

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
по курсу «Программирование»
на тему «Разработка программы сложной структуры
методом нисходящего программирования»

Выполнил
студент группы 23ВОА2
Логинов А.Ю.

Принял
Митрохин М. А.

подпись 20.12.24.

Пенза 2024

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра «Вычислительная техника»

«УТВЕРЖДАЮ»

Зав. кафедрой ВТ

Митрохин М. А.

«___» _____ 2024 г

ЗАДАНИЕ
на курсовое проектирование по курсу

Программирование

Студенту Логинову Артему Юрьевичу

Группа 23ВОА2

Тема проекта: Разработка программы сложной структуры
методом нисходящего программирования

Исходные данные (технические требования) на проектирование
Наименование программы: DSA Visualization Вариант 43

Задание. Программа должна визуализировать алгоритм сортировки «Гномья сортировка (Gnome sort)» по возрастанию над структурой данных «Динамический массив».

Обязательные требования к программе:

1. Многомодульность.
2. Использование ООП.
3. Режим работы видеосистемы — графический.
4. Устройство ввода информации — клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Операционная система — Windows.
7. Система программирования — C/C++ на базе среды Microsoft Visual Studio 2019.
8. Язык программирования — C/C++.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

UML-диаграммы использования, UML-диаграммы классов, UML-диаграммы активности, UML-диаграммы последовательности.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

1. В соответствии с графиком выполнения курсового проекта.

2.

3.

Дата выдачи задания «17» сентября 2024 г.

Дата защиты проекта «___» _____ 2024 г.

Митрохин Максим Александрович

Руководитель

Задание получил

Студент

«17» сентября 2024 г.
Логинов Артем Юрьевич

Содержание

Введение	6
1 Постановка задачи	7
2 Выбор решения	8
3 Проектирование программы.....	9
3.1 UML-диаграмма использования.....	9
3.2 UML-диаграмма классов.....	10
3.3 UML-диаграмма активности	11
3.4 UML-диаграмма последовательности	12
4 Описание программы	13
5 Отладка и тестирование	17
6 Руководство пользователя	19
Заключение.....	20
Список использованных источников.....	22
Приложение А.1 Файл «main.cpp»	23
ПриложениеА.2 Файл «Arra.cpp»	30
ПриложениеА.3 Файл « Arra.h»	35
Приложение А.4 Файл «Button.cpp»	36
Приложение А.5 Файл « Button.h».....	37
ПриложениеА.6 Файл «Label.cpp»	38
ПриложениеА.7 Файл «Label.h»	41
ПриложениеА.8 Файл «Textbox.h»	42
Приложение А.9 Файл «TextBox.cpp»	43
ПриложениеА.10 Файл «TextMultiblock.h»	44
ПриложениеА.11 Файл « TextMultiblock.cpp».....	45
ПриложениеА.12 Файл «ViewPortElem.cpp»	45
ПриложениеА.13 Файл « ViewPortElem.h»	48
Приложение Б Результаты тестирования	49

Введение

Современный мир трудно представить без компьютеров. Несмотря на это, немногие задумываются о принципах их работы и степени их интеллекта, достигнутой за последние десятилетия. Для большинства людей компьютеры остаются неизменными понятиями, хотя они уже давно перешли границы обычных вычислительных машин.

Информационное общество — это социум, в котором большинство занято производством, хранением, переработкой и реализацией информации, особенно ее высшей формы — знаний.

Целью данной практической работы является создание программы, которая сортирует динамический массив с помощью алгоритма гномьей сортировки (gnomesort). Программа включаетв себя кнопки. Разработанная программа может быть использована людьми, не имеющими квалификации программиста. Программа создана с использованием современных концепций программирования — объектно-ориентированного программирования и языка Microsoft Visual Studio C++.

Пояснительная записка включает в себя техническое задание, описание программы, программу и методику испытаний, описание применения и текст программы.

1 Постановка задачи

С помощью графики отобразить на экране монитора ход решения следующей задачи: сортировка динамического массива с помощью гномьей сортировки(gnomesort).

2 Выбор решения

Проект "GnomeSort" будет реализован с использованием событийного механизма управления программой. Основной цикл программы включает обработку пользовательских вводов и обновление состояния программы на основе этих вводов. Реализация будет проходить по следующему плану:

Основной цикл программы

Основной цикл программы будет выполнять следующие ключевые функции:

1. **Обработка событий:** программа реагирует на различные события, такие как ввод данных пользователем, нажатия кнопок и закрытие окна. Это позволяет программе быть интерактивной и отзывчивой.
2. **Мониторинг таймера:** если активен процесс сортировки, программа отслеживает время, прошедшее с последнего шага сортировки, и, при необходимости, выполняет следующий шаг. Это обеспечивает плавность анимации и корректное выполнение алгоритма сортировки.
3. **Рендеринг окна:** программа очищает окно перед отрисовкой, визуализирует текущее состояние массива и интерфейса. Это обеспечивает четкость и актуальность отображаемых данных.
4. **Отображение изменений:** обновленные данные и изменения отображаются на экране, что позволяет пользователю видеть результат своих действий и текущий статус программы.

Механизм управления программой

Управление программой будет основано на событийной модели, что позволит эффективно обрабатывать пользовательские вводы и обновлять состояние программы в режиме реального времени. Эта модель предоставляет гибкость для расширения функциональности и добавления новых элементов в будущем.

3 Проектирование программы

В данном разделе приводится описание архитектуры системы, определяются основные компоненты и их взаимодействия (рисунки 3.1 – 3.4).

3.1 UML-диаграмма использования

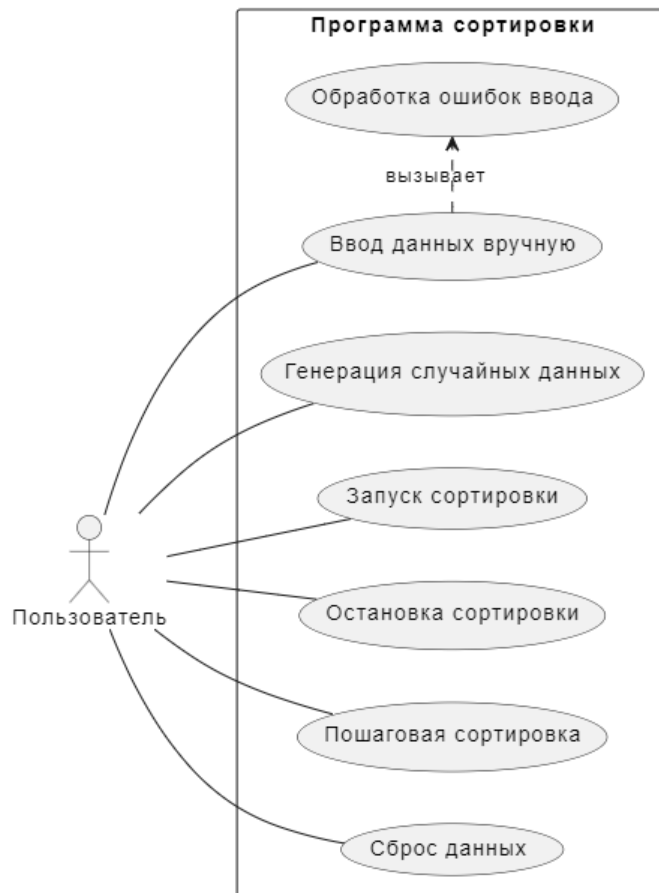


Рисунок 3.1 – Диаграмма использования

3.2 UML-диаграмма классов

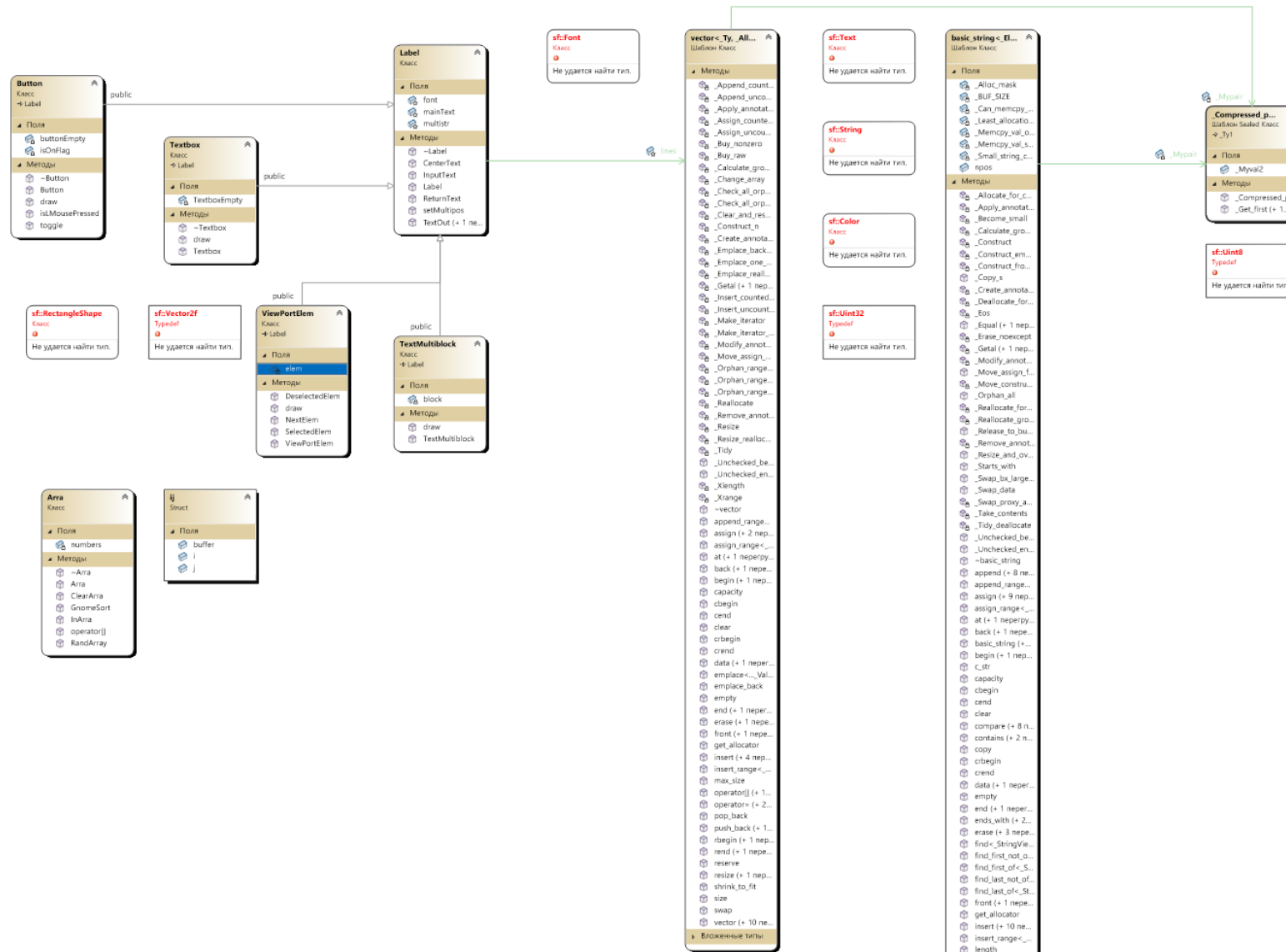


Рисунок 3.2 – Диаграмма классов

3.3 UML-диаграмма активности

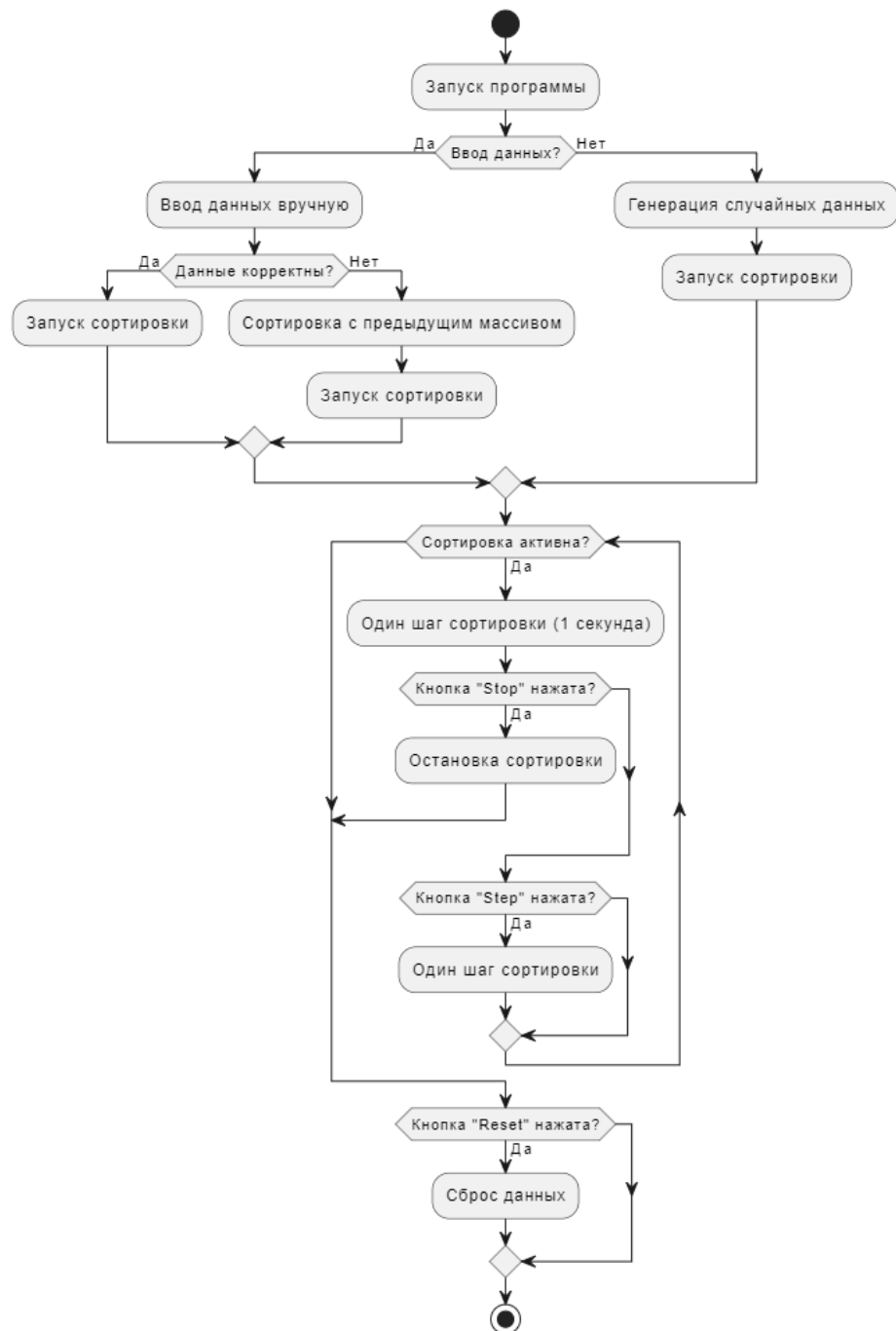


Рисунок 3.3 – Диаграмма активности

3.4 UML-диаграмма последовательности

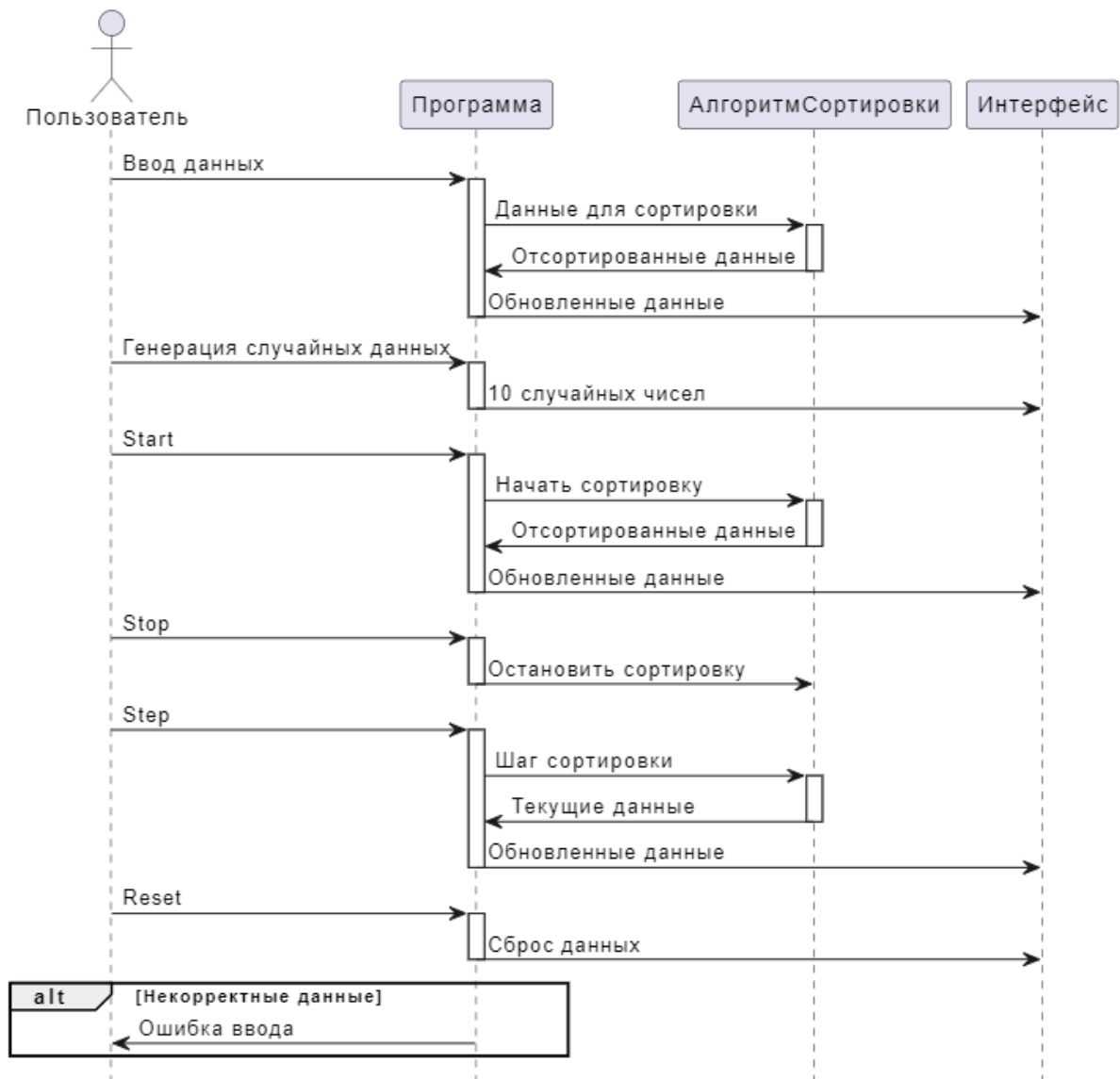


Рисунок 3.4 – Диаграмма последовательности

4 Описание программы

В данной программе выделяются следующие основные классы:

1. Класс **MyOpenGLWidget**:

Данный класс представляет собой основную часть работы программы визуализации алгоритма сортировки гномов с использованием библиотеки SFML.

Подключение библиотек и определение констант

В начале класса подключаются необходимые заголовочные файлы, включая SFML и пользовательские заголовочные файлы для кнопок, элементов интерфейса и других компонентов. Определяются константы для цветов, размеров кнопок и текстовых блоков, а также шрифта и его размера.

Основная функция

Основная функция программы выполняет следующие задачи:

1. **Инициализация окна:** Создается окно с заданными размерами и заголовком.
2. **Создание элементов интерфейса:** Инициализируются массив объектов ViewPortElem, представляющих элементы для визуализации сортировки. Также создаются кнопки, текстовые блоки и текстовые поля.
3. **Обработка событий:** Реализуется цикл обработки событий, который обрабатывает ввод текста и нажатия кнопок. В зависимости от событий выполняются соответствующие действия, такие как запуск сортировки, остановка, пошаговое выполнение, сброс и генерация случайных данных.
4. **Визуализация сортировки:** Используется алгоритм гномьей сортировки для сортировки элементов массива. Визуализация обновляется в реальном времени, отображая текущие состояния элементов.
5. **Управление памятью:** Обеспечивается правильная очистка динамически выделенной памяти и проверка утечек памяти с использованием средств отладки.

2. Класс Arra:

Подключение библиотек и определение констант

В начале класса подключаются необходимые заголовочные файлы, включая стандартные библиотеки C++ и пользовательские заголовочные файлы для работы с текстовыми полями и элементами интерфейса.

Основные методы класса Arra

1. Конструктор:

- Инициализирует массив numbers размером 10 элементов, заполняя его нулями.

2. Метод ClearArra:

- Очищает массив и текстовое поле, устанавливая все элементы массива в ноль и обновляя соответствующие элементы интерфейса.

3. Метод InArra:

- Проверяет и заполняет массив значениями из строки, разделенной запятыми. Возвращает true, если все значения корректны, и false в противном случае.

4. Метод RandArray:

- Заполняет массив случайными значениями в диапазоне от -100 до 100 и обновляет текстовое поле.

5. Метод GnomeSort:

- Реализует алгоритм гномьей сортировки для сортировки массива. Обновляет элементы интерфейса в процессе сортировки, визуализируя текущие состояния элементов.

6. Метод UserGnome:

- Вспомогательный метод для создания структуры ij, содержащей индексы и буферное значение для сортировки.

7. Перегрузка оператора []:

- Позволяет обращаться к элементам массива по индексу.

8. Деструктор:

- Освобождает память, выделенную под массив `numbers`.

Класс `Button`:

Этот класс предназначен для создания и управления кнопками в графическом интерфейсе с использованием библиотеки SFML. Класс предоставляет методы для инициализации кнопок, обработки событий нажатия и визуализации кнопок в окне приложения. Программа обеспечивает интерактивный способ взаимодействия с графическим интерфейсом, позволяя пользователю управлять состоянием кнопок и наблюдать за их изменениями:

Основные методы класса `Button`

1. Конструктор:

- Инициализирует объект кнопки с заданными размерами, позицией, цветом, шрифтом, текстом и размером текста. Устанавливает начальное состояние флага `isOnFlag` в 0.

2. Метод `draw`:

- Отрисовывает кнопку и текст на экране, используя окно SFML.

3. Метод `toggle`:

- Переключает состояние флага `isOnFlag` и возвращает его новое значение.

4. Метод `isLMousePressed`:

- Обработывает событие нажатия левой кнопки мыши. Если курсор мыши находится в пределах кнопки при нажатии, переключает состояние флага `isOnFlag` и возвращает его новое значение.

3. Класс Label:

Этот класс демонстрирует использование класса Label для создания и управления текстовыми метками в графическом интерфейсе с использованием библиотеки SFML. Класс предоставляет методы для инициализации текстовых меток, их центрирования, обновления текста и визуализации в окне приложения.

Основные методы класса Label

1. Конструктор:

- Инициализирует объект текстовой метки с заданными размерами, цветом, шрифтом, текстом и позицией. Загружает шрифт из файла и разбивает текст на строки по символу \b.

2. Метод InputText:

- Обновляет текст метки на заданный входной текст.

3. Метод CenterText:

- Центрирует текст внутри заданного прямоугольника, вычисляя новую позицию текста на основе размеров прямоугольника и текста.

4. Деструктор:

- Очищает вектор строк lines и освобождает память.

5. Метод TextOut:

- Отрисовывает текстовую метку в окне SFML.

6. Метод setMultipos:

- Устанавливает позицию текста с учетом множителя, чтобы текст располагался вертикально с отступами.

7. Перегруженный метод TextOut:

- Отрисовывает текстовую метку построчно в окне SFML, используя заданную позицию.

8. Метод ReturnText:

- Возвращает текущий текст метки.

5 Отладка и тестирование

Для проверки сортировки массива с помощью кнопки “Start”., необходимо запустить программу на выполнение (рисунок Б.1), нажать кнопку “Random”, чтобы сгенерировать массив, либо ввести его вручную и нажать кнопку “Start” (Рисунок Б.1). Массив начнет сортировку, отображая сортируемые элементы (Рисунок Б.6).

Тестовый пример 1. Сортировка массива с помощью кнопки “Step”.

Для проверки сортировки массива с помощью кнопки “Step”., необходимо запустить программу на выполнение (рисунок Б.1), нажать кнопку “Random”, чтобы сгенерировать массив, либо ввести его вручную и нажать кнопку “Step” (Рисунок Б.1). Массив начнет сортировку, отображая сортируемые элементы (Рисунок Б.7) продвигаясь между ключевыми точками при каждом нажатии кнопки “Step”.

Тестовый пример 2. Остановка сортировки массива с помощью кнопки “Stop”.

Для проверки остановки сортировки массива с помощью кнопки “Stop”, необходимо запустить программу на выполнение (рисунок Б.1), нажать кнопку “Random”, чтобы сгенерировать массив, либо ввести его вручную и нажать кнопку “Step” (Рисунок Б.1). Массив начнет сортировку, отображая сортируемые элементы (Рисунок Б.7). После этого нажать кнопку “Stop” – массив остановит сортировку и возобновит ее при нажатии на кнопки “Start” или “Step” (Рисунок Б.8).

Тестовый пример 3. Очищение массива с помощью кнопки “Reset”.

Для проверки очищения массива с помощью кнопки “Reset”, необходимо запустить программу на выполнение (рисунок Б.1), нажать кнопку “Random”, чтобы сгенерировать массив, либо ввести его вручную и нажать кнопку “Step” (Рисунок Б.1). Массив начнет сортировку, отображая сортируемые элементы (Рисунок Б.8). После этого нажать кнопку “Reset” – массив остановит сортировку, из окна ввода пропадут введенные числа, а объекты массива станут равны 0 (Рисунок Б.9).

Исходя из проведенного теста, можно сделать вывод, что программа работает корректно и без ошибок.

6Руководство пользователя

Установка и запуск

1. Установка:

- Скачайте архив с программой с официального сайта или из предоставленного источника.
- Распакуйте содержимое архива в удобную папку на вашем компьютере.

2. Запуск:

- Откройте папку с распакованной программой.
- Найдите исполняемый файл `united.exe`.
- Дважды щелкните по файлу, чтобы запустить программу.

Пользовательский интерфейс:

При запуске программы вы увидите основное меню, включающее следующие пункты:

- **10 объектов ViewPortElement:**Отображают элементы массива.
- **Кнопки «Start», «Stop», «Step», «Random», «Reset»:**добавление файлов и текстур.
- **Два многострочных текстовых поля:**Отображает информацию о структуре и методе ее сортировки.
- **Поле Input:**Отображает сгенерированные числа и позволяет пользователю осуществлять их ввод в ручную.

Заключение

В ходе выполнения данного проекта по разработке программы "Gnomesort" были достигнуты значительные успехи, которые позволили получить ценные знания и практические навыки в области программирования и разработки программ.

Приобретенные навыки и знания

Во-первых, работа над проектом способствовала углубленному изучению языка программирования C++ и библиотеки SFML, которые стали основными инструментами для создания приложения. Было детально изучено создание оконных приложений, работа с графикой, что позволило реализовать все основные элементы программы.

Неудачи и возможные улучшения

Несмотря на достигнутые успехи, проект столкнулся с рядом трудностей и ограничений. Одной из них была проблема с трудностью обработки выхода из функции с сохранением параметров, что потребовало дополнительного времени на решение.

Направления для улучшения

1. **Улучшение графики и анимаций:** Улучшение визуальных эффектов, расширение функционала интерфейса и добавление горячих клавиш, позволит сделать программу более привлекательной для широкого круга лиц.

2. **Кастомизация интерфейса:** Добавление возможности кастомизации интерфейса даст пользователям возможность использовать знакомое расположение блоков, позволит пользователю быстрее привыкнуть к программе.

Итог

Проект "Gnomesort" предоставил отличную возможность для практики и изучения ключевых аспектов разработки в направлении создания приложений. Несмотря на некоторые сложности и ограничения, были достигнуты значительные успехи, и проект представляет собой программу с широкими возможностями для дальнейшего развития и улучшения. Работая над проектом, удалось приобрести

ценные навыки и знания, которые будут полезны в будущих проектах и разработках.

Список использованных источников

1. Липачёв Е.К. Технология программирования. Базовые конструкции C/C++ - Казань: Казан.
2. C/C++. Программирование на языке высокого уровня/ Т. А. Павловская. — СПб.: Питер.
3. Хомоненко А.Д. - Программирование на C++. - Альтекс.
4. OpenGL: [Электронный ресурс] Камера — URL: <https://habr.com/ru/articles/327604/>

Приложение А.1

Файл «main.cpp»

```
#include <SFML/Graphics.hpp>

#include <iostream>

#include "button.h"

#include "ViewPortElem.h"

#include "Arra.h"

#include "Textbox.h"

#include "TextMultiblock.h"

#include <set>

#define _CRTDBG_MAP_ALLOC

#include <stdlib.h>

#include <crtDBG.h>

const sf::Color buttonColor = sf::Color(72, 128, 143);

const sf::Color TextBoxColor = sf::Color(145, 191, 204);

const sf::Color WorkspaceColor = sf::Color(47, 69, 82);

const sf::Vector2f ButtonSize = sf::Vector2f(200, 80);

const sf::Vector2f MultiblockSize = sf::Vector2f(400, 350);

const std::string Font = "Impact.ttf";

const int FontSize = 30;

std::set<char32_t> ASCII = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '-', ',', '\b' };

int main()

{

    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    sf::RenderWindow window(sf::VideoMode(1300, 900), L"Gnome Sort",
sf::Style::Titlebar | sf::Style::Close);

    ViewPortElem* Sprites[10];

    Arra pashalkoxD;
```

```

    ij returnData;

    bool f = 1;

    bool flag1 = 1;

    bool isArraFull = false;

    bool flag = 0;

    std::string inputS;

    size_t b = 150;

    size_t a = 60;

    for (size_t i = 0; i < 10; i++)
    {
        Sprites[i] = new ViewPortElem(sf::Vector2f(100, 100), sf::Vector2f(a, b),
sf::Color::White, Font, std::to_string(pashalkoxD[i]), FontSize);

        a += 120;

    }

    TextMultiblock Info1(MultiblockSize, sf::Vector2f(350, 500), buttonColor, Font,
"A dynamic array is an array with a big\bimprovement: automatic resizing.\bOne
limitation of arrays is that they're fixed \bsize,meaning you need to specify the
number\b of elements your array will hold\b ahead of time.", 20,sf::Color::White);

    TextMultiblock Info2(MultiblockSize, sf::Vector2f(850, 500), buttonColor, Font,
"Gnome sort(nicknamed stupid sort)\b is a variation of the insertion sort\b sorting
algorithm that does not\b use nested loops.Gnome sort was\b originally proposed by
Iranian\b computer scientist Hamid Sarbazi - Azad\b(professor of Computer Science\b and
Engineering at Sharif\b University of Technology", 20, sf::Color::White);

    Textbox input(sf::Vector2f(950, 80), sf::Vector2f(50, 350), TextBoxColor);

    Button start(ButtonSize, sf::Vector2f(50, 500), buttonColor, Font, "Start",
FontSize);

    Button stop(ButtonSize, sf::Vector2f(50, 590), buttonColor, Font, "Stop",
FontSize);

    Button step(ButtonSize, sf::Vector2f(50, 680), buttonColor, Font, "Step",
FontSize);

    Button reset(ButtonSize, sf::Vector2f(50, 770), buttonColor, Font, "Reset",
FontSize);

    Button random(ButtonSize, sf::Vector2f(1040, 350), buttonColor, Font, "Random",
FontSize);

    while (window.isOpen())

```

```

{

    sf::Event event;

    while (window.pollEvent(event))
    {

        if (event.type == sf::Event::TextEntered) // Если вводится текст
        {

            if (!(ASCII.find(event.text.unicode) == ASCII.end())) //
отсекаем не ASCII символы
            {

                if (event.text.unicode == '\b' && !inputS.empty())
inputS.pop_back();

                //if (event.text.unicode == '\b' && inputS.empty()) {}

                else if(inputS.length() <=50) inputS +=
static_cast<char>(event.text.unicode); // добавляем в строку вводимый символ

                input.InputText(inputS);

            }

        }

        if (event.type == sf::Event::Closed) window.close();

        flag = start.isLMousePressed(event);

        if (random.isLMousePressed(event))
        {

            if (flag) flag = start.toggle();

```

```

returnData.buffer = 0;

returnData.i = 1;

returnData.j = 1;


for (size_t i = 0; i < 10; i++)
{
    Sprites[i]->DeselectedElem();
}


pashalkoxD.RandArray(input);

isArraFull = true;

random.toggle();
}

if (stop.isLMousePressed(event))
{
    if(flag) flag = start.toggle();;
    stop.toggle();
}

if (step.isLMousePressed(event))
{
    if (isArraFull == false)
    {
        f = pashalkoxD.InArra(input.ReturnText());

        if (f) {
            isArraFull = true;
        }

        else
        {
            input.InputText("");

            inputS = "";
        }
    }
}

```

```

    }

    if (isArraFull && f)
    {
        pashalkoxD.GnomeSort(Sprites, returnData, flag1, 1, 0);
    }

    step.toggle();

}

if (reset.isLMousePressed(event))
{

    if (flag)
    {
        flag = start.toggle();
        returnData.buffer = 0;
        returnData.i = 1;
        returnData.j = 1;
    }

    for (size_t i = 0; i < 10; i++)
    {
        Sprites[i]->DeselectedElem();
    }

    pashalkoxD.ClearArra(input, Sprites);
    inputS = "";
    isArraFull = false;
    reset.toggle();

}

}

```

```

for (size_t i = 0; i < 10; i++)
{
    Sprites[i]->InputText(std::to_string(pashalkoxD[i]));

}

if (flag) {

    if (isArraFull == false)
    {
        f = pashalkoxD.InArra(input.ReturnText());
        if (f) {
            isArraFull = true;
        }
        else
        {
            input.InputText("");
            inputS = "";
            flag = 0;
            start.toggle();
        }
    }

    if (isArraFull && f)
    {
        pashalkoxD.GnomeSort(Sprites, returnData, flag1, 1, 1);
    }

    //flag = start.toggle();
}

window.clear(WorkspaceColor);

```

```

        for (size_t i = 0; i < 10; i++)
        {
            Sprites[i]->draw(window);
        }

        start.draw(window);
        stop.draw(window);
        step.draw(window);
        reset.draw(window);
        random.draw(window);
        input.draw(window);
        Info1.draw(window);
        Info2.draw(window);
        window.display();

    }

    for (size_t i = 0; i < 10; i++)
    {
        delete Sprites[i];
        Sprites[i] = nullptr;
    }

    //delete[] Sprites;

    _CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_DEBUG);
    _CrtDumpMemoryLeaks();
    return 0;

}

}

```

Приложение А.2

Файл «Arra.cpp»

```
#include "Arra.h"

#include <random>

#include <sstream>

#include "Windows.h"


Arra::Arra()

{

    numbers = new int[10]();

}

void Arra::ClearArra(Textbox& text, ViewPortElem** Sprites)

{

    text.InputText("");

    for (size_t i = 0; i < 10; i++)

    {

        numbers[i] = 0;

        Sprites[i]->InputText(std::to_string(0));

    }

}


bool Arra::InArra(std::string number)

{

    if (number.empty()) return false;

    number.erase(remove(number.begin(), number.end(), ' '), number.end());

    std::stringstream ss(number);

    std::string item;
```

```

int i = 0;

while (std::getline(ss, item, ',') && i < 10 ) {

    try

    {

        if (std::stoi(item) > 100 || std::stoi(item) < -100)

        {

            return false;

        }

        else numbers[i] = std::stoi(item);

    }

    catch (const std::exception&)

    {

        return false;

    }

    i++;

}

return true;

}

void Arra::RandArray(Textbox &text )

{

    std::random_device rd; // Получаем случайное значение из аппаратного генератора
    std::mt19937 gen(rd()); // Инициализируем генератор Mersenne Twister
    std::uniform_int_distribution<> distrib(-100, 100);
    text.InputText("");
    for (size_t i = 0; i < 10; i++)

    {

```

```

        std::string a = (text.ReturnText() == "") ? "" : ", ";

        text.InputText(text.ReturnText() + a + std::to_string(distrib(gen)));
    }

    InArra(text.ReturnText());

}

ij UserGnome(const int i, const int j, const int buffer)
{
    ij returnIJ;

    returnIJ.i = i;

    returnIJ.j = j;

    returnIJ.buffer = buffer;

    return returnIJ;
}

void Arra::GnomeSort(ViewPortElem** Sprites, ij &ijIn, bool& flag1, bool userGnome,
bool startpress)
{

    int i = ijIn.i;

    int j = ijIn.j;

    int buffer = ijIn.buffer;

    while (i < 10)
    {

        if (userGnome && flag1)
        {

            Sprites[i - 1]->SelectedElem();

            Sprites[i]->NextElem();

```

```

    flag1 = !flag1;
    ijIn = UserGnome(i, j,buffer);
    if (startpress) Sleep(500);
    return;
}

```

```

if (numbers[i - 1] < numbers[i])
{
    int iPred = i;
    i = j;
    j++;
    if (userGnome)
    {
        Sprites[iPred - 1]->DeselectedElem();
        Sprites[iPred]->DeselectedElem();
        flag1 = !flag1;
        ijIn = UserGnome(i, j, buffer);
        if (startpress) Sleep(500);
        return;
    }
}

```

```

else
{
    Sprites[i]->InputText(std::to_string(numbers[i - 1]));
    Sprites[i - 1]->InputText(std::to_string(numbers[i]));
    buffer = numbers[i - 1];
    numbers[i - 1] = numbers[i];

    numbers[i] = buffer;
    int iPred = i;
}

```

```

        i--;

        if (i == 0)
        {
            i = j;

            j++;

        }

        if (userGnome)
        {
            flag1 = !flag1;

            ijIn = UserGnome(i, j, buffer);

            Sprites[iPred - 1]->DeselectedElem();

            Sprites[iPred]->DeselectedElem();

            if (startpress) Sleep(500);

            return;

        }

    }

}

int Arra::operator[] (int number)
{
    return numbers[number];
}

Arra::~~Arra()
{
    delete[] numbers;
}

```

Приложение А.3

Файл « Arra.h »

```
#pragma once
#include <iostream>
#include "ViewPortElem.h"
#include "Textbox.h"
struct ij
{
    int i = 1;
    int j = 1;
    int buffer = 0;
};

class Arra
{
private:
    int *numbers;

public:
    Arra();

    void ClearArra(Textbox& text, ViewPortElem** Sprites);

    bool InArra(std::string number);

    void RandArray(Textbox &text);
    void GnomeSort(ViewPortElem** Sprites, ij &ijIn, bool& flag1, bool UserGnome,
bool startpress);
    int operator[] (int number);
    ~Arra();
};
```

Приложение А.4

Файл «Button.cpp»

```
#include "Button.h"

Button::Button(sf::Vector2f buttonSize, sf::Vector2f buttonPosition,
               sf::Color buttonColor, std::string font, std::string text, int textSize,
               sf::Color textColor) : Label(textSize, textColor, font, text)
{
    Label::CenterText(buttonSize, buttonPosition);
    buttonEmpty.setSize(buttonSize);
    buttonEmpty.setFillColor(buttonColor);
    buttonEmpty.setPosition(buttonPosition);
    isOnFlag = 0;
}

void Button::draw(sf::RenderWindow& window)
{
    window.draw(buttonEmpty);
    Label::TextOut(window);
}

bool Button::toggle()
{
    isOnFlag = !isOnFlag;
    return isOnFlag;
}

bool Button::isLMousePressed(const sf::Event& event)
{
    if (event.type == sf::Event::MouseButtonPressed) {
        if (event.mouseButton.button == sf::Mouse::Left) {
            sf::Vector2f mousePos(event.mouseButton.x,
event.mouseButton.y);

            if (buttonEmpty.getGlobalBounds().contains(mousePos)) {
                isOnFlag = !isOnFlag;
                return isOnFlag;
            }
        }
    }
    return isOnFlag;
}
```

Приложение А.5

Файл « Button.h »

```
#pragma once

#include <SFML/Graphics.hpp>

#include "Label.h"

class Button: public Label
{
public:
    Button(sf::Vector2f buttonSize = sf::Vector2f(10, 10),
           sf::Vector2f buttonPosition = sf::Vector2f(0, 0),
           sf::Color buttonColor = sf::Color(128, 128, 128),
           std::string font = "Impact.ttf",
           std::string text = " ", int textSize = 0,
           sf::Color textColor = sf::Color::White);

    void draw(sf::RenderWindow& window);

    bool isLeftMousePressed(const sf::Event& event) ;

    bool toggle();

    ~Button() {}

private:
    sf::RectangleShape buttonEmpty;

    bool isOnFlag;

};
```

Приложение А.6

Файл «Label.cpp»

```
#include "Label.h"

#include <string>

#include <sstream>

Label::Label(int size, sf::Color color, std::string font, std::string text,
sf::Vector2f position)

{

    this->font.loadFromFile(font);

    if (!(this->font.loadFromFile(font))) exit(0);

    mainText.setFont(this->font);

    mainText.setString(text);

    mainText.setPosition(position);

    mainText.setCharacterSize(size);

    mainText.setFillColor(color);


    std::stringstream ss(text);

    std::string item;

    int i = 0;

    while (std::getline(ss, item, '\b'))

    {

        lines.push_back(item);

    }

}

void Label::InputText(const std::string& input)

{

    mainText.setString(input);

}


void Label::CenterText(const sf::Vector2f& rectSize, const sf::Vector2f& rectPosition)

{
```

```

// Получаем размеры текста
sf::FloatRect textBounds = mainText.getGlobalBounds();

// Вычисляем позицию для центрирования
float x = rectPosition.x + (rectSize.x - textBounds.width) / 2;
float y = rectPosition.y + (rectSize.y - textBounds.height) / 2;

// Устанавливаем новую позицию текста
mainText.setPosition(x, y);

}

Label::~Label()
{
    lines.clear();
    lines.shrink_to_fit();

}

void Label::TextOut(sf::RenderWindow& window)
{
    window.draw(mainText);
}

void Label::setMultipos( sf::Vector2f& pos, int i)
{
    mainText.setPosition(pos.x, pos.y + i*(mainText.getCharacterSize() + 5));

};

void Label::TextOut(sf::RenderWindow& window, sf::Vector2f brickPos)
{
    for (size_t i = 0; i < lines.size(); i++)
    {

```

```
        mainText.setString(lines[i]);  
        setMultipos(brickPos, i);  
        window.draw(mainText);  
    }  
}
```

```
std::string Label::ReturnText()  
{  
    return mainText.getString();  
}
```

Приложение А.7

Файл «Label.h»

```
#pragma once

#include <SFML/graphics.hpp>

#include <vector>

#include <iostream>

class Label
{
protected:
    sf::Font font;
    sf::Text mainText;
    std::vector<std::string> lines;
    bool multistr = 0;
public:
    Label(int size = 30, sf::Color color = sf::Color::White,
          std::string font = "Impact.ttf", std::string text = "None",
          sf::Vector2f position = sf::Vector2f(60, 60));
    ~Label();
    void InputText(const std::string& input);
    void CenterText(const sf::Vector2f& buttonSize, const sf::Vector2f&
buttonPosition);
    void TextOut(sf::RenderWindow& window) ;
    void setMultipos(sf::Vector2f& pos, int i);
    void TextOut(sf::RenderWindow& window, sf::Vector2f brickPos);
    std::string ReturnText();
};
```

Приложение А.8

Файл «Textbox.h»

```
#pragma once

#include <SFML/Graphics.hpp>
#include "Label.h"

class Textbox : public Label
{
public:
    Textbox(sf::Vector2f TextboxSize = sf::Vector2f(10, 10),
            sf::Vector2f TextboxPosition = sf::Vector2f(0, 0),
            sf::Color TextboxColor = sf::Color(128, 128, 128));

    void draw(sf::RenderWindow& window);

    ~Textbox() {}

private:
    sf::RectangleShape TextboxEmpty;
};
```

Приложение А.9

Файл «TextBox.cpp»

```
#include "Textbox.h"

Textbox::Textbox(sf::Vector2f TextboxSize, sf::Vector2f TextboxPosition, sf::Color
TextboxColor) :Label(30, sf::Color::White, "Impact.ttf", "")
{
    Label::mainText.setPosition(60,370);

    TextboxEmpty.setSize(TextboxSize);

    TextboxEmpty.setPosition(TextboxPosition);

    TextboxEmpty.setFillColor(TextboxColor);
}

void Textbox::draw(sf::RenderWindow& window)
{
    window.draw(TextboxEmpty);

    Label::TextOut(window);
}
```

Приложение А.10

Файл «TextMultiblock.h»

```
#pragma once

#include <SFML/Graphics.hpp>

#include "label.h"

class TextMultiblock :public Label
{
public:
    TextMultiblock(sf::Vector2f elemSize, sf::Vector2f elemPosition,
        sf::Color elemColor, std::string font, std::string text, int textSize,
        sf::Color textColor = sf::Color::Black);

    void draw(sf::RenderWindow& window);

private:
    sf::RectangleShape block;
};
```

Приложение А.11

Файл «TextMultiblock.cpp»

```
#include "TextMultiblock.h"

TextMultiblock::TextMultiblock(sf::Vector2f elemSize, sf::Vector2f elemPosition,
    sf::Color elemColor, std::string font, std::string text, int textSize,
    sf::Color textColor) : Label(textSize, textColor, font, text, elemPosition)
{

    //Label::CenterText(elemSize, elemPosition);
    block.setSize(elemSize);
    block.setPosition(elemPosition);
    block.setFillColor(elemColor);

}

void TextMultiblock::draw(sf::RenderWindow& window)
{
    window.draw(block);
    Label::TextOut(window, sf::Vector2f(block.getPosition().x + 10,
block.getPosition().y + block.getPosition().y/5));
}
```

Приложение А.12

Файл «ViewPortElem.cpp»

```
#include "ViewPortElem.h"

ViewPortElem::ViewPortElem(sf::Vector2f elemSize, sf::Vector2f elemPosition,
    sf::Color elemColor, std::string font, std::string text, int textSize,
    sf::Color textColor) : Label(textSize, textColor, font, text)
{

    Label::CenterText(elemSize, elemPosition);

    elem.setSize(elemSize);

    elem.setPosition(elemPosition);

    elem.setFillColor(elemColor);

}

void ViewPortElem::draw(sf::RenderWindow& window)
{

    window.draw(elem);

    Label::TextOut(window);

}

void ViewPortElem::SelectedElem()
{

    elem.setFillColor(sf::Color::Green);

}

void ViewPortElem::NextElem()
{

    elem.setFillColor(sf::Color::Yellow);

}

void ViewPortElem::DeselectedElem()
{

    elem.setFillColor(sf::Color::White);

}
```


Приложение А.13

Файл «ViewPortElem.h»

```
#pragma once

#include <SFML/Graphics.hpp>

#include "label.h"

class ViewPortElem:public Label
{
public:
    ViewPortElem(sf::Vector2f elemSize, sf::Vector2f elemPosition,
        sf::Color elemColor, std::string font, std::string text, int textSize,
        sf::Color textColor = sf::Color::Black);

    void draw(sf::RenderWindow& window);

    void SelectedElem();

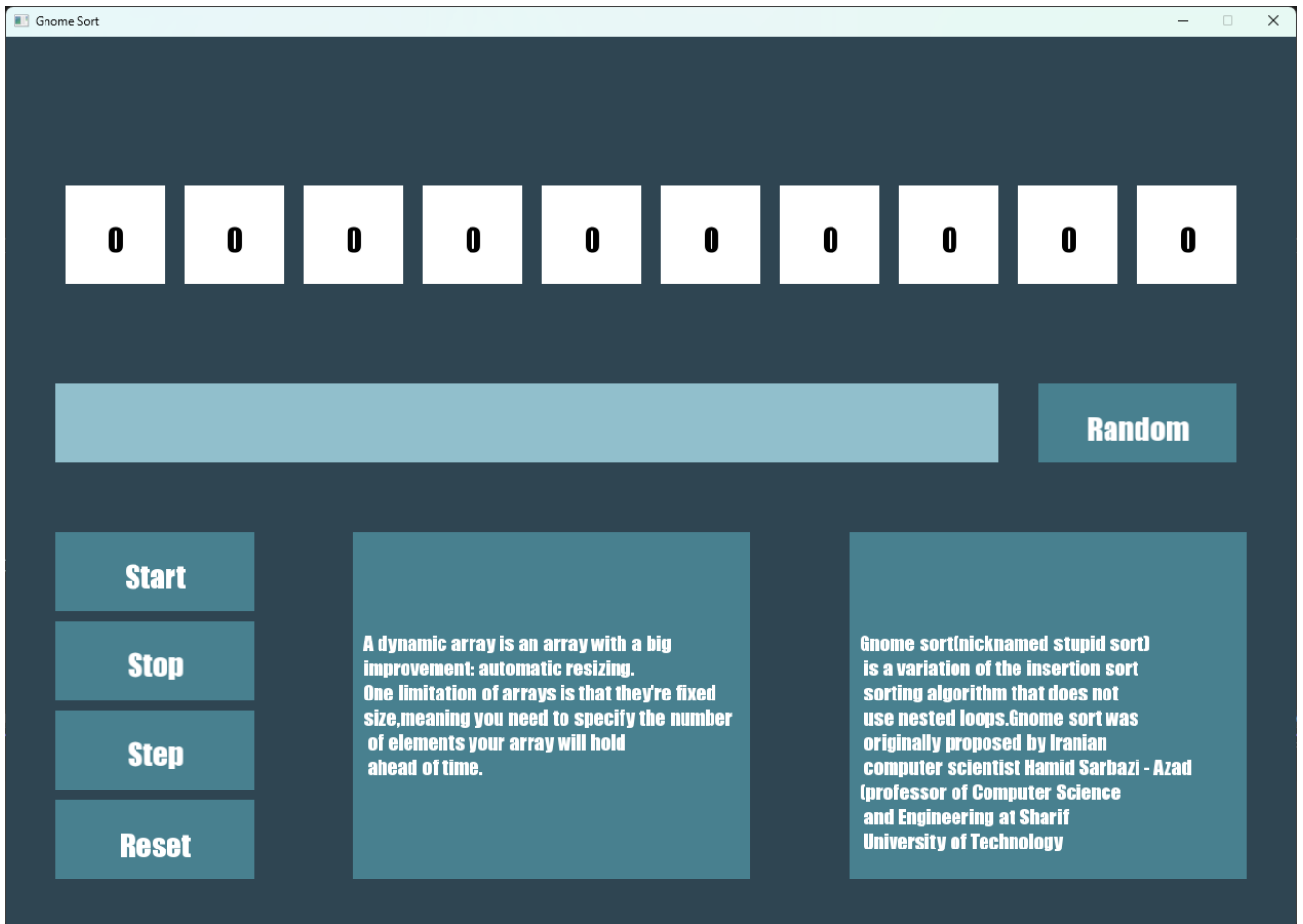
    void NextElem();

    void DeselectedElem();

private:
    sf::RectangleShape elem;
};
```

ПриложениеБ

Результатытестирования



РисунокБ.1—запущеннаянавывполнениепрограмма

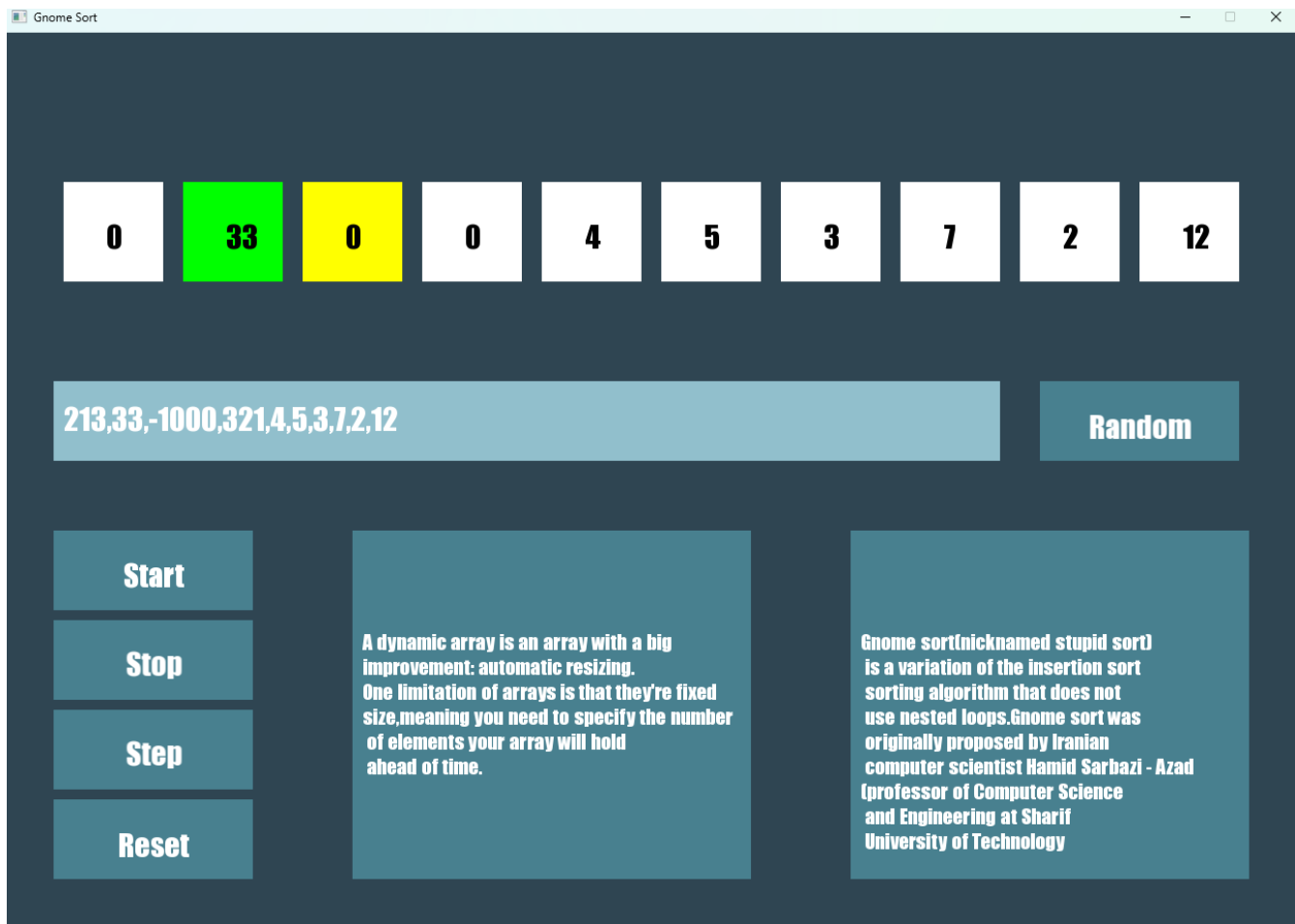
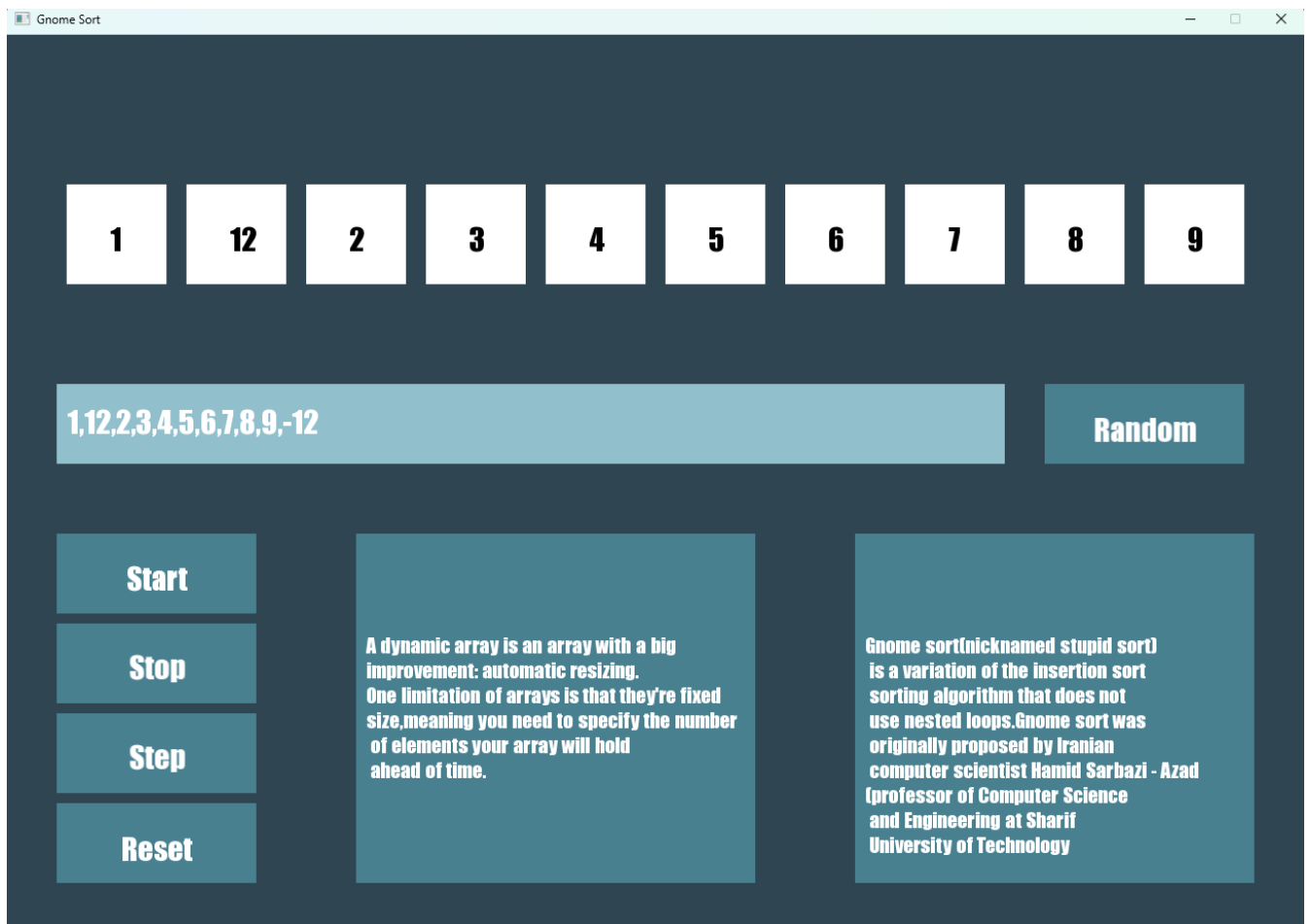
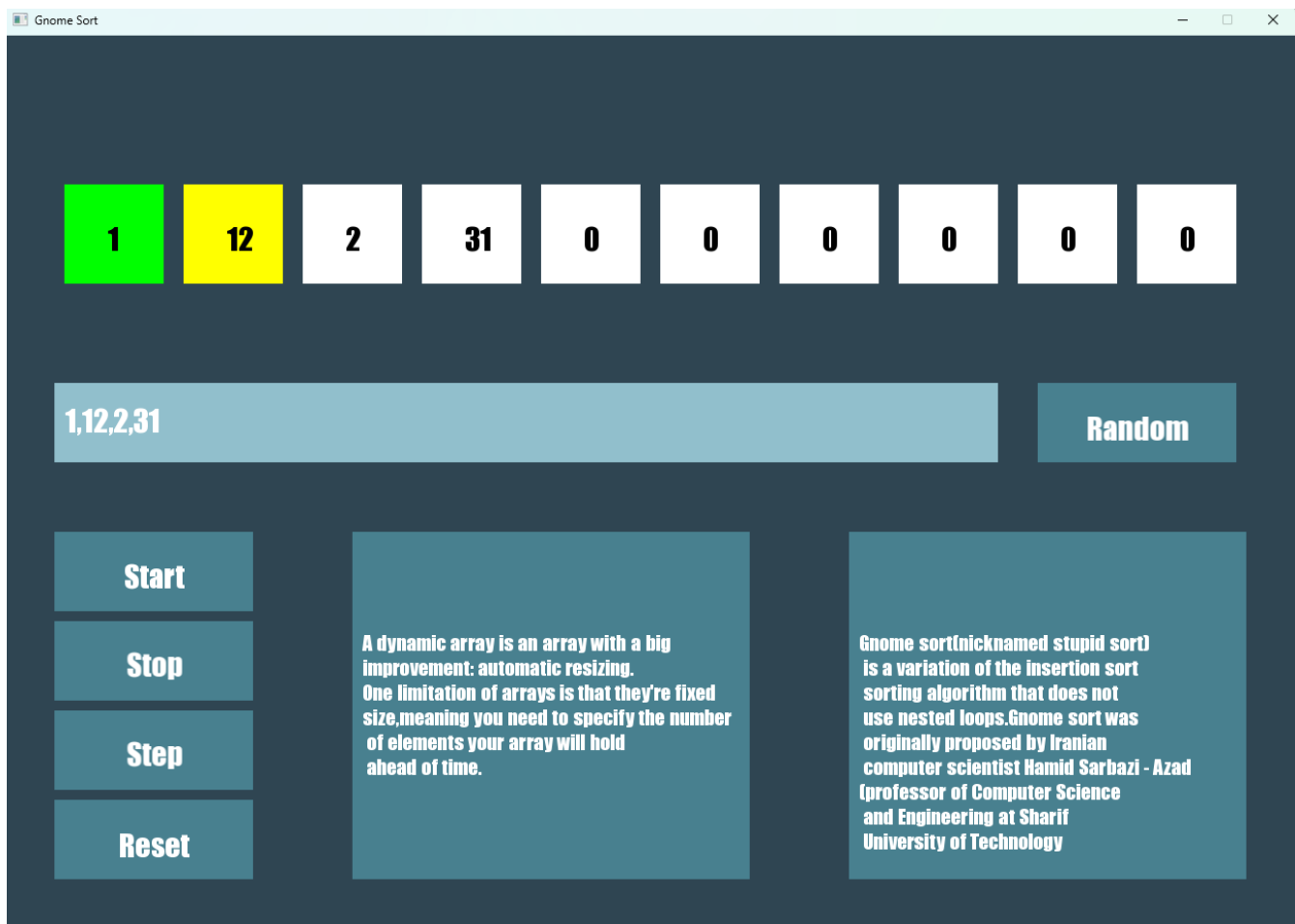


Рисунок Б.2—Ввод чисел, не входящих в диапазон



РисунокБ.3—Ввод большего количества чисел, чем ожидалось



РисунокБ.4—Ввод меньшего количества чисел, чем ожидалось

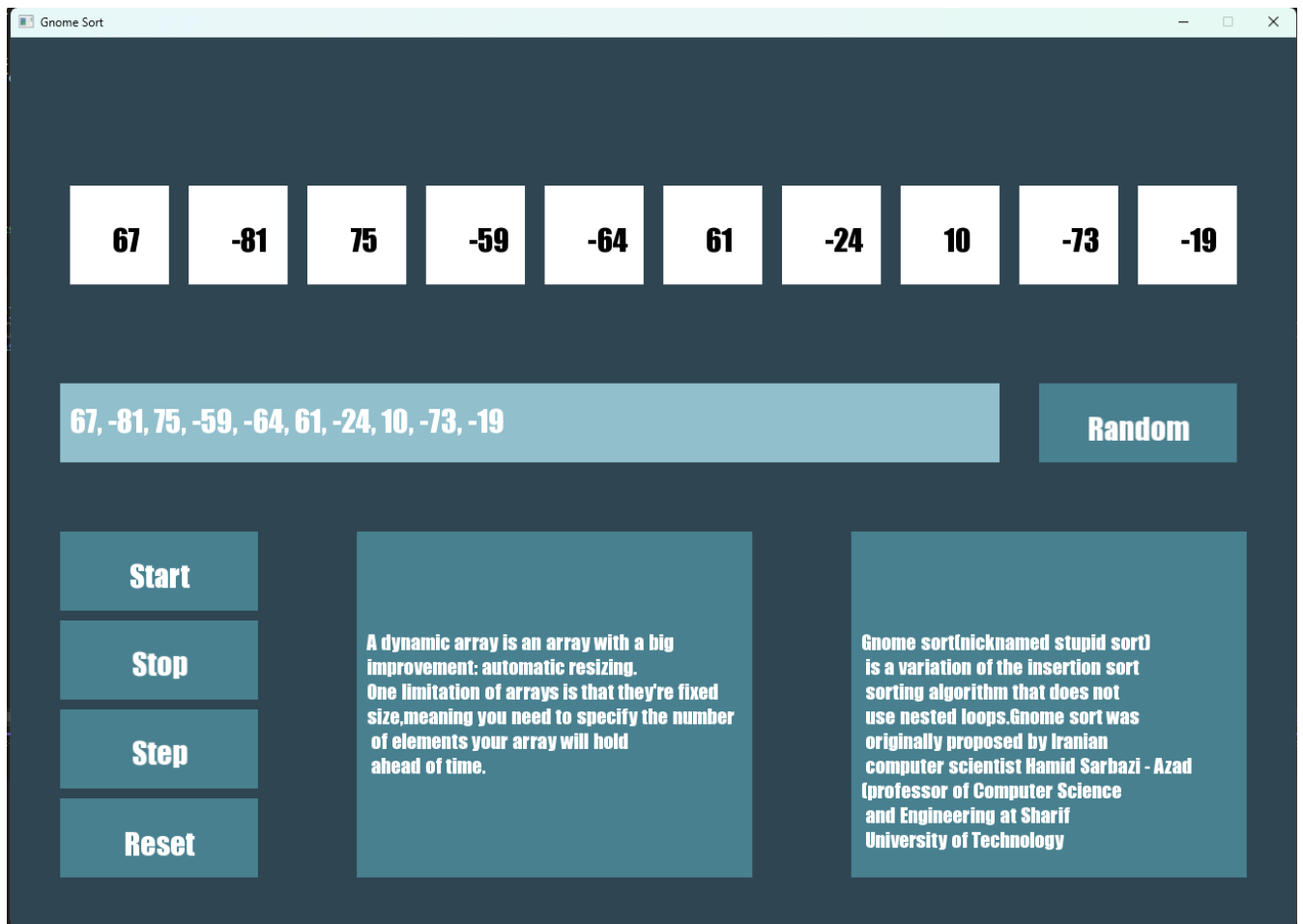


Рисунок Б.5—сгенерированный массив

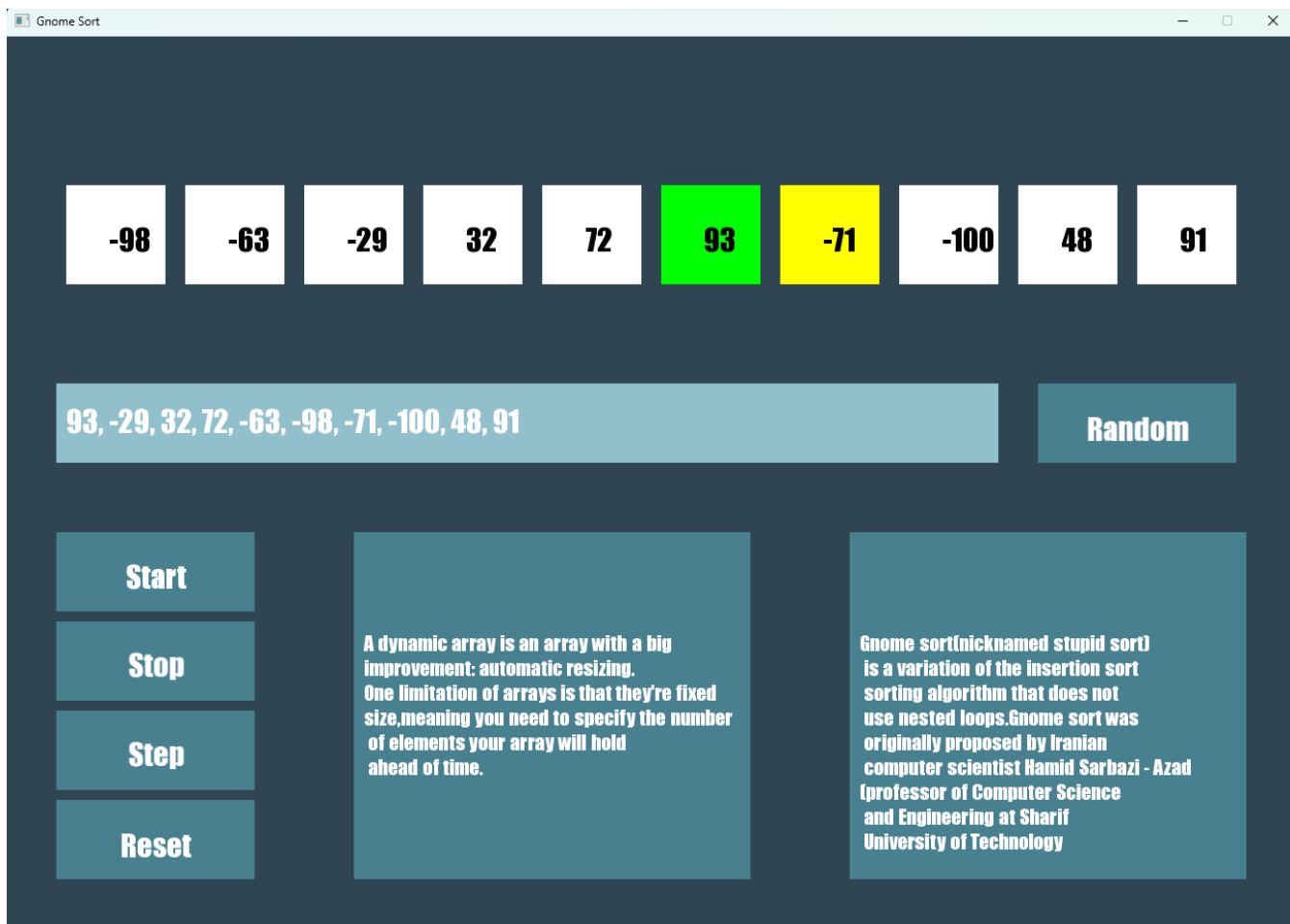


Рисунок Б.6—нажатие кнопки “Start”

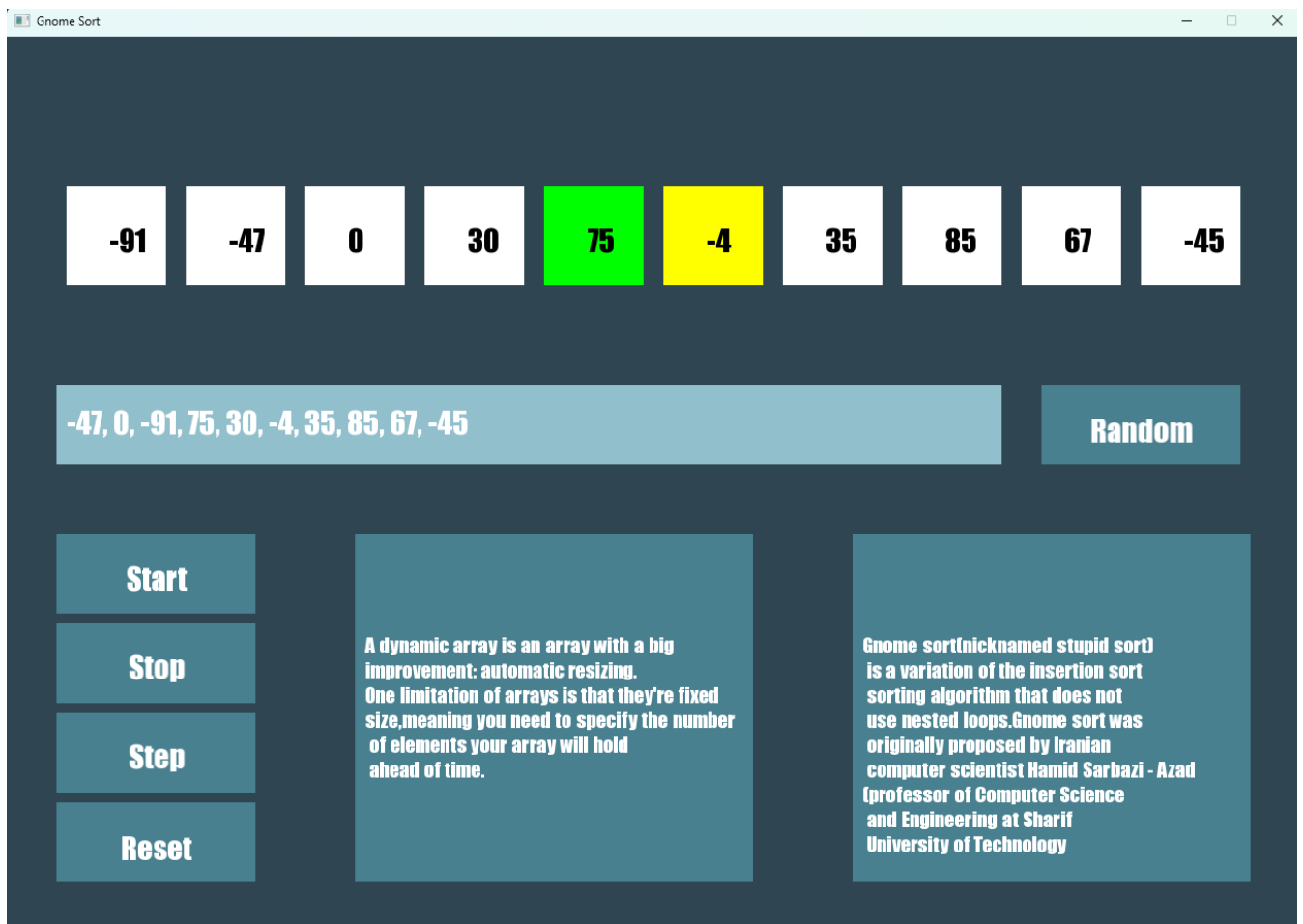
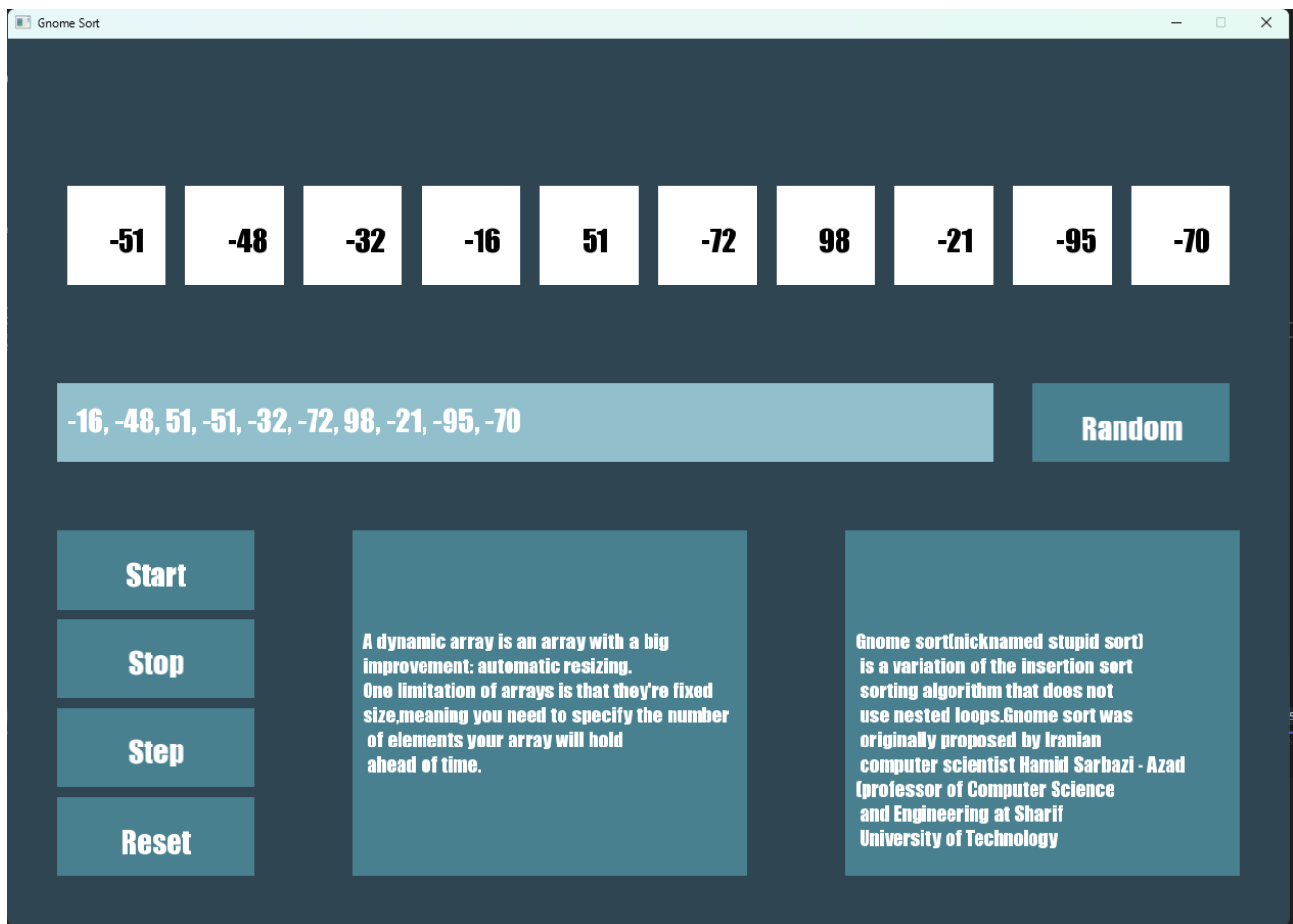


Рисунок Б.7—нажатие кнопки “Step”



РисунокБ.8—нажатие кнопки “Stop”

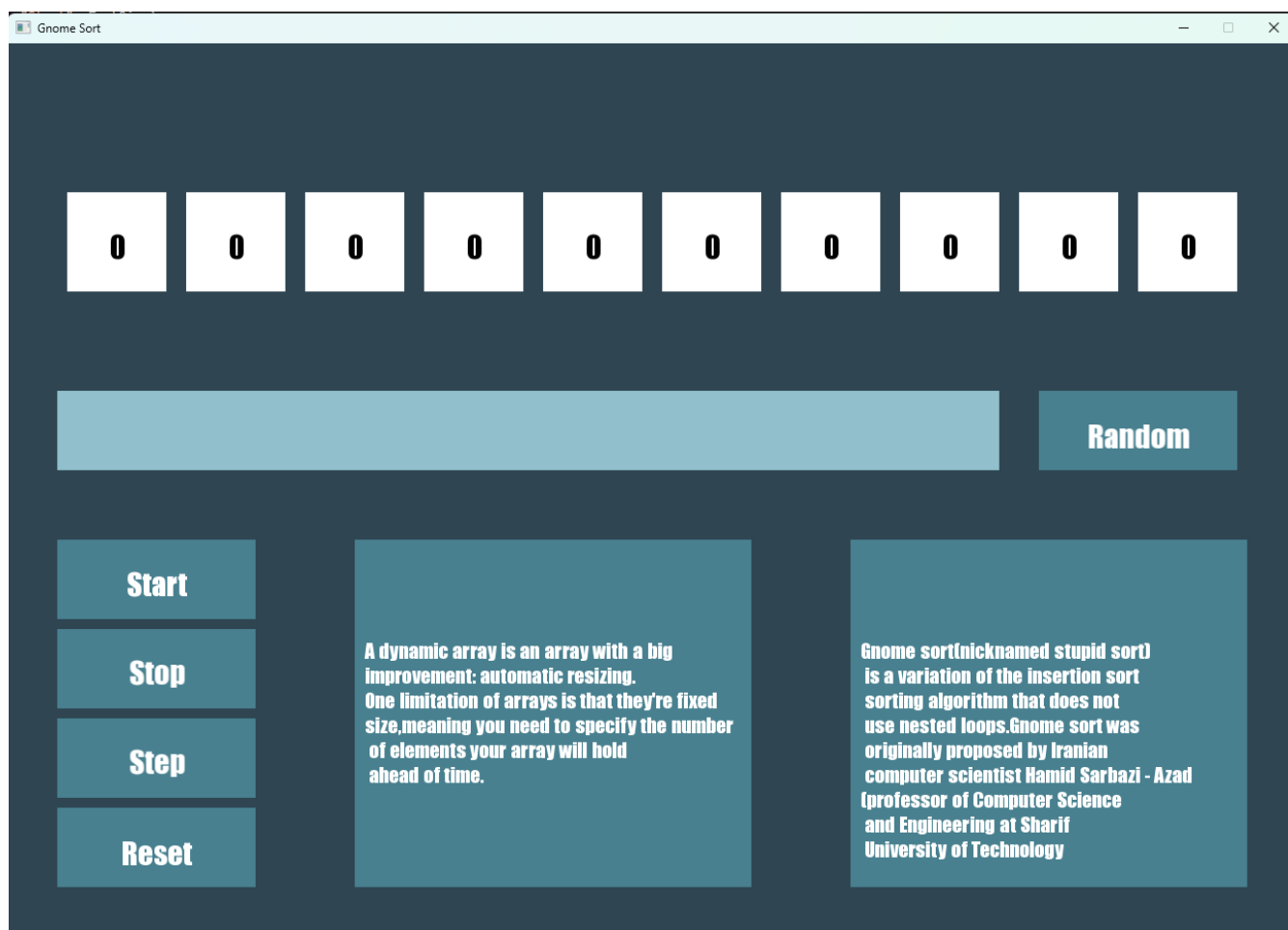


Рисунок Б.9—нажатие кнопки “Reset”