Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Федеральное государственное бюджетное образовательное учреждениевысшего образования
«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №10
по курсу «Программирование»

Разработка графического приложения

Выполнил студент группы ИВТ-11 ____________________/Рзаев А. Э./
Проверил преподаватель __________________________/Чистяков Г. А./

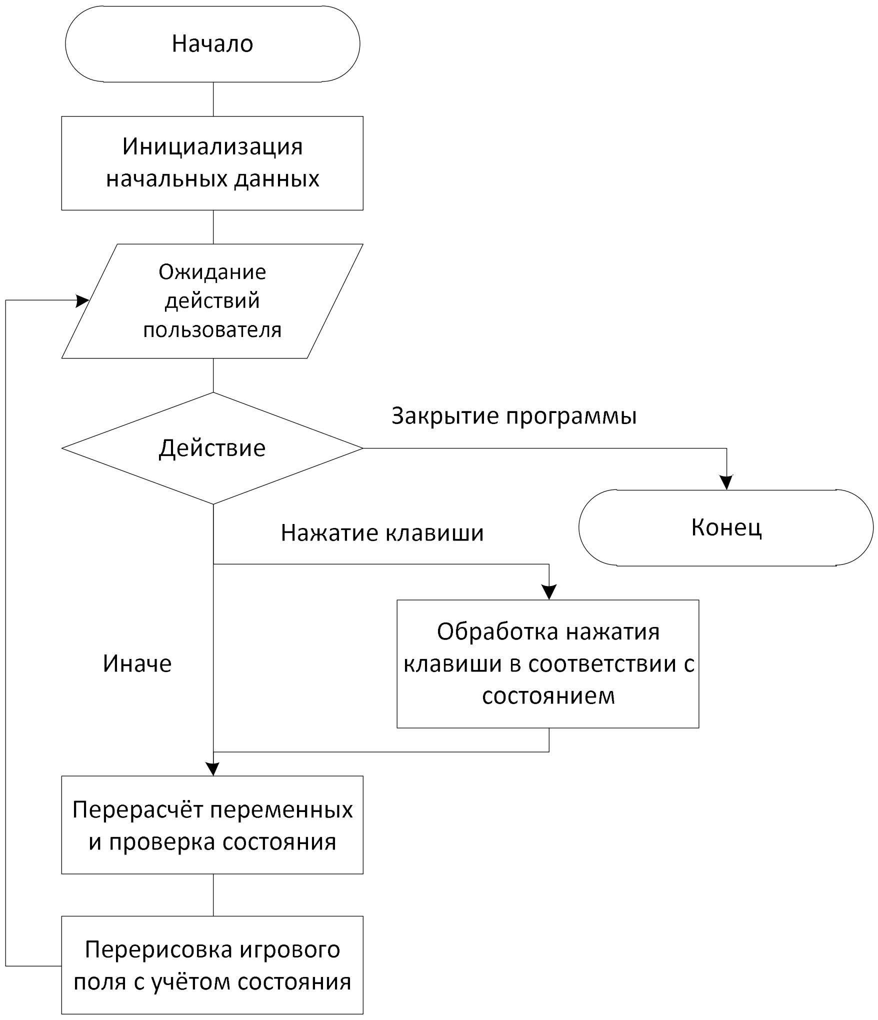
Киров 2016

Цель работы: закрепить навыки, полученные в ходе изучения курса.

Задание:

1. Разработать игру, используя графические возможности среды разработки;
2. Данные приложения (таблица рекордов) хранить с помощью базы данных.

Схема работы программы:


Начало
Инициализация начальных данных
Ожидание действий пользователя
Действие
Закрытие программы
Конец
Нажатие клавиши
Обработка нажатия клавиши в соответствии с состоянием
Иначе
Перерасчёт переменных и проверка состояния
Перерисовка игрового поля с учётом состояния

Листинг кода:
Модуль MainForm

```
unit MainForm;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, ExtCtrls,
  StdCtrls, Machine, Tetris, LCLType, Menus, LResources, Grids, Records;

const
  NormalTitle : String = 'Тетрис';
  PauseTitle : String = 'Тетрис (Пауза)';
  SquareSize = 15;
  Distance = 3;
  RecordFileName : String = 'records.dbf';
  //DataBase Edition

type

  { TGameForm }

  TGameForm = class(TForm)
    GameBox: TPaintBox;
    LabelLevel: TLabel;
    LabelScore: TLabel;
    LabelLines: TLabel;
    LabelScoreC: TLabel;
    LabelLinesC: TLabel;
    LabelLevelC: TLabel;
    MainMenu: TMainMenu;
    ItemNewGame: TMenuItem;
    ItemExitGame: TMenuItem;
    ItemPauseResumeGame: TMenuItem;
    ItemRecords: TMenuItem;
    MenuItemGame: TMenuItem;
    ShapeBox: TPaintBox;
    Timer: TTimer;
    Records: TRecordTable;
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word; Shift:
TShiftState);
    procedure ItemExitGameClick(Sender: TObject);
    procedure ItemNewGameClick(Sender: TObject);
    procedure ItemPauseResumeGameClick(Sender: TObject);
    procedure ItemRecordsClick(Sender: TObject);
    procedure TimerTimer(Sender: TObject);
    procedure RepaintArea;
    procedure UpdateState;
    procedure GameOver;
    procedure UpdateScores;
    procedure LoadRecords;
    procedure SaveRecords;
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  GameForm: TGameForm;

implementation

{$R *.lfm}

procedure RepaintFigure(var Box : TPaintBox; Sh : TShape);
var
  i, j, x1, y1, x2, y2 : Longint;
  m : TFigureMatrix;
begin
  Box.Canvas.Brush.Color := clWhite;
  Box.Canvas.Clear;
  Box.Canvas.Brush.Color := Sh.color;
  m := Figures[Sh.n][Sh.p];
  for i := 0 to ShapeHeight - 1 do
    for j := 0 to ShapeWidth - 1 do
    begin
      if (m[i][j] <> 0) then
      begin
        x1 := (j + Sh.position.x) * (SquareSize + Distance);
        y1 := (i + Sh.position.y) * (SquareSize + Distance);
        x2 := x1 + SquareSize;
        y2 := y1 + SquareSize;
        Box.Canvas.FillRect(x1, y1, x2, y2);
      end;
    end;
  Box.Canvas.Brush.Color := clBlack;
end;

procedure RepaintBomb(var Box : TPaintBox; Sh : TShape);
var
  x1, y1 : Longint;
  img : TPicture;
begin
  Box.Canvas.Brush.Color := clWhite;
  Box.Canvas.Clear;
  x1 := (Sh.position.x) * (SquareSize + Distance);
  y1 := (Sh.position.y) * (SquareSize + Distance);
  img := TPicture.Create;
  img.LoadFromFile('bomb.bmp');
  Box.Canvas.Draw(x1, y1, img.Graphic);
  Box.Canvas.Brush.Color := clBlack;
end;

procedure RepaintBonus(var Box : TPaintBox; Sh : TShape);
begin

end;

const
  RepaintShape : array [0..2] of procedure(var Box : TPaintBox; Sh : TShape)
= (
    @RepaintFigure,
    @RepaintBomb,
    @RepaintBonus
  );

{ TGameForm }

procedure TGameForm.RepaintArea;
var
  i, j, x1, y1, x2, y2 : Longint;
begin
  RepaintShape[Shape.t](GameBox, Shape);
  RepaintShape[NextShape.t](ShapeBox, NextShape);
  i := 0;
  while i < TetrisHeight do
  begin
    j := Area.offset;
    repeat
      if (Area.surface[i][j].n <> 0) then
      begin
        GameBox.Canvas.Brush.Color := Area.surface[i][j].c;
        x1 := CircleAdd(j, -Area.offset, TetrisWidth) * (SquareSize +
Distance);
        y1 := i * (SquareSize + Distance);
        x2 := x1 + SquareSize;
        y2 := y1 + SquareSize;
        GameBox.Canvas.FillRect(x1, y1, x2, y2);
      end;
      j := CircleAdd(j, 1, TetrisWidth);
    until j = CircleAdd(Area.offset, TetrisWidth, TetrisWidth);
    Inc(i);
  end;
end;

procedure TGameForm.TimerTimer(Sender: TObject);
begin
  MachineProcess(KNone);
  RepaintArea;
  UpdateState;
end;

procedure TGameForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if State = SEnd then
    exit;
  if Key = VK_LEFT then
    MachineProcess(KLeft)
  else if Key = VK_RIGHT then
    MachineProcess(KRight)
  else if Key = VK_ESCAPE then
    MachineProcess(KEsc)
  else if Key = VK_SPACE then
    MachineProcess(KSpace)
  else
    MachineProcess(KNone);
  RepaintArea;
  UpdateState;
end;
```

```
procedure TGameForm.ItemExitGameClick(Sender: TObject);
begin
  GameForm.Close;
end;

procedure TGameForm.GameOver;
begin
  Timer.Enabled := False;
  ShowMessage('Вы проиграли!');
  UpdateScores;
end;

procedure TGameForm.UpdateScores;
var
  n: String;
begin
  if Records.IsNewRecord(Score) then
  begin
    n:=InputBox('Сохранение рекорда', 'Введите имя игрока', '');
    if n <> '' then
      Records.WriteNewRecord(n, Score);
  end;
end;

procedure TGameForm.LoadRecords;
begin
  Records:=TRecordTable.Create;
  Records.LoadFromFile(RecordFileName);
end;

procedure TGameForm.SaveRecords;
begin
  Records.SaveToFile(RecordFileName);
  Records.Free;
end;

procedure TGameForm.UpdateState;
begin
  GameForm.LabelLevel.Caption := IntToStr(Level);
  GameForm.LabelScore.Caption := IntToStr(Score);
  GameForm.LabelLines.Caption := IntToStr(Lines);
  if State = SNormal then
    GameForm.Caption := NormalTitle
  else if State = SPause then
    GameForm.Caption := PauseTitle
  else if State = SEnd then
    GameOver;
end;

procedure TGameForm.ItemNewGameClick(Sender: TObject);
begin
  GameBox.Canvas.Brush.Color := clWhite;
  GameBox.Canvas.Clear;
  MachineInit;
  RepaintArea;
  UpdateState;
  //Timer.Interval := BaseInterval;
  Timer.Enabled := True;
end;
```

```
procedure TGameForm.ItemPauseResumeGameClick(Sender: TObject);
begin
  if State <> SEnd then;
    MachineProcess(KEsc);
end;

procedure TGameForm.ItemRecordsClick(Sender: TObject);
var
  i, j : Integer;
  viewer: TForm;
  table: TStringGrid;
begin
  viewer:=TForm.Create(nil);
  table:=TStringGrid.Create(viewer);
  i:=Left; j:=Top;
  with viewer do
  begin
    SetBounds(i, j, 200, 200);
    Caption:='Рекорды';
  end;
  with table do
  begin
    Parent:=viewer;
    Align:=alClient;
    AutoFillColumns:=True;
    ColCount:=2;
    RowCount:=8;
    FixedCols:=0;
    FixedRows:=0;
  end;
  j:=0;
  for i := 1 to Records.Size do
  begin
    if Records.Players [i].Name <> '' then
    begin
      table.Cells [0, j]:=Records.Players [i].Name;
      table.Cells [1, j]:=IntToStr(Records.Players [i].Score);
      Inc(j);
    end
  end;
  viewer.ShowModal;
  FreeAndNil(viewer);
end;

procedure TGameForm.FormCreate(Sender: TObject);
begin
  GameBox.Canvas.Brush.Color := clWhite;
  GameBox.Canvas.Clear;
  GameBox.Canvas.Refresh;
  LoadRecords;
end;

procedure TGameForm.FormClose(Sender: TObject; var CloseAction:
TCloseAction);
begin
  if (State <> SEnd)  then
  begin
    Timer.Enabled:=False;
    UpdateScores;
  end;
  SaveRecords;
end;

end.
```

Модуль Machine

```
unit Machine;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Tetris;

type
  TState = SmallInt;
  TKeys = SmallInt;
  THandle = function(): TState;

const
  SNormal: TState = 0;
  SPause: TState = 1;
  SEnd: TState = 2;

  KNone: TKeys = 0;
  KDown: TKeys = 1;
  KLeft: TKeys = 2;
  KRight: TKeys = 3;
  KSpace: TKeys = 4;
  KEsc: TKeys = 5;
  KQuit: TKeys = 6;

procedure MachineInit;
procedure MachineProcess(Key : TKeys);

var
  State : TState;

implementation


procedure MachineInit;
begin
  State := SNormal;
  Initialise;
end;

function MachineStep: TState;
begin
  Process;
  if IsGameOver then
    MachineStep := SEnd
  else
    MachineStep := SNormal;
end;
```

```
function MachineNothing: TState;
begin
  MachineNothing := State;
end;

function MachineDown: TState;
begin
  MoveDown;
  MachineDown := SNormal;
end;

function MachineLeft: TState;
begin
  MoveLeft;
  MachineLeft := SNormal;
end;

function MachineRight: TState;
begin
  MoveRight;
  MachineRight := SNormal;
end;

function MachineRotate: TState;
begin
  RotateShape;
  MachineRotate := SNormal;
end;

function MachinePause: TState;
begin
  MachinePause := SPause;
end;

function MachineContinue: TState;
begin
  MachineContinue := SNormal;
end;

function MachineExit: TState;
begin
  //ExitGame;
  MachineExit := SEnd;
end;

const
  CMachine : array [0..1, 0..6] of THandle = (
          ( @MachineStep, @MachineDown, @MachineLeft, @MachineRight,
@MachineRotate, @MachinePause, @MachineExit ),
          ( @MachineNothing, @MachineNothing, @MachineNothing,
@MachineNothing, @MachineNothing, @MachineContinue, @MachineExit )
  );

procedure MachineProcess(Key : TKeys);
begin
  if Key in [KDown, KLeft, KRight, KSpace, KEsc, KQuit] then
  begin
    State := CMachine[State][Key]();
  end
  else
  begin
    State := CMachine[State][KNone]();
  end;
end;

end.
```

Модуль Tetris

```
unit Tetris;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Graphics;

type
  TFigureMatrix = array [0..3, 0..3] of LongInt;
  TShapeType = SmallInt;
  TShape = record
    n : LongInt;
    p : LongInt;
    position : TPoint;
    color : TColor;
    t : TShapeType;
  end;

const
  TetrisWidth : LongInt = 20;
  TetrisHeight : LongInt = 35;
  ShapeWidth : LongInt = 4;
  ShapeHeight : LongInt = 4;
  STFigure : TShapeType = 0;
  STBomb : TShapeType = 1;
  STBag : TShapeType = 2;
  Colors : array [0..4] of TColor = (clTeal, clRed, clPurple, clBlue,
clGreen);
  Figures : array [0..6, 0..3] of TFigureMatrix = (
    (
      ( (1, 1, 1, 1), (0, 0, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //1111
      ( (0, 1, 0, 0), (0, 1, 0, 0), (0, 1, 0, 0), (0, 1, 0, 0) ), //0000
      ( (1, 1, 1, 1), (0, 0, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0000
      ( (0, 1, 0, 0), (0, 1, 0, 0), (0, 1, 0, 0), (0, 1, 0, 0) )  //0000
    ),
    (
      ( (1, 1, 0, 0), (1, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //1100
      ( (1, 1, 0, 0), (1, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //1100
      ( (1, 1, 0, 0), (1, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0000
      ( (1, 1, 0, 0), (1, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) )  //0000
    ),
    (
      ( (1, 1, 0, 0), (1, 0, 0, 0), (1, 0, 0, 0), (0, 0, 0, 0) ), //1100
      ( (1, 1, 1, 0), (0, 0, 1, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //1000
      ( (0, 1, 0, 0), (0, 1, 0, 0), (1, 1, 0, 0), (0, 0, 0, 0) ), //1000
      ( (1, 0, 0, 0), (1, 1, 1, 0), (0, 0, 0, 0), (0, 0, 0, 0) )  //0000
    ),
    (
      ( (1, 1, 0, 0), (0, 1, 0, 0), (0, 1, 0, 0), (0, 0, 0, 0) ), //1100
      ( (0, 0, 1, 0), (1, 1, 1, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0100
      ( (1, 0, 0, 0), (1, 0, 0, 0), (1, 1, 0, 0), (0, 0, 0, 0) ), //0100
      ( (1, 1, 1, 0), (1, 0, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) )  //0000
    ),
    (
      ( (1, 1, 0, 0), (0, 1, 1, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //1100
      ( (0, 1, 0, 0), (1, 1, 0, 0), (1, 0, 0, 0), (0, 0, 0, 0) ), //0110
      ( (1, 1, 0, 0), (0, 1, 1, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0000
      ( (0, 1, 0, 0), (1, 1, 0, 0), (1, 0, 0, 0), (0, 0, 0, 0) )  //0000
    ),
    (
      ( (0, 1, 1, 0), (1, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0110
      ( (1, 0, 0, 0), (1, 1, 0, 0), (0, 1, 0, 0), (0, 0, 0, 0) ), //1100
      ( (0, 1, 1, 0), (1, 1, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0000
      ( (1, 0, 0, 0), (1, 1, 0, 0), (0, 1, 0, 0), (0, 0, 0, 0) )  //0000
    ),
    (
      ( (0, 1, 0, 0), (1, 1, 1, 0), (0, 0, 0, 0), (0, 0, 0, 0) ), //0100
      ( (1, 0, 0, 0), (1, 1, 0, 0), (1, 0, 0, 0), (0, 0, 0, 0) ), //1110
      ( (0, 0, 0, 0), (1, 1, 1, 0), (0, 1, 0, 0), (0, 0, 0, 0) ), //0000
      ( (0, 1, 0, 0), (1, 1, 0, 0), (0, 1, 0, 0), (0, 0, 0, 0) )  //0000
    )
  );

type
  TMatrix = array [-4..38, 0..19] of record
    n : LongInt;
    c : TColor;
  end;

  TPoint = record
    x : LongInt;
    y : LongInt;
  end;

type TArea = record
  offset : LongInt;
  surface : TMatrix;
end;

procedure Initialise;
procedure NextLevel;
function IsNextLevelScore : Boolean;
procedure GenerateArea;
procedure GenerateShape;
procedure GenerateNextShape;
procedure MoveLeft;
procedure MoveRight;
procedure MoveDown;
procedure Process;
procedure RotateShape;
procedure AttachFigure(Sh : TShape);
procedure AttachBomb(Sh : TShape);
procedure AttachBag(Sh : TShape);
function IsGameOver: Boolean;
function CircleAdd(a, d, m : LongInt) : LongInt;
```

```
const
  AttachShape : array[0..2] of procedure(Sh : TShape) = (
    @AttachFigure,
    @AttachBomb,
    @AttachBag
  );

var
  Area : TArea;
  Shape, NextShape : TShape;
  Counter : LongInt;
  Level : LongInt;
  Lines : LongInt;
  Score : LongInt;

implementation

function CircleAdd(a, d, m : LongInt) : LongInt;
begin
  CircleAdd := (a + d + m) mod m;
end;

procedure Initialise;
begin
  Level := 1;
  Lines := 0;
  Counter := 0;
  Score := 0;
  Area.offset := 0;
  GenerateArea;
  GenerateNextShape;
  GenerateShape;
end;

function IsGameOver: Boolean;
begin
  if Shape.position.y + 1 < 0 then
    IsGameOver := True
  else
    IsGameOver := False;
end;

procedure GenerateArea;
var
  i, j : LongInt;
begin
  //test
  Randomize;
  for i := -4 to TetrisHeight - 1 do
    for j := 0 to TetrisWidth - 1 do
      if (i > TetrisHeight - 1 - Level) and (Random(100) mod 4 = 0) then
      begin //25 %
        Area.surface[i][j].n := 1;
        Area.surface[i][j].c := Colors[Random(5)];
      end
      else
        Area.surface[i][j].n := 0;
  for i := TetrisHeight to TetrisHeight + 3 do
```

```
    for j := 0 to TetrisWidth - 1 do
      Area.surface[i][j].n := 1;
end;

function CheckEdges: Boolean;
var
  i, j, offsetx, offsety : LongInt;
  m : TFigureMatrix;
begin
  m := Figures[Shape.n][Shape.p];
  for j := 0 to ShapeWidth - 1 do
  begin
    offsetx := CircleAdd(Area.offset, Shape.position.x + j, TetrisWidth);
    for i := 0 to ShapeHeight - 1 do
    begin
      offsety := Shape.position.y + i;
      if (m[i][j] = 1) and (Area.surface[offsety][offsetx].n = 1) then
      begin
        CheckEdges := False;
        exit;
      end;
    end;
  end;
  CheckEdges := True;
end;

procedure GenerateNextShape;
begin
  NextShape.n := Random(7);
  NextShape.p := Random(4);
  NextShape.color := Colors[Random(5)];
  NextShape.position.x := 1;
  NextShape.position.y := 1;
  if Random(100) mod 11 = 0 then
  begin
    NextShape.t := STBomb;
    NextShape.n := 1;
  end
  else
    NextShape.t := STFigure;
end;

procedure GenerateShape;
begin
  Randomize;
  Inc(Counter);
  Shape := NextShape;
  GenerateNextShape;
  Shape.position.y := -4;
  Shape.position.x := Random(TetrisWidth - ShapeWidth);
  while (Shape.position.y < 0) do
  begin
    Inc(Shape.position.y);
    if not CheckEdges then
    begin
      Dec(Shape.position.y);
      break;
    end;
  end;
```

```
end;

procedure Process;
var
  i, j, k, cnt : LongInt;
begin
  MoveDown;
  for i := 0 to TetrisHeight - 1 do
  begin
    cnt := 0;
    for j := 0 to TetrisWidth - 1 do
      if Area.surface[i][j].n <> 0 then
        inc(cnt);
    if cnt = TetrisWidth then
    begin
      inc(Score, 100 * Level);
      inc(Lines, 1);
      for j := 0 to TetrisWidth - 1 do
        Area.surface[i][j].n := 0;
      for k := i - 1 downto 1 do
        for j := 0 to TetrisWidth - 1 do
          Area.surface[k + 1][j] := Area.surface[k][j];
    end;
  end;
  cnt := 0;
  //Is area cleaned completely?
  for i := 0 to TetrisHeight - 1 do
    for j := 0 to TetrisWidth - 1 do
      if Area.surface[i][j].n <> 0 then
        Inc(cnt);
  if (cnt = 0) or IsNextLevelScore then
    NextLevel;
end;

function IsNextLevelScore: Boolean;
begin
  IsNextLevelScore := Score >= Level * Level * 500 div 2;
end;

procedure NextLevel;
begin
  if not IsNextLevelScore then GenerateArea;
  Inc(Score, 100 * Level);
  Inc(Level);
  GenerateShape;
end;

procedure AttachFigure(Sh : TShape);
var
  i, j, offsetx, offsety : LongInt;
  m : TFigureMatrix;
begin
  if Sh.position.y + 1 < 0 then
    exit; //Game over
  m := Figures[Sh.n][Sh.p];
  for i := 0 to ShapeHeight - 1 do
  begin
    offsety := Sh.position.y + i;
    for j := 0 to ShapeWidth - 1 do
    begin
      offsetx := CircleAdd(Area.offset, Sh.position.x + j, TetrisWidth);
      if (m[i][j] <> 0) and (offsety >= 0) then
      begin
        Area.surface[offsety][offsetx].n := m[i][j];
        Area.surface[offsety][offsetx].c := Sh.color;
      end;
    end;
  end;
  GenerateShape;
end;

procedure AttachBomb(Sh: TShape);
var
  i, j, offsetx, offsety : LongInt;
begin
  if Sh.position.y + 1 < 0 then
    exit; //Game over
  for i := -1 to 2 do //Remove surface 4x4
  begin
    offsety := Sh.position.y + i;
    for j := -1 to 2 do
    begin
      offsetx := CircleAdd(Area.offset, Sh.position.x + j, TetrisWidth);
      if (offsety < TetrisHeight) and (offsety >= 0) then
      begin
        Area.surface[offsety][offsetx].n := 0; //Clear cell
      end;
    end;
  end;
  GenerateShape;
end;

procedure AttachBag(Sh: TShape);
begin

end;

procedure MoveRight;
var
  t : LongInt;
begin
  t := Area.offset;
  Area.offset := CircleAdd(Area.offset, -1, TetrisWidth);
  if not CheckEdges then
    Area.offset := t;
end;

procedure MoveLeft;
var
  t : LongInt;
begin
  t := Area.offset;
  Area.offset := CircleAdd(Area.offset, 1, TetrisWidth);
  if not CheckEdges then
    Area.offset := t;
end;

procedure MoveDown;
```

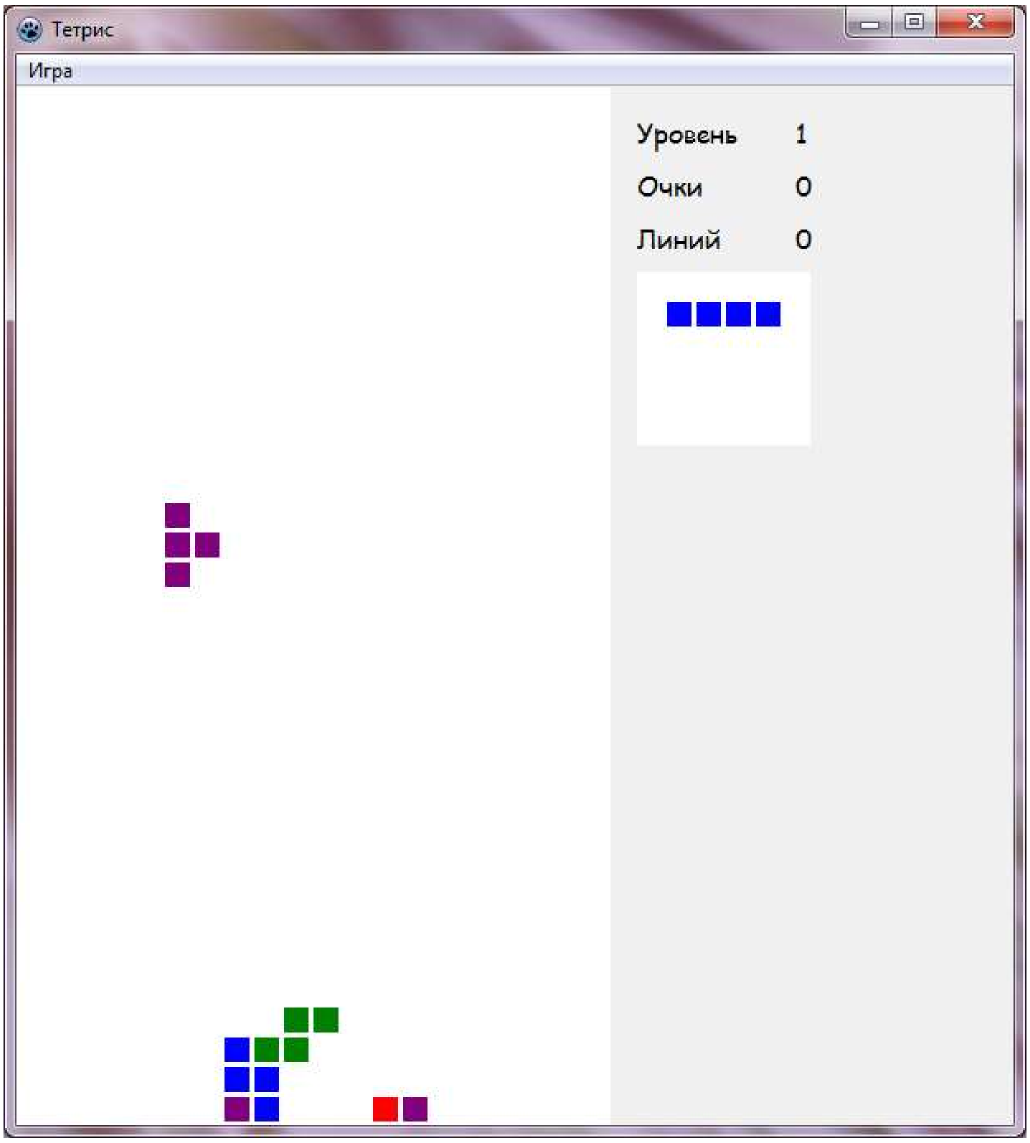
```
var
  i, j, offsetx, offsety : LongInt;
  m : TFigureMatrix;
begin
  m := Figures[Shape.n][Shape.p];
  for j := 0 to ShapeWidth - 1 do
  begin
    offsetx := CircleAdd(Area.offset, Shape.position.x + j, TetrisWidth);
    for i := 0 to ShapeHeight - 1 do
    begin
      offsety := Shape.position.y + 1 + i;
      if offsety <= 0 then
        continue;
      if (m[i][j] = 1) and (Area.surface[offsety][offsetx].n = 1) then
      begin
        AttachShape[Shape.t](Shape);
        exit;
      end;
    end;
  end;
  Inc(Shape.position.y);
end;

procedure RotateShape;
var
  t : LongInt;
begin
  t := Shape.p;
  Shape.p := CircleAdd(Shape.p, 1, 4);
  if not CheckEdges then
    Shape.p := t;
end;

end.
```

Модуль Records

```
unit Records;

{$mode objfpc}{$H+}{$M+}

interface

uses
  Classes, SysUtils, Dbf, db;

type
  TPlayerName = String[40];

  TPlayer = record
    Name: TPlayerName;
    Score: LongInt;
  end;

  TRecordTable = class(TObject)
     private
       _Size: Integer;
       _Players: array [1..8] of TPlayer;
       function _GetPlayer(i: Integer): TPlayer;
```

```
      function _GetPlayerPos(score: LongInt): Integer;
      procedure _CreateDataBase(path: String);
    public
      const Empty: TPlayer = (Name: ''; Score: 0);
      constructor Create;
      property Players[i: Integer]: TPlayer read _GetPlayer;
      property Size: Integer read _Size ;
      function IsNewRecord(score: LongInt): Boolean;
      procedure WriteNewRecord(name: TPlayerName; score: LongInt);
      procedure LoadFromFile(path: String);
      procedure SaveToFile(path: String);
    end;

implementation

  constructor TRecordTable.Create;
  var
    i: Integer;
  begin
    _Size:=8;
    for i:=1 to Size do
      _Players[i]:=Empty;
  end;

  function TRecordTable._GetPlayer(i: LongInt): TPlayer;
  begin
    if (i >= 1) and (i <= Size) then
      _GetPlayer:=_Players[i]
    else
      _GetPlayer:=Empty;
  end;

  function TRecordTable.IsNewRecord(score: LongInt): Boolean;
  begin
    IsNewRecord:=_GetPlayerPos(score) <= Size;
  end;

  procedure TRecordTable.WriteNewRecord(name: TPlayerName; score: LongInt);
  var
    i, p: Integer;
  begin
    p:=_GetPlayerPos(score);
    for i:=Size downto p + 1 do
      _Players[i]:=_Players[i - 1];
    _Players[p].Name:=name;
    _Players[p].Score:=score;
  end;

  procedure TRecordTable._CreateDataBase(path: String);
  var
    dbase: Tdbf;
    i: Integer;
  begin
    dbase:=Tdbf.Create(nil);
    try
      dbase.TableLevel:=7;
      dbase.Exclusive:=True;
      dbase.TableName:=path;
      with dbase.FieldDefs do
      begin
        Add('Id', ftAutoInc, 0, True);
        Add('Name', ftString, 40, True);
        Add('Score', ftInteger);
      end;
      with dbase do
      begin
        CreateTable;
        Open;
        for i:=1 to Size do
        begin
          Append;
          Fields[1].AsString:='';
          Fields[2].AsInteger:=0;
          Post;
        end;
        Close;
      end;
    finally
      dbase.Free;
    end;
  end;

  procedure TRecordTable.LoadFromFile(path: String);
  var
    source: TDbf;
    i: Integer;
  begin
    if FileExists(path) then
    begin
      source:=TDbf.Create(nil);
      try
        with source do
        begin
          TableLevel:=7;
          Exclusive:=True;
          TableName:=path;
          Open;
          Active:=True;
          First;
          for i:=1 to Size do
          begin
            _Players [i].Name:=Fields[1].AsString;
            _Players [i].Score:=Fields[2].AsLongint;
            Next;
          end;
          Active:=False;
          Close;
        end;
      finally
        source.Free;
      end;
    end;
  end;

  procedure TRecordTable.SaveToFile(path: String);
  var
    dest: TDbf;
    i: Integer;
  begin
    if not FileExists(path) then
      _CreateDataBase(path);
    dest:=TDbf.Create(nil);
    try
      with dest do
      begin
        TableLevel:=7;
        Exclusive:=True;
        TableName:=path;
        Open;
        Active:=True;
        First;
        for i:=1 to Size do
        begin
          Edit;
          FieldValues['Name']:=Players [i].Name;
          FieldValues['Score']:=Players [i].Score;
          Post;
          Next;
        end;
        Active:=False;
        Close;
      end;
    finally
      dest.Free;
    end;
  end;

  function TRecordTable._GetPlayerPos(score: LongInt): Integer;
  var
    i: Integer;
  begin
    _GetPlayerPos:=Size + 1;
    for i:=1 to Size do
      if score > Players[i].Score then
      begin
        _GetPlayerPos:=i;
        break;
      end;
  end;

end.
```

Экранная форма:



Уровень 1

Очки 0

Линий 0

Вывод: в ходе данной лабораторной работы были изучены средства работы с графикой, а также инструменты и методы, позволяющие организовать работу с базами данных, в среде разработки Delphi; разработано графическое приложение с использованием изученных средств.