# 《数据结构与算法设计》
# 课程设计总结


学号：**2251079**

姓名：隋建政

专业：计算机科学与技术


2024 年 7 月

# 目　录

# 第一部分 算法实现设计说明

## 1.1 题目

9 号题目

几种排序：要求随机输入一组数据，随时给出某一趟排序的变化情况

（1）直接插入排序、折半插入排序、希尔排序；

（2）冒泡排序、快速排序；

（3）简单选择排序

## 1.2 软件功能

（1）可视化窗口

本软件采用 QtC++的方式实现主体窗口的可视化，将所有组件和变量添加在继承自 *QMainWindow* 的 *MainWindow* 类中，并通过运行 *MainWindow* 对象将所有组件展现在可视化窗口中。

为了使整个页面更加美观，在 *MainWindow* 类中添加了三个 *QPixmap* 对象，对整体的页面进行区域的划分，也方便后续的开发。包括 *inputMap*（包含输入框以及所有用户可交互的按钮等）、*sortInfoMap*（包含排序的各项信息，在后续进行详细介绍）、*arrElemMap*（包含数组元组的信息）。其中 *inputMap* 主要通过 ui 设计功能对标题、按钮、输入框等组件进行编排设计，而其余 *QPixmap* 对象则通过重写 *paintEvent()*函数的方式进行设计，在 *paintEvent()*函数中我设定了框图的颜色、大小、位置以及文本的字体和排版。这种差异主要是考量到 *inputMap* 在程序运行全程保持一致，而其余二者则需要跟随程序运行实时地发生变化，通过重写 *paintEvent()*函数的方式也方便后续调用 *update()*函数对画面进行更新。

此外，画面的主体留给了本次可视化的主角——排序算法的可视化动画。在此定义了 *QGraphicsView*、*QGraphicsScene* 对象并实现了 *BarList* 类，这些组件都将在后续进行介绍。总而言之，通过自行定义的 *generateBars()*函数，就可以绘制所有所需的图形啦。

以下是最终程序页面的整体展示。

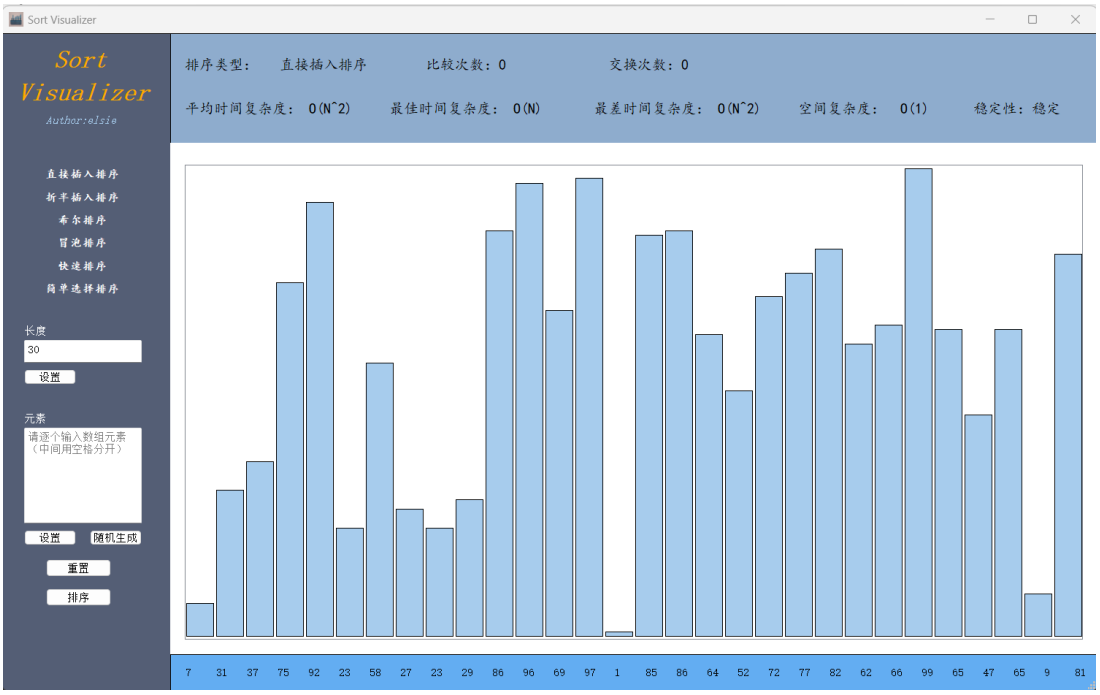



图 1 排序算法可视化程序页面的整体展示

（2）数组的输入

为了实现数组输入的功能，我首先实现了一个 *Array* 类，*Array* 具体实现将在后续给出。通过 *Array* 类的方法，用户可以对参与排序的数组进行自行设定与重置。并通过定义按钮的槽函数实现按钮功能，在两个“设置”按钮的槽函数中，通过读取文本框中的内容，调用 *Array* 类的方法对数组进行初始化设置。

数组的元素设置我规定了两种方式，用户自行输入与随机设置。为了避免产生歧义，这两种方式都需要先输入数组长度并设置才可以进行，否则就会弹出警告框。也就是具有以下的程序运行逻辑。

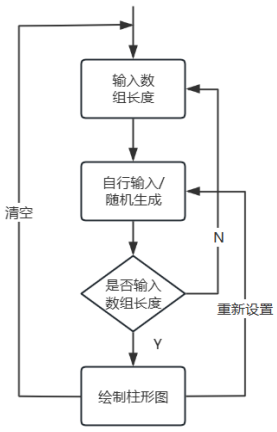



图 2 数组输入功能的逻辑设计

（3）排序方法的选择

对于不同的排序方法，实现了一个 *SortInfo* 类整合了所有的排序信息，并定义了排序信息包含的例如类型以及复杂度的宏。并在其中定义了更改不同类型排序的方法，以及将当前信息展示在可视化页面中所需要的方法，包含排序名称、平均时间复杂度、最佳时间复杂度、最差时间复杂度、空间复杂度以及稳定性。这部分内容被放在整体页面的右上方。

在本程序中，需要通过点击左侧输入栏中的各个排序类型按钮对排序方法进行选择。每当用户按下按钮，排序信息框就会相应的发生变化。

（4）排序过程的可视化

首先，在程序左侧输入框中放置了“排序”按钮，当用户设置数组完成后点击就可以进行排序。可以观察在在主体动画框中柱形图发生了相应的变化，并在上方排序信息框中实时更新排序过程中的交换次数与比较次数。

为了实现排序算法的可视化过程，引入了 *QTimer* 的计时器对象，并使用 *connect()* 函数将 *QTimer::timeout* 事件链接到相对应的槽函数中。六种需要实现的排序方法都有着各自的槽函数，当点击“排序”按钮后，将初始化 *timer* 对象，依据 *SortInfo* 对象给出的参数进行不同槽函数的选取，并将时间间隔设置为 500ms，调用 *timer* 对象的 *start()* 方法开启计时器。槽函数的具体实现将在后续的算法设计中详细给出。

## 1.3 设计思想

（1）软件的实现思路

在前文中介绍了软件可视化界面的编排设计，现在要详细说明一下软件整体的实现思路。首先软件本体被分为了输入区与展示区两个区域，其中输入区为用户与软件进行交互的区域，而展示区仅展示各类信息而无法与用户交互。

在程序内部，自行定义了 *Array*、*BarList*、*RecordList*、*SortInfo*、*Sorts* 等 *class*，旨在通过用户的设定，完成相应的软件功能。用户通过输入区对程序中参数进行设定的过程，实际上就是对 *MainWindow* 中各个 *class* 的实例化对象进行设定的过程。

输入区与展示区通过“排序”按钮进行连接，当用户按下“排序”按钮时。程序的展示区就会根据用户提前设定的对象进行可视化展现，包括上方排序信息的实时更新、下方数组信息的更新以及柱状图的交换动画。需要注意的时，为了避免在动画运行

过程中用户更改输入区中的内容导致错误，在展示区工作时输入区的功能将被禁止（这一点在程序中通过判断计时器 *timer* 是否为 NULL 来实现）。

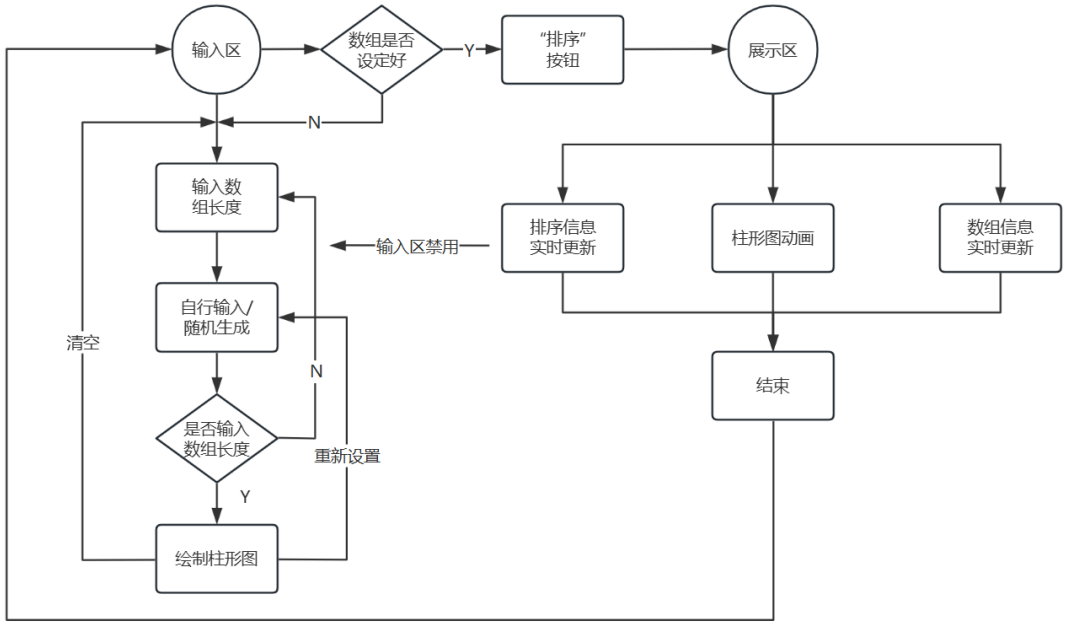



图 3 软件的整体运行逻辑示意图

（2）算法设计

在本次项目中，最核心的部分莫过于排序算法的设计，即如何在保证排序算法正确性的同时将其用动画的方式进行可视化。在上文中已经提到安排了计时器 *timer*，并通过将 *QTimer::timeout* 事件与槽函数相连接进行动画的绘制，因此算法实现的核心问题便是如何构造槽函数，下面将逐一进行详细解析。

**直接插入排序**

直接插入排序的算法思路为依次将元素插入到前方已经排好的有序数列中，对于一个元素，其找寻插入位置的方式为逐个与前方元素进行比较，直到找到一个不大于它的元素。因此直接插入排序的平均时间复杂度为 O(n^2)，最佳时间复杂度为 O(n)（已排好序的数列）。

在可视化的实现过程中，为了凸显直接插入排序的特点，忽略了内层循环与前述元素比较的过程，而是直接将元素插入到适当的位置，因此一个长度为 n 的数列最多进行 n 次插入。每当进入一次槽函数，便会将 i 对应位置的数值与前述元素进行比较，直到找到要插入的位置，更新插入之后的位置，这样就完成了一次插入。完成排序的数列会由原先的浅蓝色变为绿色。仅可视化外循环的方式有利有弊，这点会在后续折半插入排序的介绍中进行说明。

尽管忽略了内层循环的可视化，但内部逻辑依然按照直接插入排序的方式进行，即以 O(n)的方式遍历前方元素，这部分可以在比较次数中进行体现（可参考代码的实现）。

**折半插入排序**

折半插入排序是直接插入排序的改进版，其整体思路与直接插入排序一致，区别在于找寻插入位置时采用二分法进行查找而非逐一进行遍历，这使得一次查找的复杂度由 O(n)降为了 O(logn)。但其余实现均与直接插入排序一致，也就是说折半插入排序尽管需要的查找次数更少了，但交换次数与直接插入排序完全一样，也是 O(n^2)的复杂度。

前面说的忽略内层循环可视化的方式有利有弊在此处就体现出来了，由于基础逻辑一致，折半插入排序与直接插入排序的动画完全一致，这难以体现二者的差异性（仅能通过上方的排序信息进行体现，对于相同数列的排序折半的比较次数明显小于直接插入）。而其好处在于强调了两种算法的一致性，并且很好地凸显了插入排序的特点（即多次比较单次插入的特点）。下图为两种排序方式对 10-1 进行排序地记过，可以看到尽管二者的交换次数完全一致，但比较次数后者明显的小于前者。

图 4 直接插入排序与折半插入排序的结果比较

**希尔排序**

希尔排序同样是插入排序的改进型，但其实现过程与上述两种算法有着较大差别，这里选取了较为流行的希尔排序实现策略。首先需要计算间隔 h 的值，采用如下循环进行计算。

```
int h = 1;
while (h < len / 3)
    h = 3 * h + 1;
```

h 初始化完成后就可以开启三重循环排序，最外层的主排序循环每次将 h 除以 3 直到 h 为 0，这被称为主排序循环。内层双重循环中的外循环从 h 开始，到 len 结束。

而内循环则将距离为 h 的元素进行比较和交换，这个操作相当于在数组中同时进行多个不同间隔的插入排序。这个最内层的循环也是槽函数的最小运行单位，当其参数不满足内层循环的条件时便会依次跳出到外层循环。通过这种有些类似于递归函数的方式（一次次调用槽函数），实现多重循环的效果，便是所有槽函数运行的基础逻辑。

由于本次实现的基础单位为最内层循环，因此希尔排序的可视化过程将会包含所有的交换步骤，这里也能看出他与传统的插入排序算法之间有着较大的差异。

**冒泡排序**

冒泡排序的算法思想也非常简单且形象，即将相邻元素两两进行比较、交换，每趟“冒泡”都将最大的一个元素排到最后，因此平均时间复杂度为 O(n^2)。冒泡排序的槽函数的实现思路也非常简单，就是对冒泡排序双重循环的模拟（也是类似递归函数的方式）。在这里解释一下之前没有提到的问题，数组的元素交换非常简单，其实柱状图的交换也并不难实现，通过 *QGraphicsRectItem* 对象的 *setRect()* 方法对柱形的高度进行设置即可，这里通过一些计算求出柱体的宽度、间隔等参数，并根据最大值等比例进行高度的设置，一切计算都需要依照具体的数组（即根据数组本身的不同参数会有所不同）。

冒泡排序的可视化也是十分详尽的，双重循环都有所体现，排序完成的柱体会由原先的浅蓝色变为绿色。

**快速排序**

快速排序是本次实现中较为困难的一个，其算法思想并不复杂，每轮排序其实是一个划分的过程。选取一个基准元素 *pivot*，一次划分将比 *pivot* 小的元素与比 *pivot* 大的元素分别放置在 *pivot* 的左右两侧。之后在分别将左右两侧的子数列以相同的方式进行排序。快速排序利用了分治的思想，这也是可视化实现的难点，因为算法往往会使用到递归函数，而由于 *QTimer* 计时器的特性，在递归函数中使用会出现错误。在尝试了多种方法无果后，决定利用牺牲一些内存的方式进行解决。

首先将元素组进行复制，复制到一个新的 *Array* 对象 arr_。然后将 arr_按非可视化的快速排序算法进行排序，然后将排序过程中涉及到的元素交换全部用 *RecordList* 对象的实例 record 进行记录。由于对所有的交换数对进行了记录，因此快速排序的槽函数要做的工作就十分简单了，只需要每次取出一对交换数对进行交换即可。

这样的实现尽管难以称得上漂亮，但却十分有效，快速排序的可视化呈现也十分详尽，所有的交换步骤都有所体现。快速排序利用了分治法的思想，平均时间复杂度为

O(nlogn)，但由于其需要维护一个栈空间，因此本身也需要 O(logn)的空间复杂度。

**简单选择排序**

简单选择排序的算法思想也并不复杂，就是每次选出最小的元素，与首位元素进行交换。每次选择的复杂度为 O(n)，共需进行 n 次选择，因此时间复杂度为 O(n^2)，值得注意的是，由于简单选择排序的排序过程中需要与首位元素进行交换，因此排序整体是不稳定的。

简单选择排序的槽函数同样只注重了外层循环，因为内层循环只是进行元素的选择而没有交换的过程，这点与插入排序有些类似。然而简单选择排序最多只需要进行 n 次交换（n 为数组长度），这大大少于插入排序的交换次数。每次进入槽函数，都找到最小的一个元素并将其与首位元素进行交换（首位指 i）。

在这里需要说明一下，槽函数的实现离不开 MainWindow 中定义的全局变量（MainWindow 中的全局）诸如 i、j、h 的设置，他们的存在令槽函数模拟多重循环的算法思想成为了可能。

## 1.4 逻辑结构与物理结构

（1）*Array* 类

*Array* 类为记录用户设置数组的类，也是排序算法应用的对象。在本次程序中的逻辑结构为线性结构（因为排序算法本身都是基于线性结构的操作），并选取顺序结构作为物理结构，因为顺序结构存储的线性表实现起来更为简单，根据位置获取元素时的复杂度为 O(1)，且便于两个元素交换的实现，交换操作在排序算法中是十分普遍的，而链式结构的交换操作就会较为复杂。

下面给出 *Array* 类头文件中的声明，给出了涉及的变量以及各类方法。

```
class Array {
private:
    int* data; // 数组元素
    int num; // 数组长度
    bool ready; // 数组是否设置好
public:
    Array(void); // 构造函数
    Array(int n); // 构造函数（测试用）
    ~Array(void); // 析构函数
    void randomInit(int n); // 随机初始化
    int getKth(int k); // 获取第 k 个元素
```

```
    int getLen(void); // 获取数组长度
    bool isReady(void); // 判断数组是否设置好
    void changeKth(int x, int k); // 更改数组第 k 个元素为 x
    void display(void); // 在 console 中打印数组（测试用）
    void swap(int i, int j); // 交换数组中 i 和 j 的值
    void changeNum(int n); // 改变数组长度（输入框用）
    void changeElem(const QString str); // 改变数组元素（输入框用）
    void paintText(QPainter &painter); // 绘制数组长度信息（gui 用）
    void paintElemText(QPainter &painter); // 绘制数组元素信息（gui 用）
    void reset(void); // 重置数组
    void warning(void); // 弹出警告框（数组未设置时）
    int getMax(void); //  获取最大值
    void copy(Array &arr); // 复制数组
};
```

（2）*BarList* 类

*BarList* 类为记录柱形图的类，其存在旨在替代 *QVector<QGraphicsRectItem>* 这一 Qt 自带的 *Vector* 数据结构。为了实现 *BarList* 类的功能，先对其结点结构体进行定义，命名为 *barItem*，包含 *QGraphicsRectItem* 类型的指针*item 以及指向下一个结点的指针*next，其声明与构造函数如下所示。

```
struct barItem{
    QGraphicsRectItem *item;
    barItem *next;
    barItem(QGraphicsRectItem *i, barItem *n){
        item = i;
        next = n;
    }
};
```

该类的逻辑结构同样为线性结构，因为柱形图中的元素是与前面设计的 *Array* 类中的元素一一对应的。在结构体中出现了*next 指针，因此很明显 *BarList* 类的物理结构选取了链式结构，这样做的目的是方便增加新的柱形图案，简化初始化时的流程。（后来发现其实使用顺序结构也不是不可以，但与后面的 *RecordList* 保持一致性索性就没有改）。以下给出 *BarList* 类的声明。

```
class BarList{
private:
    barItem *head; // 头指针
```

```
    int num; // 链表长度
public:
    BarList(void); // 构造函数
    ~BarList(void); // 析构函数
    barItem *getHead(void); // 获取头指针
    barItem *getKth(int k); // 获取第 k 个元素的指针
    int getNum(void); // 获取链表长度
    void addBar(barItem *bar); // 添加元素
    void clear(void); // 清空链表
    void setKth(QGraphicsRectItem *bar, int k); // 设置第 k 个元素的值
};
```

（3）*RecordList* 类

*RecordList* 类实际上是快速排序专供的数据结构，也就是前文提到的不得已的解决办法，其存在也是为了取代 *QVector<int>* 的使用，用于存储快速排序算法过程中所有需要交换的项。其实现与 *BarList* 类类似，同样定义了结构体结点，并实现了一系列实用的方法。*RecordList* 的逻辑结构为线性结构，物理结构为链式结构。*RecordList* 确实有十足的理由使用链式结构，因为需要实时地添加元素，这个操作如果使用顺序结构则较为困难，而且容易造成空间地浪费。

以下给出 *RecordList* 类的声明与结点结构体的定义。

```
struct recordItem{
    int value;
    recordItem *next;
    recordItem(int v, recordItem *n){
        value = v;
        next = n;
    }
};

class recordList{
private:
    recordItem *head; // 头指针
    int num; // 链表长度
public:
    recordList(void); // 构造函数
    ~recordList(void); // 析构函数
    recordItem *getHead(void); // 获取头指针
```

```
    int getKth(int k); // 获取第 k 个元素
    int getNum(void); // 获取链表长度
    void addRecord(int v); // 添加记录元素（两位合成一个记录）
    void clear(void); // 清空
};
```

## 1.5 开发平台

应用程序开发框架：Qt（版本：5.15.2）

开发平台：Qt Creator（版本：5.0.2）

编程语言：C++

运行环境：Windows11

## 1.6 系统的运行结果分析说明

（1）调试及开发过程

本次应用最初先用控制台演示的形式进行验证，在这一部分完成 *Array*、*Sorts*、*SortInfo* 类的编写，并通过 console 控制台进行验证。这一部分代码在代码注释中均有所标注，大部分测试代码在正式程序中并不需要使用。通过前期的控制台验证，实现了 *Array* 类的功能，完成并测试了各类排序算法，并实现了通过 *SortInfo* 更改排序方式的功能。

完成初期的理论验证后，我首先着手 gui 页面的开发，首先是对整体静态页面的设计，包括框图位置、字体编排、颜色以及按钮等组件，这部分主要通过 Qt 的 ui 设计功能以及 QPixmap 组件进行，在前文中已经有所介绍。

完成前端的设计后，我首先实现的是输入区的部分，这部分的正确性通过 ui 页面下方的数组显示进行验证，输入区的功能保证程序可以正确读取用户的输入并设置相应的数组。最后实现的部分是程序的核心，也就是展示区动画演示的功能，这部分的调试是利用多组不同的数组对各个排序算法进行测试，避免 bug 的产生。同时尝试多种用户交互逻辑（按下按钮的书顺序），以发现交互过程中存在的漏洞并计时进行修改。

（2）软件成果

由于动画效果难以演示，这里仅给出排序运行最后的结果，进行排序的数组以一组随机数为例，可以观察到各个排序对同样的数组所进行的交换次数与比较次数各不相同，若希望了解实际的动画演示效果，可以观看本人上传到 b 站的演示视频，视频链

接为[排序算法可视化演示_哔哩哔哩_bilibili](BV1zmHve5EgH)。

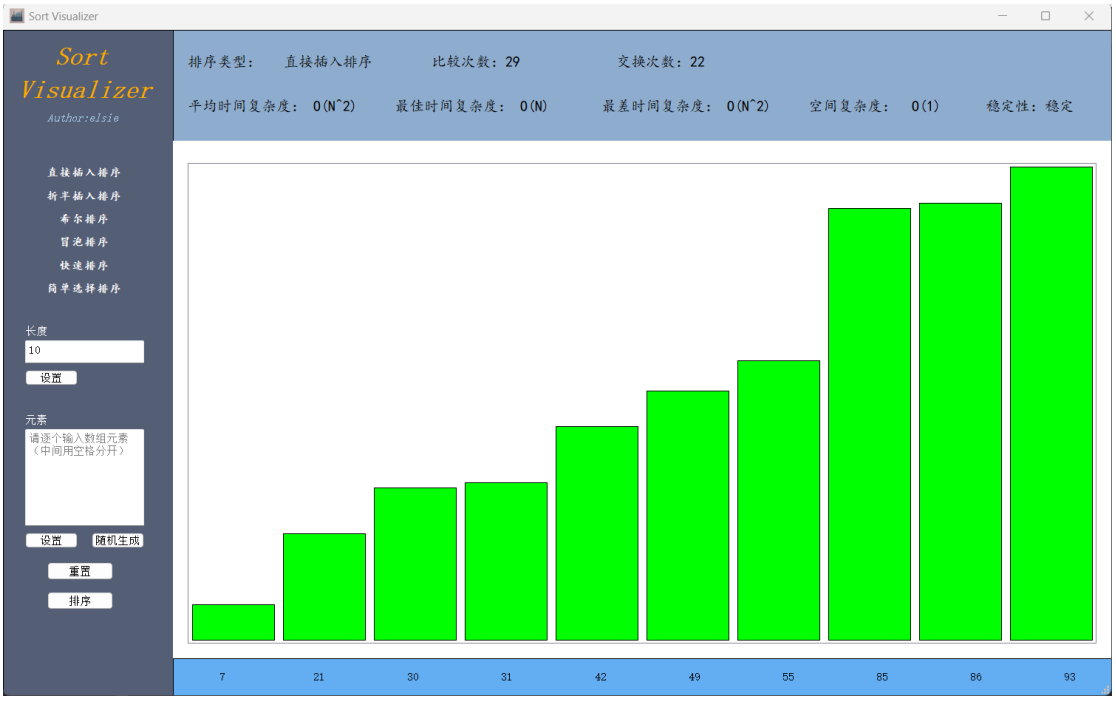



图 5 直接插入排序的结果

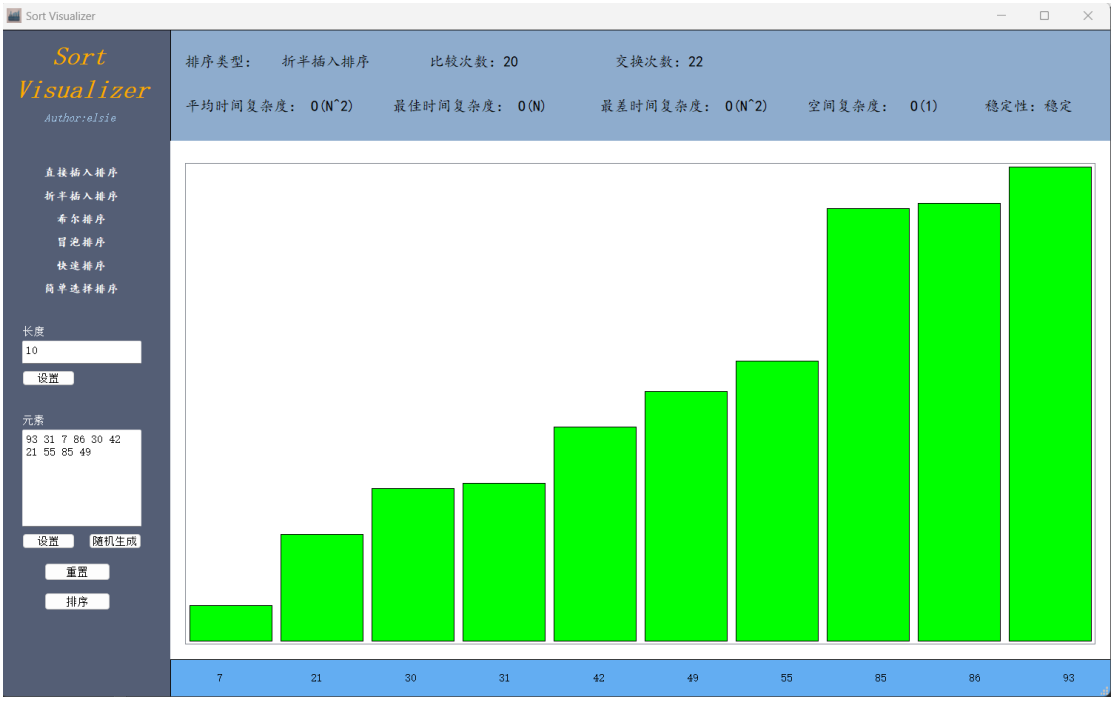



图 6 折半插入排序的结果

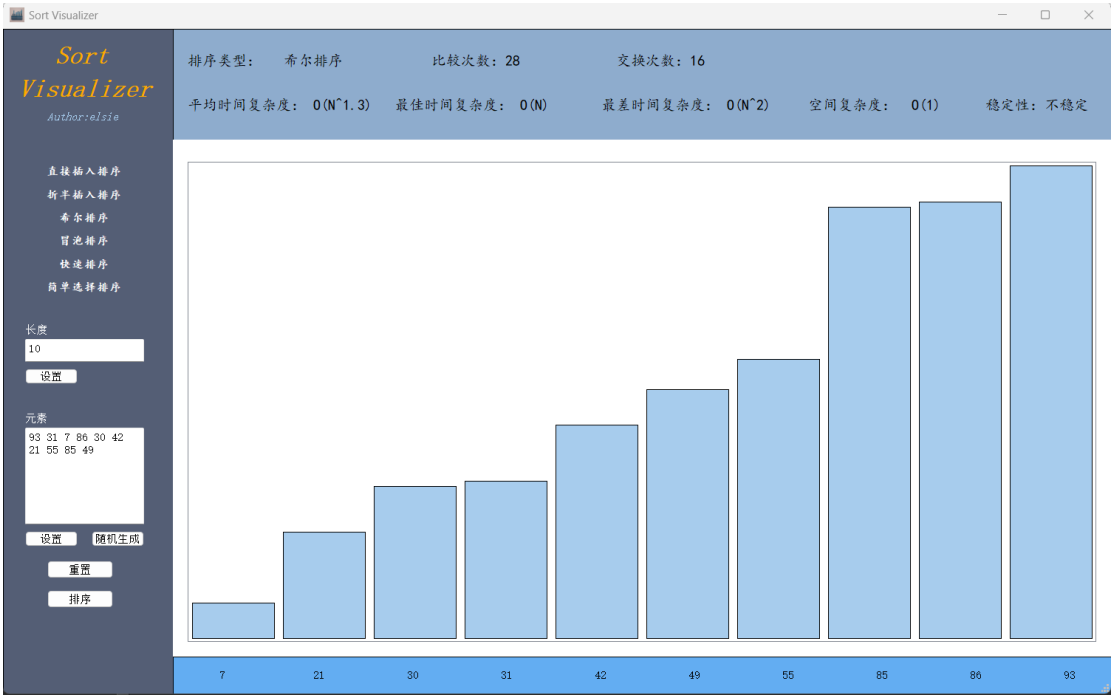



图 7 希尔排序的结果

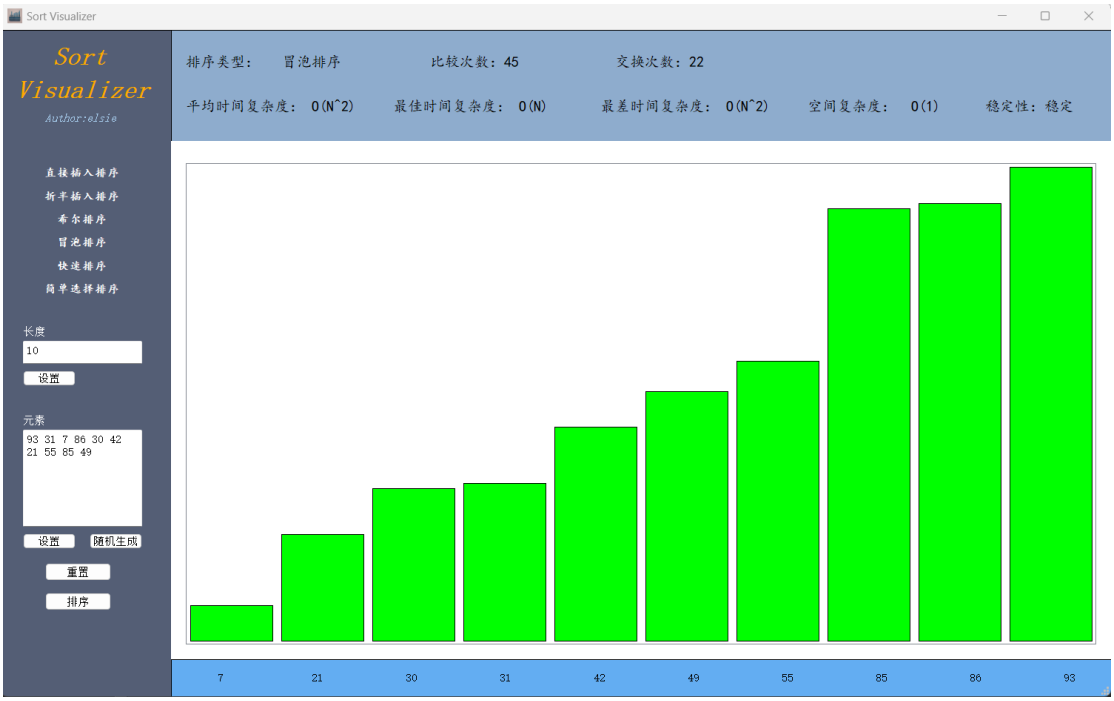



图 8 冒泡排序的结果

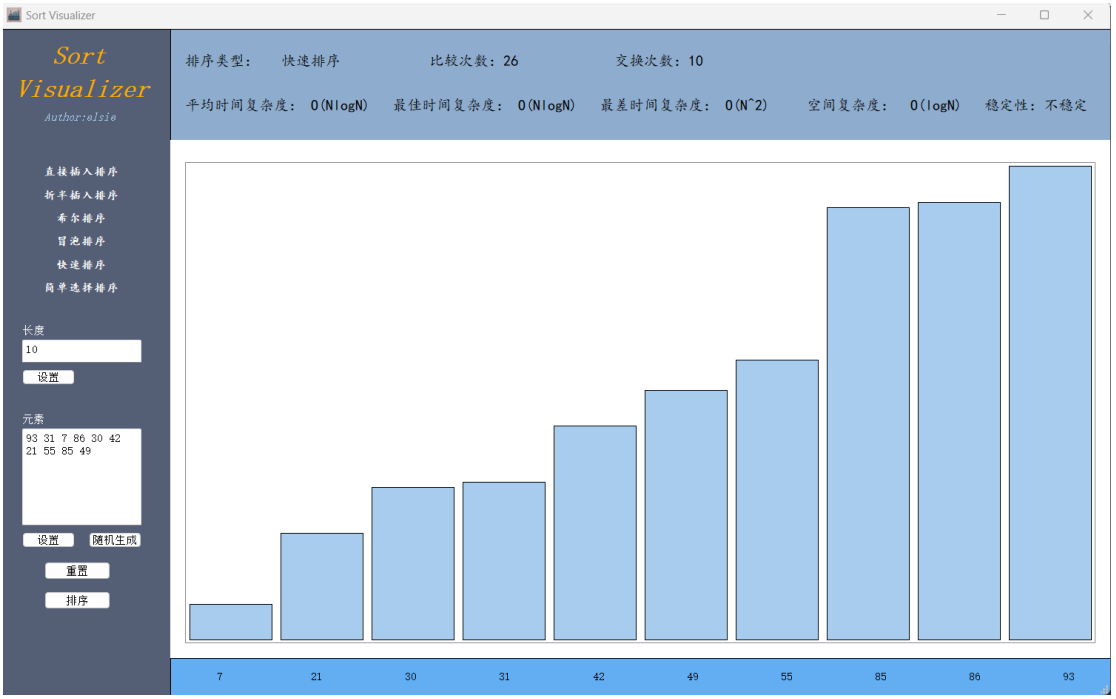



图 9 快速排序的结果

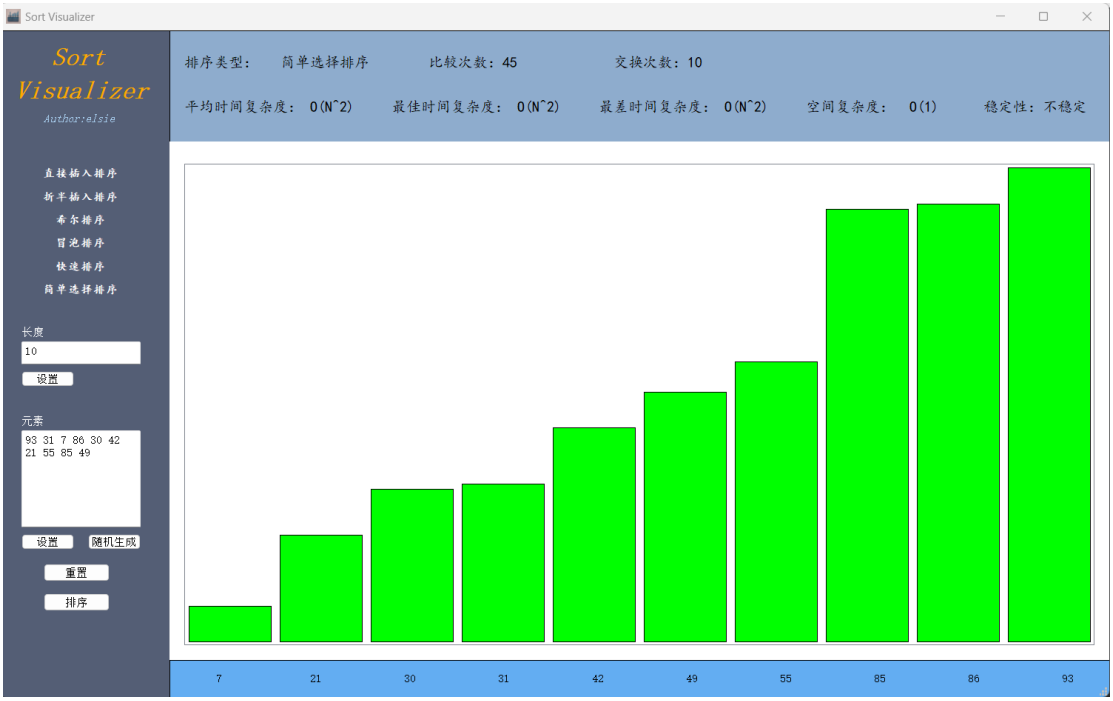



图 10 简单选择排序的结果

此外，程序本身还对一些可能出现的错误进行了处理。首先是前文提到的输入区可能干扰展示区的问题，因此每当用户按下输入区的按钮时，会首先判断展示区是否在工作，若展示区在工作则按下按钮不会产生任何的效果。

然后是输入区的一些容错性处理，首先为了避免发生歧义，用户在设置数组元素

时首先要对数组长度进行设置，如果缺少这一步骤则将弹出警告框。

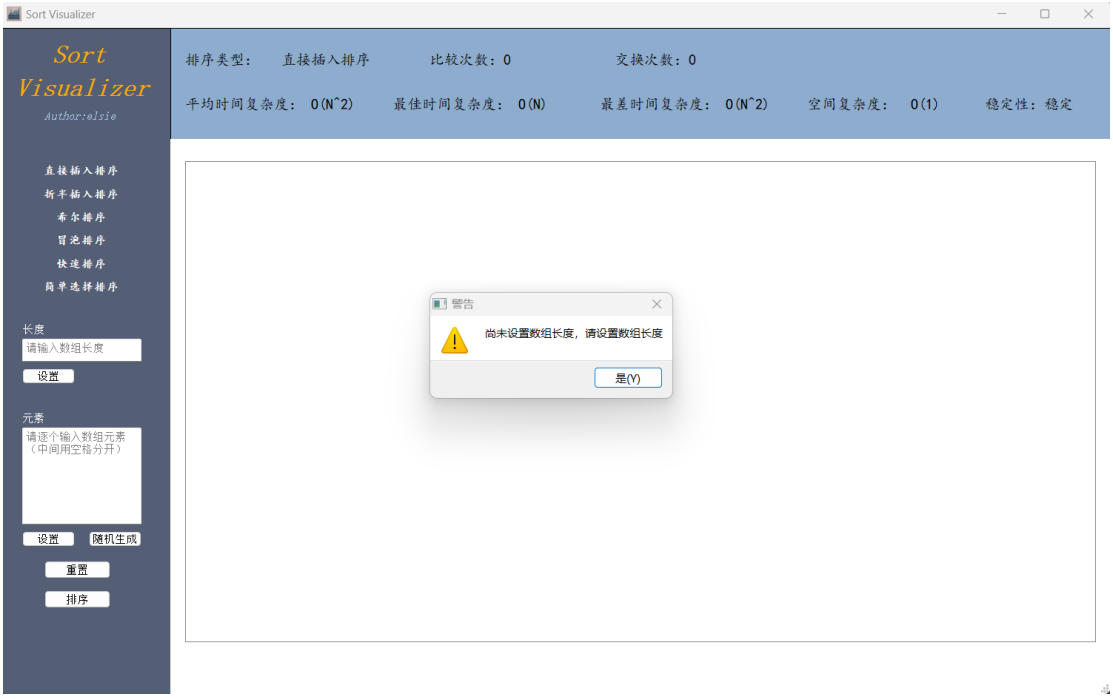



图 11 未设置长度就设置元素回弹出警告框

当用户设置好数组长度时，*Arrray* 类就会自动地生成对应长度的数组并将所有值设置为 0。此时若用户自行设置的元素数量与先前设置的长度不相同，程序也有对应的保护手段。例如当元素数量大于设定长度时，输入会自行忽略超出的部分。

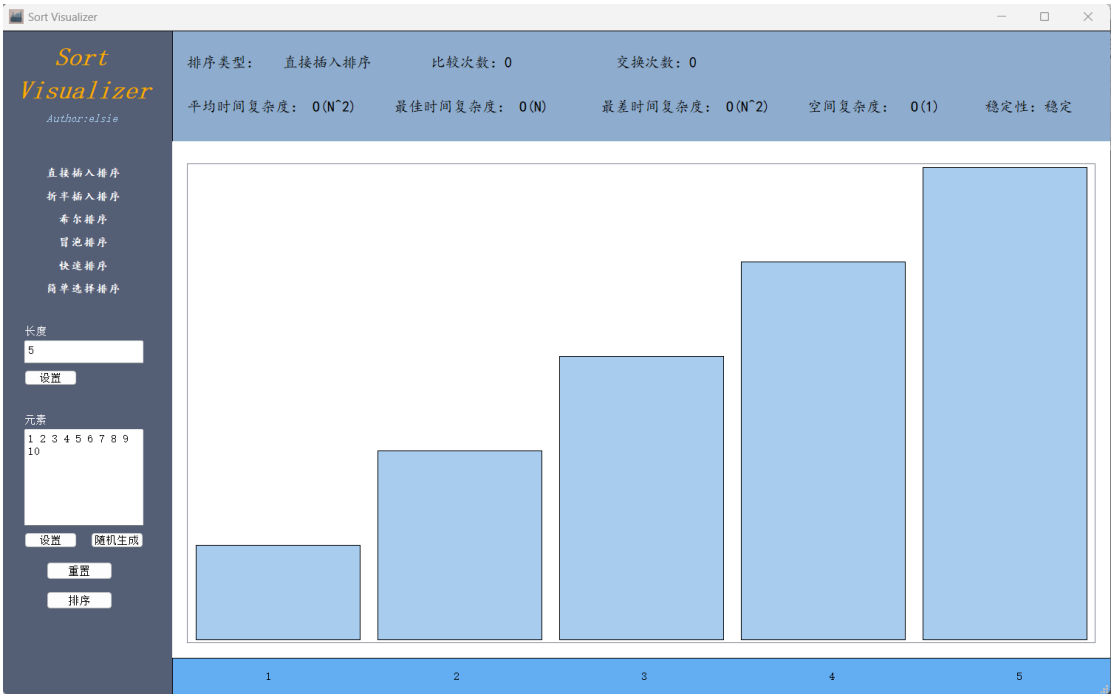



图 12 自动忽略超出设定长度的元素

而当用户输入的元素数量小于设定的长度时，程序会自行把未设置的元素记录为0，而不会报错。

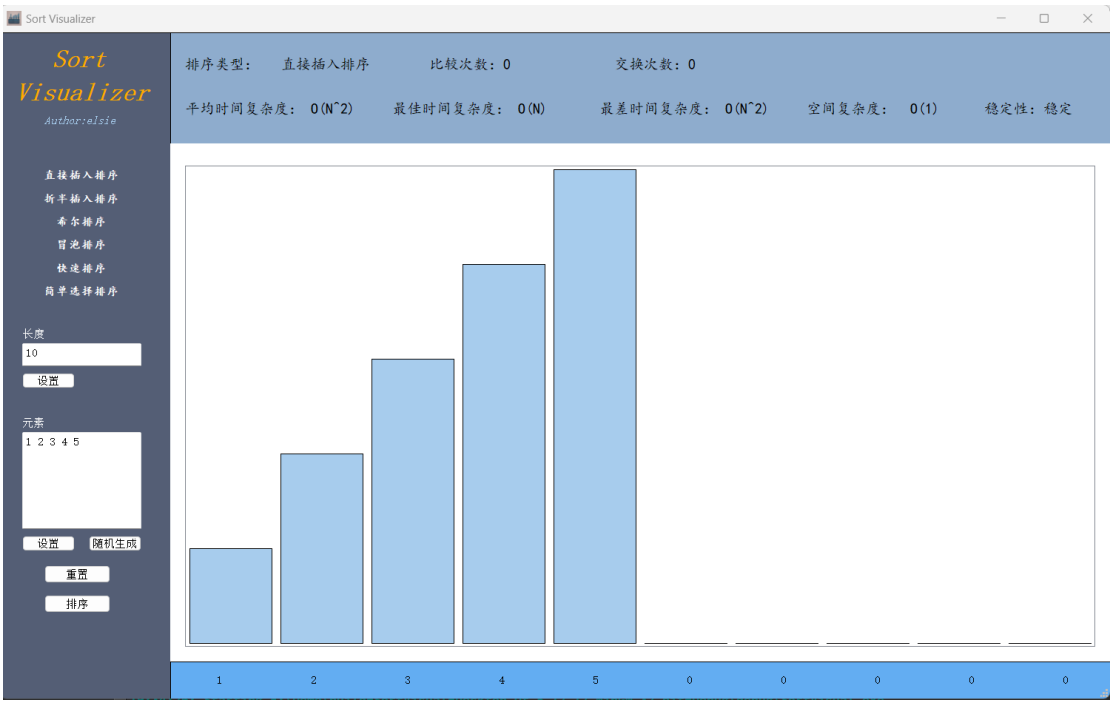



图 13 将未设定的元素补齐为 0

此外，输入本身也进行了一定的容错性处理。输入框中显示不同元素之间要用空格间隔，实际上程序会将所有非数字的字符算作空格处理，而连续多个空格也不会影响输入的正确性（因此程序不能输入负数，因为负号会被当作空格处理）。

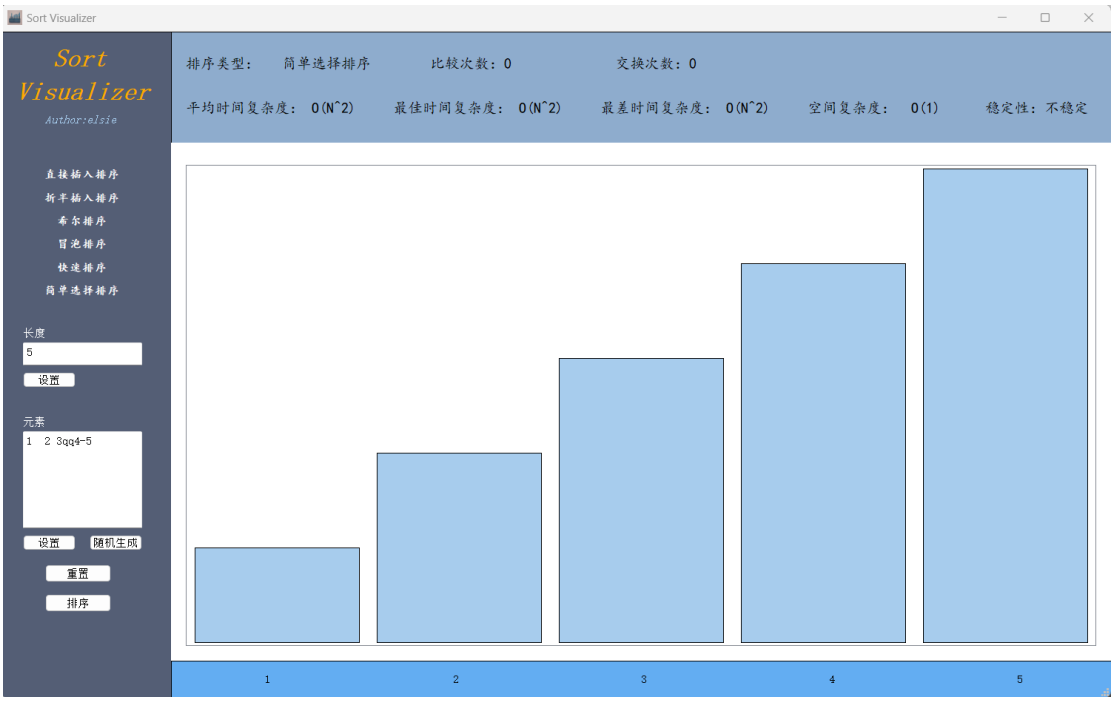



图 14 输入的容错性展示

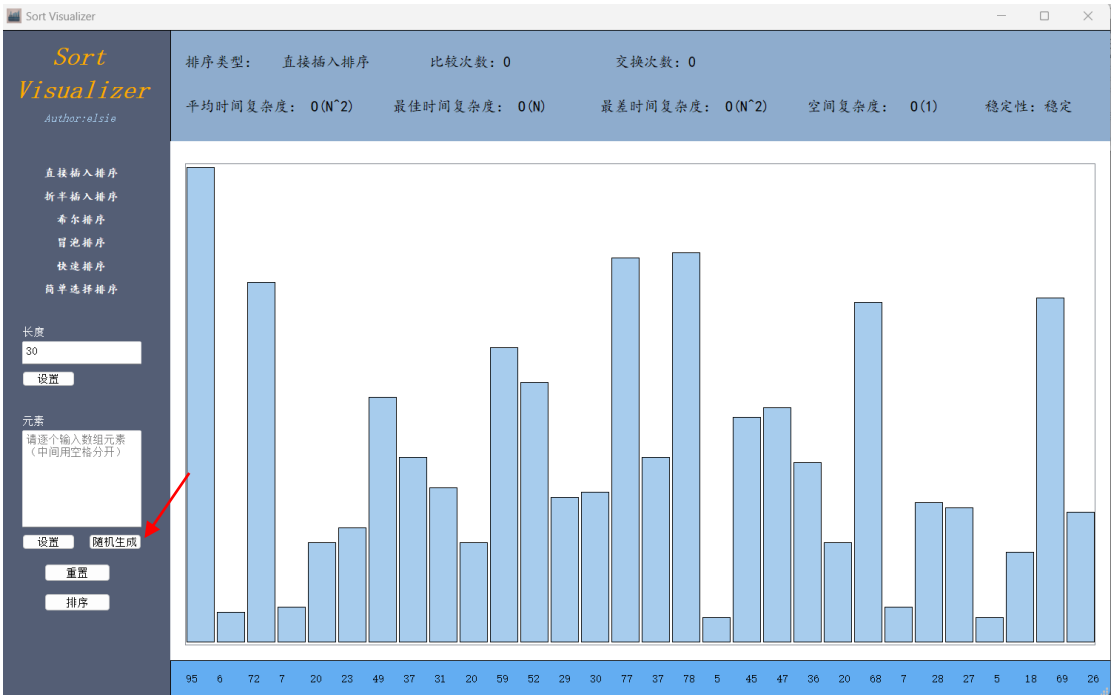
## 1.7 操作说明

（1）输入

ui 页面左侧区域为输入区。想要初始化数组，首先需要输入数组长度。在上方的输入框中输入想要设定的数组长度，并点击“设置”按钮进行设置。


设置完数组长度就可以对数组元素进行设定了，对数组元素进行设定有两种方式，一种为在下方输入框中逐个输入元素，相邻的元素用空格隔开，并点击下方的“设置”按钮。

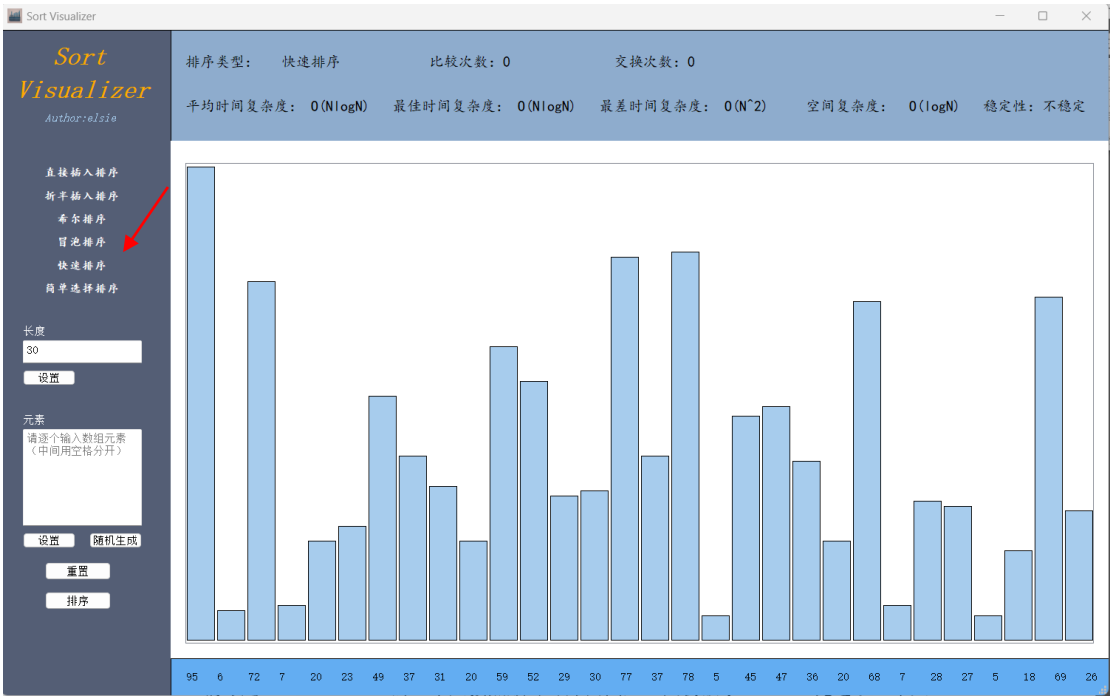
或者直接点击“随机生成”按钮，这样会根据设定的长度产生一个随机的数组。



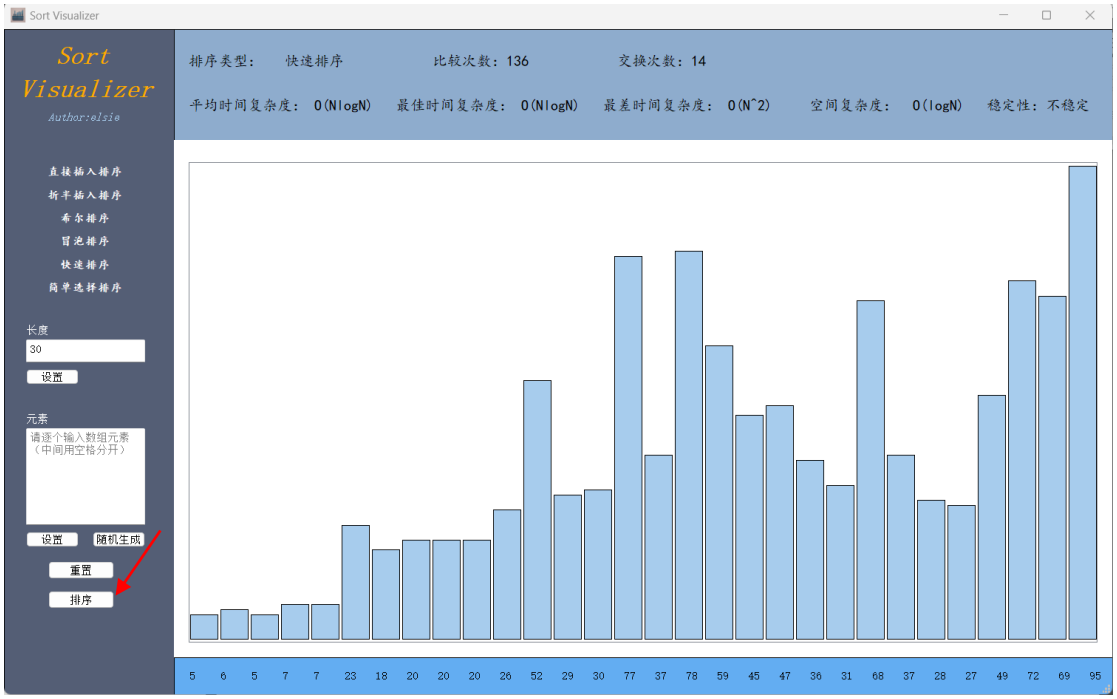

当 ui 下方的数组元素信息出现，并且中间区域的柱状图正确显示，就说明数组设置完成。

（2）排序

首先要选择想要使用的排序类型，点击左侧输入区上方相对应的按钮即可。例如这里选择快速排序，当 ui 上方的排序类型栏变为“快速排序”就说明排序类型设置成功。



设置完排序类型后，就可以点击“排序”按钮，这时中间的柱状图就会发生交换，将排序算法进行可视化演示，此时只需要等待排序动画完成即可。




（3）更改数组

按下“重置”按钮将把所有信息进行重置，包括数组、柱状图、排序信息等等，但此时先前设置的数组长度依然保留。


此时依然可以利用两种方式对数组进行设定，但需要注意的时，如果希望改变数组长度，要先更改上方输入框的内容，并点击“设置”按钮，否则数组长度会一直遵照

先前的设定。

# 第二部分 综合应用设计说明

## 2.1 题目

编号是 1，2，……，n 的 n 个人按照顺时针方向围坐一圈扔骰子（1-6），先选取一个人扔，根据扔的数字 m，从扔骰子的人开始从 1 沿顺时针方向顺序报数，报到 m 时停止报数，报 m 的人出列，然后在从他顺时针方向的下一个人扔骰子，扔完后从 1 开始报数，如此，直到剩下一个人胜出。设计一个程序模拟这一过程。

（1）通过输入框输入 1，2，……，n 个人；

（2）模拟整个游戏过程；

（3）按照出列的顺序输出各个人的编号。

## 2.2 软件功能

（1）可视化页面

程序的可视化页面沿用了上个项目排序可视化的设计风格，但依照具体的功能及进行了一定的修改。根据不同的三个功能分区将整体界面划分为了三个部分，分别为左侧的输入区、中间的模拟区以及右侧的信息区。

其中输入区主要包含用户与程序交互所要用到的各类按钮以及输入框，通过 Qt 的 ui 设计功能进行实现；模拟区为展示游戏过程的核心区域，利用 *QGraphicsView*、*QGraphicsScene* 以及 *QGraphicsEllipseItem* 三套核心组件搭配自行实现的链表类进行展示；信息区主要用于打印游戏运行过程中的各类文字信息，包含出列的玩家编号以及姓名，也包括最终获得胜利的玩家信息。整体的界面设计如下图所示。

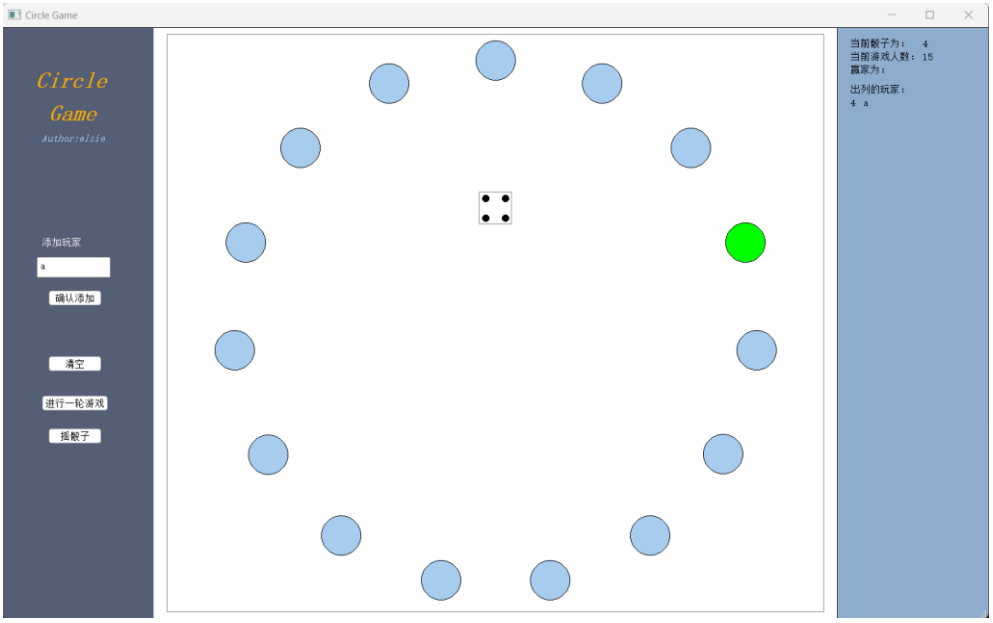


图 1 图形化界面的整体设计

（2）输入玩家

输入玩家信息的功能主要由左侧输入区进行实现，用户需要逐个输入参与游戏的玩家姓名，程序会为其自动生成编号并加入到圆圈中，每当用户添加完一个玩家后，在模拟区就会产生对应的变化。“清空”按钮可以清楚所有的玩家，并清空模拟区和信息区。“进行一轮游戏”按钮将向程序发出指令，在程序内部按照游戏规则运行的同时也在模拟区展示相应的动画。“摇骰子”按钮实际是一个测试用的功能，主要用于测试投骰子的动画演示以及功能。

（3）游戏过程的模拟

当用户按下“进行一轮游戏”按钮后，模拟区就会开始工作。首先会进行投骰子的操作，骰子的结果将会展示在模拟区中心偏上方的位置进行展示。然后代表玩家的圆圈将会根据骰子的结果，依照游戏规则进行演示。从起始玩家顺时针开始报数，报到数的玩家将由浅蓝色变为红色，而报到 m 的玩家将会被淘汰出局。此时下一个顺位的玩家将成为下一轮的起始玩家，其颜色变为绿色。这样就完成了一轮游戏的模拟。

（4）输出出列玩家

每当进行一轮游戏，就会在右侧信息区打印出列的玩家编号以及姓名。此外，当游戏玩家仅剩一名时，信息区将打印这名玩家的编号以及姓名，作为游戏最终的赢家。

## 2.3 设计思想

（1）软件的实现思路

正如前文所述，程序整体被划分为了三个功能区——输入区、模拟区以及信息区。这三个区域之间的交互可以用如下示意图进行表示。

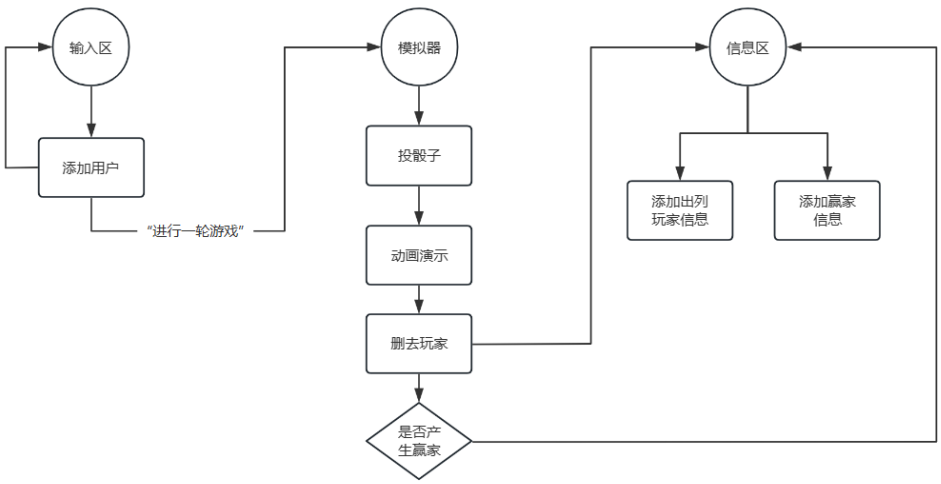


图 2 软件整体思路示意图

（2）数据结构的选取

本程序中最为核心的数据结构便是玩家序列的设计，这里选用循环链表代表玩家序列。因为本身玩家就是围成一个圈，当其逐一进行报数时会产生循环，因此选取循环链表是十分自然的想法。这里选用链表同时也便于玩家结点逐个进行插入，避免顺序结构产生空间上的浪费。

在本次程序中循环链表类被命名为 *Circle*，首先定义了 *Person* 结构体作为链表的结点，结构体中包含了玩家的姓名、编号以及指向下一位玩家的指针*next。其定义如下所示。

```
// Person 结构体
struct Person {
    std::string name; // 成员姓名
    int No; // 成员编号
    Person* next; // 下一位成员
    Person(std::string n, int no, Person* nx){
        name = n;
        No = no;
        next = nx;
    }
};
```

之后给出 *Circle* 类的各项变量与方法的声明。

```
class Circle {
public:
    Circle(void); // 构造函数
    ~Circle(void); // 析构函数
    int getNum(void); // 获取成员数量
    Person* getHead(void); // 获取头指针
    Person* getKth(Person* start, int k); // 获取第 k 个成员的指针（从 start 开始）
    Person* getKthfromHead(int k); // 获取第 k 个成员的指针（从 head 开始）
    void AddPerson(std::string name, int No); // 添加成员（姓名，编号）
    void addPerson(const QString name); // 添加成员（姓名，自动设置编号）
    void DeletePerson(Person* p); // 删除成员
    void create(int n); // 创建长度为 n 的 circle（测试用）
    void display(void); // 打印 circle 信息（console 测试用）
    void printPerson(Person* p); // 打印成员信息（console 测试用）
    Person* OneStep(Person* start, Dice* dice); // 进行一轮游戏（投骰子，淘汰一人）
    Person* play(Dice* dice); // 进行整个游戏（打印最后的赢家）
```

```
    void paint(QPainter &painter); // 绘制 circle 信息（测试用 Qt）
    void clear(void); // 清除所有成员
    int getPersonNum(Person* p);
private:
    Person* head; // 头指针
    Person* tail; // 尾指针
    int num; // 数量
    void getValue(Person*& p, std::string str, int n, Person* nextp); // 设置某一成员的值
};
```

这其中也包含一些在控制台测试时使用的方法，在正式的程序中并不会调用这些函数。

当然程序也实现了两个单链表类 *CircleList* 与 *PersonList*，与排序可视化中的两个单链表类似，这两个类的实现也是为了取代 *QVector* 的作用。其中 *CircleList* 用于记录模拟区中各个圆圈的绘图信息，其基础结点包含 *QGraphicsEllipseItem** 类型的变量，而 *PersonList* 则是用于记录出列的玩家信息，其基础节点包含 *Person** 类型的变量。这二者的操作都十分简单，仅涉及添加删除等，因此就选取单链表的数据结构进行实现，在此不多做赘述。值得一提的是，为了实现骰子的动画效果，骰子上的点也是 *CircleList* 对象，根据骰子的不同结果生成对应数量的黑点并设置相应的位置，这就使得骰子的展现变得十分美观了。

（3）算法设计

本程序的底层算法逻辑比较简单。首先是骰子的产生算法，这里调用了 *rand()* 函数，选取当前时间作为随机数种子，并加以处理使其结果范围在 1-6 之间，再根据生成的结果设置骰子的形象。这些步骤都通过 *Dice* 类进行实现。

利用循环链表模拟游戏规则的算法是程序运行的核心。其内部的底层逻辑就是根据骰子的点数，从起始位置找到相应的结点，将该结点删除，再将下一个结点作为返回值返回（设置为下一轮的起始结点）。这一过程通过 *Circle* 类的 *OneStep()* 方法进行实现，在这个方法中调用了 *getKth()* 以及 *DeletePerson()* 等方法。

而演示动画的实现同样依托计时器 *QTimer*，将 *QTimer::timeout* 事件连接到相应的 *step()* 槽函数上。槽函数首先要判断是否数到骰子的点数 m，若达到则要将该玩家添加至出列人员链表中，删除出列玩家，更新圆圈图并停止计时器。否则就要将报数的玩家颜色变为红色。值得注意的是，当骰子点数超过当前玩家数量时，报数完成一轮后要将所有玩家变回浅蓝色并继续进行报数。

这里需要对模拟图的构造进行一些说明，一个小圈就代表一个玩家，所有的小圈围成一个大圈，代表玩家围成的圈。圈中间的正方形是用来展示骰子结果的区域。所有小圈的半径和位置都经过实时的计算，会根据玩家的数量发生相应的变化。构造圆圈图的过程通过函数 *generateCircle()*进行实现。

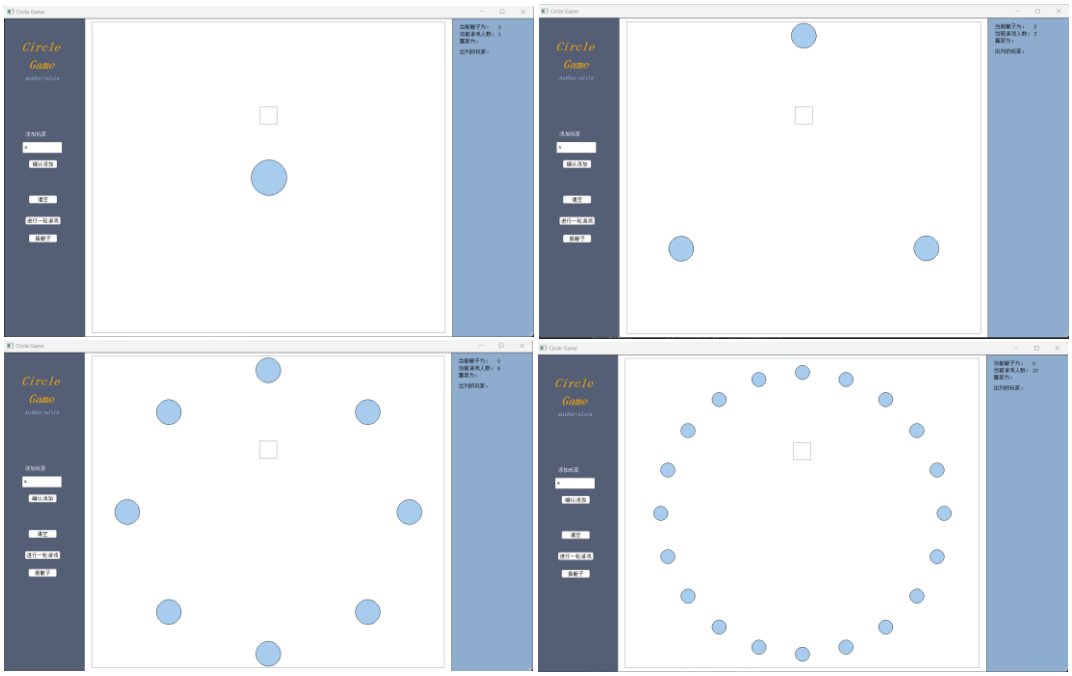


图 3 不同数量玩家的圆圈图

## 2.4 逻辑结构与物理结构

（1）*Circle* 类

逻辑结构：线性结构

物理结构：链式结构

（2）*CircleList* 类

逻辑结构：线性结构

物理结构：链式结构

（3）*PersonList* 类

逻辑结构：线性结构

物理结构：链式结构

## 2.5 开发平台

应用程序开发框架：Qt（版本：5.15.2）

开发平台：Qt Creator（版本：5.0.2）

编程语言：C++

运行环境：Windows11

## 2.6 系统的运行结果分析说明

（1）调试及开发过程

与前一个项目类似，首先利用纯控制台程序对程序的基础逻辑进行验证。主要是验证 *Dice* 与 *Circle* 类的功能正确性。随后的 ui 设计整体沿用了前一个项目的风格，只对具体功能组件的排版进行修改，这省去了重新设计 ui 的麻烦。

首先实现的是输入区与信息区的开发，通过为各个按钮添加槽函数并在右侧信息区实时地打印相关信息，观察右侧信息判断程序运行正确与否并及时进行修改。这一部分的调试实际还是非常顺利的，因为前期控制台程序的开发已经将要实现的类的功能编写得较为完善了。

最后就是实现程序的核心功能区——模拟区。这一部分调试的难点一部分在于对各个圆圈位置的计算，除此以外便是需要对用户的各种交互逻辑进行尝试，以便发现存在漏洞的交互组合并进行修改。

（2）软件成果

首先通过添加玩家功能添加十位玩家。

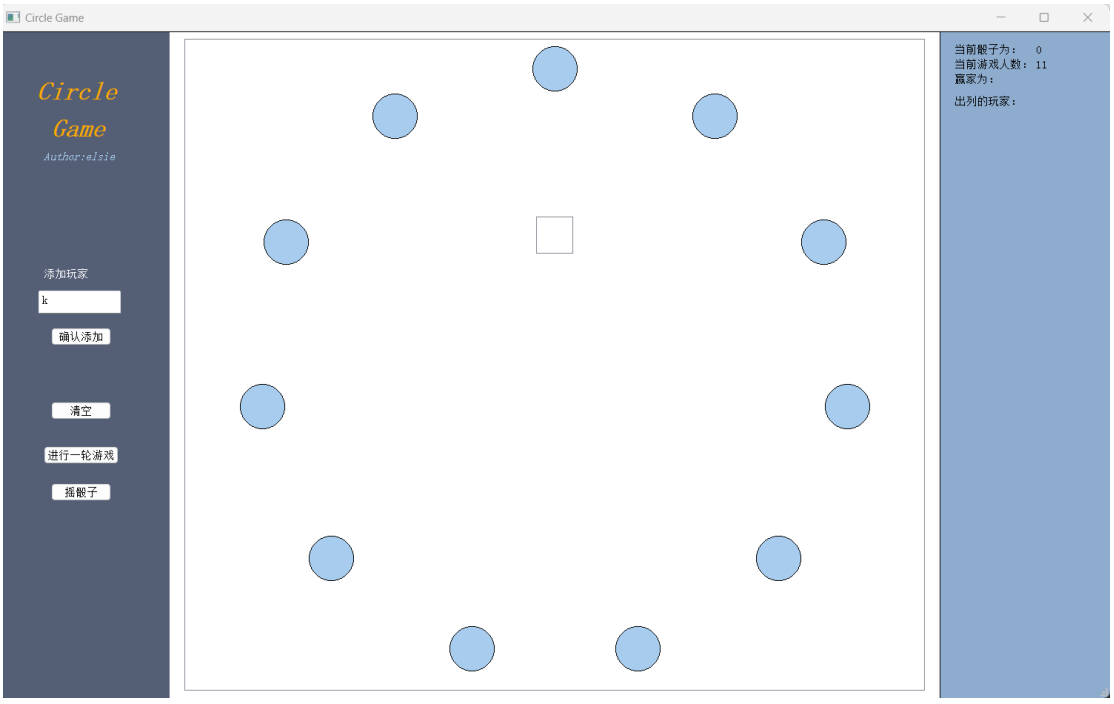



图 4 添加十名玩家

点击“进行一轮游戏”，进行动画演示，并在右侧信息区展示出列玩家的信息。连续点击该按钮进行多轮游戏。

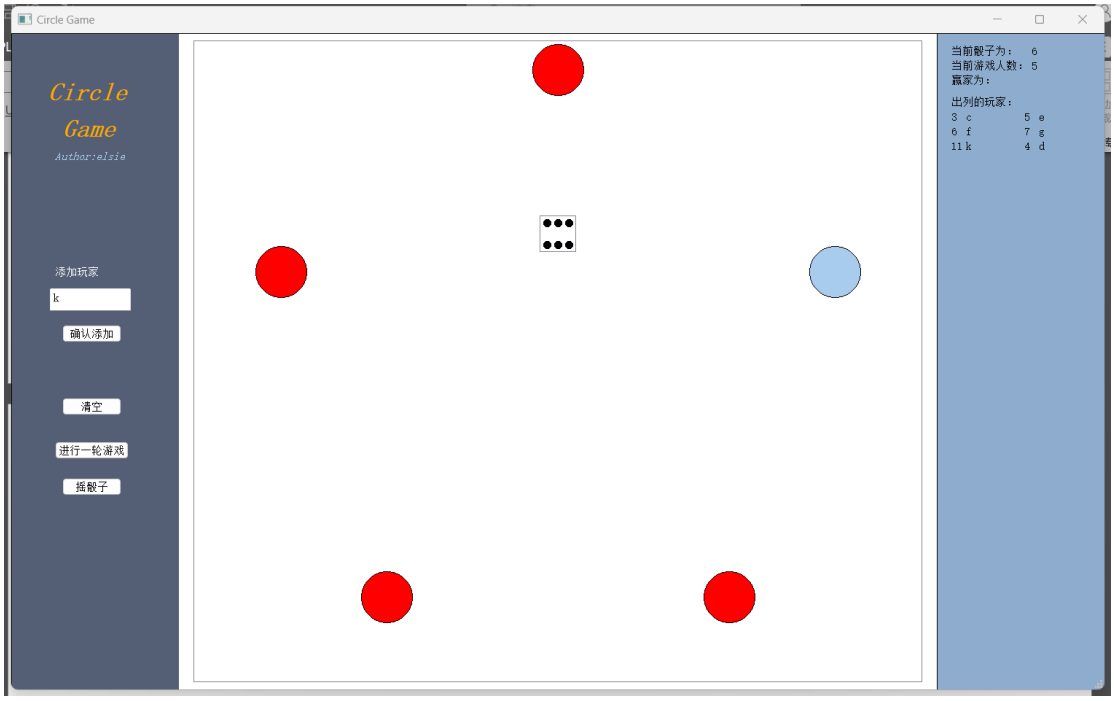



图 5 报数过程中的截图

进行完一轮游戏后，下一轮的起始玩家变为绿色。

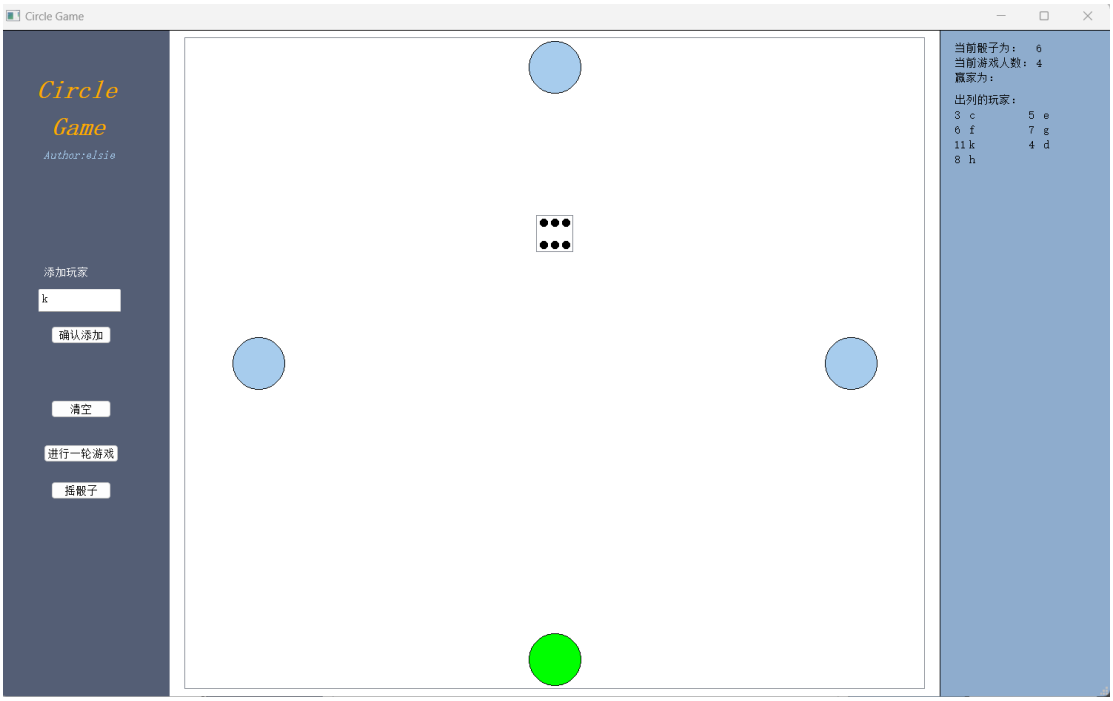



图 6 一轮游戏结束后的截图

最终仅剩一名玩家时就产生了最终的胜者，在右侧信息栏给出赢家的信息。

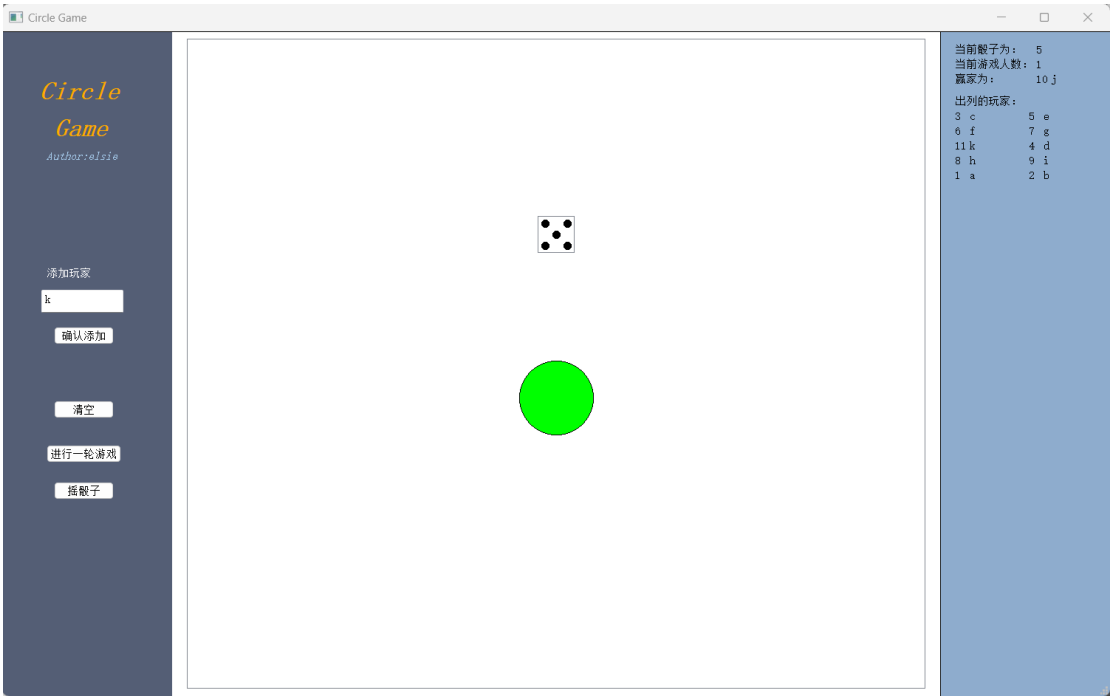



图 7 产生赢家并在右侧显示

如果觉得图片演示还不够清晰，也可以前往 b 站观看本人录制的演示视频，视频链接为[约瑟夫环游戏演示_哔哩哔哩_bilibili](#)（BV1UDHiecE4r）

当然，本程序也进行了一些容错性设计，基础的输入与上一个项目类似，在这里就不再赘述。输入区与模拟区同样禁止相互干涉，换言之，当模拟区继续工作时，输入区的功能都会被禁止。此外，还有一项容错性设计，便是当玩家仅有一人时禁止用户按下“进行一轮游戏”的按钮，这时会弹出警告框对用户进行提醒。

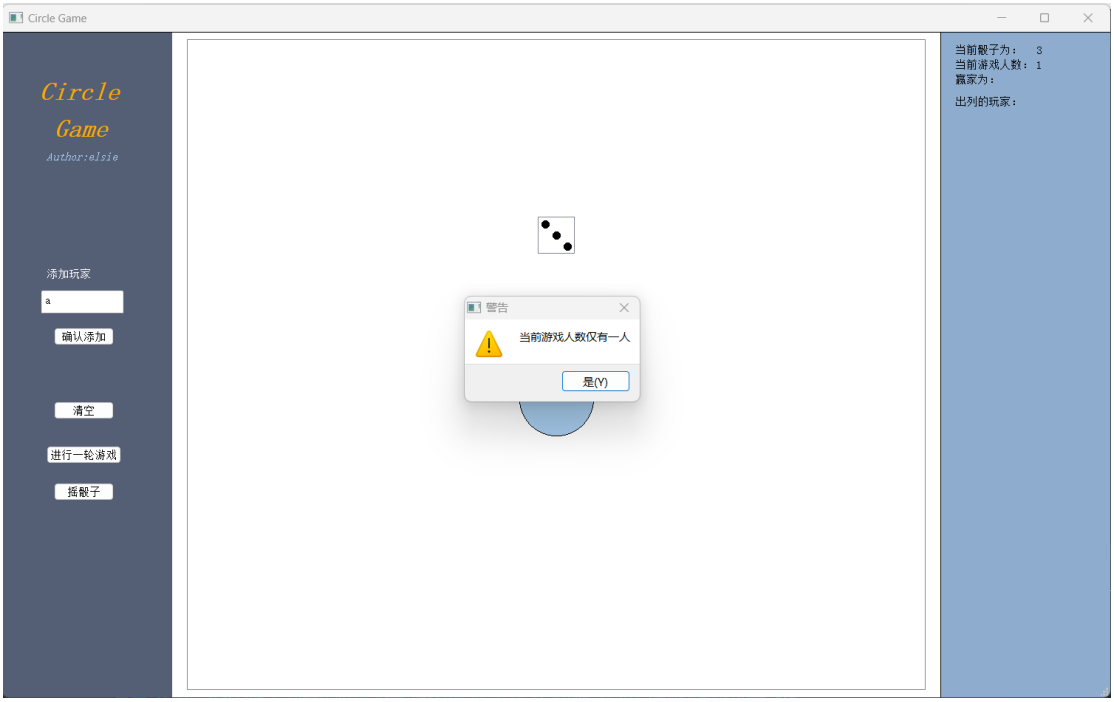
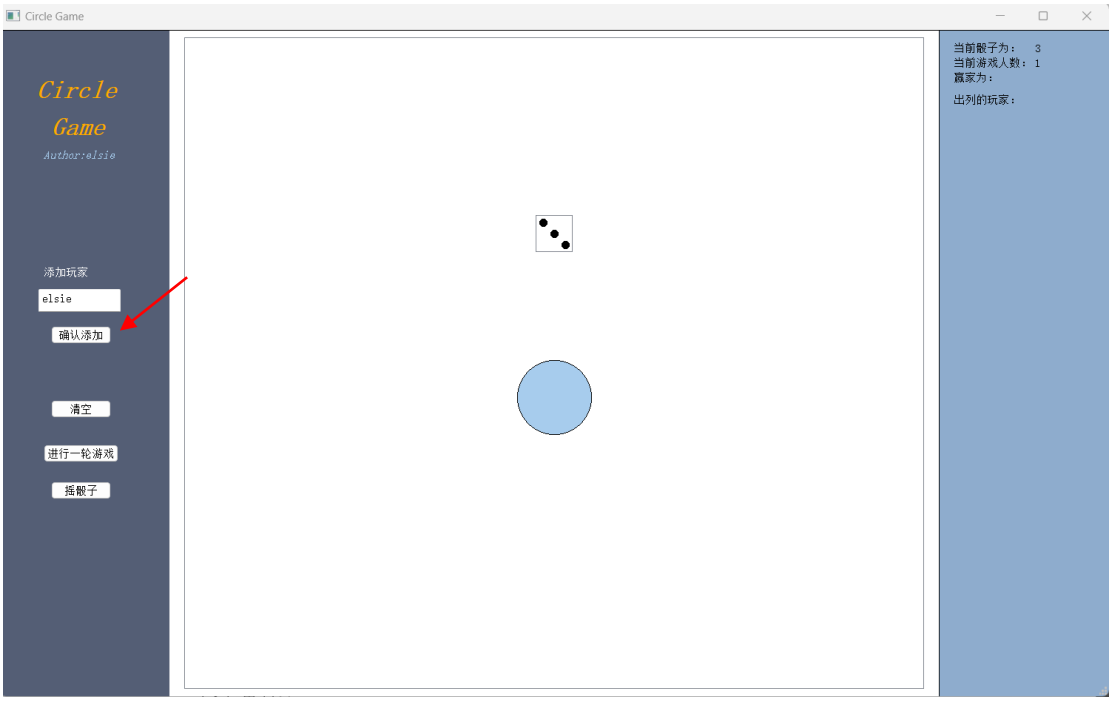


图 8 弹出警告框

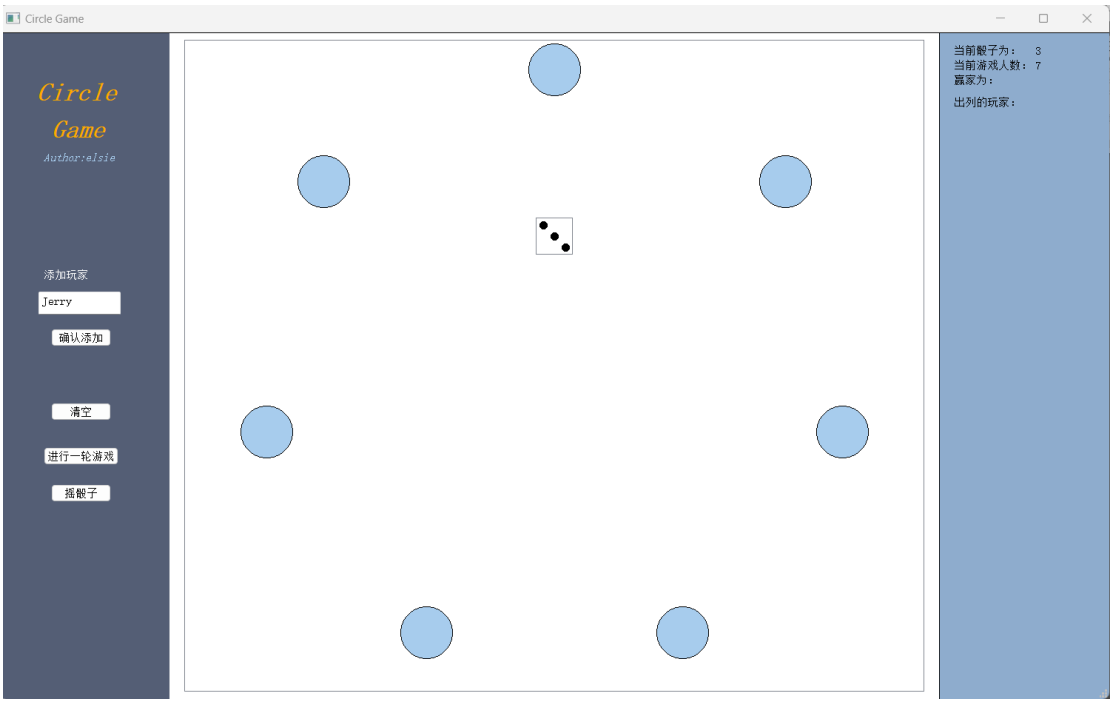
## 2.7 操作说明

（1）添加玩家

首先在左侧输入框输入玩家姓名，点击“确认添加”按钮进行添加，在中间模拟区看到代表玩家的圆圈产生，且在右侧信息区玩家数量一栏发生改变就说明玩家添加成功。

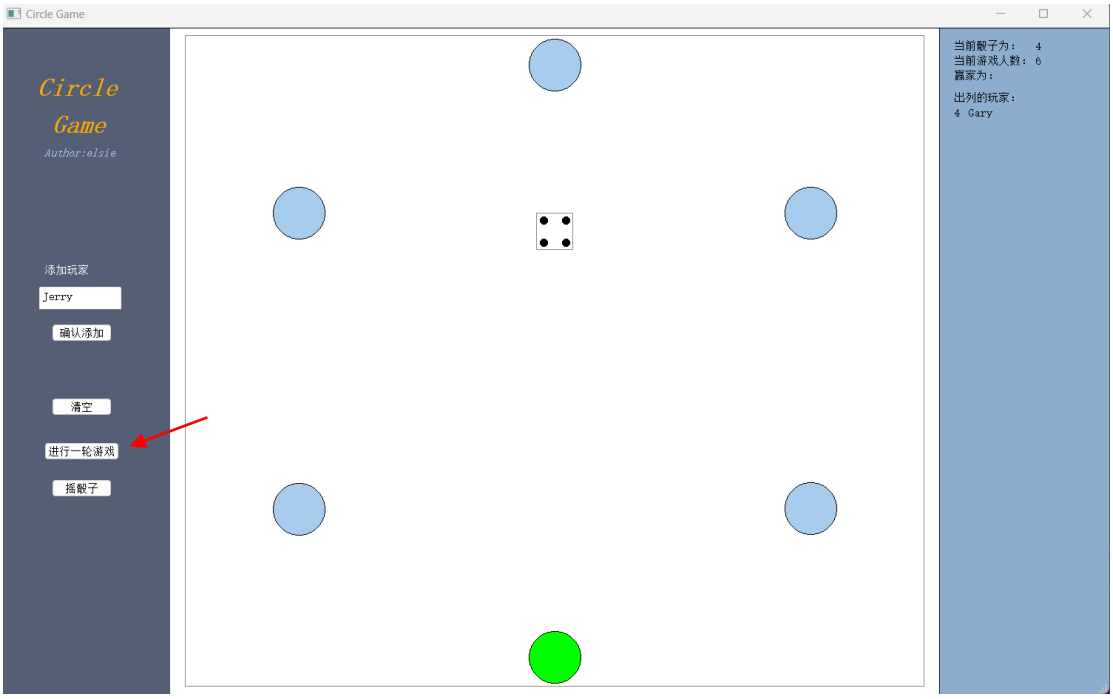



更改输入框中的姓名，重复添加玩家，直到玩家数量满足用户的需求。

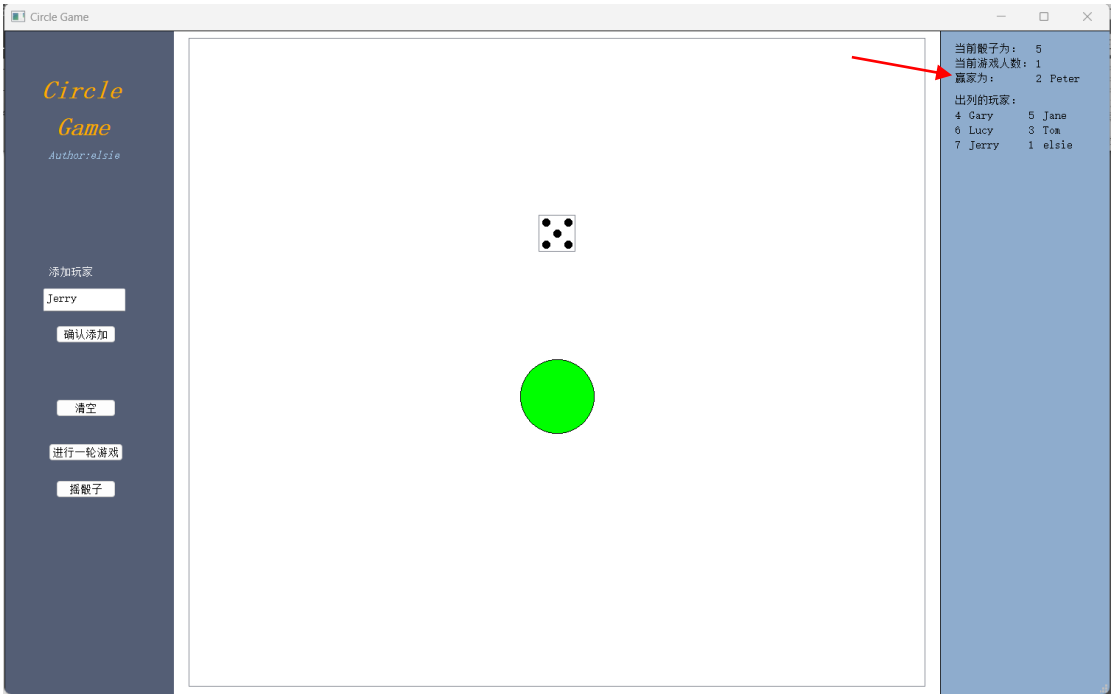


（2）模拟游戏

点击“进行一轮游戏”按钮，开启一轮游戏：投一次骰子，玩家进行报数，淘汰最后报数的玩家将其添加在出列玩家名单中，并将下一名玩家作为新的起始玩家。




重复点击该按钮，直到最后只剩一名玩家，打印赢家的信息。




（3）重置

当结束一局游戏后想重新开始新的一局游戏，可以直接从胜者的基础上进行添加，但这样就会保留上次游戏的出列玩家名单。也可以选择点击“清空”按钮，将所有信息

进行重置，这时就可以从零开始添加玩家啦（当然不角逐出最后的赢家也是可以重置的）。


# 第三部分 实践总结

## 3.1. 所做的工作

我想根据两个项目实现的流程总结我所完成的工作。

首先是初期理论验证的环节，在这个环节我自行实现了 *Array*、*SortInfo*、*Circle*、*Dice* 等一系列实用的类，同时选取了合适的数据结构设计，例如单链表、循环链表等，根据题目实际要求实现了数据结构操作所要用到的方法；此外，我还编写了六种排序算法的基础代码，结合前面完成的 *Array* 类在控制台中使用测试程序验证其正确性。

实现理论验证后我放缓了项目的进度，转而投入到 Qt 的学习中。因为 Qt 是我此前并未接触过的开发套件。在这个环节中，我安装并下载了使用 Qt 进行开发的一系列套件，学习了 Qt 代码编辑和 ui 设计功能，并学些了一些相关组件的使用方法。对 Qt 开发有了初步的认识后，我开始正式进入应用程序的开发。

我首先进行了 ui 界面的设计，包括字体排版、框图颜色、各个组件的位置等等。此后就进入功能区的开发，两个项目的开发流程较为类似，都是先进行输入区功能的实现，主要是实现各个按钮以及输入框的功能。为了验证输入区的功能，同时完成的还有显示文本信息的框图功能，这部分通过重写 *paintEvent()*进行实现。最后进行实现的就是动画区的功能，

这部分最为复杂，第一个项目需要对原先基础的排序算法进行重写使其得以适应可视化功能的需求，第二个项目也是较为类似。全部功能完成后就是对软件本体进行测试，挖掘软件可能存在的漏洞并及时进行修改，为软件的容错性提供保障。

当代码部分开发完成后，需要完成的最后一项工作就是利用 release 功能生成最终的执行文件，并生成相关的辅助文件进行打包，至此本次项目的所有工作汇报完毕。

## 3.2.总结与收获

通过完成本次项目的两个应用程序的开发，本人获益匪浅。首先是自行学习了 Qt 开发相关的知识，对 Qt 框架的基本原理以及结构进行了一定的了解，并学习了相关组件的使用方法。并在学习理论知识的同时亲自上手实践最终产出了两个较为完整的可视化应用程序。此外，也对之前课堂中学习过的许多数据结构（诸如单链表、顺序表以及循环链表）进行了回顾并亲自上手进行实践，此外也回顾了一些经典算法，比如程序中用到的多种不同的排序算法。

通过实现本次课程设计，我最大的收获是亲自实践了开发一个软件的简易流程，对内在的算法与数据结构如何与可视化界面以及用户之间形成联系与交互的原理有了初步的认识。此外，也通过实现项目的方式锻炼了自身开发较大程序的能力（与平时的小作业相比），并对所有头文件、cpp 文件以及资源文件的编排有了一定的认识。

在本次项目的实践过程中，我切实地感受到了面向对象编程的优势所在。其隐藏实现细节、封装参量与方法的特性为我整体项目的完成带来了十分优异的编程体验。最后，我也是深深感受到了工程项目的实践与平时简单算法的编程实验有着非常大的差异，平时的小型实验大多追求便捷快速的实现，而工程项目则十分追求实用性与可维护性，如果仿照平时写小实验的方式完成工程项目，例如随时随地新建变量或是为变量草草命名的话，在后续的开发与维护的流程中要会遇到十分大的阻碍。

总而言之，通过本次课程设计，我不光学习了 Qt 开发框架、亲手实践了可交互图形化程序的开发，也对数据结构与算法的基本原理进行了回顾，同时还对工程软件开发的流程与理念有了更深层次的理解。当然由于初学 Qt，整体的代码编排还是有很多不合理的地方，也有一些预想中的功能最终没能进行实现，希望可以在今后的项目开发中继续进步，不断丰富与精进自己的知识与技术。

# 第四部分 参考文献

*[1] 霍亚飞. Qt Creator 快速入门.北京：北京航空航天大学出版社，2022*

[2] 王维波，栗宝鹃，侯春望.Qt5.9C++开发指南.北京：人民邮电出版社，2018
[3] 陆文周.Qt5 开发及实例.北京:电子工业出版社，2019
[4] 严蔚敏，吴伟民.数据结构（C 语言版）.北京：清华大学出版社，2007
[5] 王晓东.计算机算法设计与分析.北京:电子工业出版社,2018
[6] 刘汝佳.算法竞赛入门经典(第 2 版).北京:清华大学出版社,2014