课程设计一 贪吃蛇游戏

计算机科学与技术系 171860574 胡育玮

一、 设计内容

在正式开始写项目代码之前，应先确定所要实现的功能。在这次课程设计中，我将使用 Qt 完成一个带 GUI 的贪吃蛇游戏。该贪吃蛇游戏将具有以下的功能和特点：

- 可以在游戏开始前*选择游戏难度*。难度越高，蛇的移动速度越快；
- 地图全空白，没有障碍物，*上下左右皆可贯穿*；
- 蛇每吃掉一个食物后，其长度加一，同时在场上的随机位置生成一个新的食物；
- 每吃掉 5 个普通食物后，附加多产生一个限时食物;若蛇在给定的移动步数内没有吃掉它，它会自动消失；
- 每吃掉一个普通食物得 1 分，每吃掉一个限时食物得 5 分；
- 若蛇撞到自己则游戏结束，结束后会显示玩家的得分，并*可以重新开始游戏*（以相同的游戏难度）；
- 游戏过程中可以*暂停游戏*和*继续游戏*;

二、设计思路

确定好全部计划实现的功能以后，下面开始规划设计思路。

1. 界面设计

首先是整体的界面设计方案。根据功能设计，本程序应包含3个界面：开始界面、游戏界面、结束界面。其中，开始界面应显示欢迎字样，并告诉玩家游戏的玩法以及注意事项（如按哪个键来暂停游戏等），同还应允许玩家选择游戏难度。游戏界面则是本程序的主体，实现游戏功能，根据开始界面选择的难度来进行游戏。结束界面应显示玩家的得分，并提供重新开始游戏的入口。

下面具体到每个界面地来对整体的界面设计方案进行细化。

- ***开始界面***：开始界面应在上部显示欢迎和提示字样，下部则显示3个按钮，分别有“简单”、“中等”、“困难”的字样，代表 3 中游戏难度。玩家用鼠标点击其中一个按钮，便可以以相应难度开始游戏。

- ***游戏界面***：游戏界面应以纵横网格作为背景，这样玩家看着比较舒服。食物随机出现在单个网格内，分别以红色和蓝色实心圆来代表普通和限时食物。蛇的身体填充一些网格，每吃掉一个食物，蛇占据的网格数加一。

- ***结束界面***：结束界面应以一行文字来显示玩家的得分。该行文字应显示在窗口的上部，窗口的下部提供一个“重新开始”按钮以供重新开始游戏。

2. 游戏逻辑和类的设计

由于已经确定该程序只有 3 个窗口，因此 Qt 项目中应包含 3 个窗体类。游戏的主体应是游戏界面，其它 2 个界面都是陪衬，故确定控制游戏界面的窗体为主窗体。因此主窗体类中应包含另外 2 个窗体类的对象作为成员以实现它们的功能。另外 2 个窗体类由于功能简单，因此不会包含其它设计游戏功能的成员。而对于实现游戏界面和游戏逻辑的主窗体类，情况要复杂得多。

本程序将使用 Qt 的 Graphics View 框架来实现游戏界面和游戏元素的交互。该框架包含 scene（场景）、view（视图）、item（物件）三种重要元素。具体到该游戏中，scene 就是游戏的网格背景与添加到这个背景中的 item（蛇和食物）的综合体，而 view 则相当于观察 scene 的“视角”，通过 view 来观察 scene。本程序的设计中，主窗体类应包含 Qt 自带的 QGraphicsScene 类和 QGraphicsView 类的对象指针，以创建 scene 和 view 来显示游戏主场景。将蛇和食物作为 item 添加到 scene 中以后，蛇的运动和吃食会由 Graphics View 框架自动处理，也即我们只需在程序中其它地方更改蛇的在 scene 中的类似坐标等信息，Graphics View

框架便会自动将运动后的蛇绘制出来。该框架同时还提供接口以检测添加到 scene 中的不同 item 间的碰撞，这使得蛇吃食物的游戏逻辑十分易于实现。

游戏场景创建完后，还需要实现游戏的控制逻辑。在本程序中，将自定义 GameController 类来实现游戏的控制逻辑。GameController 类负责实现蛇的运动、接收键盘按键事件以让蛇改变方向、蛇吃食物、产生新食物、游戏暂停和继续等功能。将一个 GameController 类的对象指针放入主窗体类中，并允许该对象对创建的 scene 进行控制，便可以实现整体的游戏逻辑，让蛇在 scene 中动起来并吃食物。GameController 类应包含一个自定义的 Snake 类对象以表示蛇，同时还需要包含 QTimer 类的计时器对象以控制游戏每一“帧”的运动，还需要包含 score 等变量以实现计分等功能。

详细的类的设计细节见下：

1. 主窗体类

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "end_window.h"
#include "startwindow.h"

class QGraphicsScene;
class QGraphicsView;
class GameController;

class MainWindow : public QMainWindow
```

```
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);


private slots:
    void adjustViewSize(); //调整 view
    void show_end_wind();  //显示结束窗口
    void new_game();       //开始新游戏

    void easy();           //以简单模式开始游戏的槽函数
    void moderate();       //以中等模式开始游戏的槽函数
    void hard();           //以困难模式开始游戏的槽函数

private:
    void initScene();           //初始化场景的大小
    void initSceneBackground(); //初始化场景的背景

    void init(int speed);   //接收到游戏速度后，进行初始化并开始游戏

    QGraphicsScene *scene;  //场景
    QGraphicsView *view;    //视图
    GameController *game;   //控制整个游戏的 game 变量
    end_window *endwind;    //游戏结束时显示的窗口
    StartWindow *startwind; //游戏开始时的界面
};

#endif // MAINWINDOW_H
```

表 1：主窗体类 MainWindow 的接口设计。

2. GameController 类

```
#ifndef GAMECONTROLLER_H
#define GAMECONTROLLER_H

#include <QObject>
#include <QTimer>

```

```
7.  class QGraphicsScene;
8.  class QKeyEvent;
9.
10. class Snake;
11. class Food;
12.
13.
14. class GameController : public QObject
15. {
16.     Q_OBJECT
17.
18. public:
19.     GameController(QGraphicsScene &scene, int speed, QObject *parent = nullp
    tr);
20.
21.     void snakeEatFood(Food *food); //蛇吃东西
22.     void del_spec(Food *food);     //删除场上的特殊食物，添加普通食物
23.     int get_score() const {return score;}  //获取最终得分
24.     void init_score() {score = 0;} //游戏开始时，将得分初始化为0
25.     void new_game();               //开始新游戏
26.
27. signals:
28.     void send_end_signal();        //发送游戏结束的信号给主窗体
29.
30. public slots:
31.     void pause();    //暂停游戏
32.     void resume();   //继续游戏
33.     void gameOver(); //游戏结束
34.
35. protected:
36.      bool eventFilter(QObject *object, QEvent *event); //事件过滤器:仅接受键盘
    事件
37.
38. private:
39.     void handleKeyPressed(QKeyEvent *event); //按键的响应
40.     void addNewFood(bool is_food_spec);      //增加新的食物
41.
42.     QTimer timer; //计时器
43.     QGraphicsScene &scene; //场景的引用,使得游戏控制器得以控制场景
44.     Snake *snake;  //蛇对象
45.
46.     Food *spec_food;  //指向场上的特殊食物的指针
47.     int score;        //得分
48.     int game_speed;   //游戏速度
```

```
};

#endif // GAMECONTROLLER_H
```

表 2：游戏逻辑控制类 GameController 的接口设计。

3. Snake 类

```
#ifndef SNAKE_H
#define SNAKE_H

#include <QGraphicsItem>
#include <QRectF>

class GameController;
class Food;

class Snake : public QGraphicsItem //蛇也是一个 Item，会被添加到 scene 中
{
public:
    enum Direction  //蛇的移动方向：可以是不移动。
    {Still, Left, Right, Up, Down};

    Snake(GameController &, int);

    QRectF boundingRect() const; //蛇的边界矩形
    QPainterPath shape() const; //提供蛇的形状以供绘制
    QPainterPath wholeShape();  //完整的形状
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *, QWidget
*); //画出蛇

    void setMoveDirection(Direction direction); //设置移动方向
    bool eat(Food *food);       //蛇吃食物

protected:
    void advance(int step);    //“帧”控制函数，控制每一“帧”中该 item 的行为

private:
    void moveLeft();    //根据移动方向来控制蛇的移动，下同
    void moveRight();
    void moveUp();
    void moveDown();
```

```
34.     void handleCollisions(); //处理蛇与食物的碰撞
35.
36.     QPointF         head;     //蛇头的位置
37.     int             add_length; //蛇的待生长的长度
38.     int             speed;    //速度
39.     QList<QPointF> tail;     //蛇的各个节点的位置
40.     int             cnt;
41.     Direction       moveDirection; //当前的移动方向
42.     GameController &controller; //将控制器也纳入自己名下，以在需要时调用控制器进
    行相应的处理
43.
44.     int normal_eaten;  //吃掉的普通食物的数量
45. };
46.
47. #endif // SNAKE_H
```

表 3：Snake 类的接口设计。Snake 类的对象将作为 item 被添加到 scene 中。该类包含控制蛇的绘制、蛇的长长、蛇的移动的方法。

4. Food 类

```
1.  #ifndef FOOD_H
2.  #define FOOD_H
3.
4.  #include <QGraphicsItem>
5.  class GameController;
6.
7.
8.  class Food : public QGraphicsItem //food 是一个 graphics item
9.  {
10. public:
11.     Food(qreal x, qreal y, bool normal, GameController *control, int speed);
     //坐标,以及是否是普通食物
12.
13.     QRectF boundingRect() const; //计算边界矩形
14.     void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QW
    idget *widget=Q_NULLPTR); //画出食物
```

```
15.     QPainterPath shape() const;  //提供食物的形状以供绘制
16.
17.     bool Is_special_food() const {return is_spec;} //返回当前食物是否是特殊食物
18.
19. private:
20.     bool is_spec; //是否是普通食物，初始化为 false
21.     int cnt;
22.     GameController *control; //将控制器也纳入自己名下，以在需要时调用控制器进行相应的处理
23.     int speed;
24.     int frames;
25.
26. protected:
27.     void advance(int step); //食物的每帧控制：特殊食物会在一定帧数后消失。
28. };
29.
30. #endif // FOOD_H
```

表 4：Food 类的接口设计。Food 类的对象也会作为 item 被添加到 scene 中。该类包含控制食物的绘制、特殊食物的在场上的计时和消失的代码。

三、运行流程

下面将通过讲解该游戏的完整运行流程来详细说明代码逻辑。

1. 开始界面

开始游戏后，首先显示的是开始界面：

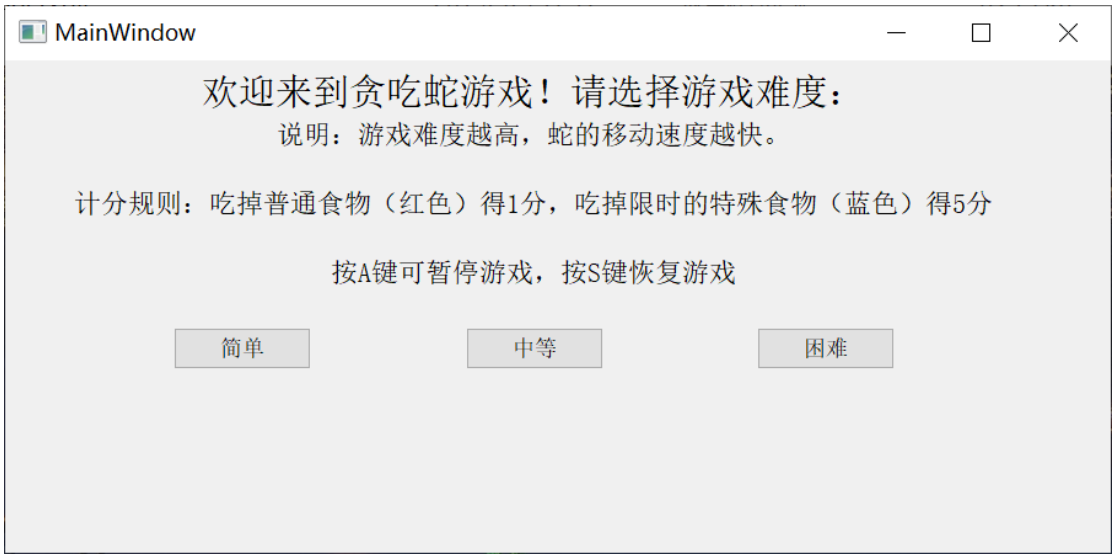


图 1：开始界面

该开始界面在 Qt 中的 UI 设计窗口中的显示情况：

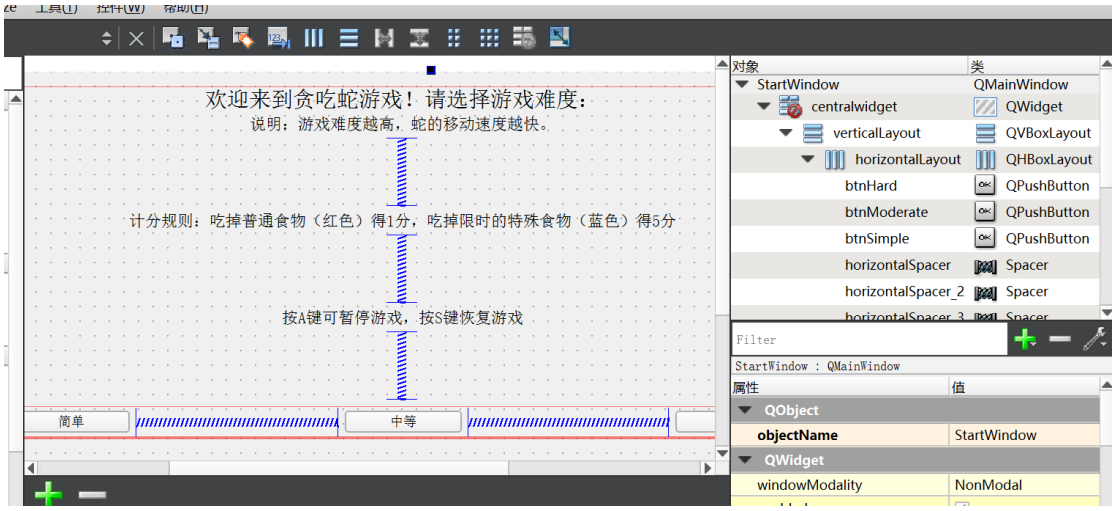


图 2：显示在 Qt 中的 UI 设计窗口中的开始界面

开始界面有 4 行文字，分别是欢迎界面、游戏说明、计分规则、暂停和恢复游戏按钮提示。

界面的下部是 3 个按钮，点击任意一个即可以相应的难度开始游戏。这种操作是通过 Qt 的信号-槽机制实现的。在开始界面

对应的类的代码中，有如下三个槽函数：

```
void StartWindow::on_btnSimple_clicked(bool checked)
{
    emit easymode();
}

void StartWindow::on_btnModerate_clicked(bool checked)
{
    emit moderatemode();
}

void StartWindow::on_btnHard_clicked(bool checked)
{
    emit hardmode();
}
```

图 3：开始界面类中的 3 个槽函数

这三个槽函数相应 Qt 自带的 clicked()信号。当用户用鼠标点击任意一个按钮后，Qt 的信号-槽框架发送 clicked()信号，图中这三个槽函数便是相应的按钮被按下后的clicked()信号的响应函数（槽函数）。可见，这 3 个槽函数中又分别发射了 easymode()、moderatemode()、hardmode()这三个信号。这三个信号不是 Qt 自带的，而是我在开始界面类中自定义的信号：

```
class StartWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit StartWindow(QWidget *parent = nullptr);
    ~StartWindow();

signals:
    void easymode();
    void moderatemode();
    void hardmode();
```

图 4：开始界面类中自定义的 3 个信号

这三个信号上述由开始界面类中的方法发射后，被主窗口类接收。主窗口类中定义了三个槽函数，分别用于相应这三个信号：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);


private slots:
    void adjustViewSize(); //调整view
    void show_end_wind();  //显示结束窗口
    void new_game();       //开始新游戏

    void easy();           //以简单模式开始游戏的槽函数
    void moderate();       //以中等模式开始游戏的槽函数
    void hard();           //以困难模式开始游戏的槽函数
```

```
void MainWindow::easy()
{
    startwind->hide();
    init(EASY);
    show();
}


void MainWindow::moderate()
{
    startwind->hide();
    init(MODERATE);
    show();
}


void MainWindow::hard()
{
    startwind->hide();
    init(HARD);
    show();
}
```

图 5：主窗口类中的 3 个槽函数

如图 5，主窗口类中的 3 个槽函数分别响应以对应难度开始

游戏的信号。它们先用 startwind->hide()隐藏掉开始界面，然后调用 init()初始化方法来用相应的难度对游戏界面进行初始化，最后调用 show()方法显示出主窗口：

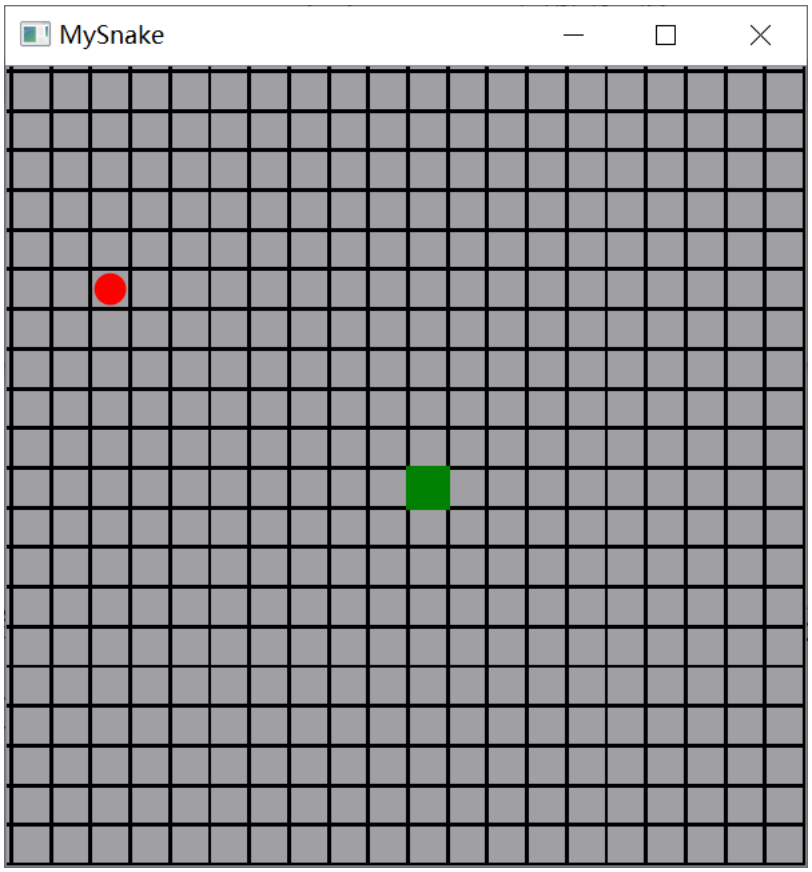


图 6：主窗口（游戏界面）显示后的情况

2. 游戏界面及游戏的控制

在主窗口类的 init()方法中，采用如下的步骤对游戏进行初始化：

```
1.  void MainWindow::init(int speed)
2.  {
3.      scene = new QGraphicsScene(this);  //初始化 scene
4.      view = new QGraphicsView(scene, this); //用 scene 来初始化 view
5.      game = new GameController(*scene, speed, this); //初始化整个游戏的控制变
    量
6.
```

```
7.      //将游戏结束的信号连接到用于显示结束界面的槽函数
8.      connect(game, SIGNAL(send_end_signal()), this, SLOT(show_end_wind()));
9.
10.     //对显示窗口进行设置
11.     setCentralWidget(view); //把 view 这个 widget 设为显示窗口的中心 widget
12.     setFixedSize(WINDSIZE, WINDSIZE);//设置显示窗口的大小，并不允许用户修改窗口
    大小
13.
14.     initScene();            //设置场景矩形
15.     initSceneBackground();  //设置场景的背景：纵横网格
16.
17.     QTimer::singleShot(0, this, SLOT(adjustViewSize())); //放入事件队列，下个
    循环周期开始调用
18. }
```

下面逐一分析该函数中的各项操作。

（1）初始化 scene 和 view

```
19.     scene = new QGraphicsScene(this);  //初始化 scene
20.     view = new QGraphicsView(scene, this); //用 scene 来初始化 view
```

这项操作给成员变量 scene 和 view 赋值。

（2）初始化 game

```
game = new GameController(*scene, speed, this); //初始化整个游戏的控制变量
```

该操作调用 GameController 类的构造函数对 game 进行初始化。该构造函数细节如下：

```
1.  GameController::GameController(QGraphicsScene &scene, int speed, QObject *pa
    rent) :
2.      QObject(parent),
3.      scene(scene),
4.      snake(new Snake(*this, speed)),
5.      game_speed(speed)
6.  {
7.      timer.start( 1000 / 30 );  //定时器设置每一次响应的间隔
8.      spec_food = nullptr;       //一开始没有特殊食物
```

```
9.      Food *first_food = new Food(-80, -50, false, this, game_speed); //设置第
    一个食物的位置
10.     scene.addItem(first_food); //往场景中添加蛇和首个食物
11.     scene.addItem(snake);       //场景中添加 snake
12.     scene.installEventFilter(this); //启动事件过滤器
13.
14.     init_score(); //初始化比分
15.     qsrand(QTime(0, 0, 0).secsTo(QTime::currentTime()));
16.     resume();      //正式开始游戏
17. }
```

这个构造函数中初始化了 Snake 类变量 snake，游戏速度 game_speed，同时设置了定时器变量 timer 的定时间隔。在本游戏中，时间间隔设置为 1000 / 30，也即每 1 / 30 秒就让 timer 发送 timeout()信号，然后让程序对 timeout()信号进行处理，这样便实现了每秒 30“帧”的机制。构造函数还将指向特殊食物的指针初始化为 nullptr，因为游戏开始时没有特殊食物。同时还初始化了一个 Food 类变量，设置其坐标并将其和蛇一起添加到了 scene 中。同时还启动事件过滤器，让 this（游戏控制器 game 变量）来处理 scene 接收的事件并进行处理。同时还将得分置为 0，设置随机数种子，同时调用 resume()以正式开始游戏循环。resume()的代码如下：

```
1.  void GameController::resume()
2.  {   //执行完这个 connect 后，整个程序的游戏循环就已经开始了
3.      connect(&timer, SIGNAL(timeout()), &scene, SLOT(advance()), Qt::UniqueCo
    nnection);
4.  }
```

可见该函数调用 Qt 的 connect 函数将 timer 的 timeout()信号连接到 scene 的 advance()方法，这样每次 timeout 后，scene 都会

有所反应，因此执行完这个连接函数后，游戏便正式开始。scene 是 Qt 内置类型 QGraphicsScene 类的对象指针，其 advance()方法会自动调用所有添加到 scene 上的 item 的 advance()方法，来实现 item 的“运动”。具体到本程序中，每次 timeout()后 scene 的 advance()方法被执行，执行过程中调用所有 scene 上的 item（也即蛇和食物）的 advance()方法。因此只需实现 Snake 类和 Food 类的 advance()方法，即可在每次 timeout 后让蛇移动并进行吃食物判断。

（4）初始化显示窗口：

```
1.      //对显示窗口进行设置
2.      setCentralWidget(view); //把 view 这个 widget 设为显示窗口的中心 widget
3.      setFixedSize(WINDSIZE, WINDSIZE);//设置显示窗口的大小，并不允许用户修改窗口
    大小
4.
5.      initScene();            //设置场景矩形
6.      initSceneBackground();  //设置场景的背景：纵横网格
7.
8.      QTimer::singleShot(0, this, SLOT(adjustViewSize())); //放入事件队列，下个
    循环周期开始调用
```

setCentralWidget()方法将当前的 view 设置为显示窗口（应用程序窗口）的中心，setFixedSize()设置显示窗口的大小。initScene()设置 scene 的边界矩形，这在 Graphics View 框架中是必须设置的，这样才能让系统自动绘制场景。initSceneBackground()绘制场景的背景，在本例中为纵横网格。绘制的代码如下：

```
1.  void MainWindow::initSceneBackground()
2.  {
```

```
3.      QPixmap bg(TILE_SIZE, TILE_SIZE); //tile_size = 10
4.      QPainter p(&bg);
5.      p.setBrush(QBrush(Qt::gray)); //bg 是一个 paint device,在它上面绘制之后，bg
    就有了东西！
6.      p.drawRect(0, 0, TILE_SIZE, TILE_SIZE);
7.      scene->setBackgroundBrush(QBrush(bg));
8.  }
```

首先创建一个 pixmap 作为绘图设备，用其初始化绘图画笔 QPainter 类对象 p，然后设置 p 的画刷颜色为灰色，最后调用 drawRect()方法在 Pixmap 变量 bg 上进行绘制，然后用绘制好的 pixmap 变量作为 scene 的背景画刷，填充整个 scene。在 drawRect()中，设置的绘制区域并没有覆盖整个 pixmap，绘制出的 bg 实际上是这样的：

图 7：绘制出的 pixmap

可见灰色并没有覆盖整个图形，图形的左侧和上侧留有黑色边框。如此一来，这样的图形在平铺填满 scene 时，便可以形成纵横网格。

至此，游戏窗口的初始化工作已完毕，游戏窗口已经可以显示出来。下面进入游戏的**逻辑控制部分**。

游戏的逻辑控制：

前面提到，在 game 变量的构造过程中，调用 resume()后，游戏的帧循环便已开始，可以认为已经开始游戏。但玩家会发现，如果不去动方向键，则蛇依然不会运动，游戏会定在下面的场景中：

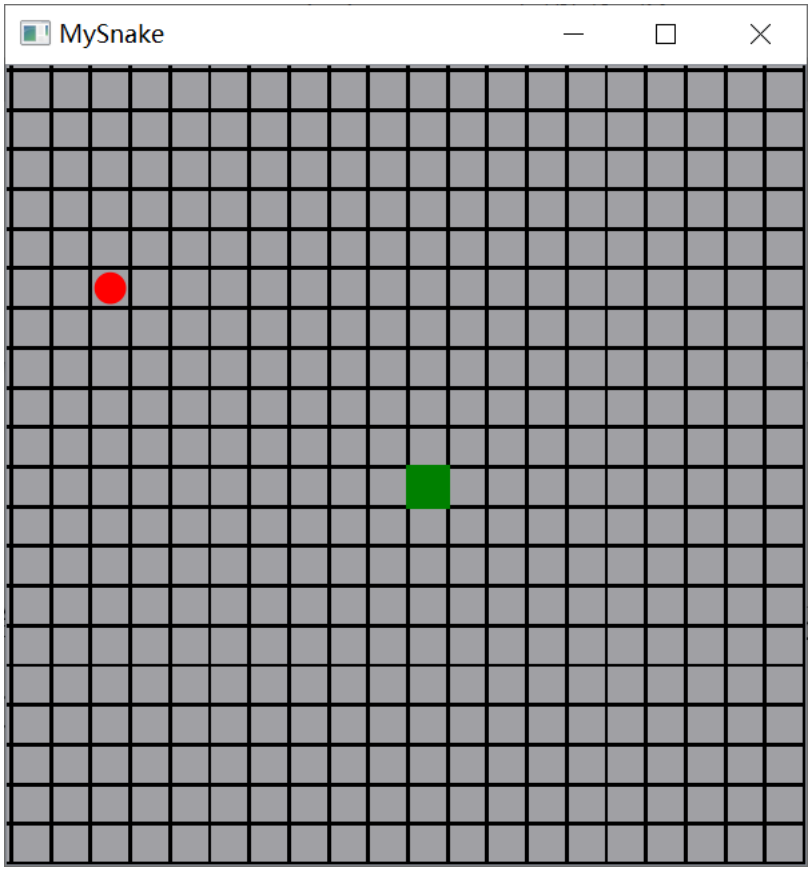


图 8：一开始只有一个不动的蛇头

这是因为蛇 snake 变量在初始化时，移动方向被置为 Still，也即不运动。只有玩家按下方向键触发了键盘事件，程序对其响应后，蛇才会动起来：

```
1.  bool GameController::eventFilter(QObject *object, QEvent *event)
```

```
{
    if (event->type() == QEvent::KeyPress)
    {
        handleKeyPressed(static_cast<QKeyEvent *>(event));
        return true;
    }

    else
        return QObject::eventFilter(object, event); //do nothing
}
```

上面的代码段为 GameController 类的 eventFillter()方法。前面提到过，game 变量在构造时初始化了事件过滤器，game 对象来处理 scene 接收到的键盘事件。这个处理过程由系统自动调用 game 对象中的 eventFilter()方法来进行处理。如上，在该方法中，首先判断事件类型是不是键盘事件，是则由本类的 handleKeyPressed()方法进一步处理，否则抛给父类的 eventFilter()进行处理。handleKeyPressed()的代码如下：

```
void GameController::handleKeyPressed(QKeyEvent *event)
{
    switch (event->key())
    {
        case Qt::Key_Left:
            snake->setMoveDirection(Snake::Left);
            break;
        case Qt::Key_Right:
            snake->setMoveDirection(Snake::Right);
            break;
        case Qt::Key_Up:
            snake->setMoveDirection(Snake::Up);
            break;
        case Qt::Key_Down:
            snake->setMoveDirection(Snake::Down);
            break;

        case Qt::Key_A:
            pause();
```

```
            break;

        case Qt::Key_S:
            resume();
            break;
    }
}
```

可见，该方法根据按下的方向键的类型来调用 snake 对象的设置方向的方法来设置蛇的移动方向。若按下的是 A 或 S 键，则暂停或恢复游戏。暂停游戏的代码编写很简单：

```
void GameController::pause()
{
    disconnect(&timer, SIGNAL(timeout()), &scene, SLOT(advance())); //每当 timeout 时，就执行 advance()
}
```

只需断开 timer 的 timeout()信号与 scene 的 advance()方法之间的连接，这样基于 timeout 的帧动作就会停止，于是实现了暂停。要想继续游戏，则调用 resume()方法，恢复信号连接即可。

前面讲过，scene 的 advance()会调用 scene 中每个 item 的 advance()方法，在本游戏中通过这种方式实现帧动作。下面详细说明 snake 和 food 的 advance()方法应如何编写。首先是 snake 的 advance()方法：

```
void Snake::advance(int step)
{
    if (!step || cnt++ % speed != 0 || moveDirection == Still)
        return;

    if (add_length > 0)
    {
```

```
8.          tail << head; //直接在食物的位置增长!
9.          add_length -= 1;
10.     }
11.
12.     else
13.     {
14.         tail.takeFirst(); //去除掉蛇尾的节点
15.         tail << head;     //追加 head 到头部
16.     }
17.
18.     switch (moveDirection) //设置下一次 advance 后，蛇转向哪里移动
19.     {
20.         case Left: moveLeft(); break;
21.         case Right: moveRight(); break;
22.         case Up: moveUp(); break;
23.         case Down: moveDown(); break;
24.         default: break;
25.     }
26.
27.     setPos(head); //设置蛇的位置
28.     handleCollisions();
29. }
```

这个函数中，add_length 是蛇将要长长的长度。这个值被构造函数初始化为 4，表明蛇一开始包括头在内在不吃食物的情况下有 5 格的身体。若 add_length 大于 0，则表明蛇应继续长长，于是将 head 添加到表示蛇的身体的各个节点的坐标的 tail 数组中，这样蛇便长长了一格。若 add_length 等于 0，则首先移除最先被添加进 tail 数组的节点，这样蛇的尾部缩短了一格；同时将 head 添加到 tail 中，这样蛇的头部又长长了一格，两者抵消，相当于蛇移动了一格，没有长长。

然后根据类中的 moveDirection 属性成员确定当前的移动方向并进行移动。switch 语句中调用的 moveLeft()等函数调整 head 的值，以决定下一帧中蛇朝哪个方向移动。

最后调用 handleCollisions()方法来处理蛇碰到自己或蛇碰到食物的情况。代码如下：

```
1.  void Snake::handleCollisions()
2.  {   //2 种碰撞情况：吃到食物，吃到自己
3.      QList<QGraphicsItem *> collisions = collidingItems(); //获取 all 与当前对象发生碰撞的物件
4.
5.      if (!collisions.isEmpty())
6.      {
7.          controller.snakeEatFood((Food *)collisions[0]);
8.          add_length += 1;
9.          return;
10.     }
11.
12.     // Check snake eating itself
13.     for (QPointF p: tail) //fix one bug
14.     {
15.         if (p == head)
16.         {
17.             controller.gameOver();
18.             return;
19.         }
20.     }
21. }
```

首先调用 collidingItems()方法来获取 scene 中与当前 item 碰撞的所有 item 组成的 QList 列表。这个方法是 Graphics View 框架给本游戏的实现提供的诸多便利之一。collidingItems()方法是 QGraphicsItem 类的方法，我们的蛇和食物都是要被添加到场景中的 item，因此 Snake 类和 Food 类都继承自 QGraphicsItem 类，因此 snake 可以调用该方法。由于与蛇碰撞的 item 只可能是食物，因此只要列表非空便可以确定蛇吃到了食物。此时调用 controller（也即 game 对象）的 snakeEatFood()方法进行蛇吃食物

的处理。处理内容包括：在 scene 中移除被吃的食物(item)；在随机位置添加新的食物；判断是否需要添加特殊食物。

接着，遍历 tail 数组，看看当前头的位置 head 是否已经存在于 tail 数组中，如果存在则说明蛇撞到了自己，此时游戏结束，调用 game 的 gameOver()方法显示结束界面。

Food 类的 advance()则比较简单：

```
1.  void Food::advance(int step)
2.  {
3.      if (0 == step || cnt++ % speed != 0 || is_spec == false) //不是特殊食物：
    不处理
4.          return;
5.
6.      frames++;
7.      if (frames == TIMELAP)
8.          control->del_spec(this);
9.  }
```

Food 类的构造函数中有一个参数表明要初始化的这个对象是否是特殊食物。如果不是，则 advance()不进行任何处理。否则，经过一定的帧数后，advance()会调用 game 的 del_spec()方法来删除掉 scene 中的特殊食物。

游戏的主体逻辑讲解完毕。下面讲解蛇和食物是怎么绘制的。在 Graphics View 框架中，图形的绘制是系统自动完成的，我们只需要在自定义的物件类（需要继承 QGraphicsItem 类）中实现 boundingRect()虚函数和 paint()虚函数即可（必要时还需实现

shape()虚函数。shape()给出要进行绘制的形状。)。boundingRect()给出 item 的边界矩形，paint()给出 item 的绘制方法。系统自动调用 paint()进行物件的绘制。

3. 结束界面

在 Snake 类的 handleCollisions()方法中，若蛇吃到自己则会调用 game 控制器的 gameOver()方法。这个方法的代码如下：

```
1. void GameController::gameOver()
2. {
3.     pause();
4.     emit send_end_signal();
5. }
```

可见该方法先暂停游戏，然后发送 send_end_signal()信号。这个信号被主窗体类中的 show_end_wind()槽函数接收：

```
1. void MainWindow::show_end_wind()
2. {
3.     int score = game->get_score();
4.     endwind->show_window(score);
5. }
```

这个槽函数首先获取游戏的得分，然后调用把结束界面显示出来：

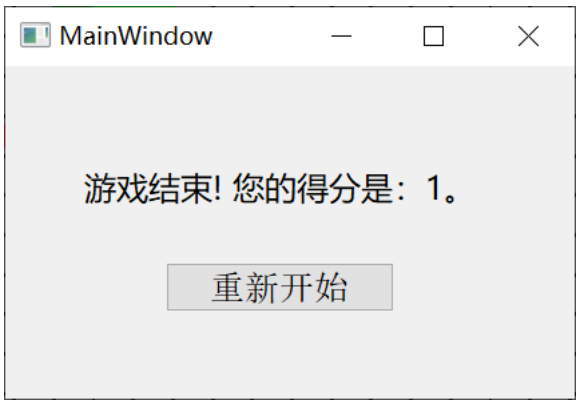


图 9：结束界面

用户点击“重新开始”后，该按钮被点击的信号会最终传导到主窗体类的 new_game()方法中，被其处理：

```
1.  void MainWindow::new_game()
2.  {
3.      endwind->hide();
4.      game->new_game();
5.  }
```

该方法首先隐藏结束窗口，然后调用 game 控制器的 new_game()方法来重新开始游戏。new_game()方法代码如下：

```
1.  void GameController::new_game()
2.  {
3.      scene.clear();
4.      spec_food = nullptr;
5.
6.      snake = new Snake(*this, game_speed);
7.      scene.addItem(snake);
8.      addNewFood(false);
9.      init_score();
10.     resume();
11. }
```

首先清空场景，然后创建一条新的蛇和新的食物并添加到场

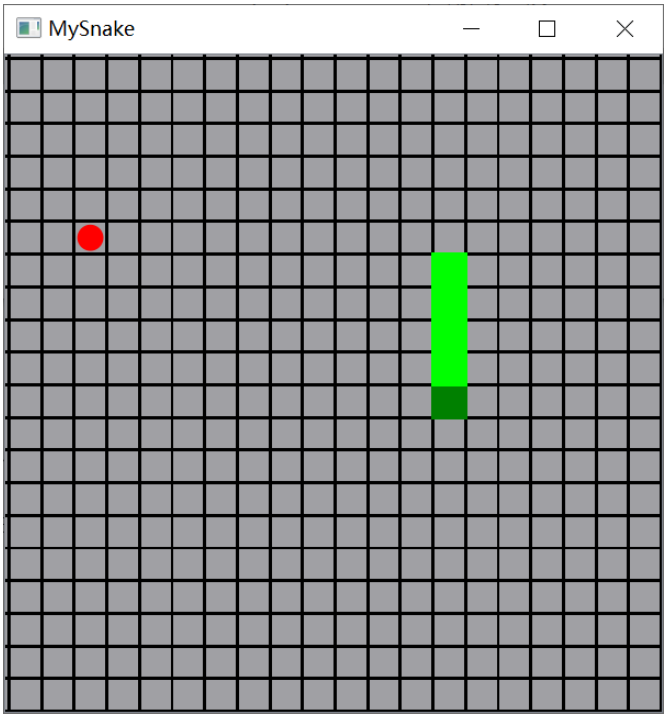
景中，接着将得分初始化为 0，最后调用 resume()方法重新建立 timeout()信号与 scene 的 advance()的连接，以开始新的游戏。

至此，整个游戏程序全部介绍完毕。

四、遇到的问题和解决方法

（1）如何绘制出纵横网格背景：采用前面提到的方法。

（2）如何让蛇的头节点和蛇身节点采用不一样的颜色（如下图）：分别绘制蛇身和蛇头，而不是将整个蛇的所有节点一起放入一个 QPainterPath 类变量中来一起绘制。



（3）游戏未暂停时若按下 S 键（继续游戏）则蛇运动速度变快，

且按 S 键次数越多速度越快：在 resume()中将 connect()函数的参数从

```
connect(&timer, SIGNAL(timeout()), &scene, SLOT(advance()));
```

改为

```
connect(&timer, SIGNAL(timeout()), &scene, SLOT(advance()), Qt::UniqueConnection);
```

增加了 Qt::UniqueConnection 参数后，信号不会多次连接，从而解决了这个问题。

（4）蛇会移动到窗口中的 20 * 20 的纵横网格以外的地方：调整蛇的移动函数，修改坐标边界值