**Министерство образования Республики Беларусь**

**УО «Мозырский государственный педагогический университет имени И.П. Шамякина»**

ФИЗИКО-ИНЖЕНЕРНЫЙ ФАКУЛЬТЕТ

Кафедра информатики и методики

преподавания информатики

Курсовая работа

**Алгоритмический игровой исполнитель**

Выполнила:  
студентка 3 курса 2 группы  
физико-инженерного факультета  
Пиляк Ксения Юрьевна

Научный руководитель:  
кандидат физико-математических наук,

доцент  
Сергиевич Н.В.

Оценка научного руководителя: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 *(оценка, дата сдачи, подпись)*

Итоговая оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Мозырь 2015

**РЕЗЮМЕ**

**Выполнила:** Пиляк Ксения Юрьевна.

**Тема работы:** Алгоритмический игровой исполнитель.

**Ключевые слова:** АЛГОРИТМ, ВЕТВЛЕНИЕ, ЦИКЛ, ПОДПРОГРАММА, ИСПОЛНИТЕЛЬ, БЛОК-СХЕМА, ПРОГРАММА, JAVA, INTELLIJ IDEA, UML.

**Объект исследования:** алгоритмизация и современные технологии разработки программного обеспечения.

**Предмет исследования:** алгоритмические исполнители и их применение на уроках информатики.

**Цель работы:** изучение современных технологий программирования, алгоритмов и структур данных, методов программирования.

**Задачи работы:**

1. изучение теоретического материала темы «Основы алгоритмизации»;
2. разработка структуры программы;
3. изучение и анализ современных средств разработки и проектирования прикладного программного обеспечения;
4. разработка, тестирование, отладка, анализ программной реализации алгоритмического игрового исполнителя в среде Java.

**Структрура работы:** данная работа включает введение, 2 главы, заключение, список использованных источников из 5 наименований, приложение.

**Объем работы:** страниц – 73, рисунков – 17.

Курсовая работа посвящена написанию алгоритмического игрового исполнителя для использования учащимися на уроках информатики. В среде программирования Java выполнена его программная реализация на компьютере.

Рассмотрены основные понятия алгоритмизации, составлена структурная схема программы на унифицированном языке моделирования (UML).

Подробно описан процесс проектирования и разработки программного продукта на языке программирования Java в среде разработки IntelliJ IDEA.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 7](#_Toc438300905)

[ГЛАВА 1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 8](#_Toc438300906)

[1.1 Описание алгоритмического игрового исполнителя 8](#_Toc438300907)

[1.2 Описание интерфейса 8](#_Toc438300908)

[1.3 Теоретические основы темы «Основы алгоритмизации» 10](#_Toc438300909)

[1.4 Язык программирования Java 12](#_Toc438300910)

[ГЛАВА 2 РАЗРАБОТКА ИГРОВОГО АЛГОРИТМИЧЕСКОГО ИСПОЛНИТЕЛЯ «РОБОТ» 14](#_Toc438300911)

[2.1 Структурная схема программы 14](#_Toc438300912)

[2.2 Интерфейс TypeOfCommands 14](#_Toc438300913)

[2.3 Класс Field 15](#_Toc438300914)

[2.4 Класс Robot 18](#_Toc438300915)

[2.5 Класс ProgramField 19](#_Toc438300916)

[2.6 Класс ControlButton 22](#_Toc438300917)

[2.7 Класс Blocks 23](#_Toc438300918)

[2.8 Класс Window 25](#_Toc438300919)

[2.9 Класс Title 31](#_Toc438300920)

[2.10 Класс Main 34](#_Toc438300921)

[ЗАКЛЮЧЕНИЕ 35](#_Toc438300922)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc438300923)

[ПРИЛОЖЕНИЯ 37](#_Toc438300924)

[ПРИЛОЖЕНИЕ А. Программная реализация класса Field 37](#_Toc438300925)

[ПРИЛОЖЕНИЕ Б Программная реализация класса Robot 40](#_Toc438300926)

[ПРИЛОЖЕНИЕ В Программная реализация класса ProgramField 42](#_Toc438300927)

[ПРИЛОЖЕНИЕ Г Программная реализация класса ControlButton 44](#_Toc438300928)

[ПРИЛОЖЕНИЕ Д Программная реализация класса Blocks 46](#_Toc438300929)

[ПРИЛОЖЕНИЕ Е Программная реализация класса Window 50](#_Toc438300930)

[ПРИЛОЖЕНИЕ Ж Программная реализация класса Title 59](#_Toc438300931)

[ПРИЛОЖЕНИЕ З Программная реализация класса Main 70](#_Toc438300932)

ПРИЛОЖЕНИЕ И UML-диаграмма классов программы………………………... 71

# ВВЕДЕНИЕ

Компьютерные технологии стали неотъемлемой частью нашей жизни. Они используются во всех сферах человеческой деятельности, в том числе в промышленности, экономике, сфере услуг. Возросла потребность в специалистах, владеющих персональным компьютером, а также в разработчиках программного обеспечения. Поэтому очень важно обучить людей основам информационных технологий. Для этих целей в учреждениях образования появились компьютеры, а учащихся стали обучать компьютерной грамотности.

Довольно давно информатика стала обязательным предметом в школе. На уроках детей учат пользоваться программами, которые могут быть полезными и необходимыми в их будущей деятельности. Это такие программы как Word, Excel, PowerPoint и другие. Однако кроме обучения на пользовательском уровне, учащихся необходимо научить основам алгоритмизации и программирования, чтобы они понимали, как создаются используемые ими программы и что скрывается за их привычным интерфейсом.

Раздел «Основы алгоритмизации и программирования» обычно вызывает трудности у большинства учащихся. Во-первых, это связано с еще недостаточным уровнем сформированности алгоритмического мышления. Во-вторых, часто случается так, что учащиеся не до конца понимают, для чего нужно программирование и как алгоритмизация связана с разработкой программ, которыми они пользуются ежедневно. В-третьих, учащиеся могут быть и не заинтересованы учиться программированию, поскольку многие задачи из школьных учебников ограничиваются вводом и выводом на экран какой-то информации и не носят творческого характера, из-за чего дети теряют интерес к предмету.

Для обучения детей программированию и алгоритмизации в разное время были созданы алгоритмические исполнители такие как «Чертежник», «Кенгуренок», «Черепашка» и другие. Они намного легче языков программирования высокого уровня, а также позволяют выполнять интересные задания. Но эти исполнители, как правило, подходят для подготовленных учащихся, уже изучивших принципы алгоритмизации.

Конечно, ничто не заинтересует ребенка так, как игра. Поэтому целью данной курсовой работы является написание алгоритмического игрового исполнителя для использования учащимися на уроках информатики. Изучая алгоритмизацию в игровой форме, ученики могут развить мышление, изучить основные виды алгоритмов, а также научиться составлять их. Поэтому задачами исследования являются: подбор материала по алгоритмизации, составление интересных заданий и разработка игровой среды.

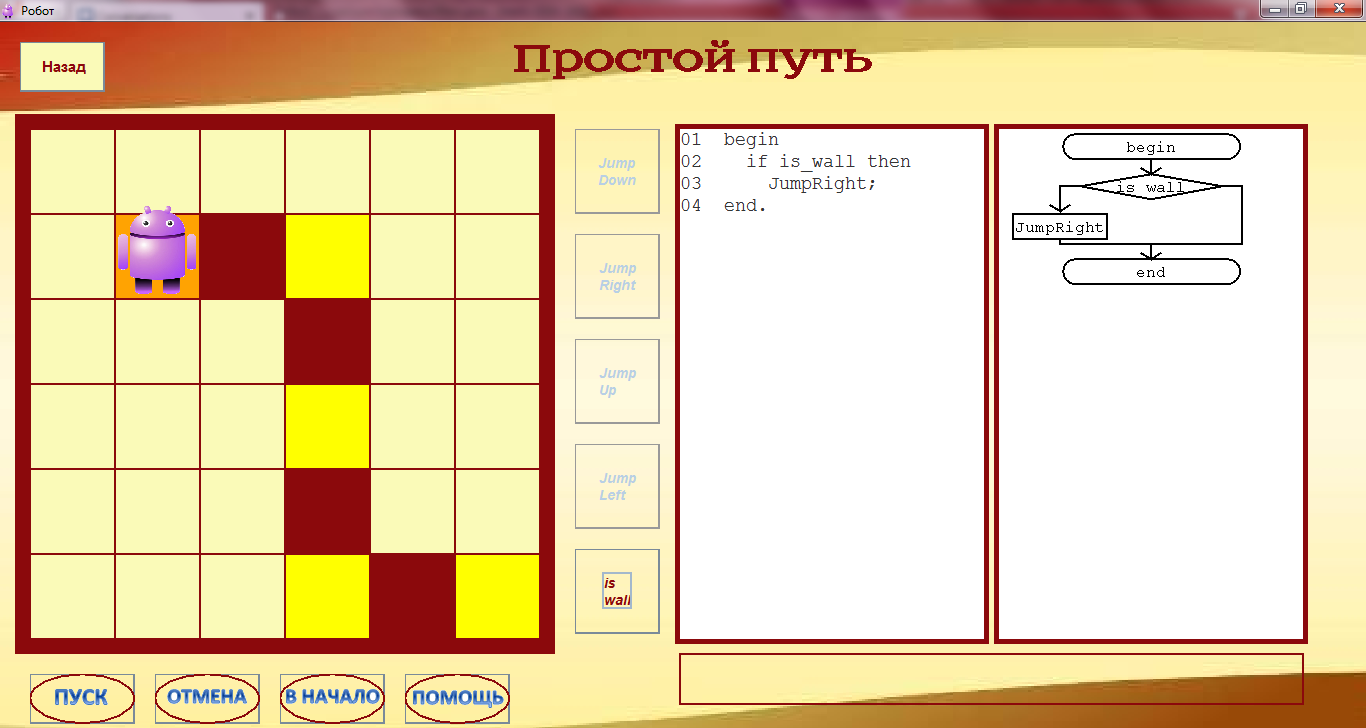
# ГЛАВА 1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## Описание алгоритмического игрового исполнителя

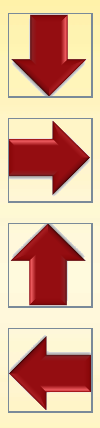
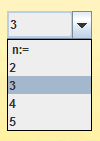
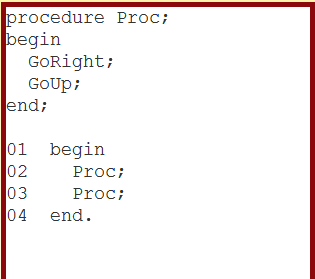
Алгоритмический игровой исполнитель «Робот» – это образовательная программа, с помощью которой осуществляется обучение учащихся основам алгоритмизации. Вся программа состоит из пяти разделов: «Линейные алгоритмы», «Алгоритмы с ветвлением», «Циклические алгоритмы», «Подпрограммы», «Контрольные задания». Каждый раздел сопровождается небольшим теоретическим материалом, в котором приводятся определение каждого типа алгоритмов, их блок-схемы, особенности использования. Также приводятся программные блоки алгоритмов на языке программирования Pascal. После теоретического материала пользователю предлагается выполнить четыре задания различных уровней сложности. В разделе «Контрольные задания» приводится сводная таблица по всем изученным видам алгоритмов и предлагается выполнить задания по одному на каждую из изученных тем. Доступ к заданиям и разделам не ограничен, таким образом, пользователь в любой момент может выбрать интересующую его тему или пройти ее заново.

## Описание интерфейса

На рабочем поле размером 6х6 находится исполнитель Робот (рис. 1.1). Некоторые клетки на поле закрашены желтым цветом. Это означает, что Роботу для достижения цели необходимо пройти все желтые клетки. В заданиях по теме «Алгоритмы с ветвлением» есть коричневые клетки – препятствия, в которые Робот попасть не может, их нужно перепрыгивать. Справаот поля находятся кнопки, которые предназначены для управления поведением Робота на поле. Подписи на некоторых кнопках и соответствующие им команды приводятся на английском языке, что способствует запоминанию учащимися английских слов, что очень важно при дальнейшем изучении программирования. Интерфейс кнопок различен для каждого вида алгоритма. В линейных алгоритмах Робот имеет возможность переходить на одну клетку вниз, вправо, вверх, влево. В алгоритмах с ветвлением для того, чтобы перепрыгнуть препятствие, необходимо сделать ветвление if is wall (кнопка «**is wall**»), после чего появится возможность использовать кнопки для перепрыгивания Робота через препятствие – они становятся активными. В условных алгоритмах в выпадающем списке можно выбирать количество повторений в цикле for. При нажатии кнопок соответствующие им команды появляются в программном блоке и в блок-схеме, а в случае вспомогательных алгоритмов вызов процедуры появляется в основной программе, а другие действия – в процедуре (рис. 1.2).

  
**Рисунок 1.1 – Интерфейс программы**

При этом Робот пока не совершает никаких действий на поле. Чтобы привести Робота в движение, необходимо нажать кнопку «**Пуск**». Чтобы отменить последнюю неверную команду, нужно нажать кнопку «**Отмена**». Кнопка «**В начало**» предназначена для возвращения Робота в исходное положение. Для получения справки по использованию программы имеется кнопка «**Помощь**». Пользователь имеет возможность использовать лишь ограниченное количество команд, что позволяет научиться составлять более эффективные алгоритмы. Внизу экрана есть строка, в которой выводятся комментарии. Если в ходе выполнения алгоритма Робот выходит за пределы поля, то выводится соответствующее сообщение об ошибке, где указывается, в какой строке была вызвана неверная команда. Если пользователь успешно справляется с заданием, то выводится надпись «Молодец!».

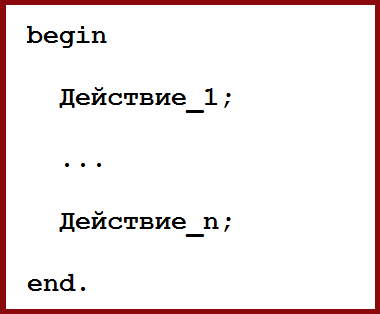
    
**Рисунок 1.2 – Элементы интерфейса программы**

## Теоретические основы темы «Основы алгоритмизации»

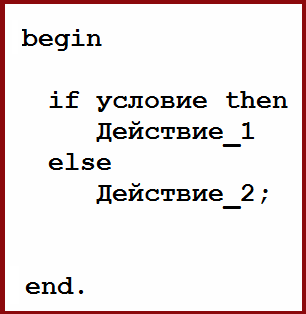
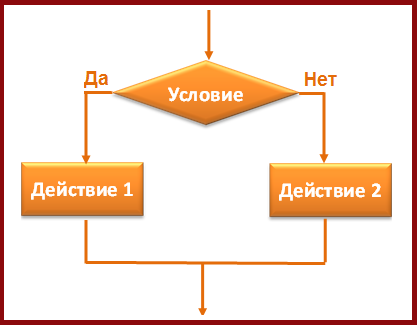
Для составления теоретической части разделов алгоритмического игрового исполнителя «Робот» подобран краткий материал по теме «Основы алгоритмизации». При этом были изучены методические разработки учителей информатики: Вилковой О.Ю. [2], Корнеевой Г.Ю. [1], Босовой Л.Л. [4]. Материал подобран с учетом возрастных особенностей пользователей. Поскольку игровой исполнитель рассчитан на детей младшего и среднего школьного возрастов, изложение материала ведется достаточно доступным и понятным для них языком. В каждый раздел включены следующие теоретические сведения:

1. **Линейные алгоритмы**

*Линейный алгоритм* – это алгоритм, в котором действия выполняются строго последовательно [1] (рис. 1.3).

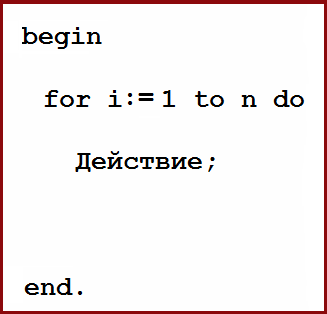
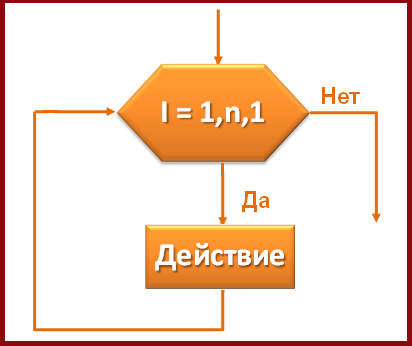
   
**Рисунок 1.3 – Линейный алгоритм**

1. **Алгоритмы с ветвлением**

   
**Рисунок 1.4 – Алгоритм с ветвлением**

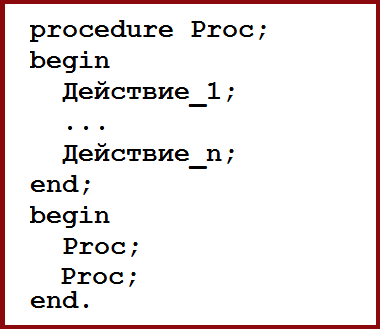
1. **Циклические алгоритмы**

*Циклический алгоритм* – это алгоритм, в котором действия выполняются определенное количество раз. На рис. 1.5 видно, что *Действие* будет выполняться *n* раз, т.е. пока условие *i* ≤ *n* истинно.

   
**Рисунок 1.5 – Циклический алгоритм**

1. **Подпрограммы**

При решении некоторых задач бывает удобно разбить их на подзадачи, каждую из которых можно оформить как самостоятельный алгоритм [2]. Каждый такой алгоритм называется подпрограммой (процедурой). Процедуру в основной программе можно вызывать неоднократно (рис. 1.5).

   
**Рисунок 1.6 – Алгоритм с подпрограммой**

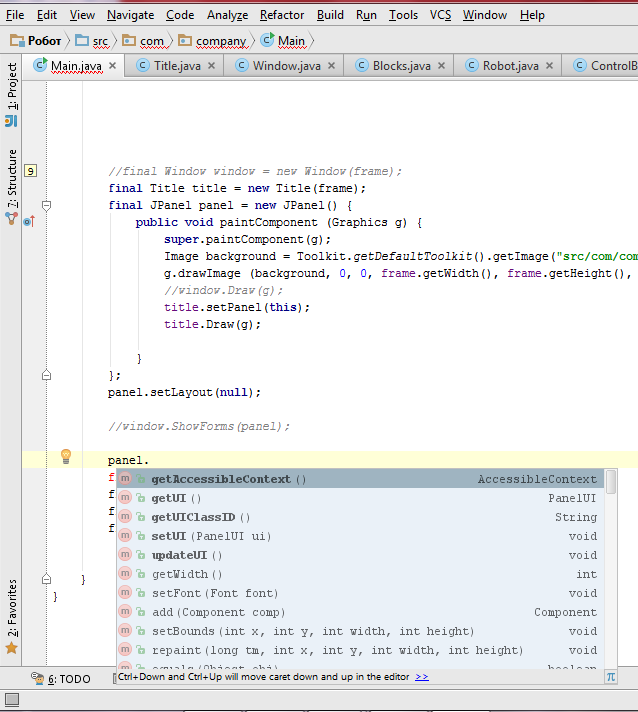
## Язык программирования Java

Алгоритмический игровой исполнитель был написан на языке программирования Java. Java является объектно-ориентированным языком программирования. Это значит, что в основе языка лежат понятия объектов и классов. Разбиение программы на объекты значительно облегчает понимание и отладку кода. В Java реализованы все три основных принципа ООП: инкапсуляция, наследование, полиморфизм. **Инкапсуляция** предполагает, что отдельный класс закрыт от других классов, они могут взаимодействовать с ним только через отдельные методы. Поля и внутренние методы класса не доступны извне, что позволяет избежать случайного изменения класса. В Java для этих целей имеются модификаторы доступа: private (закрытый, доступ внутри класса), public (открытый, доступ для любого класса), protected (защищенный, доступ для классов-наследников и для классов внутри своего же пакета). Java определяет также уровень доступа, предоставляемый по умолчанию [3] (доступ для классов внутри своего пакета). **Наследование** – процесс, в результате которого один объект получает свойство другого [3]. В Java можно наследовать классы и интерфейсы. Множественное наследование классов запрещено, но разрешено множественное наследование интерфейсов. Основной аспект **полиморфизма** – «один интерфейс, множество методов». В Java широко используется перегрузка и переопределение методов. Перегрузка позволяет использовать один и тот же метод при различных параметрах. Переопределение используется при наследовании и позволяет определять конкретные реализации всех или некоторых методов.

Одним из ключевых средств Java является многопоточность. Многопоточная программа может содержать две или более частей, которые могут выполняться одновременно. Каждая часть такой программы называется потоком, и каждый поток задает отдельный путь выполнения [3].

Программы будут разрабатываться быстрее и занимать меньше места в том случае, если большинство стандартных алгоритмов, наиболее применимых типов данных и прочих действий будет реализовано заранее и предоставлено вместе с языком программирования. Для этих целей в Java имеются стандартные библиотеки классов. Они предоставляют большую часть функциональных возможностей, обеспечиваемых средой Java [3]. Примерами таких библиотек являются библиотеки для работы с графикой (AWT, Swing), библиотека для работы с инфраструктурой Collections Framework и прочими служебными классами (java.util), библиотека обработчиков событий (java.event) и другие.

Приложения можно создавать в интегрированных средах разработки. Наиболее популярными являются Eclipse, NetBeans, IntelliJ IDEA и другие. IntelliJ IDEA – это ведущая среда быстрой разработки на языке Java. IntelliJ IDEA представляет собой высокотехнологичный комплекс тесно интегрированных инструментов программирования, включающий интеллектуальный редактор исходных текстов с развитыми средствами автоматизации, мощные инструменты рефакторинга кода, встроенную поддержку технологий J2EE, механизмы интеграции со средой тестирования Ant/JUnit и системами управления версиями, уникальный инструмент оптимизации и проверки кода Code Inspection, а также инновационный визуальный конструктор графических интерфейсов (рис. 1.7). Уникальные возможности IntelliJ IDEA избавляют программиста от груза рутинной работы, помогают своевременно устранить ошибки и повысить качество кода, поднимая продуктивность разработчика на новую высоту [5].

  
**Рисунок 1.7 – Среда разработки IntelliJ IDEA**

# ГЛАВА 2 РАЗРАБОТКА ИГРОВОГО АЛГОРИТМИЧЕСКОГО ИСПОЛНИТЕЛЯ «РОБОТ»

## Структурная схема программы

При разработке использовался объектно-ориентированный подход: это означает, что программа состоит из объектов, каждый из которых выполняет свои определенные функции. Программу можно разделить на две большие части: область теоретического обучения (класс Title) и область практического обучения (класс Window). В области теоретического обучения приводятся основные теоретические сведения по каждому из разделов. В области для практического обучения представлен пользовательский интерфейс для выполнения заданий. Его представляют следующие компоненты: рабочее поле (класс Field), исполнитель Робот (класс Robot), программный блок (класс ProgramField), блок-схема (класс Blocks), кнопки управления исполнителем (класс ControlButton). Структурная схема программы на унифицированном языке моделирования (UML) изображается на рис. 2.1

TypeOfCommands

Title

Window

Title.Buttons

ControlButton

Robot

Field

Blocks

ProgramField

Main

**Рисунок 2.1 – UML-диаграмма классов**

## Интерфейс TypeOfCommands

Поскольку в интерфейсах переменные, инициализированные нужными значениями, неявно объявляются как final static, то интерфейс можно применять для импорта совместно используемых констант в несколько классов. При включении интерфейса в класс имена всех переменных будут помещены в область констант [3]. Интерфейс TypeOfCommands содержит строковые константы – команды действий, устанавливаемых кнопкам, используемых в программе. При событии нажатия на кнопку полученная команда действия сравнивается с каждой из констант, которой соответствуют определенные действия. Подробное описание интерфейса приведено в приложении.

## Класс Field

**public class Field implements TypeOfCommands** **{**  
 private int x0, y0;  
 private int w, h;  
 static final int SQUARE\_SIZE= 85;  
 private ArrayList<Integer> Way\_X = new ArrayList<Integer>();  
 private ArrayList<Integer> Way\_Y = new ArrayList<Integer>();  
 private int[] necessary\_row;  
 private int[] necessary\_col;  
 private int[] border\_row;  
 private int[] border\_col;  
 private int robot\_x, robot\_y;

public Field (int x0, int y0, int w, int h);

public void Draw (Graphics g);

public void DrawWay (String command);

public void ClearWay ();

private void PaintCells (Graphics g);

public void setNecessaryRowCol (int[] row, int[] col);

public void setBorderRowCol (int[] row, int[] col);

public boolean AllCrossedCells ();

public void setRobotCoord (int robot\_x, int robot\_y);

**}**

Этот класс представляет собой реализацию рабочего поля исполнителя. Основными функциями данного класса являются изображение поля на экране, прорисовка траектории движения исполнителя, установка координат клеток, которые необходимо пройти (закрашиваются на поле желтым цветом), а также координаты клеток препятствий (для алгоритмов с ветвлением, закрашиваются коричневым цветом) и проверка на прохождение всех клеток исполнителем, т.е. на выполнимость задания пользователем.

Основной особенностью данного класса является наличие константы SQUARE\_SIZE, которая задает размер стороны клетки в пикселях. Она используется не только в данном классе и применяется в основном для подсчета координат Робота при движении по полю.

**Конструктор** **public Field** **(int x0, int y0, int w, int h)** устанавливает координаты x0 и y0 левого верхнего угла поля в пикселях и ширину w и высоту h поля в клетках.

**Методы**

* **public void setRobotCoord** **(int robot\_x, int robot\_y)** устанавливает начальные координаты Робота robot\_x и robot\_y, а также инициализирует массивы Way\_X и Way\_Y, в которых будут храниться координаты по x и по y всех положений Робота во время выполнения заданного пользователем алгоритма. В массивах задается середина клетки, в которой находится Робот.

this.robot\_x = robot\_x;  
this.robot\_y = robot\_y;  
Way\_X.add(robot\_x + SQUARE\_SIZE / 2);  
Way\_Y.add(robot\_y + SQUARE\_SIZE / 2);

* **public void DrawWay (String command)** в зависимости от команды command, которую должен выполнить Робот, высчитывает координаты его текущего положения в пикселях и записывает их в конец массивов Way\_X и Way\_Y. Так, например, если исполнителю была задана команда «Идти вниз» (GODOWN), то к последнему элементу массива Way\_Y прибавляется 5 пикселей и новое значение заносится в конец этого массива, координата по x остается без изменений, поэтому новый последний элемент массива будет равен предыдущему.

if (command.equals(GODOWN)) {  
 Way\_X.add(Way\_X.get(Way\_X.size() - 1));  
 Way\_Y.add(Way\_Y.get(Way\_Y.size() - 1) + 5);  
}

* **public void ClearWay** **()** очищает содержимое массивов Way\_X и Way\_Y и инициализирует их начальными координатами Робота.

Way\_X.clear();  
Way\_Y.clear();  
Way\_X.add(robot\_x + SQUARE\_SIZE / 2);  
Way\_Y.add(robot\_y + SQUARE\_SIZE / 2);

* **public void setNecessaryRowCol** **(int[] row, int[] col)** устанавливает клетки, которые нужно пройти Роботу в задании.

necessary\_row = new int[row.length];  
necessary\_col = new int[col.length];  
for (int i = 0; i < row.length; i++) {  
 necessary\_row[i] = row[i];  
 necessary\_col[i] = col[i];  
}

* **public void setBorderRowCol** **(int[] row, int[] col)** устанавливает клетки-препятствия, которые необходимо перепрыгивать Роботу в задании. Этот метод используется только для заданий с алгоритмами ветвления.

border\_row = new int[row.length];  
border\_col = new int[col.length];  
for (int i = 0; i < row.length; i++) {  
 border\_row[i] = row[i];  
 border\_col[i] = col[i];  
}

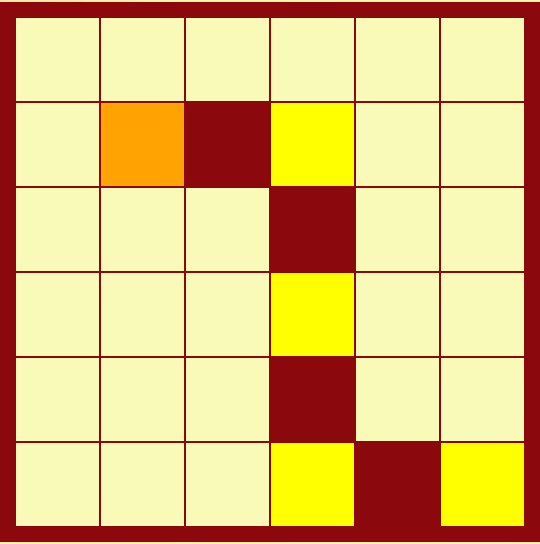
* **private void PaintCells** **(Graphics g)** закрашивает клетки, которые необходимо пройти, желтым цветом и клетки, которые нужно перепрыгивать, коричневым цветом (Приложение А).
* **public boolean AllCrossedCells** **()**проверяет, все ли клетки, которые необходимо пройти Роботу в задании, пройдены. Для этого проверяется совпадение соответствующих координат всех положений Робота (массивы Way\_X и Way\_Y) и координат клеток, которые необходимо пройти. Поскольку массивы necessary\_rowиnecessary\_colсодержат сами номера клеток, то для описанного выше сравнения необходимо найти их координаты в пикселях. Метод возвращает значение true, если все клетки были пройдены, и false – в противном случае.

int k = 0;  
for (int i = 0; i < necessary\_row.length; i++) {  
 for (int j = 0; j < Way\_X.size(); j++)  
 if (Way\_X.get(j) == (x0 + necessary\_col[i] \*

SQUARE\_SIZE + SQUARE\_SIZE / 2) &&

Way\_Y.get(j) == (y0 + necessary\_row[i] \*

SQUARE\_SIZE + SQUARE\_SIZE / 2))  
 k++;  
}  
  
if (k >= necessary\_row.length)  
 return true;  
else  
 return false;

 **Рисунок 2.2 – Рабочее поле исполнителя**

* **public void Draw** **(Graphics g)** отображает поле на экране. Кроме отрисовки клеток поля выполняется также закраска клеток, которые необходимо пройти и клеток-препятствий (вызов метода PaintCells) и изображение траектории движения Робота по данным из массивов Way\_X и Way\_Y (Приложение А). Результат работы метода показан на рис. 2.2.

## Класс Robot

**public class Robot implements TypeOfCommands {**  
 private int x0, y0;  
 private int x, y;  
 private int w, h;  
 private JFrame frame;

public Robot (int x0, int y0, int w, int h, JFrame frame);

public void Draw (Graphics g);

public void setInitialCoords ();

public int getX ();

public int getY ();

public void Move (String command);

**}**

Этот класс представляет собой реализацию исполнителя Робота. Робот способен перемещаться из клетки в клетку или перепрыгивать клетку (только в алгоритмах с ветвлением) на рабочем поле в зависимости от команды, которую задаст пользователь.

**Конструктор public Robot** **(int x0, int y0, int w, int h, JFrame frame)** устанавливает начальные координаты, ширину и высоту Робота в пикселях и фрейм, в котором изображается программа.

this.x0 = x0;  
this.y0 = y0;  
x = x0;  
y = y0;  
this.w = w;  
this.h = h;  
this.frame = frame;

**Методы**

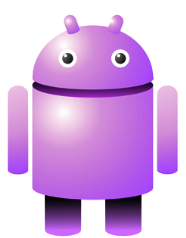
* **public void Draw** **(Graphics g)** изображает Робота на экране.

Image background = Toolkit.getDefaultToolkit().getImage(

"src/com/company/robot.png"

);  
g.drawImage (background, x, y, w, h, null);

Результат работы метода приведен на рис. 2.3.

  
**Рисунок 2.3 - Робот**

* **public void setInitialCoords** **()** устанавливает начальные координаты Робота и перерисовывает содержимое фрейма. Этот метод используется в тех случаях, когда Робота в исходное положение.

this.x = x0;  
this.y = y0;  
frame.repaint();

* **public int getX ()** и **public int getY ()** возвращают значения текущих координат Робота x и y соответственно.
* **public void Move** **(String command)** изменяет значения текущих координат Робота в зависимости от команды command, заданной пользователем и вызывает перерисовку содержимого фрейма. Эти действия имитируют передвижение Робота по полю. Если были заданы команды «Идти вниз» (GODOWN), «Идти вправо» (GORIGHT), «Идти вверх» (GOUP), «Идти влево» (GOLEFT), то значения соответствующих координат увеличиваются (уменьшаются) на 5 пикселей, поскольку Робот движется по экрану плавно. Если были заданы команды «Прыгнуть вниз» (JUMPDOWN), «Прыгнуть вправо» (JUMPRIGHT), «Прыгнуть вверх» (JUMPUP), «Прыгнуть влево» (JUMPLEFT), то значения соответствующих координат увеличиваются (уменьшаются) на две клетки, поскольку Робот прыгает через клетку мгновенно (Приложение Б).

## Класс ProgramField

**public class ProgramField implements TypeOfCommands {**  
 private JTextArea textArea;  
private String begin = "01 begin";  
 private String end = "end.";  
 private int nums\_string = 1;  
 private int n\_for;  
 private boolean isProc;  
 private String Procedure = "procedure Proc;"+"\n"+"begin";

public ProgramField (int x, int y, int w, int h);

public void AddOnPanel (JPanel panel);

public void AddText (String text);

public void AddTextToProc (String text);

public void RemoveString ();

public void RemoveStringFromProc ();

public void setNFor (int n);

public void setIsProc (boolean b);

**}**

Этот класс реализует программный блок (поле). Отражает заданные пользователем команды на экран на языке программирования, имеющий синтаксис, подобный языку Pascal. В программное поле можно как добавлять команды, так и удалять.

**Конструктор public ProgramField** **(int x, int y, int w,**

**int h)**создает текстовую область TextArea, устанавливает координаты, размеры и задает параметры области (изменяемость, шрифт).

**Методы**

* **public void AddOnPanel** **(JPanel panel)** добавляет программное поле на панель.

panel.add(textArea);

* **public void setNFor** **(int n)** устанавливает количество повторений в цикле for. Метод используется только для заданий с циклическими алгоритмами.
* **public void setIsProc** **(boolean b)** устанавливает значение переменной isProc значением b, т.е. указывает наличие процедурного блока в программе. Если переменной isProc было присвоено значение true, то в текстовую область добавляется заголовок и пустое тело процедуры.

isProc = b;  
if (isProc) {  
 textArea.removeAll();  
 textArea.setText(Procedure + "\n" + "end;" + "\n" +

"\n" + begin + "\n" + "02 " + end);  
}

* **public void AddText** **(String text)** добавляет команду text в программный блок. До добавления в текстовую область переменная text обрабатывается в зависимости от команды или оператора в соответствии с правилами оформления и форматирования кода.

if (text.equals(IF))  
 text = "if " + text + " then";  
else  
if (text.equals(FOR))  
 text = "for i:=1 to " + n\_for + " do";  
else  
if (text.equals(JUMPLEFT) || text.equals(JUMPRIGHT) ||

text.equals(JUMPDOWN) || text.equals(JUMPUP))  
 text = " " + text + ";";  
else  
 text = text + ";";

Далее обработанная строка text добавляется к строке begin. При этом ведется нумерация строк в программном блоке.

nums\_string++;  
if (nums\_string > 0 && nums\_string < 10)  
 begin = begin + "\n" + "0" + nums\_string + " " + " " + text;  
else  
 begin = begin + "\n" + nums\_string + " " + " " + text;

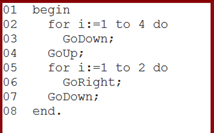
Если в программе присутствует процедурный блок, то он также добавляется в текстовую область, причем в нем строки не нумеруются.

textArea.removeAll();  
String proc = "";  
if (isProc) {  
 proc = Procedure + "\n" + "end;" + "\n" + "\n";  
}  
if (nums\_string + 1 > 0 && nums\_string + 1 < 10)  
 textArea.setText(proc + begin + "\n" + "0" +

(nums\_string + 1) + " " + end);  
else  
 textArea.setText(proc + begin + "\n" + (nums\_string +

1) + " " + end);

Результат работы метода приведен на рис. 2.4.

  
**Рисунок 2.4 – Программное поле**

* **public void AddTextToProc** **(String text)** добавляет команду text в процедурный блок. Строка text добавляется к строке Procedure и предварительно не обрабатывается, поскольку в заданиях не предусмотрены операторы ветвления или цикла в процедурном блоке.

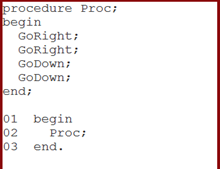
textArea.removeAll();  
Procedure = Procedure + "\n" + " " + text + ";";  
String proc = Procedure + "\n" + "end;" + "\n" + "\n";  
if (nums\_string + 1 > 0 && nums\_string + 1 < 10)  
 textArea.setText(proc + begin + "\n" + "0" +

(nums\_string + 1) + " " + end);  
else  
 textArea.setText(proc + begin + "\n" + (nums\_string

+ 1) + " " + end);

Результат работы метода приведен на рис. 2.5.

* **public void RemoveString** **()** удаляет последнюю заданную пользователем команду в основной программе. Для этого формируется новая строка, не содержащая последней строчки (Приложение В).

**  
Рисунок 2.5 – Процедурный блок в программном поле**

* **public void RemoveStringFromProc** **()** удаляет последнюю заданную пользователем команду в процедурном блоке. Этот метод аналогичен предыдущему методу (Приложение В).

## Класс ControlButton

**public class ControlButton implements TypeOfCommands** **{**  
 private int x, y, w, h;  
 private JButton button;  
 private Image bgr;

public ControlButton (int x, int y, int w, int h, Image bgr, String action\_command);

private void CreateButton(String action\_command);

public void Show (JPanel panel);

public JButton getButton();

private class RoundButton extends JButton {

public RoundButton(Icon icon)

public RoundButton (String text)

protected void paintBorder(Graphics g)

}

**}**

Создает кнопку команды или оператора или кнопку управления исполнителем (START, CANCEL, RETURN, CANCELPROC, HELP). Этот класс содержит внутри себя **закрытый** **класс** **RoundButton**, который наследуется от класса JButton и задает овальную границу кнопки (рис. 2.6). Класс RoundButton имеет два перегруженных конструктора: **public RoundButton**(Icon icon) и **public RoundButton** (String text), которые вызывают конструктор суперкласса. Метод **protected void paintBorder**(Graphics g) переопределят родительский метод (рисует овальную границу кнопки).

  
**Рисунок 2.6 – Кнопка с овальной границей**

super.paintBorder(g);  
g.setColor(getForeground());  
g.drawOval(0, 0, getSize().width - 1, getSize().height - 1);

**Конструктор** **public ControlButton** **(int x, int y, int w, int h, Image bgr, String action\_command)** задает начальные координаты, ширину и высоту кнопки в пикселях, а также фоновую картинку bgr и команду действия кнопки action\_command.

this.x = x;  
this.y = y;  
this.w = w;  
this.h = h;  
this.bgr = bgr;  
UIManager.put("Button.background", new Color (0, 0, 0, 0));  
CreateButton(action\_command);  
button.setBounds(x, y, w, h);

**Методы**

* **private void CreateButton(String action\_command)** создает экземпляр кнопки JButton (для операторов или команд) или RoundButton (для кнопок управления исполнителем) с подписью, соответствующей ее команде действия action\_command или с заданным рисунком bgr (Приложение Г).
* **public void Show** **(JPanel panel)** добавляет созданную кнопку на панель.

panel.add(button);

* **public JButton getButton ()** возвращает созданную кнопку JButton.

## Класс Blocks

**public class Blocks implements TypeOfCommands {**  
 private int x0, y0, w, h;  
 private final String BEGIN = "begin";  
 private final String END = "end";  
 private int w\_oval; *//ширина овала* private int w\_rect; *//ширина прямоугольника* private final int H\_OF\_FIGURES = 25; *//высота блока* private final int L = 15; *//длина стрелочки* private ArrayList<String> list = new ArrayList<String>();  
 private int n\_for;

public Blocks (int x0, int y0, int w, int h);

public void AddCommand (String command);

public void RemoveCommand ();

public void Draw(Graphics g);

public void setNFor (int n);

public ArrayList<String> getList ();

**}**

Этот класс предназначен для отображения блок-схемы алгоритма, заданного пользователем. При удалении или добавлении команд блок-схема перерисовывается в соответствии с текущим списком команд list.

**Конструктор public Blocks (int x0, int y0, int w, int h)** задает координаты, ширину и высоту блок-схемы и устанавливает ширину овала (для «begin» и «end») и прямоугольника (для произвольной команды).

**Методы**

* **public void setNFor** **(int n)** устанавливает количество повторений n\_for в цикле for.
* **public void AddCommand** **(String command)** добавляет команду command в список команд list. При добавлении цикла for учитывается количество повторений n\_for.

if (command.equals(FOR))  
 command = command + n\_for + ",1";  
list.add(command);

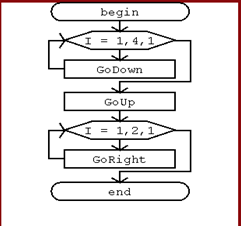
* **public void RemoveCommand** **()** удаляет последнюю введенную пользователем команду из списка команд list, если это возможно.

if (list.size() != 0)  
 list.remove(list.size() - 1);

* **public ArrayList<String> getList** **()** возвращает текущий список команд list.
* **public void Draw (Graphics g)** изображает блок-схему на экране. Поскольку циклический блок может иметь различное число повторений, то для проверки его наличия в списке list используется регулярное выражение.

String regex = "I = 1,\\d,1";  
Pattern p = Pattern.compile(regex);

Далее в соответствии со списком команд рисуется блок-схема в соответствии с ГОСТом. Слова «begin» и «end» изображаются в овалах. В блоке ветвления рисуется ромб, в циклическом блоке – шестиугольник, в линейных командах – прямоугольник. Блоки команд соединены стрелочками (Приложение Д). Результат работы метода приведен на рис. 2.7.

  
**Рисунок 2.7 –Блок-схема**

## Класс Window

**public class Window implements TypeOfCommands** **{**  
 private final int FIELD\_X = 30;  
 private int FIELD\_Y;  
 private final int FIELD\_W = 6, FIELD\_H = 6;  
 private final int PROGRAM\_FIELD\_X = FIELD\_X +

Field.SQUARE\_SIZE \* 7 + 40 + 15;  
 private int PROGRAM\_FIELD\_Y;  
 private int PROGRAM\_FIELD\_W;  
 private final int PROGRAM\_FIELD\_H = FIELD\_H \*

Field.SQUARE\_SIZE;  
 private final int ROBOT\_X = FIELD\_X;  
 private int ROBOT\_Y;  
 private final int ROBOT\_W = Field.SQUARE\_SIZE, ROBOT\_H =

Field.SQUARE\_SIZE + 10;  
 private Field field;  
 private Robot robot;  
 private ProgramField programField;  
 private Blocks blocks;  
 private int n\_for;  
 private String signal;  
 private int ms = 50;  
 private int limit;  
 private int limit\_proc;  
 private int n\_commands = 0;  
 private ArrayList<String> commands = new

ArrayList<String>();  
 private ArrayList<String> procedure = new

ArrayList<String>();  
 private JFrame frame;  
 private JPanel panel;  
 private int num = -1;  
 private int num\_for\_proc = -1;  
 public final Timer timer = new Timer(ms, new

ActionListener();

private String title;  
private ControlButton GoDown;  
private ControlButton GoRight;  
private ControlButton GoUp;  
private ControlButton GoLeft;  
private ControlButton If;  
private ControlButton Proc;  
private ControlButton JumpDown;  
private ControlButton JumpRight;  
private ControlButton JumpUp;  
private ControlButton JumpLeft;  
private ControlButton Start;  
private ControlButton Cancel;  
private ControlButton CancelProc;  
private ControlButton Return;  
private ControlButton Help;

public Window (JFrame frame, JPanel panel, int robot\_x, int robot\_y, String signal);

public void Draw (Graphics g);

public void setTitle (String title);

public void ShowForms ();

private void MakeButtons();

public Field getField ();

public void setLimit (int limit);

public void setLimitProc (int limit\_proc);

private class ControlButtonActionListener implements ActionListener {

public void actionPerformed(ActionEvent ae);

}

private class CMBActionListener implements ActionListener {  
 public void actionPerformed (ActionEvent ae)

}

**}**

Этот класс создает интерфейс пользователя для выполнения заданий. Устанавливает взаимосвязь между объектами вышеперечисленных классов: при нажатии на кнопки ControlButton определенным образом изменяется состояние объектов (например, в объекты Blocks и ProgramField добавляются или удаляются команды, Робот начинает двигаться по полю, вызывается справка по программе и т.д.).

**Конструктор public Window** **(JFrame frame, JPanel panel, int robot\_x, int robot\_y, String signal)** устанавливает координаты объектов, зависящие от размеров окна, создает объекты вышеперечисленных классов, указывает вид алгоритма текущего задания (Приложение Е).

**Методы**

* **public void setTitle** **(String title)** устанавливает заголовок (название) задания.
* **public Field getField** **()** возвращает объект класса Field.
* **public void setLimit** **(int limit)** устанавливает предел вводимых пользователем команд для конкретной задачи.
* **public void setLimitProc** **(int limit\_proc)** устанавливает предел вводимых пользователем команд для процедуры. Этот метод используется только для раздела «Подпрограммы».
* **public void Draw** **(Graphics g)** изображает созданные объекты (кроме кнопок и программного поля) на экране. Для этого используются методы Draw, которые были описаны в каждом из классов объектов.

Font font = new Font("Sreda", Font.BOLD, 45);  
g.setFont(font);  
FontMetrics metrics = g.getFontMetrics(font);  
g.setColor(new Color(139, 9, 12));  
g.drawString(title, frame.getWidth() / 2 –

metrics.stringWidth(title) / 2, 50);  
field.Draw(g);  
robot.Draw(g);  
blocks.Draw(g);  
  
*//окаймовка program\_field*  
g.setColor(new Color(139, 9, 12));  
g.fillRect(PROGRAM\_FIELD\_X - 5, PROGRAM\_FIELD\_Y - 5,

PROGRAM\_FIELD\_W + 10, PROGRAM\_FIELD\_H + 10);  
  
g.drawRect(PROGRAM\_FIELD\_X, PROGRAM\_FIELD\_Y +

PROGRAM\_FIELD\_H + 15, PROGRAM\_FIELD\_W \* 2 + 15, 50);

* **public void ShowForms** **()** добавляет все созданные формы (кнопки и программное поле) на панель (Приложение Е). Кнопки добавляются на панель в зависимости от вида алгоритма в задании (описание кнопок для каждого вида алгоритмов приведено ниже). Для кнопок используется метод Show, а для программного поля – метод AddOnPanel.
* **private void MakeButtons ()** создает кнопки команд, операторов и кнопки управления Роботом ContolButton (Приложение И). Кнопки JumpDown («Прыгнуть вниз»), JumpRight («Прыгнуть вправо»), JumpUp («Прыгнуть вверх»), JumpLeft («Прыгнуть влево») создаются только для заданий на алгоритмы с ветвлением, в остальных алгоритмах используются кнопки GoDown («Идти вниз»), GoRight («Идти вправо»), GoUp («Идти вверх»), GoLeft («Идти влево»). Кнопка If – кнопка оператора if. Для задания в программе условного оператора for используется выпадающий список JComboBox<String>, в котором пользователь может выбрать количество повторений цикла. Кнопка Proc используется для вызова процедуры в основной программе. Кнопка Start предназначена для запуска Робота. После нажатия этой кнопки Робот начинает выполнять все заданные пользователем команды, т.е. передвигаться по экрану соответствующим образом. Кнопка Cancel используется для отмены последней введенной пользователем команды в основной программе, CancelProc – для отмены последней введенной пользователем команды в процедуре (используется только в разделе «Подпрограммы»). Кнопка Return предназначена для возврата Робота в исходное положение. Кнопка Help – вызов справки по использованию программы.

**Закрытые классы.** Поскольку в программе используются кнопки и выпадающий список, то, прежде всего, к ним нужно привязать обработчики событий. Для этих целей были созданы закрытые классы ControlButtonActionListener и CMBActionListener – слушатели событий нажатия на кнопки и выбора из выпадающего списка соответственно. Оба класса реализуют интерфейс ActionListener. Этот интерфейс определяет метод actionPerformed, вызываемый при событии действия [3].

* **private class ControlButtonActionListener implements ActionListener**

public void actionPerformed(ActionEvent ae)

В зависимости от полученной слушателем команды command события ae выполняются определенные действия (Приложение Е). Количество нажатий на кнопки команд и операторов ограничивается, т.е. обработка событий происходит лишь при выполнении условий n\_commands < limit и n\_commands\_proc < limit\_proc. Кнопки **GoDown, GoRight, GoUp, GoLeft** доступны для линейных, циклических алгоритмов и подпрограмм. Если задание относится к разделу «Подпрограммы», то команды добавляются в процедурный блок ProgramField (метод AddTextToProc), а также в массив команд процедуры procedure. В блок-схему процедурные команды не добавляются. Для остальных перечисленных видов алгоритмов команды добавляются в ProgramField (метод AddText), в Blocks (метод AddCommand) и в массив команд commands, причем в количестве Field.SQUARE\_SIZE / 5, а для циклических алгоритмов в количестве n\_for \* Field.SQUARE\_SIZE / 5, где n\_for – выбранное пользователем количество повторений в цикле for, чтобы добиться плавного перемещения Робота по полю.

Кнопки **If, JumpDown, JumpRight, JumpUp, JumpLeft** присутствуют только в заданиях с алгоритмами ветвления. Они добавляются в программное поле, блок-схему и один раз в массив команд. При нажатии на кнопку If, остальные перечисленные кнопки становятся активными: теперь пользователь может задать действие в ветвлении. После этого кнопки вновь становятся неактивными до тех пор, пока вновь не будет нажата кнопка If.

При нажатии кнопки **Proc** вызов процедуры добавляется в блок-схему, программное поле в основную команду (метод AddText) и один раз в commands.

При нажатии кнопки **Start** – Робот движется по полю в зависимости от заданных пользователем команд. Для этого запускается таймер timer. Была идея вместо таймера использовать цикл для движения Робота. Но перерисовка фрейма и одновременное изменение координат Робота (метод robot.Move) невозможны, поскольку эти действия выполняются в одном потоке. В этом случае цикл будет выполнять сначала изменение координат, а лишь потом перерисовку фрейма, в результате чего пользователь не увидит на экране плавного перемещения Робота. Чтобы решить эту проблему, было принято решение воспользоваться таймером (класс Timer), который выполняет действия через определенный промежуток времени в другом потоке. В таймере последовательно извлекаются команды из массива commands или procedure и вызываются методы field.DrawWay – нарисовать траекторию движения и robot.Move – изменить координаты Робота при движении. Однако это еще не все действия таймера. Также он контролирует выход за пределы поля. Если это происходит, программа выводит на экране сообщение об ошибке с указанием, в какой строчке пользовательского программного кода была введена неверная команда. В случае, когда все клетки, которые необходимо было пройти в задании, пройдены (метод field.AllCrossedCells), выводится сообщение «Молодец!».

При нажатии кнопки **Cancel** происходит отмена последней введенной пользователем команды в основной программе. Удаление осуществляется в зависимости от оператора. Если было ветвление с действием или цикл с действием, то они удаляются вместе из программного поля, блок-схемы и массива commands (методы programFiled.RemoveString, blocks.RemoveCommand). Проверка на наличие цикла осуществляется с помощью регулярного выражения String regex = **«I = 1,\\d,1»**;, поскольку количество повторений в цикле может быть произвольным. Если был оператор без действия, то они также удаляются, как и другие команды. При этом, если был удален оператор if, то кнопки GoDown, GoRight, GoUp, GoLeft вновь становятся неактивными. Удаление действий цикла и линейных команд из массива осуществляется столько раз, сколько они добавлялись в массив.

При нажатии кнопки **CancelProc** удаляется последняя заданная пользователем команда или оператор в процедуре аналогично, как и из основной программы.

При нажатии пользователем на кнопку **Return** Робот возвращается в начальное положение (метод robot.setInitialCoords) и траектория движения очищается (метод field.ClearWay).

При нажатии на кнопку **Help** вызывается окно, в котором содержится справка по использованию программы.

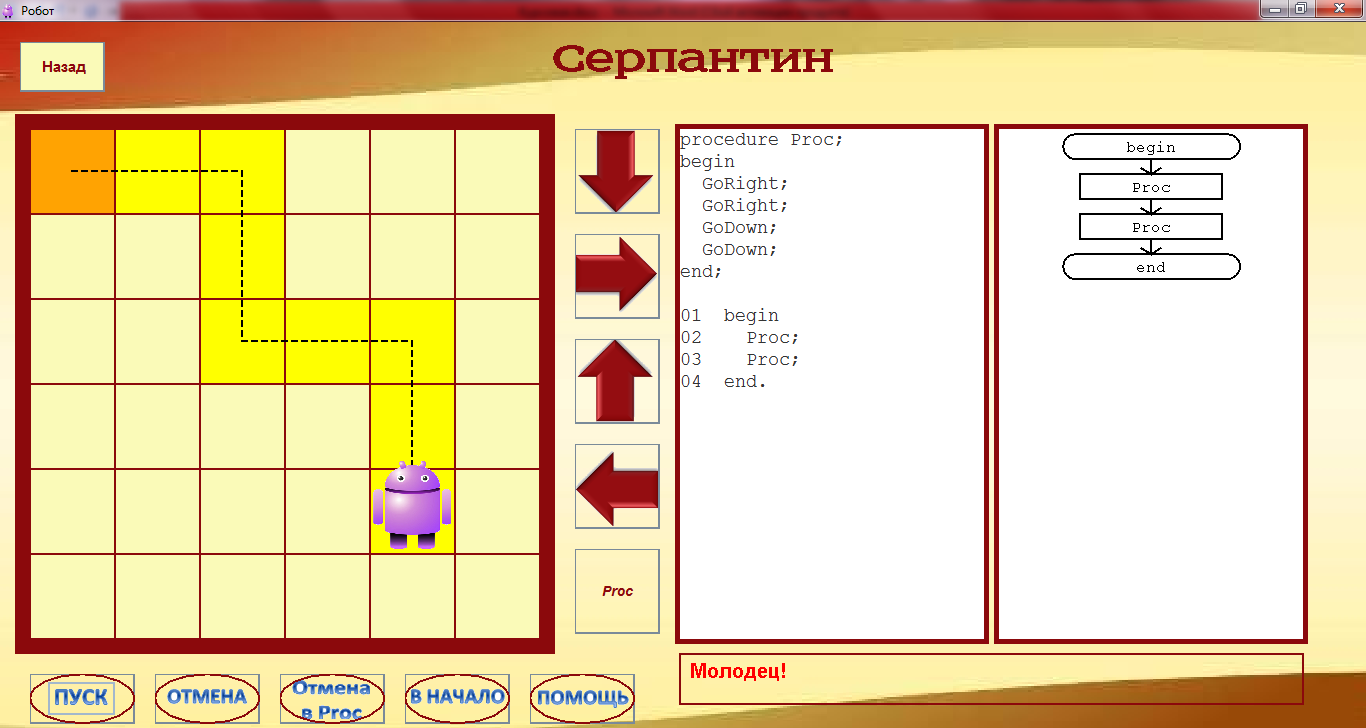
* **private class CMBActionListener implements ActionListener**

public void actionPerformed (ActionEvent ae)

Выбранное количество повторений цикла for устанавливается в блок-схему и программное поле (методы blocks.setNFor и programField.setNFor). Оператор for добавляется в программное поле, блок-схему и массив commands. Поскольку данные в выпадающем списке имеют строковый тип, то перевод из строки в число осуществляется в блоке try, чтобы перехватить исключение, если оно возникнет.

public void actionPerformed (ActionEvent ae) {  
 JComboBox box = (JComboBox)ae.getSource();  
 String item = (String)box.getSelectedItem();  
 try {  
 if (n\_commands < limit) {  
 n\_for = Integer.parseInt(item);  
 programField.setNFor(n\_for);  
 blocks.setNFor(n\_for);  
  
 programField.AddText(FOR);  
 commands.add(FOR);  
 blocks.AddCommand(FOR);  
 n\_commands++;  
 frame.repaint();  
 }  
 }  
 catch (Exception e) {  
 }  
}

Результат работы методов класса приведен на рис. 2.8.

  
**Рисунок 2.8 – Окно для выполнения заданий**

## Класс Title

**public class Title implements TypeOfCommands** **{**  
 private JFrame frame;  
 private JPanel panel;  
 private boolean isLinearAlgs = false;  
 private boolean isIfAlgs = false;  
 private boolean isForAlgs = false;  
 private boolean isProcAlgs = false;  
 private boolean isControlAlgs = false;  
 private boolean isTitle = true;  
 private boolean isWork = false;  
 private Buttons linear\_alg;private Buttons if\_alg;private Buttons for\_alg;private Buttons proc\_alg;private Buttons control\_alg;private Buttons back;private Buttons first;private Buttons second;private Buttons third;private Buttons forth;private Window window;  
 private int x\_on\_title;  
 private int y\_on\_title;

public Title (JFrame frame);

public void Draw (Graphics g);

private void MakeTitle (Graphics g, String title, String text, Image block, Image prog, Image img, String theory);

public void setPanel (JPanel panel);

private void MakeWork (int robot\_x, int robot\_y, int[] r, int[] c, String signal, String title, int limit);

private void MakeButtons();

private class ButtonActionListener implements ActionListener {  
 public void actionPerformed(ActionEvent ae);

}

private class Buttons {

public Buttons (String command, String icon, int x, int

y, int w, int h, ButtonActionListener actionListener);

public void AddOnPanel ()

public JButton getButton()

}

**}**

Этот класс осуществляет оформление титульной страницы программы и ее разделов.

Закрытый класс **Buttons** – класс, который создает кнопку Jbutton, добавляет ее на панель и возвращает ее экземпляр.

JButton button;  
public Buttons (String command, String icon, int x, int y, int w, int h, ButtonActionListener actionListener) {  
 button = new Jbutton(command);  
 button.setIcon(new ImageIcon(Toolkit.getDefaultToolkit().getImage(icon)));  
 button.setActionCommand(command);  
 button.setBounds(x, y, w, h);  
 button.addActionListener(actionListener);  
}  
  
public void AddOnPanel () {  
 panel.add(button);  
}  
  
public JButton getButton() {  
 return button;  
}

**Методы**

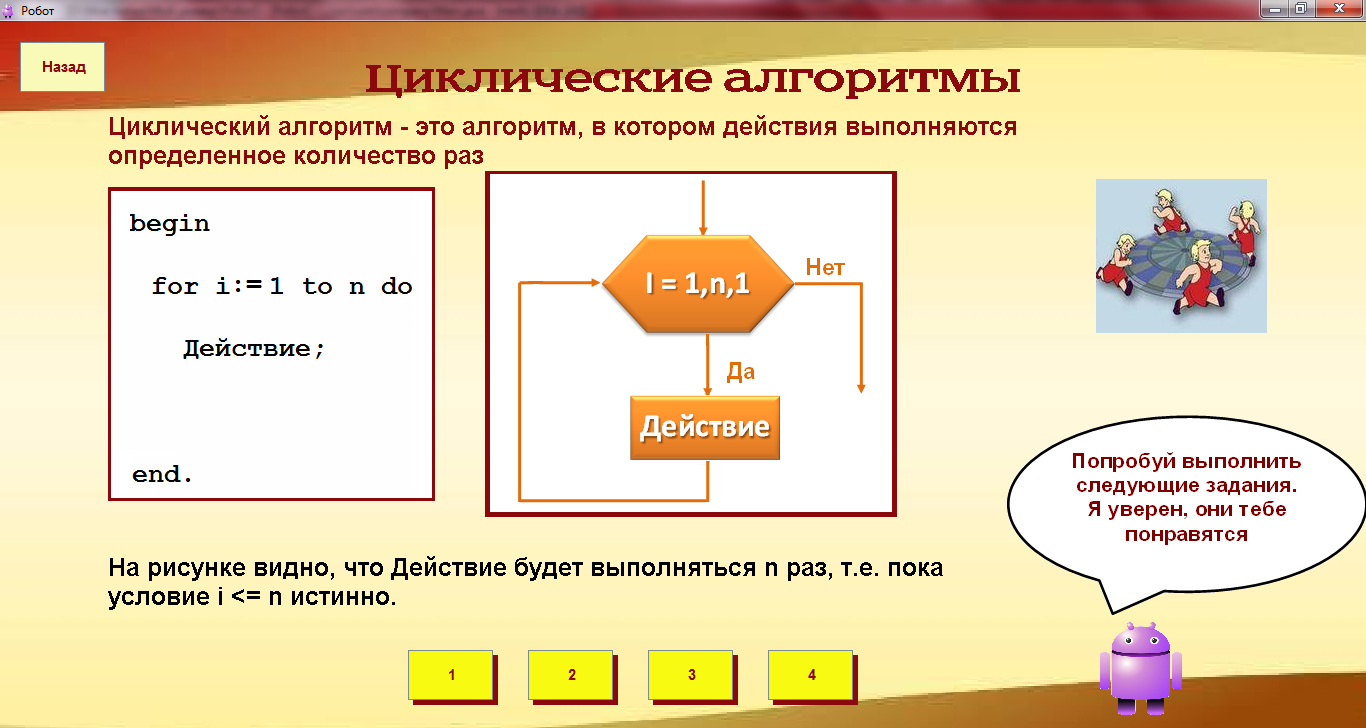
* **public void Draw** **(Graphics g)** изображает титульные страницы программы или раздела (Приложение Ж).
* **private void MakeTitle** **(Graphics g, String title, String text, Image block, Image prog, Image img, String theory)** изображает титульные страницы разделов (изображает заголовок, теорию, сопровождающие рисунки) (Приложение Ж).
* **public void setPanel** **(JPanel panel)** установить панель для рисования.
* **private void MakeButtons ()** создает кнопки Buttons: кнопки для выбора раздела («Линейные алгоритмы», «Алгоритмы с ветвлением», «Циклические алгоритмы», «Подпрограммы», «Контрольные задания») и кнопки для выбора заданий («1», «2», «3», «4») (Приложение Ж).
* **private void MakeWork** **(int robot\_x, int robot\_y, int[] r, int[] c, String signal, String title, int limit)** создает объект класса Window. Устанавливает название выбранного задания (метод window.setTitle), клетки, которые необходимо пройти (window.getField().setNecessaryRowCol) и предел вводимых команд в задании.

if (window != null)  
 window.timer.stop();  
isWork = true;  
panel.removeAll();  
window = new Window(frame, panel, robot\_x, robot\_y, signal);  
window.setTitle(title);  
window.ShowForms();  
window.getField().setNecessaryRowCol(r, c);  
window.setLimit(limit);

**Закрытый класс ButtonActionListener** реализует интерфейс ActionListener и является слушателем кнопок (Приложение Ж). В зависимости от того, какой раздел был выбран (кнопки linear\_alg, if\_alg, for\_alg, proc\_alg, control\_alg), устанавливается флаг соответствующей булевой переменной. При нажатии на кнопки first, second, third, forth задаются параметры задания пользователю: клетки, которые нужно пройти, клетки-препятствия (только для алгоритмов с ветвлением), предел вводимых команд.

Результат работы методов класса приведен на рис. 2.9, 2.10.

  
**Рисунок 2.9 – Титульная страница**

  
**Рисунок 2.10 – Титульная страница одного из разделов**

## Класс Main

Это основной класс программы, содержащий метод main (Приложение З). В этом классе создается фрейм, панель и объект класса Title. При запуске на экране появляется готовая программа.

# ЗАКЛЮЧЕНИЕ

При написании игрового исполнителя «Робот» изучены основные понятия по теме «Основы алгоритмизации» школьного курса информатики. Рассмотрены основные виды алгоритмов: линейные, условные, циклические, вспомогательные. Каждое понятие сопровождается достаточно доступным и понятным определением, блок-схемой и программной реализацией. Блок-схемы формируют у учащихся зрительное восприятие материала, а программное описание приобщает их к программированию. Для всех видов алгоритмов составлены задания. Каждое из них имеет название, созвучное содержанию задания. Эти задания не только помогут развить алгоритмическое мышление школьников, но и заинтересовать их в дальнейшем изучении алгоритмизации и программирования.

Алгоритмический игровой исполнитель «Робот» написан на языке программирования Java. Этот язык является одним из самых популярных в мире, поскольку он является языком программирования для сети Интернет, на нем разрабатываются приложения для мобильных телефонов, планшетов, компьютеров и другой электронной техники. Основной особенностью языка является его объекто-ориентированность. ООП используется для разработки современных программ, поскольку писать и отлаживать такие программы проще. ООП представлет собой методологию программирования, которая помогает организовывать сложные программы за счет использования наследования, инкапсуляции и полиморфизма [3]. Все три принципа ООП использовались при написании игрового исполнителя.

В данной работе приведено описание алгоритмического игрового исполнителя «Робот», подробно описан интерфейс, приведен теоретический материал, используемый в программе. Также показана структурная схема программы на языке UML, программная реализация на языке Java, рассмотрены взаимосвязи всех классов программы.

Алгоритмический игровой исполнитель «Робот» предназначен для обучения основам алгоритмизации детей младшего и среднего школьного возрастов. Может применяться учителем в качестве дополнительного или обучающего материала на уроках информатики, кружках и факультативах при изучении темы «Основы алгоритмизации и программирования». Перед началом выполнения заданий по каждой теме настоятельно рекомендуется изучить теоретический материал. После изучения всего курса рекомендуется пройти раздел «Контрольные задания», что укрепит полученные знания.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Корнеева, Г.Ю. Презентация «Типы алгоритмов» / Г.Ю. Корнеева // PPt4Web Хостинг презентаций [Электронный ресурс]. – 2013 – Режим доступа: <http://ppt4web.ru/informatika/tipy-algoritmov.html>. – Дата доступа: 19.12.2015.
2. Вилкова, О.Ю. Методическая разработка по информатике / О.Ю.Вилкова // Официальный сайт лицея №4 г. Саратова [Электронный ресурс]. – 2011 – Режим доступа: <http://sarlicey4.narod.ru/img_metod/robot_vilkova.doc>. – Дата доступа: 19.12.2015.
3. Шилдт, Г. Java. Полное руководство, 8-е ид / Г. Шилдт.: Пер. с англ. – М. ООО «И.Д. Вильямс », 2012. – 1104 с.
4. Босова, Л.Л. УМК по информатике для 5-7 классов / Л.Л. Босова // PPt4Web Хостинг презентаций [Электронный ресурс]. – 2007 – Режим доступа: <http://ppt4web.ru/informatika/tipy-algoritmov0.html>. Дата доступа: 19.12.2015.
5. Исаков, Б.Р. JetBrains IntelliJ IDEA 14.0.2 Ultimate (с ключом) / Б.Р. Исаков // Экспресс-файлы – сервис обмена информацией [Электронный ресурс]. – 2014 – Режим доступа: http://www.ex.ua/84164135. Дата доступа: 21.12.2015.

# ПРИЛОЖЕНИЯ

## ПРИЛОЖЕНИЕ А. Программная реализация класса Field

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.util.ArrayList;  
  
**public class** Field **implements** TypeOfCommands {  
 **private int x0**, **y0**;  
 **private int w**, **h**;  
 **static final int *SQUARE\_SIZE*** = 85;  
 **private** ArrayList<Integer> **Way\_X** = **new** ArrayList<Integer>();  
 **private** ArrayList<Integer> **Way\_Y** = **new** ArrayList<Integer>();  
 **private int**[] **necessary\_row**;  
 **private int**[] **necessary\_col**;  
 **private int**[] **border\_row**;  
 **private int**[] **border\_col**;  
 **private int robot\_x**, **robot\_y**;  
  
 **public** Field (**int** x0, **int** y0, **int** w, **int** h) {  
 **this**.**x0** = x0;  
 **this**.**y0** = y0;  
 **this**.**w** = w;  
 **this**.**h** = h;  
  
 }  
  
 **public void** Draw (Graphics g) {  
 Graphics2D g2 = (Graphics2D)g;  
 **int** x = **x0**;  
 **int** y = **y0**;  
 **int** border = 15;  
 g2.setColor(**new** Color(139, 9, 12));  
 g2.fillRect(**x0** - border, **y0** - border, **w** \* ***SQUARE\_SIZE*** + border \* 2, **h** \* ***SQUARE\_SIZE*** + border \* 2);  
 g2.setStroke(**new** BasicStroke(2));  
  
 *//рисуем клетки поля* **for** (**int** i = 0; i < **w**; i++) {  
 **for** (**int** j = 0; j < **h**; j++) {  
 g2.setColor(**new** Color(250, 250, 184));  
 g2.fillRect(x, y, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 g2.setColor(**new** Color(139, 9, 12));  
 g2.drawRect(x, y, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 x += ***SQUARE\_SIZE***;  
 }  
 y += ***SQUARE\_SIZE***;  
 x = **x0**;  
  
 }  
  
 PaintCells(g2);  
 g2.setColor(**new** Color(255, 163, 2)); *//закрасить начальную клетку робота* g2.fillRect(**robot\_x**, **robot\_y**, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 g2.setColor(**new** Color(139, 9, 12));  
 g2.drawRect(**robot\_x**, **robot\_y**, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
  
 g2.setColor(Color.***BLACK***);  
 **for** (**int** i = 1; i < **Way\_X**.size(); i += 2) {  
 g2.drawLine(**Way\_X**.get(i - 1), **Way\_Y**.get(i - 1), **Way\_X**.get(i), **Way\_Y**.get(i));  
 }  
  
  
 }  
  
 *//нарисовать траекторию движения* **public void** DrawWay(String command) {  
 **if** (command.equals(***GODOWN***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1));  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1) + 5);  
 }  
 **if** (command.equals(***GORIGHT***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1) + 5);  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1));  
 }  
 **if** (command.equals(***GOUP***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1));  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1) - 5);  
 }  
 **if** (command.equals(***GOLEFT***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1) - 5);  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1));  
 }  
 **if** (command.equals(***JUMPDOWN***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1));  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1) + Field.***SQUARE\_SIZE*** \* 2);  
 }  
 **if** (command.equals(***JUMPRIGHT***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1) + Field.***SQUARE\_SIZE*** \* 2);  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1));  
 }  
 **if** (command.equals(***JUMPUP***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1));  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1) - ***SQUARE\_SIZE*** \* 2);  
 }  
 **if** (command.equals(***JUMPLEFT***)) {  
 **Way\_X**.add(**Way\_X**.get(**Way\_X**.size() - 1) - ***SQUARE\_SIZE*** \* 2);  
 **Way\_Y**.add(**Way\_Y**.get(**Way\_Y**.size() - 1));  
 }  
 }  
  
 **public void** ClearWay () {  
 **Way\_X**.clear();  
 **Way\_Y**.clear();  
 **Way\_X**.add(**robot\_x** + ***SQUARE\_SIZE*** / 2);  
 **Way\_Y**.add(**robot\_y** + ***SQUARE\_SIZE*** / 2);  
 }  
  
 *//закрасить клетки для задания* **private void** PaintCells (Graphics g) {  
 **int** x, y;  
 **for** (**int** i = 0; i < **necessary\_row**.**length**; i++) {  
 x = **x0** + **necessary\_col**[i] \* ***SQUARE\_SIZE***;  
 y = **y0** + **necessary\_row**[i] \* ***SQUARE\_SIZE***;  
 g.setColor(**new** Color(255, 255, 0));  
 g.fillRect(x, y, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 g.setColor(**new** Color(139, 9, 12);  
 g.drawRect(x, y, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 }  
  
 **if** (**border\_row** != **null**)  
 **for** (**int** i = 0; i < **border\_row**.**length**; i++) {  
 x = **x0** + **border\_col**[i] \* ***SQUARE\_SIZE***;  
 y = **y0** + **border\_row**[i] \* ***SQUARE\_SIZE***;  
 g.setColor(**new** Color(0, 0, 0));  
 g.fillRect(x, y, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 g.setColor(**new** Color(139, 9, 12));  
 g.fillRect(x, y, ***SQUARE\_SIZE***, ***SQUARE\_SIZE***);  
 }  
 }  
  
 *//установить клетки, которые необходимо пройти* **public void** setNecessaryRowCol (**int**[] row, **int**[] col) {  
 **necessary\_row** = **new int**[row.**length**];  
 **necessary\_col** = **new int**[col.**length**];  
 **for** (**int** i = 0; i < row.**length**; i++) {  
 **necessary\_row**[i] = row[i];  
 **necessary\_col**[i] = col[i];  
 }  
  
 }  
  
 *//установть клетки-бордюры (только для условных алгоритмов)* **public void** setBorderRowCol (**int**[] row, **int**[] col) {  
 **border\_row** = **new int**[row.**length**];  
 **border\_col** = **new int**[col.**length**];  
 **for** (**int** i = 0; i < row.**length**; i++) {  
 **border\_row**[i] = row[i];  
 **border\_col**[i] = col[i];  
 }  
  
 }  
  
 *//проверить, все ли клетки в задании пройдены* **public boolean** AllCrossedCells () {  
 **int** k = 0;  
 **for** (**int** i = 0; i < **necessary\_row**.**length**; i++) {  
 **for** (**int** j = 0; j < **Way\_X**.size(); j++)  
 **if** (**Way\_X**.get(j) == (**x0** + **necessary\_col**[i] \* ***SQUARE\_SIZE*** + ***SQUARE\_SIZE*** / 2) && **Way\_Y**.get(j) == (**y0** + **necessary\_row**[i] \* ***SQUARE\_SIZE*** + ***SQUARE\_SIZE*** / 2))  
 k++;  
 }  
  
 **if** (k >= **necessary\_row**.**length**)  
 **return true**;  
 **else  
 return false**;  
 }  
  
 **public void** setRobotCoord (**int** robot\_x, **int** robot\_y) {  
 **this**.**robot\_x** = robot\_x;  
 **this**.**robot\_y** = robot\_y;  
 **Way\_X**.add(robot\_x + ***SQUARE\_SIZE*** / 2);  
 **Way\_Y**.add(robot\_y + ***SQUARE\_SIZE*** / 2);  
 }  
  
  
}

## ПРИЛОЖЕНИЕ Б Программная реализация класса Robot

**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** Robot **implements** TypeOfCommands {  
 **private int x0**, **y0**;  
 **private int x**, **y**;  
 **private int w**, **h**;  
 **private** JFrame **frame**;  
  
**public** Robot (**int** x0, **int** y0, **int** w, **int** h, JFrame frame) {  
 **this**.**x0** = x0;  
 **this**.**y0** = y0;  
 **x** = x0;  
 **y** = y0;  
 **this**.**w** = w;  
 **this**.**h** = h;  
 **this**.**frame** = frame;  
 }  
  
 **public void** Draw (Graphics g) {  
 Image background = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/robot.png"**);  
 g.drawImage (background, **x**, **y**, **w**, **h**, **null**);  
 }  
  
 **public void** setInitialCoords () {  
 **this**.**x** = **x0**;  
 **this**.**y** = **y0**;  
 **frame**.repaint();  
 }  
  
 **public int** getX() {  
 **return x**;  
 }  
  
 **public int** getY() {  
 **return y**;  
 }  
  
 **public void** Move (String command) {  
 **if** (command.equals(***GORIGHT***)) {  
 **x** += 5;  
 }  
 **if** (command.equals(***GODOWN***)) {  
 **y** += 5;  
 }  
 **if** (command.equals(***GOUP***)) {  
 **y** -= 5;  
 }  
 **if** (command.equals((***GOLEFT***))) {  
 **x** -= 5;  
  
 }  
 **if** (command.equals(***JUMPRIGHT***)) {  
 **x** += Field.***SQUARE\_SIZE*** \* 2;  
 }  
 **if** (command.equals(***JUMPDOWN***)) {  
 **y** += Field.***SQUARE\_SIZE*** \* 2;  
 }  
 **if** (command.equals(***JUMPUP***)) {  
 **y** -= Field.***SQUARE\_SIZE*** \* 2;  
 }  
 **if** (command.equals((***JUMPLEFT***))) {  
 **x** -= Field.***SQUARE\_SIZE*** \* 2;  
  
 }  
  
 **frame**.repaint();  
 }  
}

## ПРИЛОЖЕНИЕ В Программная реализация класса ProgramField

**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** ProgramField **implements** TypeOfCommands{  
 **private** JTextArea **textArea**;  
**private** String **begin** = **"01 begin"**;  
 **private** String **end** = **"end."**;  
 **private int nums\_string** = 1;  
 **private int n\_for**;  
 **private boolean isProc**;  
 **private** String **Procedure** = **"procedure Proc;"** + **"\n"** + **"begin"**;  
  
  
 **public** ProgramField (**int** x, **int** y, **int** w, **int** h) {  
 **textArea** = **new** JTextArea(**begin** + **"\n"** + **"02 "** + **end**);  
 **textArea**.setBounds(x, y, w, h);  
 **textArea**.setEditable(**false**);  
  
 **textArea**.setFont(**new** Font(**"Courier New"**, Font.***PLAIN***, 19));  
}  
  
 **public void** AddOnPanel (JPanel panel) {  
 panel.add(**textArea**);  
 }  
  
  
 *//добавить текст на панель* **public void** AddText (String text) {  
 **if** (text.equals(***IF***))  
 text = **"if is\_wall then"**;  
 **else  
 if** (text.equals(***FOR***))  
 text = **"for i:=1 to "** + **n\_for** + **" do"**;  
 **else  
 if** (text.equals(***JUMPLEFT***) || text.equals(***JUMPRIGHT***) || text.equals(***JUMPDOWN***) || text.equals(***JUMPUP***))  
 text = **" "** + text + **";"**;  
 **else** text = text + **";"**;  
 **nums\_string**++;  
 **if** (**nums\_string** > 0 && **nums\_string** < 10)  
 **begin** = **begin** + **"\n"** + **"0"** + **nums\_string** + **" "** + **" "** + text;  
 **else  
 begin** = **begin** + **"\n"** + **nums\_string** + **" "** + **" "** + text;  
 **textArea**.removeAll();  
 String proc = **""**;  
 **if** (**isProc**) {  
 proc = **Procedure** + **"\n"** + **"end;"** + **"\n"** + **"\n"**;  
 }  
 **if** (**nums\_string** + 1 > 0 && **nums\_string** + 1 < 10)  
 **textArea**.setText(proc + **begin** + **"\n"** + **"0"** + (**nums\_string** + 1) + **" "** + **end**);  
 **else  
 textArea**.setText(proc + **begin** + **"\n"** + (**nums\_string** + 1) + **" "** + **end**);  
 }  
  
 **public void** AddTextToProc (String text) {  
 **textArea**.removeAll();  
 **Procedure** = **Procedure** + **"\n"** + **" "** + text + **";"**;  
 String proc = **Procedure** + **"\n"** + **"end;"** + **"\n"** + **"\n"**;  
 **if** (**nums\_string** + 1 > 0 && **nums\_string** + 1 < 10)  
 **textArea**.setText(proc + **begin** + **"\n"** + **"0"** + (**nums\_string** + 1) + **" "** + **end**);  
 **else  
 textArea**.setText(proc + **begin** + **"\n"** + (**nums\_string** + 1) + **" "** + **end**);  
  
 }  
  
 **public void** RemoveString () {  
 String newString = **""**;  
 **int** i = **begin**.length() - 1;  
 **while** (i >= 0 && **begin**.charAt(i) != **'\n'**)  
 i--;  
 i--;  
 **for** (**int** j = i; j >= 0; j--)  
 newString = **begin**.charAt(j) + newString;  
 **nums\_string**--;  
 **begin** = newString;  
 **textArea**.removeAll();  
 String proc = **""**;  
 **if** (**isProc**)  
 proc = **Procedure** + **"\n"** + **"end;"** + **"\n"** + **"\n"**;  
 **if** (**nums\_string** + 1 > 0 && **nums\_string** + 1 < 10)  
 **textArea**.setText(proc + **begin** + **"\n"** + **"0"** + (**nums\_string** + 1) + **" "** + **end**);  
 **else  
 textArea**.setText(proc + **begin** + **"\n"** + (**nums\_string** + 1) + **" "** + **end**);  
 }  
  
 **public void** RemoveStringFromProc () {  
 String newString = **""**;  
 **int** i = **Procedure**.length() - 1;  
 **while** (i >= 0 && **Procedure**.charAt(i) != **'\n'**)  
 i--;  
 i--;  
 **for** (**int** j = i; j >= 0; j--)  
 newString = **Procedure**.charAt(j) + newString;  
 **Procedure** = newString;  
 **textArea**.removeAll();  
 String proc = **Procedure** + **"\n"** + **"end;"** + **"\n"** + **"\n"**;  
 **if** (**nums\_string** + 1 > 0 && **nums\_string** + 1 < 10)  
 **textArea**.setText(proc + **begin** + **"\n"** + **"0"** + (**nums\_string** + 1) + **" "** + **end**);  
 **else  
 textArea**.setText(proc + **begin** + **"\n"** + (**nums\_string** + 1) + **" "** + **end**);  
 }  
  
 **public void** setNFor (**int** n) {  
 **n\_for** = n;  
 }  
  
 **public void** setIsProc (**boolean** b) {  
 **isProc** = b;  
 **if** (**isProc**) {  
 **textArea**.removeAll();  
 **textArea**.setText(**Procedure** + **"\n"** + **"end;"** + **"\n"** + **"\n"** + **begin** + **"\n"** + **"02 "** + **end**);  
 }  
 }  
}

## ПРИЛОЖЕНИЕ Г Программная реализация класса ControlButton

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.geom.Ellipse2D;  
  
**public class** ControlButton **implements** TypeOfCommands {  
 **private int x**, **y**, **w**, **h**;  
 **private** JButton **button**;  
 **private** Image **bgr**;  
 **public** ControlButton (**int** x, **int** y, **int** w, **int** h, Image bgr, String action\_command) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**w** = w;  
 **this**.**h** = h;  
 **this**.**bgr** = bgr;  
 UIManager.*put*(**"Button.background"**, **new** Color (0, 0, 0, 0));  
 CreateButton(action\_command);  
 **button**.setBounds(x, y, w, h);  
  
 }  
  
 **private void** CreateButton(String action\_command) {  
 **if** (!action\_command.equals(***START***) && !action\_command.equals(***CANCEL***) && !action\_command.equals(***RETURN***) && !action\_command.equals(***CANCELPROC***) && !action\_command.equals(***HELP***))  
 **if** (**bgr** != **null**)  
 **button** = **new** JButton(**new** ImageIcon(**bgr**));  
 **else** {  
 **if** (action\_command.equals(***JUMPDOWN***))  
 **button** = **new** JButton(**"<html>Jump<br>Down</html>"**);  
 **else  
 if** (action\_command.equals(***JUMPRIGHT***))  
 **button** = **new** JButton(**"<html>Jump<br>Right</html>"**);  
 **else  
 if** (action\_command.equals(***JUMPUP***))  
 **button** = **new** JButton(**"<html>Jump<br>Up</html>"**);  
 **else  
 if** (action\_command.equals(***JUMPLEFT***))  
 **button** = **new** JButton(**"<html>Jump<br>Left</html>"**);  
 **else  
 if** (action\_command.equals(***IF***))  
 **button** = **new** JButton(**"<html>is<br>wall</html>"**);  
 **else  
 button** = **new** JButton(action\_command);  
 }  
 **else** {  
 **if** (**bgr** != **null**)  
 **button** = **new** RoundButton(**new** ImageIcon(**bgr**));  
 **else  
 button** = **new** RoundButton(action\_command);  
 }  
  
 **button**.setContentAreaFilled(**false**);  
 **button**.setFont(**new** Font(**"Arial"**, Font.***ITALIC*** | Font.***BOLD***, 14));  
  
 **button**.setActionCommand(action\_command);  
 }  
  
 **public void** Show (JPanel panel) {  
  
 panel.add(**button**);  
 }  
  
 **public** JButton getButton() {  
 **return button**;  
 }  
  
 **private class** RoundButton **extends** JButton {  
 **public** RoundButton(Icon icon) {  
 **super**(icon);  
}  
**public** RoundButton (String text) {  
 **super**(text);  
 }  
  
 **protected void** paintBorder(Graphics g) {  
 **super**.paintBorder(g);  
 g.setColor(getForeground());  
 g.drawOval(0, 0, getSize().**width** - 1, getSize().**height** - 1);  
  
 }  
 }  
  
}

## ПРИЛОЖЕНИЕ Д Программная реализация класса Blocks

**import** java.awt.\*;  
**import** java.util.ArrayList;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**public class** Blocks **implements** TypeOfCommands{  
 **private int x0**, **y0**, **w**, **h**;  
 **private final** String **BEGIN** = **"begin"**;  
 **private final** String **END** = **"end"**;  
 **private int w\_oval**; *//ширина овала* **private int w\_rect**; *//ширина прямоугольника* **private final int H\_OF\_FIGURES** = 25; *//высота блока* **private final int L** = 15; *//длина стрелочки* **private** ArrayList<String> **list** = **new** ArrayList<String>();  
  
 **private int n\_for**;  
  
 **public** Blocks(**int** x0, **int** y0, **int** w, **int** h) {  
 **this**.**x0** = x0;  
 **this**.**y0** = y0;  
 **this**.**w** = w;  
 **this**.**h** = h;  
 **w\_oval** = w / 2;  
 **w\_rect** = w / 2 - 10;  
 }  
  
 *//добавить команду в блок-схему* **public void** AddCommand (String command) {  
 **if** (command.equals(***FOR***))  
 command = command + **n\_for** + **",1"**;  
 **list**.add(command);  
 }  
  
 *//удалить команду из блок-схемы* **public void** RemoveCommand () {  
  
 **if** (**list**.size() != 0)  
 **list**.remove(**list**.size() - 1);  
 }  
  
 **public void** Draw(Graphics g) {  
 *//регулярное выражение для проверки наличия циклического блока* String regex = **"I = 1,\\d,1"**;  
 Pattern p = Pattern.*compile*(regex);  
  
  
 Graphics2D g2 = (Graphics2D)g;  
 g2.setColor(**new** Color(139, 9, 12));  
 g2.fillRect(**x0** - 5, **y0** - 5, **w** + 10, **h** + 10);  
  
 g2.setColor(Color.***white***);  
 g2.fillRect(**x0**, **y0**, **w**, **h**);  
  
 **int** y = **y0**;  
 g2.setColor(Color.***black***);  
 Font font = **new** Font(**"Courier New"**, Font.***PLAIN***, 16);  
 FontMetrics metrics = g2.getFontMetrics(font);  
 g2.setFont(font);  
 **int** x = **x0** + **w** / 2 - **w\_oval** / 2;  
  
 *//слово "begin"*

y += 5;

g2.drawLine(x, y, x + **w\_oval**, y);  
 g2.drawLine(x, y + **H\_OF\_FIGURES**, x + **w\_oval**, y + **H\_OF\_FIGURES**);  
 g2.drawArc(x - **H\_OF\_FIGURES** / 2, y, **H\_OF\_FIGURES**, **H\_OF\_FIGURES**, 90, 180);  
 g2.drawArc(x + **w\_oval** - **H\_OF\_FIGURES** / 2, y, **H\_OF\_FIGURES**, **H\_OF\_FIGURES**, 90, -180);  
 g2.drawString(**BEGIN**, x + **w\_oval** / 2 - metrics.stringWidth(**BEGIN**) / 2, y + **H\_OF\_FIGURES** / 2 + metrics.getHeight() / 3);  
  
 *//стрелочка* **int** l\_arrow = 5;  
 **float** alpha = (**float**)Math.***PI*** / 6;  
 g2.drawLine(**x0** + **w** / 2, y + **H\_OF\_FIGURES**, **x0** + **w** / 2, y + **H\_OF\_FIGURES** + **L**);  
 g2.drawLine(**x0** + **w** / 2, y + **H\_OF\_FIGURES** + **L**, (**int**)(**x0** + **w** / 2 - l\_arrow / Math.*sin*(alpha)), (**int**)(y + **H\_OF\_FIGURES** + **L** - l\_arrow / Math.*cos*(alpha)));  
 g2.drawLine(**x0** + **w** / 2, y + **H\_OF\_FIGURES** + **L**, (**int**)(**x0** + **w** / 2 + l\_arrow / Math.*sin*(alpha)), (**int**)(y + **H\_OF\_FIGURES** + **L** - l\_arrow / Math.*cos*(alpha)));  
  
 *//введенные команды* x = **x0** + **w** / 2 - **w\_rect** / 2;  
 **for** (**int** i = 0; i < **list**.size(); i++) {  
 y += **H\_OF\_FIGURES** + **L**;  
 *//блок ветвления* **if** (**list**.get(i).equals(***IF***)) {  
 g2.drawLine (x, y + **H\_OF\_FIGURES** / 2, x + **w\_rect** / 2, y);  
 g2.drawLine(x + **w\_rect** / 2, y, x + **w\_rect**, y + **H\_OF\_FIGURES** / 2);  
 g2.drawLine(x + **w\_rect**, y + **H\_OF\_FIGURES** / 2, x + **w\_rect** / 2, y + **H\_OF\_FIGURES**);  
 g2.drawLine(x + **w\_rect** / 2, y + **H\_OF\_FIGURES**, x, y + **H\_OF\_FIGURES** / 2);  
  
 *//стрелочка* g2.drawLine(x, y + **H\_OF\_FIGURES** / 2, x - 20, y + **H\_OF\_FIGURES** / 2);  
 g2.drawLine(x - 20, y + **H\_OF\_FIGURES** / 2, x - 20, y + **H\_OF\_FIGURES** / 2 + 25);  
 g2.drawLine(x - 20, y + **H\_OF\_FIGURES** / 2 + 25, (**int**)(x - 20 - l\_arrow / Math.*sin*(alpha)), (**int**)(y + **H\_OF\_FIGURES** / 2 + 25 - l\_arrow / Math.*cos*(alpha)));  
 g2.drawLine(x - 20, y + **H\_OF\_FIGURES** / 2 + 25, (**int**)(x - 20 + l\_arrow / Math.*sin*(alpha)), (**int**)(y + **H\_OF\_FIGURES** / 2 + 25 - l\_arrow / Math.*cos*(alpha)));  
  
 g2.drawLine(x + **w\_rect**, y + **H\_OF\_FIGURES** / 2, x + **w\_rect** + 20, y + **H\_OF\_FIGURES** / 2);  
 g2.drawLine(x + **w\_rect** + 20, y + **H\_OF\_FIGURES** / 2, x + **w\_rect** + 20, y + **H\_OF\_FIGURES** / 2 + 25);  
 g2.drawString(**list**.get(i), x + **w\_rect** / 2 - metrics.stringWidth(**list**.get(i)) / 2, y + **H\_OF\_FIGURES** / 2 + metrics.getHeight() / 3);  
 }  
 **else  
 if** (**list**.get(i).equals(***JUMPLEFT***) || **list**.get(i).equals(***JUMPDOWN***) || **list**.get(i).equals(***JUMPRIGHT***) || **list**.get(i).equals(***JUMPUP***)) {  
 g2.drawRect(x - 20 - **w\_rect** / 3, y, (**int**)(**w\_rect** / 1.5), **H\_OF\_FIGURES**);  
 g2.drawString(**list**.get(i), x - 20 - **w\_rect** / 3 + **w\_rect** / 3 - metrics.stringWidth(**list**.get(i)) / 2, y + **H\_OF\_FIGURES** / 2 + metrics.getHeight() / 3);  
 g2.drawLine (x - 20, y + **H\_OF\_FIGURES**, x - 20, y + **H\_OF\_FIGURES** + 5);  
 g2.drawLine(x - 20, y + **H\_OF\_FIGURES** + 5, x + **w\_rect** + 20, y + **H\_OF\_FIGURES** + 5);  
 g2.drawLine(x + **w\_rect** + 20, y + **H\_OF\_FIGURES** + 5, x + **w\_rect** + 20, y - 10);  
 y += 5;  
 }  
 **else  
 if** (p.matcher(**list**.get(i)).matches()) {  
 g2.drawLine (x, y + **H\_OF\_FIGURES** / 2, x + **w\_rect** / 4, y);  
 g2.drawLine(x + **w\_rect** / 4, y, x + (**int**) (**w\_rect** \* 0.75), y);  
 g2.drawLine(x + (**int**) (**w\_rect** \* 0.75), y, x + **w\_rect**, y + **H\_OF\_FIGURES** / 2);  
 g2.drawLine(x + **w\_rect**, y + **H\_OF\_FIGURES** / 2, x + (**int**) (**w\_rect** \* 0.75), y + **H\_OF\_FIGURES**);  
 g2.drawLine(x + (**int**) (**w\_rect** \* 0.75), y + **H\_OF\_FIGURES**, x + **w\_rect** / 4, y + **H\_OF\_FIGURES**);  
 g2.drawLine(x + **w\_rect** / 4, y + **H\_OF\_FIGURES**, x, y + **H\_OF\_FIGURES** / 2);  
 g2.drawString(**list**.get(i), x + **w\_rect** / 2 - metrics.stringWidth(**list**.get(i)) / 2, y + **H\_OF\_FIGURES** / 2 + metrics.getHeight() / 3);  
 }  
 **else** {  
 g2.drawRect(x, y, **w\_rect**, **H\_OF\_FIGURES**);  
 g2.drawString(**list**.get(i), x + **w\_rect** / 2 - metrics.stringWidth(**list**.get(i)) / 2, y + **H\_OF\_FIGURES** / 2 + metrics.getHeight() / 3);  
 *//для циклического блока* **if** (i - 1 >= 0 && p.matcher(**list**.get(i - 1)).matches()) {  
 g2.drawLine(x, y + **H\_OF\_FIGURES** / 2, x - 20, y + **H\_OF\_FIGURES** / 2);  
 g2.drawLine(x - 20, y + **H\_OF\_FIGURES** / 2, x - 20, y - **H\_OF\_FIGURES** / 2 - **L**);  
 g2.drawLine (x - 20, y - **H\_OF\_FIGURES** / 2 - **L**, x, y - **H\_OF\_FIGURES** / 2 - **L**);  
 *//стрелочка* g2.drawLine(x, y - **H\_OF\_FIGURES** / 2 - **L**, (**int**)(x - l\_arrow / Math.*cos*(alpha)), (**int**)(y - **H\_OF\_FIGURES** / 2 - **L** - l\_arrow / Math.*sin*(alpha)));  
 g2.drawLine(x, y - **H\_OF\_FIGURES** / 2 - **L**, (**int**)(x - l\_arrow / Math.*cos*(alpha)), (**int**)(y - **H\_OF\_FIGURES** / 2 - **L** + l\_arrow / Math.*sin*(alpha)));  
  
 g2.drawLine(x + **w\_rect**, y - **H\_OF\_FIGURES** / 2 - **L**, x + **w\_rect** + 20, y - **H\_OF\_FIGURES** / 2 - **L**);  
 g2.drawLine(x + **w\_rect** + 20, y - **H\_OF\_FIGURES** / 2 - **L**, x + **w\_rect** + 20, y + **H\_OF\_FIGURES** + 5);  
 g2.drawLine(x + **w\_rect** + 20, y + **H\_OF\_FIGURES** + 5, x + **w\_rect** / 2, y + **H\_OF\_FIGURES** + 5);  
 y += 5;  
 }  
 }  
 **if** (!**list**.get(i).equals(***IF***)) {  
 g2.drawLine(**x0** + **w** / 2, y + **H\_OF\_FIGURES**, **x0** + **w** / 2, y + **H\_OF\_FIGURES** + **L**);  
 g2.drawLine(**x0** + **w** / 2, y + **H\_OF\_FIGURES** + **L**, (**int**) (**x0** + **w** / 2 - l\_arrow / Math.*sin*(alpha)), (**int**) (y + **H\_OF\_FIGURES** + **L** - l\_arrow / Math.*cos*(alpha)));  
 g2.drawLine(**x0** + **w** / 2, y + **H\_OF\_FIGURES** + **L**, (**int**) (**x0** + **w** / 2 + l\_arrow / Math.*sin*(alpha)), (**int**) (y + **H\_OF\_FIGURES** + **L** - l\_arrow / Math.*cos*(alpha)));  
 }  
 }  
  
 *//слово "end"* x = **x0** + **w** / 2 - **w\_oval** / 2;  
 y += **H\_OF\_FIGURES** + **L**;  
 g2.drawLine(x, y, x + **w\_oval**, y);  
 g2.drawLine(x, y + **H\_OF\_FIGURES**, x + **w\_oval**, y + **H\_OF\_FIGURES**);  
 g2.drawArc(x - **H\_OF\_FIGURES** / 2, y, **H\_OF\_FIGURES**, **H\_OF\_FIGURES**, 90, 180);  
 g2.drawArc(x + **w\_oval** - **H\_OF\_FIGURES** / 2, y, **H\_OF\_FIGURES**, **H\_OF\_FIGURES**, 90, -180);  
 g2.drawString(**END**, x + **w\_oval** / 2 - metrics.stringWidth(**END**) / 2, y + **H\_OF\_FIGURES** / 2 + metrics.getHeight() / 3);  
  
  
 }  
 **public void** setNFor (**int** n) {  
 **n\_for** = n;  
 }  
  
 **public** ArrayList<String> getList () {  
 **return list**;  
 }  
}

## ПРИЛОЖЕНИЕ Е Программная реализация класса Window

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.util.ArrayList;  
**import** java.util.regex.Pattern;

**public class** Window **implements** TypeOfCommands {  
 **private final int FIELD\_X** = 30;  
 **private int FIELD\_Y**;  
 **private final int FIELD\_W** = 6, **FIELD\_H** = 6;  
  
 **private final int PROGRAM\_FIELD\_X** = **FIELD\_X** + Field.***SQUARE\_SIZE*** \* 7 + 40 + 15;  
 **private int PROGRAM\_FIELD\_Y**;  
 **private int PROGRAM\_FIELD\_W**;  
 **private final int PROGRAM\_FIELD\_H** = **FIELD\_H** \* Field.***SQUARE\_SIZE***;  
  
 **private final int ROBOT\_X** = **FIELD\_X**;  
 **private int ROBOT\_Y**;  
 **private final int ROBOT\_W** = Field.***SQUARE\_SIZE***, **ROBOT\_H** = Field.***SQUARE\_SIZE*** + 10;  
  
 **private** Field **field**;  
 **private** Robot **robot**;  
 **private** ProgramField **programField**;  
 **private** Blocks **blocks**;  
  
 **private int n\_for**;  
 **private** String **signal**;  
 **private int ms** = 50;  
 **private int limit**;  
 **private int limit\_proc**;  
 **private int n\_commands** = 0;  
  
  
  
 *//список команд* **private** ArrayList<String> **commands** = **new** ArrayList<String>();  
 **private** ArrayList<String> **procedure** = **new** ArrayList<String>();  
 **private** JFrame **frame**;  
 **private** JPanel **panel**;  
 **private int num** = -1;  
 **private int num\_for\_proc** = -1;  
 **public final** Timer **timer** = **new** Timer(**ms**, **new** ActionListener() {  
 *//int i = 0;* **public void** actionPerformed(ActionEvent ev) {  
 Graphics g = **panel**.getGraphics();  
  
  
  
 **if** (**num** < **commands**.size() - 1 && **robot**.getX() + **ROBOT\_W** <= **FIELD\_X** + Field.***SQUARE\_SIZE*** \* **FIELD\_W** && **robot**.getY() + **ROBOT\_H** <= **FIELD\_Y** + Field.***SQUARE\_SIZE*** \* **FIELD\_H** && **robot**.getX() >= **FIELD\_X** && **robot**.getY() >= **ROBOT\_Y**) {  
  
 **if** (!**signal**.equals(***PROCALGS***)) {  
 **num**++;  
 **field**.DrawWay(**commands**.get(**num**));  
 **robot**.Move(**commands**.get(**num**));  
 }  
 **else** {  
 **if** (**num\_for\_proc** < **procedure**.size() - 1) {  
 **num\_for\_proc**++;  
 **field**.DrawWay(**procedure**.get(**num\_for\_proc**));  
 **robot**.Move(**procedure**.get(**num\_for\_proc**));  
 }  
 **else** {  
 **num\_for\_proc** = -1;  
 **num**++;  
 }  
  
 }  
  
 }  
 **else if** ((**num** < **commands**.size()) && (**robot**.getX() + **ROBOT\_W** > **FIELD\_X** + Field.***SQUARE\_SIZE*** \* **FIELD\_W** || **robot**.getY() + **ROBOT\_H** > **FIELD\_Y** + Field.***SQUARE\_SIZE*** \* **FIELD\_H** || **robot**.getX() < **FIELD\_X** || **robot**.getY() < **ROBOT\_Y**)) {  
  
 **int** pos = **num**;  
 **if** (**signal**.equals(***PROCALGS***))  
 pos++;  
**if** (!(**commands**.get(pos).equals(***JUMPLEFT***) || **commands**.get(pos).equals(***JUMPUP***) || **commands**.get(pos).equals(***JUMPRIGHT***) || **commands**.get(pos).equals(***JUMPDOWN***)  
 || **commands**.get(pos).equals(***PROC***))) {  
 pos = pos / (Field.***SQUARE\_SIZE*** / 5);  
  
 }  
  
 g.setColor(Color.***red***);  
 g.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 20));  
 FontMetrics metrics = g.getFontMetrics(g.getFont());  
 g.drawString(**"Ошибка! Выход за границы поля. Строка "** + (pos + 2), **PROGRAM\_FIELD\_X** + 10, **PROGRAM\_FIELD\_Y** + **PROGRAM\_FIELD\_H** + 15 + metrics.getHeight());  
 }  
 **else if** (**field**.AllCrossedCells()) {  
 g.setColor(Color.***red***);  
 g.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 20));  
 FontMetrics metrics = g.getFontMetrics(g.getFont());  
 g.drawString(**"Молодец!"**, **PROGRAM\_FIELD\_X** + 10, **PROGRAM\_FIELD\_Y** + **PROGRAM\_FIELD\_H** + 15 + metrics.getHeight());  
  
 }  
  
 }  
 });  
  
 **private** String **title**;  
  
 **private** ControlButton **GoDown**;  
 **private** ControlButton **GoRight**;  
 **private** ControlButton **GoUp**;  
 **private** ControlButton **GoLeft**;  
 **private** ControlButton **If**;  
 **private** ControlButton **Proc**;  
 **private** ControlButton **JumpDown**;  
 **private** ControlButton **JumpRight**;  
 **private** ControlButton **JumpUp**;  
 **private** ControlButton **JumpLeft**;  
 **private** ControlButton **Start**;  
 **private** ControlButton **Cancel**;  
 **private** ControlButton **CancelProc**;  
 **private** ControlButton **Return**;  
 **private** ControlButton **Help**;  
  
  
 **public** Window (JFrame frame, JPanel panel, **int** robot\_x, **int** robot\_y, String signal) {  
 **this**.**frame** = frame;  
 **this**.**panel** = panel;  
 **FIELD\_Y** = frame.getHeight() / 2 - **FIELD\_H** \* Field.***SQUARE\_SIZE*** / 2 - 10;  
 **PROGRAM\_FIELD\_W** = frame.getWidth() \* 22 / 100;  
 **PROGRAM\_FIELD\_Y** = **FIELD\_Y**;  
 **ROBOT\_Y** = **FIELD\_Y** - 14;  
  
  
 **field** = **new** Field(**FIELD\_X**, **FIELD\_Y**, **FIELD\_W**, **FIELD\_H**);  
 **robot** = **new** Robot(**ROBOT\_X** + robot\_x \* Field.***SQUARE\_SIZE***, **ROBOT\_Y** + robot\_y \* Field.***SQUARE\_SIZE***, **ROBOT\_W**, **ROBOT\_H**, frame);  
 **field**.setRobotCoord(**FIELD\_X** + robot\_x \* Field.***SQUARE\_SIZE***, **FIELD\_Y** + robot\_y \* Field.***SQUARE\_SIZE***);  
 **blocks** = **new** Blocks(**PROGRAM\_FIELD\_X** + **PROGRAM\_FIELD\_W** + 15, **PROGRAM\_FIELD\_Y**, **PROGRAM\_FIELD\_W**, **PROGRAM\_FIELD\_H**);  
 **this**.**signal** = signal;  
 **if** (signal.equals(***IFALGS***))  
 **ms** = 500;  
 **timer**.setDelay(**ms**);  
 MakeButtons();  
 **programField** = **new** ProgramField(**PROGRAM\_FIELD\_X**, **PROGRAM\_FIELD\_Y**, **PROGRAM\_FIELD\_W**, **PROGRAM\_FIELD\_H**);  
 **if** (signal.equals(***PROCALGS***))  
 **programField**.setIsProc(**true**);  
 **else  
 programField**.setIsProc(**false**);  
 }  
  
 **public void** Draw (Graphics g) {  
 Font font = **new** Font(**"Sreda"**, Font.***BOLD***, 45);  
 g.setFont(font);  
 FontMetrics metrics = g.getFontMetrics(font);  
 g.setColor(**new** Color(139, 9, 12));  
 g.drawString(**title**, **frame**.getWidth() / 2 - metrics.stringWidth(**title**) / 2, 50);  
 **field**.Draw(g);  
 **robot**.Draw(g);  
 **blocks**.Draw(g);  
  
 *//окаймовка program\_field* g.setColor(**new** Color(139, 9, 12));  
 g.fillRect(**PROGRAM\_FIELD\_X** - 5, **PROGRAM\_FIELD\_Y** - 5, **PROGRAM\_FIELD\_W** + 10, **PROGRAM\_FIELD\_H** + 10);  
  
 g.drawRect(**PROGRAM\_FIELD\_X**, **PROGRAM\_FIELD\_Y** + **PROGRAM\_FIELD\_H** + 15, **PROGRAM\_FIELD\_W** \* 2 + 15, 50);  
  
 }  
  
 **public void** setTitle (String title) {  
 **this**.**title** = title;  
 }  
  
 *//показать все формы на панели* **public void** ShowForms () {  
 **if** (**signal**.equals(***LINEARALGS***) || **signal**.equals(***FORALGS***) || **signal**.equals(***PROCALGS***)) {  
 **GoDown**.Show(**panel**);  
 **GoRight**.Show(**panel**);  
 **GoUp**.Show(**panel**);  
 **GoLeft**.Show(**panel**);  
 }  
 **if** (**signal**.equals(***IFALGS***)) {  
 **If**.Show(**panel**);  
 **JumpDown**.Show(**panel**);  
 **JumpRight**.Show(**panel**);  
 **JumpUp**.Show(**panel**);  
 **JumpLeft**.Show(**panel**);  
 }  
 **if** (**signal**.equals(***PROCALGS***)) {  
 **Proc**.Show(**panel**);  
 **CancelProc**.Show(**panel**);  
 }  
 **Start**.Show(**panel**);  
 **Cancel**.Show(**panel**);  
 **Return**.Show(**panel**);  
 **Help**.Show(**panel**);  
 **programField**.AddOnPanel(**panel**);  
 }  
  
 *//сделать кнопки* **private void** MakeButtons() {  
  
 **int** y = **FIELD\_Y**;  
 **int** w = Field.***SQUARE\_SIZE***, h = w;  
 **int** h\_y = 20;  
 **int** x = **FIELD\_X** + Field.***SQUARE\_SIZE*** \* **FIELD\_W** + h\_y + 15;  
 ControlButtonActionListener controlButtonActionListener = **new** ControlButtonActionListener();  
  
 *//кнопки "Идти" и "Прыгнуть" создаются в зависимости от надобности  
 //кнопка "Идти вниз"* **if** (!**signal**.equals(***IFALGS***)) {  
 **GoDown** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/GoDown.png"**), ***GODOWN***);  
 **GoDown**.getButton().addActionListener(controlButtonActionListener);  
 }  
 **else** {  
 **JumpDown** = **new** ControlButton(x, y, w, h, **null**, ***JUMPDOWN***);  
 **JumpDown**.getButton().addActionListener(controlButtonActionListener);  
 **JumpDown**.getButton().setEnabled(**false**);  
 }  
  
 *//кнопка "Идти вправо"* y = y + h + h\_y;  
 **if** (!**signal**.equals(***IFALGS***)) {  
 **GoRight** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/GoRight.png"**), ***GORIGHT***);  
 **GoRight**.getButton().addActionListener(controlButtonActionListener);  
 }  
 **else** {  
 **JumpRight** = **new** ControlButton(x, y, w, h, **null**, ***JUMPRIGHT***);  
 **JumpRight**.getButton().addActionListener(controlButtonActionListener);  
 **JumpRight**.getButton().setEnabled(**false**);  
 }  
  
 y = y + h + h\_y;  
 **if** (!**signal**.equals(***IFALGS***)) {  
 **GoUp** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/GoUp.png"**), ***GOUP***);  
 **GoUp**.getButton().addActionListener(controlButtonActionListener);  
 }  
 **else** {  
 **JumpUp** = **new** ControlButton(x, y, w, h, **null**, ***JUMPUP***);  
 **JumpUp**.getButton().addActionListener(controlButtonActionListener);  
 **JumpUp**.getButton().setEnabled(**false**);  
 }  
  
 y = y + h + h\_y;  
 **if** (!**signal**.equals(***IFALGS***)) {  
 **GoLeft** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/GoLeft.png"**), ***GOLEFT***);  
 **GoLeft**.getButton().addActionListener(controlButtonActionListener);  
 }  
 **else** {  
 **JumpLeft** = **new** ControlButton(x, y, w, h, **null**, ***JUMPLEFT***);  
 **JumpLeft**.getButton().addActionListener(controlButtonActionListener);  
 **JumpLeft**.getButton().setEnabled(**false**);  
 }  
  
 **if** (**signal**.equals(***IFALGS***)) {  
 y = y + h + h\_y;  
 **If** = **new** ControlButton(x, y, w, h, **null**, ***IF***);  
 **If**.getButton().addActionListener(controlButtonActionListener);  
 }  
  
 **if** (**signal**.equals(***FORALGS***)) {  
 y = y + h + h\_y;  
 JComboBox<String> cmb = **new** JComboBox<String>(**new** String[]{**" n:= "**, **"2"**, **"3"**, **"4"**, **"5"**});  
 cmb.addActionListener(**new** CMBActionListener());  
 cmb.setBounds(x, y + h / 2 - h / 6, w, h / 3);  
 **panel**.add(cmb);  
 }  
  
 **if** (**signal**.equals(***PROCALGS***)) {  
 y = y + h + h\_y;  
 **Proc** = **new** ControlButton(x, y, w, h, **null**, ***PROC***);  
 **Proc**.getButton().addActionListener(controlButtonActionListener);  
 }  
  
 w = w + h\_y;  
 h = 50;  
 x = **FIELD\_X**;  
 y = **FIELD\_Y** + **FIELD\_H** \* Field.***SQUARE\_SIZE*** + h\_y + 15;  
 **Start** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/start.png"**), ***START***);  
 **Start**.getButton().addActionListener(controlButtonActionListener);  
  
 x = x + w + 20;  
 **Cancel** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/cancel.png"**), ***CANCEL***);  
 **Cancel**.getButton().addActionListener(controlButtonActionListener);  
  
 **if** (**signal**.equals(***PROCALGS***)) {  
 x = x + w + 20;  
 **CancelProc** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/cancel\_proc.png"**), ***CANCELPROC***);  
 **CancelProc**.getButton().addActionListener(controlButtonActionListener);  
 }  
  
 x = x + w + 20;  
 **Return** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/return.png"**), ***RETURN***);  
 **Return**.getButton().addActionListener(controlButtonActionListener);  
  
 x = x + w + 20;  
 **Help** = **new** ControlButton(x, y, w, h, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/help.png"**), ***HELP***);  
 **Help**.getButton().addActionListener(controlButtonActionListener);  
 }  
  
 *//слушатель кнопок* **private class** ControlButtonActionListener **implements** ActionListener {  
 **private int n\_commands\_proc** = 0;  
 **public void** actionPerformed(ActionEvent ae) {  
 String command = ae.getActionCommand();  
 *//регулярное выражения для циклических блоков* String regex = **"I = 1,\\d,1"**;  
 Pattern p = Pattern.*compile*(regex);  
 **timer**.stop();  
  
 **if** (command.equals(***GODOWN***) || command.equals(***GORIGHT***) || command.equals(***GOUP***) || command.equals(***GOLEFT***)) {  
 **if** (**signal**.equals(***PROCALGS***)) {  
 **if** (**n\_commands\_proc** < **limit\_proc**) {  
 **programField**.AddTextToProc(command);  
 **for** (**int** i = 0; i < Field.***SQUARE\_SIZE*** / 5; i++)  
 **procedure**.add(command);  
 **n\_commands\_proc**++;  
 }  
 }  
 **else** {  
 **if** (**n\_commands** < **limit**) {  
 **if** (**commands**.size() > 0 && **commands**.get(**commands**.size() - 1).equals(***FOR***))  
 **programField**.AddText(**" "** + command);  
 **else  
 programField**.AddText(command);  
 **if** (**commands**.size() > 0 && **commands**.get(**commands**.size() - 1).equals(***FOR***))  
 **for** (**int** j = 0; j < **n\_for**; j++)  
 **for** (**int** i = 0; i < Field.***SQUARE\_SIZE*** / 5; i++)  
 **commands**.add(command);  
 **else  
 for** (**int** i = 0; i < Field.***SQUARE\_SIZE*** / 5; i++)  
 **commands**.add(command);  
 **blocks**.AddCommand(command);  
 **n\_commands**++;  
 }  
 }  
 **frame**.repaint();  
 }  
  
 **if** (command.equals(***IF***)) {  
 **if** (**n\_commands** < **limit**) {  
 **programField**.AddText(command);  
 **commands**.add(command);  
 **blocks**.AddCommand(command);  
 **JumpDown**.getButton().setEnabled(**true**);  
 **JumpRight**.getButton().setEnabled(**true**);  
 **JumpUp**.getButton().setEnabled(**true**);  
 **JumpLeft**.getButton().setEnabled(**true**);  
 **n\_commands**++;  
 **frame**.repaint();  
 }  
 }  
 **if** (command.equals(***JUMPDOWN***) || command.equals(***JUMPRIGHT***) || command.equals(***JUMPUP***) || command.equals(***JUMPLEFT***)) {  
 **if** (**n\_commands** < **limit**) {  
 **programField**.AddText(command);  
 **blocks**.AddCommand(command);  
 **commands**.add(command);  
 **JumpDown**.getButton().setEnabled(**false**);  
 **JumpRight**.getButton().setEnabled(**false**);  
 **JumpUp**.getButton().setEnabled(**false**);  
 **JumpLeft**.getButton().setEnabled(**false**);  
 **n\_commands**++;  
 **frame**.repaint();  
 }  
 }  
  
 **if** (command.equals(***PROC***)) {  
 **if** (**n\_commands** < **limit**) {  
 **programField**.AddText(command);  
 **blocks**.AddCommand(command);  
 **commands**.add(command);  
 **n\_commands**++;  
 **frame**.repaint();  
 }  
 }  
 **if** (command.equals(***START***)) {  
 **timer**.start();  
 }  
 **if** (command.equals(***CANCEL***)) {  
 **if** (**commands**.size() > 0) {  
 *//если это было ветвление, то удаляем вместе с ветвлением* **if** (**commands**.get(**commands**.size() - 1).equals(***JUMPLEFT***) || **commands**.get(**commands**.size() - 1).equals(***JUMPUP***) ||  
 **commands**.get(**commands**.size() - 1).equals(***JUMPRIGHT***) || **commands**.get(**commands**.size() - 1).equals(***JUMPDOWN***)  
 ) {  
 **commands**.remove(**commands**.size() - 1);  
 **commands**.remove(**commands**.size() - 1);  
 **programField**.RemoveString(); *//удаляем действие и ветвление* **programField**.RemoveString();  
 **blocks**.RemoveCommand();  
 **blocks**.RemoveCommand();  
 **if** (**n\_commands** - 2 >= 0)  
 **n\_commands** -= 2;  
 }  
 *//если был оператор* **else if** (**commands**.get(**commands**.size() - 1).equals(***IF***) || **commands**.get(**commands**.size() - 1).equals(***FOR***)  
 || **commands**.get(**commands**.size() - 1).equals(***PROC***)) {  
 **if** (**commands**.get(**commands**.size() - 1).equals(***IF***)) {  
 **JumpDown**.getButton().setEnabled(**false**);  
 **JumpRight**.getButton().setEnabled(**false**);  
 **JumpUp**.getButton().setEnabled(**false**);  
 **JumpLeft**.getButton().setEnabled(**false**);  
 }  
 **commands**.remove(**commands**.size() - 1);  
 **programField**.RemoveString();  
 **blocks**.RemoveCommand();  
 **if** (**n\_commands** > 0)  
 **n\_commands**--;  
 }  
 **else** {  
 *//циклический оператор* **if** (**blocks**.getList().size() > 1 && p.matcher(**blocks**.getList().get(**blocks**.getList().size() - 2)).matches()) {  
 **while** (!**commands**.get(**commands**.size() - 1).equals(***FOR***))  
 **commands**.remove(**commands**.size() - 1);  
 **commands**.remove(**commands**.size() - 1);  
 **programField**.RemoveString();  
 **blocks**.RemoveCommand();  
 **programField**.RemoveString();  
 **blocks**.RemoveCommand();  
 **if** (**n\_commands** - 2 >= 0)  
 **n\_commands** -= 2;  
 }  
 **else** {  
 **for** (**int** i = 0; i < Field.***SQUARE\_SIZE*** / 5; i++) {  
 **commands**.remove(**commands**.size() - 1);  
  
 }  
 **programField**.RemoveString();  
 **blocks**.RemoveCommand();  
 **if** (**n\_commands** > 0)  
 **n\_commands**--;  
 }  
 }  
  
 **frame**.repaint();  
 }  
 }  
  
 **if** (command.equals(***RETURN***)) {  
 **field**.ClearWay();  
 **robot**.setInitialCoords();  
 **num** = -1;  
 **num\_for\_proc** = -1;  
 }  
 **if** (command.equals(***CANCELPROC***)) {  
 **if** (**procedure**.size() > 0) {  
 **for** (**int** i = 0; i < Field.***SQUARE\_SIZE*** / 5; i++) {  
 **procedure**.remove(**procedure**.size() - 1);  
 }  
 **programField**.RemoveStringFromProc();  
 **if** (**n\_commands\_proc** > 0)  
 **n\_commands\_proc**--;  
 }  
 }  
 **if** (command.equals(***HELP***)) {  
 **final** JFrame fr = **new** JFrame(**"Помощь"**);  
 fr.setSize(1000, 700);  
 fr.setResizable(**false**);  
  
 JPanel pan = **new** JPanel() {  
 **public void** paintComponent (Graphics g) {  
 **super**.paintComponent(g);  
 Image background = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/help\_window1.png"**);  
 **if** (**signal**.equals(***IFALGS***))  
 background = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/help\_window2.png"**);  
 **if** (**signal**.equals(***FORALGS***))  
 background = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/help\_window3.png"**);  
 **if** (**signal**.equals(***PROCALGS***))  
 background = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/help\_window4.png"**);  
 g.drawImage(background, 0, 0, **this**);  
  
 }  
 };  
 fr.add(pan);  
 fr.setVisible(**true**);  
 }  
  
 }  
 }  
  
 **public** Field getField () {  
 **return field**;  
 }  
  
 **private class** CMBActionListener **implements** ActionListener {  
 **public void** actionPerformed (ActionEvent ae) {  
 JComboBox box = (JComboBox)ae.getSource();  
 String item = (String)box.getSelectedItem();  
 **try** {  
 **if** (**n\_commands** < **limit**) {  
 **n\_for** = Integer.*parseInt*(item);  
 **programField**.setNFor(**n\_for**);  
 **blocks**.setNFor(**n\_for**);  
  
 **programField**.AddText(***FOR***);  
 **commands**.add(***FOR***);  
 **blocks**.AddCommand(***FOR***);  
 **n\_commands**++;  
 **frame**.repaint();  
 }  
 }  
 **catch** (Exception e) {  
 }  
 }  
 }  
  
 **public void** setLimit (**int** limit) {  
 **this**.**limit** = limit;  
 }  
  
 **public void** setLimitProc (**int** limit\_proc) {  
 **this**.**limit\_proc** = limit\_proc;  
 }  
  
}

## ПРИЛОЖЕНИЕ Ж Программная реализация класса Title

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**public class** Title **implements** TypeOfCommands {  
 **private** JFrame **frame**;  
 **private** JPanel **panel**;  
 **private boolean isLinearAlgs** = **false**;  
 **private boolean isIfAlgs** = **false**;  
 **private boolean isForAlgs** = **false**;  
 **private boolean isProcAlgs** = **false**;  
 **private boolean isControlAlgs** = **false**;  
 **private boolean isTitle** = **true**;  
 **private boolean isWork** = **false**;  
 **private** Buttons **linear\_alg**; *//кнопка "Линейные алгоритмы"* **private** Buttons **if\_alg**; *//кнопка "Условные алгоритмы"* **private** Buttons **for\_alg**; *//кнопка "Циклические алгоритмы"* **private** Buttons **proc\_alg**; *//кнопка "Подпрограммы"* **private** Buttons **control\_alg**; *//кнопка "Контрольные задания"* **private** Buttons **back**; *//кнопка "Назад"* **private** Buttons **first**; *//кнопка первого задания* **private** Buttons **second**; *//кнопка второго задания* **private** Buttons **third**; *//кнопка третьего задания* **private** Buttons **forth**; *//кнопка четвертого задания* **private** Window **window**;  
 **private int x\_on\_title**;  
 **private int y\_on\_title**;  
  
  
  
 **public** Title (JFrame frame) {  
 **this**.**frame** = frame;  
 MakeButtons();  
  
 }  
  
 **public void** Draw (Graphics g) {  
 **int** robot\_x = 90;  
 **int** title\_y = 70; *//Y заголовка* Image robot = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/robot.png"**);  
 Image phrase;  
 String s;  
 *//главная страница* **if** (**isTitle**) {  
 **panel**.removeAll();  
 Font font = **new** Font(**"Sreda"**, Font.***BOLD***, 45);  
 g.setFont(font);  
 FontMetrics metrics = g.getFontMetrics(font);  
 s = **"Алгоритмический игровой исполнитель"**;  
 g.setColor(**new** Color(139, 9, 12)*/\*new Color(85, 26, 139)\*/*);  
 g.drawString(s, **frame**.getWidth() / 2 - metrics.stringWidth(s) / 2, title\_y);  
 s = (**char**) 171 + **"Робот"** + (**char**) 187;  
 g.drawString(s, **frame**.getWidth() / 2 - metrics.stringWidth(s) / 2, title\_y + metrics.getHeight() + 10);  
  
 phrase = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/phrase1.png"**);  
 **x\_on\_title** = **frame**.getWidth() / 2 - (**int**)(**frame**.getWidth() \* 0.065886);  
 **y\_on\_title** = (**int**)(**frame**.getHeight() \* 0.57);  
 g.drawImage(robot, **frame**.getWidth() / 2 - robot\_x / 2, **y\_on\_title**, robot\_x, 100, **panel**); *//90* g.drawImage(phrase, **frame**.getWidth() / 2 - robot\_x / 2 - robot\_x, **y\_on\_title** - phrase.getHeight(**null**), **panel**); *//130* **int** h\_button = 100;  
 **int** h\_coord = 70;  
 **int** x = (**int**)(**x\_on\_title** \* 0.15);  
 **int** y0 = (**int**)(**y\_on\_title** \* 0.55);  
 **int** y = y0;  
 **int** w\_button = (**int**)(**frame**.getWidth() \* 0.2196);  
  
 **linear\_alg**.getButton().setBounds(x, y, w\_button, h\_button);  
 **int** h\_rect = 7;  
 g.fillRect(x + h\_rect, y + h\_rect, w\_button, h\_button);  
 **linear\_alg**.AddOnPanel();  
  
 Image leaf = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/leaf.png"**);  
 g.drawImage(leaf, x + w\_button / 3, y - 20 - 100, 100, 100, **panel**);  
  
 y = y + h\_coord + h\_button;  
 **if\_alg**.getButton().setBounds(x + 30, y, w\_button, h\_button);  
 g.fillRect(x + 30 + h\_rect, y + h\_rect, w\_button, h\_button);  
 **if\_alg**.AddOnPanel();  
  
 Image computer = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/computer.jpg"**);  
 g.drawImage(computer, x + w\_button / 3, y + h\_coord + h\_button, 120, 120, **panel**);  
  
 x = **frame**.getWidth() - x - w\_button;  
 y = y0;  
 **for\_alg**.getButton().setBounds(x, y, w\_button, h\_button);  
 g.fillRect(x + h\_rect, y + h\_rect, w\_button, h\_button);  
 **for\_alg**.AddOnPanel();  
  
 Image block = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/block.jpg"**);  
 g.drawImage(block, x + w\_button / 3, y - 15 - 100, 100, 100, **panel**);  
  
 y = y + h\_button + h\_coord;  
 **proc\_alg**.getButton().setBounds(x - 30, y, w\_button, h\_button);  
 g.fillRect(x - 30 + h\_rect, y + h\_rect, w\_button, h\_button);  
 **proc\_alg**.AddOnPanel();  
  
 g.drawImage(leaf, x + w\_button / 3, y + 20 + 100, 100, 100, **panel**);  
  
 x = x - (x - (**int**)(**x\_on\_title** \* 0.15) - w\_button) / 2 - w\_button / 2;  
 y = y + h\_button + h\_button / 2;  
 **control\_alg**.getButton().setBounds(x, y, w\_button, h\_button);  
 g.fillRect(x + h\_rect, y + h\_rect, w\_button, h\_button);  
 **control\_alg**.AddOnPanel();  
 }  
  
 *//страница с линейными алгоритмами* **if** (**isLinearAlgs**) {  
 **if** (**isWork**) {  
 **window**.Draw(g);  
 **back**.AddOnPanel();  
 }  
 **else** {  
 MakeTitle(g, **"Линейные алгоритмы"**, **"Линейный алгоритм - это алгоритм, в котором действия выполняются строго последовательно"**, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/linear\_block.png"**),  
 Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/linear\_prog.png"**), Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/linear.jpg"**), **""**);  
 }  
 }  
  
 **if** (**isIfAlgs**) {  
 **if** (**isWork**) {  
 **window**.Draw(g);  
 **back**.AddOnPanel();  
 }  
 **else** {  
 MakeTitle(g, **"Алгоритмы с ветвлением"**, **"Алгоритм с ветвлением - это алгоритм, в котором в зависимости&от условия выполняется то, либо иное действие"**, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/if\_block.png"**),  
 Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/if\_prog.png"**), Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/if.jpg"**),  
 **"Если условие ветвления истинно, то выполняется Действие 1, в противном случае выполняется Действие 2"**);  
 }  
 }  
  
 **if** (**isForAlgs**) {  
 **if** (**isWork**) {  
 **window**.Draw(g);  
 **back**.AddOnPanel();  
 }  
 **else** {  
 MakeTitle(g, **"Циклические алгоритмы"**, **"Циклический алгоритм - это алгоритм, в котором действия выполняются&определенное количество раз"**, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/for\_block.png"**),  
 Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/for\_prog.png"**), Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/for.jpg"**),  
 **"На рисунке видно, что Действие будет выполняться n раз, т.е. пока условие i <= n истинно."**);  
 }  
 }  
 **if** (**isProcAlgs**) {  
 **if** (**isWork**) {  
 **window**.Draw(g);  
 **back**.AddOnPanel();  
 }  
 **else** {  
 MakeTitle(g, ***PROCALGS***, **"При решении некоторых задач бывает удобно разбить их на подзадачи,&каждую из которых можно оформить как самостоятельный алгоритм."**, Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/proc\_block.png"**),  
 Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/proc\_prog.png"**), Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/linear.jpg"**),  
 **"Каждый такой алгоритм называется подпрограммой (процедурой). Процедуру в основной программе можно вызывать неоднократно"**);  
 }  
 }  
  
 **if** (**isControlAlgs**) {  
 **if** (**isWork**) {  
 **window**.Draw(g);  
 **back**.AddOnPanel();  
 }  
 **else** {  
 MakeTitle(g, **"Контрольные задания"**, **"В этом разделе приведены контрольные задания по всему курсу алгоритмизации"**,  
 Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/control\_table.png"**), **null**, **null**, **""**);  
 }  
 }  
  
 }  
  
 **private void** MakeTitle (Graphics g, String title, String text, Image block, Image prog, Image img, String theory) {  
 **panel**.removeAll();  
 **int** robot\_x = 90;  
 **int** title\_y = 70;  
 Image robot = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/robot.png"**);  
 Font font = **new** Font(**"Sreda"**, Font.***BOLD***, 45);  
 g.setFont(font);  
 g.setColor(**new** Color(139, 9, 12));  
 FontMetrics metrics = g.getFontMetrics(font);  
 g.drawString(title, **frame**.getWidth() / 2 - metrics.stringWidth(title) / 2, title\_y);  
  
  
 font = **new** Font(**"Arial"**, Font.***BOLD***, 25);  
 g.setFont(font);  
 metrics = g.getFontMetrics(font);  
 **int** pos = text.indexOf(**"&"**);  
 String text1 = **""**;  
 *//если строку надо разбить* **if** (pos > 0) {  
 text1 = text.substring(pos + 1, text.length());  
 text = text.substring(0, pos);  
 }  
 **if** (!title.equals(**"Контрольные задания"**)) {  
 g.drawString(text, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), title\_y + metrics.getHeight() + metrics.getHeight() / 2);  
 g.drawString(text1, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), title\_y + 2 \* metrics.getHeight() + metrics.getHeight() / 2);  
 }  
 **else** {  
 g.drawString(text, **frame**.getWidth() / 2 - metrics.stringWidth(text) / 2, title\_y + metrics.getHeight() + metrics.getHeight() / 2);  
 g.drawString(text1, **frame**.getWidth() / 2 - metrics.stringWidth(text) / 2, title\_y + 2 \* metrics.getHeight() + metrics.getHeight() / 2);  
 }  
  
  
 g.setColor(Color.***BLACK***);  
 g.setFont(**new** Font(**"Comic Sans MS"**, Font.***PLAIN***, 30));  
 **if** (!title.equals(**"Контрольные задания"**)) {  
 g.drawImage(robot, Toolkit.*getDefaultToolkit*().getScreenSize().**width** - robot\_x \* 3, **frame**.getHeight() - 100 - 50, robot\_x, 100, **panel**);  
 Image phrase = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/phrase2.png"**);  
 g.drawImage(phrase, Toolkit.*getDefaultToolkit*().getScreenSize().**width** - robot\_x \* 4, **frame**.getHeight() - 100 - 50 - phrase.getHeight(**null**), **panel**);  
 }  
  
 **if** (!theory.equals(**""**)) {  
 **if** (prog != **null**)  
 g.drawImage(prog, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), **frame**.getHeight() / 2 - prog.getHeight(**null**) / 2 - 50, **panel**);  
 **if** (block != **null**)  
 g.drawImage(block, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2, **frame**.getHeight() / 2 - block.getHeight(**null**) / 2 - 50, **panel**);  
 }  
 **else** {  
 **if** (prog != **null**)  
 g.drawImage(prog, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), **frame**.getHeight() / 2 - prog.getHeight(**null**) / 2, **panel**);  
 **if** (block != **null**)  
 g.drawImage(block, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2, **frame**.getHeight() / 2 - block.getHeight(**null**) / 2, **panel**);  
 }  
  
 **if** (img != **null**)  
 g.drawImage(img, Toolkit.*getDefaultToolkit*().getScreenSize().**width** - robot\_x \* 3, title\_y + 3 \* metrics.getHeight(), **panel**);  
  
 font = **new** Font(**"Arial"**, Font.***BOLD***, 25);  
 g.setFont(font);  
 metrics = **panel**.getFontMetrics(font);  
  
 text1 = **""**;  
 String text2 = **""**;  
  
 *//если строку надо разбить* **if** (!theory.equals(**""**)) {  
 String[] words = theory.split(**" "**);  
 **int** i = 0;  
 **while** (i < words.**length** && metrics.stringWidth(text1 + words[i] + **" "**) < **frame**.getWidth() / 2 + block.getWidth(**null**) / 2) {  
 text1 = text1 + words[i] + **" "**;  
 i++;  
 }  
 theory = **""**;  
 **if** (i < words.**length**)  
 **while** (i < words.**length** && metrics.stringWidth(theory + words[i] + **" "**) < **frame**.getWidth() / 2 + block.getWidth(**null**) / 2) {  
 theory = theory + words[i] + **" "**;  
 i++;  
 }  
  
 **for** (**int** j = i; j < words.**length**; j++)  
 text2 = text2 + words[j] + **" "**;  
  
  
 }  
 **if** (prog != **null**) {  
 g.drawString(text1, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), **frame**.getHeight() / 2 + prog.getHeight(**null**) / 2 + 25);  
 g.drawString(theory, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), **frame**.getHeight() / 2 + prog.getHeight(**null**) / 2 + 25 + metrics.getHeight());  
 g.drawString(text2, **frame**.getWidth() / 2 - block.getWidth(**null**) / 2 - 50 - prog.getWidth(**null**), **frame**.getHeight() / 2 + prog.getHeight(**null**) / 2 + 25 + 2 \* metrics.getHeight());  
 }  
  
  
  
 g.setColor(**new** Color(139, 9, 12));  
 **int** h\_rect = 5;  
 **back**.AddOnPanel();  
 **first**.AddOnPanel();  
 g.fillRect(**first**.getButton().getX() + h\_rect, **first**.getButton().getY() + h\_rect, **first**.getButton().getWidth(), **first**.getButton().getHeight());  
 **second**.AddOnPanel();  
 g.fillRect(**second**.getButton().getX() + h\_rect, **second**.getButton().getY() + h\_rect, **second**.getButton().getWidth(), **second**.getButton().getHeight());  
 **third**.AddOnPanel();  
 g.fillRect(**third**.getButton().getX() + h\_rect, **third**.getButton().getY() + h\_rect, **third**.getButton().getWidth(), **third**.getButton().getHeight());  
 **forth**.AddOnPanel();  
 g.fillRect(**forth**.getButton().getX() + h\_rect, **forth**.getButton().getY() + h\_rect, **forth**.getButton().getWidth(), **forth**.getButton().getHeight());  
 }  
  
 **public void** setPanel (JPanel panel) {  
 **this**.**panel** = panel;  
 }  
  
  
 **private class** ButtonActionListener **implements** ActionListener {  
 **public void** actionPerformed(ActionEvent ae) {  
 String command = ae.getActionCommand();  
 **if** (command.equals(***LINEARALGS***)) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **isLinearAlgs** = **true**;  
 **isTitle** = **false**;  
 **isForAlgs** = **false**;  
 **isIfAlgs** = **false**;  
 **isProcAlgs** = **false**;  
 **isControlAlgs** = **false**;  
 **isWork** = **false**;  
 **frame**.repaint();  
  
 }  
 **if** (command.equals(***IFALGS***)) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **isLinearAlgs** = **false**;  
 **isTitle** = **false**;  
 **isForAlgs** = **false**;  
 **isIfAlgs** = **true**;  
 **isProcAlgs** = **false**;  
 **isControlAlgs** = **false**;  
 **isWork** = **false**;  
 **frame**.repaint();  
  
 }  
 **if** (command.equals(***FORALGS***)) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **isLinearAlgs** = **false**;  
 **isTitle** = **false**;  
 **isForAlgs** = **true**;  
 **isIfAlgs** = **false**;  
 **isProcAlgs** = **false**;  
 **isControlAlgs** = **false**;  
 **isWork** = **false**;  
 **frame**.repaint();  
  
 }  
 **if** (command.equals(***PROCALGS***)) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **isLinearAlgs** = **false**;  
 **isTitle** = **false**;  
 **isForAlgs** = **false**;  
 **isIfAlgs** = **false**;  
 **isProcAlgs** = **true**;  
 **isControlAlgs** = **false**;  
 **isWork** = **false**;  
 **frame**.repaint();  
  
 }  
 **if** (command.equals(***CONTROLALG***)) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **isLinearAlgs** = **false**;  
 **isTitle** = **false**;  
 **isForAlgs** = **false**;  
 **isIfAlgs** = **false**;  
 **isProcAlgs** = **false**;  
 **isControlAlgs** = **true**;  
 **isWork** = **false**;  
 **frame**.repaint();  
  
 }  
 **if** (command.equals(***BACK***)) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **if** (**isWork**)  
 **isWork** = **false**;  
 **else** {  
 **isLinearAlgs** = **false**;  
 **isTitle** = **true**;  
 **isForAlgs** = **false**;  
 **isIfAlgs** = **false**;  
 **isProcAlgs** = **false**;  
 **isControlAlgs** = **false**;  
 }  
 **frame**.repaint();  
 }  
 **if** (command.equals(***FIRST***)) {  
 **if** (**isLinearAlgs**) {  
 **int**[] col = **new int**[] {1, 2, 3, 4};  
 **int**[] row = **new int**[] {0, 0, 0, 0};  
 MakeWork(1, 1, row, col, ***LINEARALGS***, **"Самое простое задание"**, 4);  
 }  
 **if** (**isIfAlgs**) {  
 **int**[] col = **new int**[] {3, 3, 3, 5};  
 **int**[] row = **new int**[] {1, 3, 5, 5};  
 MakeWork(1, 1, row, col, ***IFALGS***, **"Простой путь"**, 8);  
 col = **new int**[] {2, 3, 3, 4};  
 row = **new int**[] {1, 2, 4, 5};  
 **window**.getField().setBorderRowCol(row, col);  
 }  
 **if** (**isForAlgs**) {  
 **int**[] col = **new int**[] {1, 2, 3, 4, 5};  
 **int**[] row = **new int**[] {1, 1, 1, 1, 1};  
 MakeWork(0, 1, row, col, ***FORALGS***, **"Прямая"**, 2);  
 }  
 **if** (**isProcAlgs**) {  
 **int**[] col = **new int**[] {1, 2, 2, 2, 3, 4, 4, 4};  
 **int**[] row = **new int**[] {0, 0, 1, 2, 2, 2, 3, 4};  
 MakeWork(0, 0, row, col, ***PROCALGS***, **"Серпантин"**, 2);  
 **window**.setLimitProc(4);  
 }  
 **if** (**isControlAlgs**) {  
 **int**[] col = **new int**[] {1, 1, 1, 2, 3, 3, 3, 3};  
 **int**[] row = **new int**[] {4, 3, 2, 2, 2, 3, 4, 5};  
 MakeWork(1, 5, row, col, ***LINEARALGS***, **"Буква П"**, 8);  
 }  
 **frame**.repaint();  
 }  
 **if** (command.equals(***SECOND***)) {  
 **if** (**isLinearAlgs**) {  
 **int**[] col = **new int**[]{3, 3, 2, 4};  
 **int**[] row = **new int**[]{2, 4, 3, 3};  
 MakeWork(3, 3, row, col, ***LINEARALGS***, **"Крестик"**, 7);  
  
 }  
 **if** (**isIfAlgs**) {  
 **int**[] col = **new int**[] {2, 4, 0};  
 **int**[] row = **new int**[] {1, 3, 3};  
 MakeWork(2, 3, row, col, ***IFALGS***, **"Вправо, влево, вверх"**, 10);  
 col = **new int**[] {2, 3, 1};  
 row = **new int**[] {2, 3, 3};  
 **window**.getField().setBorderRowCol(row, col);  
 }  
 **if** (**isForAlgs**) {  
 **int**[] col = **new int**[]{2, 3, 4, 4, 4, 3, 2, 1};  
 **int**[] row = **new int**[]{1, 1, 1, 2, 3, 3, 3, 3};  
 MakeWork(1, 1, row, col, ***FORALGS***, **"Подкова"**, 6);  
 }  
 **if** (**isProcAlgs**) {  
 **int**[] col = **new int**[] {2, 3, 3, 3, 2, 1, 3, 3, 2, 1};  
 **int**[] row = **new int**[] {0, 0, 1, 2, 2, 2, 3, 4, 4, 4};  
 MakeWork(1, 0, row, col, ***PROCALGS***, **"Цифра 3"**, 2);  
 **window**.setLimitProc(6);  
 }  
 **if** (**isControlAlgs**) {  
 **int**[] col = **new int**[] {2, 2, 4, 4};  
 **int**[] row = **new int**[] {0, 2, 2, 4};  
 MakeWork(0, 0, row, col, ***IFALGS***, **"Сепрантин с if"**, 8);  
 col = **new int**[] {1, 2, 3, 4};  
 row = **new int**[] {0, 1, 2, 3};  
 **window**.getField().setBorderRowCol(row, col);  
 }  
 **frame**.repaint();  
 }  
 **if** (command.equals(***THIRD***)) {  
 **if** (**isLinearAlgs**) {  
 **int**[] col = **new int**[]{2, 3, 3, 3, 2, 1, 1};  
 **int**[] row = **new int**[]{1, 1, 2, 3, 3, 3, 2};  
 MakeWork(1, 1, row, col, ***LINEARALGS***, **"Квадрат"**, 8);  
 }  
 **if** (**isIfAlgs**) {  
 **int**[] col = **new int**[] {0, 2, 2, 4, 4};  
 **int**[] row = **new int**[] {3, 3, 1, 1, 3};  
 MakeWork(0, 1, row, col, ***IFALGS***, **"Змейка"**, 10);  
 col = **new int**[] {0, 1, 2, 3, 4};  
 row = **new int**[] {2, 3, 2, 1, 2};  
 **window**.getField().setBorderRowCol(row, col);  
 }  
 **if** (**isForAlgs**) {  
 **int**[] col = **new int**[]{2, 2, 2, 2, 3, 4, 4};  
 **int**[] row = **new int**[]{2, 3, 4, 5, 4, 4, 5};  
 MakeWork(2, 1, row, col, ***FORALGS***, **"Буква H"**, 6);  
 }  
 **if** (**isProcAlgs**) {  
 **int**[] col = **new int**[] {0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4};  
 **int**[] row = **new int**[] {0, 0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5};  
 MakeWork(0, 1, row, col, ***PROCALGS***, **"По кочкам с возвратом"**, 4);  
 **window**.setLimitProc(4);  
 }  
 **if** (**isControlAlgs**) {  
 **int**[] col = **new int**[]{1, 2, 3, 4, 5, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0, 0, 0, 0, 0};  
 **int**[] row = **new int**[]{0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 5, 4, 3, 2, 1};  
 MakeWork(0, 0, row, col, ***FORALGS***, **"Обойти поле"**, 8);  
 }  
 **frame**.repaint();  
 }  
 **if** (command.equals(***FORTH***)) {  
 **if** (**isLinearAlgs**) {  
 **int**[] col = **new int**[]{0, 1, 1, 2, 2, 3, 3, 4, 4};  
 **int**[] row = **new int**[]{1, 1, 2, 2, 3, 3, 4, 4, 5};  
 MakeWork(0, 0, row, col, ***LINEARALGS***, **"Лесенка"**, 9);  
 }  
 **if** (**isIfAlgs**) {  
 **int**[] col = **new int**[] {3, 3, 3, 1, 1};  
 **int**[] row = **new int**[] {1, 3, 5, 5, 3};  
 MakeWork(1, 1, row, col, ***IFALGS***, **"Прямоугольник"** ,10);  
 col = **new int**[] {2, 3, 3, 2, 1, 1};  
 row = **new int**[] {1, 2, 4, 5, 4, 2};  
 **window**.getField().setBorderRowCol(row, col);  
 }  
 **if** (**isForAlgs**) {  
 **int**[] col = **new int**[]{1, 2, 3, 4, 5, 5, 5, 4, 3, 2, 1, 0, 0, 0, 1, 2, 3, 4, 5};  
 **int**[] row = **new int**[]{0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 3, 4, 4, 4, 4, 4, 4};  
 MakeWork(0, 0, row, col, ***FORALGS***, **"Лабиринт"**, 10);  
 }  
 **if** (**isProcAlgs**) {  
 **int**[] col = **new int**[] {1, 2, 3, 4};  
 **int**[] row = **new int**[] {3, 2, 3, 2};  
 MakeWork(0, 2, row, col, ***PROCALGS***, **"Разметка"**, 2);  
 **window**.setLimitProc(4);  
 }  
 **if** (**isControlAlgs**) {  
 **int**[] col = **new int**[] {3, 3, 2, 2, 2, 1};  
 **int**[] row = **new int**[] {1, 2, 2, 3, 4, 4};  
 MakeWork(3, 0, row, col, ***PROCALGS***, **"Вниз по горе"**, 2);  
 **window**.setLimitProc(3);  
 }  
 **frame**.repaint();  
 }  
  
 }  
 }  
  
 **private void** MakeWork (**int** robot\_x, **int** robot\_y, **int**[] r, **int**[] c, String signal, String title, **int** limit) {  
 **if** (**window** != **null**)  
 **window**.**timer**.stop();  
 **isWork** = **true**;  
 **panel**.removeAll();  
 **window** = **new** Window(**frame**, **panel**, robot\_x, robot\_y, signal);  
 **window**.setTitle(title);  
 **window**.ShowForms();  
 **window**.getField().setNecessaryRowCol(r, c);  
 **window**.setLimit(limit);  
  
 }  
  
 **private void** MakeButtons() {  
 ButtonActionListener actionListener = **new** ButtonActionListener();  
 UIManager.*put*(**"Button.background"**, **new** Color(250, 174, 41));  
 UIManager.*put*(**"Button.font"**, **new** Font(**"Arial"**, Font.***BOLD***, 15));  
 UIManager.*put*(**"Button.foreground"**, **new** Color(139, 9, 12));  
  
 **int** w = Field.***SQUARE\_SIZE***, h = 50;  
 **int** x = (**int**)(**x\_on\_title** \* 0.25), y = 250;  
**int** y0 = y;  
 **int** h\_coord = 70;  
  
 **linear\_alg** = **new** Buttons(***LINEARALGS***, **"src/com/company/linear.png"**, x, y, w, h, actionListener);  
  
 y = y + h + h\_coord;  
 **if\_alg** = **new** Buttons(***IFALGS***, **"src/com/company/if.png"**, x + 30, y, w, h, actionListener);  
  
 x = x + 3 \* h\_coord + 2 \* w;  
 y = y0;  
 **for\_alg** = **new** Buttons(***FORALGS***, **"src/com/company/for.png"**, x, y, w, h, actionListener);  
  
  
 y = y + h + h\_coord;  
 **proc\_alg** = **new** Buttons(***PROCALGS***, **"src/com/company/proc.png"**, x - 30, y, w, h, actionListener);  
  
 x = x - (3 \* h\_coord + w) / 2 - w / 2;  
 y = y + h + h / 3;  
 **control\_alg** = **new** Buttons(***CONTROLALG***, **"src/com/company/control.png"**, x, y, w, h, actionListener);  
  
 x = 20;  
 y = 20;  
 **back** = **new** Buttons(***BACK***, **null**, x, y, w, h, actionListener);  
 **back**.getButton().setBackground(**new** Color(250, 250, 184));  
  
 y = Toolkit.*getDefaultToolkit*().getScreenSize().**height** - h\_coord \* 2;  
 x = Toolkit.*getDefaultToolkit*().getScreenSize().**width** / 2 - w \* 2 - h\_coord - h\_coord / 2;  
  
  
 **first** = **new** Buttons(***FIRST***, **null**, x, y, w, h, actionListener);  
 **first**.getButton().setBackground(**new** Color(248, 250, 18));  
 x += h + h\_coord;  
 **second** = **new** Buttons(***SECOND***, **null**, x, y, w, h, actionListener);  
 **second**.getButton().setBackground(**new** Color(248, 250, 18));  
 x += h + h\_coord;  
 **third** = **new** Buttons(***THIRD***, **null**, x, y, w, h, actionListener);  
 **third**.getButton().setBackground(**new** Color(248, 250, 18));  
 x += h + h\_coord;  
 **forth** = **new** Buttons(***FORTH***, **null**, x, y, w, h, actionListener);  
 **forth**.getButton().setBackground(**new** Color(248, 250, 18));  
  
  
 }  
  
 **private class** Buttons {  
 JButton **button**;  
 **public** Buttons (String command, String icon, **int** x, **int** y, **int** w, **int** h, ButtonActionListener actionListener) {  
 **button** = **new** JButton(command);  
 **button**.setIcon(**new** ImageIcon(Toolkit.*getDefaultToolkit*().getImage(icon)));  
 **button**.setActionCommand(command);  
 **button**.setBounds(x, y, w, h);  
 **button**.addActionListener(actionListener);  
 }  
  
 **public void** AddOnPanel () {  
 **panel**.add(**button**);  
 }  
  
 **public** JButton getButton() {  
 **return button**;  
 }  
  
 }  
  
  
}

## ПРИЛОЖЕНИЕ З Программная реализация класса Main

**import** java.awt.\*;  
**import** javax.swing.\*;  
**public class** Main **extends** JComponent{  
 **public static void** main(String[] args) {  
**final** JFrame frame = **new** JFrame(**"Робот"**);  
 frame.setIconImage(Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/robot.**

**png"**));  
 frame.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
Dimension dim = Toolkit.*getDefaultToolkit*().getScreenSize();  
 frame.setSize(dim.**width**, dim.**height**);  
  
 frame.setExtendedState(JFrame.***MAXIMIZED\_BOTH***);  
frame.setPreferredSize(**new** Dimension(dim.**width**, dim.**height**));  
  
 frame.pack();  
  
 **final** Title title = **new** Title(frame);  
 **final** JPanel panel = **new** JPanel() {  
 **public void** paintComponent (Graphics g) {  
 **super**.paintComponent(g);  
 Image background = Toolkit.*getDefaultToolkit*().getImage(**"src/com/company/background1.jpg"**);  
 g.drawImage (background, 0, 0, frame.getWidth(), frame.getHeight(), **this**);title.setPanel(**this**);  
 title.Draw(g);  
  
 }  
 };  
 panel.setLayout(**null**);  
 frame.add(panel);  
 frame.setVisible(**true**);  
 frame.repaint();  
 frame.repaint();  
 }  
}