```
unit LoadU;
                                                    procedure TLoad-
interface
                                                    ing_Screen.Timer1Timer(Sender: TObject);
uses
                                                    begin
 Winapi. Windows, Winapi. Messages, Sys-
                                                    Progressbar1.Position:=Progressbar1.Posi-
tem.SysUtils, System.Variants, System.Classes,
                                                    tion+25;
Vcl.Graphics,
                                                    if progressbar1.position=100 then
 Vcl.Controls, Vcl.Forms, Vcl.Dialogs,
                                                    begin
Vcl.ExtCtrls, Vcl.ComCtrls,
                                                      Timer1.Enabled:=false;
 Vcl.Imaging.GIFImg, Vcl.Imaging.pngimage,
                                                      frmLauncher.Show;
Vcl.StdCtrls;
                                                     Loading_Screen.Hide;
                                                    end;
type
 TLoading_Screen = class(TForm)
                                                    end;
  ProgressBar1: TProgressBar;
                                                    end.
  Timer1: TTimer;
                                                    unit Launcher;
  Image1: TImage;
                                                    interface
  Label1: TLabel;
                                                    uses
  procedure Timer1Timer(Sender: TObject);
                                                      Windows, Messages, SysUtils, Variants, Clas-
  procedure FormCreate(Sender: TObject);
                                                    ses, Graphics, Controls, Forms,
 private
                                                      Dialogs, StdCtrls, pngimage, ExtCtrls, Play-
  { Private declarations }
                                                    erU, EngineUI, Spin,
 public
                                                      engineclasses, math, Vcl.Menus, ShellAPI,
  { Public declarations }
                                                      Vcl.Buttons, Vcl.MPlayer;
 end:
                                                    type
var
                                                      TfrmLauncher = class(TForm)
 Loading_Screen: TLoading_Screen;
                                                       Image1: TImage;
implementation
                                                       sedMonitor: TSpinEdit;
uses Launcher;
                                                       Label1: TLabel;
                                                       Label2: TLabel:
{$R *.dfm}
procedure TLoading_Screen.FormCre-
                                                       sedWidth: TSpinEdit;
ate(Sender: TObject);
                                                       lblHeight: TLabel;
                                                       shpPlay: TShape;
begin
Image1.Picture.LoadFromFile('Im-
                                                       Label3: TLabel;
                                                       Label4: TLabel:
ages\load.png');
PostMessage(ProgressBar1.Handle, $0409, 0,
                                                       tmr: TTimer:
                                                       Label5: TLabel;
clBlue);
progressbar1.BarColor:=clblue;
                                                       shpClose: TShape;
end;
                                                       Image2: TImage;
                                                       Label6: TLabel;
```

Изм.	Лист	№докум.	Подпись	Дата

```
shpDisplaySettings: TShape;
                                                    ChessForm.Show;
                                                    if not cbxWindowed.checked then
  MainMenu1: TMainMenu;
  N1: TMenuItem;
                                                    begin
  N2: TMenuItem;
                                                     ChessForm.Top := screen.Monitors[sedMon-
  N3: TMenuItem;
                                                   itor.Value - 1].Top;
  Panel1: TPanel;
                                                      ChessForm.Left := screen.Monitors[sed-
  MediaPlayer1: TMediaPlayer;
                                                   Monitor. Value - 1]. Left;
  VolumeButton: TSpeedButton;
                                                      gameHeight := screen.Monitors[sedMoni-
  cbxWindowed: TCheckBox;
                                                   tor. Value - 1]. Height;
  procedure LaunchGame;
                                                      gameWidth := screen.Monitors[sedMoni-
  procedure FormCreate(Sender: TObject);
                                                   tor. Value - 1]. Width;
  procedure sedWidthChange(Sender:
                                                     ChessForm.WindowState := wsMaximized;
TObject);
                                                    end
  procedure tmrTimer(Sender: TObject);
                                                    else
  procedure N2Click(Sender: TObject);
                                                    begin
  procedure N3Click(Sender: TObject);
                                                     ChessForm.Top := screen.Monitors[sedMon-
  procedure FormShow(Sender: TObject);
                                                   itor.Value - 1].Top;
  procedure VolumeButtonClick(Sender:
                                                      ChessForm.Left := screen.Monitors[sed-
TObject);
                                                   Monitor. Value - 1]. Left;
                                                      gameHeight := iHeight;
 private
  { Private declarations }
                                                      gameWidth := sedWidth.Value;
 public
                                                      ChessForm.ClientHeight := gameHeight;
  isMusicAllowed:boolean;
                                                     ChessForm.ClientWidth := gameWidth;
                                                     ChessForm.roundEdges;
 end;
var
                                                    end:
 frmLauncher: TfrmLauncher:
                                                    ChessForm.reloadGame:
 showingSettings : boolean = false;
 iHeight: integer = 360;
                                                   procedure TfrmLauncher.N2Click(Sender:
 rectPlay, rectSettings: TShape;
                                                   TObject);
                                                   begin
const
 heightWOSettings = 335;
                                                   ShellExecute(0, PChar ('Open'), PChar ('New-
 heightWSettings = 460;
                                                   Project.chm'), nil, nil, SW_SHOW);
implementation
                                                   end;
{$R *.dfm}
                                                   procedure TfrmLauncher.N3Click(Sender:
procedure TfrmLauncher.LaunchGame;
                                                   TObject);
                                                   begin
begin
 ChessForm.PlayerRefresh.Enabled := false;
                                                   Application. Terminate;
 ChessForm.WindowState := wsNormal;
                                                   end;
```

I					
I					
ſ	Изм.	Лист	№докум.	Подпись	Дата

```
procedure TfrmLauncher.FormCreate(Sender:
                                                       begin
TObject);
                                                         MessageDlg('Неверный путь к файлу.
                                                    Возможно его больше не существует. По-
var
                                                    пробуйте еще.',vcl.Dialogs.mtError,
 rgn: HRGN;
begin
                                                   mbOKCancel, 0);
 isMusicAllowed:=true;
                                                         exit;
 Image1.Picture.LoadFromFile('Im-
                                                       end;
ages\Launch_horse.bmp');
 Image2.Picture.LoadFromFile('Im-
                                                     end;
ages\close.png');
                                                   end;
 label6.Caption := 'ПРИЯТНОЙ ' + #13 +
'И Г Р Ы';
 sedMonitor.MaxValue := screen.Monitor-
                                                   procedure TfrmLauncher.sed-
Count:
                                                   WidthChange(Sender: TObject);
 if screen.MonitorCount = 1 then
                                                   begin
  sedMonitor.Enabled := false;
                                                     if sedWidth.Text <> " then
 sedWidth.MaxValue := screen.Width;
                                                      iHeight := ceil(sedWidth.value/(16/9));
 lblHeight.Caption := format('X %d',
                                                     lblHeight.Caption := format('X %d',
[iHeight]);
                                                    [iHeight]);
 rgn := CreateRoundRectRgn(0,
                                                   end;
  0,
  ClientWidth,
                                                   procedure TfrmLauncher.tmrTimer(Sender:
  ClientHeight,
                                                   TObject);
                                                   begin
  20,
  20);
 SetWindowRgn(Handle, rgn, True);
                                                     if (mouse.CursorPos.X >= shpPlay.Cli-
procedure TfrmLauncher.FormShow(Sender:
                                                   entToScreen(Point(0, 0)).X) AND
TObject);
                                                      (mouse.CursorPos.X <= shpPlay.Cli-
                                                   entToScreen(Point(shpPlay.Width, 0)).X)
begin
  var path:string:=ExtractFilePath((Applica-
                                                      AND (mouse.CursorPos.Y >= shpPlay.Cli-
tion.ExeName));
                                                   entToScreen(Point(0, 0)).Y) AND
  self.MediaPlayer1.FileName := path + '\Im-
                                                      (mouse.CursorPos.Y <= shpPlay.Cli-
ages\TheHappyBride.mp3';
                                                   entToScreen(Point(0, shpPlay.Height)).Y)
                                                      then
 try
   self.MediaPlayer1.Open();
                                                     begin
    self.MediaPlayer1.Play();
                                                      while GETGVALUE(shpPlay.Brush.color) >
                                                    $BE do
  except
```

Изм.	Лист	№доким.	Подпись	Дата

```
begin
                                                       Application.ProcessMessages;
   shpPlay.Brush.color := shpPlay.Brush.color
                                                       cbxWindowed.Color := shpDisplaySet-
- $000100;
                                                    tings.Brush.Color;
   Application.ProcessMessages;
                                                      end;
                                                     end
  end:
  if GetKeyState(VK_LBUTTON) < 0 then
                                                     else
  begin
                                                     begin
   tmr.Enabled := false;
                                                      while GETGVALUE(shpDisplaySet-
   LaunchGame;
                                                    tings.Brush.color) < $DD do
  end:
                                                      begin
 end
                                                       shpDisplaySettings.Brush.color := shpDis-
                                                    playSettings.Brush.color + $000100;
 else
                                                       Application.ProcessMessages;
 begin
  while GETGVALUE(shpPlay.Brush.color) <
                                                       cbxWindowed.Color := shpDisplaySet-
$DD do
                                                    tings.Brush.Color;
                                                      end;
  begin
   shpPlay.Brush.color := shpPlay.Brush.color
                                                     end;
+ $000100;
                                                     if (mouse.CursorPos.X >= shpClose.Cli-
   Application.ProcessMessages;
                                                    entToScreen(Point(0, 0)).X) AND
  end;
                                                      (mouse.CursorPos.X <= shpClose.Cli-
 end;
                                                    entToScreen(Point(shpClose.Width, 0)).X)
 if (mouse.CursorPos.X >= shpDisplaySet-
                                                      AND (mouse.CursorPos.Y \geq shpClose.Cli-
tings.ClientToScreen(Point(0, 0)).X) AND
                                                    entToScreen(Point(0, 0)).Y) AND
  (mouse.Curso rPos.X <= shpDisplaySet-
                                                      (mouse.CursorPos.Y <= shpClose.Cli-
tings.ClientToScreen(Point(shpDisplaySet-
                                                    entToScreen(Point(0, shpClose.Height)).Y)
tings. Width, (0). (X)
                                                      then
  AND (mouse.CursorPos.Y >= shpDisplay-
                                                     begin
Settings.ClientToScreen(Point(0, 0)).Y) AND
                                                      while GETGVALUE(shpClose.Brush.color)
  (mouse.CursorPos.Y <= shpDisplaySet-
                                                    > $BE do
tings.ClientToScreen(Point(0, shpDisplaySet-
                                                      begin
tings.Height)).Y)
                                                       shpClose.Brush.color :=
  then
                                                    shpClose.Brush.color - $000100;
 begin
                                                       Application.ProcessMessages;
  while GETGVALUE(shpDisplaySet-
                                                      end:
tings.Brush.color) > $BE do
                                                      if GetKeyState(VK_LBUTTON) < 0 then
  begin
                                                      begin
   shpDisplaySettings.Brush.color := shpDis-
                                                       tmr.Enabled := false;
playSettings.Brush.color - $000100;
                                                       Application. Terminate;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
end;
                                                         MessageDlg('Неверный путь к файлу.
 end
                                                    Возможно его больше не существует. По-
                                                    пробуйте еще.',vcl.Dialogs.mtError,
 else
                                                    mbOKCancel, 0);
 begin
  while GETGVALUE(shpClose.Brush.color)
                                                         exit;
< $DD do
                                                        end;
  begin
                                                       end;
   shpClose.Brush.color :=
                                                     end;
shpClose.Brush.color + $000100;
                                                    end;
   Application.ProcessMessages;
                                                    end.
  end;
                                                    unit PlayerU;
 end;
                                                    interface
end;
                                                    uses
procedure TfrmLauncher. VolumeButton-
                                                     Windows, Messages, SysUtils, Variants, Clas-
Click(Sender: TObject);
                                                    ses, Graphics, Controls, Forms,
                                                     Dialogs, EngineClasses, jpeg, math, StdCtrls,
begin
if self.isMusicAllowed then
                                                    ImgList, ExtCtrls, pngimage,
 begin
                                                     Menus, ActnList, EngineUI, System. Actions,
  self.isMusicAllowed:=false;
                                                    System.ImageList, Vcl.Buttons;
  self.VolumeButton.Glyph.LoadFrom-
                                                    type
File(ExtractFilePath(Application.ExeName)
                                                     TChessForm = class(TForm)
+'/Images/mute.bmp');
                                                      PlayerRefresh: TTimer;
                                                      lblWhiteTitle: TLabel:
   self.MediaPlayer1.Stop
                                                      lblBlackTitle: TLabel:
 end
 else
                                                      lblWPiecesTook: TLabel;
                                                      lblBPiecesTook: TLabel:
 begin
   self.isMusicAllowed:=true;
                                                      highlightblock: TImage;
   self.VolumeButton.Glyph.LoadFrom-
                                                      imgClose: TImage;
File(ExtractFilePath(Application.ExeName)
                                                      imgCloseHover: TImage;
                                                      imgCloseDef: TImage;
+'/Images/speaker.bmp');
                                                      Settings: TActionList;
   try
                                                      setWhiteColor: TAction;
    MediaPlayer1.FileName:=Extract-
                                                      setBlackColor: TAction;
FilePath(Application.ExeName)+ '\Im-
ages\TheHappyBride.mp3';
                                                      setOutlineColor: TAction:
                                                      setBackColor: TAction;
    self.MediaPlayer1.Open();
     MediaPlayer1.Play
                                                      autoDeselect: TAction:
                                                      saveDirSet: TAction;
   except
    begin
                                                      saveSettings: TAction;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
resetSettings: TAction;
                                                       procedure FormMouseDown(Sender:
  setAssetsPath: TAction;
                                                    TObject; Button: TMouseButton;
  setUIScale: TAction;
                                                        Shift: TShiftState; X, Y: Integer);
  AssetsList: TImageList;
                                                       procedure setUIScaleExecute(Sender:
  cldlg: TColorDialog;
                                                    TObject);
  procedure FormDestroy(Sender: TObject);
                                                       procedure exitButtonClick(Sender: TObject);
  procedure FormKeyDown(Sender: TObject;
                                                       function GetData(FilePath: string; Tag:
var Key: Word; Shift: TShiftState);
                                                    string): string;
  procedure PlayerRefreshTimer(Sender:
                                                     private
TObject);
                                                       { Private declarations }
  procedure imgCloseClick(Sender: TObject);
                                                     public
  procedure imgCloseMouseEnter(Sender:
                                                       { Public declarations }
TObject);
                                                     end;
  procedure imgCloseMouseLeave(Sender:
                                                    var
                                                     ChessForm: TChessForm;
TObject);
  procedure FormCreate(Sender: TObject);
                                                      BoardMannager: TBoardMannager;
  procedure setWhiteColorExecute(Sender:
                                                     highlightblock: TImage;
TObject);
                                                      SaveManager: TSaveManager;
  procedure setBlackColorExecute(Sender:
                                                      AssetPath: string = 'default';
                                                     turnColor: TColor:
TObject);
  procedure setOutlineColorExecute(Sender:
                                                      selectColor: TColor;
TObject);
                                                     scaleMultplier : real = 1;
  procedure setBackColorExecute(Sender:
                                                    implementation
TObject);
                                                    {$R *.dfm}
  procedure autoDeselectExecute(Sender:
                                                    procedure TChessForm.autoDeselectEx-
TObject);
                                                    ecute(Sender: TObject);
  procedure SetSettings;
                                                    begin
  procedure saveSettingsExecute(Sender:
                                                     BoardMannager.autoDeselect := not Board-
TObject);
                                                    Mannager.autoDeselect;
  procedure showDebugExecute(Sender:
                                                     autoDeselect.Checked := BoardMannager.au-
TObject);
                                                    toDeselect
  procedure resetSettingsExecute(Sender:
                                                    end;
TObject);
                                                    function TChessForm.GetData(FilePath:
  procedure reloadGame;
                                                    string; Tag: string): string;
  procedure setAssetsPathExecute(Sender:
                                                    var
TObject);
                                                     tS: TextFile:
  procedure roundEdges;
                                                     s: string;
  procedure ScaleComponents;
                                                    begin
```

ı					
I					
ĺ	Изм.	Лист	№докцм.	Подпись	Дата

```
if not fileExists(filepath) then
                                                        imageSize := 32;
 begin
                                                       tempbm := TBitmap.Create;
  result := 'default';
                                                       with tempbm do
  exit;
                                                       begin
 end;
                                                        PixelFormat := pf32bit;
 AssignFile(ts, FilePath);
                                                        Height := imageSize;
 reset(ts);
                                                         Width := Height;
 while (Pos(Tag, s) = 0) AND (not eof(tS)) do
                                                       end;
  readln(ts, s);
                                                       BoardMannager := TBoardMannager.Cre-
 if eof(ts) then begin closeFile(tS); exit end;
                                                      ate(Self);
 delete(s, 1, pos('[', s));
                                                       SaveManager := TSaveManager.Create(Self);
 Result := Copy(s, 1, pos(']', s) - 1);
                                                       SaveManager.LinkedBoard := BoardMan-
 closeFile(tS);
                                                      nager;
                                                       color := rgb(102, 202, 255);
end;
procedure TChessForm.FormCreate(Sender:
                                                       SetSettings:
TObject);
                                                       if AssetPath = 'default' then
var
 tempbm: TBitmap;
                                                          AssetsList.Draw(BoardMan-
 settingDat: string;
                                                      nager.Bishop.Canvas, 0, 0, 0, true);
begin
                                                          AssetsList.Draw(BoardMannager.Cas-
 highlightblock.Picture.LoadFromFile('Im-
                                                      tle.Canvas, 0, 0, 1, true);
ages\block.png');
                                                          AssetsList.Draw(BoardMan-
 imgClose.Picture.LoadFromFile('Im-
                                                      nager.horse.Canvas, 0, 0, 2, true);
ages\close.png');
                                                          AssetsList.Draw(BoardMan-
 imgCloseDef.Picture.LoadFromFile('Im-
                                                      nager.king.Canvas, 0, 0, 3, true);
ages\white_block.png');
                                                          AssetsList.Draw(BoardMan-
 imgCloseHover.Picture.LoadFromFile('Im-
                                                      nager.pawn.Canvas, 0, 0, 4, true);
                                                          AssetsList.Draw(BoardMan-
ages\close.png');
 settingDat := getdata('_SETTINGS.DWCS',
                                                      nager.queen.Canvas, 0, 0, 5, true);
'AssetsDir');
                                                         finally
 if DirectoryExists(settingDat) then
                                                          BoardMannager.Orientation := orTop_Bot-
  AssetPath := settingDat
                                                      tom;
 else
                                                          BoardMannager.InitialDraw;
  AssetPath := 'default':
                                                        end:
 if AssetPath <> 'default' then
                                                       scalecomponents;
  imageSize := StrToInt(getdata(AssetPath +
                                                       autoDeselect.Checked := BoardMannager.au-
'\_SETUP.DWCS', 'ImageSize'))
                                                      toDeselect:
 else
                                                      end:
```

Изм.	Лист	№докум.	Подпись	Дата

```
procedure TChessForm.FormDestroy(Sender:
                                                     begin
TObject);
                                                      ReleaseCapture;
begin
                                                      Perform(WM_SYSCOMMAND,
                                                   SC_DRAGMOVE, 0);
 BoardMannager.destroy;
end;
                                                     end;
procedure TChess-
                                                  end;
Form.FormKeyDown(Sender: TObject; var
                                                  procedure TChessForm.imgCloseClick(Sender:
Key: Word;
                                                  TObject);
 Shift: TShiftState);
                                                  begin
                                                    Application.Terminate;
begin
 case key of
  VK_ESCAPE:
                                                  procedure TChessForm.imgClose-
  begin
                                                  MouseEnter(Sender: TObject);
   if boardmannager.selected then
                                                  begin
                                                    imgClose.Picture := imgCloseHover.Picture;
   begin
   boardmannager.selected := false;
   if boardmannager. Turn = 1 then
                                                  procedure TChessForm.imgClose-
                                                  MouseLeave(Sender: TObject);
    boardmannager.turn := 2
   else
                                                  begin
    boardmannager.turn := 1;
                                                    imgClose.Picture := imgCloseDef.Picture;
   end;
  end;
                                                  procedure TChessForm.PlayerRe-
  VK END:
                                                   freshTimer(Sender: TObject);
  begin
                                                   var
   BoardMannager.Clear;
                                                    sWPT, sBPT: string;
   BoardMannager.InitialDraw;
                                                    I: Integer;
  end;
                                                    y, x, newKind: Integer;
                                                  begin
 end;
end:
                                                    turnColor := RGB(GetGValue(Color), Get-
                                                  BValue(Color), GetRValue(Color));
procedure TChess-
Form.FormMouseDown(Sender: TObject; But-
                                                    selectColor := RGB(GetBValue(Color),
                                                   GetRValue(Color), GetGValue(Color));
ton: TMouseButton;
                                                    lblWhiteTitle.Caption := 'БЕЛЫЙ ИГРОК';
 Shift: TShiftState; X, Y: Integer);
                                                    lblBlackTitle.Caption := 'ЧЕРНЫЙ ИГРОК';
const
 SC_DRAGMOVE = $F012;
                                                    for I := 0 to boardmannager.getBlackTook-
begin
                                                   length do
 if WindowState = wsNormal then
                                                    begin
  if Button = mbLeft then
                                                     case boardMannager.BlackPiecesTook[i] of
```

ĺ					
I					
I	Изм.	Лист	№докцм.	Подпись	Дата

Лист

```
0:;
                                                        lblBlackTitle.font.Color := clblack;
    1: sBPT := sBPT + 'Пешка' + nl;
                                                       end
   2: sBPT := sBPT + 'Ладья' + nl;
                                                       else if (BoardMannager.turn = 1) AND
   3: sBPT := sBPT + 'Слог' + nl;
                                                      (BoardMannager.Selected) then
   4: sBPT := sBPT + 'Конь' + nl;
                                                       begin
   5: sBPT := sBPT + 'Ферзь' + nl;
                                                        lblWhiteTitle.font.Color := clblack;
  end:
                                                        lblBlackTitle.font.Color := selectcolor;
 end;
                                                       end;
 for I := 0 to boardmannager.getWhiteTook-
                                                       with boardmannager do
length do
                                                       begin
 begin
                                                        if Orientation = orTop_Bottom then
  case boardMannager.WhitePiecesTook[i] of
                                                        begin
   0:;
                                                        for y := 1 to 2 do
                                                         for x := 1 to 8 do
   1: sWPT := sWPT + 'Пешка' + nl;
   2: sWPT := sWPT + 'Ладья' + nl;
                                                          if (board[x, y * 7 - 6].kind = 1) then
   3: sWPT := sWPT + 'Слон' + nl;
                                                          begin
   4: sWPT := sWPT + 'Конь' + nl;
                                                            board[x, y * 7 - 6].Kind := 0;
   5: sWPT := sWPT + 'Ферзь' + nl;
                                                            newKind := pickpawnpromotion;
                                                            SetSquareTo(point(x, y*7-6),
  end;
                                                      newKind);
 end;
 lblWPiecesTook.caption := sWPT;
                                                           end
 lblBPiecesTook.Caption := sBPT;
                                                           else if (board[x, y * 7 - 6].kind = -1) then
 if (BoardMannager.turn = 1) AND (Not
                                                          begin
                                                            board[x, y * 7 - 6].Kind := 0;
BoardMannager.Selected) then
                                                            newKind := pickpawnpromotion;
 begin
  lblWhiteTitle.font.Color := turncolor:
                                                            SetSquareTo(point(x, y*7-6), -1 *
  lblBlackTitle.font.Color := clblack:
                                                     newKind);
 end
                                                          end:
 else if (BoardMannager.turn = 2) AND (Not
                                                        end
BoardMannager.Selected) then
                                                        else
 begin
                                                        begin
  lblWhiteTitle.font.Color := clblack;
                                                        for x := 1 to 2 do
  lblBlackTitle.font.Color := turncolor;
                                                         for y := 1 to 8 do
 end
                                                          if board[x * 7 - 6, y].kind = 1 then
 else if (BoardMannager.turn = 2) AND
                                                          begin
(BoardMannager.Selected) then
                                                            board[x * 7 - 6, y].Kind := 0;
 begin
                                                            newKind := pickpawnpromotion;
  lblWhiteTitle.font.Color := selectcolor;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
SetSquareTo(point(x * 7 - 6, y),
                                                    procedure TChessForm.resetSettingsExe-
newKind);
                                                    cute(Sender: TObject);
    end
                                                    var
                                                     tS: textfile;
    else if board[x * 7 - 6, y].kind = -1 then
                                                    begin
    begin
                                                      assignfile(ts, '_SETTINGS.DWCS');
      board[x * 7 - 6, y].Kind := 0;
      newKind := pickpawnpromotion;
                                                     rewrite(ts);
      SetSquareTo(point(x * 7 - 6, y), -1 *
                                                     write(tS, 'WhiteColor=[default]'#13#10'Black-
newKind);
                                                    Color=[default]'#13#10'OutlineColor=[de-
                                                    fault]'#13#10'BackColor=[de-
    end:
  end;
                                                    fault]'#13#10'SaveDir=[default]'#13#10'Auto-
                                                    Deselect=[default]'#13#10'ShowDebug=[de-
 end;
 if boardmannager.selected then
                                                    fault]'#13#10'AssetsDir=[de-
                                                    fault]'#13#10'END');
 begin
  highlightblock. Visible := true;
                                                     closefile(tS);
  highlightblock.Top := boardmannager.Se-
                                                     reloadGame;
lectedSqr.Top;
                                                    end;
  highlightblock.Left := boardmannager.Se-
                                                    procedure TChessForm.roundEdges;
lectedSqr.left;
                                                     var
 end
                                                     rgn: HRGN;
 else
                                                    begin
  highlightblock. Visible := false;
                                                     rgn := CreateRoundRectRgn(0,
                                                       0.
end:
procedure TChessForm.reloadGame;
                                                       chessform.ClientWidth.
                                                       chessform.ClientHeight,
begin
 SaveManager.SaveToFileOverwrite(' RE-
                                                       40.
SETTEMP.DWCS');
                                                       40);
 PlayerRefresh.Enabled := false;
                                                      SetWindowRgn(chessform.Handle, rgn,
 BoardMannager.destroy;
                                                    True):
 BoardMannager := nil;
                                                    end;
 SaveManager.Destroy;
                                                    procedure TChessForm.saveSettingsExe-
 FormCreate(nil);
                                                    cute(Sender: TObject);
 SaveManager.LoadFromFile('_RESET-
                                                    var
TEMP.DWCS');
                                                     tS: textFile:
 DeleteFile('_RESETTEMP.DWCS');
                                                     showDebug, autoDeselect: string;
 DeleteFile('_RESETTEMP.PGN');
                                                    begin
 PlayerRefresh.Enabled := true;
                                                     if NOT BoardMannager.Debug.Visible then
                                                       showdebug := 'false'
end;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
else
                                                      lblWhiteTitle.Left := 8;
  showdebug := 'true';
                                                      lblBPiecesTook.Left := 8;
 if NOT BoardMannager.AutoDeselect then
                                                      lblBPiecesTook.Top := lblWhiteTitle.Top +
  autoDeselect := 'false'
                                                    lblWhiteTitle.Height + 8;
 else
                                                      lblBlackTitle.Top := 8;
  autoDeselect := 'true';
                                                      lblBPiecesTook.Font.Size := Ceil((12 /
 assignfile(ts, '_SETTINGS.DWCS');
                                                     (1080/ClientHeight))* scaleMultplier);
                                                      lblBlackTitle.Font.Size := Ceil((20 /
 rewrite(ts);
                                                    (1080/ClientHeight))* scaleMultplier);
 write(tS, format(
   'WhiteColor=[%d]'#13#10'Black-
                                                      lblBlackTitle.Left := BoardMannager.get-
Color=[%d]'#13#10'Out-
                                                    LastSquareLeft +
lineColor=[%d]'#13#10"
                                                       BoardMannager.getSquareHeightWidth + 8;
    + 'Back-
                                                      lblWPiecesTook.Top := lblBlackTitle.Top +
Color=[%d]'#13#10'SaveDir=[%s]'#13#10'Au-
                                                    lblBlackTitle.Height + 8;
toDese-
                                                      lblWPiecesTook.Font.Size := Ceil((12 /
lect=[%s]'#13#10'ShowDebug=[%s]'#13#10'As
                                                    (1080/ClientHeight))* scaleMultplier);
setsDir=[%s]'#13#10'END',
                                                      lblWPiecesTook.Left := BoardMannager.get-
    [rgb(GetBValue(BoardMannager.White-
                                                    LastSquareLeft +
Color), GetGValue(BoardMannager.White-
                                                       BoardMannager.getSquareHeightWidth + 8;
Color), GetRValue (BoardMannager. White-
                                                      highlightblock.BringToFront;
Color)),
                                                      highlightblock.Parent := Self;
    rgb(GetBValue(BoardMannager.Black-
                                                      highlightblock.Stretch := true;
Color), GetGValue(BoardMannager.Black-
                                                     highlightblock. Visible := false;
                                                      highlightblock.Height := BoardMan-
Color), GetRValue(BoardMannager.Black-
Color)),
                                                    nager.Board[1, 1].Height;
    rgb(GetBValue(BoardMannager.Out-
                                                      highlightblock.Width := BoardMan-
lineColor), GetGValue(BoardMannager.Out-
                                                    nager.Board[1, 1].Width;
lineColor),GetRValue(BoardMannager.Out-
                                                      imgClose.Width := Ceil((45 / (1080/Clien-
lineColor)),
                                                    tHeight))* scaleMultplier);
    color, savemanager.rootDir, autoDeselect,
                                                      imgClose.Height := Ceil((45 / (1080/Clien-
showdebug, assetPath]));
                                                    tHeight))* scaleMultplier);
 closefile(tS);
                                                      imgClose.Left := chessform.Width -
end:
                                                    imgClose.Width - 8;
procedure TChessForm.ScaleComponents;
                                                      BoardMannager.Debug.Font.Size := Ceil((10
                                                    /(1080/ClientHeight))* scaleMultplier);
begin
 lblWhiteTitle.Top := 8;
 lblWhiteTitle.Font.Size := Ceil((20 /
                                                    procedure TChessForm.setAssetsPathExe-
(1080/ClientHeight)) * scaleMultplier);
                                                    cute(Sender: TObject);
```

Изм.	Лист	№докцм.	Подпись	Дата

```
begin
var
 prePath: string;
                                                     clDlg.Color := rgb(GetBValue(BoardMan-
 accept: integer;
                                                    nager.OutlineColor), GetGValue(BoardMan-
                                                    nager.OutlineColor),GetRValue(BoardMan-
begin
 prePath := AssetPath;
                                                    nager.OutlineColor));
 AssetPath := InputBox(", ", AssetPath);
                                                     clDlg.Execute();
 if prePath <> AssetPath then
                                                     if clDlg.Color = $000000 then
  accept := MessageDlg(", mtConfirmation,
                                                       clDlg.Color := $000001;
[mbYes, mbNo], 0);
                                                     if clDlg.Color = $FFFFFF then
 if accept = mrYes then
                                                       clDlg.Color := $FFFFE;
                                                     BoardMannager.OutlineColor := clDlg.Color;
 begin
  saveSettingsExecute(nil);
                                                    end;
  reloadGame;
                                                    procedure TChessForm.SetSettings;
 end;
end;
                                                     settingDat: string;
procedure TChessForm.setBackColorExe-
                                                    begin
cute(Sender: TObject);
                                                     if fileexists('_SETTINGS.DWCS') then
begin
                                                     begin
 cldlg.Color := color;
                                                       settingDat := getdata('_SETTINGS.DWCS',
                                                     'WhiteColor');
 clDlg.Execute();
 Color := clDlg.Color;
                                                       if settingDat <> 'default' then
end;
                                                        BoardMannager.WhiteColor :=
procedure TChessForm.setBlackColorExe-
                                                    StrToInt(settingDat);
cute(Sender: TObject);
                                                       settingDat := getdata('_SETTINGS.DWCS',
                                                    'BlackColor');
begin
 clDlg.Color := rgb(GetBValue(BoardMan-
                                                       if settingDat <> 'default' then
nager.BlackColor), GetGValue(BoardMan-
                                                        BoardMannager.BlackColor :=
                                                    StrToInt(settingDat);
nager.BlackColor),GetRValue(BoardMan-
nager.BlackColor));
                                                       settingDat := getdata('_SETTINGS.DWCS',
 clDlg.Execute();
                                                    'OutlineColor');
 if clDlg.Color = $000000 then
                                                       if settingDat <> 'default' then
  clDlg.Color := $000001;
                                                        BoardMannager.OutlineColor :=
 if clDlg.Color = $FFFFFF then
                                                    StrToInt(settingDat);
  clDlg.Color := $FFFFE;
                                                       settingDat := getdata('_SETTINGS.DWCS',
 BoardMannager.BlackColor := clDlg.Color;
                                                    'BackColor');
 end:
                                                       if settingDat <> 'default' then
procedure TChessForm.setOutlineColorExe-
                                                        chessForm.Color := StrToInt(settingDat);
cute(Sender: TObject);
```

Изм.	Лист	№докцм.	Подпись	Дата

```
settingDat := getdata('_SETTINGS.DWCS',
                                                      clDlg.Color := $FFFFE;
'SaveDir');
                                                     BoardMannager.WhiteColor := clDlg.Color;
  if settingDat <> 'default' then
                                                     SaveManager.SaveToFileOver-
   SaveManager.rootDir := settingDat;
                                                    write(SaveManager.rootDir +
  settingDat := getdata('_SETTINGS.DWCS',
                                                    '\_TEMPSAVE.DWCS');
'ShowDebug');
                                                     SaveManager.LoadFrom-
   if settingDat <> 'default' then
                                                    File(SaveManager.rootDir +
    if settingDat = 'false' then
                                                    '\_TEMPSAVE.DWCS');
      BoardMannager.Debug.Visible := false;
                                                     deleteFile(SaveManager.rootDir +
  settingDat := getdata('_SETTINGS.DWCS',
                                                    '\_TEMPSAVE.DWCS');
'AutoDeselect');
                                                     deletefile(SaveManager.rootDir +
  if settingDat <> 'default' then
                                                    '\_TEMPSAVE.PGN');
   if settingDat = 'false' then
                                                    end;
    BoardMannager.AutoDeselect := false;
                                                    procedure TChessForm.showDebugExe-
                                                    cute(Sender: TObject);
 end;
end;
                                                    begin
procedure TChessForm.setUIScaleExe-
                                                     BoardMannager.Debug.Visible := not Board-
cute(Sender: TObject);
                                                    Mannager.Debug.Visible;
                                                    end;
var
 newScaleM : real;
                                                    procedure TChessForm.ExitButton-
                                                    Click(Sender: TObject);
begin
 newScaleM := strtofloat(inputbox('Set new UI
                                                    begin
                                                     self.Close();
Scale', 'Enter a scale multiplier [Any real num-
ber]', FloatToStr(scaleMultplier)));
                                                    end;
 scaleMultplier := newscaleM;
                                                    end.
 ScaleComponents;
                                                    unit EngineUI;
end;
                                                    interface
procedure TChessForm.setWhiteColorExe-
                                                    uses
cute(Sender: TObject);
                                                     ExtCtrls, Windows, Messages, SysUtils, Vari-
begin
                                                    ants, Classes, Graphics, Controls,
                                                     Forms, Dialogs, Math, StdCtrls, pngimage;
 clDlg.Color := rgb(GetBValue(BoardMan-
nager.WhiteColor), GetGValue(BoardMan-
                                                    function PickPawnPromotion: byte;
nager.WhiteColor),GetRValue(BoardMan-
                                                    implementation
nager.WhiteColor));
                                                    function PickPawnPromotion: byte;
 clDlg.Execute();
                                                    var
 if clDlg.Color = $000000 then
                                                     frm: TForm;
                                                     rgn: HRGN;
  clDlg.Color := $000001;
 if clDlg.Color = $FFFFFF then
                                                     Done: boolean;
```

ı					
	Изм.	Лист	№докцм.	Подпись	Дата

```
lbl: TLabel;
                                                          case i of
 sR: array of string;
                                                           1: sr[i - 1] := 'ЛАДЬЯ';
                                                           2: sr[i - 1] := 'СЛОН';
 i, y, i2: integer;
 clickRegions: array of trect;
                                                           3: sr[i - 1] := 'KOHb';
 rects: array of TShape;
                                                           4: sr[i - 1] := '\Phi EP3b';
begin
                                                          end;
 Done := false;
                                                          lbl := TLabel.Create(frm);
 frm := TForm.CreateNew(nil, 0);
                                                          with lbl do
 frm.BorderStyle := bsNone;
                                                          begin
 frm.AlphaBlend := true;
                                                           parent := frm;
 frm.AlphaBlendValue := 0;
                                                           Font.Name := 'Arial';
 lbl := TLabel.Create(frm);
                                                           Font.Size := 18;
 lbl.Parent := frm;
                                                           Font.Color := rgb(105,97,225);
 lbl.Caption := '\Pi Р О В Е Д Е Н Н А Я \Pi Е
                                                           Font.Style := [fsBold];
ШКА';
                                                           Top := round(rects[i-1].Top + (rects[i-
 lbl.Font.Name := 'Arial';
                                                        1].Height/2) - (Height/2));
 lbl.Font.Size := 18;
                                                           Caption := sR[i-1];
 lbl.Font.Color := $5D2FFF;
                                                           Left := round((frm.Width/2) -
 lbl.Font.Style := [fsBold];
                                                       (lbl.Width/2));
 frm.ClientWidth := lbl.Width + 40;
                                                          end;
 lbl.left := round((frm.ClientWidth/2) -
                                                         end;
(lbl.Width/2));
                                                         frm.ClientHeight := rects[i-2].Top + rects[i-
 1b1.Top := 20;
                                                        2].Height + 20;
                                                         frm.Position := poScreenCenter;
 i := 0;
 for i := 1 to 4 do
                                                         rgn := CreateRoundRectRgn(0,
 begin
                                                          0.
  SetLength(rects, i);
                                                          frm.ClientWidth,
  SetLength(clickRegions, i);
                                                          frm.ClientHeight,
  SetLength(sR, i);
                                                          20.
  rects[i-1] := TShape.Create(frm);
                                                          20);
  with rects[i-1] do
                                                         SetWindowRgn(frm.Handle, rgn, True);
                                                         frm.Color := rgb(168,244,255);
  begin
   Parent := frm;
                                                         frm.DoubleBuffered := true;
   Width := frm.ClientWidth:
                                                         frm.Show:
   brush.Color := $FFDD69;
                                                         i2 := 0;
   top := i*(height+5);
                                                         while i2 < 250 do
   pen.Style := psClear;
                                                         begin
  end;
                                                          inc(i2, 2);
```

Γ	Изм.	Лист	№докцм.	Подпись	Дата

```
'\Pi': result := 2;
  frm.AlphaBlendValue := i2;
                                                             'C': result := 3;
  Application.ProcessMessages;
                                                             'K': result := 4;
 end;
                                                             '\Phi': result := 5;
 for y := 0 to i - 2 do
 begin
                                                            end:
  clickRegions[y].Left := rects[y].Cli-
                                                         end
entToScreen(point(0,0)).x;
                                                         else
  clickRegions[y].Top := rects[y].Cli-
                                                         begin
entToScreen(point(0,0)).y;
                                                          while
  clickRegions[y].Bottom := rects[y].Cli-
                                                      GETGVALUE(rects[y].Brush.Color) < $DD
entToScreen(point(0,0 + rects[y].height)).y;
                                                      do
  clickRegions[y].Right := rects[y].Cli-
                                                          begin
entToScreen(point(0 + rects[y].width,0)).x;
                                                            rects[y].Brush.Color :=
 end;
                                                      rects[y].Brush.Color + $000100;
 while not done do
                                                            Application.ProcessMessages;
 begin
                                                          end;
  frm.BringToFront;
                                                         end;
  for Y := 0 to i - 2 do
                                                         Application.ProcessMessages;
  begin
                                                        end;
   if (mouse.CursorPos.X >= clickRe-
                                                        Application.ProcessMessages;
gions[y].Left) AND
                                                       end;
     (mouse.CursorPos.X <= clickRe-
                                                       i2 := 250:
                                                       while i2 > 2 do
gions[y].Right) AND
     (mouse.CursorPos.Y >= clickRe-
                                                       begin
gions[y].Top) AND
                                                        dec(i2, 2);
     (mouse.CursorPos.Y <= clickRe-
                                                        frm.AlphaBlendValue := i2;
gions[y].Bottom) then
                                                        Application.ProcessMessages;
   begin
                                                       end;
     while
                                                       frm.Destroy;
GETGVALUE(rects[y].Brush.Color) > $BE do
                                                      end;
     begin
                                                      end.
                                                      unit EngineClasses;
      rects[y].Brush.Color :=
rects[y].Brush.Color - $000100;
                                                      interface
      Application.ProcessMessages;
                                                      uses
                                                       ExtCtrls, Windows, Messages, SysUtils, Vari-
     end;
     if GetKeyState(VK_LBUTTON) < 0 then
                                                      ants, Classes, Graphics, Controls,
      Done := True;
                                                       Forms, Dialogs, Math, StdCtrls, pngimage,
      case sR[y][1] of
                                                      EngineUI;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
property ForwardClick: TForward read
type
 TLineAddTrigger = function(const s :
                                                    FForwardClick write SetForwardClick;
string):integer of object;
                                                     end:
 TDebug = class(TMemo)
                                                     TBoard = array[1..8] of array[1..8] of
  published
                                                    TSquare;
   constructor Create(AOwner:TCompo-
                                                     PDW = ^DWORD;
nent);override;
                                                     PSQR = ^TSquare;
  public
                                                     TIntArray = array of integer;
   println : TLineAddTrigger;
                                                     TBoardMannager = class
 end;
                                                     private
 TForward = procedure(ASquare : Pointer) of
                                                      FDebug: TDebug;
object;
                                                      Fhorse: TBitmap;
 TSquare = class(TImage)
                                                      Fpawn: TBitmap;
 private
                                                      Fknight: TBitmap;
  FCords: TPoint;
                                                      Fking: TBitmap;
  FDebug: TDebug;
                                                      Fqueen: TBitmap;
  FForwardClick: TForward;
                                                      Fbishop: TBitmap;
  FKind: integer;
                                                      FSelected: boolean;
  FColor: integer;
                                                      FSelectedSqr: TSquare;
  FPreKind: integer;
                                                      FTurn: integer;
  procedure SetCords(const Value: TPoint);
                                                      InCheck: boolean;
  procedure Click(Sender:TObject);
                                                      FWhitePiecesTook: array of integer;
  procedure SetForwardClick(const Value:
                                                      FBlackPiecesTook: array of integer;
TForward);
                                                      FOrientation: Integer;
  procedure SetKind(const Value: integer);
                                                      FAutoDeselect: boolean;
  procedure SetColor(const Value: integer);
                                                      FOutlineColor: TColor:
  procedure SetPreKind(const Value: integer);
                                                      FBlackColor: TColor;
                                                      FWhiteColor: TColor:
 published
  constructor Create(AOwner : TComponent);
                                                      FPlayerNameWhite: string;
                                                      FPlayerNameBlack: string;
override;
  property Cords: TPoint read FCords write
                                                      procedure SetDebug(const Value: TDebug);
                                                      procedure Setbishop(const Value: TBitmap);
SetCords;
  property PreKind: integer read FPreKind
                                                      procedure Sethorse(const Value: TBitmap);
write SetPreKind;
                                                      procedure Setking(const Value: TBitmap);
  property Kind: integer read FKind write
                                                      procedure Setknight(const Value: TBitmap);
SetKind:
                                                      procedure Setpawn(const Value: TBitmap);
  property Color: integer read FColor write
                                                      procedure Setqueen(const Value: TBitmap);
SetColor;
```

Изм.	Лист	№докцм.	Подпись	Дата

procedure SetSelected(const Value: boolprocedure DrawBoard; ean); property AutoDeselect: boolean read FAuto-Deselect write SetAutoDeselect: procedure SetSelectedSqr(const Value: TSquare); property pawn: TBitmap read Fpawn write procedure SetTurn(const Value: integer); Setpawn; function GetBlackPiecesTook(index:inteproperty king: TBitmap read Fking write ger):integer; Setking; function GetWhitePiecesTook(index:inteproperty castle: TBitmap read Fknight write ger):integer; Setknight; procedure SetBlackPiecesTook(Index: Inteproperty queen: TBitmap read Fqueen write ger; Value: Integer); Setqueen; procedure SetWhitePiecesTook(Index: Inteproperty bishop: TBitmap read Fbishop ger; Value: Integer); write Setbishop; function Move(ASquare: TSquare; Abm: property horse: TBitmap read Fhorse write TBitmap): integer; Sethorse: procedure TakePiece(ASquare: TSquare; property Selected: boolean read FSelected Abm : TBitmap); write SetSelected; procedure SetOrientation(const Value: Inteproperty SelectedSqr: TSquare read FSelectedSqr write SetSelectedSqr; ger); property Orientation: Integer read FOrientaprocedure SetAutoDeselect(const Value: boolean); tion write SetOrientation; procedure SetBlackColor(const Value: property Turn: integer read FTurn write TColor); SetTurn: procedure SetOutlineColor(const Value: property WhiteColor: TColor read FWhite-TColor); Color write SetWhiteColor; procedure SetWhiteColor(const Value: property BlackColor: TColor read FBlack-TColor); Color write SetBlackColor; property OutlineColor: TColor read FOutprocedure SetPlayerNameBlack(const Value: lineColor write SetOutlineColor: string); procedure SetPlayerNameWhite(const property PlayerNameWhite: string read Value: string); FPlayerNameWhite write SetPlayerNamepublished White; constructor create(AOwner : TForm); property PlayerNameBlack: string read destructor destroy; FPlayerNameBlack write SetPlayerNameproperty Debug: TDebug read FDebug write Black; SetDebug; function getLastSquareLeft : integer; procedure Click(ASquare : Pointer); function getSquareHeightWidth : integer; procedure InitialDraw; function getBlackTookLength: integer;

Изм.	Лист	№докцм.	Подпись	Дата

```
function getWhiteTookLength: integer;
                                                      gameWidth, gameHeight: integer;
  procedure Clear;
                                                     const
  procedure InvalidMove;
                                                      nl = #13#10;
  procedure SetSquareTo(Location : TPoint;
                                                      orRight_Left = 1;
kind: integer);
                                                      orTop\_Bottom = 2;
 public
                                                     implementation
                                                     { TSquare }
  Board: TBoard:
  CastlingPossible : array[1..2] of boolean;
                                                     procedure TSquare.Click(Sender: TObject);
  property WhitePiecesTook[Index:integer] :
                                                     begin
integer read GetWhitePiecesTook write
                                                      ForwardClick(Self);
SetWhitePiecesTook;
                                                     end;
  property BlackPiecesTook[Index:integer] :
                                                     constructor TSquare.Create(AOwner: TCom-
integer read GetBlackPiecesTook write Set-
                                                     ponent);
BlackPiecesTook:
                                                     begin
  function CheckDetect : byte; overload;
                                                      inherited;
  function CheckDetect(APoint: TPoint; multi-
                                                      stretch := True;
plier: integer): boolean; overload;
                                                      Height := floor(gameHeight/8);
 end;
                                                      Width := height;
 TSaveManager = class
                                                      if AOwner IS TForm then
                                                       parent := TForm(AOwner);
 private
  FrootDir: string;
                                                      kind := 0;
  FLinkedBoard: TBoardMannager;
                                                      color := 0;
                                                      OnClick := Click;
  procedure SetLinkedBoard(const Value:
TBoardMannager);
                                                     end;
  procedure SetrootDir(const Value: string);
                                                     procedure TSquare.SetColor(const Value: inte-
 published
                                                     ger);
  constructor Create(AOwner:TObject);
                                                     begin
  property LinkedBoard: TBoardMannager
                                                      FColor := Value;
read FLinkedBoard write SetLinkedBoard:
                                                     end:
  property rootDir: string read FrootDir write
                                                     procedure TSquare.SetCords(const Value:
SetrootDir:
                                                     TPoint);
  procedure SaveToFile(filepath : string);
                                                     begin
                                                      FCords := Value;
  procedure SaveToFileOverwrite(filepath:
string);
                                                     end:
  procedure LoadFromFile(filepath : string);
                                                     procedure TSquare.SetForwardClick(const
                                                     Value: TForward);
 end:
                                                     begin
var
 imageSize : integer = 32;
                                                      FForwardClick := Value;
```

ı					
	Изм.	Лист	№докцм.	Подпись	Дата

```
end;
                                                        pawn1, pawn2: TPoint;
procedure TSquare.SetKind(const Value: inte-
                                                        horseLocation: array[1..8] of TPoint;
                                                      begin
ger);
                                                        result := 0;
begin
                                                        for x := 1 to 8 do
 PreKind := FKind;
 FKind := Value;
                                                         for y := 1 to 8 do
end;
                                                          if (Board[x,y].Kind = -6) then
procedure TSquare.SetPreKind(const Value:
                                                           blackCords := Point(x, y)
integer);
                                                          else if Board[x,y].Kind = 6 then
begin
                                                            whiteCords := Point(x, y);
 FPreKind := Value;
                                                        for i := 1 to 2 do
end;
                                                        begin
                                                         if i = 1 then
{ TDebug }
constructor TDebug.Create(AOwner: TCompo-
                                                         begin
                                                          searchCords := whiteCords;
nent);
begin
                                                          multiplier := -1;
 inherited;
                                                         end
 if AOwner IS TForm then
                                                         else
  parent := TForm(AOwner);
                                                         begin
 lines.Clear;
                                                          searchCords := blackCords;
 width := floor(gamewidth/2) -
                                                          multiplier := 1;
floor((gamewidth/8) * 2.25);
                                                         end;
                                                         bExit := false:
 height := floor((gameheight/8)*2);
                                                         search := searchCords:
 top := gameHeight - height;
 println := lines.Add;
                                                         while (search.x + 1 < 9) AND (search.y - 1 >
 ReadOnly := true;
                                                      0) AND (NOT bExit) do
 Enabled := false;
                                                         begin
end:
                                                          inc(search.x);
{ TBoardMannager }
                                                          dec(search.y);
function TBoardMannager.CheckDetect: byte;
                                                          if (Board[search.x, search.y].Kind = 5 *
                                                      multiplier) or (Board[search.x, search.y].Kind
 blackCords, whiteCords, searchCords, search
                                                      = 3 * multiplier) then
: TPoint;
                                                           result := i
 x: Integer;
                                                          else if Board[search.x, search.y].Kind <> 0
 y: Integer;
                                                      then
                                                           bExit := true:
 i, i2: integer;
 multiplier: integer;
                                                         end:
 bExit: boolean;
                                                         bExit := false;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
search := searchCords;
                                                           if (Board[search.x, search.y].Kind = 5 *
  while (search.x + 1 < 9) AND (search.y + 1
                                                        multiplier) or (Board[search.x, search.y].Kind
< 9) AND (NOT bExit) do
                                                        = 3 * multiplier) then
                                                             result := i
  begin
   inc(search.x);
                                                           else if Board[search.x, search.y].Kind <> 0
   inc(search.y);
                                                        then
   if (Board[search.x, search.y].Kind = 5 *
                                                             bExit := true;
multiplier) or (Board[search.x, search.y].Kind
                                                          end;
= 3 * multiplier) then
                                                          bExit := false;
     result := i
                                                          search := searchCords;
   else if Board[search.x, search.y].Kind <> 0
                                                          while (search.X - 1 > 0) AND (NOT bExit)
then
                                                        do
     bExit := true;
                                                          begin
                                                           dec(search.x);
  end;
  bExit := false;
                                                           if (Board[search.x, search.y].Kind = 5 *
  search := searchCords;
                                                        multiplier) or (Board[search.x, search.y].Kind
                                                        = 2 * multiplier) then
  while (search.x - 1 > 0) AND (search.y - 1 > 0)
0) AND (NOT bExit) do
                                                             result := i
  begin
                                                           else if Board[search.x, search.y].Kind <> 0
   dec(search.x);
                                                        then
   dec(search.y);
                                                             bExit := true;
   if (Board[search.x, search.y].Kind = 5 *
                                                          end:
multiplier) or (Board[search.x, search.y].Kind
                                                          bExit := false;
= 3 * multiplier) then
                                                          search := searchCords:
     result := i
                                                          while (search.X + 1 < 9) AND (NOT bExit)
   else if Board[search.x, search.y].Kind <> 0
                                                        do
then
                                                          begin
                                                           inc(search.x);
     bExit := true;
                                                           if (Board[search.x, search.y].Kind = 5 *
  end:
  bExit := false:
                                                        multiplier) or (Board[search.x, search.y].Kind
  search := searchCords;
                                                        = 2 * multiplier) then
                                                             result := i
  while (search.x - 1 > 0) AND (search.y + 1
< 9) AND (NOT bExit) do
                                                           else if Board[search.x, search.y].Kind <> 0
  begin
                                                        then
   dec(search.x);
                                                             bExit := true;
   inc(search.y);
                                                          end:
                                                          bExit := false:
                                                          search := searchCords;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
while (search.Y - 1 > 0) AND (NOT bExit)
                                                           pawn1 := Point(search.X - 1, search.Y +
do
                                                       multiplier);
  begin
                                                           pawn2 := pawn1;
   dec(search.Y);
   if (Board[search.x, search.y].Kind = 5 *
                                                         else if NOT (search.X > 1) then
multiplier) or (Board[search.x, search.y].Kind
                                                         begin
= 2 * multiplier) then
                                                           pawn2 := Point(search.X + 1, search.Y +
    result := i
                                                       multiplier);
   else if Board[search.x, search.y].Kind <> 0
                                                           pawn1 := Pawn2;
then
                                                         end;
    bExit := true;
                                                         if (board[Pawn1.X, Pawn1.Y].Kind = 1 *
                                                       multiplier) OR
  end;
                                                           (board[Pawn2.X, Pawn2.Y].Kind = 1 *
  bExit := false;
                                                       multiplier) then
  search := searchCords;
  while (search.Y + 1 < 9) AND (NOT bExit)
                                                           result := i;
                                                         horseLocation[1] := point(search.X - 1,
do
                                                       search.Y - 2);
  begin
                                                         horseLocation[2] := point(search.X + 1,
   inc(search.Y);
   if (Board[search.x, search.y].Kind = 5 *
                                                       search.Y - 2);
multiplier) or (Board[search.x, search.y].Kind
                                                         horseLocation[3] := point(search.X - 2,
= 2 * multiplier) then
                                                       search.Y - 1);
    result := i
                                                         horseLocation[4] := point(search.X + 2,
   else if Board[search.x, search.y].Kind <> 0
                                                       search.Y - 1);
then
                                                         horseLocation[5] := point(search.X - 2,
    bExit := true;
                                                       search.Y + 1;
  end:
                                                         horseLocation[6] := point(search.X + 2,
  bExit := false:
                                                       search.Y + 1;
  search := searchCords;
                                                         horseLocation[7] := point(search.X - 1,
  if (search.X > 1) AND (search.X < 8) then
                                                       search.Y + 2;
  begin
                                                         horseLocation[8] := point(search.X + 1,
   pawn1 := Point(search.X - 1, search.Y +
                                                       search.Y + 2:
                                                         for i2 := 1 to 8 do
multiplier);
   pawn2 := Point(search.X + 1, search.Y +
                                                         begin
multiplier);
                                                           if (horseLocation[i2].X IN [1..8]) AND
                                                       (horseLocation[i2].Y IN [1..8]) then
  end
  else if NOT (search.X < 8) then
                                                            if board[horselocation[i2].x,horseloca-
                                                       tion[i2].Y].kind = 4 * multiplier then
  begin
                                                             result := i;
```

	·			·
Изм.	Лист	№докцм.	Подпись	Дата

```
end;
                                                         while (search.x + 1 < 9) AND (search.y + 1 <
 end;
                                                        9) AND (NOT bExit) do
end;
                                                         begin
function TBoardMannager.Check-
                                                          inc(search.x);
Detect(APoint: TPoint; multiplier : integer):
                                                          inc(search.y);
boolean;
                                                          if (Board[search.x, search.y].Kind = 5 * mul-
var
                                                       tiplier) or
                                                           (Board[search.x, search.y].Kind = 3 * mul-
 searchCords, search: TPoint;
                                                       tiplier) then
 x: Integer;
 y: Integer;
                                                           result := i
                                                          else if Board[search.x, search.y].Kind <> 0
 i2: integer;
 bExit: boolean;
                                                       then
 pawn1, pawn2: TPoint;
                                                           bExit := True;
 horseLocation : array[1..8] of TPoint;
                                                         end;
                                                         bExit := false;
const
 i = true;
                                                         search := searchCords;
begin
                                                         while (search.x - 1 > 0) AND (search.y - 1 > 0)
 result := false;
                                                        0) AND (NOT bExit) do
 searchCords := APoint;
                                                         begin
 bExit := false;
                                                          dec(search.x);
 search := searchCords;
                                                          dec(search.y);
 while (search.x + 1 < 9) AND (search.y - 1 >
                                                          if (Board[search.x, search.y].Kind = 5 * mul-
0) AND (NOT bExit) do
                                                       tiplier) or
 begin
                                                           (Board[search.x, search.y].Kind = 3 * mul-
  inc(search.x);
                                                       tiplier) then
                                                           result := i
  dec(search.y);
  if (Board[search.x, search.y].Kind = 5 * mul-
                                                          else if Board[search.x, search.y].Kind <> 0
tiplier) or
                                                       then
   (Board[search.x, search.y].Kind = 3 * mul-
                                                           bExit := True:
tiplier) then
                                                         end;
   result := i
                                                         bExit := false:
                                                         search := searchCords;
  else if Board[search.x, search.y].Kind <> 0
then
                                                         while (search.x - 1 > 0) AND (search.y + 1 <
   bExit := True:
                                                        9) AND (NOT bExit) do
 end;
                                                         begin
 bExit := false:
                                                          dec(search.x);
 search := searchCords;
                                                          inc(search.y);
```

Изм.	Лист	№докцм.	Подпись	Дата

```
if (Board[search.x, search.y].Kind = 5 * mul-
                                                         search := searchCords;
tiplier) or
                                                         while (search.y - 1 > 0) AND (NOT bExit) do
   (Board[search.x, search.y].Kind = 3 * mul-
                                                         begin
tiplier) then
                                                          dec(search.y);
   result := i
                                                          if (Board[search.x, search.y].Kind = 5 * mul-
  else if Board[search.x, search.y].Kind <> 0
                                                        tiplier) or
then
                                                            (Board[search.x, search.y].Kind = 2 * mul-
   bExit := True;
                                                        tiplier) then
 end;
                                                            result := i
 bExit := false;
                                                          else if Board[search.x, search.y].Kind <> 0
 search := searchCords;
                                                        then
 while (search.x - 1 > 0) AND (NOT bExit) do
                                                            bExit := True;
 begin
                                                         end;
                                                         bExit := false;
  dec(search.x);
  if (Board[search.x, search.y].Kind = 5 * mul-
                                                         search := searchCords;
                                                         while (search.y + 1 < 9) AND (NOT bExit) do
tiplier) or
   (Board[search.x, search.y].Kind = 2 * mul-
                                                         begin
tiplier) then
                                                          inc(search.y);
   result := i
                                                          if (Board[search.x, search.y].Kind = 5 * mul-
  else if Board[search.x, search.y].Kind <> 0
                                                        tiplier) or
                                                            (Board[search.x, search.y].Kind = 2 * mul-
then
   bExit := True;
                                                        tiplier) then
                                                            result := i
 end:
 bExit := false;
                                                          else if Board[search.x, search.y].Kind <> 0
 search := searchCords:
                                                        then
 while (search.x + 1 < 9) AND (NOT bExit) do
                                                            bExit := True:
 begin
                                                         end:
  inc(search.x);
                                                         bExit := false:
  if (Board[search.x, search.y].Kind = 5 * mul-
                                                         search := searchCords:
                                                         if (search.x > 1) AND (search.x < 8) then
tiplier) or
   (Board[search.x, search.y].Kind = 2 * mul-
tiplier) then
                                                          pawn1 := Point(search.x - 1, search.y + mul-
   result := i
                                                        tiplier);
  else if Board[search.x, search.y].Kind <> 0
                                                           pawn2 := Point(search.x + 1, search.y + mul-
then
                                                        tiplier);
   bExit := True;
                                                         end
 end:
                                                         else if NOT(search.x < 8) then
 bExit := false;
                                                         begin
```

Изм.	Лист	№докцм.	Подпись	Дата

```
pawn1 := Point(search.x - 1, search.y + mul-
                                                           horseLocation[i2].y].Kind = 4 * multiplier
tiplier);
                                                       then
  pawn2 := pawn1;
                                                           result := i;
                                                        end;
 end
 else if NOT(search.x > 1) then
                                                      end;
                                                      procedure TBoardMannager.Clear;
 begin
  pawn2 := Point(search.x + 1, search.y + mul-
tiplier);
                                                        y, x, i: Integer;
  pawn1 := pawn2;
                                                        t1, t2: integer;
                                                      begin
 if (Board[pawn1.x, pawn1.y].Kind = 1 * mul-
                                                        t1 := GetTickCount;
tiplier) OR
                                                        selected := false;
  (Board[pawn2.x, pawn2.y].Kind = 1 * multi-
                                                        for I := 0 to getBlackTooklength do
plier) then
                                                         FBlackPiecesTook[i] := 0;
  result := i;
                                                        for I := 0 to getWhiteTookLength do
 horseLocation[1] := Point(search.x - 1,
                                                         FWhitePiecesTook[i] := 0;
search.y - 2);
                                                        SetLength(fwhitePiecesTook, 1);
 horseLocation[2] := Point(search.x + 1,
                                                        SetLength(fblackPiecesTook, 1);
search.y - 2);
                                                        turn := 1;
 horseLocation[3] := Point(search.x - 2,
                                                        for y := 1 to 8 do
search.y - 1);
                                                         for x := 1 to 8 do
 horseLocation[4] := Point(search.x + 2,
                                                          with Board[x, y] do
                                                           Kind := 0;
search.y - 1);
 horseLocation[5] := Point(search.x - 2,
                                                        t2 := GetTickCount:
                                                        Debug.lines.Clear;end;
search.y + 1;
 horseLocation[6] := Point(search.x + 2,
                                                      procedure TBoardMannager.Click(ASquare:
search.y + 1;
                                                       Pointer);
 horseLocation[7] := Point(search.x - 1,
                                                       var
search.y + 2;
                                                        Square: TSquare;
 horseLocation[8] := Point(search.x + 1,
                                                        sDebugMSG: string;
search.y + 2;
                                                        difInY, difInX:integer;
 for i2 := 1 to 8 do
                                                        difPawnForward, difPawnSide: integer;
 begin
                                                        bm: TBitmap;
  if (horseLocation[i2].x IN [1..8]) AND
                                                        x,y:integer;
(horseLocation[i2].y IN [1..8])
                                                        pbase, p: PDW;
                                                        xMultiplier, yMin, yMax, newKind: integer;
   then
   if Board[horseLocation[i2].x,
                                                        I: Integer;
```

Изм.	Лист	№докум.	Подпись	Дата

```
startcheckX, endcheckx, startchecky, end-
                                                         begin
checky: integer;
                                                          xMultiplier := 1;
 possibleCastling: boolean;
                                                          yMax := -1;
 presquareKind: integer;
                                                          yMin := -6;
begin
                                                          newKind := Selectedsqr.Kind;
 InCheck := false;
                                                         end;
 bm := TBitmap.Create;
                                                        end;
 with bm do
                                                        case SelectedSqr.Kind of
 begin
                                                         1, -1:
  PixelFormat := pf32bit;
                                                          begin
  height := 1;
                                                            case Orientation of
  width := 1;
                                                             orRight_Left:
 end;
                                                              begin
                                                               difPawnForward := difInX;
 Square := TSquare(ASquare);
 sDebugMSG := 'Clicked On: X:' + IntTo-
                                                               difPawnSide := difInY;
Str(Square.Cords.X) + 'Y:' + IntTo-
                                                              end;
Str(Square.Cords.Y);
                                                             orTop_Bottom:
 else
                                                              begin
                                                               difPawnForward := -difInY;
 begin
  difInY := Square.Cords.Y - Selected-
                                                               difPawnSide := -difInX:
Sqr.Cords.Y;
                                                              end;
  difInX := Square.Cords.X - Selected-
                                                            end:
                                                            if difPawnForward = (1 * xMultiplier)
Sqr.Cords.X;
  case SelectedSqr.Color of
                                                      then
   1:
                                                            begin
    bm.Canvas.Pixels[0, 0] := \$0;
                                                             if (Square.Kind <> 0) AND (difPawn-
                                                      Side = 0) then
     bm.Canvas.Pixels[0, 0] := $FFFFFF;
                                                             begin
                                                              invalidmove:
  end:
  case selectedsqr.kind of
                                                              Exit;
   -6..-1:
                                                             end:
                                                             if (difPawnSide = 0) then
   begin
     xMultiplier := -1;
                                                             begin
    yMax := 6;
                                                              move(square, bm);
    yMin := 1;
                                                              exit;
     newKind := Selectedsqr.Kind;
                                                             end:
                                                             if difPawnSide <> 0 then
   end;
   1..6:
```

Изм.	Лист	№доким.	Подпись	Дата

```
if (difPawnSide = 1) or (difPawnSide
                                                            begin
                                                             invalidmove;
= -1) then
        begin
                                                             Exit;
         if (Square.Kind >= yMin) AND
                                                            end;
(Square.Kind <= yMax) then
                                                           end;
                                                          2, -2:
         begin
           takepiece(square, bm);
                                                           begin
           if not ((Square.Kind >= yMin) AND
                                                            if (difInY <> 0) and (difInX <> 0) then
(Square.Kind <= yMax)) then
                                                            begin
                                                             invalidmove;
           exit;
          end
                                                             Exit;
                                                            end
          else
                                                            else if difInY <> 0 then
         begin
           invalidmove;
                                                            begin
           Exit;
                                                             if difInY < 0 then
         end;
                                                              for I := Square.Cords.y + 1 to Selected-
                                                      Sqr.Cords.y - 1 do
        end
        else
                                                             begin
                                                               if Board[Square.Cords.x, I].Kind <> 0
        begin
         invalidmove;
                                                      then
         Exit;
                                                               begin
                                                                invalidmove;
        end;
                                                                Exit:
      end
      else if (difPawnForward = (2*xMulti-
                                                               end;
plier)) AND
                                                              end
           (board[square.cords.x,
                                                              else if difInY > 0 then
square.cords.y + (1 * xMultiplier)].Kind = 0)
                                                             for I := SelectedSqr.Cords.y + 1 to
AND
                                                      Square.Cords.y - 1 do
           ((selected sqr. Cords. Y = 7) or (select-
                                                             begin
edsqr.Cords.Y = 2)) then
                                                               if Board[Square.Cords.x, I].Kind <> 0
      begin
                                                      then
       if (difPawnSide = 0) then
                                                               begin
                                                                invalidmove;
       begin
        move(square, bm);
                                                                Exit:
                                                               end;
        exit;
       end;
                                                              end;
                                                             if (Square.Kind >= yMin) AND
      end
      else
                                                      (Square.Kind <= yMax) then
```

Изм.	Лист	№докцм.	Подпись	Дата

```
begin
                                                               begin
                                                                 invalidmove;
        takepiece(square, bm);
        if not ((Square.Kind >= yMin) AND
                                                                Exit;
(Square.Kind <= yMax)) then
                                                               end;
        castlingPossible[turn] := false;
                                                              end;
        exit;
                                                              if (Square.Kind >= yMin) AND
                                                       (Square.Kind <= yMax) then
       end
       else if Square.Kind = 0 then
                                                              begin
                                                               takepiece(square,bm);
       begin
        move(square, bm);
                                                               if not ((Square.Kind >= yMin) AND
        castlingPossible[turn] := false;
                                                       (Square.Kind <= yMax)) then
        exit;
                                                               castlingPossible[turn] := false;
       end
                                                               exit;
       else
                                                              end
                                                              else if Square.Kind = 0 then
       begin
        invalidmove;
                                                              begin
        Exit;
                                                               move(square, bm);
       end;
                                                               castlingPossible[turn] := false;
                                                               exit;
      end
      else if difInX <> 0 then
                                                              end
      begin
                                                              else
       if difInX < 0 then
                                                              begin
       for I := Square.Cords.X + 1 to Selected
                                                               invalidmove;
Sqr.Cords.X - 1 do
                                                               Exit:
       begin
                                                              end;
        if Board[I, square.Cords.y].Kind <> 0
                                                             end:
then
                                                            end;
                                                          3, -3:
        begin
          invalidmove;
                                                            begin
                                                             if difInX > 0 then
         Exit;
        end;
       end
                                                              if (difInY = difInX) or (difInY = -
       else if difInX > 0 then
                                                       difInX) then
       for I := SelectedSqr.Cords.X + 1 to
                                                              begin
Square.Cords.X - 1 do
                                                               i := 0;
       begin
                                                               for x := SelectedSqr.Cords.x + 1 to
        if Board[I, square.Cords.y].Kind <> 0
                                                       Square.Cords.x - 1 do
then
                                                               begin
```

Изм.	Лист	№докцм.	Подпись	Дата

```
inc(i);
                                                               if (difInY = difInX) or (difInY = -
          if difInY = difInX then
                                                       difInX) then
           y := SelectedSqr.Cords.y + i
                                                              begin
          else
                                                                i := 0;
           y := SelectedSqr.Cords.y - i;
                                                                for x := SelectedSqr.Cords.x - 1
          if Board[x, y]. Kind \Leftrightarrow 0 then
                                                       downto Square.Cords.x + 1 do
          begin
                                                                begin
           invalidmove;
                                                                 dec(i);
           Exit;
                                                                 if difInY = difInX then
          end;
                                                                  y := SelectedSqr.Cords.y + i
        end;
                                                                 else
        if Square.Kind = 0 then
                                                                  y := SelectedSqr.Cords.y - i;
                                                                 if Board[x, y].Kind <> 0 then
        begin
          move(square, bm);
                                                                 begin
          exit;
                                                                  invalidmove;
        end;
                                                                  Exit:
        if (Square.Kind >= yMin) AND
                                                                 end;
(Square.Kind <= yMax) then
                                                                end;
                                                                if Square.Kind = 0 then
        begin
          takepiece(square, bm);
                                                                begin
          if not ((Square.Kind >= yMin) AND
                                                                 move(square, bm);
(Square.Kind <= yMax)) then
                                                                 exit;
          exit;
                                                                end:
                                                                if (Square.Kind >= yMin) AND
         end
                                                       (Square.Kind <= yMax) then
         else
        begin
                                                                begin
         invalidmove;
                                                                 takepiece(square, bm);
                                                                 if not ((Square.Kind >= yMin) AND
         Exit;
                                                       (Square.Kind <= yMax)) then
        end:
       end
                                                                 exit;
       else
                                                                end
       begin
                                                                else
        invalidmove;
                                                                begin
                                                                 invalidmove;
        Exit:
       end;
                                                                 Exit;
      end
                                                                end:
      else if difInX < 0 then
                                                               end
      begin
                                                               else
```

Изм.	Лист	№докцм.	Подпись	Дата

```
begin
                                                               begin
         invalidmove;
                                                                 invalidmove;
         Exit;
                                                                Exit;
                                                               end;
       end;
      end
                                                              end;
                                                             5, -5:
      else
      begin
                                                              begin
       invalidmove;
                                                               if (difInY = difInX) or (difInY = -difInX)
       Exit;
                                                         then
      end;
                                                               begin
     end;
                                                                 if difInX > 0 then
    4, -4:
                                                                 begin
     begin
                                                                  i := 0;
      if (((difInX = 2) \text{ or } (difInX = -2)) \text{ and }
                                                                  for x := SelectedSqr.Cords.x + 1 to
((difInY = 1) or (difInY = -1)))
                                                         Square.Cords.x - 1 do
      or (((difInY = 2) \text{ or } (difInY = -2)) \text{ and }
                                                                  begin
((difInX = 1) \text{ or } (difInX = -1))) \text{ then }
                                                                   inc(i);
      begin
                                                                   if difInY = difInX then
       if Square.Kind = 0 then
                                                                     y := SelectedSqr.Cords.y + i
       begin
                                                                   else
         move(square, bm);
                                                                    y := SelectedSqr.Cords.y - i;
                                                                   if Board[x, y].Kind <> 0 then
         exit;
                                                                   begin
       end
       else if (Square.Kind >= yMin) AND
                                                                    invalidmove;
(Square.Kind <= yMax) then
                                                                    Exit;
       begin
                                                                   end:
         takepiece(square,bm);
                                                                  end;
         if not ((Square.Kind >= yMin) AND
                                                                  if Square.Kind = 0 then
(Square.Kind <= yMax)) then
                                                                  begin
         exit;
                                                                   Move(Square, bm);
       end
                                                                   Exit:
       else
                                                                  end;
                                                                  if (Square.Kind >= yMin) AND
       begin
         invalidmove;
                                                         (Square.Kind <= yMax) then
         Exit;
                                                                  begin
       end;
                                                                   TakePiece(Square, bm);
                                                                   if not ((Square.Kind >= yMin) AND
      end
      else
                                                         (Square.Kind <= yMax)) then
```

Изм.	Лист	№докцм.	Подпись	Дата

```
Exit;
                                                                else
         end
                                                                begin
         else
                                                                 invalidmove;
                                                                 Exit;
        begin
         invalidmove;
                                                                end;
         Exit;
                                                               end
        end;
                                                               else
       end
                                                              begin
       else if difInX < 0 then
                                                               invalidmove;
       begin
                                                               Exit:
        i := 0;
                                                              end;
        for x := SelectedSqr.Cords.x - 1
                                                             end
downto Square.Cords.x + 1 do
                                                             else if (difInY \ll 0) and (difinx = 0) then
        begin
                                                             begin
          dec(i);
                                                              if difInY < 0 then
          if difInY = difInX then
                                                               for I := Square.Cords.y + 1 to Selected-
           y := SelectedSqr.Cords.y + i
                                                       Sqr.Cords.y - 1 do
          else
                                                              begin
                                                                if Board[Square.Cords.x, I].Kind <> 0
           y := SelectedSqr.Cords.y - i;
          if Board[x, y].Kind \Leftrightarrow 0 then
                                                       then
          begin
                                                                begin
           invalidmove;
                                                                 invalidmove;
           Exit:
                                                                 Exit:
          end;
                                                                end:
        end;
                                                               end
        if Square.Kind = 0 then
                                                               else if difInY > 0 then
        begin
                                                              for I := SelectedSqr.Cords.y + 1 to
                                                       Square.Cords.y - 1 do
          Move(Square, bm);
         Exit:
                                                              begin
                                                                if Board[Square.Cords.x, I].Kind <> 0
         end;
        if (Square.Kind >= yMin) AND
                                                       then
(Square.Kind <= yMax) then
                                                                begin
                                                                 invalidmove;
        begin
                                                                 Exit:
          TakePiece(Square, bm);
          if not ((Square.Kind >= yMin) AND
                                                                end;
(Square.Kind <= yMax)) then
         Exit:
                                                              if (Square.Kind >= yMin) AND
         end
                                                       (Square.Kind <= yMax) then
```

Изм.	Лист	№докцм.	Подпись	Дата

```
begin
                                                                Exit;
                                                               end;
        takepiece(square,bm);
        if not ((Square.Kind >= yMin) AND
                                                             end;
(Square.Kind <= yMax)) then
                                                             if (Square.Kind >= yMin) AND
                                                      (Square.Kind <= yMax) then
          exit;
       end
                                                             begin
       else if Square.Kind = 0 then
                                                               takepiece(square,bm);
                                                               if not ((Square.Kind >= yMin) AND
       begin
        move(square, bm);
                                                      (Square.Kind \le yMax)) then
        exit;
                                                               exit;
       end
                                                             end
                                                             else if Square.Kind = 0 then
       else
       begin
                                                             begin
        invalidmove;
                                                               move(square, bm);
        Exit;
                                                              exit;
       end;
                                                             end
      end
                                                             else
      else if (difInX <> 0) and (difiny = 0) then
                                                             begin
                                                              invalidmove;
      begin
       if difInX < 0 then
                                                              Exit;
       for I := Square.Cords.X + 1 to Selected
                                                             end;
Sqr.Cords.X - 1 do
                                                            end
       begin
                                                            else
        if Board[I, square.Cords.y].Kind <> 0
                                                            begin
                                                             invalidmove;
then
        begin
                                                             Exit:
         invalidmove;
                                                            end;
         Exit:
                                                           end;
        end:
                                                          6, -6:
       end
                                                           begin
       else if difInX > 0 then
                                                            if ((difInX < 2) AND (difInX > -2))
       for I := SelectedSqr.Cords.X + 1 to
                                                      AND ((difInY < 2) AND (difInY > -2)) then
Square.Cords.X - 1 do
                                                            begin
                                                             startcheck X := square.Cords. X - 1;
       begin
        if Board[I, square.Cords.y].Kind <> 0
                                                             if startcheck X < 1 then
                                                               startcheckx := 1;
then
        begin
                                                             endcheckX := square.Cords.X + 1;
         invalidmove;
                                                             if endcheckX > 8 then
```

ı					
I					
ĺ	Изм.	Лист	№докцм.	Подпись	Дата

```
endcheckx := 8;
                                                             else if ((((difInX > 1) or (difinx < -1))
       startcheckY := square.Cords.Y - 1;
                                                       and (Orientation = orTop_Bottom)) and
       if startcheck Y < 1 then
                                                       ((square.Kind = 0) AND (castlingpossi-
        startcheckY := 1;
                                                       ble[turn]))) then
       endcheckY := square.Cords.Y + 1;
                                                             begin
       if endcheckY > 8 then
                                                               possibleCastling := false;
        endcheckY := 8;
                                                               if ((board[8, square.Cords.y].kind = 2 *
       for y := startcheckY to endcheckY do
                                                       xmultiplier)) then
        for x := startcheckX to endcheckX do
                                                                 possibleCastling := true;
        begin
                                                              if square.Cords.X = 7 then
         if board[x,y].Kind = selectedsqr.kind
                                                                if ((board[8, square.Cords.y].kind = 2)
                                                       * xmultiplier)) then
* -1 then
         begin
                                                                 possibleCastling := true
                                                               else if square.Cords.X = 2 then
           beep;
           Exit;
                                                                if (board[1, square.Cords.y].Kind = 2
                                                       * xmultiplier) then
          end;
                                                                 possibleCastling := true;
        end:
       if square.Kind = 0 then
                                                              if possibleCastling then
                                                                if SelectedSqr.Cords.x <
       begin
        Move(square, bm);
                                                       Square.Cords.x then
        castlingPossible[turn] := false;
                                                                begin
        exit;
                                                                 for i := SelectedSqr.Cords.x + 1 to
                                                       Square.Cords.x - 1 do
       end
                                                                  if Board[i, Square.Cords.y].Kind <>
       else if (Square.Kind >= yMin) AND
(Square.Kind <= yMax) then
                                                       0 then
       begin
                                                                  begin
        takepiece(square,bm);
                                                                   possibleCastling := false;
        if not ((Square.Kind >= yMin) AND
                                                                  end:
(Square.Kind <= yMax)) then
                                                                end
        castlingPossible[turn] := false;
                                                                else
        exit:
                                                                 for i := SelectedSqr.Cords.x - 1
       end
                                                       downto Square.Cords.x + 1 do
                                                                  if Board[i, Square.Cords.y].Kind <>
       else
                                                       0 then
       begin
        invalidmove;
                                                                  begin
        Exit:
                                                                    possibleCastling := false;
       end:
                                                                  end;
      end
                                                              if possibleCastling then
```

h	1зм.	Лист	№докцм.	Подпись	Дата

```
begin
                                                                 board[1, square.Cords.y].Pic-
                                                      ture.Bitmap := bm;
        if SelectedSqr.Cords.x <
Square.Cords.x then
                                                                 SetSquareTo(point(square.Cords.X
                                                      + 2, square.Cords.Y), 2 * xMultiplier);
         if CheckDetect(Point(Selected-
Sqr.Cords.x + 2, Square.Cords.y),
                                                                end;
           xMultiplier * -1) = false then
                                                               end;
         begin
                                                              end;
           CastlingPossible[turn] := false;
                                                            end
           move(square, bm);
                                                            else
           setsquareto(point(8,
                                                            begin
square.Cords.y), 0);
                                                             invalidmove;
           if board[8, square.Cords.y].Color =
                                                             Exit;
1 then
                                                            end;
            bm.Canvas.Pixels[0,0] := $0
                                                           end;
           else
                                                         end;
            bm.Canvas.Pixels[0,0] := $ffffff;
                                                       end;
           board[8, square.Cords.y].Pic-
                                                       freeAndNil(bm);
ture.Bitmap := bm;
                                                      end;
           SetSquareTo(point(square.Cords.X
                                                      constructor TBoardMannager.create(AOwner:
- 1, square.Cords.Y), 2 * xMultiplier);
                                                      TForm);
         end;
                                                      var
        if SelectedSqr.Cords.x >
                                                       y, x, firstX: Integer;
Square.Cords.x then
                                                       bm: TBitmap;
        begin
                                                       t1, t2 : integer;
         if CheckDetect(Point(Selected-
                                                      begin
Sqr.Cords.x - 2, Square.Cords.y),
                                                       BlackColor := $1F2635:
           xMultiplier * -1) = false then
                                                        WhiteColor := $BED5FF;
         begin
                                                       OutlineColor := $505050;
           CastlingPossible[turn] := false;
                                                       AutoDeselect := true:
           move(board[SelectedSqr.Cords.x -
                                                       incheck := false;
2, Square.Cords.y], bm);
                                                        selected := false:
                                                       firstX := floor(gameWidth/2) -
           setsquareto(point(1,
                                                      floor((gameWidth/8) * 2.25);
square.Cords.y), 0);
           if board[1, square.Cords.y].Color =
                                                       SetLength(fwhitePiecesTook, 1);
1 then
                                                       SetLength(fblackPiecesTook, 1);
            bm.Canvas.Pixels[0,0] := $0
                                                       turn := 1;
                                                       Debug := TDebug.Create(AOwner);
           else
            bm.Canvas.Pixels[0,0] := $ffffff;
                                                       debug.Visible:=false;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
for y := 1 to 8 do
                                                        width := height;
 for x := 1 to 8 do
                                                       end;
 begin
                                                       king := TBitmap.Create;
  Board[x, y] := tsquare.Create(AOwner);
                                                       with king do
  with board[x, y] do
                                                       begin
  begin
                                                        PixelFormat := pf32bit;
   top := (y - 1) * Height;
                                                        height := imagesize;
   left := (x - 1) * Height + firstX;
                                                        width := height;
   Cords := Point(x, y);
                                                       end;
   ForwardClick := self.Click;
                                                       queen := TBitmap.Create;
  end:
                                                       with queen do
                                                       begin
 end;
pawn := TBitmap.Create;
                                                        PixelFormat := pf32bit;
                                                        height := imagesize;
with pawn do
                                                        width := height;
begin
 PixelFormat := pf32bit;
                                                       end;
                                                       Orientation := orRight_Left;
 height := imagesize;
 width := height;
                                                      end;
end;
                                                      destructor TBoardMannager.destroy;
bishop := TBitmap.Create;
                                                      var
with bishop do
                                                       x, y: integer;
begin
                                                     begin
                                                       for y := 1 to 8 do
 PixelFormat := pf32bit;
 height := imagesize;
                                                        for x := 1 to 8 do
 width := height;
                                                         freeandnil(Board[x,y]);
end:
                                                       debug.Destroy;
castle := TBitmap.Create;
                                                       pawn.Destroy;
with castle do
                                                       bishop.Destroy;
begin
                                                       castle.Destroy;
 PixelFormat := pf32bit;
                                                       horse.Destroy;
 height := imagesize;
                                                       king.Destroy;
 width := height;
                                                       queen.Destroy;
end;
                                                      end;
horse := TBitmap.Create;
                                                     procedure TBoardMannager.DrawBoard;
with horse do
                                                      var
begin
                                                      bm : TBitmap;
 PixelFormat := pf32bit;
                                                      x, y: integer;
 height := imagesize;
                                                      t1, t2: integer;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
begin
                                                       result := length(FBlackPiecesTook) - 1;
 t1 := GetTickCount;
                                                      end:
 bm := TBitmap.Create;
                                                      function TBoardMannager.getLastSquareLeft:
 with bm do
                                                      integer;
 begin
                                                      begin
  PixelFormat := pf32bit;
                                                       result := board[8,1].Left
  height := 1;
                                                      end:
  width := 1;
                                                      function TBoardMannager.getSquare-
 end;
                                                      HeightWidth: integer;
 for y := 1 to 8 do
                                                      begin
  for x := 1 to 8 do
                                                       result := board[1,1].Height;
  begin
                                                      end;
   with board[x, y] do
                                                      function TBoardMannager.GetWhite-
   begin
                                                      PiecesTook(index: integer): integer;
   if odd(x + y - orientation) then
                                                      begin
                                                       result := FWhitePiecesTook[index];
     begin
      bm.Canvas.Pixels[0,0] := \$000000;
                                                      end;
      color := 1;
                                                      function TBoardMannager.getWhiteTook-
     end
                                                      Length: integer;
     else
                                                      begin
                                                       result := length(FWhitePiecesTook) - 1;
     begin
      bm.Canvas.Pixels[0,0] := $ffffff;
      color := 2;
                                                      procedure TBoardMannager.InitialDraw;
     end;
                                                      var
     picture.Bitmap := bm
                                                       pbase, p: PDW;
   end:
                                                       y, y1, x, x1, i : integer;
  end;
                                                       tempbm: TBitmap;
 bm.Destroy;
                                                       t1, t2: integer;
 t2 := GetTickCount:
                                                      begin
                                                       drawboard;
end;
function TBoardMannager.GetBlack-
                                                       castlingPossible[1] := true;
                                                       CastlingPossible[2] := true;
PiecesTook(index: integer): integer;
begin
                                                       t1 := GetTickCount;
 result := FBlackPiecesTook[index];
                                                       if Orientation = orTop_Bottom then
end;
                                                       begin
                                                         for x := 1 to 8 do
function TBoardMannager.getBlackTook-
Length: integer;
                                                        begin
begin
                                                          SetSquareTo(Point(x, 7), 1);
```

	·			·
Изм.	Лист	№докцм.	Подпись	Дата

```
SetSquareTo(Point(x, 2), -1);
                                                     end;
 end;
                                                     procedure TBoardMannager.InvalidMove;
 for x := 1 to 2 do
                                                     begin
 begin
                                                      beep;
  SetSquareTo(Point( x * 7 - 6, 8), 2);
                                                      if autoDeselect then
  SetSquareTo(Point(x * 7 - 6, 1), -2);
                                                      begin
  SetSquareTo(Point( x * 3,8), 3);
                                                       selected := false;
                                                       if turn = 1 then
  SetSquareTo(Point(x * 3,1), -3);
  SetSquareTo(Point( x * 5 - 3, 8), 4);
                                                         turn := 2
  SetSquareTo(Point( x * 5 - 3, 1), -4);
                                                       else
                                                         turn := 1;
 end;
 SetSquareTo(Point(5, 8), 6);
                                                      end;
 SetSquareTo(Point(5, 1), -6);
                                                     end;
 SetSquareTo(Point(4, 8), 5);
                                                     function TBoardMannager.Move(ASquare:
 SetSquareTo(Point(4, 1), -5);
                                                     TSquare; Abm: TBitmap): integer;
end
                                                     var
else
                                                      Atempbm: Tbitmap;
                                                      pbase, p: PDW;
Begin
 for y := 1 to 8 do
                                                      y,x: integer;
 begin
                                                      CheckTurn: byte;
  SetSquareTo(Point(2, y), 1);
                                                      reverseSelected, reverseSquare: TSquare;
  SetSquareTo(Point(7, y), -1);
                                                      squarebm: TBitmap;
 end:
                                                     begin
 for y := 1 to 2 do
                                                      result := ASquare.kind;
                                                      reverseSelected := ASquare;
 begin
  SetSquareTo(Point(1, y * 7 - 6), 2);
                                                      reverseSquare := SelectedSqr;
  SetSquareTo(Point(8, y * 7 - 6), -2);
                                                      squarebm := TBitmap.Create;
  SetSquareTo(Point(1, y * 3), 3);
                                                      with squarebm do
  SetSquareTo(Point(8, y * 3), -3);
                                                      begin
  SetSquareTo(Point(1, y * 5 - 3), 4);
                                                       PixelFormat := pf32bit;
  SetSquareTo(Point(8, y * 5 - 3), -4);
                                                       Height := imageSize;
                                                       Width := Height;
 end;
 SetSquareTo(Point(1, 5), 6);
                                                      end;
 SetSquareTo(Point(8, 5), -6);
                                                      squarebm.Assign(ASquare.Picture.Bitmap);
                                                      if Turn = 1 then
 SetSquareTo(Point(1, 4), 5);
 SetSquareTo(Point(8, 4), -5);
                                                        CheckTurn := 2
End:
                                                      else
t2 := GetTickCount;
                                                       CheckTurn := 1;
```

Изм.	Лист	№докцм.	Подпись	Дата

```
if asquare.Color <> selectedsqr.color then
                                                       SelectedSqr.picture.Bitmap := Abm;
                                                       Selected := false;
 begin
 Atempbm := TBitmap.Create;
                                                      end:
 with Atempbm do
                                                      if CheckDetect = CheckTurn then
 begin
                                                      begin
  PixelFormat := pf32bit;
                                                       if not InCheck then
  Height := imageSize;
                                                       begin
  Width := Height;
                                                        incheck := true;
 end;
                                                        showmessage('Wax!');
 Atempbm.Assign(SelectedSqr.picture.Bit-
                                                        beep;
                                                        SelectedSqr := reverseSelected;
map);
 for y := 0 to imageSize - 1 do
                                                        Move(reverseSquare, squarebm);
  for x := 0 to imageSize - 1 do
                                                        ASquare.Kind := result;
                                                        selectedSqr := reverseSquare;
  begin
   pbase := Atempbm.ScanLine[y];
                                                        Turn := CheckTurn;
   p := PDW(DWORD(pbase) + (x shl 2));
                                                       end
   case ASquare.Color of
                                                       else
    2:
                                                        InCheck := false;
     if p^{*} = $0 then
                                                      end
       p^{\wedge} := FFFFFF;
                                                      else if CheckDetect <> 0 then
     1:
      selectedSqr := reverseSquare;
       p^{*} := $0;
   end;
                                                      squarebm.Destroy;
                                                    end;
  end:
 ASquare.picture.Bitmap := Atempbm;
 SelectedSqr.picture.Bitmap := Abm;
                                                    procedure TBoardMannager.SetAutoDese-
 ASquare.Kind := SelectedSqr.Kind;
                                                    lect(const Value: boolean);
 SelectedSqr.Kind := 0;
                                                    begin
 Selected := false;
                                                      FAutoDeselect := Value;
 freeandnil(atempbm);
                                                    end;
 end
 else
                                                    procedure TBoardMannager.Setbishop(const
                                                     Value: TBitmap);
 begin
  ASquare.picture.Bitmap := SelectedSqr.pic-
                                                    begin
ture.Bitmap;
                                                      Fbishop := Value;
  ASquare.Kind := selectedsqr.Kind;
                                                    end;
  SelectedSqr.Kind := 0;
```

Изм.	Лист	№докцм.	Подпись	Дата

<pre>nager.SetOutlineColor(const Value: TColor); begin FOutlineColor := rgb(GetBValue(value), GetGValue(Value),GetRValue(Value)); end; procedure TBoardMannager.Setpawn(const Value: TBitmap); begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName- White(const Value: string);</pre>
FOutlineColor := rgb(GetBValue(value), GetGValue(Value),GetRValue(Value)); end; procedure TBoardMannager.Setpawn(const Value: TBitmap); begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
GetGValue(Value),GetRValue(Value)); end; procedure TBoardMannager.Setpawn(const Value: TBitmap); begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
end; procedure TBoardMannager.Setpawn(const Value: TBitmap); begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
<pre>procedure TBoardMannager.Setpawn(const Value: TBitmap); begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-</pre>
Value: TBitmap); begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
begin Fpawn := Value; end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
end; procedure TBoardMannager.SetPlayerName- Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
Black(const Value: string); begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
begin FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
FPlayerNameBlack := Value; end; procedure TBoardMannager.SetPlayerName-
end; procedure TBoardMannager.SetPlayerName-
procedure TBoardMannager.SetPlayerName-
begin
FPlayerNameWhite := Value;
end;
procedure TBoardMannager.Setqueen(const
Value: TBitmap);
begin
Fqueen := Value;
end;
procedure TBoardMannager.SetSelected(const
Value: boolean);
begin
FSelected := Value;
end;
procedure TBoardMannager.SetSelected-
Sqr(const Value: TSquare);
begin
FSelectedSqr := Value;
end;
procedure TBoardMannager.SetSquareTo(Lo-
cation: TPoint; Kind: integer);

Изм.	Лист	№докум.	Подпись	Дата

```
for x := 0 to imageSize - 1 do
var
 tempbm: TBitmap;
                                                         begin
 x, y: integer;
                                                          pbase := tempbm.ScanLine[y];
 pbase, p: PDW;
                                                          p := PDW(DWORD(pbase) + (x shl 2));
begin
                                                          case p^ of
 if (Location.x IN [1 .. 8]) AND (Location.y IN
                                                           $0000FF:
[1 .. 8]) then
                                                             if odd(Location.y + Location.x - orien-
 begin
                                                     tation) then
                                                              p^{\wedge} := $000000
  tempbm := TBitmap.Create;
  with tempbm do
                                                             else
  begin
                                                              p^ := $FFFFF;
   PixelFormat := pf32bit;
                                                            $00FF00:
                                                             p^ := outlineColor;
   Height := imageSize;
                                                            $FF0000:
   Width := Height;
  end:
                                                             begin
  case Kind of
                                                              if Kind > 0 then
                                                               p^ := WhiteColor
   1, -1:
                                                              else
    tempbm.Assign(pawn);
   2, -2:
                                                               p^ := BlackColor;
    tempbm.Assign(castle);
                                                             end;
   3, -3:
                                                          end;
    tempbm.Assign(bishop);
                                                         end:
                                                        if Kind <> 0 then
   4, -4:
                                                        Board[Location.x, Location.y].picture.Bit-
    tempbm.Assign(horse);
   5, -5:
                                                     map := tempbm;
    tempbm.Assign(queen);
                                                        Board[Location.x, Location.y].Kind := Kind;
                                                        tempbm.Destroy;
   6, -6:
                                                       end;
    tempbm.Assign(king);
   0:
                                                     end:
   begin
                                                     procedure TBoardMannager.SetTurn(const
     for y := 0 to imageSize - 1 do
                                                     Value: integer);
      for x := 0 to imageSize - 1 do
                                                     begin
                                                      FTurn := Value;
       pbase := tempbm.ScanLine[y];
       p := PDW(DWORD(pbase) + (x shl 2));
                                                     end;
       p^{*} := $0000FF;
                                                     procedure TBoardMannager.SetWhite-
                                                     Color(const Value: TColor);
   end:
  end;
                                                     begin
  for y := 0 to imageSize - 1 do
```

	·			·
Изм.	Лист	№докцм.	Подпись	Дата

```
FWhiteColor := rgb(GetBValue(value),
                                                       LinkedBoard := nil;
                                                       rootdir := ";
GetGValue(Value),GetRValue(Value));
                                                      end;
end;
procedure TBoardMannager.SetWhite-
                                                      procedure TSaveManager.LoadFrom-
PiecesTook(Index: Integer; Value: Integer);
                                                      File(filepath: string);
begin
                                                      var
 FWhitePiecesTook[Index] := Value;
                                                       tS: TextFile;
end;
                                                       x: Integer;
procedure TBoardMan-
                                                       y: Integer;
nager.TakePiece(ASquare: TSquare; Abm:
                                                       i: Integer;
TBitmap);
                                                       s: string;
var
                                                       t1, t2: integer;
 i, oKind: integer;
                                                       PGNPath: string;
begin
                                                      begin
 okind := selectedsqr.Kind;
                                                       PGNPath := filepath;
 i := Move(ASquare, Abm);
                                                       delete(PGNPath, pos('.', PGNPath), 6);
 if ASquare.Kind = oKind then
                                                       PGNPath := PGNPath + '.PGN';
  case i of
                                                       if FileExists(PGNPath) then
   1 .. 6:
                                                       assignFile(tS, filepath);
    begin
                                                       reset(tS);
      WhitePiecesTook[ high(FWhite-
                                                       readln(tS, s);
PiecesTook)] := i;
                                                       with LinkedBoard do
      SetLength(FWhitePiecesTook,
                                                       begin
length(FWhitePiecesTook) + 1);
                                                        Clear;
     end:
                                                        drawboard;
   -6...-1:
                                                        t1 := GetTickCount:
    begin
                                                        turn := strtoint(s);
      BlackPiecesTook[ high(FBlack-
                                                        for y := 1 to 8 do
PiecesTook)] := i * -1;
                                                        begin
      SetLength(FBlackPiecesTook,
                                                         readln(tS, s);
length(FBlackPiecesTook) + 1);
                                                         for x := 1 to 8 do
     end;
                                                         begin
  end;
end;
                                                      SetSquareTo(Point(x,y),strtoint(copy(s,1,2)));
                                                           delete(s, 1, 2);
{ TSaveManager }
constructor TSaveManager.Create(AOwner:
                                                         end;
TObject);
                                                        end;
begin
                                                        readln(tS, s);
```

Изм.	Лист	№доким.	Подпись	Дата

```
SetLength(FWhitePiecesTook, length(S));
                                                          y: Integer;
  for i := 0 to length(s) - 1 do
                                                          s: Integer;
  begin
                                                          pgnpath: string;
   WhitePiecesTook[i] := strtoint(copy(s, 1,
                                                        begin
                                                          PGNPath := filepath;
1));
                                                          delete(PGNPath, pos('.', PGNPath), 6);
   delete(s, 1, 1);
  end;
                                                          PGNPath := PGNPath + '.PGN';
  readln(tS, s);
                                                          assignfile(tS, PGNPath);
  SetLength(FBlackPiecesTook, length(S));
                                                          rewrite(ts);
  for i := 0 to length(s) - 1 do
                                                          closefile(ts);
  begin
                                                          assignFile(tS, filepath);
    BlackPiecesTook[i] := strtoint(copy(s, 1,
                                                          rewrite(tS);
                                                          with LinkedBoard do
1));
   delete(s, 1, 1);
                                                          begin
                                                           if selected then
  end:
  readln(ts, s);
                                                           begin
  if s = 'TRUE' then
                                                            selected := false;
   CastlingPossible[1] := true
                                                            if Turn = 1 then
  else
                                                             turn := 2
   CastlingPossible[1] := false;
                                                            else
  readln(ts, s);
                                                             turn := 1;
  if s = 'TRUE' then
                                                           end;
   CastlingPossible[2] := true
                                                           writeln(tS, turn);
  else
                                                           for y := 1 to 8 do
   CastlingPossible[2] := false;
                                                           begin
  readln(tS, s);
                                                            for x := 1 to 8 do
  PlayerNameWhite := s;
                                                            begin
  readln(tS, s);
                                                             if Board[x, y].Kind >= 0 then
  PlayerNameBlack := s;
                                                               write(tS, FormatFloat('00', Board[x,
                                                        y].Kind))
 end;
 closefile(ts);
                                                             else
 t2 := GetTickCount;
                                                               write(tS, Board[x, y].Kind);
end:
                                                            end;
procedure TSaveManager.SaveToFile(filepath:
                                                            write(tS, #13#10);
                                                           end;
string);
                                                           for s := 0 to getWhiteTookLength do
 tS: TextFile;
                                                            write(tS ,WhitePiecesTook[s]);
 x: Integer;
                                                           write(tS, #13#10);
```

Изм.	Лист	№докцм.	Подпись	Дата

```
for s := 0 to getBlackTookLength do
                                                           selected := false;
                                                           if Turn = 1 then
    write(tS, BlackPiecesTook[s]);
  write(tS, #13#10);
                                                            turn := 2
  writeln(ts, CastlingPossible[1]);
                                                           else
  writeln(ts, CastlingPossible[2]);
                                                            turn := 1;
  writeln(tS, PlayerNameWhite);
                                                          end;
  writeln(tS, PlayerNameBlack);
                                                          writeln(tS, turn);
                                                          for y := 1 to 8 do
 end;
 closefile(tS);
                                                          begin
 assignFile(tS, rootDir + '\_LOG.DWCS');
                                                           for x := 1 to 8 do
 if not fileExists(rootDir + '\_LOG.DWCS')
                                                           begin
                                                            if Board[x, y]. Kind \geq 0 then
then
                                                             write(tS, FormatFloat('00', Board[x,
  rewrite(tS);
 Append(tS);
                                                       y].Kind))
 writeLn(tS, filepath);
                                                            else
 closefile(tS);
                                                             write(tS, Board[x, y].Kind);
end;
                                                           end;
procedure TSaveManager.SaveToFileOver-
                                                           write(tS, #13#10);
write(filepath: string);
                                                          end;
                                                          for s := 0 to getWhiteTookLength do
var
 tS: TextFile;
                                                           write(tS ,WhitePiecesTook[s]);
 x: Integer;
                                                          write(tS, #13#10);
                                                          for s := 0 to getBlackTookLength do
 y: Integer;
 s: Integer;
                                                           write(tS, BlackPiecesTook[s]);
 pgnpath: string;
                                                          write(tS, #13#10);
begin
                                                          writeln(ts, CastlingPossible[1]);
 PGNPath := filepath;
                                                          writeln(ts, CastlingPossible[2]);
 delete(PGNPath, pos('.', PGNPath), 6);
                                                          writeln(tS, PlayerNameWhite);
 PGNPath := PGNPath + '.PGN';
                                                          writeln(tS, PlayerNameBlack);
 assignfile(tS, PGNPath);
                                                        end;
 rewrite(ts);
                                                        closefile(tS);
 closefile(ts);
                                                       end;
 assignFile(tS, filepath);
                                                       procedure
 rewrite(tS);
                                                       TSaveManager.SetLinkedBoard(const Value:
 with LinkedBoard do
                                                       TBoardMannager);
                                                       begin
 begin
  if selected then
                                                        FLinkedBoard := Value:
  begin
                                                       end:
```

Изм.	Лист	№докцм.	Подпись	Дата

 $procedure\ TS ave Manager. Setroot Dir (const$ Value: string); begin $if \ not \ Directory Exists (value) \ then$ CreateDir(value); FrootDir := Value; end; end. Лист KΠ 2-40 01 01.35.40.13.24 70 Изм. Лист №докцм. Подпись Дата