中国矿业大学计算机学院

2019 级本科生课程设计报告

课程名称 程序设计综合实践

报告时间 2021.1.6

学生姓名 李 春 阳

学　　号 10193856

专　　业 信息安全 2019-1 班

任课教师 刘 佰 龙

目　录

目录

实验一 多彩计算器

1.1.程序介绍

该程序为名称为多彩计算器，设计的思路为仿照 window10 官方的计算器，用 Qt 框架配合 c++语言来完成。在原有计算器加减乘除的功能下扩展为支持浮点运算，进一步利用栈来解决运算优先级的问题，在页面引入 e，π 等常见的数学字符,还扩展了乘方和进制转换的功能。为了更好的呈现一个美好的 UI 界面，在保证计算器严谨的同时，添加彩色触显，来增强用户体验感。

1.2.操作说明

1.2.1 清除和退格操作

这里先输入 20201231，如图 1-1 所示，再进行退格操作（按钮 C），后显示如图 1-2 所示，再进行清空操作（按钮 AC），效果如图 1-3 所示。

图 1 清除和退格展示图

1.2.2 加减乘除操作

这里为了一次性检验加减乘除，先输入 8*2/4+5-2 如图 2-1 所示，运算结果应该为 7，按下等于检验结果如图 2-2 所示。

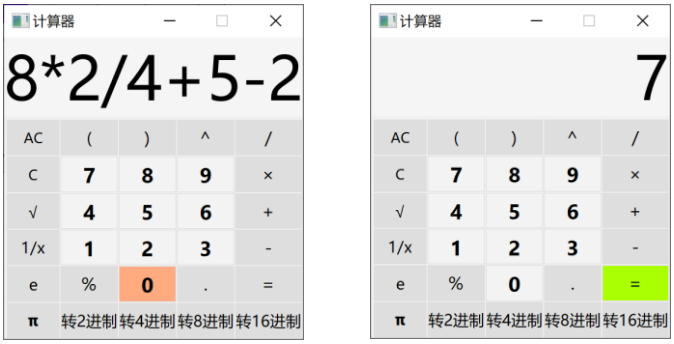


图 2 加减乘除运算图

1.2.3 扩展高级操作

1. 乘方操作

这里输入 5^2 来进行验证，结果如图 3 所示。

图 3 乘方操作

2. 分数操作

这里输入 25 再按（1/x 按钮）来进行验证，结果如图 4 所示。

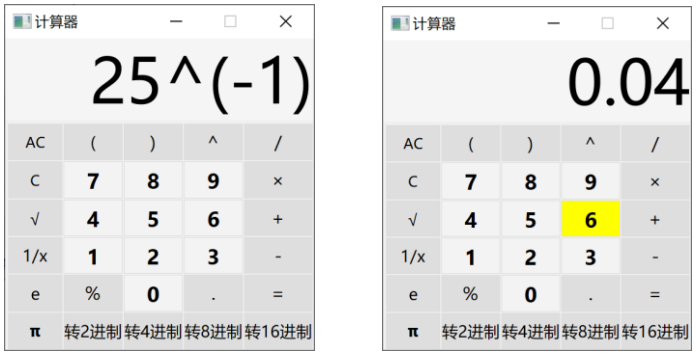


图 4 分数操作

3. 取余操作

这里输入 25%4 来进行验证，结果如图 5 所示。

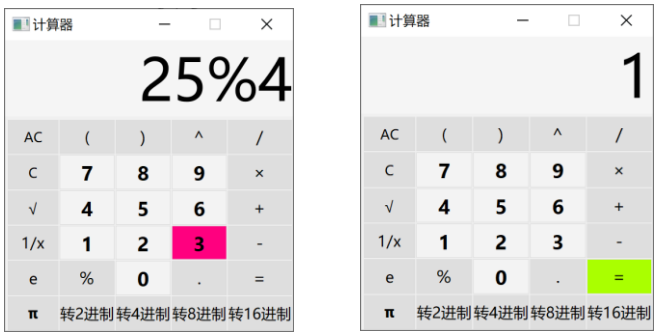


图 5 取余操作

4. 根号操作

这里输入 25 再按（√ 按钮）来进行验证，结果如图 6 所示。

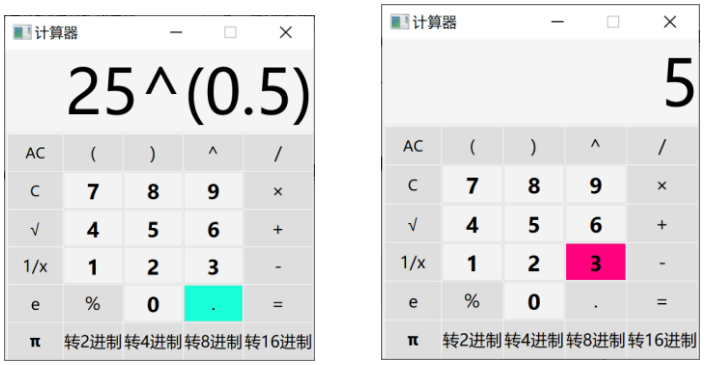


图 6 根号操作

1.2.4 特色字符展示

1. 圆周率 π

这里输入 π*2 来进行验证，结果如图 7 所示。

图 7 圆周率

2. 自然指数 e

这里输入 e*2 来进行验证，结果如图 8 所示。

图 8 自然指数

1.2.5 优先级与小数展示

这里输入（2.5+1）*2 来进行验证，结果如图 9 所示。

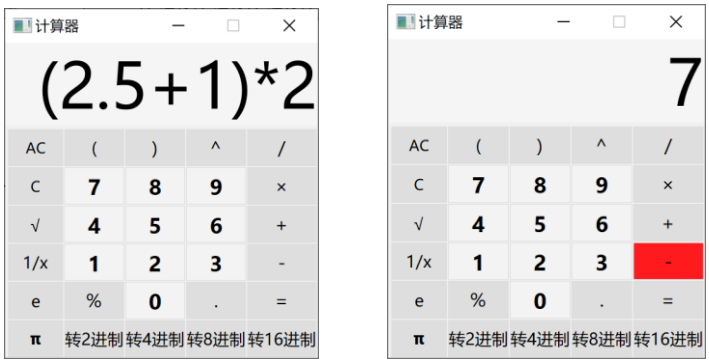


图 9 优先级展示

1.2.6 进制转换

这里我们输入 10 为例，输出结果如图 10 所示.

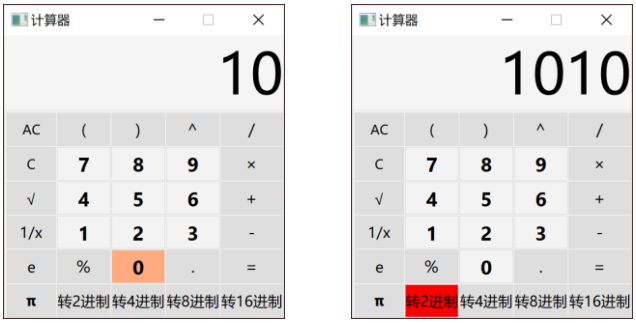


图 10 进制转换

1.3.设计理念

1.3.1 设计目标

该项目是想设计一个支持连续计算的四则运算计算器，通过点击相应的数字

按钮和运算按钮，输入并完成如 4*6+3、或者取倒数等等类似的计算，并将运算结果显示输出在文本框中，同时此计算器也具备清空、退格等其他功能，页面采用灰白配色，在按动时有不同颜色彩蛋。其运行界面如图 11 所示。

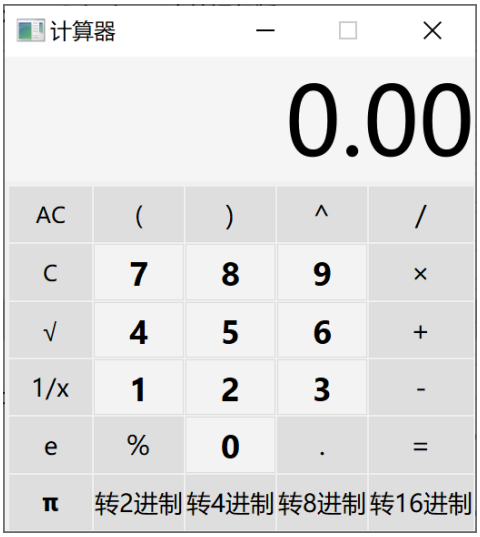


图 11 计算器页面

1.3.2 设计分析和算法分析

1.3.2.1 页面搭建

这里采用 Qt 框架自带的 UI 界面进行美化排布，效果如下图所示，利用 UI 自动生成按钮等类，在页面美化中，添加样本样式，利用 Qt 自带的 QSS 进行美化（操作如前端 CSS 类似）。

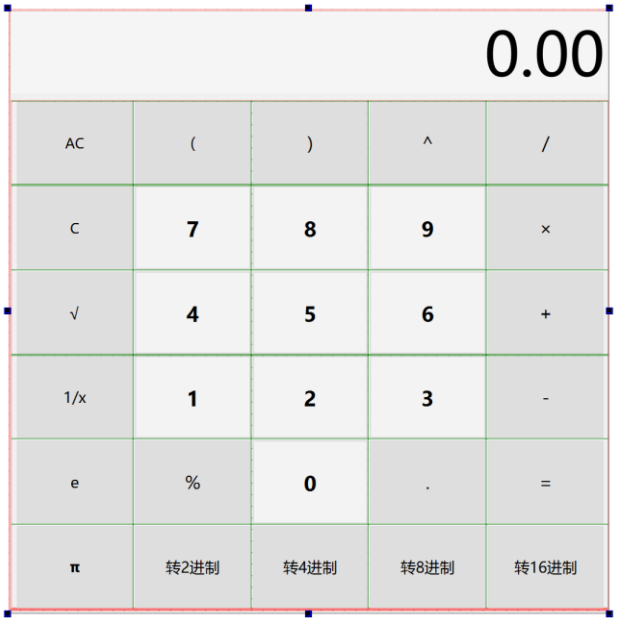


图 12UI 界面美化

1.3.2.2 按钮绑定

利用 UI 生成的类，添加槽函数进行关联绑定，生成类如图 13 所示，在 mainwindow.cpp 析构函数中进行绑定操作。代码如下：

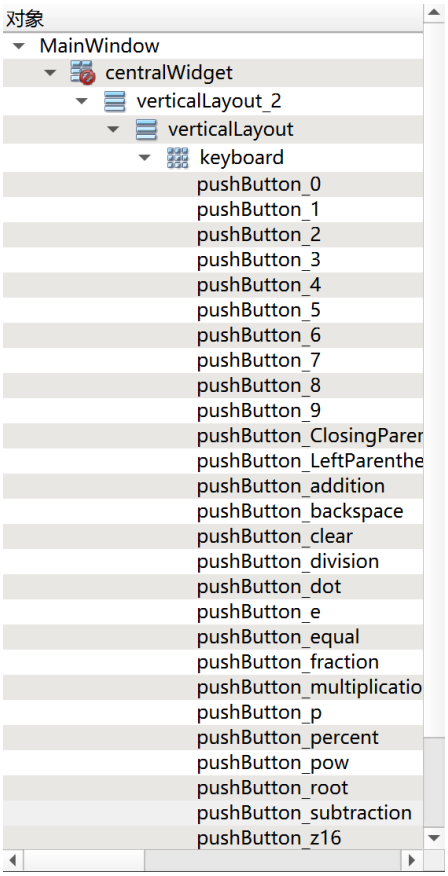


图 13 按钮类

```
    ui->setupUi(this);
    //清空a，b
    a.clear();
    b.clear();

    //绑定按键0 与处理函数
        connect(ui->pushButton_0,&QPushButton::clicked,[=](){btn_logic(0);});
        //绑定按键1 与处理函数
        connect(ui->pushButton_1,&QPushButton::clicked,[=](){btn_logic(1);});
        //绑定按键2 与处理函数
        connect(ui->pushButton_2,&QPushButton::clicked,[=](){btn_logic(2);});
        //绑定按键3 与处理函数
        connect(ui->pushButton_3,&QPushButton::clicked,[=](){btn_logic(3);});
        //绑定按键4 与处理函数
        connect(ui->pushButton_4,&QPushButton::clicked,[=](){btn_logic(4);});
        //绑定按键5 与处理函数
        connect(ui->pushButton_5,&QPushButton::clicked,[=](){btn_logic(5);});
        //绑定按键6 与处理函数
        connect(ui->pushButton_6,&QPushButton::clicked,[=](){btn_logic(6);});
        //绑定按键7 与处理函数
        connect(ui->pushButton_7,&QPushButton::clicked,[=](){btn_logic(7);});
        //绑定按键8 与处理函数
        connect(ui->pushButton_8,&QPushButton::clicked,[=](){btn_logic(8);});
        //绑定按键9 与处理函数
        connect(ui->pushButton_9,&QPushButton::clicked,[=](){btn_logic(9);});
        //绑定按键点与处理函数
        connect(ui->pushButton_dot,&QPushButton::clicked,[=](){btn_logic(0,".");});

        //绑定按键+ 与处理函数
        connect(ui->pushButton_addition,&QPushButton::clicked,[=](){btn_logic(0,"+");});
        //绑定按键- 与处理函数
        connect(ui->pushButton_subtraction,&QPushButton::clicked,[=](){btn_logic(0,"-
");});
        //绑定按键* 与处理函数

connect(ui->pushButton_multiplication,&QPushButton::clicked,[=](){btn_logic(0,"*");});
        //绑定按键/ 与处理函数
        connect(ui->pushButton_division,&QPushButton::clicked,[=](){btn_logic(0,"/");});
        //绑定按键% 与处理函数
        connect(ui->pushButton_percent,&QPushButton::clicked,[=](){btn_logic(0,"%");});
        //绑定按键^ 与处理函数
```

```
        connect(ui->pushButton_pow,&QPushButton::clicked,[=](){btn_logic(0,"^");});

        //绑定按键(与处理函数

connect(ui->pushButton_LeftParenthesis,&QPushButton::clicked,[=](){btn_logic(0,"(");});
        //绑定按键)与处理函数

connect(ui->pushButton_ClosingParenthesis,&QPushButton::clicked,[=](){btn_logic(0,")");});

        //绑定按键 e 与处理函数
        connect(ui->pushButton_e,&QPushButton::clicked,[=](){btn_logic(0,"e");});
        //绑定按键 p 与处理函数
        connect(ui->pushButton_p,&QPushButton::clicked,[=](){btn_logic(0,"p");});
        //绑定按键根号与处理函数
        connect(ui->pushButton_root,&QPushButton::clicked,[=](){btn_logic(0,"root");});
        //绑定按键 1/x 与处理函数

connect(ui->pushButton_fraction,&QPushButton::clicked,[=](){btn_logic(0,"pushButton_fracti
on");});

        //绑定按键 AC 与处理函数
        connect(ui->pushButton_clear,&QPushButton::clicked,[=](){
            a.clear();
            chh.clear();
            b.clear();
            ui->lineEdit->setText(a);
        });

        //绑定按键退格-> 与处理函数
        connect(ui->pushButton_backspace,&QPushButton::clicked,[=](){
            //删除 a.pop
            a.chop(1);
            chh.chop(1);
            ui->lineEdit->setText(a);
        });

        //绑定按键转 2 进制与处理函数
        connect(ui->pushButton_z2,&QPushButton::clicked,[=](){

            int number = a.toInt();
            a.clear();
            decimalToAny(number,2);
```

```
        ui->lineEdit->setText(a);
    });
    //绑定按键转 4 进制与处理函数
    connect(ui->pushButton_z4,&QPushButton::clicked,[=](){

        int number = a.toInt();
        a.clear();
        decimalToAny(number,4);
        ui->lineEdit->setText(a);
    });
    //绑定按键转 8 进制与处理函数
    connect(ui->pushButton_z8,&QPushButton::clicked,[=](){

        int number = a.toInt();
        a.clear();
        decimalToAny(number,8);
        ui->lineEdit->setText(a);
    });
    //绑定按键转 16 进制与处理函数
    connect(ui->pushButton_z16,&QPushButton::clicked,[=](){

        int number = a.toInt();
        a.clear();
        decimalToAny(number,16);
        ui->lineEdit->setText(a);
    });


    //绑定按键= 与处理函数
    connect(ui->pushButton_equal,&QPushButton::clicked,[=](){den_logic();});
```

代码块 1 析构函数代码

1.2.2.3 进制转换算法

构建 cimalToAny 函数，输入转换的 10 进制数 n，和转换进制 d，利用辗转相除法求余数来构建。例如以 2 进制为例

除 2 取余法，即每次将整数部分除以 2，余数为该位权上的数，而商继续除以 2，余数又为上一个位权上的数，这个步骤一直持续下去，直到商为 0 为止，

最后读数时候,从最后一个余数读起,一直到最前面的一个余数。算法代码如下:

```
void MainWindow::decimalToAny(int n,int d)//10 进制转换成任意进制
{
    if(n==0)
        return ;
    else
    {
        decimalToAny(n/d,d);
        if(d>=10)//如果是 10 进制以上
        {
            if(n%d>=10)
            {
                //printf("%c",(char)((n%d-10)+'A'));
                a += char((n%d-10)+'A');
            }
            else//如果余数小于 10，则直接输出
            {
                //printf("%d",n%d);
                a += QString::number(n%d);
            }
        }
        //如果进制小于 10，不会有字母的问题
        else
        {
            //printf("%d",n%d);
            a += QString::number(n%d);
        }

    }
}
```

代码块 2 进制转换代码

1.2.2.4 字符转换

根据按钮调用此函数 btn_logic,对与屏幕进行输入,输入类型为 QString 类,改变输入并刷新屏幕。代码如下:

```
void MainWindow::btn_logic(int x , QString i)
{
    if(i != " " )
    {
        if(i == "pushButton_fraction")
        {
            a += "^(-1)";
            chh += "^(-1)";
        }
        else if (i == "root")
        {
            a += "^(-1)";
            chh += "^(-1)";
        }
        else if (i == "p")
        {
            a += "π";
            chh += "p";
        }
        else
        {
            a += i;
            chh += i;
        }
    }
    else
    {
        a += QString::number(x);
        chh += QString::number(x);
    }
    ui->lineEdit->setText(a);

}
```

代码块 3btn_logic 函数代码

1.2.2.5 逻辑运算代码

这里采用面向对象的方法，构建一个运算类 calc 和检验类 check，通过构建俩个栈，一个操作数一个运算符，将输入的中序运算改为后序运算并计算结果输

出，下代码块相关实现：

```
#ifndef CLAC_H
#define CLAC_H
#include <iostream>
#include <fstream>
#include <stack>
#include <cctype>
#include <sstream>
#include <iomanip>
#include <cmath>

using namespace std;

class calc
{
private:
    stack<char> operators;
    stack<double> operands;

    char getSign(void)
    {
        return operators.top();
    }

    double getNum(void)
    {
        return operands.top();
    }

    void popSign(void)
    {
        operators.pop();
    }

    void popNum(void)
    {
        operands.pop();
    }

    double popAndGetNum(void)
    {
        double num = getNum();
```

```
        popNum();
        return num;
    }

    char popAndGetSign(void)
    {
        char sign = getSign();
        popSign();
        return sign;
    }
public:
    double final_result;

    void push(double num)
    {
        operands.push(num);
    }

    void push(char sign)
    {
        operators.push(sign);
    }

    char get(void)
    {
        return getSign();
    }

    void pop(void)
    {
        popSign();
    }

    double result(void)
    {
        if (!operands.empty())
            return getNum();
        return 0.0;
    }

    void calculate(void)
    {
```

```
        double post = popAndGetNum();
        char sign = popAndGetSign();
        double pre = popAndGetNum();
        double result = 0.0;
        switch (sign)
        {
        case '+':
            result = pre + post;
            break;
        case '-':
            result = pre - post;
            break;
        case '*':
            result = pre * post;
            break;
        case '/':
            if (fabs(post) < 1e-6)
            {
                calc::writeResult("#1");
                exit(EXIT_SUCCESS);
            }
            else
                result = pre / post;
            break;
        case '^':
            result = pow(pre, post);
            break;
        case '%':
            result = static_cast<int>(pre) % static_cast<int>(post);
            break;
        }
        //cout << result << endl;
        final_result = result;
        push(result);
    }

    bool canCalculate(char sign)
    {
        if (sign == '(' || sign == '[' || sign == '{' || operators.empty())
            return false;
        char t = getSign();
        if (t == '^')
```

```
            return true;
        switch (t)
        {
        case '+':
        case '-':
            return sign == '+' || sign == '-';
        case '*':
        case '/':
        case '%':
            return sign == '+' || sign == '-' || sign == '*' || sign == '/'
|| sign == '%';
        }
        return false;
    }

    bool empty(void)
    {
        return operators.empty();
    }

    static void writeResult(const string& s)
    {
        ofstream out;
        out.open("result.txt");
        out << s;
        out.close();
    }

    void writeResult(void)
    {
        ofstream out;
        out.open("result.txt");
        if (result() < 1e15)
        {
            out << fixed;
            out << setprecision(12);
        }
        out << result();
        out.close();
    }
};
```

```
class check
{
private:
    string s;
    size_t len;
    char c;
    bool valid(void)
    {
        if (isSignOrDot(0) || isRightBrace(0))
            return false;
        len = s.size();
        stack<char> braces;
        for (size_t i = 0; i < len; ++i)
        {
            if (isLeftBrace(i))
            {
                if (isSignOrDot(i + 1))
                {
                    if (s[i + 1] != '-' && s[i + 1] != '+')
                        return false;
                }
                if (isRightBrace(i + 1))
                    return false;
                braces.push(s[i]);
            }
            else if (isRightBrace(i))
            {
                if (isDot(i + 1) || isDigit(i + 1) || isLeftBrace(i + 1))
                    return false;
                if (isRightBrace(i + 1))
                {
                    stack<char> braces_copy(braces);
                    if (braces_copy.empty())
                        return false;
                    braces_copy.pop();
                    if (braces_copy.empty())
                        return false;
                }
```

```
                if (braces.empty())
                    return false;
                char brace = braces.top();
                if ((brace == '(' && s[i] != ')') || (brace == '[' &&
s[i] != ']') || (brace == '{' && s[i] != '}'))
                    return false;
                braces.pop();
            }
            else if (isSign(i))
            {
                if (isSign(i + 1) || isDot(i + 1) || isRightBrace(i + 1))
                    return false;
            }
            else if (isDot(i))
            {
                if (isSignOrDot(i + 1) || isBrace(i + 1))
                    return false;
            }
            else if (isDigit(i))
            {
                if (isRightBrace(i + 1))
                {
                    if (braces.empty())
                        return false;
                    char brace = braces.top();
                    if ((brace == '(' && s[i + 1] != ')') || (brace == '['
&& s[i + 1] != ']') || (brace == '{' && s[i + 1] != '}'))
                        return false;
                }
            }
        }
        return braces.empty();
    }

    bool set(size_t i)
    {
        if (i >= len)
            return false;
        c = s[i];
        return true;
    }
```

```
    bool isSign(size_t i)
    {
        if (set(i))
            return c == '+' || c == '-' || c == '*' || c == '/' || c == '^'
|| c == '%';
        return false;
    }

    bool isDot(size_t i)
    {
        if (set(i))
            return c == '.';
        return false;
    }

    bool isBrace(size_t i)
    {
        return isLeftBrace(i) || isRightBrace(i);
    }

    bool isLeftBrace(size_t i)
    {
        if (set(i))
            return c == '(' || c == '[' || c == '{';
        return false;
    }

    bool isRightBrace(size_t i)
    {
        if (set(i))
            return c == ')' || c == ']' || c == '}';
        return false;
    }

    bool isSignOrDot(size_t i)
    {
        return isSign(i) || isDot(i);
    }

    bool isDigit(size_t i)
    {
        if (set(i))
```

```
            return isdigit(c);
        return false;
    }

public:

    check() {}
    bool valid(const string& s)
    {
        this->s = s;
        return valid();
    }

    string getResult(void)
    {
        size_t len = s.size();
        for (size_t i = 0; i < len; ++i)
        {
            if (s[i] == '(' && (s[i + 1] == '-' || s[i + 1] == '+'))
                s.insert(i + 1, "0");
        }
        return s;
    }
};









    #endif // CLAC_H
```

代码块 4 运算类代码

1.2.2.6 等于函数

在按下等于后执行这个 den_logic 函数，将 QString 类型转换为 string 类，并将运算的字符串进行分割，利用 isdigit（）函数来进行有效的判断分割 double 类数字，将运算符压入运算符栈中，并进行栈内运算，调用逻辑运算类进行运算，

输入给 QString 并刷新显示。代码如下：

```
void MainWindow::den_logic()
{
    string s = chh.toStdString();
    a.clear();
    check chk;
    if (chk.valid(s))
        s = chk.getResult();
    else
    {
        calc::writeResult("#");
        a += "输入错误";
    }

    size_t len = s.size();
    calc c;
    for (size_t i = 0; i < len; ++i)
    {
        if (isdigit(s[i]))
        {
            double num;
            size_t i1 = i + 1;
            while (i1 < len && (isdigit(s[i1]) || s[i1] == '.'))
                ++i1;
            istringstream input(s.substr(i, i1));
            input >> num;
            i = i1 - 1;
            c.push(num);
        }
        else if (s[i] == '}' || s[i] == ']' || s[i] == ')')
        {
            char sign;
            char start = (s[i] == '}' ? '{' : (s[i] == ']' ? '[' : '('));
            while ((sign = c.get()) != start)
                c.calculate();
            c.pop();
        }
        else if(s[i] == 'p' || s[i] == 'e')
        {
            double number;
            if(s[i] == 'e')
            {
```

```
                number = 2.71828182846;
                c.push(number);
            }
            else
            {
                number = 3.14159265;
                c.push(number);
            }
        }
        else                            //s[i] is [ ( { + - * / ^ %
        {
            while (c.canCalculate(s[i]))
                c.calculate();
            c.push(s[i]);
        }
    }
    while (!c.empty())
    {
        c.calculate();
    }
    a += QString::number(c.final_result);

    ui->lineEdit->setText(a);
}
```

1.3.3 类图关系

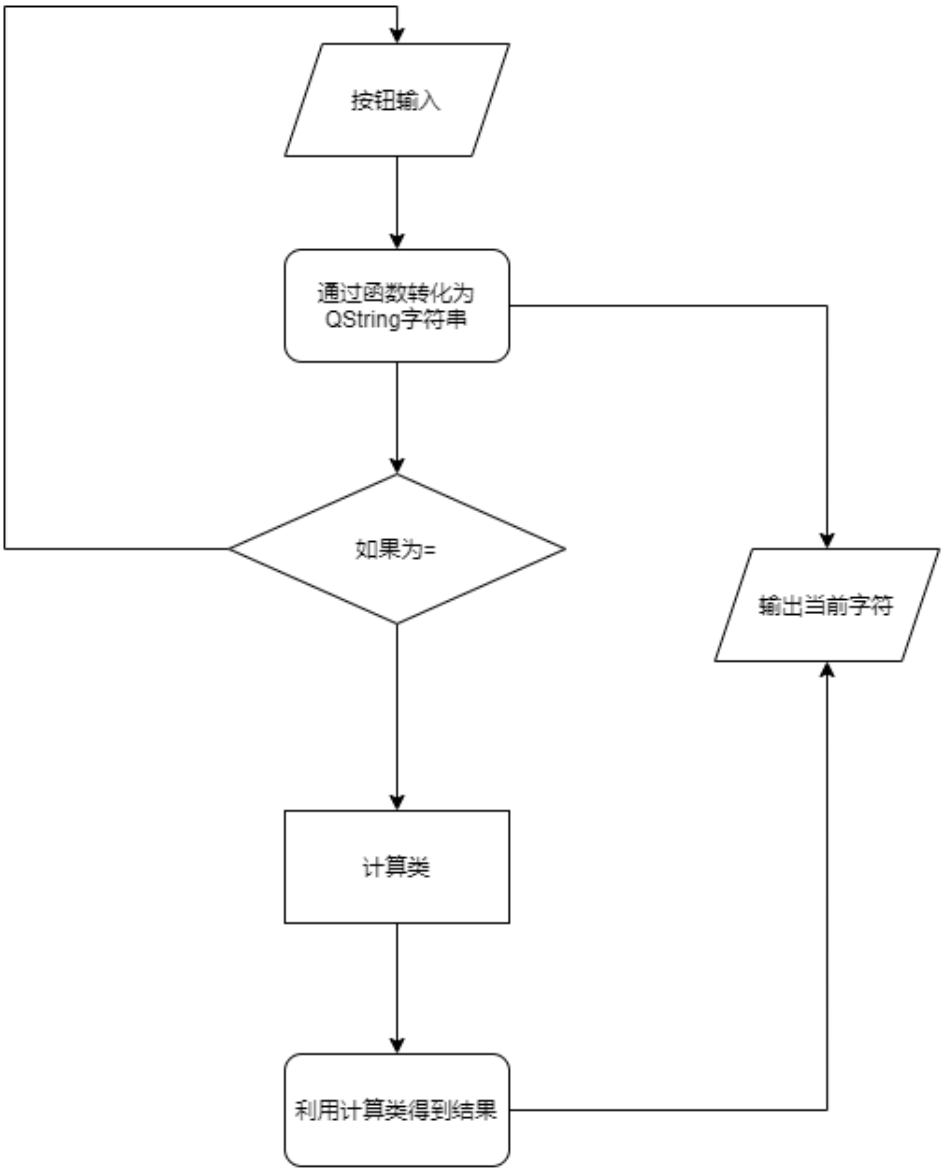


图 14 类关系流程图

1.4.程序展示

一些操作实际展示，展示效果如下图：

图 15 成果展示图

1.5.总结思考

通过使用 Qt 应用框架实现了人机交互界面的计算器，采用 Qt 信号槽机制实现计算器的加减乘除及带括号的四则混合运算功能。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。

实验二 利率计算器

2.1.程序介绍

该程序为名称为利率计算器，设计的思路为仿照招商银行在线利率计算器，用 Qt 框架配合 c++语言来完成。这里实现了等额本息和等额本金俩种贷款模式，只需要输入贷款年限，贷款金额，贷款利率就可以进行计算出月均还款和利息总额。为了更好的呈现一个美好的 UI 界面，在保证计算器严谨的同时，添加彩色触显，来增强用户体验感。

2.2.操作说明

2.2.1 等额本息

等额本息是指一种贷款的还款方式，指在还款期内，每月偿还同等数额的贷款(包括本金和利息)。这里选择等额本息，以 20 年贷款 20 万.利率 5%计算，结果如图 1 所示

图 16 等额本息展示

2.2.2 等额本金

等额本金是指一种贷款的还款方式，是在还款期内把贷款数总额等分，每月偿还同等数额的本金和剩余贷款在该月所产生的利息，这样由于每月的还款本金

额固定，而利息越来越少，借款人起初还款压力较大，但是随时间的推移每月还款数也越来越少。这里选择等额本金，以 20 年贷款 20 万.利率 5%计算，结果如图 2 所示

图 17 等额本金展示

2.3.设计理念

2.3.1 设计目标

该项目是想设计一个利率计算的程序，这里实现了等额本息和等额本金俩种贷款模式，分别对应不同的需求，在程序中只需要输入贷款年限，贷款金额，贷款利率就可以进行计算出月均还款和利息总额。其运行界面如图 3 所示。

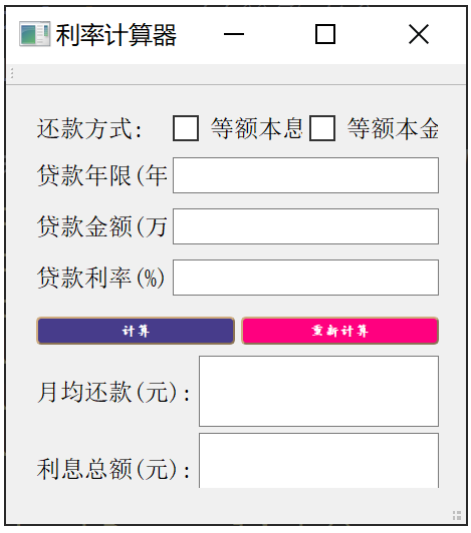


图 18 利率计算器页面

2.3.2 设计分析和算法分析

2.3.2.1 页面搭建

这里采用 Qt 框架自带的 UI 界面进行美化排布，效果如下图所示，利用 UI 自动生成按钮等类，在页面美化中，添加样本样式，利用 Qt 自带的 QSS 进行美化（操作如前端 CSS 类似）。

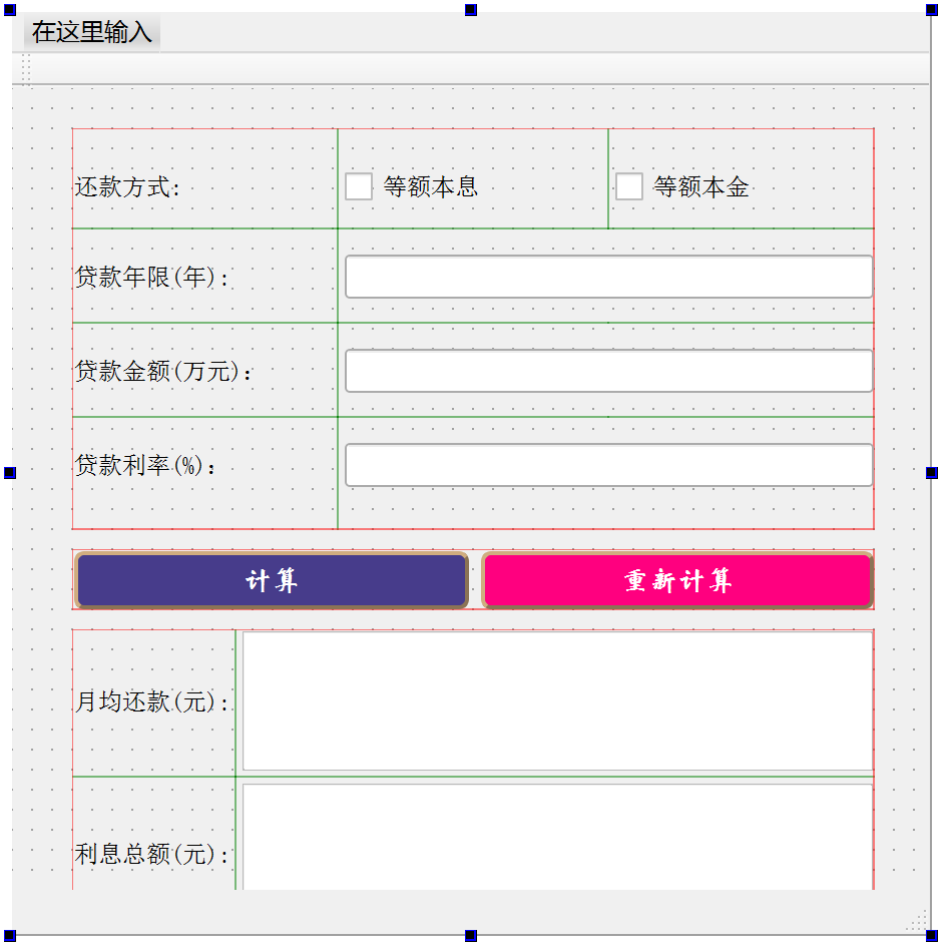


图 19UI 界面美化

2.3.2.2 按钮绑定

利用 UI 生成的类，添加槽函数进行关联绑定，生成类如图 13 所示，在 mainwindow.cpp 析构函数中进行绑定操作。代码如下：

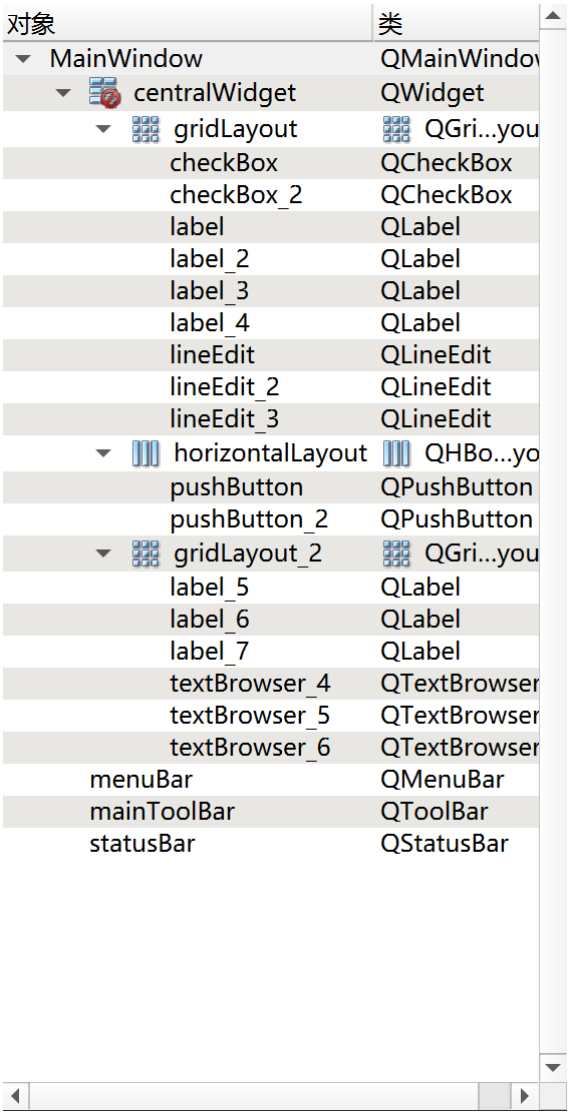


图 20 按钮类

2.2.2.3 模式判断算法

这里有俩种计算利率的方法。只能勾选一种，在这里我们利用一个全局变量来控制计算模式如何，对于还款方式进行构建，代码如下：

```
//等额本息
void MainWindow::on_checkBox_clicked()
{
    if(temp_1==1) temp_1=0;
    else temp_1=1;
    if(temp_2==2&&err == 0)
    {
        QMessageBox::critical(NULL, "错误信息", "请选择一种还款方式", QMessageBox::Yes |
QMessageBox::No, QMessageBox::Yes);
        ui->textBrowser_6->clear();
```

```
        ui->textBrowser_5->clear();
        ui->textBrowser_4->clear();
        ui->lineEdit->clear();
        ui->lineEdit_2->clear();
        ui->lineEdit_3->clear();
        ui->checkBox_2->click();
        //err = 1;
    }

}

//等额本金
void MainWindow::on_checkBox_2_clicked()
{
    if(temp_2==2) temp_2=0;
    else temp_2=2;
    if(temp_1==1)
    {
        QMessageBox::critical(NULL, "错误信息", "请选择一种还款方式", QMessageBox::Yes |
QMessageBox::No, QMessageBox::Yes);
        ui->textBrowser_6->clear();
        ui->textBrowser_5->clear();
        ui->textBrowser_4->clear();
        ui->lineEdit->clear();
        ui->lineEdit_2->clear();
        ui->lineEdit_3->clear();

        ui->checkBox->click();
    }
}
```

代码块 5 模式判断代码

2.2.2.4 重新计算算法

在计算后考虑到用户需要清空，这里重新计算提供一个清空操作，将 UI 中 lineEdit 类都置空，并模式初始化为 0.代码如下：

```
void MainWindow::on_pushButton_2_clicked(bool checked)
{
    ui->textBrowser_6->clear();
    ui->textBrowser_5->clear();
```

```
    ui->textBrowser_4->clear();
    ui->lineEdit->clear();
    ui->lineEdit_2->clear();
    ui->lineEdit_3->clear();
    temp_1=0;
    temp_2=0;
}
```

代码块 6 重新计算代码

2.2.2.5 计算算法

这里首先要明白计算原理

1. 等额本息

等额本息是指一种贷款的还款方式，指在还款期内，每月偿还同等数额的贷款(包括本金和利息)。等额本息还款法即借款人每月按相等的金额偿还贷款本息，其中每月贷款利息按月初剩余贷款本金计算并逐月结清。

计算公式如下：

每月还款额=[贷款本金×月利率×（1+月利率）^还款月数]÷[（1+月利率）^还款月数－1]

2. 等额本金

等额本金是指一种贷款的还款方式，是在还款期内把贷款数总额等分，每月偿还同等数额的本金和剩余贷款在该月所产生的利息，这样由于每月的还款本金额固定，而利息越来越少，借款人起初还款压力较大，但是随时间的推移每月还款数也越来越少。

计算公式如下：

每月还款金额= （贷款本金/还款月数）+（本金—已归还本金累计额）×每月利率

在计算的基础上加入报错功能，提醒功能，增强人机交互感，代码如下：

```
void MainWindow::on_pushButton_clicked()
{

    double num1=ui->lineEdit->text().toInt();   //贷款年限（年）
    double num2=ui->lineEdit_2->text().toInt();    //贷款金额（万元）
    double num3=ui->lineEdit_3->text().toInt()/100.0;   //贷款利率
```

```
    double yue_num3=num3/12.0;     //月利率
    double yue_num2=num2/12.0;     //
    if(temp_1==0&&temp_2==0)
    {
        QMessageBox::critical(NULL, "错误信息", "未选择还款方式", QMessageBox::Yes |
QMessageBox::No, QMessageBox::Yes);
        //err = 1;
    }
    else if(temp_1==1&&temp_2==2)
    {
        QMessageBox::critical(NULL, "错误信息", "请选择一种还款方式", QMessageBox::Yes |
QMessageBox::No, QMessageBox::Yes);
        //err = 1;
    }
    else if(num1<=0)
         QMessageBox::critical(NULL, "错误信息", "未输入有效贷款年限信息",
QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
    else if(num2<=0)
         QMessageBox::critical(NULL, "错误信息", "未输入有效贷款金额信息",
QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
    else if(num3<=0)
         QMessageBox::critical(NULL, "错误信息", "未输入有效贷款利率信息",
QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
    double ans=num1*num2*num3;
    if(num1<=0||num2<=0||num3<=0||(temp_1==0&&temp_2==0)||(temp_1==1&&temp_2==2))
    {
        ui->textBrowser_6->clear();
        ui->textBrowser_5->clear();
        ui->textBrowser_4->clear();
        ui->lineEdit->clear();
        ui->lineEdit_2->clear();
        ui->lineEdit_3->clear();
        return;
    }
    if(temp_1==1)
    {
        double nn=-((num2*num3*((1+yue_num3)*num1*12-(1+yue_num3)*(num1*12-
1))/((1+yue_num3)*num1*12-1))*12-num2);
        ui->textBrowser_6->setText(QString::number((nn/10.0+num2)*10000));     //还款总额
        ui->textBrowser_5->setText(QString::number(nn*1000));     //利息总额
```

```
ui->textBrowser_4->setText(QString::number((num2*yue_num3*(1+yue_num3)*num1*12.0)/(1+yue_num3)*num1*12.0-1000));  //月均还款
    }
    if(temp_2==2)
    {
        double yue_huan=(num2/num1*12)+(num2)*yue_num3;
        double yue_lixi=(num2)*yue_num3;
        double total_lixi=((num2/num1*12+num2*yue_num3)+num2/num1*12*(1+yue_num3))/2*num1*12-num2;
        ui->textBrowser_6->setText(QString::number(((total_lixi/10000+num2)*10000)));
//还款总额
        ui->textBrowser_5->setText(QString::number(total_lixi));    //利息总额
        ui->textBrowser_4->setText(QString::number(yue_huan*100));
    }
}
```

代码块 7 计算原理代码

2.3.3 类图关系

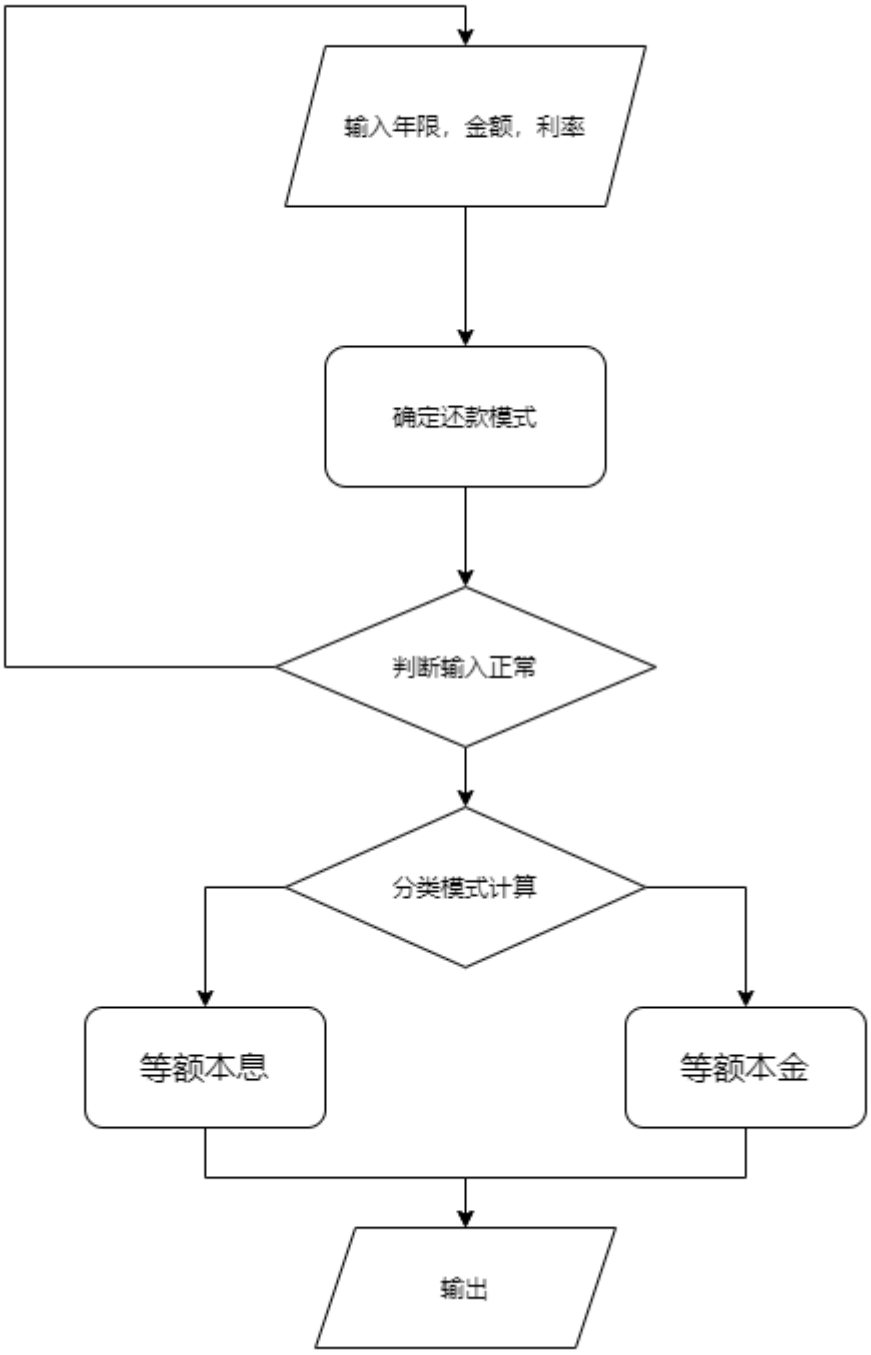


图 21 利率计算流程图

2.4.程序展示

一些操作实际展示，展示效果如下图 7:

图 22 成果展示图

2.5.总结思考

通过使用 Qt 应用框架实现了人机交互界面的利率计算器，采用 Qt 信号槽机制实现利率的运算并对可能出现的错误进行预判，并反馈警告。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。

实验三 2048 游戏

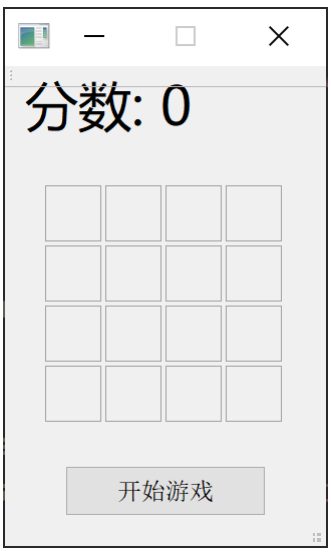
3.1.程序介绍

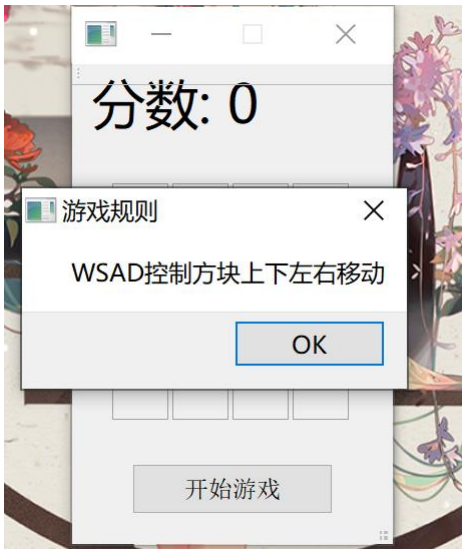
该程序为名称为 2048 游戏。2048 是一款非常有趣的益智游戏，该游戏使用方向键让方块整体上下左右移动。如果两个带有相同数字的方块在移动中碰撞，则它们会合并为一个方块，且所带数字变为两者之和。每次移动时，会有一个值为 2 或者 4 的新方块出现，所出现的数字都是 2 的幂。当值为 2048 的方块出现时，游戏即胜利，该游戏因此得名。本项目利用 Qt 框架结合 c++来实现器游戏规则。

3.2.操作说明

每次控制所有方块向同一个方向运动，两个相同数字的方块撞在一起之后合并成为他们的和，每次操作之后会在空白的方格处随机生成一个 2 或者 4，最终得到一个“2048”的方块就算胜利了。如果 16 个格子全部填满并且相邻的格子都不相同也就是无法移动的话，那么恭喜你，gameover。

点击开始游戏，WSAD 为上下左右控制，记录分数，效果如图 1 所示。





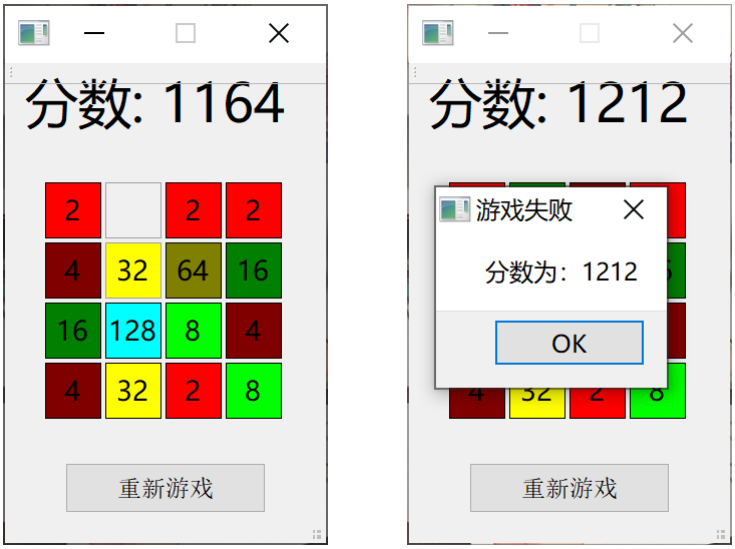


图 23 操作展示图

3.3.设计理念

3.3.1 设计目标

该项目是想设计一个益智小游戏 2048，通过该游戏使用方向键让方块整体上下左右移动。如果两个带有相同数字的方块在移动中碰撞，则它们会合并为一个方块，且所带数字变为两者之和。每次移动时，会有一个值为 2 或者 4 的新方块出现，所出现的数字都是 2 的幂。不同数字对应不同颜色的出现直到出现 2048 游戏胜利。

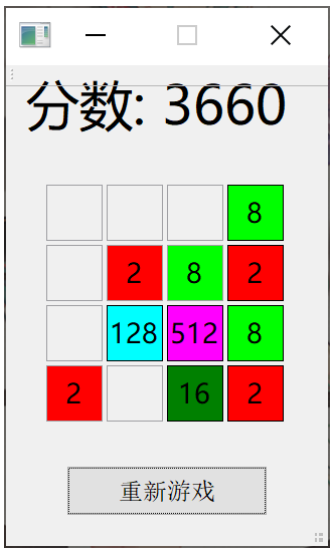


图 24 2048 整体展示

3.3.2 设计分析和算法分析

3.3.2.1 设计整体思路

1. 游戏界面初始化，共有 4 行 4 列，总计 16 个位置，游戏开始时，在任意的两个位置上，随机产生数字 2；这里用二维数组来表示。
2. 玩家可通过 W\A\S\D 或者键盘方向键来控制所有数字的移动，游戏过程中，要符合 2048 游戏的基本规则；
3. 当游戏中无空余位置，且相邻数字之间无法合并，则 game over；
4. 数字移动和合并的算法实现在 2048 游戏中，数字移动和合并为游戏的核心，在游戏过程中，无论数字向那个方向移动，其实现所用的算法都是相同的。
5. 对于不同数字用不同颜色进行显示，初始化为 0（灰色）。每次操作后在一个随机位置生成一个 2，并判断游戏是否继续。

3.3.2.2 整体的成员和成员函数展示

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QKeyEvent>
#include <QPushButton>
#include <QPainter>
#include <QTime>
#include <QDebug>
#include <QMessageBox>


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
```

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

    void paintEvent(QPaintEvent *);
    void keyPressEvent(QKeyEvent *event);

    void PressUp();
    void PressDown();
    void PressLeft();
    void PressRight();

    void myRand();

    //开始游戏的按钮，指针
    QPushButton *button;
    //4*4 的数据集
    int s[4][4];
    //得分
    int score = 0;
    //状态
    bool state;

    //随机的一个点
    struct Ns
    {
        int i;
        int j;
    };


 //槽函数
public slots:
    void slotStart();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

代码块 8 成员函数展示

3.3.2.3 开始按钮

在点击开始游戏按钮后，游戏开始，随机生成一个 2，并刷新屏幕显示。

```
//开始游戏按钮
void MainWindow::slotStart()
{
    QMessageBox::about(this,"游戏规则","WSAD 控制方块上下左右移动");
    score = 0;
    for (int i = 0;i < 4;i++)
    {
        for (int j = 0;j < 4;j++)
        {
            s[i][j] = 0;
        }
    }

    button->setText("重新游戏");

    //随机生成第一个 2
    int randi = qrand() % 4;
    int randj = qrand() % 4;
    s[randi][randj] = 2;

    //结束状态
    state = true;
    //刷新
     update();
}
```

代码块 9 开始游戏代码

3.3.2.4 刷新绘制算法

这里应用 Qt 自带的 paintEvent(QPaintEvent *)函数进行重载。用于主要游戏界面绘制，内置刷新函数进行更新。先绘制顶层分数，进行显示。后遍历 4×4 数组。对不同数字进行不同颜色绘制。代码如下：

```
//绘制每一个方块
void MainWindow::paintEvent(QPaintEvent *)
{
    //基本绘图，开始构建方块
    QPainter p(this);
    p.setBrush(Qt::blue);
    p.setFont(QFont("微软雅黑",20,700,false));
    //分数显示
    QString strscore;
    p.drawText(QPoint(20,60),"分数: "+QString::number(score));
    for (int i = 0;i < 4;i++)
    {
        for (int j = 0;j < 4;j++)
        {
            p.setBrush(Qt::transparent);
            //值为0时为灰色
            if(s[i][j] == 0)
            {
                p.setPen(Qt::gray);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
            }
            else if (s[i][j] == 2)
            {
                p.setBrush(Qt::red);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(2),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==4)
            {
                p.setBrush(Qt::darkRed);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
```

```
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(4),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==8)
            {
                p.setBrush(Qt::green);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(8),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==16)
            {
                p.setBrush(Qt::darkGreen);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(16),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==32)
            {
                p.setBrush(Qt::yellow);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(32),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==64)
            {
                p.setBrush(Qt::darkYellow);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(64),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==128)
            {
                p.setBrush(Qt::cyan);
```

```
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(128),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==256)
            {
                p.setBrush(Qt::darkCyan);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(256),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==512)
            {
                p.setBrush(Qt::magenta);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(512),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==1024)
            {
                p.setBrush(Qt::darkMagenta);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(1024),QTextOption(Qt::AlignCenter));
            }
            else if (s[i][j]==2048)
            {
                p.setBrush(Qt::blue);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(2048),QTextOption(Qt::AlignCenter));
```

```
                QMessageBox::about(this,"游戏胜利","分数为：" + QString::number(score)+"
");
                return;
            }
            else
            {
                p.setBrush(Qt::darkBlue);
                p.drawRect(i * 60 + 40, j * 60 + 120, 55, 55);
                p.setPen(Qt::black);
                p.setFont(QFont("微软雅黑",10,700,false));
                p.drawText(QRectF(i * 60 + 40, j * 60 + 120, 55,
55),QString::number(s[i][j]),QTextOption(Qt::AlignCenter));
            }
        }
    }
}
```

代码块 10 页面函数代码

3.3.2.5 游戏操作设计

调用 keyPressEvent 函数检测键盘操作，并调用先对应函数。

```
//按键控制 wsad,
void MainWindow::keyPressEvent(QKeyEvent *event)
{
    if(!state)
    {
        return ;
    }
    switch (event->key())
    {
    case Qt::Key_W:
        PressUp();
        break;
    case Qt::Key_S:
        PressDown();
        break;
    case Qt::Key_A:
        PressLeft();
        break;
    case Qt::Key_D:
        PressRight();
```

```
        break;
    default:
        //忽略其他按钮
        return;
    }

    //随机生成一个2
    myRand();
    //强制刷新
         update();
}
```

代码块 11keyPressEvent 函数代码

W/S/A/D 操作函数算法原理一致，先进行移动操作，后进行判断，如果这个方向上前一个数字与这个相同着进行合并操作。应用俩次数组遍历来实现。

```
void MainWindow::PressUp()
{
    //移动
    for (int i = 0;i < 4;i++)
    {
        for (int j = 1;j < 4;j++)
        {
            if(s[i][j] == 0)
            {
                continue;
            }
            for (int p = 0;p < j;p++)
            {
                //查看前面是否有空格子可移动
                if(s[i][p] == 0)
                {
                    s[i][p] = s[i][j];
                    s[i][j] = 0;
                    break;
                }
            }
        }
    }
    //相加
    for (int i = 0;i < 4;i++)
    {
        for (int j = 0;j < 3;j++)
```

```
        {
            if(s[i][j] == s[i][j+1])
            {
                s[i][j] = 2 * s[i][j];
                s[i][j+1] = 0;
                score += s[i][j];
                for (int p = j + 2;p < 4;p++)
                {
                    s[i][p-1] = s[i][p];
                }
            }
        }
    }
}


void MainWindow::PressDown()
{
    //移动
    for (int i = 0;i < 4;i++)
    {
        for (int j = 2;j >= 0;j--)
        {
            if(s[i][j] == 0)
            {
                continue;
            }
            for (int p = 3;p > j;p--)
            {
                //查看前面是否有空格子可移动
                if(s[i][p] == 0)
                {
                    s[i][p] = s[i][j];
                    s[i][j] = 0;
                    break;
                }
            }
        }
    }
    //相加
    for (int i = 0;i < 4;i++)
    {
```

```
        for (int j = 3;j > 0; j--)
        {
            if(s[i][j] == s[i][j-1])
            {
                s[i][j] = 2*s[i][j];
                s[i][j-1] = 0;
                score += s[i][j];
                for (int p = j - 2;p >= 0;p--)
                {
                    s[i][p+1] = s[i][p];
                }
            }
        }
    }
}


void MainWindow::PressLeft()
{
    //移动
    for (int j = 0;j < 4;j++)
    {
        for (int i = 1;i < 4;i++)
        {
            if(s[i][j] == 0)
            {
                continue;
            }
            for (int p = 0;p < i; p++)
            {
                //查看前面是否有空格可移入
                if(s[p][j] == 0)
                {
                    s[p][j] = s[i][j];
                    s[i][j] = 0;
                    break;
                }
            }
        }
    }
    //相加
    for (int j = 0;j < 4;j++)
```

```
    {
        for (int i = 0;i < 3;i++)
        {
            if(s[i][j] == s[i+1][j])
            {
                s[i][j] = s[i][j] * 2;
                score += s[i][j];
                s[i+1][j] = 0;
                for(int p = i + 2;p < 4; p++)
                {
                    s[p-1][j] = s[p][j];
                }
            }
        }
    }
}


void MainWindow::PressRight()
{
    //移动
    for (int j = 0;j < 4; j++)
    {
        for (int i = 2;i >= 0; i--)
        {
            if(s[i][j] == 0)
            {
                continue;
            }
            for (int p = 3;p > i;p--)
            {
                //查看前面是否有空格可移入
                if(s[p][j] == 0)
                {
                    s[p][j] = s[i][j];
                    s[i][j] = 0;
                    break;
                }
            }
        }
    }
    //相加
```

```
    for (int j = 0;j < 4;j++)
    {
        for (int i = 3;i >= 0;i--)
        {
            if(s[i][j] == s[i-1][j])
            {
                s[i][j] = s[i][j] * 2;
                s[i-1][j] = 0;
                score += s[i][j];
                for(int p = i - 2; p >= 0; p--)
                {
                    s[p+1][j] = s[p][j];
                }
            }
        }
    }
}
```

代码块 12 操作控制代码

3.3.2.6 随机生成原理

这里采用一个 struct 结构来说实现，每次寻找空闲的格子，随机生成一个数对应这个格子。代码如下：

```
//随机生成
void MainWindow::myRand()
{
    int i = 0;
    int j = 0;
    //找出格子
    struct Ns n[15];
    int ni = 0;
    for (i = 0;i < 4;i++)
    {
        for (j = 0; j < 4; j++)
        {
            if(s[i][j] == 0)
            {
                n[ni].i = i;
                n[ni].j = j;
                ni++;
            }
```

```
        }
    }


    //判断游戏是否结束
    if (ni == 0)
    {
        for (i = 0;i < 4;i++)
        {
            for (j = 0;j < 3;j++)
            {
                if(s[i][j] == s[i][j+1])
                {
                    return;
                }
            }
        }
        for (j = 0;j < 4;j++)
        {
            for (i = 0;i < 3;i++)
            {
                if(s[i][j] == s[i+1][j])
                {
                    return;
                }
            }
        }
        QMessageBox::about(this,"游戏失败","分数为：" + QString::number(score)+" ");
        return;
    }

    int rand = qrand() % ni;
    s[n[rand].i][n[rand].j] = 2;

}
```

代码块 13 随机函数代码

3.3.4 类图关系

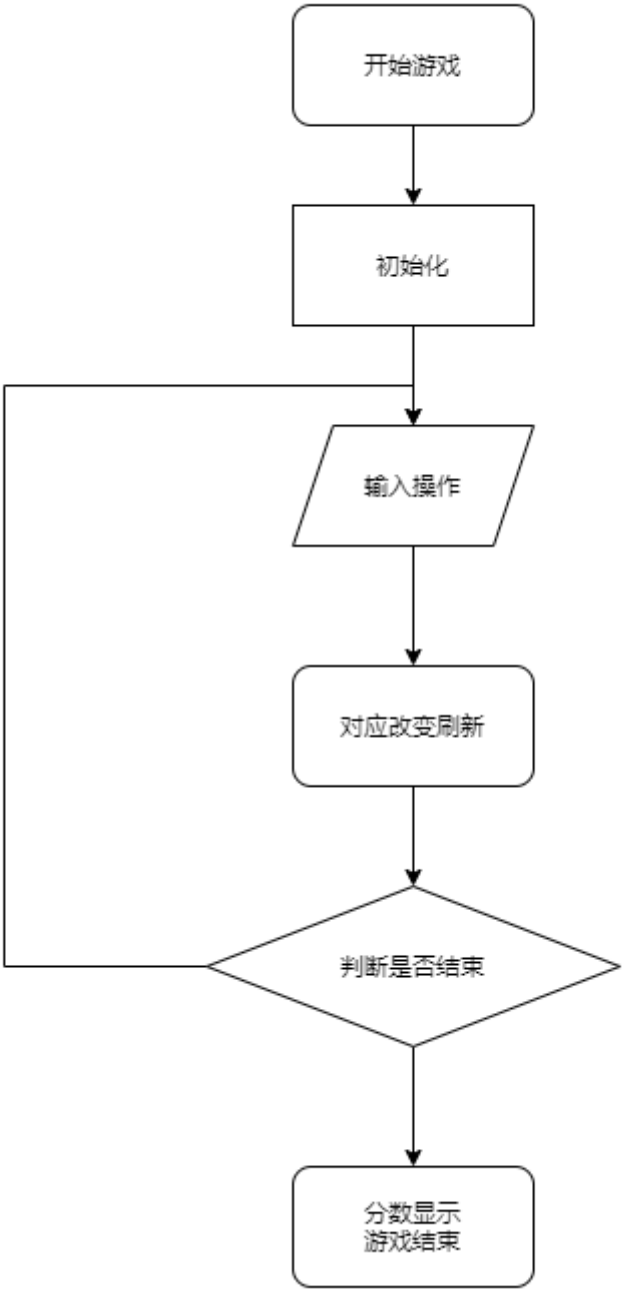


图 25 类关系流程图

3.4 程序展示

一些操作实际展示，展示效果如下图：

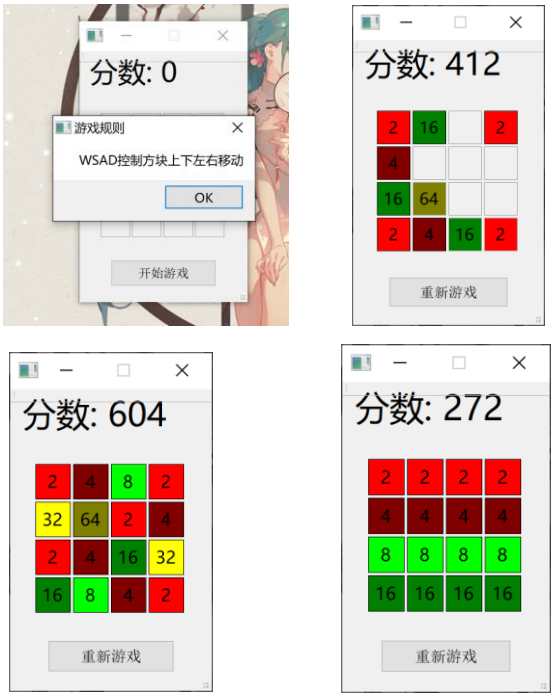


图 26 成果展示图

3.5 总结思考

通过使用 Qt 应用框架实现了 2048 小游戏，采用 Qt 的 QPaintEvent 事件解决了游戏页面刷新布局问题，用 keyPressEvent 实现按键的功能绑定设置。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。

实验四 动物图像识别

4.1 程序介绍

该程序为名称为动物图像识别，设计的思路为调用百度智能云平台 API 接口进行图像处理，本地加载处理识别图片，接受请求后返回的 JSON 文本参数，进行解析给出识别结果。这里利用 Qt 搭建界面，c++发送请求和接受应答。整体为用户给出动物图片，点击开始识别，会显示结果。

4.2 操作说明

点击文件选择打开，选择要识别的动物图像，这里以袋鼠照片为例，点击开始识别按钮就可以看到结果，如图 1 所示。

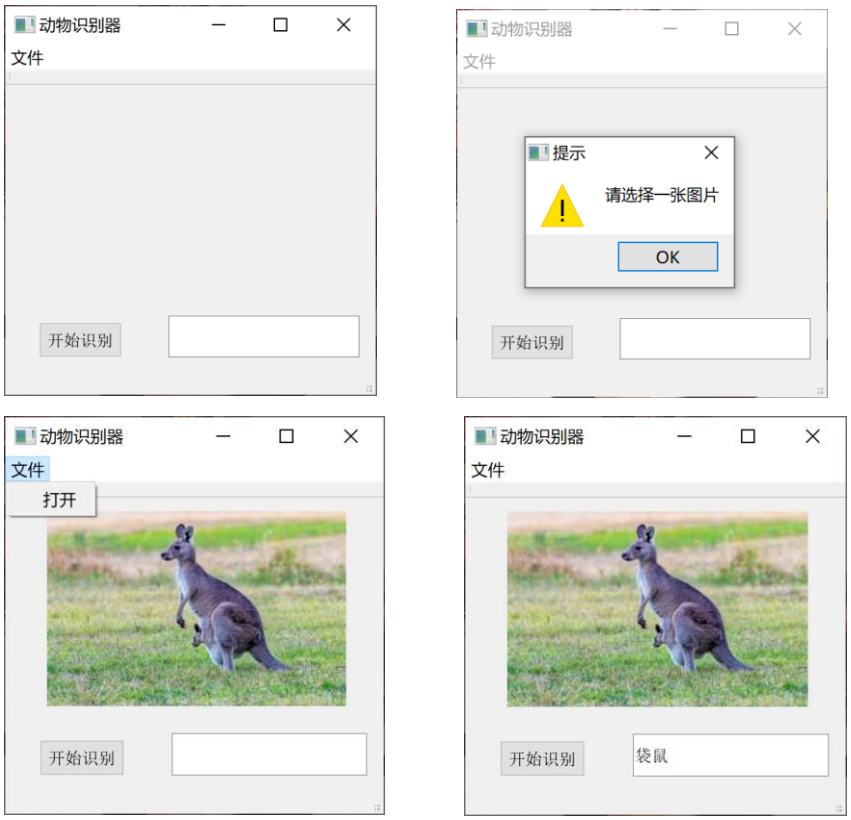


图 27 操作展示图

4.3 设计理念

4.3.1 设计目标

该项目是想设计一个可以自动识别动物图片的软件，对于人们不了解的动物给出其名称，这里采用百度 AI 平台的接口，利用其云数据学习的结果进行匹配，其运行界面如图 2 所示。

图 28 动物图像识别页面

4.3.2 设计分析和算法分析

4.3.2.1 整体思路

搭建一个图片处理的界面，进行图片的读取，这里图像识别主要借助百度云 AI 平台搭建项目服务器，进行访问请求，按照上传要求，先要将图片进行处理，转化为 base64 编码。第一次请求获得 Access Token，第二次请求获得结果，解析结果输出。这里构建一个 Image 类和 Http 类，用来处理图片和发送请求。

4.3.2.2 图像处理算法

将图像转化为 base64，这里利用 QByteArray 类和 QBuffer 类来实现，具体代码如下：

```
#ifndef IMAGE_H
#define IMAGE_H
#include <QString>
#include <QImage>
#include <QByteArray>
#include <QBuffer>
```

```
#include <QTextCodec>


class Image
{
public:
    Image();

    static QByteArray imageToBase64(QString imgPath);
    void urlEncode();
};

#endif // IMAGE_H
```

代码块 14 Image 类头文件

```
#include "image.h"

Image::Image()
{

}



QByteArray Image::imageToBase64(QString imgPath)
{
    QImage img(imgPath);
    QByteArray ba;
    //用 QByterArray 构造 QBuffer
    QBuffer buf(&ba);
    buf.open(QIODevice::WriteOnly);
    //把 img 写入 QBuffer
    img.save(&buf,"JPG");
    //对图片做 base64 编码，不包含编码头
    QByteArray base64 = ba.toBase64();

    QTextCodec* codec = QTextCodec::codecForName("UTF-8");
    QByteArray imgDate = codec->fromUnicode(base64).toPercentEncoding();

    return imgDate;

}
```

代码块 15 Image 类实现代码

4.3.3.3 百度 AI 平台搭建

这里在百度 AI 开发平台注册，创建项目，如图 3 所示，在开发者文档中有 API 教程。

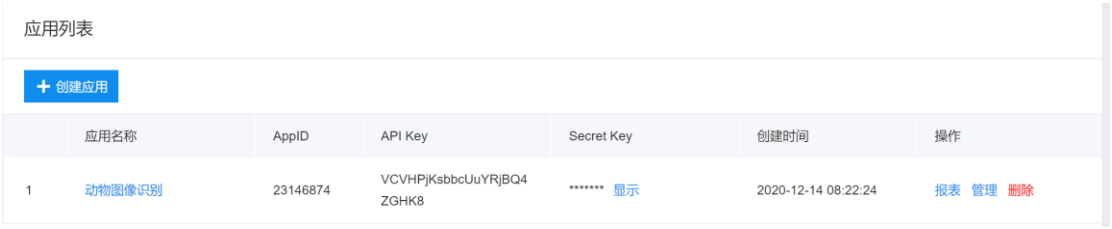


图 29 百度 AI 项目

以本次动物识别为例，请求方法如下图所示，将相关的网站参数用 QString 类进行记录。

请求示例

HTTP 方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/image-classify/v1/animal`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式。**注意：图片需要base64编码、去掉编码头后再进行urlencode。**
top_num	否	integer	6	返回预测得分top结果数，默认为6
baike_num	否	integer	0	返回百科信息的结果数，默认不返回

图 30 请求参数

请求代码示例如下：

curl -i -k 'https://aip.baidubce.com/rest/2.0/image-classify/v1/animal?access_token=【调用鉴权接口获取的 token】' --data 'image=【图片 Base64 编码，需 UrlEncode】

' -H 'Content-Type:application/x-www-form-urlencoded'

对返回的参数进行识别，返回类型如下图所示，对返回的 JSON 文本进行解析，获得结果，返回示例如下：

返回参数

参数	类型	是否必须	说明
log_id	uint64	是	唯一的log id，用于问题定位
result	arrry(object)	是	识别结果数组
+name	string	是	动物名称，示例：蒙古马
+score	string	是	置信度，示例：0.5321
+baike_info	object	否	对应识别结果的百科词条名称
++baike_url	string	否	对应识别结果百度百科页面链接
++image_url	string	否	对应识别结果百科图片链接
++description	string	否	对应识别结果百科内容描述

图 31 返回参数

```
HTTP/1.1 200 OK
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Tue, 18 Oct 2016 02:21:01 GMT
Content-Type: application/json;charset=UTF-8
{
    "log_id": 7392482912853822863,
    "result": [{
        "score": "0.993811",
        "name": "叉角羚",
        "baike_info": {
            "baike_url":
"http://baike.baidu.com/item/%E5%8F%89%E8%A7%92%E7%BE%9A/801703",
```

"description":"叉角羚(学名：Antilocapra americana)：在角的中部角鞘有向前伸的分枝，故名。体型中等，体长 1-1.5 米，尾长 7.5-10 厘米，肩高 81-104 厘米，成体重 36-60 千克，雌体比雄体小；背面为红褐色，颈部有黑色鬃毛，腹部和臀部为白色，颊面部和颈部两侧有黑色块斑；毛被下面为绒毛，上覆以粗糙、质脆的长毛，由于某些皮肤肌的作用，能使其毛被呈不同角度，以利于保暖或散热。植食。叉角羚奔跑速度非常快，最高时速达 100 千米。一次跳跃可达 3.5-6 米。善游泳。夏季组成小群活动，冬季则集结成上百只的大群。为寻找食物和水源，一年中常进行几次迁移。性机警，视觉敏锐，能看到数千米外的物体。遇险时，臀部的白色毛能立起，向同伴告警。分布于北美洲。
"
        }
    },
    {
        "score": "0.000289439",
        "name": "印度羚"
    },
    {
        "score": "0.000186248",
        "name": "藏羚羊"
    },
    {
        "score": "0.000147176",
        "name": "跳羚"
    },
    {
        "score": "0.000134434",
        "name": "驯鹿"
    },
    {
        "score": "9.86555e-05",

```
            "name": "高鼻羚羊"
        }]
}
```

代码块 16 返回示例

4.3.3.4 请求类实现

先导入一些网络请求头文件，利用 QNetworkReply 发送请求，先将请求内容进行拼接组合，确定请求 Url 和请求 data。设置循环请求直到接受返回值。

```
#ifndef HTTP_H
#define HTTP_H

#include <QString>
#include <qnetwork.h>
#include <QtNetwork/QNetworkAccessManager>
#include <QtNetwork/QNetworkReply>
#include <QtNetwork/QNetworkRequest>
#include <QEventLoop>
#include <QJsonObject>
#include <QNetworkReply>
#include <QJsonDocument>
#include <QObject>


const QString BaiduTokenUrl =
"https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=%1&clie
nt_secret=%2";
const QString client_id = "VCVHPjKsbbcUuYRjBQ4ZGHK8";
const QString secret_id = "lGKOocvMR5nhhj6DMsB53vjMSFU6HrZO";
const QString baiduImageUrl = "https://aip.baidubce.com/rest/2.0/image-
classify/v1/animal?access_token=";



class Http : public QObject
{
    Q_OBJECT
public:
    Http();
```

```
    static bool post_sync(QString Url, QMap<QString, QString> header, QByteArray
&requestData, QByteArray &replyData);
};

#endif // HTTP_H
```

代码块 17 Http 头文件

```
#include "http.h"

Http::Http()
{

}


bool Http::post_sync(QString Url, QMap<QString, QString> header, QByteArray &requestData,
QByteArray &replyData)
{
    QNetworkAccessManager manager;            //发送请求的动作

    QNetworkRequest request;                  //请求内容（包含 url 和头）
    request.setUrl(Url);
    QMapIterator<QString, QString> it(header);
    while (it.hasNext())
    {
        it.next();
        request.setRawHeader(it.key().toLatin1(),it.value().toLatin1());

    }

    //发起请求
    QNetworkReply *replay = manager.post(request, requestData);
    QEventLoop l;

    connect(replay,&QNetworkReply::finished, &l, &QEventLoop::quit);

    l.exec();

    if(replay != nullptr && replay->error() == QNetworkReply::NoError)
    {
        replyData = replay->readAll();
        return  true;
```

```
    }
    else
    {
        return  false;
    }
}
```

代码块 18 Http 实现代码

4.3.3.5 打开文件函数

打开文件，选择文集路径，并保存，在 label 中显示。代码如下：

```
void MainWindow::openImageSlot()
{
    imgPath = QFileDialog::getOpenFileName(this,"选择图片
",QCoreApplication::applicationFilePath());
    if(imgPath.isEmpty())
    {
        QMessageBox::warning(this,"提示","请选择一张图片");

    }

    QPixmap pix(imgPath);
    ui->label->setPixmap(pix);
}
```

代码块 19 打开文件代码

4.3.3.6 开始识别按钮

点击开始识别，将图片进行处理，拼接获取 access_token，如何发送请求获得结果。代码如下：

```
void MainWindow::on_pushButton_clicked()
{
    QByteArray img = Image::imageToBase64(imgPath);
    QByteArray imgData = "image=" + img;

    //获取 access_token
    QByteArray replyData;

    QString url = QString(BaiduTokenUrl).arg(client_id).arg(secret_id);
```

```
    QMap<QString, QString> header;
    header.insert(QString("Content-Type"), QString("application/x-www-form-urlencoded"));

    QString accessToken;
    bool result;
    result = Http::post_sync(url, header, imgData, replyData);
    if(result)
    {
        QJsonObject obj = QJsonDocument::fromJson(replyData).object();
        qDebug() << obj.value("access_token").toString();
        accessToken = obj.value("access_token").toString();

    }

    char*  ch;
    QByteArray ba = accessToken.toLatin1();
    ch=ba.data();

    replyData.clear();
    QString imgUrl = baiduImageUrl + accessToken;
    qDebug() << imgUrl;

    //发起第二次请求
    result = Http::post_sync(imgUrl,header,imgData,replyData);
    if(result)
    {
        //没办法解析？？？
        QJsonObject obj = QJsonDocument::fromJson(replyData).object();

        qDebug() << obj.value("log_id").toString();
        QJsonValue val = obj.value("result");
        if(val.isArray())
        {
            QJsonValue first = val.toArray().at(0);

            if(first.isObject())
            {
                QString name = first.toObject().value("name").toString();
                qDebug() << first.toObject().value("name").toString();
                ui->lineEdit->setText(name);
                return;
            }
```

```
        }

    }

    ui->lineEdit->setText("识别错误，这是什么呢？");
    return;
}
```

代码块 20 开始识别代码

4.3.3 类图关系

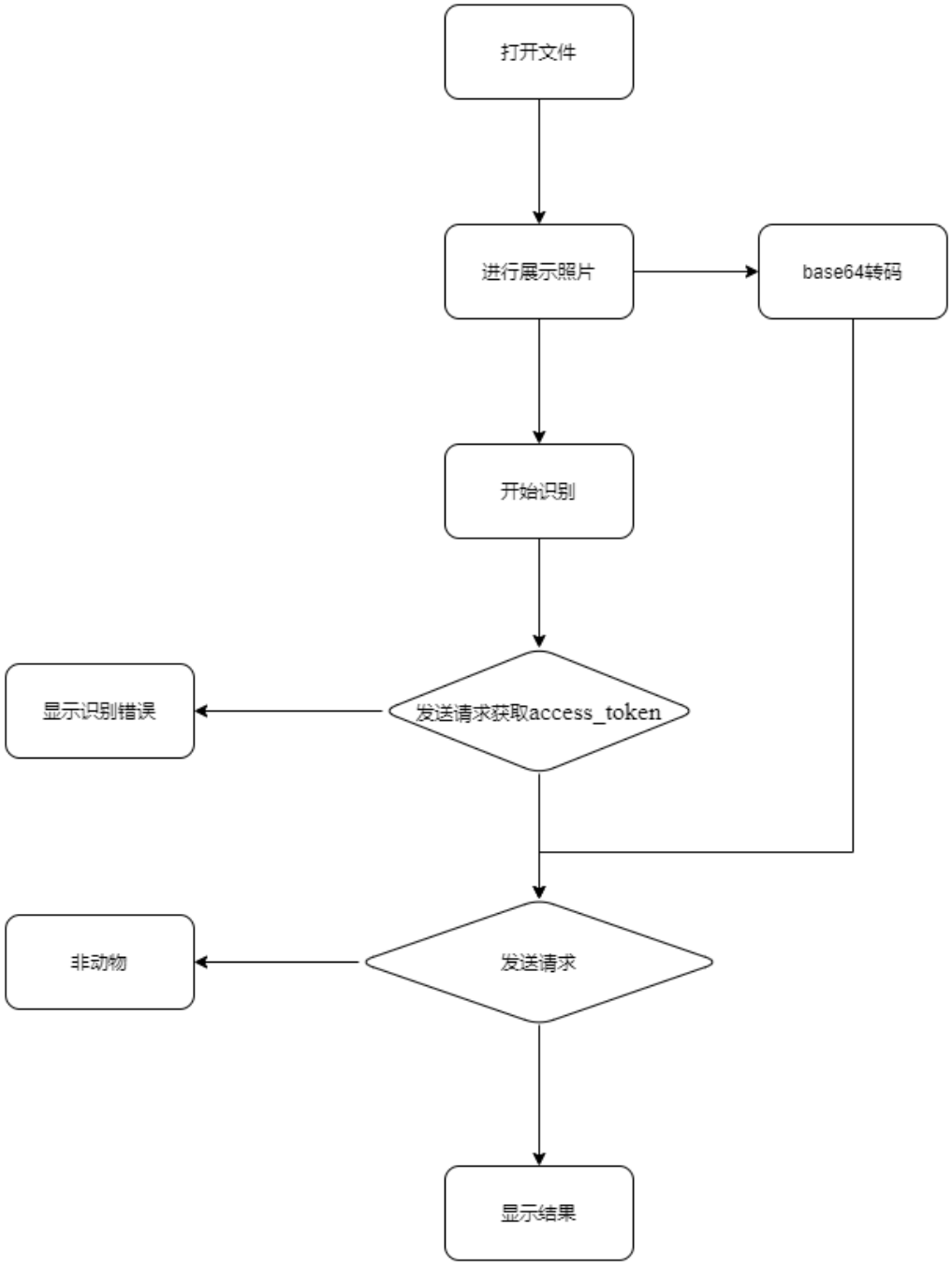


图 32 类关系流程图

4.4 程序展示

一些操作实际展示，展示效果如下图：

图 33 调用流量显示

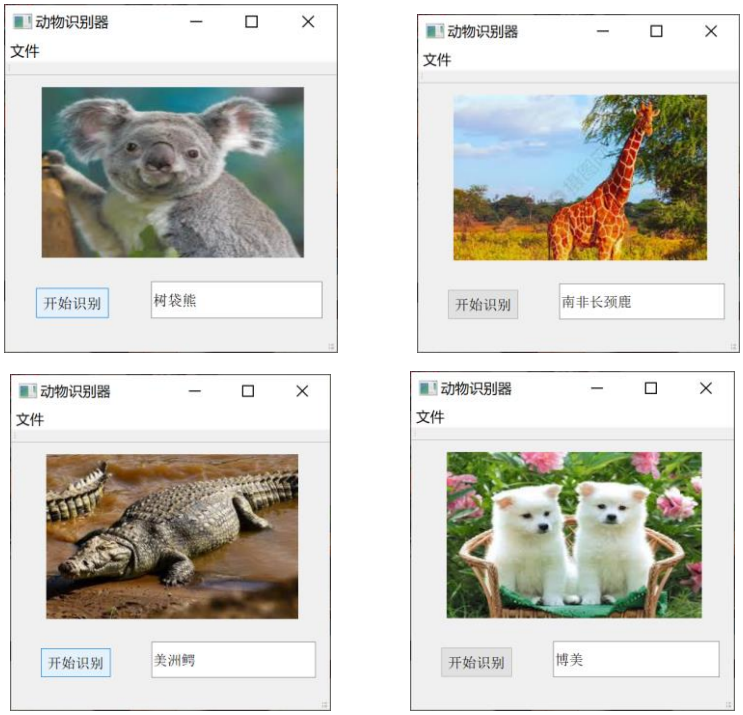


图 34 成果展示图

4.5 总结思考

通过使用 Qt 应用框架实现了人机交互界面的计算器，采用 Qt 实现页面和发送请求，通过使用百度 AI 的 API 接口，实现了图像识别功能。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。

实验五 飞机大战

5.1 程序介绍

该程序为名称为飞机大战。作为早期最经典游戏之一的飞机大战，它是一种传统的电脑游戏，经常出现在便携式终端、手机和计算机中。本游戏是一个基于面向对象编程思想,选用 Qt 框架和 c++语言来实现,PC 端的一款飞机大战游戏。该游戏操作灵活简单，趣味性较强，玩家可以通过鼠标进行游戏操作，通过移动躲避敌方飞机子弹，并发射子弹消灭敌方飞机，从而获取积分。游戏主要涉及了实现飞机移动、发射子弹、碰撞检测、敌方飞机、事件监听,刷新游戏画面及音乐,记录游戏分数和使用者的信息，使游戏界面更多样化，展现出游戏的整体的开发创新流程和设计想法。

5.2 操作说明

5.2.1 游戏开始界面

游戏以开始背景为起点，显示出游戏的背景和我方飞机状态，在点击开始游戏后即可进入游戏。有游戏规则提示窗口,选择是即可进入游戏。效果如图所示:

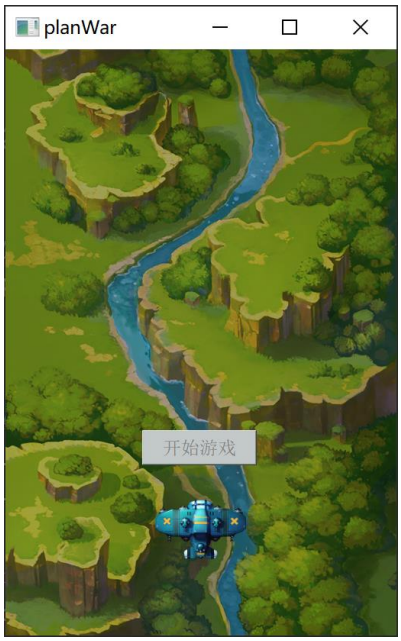


图 35 开始页面

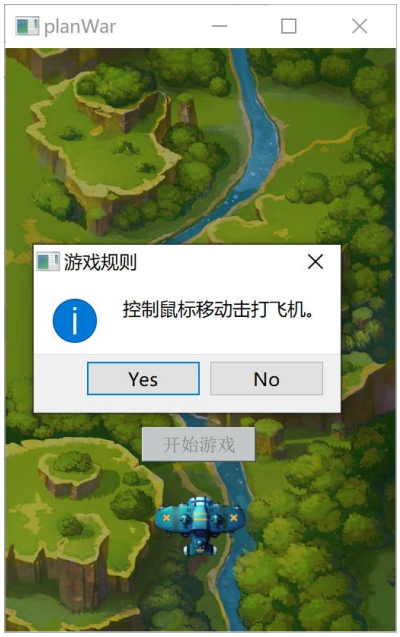


图 36 确认开始页面

5.2.2 游戏状态展示

这里以模拟游戏进行时为操作，展示游戏的操作性，效果如图所示：

图 37 游戏状态

5.2.3 游戏结束展示

如果触碰到敌机，游戏结束。显示得分情况。

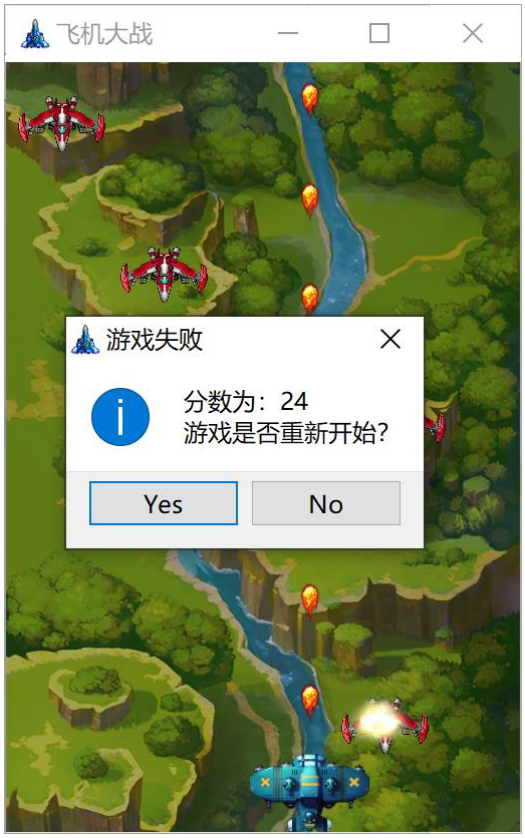


图 38 游戏结束

5.3 设计理念

5.3.1 设计目标

该项目是想设计一个经典的游戏飞机大战,选用 Qt 框架和 c++语言来实现。期望游戏操作灵活简单，趣味性较强，玩家可以通过鼠标进行游戏操作，通过移动躲避敌方飞机子弹，并发射子弹消灭敌方飞机，从而获取积分。游戏主要涉及了实现飞机移动、发射子弹、碰撞检测、敌方飞机、事件监听,刷新游戏画面及音乐,记录游戏分数和使用者的信息。

在整个游戏中，我们看到的所有内容，我们都可以理解为游戏对象，每一个游戏对象,都由一个单独的类来创建;在游戏中主要游戏对象:我方飞机,子弹，背景，敌方飞机，爆炸效果。

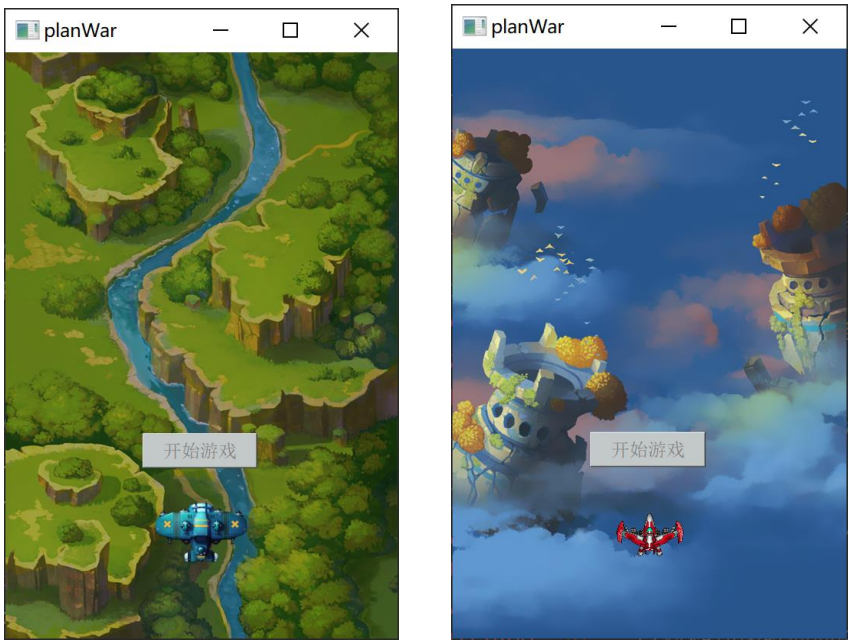


图 39 游戏页面

5.3.2 设计分析和算法分析

5.3.2.1 资源导入

游戏所需要的资源较多，这里我选用网上的素材包，进行一次性导入，因为资源内存较大，qrc 文件无法一次通过，这里 采用 rcc 注册二进制文件的方法导入资源。资源文件如图所示，步骤如下：

1. 生成 qrc 文件
2. 项目同级目录下创建 res 文件夹并将资源粘贴过来
3. 编辑 qrc，加入前缀和文件
4. 利用 qrc 生成二进制文件 rcc
5. rcc 文件放入到 debug 同级目录下
6. 注册二进制文件
7. 添加图标资源

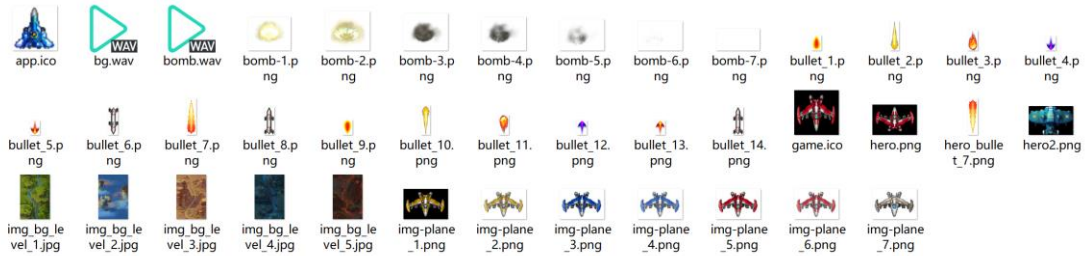


图 40 资源文件

在导入资源生成 rcc 文件后在 main 函数中用 QResource 注册外部的二进制资源文件。代码如下：

```
#include "mainscreen.h"
#include <QApplication>
#include <QResource>
#include "config.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    //注册外部的二进制资源文件
    QResource::registerResource(GAME_RES_PATH);

    MainScreen w;
    w.show();

    return a.exec();
}
```

代码块 21 main 代码

为了方便管理资源参数以及改变游戏背景，资源属性，这里参考网上游戏开发教程，单独生成资源管理头文件。Config 头文件，全部代码如下：

```
#ifndef CONFIG_H
#define CONFIG_H
//配置文件

/********** rcc 文件路径 **********/
#define GAME_RES_PATH  "./plane.rcc" //rcc 文件路径
```

```
/********** 游戏配置数据 **********/
#define GAME_WIDTH  512  //宽度
#define GAME_HEIGHT 768  //高度
#define GAME_TITLE "飞机大战" //标题

#define GAME_RES_PATH "./plane.rcc" //二进制资源路径
#define GAME_ICON ":/res/app.ico"  //图标路径
#define GAME_RATE  10               //单位毫秒

/********** 地图配置数据 **********/
#define MAP_PATH  ":/res/img_bg_level_1.jpg" //地图图片路径
#define MAP_SCROLL_SPEED 2  //地图滚动速度

/********** 飞机配置数据 **********/
#define HERO_PATH ":/res/hero2.png"

/********** 子弹配置数据 **********/
#define BULLET_PATH ":/res/bullet_11.png"   //子弹图片路径
#define BULLET_SPEED 5  //子弹移动速度

#define BULLET_NUM 30   //弹匣中子弹总数
#define BULLET_INTERVAL 20 //发射子弹时间间隔

/********** 敌机配置数据 **********/
#define ENEMY_PATH  ":/res/img-plane_5.png"  //敌机资源图片
#define ENEMY_SPEED 5  //敌机移动速度
#define ENEMY_NUM   20  //敌机总数量
#define ENEMY_INTERVAL  30  //敌机出场时间间隔

/********** 爆炸配置数据 **********/
#define BOMB_PATH  ":/res/bomb-%1.png"   //爆炸资源图片
#define BOMB_NUM  20      //爆炸数量
#define BOMB_MAX  7       //爆炸图片最大索引
#define BOMB_INTERVAL 20   //爆炸切图时间间隔

#define SOUND_BACKGROUND ":/res/bg.wav"
#define SOUND_BOMB ":/res/bomb.wav"

#endif // CONFIG_H
```

代码块 22 Config 头文件

5.3.2.2 地图类的构建

这里用来控制背景图像，采用一个俩个图片拼接，每次刷新，向下移动实现滚动效果，这里难点是控制地图滚动幅度和游戏刷新逻辑保持一致。头文件及实现代码如下：


图 41 背景图资源

```
#ifndef MAP_H
#define MAP_H
#include <QPixmap>

class Map
{
public:
    //构造函数
    Map();

    //地图滚动坐标计算
    void mapPosition();

public:

    //地图图片对象
    QPixmap m_map1;
    QPixmap m_map2;
```

```
    //地图Y轴坐标
    int m_map1_posY;
    int m_map2_posY;

    //地图滚动幅度
    int m_scroll_speed;
};

#endif // MAP_H
```

代码块 23 Map 头文件

```
#include "map.h"
#include "config.h"

Map::Map()
{
    //初始化加载地图对象
    m_map1.load(MAP_PATH);
    m_map2.load(MAP_PATH);

    //设置坐标
    m_map1_posY = -GAME_HEIGHT;
    m_map2_posY = 0;

    //设置滚动速度
    m_scroll_speed = MAP_SCROLL_SPEED;
}

void Map::mapPosition()
{
    //处理第一张图片滚动
    m_map1_posY += m_scroll_speed;
    if(m_map1_posY >= 0)
    {
        m_map1_posY = -GAME_HEIGHT;
    }

    //处理第二张图片滚动
    m_map2_posY += m_scroll_speed;
    if(m_map2_posY >= GAME_HEIGHT)
    {
        m_map2_posY = 0;
```

```
    }

}
```

代码块 24 Map 实现代码

5.3.2.3 我方飞机构建

这里是创建一个我方飞机类，有坐标 x，y 及边界参数，顾应该有一个移动的操作函数,用来和鼠标移动所配合,因为这里要实现打飞机,后面引入子弹类,用于射击。相关 HeroPlane 类的头文件及实现代码如下:

图 42 我方飞机展示

```
#ifndef HEROPLANE_H
#define HEROPLANE_H
#include <QPixmap>
#include "bullet.h"

class HeroPlane
{
public:
    HeroPlane();

    //发射子弹
    void shoot();

    //设置飞机位置
    void setPosition(int x, int y);

public:
    //飞机资源 对象
    QPixmap m_Plane;

    //飞机坐标
    int m_X;
    int m_Y;

    //飞机的矩形边框
```

```
    QRect m_Rect;

    //弹匣
    Bullet m_bullets[BULLET_NUM];

    //发射间隔记录
    int m_recorder;
};
#endif // HEROPLANE_H
```

代码块 25 HeroPlane 头文件

```
#include "heroplane.h"
#include "config.h"

HeroPlane::HeroPlane()
{
    //加载飞机图片资源
    m_Plane.load(HERO_PATH);

    //初始化坐标(通过计算飞机在屏幕坐标得)
    m_X = GAME_WIDTH * 0.5 - m_Plane.width() *0.5;
    m_Y = GAME_HEIGHT - m_Plane.height() - 100;

    //矩形边框 碰撞检测用
    m_Rect.setWidth(m_Plane.width());
    m_Rect.setHeight(m_Plane.height());
    m_Rect.moveTo(m_X,m_Y);

    //初始化间隔记录变量
    m_recorder = 0;
}

//设置飞机位置
void HeroPlane::setPosition(int x, int y)
{
    m_X = x;
    m_Y = y;
    m_Rect.moveTo(m_X,m_Y);
}

//发射子弹
void HeroPlane::shoot()
```

```
{
    //累加事件间隔记录的变量
    m_recorder++;


    //如果记录数字 未达到发射间隔，直接 return
    if(m_recorder < BULLET_INTERVAL)
    {
        return;
    }

    m_recorder = 0;

    //发射子弹
    for(int i = 0 ; i < BULLET_NUM;i++)
    {
        //如果是空闲状态的子弹，发射子弹
        if(m_bullets[i].m_Free)
        {
            m_bullets[i].m_Free = false;
            m_bullets[i].m_X = m_X+m_Rect.width()*0.5 - 10;
            m_bullets[i].m_Y = m_Y - 25 ;
            break;
        }

    }
}
```

代码块 26 HeroPlane 实现代码

5.3.2.4 子弹构建

这里为了实现射击函数，创建子弹类，子弹和我方飞机类似，拥有坐标 x，y 及边界，图片资源，为了进一步控制这里采用控制子弹移动速度和状态，状态来判断碰撞后是否存在，来实现游戏的基础逻辑。Bullet 类的头文件及实现代码如下：

图 43 子弹资源展示

```
#ifndef BULLET_H
#define BULLET_H
#include "config.h"
#include <QPixmap>

class Bullet
{
public:
    Bullet();

    //更新子弹坐标
    void updatePosition();

public:
    //子弹资源对象
    QPixmap m_Bullet;
    //子弹坐标
    int m_X;
    int m_Y;
    //子弹移动速度
    int m_Speed;

    //子弹是否闲置
    bool m_Free;

    //子弹的矩形边框（用于碰撞检测）
    QRect m_Rect;
};

#endif // BULLET_H
```

代码块 27 Bullet 头文件

```
#include "bullet.h"

Bullet::Bullet()
{
    //加载自动资源
    m_Bullet.load(BULLET_PATH);

    //子弹坐标初始化
    m_X = GAME_WIDTH *0.5 - m_Bullet.width() * 0.5;
    m_Y = GAME_HEIGHT;
```

```
    //子弹空闲状态
    m_Free = true;

    //子弹速度
    m_Speed = BULLET_SPEED;

    //子弹矩形边框 （碰撞检测）
    m_Rect.setWidth(m_Bullet.width());
    m_Rect.setHeight(m_Bullet.height());
    m_Rect.moveTo(m_X,m_Y);
}

void Bullet::updatePosition()
{
    //如果子弹是空闲状态，不需要计算坐标
    if(m_Free)
    {
        return;
    }

    //子弹向上移动
    m_Y -=m_Speed;
    m_Rect.moveTo(m_X,m_Y);

    if(m_Y <= - m_Rect.height())
    {
        m_Free = true;
    }

}
```

代码块 28 Bullet 实现函数

5.3.2.5 敌方飞机构建

这里类似子弹类，拥有坐标 x，y 及边界，图片资源，为了进一步控制这里采用控制敌方飞机移动速度和状态，状态来判断碰撞后是否存在，来实现游戏的基础逻辑。EnemyPlane 类头文件及代码如下：

图 44 敌方飞机资源展示

```
#ifndef ENEMYPLANE_H
#define ENEMYPLANE_H



#include <QPixmap>

class EnemyPlane
{
public:
    EnemyPlane();

    //更新坐标
    void updatePosition();
public:
    //敌机资源对象
    QPixmap m_enemy;

    //位置
    int m_X;
    int m_Y;

    //敌机的矩形边框（碰撞检测）
    QRect m_Rect;

    //状态
    bool m_Free;

    //速度
    int m_Speed;
};

#endif // ENEMYPLANE_H
```

代码块 29 EnemyPlane 头文件

```
#include "enemyplane.h"
#include "config.h"
```

```
EnemyPlane::EnemyPlane()
{
    //敌机资源加载
    m_enemy.load(ENEMY_PATH);

    //敌机位置
    m_X = 0;
    m_Y = 0;

    //敌机状态
    m_Free = true;

    //敌机速度
    m_Speed = ENEMY_SPEED;

    //敌机矩形
    m_Rect.setWidth(m_enemy.width());
    m_Rect.setHeight(m_enemy.height());
    m_Rect.moveTo(m_X,m_Y);
}

void EnemyPlane::updatePosition()
{
    //空闲状态，不计算坐标
    if(m_Free)
    {
        return;
    }

    m_Y += m_Speed;
    m_Rect.moveTo(m_X,m_Y);

    if(m_Y >= GAME_HEIGHT)
    {
        m_Free = true;
    }
}
```

代码块 30 EnemyPlane 实现代码

5.3.2.6 爆炸类构建

爆炸类为当飞机与敌机碰撞（边界碰撞），或子弹与敌机碰撞（边界碰撞）时，在这个位置产生爆炸效果及爆炸图片展示。其和我方飞机构建原理类似。Bomb 类头文件及实现代码如下：

图 45 爆炸资源展示

```
#ifndef BOMB_H
#define BOMB_H

#include "config.h"
#include <QPixmap>
#include <QVector>

class Bomb
{
public:
    Bomb();

    //更新信息（播放图片下标、播放间隔）
    void updateInfo();

public:

    //放爆炸资源数组
    QVector<QPixmap> m_pixArr;

    //爆炸位置
    int m_X;
    int m_Y;

    //爆炸状态
    bool m_Free;

    //爆炸切图的时间间隔
    int m_Recoder;
```

```
    //爆炸时加载的图片下标
    int m_index;
};

#endif // BOMB_H
```

代码块 31 Bomb 头文件

```
#include "bomb.h"

Bomb::Bomb()
{
    //初始化爆炸图片数组
    for(int i = 1 ;i <= BOMB_MAX ;i++)
    {
        //字符串拼接，类似 ":/res/bomb-1.png"
        QString str = QString(BOMB_PATH).arg(i);
        m_pixArr.push_back(QPixmap(str));
    }

    //初始化坐标
    m_X = 0;
    m_Y = 0;

    //初始化空闲状态
    m_Free = true;

    //当前播放图片下标
    m_index = 0;

    //爆炸间隔记录
    m_Recoder = 0;
}

void Bomb::updateInfo()
{
    //空闲状态
    if(m_Free)
    {
        return;
    }

    m_Recoder++;
```

```
    if(m_Recoder < BOMB_INTERVAL)
    {
        //记录爆炸间隔未到，直接 return，不需要切图
        return;
    }
    //重置记录
    m_Recoder = 0;

    //切换爆炸播放图片
    m_index++;

    //如果计算的下标大于 6，重置为 0
    if(m_index > BOMB_MAX-1)
    {
        m_index = 0;
        m_Free = true;
    }
}
```

代码块 32 Bomb 实现代码

5.3.2.7 游戏整体构建

这里 MainScreen 继承所有构建类，在 MainScreen 类中实现相关算法以及游戏逻辑，这里首先是音乐的添加，游戏的初始化，控制游戏进程采用定时器对象 QTimer 类来实现，这里应该包含敌机随机产生，子弹的发射，paintEvent 的事件刷新时，各单位的移动操作，以及碰撞检测（及边界检测）和游戏结束函数和重新开始函数。下面展示游戏主体逻辑 MainScreen 的头文件：

```
#ifndef MAINSCREEN_H
#define MAINSCREEN_H

#include <QWidget>
#include "config.h"
#include "map.h"
#include "heroplane.h"
#include "bullet.h"
#include "enemyplane.h"
#include "bomb.h"
#include "start.h"
#include <QIcon>
```

```
#include <QTimer>
#include <QPainter>
#include <QEvent>
#include <QMouseEvent>
#include <QSound>
#include <ctime>
#include <QMessageBox>
#include <QPushButton>

class MainScreen : public QWidget
{
    Q_OBJECT

public:
    MainScreen(QWidget *parent = 0);
    ~MainScreen();

    //初始化场景
    void initScene();

    //游戏重新开始
    void slotStart();

    //初始化显示
    void initialDisplay();
    //启动游戏  用于启动定时器对象
    void playGame();
    //更新坐标
    void updatePosition();
    //绘图事件
    void paintEvent(QPaintEvent *event);

    //重写鼠标移动事件
    void mouseMoveEvent(QMouseEvent *);

    //敌机出场
    void enemyToScene();

    //碰撞检测
    void collisionDetection();
```

```
    //地图对象
    Map m_map;

    //创建飞机对象
    HeroPlane m_hero;


    //定时器对象
    QTimer m_Timer;


    //敌机数组
    EnemyPlane m_enemys[ENEMY_NUM];

    //敌机出场间隔记录
    int m_recorder;

    //爆炸数组
    Bomb m_bombs[BOMB_NUM];


    //得分计算
    int score;

    int Inumber;

    Start ui;
    //测试子弹
    //Bullet temp_Bullet;
};

#endif // MAINSCREEN_H
```

代码块 33 MainScreen 头文件

5.3.2.8 背景音乐

这里背景音乐采用 Qt 框架的 QSound 类，实现游戏背景音乐的循环播放。代码如下：

```
    //启动背景音乐(循环播放)
    //QSound::play(SOUND_BACKGROUND);
```

```
    QSound *sound = new QSound(SOUND_BACKGROUND, this);
    sound->setLoops(-1);
    sound->play();
```

代码块 34 背景音乐代码

5.3.2.9 开始按钮构建

这里在原背景暂停，构建一个开始游戏按钮，在构造函数时调用，设置槽函数搭建初始化连接，代码如下：

```
MainScreen::MainScreen(QWidget *parent)
    : QWidget(parent)
{

    //启动背景音乐(循环播放)
    //QSound::play(SOUND_BACKGROUND);
    QSound *sound = new QSound(SOUND_BACKGROUND, this);
    sound->setLoops(-1);
    sound->play();


    QPushButton *button=new QPushButton(this);
    button->setText("开始游戏");
    button->setStyleSheet("background:#C3C8C9;color:gray");
    resize(GAME_WIDTH,GAME_HEIGHT);
    button->move(width()*0.35,height()*0.65);

    Inumber = 0;
    score = 0;

    connect(button,&QPushButton::clicked,[=](){

        QMessageBox::StandardButton rb = QMessageBox::information(this,"游戏规则","控制鼠标移动击打飞机。",QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
        if(rb == QMessageBox::Yes)
        {
            initScene();
        }
        else if (rb == QMessageBox::No)
        {
```

```
            exit(1);
        }
        button->hide();
        //this->hide();
        });
}
```

代码块 35 开始按钮

5.3.2.10 初始化函数

设置页面大小（与背景图片大小保持一致），调用游戏开始接口，实现游戏逻辑基本框架，随机数的刷新,，为了方便后面死亡后二次调用初始化，做出微量调整，代码如下：

```
void MainScreen::initScene()
{
    //初始化窗口大小
    setFixedSize(GAME_WIDTH,GAME_HEIGHT);

    //设置窗口标题
    setWindowTitle(GAME_TITLE);

    //设置窗口标题
    setWindowIcon(QIcon(GAME_ICON));

    //设置定时器间隔
    //m_Timer.setSingleShot(true);
    m_Timer.setInterval(GAME_RATE);

    //QMessageBox::about(this,"游戏规则","鼠标左键移动");


    //清屏
    //system ("clear");
    //initialDisplay();
    //调用启动游戏接口
    playGame();

    //敌机出场纪录变量 初始化
    m_recorder = 0;
```

```
    //随机数种子
    srand((unsigned int)time(NULL));
}

void MainScreen::slotStart()
{
    //QMessageBox::about(this,"游戏规则","鼠标左键移动");


    //设置定时器间隔
    m_Timer.setInterval(GAME_RATE);
    //initialDisplay();

    //system ("clear");
    initialDisplay();

    playGame();

    //敌机出场纪录变量 初始化
    m_recorder = 0;

    //随机数种子
    srand((unsigned int)time(NULL));
}
```

代码块 36 初始化函数

5.3.2.11 刷新展示

这里实现主见面的刷新主体，绘制飞机，绘制子弹，绘制敌机，绘制爆炸图片，生成随机数种子，敌机出场，更新游戏中所有元素的坐标，把游戏中的元素绘制到屏幕中。代码如下：

```
void MainScreen::initialDisplay()
{

    //绘制飞机
    m_hero.m_X = GAME_WIDTH * 0.5 - m_hero.m_Plane.width() *0.5;
    m_hero.m_Y = GAME_HEIGHT - m_hero.m_Plane.height() - 100;


    //绘制子弹
```

```
    for(int i = 0 ;i < BULLET_NUM;i++)
    {
        m_hero.m_bullets[i].m_X = GAME_WIDTH *0.5 - m_hero.m_bullets[i].m_Bullet.width()
* 0.5;
        m_hero.m_bullets[i].m_Y = GAME_HEIGHT;
        m_hero.m_bullets[i].m_Free == true;
    }



    //绘制敌机
    for(int i = 0 ; i< ENEMY_NUM;i++)
    {
        m_enemys[i].m_X = 0;
        m_enemys[i].m_Y = 0;

        m_enemys[i].m_Free == true;
    }


    //绘制爆炸图片
    for(int i = 0 ; i < BOMB_NUM;i++)
    {
       m_bombs[i].m_Free == true;
    }


    m_recorder = 0;

    //随机数种子
    srand((unsigned int)time(NULL));

    m_Timer.setTimerType(Qt::CoarseTimer);
    m_Timer.setInterval(GAME_RATE + Inumber);

    //敌机出场
    enemyToScene();
    //更新游戏中所有元素的坐标
    updatePosition();
    //游戏中的元素 绘制到屏幕中
    update();
```

```
}
```

代码块 37 刷新函数

```
void MainScreen::updatePosition()
{
    //更新地图坐标
    m_map.mapPosition();

    //发射子弹
    m_hero.shoot();

    //计算子弹坐标
    for(int i = 0 ;i < BULLET_NUM;i++)
    {
        //如果子弹状态为非空闲，计算发射位置
        if(m_hero.m_bullets[i].m_Free == false)
        {
            m_hero.m_bullets[i].updatePosition();
        }

    }

    //敌机坐标计算
    for(int i = 0 ; i< ENEMY_NUM;i++)
    {
        //非空闲敌机 更新坐标
       if(m_enemys[i].m_Free == false)
       {
```

```
            m_enemys[i].updatePosition();
        }
    }

    //计算爆炸播放的图片
    for(int i = 0 ; i < BOMB_NUM;i++)
    {
        if(m_bombs[i].m_Free == false)
        {
            m_bombs[i].updateInfo();
        }
    }

    //测试子弹
//    temp_Bullet.m_Free = false;
//    temp_Bullet.updatePosition();
}

void MainScreen::paintEvent(QPaintEvent *event)
{
    //利用画家画图图片
    QPainter painter(this);

    //绘制地图
    painter.drawPixmap(0,m_map.m_map1_posY,m_map.m_map1);
    painter.drawPixmap(0,m_map.m_map2_posY,m_map.m_map2);

    //绘制飞机
    painter.drawPixmap(m_hero.m_X,m_hero.m_Y,m_hero.m_Plane);

    //绘制子弹
    for(int i = 0 ;i < BULLET_NUM;i++)
```

```
    {
        //如果子弹状态为非空闲 ， 绘制图片
        if(m_hero.m_bullets[i].m_Free == false)
        {

painter.drawPixmap(m_hero.m_bullets[i].m_X,m_hero.m_bullets[i].m_Y,m_hero.m_bullets[i].m_
Bullet);
        }

    }

    //绘制敌机
    for(int i = 0 ; i< ENEMY_NUM;i++)
    {
        if(m_enemys[i].m_Free == false)
        {
            painter.drawPixmap(m_enemys[i].m_X,m_enemys[i].m_Y,m_enemys[i].m_enemy);
        }
    }

    //绘制爆炸图片
    for(int i = 0 ; i < BOMB_NUM;i++)
    {
        if(m_bombs[i].m_Free == false)
        {

painter.drawPixmap(m_bombs[i].m_X,m_bombs[i].m_Y,m_bombs[i].m_pixArr[m_bombs[i].m_index])
;
        }
    }

    //测试子弹
    //painter.drawPixmap(temp_Bullet.m_X,temp_Bullet.m_Y,temp_Bullet.m_Bullet);
}
```

代码块 38 位置更新函数

5.3.2.12 游戏主体

这里游戏主体采用 QTimer 来实现，QTimer 在多线程程序中，可以在一个有事件循环的任何线程中使用 QTimer。使用 QThread::exec()，从非 GUI 线程启动

一个事件循环。Qt 使用定时器的线程关联，以确定哪个线程会发出 timeout()信号。正因为如此，你必须在它的线程中启动和停止定时器，不可能从另一个线程启动定时器。在这里来控制游戏进度时间触发推进。代码如下：

```
void MainScreen::playGame()
{



    //玩游戏 启动定时器
    m_Timer.start();

    //监听定时器的信号
    connect(&m_Timer , &QTimer::timeout,[=](){



        //敌机出场
        enemyToScene();
        //更新游戏中所有元素的坐标
        updatePosition();
        //游戏中的元素 绘制到屏幕中
        update();
        //再调用 paintEvent 函数

        //碰撞检测
        collisionDetection();
    });
}



void MainScreen::enemyToScene()
{
    //累加出场间隔
    m_recorder++;
    if(m_recorder < ENEMY_INTERVAL)
    {
        return;
    }

    m_recorder = 0;

    for(int i = 0 ; i< ENEMY_NUM;i++)
```

```
    {
        if(m_enemys[i].m_Free)
        {
            //敌机空闲状态改为 false
            m_enemys[i].m_Free = false;
            //设置坐标
            m_enemys[i].m_X = rand() % (GAME_WIDTH - m_enemys[i].m_Rect.width());
            m_enemys[i].m_Y = -m_enemys[i].m_Rect.height();
            break;
        }
    }
}
```

代码块 39 开始游戏函数

5.3.2.13 鼠标控制绑定

通过 mouseMoveEvent 函数来实现定位鼠标坐标与我方飞机的绑定操作，因为飞机资源为图像，为了使鼠标与飞机中心完美匹配，对 x，y 设计一定的偏移量，代码如下：

```
void MainScreen::mouseMoveEvent(QMouseEvent * event)
{
    int x =  event->x() -m_hero.m_Rect.width() * 0.5;
    int y =  event->y() -m_hero.m_Rect.height() *0.5;

    //边界检测

    if(x <= 0 )
    {
        x = 0;
    }
    if(x >= GAME_WIDTH - m_hero.m_Rect.width())
    {
        x = GAME_WIDTH - m_hero.m_Rect.width();
    }
    if(y <= 0)
    {
        y = 0;
    }
```

```
    if(y >= GAME_HEIGHT - m_hero.m_Rect.height())
    {
        y = GAME_HEIGHT - m_hero.m_Rect.height();
    }

    m_hero.setPosition(x,y);
}
```

代码块 40 鼠标绑定函数

5.3.2.14 碰撞检测

每个资源类都可以看成一个长方形，具有边界范围，利用循环遍历是否发生碰撞，如果发生碰撞，产生爆炸效果，并调用相关函数和音效，如果我方飞机与敌方飞机碰撞，则游戏结束。代码如下：

```
void MainScreen::collisionDetection()
{
    //遍历所有非空闲的敌机
    for(int i = 0 ;i < ENEMY_NUM;i++)
    {
        if(m_enemys[i].m_Free)
        {
            //空闲飞机 跳转下一次循环
            continue;
        }

        //遍历所有 非空闲的子弹
        for(int j = 0 ; j < BULLET_NUM;j++)
        {
            if(m_hero.m_bullets[j].m_Free)
            {
                //空闲子弹 跳转下一次循环
                continue;
            }

            //如果子弹矩形框和敌机矩形框相交，发生碰撞，同时变为空闲状态即可
            if(m_enemys[i].m_Rect.intersects(m_hero.m_bullets[j].m_Rect))
            {
                //播放音效
                QSound::play(SOUND_BOMB);
```

```
                m_enemys[i].m_Free = true;
                m_hero.m_bullets[j].m_Free = true;

                //调用爆炸特效
                //播放爆炸效果
                score++;
              for(int k = 0 ; k < BOMB_NUM;k++)
              {
                  if(m_bombs[k].m_Free)
                  {
                      //爆炸状态设置为非空闲
                      m_bombs[k].m_Free = false;
                      //更新坐标
                      m_bombs[k].m_X = m_enemys[i].m_X;
                      m_bombs[k].m_Y = m_enemys[i].m_Y;
                      break;
                  }
              }

          }
      }
  }

  //遍历所有非空闲的敌机
  for(int i = 0 ;i < ENEMY_NUM;i++)
  {
      if(m_enemys[i].m_Free)
      {
          //空闲飞机 跳转下一次循环
          continue;
      }
      if(m_enemys[i].m_Rect.intersects(m_hero.m_Rect))
      {
          QSound::play(SOUND_BOMB);

          m_Timer.stop();

          for(int k = 0 ; k < BOMB_NUM;k++)
          {
              if(m_bombs[k].m_Free)
```

```
                {
                    //爆炸状态设置为非空闲
                    m_bombs[k].m_Free = false;
                    //更新坐标
                    m_bombs[k].m_X = m_enemys[i].m_X;
                    m_bombs[k].m_Y = m_enemys[i].m_Y;
                    break;
                }
            }


            QMessageBox::StandardButton rb = QMessageBox::information(this,"游戏失败","分数为：" + QString::number(score)+"\n"+"游戏是否重新开始？",QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
            if(rb == QMessageBox::Yes)
            {
                score = 0;
                Inumber += 10;
                initialDisplay();
                //initScene();
                playGame();
                //需要一个游戏开始的页面
                //slotStart();
            }
            else if (rb == QMessageBox::No)
            {
                exit(1);
            }
            //QMessageBox::about(this,"游戏失败","分数为：" + QString::number(score)+"");
            //exit(1);
        }

    }
}
```

代码块 41 碰撞检测函数

5.3.3 类图关系

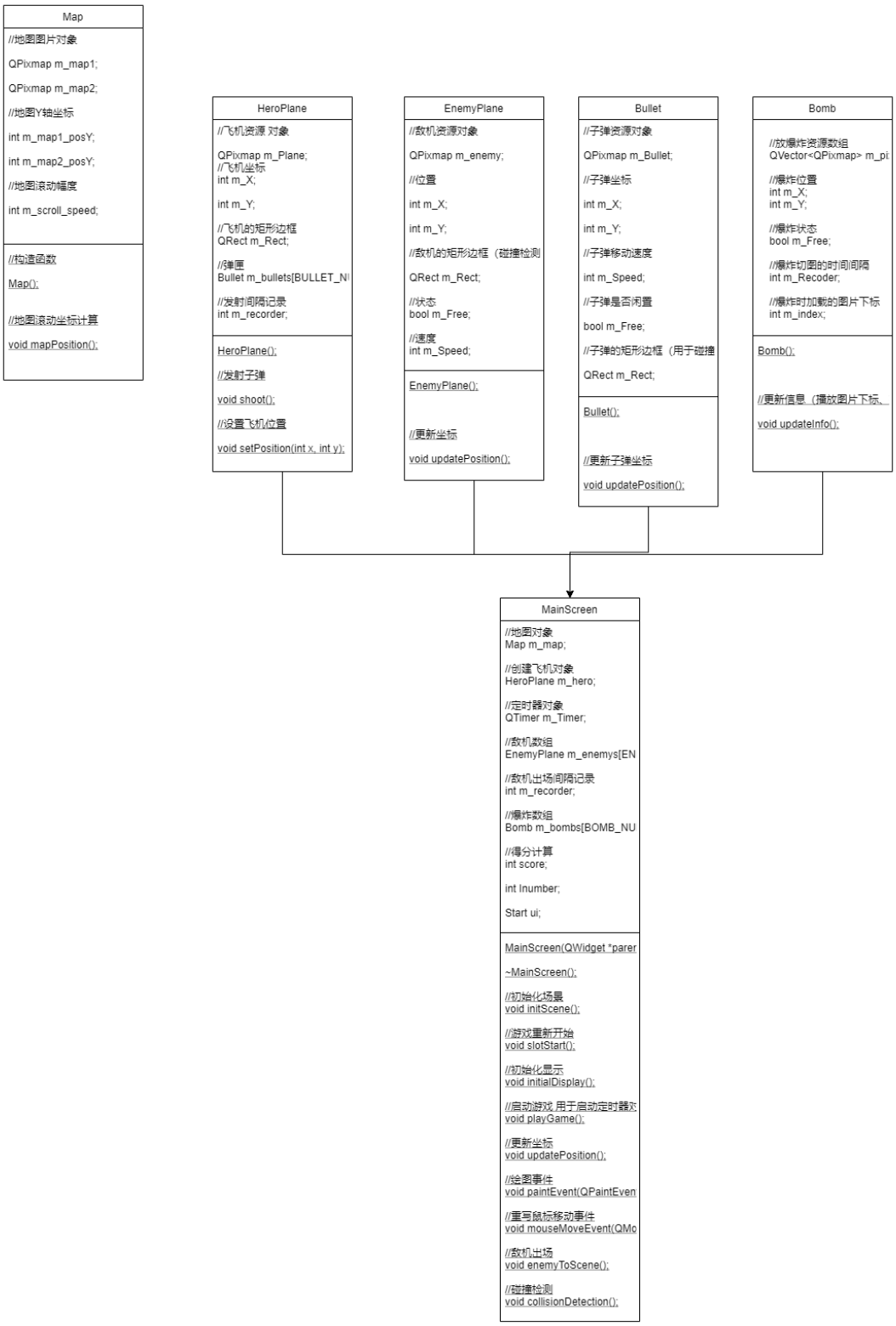


图 46 类关系图

5.4 程序展示

一些操作实际展示，展示效果如下图：

图 47 丛林版

图 48 熔岩版

图 49 云顶版

5.5 总结思考

通过使用 Qt 应用框架实现了飞机大战游戏，并通过 QTimer 控制游戏进程，采用面向对象的思想，将每个资源费封装，根据资源不同可以换不同的版本，可玩性较高，基本上实现了游戏目标。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。

实验六 俄罗斯方块游戏

6.1 程序介绍

该程序为名称为俄罗斯方块游戏,设计的思路为构建一个具有可玩性的俄罗斯方块游戏。俄罗斯方块是由七种四格骨牌构成，全部都由四个方块组成。开始时，一个随机的方块会从区域上方开始缓慢继续落下。落下期间，玩家可以以 90 度为单位旋转方块，以格子为单位左右移动方块，或让方块加速落下。当方块下落到区域最下方或着落到其他方块上无法再向下移动时，就会固定在该处，然后一个新的随机的方块会出现在区域上方开始落下。当区域中某一横行的格子全部由方块填满时，则该列会被消除并成为玩家的得分。同时消除的行数越多，得分指数级上升。在游戏中的目的就是尽量得分。当固定的方块堆到区域最顶端而无法消除层数时，游戏就会结束。

6.2 操作说明

6.2.1 基本游戏规则

一个用于摆放小型正方形的平面虚拟场地，其标准大小：行宽为 10，列高为 20，以每个小正方形为单位。2、一组由 4 个小型正方形组成的规则图形，英文称为 Tetromino，中文通称为方块共有 7 种，分别以 S、Z、L、J、I、O、T 这 7 个字母的形状来命名。

玩家可以做的操作有：以 90 度为单位旋转方块，以格子为单位左右移动方块，让方块加速落下。

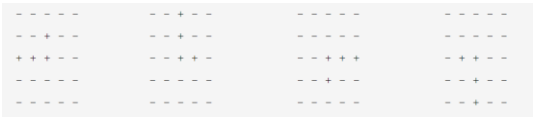
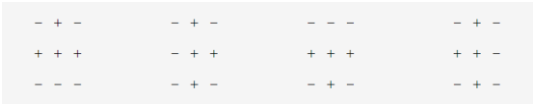
方块移到区域最下方或是着地到其他方块上无法移动时，就会固定在该处，而新的方块出现在区域上方开始落下。

当区域中某一列横向格子全部由方块填满,则该列会消失并成为玩家的得分。同时删除的列数越多，得分指数上升。

当固定的方块堆到区域最上方而无法消除层数时，则游戏结束。

俄罗斯方块类型如下:

表格 1 俄罗斯方块类型及表示

类型	功能	描述
I	一次最多消除四层	
J	最多消除三层，或消除二层	
L	最多消除三层，或消除二层	
O	消除一至二层	
S	最多二层，容易造成孔洞	
Z	最多二层，容易造成孔洞	
T	最多三层	

6.2.2 游戏界面

页面布局为右边提示窗口，左边游戏界面，致敬经典 PSP 游戏风格。在打开游戏会有欢快的游戏背景音乐。

图 50 游戏页面

6.2.3 游戏开始

整体游戏可以上下左右来控制，左右移动，上变换，下加速，空格暂停。实践体验如图所示：

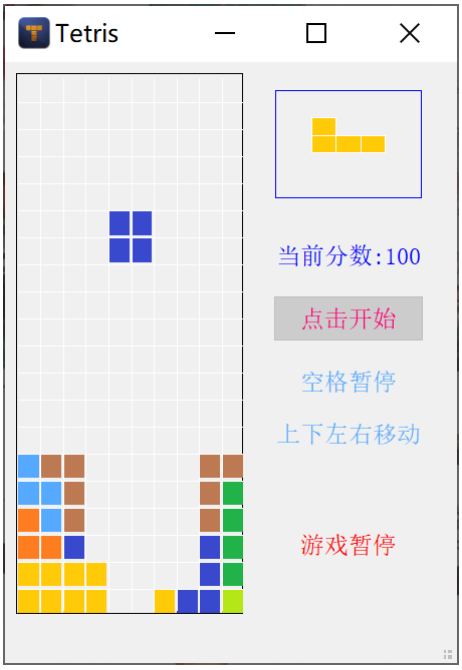


图 51 游戏暂停状态

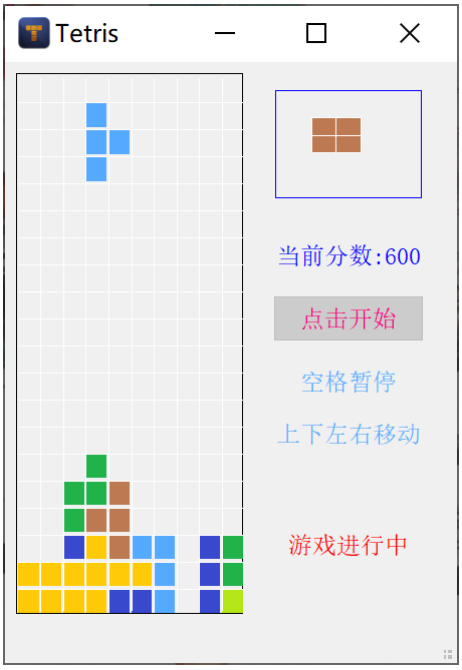


图 52 游戏进行时

6.2.4 游戏结束

游戏结束提示分数，可以选择继续游戏重新开始。

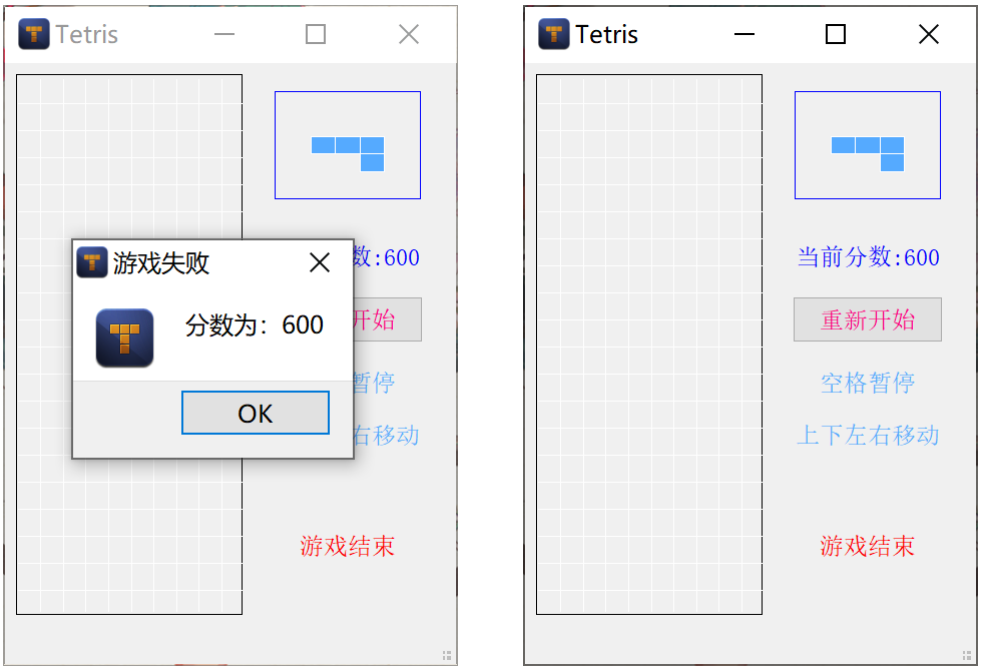


图 53 游戏结束展示

6.3 设计理念

6.3.1 设计目标

俄罗斯方块其实就是一副能够实现人机交互的动画，可以让人来控制四格拼版个的图形样式和位置的呈现。基本功能包括：游戏主界面显示模块、方块及数据显示模块、方块移动控制模块、游戏界面颜色控制模块、游戏进度、等级控制模块等。本项目结构如下：其运行界面如图所示。

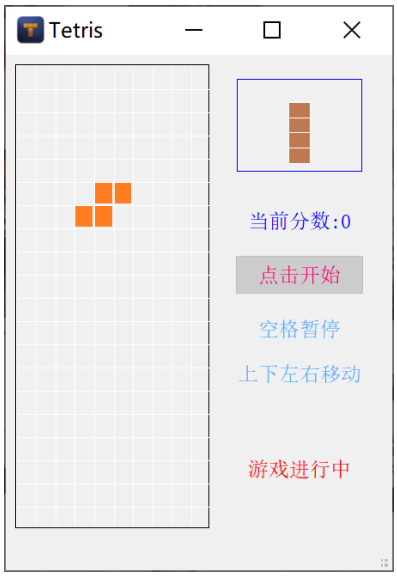


图 54 俄罗斯方块页面

6.3.2 设计分析和算法分析

6.3.2.1 页面搭建

这里采用 Qt 框架自带的 UI 界面进行美化排布，效果如下图所示，利用 UI 自动生成按钮等类，在页面美化中，添加样本样式，利用 Qt 自带的 QSS 进行美化（操作如前端 CSS 类似）。

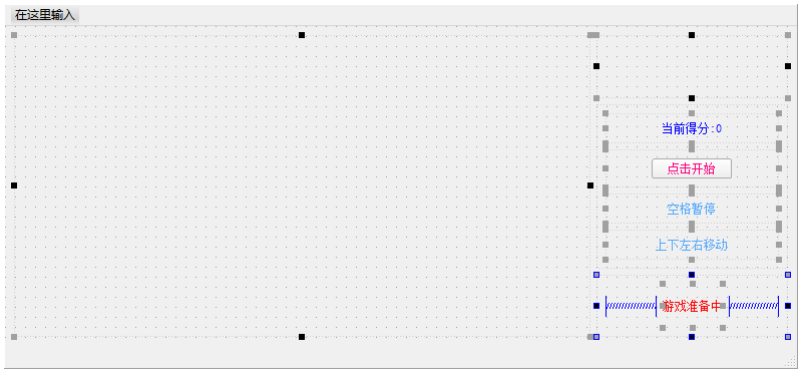


图 55UI 界面美化

6.3.2.2 页面生成

利用 UI 生成的类，进行类的升级关联绑定，生成类如图 13 所示，在 mainwindow.cpp 析构函数中进行绑定操作。代码如下：

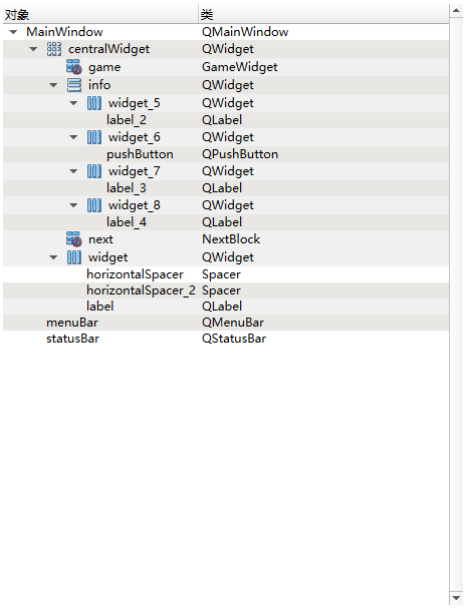
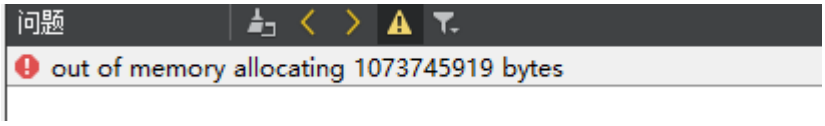


图 56UI 生成类

6.2.2.3 资源的导入

这里因为涉及音乐处理，资源文件过大，采用二进制注册的方法，先导入资源生成 res.qrc 文件。如果直接运行会报错，如图所示。



由于资源文件 qrc 过大，超出分配的内存范围，因此我们需要利用二进制资源，而生成二进制资源就需要我们刚刚的 qrc 文件利用 cmd 打开终端，定位到

res.qrc 的目录下，输入命令 rcc -binary .\res.qrc -o res.rcc 即可。

```
#include "mainwindow.h"
#include <QApplication>
#define GAME_RES_PATH  "./res.rcc" //rcc 文件路径


int main(int argc, char *argv[])
{


    //注册外部的二进制资源文件
    QResource::registerResource(GAME_RES_PATH);


    QApplication a(argc, argv);
    MainWindow w;
    w.show();


    return a.exec();
}
```

代码块 42 主函数调用注册二进制

6.2.2.4 控制类构建

这里实现俄罗斯方块的基本游戏规则，生成每个方块的数组和他的变换，生成方块函数采用随机数访问数组下标，控制边界检查，始终在访问范围内，每次下移动操作的实现和游戏消除以及结束检查。为了方便预测值传给主游戏界面，这里重载了等于=操作符。Control 类头文件及实现代码如下：

```
#ifndef CONTROL_H
#define CONTROL_H
#include<QVector>
```

```
#include<QPoint>

#include<QDebug>


class Control

{

public:

    Control();

    //当前处于的 x 坐标

    int x;

    //当前处于的 y 坐标

    int y;

    //方块的类型

    int type;

    //方块类型变成了那种了

    int process;

    //颜色类型

    int color;



    static QVector<QVector<int>> map; //map 类  地图信息

    static QVector<QVector<QVector<QPoint>>> block; //留存砖块形状 配合 control 类生成方块

    static QVector<QVector<int>> colorMap;


    //生成新的砖块

    void newBlock();


    //砖块发生移动的时候 检查下标

    bool indexCheck(int moveX,int moveY,int newProcess);
```

```
    //移动操作
    void move(int type);

    //垃圾回收 即发生移动过后 清楚上一步的痕迹
    void gc();

    //重新绘制
    void rePaint();

    //判断清除一行
    void clearRow();

    //重置等于操作
    void operator=(Control&);

    void gameOver();
};

#endif // CONTROL_H
```

代码块 43 Control 类头文件

```
#include "control.h"
#include<QRandomGenerator>
#include"gamewidget.h"
#include<QTimer>
#include"mainwindow.h"

//方块类型
QVector<QVector<QVector<QPoint>>> Control::block=
{
  {
```

```
    //四种变换
    {{0, 0}, {1, 0}, {1, 1}, {1, 2}},  // *
    {{2, 0}, {0, 1}, {1, 1}, {2, 1}},  // *
    {{1, 0}, {1, 1}, {1, 2}, {2, 2}},  // ** L 类型方块
    {{0, 1}, {1, 1}, {2, 1}, {0, 2}}
},//这个是 L
{
    {{1, 0}, {1, 1}, {1, 2}, {1, 3}}, // *
    {{0, 1}, {1, 1}, {2, 1}, {3, 1}}, // *
    {{1, 0}, {1, 1}, {1, 2}, {1, 3}}, // *
    {{0, 1}, {1, 1}, {2, 1}, {3, 1}}  // *  直方块
},//这个是直线
{
    {{1, 0}, {2, 0}, {1, 1}, {1, 2}}, //  *
    {{0, 1}, {1, 1}, {2, 1}, {2, 2}}, //  *
    {{1, 0}, {1, 1}, {0, 2}, {1, 2}}, // **  反 L
    {{0, 0}, {0, 1}, {1, 1}, {2, 1}}
},//这个是反 L
{
    {{0, 0}, {1, 0}, {2, 0}, {1, 1}},
    {{0, 0}, {0, 1}, {0, 2}, {1, 1}}, //  *
    {{0, 2}, {1, 2}, {2, 2}, {1, 1}},//  ***  就这个类似三角形的方块
    {{2, 0}, {2, 1}, {2, 2}, {1, 1}}
},//这个是 土
{
    {{0, 0}, {1, 0}, {0, 1}, {1, 1}},
    {{0, 0}, {1, 0}, {0, 1}, {1, 1}}, // **
    {{0, 0}, {1, 0}, {0, 1}, {1, 1}}, // **  正方形方块
    {{0, 0}, {1, 0}, {0, 1}, {1, 1}}
},//这个是田
{
    {{1, 1}, {2, 1}, {2, 0}, {1, 2}},
    {{1, 1}, {0, 1}, {1, 2}, {2, 2}},  //   *
    {{1, 1}, {2, 1}, {2, 0}, {1, 2}}, //   **
    {{1, 1}, {0, 1}, {1, 2}, {2, 2}}  //   *   旋转是 Z 的方块
},//这个是 Z
{
    {{0, 0}, {0, 1}, {1, 1}, {1, 2}},//   *
    {{0, 1}, {1, 1}, {1, 0}, {2, 0}},//   **
    {{0, 0}, {0, 1}, {1, 1}, {1, 2}},//    * 旋转式反 Z 的方块
    {{0, 1}, {1, 1}, {1, 0}, {2, 0}}
},//这个是反 Z
```

```
};



QVector<QVector<int>> Control::map;
QVector<QVector<int>> Control::colorMap;

Control::Control()
{
}


//生成砖块
void Control::newBlock()
{
        x = 3;
        y = 0;

        //Qt5 新随机数方法 返回[0,7)
        type=QRandomGenerator::global()->bounded(0,7);
        process=QRandomGenerator::global()->bounded(0,4);

        qDebug()<<type<<process;
        color=QRandomGenerator::global()->bounded(0,8);
}


// moveX 代表 x 坐标位移大小 moveY 代表 moveY 坐标大小 newProcess 代表用户想要进行砖块转换
bool Control::indexCheck(int moveX,int moveY,int newProcess)
{

    for(int i=0;i<4;i++)
    {
        //i<4 因为每种砖块都是由 4 个构成
        int tempx=this->x+block[this->type][newProcess][i].x()+moveX;
        int tempy=this->y+block[this->type][newProcess][i].y()+moveY;

        //条件说明 x 超过边界 y 超过边界 想要移动的地方已经具有砖块 都返回 false 不允许当前
移动操作
        if(tempx<0||tempx>9||tempy<0||tempy>19||map[tempx][tempy])
            return false;
    }
```

```
    return true;
}



void Control::move(int type)
{
    //标志是否停止
    bool stopFlag=false;
    //移除刚才的痕迹 进行下一步绘制
    gc();

    //1 代表想上 即方块进行变化
    if(type==1){
        int newProcess=(this->process+1)%4;
        if(indexCheck(0,0,newProcess))
        {
            this->process=newProcess;
        }
    }
    //2 代表向下移动 如果落地 新建状况 并且检测是否有行可以消除
    else if(type==2){
        stopFlag=indexCheck(0,1,this->process);
        if(stopFlag)
        {
            y++;
        }
        else
        {
            //回溯一次 下面的 paint 准备绘制新方块
            rePaint();
            //有方块落地 检测是否可以消行
            clearRow();
            *this=GameWidget::anotherCon;
        }
    }
    //3 代表向左移动
    else if(type==3)
    {
        if(indexCheck(-1,0,this->process))
        {
            x--;
```

```
        }
    }
    //4 代表向右移动
    else if(type==4)
    {
        if(indexCheck(1,0,this->process))
        {
            x++;
        }
    }
    rePaint();
}

//垃圾回收 即发生移动过后 清楚上一步的痕迹
void Control::gc(){
    for(int i=0;i<4;i++)
    {

map[this->x+block[this->type][this->process][i].x()][this->y+block[this->type][this->proc
ess][i].y()]=0;
    }
}



void Control::rePaint()
{
    for(int i=0;i<4;i++)
    {
        int tempx=this->x+block[this->type][this->process][i].x();
        int tempy=this->y+block[this->type][this->process][i].y();

        //新砖块刷新 可地图已经有了内容 即死亡
        if(map[tempx][tempy])
        {
            GameWidget::timer->stop();
            GameWidget::gameOver=true;
            MainWindow::running=false;
        }
        else
        {
            colorMap[tempx][tempy]=this->color;
            map[tempx][tempy] = 1;
```

```
        }
    }
}


void Control::clearRow(){
    int i,j;
    //因为是按 x y 格式进行标记的 所以删除方式比较特殊
    for(j=0;j<map[0].size();j++)
    {
        int flag1=true;
        for(i=0;i < map.size();i++)
        {
            if(map[i][j]==0)
                flag1=false;
        }
        //清除整行向下移动
        if(flag1)
        {
            for(int k=0;k < map.size();k++)
            {
                map[k].remove(j);
                map[k].insert(0,0); //插入到最前！！达到自动更新坐标
            }
            MainWindow::score+=100;
        }
    }
}

void Control::operator=(Control &another)
{
    this->x=another.x;
    this->y=another.y;
    this->type=another.type;
    this->process=another.process;
    this->color=another.color;
    another.newBlock();
}
```

代码块 44 Control 类实现代码

6.2.2.5 游戏主界面

游戏主界面，主要用来刷新控制页面显示，这里实现尺寸和每一个小方块的大小，颜色设置，调用 paintEvent 来实现控制，这里设置 Control 类才表示下落的俄罗斯方块类型。GameWidget 继承于 QWidget，头文件及实现代码如下：

```
#ifndef GAMEWIDGET_H

#define GAMEWIDGET_H


#include <QWidget>

#include "control.h"

#include<QPainter>

#include<QPen>

#include<QKeyEvent>

#include<QTimer>

#include<mainwindow.h>



class GameWidget : public QWidget

{

    Q_OBJECT

public:

    //这个是游戏主窗口

    //这里主要绘制图像和捕捉按键事件 而具体的逻辑 交给 control 类实现

    explicit GameWidget(QWidget *parent = nullptr);


    static Control con;

    static Control anotherCon;
```

```
    static QTimer *timer;

    static QVector<QColor> color;

    static bool gameOver;

signals:
    void GameOver();

public slots:
protected:
    //virtual void keyPressEvent (QKeyEvent *);
    virtual void paintEvent (QPaintEvent *);
};

#endif // GAMEWIDGET_H
```

代码块 45 GameWidget 头文件

```
#include "gamewidget.h"

Control  GameWidget::con;
Control  GameWidget::anotherCon;
QTimer* GameWidget::timer;

bool GameWidget::gameOver=false;

QVector<QColor> GameWidget::color=
```

```
{
    QColor::fromRgb(85,170,255),QColor::fromRgb(255,125,33),
    QColor::fromRgb(33,178,74),QColor::fromRgb(189,121,82),
    QColor::fromRgb(140,0,16),QColor::fromRgb(57,73,206),
    QColor::fromRgb(255,203,8),QColor::fromRgb(181,231,24)
};

GameWidget::GameWidget(QWidget *parent) : QWidget(parent)
{
    //设置尺寸
    this->resize(350,700);
    //调整尺寸
    for(int i=0;i<10;i++)
    {
        Control::map.push_back(QVector<int>(20,0));
    }
    for(int i=0;i<10;i++)
    {
        Control::colorMap.push_back(QVector<int>(20,0));
    }
    //con.rePaint();
}

void GameWidget::paintEvent (QPaintEvent *)
{
    QPainter p(this);
    p.setViewport(this->rect());
```

```
p.setWindow(-1,-1,101,201);
QPen pen;
pen.setWidth(0);
pen.setColor("black");
p.setPen(pen);
p.drawRect(-1,-1,100,200);
//根据 1 的标志 来绘画砖块
for(int i=0;i<Control::map.size();i++)
{
    for(int j=0;j<Control::map[i].size();j++)
    {
        if(Control::map[i][j])
        {
            p.fillRect(i*10, j*10,9,9,color[Control::colorMap[i][j]]);
        }
    }
}
pen.setWidth(0);
pen.setColor("white");
p.setPen(pen);

//画线
for(int i=1;i<20;i++)
{
    p.drawLine(0,i*10,99,i*10);
}
for(int i=1;i<10;i++)
{
```

```
            p.drawLine(i*10,1,i*10,198);
        }
        if(gameOver)
        {
            delete timer;
            timer = nullptr;
            emit GameOver();
        }
}
```

代码块 46 GameWidget 实现代码

6.2.2.6 预测窗口代码

其实现逻辑于主页面类似，不需要移动，只需要在游戏改变状态时随机生成一个方块并显示。NextBlock 头文件及实现代码如下：

```
#ifndef NEXTBLOCK_H
#define NEXTBLOCK_H
#include <QWidget>
#include<QPen>
#include<QPainter>
#include"gamewidget.h"
#include<QColor>
#include "mainwindow.h"

class NextBlock : public QWidget
{
    Q_OBJECT
public:
    explicit NextBlock(QWidget *parent = nullptr);
```

```
    void paintEvent (QPaintEvent *e);
signals:

public slots:
};

#endif // NEXTBLOCK_H
```

代码块 47 NextBlock 头文件

```
#include "nextblock.h"

NextBlock::NextBlock(QWidget *parent) : QWidget(parent)
{
}



//{0, 0}, {1, 0}, {0, 1}, {1, 1}
void NextBlock::paintEvent (QPaintEvent *e)
{
    Q_UNUSED(e);
    QPainter p(this);
    p.setViewport(this->rect());
    p.setWindow(0,0,80,80);

    QPen pen;
    pen.setWidth(0);
    pen.setColor("blue");
```

```
    p.setPen(pen);
    p.drawRect(10,10,60,60);


GameWidget::anotherCon.block[GameWidget::anotherCon.type][GameWidget::anotherCon.process]
;


    for(int i=0;i<4;i++)
    {
        int width =
GameWidget::anotherCon.block[GameWidget::anotherCon.type][GameWidget::anotherCon.process]
[i].x();
        int height =
GameWidget::anotherCon.block[GameWidget::anotherCon.type][GameWidget::anotherCon.process]
[i].y();

p.fillRect(25+width*10,25+height*10,10,10,GameWidget::color[GameWidget::anotherCon.color]
);


        pen.setWidth(0);
        pen.setColor("white");
        p.setPen(pen);
        p.drawRect(25+width*10,25+height*10,10,10);
    }


}
```

代码块 48 NextBlock 实现代码

6.2.2.7 背景音乐及初始化

这里使用 Qt 框架下的 QSound 类，来实现多线程播放背景音乐。用 QIcon 类实现图标的插入,初始化地图信息。与 GameWidget 的信号 signals::GameOver();

进行槽连接。代码如下：

```
int MainWindow::running=false;
int MainWindow::score=0;
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    this->resize(450,600);
    ui->next->hide();

    //启动背景音乐(循环播放)
    //QSound::play(SOUND_BACKGROUND);
    QSound *sound = new QSound(":/BG1.wav",this);
    sound->setLoops(-1);
    sound->play();


    connect(ui->game,&GameWidget::GameOver,[=](){
       ui->pushButton->setText("重新开始");
       ui->pushButton->setDisabled(false);
       //"游戏结束""最终得分为:%1"
//        QString str = "游戏结束最终得分为:%1"
       ui->label->setText(QString("游戏结束"));
       //QMessageBox::about(this,"游戏失败","分数为： " + QString::number(score)+" ");

       GameWidget::gameOver=false;

       for(int i=0;i<10;i++)
       {
           for(int j=0;j<20;j++)
           {
               Control::map[i][j]=0;
               Control::colorMap[i][j]=0;
           }
       }

       QMessageBox::about(this,"游戏失败","分数为： " + QString::number(score)+" ");

       score=0;
    });
```

```
    this->setWindowIcon(QIcon(":/Tetris.ico"));
    this->setWindowTitle("Tetris");
}
```

代码块 49 背景音乐及初始化

6.2.2.8 游戏开始按钮

点击按钮，匹配相应槽函数，主要实现信息的更新与显示。代码如下：

```
void MainWindow::on_pushButton_clicked()
{
    running=true;

    ui->game->timer = new QTimer(this);
    ui->game->timer->start(1000);
    update();
    ui->game->con.newBlock();
    ui->game->anotherCon.newBlock();
    ui->game->con.rePaint();

    connect(ui->game->timer,&QTimer::timeout,[=](){
        ui->game->con.move(2);
        update();
    });

    ui->next->show();
    ui->pushButton->setDisabled(true);
    ui->label->setText("游戏进行中");
}
```

代码块 50 开始按钮代码

6.2.2.9 键盘操作控制

主要实现 W/S/A/D 于↑/↓/←/→操作绑定，调用在控制类中的 move（）函数实现，代码如下：

```
void MainWindow::keyPressEvent(QKeyEvent *e){
    if(running)
    {
        if(e->key()==Qt::Key_Up||e->key()==Qt::Key_W)
```

```
        {
            ui->game->con.move(1);
        }
        else if(e->key()==Qt::Key_Down||e->key()==Qt::Key_S)
        {
            ui->game->con.move(2);
        }
        else if(e->key()==Qt::Key_Left||e->key()==Qt::Key_A)
        {
            ui->game->con.move(3);
        }
        else if(e->key()==Qt::Key_Right||e->key()==Qt::Key_D)
        {
            ui->game->con.move(4);
        }
        update();
        ui->label_2->setText(QString("当前分数:%1").arg(score));
    }
    if(e->key()==Qt::Key_Space)
    {
        if(ui->game->timer)
        {
            if(running){
                running=false;
                ui->game->timer->stop();
                ui->label->setText(QString("游戏暂停"));
                update();
            }
            else
            {
                running=true;
                ui->game->timer->start(1000);
                ui->label->setText("游戏进行中");
            }
        }
    }
}
```

代码块 51 键盘控制代码

6.3.3 类图关系

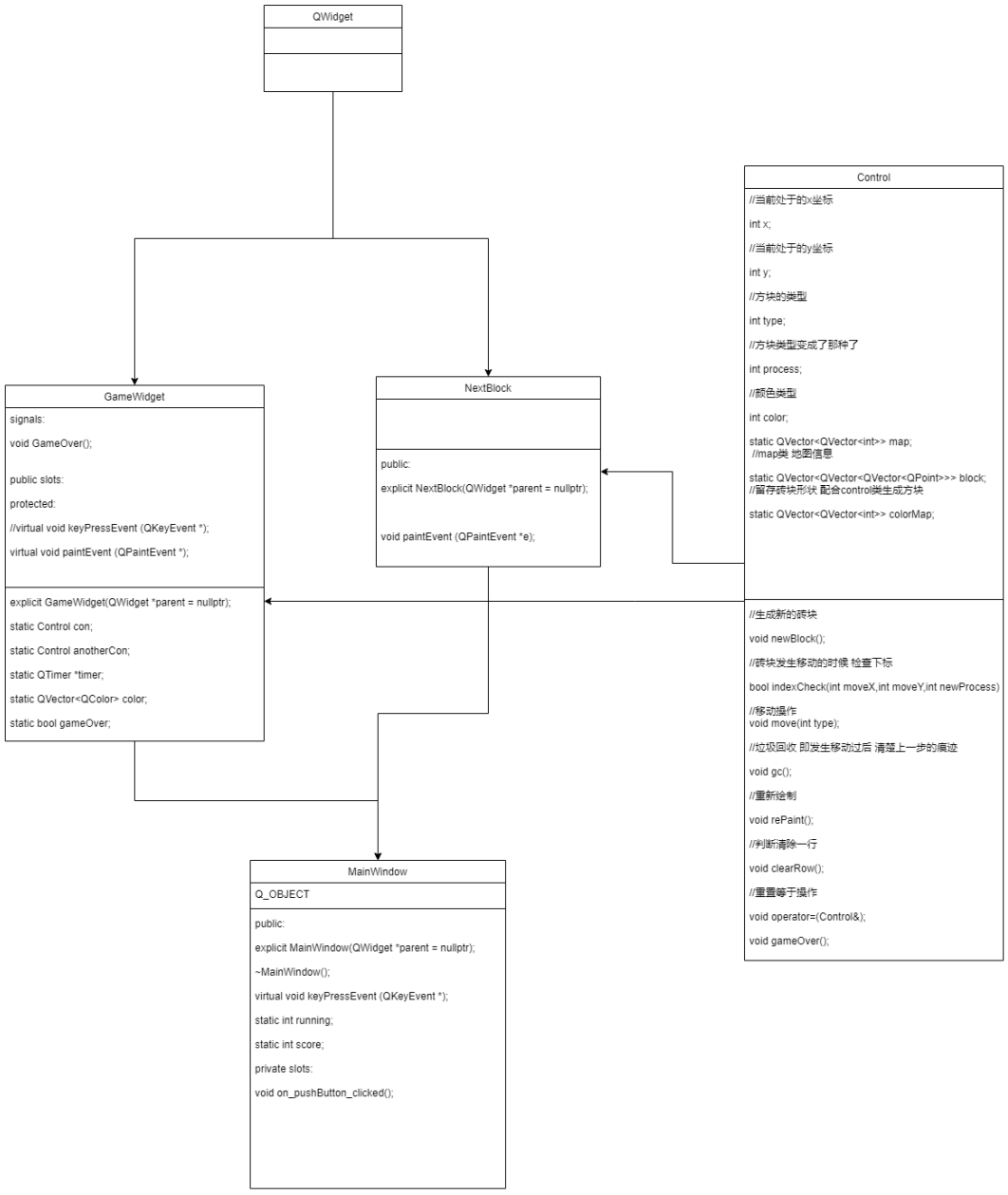


图 57 类图关系

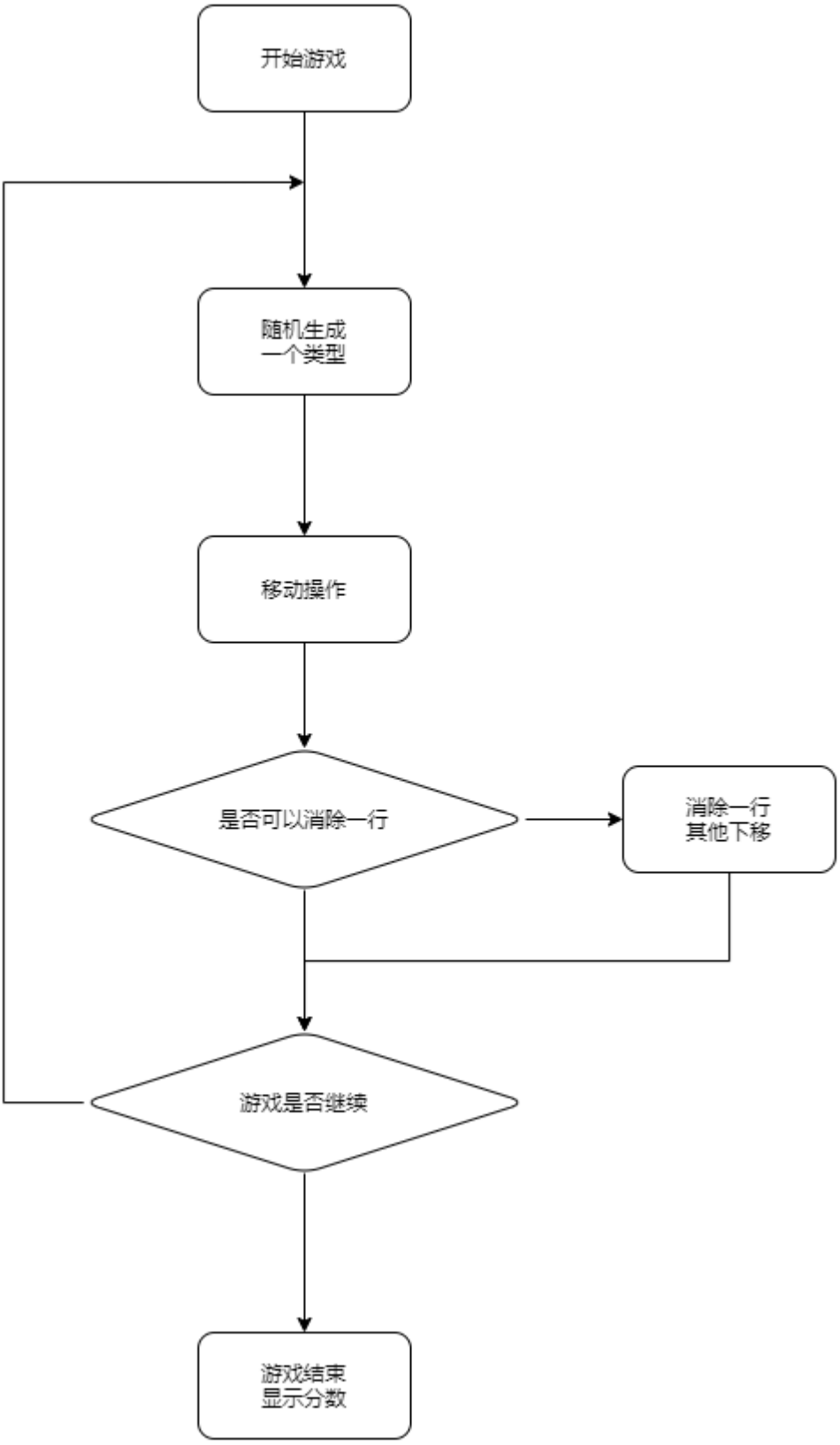


图 58 类关系流程图

6.4 程序展示

一些操作实际展示，展示效果如下图：

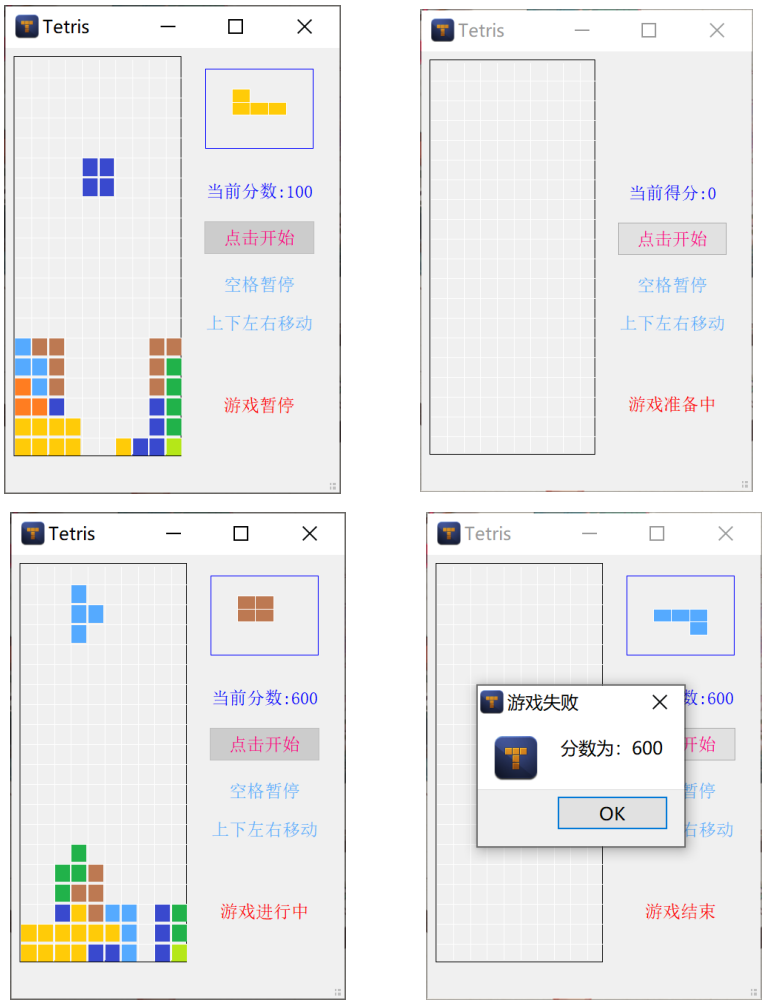


图 59 成果展示图

6.5 总结思考

通过使用 Qt 应用框架实现俄罗斯方块的基本游戏结构，对页面进行处理，采用继承思想，实现了游戏界面类的开发。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。