# 基于C++ Qt的“躲避球”游戏设计与实现

## 1 项目概述

本项目基于C++ Qt制作了一个“躲避球”的小游戏。Qt 是一个1991年由Qt Company开发的跨平台C++图形用户界面应用程序开发框架。它是面向对象的框架，很容易扩展，并且允许组件编程。本设计所选则的Qt版本为5.12.3。

首先，游戏的主要效果图如图1所示。游戏中玩家用鼠标控制飞船的移动，躲避黄色子弹，如果不幸撞到子弹则损失一个生命值。同时，图1中的红心为加血球，碰撞到加血球后生命值加一。同时屏幕重要会显示当前游戏进行的时间与剩余生命值。



图 1 游戏主界面

由于我无游戏开发经验，所以本项目的设计与实现是先阅读了一些博文与游戏demo[1]，在理解游戏设计思路以及学习C++语法和Qt设计模式后，完成了自主项目的开发，其不同之处在于：

1. [1]中采用的编程工具为Visual Studio，主要使用了C++图形库graphics.h，而Qt与Visual Studio的编程模式存在巨大差异（我其实真的不想说我是怎么知道的）。因为首先，我想除单一的游戏界面外，还增加“开始游戏”界面与“游戏结束”界面，使游戏更具完整性。而当我使用Visual Studio实现了游戏的主体功能，准备在此基础上增加“开始游戏”界面与“游戏结束”界面时，却发现无法使用graphics.h图形库实现这一目标。当我了解到Qt是面向图形用户界面应用程序的开发工具时，我决定转战Qt。可是我又发现，如果只是简单的对Visual Studio中的代码进行Ctrl\_C与Ctrl\_V至Qt，那是完全不能运行的，于是我查阅了Qt的一些游戏案例，故而发现Qt与Visual Studio在游戏实现的套路上存在很大不同，若想使用Qt完成本项目只能从头开始，别人的肩膀并不是轻易可站的。
2. 考虑到玩家体验性，希望玩家在游玩时能从易到难，所以在子弹效果上，我利用定时器，实现随着游戏时间的推进，屏幕上的子弹数量由少至多，直至设定的子弹数量上限。
3. 增加了加血球的功能，玩家不仅仅只是被动的掉血，也能够恢复生命值。
4. 增加“开始游戏”界面与“游戏结束”界面。“开始游戏”界面有两个按钮——“开始游戏”与“历史记录”，点击“开始游戏”可以跳转至游戏界面，点击“历史记录”可以查看过去游戏的成绩。在游戏结束后，会出现“游戏结束”界面，界面会显示本局游戏所坚持的时间，以及一个“再玩一局”按钮，点击此按钮会回到“开始游戏”的界面。
5. 增加Mysql模块，主要是记录每局的坚持时间，同时，“开始游戏”界面上的“历史记录”按钮本质上是查询Mysql中所存储的数据，按从高到低的顺序显示过往的游戏记录。本项目之所以使用Qt5.12.3，是因为Qt5.12.3是最后提供Mysql数据库插件的版本，往后的版本需要自行编译对应的Mysql数据库插件，而Qt5.12.3只需导入数据库相关头文件即可进行数据库操作。

综上所述，本项目所实现的主要内容有：

1、游戏的三个界面：“开始游戏”界面、主界面、“游戏结束”界面，且三个界面可以实现跳转；

2、碰撞检测：主要用于检测飞船与子弹的碰撞、飞船与加血球的碰撞，以实现生命值的加减；

3、难度递增：随着游戏的进行，屏幕上的子弹数增多，游戏难度加大；

4、MySQL模块：将每一次游戏的进行时间作为成绩，存入数据库，并在开始界面增加查看历史成绩的选项，即通过读取MySQL数据库，由高到低，得到历史记录。

## 2 基本配置与类成员函数

本游戏的主要成员有四个：游戏背景、飞船、子弹、加血球，如表1所示。

表 1 游戏的主要成员

|  |  |  |
| --- | --- | --- |
| 成员 | 代码中的类名 | 备注 |
| 游戏背景 | Map | 游戏界面的背景图片 |
| 飞船 | Rocket | 由玩家控制，通过躲避子弹与吃加血球延长游戏时间 |
| 子弹 | EnemyPlane | 若玩家触碰到子弹，则生命值减一 |
| 加血球 | Heart | 若玩家触碰到加血球，则生命值加一 |

### 2.1 基本配置数据

创建新的头文件为 config.h 主要记录程序中所有的配置数据，如有需要方便后期修改。

1. #ifndef CONFIG\_H
2. #define CONFIG\_H
3. #define PI 3.14159
4. #define GAME\_WIDTH 448 *//游戏窗口的宽度*
5. #define GAME\_HEIGHT 640 *//游戏窗口的高度*
6. #define GAME\_TITLE "无处可躲"
7. #define GAME\_RES\_PATH "./plane.rcc" *//rcc文件路径*
8. #define GAME\_ICON ":/resource/ufo1.png" *//游戏窗口左上角的图标路径*
9. #define MAP\_PATH ":/resource/background.png" *//游戏背景图存放路径*
10. #define GAME\_RATE 10 *//刷新间隔，帧率 单位毫秒*
11. #define HERO\_PATH ":/resource/rocket.png" *//飞船图片存放路径*
12. #define ENEMY\_PATH ":/resource/bullet.png"*//子弹图片存放路径*
13. #define ENEMY\_NUM 17 *//子弹的最大数量*
14. #define ENEMY\_INTERVAL 20
15. #define HEART\_PATH ":/resource/ufo.png"
16. #endif *// CONFIG\_H*

### 2.2 主场景基本设置

在mainscene.h中添加新的成员函数initScene() 用来初始化游戏场景。

1. *//初始化界面*
2. void initScene();

在mainscene.cpp中实现如下代码。

1. void MainScene::initScene()
2. {
3. life = 3;   *//初始生命数量为3*
4. num\_enemy = 1; *//屏幕上的子弹数量先从1开始*
5. cout << "Begining life: " << life << "\n";
6. *//初始化窗口大小*
7. setFixedSize(GAME\_WIDTH,GAME\_HEIGHT);
8. *//设置窗口标题*
9. setWindowTitle(GAME\_TITLE);
10. *//设置图标资源*
11. setWindowIcon(QIcon(GAME\_ICON));
12. *//定时器设置*
13. m\_Timer.setInterval(GAME\_RATE);
14. m\_recorder = 0;
15. h\_recorder = 0;
16. srand((unsigned int)time(NULL));
17. }

### 2.3 创建游戏背景

首先，设置成员属性。这里需要简单介绍一下Qt坐标系。Qt中每一个窗口都有一个坐标系，默认窗口左上角为坐标原点，然后水平向右依次增大，水平向左依次减小，垂直向下依次增大，垂直向上依次减小。原点即为（0，0）点，以像素为单位增减。

1. #ifndef MAP\_H
2. #define MAP\_H
3. #include <QPixmap>
4. class Map
5. {
6. public:
7. Map();
8. void mapPosition();
9. public:
10. *//背景图片对象*
11. QPixmap m\_map1;
12. *//背景图片Y轴坐标*
13. int m\_map1\_posY;
14. };
15. #endif *// MAP\_H*

接着，在map.cpp中实现成员函数。函数mapPosition()可以在之后的mainscene.cpp中被引用，其功能是更新地图坐标。通过代码我们可以看到，在初始化时背景图片的纵坐标为-GAME\_HEIGHT，通过控制背景图片的纵坐标可以使得此时的背景图片不出现在界面中。而当游戏开始时，调用mapPosition()函数，将背景图片的纵坐标设为0，此时背景图片的左上角将恰好位于界面坐标（0，0）处，实现背景图片对界面的覆盖。

1. #include "map.h"
2. #include "config.h"
3. Map::Map()
4. {
5. m\_map1.load(MAP\_PATH);
6. m\_map1\_posY = -GAME\_HEIGHT;
7. }
8. void Map::mapPosition()
9. {
10. m\_map1\_posY = 0;
11. }

### 2.4 创建飞船

首先，在rocket.h中添加代码。

1. #ifndef ROCKET\_H
2. #define ROCKET\_H
3. #include <QPixmap>
4. class Rocket
5. {
6. public:
7. Rocket();
8. void setPosition(int x, int y);
9. public:
10. QPixmap m\_Rocket;
11. *//飞船坐标*
12. int m\_X;
13. int m\_Y;
14. *//飞船矩形边框*
15. QRect m\_Rect;
16. bool m\_RktFree;
17. };
18. #endif *// ROCKET\_H*

接着在rocket.cpp中实现成员函数代码。由于飞船的坐标（m\_X，m\_Y）是位于飞船图片的左上角，所以通过初始化坐标，将飞船的初始位置设置在界面的最下方正中央。初始化矩形框的目的是为之后的碰撞检测而做准备。

1. #include "rocket.h"
2. #include "config.h"
3. Rocket::Rocket()
4. {
5. *//初始化加载飞船图片资源*
6. m\_Rocket.load(HERO\_PATH);
7. *//初始化坐标*
8. m\_X = GAME\_WIDTH \* 0.5 - m\_Rocket.width()\*0.5;
9. m\_Y = GAME\_HEIGHT - m\_Rocket.height();
10. *//初始化矩形框*
11. m\_Rect.setWidth(m\_Rocket.width());
12. m\_Rect.setHeight(m\_Rocket.height());
13. m\_Rect.moveTo(m\_X,m\_Y);
14. m\_RktFree = true;
15. }
16. void Rocket::setPosition(int x, int y)
17. {
18. m\_X = x;
19. m\_Y = y;
20. m\_Rect.moveTo(m\_X,m\_Y);
21. }

### 2.5 创建子弹

同理，首先设置成员属性enemyplane.h。这里有一个布尔变量m\_Free，该变量的作用主要是控制屏幕上的子弹数量，当m\_Free为true时，子弹为空闲子弹，不在屏幕中显示；当m\_Free为false时，子弹需要在屏幕中显示，通过位置更新函数，实现移动。

1. #ifndef ENEMYPLANE\_H
2. #define ENEMYPLANE\_H
3. #include <QPixmap>
4. class EnemyPlane
5. {
6. public:
7. EnemyPlane();
8. *//更新坐标*
9. void updatePosition();
10. public:
11. *//子弹资源对象*
12. QPixmap m\_enemy;
13. *//位置*
14. int m\_X;
15. int m\_Y;
16. *//速度*
17. float m\_VX;
18. float m\_VY;
19. *//子弹的矩形边框（碰撞检测）*
20. QRect m\_Rect;
21. *//状态*
22. bool m\_Free;
23. };
24. #endif *// ENEMYPLANE\_H*

然后编写成员函数。在成员函数中，（m\_X，m\_Y）表示的是当前子弹的坐标值，m\_VX和m\_VY分别表示飞船的水平速度与竖直速度。这里，子弹的矩形框之所以设置成子弹图片的3/5，是因为子弹在游戏中呈现形式是圆形，但实际上子弹的原始图片属于方形，如图2，而如果将矩形框的高度设置成与图片等高等宽，那么在进行游戏时会出现这样的情况：当玩家所操纵的飞船与子弹距离比较接近时，玩家可能直观视觉上并未碰到子弹，但依然会导致生命值减一，这是因为，虽然飞船并没有触碰到子弹的圆形边，但实际上已经触碰到子弹原图的边界。经过不断调参比较后，将矩形框设为原始图片长宽的3/5时，体验感比较好。

bullet

图 2 子弹的原始图片资源

同时，子弹必须只能从界面下方离开页面，当碰到两侧边界时需要进行反弹，这里将变量m\_VX设置为相反数即可实现反弹效果。当子弹从最下方边界从屏幕中消失后，将子弹的状态位m\_Free重设为true。

1. #include "enemyplane.h"
2. #include "config.h"
3. #include <conio.h>
4. #include <math.h>
5. #include <iostream>
6. using namespace std;
7. EnemyPlane::EnemyPlane()
8. {
9. *//子弹资源加载*
10. m\_enemy.load(ENEMY\_PATH);
11. *//子弹位置*
12. m\_X = 0;
13. m\_Y = 0;
14. m\_VY = 0;
15. *//Y轴速度小于1会出现“在顶部平移的子弹”*
16. while(m\_VY <= 1){
17. float angle = (rand() / double(RAND\_MAX) - 0.5) \* 0.9 \* PI;
18. float scalar = 2 \* rand() / double(RAND\_MAX) + 2;
19. m\_VX = scalar \* sin(angle);
20. m\_VY = scalar \* cos(angle);
21. }
22. *//子弹状态*
23. m\_Free = true;
24. *//子弹矩形*
25. m\_Rect.setWidth(m\_enemy.width() \* 3 / 5);
26. m\_Rect.setHeight(m\_enemy.height() \* 3 / 5);
27. m\_Rect.moveTo(m\_X,m\_Y);
28. }
29. void EnemyPlane::updatePosition()
30. {
31. *//空闲状态，不计算坐标*
32. if(m\_Free)
33. {
34. return;
35. }
36. *//更改行进间的轨迹*
37. m\_X += m\_VX;
38. m\_Y += m\_VY;
39. *//子弹触碰到两边的壁后实现反弹*
40. if (m\_X <= 0 || m\_X >= GAME\_WIDTH){
41. m\_VX = -m\_VX;
42. }
43. m\_Rect.moveTo(m\_X,m\_Y);
44. *//子弹出界后状态返回为true*
45. if(m\_Y >= GAME\_HEIGHT)
46. {
47. m\_Free = true;
48. }
49. }

### 2.5 创建加血球

加血球成员属性代码。这里也有一个bool变量h\_Free，其功能与2.4中的m\_Free相同，只有当h\_Free为false时，加血球才会出现在屏幕上，开始移动。

1. #ifndef HEART\_H
2. #define HEART\_H
3. #include <QPixmap>
4. class Heart
5. {
6. public:
7. Heart();
8. *//更新坐标*
9. void updatePosition();
10. public:
11. *//加血球资源对象*
12. QPixmap h\_heart;
13. *//位置*
14. int h\_X;
15. int h\_Y;
16. float h\_VX;
17. float h\_VY;
18. *//加血球的矩形边框（碰撞检测）*
19. QRect h\_Rect;
20. *//状态*
21. bool h\_Free;
22. };
23. #endif *// HEART\_H*

加血球成员函数代码。其目的也是为了实现加血球在屏幕中的随机移动。

1. #include "heart.h"
2. #include "config.h"
3. #include <conio.h>
4. #include <math.h>
5. Heart::Heart()
6. {
7. *//加血球对象资源加载*
8. h\_heart.load(HEART\_PATH);
9. *//加血球对象位置*
10. h\_X = 0;
11. h\_Y = 0;
12. h\_VY = 0;
13. while(h\_VY <= 1){
14. float angle = (rand()/double(RAND\_MAX) - 0.5) \* 0.9 \* PI;
15. float scalar = 2 \* rand() / double(RAND\_MAX) + 2;
16. h\_VX = scalar \* sin(angle);
17. h\_VY = scalar \* cos(angle);
18. }
19. *//加血球对象状态*
20. h\_Free = true;
21. *//加血球对象矩形*
22. h\_Rect.setWidth(h\_heart.width() \* 3 / 5);
23. h\_Rect.setHeight(h\_heart.height() \* 3 / 5);
24. h\_Rect.moveTo(h\_X,h\_Y);
25. }
26. void Heart::updatePosition()
27. {
28. *//空闲状态，不计算坐标*
29. if(h\_Free)
30. {
31. return;
32. }
33. *//更改行进间的轨迹*
34. h\_X += h\_VX;
35. h\_Y += h\_VY;
36. if (h\_X <= 0 || h\_X >= GAME\_WIDTH){
37. h\_VX = -h\_VX;
38. }
39. h\_Rect.moveTo(h\_X,h\_Y);
40. *//加血球对象出界后状态返回为true*
41. if(h\_Y >= GAME\_HEIGHT)
42. {
43. h\_Free = true;
44. }
45. }

## 3 显示对象

在创建好主要类并编写完类成员函数代码后，需要将这些类成员在屏幕中显示出来，这里使用了QPainter类，QPainter可以绘制从简单的直线到像饼图和弦这样的复杂形状。它也可以绘制排列的文本和像素映射。QPainter 一般在部件的绘图事件 paintEvent() 中进行绘制，首先创建 QPainter 对象，然后进行图形的绘制。

QPainter类中的drawPixmap()函数用于绘制对象，该函数共有三个参数，参数一和参数二分别为对象的横纵坐标，参数三为所需要绘制类中的QPixmap对象，该对象与图片资源关联在一起。同时，在游戏界面的中上方以文字的形式提醒玩家当前游戏的进行时间与剩余生命值。

1. void MainScene::paintEvent(QPaintEvent \*event)
2. {
3. QPainter painter(this);
4. *//绘制地图*
5. painter.drawPixmap(0,m\_map.m\_map1\_posY, m\_map.m\_map1);
6. *//绘制飞船*
7. painter.drawPixmap(m\_hero.m\_X,m\_hero.m\_Y,m\_hero.m\_Rocket);
8. *//绘制子弹*
9. for(int i = 0 ; i< ENEMY\_NUM;i++)
10. {
11. if(m\_enemys[i].m\_Free == false)
12. {
13. painter.drawPixmap(m\_enemys[i].m\_X,m\_enemys[i].m\_Y,m\_enemys[i].m\_enemy);
14. }
15. }
16. *//加血球绘制*
17. if (h\_hearts.h\_Free == false){
18. painter.drawPixmap(h\_hearts.h\_X,h\_hearts.h\_Y,h\_hearts.h\_heart);
19. }
20. QPainter p(this);
21. QFont font("宋体",13,75);
22. p.setFont(font);
23. p.setPen(QColor("#F0FFFF"));*//字体颜色*
24. QString s = QString::number(life,10);
25. s = "剩余生命值："+ s;
26. QString title(s);
27. int widthOfTitle = p.fontMetrics().width(title);*//字符串显示的像素大小*
28. p.drawText(this->width()/2-widthOfTitle/2,100,title);
29. QPainter p2(this);
30. QFont font2("宋体",13,75);
31. p2.setFont(font2);
32. p2.setPen(QColor("#F0FFFF"));
33. QString s2 = QString::number(timer\_count,10);
34. s2 = "您已坚持："+ s2 + "秒";
35. QString title2(s2);
36. int widthOfTitle2 = p2.fontMetrics().width(title2);
37. p2.drawText(this->width()/2-widthOfTitle2/2,50,title2);
38. }

## 4 移动飞船

在本游戏中，玩家需要用鼠标移动飞船以躲避子弹，这就需要添加代码实现此功能。

1. void MainScene::mouseMoveEvent(QMouseEvent \*event)
2. {
3. int x = event->x() - m\_hero.m\_Rect.width()\*0.5; //鼠标位置 - 飞机矩形的一半
4. int y = event->y() - m\_hero.m\_Rect.height()\*0.5;
5. //使飞船只能在游戏界面内部移动
6. if(x <= 0 ){
7. x = 0;
8. }
9. if(x >= GAME\_WIDTH - m\_hero.m\_Rect.width()){
10. x = GAME\_WIDTH - m\_hero.m\_Rect.width();
11. }
12. if(y <= 0){
13. y = 0;
14. }
15. if(y >= GAME\_HEIGHT - m\_hero.m\_Rect.height()){
16. y = GAME\_HEIGHT - m\_hero.m\_Rect.height();
17. }
18. m\_hero.setPosition(x,y);
19. }

这里有两行代码：

1. int x = event->x() - m\_hero.m\_Rect.width()\*0.5;
2. int y = event->y() - m\_hero.m\_Rect.height()\*0.5;

等号右侧若不减去m\_hero.m\_Rect.width()\*0.5与m\_hero.m\_Rect.height()\*0.5，则移动飞船时会发现鼠标的光标会位于飞船的左上角，这样会显得比较别扭。减去之后会使得鼠标光标位于飞船的正中央，这样对于玩家观感会较为友好。

同时，由于飞船位置与鼠标关联在一起，当鼠标移动到界面以外的区域时，飞船会消失在界面之中。为防止这种情况的发生，即需要保持飞船在界面之内移动，所以需要对飞船坐标进行判断。上述代码的第6行至第21行有四个if判断，该目的是为了判断鼠标是否位于上下左右四个界面之外，如果在左右边界之外，则保持飞船横坐标不变；如果在上下边界之外，则保持飞船纵坐标不变。

## 5 计时器实现

本项目中共有三个定时器：m\_Timer、timer1和timer2，代码如下所示。

1. void MainScene::playGame()
2. {
3. timer\_count = 0;
4. timer\_count2 = 0;
5. m\_Timer.start();
6. *//监听定时器*
7. connect(&m\_Timer,&QTimer::timeout,[=](){
8. if (m\_flag == 0){
9. *//敌机出场*
10. enemyToScene();
11. heartToScene();
12. *//更新游戏中元素的坐标*
13. updatePosition();
14. *//重新绘制图片*
15. update();
16. collisionDetection();
17. heartcollisionDetection();
18. if (init\_flag == 1){
19. this->close();
20. *//this->destroy();*
21. Gameover \*gameover = new Gameover();
22. gameover->show();
23. timer\_count = 0;
24. timer\_flag = 0;
25. m\_Timer.stop();  *//一定要加stop，不然如果玩家选择“再玩一局”，那就是几个定时器并行*
26. timer1.stop();
27. timer2.stop();
28. }
29. }
30. });
31. *// 启动定时器,单位是毫秒*
32. if (timer\_flag == 0){
33. ++timer\_flag;
34. timer1.start(5000);
35. }
36. *// timer发出timeout（时间到了）的信号,*
37. connect(&timer1,&QTimer::timeout,[=](){
38. *//每隔5秒给屏幕上增加一个子弹*
39. addEnemyNumber();
40. });
41. timer2.start(1000);
42. connect(&timer2,&QTimer::timeout,[=](){
43. *//timer\_counter表示坚持的时间，以秒为单位*
44. timer\_count++;
45. });
46. }

### 5.1 计时器实现游戏的循环

m\_Timer的功能是启动游戏（上述代码第7行至第30行），使游戏的各部分运行起来，其本质是不断调用游戏功能函数。这里有一个标志位init\_flag，初始化为0。当init\_flag为1时，表示游戏结束，进行游戏的界面被关闭，显示“游戏结束界面”。

### 5.2 计时器实现子弹的数量随时间递增

控制子弹数量随时间增加的定时器为timer1，为上述代码的第31到40行，该定时器以五秒为周期，每五秒增加一颗子弹，直至增加到子弹设定的最大数量，以达到随游戏进行难度不断提升的目的。

### 5.3 计时器实现计时功能

实现计时功能的定时器为timer2，为上述代码的第41到46行，该定时器以秒为周期，timer\_count为记载时间的变量。

## 6 子弹功能实现

### 6.1 子弹状态位功能

2.5中提到了子弹属性中的m\_Free，该变量作为子弹的状态位，当值为true时，表示子弹的当前状态为空闲子弹，当值为false时，表示状态为非空闲，需要在屏幕中显示。下述代码即实现该过程。

首先，设置变量ENEMY\_INTERVAL的目的是控制新子弹出现的时间间隔，如果不设置此变量，那么在5.1计时器代码中，由于调用函数enemyToScene()的频率非常快，那么就会导致各个子弹出现在屏幕上时几乎没有时间间隔，屏幕上将充满了子弹。

1. void MainScene::enemyToScene()
2. {
3. m\_recorder++;
4. *//如果未到间隔时间，则直接return*
5. if(m\_recorder < ENEMY\_INTERVAL)
6. {
7. return;
8. }
9. m\_recorder = 0;
10. for(int i = 0 ; i< num\_enemy;i++)
11. {
12. if(m\_enemys[i].m\_Free)
13. {
14. *//子弹空闲状态改为false*
15. m\_enemys[i].m\_Free = false;
16. *//设置坐标*
17. m\_enemys[i].m\_X = rand() % (GAME\_WIDTH - m\_enemys[i].m\_Rect.width());
18. m\_enemys[i].m\_Y = -m\_enemys[i].m\_Rect.height();
19. break;
20. }
21. }
22. }

上述代码第17至第18行是初始化子弹坐标，保证各子弹从界面的上边界出现。

### 6.2 子弹数量随时间递增

在5.2小节的相关代码中，我们可以看到，负责控制子弹数量随时间增加的计时器timer1每5秒会执行函数addEnemyNumber()，该函数中实现将屏幕中允许出现的子弹数量加一，直至设置的子弹上限值，该部分代码如下所示。

1. void MainScene::addEnemyNumber()
2. {
3. ++add\_time;
4. *//如果子弹数量未达到子弹的最大值，则加1*
5. if (num\_enemy < ENEMY\_NUM){
6. num\_enemy = num\_enemy + 1;
7. }
8. }

### 6.3 子弹与飞船的碰撞检测

碰撞检测的本质是检测子弹矩形框和飞船矩形框是否相交，如果相交则代表碰撞，飞船生命值减一。

1. void MainScene::collisionDetection()
2. {
3. *//遍历所有非空闲的子弹*
4. for (int i = 0; i < ENEMY\_NUM; i++)
5. {
6. if (m\_enemys[i].m\_Free)
7. {
8. *//空闲子弹 跳转下一次循环*
9. continue;
10. }
11. *//如果子弹矩形框和飞船矩形框相交，发生碰撞，则变为空闲状态即可*
12. if (m\_enemys[i].m\_Rect.intersects(m\_hero.m\_Rect))
13. {
14. --life;
15. cout << "Current Live: "  << life <<"\n";
16. cout << "Spend time: " << timer\_count << "\n";
17. timer\_count2 = timer\_count - 5;
18. h\_hearts.h\_Free = true; *//如果加血球不清零的话，玩家可以通过主动撞击子弹清空屏幕，并立即获取生命值，从而无限游玩*
19. if (life == 0){
20. life = -1;
21. init\_flag = 1;
22. break;
23. *//endGame();*
24. }
25. *//清空屏幕上的子弹*
26. refreshEnemyNumber();
27. }
28. }
29. }

### 6.4 清空屏幕子弹

当飞船碰撞子弹后，飞船生命值减一，同时需要清空屏幕上的子弹，因为如果不将子弹清空，那么当飞船重新显示在界面中时，如果重新出现的位置附近有多个子弹，则很容易出现再次碰撞丢失生命值的情况，导致接连丢命，这不利于玩家的游戏体验。

同时，这也是为什么需要第18行h\_hearts.h\_Free = true，该行代码的意义是：当飞船触碰到子弹时，如果此时界面中也存在加血球，那么此时除了情况屏幕中的子弹，也需要清空屏幕中的加血球。否则，当子弹清空后，只留下加血球，那么玩家在飞船重新显示在界面中后将很轻易与加血球进行触碰，实现生命值加一，那么游戏将会无限进行下去。

清空屏幕子弹实际上是将所有子弹的状态位m\_Free全部设置为true。

1. *//清空屏幕上的子弹*
2. void MainScene::refreshEnemyNumber()
3. {
4. *//将所有子弹的状态都值为空闲*
5. for(int i = 0 ; i< ENEMY\_NUM;i++)
6. {
7. m\_enemys[i].m\_Free = true;
8. }
9. enemyToScene();
10. }

## 7 加血球功能实现

### 7.1 加血球状态位功能

与子弹状态位类似，加血球的状态位为h\_Free，当值为true时，加血球不显示，当值为false时，表示状态为非空闲，需要在屏幕中显示。下述代码实现该过程。

1. void MainScene::heartToScene()
2. {
3. *//30秒后开始出现加血球，因为开始时子弹数量太少，比较简单*
4. *//每一次加血球的出现都与上一次加血的时间隔15秒的倍数再出现*
5. if (timer\_count >= 30 && (timer\_count - timer\_count2) % 15 == 0 && timer\_count - timer\_count2 >= 15){
6. if(h\_hearts.h\_Free)
7. {
8. *//子弹空闲状态改为false*
9. h\_hearts.h\_Free = false;
10. *//设置坐标*
11. h\_hearts.h\_X = rand() % (GAME\_WIDTH - h\_hearts.h\_Rect.width());
12. h\_hearts.h\_Y = -h\_hearts.h\_Rect.height();
13. }
14. }
15. else{
16. return ;
17. }
18. }

### 7.2 加血球与飞船的碰撞检测

1. void MainScene::heartcollisionDetection()
2. {
3. if (h\_hearts.h\_Free)
4. {
5. return ;
6. }
7. *//如果加血球矩形框和飞船矩形框相交，发生碰撞，变为空闲状态即可*
8. if (h\_hearts.h\_Rect.intersects(m\_hero.m\_Rect))
9. {
10. timer\_count2 = timer\_count; *//记录下此次加血的时间，15秒后加血球再次出现*
11. h\_hearts.h\_Free = true;
12. ++life;
13. cout << "Current Live: "  << life <<"\n";
14. cout << "Spend time: " << timer\_count << "\n";
15. }
16. }

## 8 Mysql部分

### 8.1 连接数据库

Qt可以和Mysql结合实现一些功能的开发。本项目之所以使用Qt5.12.3，是因为Qt5.12.3是最后提供Mysql数据库插件的版本，往后的版本需要自行编译对应的Mysql数据库插件，而Qt5.12.3只需导入数据库相关头文件即可进行数据库操作。

1. QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");     *//驱动*
2. QStringList list = QSqlDatabase::drivers();
3. qDebug() << list;
4. *// 数据库地址*
5. db.setHostName("localhost");
6. *// 数据库名，这里选择之前所做项目的一个数据库，就没有新建立*
7. db.setDatabaseName("student\_grades");
8. db.setUserName("root");
9. db.setPassword("123456");
10. db.setPort(3306);                     *// 端口，默认是3306*
11. *// 打开数据库*
12. if (!db.open()) { *// 数据库打开失败*
13. qDebug() << "失败原因" << db.lastError().text();
14. } else {
15. qDebug() << "连接成功!!!!";
16. }

### 8.2 添加数据

1. QSqlQuery query;
2. QString sql = QString("insert into timer\_test1 values(%1)").arg(timer\_count);
3. db.transaction();
4. bool flag = query.exec(sql);
5. if (flag){
6. db.commit();
7. }
8. else{
9. db.rollback();
10. }
11. db.close();

### 8.3 查询数据

变量ans类型为vector，用于存储查询数据后的结果，即所有历史游戏成绩。

1. QSqlQuery query;
2. QString sql;
3. sql = "select \* from timer\_test1";
4. query.exec(sql);
5. vector<int> ans;
6. while(query.next()){
7. qDebug() << query.value(0).toInt();
8. ans.push\_back(query.value(0).toInt());
9. }
10. db.close();

## 9 “开始游戏”界面

界面如图3所示。



图 3 “开始游戏”界面

点击“开始游戏”按钮，进入游戏主界面。

1. void Startgame::on\_pushButton\_clicked()
2. {
3. MainScene \*m = new MainScene();
4. this->close();
5. m->show();
6. }

点击“历史记录”按钮，可以查看过往进行游戏的成绩，按从高至低的顺序，可以查看前六高的成绩记录。

1. void Startgame::on\_pushButton\_2\_clicked()
2. {
3. Record \*r = new Record();
4. r->show();
5. }

首先，将8.3代码中的vector类型变量ans按从高至低排序，然后，选择前六高的成绩，以界面的形式展示，如图4所示。

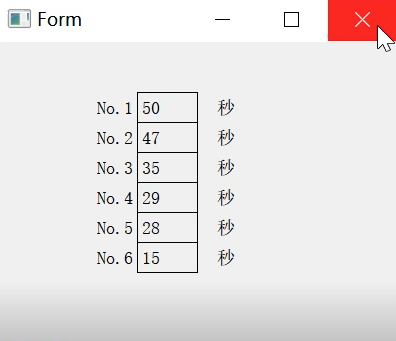


图 4 历史成绩榜

1. for (int i = 0; i < 6; ++i){
2. ans.push\_back(0);
3. }
4. sort(ans.begin(),ans.end());
5. reverse(ans.begin(),ans.end());
6. ui->label->setText(QString::number(ans[0]));
7. ui->label\_2->setText(QString::number(ans[1]));
8. ui->label\_3->setText(QString::number(ans[2]));
9. ui->label\_4->setText(QString::number(ans[3]));
10. ui->label\_5->setText(QString::number(ans[4]));
11. ui->label\_6->setText(QString::number(ans[5]));

之所以要在这里设置一个循环（第1至3行），是因为，如果玩家游玩的次数并没有达到六次，那么ans的长度将小于6，此时如果直接取数将会导致越界，所以需要先在vector中存入6个0。如果不这么做，也有替代的做法，不过这将比较繁琐，以下是我原先的代码，用了六次if判断。

1. sort(ans.begin(),ans.end());
2. reverse(ans.begin(),ans.end());
3. if (ans.size() == 0){
4. ui->label->setText(QString::number(0));
5. ui->label\_2->setText(QString::number(0));
6. ui->label\_3->setText(QString::number(0));
7. ui->label\_4->setText(QString::number(0));
8. ui->label\_5->setText(QString::number(0));
9. ui->label\_6->setText(QString::number(0));
10. }
11. else if (ans.size() == 1){
12. ui->label->setText(QString::number(ans[0]));
13. ui->label\_2->setText(QString::number(0));
14. ui->label\_3->setText(QString::number(0));
15. ui->label\_4->setText(QString::number(0));
16. ui->label\_5->setText(QString::number(0));
17. ui->label\_6->setText(QString::number(0));
18. }
19. else if (ans.size() == 2){
20. ui->label->setText(QString::number(ans[0]));
21. ui->label\_2->setText(QString::number(ans[1]));
22. ui->label\_3->setText(QString::number(0));
23. ui->label\_4->setText(QString::number(0));
24. ui->label\_5->setText(QString::number(0));
25. ui->label\_6->setText(QString::number(0));
26. }
27. else if (ans.size() == 3){
28. ui->label->setText(QString::number(ans[0]));
29. ui->label\_2->setText(QString::number(ans[1]));
30. ui->label\_3->setText(QString::number(ans[2]));
31. ui->label\_4->setText(QString::number(0));
32. ui->label\_5->setText(QString::number(0));
33. ui->label\_6->setText(QString::number(0));
34. }
35. else if (ans.size() == 4){
36. ui->label->setText(QString::number(ans[0]));
37. ui->label\_2->setText(QString::number(ans[1]));
38. ui->label\_3->setText(QString::number(ans[2]));
39. ui->label\_4->setText(QString::number(ans[3]));
40. ui->label\_5->setText(QString::number(0));
41. ui->label\_6->setText(QString::number(0));
42. }
43. else if (ans.size() == 5){
44. ui->label->setText(QString::number(ans[0]));
45. ui->label\_2->setText(QString::number(ans[1]));
46. ui->label\_3->setText(QString::number(ans[2]));
47. ui->label\_4->setText(QString::number(ans[3]));
48. ui->label\_5->setText(QString::number(ans[4]));
49. ui->label\_6->setText(QString::number(0));
50. }
51. else{
52. ui->label->setText(QString::number(ans[0]));
53. ui->label\_2->setText(QString::number(ans[1]));
54. ui->label\_3->setText(QString::number(ans[2]));
55. ui->label\_4->setText(QString::number(ans[3]));
56. ui->label\_5->setText(QString::number(ans[4]));
57. ui->label\_6->setText(QString::number(ans[5]));
58. }

## 10 “游戏结束”界面

“游戏结束”界面如图4所示。

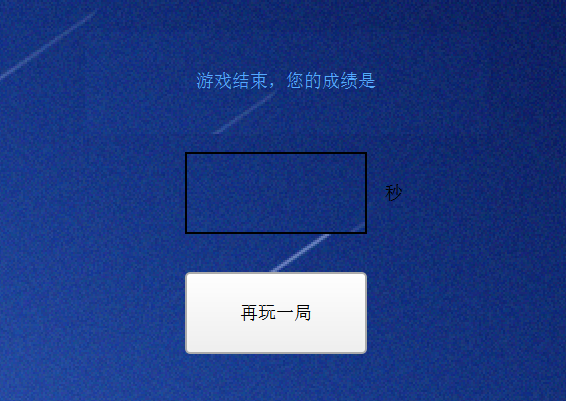


图 5 “游戏结束”界面

每局游戏结束后，成绩将会显示在中心的文本框中。

点击“再玩一局”按钮，将会回到“开始游戏”的界面。

1. void Gameover::on\_pushButton\_clicked()
2. {
3. ++num;
4. Startgame \*s = new Startgame();
5. this->close();
6. s->show();
7. }

## 11 奇怪的Bug

### 11.1 在顶部平移的子弹

在2.5小节，为子弹随机设置速度的代码里，有一个条件判断while(m\_VY <= 1)（成员函数代码的第17行），其意义在于，如果随机生成的Y轴方向的速度不大于1，则重新设置速度。

起初，这里是没有这一行判断，但是在运行游戏时发现有时会出现如图5所示的情况，即有些子弹一直在顶部平移。一开始，我以为是出现了在Y轴方向速度为0的子弹，于是先添加了判断语句while(m\_VY == 0)，但是依然会出现在顶部平移的子弹。

为解决该bug，我将每个子弹的Y轴速度都打印出来，如图6所示，发现每当出现在顶部平移的子弹时，会出现Y轴方向速度小于1的子弹。于是我觉得只有当Y轴方向速度大于一时，才不会出现顶部平移的子弹，所以我将判断语句修改为while(m\_VY <= 1)，解决了这一奇怪的bug。



图 6 在顶部平移的子弹



图 7 打印子弹的速度

### 11.2 并行的计时器

一开始，我是将计时器timer1、timer2定义在mainscene.cpp的playGame()函数中，代码如下所示。

1. QTimer \*timer1 = new QTimer(this);
2. *// timer发出timeout（时间到了）的信号,*
3. connect(&timer1,&QTimer::timeout,[=](){
4. addEnemyNumber();
5. });
6. QTimer \*timer2 = new QTimer(this);
7. timer2.start(1000);
8. connect(&timer2,&QTimer::timeout,[=](){
9. timer\_count++;
10. });

问题出现在，如果当一局游戏结束，出现“游戏结束”界面时，玩家选择“再玩一局”，那么在第二局中，如果玩家总共坚持了t秒，那么在“游戏结束”界面中显示成绩的方框内，其数值为2t，即显示的成绩与实际的时间不相符。而且，如果玩家继续选择“再玩一局”，那么往后成绩与时间时间的倍数将逐次增加。

经查阅资料，终于发现这应该是计时器并行所导致的。由于游戏结束时，界面只是从游戏界面跳转至“游戏结束”界面，而跳转的实质是将游戏界面暂时关闭，同时将“游戏结束”界面显示出来，因此计时器变量timer1和timer2并未被销毁。而当玩家选择“再玩一局”后，程序会再次执行一下两行代码

QTimer \*timer1 = new QTimer(this);

QTimer \*timer2 = new QTimer(this);

而每执行一次这两行代码，都会给给计时器增加一个新的链接函数，而原来的链接函数依然在。这意味着，如果玩家第N次选择“再玩一局”，那么计时器就有N个链接函数，当计时器timeout时就会并行地执行这些函数。

为解决此bug，我将timer1和timer2的定义放在了头文件mainscene.h中，同时，再每一局结束时，将其stop()，避免了反复定义带来的重复链接与并行计时问题。

**参考资料**

[1]C++入门——实现“坚持一百秒”游戏[OL].(2021-10-21). https://blog.csdn.net/beilizhang/article/details/120888569.