**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**Отчет**

**по учебной практике**

**«Визуализация алгоритма на языке Java»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6381 |  | Вероха В. Н. |
| Студент гр. 6381 |  | Вергейчик Г. Л. |
| Студент гр. 6381 |  | Ширяев Я. А. |
| Руководитель |  | Фирсов М. А. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Вероха В. Н. группы 6381 | | |
| Студент Вергейчик Г. Л. группы 6381 | | |
| Студент Ширяев Я. А. группы 6381  Тема практики: Визуализация алгоритма на языке Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: Backtracking (поиск с возвратом) на примере квадрирования квадрата. | | |
| Сроки прохождения практики: 27.06.2018 – 10.07.2018 | | |
| Дата сдачи отчета: 00.07.2018 | | |
| Дата защиты отчета: 00.07.2018 | | |
|  | | |
| Студентка |  | Вероха В. Н. |
| Студент |  | Вергейчик Г. Л. |
| Студент |  | Ширяев Я. А. |
| Руководитель |  | Фирсов М. А. |

**Аннотация**

Целью данной работы является разработка визуализатора алгоритма backtracking (поиск с возвратом) на примере квадрирования квадрата.

При разработке программы использовался язык Java и среда IntelliJ IDEA. Для создания графического интерфейса – библиотека Swing.

**Summary**

The goal of this work is developing a visualizer of the algorithm backtracking (search with a return) for the example of square squaring.

For the development of the program used the Java language and the IntelliJ IDEA environment. For graphical interface - the Swing library.

**Содержание**

[Введение 5](#_Toc518647229)

[1. Требования к программе 6](#_Toc518647230)

[1.1. Формулировка задания 6](#_Toc518647231)

[1.2. Формальная постановка задачи 6](#_Toc518647232)

[1.3. Объяснение алгоритма 6](#_Toc518647233)

[1.3.1. Теоретическая справка 6](#_Toc518647234)

[1.3.2. Описание алгоритма 7](#_Toc518647235)

[1.4. Исходные требования к программе 8](#_Toc518647236)

[1.4.1. Требования к вводу исходных данных 8](#_Toc518647237)

[1.4.2. Требования к визуализации 8](#_Toc518647238)

[1.5. Уточнение требований после сдачи первой версии 8](#_Toc518647239)

[1.6. Уточнение требований после сдачи второй версии 8](#_Toc518647240)

[2. План разработки и распределения ролей в бригаде 9](#_Toc518647241)

[2.1. План разработки 9](#_Toc518647242)

[2.2. План разработки 9](#_Toc518647243)

[3. Особенности реализации 9](#_Toc518647244)

[3.1. Спецификация 9](#_Toc518647245)

[3.1.1. Use case диаграмма 9](#_Toc518647246)

[3.1.2. Графический интерфейс 10](#_Toc518647247)

[3.2. Основные классы программы 16](#_Toc518647248)

[3.3. Диаграмма классов 17](#_Toc518647249)

[4. Тестирование программы 18](#_Toc518647250)

[Заключение 25](#_Toc518647251)

[Приложение A. Код програмы. 26](#_Toc518647252)

[Приложение Б. Код unit-тестов. 40](#_Toc518647253)

# Введение

В данной работе разрабатывается визуализатор алгоритма backtracking (поиск с возвратом) на примере квадрирования квадрата.

Разработанная программа решает задачу о разбиении квадрата на конечное число меньших по размеру квадратов. Пользователь вводит размеры квадрата, а на выходе получает его разбиение на меньшие квадраты. При этом количество квадратов минимально.

Целью работы в проекте является получение навыков работы в команде, а также усовершенствование знаний о языке программирования Java и системы контроля версий GitHub.

# Требования к программе

## Формулировка задания

Разработать визуализатор алгоритма backtracking (поиск с возвратом) на примере квадрирования квадрата с помощью языка программирования Java.

## Формальная постановка задачи

Дан квадрат, размеры которого задает пользователь. Большой квадрат (поле) требуется разбить на меньшие по размеру квадраты.

Должны быть соблюдены следующие требования:

* количество квадратов разбиения минимально;
* внутри поля нет пустот;
* меньшие квадраты не перекрываются и не выходят за пределы поля.

Входные данные:

Размер поля в клетках - одно целое число **N** (2≤**N**≤40).

Выходные данные:

Разбиение поля на минимальное количество меньших квадратов. Визуализация работы алгоритма.

## Объяснение алгоритма

### 1.3.1. Теоретическая справка

Поиск с возвратом (англ. backtracking) - общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве. Как правило позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …», «Существует ли объект…» и т. п.

Термин backtracking был введен в 1950 году американским математиком Дерриком Генри Лемером.

Незначительные модификации метода поиска с возвратом, связанные с представлением данных или особенностями реализации, имеют и иные названия: метод ветвей и границ, поиск в глубину, метод проб и ошибок и т. д. Поиск с возвратом практически одновременно и независимо был изобретен многими исследователями ещё до его формального описания.

### 1.3.2. Описание алгоритма

Любое целое число может быть представлено в виде произведения конечного числа простых чисел. Поэтому задача квадрирования квадрата сводится к нахождению наименьшего простого делителя размера поля (большего квадрата), нахождения его разбиения, а затем масштабирования.

Для всех квадратов, стороны которых являются простыми числами в пределах диапазона от 2 до 61 (в условиях нашей задачи диапазон – [2, 40]), уже существует разбиение на минимальное число квадратов меньших размеров (см. рис. 1).

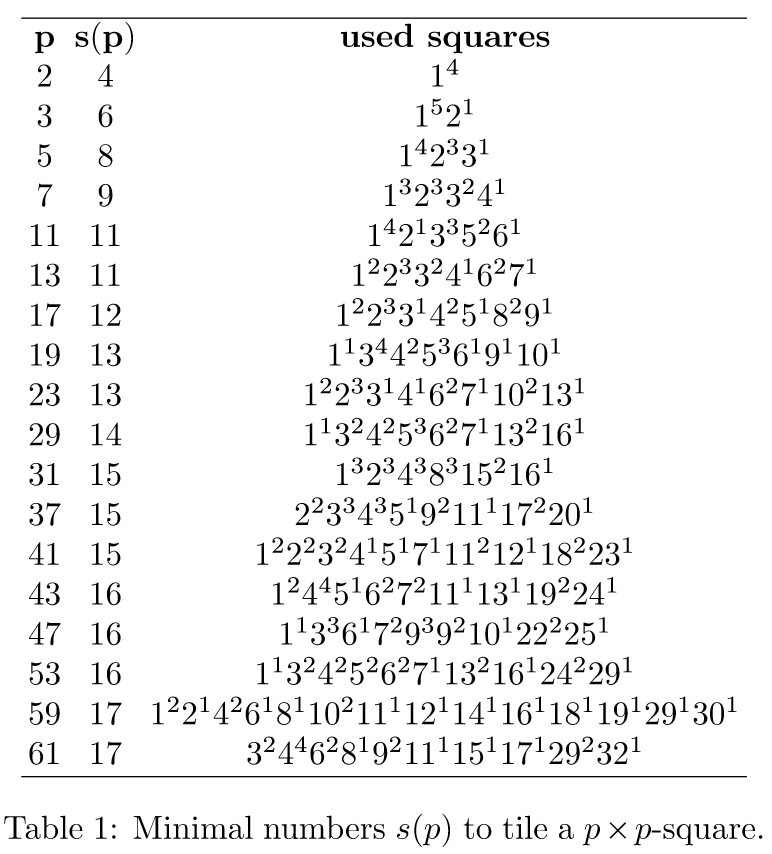


Рисунок 1. Разбиение квадратов, размер поля которого – простое число. Обозначения: p – размер поля (сторона квадрата, который нужно квадрировать) – простое число, s(p) – минимальное число квадратов, требуемое для разбиения большего квадрата.

В программе для размера поля, являющегося простым числом, задано количество меньших квадратов определенных размеров, с помощью которых может быть заполнен больший квадрат.

При квадрировании находится наименьший простой делитель введенного числа (размера поля). Далее применяется алгоритм поиска с возвратом на квадрате, сторона которого является простым числом, а затем масштабирует его, получая разбиение изначально заданного квадрата.

## Исходные требования к программе

### 1.4.1. Требования к вводу исходных данных

Программа предоставляет пользователю графический интерфейс. Входные данные считываются при вводе размера поля (большего квадрата) пользователем. После запуска алгоритма, пользователь наблюдает результаты квадрирования большего квадрата по шагам, либо конечный результат.

### Требования к визуализации

Программа предоставляет интерфейс с пояснениями и схематичным изображением поля большего квадрата, а также квадратов вставки. Визуализация должна представлять собой окно с меню.

## Уточнение требований после сдачи первой версии

* Связать алгоритм с графическим интерфейсом.
* В разделе меню «About programm» отредактировать текст с информацией.
* Доработать раздел меню «Enter size».

## Уточнение требований после сдачи второй версии

* При переходе по шагам результата добавить кнопку «Result» и «Menu».
* Сдвинуть поле расстановки квадратов от края экрана.
* Сделать расставляемые квадраты разных цветов.
* Добавить status bar с номером шага. Например, 2/10.
* При вводе размера поля указать диапазон значений.

# План разработки и распределения ролей в бригаде

## План разработки

28. 06. 2018 – разработка спецификации программы.

30. 06. 2018 – написание логики программы.

02. 07. 2018 – разработка алгоритма и визуализация меню.

04. 07. 2018 – связь алгоритма и графического интерфейса.

05. 07. 2018 – доработка графического интерфейса, визуализация хода работы алгоритма.

## План разработки

Распределение ролей представлено в таблице 1.

|  |  |
| --- | --- |
| Студент | Роль |
| Вероха В. Н. | Спецификация, отчет, тестирование |
| Вергейчик Г. Л. | Визуалицая |
| Ширяев Я. А. | Алгоритм |

Таблица 1. Распределение ролей.

# Особенности реализации

## Спецификация

### 3.1.1. Use case диаграмма

Ниже представлена диаграмма вариантов использования, показывающая взаимоотношения между пользователем и программой.

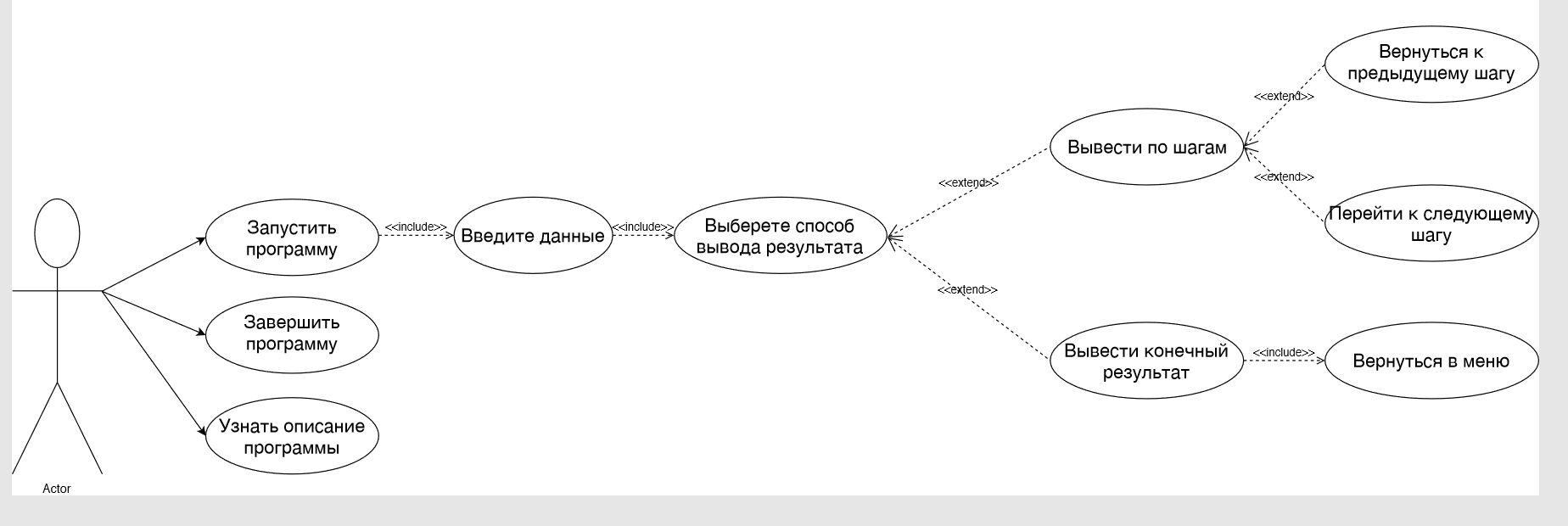


Рисунок 2. Use case диаграмма.

### 3.1.2. Графический интерфейс

Ниже представлены эскизы работы программы.

При запуске программы появляется меню (рис. 3) со следующими кнопками:

* «Enter the size» – запуск работы алгоритма (рис. 4);
* «About the program» – описание программы (рис. 12);
* «Exit» – завершение программы (рис. 13).



Рисунок 3. Меню программы.

При нажатии кнопки «Enter the size» программа запрашивает у пользователя размеры поля (рис. 4).

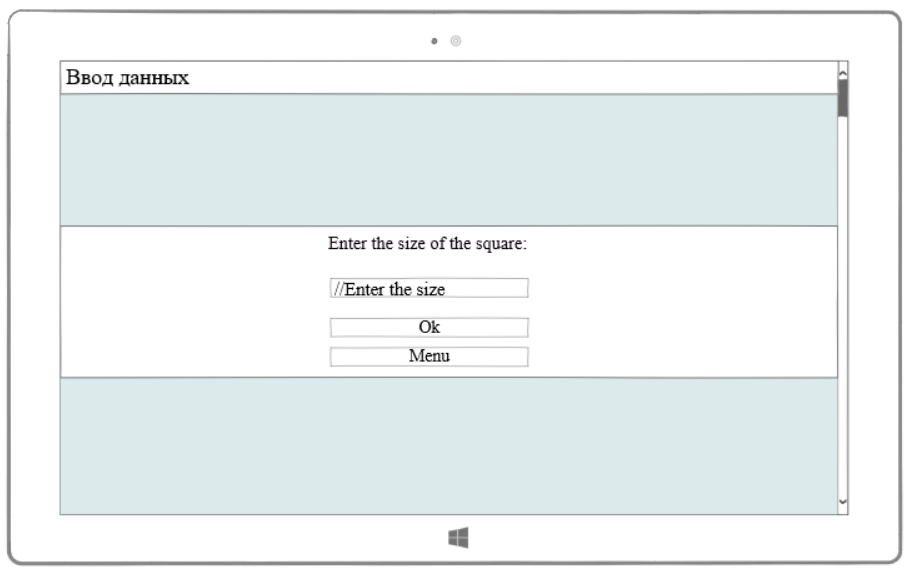


Рисунок 4. Ввод данных.

При неверном вводе размеров поля (большего квадрата) всплывает предупреждении (рис. 5). И у пользователя повторно запрашиваются размеры поля (рис. 6).

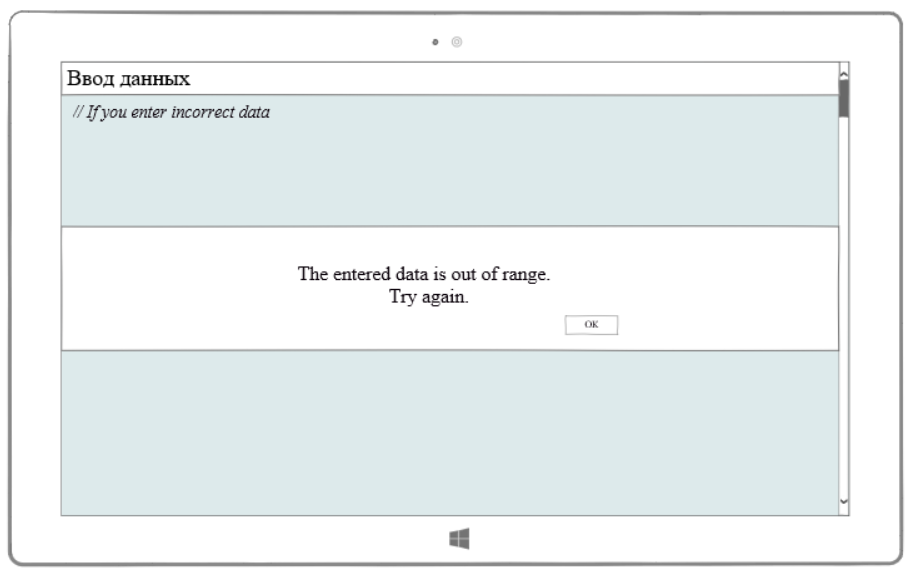


Рисунок 5. Предупреждении о неверно введенных данных.

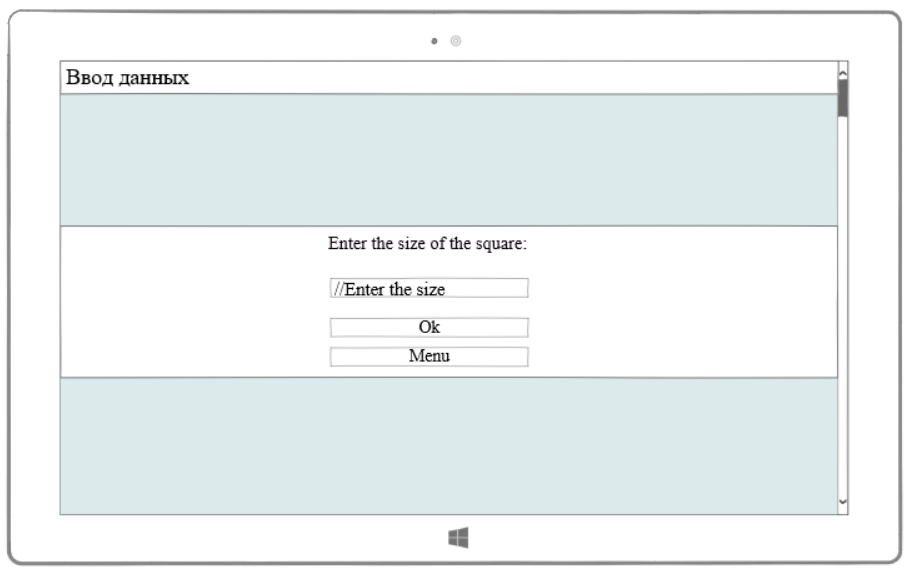


Рисунок 6. Повторный ввод данных.

Далее на примере поля, сторона которого равна 2, рассмотрим работу программы (рис. 7 – 11).

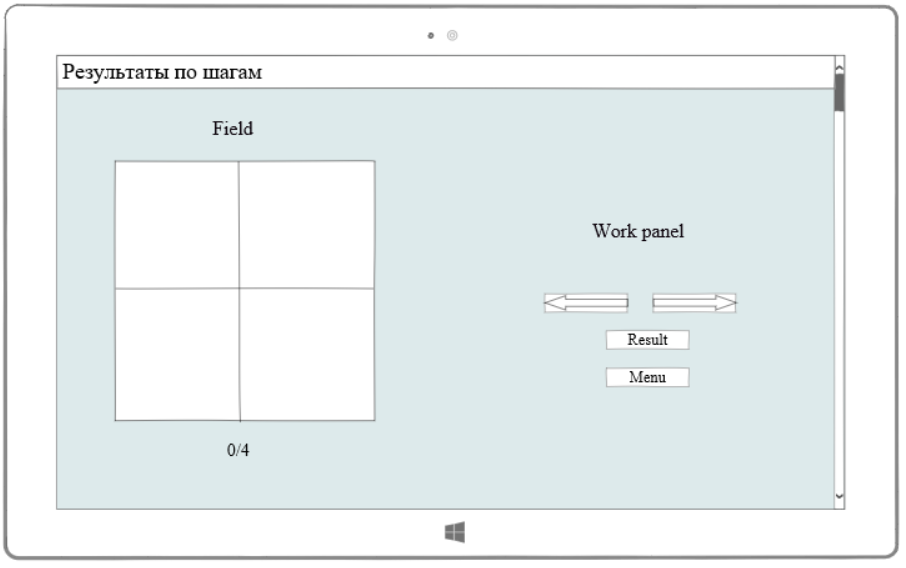


Рисунок 7. Результаты по шагам.

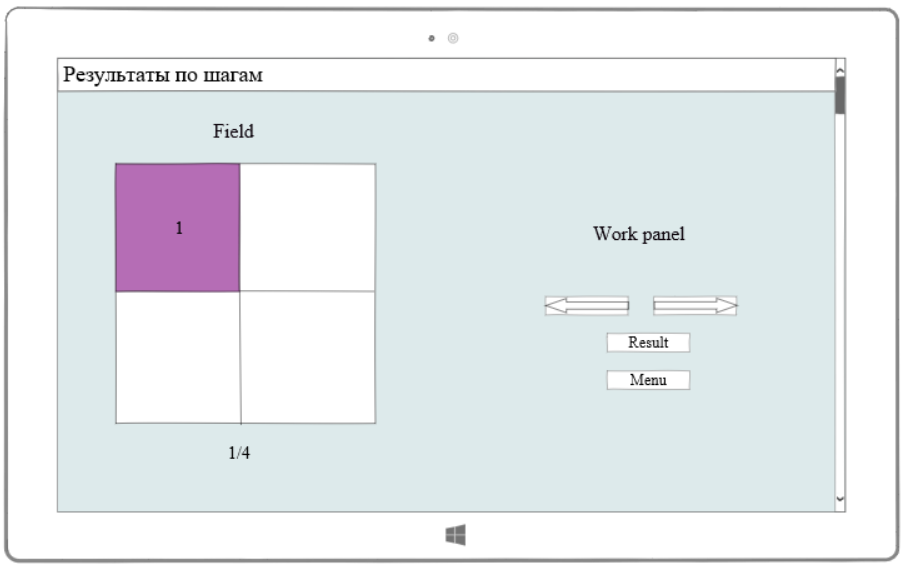


Рисунок 8. Результаты по шагам.

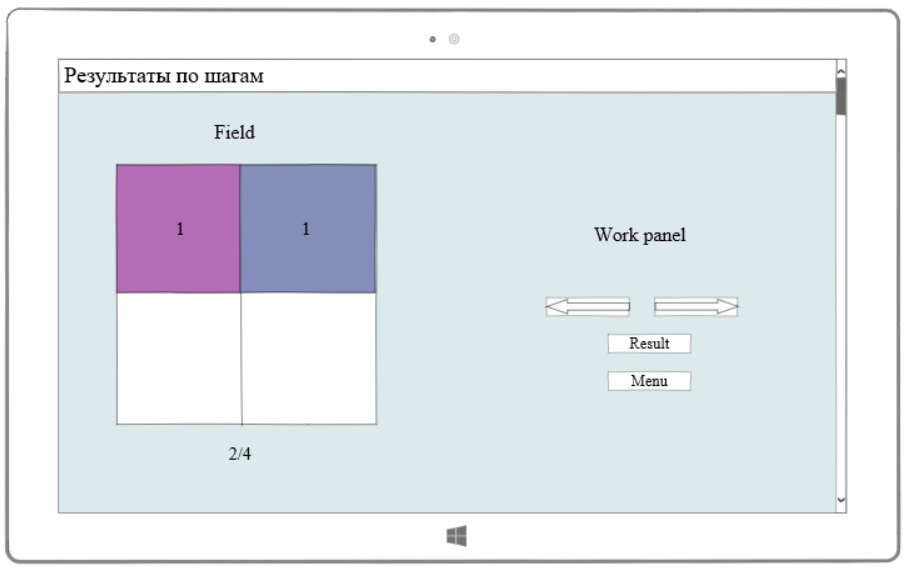


Рисунок 9. Результаты по шагам.

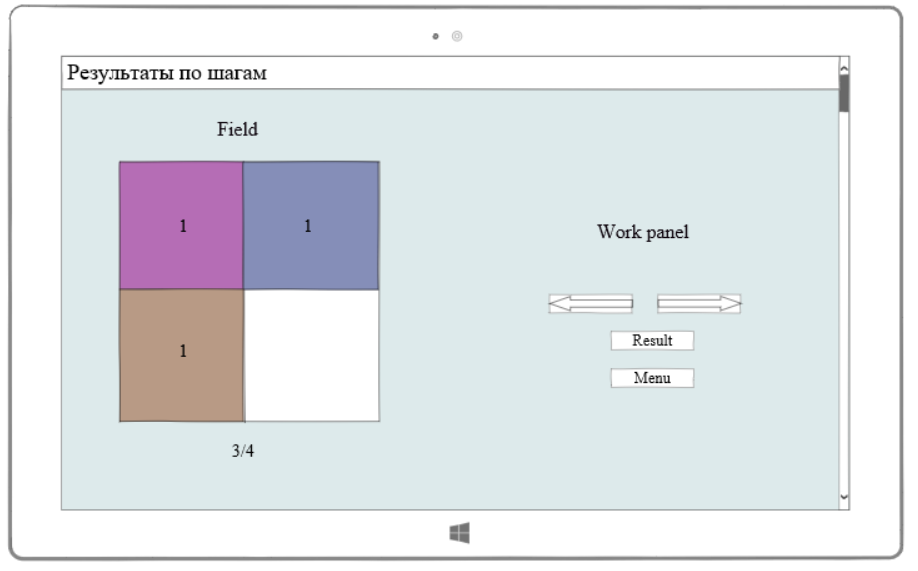


Рисунок 10. Результаты по шагам.

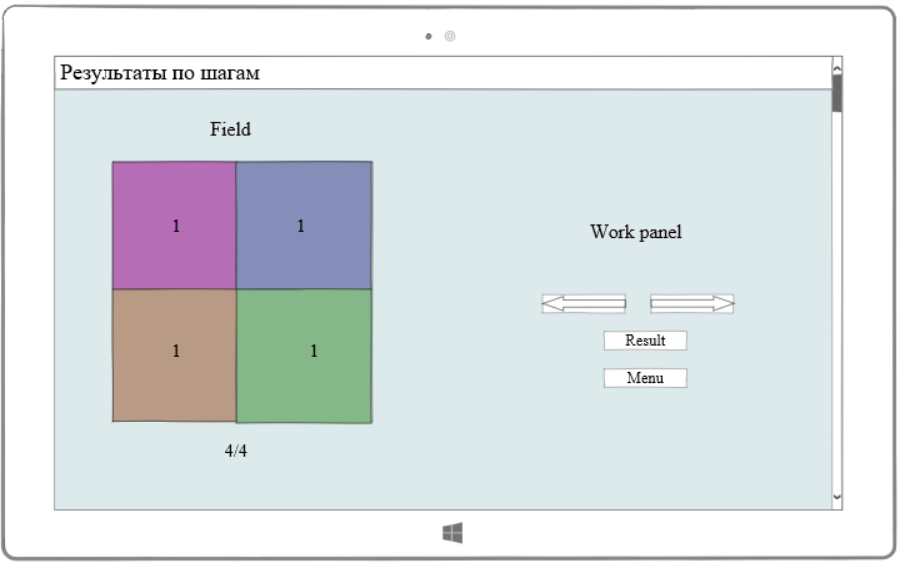


Рисунок 11. Конечный результат.

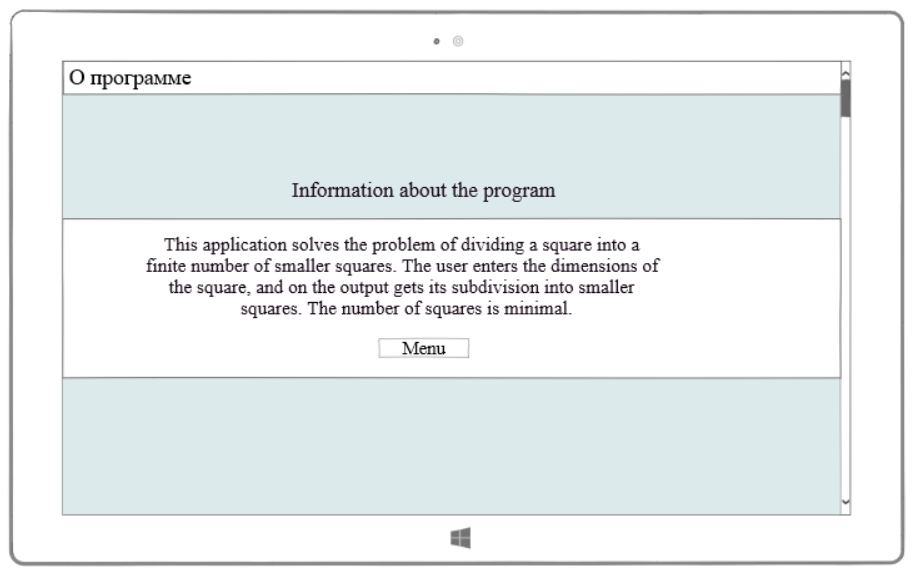


Рисунок 12. Описание программы.

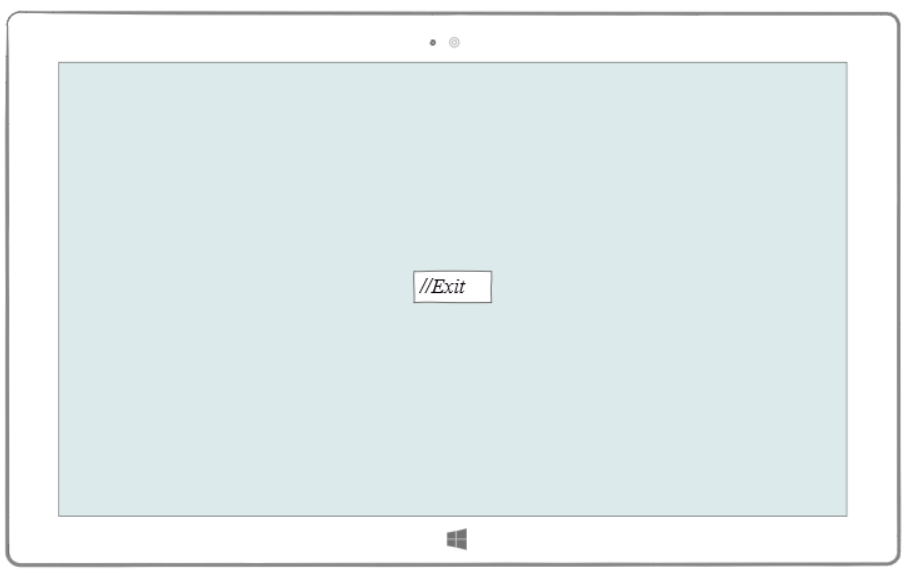


Рисунок 13. Завершение программы.

## Основные классы программы

Ниже представлены основные классы программы:

*class Controller* – инициализирует Model и View;

*class View* – окно, вызывающее эскизы;

*class Model* – описывает алгоритм поиска с возвратом;

*class Field* – задает поле для расстановки квадратов;

*class Square* – описывает меньшие квадраты, которые нужно расставить;

*class PrimeNumbers* – задает разбиение большего квадрата, сторона которого простое число, на меньшие квадраты (см. рис. 1);

*class Number* – производит поиск простого числа в заданном размере поля для упрощения вычислений;

*class StartApplication* – вывод начального меню;

*class EnterApplication* – обрабатывает кнопку ввода размера поля;

*class InformationApplication* – обрабатывает кнопку «About programm»;

*class WorkApplication* – обрабатывает кнопки рабочей области при просмотре результатов;

*class Conditions* – описывает все состояния квадратов по шагам.

## Диаграмма классов

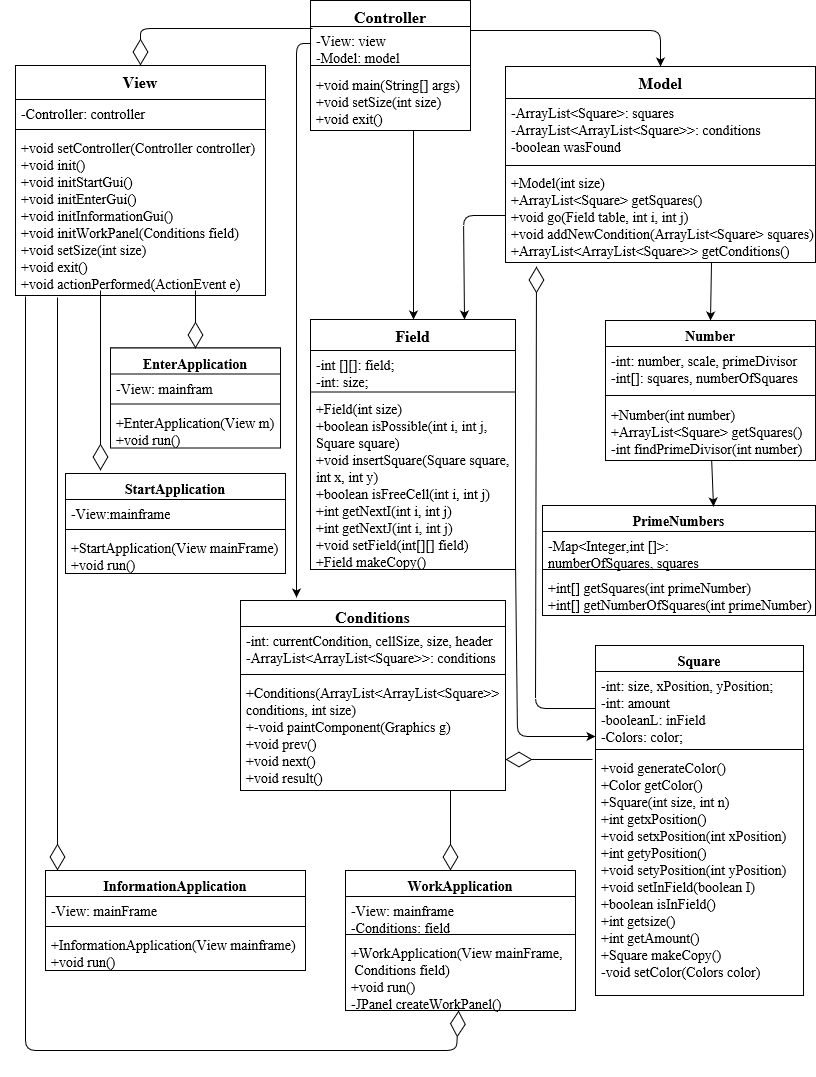


Рисунок 14. Диаграмма классов.

# Тестирование программы

При запуске программы пользователь видит начальное меню (рис. 15).

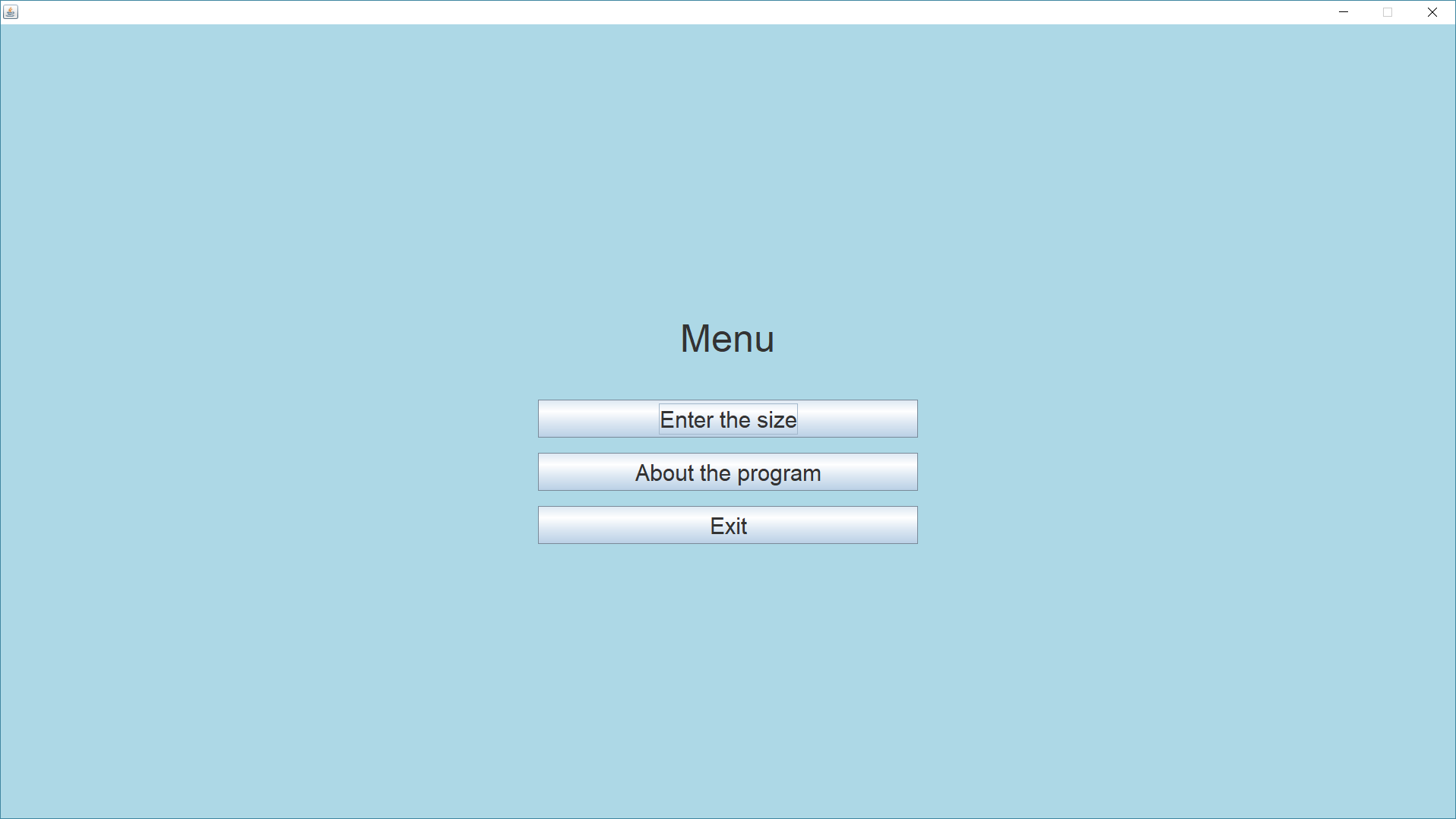


Рисунок 15. Начальное меню.

При нажатии кнопки «About the program» пользователь может ознакомиться с информацией о программе (рис.16). При нажатии на кнопку «Menu» пользователь возвращается в начальное меню.

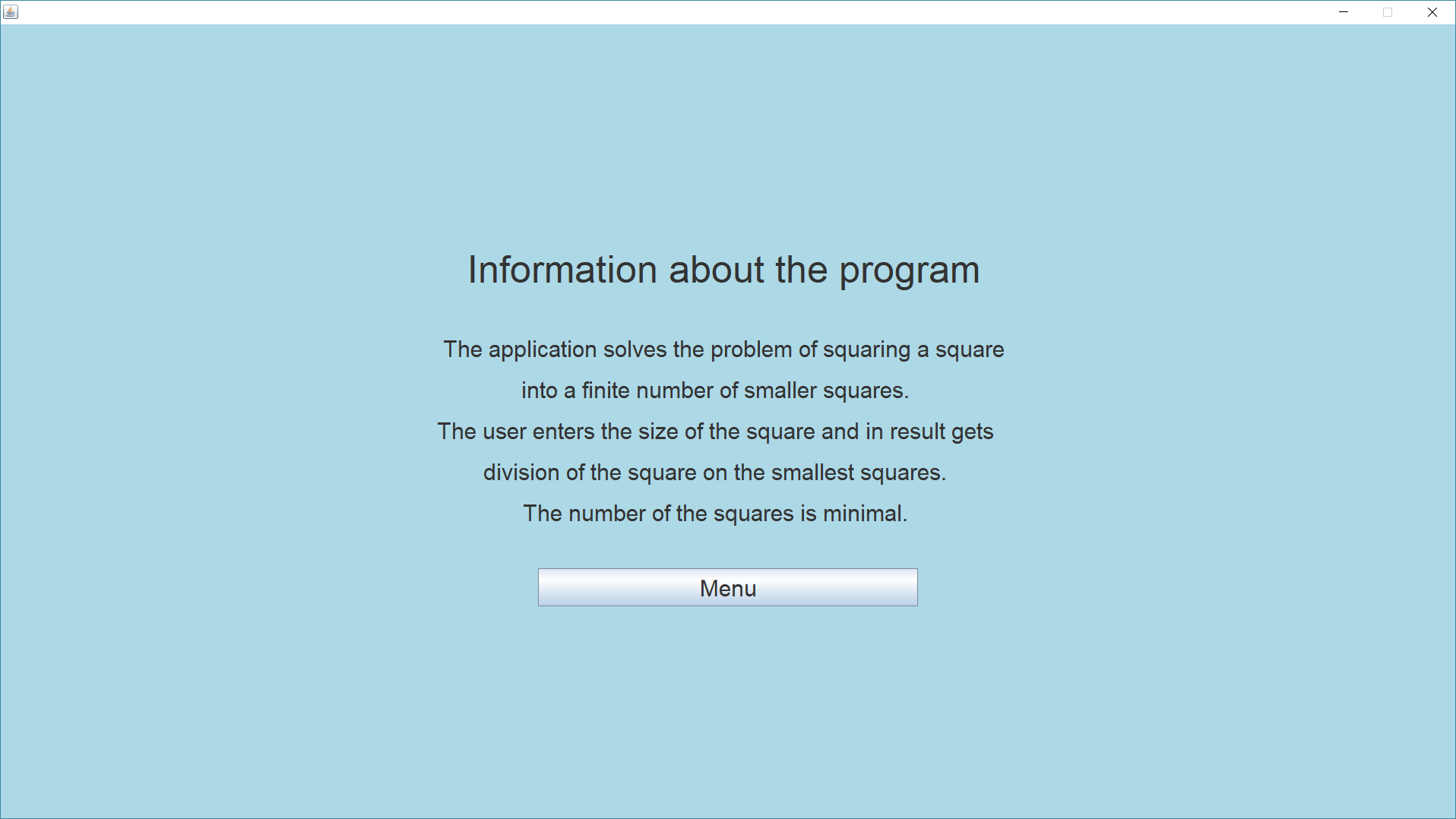


Рисунок 16. Тестирование кнопки «About the program».

При нажатии кнопки «Exit» программа успешно завершается.

Далее проведем тестирование кнопки «Enter the size». При нажатии пользователь видит окно ввода размера поля (рис. 17). Также пользователь может вернуться к начальному меню.

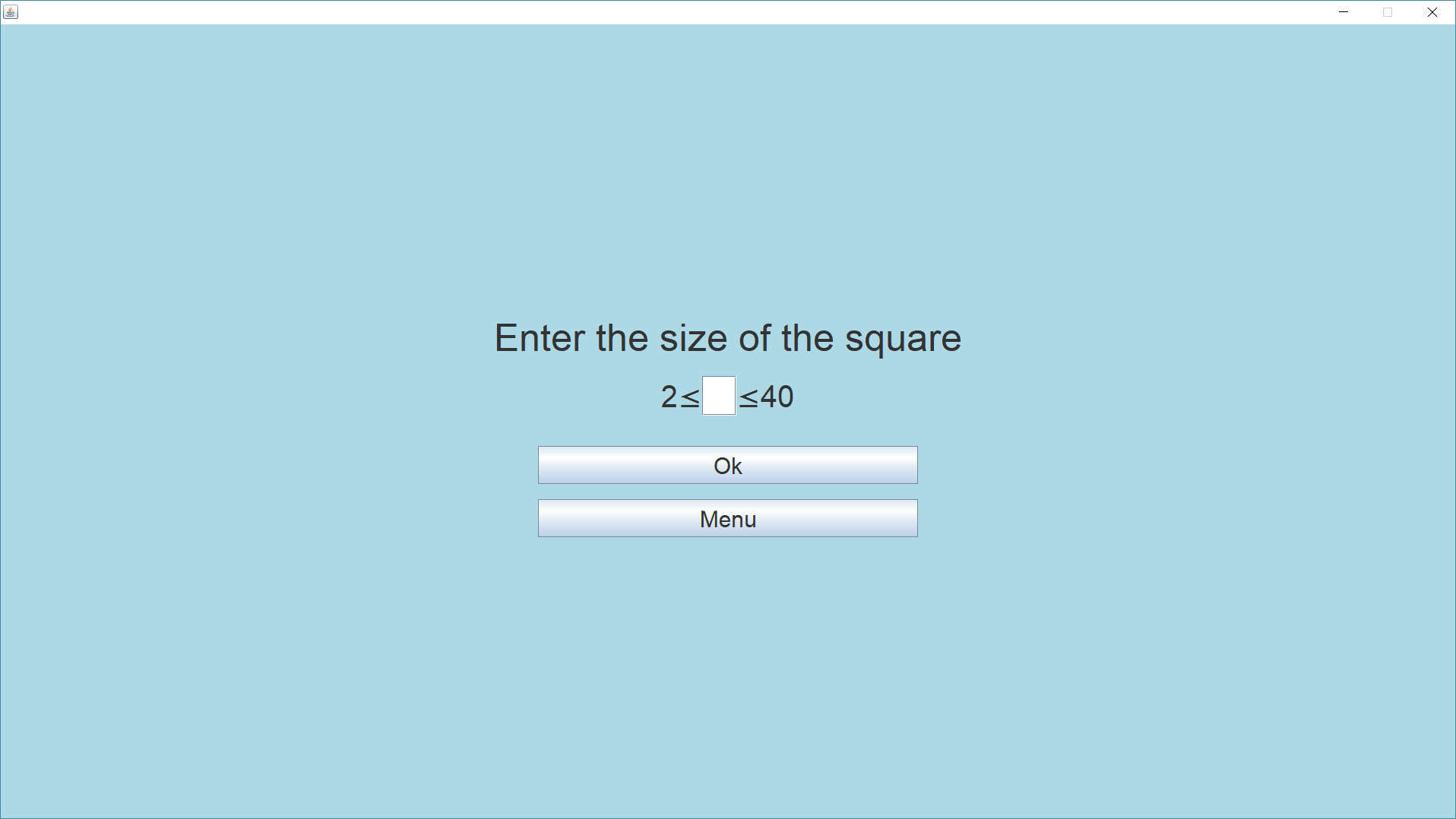


Рисунок 17. Тестирование кнопки «Enter the size».

Пользователь может ввести числа от 2 до 40. При неверном вводе всплывает окно с предупреждением об ошибке (рис. 18).

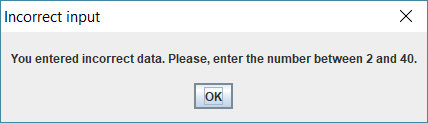


Рисунок 18. Предупреждение о некорректном вводе размера поля.

После нажатия кнопки «OK» пользователь возвращается в рабочую область кнопки «Enter the size» (рис. 17) и продолжает ввод размера поля.

После того, как пользователь задал размеры поля (рис. 19), можно просмотреть результаты работы алгоритма по шагам.

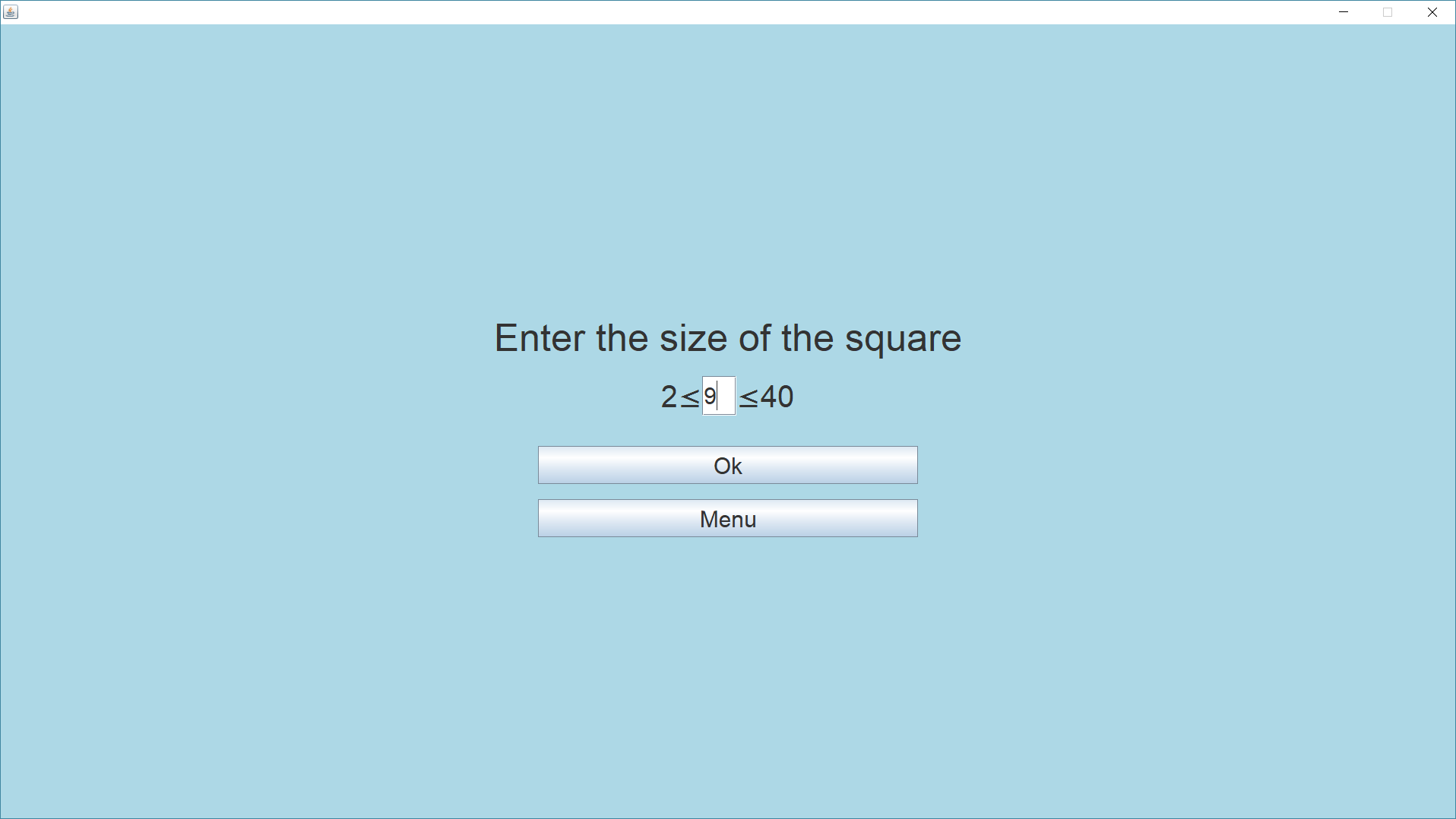


Рисунок 19. Ввод пользователем размера поля 9 на 9.

Рассмотрим работу программы на примере квадрата 9 на 9 (рис. 20).

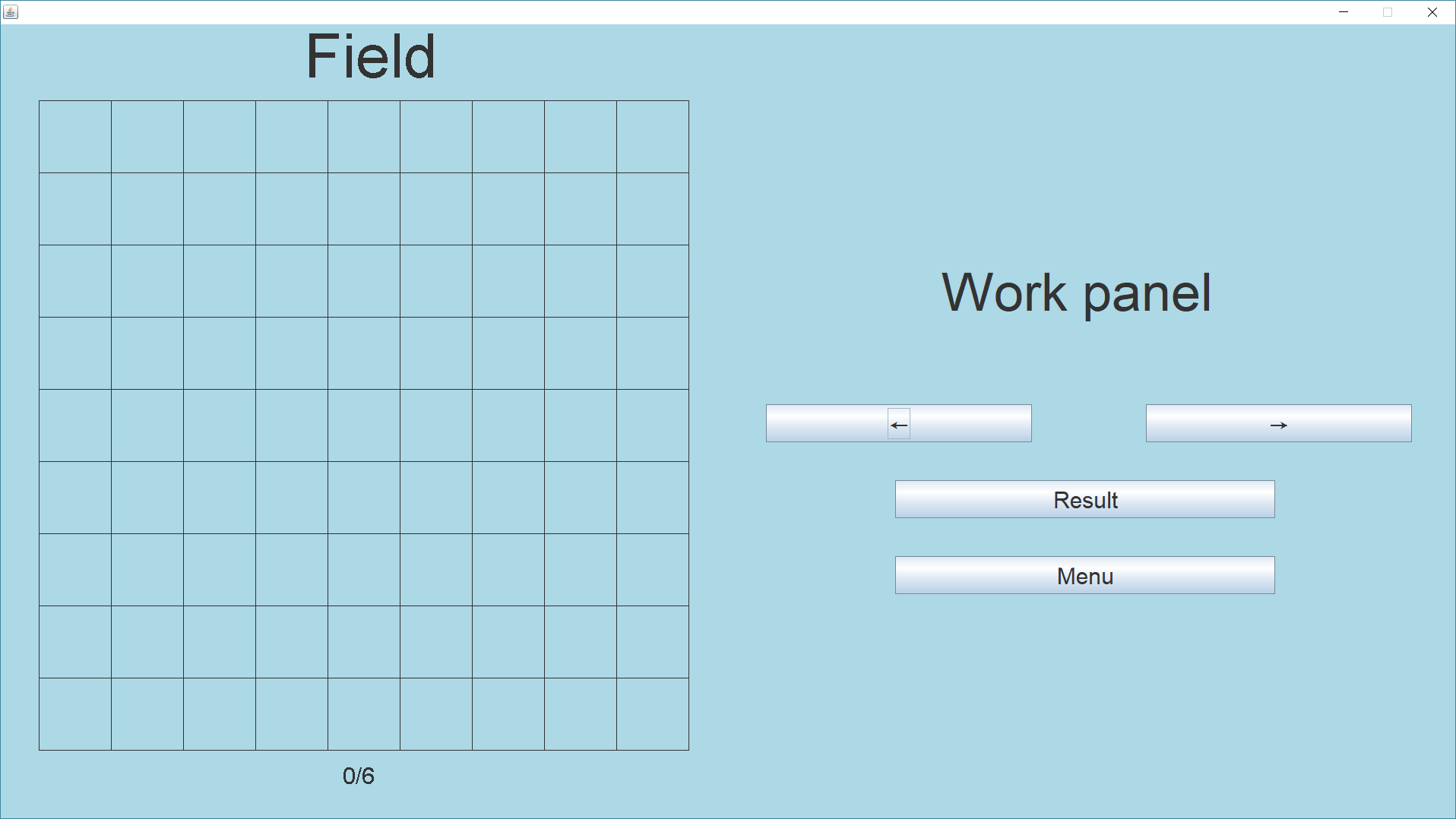


Рисунок 20. Работа алгоритма на примере квадрата 9 на 9.

На этом этапе программы пользователь может вернуться в начальное меню с помощью кнопки «Menu», посмотреть конечный результат – кнопка «Result», а также сделать шаг назад и шаг вперед (рис. 21 – 26).

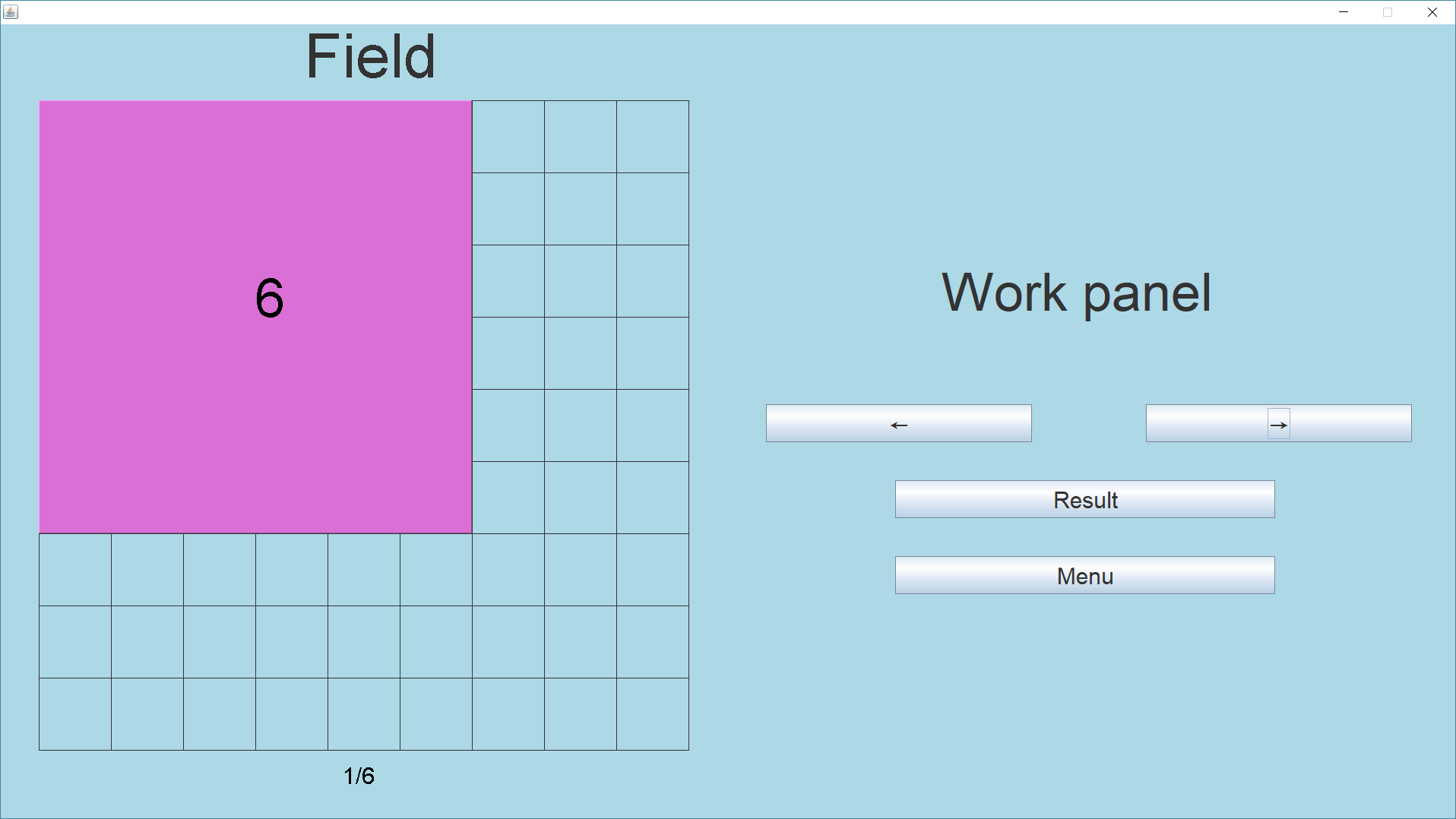


Рисунок 21. Шаг 1 работы алгоритма.

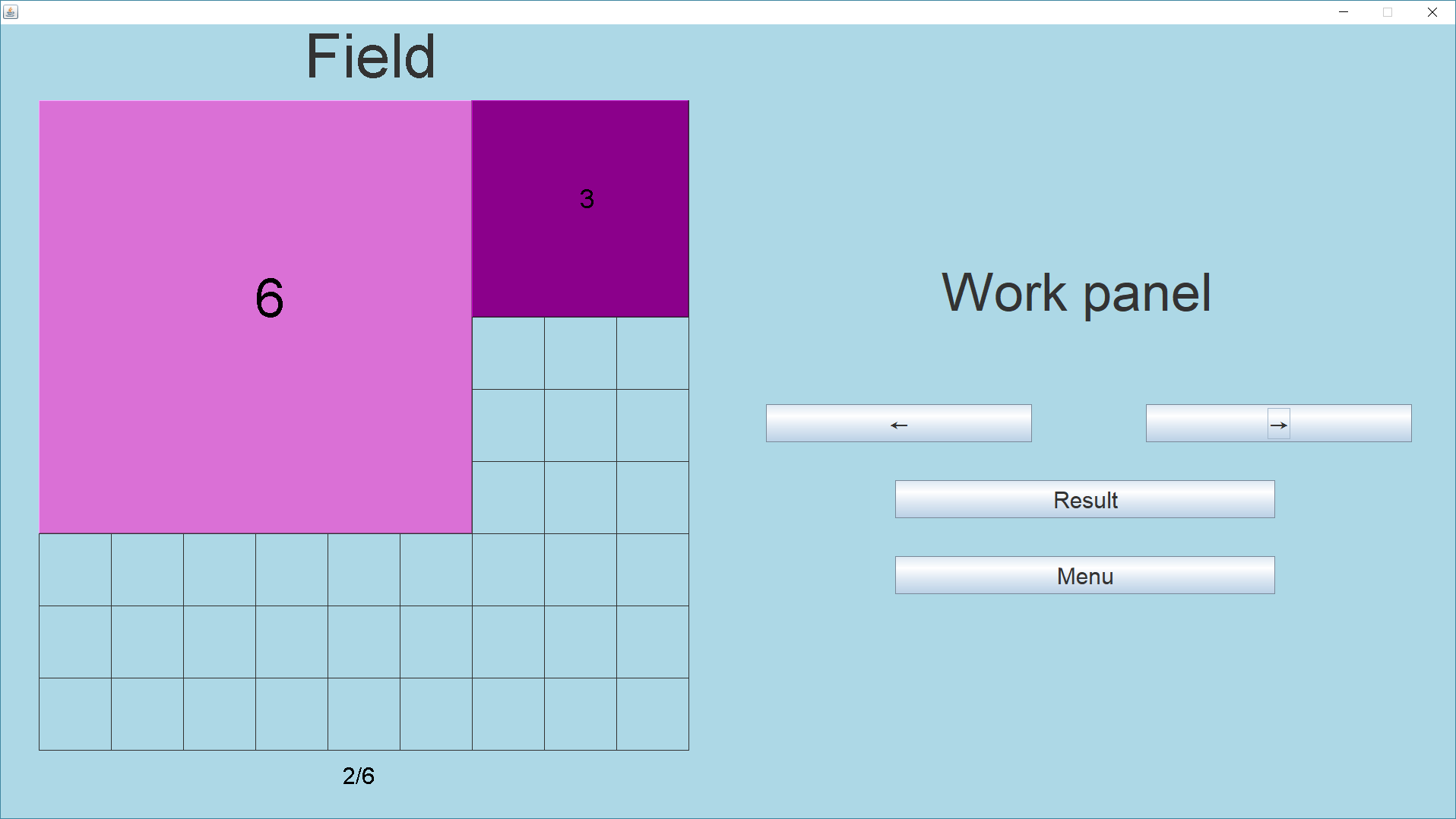


Рисунок 22. Шаг 2 работы алгоритма.

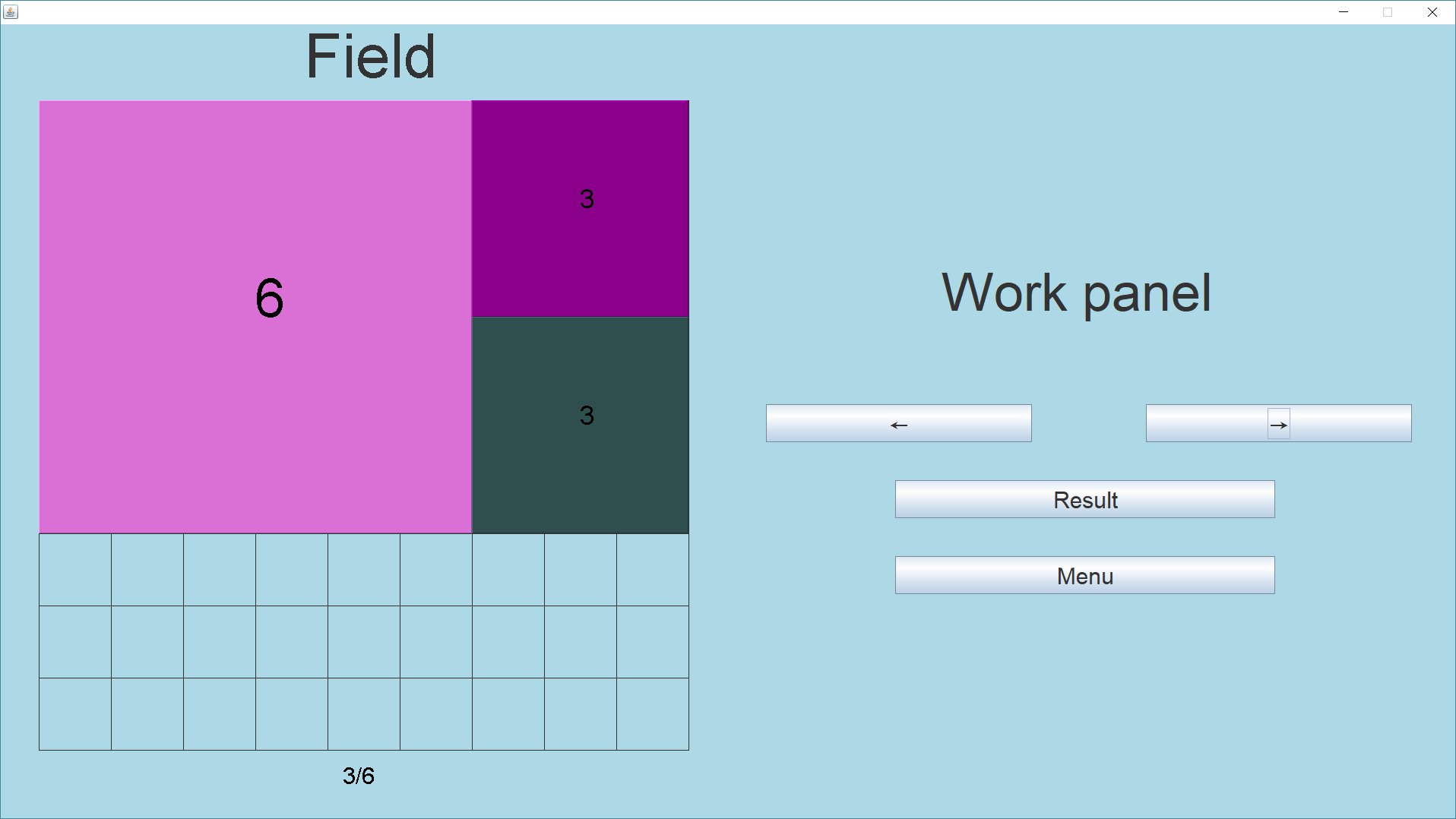


Рисунок 23. Шаг 3 работы алгоритма.

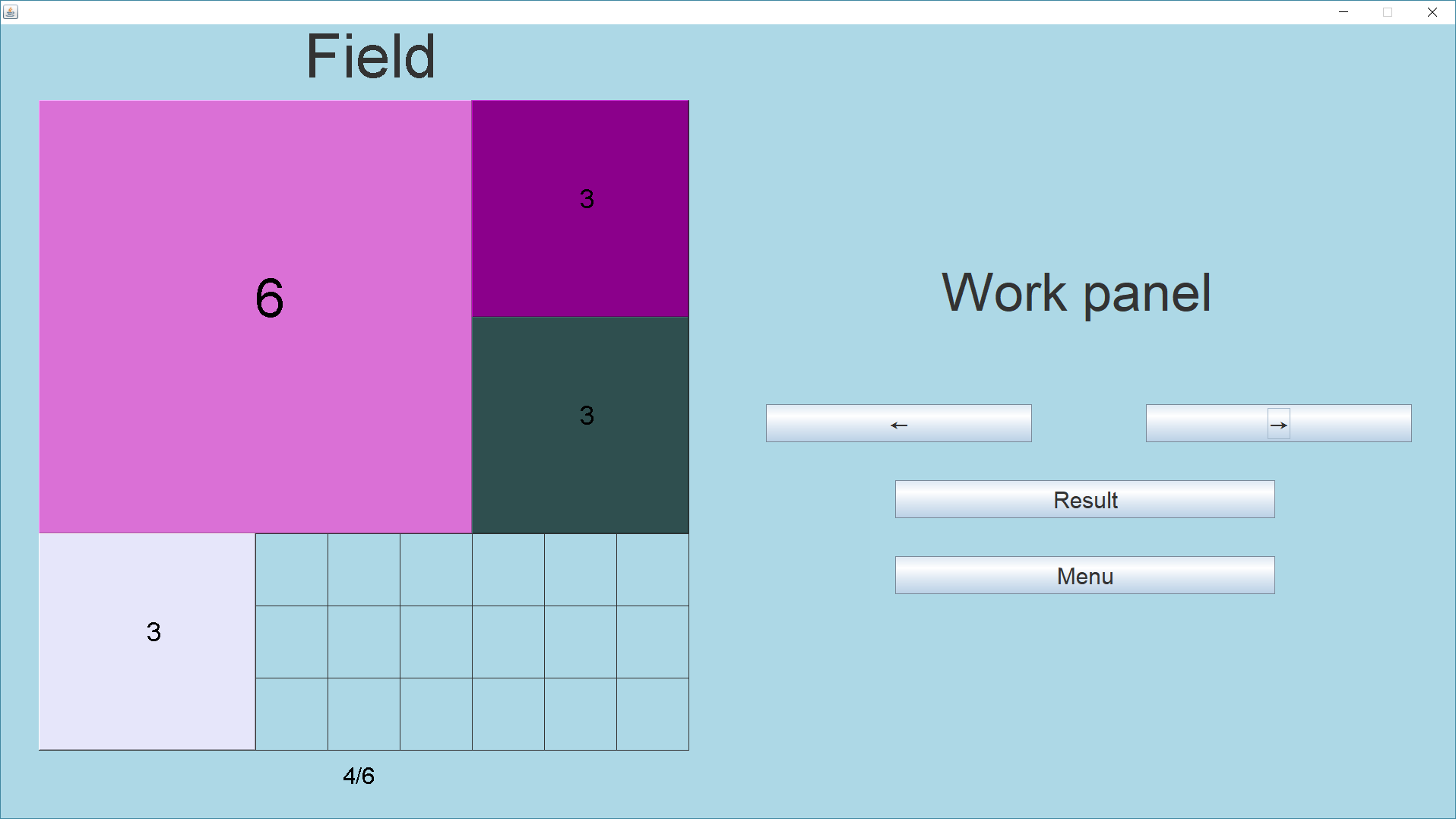


Рисунок 24. Шаг 4 работы алгоритма.

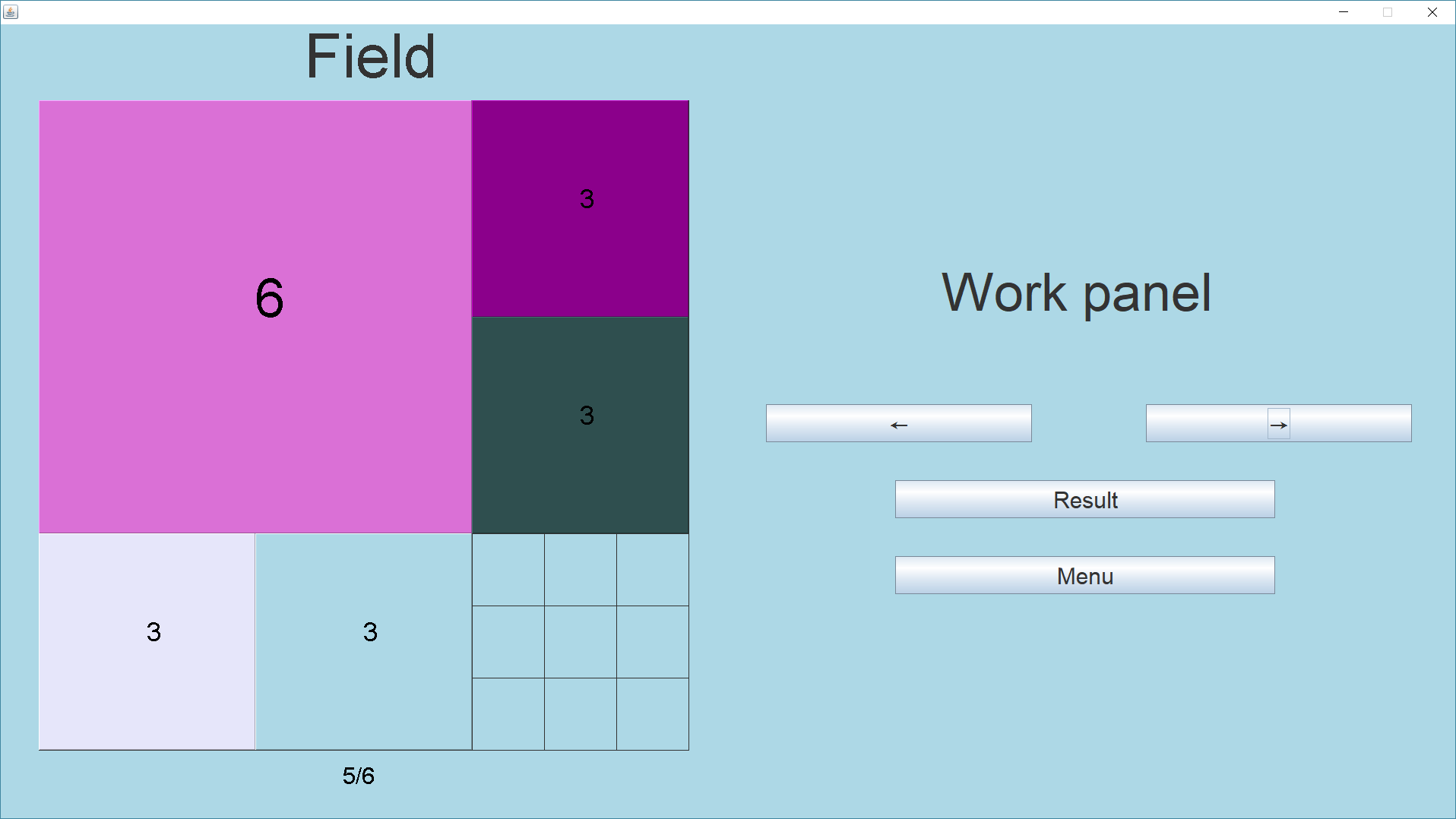


Рисунок 25. Шаг 5 работы алгоритма.

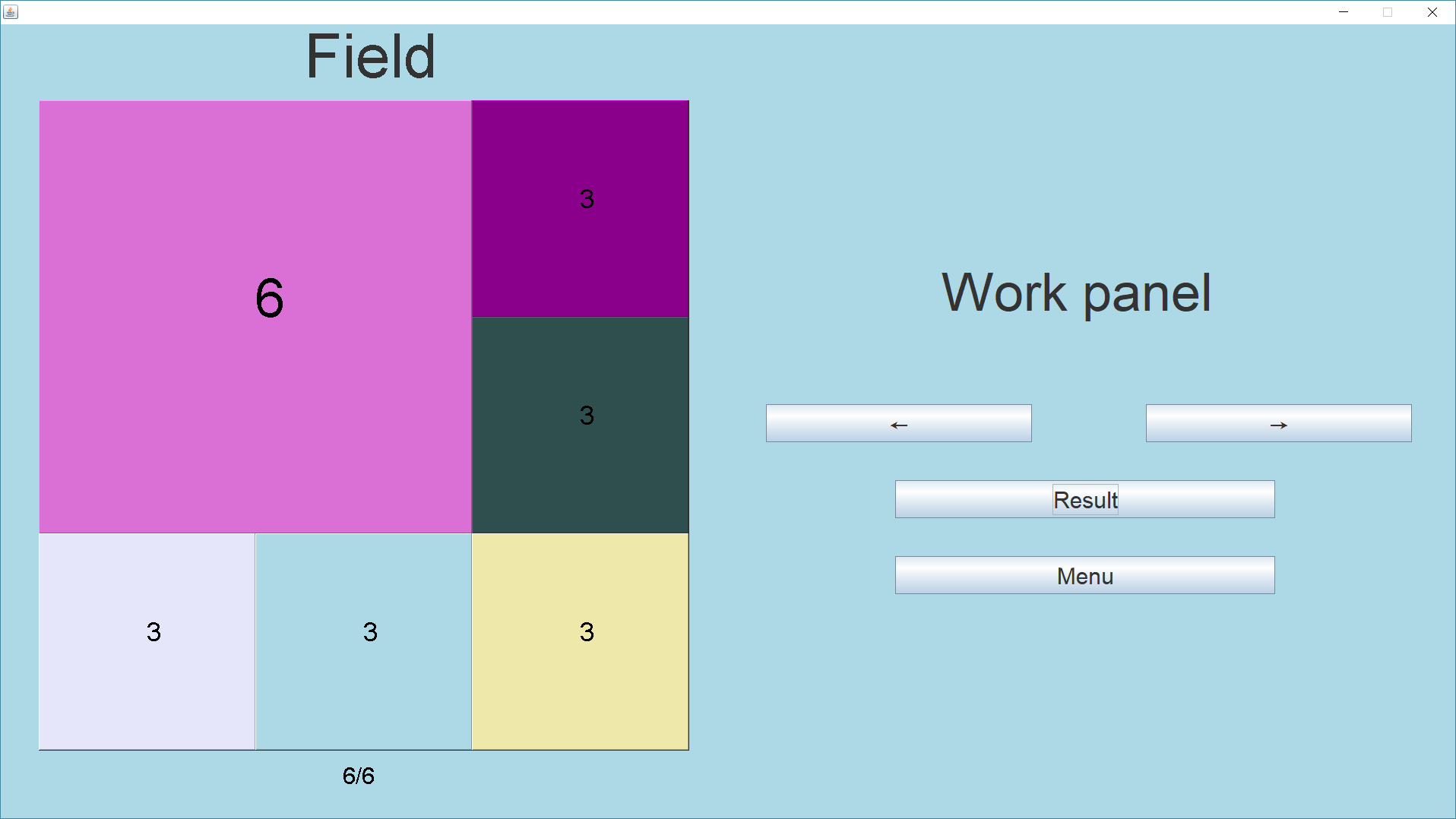


Рисунок 26. Просмотр конечного результата (кнопка «Result»), шаг 6 работы алгоритма.

Также было проведено unit-тестирование со следующими размерами поля: -173, 2, 7, 40, 89. При значениях -173 и 89, выходящих за пределы диапазона [2, 40], программа ловит исключение. Результаты unit- тестирования (рис. 27):

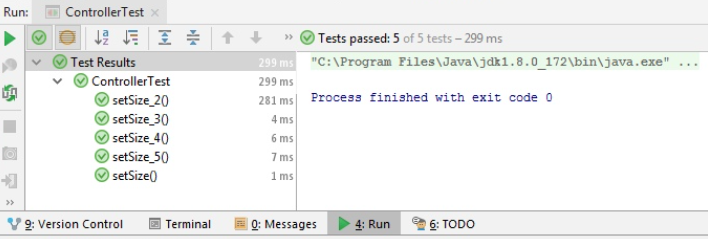


Рисунок 27. Результаты unit- тестирования.

# Заключение

В ходе данной учебной практики была создана программа, которая визуализирует процесс выполнения алгоритма backtracking на примере задачи квадрирования квадрата, используя язык программирования Java.

Программа имеет понятный для пользователя интерфейс и пошагово показывает алгоритм backtracking с помощью простых методов визуализации, а также соответствует важным параметрам парадигмы ООП.

Были получены навыки работы с онлайн-репозиторием на GitHub, а также навыки работы в команде.

# Приложение A. Код програмы.

**Controller. Java**

package com.company.Graphic;  
  
import com.company.Graphic.Model.Field;  
import com.company.Graphic.Model.Model;  
import com.company.Graphic.View.Conditions;  
import com.company.Graphic.View.View;  
  
public class Controller {  
 private View view;  
  
 private Controller(View view) {  
 this.view = view;  
 }  
  
 public static void main(String[] args) {  
 View view = new View();  
 Controller controller = new Controller(view);  
 view.setController(controller);  
 view.init();  
 }  
  
 public void setSize(int size)  
 {  
 Model model = new Model(size);  
 Field field = new Field(size);  
 model.go(field,0,0);  
 Conditions conditions = new Conditions(model.getConditions(), size);  
 view.initWorkPanel(conditions);  
 }  
  
 public void exit()  
 {  
 System.*exit*(0);  
 }  
  
}

**Conditions.java**

package com.company.Graphic.View;  
  
import com.company.Graphic.Model.Square;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.ArrayList;  
  
public class Conditions extends JPanel{  
 private ArrayList<ArrayList<Square>> conditions;  
 private int currentCondition;  
 private int cellSize ;  
 private int size;  
 private int header = 50;  
  
 public Conditions(ArrayList<ArrayList<Square>> conditions, int size) {  
 this.conditions = conditions;  
 this.cellSize = 860/size;  
 this.size = size;  
 this.currentCondition = 0;  
 setBackground(new Color(173, 216, 230));  
 }  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 setLayout(null);  
 Graphics2D graphics2D = (Graphics2D) g;  
 g.setFont(new Font("TimesRoman",Font.*PLAIN*,80));  
 graphics2D.drawString("Field", 400,70);  
 for (int x = 0; x < size; x++) {  
 for (int y = 0; y < size; y++) {  
 graphics2D.drawRect(x \* cellSize+header, y \* cellSize+2\*header, cellSize, cellSize);  
 }  
 }  
  
 for (Square s:conditions.get(currentCondition)  
 ) {  
 if(s.isInField()) {  
 graphics2D.setColor(s.getColor());  
 graphics2D.fill3DRect(s.getyPosition() \* cellSize + header, s.getxPosition() \* cellSize + 2\*header, cellSize \* s.getsize(), cellSize \* s.getsize(), true);  
 graphics2D.setColor(Color.*BLACK*);  
 graphics2D.setFont(new Font("TimesRoman",Font.*PLAIN*,cellSize\*s.getsize()/8));  
 graphics2D.drawString(s.getsize()+"",(int)(s.getyPosition() \* cellSize + cellSize\* s.getsize()/2 + header),(int)(s.getxPosition() \*cellSize + cellSize\* s.getsize()/2 + 2\*header));  
 }  
 }  
 graphics2D.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 graphics2D.drawString(currentCondition+"/"+(conditions.size()-1), 450,1000);  
 }  
  
 public void prev()  
 {  
 if(currentCondition>0)  
 currentCondition--;  
 }  
 public void next()  
 {  
 if(currentCondition<conditions.size()-1)  
 currentCondition++;  
 }  
 public void result()  
 {  
 currentCondition = conditions.size() -1;  
 }  
  
}

**EnterApplication.java**

package com.company.Graphic.View;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
  
public class EnterApplication implements Runnable{  
 private View mainFrame;  
 String text;  
  
 public EnterApplication(View mainFrame) {  
 this.mainFrame = mainFrame;  
 }  
  
 public EnterApplication(View mainFrame, String text) {  
 this.mainFrame = mainFrame;  
 this.text = text;  
 }  
  
 @Override  
 public void run() {  
 mainFrame.getContentPane().setBackground(new Color(173, 216, 230));  
 JLabel label = new JLabel("Enter the size of the square");  
 label.setFont(new Font("TimesRoman",Font.*PLAIN*,50));  
  
 JLabel moreThan = new JLabel("2≤");  
 moreThan.setFont(new Font("TimesRoman",Font.*PLAIN*,40));  
  
 JLabel lessThan = new JLabel("≤40");  
 lessThan.setFont(new Font("TimesRoman",Font.*PLAIN*,40));  
  
  
 JTextField field = new JTextField();  
 field.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 field.setPreferredSize(new Dimension(45,50));  
 field.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyTyped(KeyEvent e) {  
 if (field.getText().length() == 2) {  
 e.consume();  
 }  
 }  
 });  
  
 JButton buttonOk = new JButton("Ok");  
 buttonOk.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonOk.setPreferredSize(new Dimension(500,50));  
 buttonOk.addActionListener(e -> {  
 text = field.getText();  
 int size = 0;  
 try {  
 size = Integer.*parseInt*(text);  
 if (size > 1 && size < 41) {  
 mainFrame.setSize(size);  
 }  
 else {  
 JOptionPane.*showMessageDialog*(mainFrame, "You entered incorrect data. Please, enter the number between 2 and 40.",  
 "Incorrect input", JOptionPane.*PLAIN\_MESSAGE*);  
 field.setText("");  
 }  
 } catch (NumberFormatException e1) {  
 JOptionPane.*showMessageDialog*( mainFrame, "You entered incorrect data. Please, enter the number between 2 and 40.",  
 "Incorrect input", JOptionPane.*PLAIN\_MESSAGE*);  
 field.setText("");  
 }  
 });  
  
 JButton buttonMenu = new JButton("Menu");  
 buttonMenu.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonMenu.setPreferredSize(new Dimension(500,50));  
 buttonMenu.addActionListener(e -> mainFrame.initStartGui());  
  
 GridBagLayout layout = new GridBagLayout();  
 mainFrame.setLayout(layout);  
  
 GridBagConstraints c = new GridBagConstraints();  
 c.anchor = GridBagConstraints.*CENTER*;  
 c.fill = GridBagConstraints.*NONE*;  
 c.gridheight = 1;  
 c.gridwidth = GridBagConstraints.*REMAINDER*;  
 c.gridx = GridBagConstraints.*RELATIVE*;  
 c.gridy = GridBagConstraints.*RELATIVE*;  
 c.insets = new Insets(10, 0, 0, 0);  
  
  
 JPanel sizes = new JPanel();  
 sizes.setBackground(new Color(173, 216, 230));  
 sizes.setLayout(new BoxLayout(sizes, BoxLayout.*X\_AXIS*));  
 sizes.add(moreThan);  
 sizes.add(field);  
 sizes.add(lessThan);  
  
 layout.addLayoutComponent(label,c);  
 mainFrame.getContentPane().add(label);  
  
 c.insets = new Insets(20, 0, 0, 0);  
 layout.addLayoutComponent(sizes,c);  
 mainFrame.add(sizes);  
  
 c.insets = new Insets(40, 0, 0, 0);  
 layout.addLayoutComponent(buttonOk,c);  
 mainFrame.add(buttonOk);  
  
 c.insets = new Insets(20, 0, 0, 0);  
 layout.addLayoutComponent(buttonMenu,c);  
 mainFrame.add(buttonMenu);  
  
 mainFrame.getContentPane().revalidate();  
 mainFrame.getContentPane().repaint();  
 }  
  
  
}

**InformationApplication.java**  
package com.company.Graphic.View;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class InformationApplication implements Runnable {  
 private View mainFrame;  
  
 public InformationApplication(View mainFrame) {  
 this.mainFrame = mainFrame;  
 }  
  
 @Override  
 public void run() {  
  
 mainFrame.getContentPane().setBackground(new Color(173, 216, 230));  
 Label label = new Label("Information about the program");  
 label.setFont(new Font("TimesRoman",Font.*PLAIN*,50));  
  
 Label label1 = new Label("The application solves the problem of squaring a square");  
 label1.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
  
 Label label2 = new Label("into a finite number of smaller squares.\n");  
 label2.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
  
 Label label3 = new Label("The user enters the size of the square and in result gets\n");  
 label3.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
  
 Label label4 = new Label("division of the square on the smallest squares.\n");  
 label4.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
  
 Label label5 = new Label("The number of the squares is minimal.\n");  
 label5.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
  
 JButton buttonMenu = new JButton("Menu");  
 buttonMenu.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonMenu.setPreferredSize(new Dimension(500,50));  
 buttonMenu.addActionListener(e -> mainFrame.initStartGui());  
  
  
 GridBagLayout layout = new GridBagLayout();  
 mainFrame.setLayout(layout);  
  
 GridBagConstraints c = new GridBagConstraints();  
 c.anchor = GridBagConstraints.*NORTH*;  
 c.fill = GridBagConstraints.*NONE*;  
 c.gridheight = 1;  
 c.gridwidth = GridBagConstraints.*REMAINDER*;  
 c.gridx = GridBagConstraints.*RELATIVE*;  
 c.gridy = GridBagConstraints.*RELATIVE*;  
 c.insets = new Insets(10, 0, 0, 0);  
  
  
 layout.addLayoutComponent(label,c);  
 mainFrame.getContentPane().add(label);  
  
  
 c.insets = new Insets(50, 0, 0, 0);  
 layout.addLayoutComponent(label1,c);  
 mainFrame.getContentPane().add(label1);  
  
 c.insets = new Insets(10, 0, 0, 0);  
 layout.addLayoutComponent(label2,c);  
 mainFrame.getContentPane().add(label2);  
  
 c.insets = new Insets(10, 0, 0, 0);  
 layout.addLayoutComponent(label3,c);  
 mainFrame.getContentPane().add(label3);  
  
 c.insets = new Insets(10, 0, 0, 0);  
 layout.addLayoutComponent(label4,c);  
 mainFrame.getContentPane().add(label4);  
  
 c.insets = new Insets(10, 0, 0, 0);  
 layout.addLayoutComponent(label5,c);  
 mainFrame.getContentPane().add(label5);  
  
 c.insets = new Insets(50, 0, 0, 0);  
 layout.addLayoutComponent(buttonMenu,c);  
 mainFrame.add(buttonMenu);  
 mainFrame.getContentPane().revalidate();  
 mainFrame.getContentPane().repaint();  
 }  
}

**StartApplication.java**

package com.company.Graphic.View;  
  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class StartApplication implements Runnable {  
 private View mainFrame;  
  
 public StartApplication(View mainFrame) {  
 this.mainFrame = mainFrame;  
 }  
  
 @Override  
 public void run() {  
 mainFrame.getContentPane().setBackground(new Color(173, 216, 230));  
 JButton buttonEnter = new JButton("Enter the size");  
 buttonEnter.setPreferredSize(new Dimension(500,50));  
 buttonEnter.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonEnter.addActionListener(e -> mainFrame.initEnterGui());  
  
 JButton buttonExit = new JButton("Exit");  
 buttonExit.setPreferredSize(new Dimension(500,50));  
 buttonExit.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonExit.addActionListener(e -> mainFrame.exit());  
  
 JButton buttonInformation = new JButton("About the program");  
 buttonInformation.setPreferredSize(new Dimension(500,50));  
 buttonInformation.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonInformation.addActionListener(e -> mainFrame.initInformationGui());  
  
 JLabel label = new JLabel("Menu");  
 label.setFont(new Font("TimesRoman",Font.*PLAIN*,50));  
  
 GridBagLayout layout = new GridBagLayout();  
 mainFrame.setLayout(layout);  
  
 GridBagConstraints c = new GridBagConstraints();  
 c.anchor = GridBagConstraints.*NORTH*;  
 c.fill = GridBagConstraints.*NONE*;  
 c.gridheight = 1;  
 c.gridwidth = GridBagConstraints.*REMAINDER*;  
 c.gridx = GridBagConstraints.*RELATIVE*;  
 c.gridy = GridBagConstraints.*RELATIVE*;  
 c.insets = new Insets(20, 0, 0, 0);  
 c.ipadx = 0;  
 c.ipady = 0;  
 c.weightx = 0;  
 c.weighty = 0;  
  
 layout.addLayoutComponent(label,c);  
 mainFrame.getContentPane().add(label);  
  
 c.insets = new Insets(50, 0, 0, 0);  
 layout.setConstraints(buttonEnter,c);  
 mainFrame.getContentPane().add(buttonEnter);  
  
 c.insets = new Insets(20, 0, 0, 0);  
 layout.setConstraints(buttonInformation,c);  
 mainFrame.getContentPane().add(buttonInformation);  
  
 c.insets = new Insets(20, 0, 0, 0);  
 layout.setConstraints(buttonExit,c);  
 mainFrame.getContentPane().add(buttonExit);  
 mainFrame.getContentPane().revalidate();  
 mainFrame.getContentPane().repaint();  
 }  
  
}

**View.java**

package com.company.Graphic.View;  
import com.company.Graphic.Controller;  
import com.company.Graphic.Model.Square;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.ArrayList;  
  
public class View extends JFrame implements ActionListener {  
  
 private Controller controller;  
  
 public void setController(Controller controller) {  
 this.controller = controller;  
 getContentPane().setBackground(new Color(173, 216, 230));  
 }  
  
 public void init() {  
 initStartGui();  
 setVisible(true);  
 setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*);  
 setPreferredSize(Toolkit.*getDefaultToolkit*().getScreenSize());  
 setResizable(false);  
 pack();  
 setLocationRelativeTo(null);  
 }  
  
  
 public void initStartGui() {  
 getContentPane().removeAll();  
 SwingUtilities.*invokeLater*(new StartApplication(this));  
  
 }  
  
 public void initEnterGui()  
 {  
 getContentPane().removeAll();  
 SwingUtilities.*invokeLater*(new EnterApplication(this));  
 }  
  
 public void initInformationGui()  
 {  
 getContentPane().removeAll();  
 SwingUtilities.*invokeLater*(new InformationApplication(this));  
 }  
  
 public void initWorkPanel(Conditions field)  
 {  
 getContentPane().removeAll();  
 SwingUtilities.*invokeLater*(new WorkApplication(this, field));  
 }  
  
 public void setSize(int size)  
 {  
 controller.setSize(size);  
 }  
  
 public void exit()  
 {  
 controller.exit();  
 }  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 }  
  
}

**WorkApplication.java**

package com.company.Graphic.View;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
  
public class WorkApplication implements Runnable {  
 private View mainFrame;  
 private Conditions field;  
  
 WorkApplication(View mainFrame, Conditions field) {  
 this.mainFrame = mainFrame;  
 this.field = field;  
 }  
  
 @Override  
 public void run() {  
 JPanel rootPanel = new JPanel();  
 rootPanel.setLayout(new BoxLayout(rootPanel, BoxLayout.*X\_AXIS*));  
 rootPanel.add(field);  
 rootPanel.add(createWorkPanel());  
 mainFrame.setContentPane(rootPanel);  
  
 mainFrame.revalidate();  
 mainFrame.repaint();  
  
 }  
  
 private JPanel createWorkPanel()  
 {  
 JPanel workPanel = new JPanel();  
 workPanel.setLayout(null);  
  
 JLabel text = new JLabel("Work panel");  
 text.setFont(new Font("TimesRoman",Font.*PLAIN*,70));  
 text.setBounds(280, 300, 600, 100);  
  
 JButton buttonNext = new JButton("→");  
 buttonNext.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonNext.setBounds(550,500,350,50);  
 buttonNext.addActionListener(e -> {  
 field.next();  
 field.revalidate();  
 field.repaint();  
 });  
 workPanel.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent e) {  
 if (e.getKeyCode()== KeyEvent.*VK\_RIGHT*) {  
 buttonNext.doClick();  
 }  
 }  
 });  
  
 JButton buttonPrev = new JButton("←");  
 buttonPrev.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonPrev.setBounds(50,500,350,50);  
 buttonPrev.addActionListener(e -> {  
 field.prev();  
 field.revalidate();  
 field.repaint();  
 });  
 workPanel.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent e) {  
 //super.keyPressed(e);  
 if (e.getKeyCode()==KeyEvent.*VK\_LEFT*) {  
 buttonPrev.doClick();  
 }  
 }  
 });  
  
 JButton buttonResult = new JButton("Result");  
 buttonResult.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonResult.setBounds(220,600,500,50);  
 buttonResult.addActionListener(e -> {  
 field.result();  
 field.revalidate();  
 field.repaint();  
 });  
  
  
 JButton buttonMenu = new JButton("Menu");  
 buttonMenu.setFont(new Font("TimesRoman",Font.*PLAIN*,30));  
 buttonMenu.setBounds(220,700,500,50);  
 buttonMenu.addActionListener(e -> mainFrame.initStartGui());  
  
 workPanel.setBackground(new Color(173, 216, 230));  
 workPanel.add(text);  
 workPanel.add(buttonNext);  
 workPanel.add(buttonPrev);  
 workPanel.add(buttonResult);  
 workPanel.add(buttonMenu);  
  
  
 return workPanel;  
  
 }  
}

**Field.java**

package com.company.Graphic.Model;  
  
  
public class Field {  
 private int [][] field;  
 private int size;  
  
 public Field(int size) {  
 this.size = size;  
 field = new int [size][size];  
 }  
  
 public boolean isPossible(int i, int j, Square square)  
 {  
 if ((i + square.getsize()) > size || (j + square.getsize()) > size)  
 return false;  
 for (int l = i; l < i + square.getsize(); l++)  
 for (int m = j; m < j + square.getsize(); m++)  
 if (field[l][m] != 0)  
 return false;  
 return true;  
 }  
  
 public void insertSquare(Square square, int x, int y)  
 {  
 for (int i = x; i < x+square.getsize(); i++)  
 {  
 for (int j = y; j < y+square.getsize() ; j++)  
 {  
 field[i][j] = square.getsize();  
  
 }  
 }  
 square.setxPosition(x);  
 square.setyPosition(y);  
  
  
 }  
  
 public boolean isFreeCell(int i, int j)  
 {  
 for (int l = i; l < size; l++)  
 for (int m = 0; m < size; m++)  
 if (field[l][m] == 0)  
 return true;  
 return false;  
 }  
  
 public int getNextI(int i, int j)  
 {  
 for (int l = i; l < size; l++)  
 for (int m = 0; m < size; m++)  
 if (field[l][m] == 0)  
 return l;  
 return -1;  
 }  
  
 public int getNextJ(int i, int j)  
 {  
 for (int l = i; l < size; l++)  
 for (int m = 0; m < size; m++)  
 if (field[l][m] == 0)  
 return m;  
 return -1;  
 }  
  
 private void setField(int[][] field) {  
 for (int i = 0; i < field.length; i++) {  
 for (int j = 0; j < field[i].length; j++) {  
 this.field[i][j] = field[i][j];  
 }  
 }  
 }  
  
 public Field makeCopy()  
 {  
 Field field1 = new Field(size);  
 field1.setField(field);  
 return field1;  
 }  
}

**Model.java**

package com.company.Graphic.Model;  
  
import java.util.ArrayList;  
  
public class Model {  
 private ArrayList<Square> squares;  
 private ArrayList<ArrayList<Square>> conditions = new ArrayList<>();  
 private boolean wasFound = false;  
  
  
 public Model(int size)  
 {  
 Number num = new Number(size);  
 squares = num.getSquares();  
 addNewCondition(squares);  
 }  
  
 public ArrayList<Square> getSquares() {  
 return squares;  
 }  
  
 public void go(Field table, int i, int j) {  
 if (!wasFound) {  
 for (int k = 0; k < squares.size(); k++) {  
 Field condition = table.makeCopy();  
 Square tmp = squares.get(k);  
 if (!tmp.isInField() && !wasFound)  
 {  
 if(condition.isPossible(i,j,tmp))  
 {  
 condition.insertSquare(tmp, i, j);  
 tmp.setInField(true);  
 addNewCondition(squares);  
 if (condition.isFreeCell(i, j)) {  
 go(condition, condition.getNextI(i, j), condition.getNextJ(i, j));//go to free cell  
 } else {  
 wasFound = true; //to exit the recursive function  
 break;  
 }  
 tmp.setInField(false);  
 if (!wasFound) {  
 addNewCondition(squares);  
 }  
 }  
  
 k = k + squares.get(k).getAmount() - 1;  
 }  
 }  
 }  
 }  
  
 private void addNewCondition(ArrayList<Square> squares)  
 {  
 ArrayList<Square> list = new ArrayList<>();  
 for (Square s: squares  
 ) {  
 list.add(s.makeCopy());  
 }  
 conditions.add(list);  
 }  
  
 public ArrayList<ArrayList<Square>> getConditions() {  
 return conditions;  
 }  
}

**Number.java**

package com.company.Graphic.Model;  
import java.util.ArrayList;  
import java.awt.Color;  
  
public class Number {  
 private int scale;  
 private int[] squares;  
 private int[] numberOfSquares;  
  
 Number(int number) {  
 int primeDivisor = findPrimeDivisor(number);  
 this.scale = number / findPrimeDivisor(number);  
 PrimeNumbers primeNumbers = new PrimeNumbers();  
 this.squares = primeNumbers.getSquares(primeDivisor);  
 this.numberOfSquares = primeNumbers.getNumberOfSquares(primeDivisor);  
 }  
 private int findPrimeDivisor(int number)  
 {  
 int[] primeNumbers = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};  
 for (int i = 0; i < primeNumbers.length; i++)  
 if (number%primeNumbers[i] == 0)  
 return primeNumbers[i];  
 return -1;  
 }  
  
 public ArrayList<Square> getSquares(){  
 ArrayList<Square> result = new ArrayList<Square>();  
 for(int i = squares.length - 1 ; i >= 0 ; i--)  
 {  
 int n = numberOfSquares[i];  
 for(int j = numberOfSquares[i] - 1 ; j >= 0 ; j--)  
 {  
 Square new\_square = new Square(squares[i] \* scale, n);  
 result.add(new\_square);  
 n--;  
 }  
 }  
 return result;  
 }  
}

**PrimeNumbers.java**

package com.company.Graphic.Model;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public final class PrimeNumbers {  
 private Map<Integer,int []> numberOfSquares = new HashMap<>();  
 private Map<Integer, int []> squares = new HashMap<>();  
  
 PrimeNumbers()  
 {  
 squares.put(2,new int[]{1});  
 squares.put(3,new int[]{ 1 , 2 });  
 squares.put(5,new int[]{ 1, 2 ,3 });  
 squares.put(7,new int[]{ 1, 2, 3, 4 });  
 squares.put(11,new int[]{ 1, 2, 3, 5, 6 });  
 squares.put(13,new int[]{ 1, 2, 3, 4, 6, 7 });  
 squares.put(17,new int[]{ 1, 2, 3, 4, 5, 8, 9 });  
 squares.put(19,new int[]{ 1, 3, 4, 5, 6, 9, 10 });  
 squares.put(23,new int[]{ 1, 2, 3, 4, 6, 7, 10, 13 });  
 squares.put(29,new int[]{ 1, 3, 4, 5, 6, 7, 13, 16 });  
 squares.put(31,new int[]{ 1, 2, 4, 8, 15, 16 });  
 squares.put(37,new int[]{ 2, 3, 4, 5, 9, 11, 17, 20 });  
 squares.put(41, new int[]{1,2,3,4,5,7,11,12,18,23});  
 squares.put(43, new int[]{1,4,5,6,7,11,13,19,24});  
 squares.put(47, new int[]{1,3,6,7,8,9,10,22,25});  
 squares.put(53, new int[]{1,3,4,5,6,7,13,16,24,29});  
 squares.put(59, new int[]{1,2,4,6,8,10,11,12,14,16,18,19,29,30});  
 squares.put(61, new int[]{3,4,6,8,9,11,15,17,29,32});  
  
 numberOfSquares.put(2,new int[]{4});  
 numberOfSquares.put(3,new int[]{ 5, 1 });  
 numberOfSquares.put(5,new int[]{ 4, 3, 1 });  
 numberOfSquares.put(7,new int[]{ 3, 3, 2, 1 });  
 numberOfSquares.put(11,new int[]{ 4, 1, 3, 2, 1 });  
 numberOfSquares.put(13,new int[]{ 2, 3, 2, 1, 2, 1 });  
 numberOfSquares.put(17,new int[]{ 2, 3, 1, 2, 1, 2, 1 });  
 numberOfSquares.put(19,new int[]{ 1, 4, 2, 3, 1, 1, 1 });  
 numberOfSquares.put(23,new int[]{ 2, 3, 1, 1, 2, 1, 2, 1 });  
 numberOfSquares.put(29,new int[]{ 1, 2, 2, 3, 2, 1, 2, 1 });  
 numberOfSquares.put(31,new int[]{ 3, 3, 3, 3, 2, 1 });  
 numberOfSquares.put(37,new int[]{ 2, 3, 3, 1, 2, 1, 2, 1 });  
 numberOfSquares.put(41, new int[]{2,2,2,1,1,1,2,1,2,1});  
 numberOfSquares.put(43, new int[]{2,4,1,2,2,1,1,2,1});  
 numberOfSquares.put(47, new int[]{1,3,1,2,3,2,1,2,1});  
 numberOfSquares.put(53, new int[]{1,2,2,2,2,1,2,1,2,1});  
 numberOfSquares.put(59, new int[]{2,1,2,1,1,2,1,1,1,1,1,1,1,1});  
 numberOfSquares.put(61,new int[]{2,4,2,1,2,1,1,1,2,1});  
  
 }  
  
 public int[] getSquares(int primeNumber)  
 {  
 return squares.get(primeNumber);  
 }  
  
 public int[] getNumberOfSquares(int primeNumber)  
 {  
 return numberOfSquares.get(primeNumber);  
 }  
  
  
}

**Square.java**

package com.company.Graphic.Model;  
  
import java.awt.Color;  
import java.util.Random;  
  
public class Square {  
 private int size;  
 private int xPosition;  
 private int yPosition;  
 private boolean inField;  
 private int amount;  
 private Colors color;  
  
 public enum Colors  
 {  
 *THISTLE*,  
 *ORCHID*,  
 *MEDIUMPURPLE*,  
 *LIGHTSTEELBLUE*,  
 *DARKMAGENTA*,  
 *PALETURQUOISE*,  
 *LIGHTSLATEGREY*,  
 *PEACHPUFF*,  
 *LAVENDER*,  
 *DARKSLATEGREY*,  
 *LIGHTBLUE*,  
 *PALEGOLDENROD*,  
 *PALEGREEN*,  
 *DARKSEAGREEN*;  
 };  
  
 private void generateColor()  
 {  
 int r = new Random().nextInt(Colors.*values*().length);  
 color = Colors.*values*()[r];  
 }  
  
 public Color getColor()  
 {  
 switch (color)  
 {  
 case *ORCHID*:  
 return new Color(218, 112, 214);  
 case *THISTLE*:  
 return new Color(216, 191, 216);  
 case *MEDIUMPURPLE*:  
 return new Color(147, 112, 219);  
 case *LIGHTSTEELBLUE*:  
 return new Color(176, 196, 222);  
 case *DARKMAGENTA*:  
 return new Color(139, 0, 139);  
 case *PALETURQUOISE*:  
 return new Color(175, 238, 238);  
 case *LIGHTSLATEGREY*:  
 return new Color(119, 136, 153);  
 case *PEACHPUFF*:  
 return new Color(255, 218, 185);  
 case *LAVENDER*:  
 return new Color(230, 230, 250);  
 case *DARKSLATEGREY*:  
 return new Color(47, 79, 79);  
 case *LIGHTBLUE*:  
 return new Color(173, 216, 230);  
 case *PALEGOLDENROD*:  
 return new Color(238, 232, 170);  
 case *PALEGREEN*:  
 return new Color(152, 251, 152);  
 case *DARKSEAGREEN*:  
 return new Color(143, 188, 143);  
 }  
 return null;  
 }  
 Square(int size, int n)  
 {  
 this.size = size;  
 this.amount = n;  
 generateColor();  
 }  
  
  
 public int getxPosition() {  
 return xPosition;  
 }  
  
 public void setxPosition(int xPosition) {  
 this.xPosition = xPosition;  
 }  
  
 public int getyPosition() {  
 return yPosition;  
 }  
  
 public void setyPosition(int yPosition) {  
 this.yPosition = yPosition;  
 }  
  
 public void setInField(boolean i){ this.inField = i; }  
  
 public boolean isInField() {  
 return inField;  
 }  
  
 public int getsize() {  
 return size;  
 }  
  
 public int getAmount() { return amount; }  
  
 private void setColor(Colors color) {  
 this.color = color;  
 }  
  
 public Square makeCopy()  
 {  
 Square s = new Square(size, amount);  
 s.setxPosition(xPosition);  
 s.setyPosition(yPosition);  
 s.setInField(inField);  
 s.setColor(color);  
 return s;  
 }  
}

# Приложение Б. Код unit-тестов.

package com.company.Graphic;

import com.company.Graphic.Model.Field;

import com.company.Graphic.Model.Model;

import com.company.Graphic.Model.Square;

import org.junit.jupiter.api.Test;

import java.io.IOException;

import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.\*;

class ControllerTest {

@Test

void setSize() {

Throwable exception = assertThrows(NullPointerException.class, () -> {

new Model(-173);

});

assertNull(exception.getMessage());

}

@Test

void setSize\_2() {

Model model = new Model(7);

Field field = new Field(model,7);

model.go(field,0,0);

ArrayList<ArrayList<Square>> conditions = model.getConditions();

double area = 0;

for (Square s: conditions.get(0)

) {

assertEquals(s.isInField(),false);

area += Math.pow(s.getsize(),2);

}

assertEquals(49, area, 0.000001);

for (Square s: conditions.get(conditions.size()-1)

) {

assertEquals(s.isInField(),true);

}

}

@Test

void setSize\_3(){

Model model2 = new Model(2);

Field field2 = new Field(model2,2);

model2.go(field2,0,0);

ArrayList<ArrayList<Square>> conditions2 = model2.getConditions();

double area2 = 0;

assertEquals(5, conditions2.size());

for (Square s: conditions2.get(0)

) {

assertEquals(s.isInField(),false);

area2 += Math.pow(s.getsize(),2);

}

assertEquals(4, area2, 0.000001);

for (Square s: conditions2.get(conditions2.size()-1)

) {

assertEquals(s.isInField(),true);

}

}

@Test

void setSize\_4(){

Model model3 = new Model(40);

Field field3 = new Field(model3,40);

model3.go(field3,0,0);

ArrayList<ArrayList<Square>> conditions3 = model3.getConditions();

double area3 = 0;

assertEquals(5, conditions3.size());

for (Square s: conditions3.get(0)

) {

assertEquals(s.isInField(),false);

area3 += Math.pow(s.getsize(),2);

}

assertEquals(1600, area3, 0.000001);

for (Square s: conditions3.get(conditions3.size()-1)

) {

assertEquals(s.isInField(),true);

}

}

@Test

void setSize\_5() {

Throwable exception = assertThrows(NullPointerException.class, () -> {

new Model(89);

});

assertNull(exception.getMessage());

}

}