青岛大学
QINGDAO UNIVERSITY

《C 语言课程设计》报告

题目	单词贪吃蛇
班级	23 图灵
组长	王华召　2023204286
组员	苏奕成　2023204204 石贞豪　2023204278
指导教师	杜祥军

计算机科学技术学院

2024 年 6 月

目 录

详情概括

本学期C语言程序设计与基础算法课,我们小组选择的项目是贪吃蛇小游戏.我们组全程使用 QT 软件来完成这个项目，使用的语言是 c＋＋。这个项目的主要内容是，在传统贪吃蛇游戏的基础上，把要吃的目标（比如说苹果图案）修改成英文字母，这样的话，玩家每完整地吃完一个单词的所有字母，这个单词可以显示在下方的框中并得以记录，可以让玩家在体验游戏的过程中进行英语单词的学习。

成员分工:

·王华召：负责图像界面设计与美化，音效处理，随心吃模式的字典树和退出游戏

·石贞豪：前三个模式中关于单词的部分：单词的提取，单词分布及消失，蛇吃字母判断得分

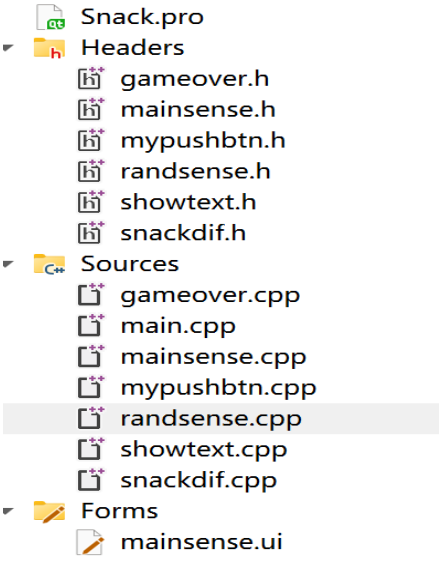
·苏奕成：贪吃蛇身体的初始化，蛇的变向和移动

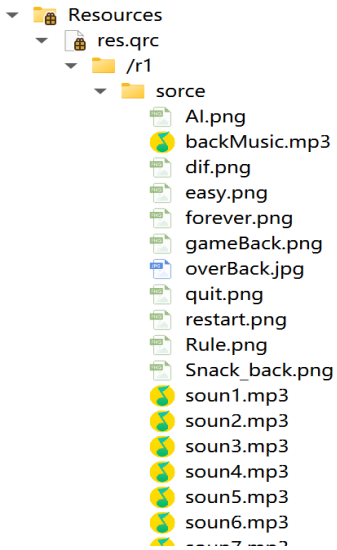
-2-

一．图像界面的设计

1.1 主界面

首先,我们的主文件夹名字是 snack(一开始创建时拼错了,写了一半才发现,就不想改了,其实应该是 snake),在主文件夹下,除去原有的 main 函数，我们一共添加了六个新的类，分别是 gameover，mainsense，mypushbtn，randsense，showtext 和 snackdif。然后把所需要使用的图片资源和音频资源存放在 resources 中新建的 res 文件里，副文件命名为 sorce。





我们的游戏共分为四个模式，分别是简单模式，困难模式，无尽模式和随心吃模式。在正式开始游戏主体功能实现之前，需要先给游戏制作图标和主界面，以及配背景音乐。

```
void Mainsense::paintEvent(QPaintEvent*)
{
    QPainter painter(this);
    QPixmap pix;
    pix.load(":/r1/sorce/Snack_back.png");
    painter.drawPixmap(0,0,this->width(),this->height(),pix);
}
```

这一段利用函数 paintEvent，可以在窗口上绘制一个图片，只需要先创建一个 QPainter 的对象，再利用 qt 自带的类 QPixmap 创建一个 pix 来继承想要放置为背景的图。QPainter

的对象调用函数 drawpixmap 可以实现对所放置图片的定位，函数中所带的五个参数的意思分别为图片起始位置的横纵坐标，横向延伸长度，纵向延伸长度，目标图片的参量名。

1.2 其余图标

然后为了激活四个模式的入口，我们需要设置四个小图标，类似于 ppt 中超链接的原理，在点击对应图标后可以让玩家自动传送至对应难度的游戏。而实现这个功能，还需要先自定义一个控制局内图标的函数，这里命名为 MyPushBtn，这是一个根据 qt 原有的 Qpushbutton 类自创的类，配合 Qmixmap 类，setIcon，setIconSize 等函数，就可以实现界面内小图标的放置，大小和形状的控制。

```
MyPushBtn::MyPushBtn(QString normalI,QString pressI)
{
    this->nPath=normalI;
    this->pPath=pressI;
    QPixmap pix;
    pix.load(nPath);
    this->setFixedSize(240,75);
    this->setStyleSheet("QPushButton{border:opx}");
    this->setIcon(pix);
    this->setIconSize(QSize(pix.width(),pix.height()));
}
```

于是对各个图标的设置和定位如下所示：

```
this->sound=new QSound(":/r1/sorce/soun1.mp3",this);
sound->setParent(this);
setFixedSize(1080,720);
setWindowTitle("单词贪吃蛇");//设置主页面的标题
setWindowIcon(QIcon(":/r1/sorce/start.png"));//设置游戏的图标
btn=new MyPushBtn(":/r1/sorce/dif.png");
btn->move(80,590);//这是较难模式的开始按钮
btn2=new MyPushBtn(":/r1/sorce/easy.png");
btn3 =new MyPushBtn(":/r1/sorce/forever.png");
btn4=new MyPushBtn(":/r1/sorce/AI.png");
btn->setParent(this);
btn2->setParent(this);
btn3->setParent(this);
btn4->setParent(this);
btn2->move(320,590);
btn3->move(570,590);
btn4->move(830,590);
```

接下来就可以用 connect 函数实现点击对应图标传送至对应难度游戏的功能了，这也是 qt 自带的一个函数，其中包含的主要参数主要是信号发送者，信号本身以及所对应的接收

者，接收者的相应方式。这个函数通过将信号和槽连接起来，使得在事件发生时能够自动调用相应的处理函数，例如玩家点击鼠标，这样就能激活事件，触发后续处理方式，自动跳转至对应难度的游戏场景。其使用形式大致为

```
connect (btn,&QPushButton::clicked,[=](){
        this->sound->play();
```



```
this->hide();
        dif=new SnackDif(1);
        dif->show();
        dif->setFixedSize(1080,720);
    });
```

二．贪吃蛇的构造和移动

2.1 蛇的变向

插放背景音乐和特殊音效的功能稍后再议，这里已经视作完成了准备工作，接下来就要步入游戏的主要部分，实现贪吃蛇的移动和单词（即所吃目标）的分配。对于贪吃蛇的移动，众所周知，这是个二维游戏，所以蛇的移动也只有四个方向（命令禁止轮盘操作，所以没有斜向移动功能），分别是上下左右。这里可以设置四个按钮变量来对应之，即定义一个包含四个枚举值的枚举类型：

```
    enum Direct
{
  Dir_UP,
  Dir_Down,
  Dir_Left,
  Dir_Right,
};
```

然后再自定义一个 keyPressEvent 函数，把键盘上的上下左右四个键对应到蛇的上下左右方向的变化，根据按下的键来操作蛇的变向或者控制游戏的开始或暂停：

```
void SnackDif::keyPressEvent(QKeyEvent* event)
{
    switch (event->key()) {
    case Qt::Key_Up:
        if(moveFlag!=Dir_Down)
```

```
            moveFlag=Dir_UP;
        break;
    case Qt::Key_Down:
         if(moveFlag!=Dir_UP)
            moveFlag=Dir_Down;
        break;
```



```
case Qt::Key_Left:
         if(moveFlag!=Dir_Right)
           moveFlag=Dir_Left;
        break;
    case Qt::Key_Right:
         if(moveFlag!=Dir_Left)
           moveFlag=Dir_Right;
        break;
    case Qt::Key_Space:
        if(gameStart==false)
        {
            gameStart=true;
            //游戏开始之后定时器启动
            timer->start(300);//每三百毫秒就发出超时信号;
        }
        else
        {
            gameStart=false;
            timer->stop();
        }
        break;
    default:
        break;
    }
}
```

具体的移动要求是这样的：先定义一个 moveflag 变量来表示当前蛇移动的方向，当我们按了键盘上的上箭头时，如果 moveflag 为向下，那么蛇就不能立即执行 Dir_up,只有在当前移动方向不为向下时，key_up 才能发挥作用，否则就是一个无效命令，无法激活对应事件的执行。其他三个方向的移动键也是如此，如果当前移动方向和你按下的键相反，那么就不能执行移动命令（原理就是贪吃蛇不能瞬间调转身子，玩过游戏的应该都知道蛇只能转弯来实现调头功能）。

2.2 定时器

说完四个方向键，再说一下这段代码里面关于暂停功能的部分。为了控制游戏开始或者结束，我们创造了一个布尔变量 gamestart，他的值只有 true 和 false 两种，可以看做是两种开关，true 对应游戏开始，false 对应游戏结束。根据个人习惯，我们把 gameStart 开关的控制权交给空格键，当按下空格时，如果游戏未开始，则将 gameStart 设置为 true，启动定时器；如果游戏已经开始，则将 gameStart 设置为 false，停止定时器。

这里再插入一条关于定时器的注释，定时器 timer 是自设的属于 qt 自带类 Qtimer 的一个对象。Qtimer 类提供了一种简单而有效的方式来处理时间相关的事件，它允许开发者在指定的时间间隔后执行特定的操作。而这个功能就是贪吃蛇移动的关键，前面用

的四个枚举变量只能控制蛇方向的变换，如何让蛇动起来就需要定时器了。利用 start 加参数的方式启动定时器，参数即为执行特定操作的时间间隔，这里设置为 300 毫秒，再利用 connect 将 timeout 信号连接到自定义的函数 snackmove（蛇的移动函数），该函数就会在定时器超时（自计时起达到参数 300 毫秒后）时被调用，至于这个移动函数，我们放在后面讲。

```
connect(timer,&QTimer::timeout,[=](){
        this->snackMove();
```

2.3 蛇身体的初始化

蛇在移动起来之前，先要进行初始化，代码实现比较简单：

```
QRectF rect(300,180,nodeW,nodeH);
    snackBody.append(rect);
    addTop();
    addTop();
```

这段代码创建了一个 QRectF 对象 rect，并将其添加到 snackBody 列表中。其中，QRectF 是一个表示矩形的类，它的构造函数需要四个参数：左上角的 x 坐标、左上角的 y 坐标、矩形的宽度和高度。在这里，rect 的左上角坐标为(300,180)，宽度为 nodeW，高度为 nodeH。其初始化工作在两次调用 addtop 的时候实现。这里我们把蛇的身体设置为一个一个小方块，初始时蛇拥有三个小方块，当蛇每多吃到一个字母，蛇的身体就会变长，即增加一个方块。

2.4 蛇的移动

在展示单独的 addtop 函数之前，我们先看一下 snackmove，这是一个总函数，相当于控制移动的总开关，底下分为四条分支，分别是 addtop，addleft，addright 和 adddown。游戏过程中根据玩家的转向命令分别执行这四条函数中的一个。下面是 snackmove 函数的代码：

```
void SnackDif::snackMove()
```

```
{
    int cnt=wordHide()+1;
    while(cnt--)
    {
        switch (moveFlag)
        {
           case Dir_UP:
              addTop();
              break;
```



```
case Dir_Down:
              addDown();
              break;
          case Dir_Left:
              addLeft();
              break;
          case Dir_Right:
              addRight();
              break;
       }
    }
    this->snackBody.removeLast();
    update();//每一次处理好这个链表之后就要进行重新画蛇的身体
}
```

这里先不解释 cnt 和函数 wordhide，只需知道这是执行移动命令的先决判断条件。对于蛇的移动，只需要使用一个 Switch 函数，将枚举变量所代表的四个方向和四个函数一一对应，就能激活各个方向蛇的移动（本质上是小方块的位移）。最后奉上最重要的 add 代码，以 addtop 为例：

```
void SnackDif::addTop()
{
    QPointF l(snackBody[0].x(),snackBody[0].y()-nodeH);
    QPointF r(snackBody[0].topRight());
    if(l.y()<=-15)
    {
        if(this->model==3)
        {
            QPointF l1(l.x(),this->height()-nodeH);
            QPointF r2(r.x(),this->height());
            l=l1;
            r=r2;
        }
        else
```

```
        {
            gameStart=false;
            this->timer->stop();
            this->gameover->show();
            return ;
        }
    }
    snackBody.insert(0,QRectF(l,r));
}
```

这段代码通过以下步骤在蛇的身体顶部添加一个新的矩形块：

首先，它创建了两个 QPointF 对象 l 和 r，分别表示新矩形块的左上角和右上角坐标。这些坐标是通过获取蛇身体的第一个矩形块的左上角坐标，并将其 y 坐标减去 nodeH（节点高度）来计算的。同时，它还获取了第一个矩形块的右上角坐标作为新矩形块的右上角坐标。然后，判断 l.y()是否小于等于-15，如果是，则执行以下操作：如果 this->model 等于 3，那么将 l 和 r 的值更新为新的坐标，使蛇身体在边界内移动。这里，它将 l 的 y 坐标设置为 this->height()-nodeH，并将 r 的 y 坐标设置为 this->height()。否则，设置 gameStart 为 false，停止计时器，并显示游戏结束界面。这意味着蛇已经撞到了边界或者自己，游戏结束。最后，将新的矩形块插入到 snackBody 列表的开头。这是通过调用 snackBody.insert(0,QRectF(l,r))实现的，其中 0 表示插入的位置，QRectF(l,r)表示新创建的矩形块。

其余三个方向的移动函数与 addtop 异曲同工，差异在对左右坐标的更新方法上，这里为了节省篇幅不再一一赘述，只附上代码：

Adddown:

```
void SnackDif::addDown()
{
    QPointF l(snackBody[0].bottomLeft());
    QPointF
    r(snackBody[0].bottomRight().x(),snackBody[0].bottomRight().y()+n
odeH);
    if(r.y()>=this->height()+15)
    {
        if(this->model==3)
        {
            QPointF l1(l.x(),0);
            QPointF r2(r.x(),nodeH);
            l=l1;
            r=r2;
        }
        else
        {
            gameStart=false;
```

```
            this->timer->stop();
            this->gameover->show();
            return ;
        }
    }
    snackBody.insert(0,QRectF(l,r));
}
```


addLeft:

```
void SnackDif::addLeft()
{
    QPointF l(snackBody[0].x()-nodeW,snackBody[0].y());
    QPointF r(snackBody[0].bottomLeft());
    if(l.x()<=-15)
    {
        if(this->model==3)
        {
            QPointF l1(this->width()-nodeW,l.y());
            QPointF r2(this->width(),r.y());
            l=l1;
            r=r2;
        }
        else
        {
            gameStart=false;
            this->timer->stop();
            this->gameover->show();
            return ;
        }
    }
    snackBody.insert(0,QRectF(l,r));
}
```

Addright:

```
void SnackDif::addRight()
{
    QPointF l(snackBody[0].topRight());
```

```
    QPointF
r(snackBody[0].bottomRight().x()+nodeW,snackBody[0].bottomRight().y()
);
    if(r.x()>=this->width()+15)
    {
        if(this->model==3)
        {
            QPointF l1(0,l.y());
            QPointF r2(nodeH,r.y());
            l=l1;
            r=r2;
        }
```



```
 else
        {
            gameStart=false;
            this->timer->stop();
            this->gameover->show();
            return ;
        }
    }
    snackBody.insert(0,QRectF(l,r));
}
```

三．局内单词显示和布局

3.1 单词的导入和存储初始化

至此，有关蛇的移动的实现方式已经全部讲完了。接下来轮到另外一个主角登场了——贪吃蛇要吃的目标字母。根据传统贪吃蛇的规则，字母应该随机分布在游戏界面中。那么这些字母又是从何而来呢？既然要调用，那么必定有储存，说到储存，最简洁直接的方式就是用数组储存。为了方便后续的得分统计，我们设置了三个数组以及一个计数变量：

```
wordType  wordSum[1000];//这个是从文本文件中读取的（即全部单词）
wordType  eatWordSum[1000];//这个是游戏过程中吃对的
gameWordType  gameWordSum[1000];//这个是游戏过程中总的出现的
int appear=0;//已经出现的英文单词个数
```

在游戏的资源库里，我们提前植入了一个文本文件：

在这个文本文件中，事先导入一千个单词的英文拼写和对应中文翻译，接着利用 initwordsum 函数来初始化数组 wordsum 中的单词：


```
void SnackDif::initWordSum()
{
    QFile file(":/r1/sorce/word.txt");
    if(!file.open(QIODevice::ReadOnly|QIODevice::Text))
    {
        qDebug()<<"dakaishibai";
    }
    QTextStream in(&file);
    in.setCodec("UTF-8");
    QChar ch;
    int i=0;
    while(!in.atEnd())
    {
        in>>ch;
        in>>ch;
        while(ch!=' ')
        {
            wordSum[i].arr+=ch;
            in>>ch;
        }
        in>>ch;
        wordSum[i].brr=in.readLine();
        i++;
    }
    file.close();
}
```

先使用 QFile 类创建一个文件对象，打开名为"word.txt"的文件。检查文件是否成功打开，如果失败则输出"dakaishibai"。使用 QTextStream 类创建一个输入流对象，用于从文件中读取数据，设置输入流的编码为"UTF-8"。定义一个 QChar 类型的变量 ch，用于存储从文件中读取的字符。定义一个整型变量 i，用于记录当前处理的单词在 wordSum 数组中的下标。使用 while 循环遍历文件中的所有字符，直到文件结束。每次循环中，先读取两个字符，然后判断第二个字符是否为空格。如果不是空格，则将该字符添加到 wordSum[i].arr 中，并继续读取下一个字符。如果是空格，则跳出循环。读取下一个字符，将其赋值给 ch。使用 in.readLine()方法读取当前行剩余的内容，并将其赋值给 wordSum[i].brr，索引 i 加 1，准备处理下一个单词。这里的 arr 和 brr 分别是存储英文单词和中文翻译的数组。

3.2 游戏内所需单词的添加方式

接下来是添加字母的函数 addword：

```
void SnackDif::addWord()
{
    int index=gameWordSum[appear].wIndex=qrand()%100;
    QString tmp=gameWordSum[appear].a=wordSum[index].arr;
    gameWordSum[appear].b=wordSum[index].brr;
    int& i=gameWordSum[appear].psize;
    for( i=0;i<tmp.size();i++)
    {
        QRectF rect(qrand()%(this->width()/20)*20+10,
                    qrand()%(this->height()/20)*20+10,
                    35,
                    35);
        gameWordSum[appear].point[i]=rect;
        gameWordSum[appear].fpoint[i]=false;//一开始单词的标志变量都为0
    }
    qDebug()<<tmp<<' '<<gameWordSum[appear].b<<' '<<tmp.size();
    QFont labelFont("Arial", 16);
    QFont labelFont2("Arial", 10);
    this->label33->setFont(labelFont2);
    this->label33->setText("");
```

```
        this->label33->setFont(labelFont2);
        this->label33->setText("当前的得分是:");
        this->labelInt->clear();
        this->labelInt->setFont(labelFont2);
        this->labelInt->setNum(getScore);
        this->labelInt->move(108,42);
        this->label33->show();
        qDebug()<<getScore;
        appear++;//可以在 appear 这里重新写
        if(showFlag)
        {
            QString word=gameWordSum[appear-1].a;
            QString mean=gameWordSum[appear-1].b;
            qDebug()<<mean<<'
'<<word<<"0000"<<gameWordSum[appear-1].a.size()<<eatNum;
            this->label11->setText("");
            this->label11->setText(word);
```



```
    this->label11->setFixedSize(170,65);
            this->label11->move(940,70);
            this->label11->setFont(labelFont);
            this->label22->setFixedSize(170,65);
            this->label22->setText("");
            this->label22->setText(mean);
            this->label22->move(940,100);
            this->label22->setFont(labelFont);
            this->label11->setAlignment(Qt::AlignHCenter);//居中显示
            this->label22->setAlignment(Qt::AlignHCenter);
        }
    }
```

```
    QLabel* label33=NULL;//显示:当前的得分是:
    QLabel* label11=NULL;//显示要吃的英语
    QLabel* label22=NULL;//显示要吃的汉语
    QLabel* labelInt=NULL;//显示数字:getscore
    QLabel* label55=NULL;//显示:已经吃的字母是:
    QLabel* label66=NULL;//显示:时刻显示吃到的字母是什么
```

这里 addword 函数和一部分统计得分的功能融合在了一起，先解释添加新单词的功能，第一句代码用来生成一个随机数作为索引，用到了 qrand 函数，这个函数是 Qt 库中的一个函数，用于生成一个随机数。它通常与 C++的%操作符结合使用，以生成一个指定范围内的随机数。例如，qrand() % 100 将生成一个 0 到 99 之间的随机数。然后从 wordSum 数组中获取一个

单词和对应的解释，并将它们存储在 gameWordSum[appear]结构体中，appear 是游戏中出现的单词数目。遍历单词的每个字符，生成一个随机的位置，并创建一个矩形对象，将其存储在 gameWordSum[appear].point 数组中。同时，将 gameWordSum[appear].fpoint 数组中的对应位置设置为 false，表示该字符尚未被选中。剩下的两段关于 Qlabel 类的对象的代码，主要是起到显示信息的作用，每个对象的含义放在上方图中，不再一一解释。

Addword 函数的调用规则是这样的：

```
if(this->eatNum==gameWordSum[appear-1].a.size())
        {
             this->judgeRight(gameWordSum[appear-1].a);//吃到指定数目
的单词之后
            //就开始调用新的添加单词
             addWord();
             this->eatNum=0;
        }
   });
```

这里面用到了一个新变量 eatnum 和判断函数 judgeright，eatnum 是指游戏过程中吃到的字母数量（无论对错），执行 addword 需要两层先决判断条件，第一个判断条件是 eatnum 是否等于当前指定单词的长度（即当前指定单词的字母个数），在确保单词数量正确的情况下，

再进行第二个判断条件：如果当前指针指向的字母和 gamewordsum 库中当前需要你吃的单词的字母相同，这个时候就可以调用 addword 进行游戏界面上单词的刷新了。对变量 appear 的自增指令在 addword 函数中实现。

```
void SnackDif::judgeRight(QString a)
{
  if(gameWordSum[appear-1].eatStr==a)
  {
      this->getScore++;
      qDebug()<<"判断了    "<<getScore;
  }
  this->wordTmp="";
}
```

3.3 吃掉单词后隐藏

每次一个字母被吃掉（无论对错），它都应该在界面上消失并且蛇的身体同步变长，实现这个功能需要函数 wordhide：

```
int SnackDif::wordHide()
{
   int tmp=appear-1;//appear 为现在出现的单词的个数多一个
   int len=gameWordSum[tmp].a.size();
   for(int i=0;i<len;i++)
```

```
    {
        if(snackBody[0].intersects(gameWordSum[tmp].point[i]))
        {
            this->s1->play();
            eatNum+=1;
            qDebug()<<"eatNum "<<eatNum;
            this->wordTmp+=gameWordSum[tmp].a[i];
            qDebug()<<eatNum<<"    0"<<gameWordSum[tmp].a[i];
            gameWordSum[tmp].eatStr+=gameWordSum[tmp].a[i];
            gameWordSum[tmp].fpoint[i]=true;
            //吃完之后就让这个点的位置清 0;
            gameWordSum[tmp].point[i].moveTo(-10,-10);
            gameWordSum[tmp].point[i].moveTo(-10,-10);
            return 1;
        }
    }
    return 0;
}
```

首先设置一个 tmp 变量获取当前出现的单词的个数（appear-1），因为 appear 表示现

在出现的单词的个数多一个，然后获取该单词的长度 len。遍历该单词的每个字符，判断是否与 snackBody[0]相交（snackbody[0]是当前蛇身体的头部矩形，即第一个矩形）。如果相交，播放 s1 音频文件（这个在后面系统讲解），增加 eatNum 计数器，并将该字符添加到 wordTmp 字符串中。将 gameWordSum[tmp].fpoint[i]设置为 true，表示该点已经被吃掉。将 gameWordSum[tmp].point[i]移动到(-10, -10)位置，表示清空该点的位置。如果有一个字符被吃掉，返回 1；否则返回 0。

当然，如果仅仅到此还不足以真正体现这个游戏的意义，为了让玩家在玩游戏的同时能够学到英语，我们需要将吃对的单词全部存储到 eatwordsum 这个数组中，并且设立一个 getscore 变量来统计得分。当 judgeright 函数中二重判断条件通过之后，得分就会自增。

```
    label11=new QLabel(this);
    label22=new QLabel(this);
    this->label33=new QLabel(this);
    label33->setFixedSize(200,100);
    label33->setText(QString("当前的得分是: "+(QString)getScore));
    label33->show();
    labelInt=new QLabel(this);
    labelInt->move(label33->x()+10,label33->y());
    this->label55=new QLabel(this);
    this->label66=new QLabel(this);
    label55->setText("已经吃到的字母是:");
    font.setPointSize(10);
```

```
    label55->setFont(font);
    this->label66->setFont(font);
    label55->move(2,10);
    label66->move(153,-28);//这个是时刻显示出字母的位置
    label66->setFixedSize(100,100);
```

根据前文给出的各项 label 变量的具体含义，这里展示的是对各自的定义和显示位置的确定。

四．退出游戏

4.1 退出游戏及对应后续选项

接下来要讲的是控制游戏结束的模块，这个功能我们单独分了一个 gameover 类，在里面创建了一些触发 gameover 时所需的图片和音效：

```
GameOver::GameOver(QWidget *parent) : QWidget(parent)
{
  this->restart=new MyPushBtn(":/r1/sorce/restart.png");
  this->quit=new MyPushBtn(":/r1/sorce/quit.png");
  this->sound=new QSound(":/r1/sorce/sound1.wav",this);
  restart->move(20,610);
  quit->move(460,610);
  restart->setParent(this);
  quit->setParent(this);
  this->setFixedSize(720,720);
}
```

那么如何触发 gameover 呢，原理很简单，只要让程序判断蛇的身体是否触碰到界面边缘就行了，因此这个判断条件需要添加在每一个 add 函数中（addtop，adddown，addleft，addright），确保在蛇的头部矩形的端点坐标超出临界坐标时触发 gameover，以下是截取的 addtop 函数中的 else 环节代码（即蛇碰到边界后触发的代码）：

```
else
        {
            gameStart=false;
            this->s2->play();
            this->timer->stop();
            this->gameover->show();
            return ;
        }
```

在触发 gameover 后，会再跳出两个选项，一个是退出游戏（即 quit），还有一个是 restart（重新开始），点击 restart 之后就会重置你的游戏记录，把 eatnum 变量清零，从头开始各项流程。

```
this->gameover=new GameOver();
    connect(gameover->quit,&QPushButton::clicked,[=](){
        this->gameover->sound->play();
        exit(0);
    });
    connect(this->gameover->restart,&QPushButton::clicked,[=](){
        this->gameover->sound->play();
        this->snackBody.clear();
        this->eatNum=0;
```


4.2 单词框以及音效

这里再交待一些杂七杂八的边角料工程，例如单词框的显示，音效，背景音乐的插入等等。

```
#include "showtext.h"
ShowText::ShowText(QWidget *parent) : QWidget(parent)
{
    this->text=new QTextEdit(this);
    this->lay=new QVBoxLayout(this);
    lay->addWidget(text);
    this->setFixedSize(300,720);
}
this->showtext=new ShowText;
this->showtext->text->append("游戏中出现的单词:");
this->showtext->show();
```

先展示单词框，用 showtext 类完成，在前三个模式中，只要游戏开始，就会自动在屏幕上多显示一个单词框，用来记录吃对的单词（随心吃模式没有配备）。
然后是为了满足我们想整活的欲望而设置的各种音效：

```
this->forBtn=new MyPushBtn(":/r1/sorce/AI.png");
this->s31=new QSound(":/r1/sorce/first.wav",this);
this->s1=new QSound(":/r1/sorce/sound2.wav",this);
this->s2=new QSound(":/r1/sorce/finish.wav",this);
this->s32=new QSound(":/r1/sorce/beautiful.wav",this);
this->s4=new QSound(":/r1/sorce/defeat.wav",this);
this->s3=new QSound(":/r1/sorce/amazing.wav",this);
this->s33=new QSound(":/r1/sorce/unbelieve.wav",this);
this->s34=new QSound(":/r1/sorce/nicekilling.wav",this);
```


五.随心吃模式和字典树

至此，前三个模式的基本思路已经讲解完毕了，不同之处在于，简单模式会给玩家显示当前要吃的字母，而困难模式不会给玩家显示要吃的字母，要根据游戏界面上分散的字母自行判断当前要吃的字母是什么（这既考验英语水平又考验反应速度）。而无尽模式难度更是降低了，无尽模式中没有死亡这一概念，蛇的身体可以穿墙而过，例如从最右侧进，最左侧出。

而最后一个 AI 智能模式，原本计划用来做 AI 自动操作蛇跑图的功能，后来转变为用来实现字典树查找。这个模式的思路和前几个模式不一样，这里进行详细的介绍：

这个模式是以字典树为基础进行架构的。首先是构建字典树，先将我们系统里所有的单词读入 gameWordsum 这个总的数组里，通过一个 for 循环来构建字典树，利用一个二维数组来存放每个已经构建好的节点的编号。初始时刻二维数组里面元素的初始值都为 0，表示只有根节点创建好了。二维数组的第一维表示的是父节点,第二维的数字表示该字母减去’a’之后对应的编号,也就是说把二十六个英文字母按照这样的规则一一映射到二十六个数字中.首先规定根结点的编号为 0。之后通过 for 循环构建该单词的字典树。从根结点依次往下寻找下一个字母。如果父亲节点下面没有该字母节点。就把该节点对应的在数组里面的值改为

++idx。直到创建到该单词的最后一个字母为止。在创建节点的过程中，为了便于书写 mfind 函数，我们应用一个标志数组来存放每个父亲节点下面是否存在子节点，如果存在子节点，那么它的值就不为零，否则他的值就为零。

接下来就是 mfind 函数的书写，传进来的参数是玩家已经吃到的字母组合。

```
int RandSense::mfind(QString word)
{
    int p=0;
    for(int i=0;i<word.size();i++)
    {
        int j=word[i].toLatin1()-'a';
        if(tree[p][j]==0)
        {
            return 0;
        }
        p=tree[p][j];
    }
    //下面的代码是判断有没有吃到底
    if(cun[p]==1)//也就是说他没有字节点
        return -1;
    else
       return 1;//他还有字节点那么就是没有吃到底
}
```


通过 for 循环来逐个遍历它每个字母是否能够在我们已经构建好的字典树中找到合法的英文单词。方法是通过他的字母来找到他的下一个子节点。因为是每吃到一个字母就进行检查一次，所以分三种情况。第 1 种情况是组合成的字母在我们构建好的字典树中无法组成合法的英文单词。第 2 种情况是吃对了，也就是说他吃到的字母组合能够在我们构建好的字典树中找到相应的英文单词。最后一种情况就是他吃到的字母组合的最后一个字母下面还有子节点。也就是俗称的还没有吃到底。第 1 种情况是最好判断的。直接利用第 1 层 for 循环，如果吃错了，那么它对应的子节点的编号就为零就直接 return 0。第二种情况是在没有吃错的前提下进行判断的,通过在构建字典树的时候创建的标志数组来判断有没有吃到底,如果他的下面还有节点,表示没有吃到底,return -1.最后在没有返回 0 和-1 的情况下直接返回 1 表示吃对了。

到这里整个项目大致就介绍完毕了，由于时间原因还未来得及做蛇碰到自己身体触发 gameover 的功能，其余部分都已完善。