Qt 大作业报告

87-摆摆巴士组： 刘昕垚 邱显童 石清扬

一、程序功能介绍

我们组的 qt 项目是用面向对象等思想复刻经典益智类小游戏“黄金矿工”，并以“还原”为核心，尽量还原原游戏的逻辑与玩法。

点开游戏，进入界面后，点击开始即可开始游戏进入第一关；在每一关内按下下键放下钩子，碰到物品便提起物品并计算钱数，若没有碰到物品，碰壁后则立刻缩回。倒计时 60s，60s 到了之后若达到目标钱数则进入商店界面，选择相应想购买的物品并扣除相应钱数，并进入下一关。若没达到目标钱数则宣布游戏失败，可以选择退出游戏或者重新开始游戏。重新开始则从开始界面从头开始游戏。

在每一关里，若点击“退出本关”，会根据是否到达目标钱数来判断是进入商店界面进入下一关还是宣布游戏失败。

我们的游戏每一关所包含的物品种类与数量不同，坐标位置也不一样。不同大小，不同种类的物品的重量和价值也相应不同，猪会左右移动，碰到 TNT 会将周围物品炸掉。这些细节讲解会在下一个版块呈现。

在商店里可以购买不同的道具，分别有炸药，生力水，幸运草，石头书和优质钻石。炸药可以将抓起来的东西在上升过程中炸掉；生力水使抓物品速度更快；幸运草让福袋里得到更好的东西；石头

收藏书让石头价值乘三倍；优质钻石让钻石价值增加 200。除开炸药可累计购买，其他道具的效力都只维持一关。不同道具的编写细节讲解也会在下一个版块呈现。

总体来说，我们组用 qt 尽力原样复现了黄金矿工这一款游戏。接下来将从各个版块细节叙述我们的实现逻辑。

二、项目各模块与类设计细节

1. 物品相关的实现

(1) 物品类的实现

在物品类里，我们运用了继承的思想方便后续检测碰撞等。不同 grade 对应不同的大小和价值重量，而 rect 作为 qt 自带的长方形类方便我们进行贴图以及碰撞检测。

```
class _Object {
public:
    string name;
    int grade;
    int size;//离中心点上下左右的距离
    int value;
    double lift_speed;
    int x_position;
    int y_position;
    int ex_speed;
    int num;

    int ex_x;
    int ex_y;
    bool caugh=0;
    int pig_speed=1;
    int boom=0;

    QRectF rect;
    QRectF rect_up;
    QRectF rect_r1, rect_r2, rect_r3, rect_l1, rect_l2, rect_l3;

    QRectF rect_boom;

    _Object();
    _Object(int _grade, int _x, int _y);

    virtual ~_Object();
};
```

```
class Gold : public _Object{
public:
    string name = "gold";
    int grade; // 分为1,2,3,4四种等级大小
    int x_position;
    int y_position;

    Gold();
    Gold(int _grade, int _x, int _y);

    ~Gold();
};
```

(2) 特殊物品的实现

① 猪

我们进行了三帧的动画实现，轮流播放猪的三个贴图，向左向右各对应三张。同时，我们将猪的初始位置存下，若超过范围则折返（速度变为负）。

同时，我们保留了贴图为透明的原始 rect 跟随猪一起动，以方便进行碰撞检测。

```
for(std::vector<_Object>::iterator itt=v.begin();itt!=v.end();itt++){
    if((itt->name=="pig" || itt->name=="d_pig") && itt->caugh==0) {
        if(itt->x_position>itt->ex_x+300 || itt->x_position<itt->ex_x) {
            itt->pig_speed=-itt->pig_speed;
            if(itt->pig_speed>0)
                itt->x_position=itt->ex_x;
            else
                itt->x_position=itt->ex_x+300;
        }

        itt->x_position+=itt->pig_speed;
        itt->rect.setRect(itt->x_position,itt->ex_y,2*itt->size,2*itt->size);
        paint_pig(itt);
    }
}
```

```
void paint_pig(std::vector<_Object>::iterator itt) {
    if(itt->pig_speed>0) {
        if((itt->x_position-itt->ex_x)%30<=10) {
            itt->rect_r1.setRect(itt->x_position,itt->y_position,2*itt->size,2*itt->size);
            itt->rect_r2.setRect(0,0,0,0);
            itt->rect_r3.setRect(0,0,0,0);
            itt->rect_l1.setRect(0,0,0,0);
            itt->rect_l2.setRect(0,0,0,0);
            itt->rect_l3.setRect(0,0,0,0);
        }
        else if((itt->x_position-itt->ex_x)%30>=20) { ...}
        else { ...}
    }
    else { ...}
}
```

② **TNT**

由于 TNT 为范围爆炸效果，因此我们用检测以 TNT 为中心的一个大矩形和物品矩形的碰撞来判断消除哪些物品。需要消除的物品成员含量 boom 变为 1，方便后续判断以进行动画编写。

```
if(itt->name=="TNT") {
    flag1=1;
    flag2=0;
    flag3=0;
    for(std::vector<_Object>::iterator it=v.begin();it!=v.end();it++) {
        if(it->rect.intersects(QRect(itt->rect.x()-150,itt->rect.y()-150,2*(itt->size+150),2*(itt->size+150)))) {
            it->boom=1;
            it->caugh=1;
        }
    }
    break;
}
```

在爆炸动画编写中，我们以一个 expand 的速度均匀改变矩形的长和宽以实现爆炸由小到大的效果。在到达一个临界值时消失，并恢复各种初始化。

```
//爆炸动画
for(std::vector<_Object>::iterator itt=v.begin();itt!=v.end();itt++){
    if(itt->boom) {
        if(itt->name=="TNT")
            expand=10;
        else
            expand=8;

        boom_x+=expand;
        boom_y+=expand;
        itt->rect_boom.setRect(itt->x_position-boom_x/2+itt->size,itt->y_position-boom_y/2+itt->size,boom_x,boom_y);
        itt->rect_up.setRect(0,0,0,0);
        itt->rect.setRect(0,0,0,0);
        itt->caugh=1;

        if(itt->name=="TNT") { ...}
        else {
            if(boom_x>100*itt->size/20) {
                itt->boom=0;
                itt->rect_boom.setRect(0,0,0,0);
                boom_x=80;
                boom_y=80;
            }
        }
    }
}
```

(3) 物品地图绘制

我们运用了父类的动态数组（vector）存放不同大小，位置的物品，并在存入 vector 时提前初始化完毕。接下来就运用 Qrectf 的自带贴图方式用初始化的数据进行贴图。若要改变物品的位置和大小，只需运用 Qrectf.setRect(x,y,width,height)函数即可改变。

在不同关卡中，地图的贴图不同，因此要对物品位置进行初始化。为了让 mainwindow.cpp 尽量简

洁，我们将地图绘制存入函数，放进 level.h 和 level.cpp 中，在关卡变化时只用调用不同函数即可。

```
Gold g1(1,700,490);
Gold g2(2,400,550);
Gold g3(3,100,500);
Gold g4(3,800,600);
Gold g5(4,250,550);
Stone s1(1,600,450);
Stone s2(2,200,300);
Stone s3(2,200,300);
Diomond d1(850,300);
Diomond d2(900,300);
Lucky_Bag l1(300,400);
Lucky_Bag l2(500,300);
Pig p1(500,200);
D_Pig dp1(100,200);
TNT tnt1(300,300);
vector<_Object> v={g1,g2,g3,g4,g5,s1,s2,s3,d1,d2,l1,l2,p1,dp1,tnt1};
```

```
for (std::vector<_Object>::iterator it = v.begin(); it != v.end(); it++) {
    QPixmap pixmap1;
    if(it->name=="gold"){
        QPixmap pixmap(":/picture/gold4.png");
        pixmap1 = pixmap;
    }
    else if(it->name=="stone"){ ...}
    else if(it->name=="lucky_bag"){ ...}
    else if(it->name=="diomond"){ ...}
    else if(it->name=="pig" || it->name=="d_pig") { ...}
    else if(it->name=="TNT") { ...}

    painter.drawPixmap(
        it->rect.x(),     // x 坐标
        it->rect.y(),     // y 坐标
        it->rect.width(), // 宽度
        it->rect.height(),// 高度
        pixmap1           // 贴图对象
        );
```

2. 道具相关的实现

(1) 商店界面

商店界面道具的贴图我们使用了 ui 的按钮（button）和标签（label），按钮进行道具的贴图，标签进行价格的展示。

stoneButton	QPushButton
stonelabel1	QLabel
stonelabel2	QLabel
strongButton	QPushButton
stronglabel1	QLabel
stronglabel2	QLabel

在算法部分，我们使用了 5 个 flag 来判断道具是否出现，初始化为随机数对 2 求余，来实现“随机出现”的效果。在购买后也设置为 0，即在此后不会再出现。

同时，我们使用了 5 个名为 ed 的 flag 判断是否买过道具，买过几个（炸药可叠加），以进行后续特效的叠加。

```
int f1=rand()%2,f2=rand()%2,f3=rand()%2,f4=rand()%2,f5=rand()%2;//是否出现
int ed=0,ed1=0,ed2=0,ed3=0,ed4=0,ed5=0;//是否买过（12345分别对应摆放顺序，ed1表示买过几个，用过要--）
int pr1=rand()%200, pr2=rand()%200, pr3=rand()%200, pr4=rand()%200, pr5=rand()%200; //价格
```

为了简化 mainwindow.cpp 的代码长度，我们将一些绘制函数写在 goods.cpp 中，并进行相应的调用。

(2) 道具效果（单关）

通过相应的 ed 变量来判断是否买过，若买过进行相应的调参就好。需要注意的是，我们不能改变类的原始值，也要记住在每一关结束后全部初始化为 0，避免出现效力不只有一关的情况。

```
hook_speed=itt->lift_speed;
value_tem=itt->value;
if(ed2!=0){
    hook_speed=4.5;
}
size_tem=2 * itt->size;
if(ed3==1&&itt->name=="lucky bag"){
    value_tem=rand()%500+300;
}
if(ed4!=0&&itt->name=="stone"){
    value_tem*=3;
}
if(ed5!=0&&(itt->name=="diomond" ||itt->name=="d_pig")){
    value_tem+=200;
}
```

(3) 炸药效果

由于炸药效果特殊，则单独写。在抓起物品后按下上键，若有炸药则将物品炸掉，恢复为直接收起钩子的 flag 组合。爆炸动画效果与 TNT 的爆炸效果相同，即检测 boom 是否为 1。

```
if (pressedKeys.contains(Qt::Key_Up)) {
    if(ed1&&(!isSwinging)&&flag3) {
        b_flag=1;
    }
}
```

```
if(b_flag){
    player8->play();
    b_flag=0;
    ed1--;
    flag1=1;
    flag2=0;
    flag3=0;
    *boom_tem=1;
}
```

炸药还需要贴图，因此我们使用了 label 进行相应的贴图（最多五个）。为了让贴图随所持炸药数量变化，我们运用了两个循环，分别进行炸药的贴图和透明底的贴图。

3. 物理相关的实现

(1) 碰撞检测

由于钩子和物品我们都利用了 Qrectf 进行初始化，再在其中贴图，因此我们运用了 Qrectf 自带的长方形相交检测来进行碰撞检测；又因为图形不规整，我们进行了一些调参让碰撞更真实。

同时，因为我们的物品是存储在动态数组里的，所以我们可以反复遍历数组来探测是否发生碰撞。若发生碰撞则进行钩子伸缩，同时将贴图改为钩子伸缩时物品上带钩子的贴图。

```
// 碰撞检测，返回true即为成功
bool MainWindow::collision_detection(QRectF rect){
    int x = hookEnd.x() - 8 * cos(angle);
    int y = hookEnd.y() - 5 * sin(angle);
    if(rect.intersects(QRect(x,y,16,10)))
        return true;
    else
        return false;
}
```

```
for(std::vector<_Object>::iterator itt=v.begin();itt!=v.end();itt++){
    // 如果与其中任何一个物体发生了碰撞
    // flag2：是否碰撞
    // flag3：是否已经发生过碰撞
    if (collision_detection(itt->rect)&&flag3==0&&itt->caugh==0){
        switch(itt->num){ ...}

        if(itt->name=="TNT") { ...}

        hookRect.setRect(hookEnd.x() - 15, hookEnd.y(),0,0);
        itt->rect.setRect(0,0,0,0);
        clear_pig(itt);
        goldRect=&itt->rect_up;
        goldRect->setRect(hookEnd.x() - itt->size, hookEnd.y(), 1.9*itt->size, 1.9*itt->size);

        boom_tem=&itt->boom;
        ex_x_tem=&itt->x_position;
        ex_y_tem=&itt->y_position;

        flag2=1;
        flag3=1;
        hook_speed=itt->lift_speed;
        value_tem=itt->value;
```

(2) 钩子摆动与伸缩

我们在 paintEvent() 中使用的钩子起点和终点坐标 hookStart 和 hookEnd 是 QPoint 对象，运用相关计算更新钩子起点和终点的坐标，使钩子移动或摆动。

① 钩子摆动

```
if (swingDirection) {
    angle += angularSpeed; // 增加摆动角度
    if (angle >= 2.9) {
        swingDirection = false; // 到达最大角度后改变摆动方向
    }
} else {
    angle -= angularSpeed; // 减小摆动角度
    if (angle <= 0.28) {
        swingDirection = true; // 到达最小角度后改变摆动方向
    }
}
// 计算钩子起点和终点的坐标
c=sqrt((hookStart.x()-hookEnd.x())*(hookStart.x()-hookEnd.x())+(hookStart.y()-hookEnd.y())*(hookStart.y()-hookEnd.y()));
cos1=-(hookStart.x()-hookEnd.x())/c;
sin1=-(hookStart.y()-hookEnd.y())/c;
hookStart = QPointF(width() / 2, 80); // 圆心位置
hookEnd = hookStart + QPointF(radius * cos(angle), radius * sin(angle)); // 计算终点坐标
```

由于钩子头部是贴图，因此还涉及图片的旋转。我们先保存原画布，再以线尾为旋转中心旋转对应的角度，如此钩子和线相对位置就会不变，可以进行正常的长方形绘制与贴图。

```
// 绘制钩子头部
QPixmap pixmap(":/picture/hook.png");

//头部旋转
//保存绘制的状态
painter.save();
painter.translate(hookEnd.x(), hookEnd.y());
painter.rotate(angle/3.14*180 - 90);
painter.translate(-hookEnd.x(), -hookEnd.y());
//绘制在新的画布位置上
painter.drawPixmap(
    hookEnd.x() - 15, hookEnd.y(), hookRect.width(), hookRect.height(), pixmap
    );
```

② 钩子伸缩

以 angle 角度放下钩子

```
hookEnd = hookEnd + QPointF((float)hook_speed_down*cos(angle), (float)hook_speed_down*sin(angle));
```

若碰壁则以 angle 角并以空钩子速度回收钩子，并在判断钩子收回到原始位置时再次开始摆动。

```
if(hookEnd.x()>width()-25||hookEnd.x()<25||hookEnd.y()>691-30||flag1==1||b_flag==1){
    if(b_flag){ ...}
    if(flag5){ ...}

    hookRect.setRect(hookEnd.x() - 15, hookEnd.y(),30,30);
    flag1=1;
    hookEnd = hookEnd - QPointF((float)hook_speed_up*cos(angle), (float)hook_speed_up*sin(angle));

    // 钩子回收成功（空钩子）
    if(hookEnd.y()<(hookStart.y()+radius * sin(angle))){

        QPalette palette;
        QPixmap pixmap(":/picture/gamewholeback1.png");
        palette.setBrush(QPalette::Window, QBrush(pixmap));
        this->setPalette(palette);
        player4->stop();
        flag5=1;
        w=0;

        isSwinging=true;
        hookEnd = hookStart + QPointF(radius * cos(angle), radius * sin(angle));
        flag1=0;
        update();
    }
    else
        update();
}
```

若检测到物体碰撞则以 angle 角并以该物体的速度回收钩子，在判断钩子收回到原始位置时再次开始摆动，并计算钱数，同时消除物品。

```
isSwinging=true;
hookEnd = hookStart + QPointF(radius * cos(angle), radius * sin(angle));
flag2=0;
flag3=0;
goldRect->setRect(0,0,0,0);
hookRect.setRect(hookEnd.x() - 15, hookEnd.y(),30,30);
mo.money_num+=value_tem;
ui->mymoneylable2->setText(QString::number(mo.money_num));
update();
```

4. 时间相关的实现

在倒计时的实现方面，我们使用了 QTimer 来计时，即倒计时 60s，每 1000ms（1s）进行一次更新，实现倒计时的功能。在倒计时小于等于零，即倒计时结束后游戏界面结束，进入其他界面。

需要注意的是，在倒计时结束时，需要把游戏算法中判断状态的变量初始化，以避免再进入下一

关时出现动画乱动的问题。

```
remainingTime = 60;
ui->timelable->setText("60");
timer = new QTimer(this);
connect(timer, &QTimer::timeout, this, &MainWindow::updateTimer);
timer->start(1000);//倒计时定时器
```

```
void MainWindow::updateTimer()
{
    if (beginflag&&remainingTime > 0) {
        ui->timelable->setText(QString::number(remainingTime));
        remainingTime--;
    }
    else if(beginflag&&remainingTime <= 0){
        timer->stop();
        remainingTime = 60;

        isSwinging=true;
        flag1=0;
        flag2=0;
        flag3=0;
        flag4=1;//放下时是否更改过人物及播放音乐
        flag5=1;//收回时是否更改过人物及播放音乐
        w=0;
        r=0;

        ui->timelable->setText("60");
        if(judge()){ ...}
        else{ ...}
    }
}
```

除此之外，我们还用 QTimer 的计时器，每 10ms 计时更新图片来实现钩子的动画。

```
timer1 = new QTimer(this);
connect(timer1, &QTimer::timeout, this, &MainWindow::animateHook);
timer1->start(10); // 每10毫秒触发一次定时器，控制钩子动画速度
this->ui->stackedWidget->setCurrentIndex(2);
```

5. UI 的实现

(1) 游戏进程的实现

我们将各个不同界面分开写进了不同函数，再在每个函数中运用 QTimer::singleShot(int msec, const QObject *receiver, const char *member)来进行各个画面间的连接。

```
void MainWindow::time_update1(){ ...}

void MainWindow::time_update2(){ ...}

void MainWindow::time_update3(){ ...}

void MainWindow::time_update4(){ ...}

void MainWindow::time_update5(){this->ui->stackedWidget->setCurrentIndex(7);}

void MainWindow::on_beginButton_clicked() {...}

void MainWindow::on_quitButton_clicked() {...}

void MainWindow::on_nextButton_clicked() {...}

void MainWindow::on_replayButton_clicked() { ...}

void MainWindow::on_closeButton_clicked() { ...}
```

(2) UI 画面的实现

在画面实现部分，我们运用了大量 Qt 的库。

首先，我们从原游戏截图，再运用 PS 抠图将各个版块需要的图片用透明背景的格式保存下来以便后续 UI 使用；同时利用录屏为 gif 形式动图来实现大部分动画部分。

对于各个按钮部分，我们主要运用了 QPushButton 的 setStyleSheet 函数来改变按钮的形状，并利用其自带的 hover 与 clicked 来改变悬停以及点击时的按钮；同时利用 setFont、setStyleSheet、setText 来分别设置按钮的字体，颜色和内容。

```
//结束界面按钮
QFont font;
font.setPointSize(18); // 设置字体大小
font.setBold(true); // 设置字体加粗
ui->replayButton->setFont(font);
ui->replayButton->setStyleSheet("color: #000000;");
ui->replayButton->setText("重 玩");
ui->replayButton->setCursor(QCursor(Qt::PointingHandCursor));
ui->replayButton->setStyleSheet("QPushButton{border-image:url(:/picture/endbutton1.png);}"
                                "QPushButton:hover{border-image:url(:/picture/endbutton2.png);}"
                                );
```

动画部分运用 QMovie 导入 gif 文件，并利用 start 函数循环播放实现各种较复杂动画，例如光晕，人物眨眼，对话出现等。

```
QMovie *movie4 = new QMovie(":/picture/shopmanspeakgif.gif");
ui->shopmanlable->setMovie(movie4);
QMovie *movie5 = new QMovie(":/picture/shoptextgif.gif");
ui->shoptextlable->setMovie(movie5);
movie4->start();//人物说话
movie5->start();//打字动画
```

音乐部分运用 QMediaPlayer 和 QAudioOutput 分别设置音频文件路径和播放控制。若要循环播放则使用 QMediaPlayer.setLoops(-1)。

```
// 播放音乐
QMediaPlayer *player1= new QMediaPlayer(this);
QAudioOutput *audio1= new QAudioOutput(this);
player1->setAudioOutput(audio1);
QString filePath = "qrc:/music/break.mp3";
player1->setSource(QUrl(filePath));
player1->play();
```

三、小组成员分工情况

刘昕垚：物品相关的实现；动画的实现；封装整合调整；报告撰写视频录制

邱显童：UI 的实现（包括背景抠图与音乐）；游戏进程、时间的实现

石清扬：游戏主体算法，例如物理相关的实现，道具效果

四、项目总结与反思

这次 Qt 作业是我们大学生活第一次以小组形式完成一个较大型的 project。这学期的程设课前半学期学习了面向对象这一十分重要的思想，也运用到了 Qt 大作业中。Qt 脱胎于 C++，但是又有自己十分丰富的库函数以及不同的类，因此可以较方便地进行 ui 的书写。在完成大作业的过程中，我们不仅学习了 Qt 的使用方法，还巩固了许多面向对象的知识，可以说是收获颇丰。

虽然由于实力原因，我们组完成的项目并不是特别复杂，整体而言也并不是特别精美，但是能够完成

一个可以正常运行的游戏也让我们非常有成就感。在这次的合作过程中，我们学到了分工合作的重要性，以及对于 ddl 时间把控的重要性。从开始前的合理分工，到过程中大家及时的反馈交流以及代码的分享整合维护，都离不开一个小组三个人共同的努力。尽量给其他人留下足够的时间，如此合作才可以更加快乐与顺利。

同时，在这次合作完成项目的过程中，我们也意识到了代码规范的重要性。清晰规范的命名，以及对相关变量，类，较复杂函数的注释可以很大地减少工作量，可以让代码的维护整合更加轻松。因此，在团队完成较大型项目的过程中，规范性命名也是十分重要的。

同时，我（刘昕垚）作为后期 debug 整合的成员，更加深刻了解到不论在写什么部分都一定要注意初始化，否则会出现各种各样奇怪的 bug。

在最后的最后，感谢老师一学期风趣幽默的讲解与细致的教导，也感谢助教耐心的辅导，让我们学习到了知识，也收获了乐趣，完美结束了大一的学习生活！

2023 年 7 月 7 日