件 - 实现正式点击效果 这里我们通过鼠标按压 [对应 MouseEvent] 显示第一张图片，然后通过鼠标释放 [对应 MouseReleaseEvent] 切换为第二张图片，最后我们实现图片明暗切换，实现正式点击效果。

```

void MyStartBtn::mousePressEvent(QMouseEvent *e)
{
    if ( this->pressImgPath != "" )
    {
        QPixmap pix;
        bool ret = pix.load( this->pressImgPath );
        if (! ret )
        {
            QMessageBox::warning(this,"警告","图片文件初始化失败");
            return;
        }

        //让父类执行其他内容：
        return QPushButton::mousePressEvent(e);
    }
}

void MyStartBtn::mouseReleaseEvent(QMouseEvent *e)
{
    if ( this->pressImgPath != "" )
    {

```

```

QPixmap pix;

bool ret = pix.load( this->normalImagePath );

if (! ret )
{
    QMessageBox::warning(this,"警告","图片文件初始化失败");

    return;
}

//让父类执行其他内容:
return QPushButton::mousePressEvent(e);
}

```

兼容各种形状按钮图片 因为 MyStartBtn 继承自我们的 `<QPushButton>`, 同时它继承自 `<QAbstractButton>`, 那么我们调用其中的 Public 函数实现效果, 见图片 2.4

```

//设置图片固定大小
this->setFixedSize(QSize(pix.width() ,pix.height()));

//设置不规则图片样式
this->setStyleSheet("QPushButton{border:0px;}");

//设置图标
this->setIcon(pix) ;

//设置图标大小
this->setIconSize(QSize(pix.width(),pix.height()));

```

2.3.2 按钮 2 - 选择按钮跳转界面

- 首先, 按钮跳转界面跨越了两个界面, 并且画面跳转有两个方向: **返回上一个界面**和**前进到下一个界面**。

自定义信号

```

//在界面类的.h的类内

signals :

//写一个可以告诉主场景的我要返回的信号

void chooseSceneBack();

```


Public Functions	
<code>QAbstractButton(QWidget *parent = nullptr)</code>	
<code>virtual ~QAbstractButton()</code>	
<code>bool autoExclusive() const</code>	
<code>bool autoRepeat() const</code>	
<code>int autoRepeatDelay() const</code>	
<code>int autoRepeatInterval() const</code>	
<code>QButtonGroup * group() const</code>	
<code>QIcon icon() const</code>	
<code>QSize iconSize() const</code>	
<code>bool isCheckedable() const</code>	
<code>bool isCheckeded() const</code>	
<code>bool isDown() const</code>	
<code>void setAutoExclusive(bool)</code>	
<code>void setAutoRepeat(bool)</code>	
<code>void setAutoRepeatDelay(int)</code>	
<code>void setAutoRepeatInterval(int)</code>	
<code>void setCheckable(bool)</code>	
<code>void setDown(bool)</code>	
<code>void setIcon(const QIcon &icon)</code>	
<code>void setShortcut(const QKeySequence &key)</code>	
<code>void setText(const QString &text)</code>	
<code>QKeySequence shortcut() const</code>	
<code>QString text() const</code>	

图 2.4 QPushButton Public Function

自定义槽函数

```
// 构建选择模式的窗口
chooseModeS = new ChooseModeScene;

// 监听上一个界面的信号，使用Connect+lambda表达式实现自定义槽函数
connect(chooseModeS,&ChooseModeScene::chooseSceneBack,[=]() {
    this->show();
    chooseModeS->hide();
});
```

同理，我们就可以实现前进到下一个界面的功能。

2.3.3 界面 1 - 关卡的界面设计

具体界面设计，我们可以参考我在 Github 上上传的代码段【具体见附录链接】，具体可以参考这些文件 <ChooseLevelScene>、<ChooseModeScene>、<Main-Scene>。

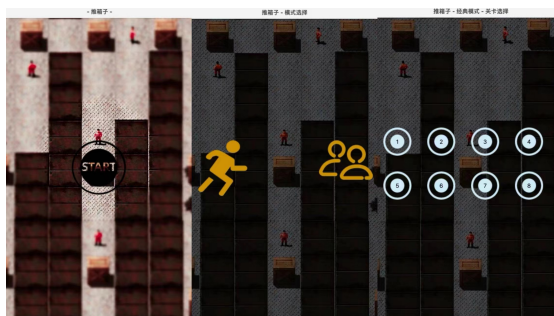


图 2.5 主界面 - 模式选择界面 - 关卡选择界面

第四节 具体实现 3.0 - 双人模式设计

这一部分主要是使用了 **QT += network** 的模块。使用套接字 (QTcpSocket) 的原理实现数据流在客户端以及服务器端的传输。

同时我们还添加了 client.ui、sever.ui 两个 ui 界面文件辅助快速完成对于界面的设计。

2.4.1 功能 1 - 服务器通讯

主体

```
setWindowTitle("服务器");

//创建监听的服务器对象
m_s = new QTcpServer(this); //?可能会出现问题!

connect(m_s,&QTcpServer::newConnection,this,[=]() {
    m_tcp = m_s->nextPendingConnection();
    m_status->setPixmap(QPixmap(":/Ele/dui.png").scaled(20,20));

    //检测是否可以接受数据
    connect(m_tcp,&QTcpSocket::readyRead,this,[=]() {

        QByteArray data = m_tcp->readAll();
        ui->Record->append("客户端: " + data);
    });

    connect(m_tcp,&QTcpSocket::disconnected,this,[=]() {
        m_tcp->close();
        m_tcp->deleteLater(); // delete;
        m_status->setPixmap(QPixmap(":/Ele/cuowu.png").scaled(20,20));

    });
});
```

监听服务函数

```
void Sever:: on_setListen_clicked ()
{
    unsigned short port = ui->port->text().toUShort();
```



图 2.6 房主端口 - 服务器端

```

m_s->listen(QHostAddress::Any,port);
ui->setListen->setDisabled(true);
}

```

服务器启动游戏

```

void Sever::on_StartGame_clicked()
{
    QString mesg = "#[系统自动]游戏即将开始……";
    m_tcp->write(mesg.toUtf8());
    // m_tcp->close();
    // this->hide();
    netGame = new Net_game;

    netGame->setWindowTitle("房主的room");
    netGame->show();
    connect(netGame,&Net_game::whowin,[=]() {
        QString mesg = "[系统自动]1房主（服务器端）完成了胜利";
        m_tcp->write(mesg.toUtf8());
        emit this->endGame();
    });
}

```

```
});
}
```

2.4.2 功能 2 - 客户端连接与通讯

具体关于按钮我们不再赘述，我们重点关注我们的客户端与服务器端的链接主体部分：

```
connect(m_tcp,&QTcpSocket::readyRead,this,[=]() {
    QByteArray data = m_tcp->readAll();
    ui->record->append("服务器端： " + data);

    if (data[0] == '#') //这里偷了个小懒~ 可以有更好的方法
    {
        netGame->show();
    }
    if (data[0] == '!' && data[7] == '1')
    {
        QMessageBox::warning(this,"失败","你失败了");
        netGame->hide();
    }
});

connect(m_tcp,&QTcpSocket::disconnected,this,[=]() {
    //QString mesg = "服务器已经断开和你的连接……";
    //m_tcp->write(mesg.toUtf8());
    m_tcp->close();
    m_tcp->deleteLater(); // delete ;
    m_status->setPixmap(QPixmap(":/Ele/cuowu.png").scaled(20,20));
    ui->record->append("□-□服务器已经断开和你的连接……");

});

connect(m_tcp,&QTcpSocket::connected,this,[=]() {
    m_status->setPixmap(QPixmap(":/Ele/dui.png").scaled(20,20));
    ui->record->append("□-□已成功连接到服务器……");
    ui->connectToH->setDisabled(true);
    ui->disconnectbtn->setEnabled(true);
});
```



图 2.7 游客端口 - 客户端

```
});

//状态栏目
m_status = new QLabel;
m_status->setPixmap(QPixmap(":/Ele/cuowu.png").scaled(20,20));
ui->statusbar->addWidget(new QLabel("连接状态: "));
ui->statusbar->addWidget(m_status);
```

2.4.3 功能 3 - 网络游戏构建

网络游戏的构建原理和经典模式的游戏构建原理差不多。大致上只需要基于经典模式的地图映射、人物移动逻辑即可，我们发现只需要改变 win() 函数。

win 函数

```
void Net_game::win()//胜利检测函数：没有优化过，可能会出现突然卡住的情况：我们优化的考量
    是：开局就检测所有4，然后观察这些4是否exist;
{
```

```

for(int i = 0;i<mPMap->mRow;i++)
{
    for(int j = 0;j<mPMap->mCol;j++)
    {
        if(mPMap->mPArr[i][j] == 4)
        {
            return;
        }
    }
}

QMessageBox::warning(this,"胜利！","哥们，您已经完成胜利□");
//上一个场景监听这个发出的信号,并且实现延时效果
QTimer::singleShot(100, this ,[=]() {
    emit this->whowin();
    emit this->chooseSceneBack();
});
}

```

第五节 单元检测

单元检测主要运用 <QDebug> 头文件以及”QDebug()” 函数在控制台检测输出情况。

2.5.1 检测 1 - 观察地图容纳入链表容器的分割过程是否正确

```

//读取所有内容
QByteArray arrAll = file .readAll(); //注意这是字节数组
arrAll .replace("\r\n","");
QList<QByteArray>linelist = arrAll . split ('\n'); //使用链表容器来容纳
mRow = linelist . size (); //确定行数

// for (int i=0; i< linelist . size (); ++i)
// {
//     qDebug() << linelist . at(i);
// } //用作调试观察是否正确分割。。。

```

2.5.2 检测 2 - 观察图片是否载入 GUI

```
QImage img(imgUrl);  
// qDebug()<<imgUrl<<" - "<<QRect(j*img.width(),i*img.height(),img.width(),img.height());
```

其他检测都是通过图形化界面交互检测是否运转正常。

附 录

下面列出：介绍视频观看地址、Github 上传地址、Qt 学习参考路径

- 视频观看地址：https://www.bilibili.com/video/BV1is421N7Gs/?spm_id_from=333.1007.0.0&vd_source=88ed50b385f354ed4e0a1345a135f69d
- Github 文件地址：https://github.com/Thinking-builder/Qt_tuixiangzi
- Qt 学习路径参考
- 我的 QT 开发日志：https://www.notion.so/Qt_-_-09e1466e1da64befbac0c6254f0c2545?pvs=4
- 个人学习证明 - 个人努力打卡「图 4，图 5」：



图 .8 Qt 学习开发日志

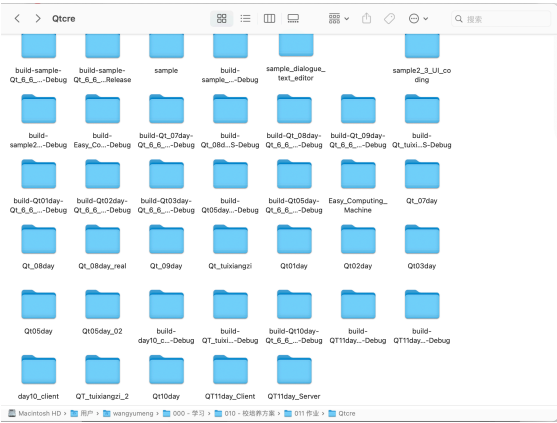


图 .9 学习全过程记录

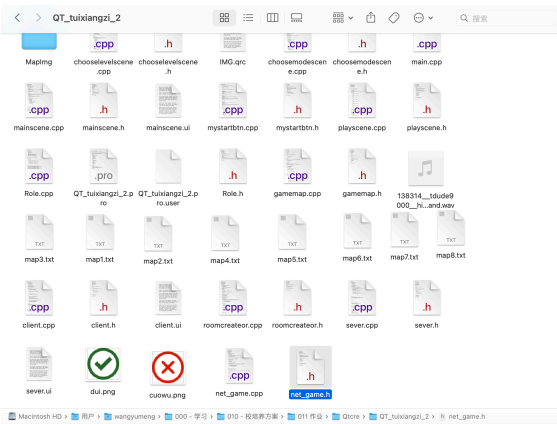


图 .10 推箱子开发全栈文件