**Департамент образования города Москвы**

Государственное образовательное учреждение высшего профессионального образования

**МОСКОВСКИЙ ГОРОДСКОЙ ПСИХОЛОГО-**

**ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
|  |

**Факульт**е**т** *Информационных технологий*

**Кафедра**  *Прикладной математики*

**ВЫПУСКНАЯ**

**КВАЛИФИКАЦИОННАЯ РАБОТА**

*Тема****: «Разработка интерактивного компьютерного учебного теста по теме «Прямые на плоскости»»***

***Автор*** *Лизуткина Ольга Александровна ­*

***Научный руководитель работы*** *док. физ.-мат. н, проф. А. Д Яшин, канд. техн. н. проф. В. В. Балтрушайтис*

*(подпись, дата)*

***Консультанты* (с указанием относящихся к ним разделов):**

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Рецензент*** *канд. физ. – мат. н., проф. Е. Г. Куланин*

***Работа допущена к защите***

***Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(А.Д. Яшин)***

«*15*» мая20*09*г.

**Департамент образования города Москвы**

Государственное образовательное учреждение высшего профессионального образования

**МОСКОВСКИЙ ГОРОДСКОЙ ПСИХОЛОГО-**

**ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
|  |

**Факульт**е**т** *Информационных технологий*

**Кафедра**  *Прикладной математики*

**ЗАДАНИЕ**

**по подготовке выпускной квалификационной работы**

Студент Лизуткина Ольга Александровна *5.2*

Ф.И.О. Группа

Получаемая квалификация (степень)  *математик-программист*

**Тема:**  ***«Разработка интерактивного компьютерного учебного теста по теме «Прямые на плоскости»»***

**Утверждена:** *12.03.09 ­ 07-03/203-у*

дата № приказа

Научный руководитель *док. физ.-мат. н, проф. А. Д. Яшин, канд. техн. н. проф. В. В. Балтрушайтис*

должность звание Ф.И.О.

Консультант (ы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

должность звание Ф.И.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

должность звание Ф.И.О.

Срок сдачи студентом оформленной выпускной квалификационной работы

*28.05.09 ­*

**СОДЕРЖАНИЕ ЗАДАНИЯ**

1. Сбор рекомендуемой литературы по теме.

2. Разработка интерфейса

3.Разработка методов генерации заданий

4.Разработка методов проверки введенных ответов

**КАЛЕНДАРНЫЙ ГРАФИК РАБОТЫ**

|  |  |  |  |
| --- | --- | --- | --- |
| *№п/п* | *Содержание разделов* | *Срок выполнения* | *Отметка о*  *выполнении* |
| 1 | Анализ предложенной литературы. | 27.11.2008 |  |
| 2 | Поиск и анализ тестов. | 14.12.2008 |  |
| 3 | Разработка теоретической части. Подбор задач. | 11.01.2009 |  |
| 4 | Составление схемы приложения | 16.01.2009 |  |
| 5 | Разработка алгоритмов приложения | 07.02.2009 |  |
| 6 | Создание пользовательского интерфейса | 10.03.2009 |  |
| 7 | Создание функции проверки данных | 27.03.2009 |  |
| 8 | Тестирование и отладка приложений | 26.04.2009 |  |
| 9 | Обобщение данных. Написание текста работы | 09.05.2009 |  |

**РЕКОМЕНДУЕМАЯ ИСХОДНАЯ ЛИТЕРАТУРА**

Прасолов В.В. Задачи по планиметрии. — 4-е изд., дополненное — М.: Изд-во Московского центра непрерывного математического образования, 2001.

Ефимов Н. В. Краткий курс аналитической геометрии: Учебн. пособие. — 13-е изд., стереот. — М.: ФИЗМАТЛИТ, 2005. — 240 с.

Фленов М.Е. Библия Delphi, БХВ-Петербург, 2005. — 880с.

Сухарев М.В., Основы Delphi. Профессиональный подход, Наука и техника, 2004. — 600с.

Ермакова М.Г., Андреева Л.Е. Вопросы разработки тестирующих программ. //Информатика и образование. – 1997. №3.

Пак Н.И., Филиппов В.В. О технологии создания компьютерных тестов. //информатика и образование. – 1997. №5.

*Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Научный руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Оглавление

Аннотация 5

Введение 6

Постановка задачи 7

Краткое описание внутреннего устройства: 8

Интерфейс 10

Использованные функции геометрических типов данных и операций над ними. 13

Примеры решения разных типов задач и алгоритмы генерации задач программой 15

1 Найти 2 точки, лежащие на этой прямой. 15

2 Найти тангенс угла наклона прямой к оси Ox 17

3 Составить уравнение в отрезках. 18

4 Найти нормальный вектор и направляющий вектор. 20

5 Найти отрезки, отсекаемые прямой на осях координат. 21

6 Записать каноническое уравнение прямой. 23

7 Записать параметрическое уравнение 26

8 Найти расстояние от точки M0 (3, 5) до прямой 28

9 Составить уравнение перпендикуляра, опущенного на эту прямую из точки (7,10). 31

10 Составить уравнение прямой, проходящей через точку (4,-6) параллельно этой прямой. 33

11 Найти координаты точки пересечения данной прямой с прямой y=-3x+1 35

12 Найти проекцию точки M(4,-6) на данную прямую 36

13 Составить уравнение прямой, проходящей через точку (1,1) под углом 45° 38

14 Составить уравнение прямой, отстоящей от данной на расстояние d=4. 40

15 Пересекает ли эта прямая отрезок ****, если A(-2,6), B(3,-4) 42

Вывод результатов 45

Заключение 46

Список используемой литературы 47

Приложение 49

Аннотация

Данная работа направлена на разработку открытого интерактивного тестирования по теме «Прямые на плоскости».

В данной работе представлена теоретическая часть, посвященная поиску методов генерации заданий и методов проверки введенных ответов.

Отличительными особенностями этого тестирования является, во-первых, его открытость. То есть ответ не выбирается из списка, а вводится с клавиатуры. А во-вторых, разработан алгоритм генерации исходных данных.

В приложении представлен программный код продукта, разработанного для данной работы.

Введение

Большинство тестирований построены по схеме выбора одного или нескольких ответов из предложенного списка. Такой подход имеет ряд недостатков. Например, трудоемкий процесс подбора заданий, ответов, причем составителю теста приходится подбирать как правильный, так и неправильные ответы. Также существует возможность угадывания верного ответа, причем процент , что это случится очень велик.

Для решения таких проблем, как угадывание целесообразно проводить открытые тестирования, где ученику предлагается вводить ответ, а не выбирать его.

Для проведения подобного тестирования проверяющему знания студентов необходимо подобрать разнообразные задачи для каждого варианта. Основная сложность этого подбора состоит в том, что необходимым условием является одинаковый уровень сложности для каждого варианта.

Актуальным решением проблемы является автоматизированное порождение тестирований для проверки знаний по выбранной теме. Однако, если генерировать задания случайно, то при решении таких заданий ученик будет сталкиваться с трудностями, например с иррациональностью подсчета. Отсюда можно сделать вывод: нужен направленный перебор исходных значений. Формирование теста для каждого варианта происходит по следующей схеме: даны основные типы задач (набор задач одинаков для каждого варианта), а такие численные параметры, как, например коэффициенты в исходных уравнениях порождаются самой программой по индивидуальной схеме для каждой задачи.

В основной части дипломной работы подобранны некоторые типы задач по теме «Прямые на плоскости» и алгоритмы их решения тестируемым. А также алгоритмы подбора исходных коэффициентов для получения так называемых «красивых ответов».

Постановка задачи

Для разработки автоматизированного интерактивного компьютерного тестирования необходимо решить ряд первостепенных задач. Как и любой программе, тестированию нужен понятный интерфейс, который позволит ученику затрачивать внимание в основном на решение задач, но в свою очередь даст возможность преподавателю устанавливать некоторые настройки.

Необходимо разработать методы генерации исходных значений, которые позволят при решении заданий не испытывать трудностей с вычислениями.

А также разработка методов проверки правильности введенного учеником ответа.

Глава 1. Особенности автоматической генерации задач

****Тестирование в учебных заведениях****

Преподаватель в школе или университете должен проверять, какие знания присутствуют у его учеников. Для этого хорошо подходят контрольные задания. Ученикам выдается набор задач, которые они должны решить самостоятельно, но обеспечить это оказывается не очень просто. Этому мешает ряд проблем:

* Варианты должны быть различные у каждого ученика. Составление каждого варианта задачи ложится на преподавателя;
* Если варианты все таки повторяются, необходимо исключить возможность списывания. Обеспечить это часто бывает затруднительно;

В рамках этой работы рассмотрим вариант с отдельными вариантами для каждого ученика. Причем недостаточно сделать задачи для одной группы, так как тогда следующая за ней группа уже будет знать ответы. Поэтому необходимо делать отдельный вариант на каждого человека на курсе (в идеале, на каждого отдельного человека). Если в контрольной 4 задачи, а в группе 25 человек, то требуется создать 100 вариантов. Это может сделать очень трудолюбивый преподаватель, либо же это может быть сделано автоматически, с помощью алгоритмов генерации задач.

****Автоматическая генерация задач****

Такой автоматический генератор задач (или просто генератор) должен удовлетворять некоторым требованиям:

* Варианты не должны сильно отличаться по сложности
* Исходные данные и ответ должны быть “красивыми”, то есть не должны быть слишком большими или слишком маленькими
* Вариантов ответов может быть больше одного (иногда бесконечное количество), генератор должен быть способен корректно их обработать

В их соблюдении есть некоторые сложности и они варьируются в зависимости от задачи. Для обеспечения “красивости” данных в некоторых задачах можно просто генерировать “красивые” исходные данные – ответ сгенерируется посредством формул. В большинстве же случаев придется решать второстепенную задачу, уникальную для каждого типа задачи.

****Простой способ генерации общего уравнения прямой****

Для создания задач на тему прямых на плоскости можно оперировать общим уравнением прямой:

В таком случае, в большинстве типов задач для генерации прямой будет достаточно задать 3 случайных числа, удовлетворяющим условиям:

1. Коэффициент A должен быть положительным;
2. Все числа должны быть взаимно простыми;
3. Все числа должны быть целыми.

Последнее важно для обеспечения “красивости” условия задачи и ответа на него.

Для данных целей можно сгенерировать 3 числа в определенном диапазоне, а затем провести над ними операцию нормирования:

1. Если А отрицательный – умножаем все коэффициенты на -1;
2. Делим каждый из коэффициентов на НОД всех трех.

**Нахождение НОД.**

НОД двух чисел находится с помощью алгоритма Эвклида:

1. Большее число делим на меньшее;
2. Если делится без остатка, то меньшее число и есть НОД;
3. Если есть остаток, то большее число заменяем на остаток от деления;
4. Переходим к пункту 1.

Нахождение НОД трех чисел:

Следовательно, имея алгоритм нахождения НОД для двух чисел, его можно с помощью рекурсии преобразовать в алгоритм находения НОД произвольного числа чисел.

**Другие примеры генерации задач**

Однако не все задачи получится сгенерировать таким образом. В некоторых случаях использование данного способа приведет к дробным, а иногда даже к иррациональным ответам.

Например, в задаче нахождения расстояния от точки до прямой расстояние в общем случае будет числом “некрасивым”. В общем случае эта задача звучит так:

Дана прямая, заданная уравнением:

Найти расстояние от нее до точки .

Для решения этой задачи необходимо обратиться к формуле нахождения расстояния:

В данном случае числитель не представляет проблем, так как если все числа будут целыми, числитель тоже будет целым. Для того, чтобы в знаменателе не получилось иррациональное число, необходимо соблюдение условия:

,

где C – целое число. Тогда при извлечении корня получится целое число C, что в итоге даст дробное значение расстояния.

Такие тройки чисел называются пифагоровыми тройками. Формула Эвклида является основным средством их построения. Согласно ей для любой пары чисел m и n (m > n) целые числа , , образуют пифагорову тройку. Любая примитивная тройка получается из пары взаимно простых чисел m и n, одно из которых четно. Все остальные (не примитивные) тройки могут быть образованы путем умножения целого числа k на A, B и C:

, ,

Таким образом, можно расписать алгоритм генерации такой задачи:

1. Генерируются числа m, n и k;
2. Вычисляются A, B, C по формулам Эвклида;
3. Выбираются случайные числа C, и ;
4. Проводится операция нормирования коэффициентов прямой.

В получившейся задаче исходные данные будут состоять из целых чисел, а ответ будет либо целым либо дробным (но не иррациональным).

**Генерация ответов к задачам**

Когда условия задачи уже сгенерированы, есть возможность сгенерировать ответ. Такой подход имеет ряд недостатков:

* Задача может иметь несколько или даже бесконечное множество решений
* Ответ нужно будет где то хранить и эту информацию будет возможно извлечь

Поэтому целесообразнее (а в некоторых случаях других вариантов просто нет) не генерировать ответ заранее. Вместо этого, введенный учеником ответ можно подставлять в готовую формулу проверки решения.Краткое описание внутреннего устройства:

1. среда разработки Delphi 7
2. базовые типы данных, используемые в программе:

* тип ответа - TAnswerType, определяет тип ответа - да/нет, выбор из списка, точка (x,y), линия (a,b,c), набор точек, набор линий.
* формат ответа - TAnswerFormat, определяет формат ответа. Для ответа типа "выбор из списка" содержит список вариантов.
* сам ответ - TAnswer. Интерфейс ответа. Данные ответа определяются его типом (TAnswerType). Реализованы следующие интерфейсы:
  + TAnswerBool - ответ типа да/нет
  + TAnswerSelection - ответ типа выбор из списка вариантов
* Вопрос. Базовый интерфейс - TQuestion. Любой вопрос имеет следующие свойства:
  + формат ответа (да/нет, точка, линия и пр.)
  + текст вопроса
  + метод проверки правильности ответа. В качестве аргумента передается объект, реализующий интерфейс TAnswer.

В программе есть глобальный список доступных вопросов, который задается в файле QuestionFactory, в массиве ReqistredQuestions.

* Список вопросов - TQuestions. Фактически это список объектов, реализующих интерфейс TQuestion. Список формируется перед началом теста. В этот момент и формируются уникальные вопросы (уникальные значения параметров).
* Для отображения вопроса используется панель TQuestionPanel, которая содержит текст вопроса и динамически создаваемую панель для ввода ответа. Типа панели определяется типом TAnswerType. Для каждого такого типа есть панель ввода, которая наследуюется о панели типа TAnswerPanel. Например для ответа типа да/нет - панель - TAnswerPanelBool, для выбора из списка - TAnswerPanelSelection. Каждая панель содержит два основных метода:
  + GetInput - проверить, насколько полно введен ответ. Если ничего не введено, то ответ пропускается. Если полностью - то ответ проверяется.
  + GetAnswer - получить ответ в виде объекта с интерфейсом TAnswer. Этот объект напрямую передается в метод Check объекта с интерфейсом TQuestion для проверки правильности его ответа.

1. Логика перехода между вопросами реализуется в основной форме приложения - TMainForm. см. методы BeginTest, NextQuestion, BeginQuestion, CompleteQuestion, GotoNextQuestion, FinishTest.

Интерфейс

При запуске программы открывается диалоговое окно для ввода фамилии, имени и группы тестируемого (Рис. 1).

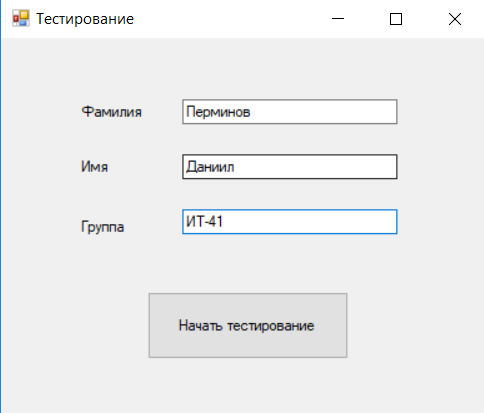


Рис. 1

Реализована возможность выбора отдельных вопросов при нажатии на кнопку «Настройка» (Рис. 2). Это функция сделана для преподавателя для того, чтобы он имел возможность, как проверить общие знания по окончанию изучения темы целиком, так и проверки остаточных знаний после изучения каждой из подтем.

Причем открыть настройки можно, только по паролю. Пароль устанавливается в текстовом документе «p.txt», он должен находиться в корневом каталоге с программой. Если не устанавливать пароль - по умолчанию устанавливается дата формата ЧЧММГГГГ (например «25052009»).

Рис. 2

Рис. 3

Также есть возможность задания времени для решения теста учеником. Эта возможность может использоваться как для сокращения времени при уменьшении количества вопросов, так и для уменьшения времени, отводимого для прохождения теста для групп с повышенной успеваемости и соответственно для увеличения времени для групп с пониженной успеваемостью. Реализована возможность установки одинаковых исходных значений, записи результатов тестирования в файл и вывода правильности ответа, для каждого вопроса (Рис.3).

После того, как приняты выбранные вопросы и установлено время, тестирование начинается при нажатии кнопки «Начать» в главном окне программы.

При прохождении тестирования в нижней части окна расположен список номеров вопросов. Еще не отвеченные и пропущенные вопросы отмечены серым цветом. Вопросы, на которые тестируемый уже ответил, отмечены синим цветом.(Рис.4)

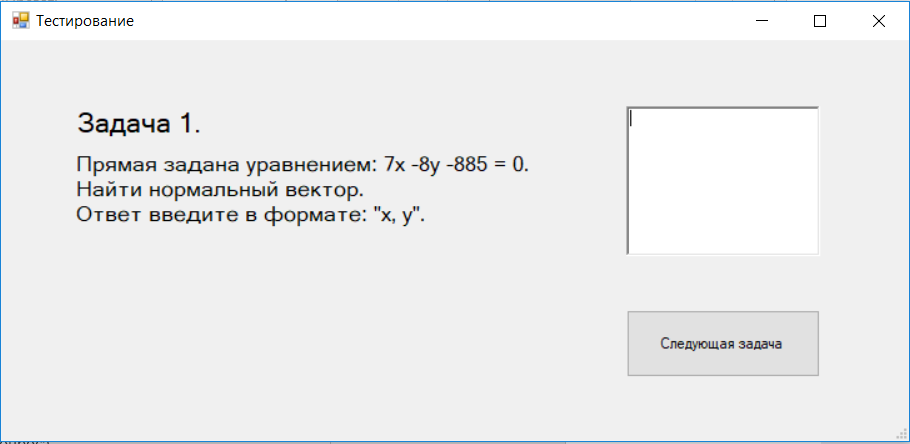


Рис. 4

Использованные функции геометрических типов данных и операций над ними.

Задание и реализация типов и функций см. Приложение из файла Geometry.pas

Геометрические типы:

1. Точка в двухмерном пространстве

TPoint2D

1. Линия в двухмерном пространстве

TLine2D

1. Отрезок в двухмерном пространстве

TSegment2D

Функции операций над геометрическими типами данных

1. Проверить, принадлежит ли точка линии

function IsPointOnLine

1. Форматирование части уравнения прямой

function FormatP

1. Форматирование строки в виде ax+by+c=0

function FormatLine2D

1. Форматирование точки в виде (x,y)

function FormatPoint2D

1. Проверить, является ли вектор нулевым

function IsNullVector

1. Проверить, является вектор нормальным к прямой

function IsNormalVector

1. Проверить, является вектор направляющим к прямой

function IsDirectionVector

1. Получить нормальный вектор

function GetNormalVector

1. Получить направляющий вектор

function GetDirectionVector

1. Найти расстояние от точки до прямой

function GetDistance

1. Найти расстояние между параллельными прямыми

function GetDistance

1. Проверить, параллельны ли прямые

function IsLinesParallel

1. Проверить, пересекаются ли прямые

function IsLinesCrossed

1. Проверить, являются прямые перпендикулярными

function IsLinesNormal

1. Найти точку пересечения прямых

function GetCrossPoint

1. Получить уравнение прямой, проходящей через две точки

function GetLineFromPoints

1. Получить уравнение прямой, проходящей через две точки

function GetLineFromSegment

1. Найти тангенс угола между прямыми

function GetLinesTangent

Примеры решения разных типов задач и алгоритмы генерации задач программой

В этом разделе речь пойдет о том как бы решал предложенные задания ученик, как формируется задача и как осуществляется проверка правильности ответа введенного учеником.

Прямая задана своим уравнением l: 3x+4y-5=0

1. **Найти 2 точки, лежащие на этой прямой.**

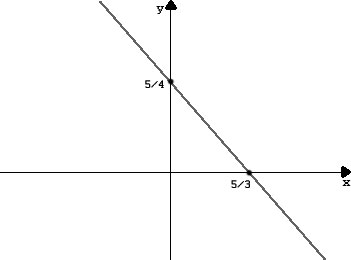
****

Рис. 5

Решение: Так как мы работаем с прямой, то логично, что эта прямая пересекает обе оси координат. Таким образом, можно найти 2 точки (Рис.5):

* 1. Возьмем *x=0*, тогда

*4y-5=5*

*y=5/4*

*y=-C/B*

* 1. Возьмем *y=0*, тогда

*3x-5=0*

*x=5/3*

*x=-C/A*

Так если прямая задана общим уравнением *Ax+By+C=0*, то имея только уравнение прямой мы можем найти 2 точки, принадлежащие это прямой *(0, -C/B) и (-C/A,0*).

Алгоритм генерации задания программой:

1. Прямая задается случайно.

2. Ученик сам находит 2 точки.

3. Координаты подставляются в исходное уравнение прямой, если тождество сохраняется, то ученик ответил правильно, иначе ответ неверен.

constructor TQ001.Create;

begin

inherited Create;

Format := TAnswerFormat.Create(TwoPoint2D);

Line := GenerateLine2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти две точки, лежащие на этой прямой';

end;

function TQ001.Check(answer : TAnswer) : Boolean;

var

p1, p2: TPoint2D;

begin

p1 := (answer as TAnswerTwoPoint2D).Point1;

p2 := (answer as TAnswerTwoPoint2D).Point2;

Result := IsPointOnLine(Line, p1) and IsPointOnLine(Line, p2);

end;

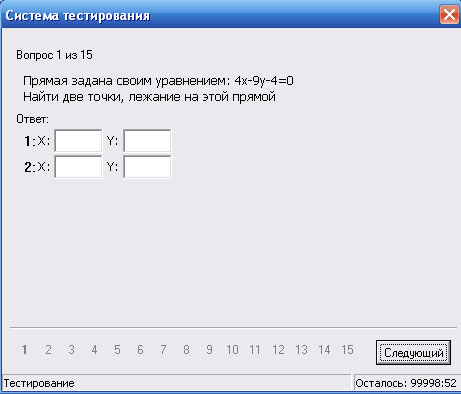


Рис. 6

1. **Найти тангенс угла наклона прямой к оси Ox**

Решение: *tgα=*. Подставляя числа *tgα=*.

Алгоритм генерации задания программой:

1. Прямая задается случайным образом
2. Ученик вводит ответ
3. Программа сравнивает введенный учеником ответ и *tgα=*.

constructor TQ002.Create;

begin

inherited Create;

Format := TAnswerFormatNumber.Create('Тангенс:');

Line := GenerateLine2D;

// Для данного вопроса B не может быть равно 0

if Line.b = 0 then

Line.b := 1;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти тангенс угла наклона прямой к оси Ox';

end;

function TQ002.Check(answer : TAnswer) : Boolean;

var

tangent : Extended;

begin

tangent := (answer as TAnswerNumber).Value;

// Сравним введенное значение с tan(-A/B)

Result := IsValueEq(tangent, Tan(-Line.a/Line.b));

end;

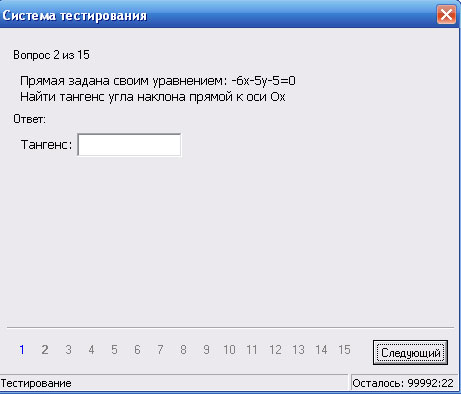


Рис. 7

1. **Составить уравнение в отрезках.**

Решение: В общем уравнении прямой , то, разделив на *–С,* получим  - это и есть уравнение прямой в отрезках. Подставляя числа получаем: 

Алгоритм генерации задания программой:

1. Прямая задается случайно
2. Программа подсчитывает значения для -A/C и -B/C и сравнивает с введенными учеником ответами.

constructor TQ003.Create;

begin

inherited Create;

Format := TAnswerFormat.Create(SegmentEq);

Line := GenerateLine2D;

if Line.c = 0 then

Line.c := 1;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Составить уравнение в отрезках';

end;

function TQ003.Check(answer : TAnswer) : Boolean;

var

p : TPoint2D;

begin

p := (answer as TAnswerPoint2D).Value;

// Сравним x c -A/C и y с -B/C

Result := IsValueEq(-Line.a/Line.c, p.x) and

IsValueEq(-Line.b/Line.c, p.y);

end;

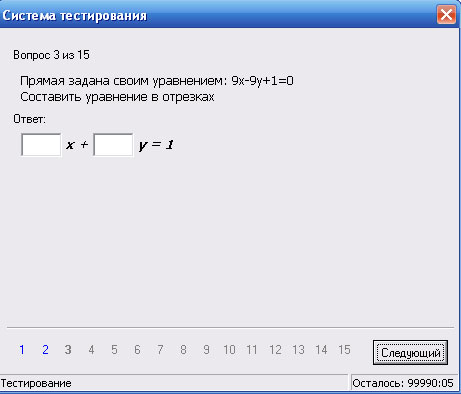


Рис. 8

1. **Найти нормальный вектор и направляющий вектор.**

# Решение: Пусть прямая задана общим уравнением Ax+By+C=0 в прямоугольной декартовой системе координат. Тогда мы знаем, что вектор  – направляющий вектор прямой,  – ее нормальный вектор.

Алгоритм генерации задания программой:

1. Прямая задается случайным образом.
2. Программа ищет нормальный вектор  и направляющий вектор 
3. Ученик вводит ответ для нормального вектора 
4. Программа сравнивает ответ ученика и свой ответ.

constructor TQ004.Create;

begin

inherited Create;

Format := TAnswerFormat.Create(TwoVector2D);

Line := GenerateLine2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти нормальный вектор и направляющий вектор';

end;

function TQ004.Check(answer : TAnswer) : Boolean;

var

vn, vd : TVector2D; // нормальный и направляющий вектор

begin

vn := (answer as TAnswerTwoVector2D).vec1;

vd := (answer as TAnswerTwoVector2D).vec2;

Result := IsNormalVector(vn, Line) and IsDirectionVector(vd, Line);

end;

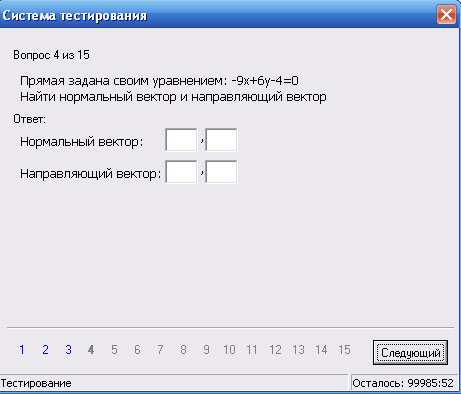


Рис. 9

1. **Найти отрезки, отсекаемые прямой на осях координат.**

Решение: Точки пересечения прямой с осями координат с осью *Ox*

*,* с осью *Oy* , то есть отрезок, отсекаемый прямой на оси *Ox* имеет координаты концов *(0,0),(,0).* А на оси *Oy: (0,0 ),(0, )*

Алгоритм генерации задания программой:

1. Прямая задается с учетом, того, что прямая недолжна быть параллельной ни одной из осей координат и не должна проходить через начало координат.
2. Ученик вводит свои значения в соответствующие поля.
3. Программа сравнивает введенные учеником ответы с подсчитанными ей самой ответами:

координаты концов отрезков (0,0), (-С/A,0) – на оси OX

координаты концов отрезков (0,0), (0, -C/B) – на оси OY

constructor TQ005.Create;

begin

inherited Create;

Format := TAnswerFormat.Create(TwoSegment2D);

Line := GenerateLine2D;

{ Прямая на должна быть параллельна оси x }

if Line.b = 0 then

Line.b := 1;

{ Прямая на должна быть параллельна оси y }

if Line.a = 0 then

Line.a := 1;

{ Прямая не должна проходить через начало координат }

if Line.c = 0 then

Line.c := 1;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти отрезки, отсекаемые прямой на осях координат';

end;

function TQ005.Check(answer : TAnswer) : Boolean;

var

x, y : TSegment2D;

nullp : TPoint2D; // Точка с координатами 0,0

sx, sy : TSegment2D;

begin

nullp.x := 0;

nullp.y := 0;

x := (answer as TAnswerTwoSegment2D).seg1;

y := (answer as TAnswerTwoSegment2D).seg2;

{ Пересечение с осю Ox - (0,0) - (-C/A, 0) }

sx.a := nullp;

sx.b.x := -Line.c/Line.a;

sx.b.y := 0;

{ Пересечение с осю Oy - (0,0) - (0, -C/B) }

sy.a := nullp;

sy.b.x := 0;

sy.b.y := -Line.c/Line.b;

Result := IsValueEq(x, sx) and IsValueEq(y, sy);

end;

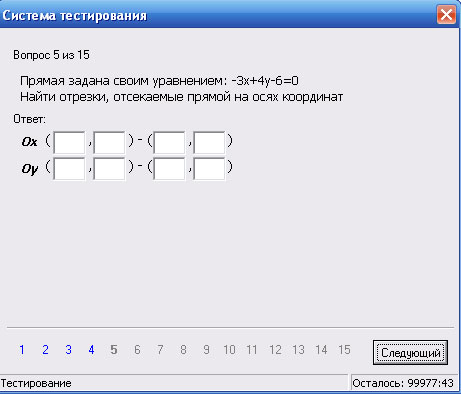


Рис. 10

1. **Записать каноническое уравнение прямой.**

Решение: Возьмем точку, принадлежащую прямой *l*, например A(0, 5/4)

Имеет вид: , где  - координаты точки принадлежащей этой прямой, а *m* и *n* – коэффициенты направляющего вектора. Получаем каноническое уравнение:



Формула для вывода канонического уравнения:



Алгоритм генерации задания программой:

1. Прямая задается случайно.
2. Ученик вводит координаты точки, принадлежащей этой прямой и ее нормальный вектор в соответствующие поля.
3. Верность ответов для координат точки проверяется аналогично заданию 1, а для нормального вектора, аналогично заданию 4.

constructor TQ006.Create;

begin

inherited Create;

Format := TAnswerFormat.Create(CanonicalLine2DEq);

Line := GenerateLine2D;

if Line.b = 0 then

Line.b := 1;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Записать каноническое уравнение прямой';

end;

function TQ006.Check(answer : TAnswer) : Boolean;

var

x, y, m, n : Extended;

p0 : TPoint2D;

p : TPoint2D;

begin

x := (answer as TAnswerParamLine2DEq).x;

y := (answer as TAnswerParamLine2DEq).y;

m := (answer as TAnswerParamLine2DEq).m;

n := (answer as TAnswerParamLine2DEq).n;

if IsValueNull(m) or IsValueNull(n) then

begin

Result := False;

Exit;

end;

{ Проверим, что точки x и y лежат на прямой }

p0.x := x;

p0.y := y;

if not IsPointOnLine(Line, p0) then

begin

Result := False;

Exit;

end;

{ Возьмем точку пересечения прямой с осью 0x(0, -C/B) }

p.x := 0;

p.y := - Line.c / Line.b;

{ И проверим равенство x-x0/m = y-y0/n }

Result := IsValueEq((p.x - x)/m , (p.y - y)/n);

end;

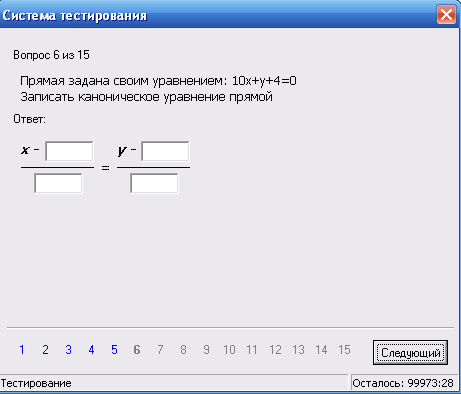


Рис. 11

1. **Записать параметрическое уравнение**

Решение:

*x=x0+mt*

*y=y0+nt,* где (*x0, y0 )* – координаты точки, принадлежащей этой прямой, а *m* и *n* – коэффициенты направляющего вектора.

Возьмем точку A(0,5/4), тогда

*x=-4t*

*y=5/4+3t*

Алгоритм генерации задания программой:

1. Прямая задается случайно.
2. Ученик вводит координаты точки, принадлежащей этой прямой и ее нормальный вектор в соответствующие поля.
3. Верность ответов для координат точки проверяется аналогично заданию 1, а для нормального вектора, аналогично заданию 4.

constructor TQ007.Create;

begin

inherited Create;

Format := TAnswerFormat.Create(ParamLine2DEq);

Line := GenerateLine2D;

if Line.a = 0 then

Line.a := 1;

if Line.b = 0 then

Line.b := 1;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Записать параметрическое уравнение';

end;

function TQ007.Check(answer : TAnswer) : Boolean;

var

p : TPoint2D;

v : TVector2D;

begin

p.x := (answer as TAnswerParamLine2DEq).x;

p.y := (answer as TAnswerParamLine2DEq).y;

v.a := (answer as TAnswerParamLine2DEq).m;

v.b := (answer as TAnswerParamLine2DEq).n;

{ Проверим, что x и y лежат на прямой

и m и n являются коэффициентами направляющего вектора }

Result := IsPointOnLine(Line, p) and IsDirectionVector(v, Line);

end;

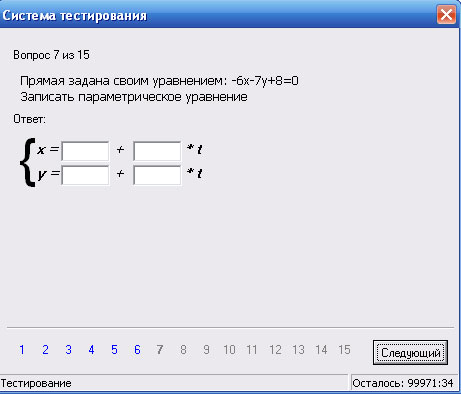


Рис. 12

1. **Найти расстояние от точки M0 (3, 5) до прямой**

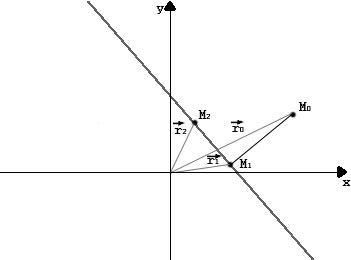
****

Рис. 13

Решение:

Опустим перпендикуляр к прямой из *M0.* Обозначим точку *M1(x1,y1)*. Пусть *M2(x2,y2) –* некоторая точка на прямой *l,* отличная от *M1(x1,y1)*. Тогда уравнение прямой запишем в нормальной векторной форме *A(x- x2)+B(y- y2)=0,* где *С= -A x2-By2*, а  - вектор нормали. Или в векторной форме *(**)=0.* Очевидно справедливо следующие равенство, причем , поэтому . Умножив обе части равенства на вектор , получим (Рис.13).

Так как точка *M1* лежит на прямой *l*, то , и следовательно . Подставляя в исходное равенство, найдем . Отсюда и *ρ(M0,l)=.* Переходя к координатной форме записи и учитывая, что *С= -A x2-By2*, , имеем *ρ(M0,l)==.*

Подставляя числа получаем *ρ(M0,l)=*

Алгоритм генерации задания программой:

1. Случайно задается точка.
2. Для генерации прямой используются заранее описанные пифагоровы числа (тройки)

C задается случайно.

1. Программа считает расстояние по формуле ρ(M0,l)=.
2. Программа сравнивает получившийся ответ с ответом введенным учеником, если они совпадают, значит, ученик ответил верно.

constructor TQ008.Create;

begin

inherited Create;

Format := TAnswerFormatNumber.Create('Расстояние');

Line := GenerateLine2DPi;

Point := GeneratePoint2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти расстояние от точки М' + FormatPoint2D(Point) +

' до прямой';

end;

function TQ008.Check(answer : TAnswer) : Boolean;

var

d : Extended;

begin

d := (answer as TAnswerNumber).Value;

Result := IsValueEq(GetDistance(Point, Line), d);

end;

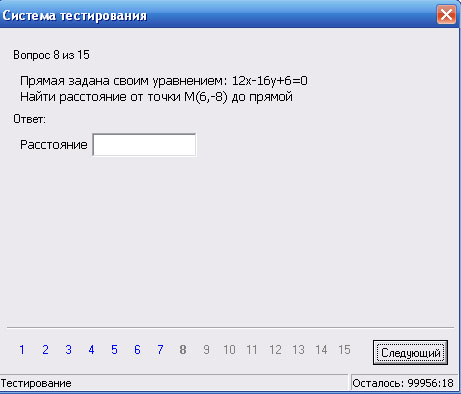


Рис. 14

1. **Составить уравнение перпендикуляра, опущенного на эту прямую из точки (7,10).**

Решение: Направленный вектор искомой прямой, это вектор нормали исходной прямой.

, где *(x, y),* координаты заданной точки, а *(m,n)*коэффициенты вектора нормали прямой *l.*

** - каноническое уравнение прямой.

*4x-3y+2=0 -* Общее уравнение прямой.

Алгоритм генерации задания программой:

1. Прямая задается случайно.
2. Точка задается случайно.
3. Прямая проходящая через точку является перпендикуляром для исходной прямой, если точка принадлежит искомой прямой и направляющий вектор искомой прямой совпадает с нормальным вектором исходной прямой.

constructor TQ009.Create;

begin

inherited Create;

Format := TAnswerFormatLine2D.Create('Перпендикуляр');

Line := GenerateLine2D;

Point := GeneratePoint2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Составить уравнение перпендикуляра, опущенного на эту прямую из точки '+

FormatPoint2D(Point);

end;

function TQ009.Check(answer : TAnswer) : Boolean;

var

l : TLine2D;

begin

l := (answer as TAnswerLine2D).Value;

{ Прямая l проходящая через точку Point является перпендикуляром

для прямой Line, если Point принадлежит прямой l и

направляющий вектор l совпадает с нормальным вектором Line }

Result := IsPointOnLine(Point, l) and

IsLinesNormal(Line, l);

end;

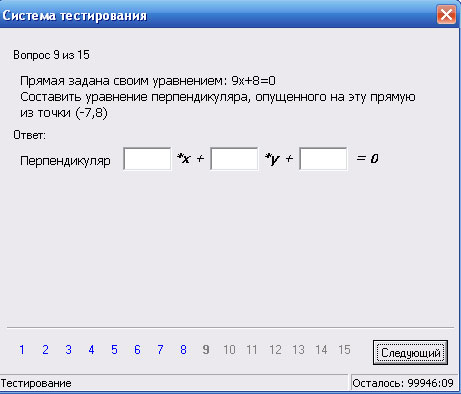
**

Рис. 15

1. **Составить уравнение прямой, проходящей через точку (4,-6) параллельно этой прямой.**

Решение: Направляющий вектор у искомой прямой будет совпадать с направляющим вектором у исходной прямой. Тогда каноническом виде прямая будет иметь вид , а в общем *3x+4y+12=0.*

Алгоритм генерации задания программой:

1. Прямая задается случайно.
2. Точка задается случайно.
3. Прямая проходящая через точку является перпендикуляром для исходной прямой, если точка принадлежит искомой прямой и направляющие векторы исходной и искомой прямой совпадают.

constructor TQ010.Create;

begin

inherited Create;

Format := TAnswerFormatLine2D.Create('Прямая');

Line := GenerateLine2D;

Point := GeneratePoint2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Составить уравнение прямой, проходящей через точку ' + FormatPoint2D(Point) +

' параллельно этой прямой';

end;

function TQ010.Check(answer : TAnswer) : Boolean;

var

l : TLine2D;

begin

l := (answer as TAnswerLine2D).Value;

{ Прямая l походящяя через точку Point является параллельной

прямой Line, если Point принадлежит прямой l и

направляющий вектор l совпадает с направляющим вектором Line }

Result := IsPointOnLine(Point, l) and

IsLinesParallel(Line, l);

end;

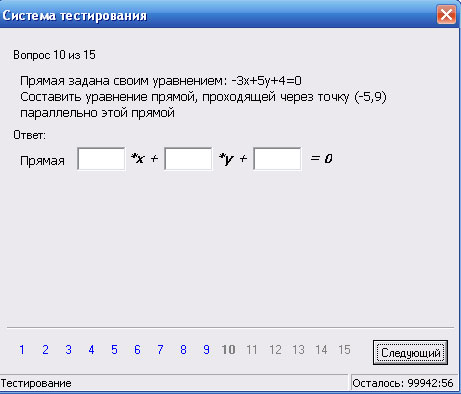
**

Рис. 16

1. **Найти координаты точки пересечения данной прямой с прямой y=-3x+1**

Решение: Составляем систему уравнений

*A1+B1y+C1=0*

*A2+B2y+C2=0*

Решая уравнение относительно переменный *x* и *y,* получаем  

Подставляя числа получим точку с координатами ()

Алгоритм генерации задания программой:

1. Задаются случайно 2 прямые

*L1:A1\*x+B1+C1=0*

*L2:A2\*x+B2+C2=0*

1. Проверяется, чтобы их нормали были не коллинеарные

*A1\*B1≠A2\*B1* , если нормали исходных прямых коллинеарные, то генерируется другая прямая *L2.*

1. Ученик вводит ответ.
2. Программа проверяет, принадлежит ли эта точка одновременно обеим прямым, если да, то ученик ответил правильно.

constructor TQ011.Create;

begin

inherited Create;

Format := TAnswerFormatPoint2D.Create('Точка пересечения');

Line := GenerateLine2D;

if Line.a = 0 then

Line.a := 1;

CrossLine := GenerateLine2D;

if CrossLine.a = 0 then

CrossLine.a := 1;

{ Линии не должны пересекаться }

if IsLinesParallel(Line,CrossLine) then

CrossLine.a := -CrossLine.a;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти координаты точки пересечения данной прямой с прямой ' + FormatLine2D(CrossLine);

end;

function TQ011.Check(answer : TAnswer) : Boolean;

var

p : TPoint2D;

begin

p := (answer as TAnswerPoint2D).Value;

Result := IsValueEq(GetCrossPoint(Line, CrossLine), p);

end;

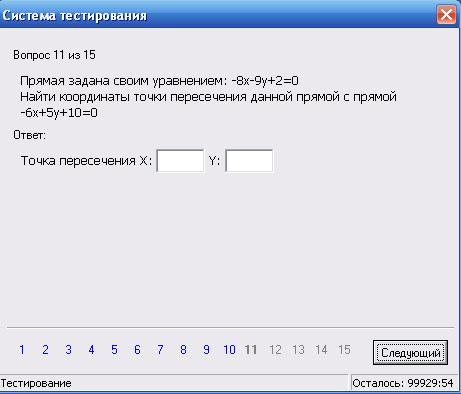


Рис. 17

1. **Найти проекцию точки M(4,-6) на данную прямую**

Решение: Постоим прямую по точке *M(4,-6)* и нормали исходной прямой . Уравнение этой прямой *4x-3y-34=0.*И найдем координаты пересечения этих прямых по формулам  . Подставляя числа получаем точку с координатами (-22)

Алгоритм генерации задания программой:

1. Задается случайно прямая.
2. Задается случайно точка.
3. Программа порождает перпендикулярную к исходной прямой по формуле

*A\*x-B\*y+B\*y1-A\*x1=0*

1. Ученик вводит свой ответ в соответствующие поля.
2. Программа проверяет принадлежит ли точка, введенная учеником исходной прямой и перпендикулярна ли прямая, построенная по исходной точке и точке проекции если, да, то ученик ответил верно.

constructor TQ012.Create;

begin

inherited Create;

Format := TAnswerFormatPoint2D.Create('Проекция');

Line := GenerateLine2D;

Point := GeneratePoint2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Найти проекцию точки М' + FormatPoint2D(Point) + ' на данную прямую';

end;

function TQ012.Check(answer : TAnswer) : Boolean;

var

p : TPoint2D; // Введенная точка

lp : TLine2D; // Прямая проходящяя через две точки

begin

p := (answer as TAnswerPoint2D).Value;

{ Получим прямую, проходящую через две точки }

lp := GetLineFromPoints(p, Point);

{ Проекция верна, если введенная точка лежит на прямой Line

и прямая LP является перпендикуляром к Line }

Result := IsPointOnLine(p, Line) and IsLinesNormal(lp, Line);

end;

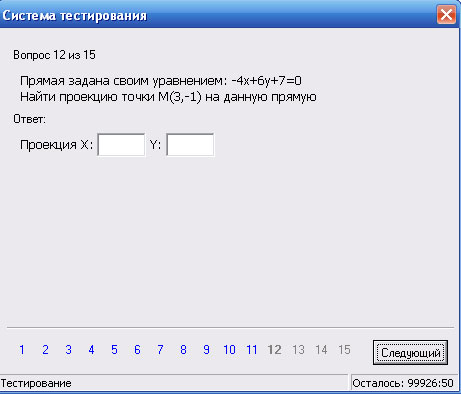


Рис. 18

1. **Составить уравнение прямой, проходящей через точку (1,1) под углом 45°**

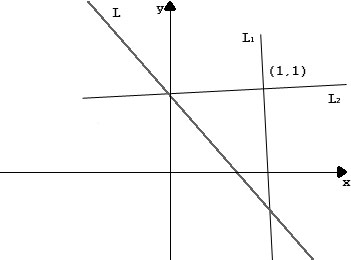
****

Рис. 19

Решение: Тангенс угла между прямыми можно вычислить с помощью формулы разности аргументов для тангенса, тогда запишем уравнение исходной прямой в виде уравнения с угловыми коэффициентами *y=*, а искомое . *φ(y,y1)*. tg *φ=*. Подставляя числа получаем *k1=.* Зная, что прямая проходит через точку *(1,1)* находим *b=.* Отсюда уравнение искомой прямой , а в общем виде

*x-7y+6=0*

Также можно провести еще одну прямую, проходящую через точку *(1,1)* и расположенную под углом 45° к исходной прямой. Уравнение этой прямой находится аналогично первой. α*(y,y2)*. tg*α=*. Подставляя числа получаем *k2=7.* Зная, что прямая проходит через точку *(1,1)* находим *b=-6.* Отсюда уравнение искомой прямой , а в общем виде *7x-y-6=0* (Рис.19)

Алгоритм генерации задания программой:

1. Задается случайно уравнение исходной прямой.
2. Задается случайно точка.
3. Ученик вводит ответ в соответствующие поля.
4. Программа проверяет, принадлежит ли заданная точка прямой, которую ввел ученик и проверяет *tg* угла наклона между исходной прямой и прямой, которую нашел ученик, если tg *φ=1*, то ответ верен.

constructor TQ013.Create;

begin

inherited Create;

Format := TAnswerFormatLine2D.Create('Прямая');

Line := GenerateLine2D;

Point := GeneratePoint2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Составить уравнение прямой, проходящей через точку ' + FormatPoint2D(Point) +

' под углом 45 градусов.';

end;

function TQ013.Check(answer : TAnswer) : Boolean;

var

l : TLine2D;

begin

l := (answer as TAnswerLine2D).Value;

{ Точка должна лежать на прямой и тангенс между углами должен быть 1}

Result := IsPointOnLine(Point, l) and

IsValueEq(abs(GetLinesTangent(Line, l)), 1);

end;

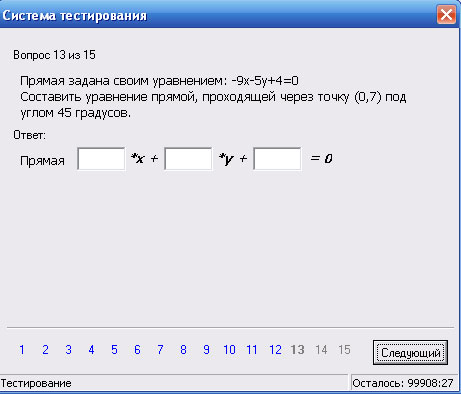


Рис. 20

1. **Составить уравнение прямой, отстоящей от данной на расстояние d=4.**

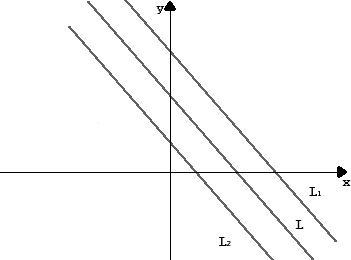


Рис. 21

Решение: Воспользуемся формулой расстояния от точки до прямой *d=.* Подставляя числа, получаем



**

Задача разбивается на 2 случая (Рис.21)

1. 
2. 

Untitled-1

Р

Алгоритм генерации задания программой:

1. Коэффициенты A и B задаются из заранее подобранных пифагоровых чисел (троек). *C* задается случайно.
2. Ученику предлагается ввести коэффициенты уравнения в соответствующие поля.
3. Программа проверяет на параллельность прямую, введенную учеником и исходную прямую.
4. И подсчитывает расстояние между ними.

constructor TQ014.Create;

begin

inherited Create;

Format := TAnswerFormatLine2D.Create('Прямая');

Line := GenerateLine2DPi;

d := abs(GenerateNumber);

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(Line) + #13#10 +

'Составить уравнение прямой, отстоящей от данной на расстоянии d=' +

FloatToStr(d);

end;

function TQ014.Check(answer : TAnswer) : Boolean;

var

l : TLine2D;

begin

l := (answer as TAnswerLine2D).Value;

{ Прямые должны быть параллельны }

Result := IsLinesParallel(l, Line) and

IsValueEq(GetDistance(l, Line), d);

end;

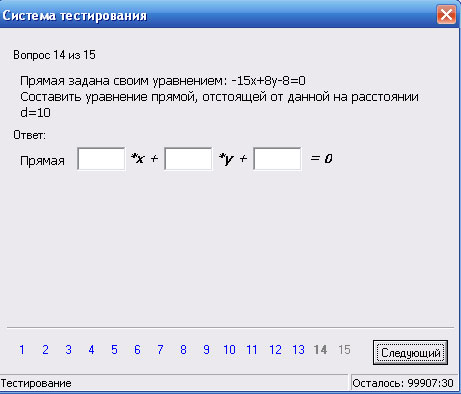


Рис. 22

1. **Пересекает ли эта прямая отрезок , если A(-2,6), B(3,-4)**

Решение: *f(-2,6)* >0

*f(3,-4)* <0

Если знаки совпадают, то отрезок не пересекает прямую, иначе пересекает.

Алгоритм генерации задания программой:

1. Задается случайно исходная прямая.
2. Задается случайно 2 точки *A(x1,y1)* и *B(x2,y2)*
3. *f(x1,y1)* программа проверяет ее знак >0 или <0

*f(x2,y2)* программа проверяет ее знак >0 или <0

Если знаки совпадают, то отрезок не пересекает прямую, иначе пересекает.

1. Программа проверяет ответ ученика.

constructor TQ015.Create;

var

l : TLine2D;

s : TSegment2D;

v1, v2 : Extended;

begin

inherited Create;

Format := TAnswerFormat.Create(Bool);

l := GenerateLine2D;

s.a := GeneratePoint2D;

s.b := GeneratePoint2D;

Text := 'Прямая задана своим уравнением: ' + FormatLine2D(l) + #13#10 +

'Пересекает ли эта прямая отрезок [AB], если A' +

FormatPoint2D(s.a) + ', B' + FormatPoint2D(s.b);

{ Подставим в исходное уравнение координаты точек }

v1 := l.a\*s.a.x + l.b\*s.a.y + l.c;

v2 := l.a\*s.b.x + l.b\*s.b.y + l.c;

{ Если знаки разные - то отрезок пересекает прямую }

Answer := ((v1\*v2) < 0);

end;

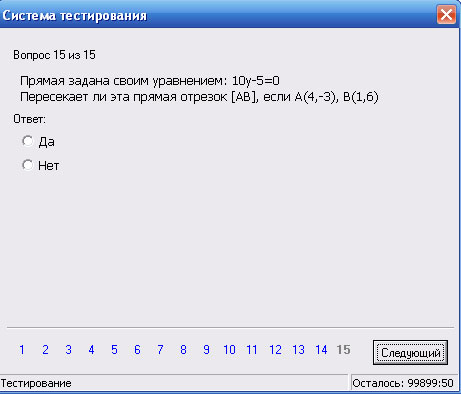


Рис. 23

Вывод результатов

После того, как ученик проделал все предложенные ему задания или по окончанию времени программа выводит результаты (Рис.24). Результаты организованны в структурном виде и заносятся в файл results.txt. В них (результатах) указывается численно и в процентном соотношении на сколько вопросов ученик ответил правильно, на сколько неправильно, сколько вопросов ученик пропустил и сколько времени потратил на тестирование.

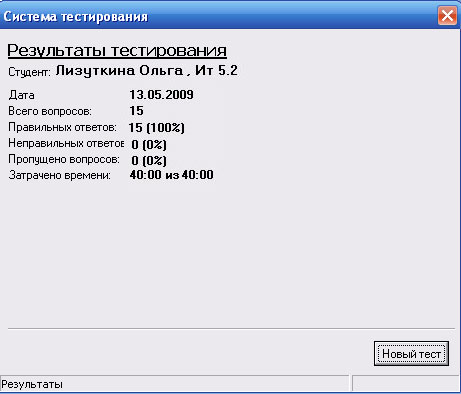


Рис. 24

Заключение

Выполнена поставленная задача разработки интерфейса, который позволяет ученику легко ориентироваться в приложении «Система тестирования» при его прохождении. А преподавателю иметь возможность проведения, как кратковременных проверочных работ с малым количеством заданий, так и полноценных итоговых тестирований. Причем исключена возможность использования преподавательских настроек учениками. Это достигнуто установкой пароля на задание (изменение) настроек тестирования.

Разработаны методы генерации заданий. Трудоемкий процесс подбора задний автоматизирован. Благодаря этому преподавателю не нужно тратить время на разработку различных вариантов тестирования.

Благодаря открытости тестирования, практически полностью исключена возможность угадывания ответов тестируемым. Исходные значения для теста каждый раз новые, что исключает возможность списывания.

Генерация исходных данных реализована таким образом, что при решении заданий тратится минимум времени на побочные вычисления. А именно при решении заданий получаются рациональные ответы. А проблема экономии времени актуальна как для преподавателей, так и для учеников.

Разработаны уникальные методики проверки правильности ответов. Причем особенность программы такова, что она не решает задания параллельно с тестируемым учеником, а проверяет правильность введенных им ответов. Практически каждое задание имеет как индивидуальную схему генерации заданий, так и уникальную систему проверки правильности введенных ответов.

Список используемой литературы

1. Математическая энциклопедия (в 5-и томах), Москва, «Советская Энциклопедия», 1982 г.
2. Прасолов В.В. Задачи по планиметрии. — 4-е изд., дополненное — М.: Изд-во Московского центра непрерывного математического образования, 2001.
3. **Справочник по элементарной математике. Геометрия, тригонометрия, векторная алгебра.** Под редакцией П. Ф. Фильчакова. 1967 г.
4. Ефимов Н. В. Краткий курс аналитической геометрии: Учебн. пособие. — 13-е изд., стереот. — М.: ФИЗМАТЛИТ, 2005. — 240 с.
5. Фленов М.Е. Библия Delphi, БХВ-Петербург, 2005. — 880с.
6. Сухарев М.В., Основы Delphi. Профессиональный подход, Наука и техника, 2004. — 600с.
7. А. Я. Архангельский, [100 компонентов общего назначения библиотеки Delphi 5](http://progbook.net/delphi/213-100-komponentov-obshhego-naznachenija.html), Бином, 1999 — 272 с.
8. Бобровский С.И., Delphi 7: Учебный курс, Питер, 2005 — 736с.
9. Кожушко В.В., Веременюк В.В., Практикум по математике: Подготовка к тестированию и экзамену Изд. 6-е, доп., ТетраСистемс, 2009 г, — 200с.
10. Шаповалов А.И., Зинченко Ю.Е., [Методика проверки хода решения математических задач](http://masters.donntu.edu.ua/2007/fvti/shapovalov/library/article1.htm). Доклад на первой Всеукраинской научно-технической конференции "Информационные процессы и технологии" г. Севастополь, 2007 г.
11. Требования к Составлению тестовых заданий URL http://[www.nspu.net/uploads/media/TREBOVANIA.doc](http://www.nspu.net/uploads/media/TREBOVANIA.doc)
12. Ермакова М.Г., Андреева Л.Е. Вопросы разработки тестирующих программ. //Информатика и образование. – 1997. №3.
13. Пак Н.И., Филиппов В.В. О технологии создания компьютерных тестов. //информатика и образование. – 1997. №5.
14. Родионов Б.У., Татур А.О. Стандарты и тесты в образовании. М., 1995.
15. Чернигин А.Н. Инструментальная система контроля знаний.// Информатика и образование. – 1999. №10.

Приложение

[Здесь был программный код]

