МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

КАЛИНИНГРАДСКОЙ ОБЛАСТИ

ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ

«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СТРОИТЕЛЬСТВА»

(ГБУ КО ПОО «КИТиС»)

|  |  |  |
| --- | --- | --- |
| Курсовой проект  допущен к защите  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата)  Зам.директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_\_Павленко Г.Я.  (подпись) |  | Курсовой проект  защищен с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Пояснительная записка к курсовому проекту

по дисциплине: МДК 01.01 Разработка программных модулей

Тема: Программная реализация графических алгоритмов с использованием среды программирования Intellij IDEA

Специальность 09.02.07 «Информационные системы и программирование»

|  |  |
| --- | --- |
| Выполнил: | студент 3 курса,  группы ИСп20-1  Даховник Анна Максимовна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель: | Большакова-Стрекалова Анна Викторовна, преподаватель ГБУ КО ПОО КИТиС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Калининград

2023

Аннотация

Курсовой проект на тему: Программная реализация графических алгоритмов с использованием среды программирования Intellij IDEA.

Отчет по курсовому проекту включает в себя: введение, три основные главы, заключение, список использованной литературы, приложение.

Во введении кратно обосновывается актуальность данной темы, цель и содержание поставленных задач.

В первой главе «Описание предметной области» рассматриваются аналоги разрабатываемого приложения, описывается техническое задание и структура данного приложения.

Вторая глава «Описание разработки приложения» включает в себя:

* Обоснования средств разработки, в которой указывается выбранная среда программирования и язык для приложения;
* Разработка интерфейса;
* Разработка логики работы приложения с схемой взаимодействия компонентов проекта.

Третья глава «Тестирование и установка приложения» содержит информацию о методе тестирования приложения, результат тестирования и установку приложения.

В заключении приведены основные выводы, полученные в результате работы курсового проекта.

Список использованной литературы насчитывает 7 ссылок на источники.

В приложениях содержатся листинги рабочего кода и руководство пользователя.

Ссылка на GitHub- https://github.com/dahovnikanna/graphics1.git

Содержание

[Введение 4](#_Toc134378577)

[1 Описание предметной области 5](#_Toc134378578)

[1.1 Аналоги разрабатываемого приложения 5](#_Toc134378579)

[1.2 Техническое задание 5](#_Toc134378580)

[1.3 Описание структуры приложения 5](#_Toc134378581)

[2 Описание разработки приложения 7](#_Toc134378582)

[2.1 Обоснование средств разработки 7](#_Toc134378583)

[2.2 Разработка интерфейса 8](#_Toc134378584)

[2.3 Разработка логики работы приложения. Схема взаимодействия компонентов проекта 14](#_Toc134378585)

[2.4 Описание переменных, компонентов, классов и подпрограмм 15](#_Toc134378586)

[3 Тестирование и установка приложения 17](#_Toc134378587)

[Заключение 19](#_Toc134378588)

[Список использованной литературы 20](#_Toc134378589)

[Приложение А. Листинг рабочего кода 21](#_Toc134378590)

[Приложение Б. Руководство пользователя. 30](#_Toc134378591)

# Введение

Современный мир тесно связан с графикой и визуализацией. Графические алгоритмы используются во многих областях, таких как игровая индустрия, медицинская визуализация, архитектурное проектирование и многие другие. Одним из ключевых факторов при работе с графическими алгоритмами является удобная и эффективная среда программирования.

Цель проекта:

Целью данного курсового проекта является программная реализация графических алгоритмов с использованием среды программирования IntelliJ IDEA.

Задачи проекта:

* Изучение графических алгоритмов и их применения;
* Изучение среды программирования IntelliJ IDEA и ее возможностей для работы с графикой;
* Разработка программы для реализации графических алгоритмов;

Актуальность проекта:

Современные технологии непрерывно развиваются, и графические алгоритмы играют все более важную роль в различных областях. Поэтому, разработка программного обеспечения для реализации графических алгоритмов с использованием среды программирования IntelliJ IDEA имеет большую актуальность. Это позволит упростить и ускорить процесс создания и отладки графических приложений, а также повысить качество их работы.

# 1 Описание предметной области

* 1. Аналоги разрабатываемого приложения

Проект является учебным. Не было найдено ни одного аналога данного приложения, в силу низкого спроса на приложения такого типа.

* 1. Техническое задание

Требуется создать программу, которая будет показывать, как визуализируются графические алгоритмы.

Используется три вида графики: растровая, векторная, объёмная.

Требования к программе:

* Программа должна иметь графический интерфейс пользователя;
* Программа должна позволять работать с растровой, векторной и объемной графикой;
* Программа должна позволять создавать и сохранять изображения;
* Программа должна быть реализована на языке Java с использованием среды программирования IntelliJ IDEA.
  1. Описание структуры приложения

Для программной реализации графических алгоритмов с использованием среды программирования Intelli IDEA и языка Java можно использовать следующую структуру приложения:

* Главный класс приложения, который содержит метод main и отвечает за запуск всего приложения.
* Классы для работы с каждым типом графики: растровой, векторной и объемной: VectorDice.java, RastrDice.java, Volum.java;
* Класс для работы с файлами и сохранения графических данных в нужном формате: RastrDice.java;
* Классы для работы с пользовательским интерфейсом: UA.java, Picture.java;
* Для обработки событий нажатия кнопок и выбора пунктов меню используется интерфейс слушателя ActionListener;
* Для создания и отрисовки элементов пользовательского интерфейса, отображения графики на экране используются классы из библиотеки JavaSwing, AWT;
* Для создания объёмной графики используется библиотека JavaFX.[3][5]

Таким образом, структура приложения может быть достаточно сложной и включать в себя множество классов, методов и библиотек. Однако, правильно спроектированное приложение позволит эффективно работать с графикой и обеспечит удобный пользовательский интерфейс для работы с программой.

# Описание разработки приложения

## Обоснование средств разработки

Для программной реализации графических алгоритмов на языке Java с использованием трех видов графики (растровой, векторной и объемной) подходит среда разработки IntelliJ IDEA.

Обоснование выбора данной среды разработки следующее:

1. Поддержка Java: IntelliJ IDEA является одним из лучших инструментов для разработки на Java, обеспечивая высокую производительность и возможность отладки Java-приложений.
2. Широкие возможности по работе с графикой: IntelliJ IDEA поддерживает большое количество библиотек и инструментов для работы с графикой на Java, включая JavaFX, OpenGL, AWT и Swing.
3. Разнообразие функций и плагинов: IntelliJ IDEA предоставляет широкий набор функций, включая интегрированную систему контроля версий, инструменты автоматического тестирования и отладки, а также возможность подключения плагинов для расширения функциональности.
4. Легкость использования: IntelliJ IDEA имеет простой и интуитивно понятный пользовательский интерфейс, что облегчает работу с средой разработки и сокращает время на ее изучение.
5. Большое сообщество разработчиков: IntelliJ IDEA имеет большое сообщество пользователей и разработчиков, которые помогают друг другу решать проблемы и делятся опытом в использовании среды разработки.

В результате, использование среды разработки IntelliJ IDEA позволяет эффективно и быстро создавать графические приложения на языке Java с использованием различных видов графики и графических алгоритмов.

## 2.2 Разработка интерфейса

Графический интерфейс служит для упрощения работы пользователя.

Для создания пользовательского интерфейса в данной работе используется класс JFrame.[1]

На главном экране можно определить кнопки, текстовое, которые будут отображаться на экране устройства. [10]

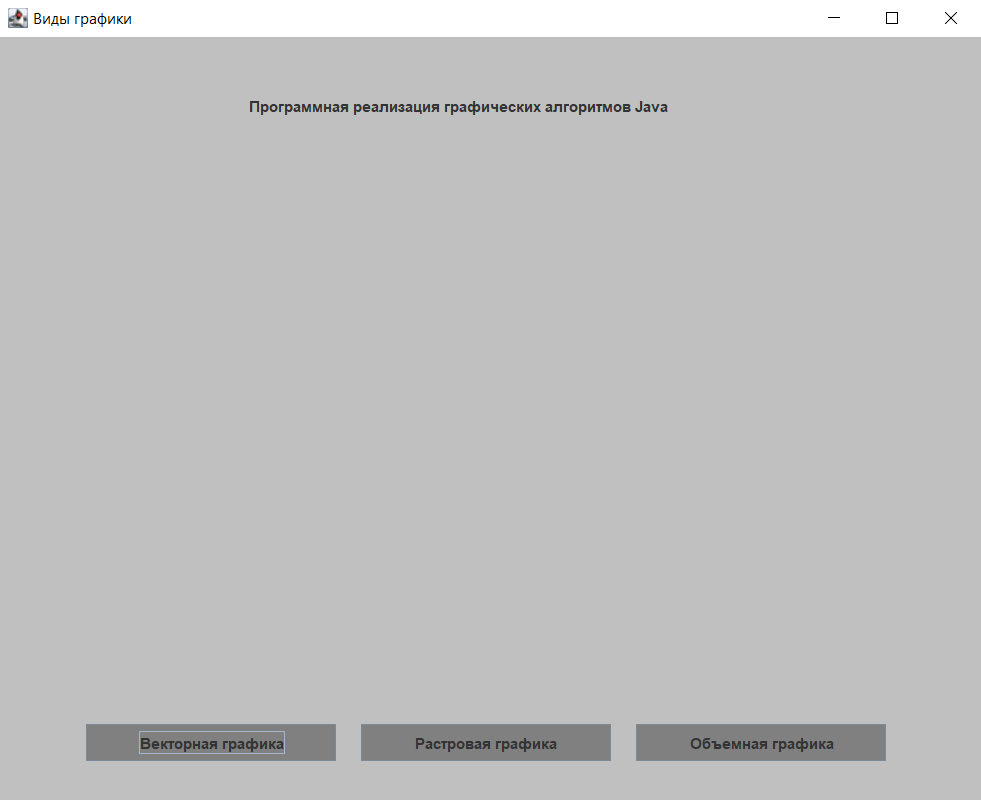
Дизайн данной программы состоит из 4 окон. Макет первого окна представлен на рисунке 1. Данное окно представляет с собой основное окно программы. Макет создан с помощью JFrame на котором располагаются JLable[6] и JButtom.[2]

Рисунок 1 – Макет главного окна программы

При нажатии на кнопку «Векторная графика» на основном окне появляется дополнительное окно, на котором изображена игральная кость, нарисованная с помощью векторной графики. (Рис.2)

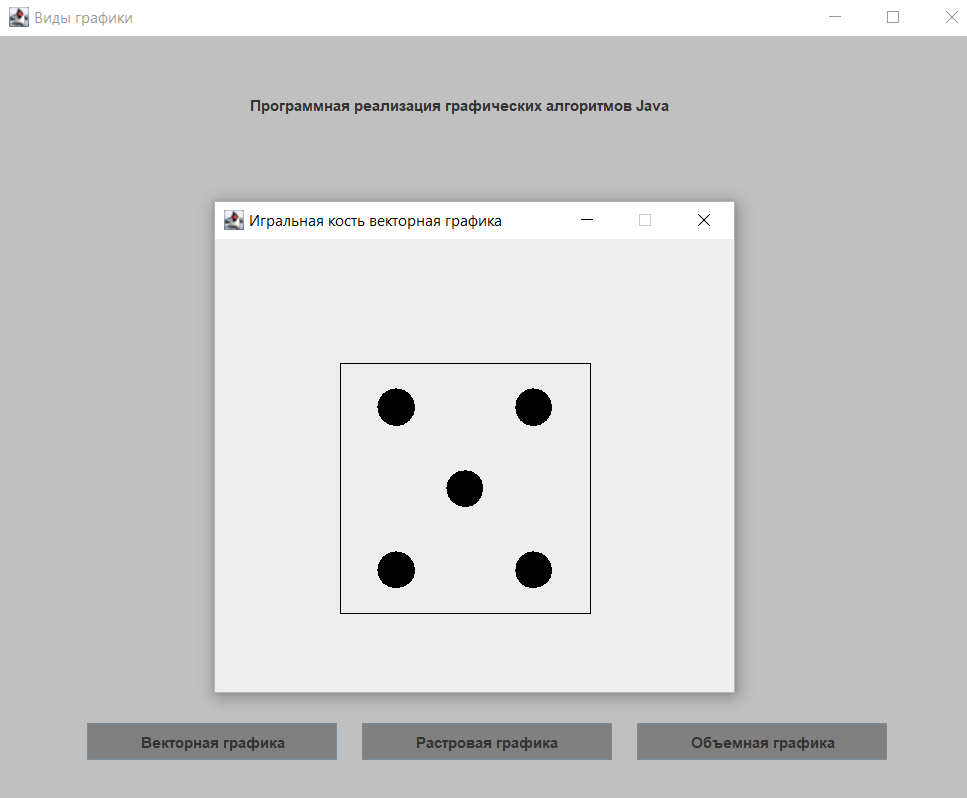


Рисунок 2 – Дополнительное окно «Игральная кость векторная графика»

При нажатии на кнопку «Растровая графика» появляется окно, на котором изображена игральная кость, нарисованная с помощью растровой графики. (Рис.3)

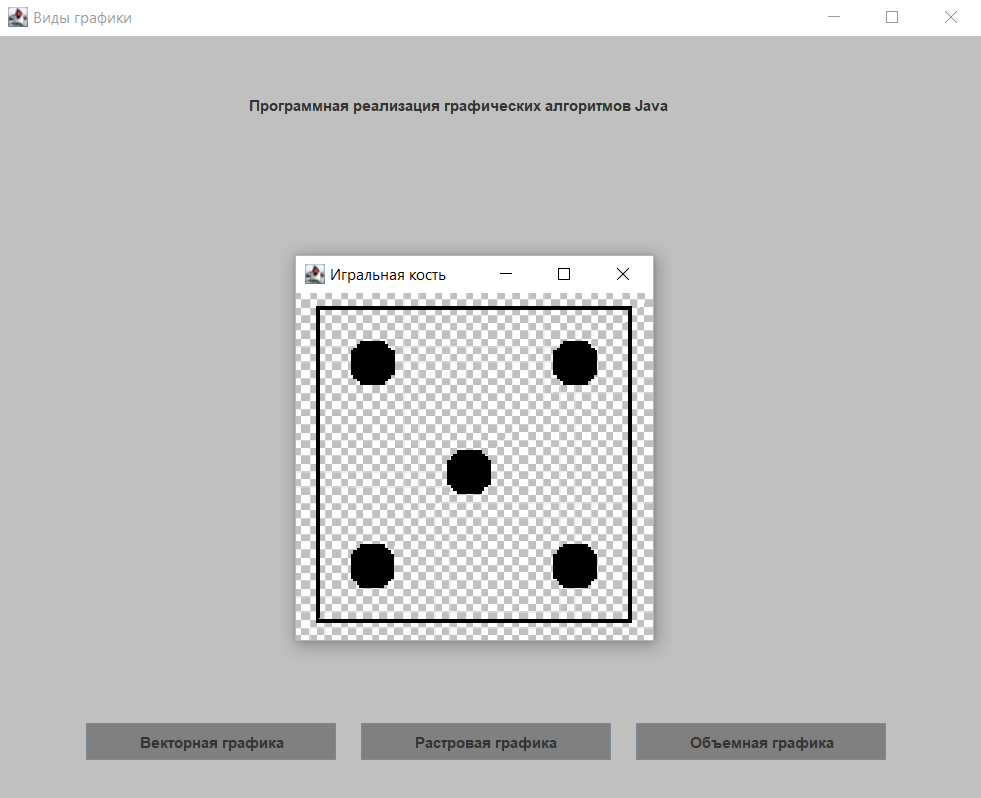


Рисунок 3 – Дополнительное окно «Игральная кость растровая графика»

При нажатии на кнопку «Объемная графика» появляется окно, на котором изображен куб, отрисованный с помощью библиотеки JavaFX.[7] (Рис.4)

При нажатии на куб один раз кнопкой мыши он увеличивается в два раза (Рис.5), если нажать кнопкой мыши на увеличенный куб он вернет свой исходный размер.

На кнопках английской раскладки A, W, S, D клавиатуры привязаны события:

* при нажатии на «A» куб двигается вправо (Рис.6);
* при нажатии на «D» куб двигается влево (Рис.7);
* при нажатии на «W» куб увеличивается до нужного пользователю размера (Рис.8);
* при нажатии на «S» куб уменьшается до нужного пользователю размера (Рис.9).

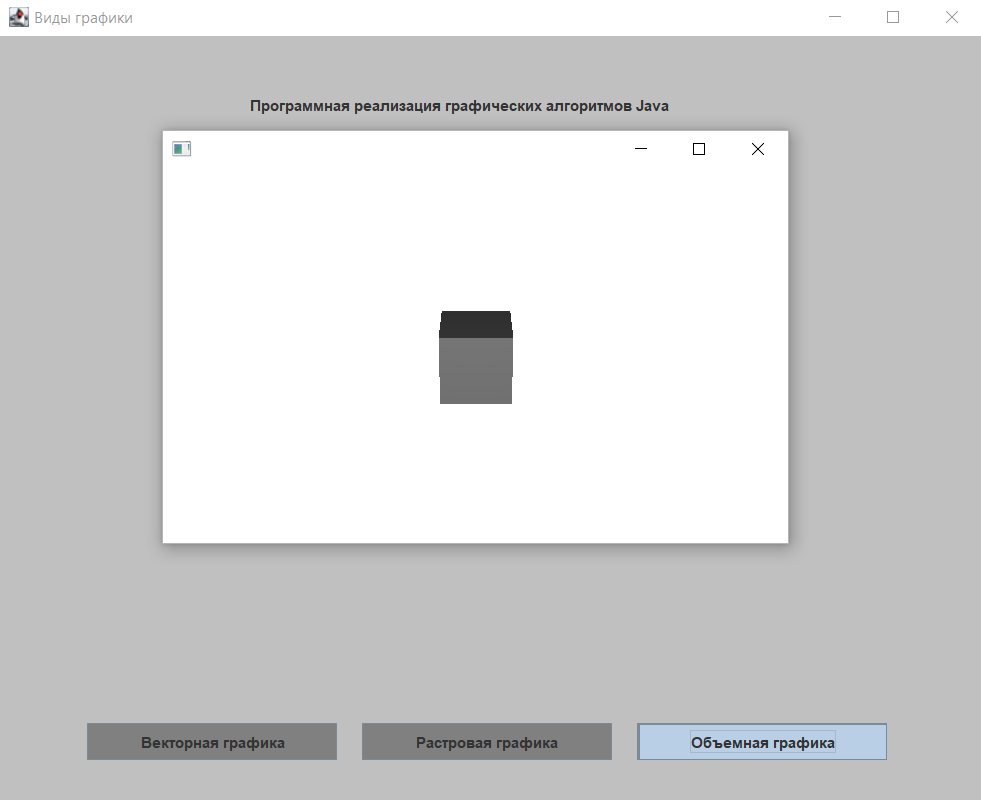


Рисунок 4 – Дополнительное окно, куб, отрисованный с помощью JavaFX

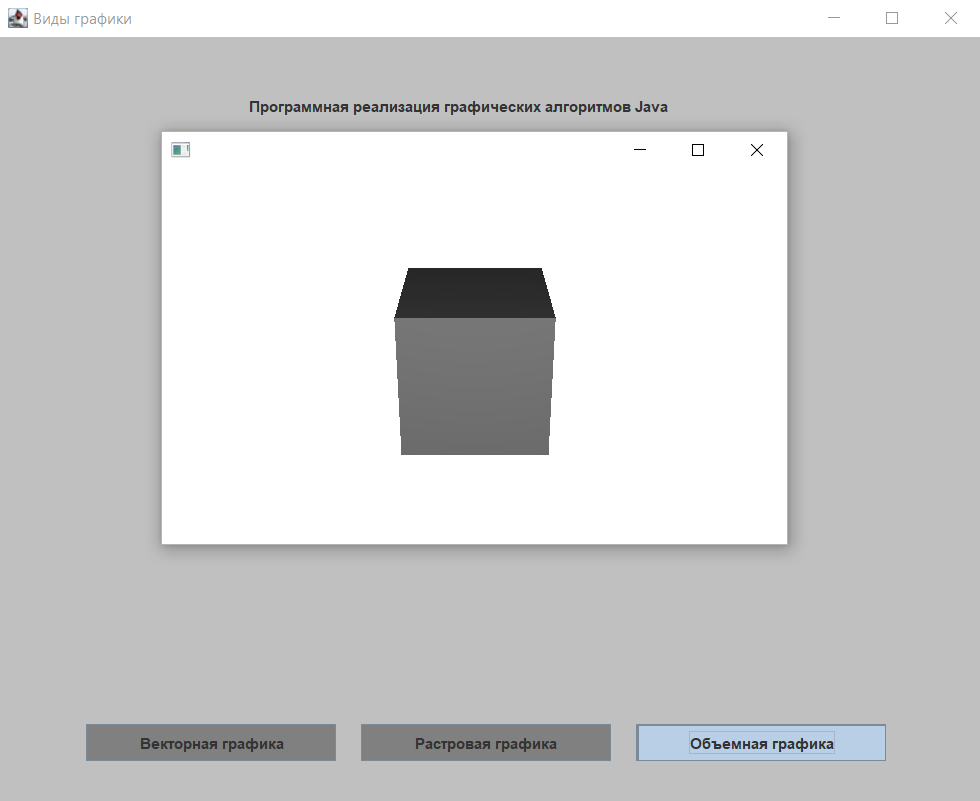


Рисунок 5 – Увеличение куба по щелчку мышки

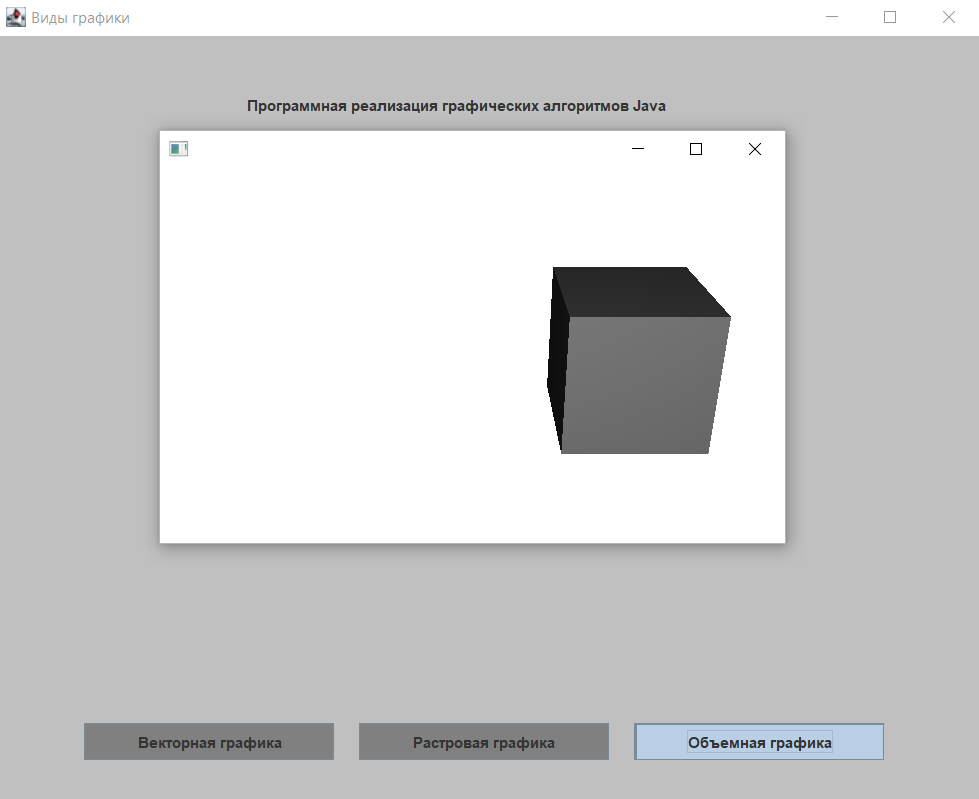


Рисунок 6 – При нажатии на «A» куб двигается вправо

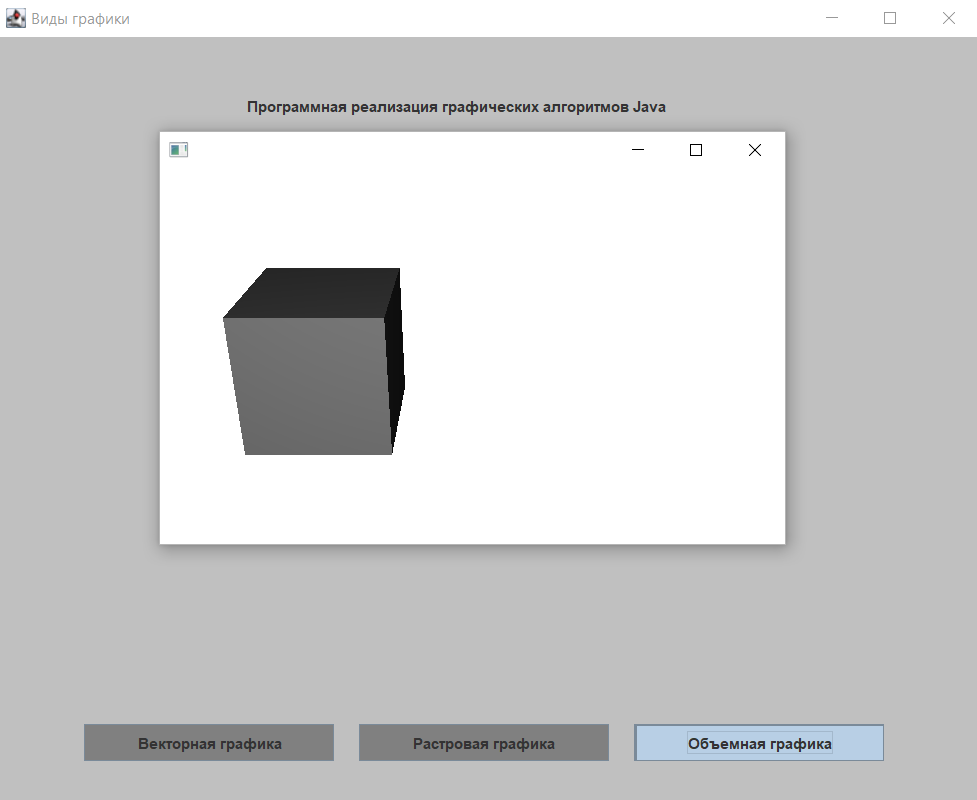


Рисунок 7 – При нажатии на «D» куб двигается влево

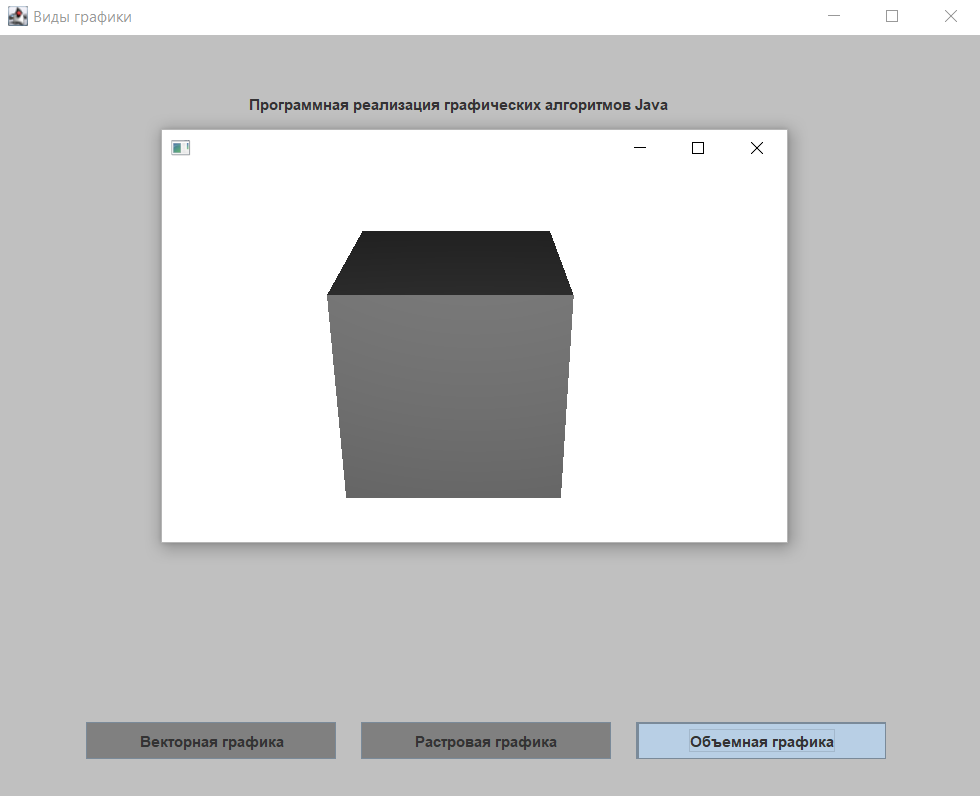


Рисунок 8 – При нажатии на «W» куб увеличивается до нужного пользователю размера

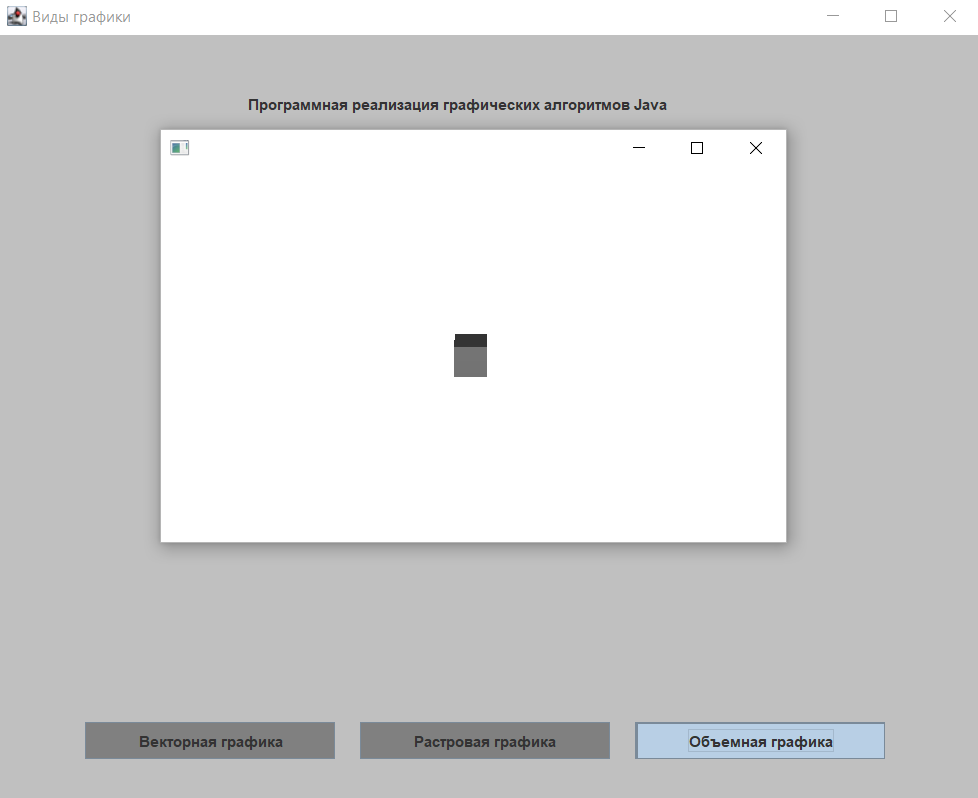


Рисунок 9 – При нажатии на «S» куб уменьшается до нужного пользователю размера

## 2.3 Разработка логики работы приложения. Схема взаимодействия компонентов проекта

Логика приложения состоит в следующем:

При запуске программы появляется главное окно JFrame, данное окно содержит в себе текстовое поле JLable и три кнопки JButton [9]«Векторная графика» vectorBtn, «Растровая графика» rasterBtn, «Объемная графика» volumeBtn.[4]

При нажатии на кнопку «Векторная графика»[8] появляется окно VectorDice в котором находится векторное изображение, так же данное окно можно закрыть, нажав на крестик.

При нажатии на кнопку «Растровая графика» появляется окно RastrDice в котором находится растровое изображение, окно можно закрыть при нажатии на крестик.

При нажатии на кнопку «Объемная графика» появляется окно VolumeDice в котором находится объёмное изображение, которое можно перемещать и увеличивать с помощью кнопок английской раскладки клавиатуры: A,W,S,D.

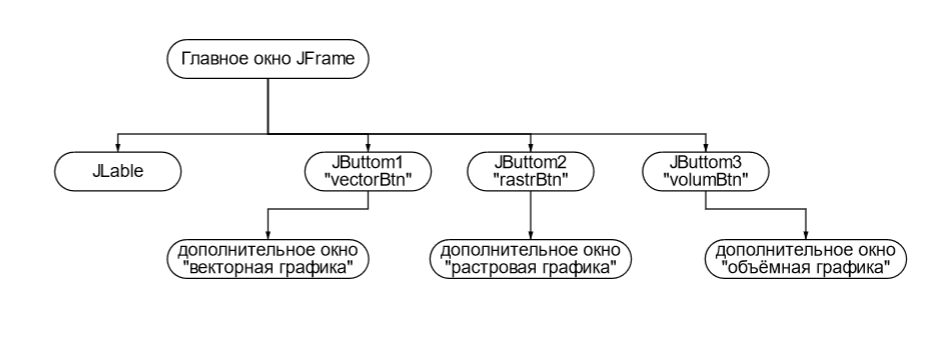
Схема логики приложения. (Рис.10)

Рисунок 10 – Схема логики приложения

Схема взаимодействия компонентов (Рис.11)

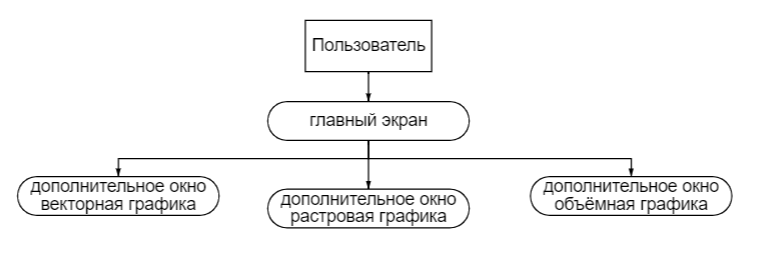


Рисунок 11 – Схема взаимодействия компонентов

## 2.4 Описание переменных, компонентов, классов и подпрограмм

Таблица 1 – Описание классов

|  |  |
| --- | --- |
| Название класса | Описание |
| Public class Main | Публичный главный класс предназначен для запуска всей программы |
| Public class UI | Публичный класс для создания графического интерфейса (кнопки, текстовое поле, цвет окна) |
| Public class VectorDice | Публичный класс для создания фигуры в векторной графике |
| Public class RastrDice | Публичный класс для создания фигуры в растровой графике |
| Public class Volum | Публичный класс для создания объёмной фигуры |
| Public class Picture | Публичный класс для окна JFrame в которое вставляется изображение |
| JFrame | Класс для создания окна, в которое добавляется объект, с которым может взаимодействовать пользователь |

Таблица 2 – Описание методов

|  |  |
| --- | --- |
| Название метода | Описание |
| add | Вставляет элемент в список по заданному индексу |
| paint | Осуществляет рисование компонентов |
| paintComponent | Вызывается для прорисовки компонента первым, а именно рисует сам компонент |
| getGraphics | Возвращает Graphics2D |
| getContentPane | Извлекает слой панели компонента, чтобы к нему можно было добавить объект |
| getChildren | Возвращает изменяемый список дочерних элементов |
| getTranslateZ | Определяет координату Z |
| getTranslateX | Определяет координату X |
| setLocation | Задает координаты элемента |
| setSize | Задает размер элемента |
| setVisible | Для отображения окна JFrame |
| setResizable | Задает может ли пользователь менять размер фрейма |
| setBounds | Перемещает и изменяет размер компонента |
| setColor | Устанавливает текущий цвет компонента |
| setFarClip | Задает расстояние от глаза до дальней плоскости |
| setDiffuseColor | Задает рассеянный цвет компонента |
| setMaterial | Устанавливает для материального объекта значение указанного объекта |
| setRotate | Устанавливает угол наклона объекта |
| setOnMouseClicked | Определяет функцию, которая вызывается при нажатии кнопки мыши |
| setScaleX | Определяет коэффициент, на который масштабируется координаты относительно центра объекта вдоль оси X |
| setScaleY | Определяет коэффициент, на который масштабируется координаты относительно центра объекта вдоль оси Y |
| setScaleZ | Определяет коэффициент, на который масштабируется координаты относительно центра объекта вдоль оси Z |
| setCamera | Указывает тип камеры, используемой для отрисовки |

# 3 Тестирование и установка приложения

Программа протестирована, все компоненты функционируют должным образом.

Таблица 3 – Функциональность компонентов программы

|  |  |  |
| --- | --- | --- |
| Компонент | Как должен работать (описание) | Работает (Да/Нет) |
| Кнопка «Векторная графика» | При нажатии на данную кнопку появляется дополнительное окно «Векторная графика», с изображение игральной кости, отрисованной в векторной графике. | Да |
| Окно «Векторная графика» | При нажатии на кнопку «Векторная графика», должно открываться данное окно, с изображением игральной кости, отрисованной в векторной графике. Так же данное окно закрывается при нажатии на «крестик» в правом верхнем углу, и сворачивается при нажатии на «-» в правом верхнем углу. | Да |
| Кнопка «Растровая графика» | При нажатии на данную кнопку появляется дополнительное окно «Растровая графика», с изображение игральной кости, отрисованной в растровой графике. | Да |
| Окно «Растровая графика» | При нажатии на кнопку «Растровая графика», должно открываться данное окно, с изображением игральной кости, отрисованной в растровой графике. Так же данное окно закрывается при нажатии на «крестик» в правом верхнем углу, и сворачивается при нажатии на «-» в правом верхнем углу. | Да |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| Кнопка «Объемная графика» | При нажатии на данную кнопку появляется дополнительное окно «Объемная графика», с изображением куба, отрисованным с помощью объемной графики. | Да |
| Окон «Объемная графика» | При нажатии на кнопку «Объемная графика», должно открываться данное окно, с изображением куба, отрисованным с помощью объемной графики. Так же данное окно закрывается при нажатии на «крестик» в правом верхнем углу, и сворачивается при нажатии на «-» в правом верхнем углу. | Да |
| Главное окно «Виды графики» | Открывается при нажатии на файл «graphics.jar». Окно закрывается при нажатии на «крестик» в правом верхнем углу, и сворачивается при нажатии на «-» в правом верхнем углу. | Да |

# Заключение

В заключении можно отметить, что в рамках данного курсового проекта была проведена разработка и реализация графических алгоритмов с использованием среды программирования IntelliJ IDEA.

В рамках проекта был рассмотрен процесс создания и настройки проекта в IntelliJ IDEA, а также возможности среды для создания графических интерфейсов пользователя.

В ходе выполнения проекта были изучены и реализованы различные графические алгоритмы, такие как алгоритмы рисования линий и кривых, объёмных фигур, алгоритмы заливки областей и т.д. Эти алгоритмы могут быть использованы в различных областях, таких как компьютерная графика, игровая разработка, анализ данных и т.д. Так же была изучена новая библиотека такая как JavaFX.

Таким образом, данный курсовой проект позволил изучить основы разработки графических алгоритмов и использования среды программирования IntelliJ IDEA для их реализации. Полученные знания и навыки могут быть полезны в будущей профессиональной деятельности в области программирования и компьютерной графики.

# Список использованной литературы

(Электронные ресурсы)

1. <https://docs.oracle.com/javase/tutorial/uiswing/> - Изучение библиотеки java.SWING
2. <https://habr.com/ru/companies/first/articles/673608/>- Разработка элементов интерфейса
3. <https://openjfx.io/> - скачивание библиотеки JavaFX
4. <https://java-online.ru/java-listener.xhtml> - Изучение событий и слушателей Java
5. <https://javarush.com/groups/posts/2560-vvedenie-v-java-fx> – Введение в JavaFX
6. <https://javaswing.wordpress.com/2009/08/11/work_with_jlabel/> - Изучение JLable для окна JFrame
7. <https://coderlessons.com/tutorials/java-tekhnologii/vyuchi-javafx/javafx-3d-shapes> – Создание объёмных фигур с помощью библиотеки JavaFX
8. <https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html> – Изучение работы ActionListener
9. [https://java-online.ru/swing-jbutton.xhtml -](https://java-online.ru/swing-jbutton.xhtml%20%20-) Расположение кнопок JButton
10. <https://java-online.ru/swing-windows.xhtml> – Методы JFrame

Приложение А. Листинг рабочего кода

Main.java

package com.example.demo;

public class Main {

public static void main(String[] args) throws Exception {

UI inter = new UI();

inter.setVisible(true);

}

}

UI.java

package com.example.demo;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class UI extends JFrame {

// создание главного окна

public UI() {

super("Виды графики");

super.setBounds(350, 70, 800, 650);

super.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

super.getContentPane().setBackground(Color.lightGray);

super.getContentPane().setLayout(null);

JLabel label = new JLabel("Программная реализация графических алгоритмов Java");

label.setBounds(200, 30, 400, 50);

super.getContentPane().add(label);

Container container = super.getContentPane();

//region Создание кнопок

//region VectorBtn

JButton vectorBtn = new JButton("Векторная графика");

vectorBtn.setBounds(70, 550, 200, 30);

vectorBtn.setBackground(Color.gray);

ActionListener listener1 = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

VectorDice vectorDice = new VectorDice();

vectorDice.show\_frame();

}

};

vectorBtn.addActionListener(listener1);

//endregion

// region RasterBtn

JButton rasterBtn = new JButton("Растровая графика");

rasterBtn.setBounds(290, 550, 200, 30);

rasterBtn.setBackground(Color.gray);

ActionListener listener2 = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

Picture picture = new Picture();

picture.rasterImage();

}

};

rasterBtn.addActionListener(listener2);

// endregion

// region VolumeBtn

JButton volumeBtn = new JButton("Объемная графика");

volumeBtn.setBounds(510, 550, 200, 30);

volumeBtn.setBackground(Color.gray);

ActionListener listener3 = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

VolumeDice volumeDice = new VolumeDice();

volumeDice.show();

}

};

volumeBtn.addActionListener(listener3);

//endregion

container.add(vectorBtn);

container.add(rasterBtn);

container.add(volumeBtn);

}

}

VectorDice.java

package com.example.demo;

import javax.swing.\*;

import java.awt.\*;

public class VectorDice extends JPanel {

private static final int SIZE = 200;

public void paint(Graphics g) {

super.paint(g);

// Основа кости-прямоугольник

g.setColor(Color.BLACK);

g.drawRect(100, 100, SIZE, SIZE);

// круги

g.fillOval(130, 120, 30, 30);

g.fillOval(185, 185, 30, 30);

g.fillOval(130, 250, 30, 30);

g.fillOval(240, 120, 30, 30);

g.fillOval(240, 250, 30, 30);

}

public static void show\_frame() {

JFrame frame = new JFrame("Игральная кость векторная графика");

frame.setSize(430, 400);

VectorDice panel = new VectorDice();

frame.getContentPane().setLayout(new BorderLayout());

frame.getContentPane().add(panel, BorderLayout.CENTER);

frame.setResizable(false);

frame.setVisible(true);

frame.setLocationRelativeTo(null);

}

public static void main(String[] args) {

show\_frame();

}

}

RasterDice.java

package com.example.demo;

import javax.imageio.ImageIO;

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.File;

public class RasterDice extends JPanel {

private static final int SIZE = 100;

private static final String FILENAME = "dice.png";

private static void show\_frame() {

// Создать новое изображение с заданными размерами и типом

BufferedImage image = new BufferedImage(200, 200, BufferedImage.TYPE\_INT\_ARGB);

// Получить объект Graphics для рисования на изображении

Graphics g = image.getGraphics();

Image img = new ImageIcon("res/Dice.PNG").getImage();

g.drawImage(img, (int) 0, (int) 0, null);

// Нарисовать кость

g.setColor(Color.BLACK);

g.drawRect(10, 10, SIZE, SIZE);

// Нарисовать круги

g.fillOval(20, 20, 15, 15);

g.fillOval(51, 55, 15, 15);

g.fillOval(20, 85, 15, 15);

g.fillOval(85, 20, 15, 15);

g.fillOval(85, 85, 15, 15);

// Сохранить изображение в файл

try {

ImageIO.write(image, "png", new File(FILENAME));

System.out.println("Изображение сохранено в файл " + FILENAME);

} catch (Exception e) {

System.out.println("Ошибка при сохранении изображения: " + e.getMessage());

}

}

public static void main(String[] args) {

show\_frame();

}

}

VolumeDice.java

package com.example.demo;

import javafx.application.Application;

import javafx.scene.\*;

import javafx.scene.input.KeyCode;

import javafx.scene.paint.Color;

import javafx.scene.paint.PhongMaterial;

import javafx.scene.shape.Box;

import javafx.scene.transform.Rotate;

import javafx.stage.Stage;

public class VolumeDice extends Application {

private final double cameraModifier = 50.0;

private final double cameraQuantity = 10.0;

public void show() {

launch();

}

@Override

public void start(Stage primaryStage) throws Exception {

Camera camera = new PerspectiveCamera(true);

camera.setFarClip(50000.0);

camera.setTranslateZ(-1000);

// создание куба - параметры

Box box = new Box(100, 100, 100);

PhongMaterial greyMaterial = new PhongMaterial();

greyMaterial.setDiffuseColor(Color.GRAY);

greyMaterial.setSpecularColor(Color.DARKGREY);

box.setMaterial(greyMaterial);

// наклон

box.setRotationAxis(Rotate.X\_AXIS);

box.setRotate(25);

// камера

Group group = new Group();

group.getChildren().add(box);

Scene scene = new Scene(group, 500, 300);

scene.setCamera(camera);

scene.setOnMouseClicked(mouseEvent -> {

Node picked = mouseEvent.getPickResult().getIntersectedNode();

if (null != picked) {

double scalar = 2;

if (picked.getScaleX() > 1)

scalar = 1;

picked.setScaleX(scalar);

picked.setScaleY(scalar);

picked.setScaleZ(scalar);

}

});

// события на кнопки: w,s,a,d

scene.setOnKeyPressed(keyEvent -> {

double change = cameraQuantity;

if (keyEvent.isShiftDown()) {

change = cameraModifier;

}

KeyCode keyCode = keyEvent.getCode();

if (keyCode == KeyCode.W) {

camera.setTranslateZ(camera.getTranslateZ() + change);

}

if (keyCode == KeyCode.S) {

camera.setTranslateZ(camera.getTranslateZ() - change);

}

if (keyCode == KeyCode.A) {

camera.setTranslateX(camera.getTranslateX() - change);

}

if (keyCode == KeyCode.D) {

camera.setTranslateX(camera.getTranslateX() + change);

}

});

primaryStage.setScene(scene);

primaryStage.show();

}

}

Picture.java

package com.example.demo;

import javax.imageio.ImageIO;

import javax.swing.\*;

import java.awt.\*;

import java.io.IOException;

public class Picture extends JPanel {

private static final JFrame frame = new JFrame("Игральная кость");

public void paintComponent(Graphics g) {

super.paintComponent(g);

Image image;

try (var in = Picture.class.getResourceAsStream("/Dice.PNG")) {

assert in != null;

image = ImageIO.read(in);

} catch (IOException e) {

throw new RuntimeException(e);

}

g.drawImage(image, 0, 0, null);

}

public static void rasterImage() {

frame.setSize(300, 315);

Picture panel = new Picture();

frame.getContentPane().setLayout(new BorderLayout());

frame.getContentPane().add(panel, BorderLayout.CENTER);

frame.add(new Picture());

frame.setVisible(true);

frame.setLocationRelativeTo(null);

}

public static void main(String[] args) {

rasterImage();

}

}

Приложение Б. Руководство пользователя.

Данную программу можно открыть с помощью файла graphics.jar. (Рис.12)



Рисунок 12- файл «graphics.jar»

При открытии появляется главное окно и кнопками. (Рис.13)

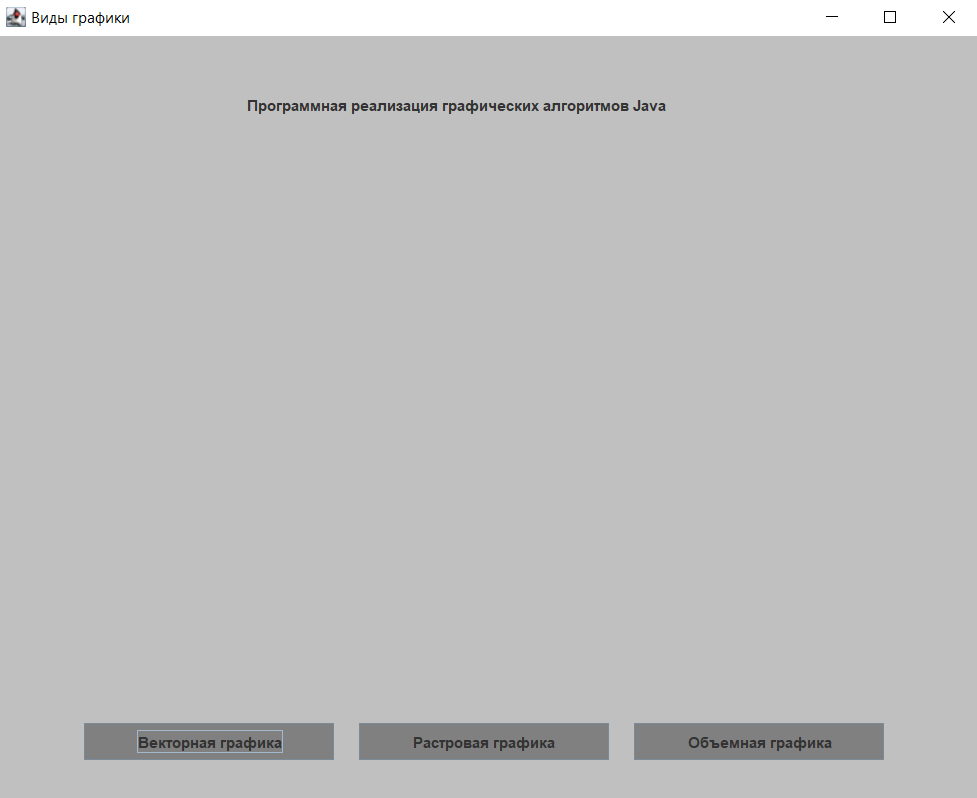
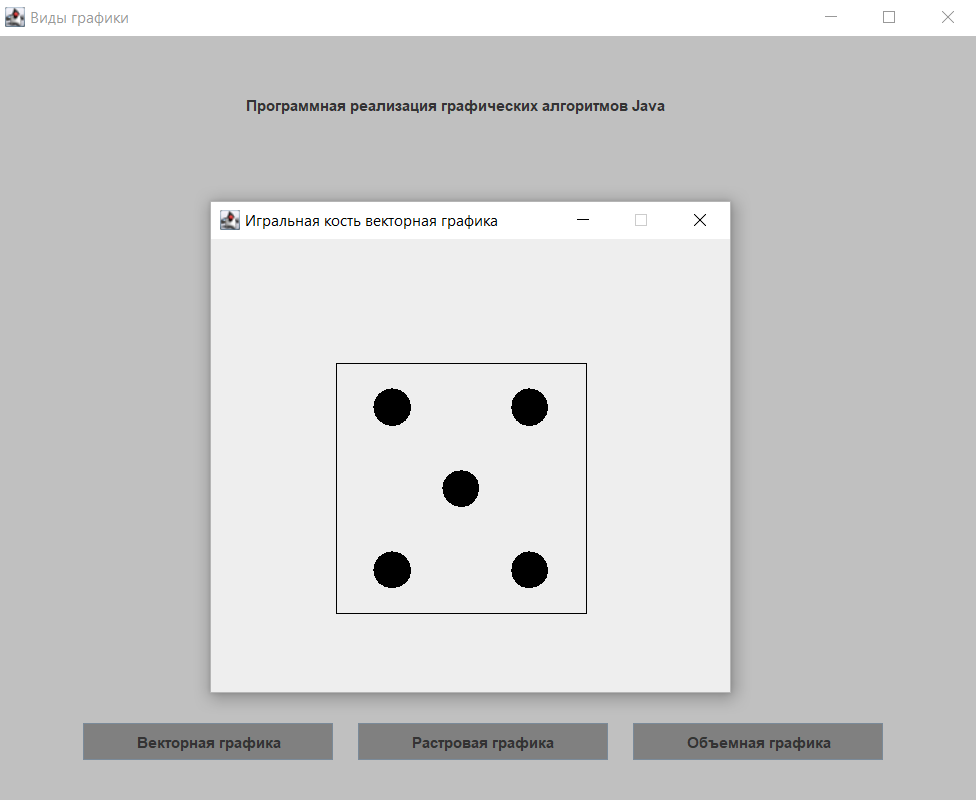


Рисунок 13 - Главное окно

При нажатии на кнопку появится дополнительное окно с изображением.

Кнопка «Векторная графика» (1) откроет окно с изображением игральной кости (2) в векторной графике. (Рис.14)

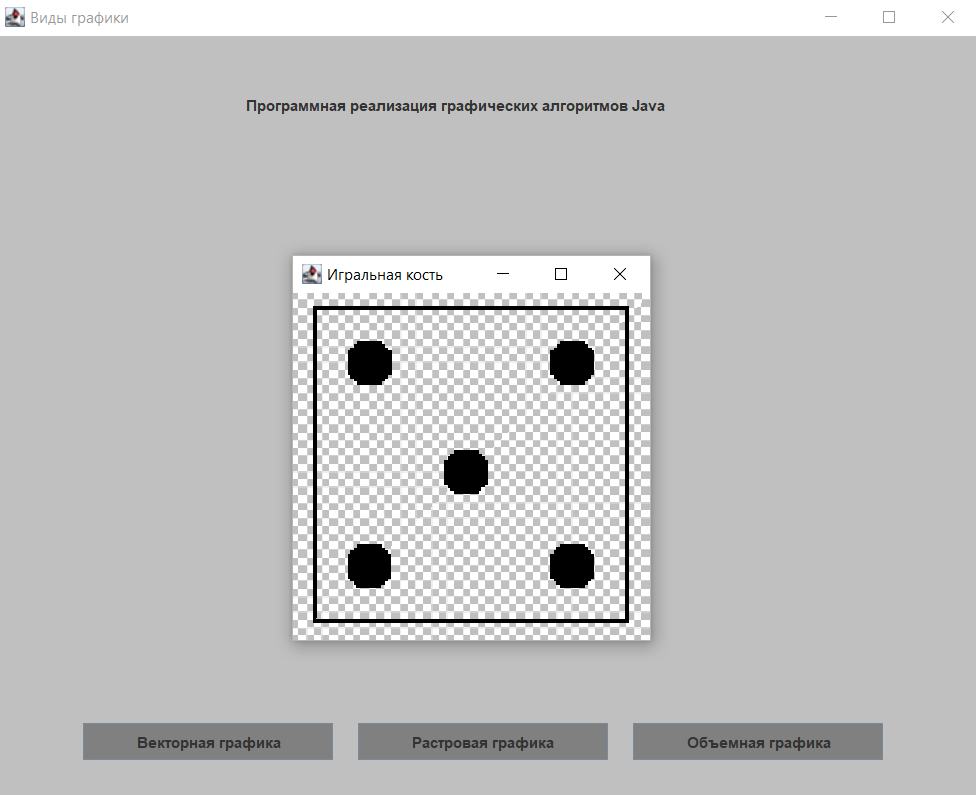


(2)

(1)

Рисунок 14 – Открытие дополнительного окна «Векторная графика»

При нажатии на кнопку «Растровая графика» (3) откроется окно с изображением игральной кости (4) в растровой графике. (Рис.15)

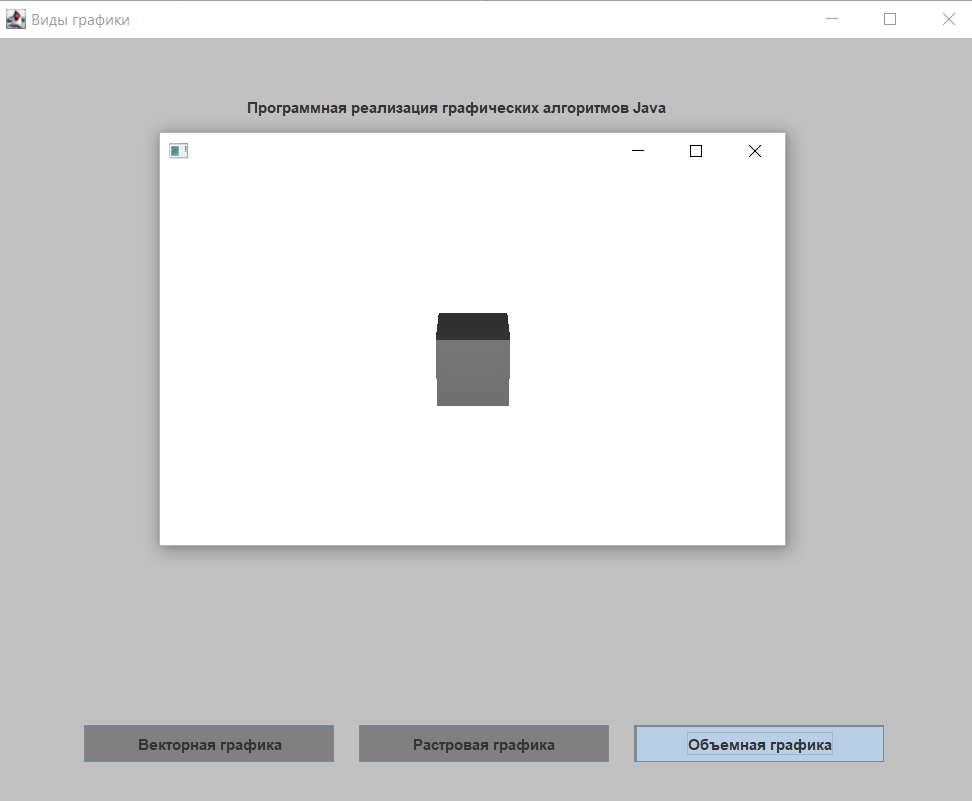


(4)

(3)

Рисунок 15 – Открытие дополнительного окна «Растровая графика»

При нажатии на кнопку «Объемная графика» (5) откроется окно с изображением куба (6) в объёмной графике. (Рис.16)



(6)

(5)

Рисунок 16 – Открытие дополнительного окна «Объёмная графика»

Каждое дополнительное окно (7) как и главное окно (8) можно закрыть, нажав на крестик в правом верхнем углу. (Рис.17)

(8)

(7)

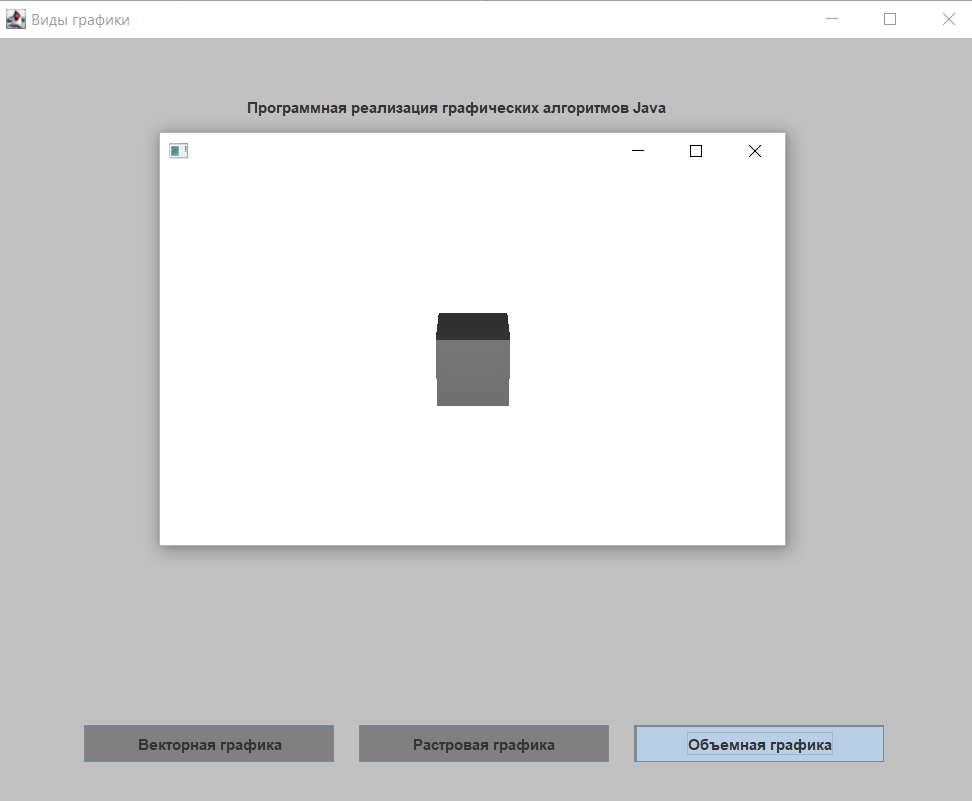


Рисунок 17 – Закрытие каждого окна