МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МОЗЫРСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ

ИМЕНИ И.П. ШАМЯКИНА»

Республиканский конкурс научных работ студентов

Педагогика, теория и методика обучения и воспитания. Социальные проблемы воспитания. Информационные технологии в образовании. Воинское обучение и воспитание

**Алгоритмический игровой исполнитель**

Пиляк Ксения Юрьевна  
студентка 4 курса

Сергиевич Николай Владимирович, доцент кафедры теоретической физики и прикладной информатики, кандидат физико-математических наук, доцент

МОЗЫРЬ, 2016 год

**РЕФЕРАТ**

Работа 73 с., 17 рис., 5 источников, 9 прил.

АЛГОРИТМ, ВЕТВЛЕНИЕ, ЦИКЛ, ПОДПРОГРАММА, ИСПОЛНИТЕЛЬ, БЛОК-СХЕМА, ПРОГРАММА, JAVA, INTELLIJ IDEA, UML, ШАБЛОН «ФАБРИКА»

Объектом исследования являются алгоритмизация и современные технологии разработки программного обеспечения.

Цель работы – разработка программного обеспечения для обучения основам алгоритмизации в средней школе.

В процессе работы рассмотрены основные понятия алгоритмизации, составлена структурная схема программы на унифицированном языке моделирования (UML), выполненена ее программная реализция на языке программирования Java в среде разработки IntelliJ IDEA.

В результате работы создано приложение «Алгоритмический игровой исполнитель «Робот»».

Внедрено в учебный процесс общеобразовательных утверждений

Приложение может применяться в образовательном процессе в средних общеобразовательных учреждениях в качестве дополнительного электронного средства обучения.

В дальшейшем планируется написание веб-приложения по данному объекту исследования, что обеспечит более удобное использование приложения, а также расширение его функциональности.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 5](#_Toc462930586)

[1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 6](#_Toc462930587)

[1.1 Описание алгоритмического игрового исполнителя 6](#_Toc462930588)

[1.2 Описание интерфейса 6](#_Toc462930589)

[1.3 Теоретические основы темы «Основы алгоритмизации» 7](#_Toc462930590)

[1.4 Язык программирования Java 9](#_Toc462930591)

[2 РАЗРАБОТКА ИГРОВОГО АЛГОРИТМИЧЕСКОГО ИСПОЛНИТЕЛЯ «РОБОТ» 12](#_Toc462930592)

[2.1 Структурная схема программы 12](#_Toc462930593)

[2.2 Интерфейс TypeOfCommands 12](#_Toc462930594)

[2.3 Класс Field 12](#_Toc462930595)

[2.4 Класс Robot 15](#_Toc462930596)

[2.5 Класс ProgramField 16](#_Toc462930597)

[2.6 Класс ControlButton 19](#_Toc462930598)

[2.7 Класс Blocks 20](#_Toc462930599)

[2.8 Класс Task 21](#_Toc462930600)

[2.9 Класс Theory 26](#_Toc462930601)

[2.10 Класс Main 29](#_Toc462930602)

[ЗАКЛЮЧЕНИЕ 30](#_Toc462930603)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31](#_Toc462930604)

# 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 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.

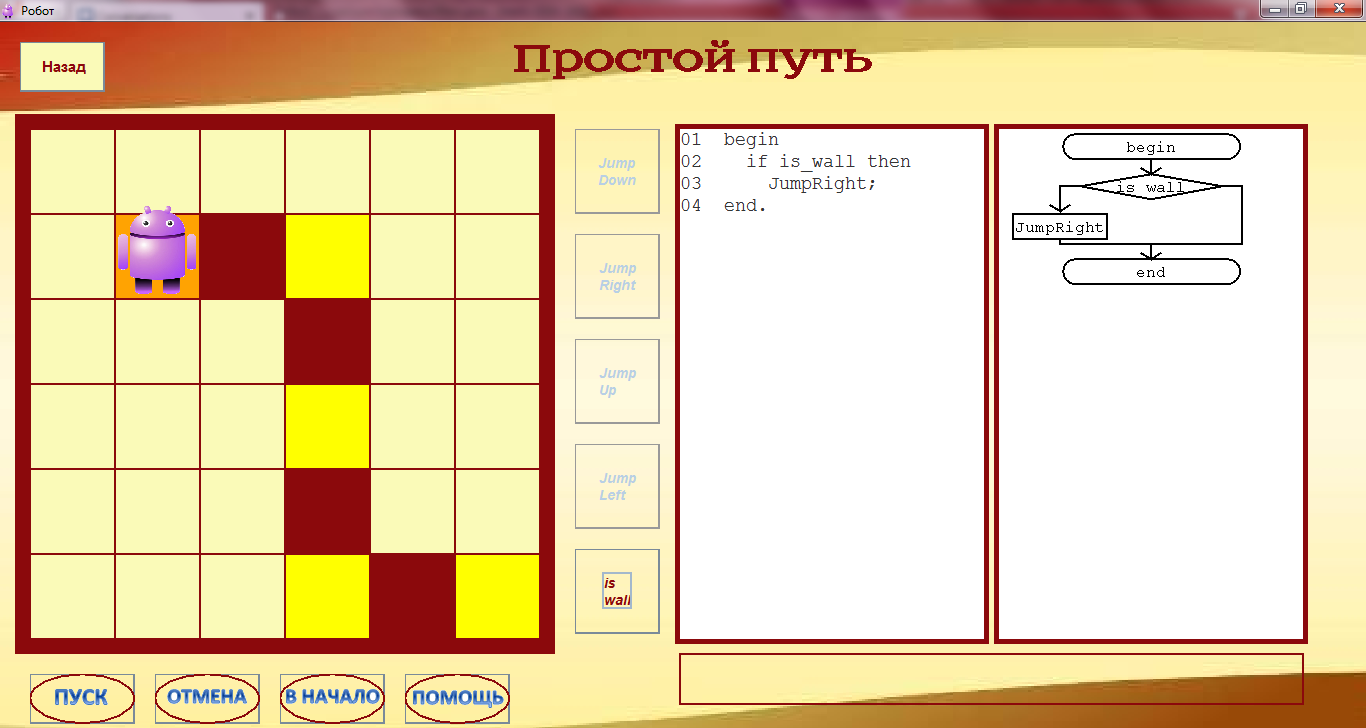
# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## Описание алгоритмического игрового исполнителя

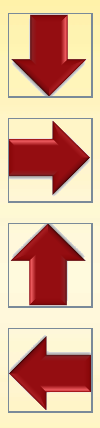
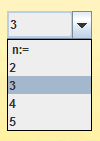
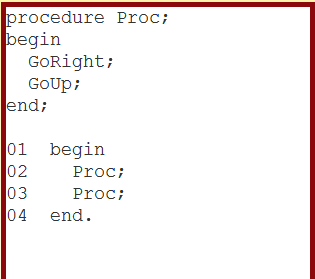
Алгоритмический игровой исполнитель «Робот» – это образовательная программа, с помощью которой осуществляется обучение учащихся основам алгоритмизации. Вся программа состоит из пяти разделов: «Линейные алгоритмы», «Алгоритмы с ветвлением», «Циклические алгоритмы», «Подпрограммы», «Контрольные задания». 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.

  
**Рисунок 1.1 – Интерфейс программы**

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).

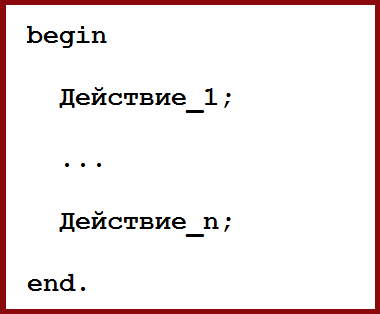
    
**Рисунок 1.2 – Элементы интерфейса программы**

## Теоретические основы темы «Основы алгоритмизации»

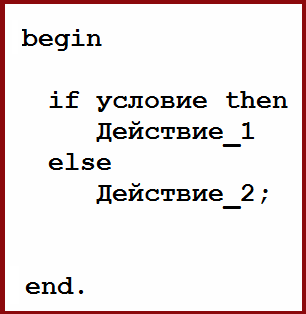
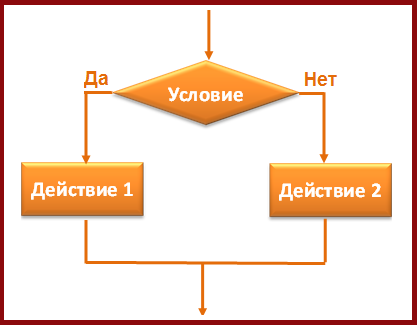
Для составления теоретической части разделов алгоритмического игрового исполнителя «Робот» подобран краткий материал по теме «Основы алгоритмизации». При этом были изучены методические разработки учителей информатики: Вилковой О.Ю. [2], Корнеевой Г.Ю. [1], Босовой Л.Л. [4]. Материал подобран с учетом 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).

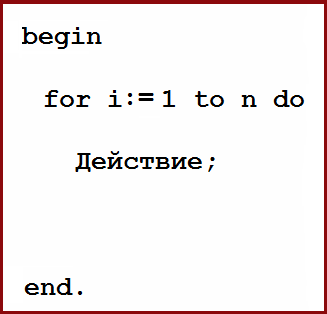
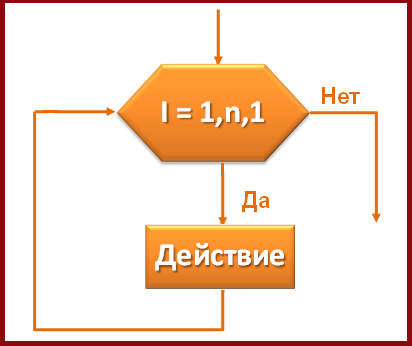
   
**Рисунок 1.3 – Линейный алгоритм**

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.

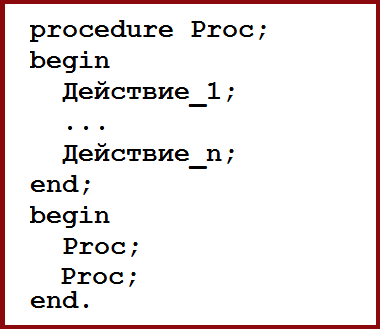
   
**Рисунок 1.5 – Циклический алгоритм**

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).

## 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.

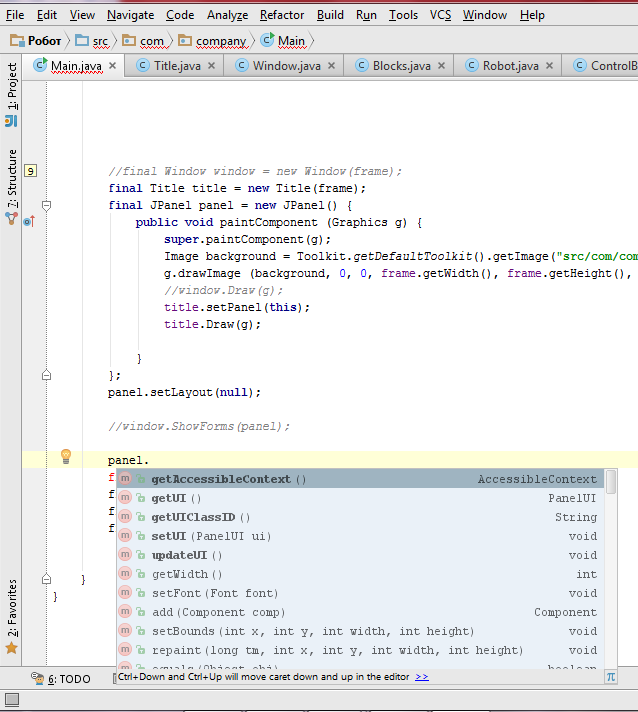
   
**Рисунок 1.6 – Алгоритм с подпрограммой**

**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**

# 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 Teory) и область практического обучения (класс Task). 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

CommandFactory

Theory

Task

Theory.Button

ControlButton

Robot

Field

Blocks

ProgramField

Main

**Рисунок 2.1 – UML-диаграмма классов**

## Шаблон Factory Method и его реализация в приложении

При разработке предусмотрено, что программа неким образом будет реагировать на действия пользователя (нажатия на кнопки). Соответственно, нужно описать действия программы, которые будут совершаться при нажатии на каждую кнопку. Конечно, это можно было сделать в одном классе-слушателе событий, однако это приводит к увеличению размеров кода, что сокращает читаемость, а в дальнейшем – усложняется отладка. Поэтому верным решением является использование шаблона Factory Method (Фабричный метод). Он определяет интерфейс для создания объектов из иерархического семейства классов на основе передаваемых данных. Классу-фабрике и его подклассам разрешается создавать экземпляры классов с помощью общего для всех интерфейса, причем именно подклассы решают, какой экземпляр следует создавать. Этот интерфейс может содержаться в абстрактном классе, интерфейсе или даже в обычном классе. Фабричный метод не должен владеть информацией о конкретном типе создаваемого объекта, но знает о классе-вершине иерархии создаваемых объектов [6].

В нашем случае классом-вершиной иерархии является абстрактный класс Command. В нем представлены методы без реализации, которые могут переопределяться в классах-наследниках. Перечисление CommandEnum – названия всех кнопок программы. Каждую описваемую кнопку представляет класс – наследник Command. Это такие классы как GoDown, JumpLeft, LinearAlgorithm, Return, Second и т.д. В них переопределяются и реализуются нужные методы. Так, если взять метод move, то в классе GoDown координата по y увеличивается на 5, а в методе GoUp – уменьшается на 5.

Класс CommandFactory – фабрика, в конструкторе которой создается объект command типа Command в зависимости от параметра – значения перечисления. Также в фабрике реализованы методы, совпадающие по имени и сигнатурам с методами из класса Command. В них вызывается одноименный метод объекта command.

Таким образом, чтобы выполнить некторые действия в ответ нажатия на кнопку, неоходимо в слушателе кнопок создать фабрику с параметром, соответствущим нажатой кнопке (значение перечисления), и вызвать нужный метод.

Преимущество фабричного метода заключается в простом использовании: не нужно задумываться о типе объекта, просто нужно указать перечисление, а JVM сама «узнает», какой метод нужно вызвать. Недостатком является увеличение количества классов в программе.

## Класс Field

**public class Field {**  
 private int x0, y0;  
 private int width, height;  
 public static final int *SQUARE\_SIZE* = 85;  
 private ArrayList<Point> robotWay = new ArrayList<>();  
 private Point[] necessaryCells;  
 private Point[] barrierCells;  
 private Point robotCoords;

public Field(int x0, int y0, int width, int height);

public void draw (Graphics g);

public void drawWay (CommandEnum command);

public void clearWay ();

private void paintCells (Graphics g);

public void setNecessaryCells(Point[] necessaryCells);

public void setBarrierCells(Point[] barrierCells);

public boolean isAllCellsCrossed();

public void setRobotCoords (Point robotCoords);

public ArrayList<Point> getRobotWay();

**}**

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 width, int height)** устанавливает координаты x0 и y0 левого верхнего угла поля в пикселях и ширину w и высоту h поля в клетках.

**Методы**

* **public void setRobotCoords(Point robotCoords)** устанавливает начальные координаты Робота robotCoords, а также добавляет их в список robotWay, в котором будут храниться координаты всех положений Робота во время выполнения заданного пользователем алгоритма. В списке задается середина клетки, в которой находится Робот.

this.robotCoords = robotCoords;  
robotWay.add(new Point(robotCoords.getX() + SQUARE\_SIZE / 2, robotCoords.getY() + SQUARE\_SIZE / 2));

* **public void drawWay(CommandEnum command)** вызывает одноименный метод фабрики.В зависимости от команды command, которую должен выполнить Робот, фабрика высчитывает координаты его текущего положения в пикселях и записывает их в конец списка robotWay. Так, например, если исполнителю была задана команда «Идти вниз» (GO\_DOWN), то к последнему элементу списка robotWay 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.

ArrayList<Point> robotWay = field.getRobotWay();  
robotWay.add(new Point(robotWay.get(robotWay.size() - 1).getX(), robotWay.get(robotWay.size() - 1).getY() + 5));

* **public void clearWay()** очищает содержимое списка robotWay и инициализирует их начальными координатами Робота.

robotWay.clear();  
robotWay.add(new Point(robotCoords.getX() + SQUARE\_SIZE / 2, robotCoords.getY() + SQUARE\_SIZE / 2));

* **public void setNecessaryCells(Point[] necessaryCells)** устанавливает клетки, которые нужно пройти Роботу в задании.

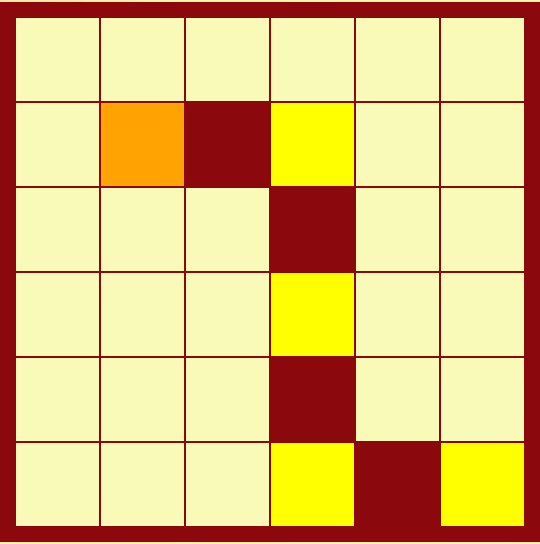
this.necessaryCells = necessaryCells.clone();

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

this.barrierCells = barrierCells.clone();

* **private void paintCells(Graphics g)** закрашивает клетки, которые необходимо пройти, желтым цветом и клетки, которые нужно перепрыгивать, коричневым цветом.
* **public boolean isAllCellsCrossed()** проверяет, все ли клетки, которые необходимо пройти Роботу в задании, пройдены. Для этого проверяется совпадение соответствующих координат всех положений Робота (список robotWay) и координат клеток, которые необходимо пройти. Поскольку список necessaryCellsсодержит 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 < necessaryCells.length; i++) {  
 for (int j = 0; j < robotWay.size(); j++)  
 if (robotWay.get(j).getX() == (x0 + necessaryCells[i].getX() \* SQUARE\_SIZE + SQUARE\_SIZE / 2) && robotWay.get(j).getY() == (y0 + necessaryCells[i].getY() \* SQUARE\_SIZE + SQUARE\_SIZE / 2))  
 k++;  
}  
return k >= necessaryCells.length;

 **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 списка robotWay. Результат работы метода показан на рис. 2.2.

## Класс Robot

**public class Robot {**  
 private int x0, y0;  
 private int x, y;  
 private int width, height;  
 private JFrame frame;

public Robot(int x0, int y0, int width, int height, JFrame frame);

public void draw(Graphics g);

public void setInitialCoords();

public int getX();

public int getY();

public void move(CommandEnum 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 width, int height, 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.width = width;  
this.height = height;  
this.frame = frame;

**Методы**

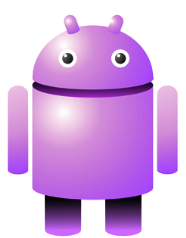
* **public void draw(Graphics g)** изображает Робота на экране.

Image background = Toolkit.getDefaultToolkit().getImage(

"src/com/company/robot.png"

);  
g.drawImage (background, x, y, width, height, 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 нужно установить 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** **(CommandEnum command)** вызывает одноименный метод фабрики, который 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» (GO\_DOWN), «Идти вправо» (GO\_RIGHT), «Идти вверх» (GO\_UP), «Идти влево» (GO\_LEFT), то значения соответствующих координат увеличиваются (уменьшаются) на 5 пикселей, поскольку Робот движется по экрану плавно. Если были заданы команды «Прыгнуть вниз» (JUMP\_DOWN), «Прыгнуть вправо» (JUMP\_RIGHT), «Прыгнуть вверх» (JUMP\_UP), «Прыгнуть влево» (JUMP\_LEFT), то значения соответствующих координат увеличиваются (уменьшаются) на две клетки, поскольку Робот прыгает через клетку мгновенно.

## Класс ProgramField

**public class ProgramField {**  
 private JTextArea textArea;  
 private String begin = "01 begin";  
 private String end = "end.";  
 private int numberOfCurrentString = 1;  
 private int nFor;  
 private boolean isProcedure;  
 private boolean isForBlock = false;  
 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 (CommandEnum command);

public void addTextToProcedure (CommandEnum command);

public void removeString ();

public void removeStringFromProcedure ();

public void setNFor (int n);

public void setProcedure (boolean b);

public void setForBlock(boolean isForBlock);

**}**

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 setProcedure(boolean b)** и устанавливает значение переменной isProcedure значением b, т.е. указывает наличие процедурного блока в программе. Если переменной isProcedure было присвоено значение true, то в текстовую область добавляется заголовок и пустое тело процедуры.

isProcedure = b;  
if (isProcedure) {  
 textArea.removeAll();  
 textArea.setText(procedure + "\n" + "end;" + "\n" + "\n" + begin + "\n" + "02 " + end);  
}

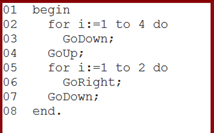
* **public void setForBlock(boolean isForBlock)** устанавливает значение переменной isForBlock.
* **public void addText** **(CommandEnum command)** добавляет команду command в программный блок. До добавления в текстовую область переменная text обрабатывается в зависимости от команды или оператора в соответствии с правилами оформления и форматирования кода. Для этого вызывается одноименный метод фабрики. Далее обработанная строка text добавляется к строке begin. При этом ведется нумерация строк в программном блоке.

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

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

textArea.removeAll();  
String procedure = "";  
if (isProcedure) {  
 procedure = this.procedure + "\n" + "end;" + "\n" + "\n";  
}  
if (numberOfCurrentString + 1 > 0 && numberOfCurrentString + 1 < 10)  
 textArea.setText(procedure + begin + "\n" + "0" + (numberOfCurrentString + 1) + " " + end);  
else  
 textArea.setText(procedure + begin + "\n" + (numberOfCurrentString + 1) + " " + end);

Результат работы метода приведен на рис. 2.4.

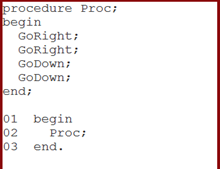
  
**Рисунок 2.4 – Программное поле**

* **public void addTextToProcedure(CommandEnum command)** добавляет команду command в процедурный блок. Строка command 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" + " " + commandEnum.getName() + ";";  
String procedure = this.procedure + "\n" + "end;" + "\n" + "\n";  
if (numberOfCurrentString + 1 > 0 && numberOfCurrentString + 1 < 10)  
 textArea.setText(procedure + begin + "\n" + "0" + (numberOfCurrentString + 1) + " " + end);  
else  
 textArea.setText(procedure + begin + "\n" + (numberOfCurrentString + 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.

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

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

## Класс ControlButton

**public class ControlButton**  **{**  
 private int x, y, width, height;  
 private JButton button;  
 private Image bgr;

public ControlButton (int x, int y, int width, int height, Image bgr, CommandEnum actionCommand);

private void createButton(CommandEnum actionCommand);

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, CANCEL\_IN\_PROCEDURE, 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 width, int height, Image bgr, CommandEnum actionCommand)** задает начальные координаты, ширину и высоту кнопки в пикселях, а также фоновую картинку bgr и команду действия кнопки actionCommand.

this.x = x;  
this.y = y;  
this.width = width;  
this.height = height;  
this.bgr = bgr;  
UIManager.put("Button.background", new Color (0, 0, 0, 0));  
createButton(actionCommand);  
button.setBounds(x, y, width, height);

**Методы**

* **private void createButton(CommandEnum actionCommand)** создает экземпляр кнопки JButton (для операторов или команд) или RoundButton (для кнопок управления исполнителем) с подписью, соответствующей ее команде действия actionCommand или с заданным рисунком bgr.
* **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 {**  
 private int x0, y0, width, height;  
 private final String BEGIN = "begin";  
 private final String END = "end";  
 private final int OVAL\_WIDTH; //ширина овала  
 private final int RECTANGLE\_WIDTH; //ширина прямоугольника  
 public static final int FIGURE\_HEIGHT = 25; //высота блока  
 public static final int BIG\_ARROW\_LENGTH = 15; //длина стрелочки  
 private ArrayList<CommandEnum> commands = new ArrayList<>();  
 public static final int SMALL\_ARROW\_LENGTH = 5;  
 public static final float ALPHA = (float) Math.PI / 6;  
 private int nFor;

public Blocks(int x0, int y0, int width, int height);

public void addCommand(CommandEnum command);

public void removeCommand();

public void draw(Graphics g);

public void setNFor (int n);

public ArrayList<CommandEnum> getCommands();

**}**

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

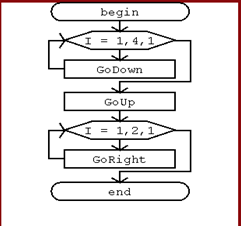
**Конструктор public Blocks(int x0, int y0, int width, int height)** задает координаты, ширину и высоту блок-схемы и устанавливает ширину овала (для «begin» и «end») и прямоугольника (для произвольной команды).

**Методы**

* **public void setNFor(int n)** устанавливает количество повторений n\_for в цикле for.
* **public void addCommand(CommandEnum command)** добавляет команду command в список команд commands.
* **public void removeCommand()** удаляет последнюю введенную пользователем команду из списка команд commands, если это возможно.

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

* **public ArrayList<CommandEnum> getCommands()** возвращает текущий список команд commands.
* **public void draw(Graphics g)** изображает блок-схему на экране согласно списку команд путем вызова одноименного метода. 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. Результат работы метода приведен на рис. 2.7.

  
**Рисунок 2.7 –Блок-схема**

## Класс Task

**public class Task** **{**  
 private int fieldX = 30;  
 private int fieldY;  
 private final int FIELD\_WIDTH = 6, FIELD\_HEIGHT = 6;  
 private int programFieldX = fieldX + Field.SQUARE\_SIZE \* 7 + 40 + 15;  
 private int programFieldY;  
 private int programFieldWidth;  
 private int programFieldHeight = FIELD\_HEIGHT \* Field.SQUARE\_SIZE;  
 private int robotX = fieldX;  
 private int robotY;  
 private final int ROBOT\_WIDTH = Field.SQUARE\_SIZE, ROBOT\_HEIGHT = Field.SQUARE\_SIZE + 10;  
 private Field field;  
 private Robot robot;  
 private ProgramField programField;  
 private Blocks blocks;  
 private int nFor;  
 private CommandEnum signal;  
 private int ms = 50;  
 private int lineLimit;  
 private int lineLimitInProcedure;  
 private int numberOfCommands = 0;  
 //список команд  
 private ArrayList<CommandEnum> commands = new ArrayList<>();  
 private ArrayList<CommandEnum> commandsInProcedure = new ArrayList<>();  
 private JFrame frame;  
 private JPanel panel;  
 private int num = -1;  
 private int numForProcedure = -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 Task(JFrame frame, JPanel panel, int robot\_x, int robot\_y, CommandEnum signal);

public void draw(Graphics g);

public void setTitle(String title);

public void showForms();

private void makeButtons();

public int getLineLimit();  
 public CommandEnum getSignal();  
 public int getLineLimitInProcedure();  
 public ProgramField getProgramField();  
 public ArrayList<CommandEnum> getCommandsInProcedure();  
 public int getNumberOfCommands();  
 public void setNumberOfCommands(int numberOfCommands);  
 public ArrayList<CommandEnum> getCommands();  
 public int getNFor();  
 public Blocks getBlocks();  
 public JFrame getFrame();  
 public ControlButton getIf();  
 public ControlButton getJumpDown();  
 public ControlButton getJumpRight();  
 public ControlButton getJumpUp();  
 public ControlButton getJumpLeft();  
 public ControlButton getReturn();

public Field getField();

public void setLineLimit(int lineLimit);

public void setLineLimitInProcedure(int lineLimitInProcedure);

public Robot getRobot();

public void setNum(int num);

public void setNumForProcedure(int numForProcedure);

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 Task(JFrame frame, JPanel panel, int robot\_x, int robot\_y, CommandEnum signal)** устанавливает координаты объектов, зависящие от размеров окна, создает объекты вышеперечисленных классов, указывает вид алгоритма текущего.

**Методы**

* **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);  
  
  
g.setColor(new Color(139, 9, 12));  
g.fillRect(programFieldX - 5, programFieldY - 5, programFieldWidth + 10, programFieldHeight + 10);  
  
g.drawRect(programFieldX, programFieldY + programFieldHeight + 15, programFieldWidth \* 2 + 15, 50);

* **public void showForms()** добавляет все созданные формы (кнопки и программное поле) на панель. 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. 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. Количество нажатий на кнопки команд и операторов ограничивается, т.е. обработка событий происходит лишь при выполнении условий numberOfCommands < lineLimit и numForProcedure < lineLimitInProcedure. 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 addTextToProcedure), 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 nFor \* Field.SQUARE\_SIZE / 5, где nFor – 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). 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.

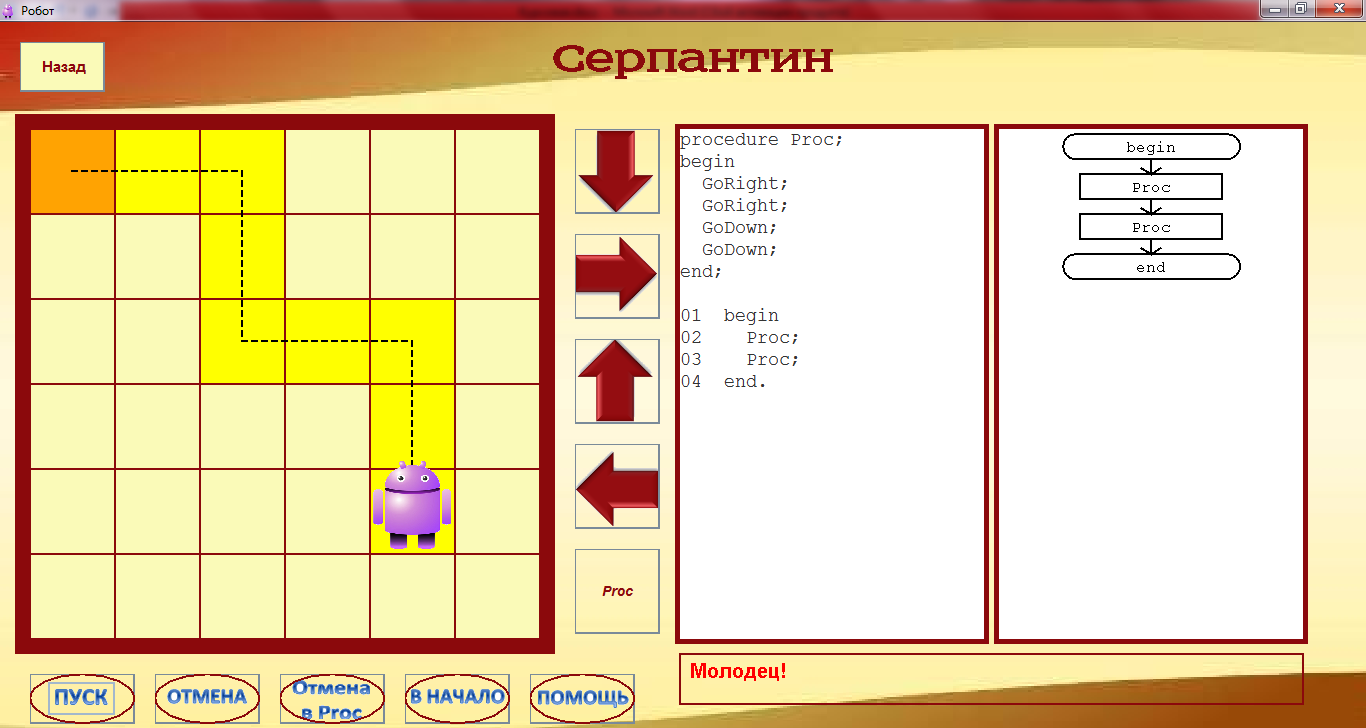
* **private class CMBActionListener implements ActionListener**

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 (numberOfCommands < lineLimit) {  
 nFor = Integer.parseInt(item);  
 programField.setNFor(nFor);  
 blocks.setNFor(nFor);  
  
 programField.addText(CommandEnum.FOR);  
 commands.add(CommandEnum.FOR);  
 blocks.addCommand(CommandEnum.FOR);  
 numberOfCommands++;  
 frame.repaint();  
 }  
 } catch (Exception e) {  
 }  
}

Результат работы методов класса приведен на рис. 2.8.

  
**Рисунок 2.8 – Окно для выполнения заданий**

## Класс Theory

**public class Theory** **{**  
 private JFrame frame;  
 private JPanel panel;  
 private CommandEnum command;  
 private boolean isTheory = true;  
 private boolean isWork = false;  
 private Button linearAlg; //кнопка "Линейные алгоритмы"  
 private Button ifAlg; //кнопка "Условные алгоритмы"  
 private Button forAlg; //кнопка "Циклические алгоритмы"  
 private Button procAlg; //кнопка "Подпрограммы"  
 private Button controlAlg; //кнопка "Контрольные задания"  
 private Button back; //кнопка "Назад"  
 private Button first; //кнопка первого задания  
 private Button second; //кнопка второго задания  
 private Button third; //кнопка третьего задания  
 private Button forth; //кнопка четвертого задания  
 private Task task;  
 private int xOnTitle;  
 private int yOnTitle;

public Title (JFrame frame);

public void draw(Graphics g);

public void makeTitle(Graphics g, String title, String text, Image block, Image prog, Image img, String theory);

public Button getBack();

public Task getTask();

public boolean isWork();

public void setTheory(boolean isTheory);

public void setWork(boolean isWork);

public void setCommand(CommandEnum command);

public JFrame getFrame();

public CommandEnum getCommand();

public void setPanel (JPanel panel);

public void makeWork(int robotX, int robotY, Point[] cells, CommandEnum 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 Button (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.
* **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).
* **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 robotX, int robotY, Point[] cells, CommandEnum signal, String title, int limit)**создает объект класса Task. Устанавливает название выбранного задания (метод task.setTitle), клетки, которые необходимо пройти (task.getField().setNecessaryCells) и предел вводимых команд в задании.

if (task != null)  
 task.TIMER.stop();  
isWork = true;  
panel.removeAll();  
task = new Task(frame, panel, robotX, robotY, signal);  
task.setTitle(title);  
task.showForms();  
task.getField().setNecessaryCells(cells);  
task.setLineLimit(limit);

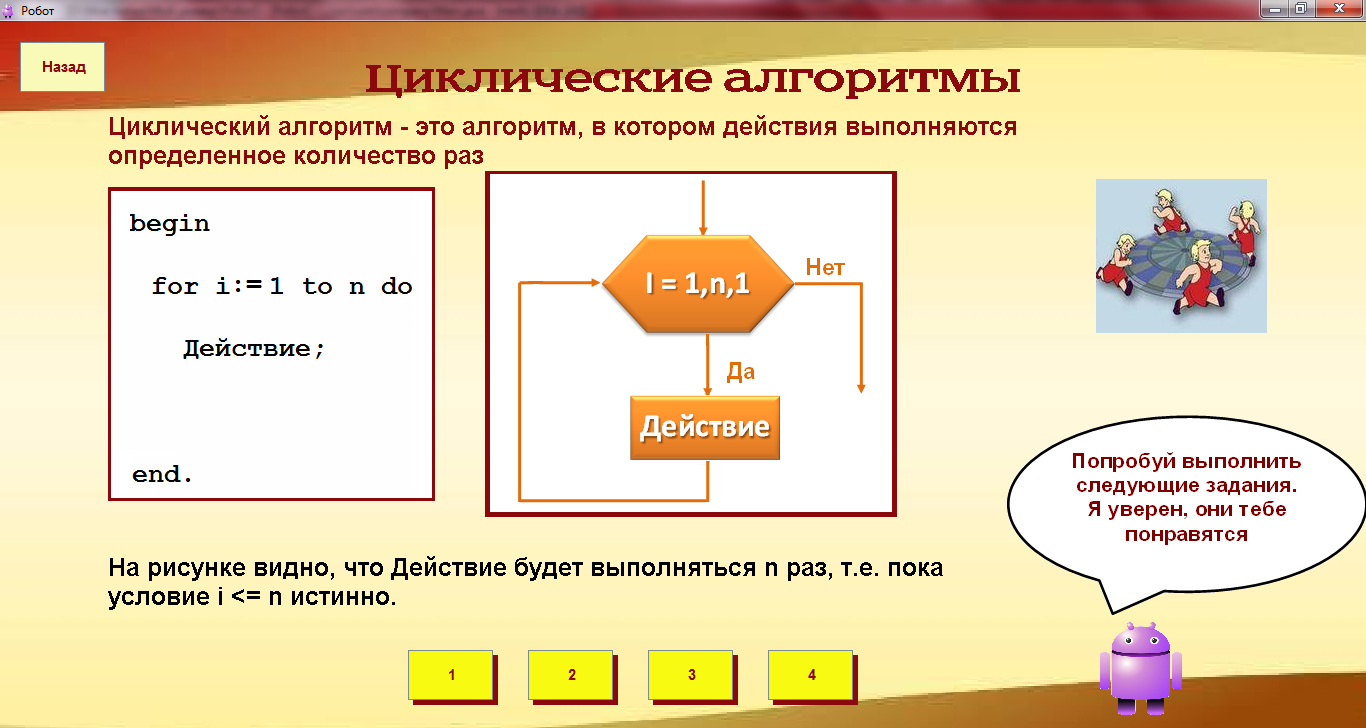
**Закрытый класс ButtonActionListener** реализует интерфейс ActionListener и является слушателем кнопок. В зависимости от того, какой раздел был, устанавливается флаг соответствующей булевой переменной. При нажатии на кнопки first, second, third, forth задаются параметры задания пользователю: клетки, 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**

## 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.

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

# 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.
6. Блинов, И.Н. Java. Методы программирования: уч.-мет. пособие / И.Н. Блинов, В.С. Романчик. – Мн: издательство «Четыре четверти», 2013. – 896 с.