Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Московский государственный университет технологии и управления имени К.Г. Разумовского (Первый казачий университет)
Университетский колледж информационных технологий

Специальность 09.02.03 Программирование в компьютерных системах

КУРСОВОЙ ПРОЕКТ

Модуль ПМ.01 Разработка программных модулей программного обеспечения для компьютерных система
МДК.01.02 Прикладное программирование

на тему «РАЗРАБОТКА ПРОГРАММЫ «Пинбол»

Пояснительная записка УКИТ
09.02.03.2015 304.021ПЗ

Группа *П-304*

Студент _____ Орищенко А.Э

(*личная подпись*)

Руководители проекта _____ Глускер А.И.

(*личная подпись*)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Суть курсового проекта разработка программы «Пинбол». Программа должна создать два игровое поле, осуществлять движение шара.

Данная программа актуальна тем, что игра развивает логику и мышление игрока. Основной целью курсового проектирования является самостоятельная реализация программногопродукта.

Задачи курсового проекта приобретение знаний по курсу прикладного программирования, разработка программы на выбранном языке программирования, овладение методами отладки и тестирования программ.

Структура пояснительной записки состоит из этапов разработки программного продукта, оформленных по ГОСТам и объединенных в одном документе. Состав пояснительной записки:

-Титульный лист

-Содержание

Включает в себя название разделов и номера их страниц.

-Введение

-Основная часть

1. Спецификация

Этот этап нужен для формирования структуры программного продукта.

2. Программа и методика испытаний

Этот этап нужен для проведения приемочных испытаний и разрабатывается на основе документации спецификации.

3. Технический проект программного изделия

Этот этап нужен для определения внутренних свойств программы и детализации внешних свойств. Процесс проектирования и его результаты, зависят не только от требований, изложенных в этапах выше, но и от выбранной модели процесса.

4. Реализация программного изделия на языке программирования

Этот этап нужен для создания работающей программы на выбранном языке программирования, в ходе которого осуществляется тестированиеи отладка программного продукта.

5. Тестирование программного продукта

Этот этап нужен для выявления степени соответствия готового программного продукта спецификации, разработанной на первом этапе

с помощью программы и методики испытаний, разработанной на втором этапе.

-Заключение

-Приложения

Основные методы, которые были использованы:

Анализ - представляет собой процессов или явлений на составные компоненты и предполагает их дальнейшее изучение.

Синтез – предполагает соединение нескольких свойств исследуемого объектав один компонент.

Структурное и модульное программирование. Принципы структурного программирования - управляющие структуры. Можно использовать только 3 типа управляющих структур:

- Следование (операторы)
- Условные конструкции
- Циклы

Программа представляет собой иерархию структур состоящих из таких структур:

if… then…

else…

запрещается использовать следующие операторы:

- goto – это объявления меток
- break – выход из внутреннего цикла
- continue – переход в завершающую частьцикла
- exit – выход из подпрограммы
- halt – выход из программы

Процедурное программирование – программа, написанная на процедурном языке, представляет собой последовательность команд, определяющих алгоритм решения задачи. Основная идея процедурного программирования – использование памяти для хранения данных. Основная команда – присвоение, с помощью которой определяется и меняется память компьютера.

ОСНОВНАЯ ЧАСТЬ

1. СПЕЦИФИКАЦИЯ

Данный этап нужен для формирования структуры программного продукта. Здесь описаны требования к функциональным характеристикам, требования к интерфейсу программы, требования к надежности, условия эксплуатации, требования к составу и параметрам технических средств, требования к исходным кодам, требования к программной документации.

1.1. Требования к функциональным характеристикам

1.1.1. Требования к составу выполняемых функций

1.1.1.1. Начало игры по нажатию кнопки пробел.

1.1.1.2. Вычисление траекторию полета с учетом абсолютно упругого отскока.

1.1.1.3. Игровой процесс включает в себя возможность управления битками с помощью стрелок.

1.1.2. Требования к интерфейсу программы

Интерфейс должен быть стандартным для операционной системы Windows.

1.2. Требования к надёжности.

Контроль за корректностью введённых данных.

1.3. Условия эксплуатации.

1.3.1. Минимальное количество пользователей – один человек, умеющий пользоваться клавиатурой.

1.3.2. Другие специальные требования к условиям эксплуатации не предъявляются.

1.4. Требования к составу и параметрам технических средств.

В состав технических средств должен входить компьютер, включающий:

- Процессор семейства intel (Intel Core i5 - 4690K);
- Клавиатуру;
- Видеокарту (Asus GeForce GTX670);
- Монитор;
- HDD или SSD-диск.

1.5. Требования к информационной и программной

Должны быть реализованы на языке Delphi. В качестве интегрированной среды разработки программы должна быть использована среда Lazarus(Delphi).

1.7. Программа должна работать под управлением операционной системы Windows XP.

1.8. Специальные требования

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса.

1.9. Программа должна работать под управлением операционной системы Windows XP.

1.10 Специальныетребования

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса.

1.11 .Требования к программнойдокументации.

1.11.1.Состав программнойдокументации

Состав программной документации долженвключать:

• техническое задание;
• пояснительную записку;
• текст программы;
• текст программы, осуществляющей автоматическое тестирование программы «Пинбол»;
• программу и методику испытаний.

1.10. Стадии и этапы разработки.

1.10.1. Стадии разработки.

1.10.1.1. Разработка осуществляется в тристадии:

• техническое задание;
• рабочее проектирование;
• рабочий проект.

1.10.2. Этапы разработки.

На стадии технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1.разработка программы;
2.разработка программной документации;
3.испытания программы.

На стадии внедрения должен быть выполнен этап разработки -подготовка и передача программы.

1.11. Порядок контроля и приемки.

Приемосдаточные испытания должны проводиться в соответствии с программой и методикой испытаний, разработанной, согласованной и утвержденной не позднее 31 декабря.

2. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Этот этап нужен для проведения приемочных испытаний и разрабатывается на основе документации спецификации. Здесь описаны программные средства, порядок проведения испытаний, методы испытаний, метод проверки требований к составу программной документации, метод проверки требований к исходным кодам.

2.1 Программные средства, используемые при проведении испытаний

В состав программных средств входит:

- лицензионная копия операционной системы Windows XP(любой версии), локализованная для работы в Российской Федерации;
- Lazarus (вариант, предназначенный для работы в среде Windows).

2.2 Порядок проведения испытаний

2.2.1 Подготовка к проведению испытаний заключается в обеспечении наличия компьютера, описанного в п. 1.5 (спецификации), и программных средств, указанных в пункте выше (п.2.1), установленных на этом компьютере

2.2.2 Ход проведения испытаний документируется в протоколе, где указывается перечень проводимых испытаний, результат каждого испытания и возможно замечания.

2.2.3 Состав испытаний:

2.2.3.1 Проверка состава программной документации в соответствии с методом, описанном в п. 3.2

2.2.3.2 Проверка требований кпрограмме

Проверка обеспечения требований к программе (п. 1.1.1(спецификации))в соответствии с методом, описанным в п. 3.1

2.2.3.3 Проверка требований к программной документации

2.2.3.3.1 Проверка пояснительной записки (п. 2 (спецификации)) в соответствии с методом, описанным в п. 3.3

2.2.3.3.2 Проверка текстов программ (п. 1.6.1(спецификации)) в соответствии с методом, описанным в п. 3.4

2.2.3.3.3 Проверка текстов программ (п. 1.6.2 (спецификации)) в соответствии с методом, описанным в п. 3.5

3. Методы испытаний

3.1. Список тестов:

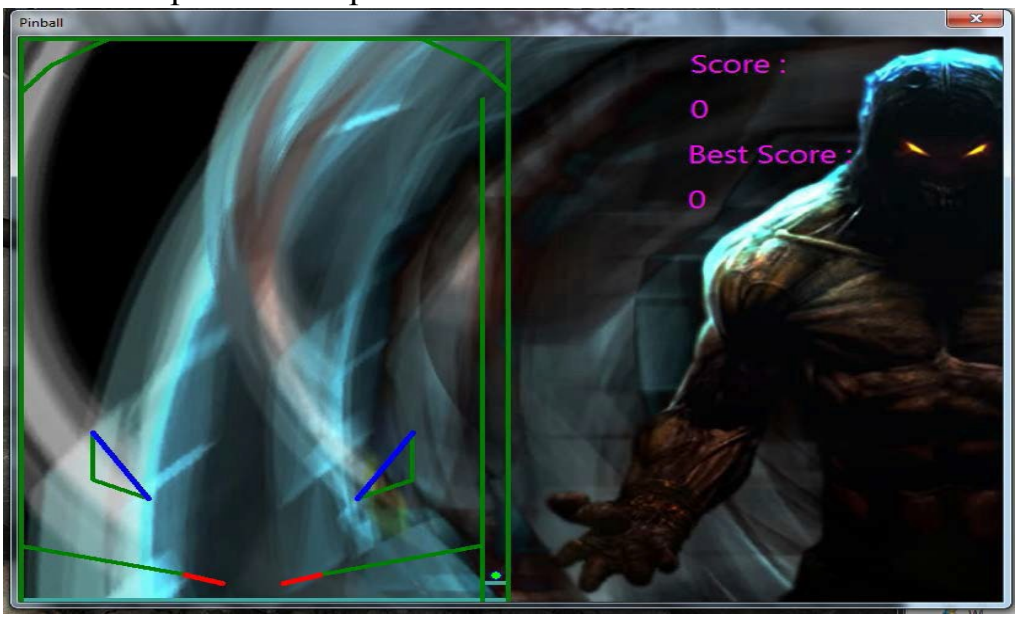
3.1.1. Запустите программу и начните игру путем нажатием клавиши пробел. Далее, если шар вылетел значит функция запуска шара работает.

3.1.2. Сделайте то же самое, что в пункте один, но используйте клавиши, проверьте работоспособность флипперов.

3.1.3. Записать видео отскока, и проверить угол отскока.

3.1.4. Учитывать силу притяжения, проверить траекторию.

3.1.5. Отображение игрового поля



3.2. Метод проверки требований к пояснительной записке

Проверка состоит из следующихэтапов:

- проверка наличия блок-схемы (блок-схем) в пояснительной записке;
- проверка соблюдения требований ГОСТ 19.701-90 для каждой блок-схемы;
- проверка соблюдения локальных стандартов для блок-схем;
- проверка соответствия каждой блок-схемы алгоритму, закодированному в программе. Проверка соблюдения требований ГОСТ 19.701-90 состоит из следующих работ:
- проверка использования только тех символов, которые указаны как применимые к схемам программ в п. 5 ГОСТ19.701-90;
- проверка соответствия символов их назначению (экспертная оценка лица, проводящего испытания);
- проверка правильности выполнения соединения линий (п. 4.2.3 ГОСТ 19.701-90);
- проверка того, что линии потока управления, выходящие из символа «решение» подписана (п. 4.3.1.2 ГОСТ 19.701-90); Проверка соблюдения локальных стандартов для блок-схем состоит из следующих работ:
- проверка того, что все символы имеют одинаковые размеры;
- проверка того, что отношение ширины к высоте составляет 2 к 1 для каждого символа.

- проверка того, что подписи к линиям не находятся на самих линиях. Проверка соответствия каждой блок-схемы алгоритму, закодированному в
- программе, осуществляется путем экспертной оценки лицом, осуществляющим проведение испытаний. В случае, если все вышеприведенные проверки прошли успешно, в протокол заносится запись: «Специальные требования к пояснительной записке» – соответствует; в противном случае «Специальные требования к пояснительной записке» – не соответствует.

3.3. Метод проверки требований к исходным кодам

Изложенный ниже метод применяется ко всем файлам, содержащим исходный текст, и входящим в состав программной документации по отдельности. Для каждого файла вносится в протокол запись: «Требования к исходным кодам для файла ####» – соответствует/не соответствует (где вместо #### указывается название файла). Проверка состоит из следующих этапов:

- Наличие комментария в начале файла, содержащего автора работы, номера задания и варианта, краткой формулировки задания (или его части)
- Для каждой подпрограммы наличие комментария, содержащего описание ее работы. Достаточность этого комментария для возможности использовать подпрограмму в других программах (без изучения собственно текста подпрограммы).
- Для каждой глобальной переменной указание ееназначения.
- Для всех переменных, кроме переменных цикла, использование «говорящих» названий
- Для всех подпрограмм использование говорящих названий.
- Использование одного оператора на одной строке программы.
- Количество пробелов перед строкой программы должно соответствовать уровню вложенности .
- Begin и End, соответствующие друг другу, располагаются строго с одной и той же позиции по вертикали.
- Отсутствие операторов goto, break, continue; функций halt и exit.
- Проверка того, что вместо явно указанных значений чисел, в тексте программы используются константы.

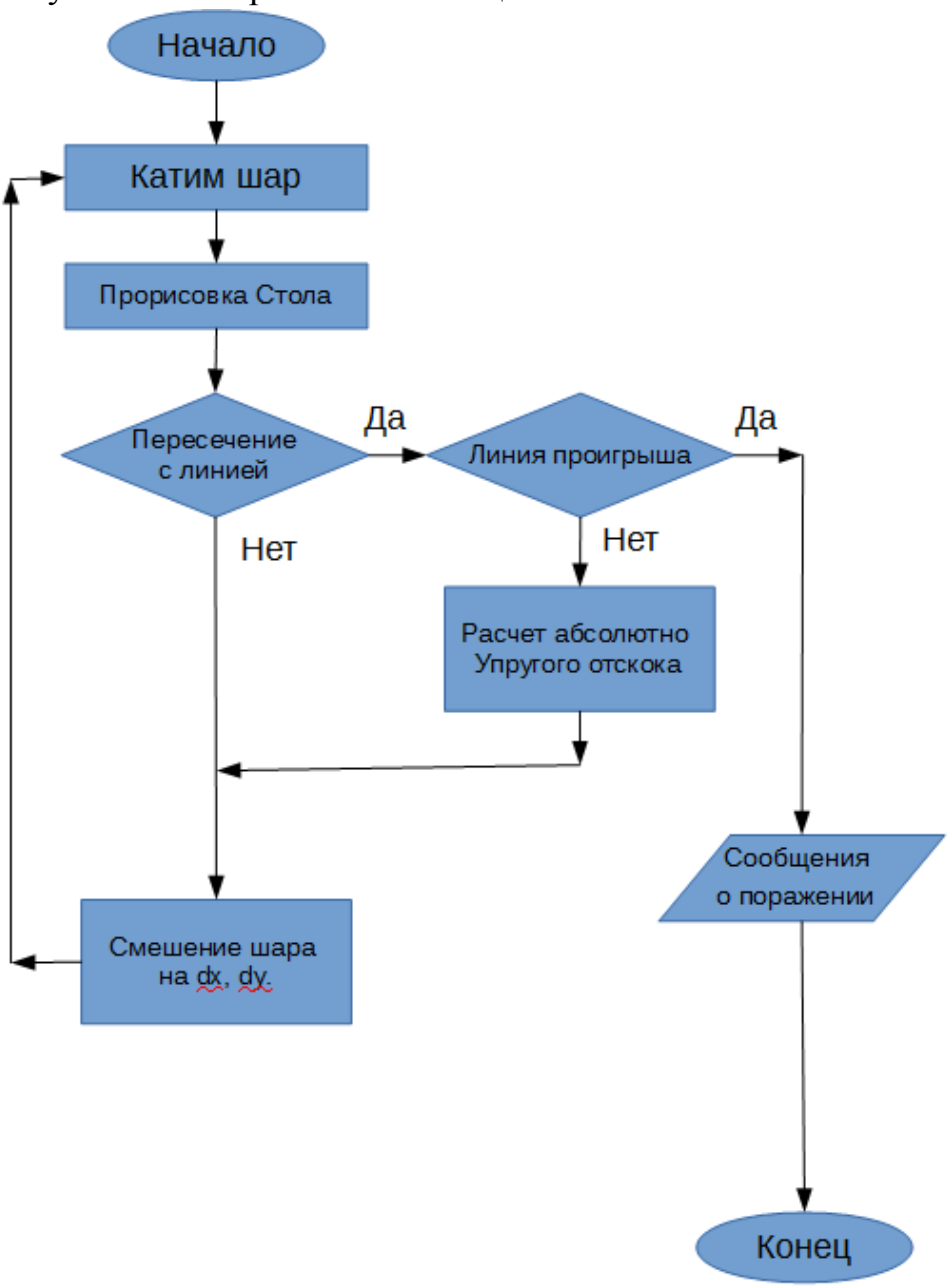
4. ТЕХНИЧЕСКИЙ ПРОЕКТ ПРОГРАММНОГО ИЗДЕЛИЯ

Этот этап нужен для определения внутренних свойств программы и детализации внешних свойств. Процесс проектирования и его результаты, зависят не только от требований изложенных в этапах выше, но и от выбранной модели процесса. Здесь нарисована блок-схема по алгоритму хода компьютера и будет описано, какой язык программирования был выбран и программы, которые были использованы в ходе выполнения курсовогопроекта.

4.1. Технические характеристики

4.1.1. Описание алгоритма Блок-схема

Рисунок 1. Алгоритм “полного цикла качения”



4.1.2. Описание и обоснование выбора языка программирования и программных средств

Для разработки программного кода был выбран язык Delphi с использованием разработки Lazarus. Так как в этой среде я уже научился работать, я выбрал именно ее, мне в ней было удобнее всего написать такую программу.

Robodoc – единственная программа которая делает авто-документацию которую я знаю.

LibreOffice- ибо халва, и в нем можно нормально работать с блок-схемами.

LibreOffice Draw — рисование блок-схем.

Git – с помощью этой программы можно создавать коммиты, дабы не потерять нужную информацию, и при возможности использовать ветки для удобного отката до выбранногокоммита.

4.2. РЕАЛИЗАЦИЯ ПРОГРАММНОГО ИЗДЕЛИЯ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ

Этот этап нужен для создания работающей программы на выбранном языке программирования, в ходе которого осуществляется тестирование и отладка программного продукта. Здесь описаны ошибки и трудности, с которыми я сталкивался и их решения.

Была разработана программа «Pinball». Исходный код и Robodoc и Git будут приложены пояснительной записке. Ниже будут представлены некоторые ошибки, и описание как я их исправлял.

Некоторые ошибки:

4.3. Синтаксические ошибки (исправлял синтаксис)

4.3.1. Перепутал название переменных.

4.4. Некоторые трудности:

4.4.1. Были трудности в расчетах отражения шара от линий. (долго рассчитывал, и в итоге всёполучилось)

4.4.2. Были трудности в алгоритме расчета расстояния от шара до отрезка.

На этом этапе я разрабатывал программный продукт, и описал ошибки и трудности, которые у меня были.

5. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

Этот этап нужен для выявления степени соответствия готового программного продукта спецификации, разработанной на первом этапе, с помощью программы и методики испытаний, разработанной на втором. Здесь приведена таблица автоматического тестирования программного продукта.

Ниже приведена таблица тестирования программного продукта (Таблица1)

№	Функция	Описание
1	Отображение полей	Выполнено успешно
2	Расчет траектории полета шара	Выполнено успешно
3	Движение левого битка	Выполнено успешно
4	Движение правого битка	Выполнено успешно
5	Отскок шара от линии	Выполнено успешно
6	Добавление очков за линию	Выполнено успешно

Таблица 1. Тестирование программного продукта

Я сделал тестирование своего программного продукта по всем функциям которые были описаны на предыдущих этапах. Когда я их проверял, все они были завершены успешно.

III. ЗАКЛЮЧЕНИЕ

Была разработана программа «Pinball». Я считаю что все поставленные задачи я выполнил. В ходе выполнения курсового проекта я научился работать в Lazarus. В основном из электронных источников я улучшил свои знания и умения в работе с Lazarus, и улучшения знаний библиотек Lazarus. Также одной я изучил математику отражения шара от лини расположенных под разным углом относительно горизонтальных и вертикальных линий. Интерфейс мог быть лучше если бы был использован Unity.

В заключение, хочу сказать, что в дальнейшем возможно продолжение разработки данной программы с целью улучшения и добавления новых препятствий для шара, с целью усложнения карты игры.

СПИСОК ЛИТЕРАТУРЫ

Lazarus Documentation / M.: «freepascal».
URL: http://wiki.freepascal.org/Lazarus_Documentation (дата обращения: 15.12.2015)

Пинбол / М.: «Википедия».
URL: https://ru.wikipedia.org/wiki/Пинбол (дата обращения: 12.12.2015)

Вычислительная геометрия / М.: «Хабрахабр».
URL:http://habrahabr.ru/post/148325 (дата обращения: 26.12.2015)

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Код программного продукта

```
// Автор: Орищенко Андрей
// Группа: П-304
   program project1;
   {$mode objfpc}{$H+}
   uses
     {$IFDEF UNIX}{$IFDEF
     UseCThreads} cthreads,
     {$ENDIF}{$ENDIF}
     Interfaces, // this includes the LCL
     widgetset Forms, Unit1, Unit2, Unit3;
     {$R *.res} begin
     RequireDerivedFormResource :=
     True; Application.Initialize;
     Application.CreateForm(TDisturbed,
     Disturbed); Application.CreateForm(TForm2,
     Form2); Application.CreateForm(TSlipknot,
     Slipknot); Application.Run;
 end.

{Автор: Орищенко Андрей
 Группа: П-304
 Основная часть кода отвечающая за     перемещение шара по столу.}
   unit Unit1;
   {$mode
   objfpc}{$H+}
   interface
   uses
     Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs,
     ExtCtrls, StdCtrls, Math, IntfGraphics, Menus;
   type
   {****t* Engine/TBall
   * NAME
  * TBall
  * USAGE
```

```
 * Объект описывающий шар
 * INPUTS
* SYNOPSIS
* :TBall
* EXAMPLE
 * exemple:TBall;
 ****}
  TBall = record
    x, y: extended; //координаты центра шара
    dx, dy: extended; // текущее изменения координат
    r, d: float; //radius,diameter
  end;
  {****t* Engine/TLine
  * NAME
  * TLine
  * USAGE
  * Объект описывающий линию
  * INPUTS
  * координаты  концов  линии
  *  x1, y1, x2, y2: longint;
  * очки заработанные за  столкновение с  этой линией.
  * score: integer;
  * SYNOPSIS
  * :TLine
  * EXAMPLE
  * exemple:TLine;
   ****}
  TLine = record
```

```
 x1, y1, x2, y2: longint;     //координаты  концов  линии
  power: extended; //энергия которую передаем шару при столкновении
  ugol: extended; //угол  прибавляемый к   углу  отскока
  score: integer; //очки заработанные за столкновение с этой линией
end;
{ TDisturbed }
{****t* Engine/TDisturbed
* NAME
* TDisturbed
* USAGE
* Объект описывающий всю форму
* SYNOPSIS
* :TDisturbed
* EXAMPLE
* exemple:TDisturbed;
 ****}
TDisturbed = class(TForm)
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  PaintBox2: TPaintBox;
  Shape1: TShape;
  Timer1: TTimer;
  procedure FormCloseQuery(Sender: TObject; var CanClose: boolean);
  procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
  procedure FormKeyDown(Sender: TObject; var Key: word; Shift: TShiftState);
  procedure Label3Click(Sender: TObject);
  procedure PaintBox2Paint(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
private
  {private declaration}
```

```
  public
    Sh_data: TBall; //шар
    CountLine: integer; //кол-во линий
    Lines: array[0..100] of TLine;//массив линий
    img: TLazIntfImage;
    score, best_score: longint; //очки
    scopuli: boolean;
    lf_bt, rg_bt: real; // углы поворота левого и правого битка
    lf_dx, rg_dx: real; // скорость левого и правого битков
    procedure AddLine(scr, power, ugol, x1, y1, x2, y2: longint); //добавление линии
    procedure RollBall; // процедура движения шара
    procedure DrawTable; // рисование стола
    procedure Init; // расстановка линий и шара
    procedure ColorOfPowe(i: integer);
    procedure left_bitok;
    procedure right_bitok;
    procedure Interceptor(n: integer);
    function InterceptorCircleLine(cx, cy, radius, p1x, p1y, p2x,
      p2y: extended): boolean;
    function GetAngle(x1, y1, x2, y2: extended): extended; //получаем угол
  function SizeTwoDot(x1, y1, x2, y2: extended): real; //получаем растояние
  end;
var
  Disturbed: TDisturbed;
implementation
uses unit2;

{$R *.lfm}

{ TDisturbed }
const
  Grav_acel = 0.002581;
```

```
{****p* logic/TDisturbed.Timer1Timer
* NAME
* TDisturbed.Timer1Timer
* USAGE
* Процедура таймера
* INPUTS
* Sender: TObject;
* SYNOPSIS
*  TDisturbed.Timer1Timer(Sender: TObject);
* EXAMPLE
*  Timer1Timer(self);
****}
procedure TDisturbed.Timer1Timer(Sender: TObject);
begin
  RollBall;
left_bitok;
right_bitok;
end;

{****p* logic/TDisturbed.AddLine
* NAME
* TDisturbed.AddLine
* USAGE
* Процедура добавления линии
* INPUTS
* счет, сила, угол, точки начала и конца линии.
* (scr, power, ugol, x1, y1, x2, y2: longint);
* SYNOPSIS
*  TDisturbed.AddLine(scr, power, ugol, x1, y1, x2, y2: longint);
* EXAMPLE
*  AddLine(0,0,0,0,0,1,1);
 ****}
procedure TDisturbed.AddLine(scr, power, ugol, x1, y1, x2, y2: longint);
begin
```

```
  Inc(CountLine);
  Lines[CountLine].x1 := x1;
  Lines[CountLine].x2 := x2;
  Lines[CountLine].y1 := y1;
  Lines[CountLine].y2 := y2;
  Lines[CountLine].score := round(scr);
  Lines[CountLine].power := power;
  Lines[CountLine].ugol := ugol;
end;

{****p* logic/TDisturbed.RollBall
* NAME
* TDisturbed.RollBall
* USAGE
* Процедура перемешения шара
* SYNOPSIS
*  TDisturbed.RollBall;
* EXAMPLE
*  RollBall;
 ****}
procedure TDisturbed.RollBall;
var
  i: integer;
begin
  scopuli := True;
  Paintbox2.Canvas.Pen.color := clwhite;
  for i := 0 to countLine do
  Interceptor(i);
  if scopuli then
  begin
    sh_data.x := sh_data.x + sh_data.dx;
  sh_data.y := sh_data.y + sh_data.dy;
  Shape1.Top := round(sh_data.y);
  Shape1.Left := round(sh_data.x);
  end;
```

```
  sh_data.dy := sh_data.dy + Grav_acel;
end;
{****p* graph/TDisturbed.DrawTable
* NAME
* TDisturbed.DrawTable
* USAGE
* Процедура прорисовки стола
* SYNOPSIS
*  TDisturbed.DrawTable;
* EXAMPLE
*  DrawTable;
 ****}
procedure TDisturbed.DrawTable;
var
  i: integer;
begin
  for i := 0 to countline do
  begin
    with PaintBox2.canvas do
    begin
      ColorOfPowe(i);
      Line(Lines[i].x1, Lines[i].y1, Lines[i].x2, Lines[i].y2);
    end;
  end;
end;

{****p* logic/TDisturbed.Init
* NAME
* TDisturbed.Init
* USAGE
* Процедура инициализации
* SYNOPSIS
*  TDisturbed.Init;
* EXAMPLE
```

```
*  Init;
 ****}
procedure TDisturbed.Init;
begin
  score := 0;
  label2.Caption := IntToStr(score);
  countline := -1;
  sh_data.dx := 0;
  sh_data.dy := 0;
  lf_bt := 30;
  rg_bt := 150;
  lf_dx := 0;
  rg_dx := 0;

  addline(500, 666, 0, 135, 571, 165, 580);
  addline(500, 666, 0, 215, 580, 245, 571);
  addline(100, 0, 0, 0, 540, 132, 570);
  addline(100, 0, 0, 248, 570, 376, 540);

  addline(0, -1, 0, 378, 580, 400, 580);
  addline(0, -1, 0, 0, 598, 398, 598);

  addline(100, 0, 0, 0, 2, 400, 2);
  addline(100, 0, 0, 398, 0, 398, 600);
  addline(100, 0, 0, 2, 0, 2, 600);
  addline(100, 0, 0, 377, 65, 377, 600);

  addline(100, 0, 0, 275, 490, 320, 470);
  addline(100, 0, 0, 320, 470, 320, 420);
  addline(250, 100, 0, 275, 490, 320, 420);

  addline(100, 0, 0, 105, 490, 60, 470);
  addline(100, 0, 0, 60, 470, 60, 420);
  addline(250, 100, 0, 105, 490, 60, 420);
```

```
  {addline(100, 0, 0, 275, 515, 345, 483);
  addline(100, 0, 0, 345, 483, 345, 410);
  addline(100, 0, 0, 105, 515, 35, 483);
  addline(100, 0, 0, 35, 483, 35, 410);}

  addline(100, 0, 0, 400, 60, 375, 30);
  addline(100, 0, 0, 375, 30, 325, 0);

  addline(100, 0, 0, 40, 100, 70, 130);
  addline(100, 0, 0, 70, 130, 100, 100);
  addline(100, 0, 0, 100, 100, 70, 70);
  addline(100, 0, 0, 70, 70, 40, 100);

  addline(100, 0, 0, 240, 100, 270, 130);
  addline(100, 0, 0, 270, 130, 300, 100);
  addline(100, 0, 0, 300, 100, 270, 70);
  addline(100, 0, 0, 270, 70, 240, 100);

  addline(100, 0, 0, 125, 200, 225, 200);
  addline(100, 0, 0, 225, 200, 225, 300);
  addline(100, 0, 0, 225, 300, 125, 300);
  addline(100, 0, 0, 125, 300, 125, 200);

  addline(100, 0, 0, 0, 60, 25, 30);
  addline(100, 0, 0, 25, 30, 75, 0);

  sh_data.x := 383;
  sh_data.y := 567;
end;

 {****p* graph/TDisturbed.ColorOfPowe
* NAME
* TDisturbed.ColorOfPowe
* USAGE
```

```
* INPUTS
* Номер линии.
* i: integer;
* SYNOPSIS
*  TDisturbed.ColorOfPowe(i: integer);
* EXAMPLE
*  ColorOfPowe(5);
 ****}
procedure TDisturbed.ColorOfPowe(i: integer);
begin
  with PaintBox2.canvas do
  begin
    if ((Lines[i].power >= 500)) then
    begin
      pen.color := clRed;
    pen.Width := 5;
    end;
    if (Lines[i].power > 0) and (Lines[i].power < 500) then
    begin
      pen.color := clBlue;
      pen.Width := 5;
    end;
    if Lines[i].power = 0 then
    begin
      pen.color := clGreen;
      pen.Width := 4;
    end;
    if Lines[i].power < 0 then
    begin
      pen.Color := $3AA3AA3A;
      pen.Width := 4;
    end;
end;
end;
```

```
{****p* graph/TDisturbed.left_bitok
* NAME
* TDisturbed.left_bitok
* USAGE
* Процедура управления левым битком
* SYNOPSIS
*  TDisturbed.left_bitok;
* EXAMPLE
*  left_bitok;
 ****}
procedure TDisturbed.left_bitok;
begin
  lf_dx := lf_dx + 0.5;
  lf_bt := lf_bt + lf_dx;
  if lf_bt > 30 then
  begin
    lf_bt := 30;
    lf_dx := 0;
  end;
  if lf_bt < -25 then
  begin
    lf_bt := -25;
    lf_dx := 0;
  end;
  ColorOfPowe(0);
  if (round(Lines[0].y1) <> (round(Lines[0].y2 + 35 * sin(rg_bt * pi / 180)))) then
    Paintbox2.Invalidate;
  Lines[0].x2 := round(Lines[0].x1 + 35 * cos(lf_bt * pi / 180));
  Lines[0].y2 := round(Lines[0].y1 + 35 * sin(lf_bt * pi /180));
  PaintBox2.canvas.Line(Lines[0].x1, Lines[0].y1, Lines[0].x2,Lines[0].y2);
end;

{****p* graph/TDisturbed.right_bitok
* NAME
```

```
* TDisturbed.right_bitok
* USAGE
* Процедура управления левым битком
* SYNOPSIS
*  TDisturbed.right_bitok;
* EXAMPLE
*  right_bitok;
 ****}
procedure TDisturbed.right_bitok;
begin
  rg_dx := rg_dx - 0.5;
  rg_bt := rg_bt + rg_dx;
  if rg_bt > 205 then
  begin
    rg_bt := 205;
    rg_dx := 0;
  end;
  if rg_bt < 150 then
  begin
    rg_bt := 150;
    rg_dx := 0;
  end;
  ColorOfPowe(0);
  if (round(Lines[1].y1) <> (round(Lines[1].y2 + 35 * sin(rg_bt * pi / 180)))) then
    Paintbox2.Invalidate;
  Lines[1].x1 := round(Lines[1].x2 + 35 * cos(rg_bt * pi / 180));
  Lines[1].y1 := round(Lines[1].y2 + 35 * sin(rg_bt * pi /180));
  PaintBox2.canvas.Line(Lines[1].x1, Lines[1].y1, Lines[1].x2, Lines[1].y2);
end;
```

```
{****p* logic/TDisturbed.Interceptor
* NAME
* TDisturbed.Interceptor
* USAGE
* Процедура упругого отскока
* INPUTS
* Номер линии.
* n: integer;
* SYNOPSIS
*  TDisturbed.Interceptor(n: integer);
* EXAMPLE
*  Interceptor(4);
 ****}
procedure TDisturbed.Interceptor(n: integer);
var
  a, b, y, c: extended;
  str: string;
begin
  if (InterceptorCircleLine(shape1.Left + 5, Shape1.Top + 5, Shape1.Height /
    2, Lines[n].x1, Lines[n].y1, Lines[n].x2, Lines[n].y2))then
  begin
    Inc(score, Lines[n].score);
    label2.Caption := IntToStr(score);
    scopuli := False;
    {if (SizeTwoDot(sh_data.x,sh_data.y,lines[n].x1,lines[n].y1)<7)
    or  (SizeTwoDot(sh_data.x,sh_data.y,lines[n].x2,lines[n].y2)<7)
    then
    Begin
      sh_data.dx:=-sh_data.dx;
    sh_data.dy:=-sh_data.dy;
    end else }
    Begin
    a := GetAngle(sh_data.x, sh_data.y,sh_data.x+sh_data.dx,sh_data.y+sh_data.dy);
    b := GetAngle(Lines[n].x1, Lines[n].y1, Lines[n].x2,Lines[n].y2);
```

```
    y := 2 * b - a;
    c := (Sqrt(Sqr(Sh_data.dx) + sqr(Sh_data.dy)));
    label1.Caption:=floattostr(b);
     sh_data.dx := c * cos(y * pi / 180); //высчитываем приращения для
движения
     sh_data.dy := c * sin(y * pi / 180);
    end;
    if Lines[n].power < 0 then

    begin
      Timer1.Enabled := False;
      ShowMessage('You lose');
      Form2.Slipknot4(score);
      str := InputBox('Сохранение игрока', 'Введите ваше имя :', 'player1');
      Form2.StringGrid1.Cells[1, 6] := str;
      Form2.StringGrid1.Cells[2, 6] := IntToStr(score);
      Form2.BS_sort;
      Form2.SaveStringGrid(Form2.StringGrid1);
      init;
    end;
    //if (sh_data.dx > -0.2) and (sh_data.dx < 0)then
      //sh_data.dx := -0.2;
    //if (sh_data.dx < 0.2) and (sh_data.dx > 0)then
      //sh_data.dx := 0.2;
    //if (sh_data.dy < 0.2) and (sh_data.dy > 0)then
      //sh_data.dy := 0.2;
    sh_data.x := sh_data.x + sh_data.dx; //и сразу двигаем шарsh_data.y
    := sh_data.y + sh_data.dy;
  shape1.Top   :=   round(sh_data.y);
  shape1.Left   :=     round(sh_data.x);
  end;
end;
```

```
{****f* logic/TDisturbed.InterceptorCircleLine
* NAME
* TDisturbed.InterceptorCircleLine
* USAGE
* Процедура для проверки пересечения линии
* INPUTS
* cx, cy: extended; координаты центра шара
* radius :extended; радиус
* p1x, p1y, p2x, p2y: extended; координаты начала и конца линии
* SYNOPSIS
*  TDisturbed.InterceptorCircleLine(cx, cy, radius, p1x, p1y, p2x, p2y: extended):
boolean;
* EXAMPLE
* InterceptorCircleLine(0,0,0,0,0,0,0);
 ****}
function TDisturbed.InterceptorCircleLine(cx, cy, radius, p1x, p1y, p2x, p2y:
  extended): boolean;
var
  dx, dy, a, b, c, x01, x02, y01, y02: extended; begin
  radius := radius * 1.1; x01 :=
  p1x - cx;
  y01 := p1y - cy; x02 :=
  p2x - cx; y02 := p2y -
  cy; dx := x02 - x01;    dy
  := y02 - y01;

  a := dx * dx + dy * dy;
  b := 2.0 * (x01 * dx + y01 * dy);
  c := x01 * x01 + y01 * y01 - radius * radius; if (-b <
  0) then
    Result := (c < 0)
  else if (-b < (2.0 * a)) then
    Result := (4.0 * a * c - b * b < 0) else
    Result := (a + b + c < 0); end;
```

```
{****f* logic/TDisturbed.GetAngle
* NAME
* TDisturbed.GetAngle
* USAGE
* Процедура для получения угла относительногоризонтали
* INPUTS
* x1, y1, x2, y2: single; координаты начала и концалинии
* SYNOPSIS
*  TDisturbed.GetAngle(x1, y1, x2, y2: single):single;
* EXAMPLE
*  GetAngle(0,0,0,0);
 ****}
function TDisturbed.GetAngle(x1, y1, x2, y2: extended): extended;
begin
  GetAngle := (ArcTan2(y2 - y1, x2 - x1)) * 180 / pi;
end;
{****f* logic/TDisturbed.SizeTwoDot
* NAME
* TDisturbed.SizeTwoDot
* USAGE
* Процедура для получения длины линии
* INPUTS
* x1, y1, x2, y2: extended; координаты начала и концалинии
* SYNOPSIS
*  TDisturbed.SizeTwoDot(x1, y1, x2, y2: extended):real;
* EXAMPLE
*  SizeTwoDot(0,0,0,0);
 ****}
function TDisturbed.SizeTwoDot(x1, y1, x2, y2: extended): real;
begin
  Result := Sqrt(sqr(x1 - x2) + sqr(y1 - y2));
end;
```

```
{****p* graph/TDisturbed.FormCreate
* NAME
* TDisturbed.FormCreate
* USAGE
* Процедура создания формы
* INPUTS
* Sender: TObject;
* SYNOPSIS
*  TDisturbed.FormCreate(Sender: TObject);
* EXAMPLE
*  FormCreate(self);
 ****}
procedure TDisturbed.FormCreate(Sender:
TObject); begin
  sh_data.d :=
  Shape1.Width;
  sh_data.r :=
  Shape1.Width / 2;
  Init;
  Shape1.Top :=
  round(sh_data.y);
  Shape1.Left :=
  round(sh_data.x);
  Brush.Bitmap :=
  TBitMap.Create; img :=
  brush.Bitmap.CreateIntfImag
  e;
  img.LoadFromFile('.\img\ba
  ck.bmp');
  Brush.Bitmap.LoadFromIntfImage(img);
Paintbox2.Canvas.Brush.Bitmap := TBitMap.Create;
Paintbox2.Canvas.Brush.Bitmap.LoadFromIntfImage(
img); end;
```

```
{****p* logic/TDisturbed.FormCloseQuery
* NAME
* TDisturbed.FormCloseQuery
* USAGE
* Процедура быстрого закрытияпрограммы
* INPUTS
* Sender: TObject;
* CanClose: boolean; можем закрыть илинет
* SYNOPSIS
*  TDisturbed.FormCloseQuery(Sender: TObject; var CanClose:boolean);
* EXAMPLE
*  FormCloseQuery(self,true);
 ****}
procedure TDisturbed.FormCloseQuery(Sender: TObject; var CanClose:
boolean); begin
  Timer1.Enabled := False;
    [mbYes, mbNo], 0)
  = mrYes; if not
  (canclose) then
  Timer1.Enabled :=
  True;
end;

{****p* logic/TDisturbed.FormDestroy
* NAME
* TDisturbed.FormDestroy
* USAGE
* Процедура закрытия программы
* INPUTS
* Sender: TObject;
* SYNOPSIS
*  TDisturbed.FormDestroy(Sender: TObject);
* EXAMPLE
*  FormDestroy(self);
```

```
 ****}
procedure TDisturbed.FormDestroy(Sender:
TObject); begin
  Brush.Bitmap.De
stroy; end;

{****p* logic/TDisturbed.FormKeyDown
* NAME
* TDisturbed.FormKeyDown
* USAGE
* Процедура перехвата нажатия клавиш
* INPUTS
* Sender: TObject;
* Key: word; номер нажатой кнопки
* SYNOPSIS
*  TDisturbed.FormKeyDown(Sender: TObject; var Key: word; Shift:
   TShiftState);
* EXAMPLE
*  FormKeyDown(self,13,bt,ssShift);
 ****}
procedure TDisturbed.FormKeyDown(Sender: TObject; var Key: word;
Shift: TShiftState);
begin
  if key = 32 then
  //start(space) begin
    if not (Timer1.Enabled)
    then begin
    Timer1.Enabled :=
    True; sh_data.dx :=
      0;
      sh_data.dy :=
    -2.4; end;
  end;
  if key = 37 then //left
  bitok begin
```

```
    lf_dx := lf_dx
  - 10; end;
  if key = 39 then //right
  bitok begin
    rg_dx := rg_dx
  + 10; end;
  if key = 27 then //exit
  (esc) begin
    C
los;
end;
end;
{****p* score/TDisturbed.Label3Click
* NAME
* TDisturbed.Label3Click
* USAGE
* Процедура отображения счета
* INPUTS
* Sender: TObject;
* SYNOPSIS
*  TDisturbed.Label3Click(Sender: TObject);
* EXAMPLE

 *  Label3Click(self);
  ****}
 procedure TDisturbed.Label3Click(Sender:
 TObject); var
 game_stop:
 boolean; begin
 game_stop :=
 Timer1.Enabled;
 Timer1.Enabled := False;
 form2.ShowModal;
 Timer1.Enabled := game_stop;
 end;
```

```
  {****p* graph/TDisturbed.PaintBox2Paint
  * NAME
  * TDisturbed.PaintBox2Paint
  * USAGE
  * Процедура рисовки стола при обновлении канвы
  * INPUTS
  * Sender: TObject;
  * SYNOPSIS
  *  TDisturbed.PaintBox2Paint(Sender: TObject);
  * EXAMPLE
  *  PaintBox2Paint(self);
   ****}
  procedure TDisturbed.PaintBox2Paint(Sender:
  TObject); begin
    DrawTabl;
  end;
end.

// Автор: Орищенко Андрей
// Группа: П-304
// База лучших играков, с сохранением.
  unit Unit2;
  {$mode objfpc}{$H+} interface
 uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, Grids,
  StdCtrls;
 type
  { TForm2 }
  player = record
    Name: string[25];
  score:  string  [25];
  end;
  {****t* score/TForm2
 * NAME
 * TForm2
```

```
* SYNOPSIS
* :TForm2
* EXAMPLE
* exemple:TForm2;
 ****}
  TForm2 = class(TForm)
    Label1: TLabel;
    StringGrid1: TStringGrid;
    procedure FormCreate(Sender: TObject);
  private
    { private declarations }
  public
    procedure BS_sort;
    procedure  SaveStringGrid(StringGrid:  TStringGrid);
    procedure  LoadStringGrid(StringGrid: TStringGrid);
    procedure Slipknot4(o4ivment:integer);
  end;
var
  Form2: TForm2;

implementation

uses unit1,unit3;

{$R *.lfm}

{ TForm2 }


{****p* score/TForm2.FormCreate
* NAME
* TForm2.FormCreate
* USAGE
* Процедура обновления счета
* INPUTS
* Sender: TObject;
* SYNOPSIS
*  TForm2.FormCreate(Sender: TObject);
```

```
* EXAMPLE
*  FormCreate(self);
****}
procedure TForm2.FormCreate(Sender: TObject);
begin
  LoadStringGrid(StringGrid1);
end;

{****p* score/TForm2.BS_sort
* NAME
* TForm2.BS_sort
* USAGE
* Процедура сортировки счета
* SYNOPSIS
*  TForm2.BS_sort;
* EXAMPLE
*  BS_sort(self);
****}
procedure TForm2.BS_sort;
var
  cik1, cik2, last: integer;
  Count: array[1..2] of string;
  n_izm: boolean;
begin
  repeat
    n_izm := True;
    for cik1 := 2 to 5 do
    begin
      if (StringGrid1.Cells[2, cik1] = '') and
      (StringGrid1.Cells[2, cik1 + 1] <> '') then
      begin
        n_izm := False;
        for cik2 := 1 to 2 do
        begin
          Count[cik2] := StringGrid1.Cells[cik2, cik1 + 1];
```

```
          StringGrid1.Cells[cik2, cik1 + 1] := StringGrid1.Cells[cik2, cik1];
          StringGrid1.Cells[cik2, cik1] :=Count[cik2];
        end;
    end;
    end;
  until n_izm;
  for cik1 := 1 to 6 do
    if StringGrid1.Cells[2, cik1] <> '' then
      last := cik1;
  repeat
    n_izm := True;
    for cik1 := 1 to last - 1 do
    begin
      if (StrToInt(StringGrid1.Cells[2, cik1]) <
      StrToInt(StringGrid1.Cells[2, cik1 + 1])) then
      begin
        n_izm := False;
        for cik2 := 1 to 2 do
        begin
          Count[cik2] := StringGrid1.Cells[cik2, cik1 + 1];
          StringGrid1.Cells[cik2, cik1 + 1] := StringGrid1.Cells[cik2, cik1];
          StringGrid1.Cells[cik2, cik1] :=Count[cik2];
        end;
    end;
    end;
  until n_izm;
end;

{****p* score/TForm2.SaveStringGrid
* NAME
* TForm2.SaveStringGrid
* USAGE
* Процедура сохранения счета
* INPUTS
* таблица для сохранения
```

```
* StringGrid: TStringGrid;
* SYNOPSIS
*  TForm2.SaveStringGrid(StringGrid:  TStringGrid);
* EXAMPLE
*  SaveStringGrid(StringGrid1);
****}
procedure TForm2.SaveStringGrid(StringGrid: TStringGrid);
var
  f: file of player;
i: integer;
one_pay: player;
begin
   {$I-}
  AssignFile(f, 'score.sav');
  Rewrite(f);
   {$I+}
  if IOresult <> 0 then
  begin
    ShowMessage('ERRoR Save');
  end
  else
  begin
    with StringGrid do
    begin
      for i := 1 to 5 do
      begin
        one_pay.Name := Cells[1, i];
        one_pay.score := Cells[2, i];
    {$I-}
        Write(F, one_pay);
    {$I+}
        if IOresult <> 0 then
          ShowMessage('ERRoR Save');
      end;
    end;
```

```
    CloseFile(F);
  end;
end;

{****p*score/TForm2.LoadStringGrid
* NAME
* TForm2.LoadStringGrid
* USAGE
* Процедура загрузки счета
* INPUTS
* таблица для загрузки
* StringGrid: TStringGrid;
* SYNOPSIS
*  TForm2.LoadStringGrid(StringGrid:  TStringGrid);
* EXAMPLE
*  LoadStringGrid(StringGrid1);
****}
procedure TForm2.LoadStringGrid(StringGrid: TStringGrid);
var
  f: file of player;
i, j: integer;
strtemp: player;
begin
 {$I-}
  AssignFile(f, 'score.sav');
  Reset(f);
 {$I+}
  if IOresult <> 0 then
  begin
    for i := 1 to 2 do
      for j := 1 to 5 do
        StringGrid.Cells[i, j] := '';
  end
  else
  begin
```

```
  with StringGrid do
  begin
    for i := 1 to 5 do
    begin
  {$I-}
      Read(f, strtemp);
  {$I+}
      if IOresult <> 0 then
      begin
        ShowMessage('ERRoR Save');
        Close;
      end;
      Cells[1, i] := strtemp.Name;
      Cells[2, i] := strtemp.score;
      end
    ;
    end;
    CloseFile(f)
  ; end;
end;

{****p* Slipknot/TForm2.Slipknot4
* NAME
* TForm2.Slipknot4
* USAGE
* Процедура проверки Slipknot
* INPUTS
* очки
* x, y: extended;
* o4ivment:integer;
* SYNOPSIS
* TForm2.Slipknot4(o4ivment:integer);
* EXAMPLE
* Slipknot4(1488);
```

```
    ****}
   procedure
   TForm2.Slipknot4(o4ivment:integer); begin
      if (o4ivment > 50000) and (o4ivment < 66666)
      then begin
        showmessage('Achievement unlocked
        "Slipknot".'); Brush.Bitmap := TBitMap.Create;
        Disturbed.img := brush.Bitmap.CreateIntfImage;
        Disturbed.img.LoadFromFile('.\img\666.bmp');
        Slipknot.Brush.Bitmap := TBitMap.Create;
        Slipknot.Brush.Bitmap.LoadFromIntfImage(Disturbed.img);
        Slipknot.ShowModal;
      end
    ; end;
  end.
// Автор: Орищенко Андрей
// Группа: П-304
    unit Unit3;
  {$mode objfpc}{$H+}

  interface

  uses
    Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs;

  type

    { TSlipknot }

  {****t* Slipknot/TSlipknot
  * NAME
  * TSlipknot
  * USAGE
  * Объект формы Slipknot
  * SYNOPSIS
  * :TSlipknot
```

```
* EXAMPLE

* exemple:TSlipknot ;
 ****}
  TSlipknot = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Slipknot: TSlipknot;
implementation
{$R *.lfm}{
TSlipknot  }
end.
```

ПРИЛОЖЕНИЕ Б

Техническая документация(Robodoc)

TABLE OF CONTENTS

Functions

Procedures

TDisturbed.left_bitok
TDisturbed.PaintBox2Paint
TDisturbed.right_bitok
TDisturbed.RollBall
TDisturbed.Timer1Timer
TForm2.BS_sort
TForm2.FormCreate
TForm2.LoadStringGrid
TForm2.SaveStringGrid
TForm2.Slipknot4

./Pinball/

- unit1.pas
- unit2.pas
- unit3.pas

Types

TBallTDisturbed

TForm2TLineTSlipknot

Engine/TBall [Types]

[Top] [Types]

NAME

TBall

USAGE

```
Объект описывающий шар
```

INPUTS

```
координаты центра шара

x, y: extended;

текущее изменения координат

dx, dy: extended;
```

SYNOPSIS

:**TBall**

EXAMPLE

exemple:**TBall**;

Engine/TDisturbed [Types]

[Top] [Types]

NAME

TDisturbed

USAGE

Объект описывающий всю форму

SYNOPSIS

:**TDisturbed**

EXAMPLE

exemple:**TDisturbed**;

Engine/TLine [Types]

[Top] [Types]

NAME

TLine

USAGE

Объект описывающий линию

INPUTS

координаты концов линии

x1, y1, x2, y2: longint;

очки заработанные за столкновение с этой линией.

score: integer;

SYNOPSIS

:**TLine**

EXAMPLE

exemple:**TLine**;

---

graph/TDisturbed.ColorOfPowe [Procedures]

[Top] [Procedures]

NAME

TDisturbed.ColorOfPowe

USAGE

Процедура для определения цвета линии

INPUTS

Номер линии.

```
i: integer;
```

SYNOPSIS

```
TDisturbed.ColorOfPowe(i: integer);
```

EXAMPLE

```
ColorOfPowe(5);
```

---

graph/TDisturbed.DrawTable [Procedures]

[Top] [Procedures]

NAME

TDisturbed.DrawTable

USAGE

Процедура прорисовки стола

SYNOPSIS

```
TDisturbed.DrawTable;
```

EXAMPLE

```
DrawTable;
```

---

graph/TDisturbed.FormCreate [Procedures]

[Top] [Procedures]

NAME

TDisturbed.FormCreate

USAGE

Процедура создания формы

INPUTS

```
Sender: TObject;
```

SYNOPSIS

```
TDisturbed.FormCreate(Sender: TObject);
```

EXAMPLE

```
FormCreate(self);
```

---

graph/TDisturbed.left_bitok [Procedures]

[Top] [Procedures]

NAME

TDisturbed.left_bitok

USAGE

Процедура управления левым битком

SYNOPSIS

```
TDisturbed.left_bitok;
```

EXAMPLE

```
left_bitok;
```

---

graph/TDisturbed.PaintBox2Paint [Procedures]

[Top] [Procedures]

NAME

TDisturbed.PaintBox2Paint

USAGE

Процедура рисовки стола при обновлении канвы

INPUTS

```
Sender: TObject;
```

SYNOPSIS

```
TDisturbed.PaintBox2Paint(Sender: TObject);
```

EXAMPLE

```
PaintBox2Paint(self);
```

---

graph/TDisturbed.right_bitok [Procedures]

[Top] [Procedures]

NAME

TDisturbed.right_bitok

USAGE

Процедура управления левым битком

SYNOPSIS

TDisturbed.right_bitok;

EXAMPLE

right_bitok;

---

logic/TDisturbed.AddLine [Procedures]

[Top] [Procedures]

NAME

TDisturbed.AddLine

USAGE

Процедура добавления линии

INPUTS

счет, сила, угол, точки начала и конца линии.

(scr, power, ugol, x1, y1, x2, y2: longint);

SYNOPSIS

TDisturbed.AddLine(scr, power, ugol, x1, y1, x2, y2: longint);

EXAMPLE

AddLine(0,0,0,0,0,1,1);

---

logic/TDisturbed.FormCloseQuery [Procedures]

[Top] [Procedures]

NAME

TDisturbed.FormCloseQuery

USAGE

Процедура быстрого закрытия программы

INPUTS

Sender: TObject;

CanClose: boolean; можем закрыть или нет

SYNOPSIS

TDisturbed.FormCloseQuery(Sender: TObject; var CanClose: boolean);

EXAMPLE

FormCloseQuery(self,true);

---

logic/TDisturbed.FormDestroy [Procedures]

[Top] [Procedures]

NAME

TDisturbed.FormDestroy

USAGE

Процедура закрытия программы

INPUTS

Sender: TObject;

SYNOPSIS

TDisturbed.FormDestroy(Sender: TObject);

EXAMPLE

FormDestroy(self);

---

logic/TDisturbed.FormKeyDown [Procedures]

[Top] [Procedures]

NAME

TDisturbed.FormKeyDown

USAGE

Процедура перехвата нажатия клавиш

INPUTS

Sender: TObject;

Key: word; номер нажатой кнопки

SYNOPSIS

TDisturbed.FormKeyDown(Sender: TObject; var Key: word; Shift: TShiftState);

EXAMPLE

FormKeyDown(self,13,bt,ssShift);

---

logic/TDisturbed.GetAngle [Functions]

[Top] [Functions]

NAME

TDisturbed.GetAngle

USAGE

Процедура для получения угла относительно горизонтали

INPUTS

x1, y1, x2, y2: single; координаты начала и конца линии

SYNOPSIS

```
TDisturbed.GetAngle(x1, y1, x2, y2: single): single;
```

EXAMPLE

```
GetAngle(0,0,0,0);
```

---

logic/TDisturbed.Init [Procedures]

[Top] [Procedures]

NAME

TDisturbed.Init

USAGE

Процедура инициализации

SYNOPSIS

```
TDisturbed.Init;
```

EXAMPLE

```
Init;
```

---

logic/TDisturbed.Interceptor [Procedures]

[Top] [Procedures]

NAME

TDisturbed.Interceptor

USAGE

Процедура упругого отскока

INPUTS

Номер линии.

```
n: integer;
```

SYNOPSIS

```
TDisturbed.Interceptor(n: integer);
```

EXAMPLE

```
Interceptor(4);
```

---

logic/TDisturbed.InterceptorCircleLine [Functions]

[Top] [Functions]

NAME

TDisturbed.InterceptorCircleLine

USAGE

Процедура для проверки пересечения линии

INPUTS

cx, cy: extended; координаты центра шара

radius :extended; радиус

p1x, p1y, p2x, p2y: extended; координаты начала и конца линии

SYNOPSIS

```
TDisturbed.InterceptorCircleLine(cx, cy, radius, p1x, p1y, p2x, p2y: extended): boolean;
```

EXAMPLE

```
InterceptorCircleLine(0,0,0,0,0,0,0);
```

---

logic/TDisturbed.RollBall [Procedures]

[Top] [Procedures]

NAME

TDisturbed.RollBall

USAGE

Процедура перемешения шара

SYNOPSIS

```
TDisturbed.RollBall;
```

EXAMPLE

```
RollBall;
```

---

logic/TDisturbed.SizeTwoDot [Functions]

[Top] [Functions]

NAME

TDisturbed.SizeTwoDot

USAGE

Процедура для получения длины линии

INPUTS

x1, y1, x2, y2: extended; координаты начала и конца линии

SYNOPSIS

```
TDisturbed.SizeTwoDot(x1, y1, x2, y2: extended): real;
```

EXAMPLE

```
SizeTwoDot(0,0,0,0);
```

---

logic/TDisturbed.Timer1Timer [Procedures]

[Top] [Procedures]

NAME

TDisturbed.Timer1Timer

USAGE

Процедура таймера

INPUTS

Sender: TObject;

SYNOPSIS

```
TDisturbed.Timer1Timer(Sender: TObject);
```

EXAMPLE

```
Timer1Timer(self);
```

---

score/TDisturbed.Label3Click [Procedures]

[Top] [Procedures]

NAME

TDisturbed.Label3Click

USAGE

Процедура отображения счета

INPUTS

Sender: TObject;

SYNOPSIS

TDisturbed.Label3Click(Sender: TObject);

EXAMPLE

Label3Click(self);

score/TForm2 [Types]

[Top] [Types]

NAME

TForm2

USAGE

Объект форму 2

SYNOPSIS

:**TForm2**

EXAMPLE

exemple:**TForm2**;

---

score/TForm2.BS_sort [Procedures]

[Top] [Procedures]

NAME

TForm2.BS_sort

USAGE

Процедура сортировки счета

SYNOPSIS

TForm2.BS_sort;

EXAMPLE

```
BS_sort(self);
```

score/TForm2.FormCreate [Procedures]

[Top] [Procedures]

NAME

TForm2.FormCreate

USAGE

Процедура обновления счета

INPUTS

```
Sender: TObject;
```

SYNOPSIS

```
TForm2.FormCreate(Sender: TObject);
```

EXAMPLE

```
FormCreate(self);
```

score/TForm2.LoadStringGrid [Procedures]

[Top] [Procedures]

NAME

TForm2.LoadStringGrid

USAGE

Процедура загрузки счета

INPUTS

таблица для загрузки

```
StringGrid: TStringGrid;
```

SYNOPSIS

```
TForm2.LoadStringGrid(StringGrid: TStringGrid);
```

EXAMPLE

```
LoadStringGrid(StringGrid1);
```

score/TForm2.SaveStringGrid [Procedures]

[Top] [Procedures]

NAME

TForm2.SaveStringGrid

USAGE

Процедура сохранения счета

INPUTS

таблица для сохранения

StringGrid: TStringGrid;

SYNOPSIS

TForm2.SaveStringGrid(StringGrid: TStringGrid);

EXAMPLE

SaveStringGrid(StringGrid1);

Slipknot/TForm2.Slipknot4 [Procedures]

[Top] [Procedures]

NAME

TForm2.Slipknot4

USAGE

Процедура проверки Slipknot

INPUTS

очки

x, y: extended;

o4ivment:integer;

SYNOPSIS

TForm2.Slipknot4(o4ivment:integer);

EXAMPLE

Slipknot4(1488);

Slipknot/TSlipknot [Types]

[Top] [Types]

NAME

TSlipknot

USAGE

```
Объект формы Slipknot
```

SYNOPSIS

```
:TSlipknot
```

EXAMPLE

```
exemple:TSlipknot ;
```

ПРИЛОЖЕНИЕ В

Протокол системы контроля версий (Git)

```
commit 8a126e08c6663bb9bb84eb721692275effe0cda1
Author: ZaXeLoN<1@zaxelon.hol.es>
Date:   Sun Jan 10 04:09:08 2016 +0300

    Alpha v02

commit 8a126e08c6663bb9bb84eb721692275effe0cda1
Author: ZaXeLoN<1@zaxelon.hol.es>
Date:   Sun Jan 10 03:12:44 2016 +0300

    bags fixed

commit 292c9c8991017af96a7bc39f19ded42ff626e357
Author: ZaXeLoN<1@zaxelon.hol.es>
Date:   Sun Jan 10 02:27:51 2016 +0300

    add comments

commit e11e0046e8662d36ea5f214756e524dd3f8e0f21
Author: ZaXeLoN<1@zaxelon.hol.es>
Date:   Sun Jan 10 02:06:54 2016 +0300

    deleted line

commit b2e4bc1182e01024b4d36108e46528f5aca02106
Author: ZaXeLoN<1@zaxelon.hol.es>
Date:   Sun Jan 10 01:28:09 2016 +0300

    add robodoc coments

commit af5b25396632c0b2acc6a98cdba2bbb61580a180
Author: ZaXeLoN<1@zaxelon.hol.es>
Date:   Sat Jan 9 12:06:56 2016 +0300

    start using git
```

ПРИЛОЖЕНИЕ Г

Текст доклада

Введение(разрекламировать себя как программиста и свой программный продукт)

Здравствуйте, Я Андрей Орищенко, учусь на 3 курсе колледжа информационных технологий на специальности программирование в компьютерных системах. За два года я получил практический опыт в разработке программ и алгоритмов по поставленной задаче и реализации его средствами автоматизированного программирования. Научился разрабатывать код связанный с вычислительной геометрией, физикой, и криптографией. Так же использование инструментальных средств на этапе отладки программного продукта и проведения тестирования программного продукта по определенному сценарию. Я умею разрабатывать код на современных языках программирования, таких как Delphi, C++, php, и на разных платформах, таких как Windows и POSIX.

Создавать программу по разработанному алгоритму, и разрабатывать алгоритмы разных сложностей. Научился корректно обрабатывать всевозможные ошибки, которые позволяют обрабатывать стандарты языков, на которых ведется разработка. Выполнять отладку и тестирование программы на уровне модуля. Оформлять документацию на программные средства. Использовать инструментальные средства для автоматизации оформления документации.

В настоящее время меня больше интересует криптография, хеширование и удаленное соединение по принципу клиент\сервер, которое позволит создать программу для удаленного администрирования ЭВМ, но смысла от внедрения данных технологий в эту программу не вижу.