高级语言程序设计

实验报告

南开大学 计算机大类

姓名：罗恩琪

学号：2311008

班级：0444

2024年5月13日

目录

[一、作业题目 2](#_Toc166534253)

[二、开发软件 2](#_Toc166534254)

[三、课题要求 2](#_Toc166534255)

[四、主要流程 3](#_Toc166534256)

[1、整体流程 3](#_Toc166534257)

[2、重要算法和实现 3](#_Toc166534258)

[（1）人物移动 3](#_Toc166534259)

[（2）敌人自动寻路 4](#_Toc166534260)

[（3）子弹的移动和发射 5](#_Toc166534261)

[（4）敌人的随机生成 5](#_Toc166534262)

[（5）结束画面（以失败为例） 7](#_Toc166534263)

[五、单元测试 7](#_Toc166534264)

[六、收获 9](#_Toc166534265)

# 一、作业题目

Survival肉鸽类类幸存者游戏

# 二、开发软件

QT6.7.0

# 三、课题要求

1) 面向对象

2) 单元测试

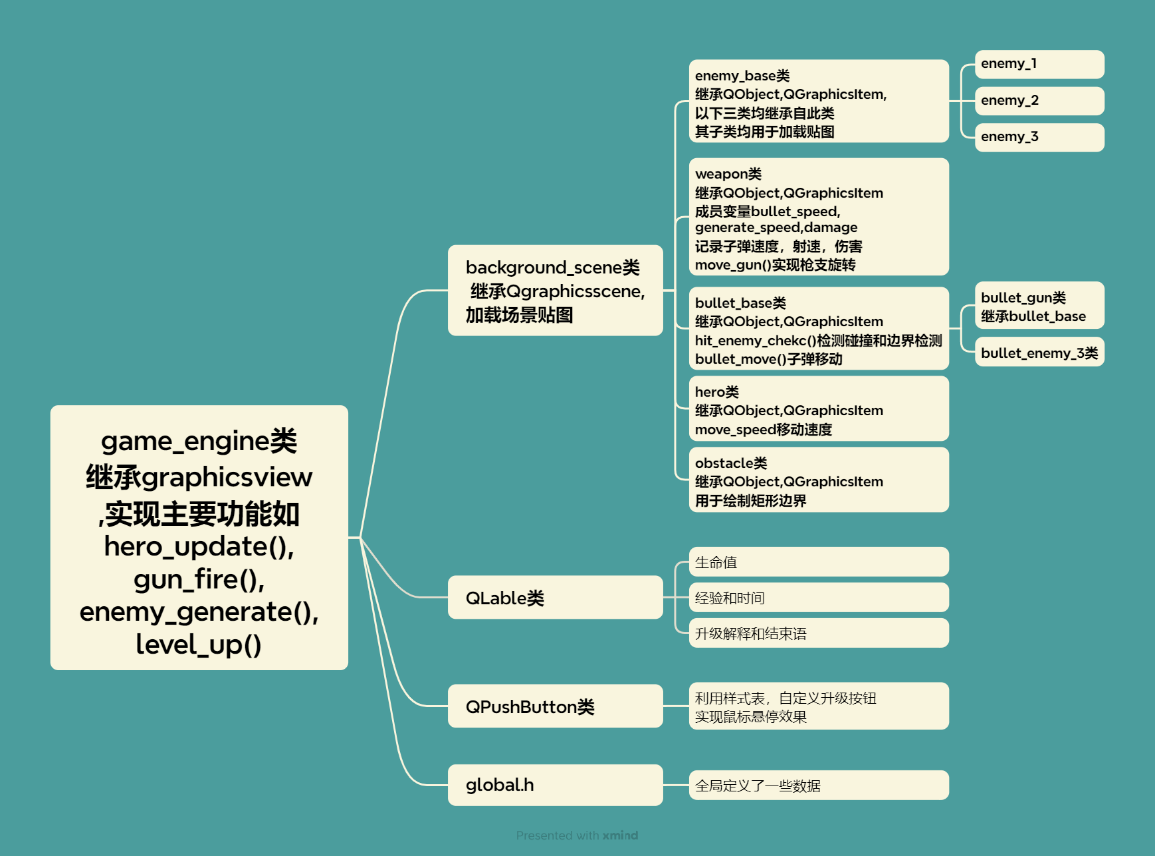
3) 模型部分

4) 验证

# 四、主要流程

## 1、整体流程

主要利用QGraphicsView,QGraphicsScene,QGraphicsItem实现

用样式表自定义了QPushButton， QLable增加了文字和一些贴图

## 2、重要算法和实现

### （1）人物移动

void game\_engine::hero\_update()

{

    map\_scene->hero\_item->movie->stop();

    bool flag=false;

    for (QMap<int, bool>::iterator it = keymap.begin(); it != keymap.end(); ++it)

    {

        if(it.value())

        {

            flag=true;

            if(it.key()==Qt::Key\_W || it.key()==Qt::Key\_S)

            {

                if(lastkey==Qt::Key\_D)

                {

                    map\_scene->hero\_item->movie->setFileName("://image/hero\_run\_right.gif");

                }

                if(lastkey==Qt::Key\_A)

                {

                    map\_scene->hero\_item->movie->setFileName("://image/hero\_run\_left.gif");

                }

                if(it.key()==Qt::Key\_W)dy-=mobile\_step;

                if(it.key()==Qt::Key\_S)dy+=mobile\_step;

            }

            if(it.key()==Qt::Key\_D)

            {

                map\_scene->hero\_item->movie->setFileName("://image/hero\_run\_right.gif");

                dx+=mobile\_step;

            }

            if(it.key()==Qt::Key\_A){

                map\_scene->hero\_item->movie->setFileName("://image/hero\_run\_left.gif");

                dx-=mobile\_step;

            }

        }

    }

    if(!flag)

    {

        for (QMap<int, bool>::iterator it = keymap.begin(); it != keymap.end(); ++it)

        {

            if(!it.value()){

                if(lastkey==Qt::Key\_D){

                    map\_scene->hero\_item->movie->setFileName("://image/hero\_stand\_right.gif");

                }

                else if(lastkey==Qt::Key\_A){

                    map\_scene->hero\_item->movie->setFileName("://image/hero\_stand\_left.gif");

                }

            }

        }

    }

    //边界检测

    qreal hero\_speed = map\_scene -> hero\_item -> gain\_move\_speed();

    qreal width = map\_scene->hero\_item->boundingRect().width();

    qreal height = map\_scene ->hero\_item->boundingRect().height();

    if(map\_scene->hero\_item->x()-width/2+dx>=map\_left &&

        map\_scene->hero\_item->y()-height/2+dy>=map\_top &&

        map\_scene->hero\_item->x()+width/2+dx<=map\_right &&

        map\_scene->hero\_item->y()+height/2+dy<=map\_underneath)

    {

        map\_scene->hero\_item->setPos(map\_scene->hero\_item->x()+dx\*hero\_speed,map\_scene->hero\_item->y()+dy\*hero\_speed);

    }

    //人物与障碍物碰撞检测

    for(int i=0;i<5;++i){

        if(map\_scene->hero\_item->collidesWithItem(obstacles[i]))

        {

            map\_scene->hero\_item->setPos(map\_scene->hero\_item->x()-dx\*hero\_speed,map\_scene->hero\_item->y()-dy\*hero\_speed);

           // qDebug()<<"test";

        }

    }

    map\_scene->hero\_item->movie->start();

    dx=0;

    dy=0;

}

主要利用键盘事件KeyPressEvent,KeyRealseEvent修改QMap<int,bool>（int型为QT中鼠标键值）实现了长按移动，记录last\_key用于更新贴图，（如按下D后释放D，再长按W，可以实现人物朝右并向上移动），实现了人物八个方向的移动。最后边界检测，防止人物抛出地图，碰撞检测，防止人物通过障碍物。

### （2）敌人自动寻路

//敌人自动寻路

void game\_engine::enemy\_update()

{

        for (QList<enemy\_base\*>::iterator it1 = enemy\_list.begin(); it1 != enemy\_list.end(); ++it1) {

            bool flag=true;

            qreal angle=(\*it1)->move\_angle(map\_scene->hero\_item->scenePos());

            for(QList<obstacle\*>::iterator it2=obstacles.begin();it2 != obstacles.end();++it2){

                if(enemy\_hit\_obstacle\_check(\*it1,\*it2,angle)){

                    flag=false;

                    QList<QPair<qreal,qreal>>list;

                    //dfs回溯寻找方向

                    for(int i=0;i<4;++i){

                        (\*it1)->enemy\_move(M\_PI/2\*i);

                        qreal distance = gain\_points\_distance((\*it1)->scenePos(),map\_scene->hero\_item->scenePos());

                        if(!(\*it1)->collidesWithItem(\*it2)){

                            list.push\_back({M\_PI/2\*i,distance});

                        }

                        (\*it1)->enemy\_move(M\_PI/2\*i-M\_PI);

                    }

                    //取distance最短的路径

                    std::sort(list.begin(),list.end(),[](QPair<qreal,qreal> &a,QPair<qreal,qreal> &b){

                        return a.second<=b.second;

                    });

                    (\*it1)->enemy\_move(list[0].first);

                    (\*it1)->distance\_hero =gain\_points\_distance((\*it1)->scenePos(),map\_scene->hero\_item->scenePos());

                    break;

                }

            }

            if(flag){

                (\*it1)->distance\_hero = gain\_points\_distance((\*it1)->scenePos(),map\_scene->hero\_item->scenePos());

                (\*it1)->enemy\_move(angle);

            }

        }

}

利用回溯法判断方向，如果前进方向没有障碍物就前进，如果有障碍物进入四个方向的dfs搜索判断（遇到障碍物只可能往上下左右四个方向移动），同样利用回溯法，如果这个方向前进和hero距离缩短最大，就往这个方向前进。

Bug:当人物和敌人的两点直线和障碍物边界矩形垂直时，敌人会被障碍物卡住不动；改进方向：可以利用搜图算法，如A\*算法。

### （3）子弹的移动和发射

在bullet\_base类中定义计时器控制了子弹的移动和子弹碰撞与边界检查，如果子弹和敌人发生碰撞就调用enemy->gain\_hurt(),敌人受到伤害。子弹移动基于坐标系，利用三角函数计算移动距离和方向。

bullet\_base::bullet\_base(QPointF pos\_, qreal angle\_, weapon \*weapon\_, QGraphicsScene \*scene\_, QObject \*parent)

    : QObject{parent},bullet\_point(pos\_),bullet\_angle(angle\_),hero\_weapon(weapon\_)

{

    bullet\_damage = hero\_weapon ->gain\_damage();

    bullet\_speed =hero\_weapon -> gain\_bullet\_speed();

    scene\_->addItem(this);

    setPos(bullet\_point.x(),bullet\_point.y());

    qreal angle\_degrees = qRadiansToDegrees(angle\_);

    setRotation(angle\_degrees);

    setScale(4);

    bullet\_timer =new QTimer(this);

    bullet\_timer->start(18/bullet\_speed);

    connect(bullet\_timer,&QTimer::timeout,this,&bullet\_base::bullet\_move);

    connect(bullet\_timer,&QTimer::timeout,this,&bullet\_base::hit\_enemy\_check);

}

//子弹碰撞和边界检查

void bullet\_base::hit\_enemy\_check()

{

    QList<QGraphicsItem\*> colliding\_items = collidingItems();

    for(QGraphicsItem\* item : colliding\_items) {

        enemy\_base\* enemy = dynamic\_cast<enemy\_base\*>(item);

        if(enemy != nullptr) {

            deleteLater();

            enemy->gain\_hurt(bullet\_damage);

        }

        }

    if (x() < 0 || x() > 4800 || y() < 0 || y() > 2700)

        deleteLater();

}

//子弹移动函数

void bullet\_base::bullet\_move()

{

    qreal angle =M\_PI/2-bullet\_angle;

    bullet\_speed=bullet\_gun\_speed;

    qreal distance=bullet\_speed;

    qreal dx=distance\*qSin(angle);

    qreal dy=distance\*qCos(angle);

    setPos(x()+dx,y()+dy);

}

在game\_engine中定义gun\_fire，并与计时器绑定，每次搜索和hero距离最小的敌人，并向其发射子弹，同时调用move\_gun()，转动枪支。

void game\_engine::gun\_fire()

{

        if(!enemy\_list.empty())

        {

            enemy\_base\* target\_enemy = enemy\_list[0];

        for(auto enemy:enemy\_list){

                if(enemy->distance\_hero<target\_enemy->distance\_hero){

                target\_enemy = enemy;

            }

        }

        qreal angle = gain\_angle(map\_scene->hero\_item->scenePos(),target\_enemy->scenePos());

        if(target\_enemy->distance\_hero<=attack\_extent){

            bullet\_gun \*bullet=new bullet\_gun(map\_scene->hero\_item->scenePos(),angle,gun,map\_scene);

            gun->gain\_angle(target\_enemy->scenePos());

            gun->move\_gun();

        }

        }

}

敌人子弹的生成同理，但调整了了射速和子弹移动速度。

### （4）敌人的随机生成

QList<QPointF> enemy\_generate\_poses={

    QPointF(map\_left,map\_top),

    QPointF(map\_left,map\_underneath),

    QPointF(map\_right,map\_top),

    QPointF(map\_right,map\_underneath),

    QPointF(map\_right-1500,map\_top),

    QPointF(map\_right-3000,map\_top),

    QPointF(map\_right-1500,map\_underneath),

    QPointF(map\_right-3000,map\_underneath),

};

//敌人生成器

void game\_engine::enemy\_generate()

{

    int i = QRandomGenerator::global()->bounded(8);//随机出生点

    int x = QRandomGenerator::global()->bounded(3);//随机出生敌人

    if(enemy\_num<=enemy\_num\_top){

        switch (x) {

        case 0:

            enemy\_list.push\_back(new enemy\_1(enemy\_generate\_poses[i],18,map\_scene));

            break;

        case 1:

            enemy\_list.push\_back(new enemy\_2(enemy\_generate\_poses[i],20,map\_scene));

            break;

        case 2:

            enemy\_list.push\_back(new enemy\_3(enemy\_generate\_poses[i],16,map\_scene));

            break;

        default:

            break;

        }

        qreal dx = map\_scene->hero\_item->x()-enemy\_list.last()->x(),

            dy =map\_scene->hero\_item->y()-enemy\_list.last()->y();

        enemy\_list.last()->distance\_hero =sqrt(pow(dx,2)+pow(dy,2));

        ++enemy\_num;

    }

}

预先定义几个出生点，并利用随机数生成，确定每一个敌人的出生位置和敌人种类，保存在Qlist<enemy\_base>中。

QList<QString> level\_up\_button\_normal ={

    "://image/icons/upgrades/u\_hollowpoint\_idle.png",

    "://image/icons/upgrades/u\_streamline\_idle.png",

    "://image/icons/upgrades/u\_inthewind\_idle.png",

    "://image/icons/upgrades/u\_heavybarrel\_idle.png",

    "://image/icons/upgrades/u\_vitality\_idle.png"

};

QList<QString> level\_up\_button\_hovered ={

    "://image/icons/upgrades/u\_hollowpoint\_hover.png",

    "://image/icons/upgrades/u\_streamline\_hover.png",

    "://image/icons/upgrades/u\_inthewind\_hover.png",

    "://image/icons/upgrades/u\_heavybarrel\_hover.png",

    "://image/icons/upgrades/u\_vitality\_hover.png"

};

QList<QString> level\_up\_introduce ={

    "子弹速度提高35%",

    "射速提高30%",

    "人物移动速度提高10%",

    "伤害提高50%",

    "生命值上限提高1"

};

void game\_engine::up\_select()

{

    timer\_stop();

    QList<int> numbers;

    while (numbers.size() < 3) {

        int number = QRandomGenerator::global()->bounded(5);

        if (!numbers.contains(number)) {

            numbers.append(number);

        }

    }

    for(int i=0;i<3;++i){

        QPixmap pixmap\_normal = level\_up\_button\_normal[numbers[i]];

        pixmap\_normal.scaled(up\_button\_width,up\_button\_height,Qt::IgnoreAspectRatio);

        up\_buttons[i]->setStyleSheet(QString("QPushButton{border-image:url(%1);}"//用样式表设置按钮形式，实现鼠标悬浮效果

                                       "QPushButton:hover{border-image:url(%2);}")

                                         .arg(level\_up\_button\_normal[numbers[i]])

                                         .arg(level\_up\_button\_hovered[numbers[i]])

                               );

        up\_buttons[i] -> show();

        up\_introductions[i] -> setText(level\_up\_introduce[numbers[i]]);

        up\_introductions[i] ->adjustSize();

        up\_introductions[i] ->move(up\_buttons[i]->pos().x()+up\_button\_width/2-up\_introductions[i]->width()/2,

                                  up\_buttons[i]->pos().y()+up\_button\_height+8);

        up\_introductions[i] ->show();

        connect(up\_buttons[i],&QPushButton::pressed,[=](){

            up\_function\_select(numbers[i]);

            emit up\_selection\_make();

        });

        }

}

//升级选择处理

void game\_engine::up\_function\_select(int num)

{

    switch (num) {

    case 0:

        gun->up\_bullet\_speed();

        break;

    case 1:

        up\_bullet\_generate\_speed();

        break;

    case 2:

        map\_scene -> hero\_item ->up\_move\_speed();

        break;

    case 3:

        gun -> up\_damage();

        break;

    case 4:

        up\_health();

        break;

    default:

        break;

    }

}

//升级结束处理

void game\_engine::up\_hide()

{

    for(int i=0;i<3;++i){

        up\_buttons[i]-> hide();

        up\_introductions[i] -> hide();

    }

    for(auto button:up\_buttons){

        button->disconnect();

    }

    timer\_start();

}

预先定义好每一个按钮样式路径，以及解释词，在升级时显示，利用随机数生成确定每次三个升级，利用QPushButton的pressed以及信号和lambda表达式实现升级确定。样式表实现当鼠标悬停在按钮上时的效果和离开时的变化。

### （5）结束画面（以失败为例）

void game\_engine::hero\_gain\_hurt()

{

    if(health\_now >= 0){//防止越界访问

    for(auto enemy: enemy\_list){

        if(enemy->collidesWithItem(map\_scene->hero\_item)){

            lose\_heart();

        }

    }

    if(health\_now <= 0)hero\_die();

    }

}

void game\_engine::hero\_die()

{

    timer\_stop();

    delete\_item();

    loser\_end -> show();

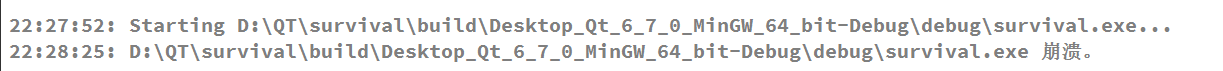
    setStyleSheet("background-color: rgb(34,34,34);");

}

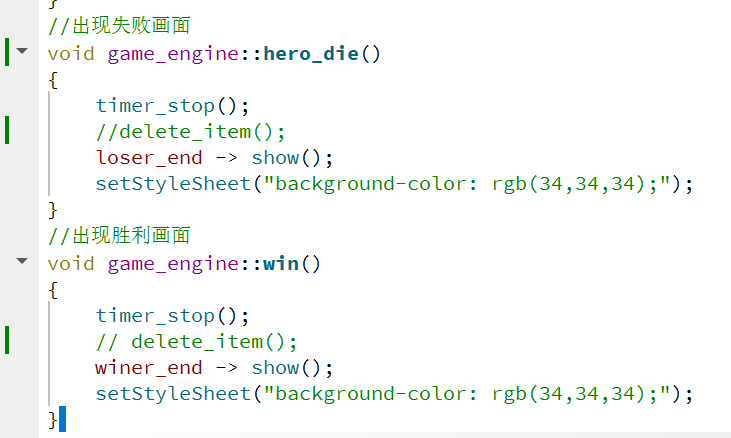
在每次受伤时加入结束判断，如果生命值为0，就显示失败画面。

胜利同理，在每次倒计时加入结束判断，如果倒计时为0，就显示胜利判断。

# 五、单元测试



游戏结束时偶尔会发生崩溃，可能是发生了内存泄漏

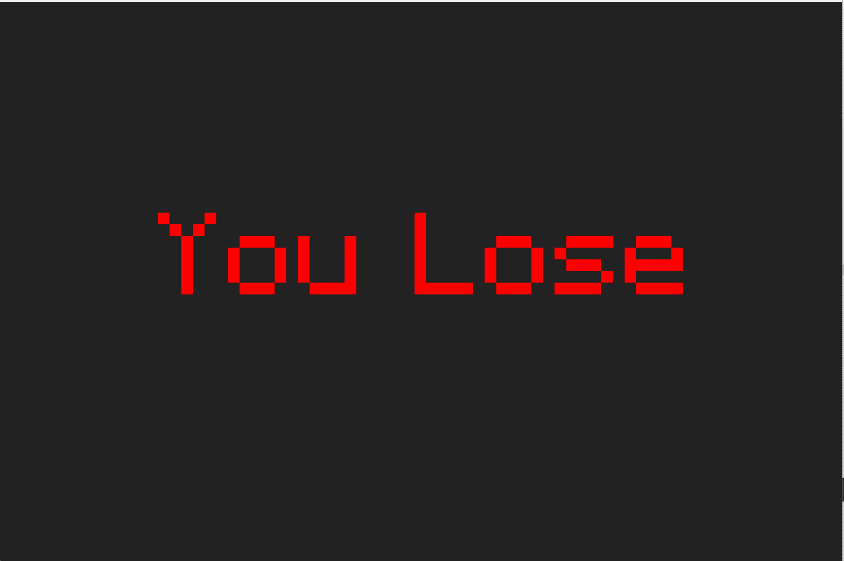


暂时以注释释放内存的内容为解决方案。



升级系统正常





能够正常结束

测试结果：

非常需要优化内存管理。游戏整体继承架构稍有问题。

# 六、收获

在各种代码实现中，强化了对于简单算法和容器的使用，如链表和映射。特别的，提高了对面向对象编程的认识，对于动态联编的纯虚函数、抽象基类有了更进一步的认识。项目的架构中，深化了对于继承的理解。

同时，对于内存泄漏的致命bug有了更深的理解，应该在内存管理上下更大功夫。