МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Подп. и дата

Инв. № дубл.

Заим. инв. №

Подп. и дата

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МИНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

**Головоломка «Уголки»**

Пояснительная записка

к курсовому проекту по предмету:

«Основы алгоритмизации и программирования»

КП 03491.002206.081 ПЗ

Руководитель Назаров Л.В.

Учащаяся Варченя И.В.

Ин. № подл.

Минск 2017

***Содержание***

[Введение 4](#_Toc481398716)

[1 Постановка задачи 5](#_Toc481398717)

[1.1 Назначение и функции программы 5](#_Toc481398718)

[1.2 Список составляющих программы 5](#_Toc481398719)

[1.3 Метод реализации основных функций программы 5](#_Toc481398720)

[1.4 Описание алгоритмов программы 5](#_Toc481398721)

[2 Описание программы 7](#_Toc481398722)

[2.1 Окна и формы программы 7](#_Toc481398723)

[2.2 Справочная система 7](#_Toc481398724)

[2.3 Графический дизайн 7](#_Toc481398725)

[2.4 Описание функций программы 7](#_Toc481398726)

[3 Описание программы 9](#_Toc481398727)

[3.1 Порядок работы с программой 9](#_Toc481398728)

[3.2 Установка программы 9](#_Toc481398729)

[Заключение 10](#_Toc481398730)

[Список используемых источников 11](#_Toc481398731)

[Приложение А](#_Toc481398732) [Схема принципа работы программы 12](#_Toc481398734)

[Приложение Б](#_Toc481398735) [Основные формы программы 13](#_Toc481398737)

[Приложение В](#_Toc481398738) [Текст программы 15](#_Toc481398740)

*Головоломка*

*«Уголки»*

*Пояснительная записка*

*Листов*

*Лист.*

*Лит.*

*Лист*

*КП 03491.002206.081*

*МРК*

*Дата*

*Подпись*

*Пров.*

*Разраб.*

*Реценз.*

*Н. Контр.*

*Назаров Л.В.*

*Варченя И.В.*

*Изм.*

*Лист.*

*№ докум.*

*3*

*43*

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники филиал «Минский Радиотехнический Колледж»

УТВЕРЖДАЮ

Проректор

По учебной работе\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_\_г.

**ЗАДАНИЕ**

Для курсового проектирования по предмету *Основы алгоритмизации и программирования* студенту\_\_ *II* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_курса\_\_\_\_*\_52492*\_\_\_\_\_\_\_\_\_\_группы

Специальности *Программное обеспечение информационных технологий*

*Варчене Илье Владимировичу*

Тема курсового проекта (работы) и исходные данные

*«Уголки» (головоломка)*

*Исходные данные: текстовые файлы с данными*

При выполнении курсового проекта (работы) по названной теме должны быть представлены:

1. **Пояснительная записка**

*Введение 1 Постановка задачи 1.1 Назначение и функции программы 1.2 Список составляющих программы 1.3.Метод реализации основных функций программы 2 Описание программы 2.1 Окна и формы программы 2.2 Справочная система 2.3 Графический дизайн*

*2.4 Описание функций программы 3* [*Описание программы*](#_Toc296275727) *3.1*  [*Порядок работы с программой*](#_Toc296275727) *3.2* [*Установка программы Заключение*](#_Toc296275727)

1. **Графическая часть проекта (работы)**

Лист 1 *Схема основного алгоритма программы*

Лист 2

Лист 3

Лист 4

Дата выдачи *«\_9\_»* \_\_\_\_\_*\_февраля*\_\_\_\_\_\_*2017*г.

Срок завершения «\_*2*\_» \_\_\_\_\_\_*мая\_\_\_\_\_\_*20*17*г.

**Преподаватель-руководитель**

**курсового проекта (работы):**

**Заведующий цикловой комиссии:**

Введение

Уголки — настольная логическая игра на квадратном поле размером 8x8 клеток для двух участников.

Игра, скорее всего, возникла как домашний вариант более древней игры халма.

Цель игры переставить все свои шашки в дом соперника. Игрок, сделавший это первым, выигрывает.

Оригинальная версия играется на доске 16x16, в России более популярен вариант игры на шахматной доске, 8х8 клеток.

В начальном положении шашки обоих игроков занимают свои стартовые позиции (дома). Дом может иметь три варианта размещения шашек: 3х3, 3х4, и углом.

Каждый игрок может за один ход переместить одну шашку. Шашки можно перемещать в вертикальном и горизонтальном направлении на соседнюю пустую клетку, шашки могут перепрыгивать через свои и чужие шашки. Перепрыгивать можно, если за шашкой есть пустая клетка. Прыжки могут быть многократными, при этом перепрыгивать шашка может как свои шашки, так и шашки противника. Длина прыжка не принудительна, то есть игрок может решить в любое время прекратить многократный ход.

Игра завершается, если один из игроков переместил все свои шашки в дом соперника, и, как следствие, он победил.

1 Постановка задачи

1.1 Назначение и функции программы

Данная программа представляет собой игру-головоломку «Уголки». Цель игры – быстрее оппонента передвинуть все свои шашки на то место, где изначально располагались его шашки.

Главной функцией данной программы, является предоставление игрового поля пользователю. Данная программа позволяет в любой момент по желанию пользователя изменить выбранное им поле для игры, а также программа проверяет правильность хода и позволяет играть с компьютером.

Созданная программа может использоваться пользователем для развлекательных целей, а так же в целях развития логического мышления.

1.2 Список составляющих программы

* Игровое поле;
* Предоставление различных стартовых позиций шашек;
* Проверка хода;
* Генерация автоматического поиска и совершения наилучшего хода;
* Справочная система.

1.3 Метод реализации основных функций программы

Игровое поле, в программе представляет собой обработанную таблицу DrawGrid. При помощи графических возможностей языка программирования Delphi, обычная таблица приобрела вид поля головоломки «уголки» при помощи «таблицы из картинок».

Загрузка первоначального игрового поля происходит после запуска программы пользователем.

Проверка решения происходит посредством правил игры, т.е. проверяются условия хода: ход не далее одной клетки по горизонтали и вертикали, прыжок только через шашки по диагонали и вертикали. При неверном ходе на экран выводится вспомогательное окно, которое информирует об этом.

Ходы совершаются игроками последовательно.

Генерация автоматического решения реализуется при помощи первоначальной обработки, всевозможных значений в каждой ячейке игрового поля, т.е. (вычеркивание одинаковых элементов, расстановки единственного варианта в ячейку игрового поля), А далее при помощи составления дерева ходов и поиска наилучшего из них.

Справочная система, которая выводится в отдельную форму. Справочная система дает первое общее представление пользователю о программе, в справочной системе указаны правила игры, а так же показаны формы головоломки, что позволяет, более точно узнать как работает программа, и какие возможности она предусматривает.

1.4 Описание алгоритмов программы

Основной принцип работы программы показан в блок-схеме (Приложение А).

После загрузки программы пользователю предоставляется первое игровое поле, которое он по желанию может изменить на иные поля из предложенных. Если пользователь захочет изменить поле в ходе игры, то на экране появится окошко с предупреждением. Условие каждого заданного поля закодировано в программе.

Первый ход всегда за белыми. Ход игрока проверяется на правильность условиям игры. При неправильном ходе появиться окошко с уведомлением «Невозможный ход».

Пользователь может играть против компьютера и против «живого» оппонента. Если играющий желает сделать автоматический ход (ход компьютера), то он нажимает на кнопку с символом « >>> ».

При генерации автоматического решения, создаётся динамическое дерево всевозможных ходов, а после из него выбирается наилучший ход.

При желании можно повернуть игровое поле на 180 градусов.

По завершению игры появляется уведомление о том, какого цвета шашки победили.

2 Описание программы

2.1 Окна и формы программы

**Окна - объекты графического интерфейса.** Основу любого графического интерфейса пользователя составляет организованная и хорошо продуманная система окон и других графических объектов.

**Окно -** обрамленная прямоугольная область на экране монитора, в которой отображаются приложение, сообщение. Окно является активным, если с ним в данный момент работает пользователь.

Данная программа состоит из двух форм.

Главная форма (Приложение В, рис.1) открывается сразу же при запуске программы. На главной форме содержатся:

* Игровое поле;
* Кнопка «Первого поля» (выбор первой расстановки шашек);
* Кнопка «Второго поля» (выбор второй расстановки шашек);
* Кнопка «Третьего поля» (выбор третьей расстановки шашек);
* Кнопка «Поворот поля» (поворачивает игровое поле на 180 градусов);
* Кнопка «Автоматический ход» (генерация авто хода);
* Кнопка «Прервать автоматический ход»;
* Список выбора глубины авто хода;
* Кнопка «Выход»;
* Список ведения ходов;
* Меню программы.

Вспомогательные формы и окна:

1. Форма «Помощь» (Приложение В, рис.2)

Данная форма открывается при нажатии кнопки меню «Помощь» и предоставляет пользователю правила игры и правила использования программы. Данная форма содержит картинку с правилами и кнопку возврата к окну с игровым полем.

1. Окно предупреждения о невозможном ходе (Приложение В, рис.3)

Данное окно предупреждает пользователя о том, что он совершил невозможный ход.

1. Окно «Объявление победителя» (Приложение В, рис.4)

Данное окно информирует пользователя в конце игры о том, какого цвета шашки победили.

2.2 Справочная система

Для программы предусмотрена справочная система, представленная формой «Помощь» (Приложение В, рис.2), которая выводится при нажатии одноимённой кнопки меню. Справочная система дает первое общее представление пользователю о самой программе и о правилах игры.

2.3 Графический дизайн

Графический дизайн — художественно-проектная деятельность по созданию гармоничной и эффективной визуально-коммуникативной среды.

Для головоломки «Уголки»» был создан уникальный интерфейс. Цвета форм подобраны так, чтобы пользователь не отвлекался от самого решения головоломки.

2.4 Описание функций программы

Для составления программы использовались следующие функции:

* procedure FormCreate(Sender: TObject); {создание формы}
* procedure DrawGridDrawCell(Sender: TObject; ACol, ARow: Integer; {создание игрового поля}
* Rect: TRect; State: TGridDrawState);
* procedure DrawGridClick(Sender: TObject);
* procedure RotateClick(Sender: TObject); {переворот игрового поля}
* procedure AddHistoryString(Count: Integer; Turn: State; Move: TMove); {ведение истории ходов}
* procedure CompMoveButtonClick(Sender: TObject); {авто ход}
* procedure DrawGridExit(Sender: TObject); {работа с полем}
* procedure FormClose(Sender: TObject; var Action: TCloseAction); {переход формы}
* procedure BreakButtonClick(Sender: TObject); {прерывание ато хода}
* procedure Start1Click(Sender: TObject); {выбор первого игрового поля}
* procedure Start2Click(Sender: TObject); {выбор второго игрового поля}
* procedure btnStart3Click(Sender: TObject); {выбор третьего игрового поля}
* procedure N3Click(Sender: TObject); {Кнопки меню}
* procedure N11Click(Sender: TObject);
* procedure N21Click(Sender: TObject);
* procedure N31Click(Sender: TObject);
* procedure N2Click(Sender: TObject).

3 Описание программы

3.1 Порядок работы с программой

При загрузке программы происходит запуск главной формы, содержащей первое игровое поле, чтобы перейти к другой раскладке, он может выбрать поле путём нажатия на кнопки выбора расстановки шашек. Также в любой момент игры пользователь может остановить текущую игру и выбрать другую расстановку шашек, выбрать автоматический ход, вызвать справку.

3.2 Установка программы

Для установки программного средства необходимо 20Мb свободного места на жестком диске и установленную на компьютере операционную систему Windows 9x/2000/ME/NT/XP/7/8/8.1/10.

Установка производится путем копирования папки, не нарушая внутреннюю структуру каталогов (это повлечет за собой различного рода сбои и ошибки при работе программы), на любой магнитный носитель позволяющий перезаписывать и дозаписывать данные.

Заключение

Созданная программа демонстрирует потенциальные возможности языка Delphi в области разработки игровых приложений. Использование компонентов Delphi в значительной мере упростило процесс написания программы и сократило код проекта. Программа не претендует на место серьезного программного продукта, однако является хорошим примером при изучении данного языка.

Поскольку Delphi в настоящее время является программным продуктом, в котором отражены самые современные достижения компьютерных технологий, то он позволяет создавать сложные Windows-приложения с применением широкого спектра различных компонентов. В разработанной программе используются для управления, отображения и редактирования информации стандартные компоненты*,* однако эти же функции могут выполнять и другие более сложные элементы управления.

Список используемых источников

**Основная:**

* *Архангельский А.Я.*Delphi 7. Справочное пособие. –М:ООО «Бином-Пресс», 2003.
* *Баженова И.Ю.*Delphi 7. Самоучитель программиста. – М.:КУДИЦ-ОБРАЗ.
* *Бакнелл Дж*. Фундаментальные алгоритмы и структуры данных в Delphi. Библиотека программиста. – М.: ООО «ДиаСофтЮП»; СПб Питер, 2006.
* *Бобровский С.И.*Delphi 7. Учебный курс. – СПб:Питер, 2003.
* *Вирт Н.*Алгоритмы и структуры данных: Пер. с англ. – М.: Мир, 1980.
* Фленов М. Библия Delphi. – СПб.: Питер, 2004. – 884 с.
* Архангельский А. Я. 100 компонентов общего назначения библиотеки Delphi5, 2002.
* Баженова И. Ю. Самоучитель программиста, Москва, 2003.

**Дополнительная:**

* *Культин Н.Б*. Delphi в задачах и примерах. – СПб.:БХВ-Петербург, 2003. – 288 с.: ил.
* *Озеров В.*Delphi. Советы программистов. –СПб:Символ-Плюс, 2004. – 976с., ил.

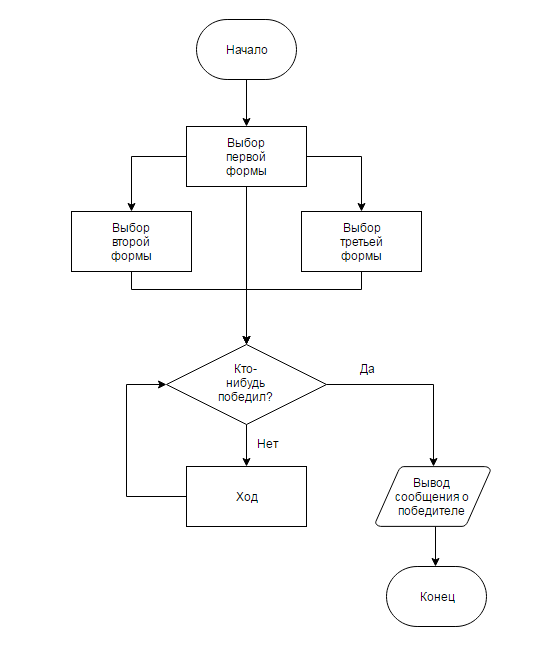
**Список интернет источников:**

* + - ru.wikipedia.org;
    - [www.delphisources.ru](http://www.delphisources.ru);
    - [www.cyberforum.ru](http://www.cyberforum.ru);
    - www.softholm.com;
    - [www.gambler.ru](http://www.gambler.ru).

Приложение А

(Справочное)

Схема принципа работы программы



Приложение Б

(справочное)

Основные формы программы

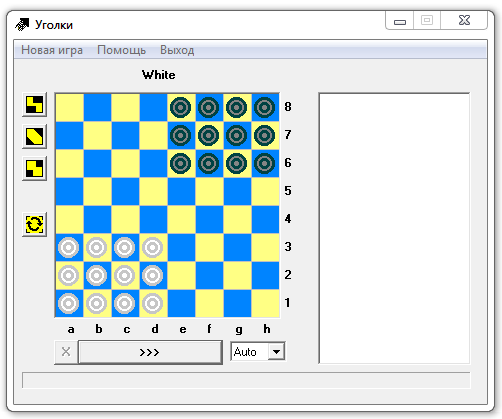


Рис. 1. Стартовая форма программы с классической расстановкой шашек

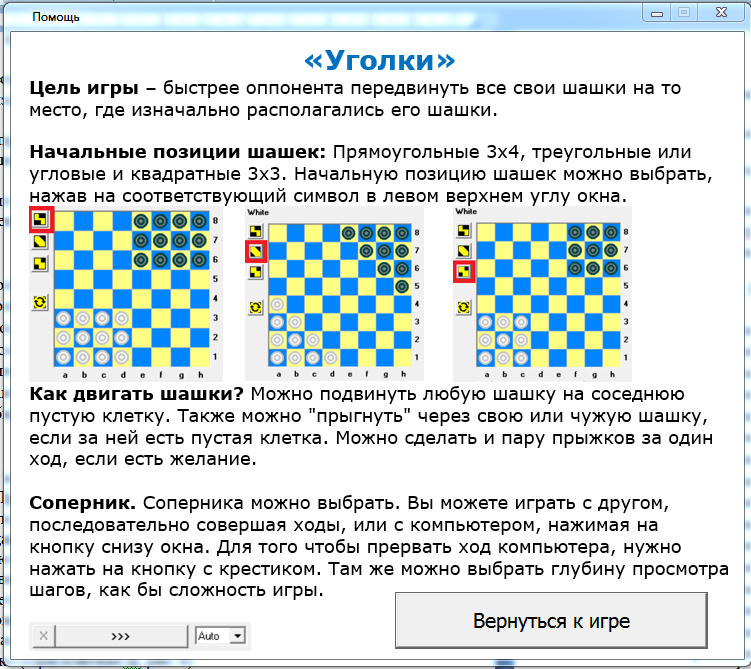


Рис. 2. Окно «Помощь»

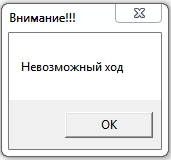


Рис. 3. Окно предупреждения о невозможном ходе

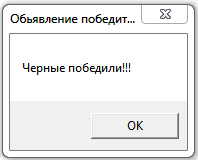


Рис. 4. Окно «Объявление победителя»

Приложение В

(Обязательное)

Текст программы

program Corner;

uses

Forms,

MFrame in 'MFrame.pas' {Form1},

Position in 'Position.pas',

Unit2 in 'Unit2.pas' {Form2};

{$R \*.RES}

begin

Application.Initialize;

Application.CreateForm(TForm1, Form1);

Application.CreateForm(TForm2, Form2);

Application.Run;

end.

unit MFrame;

//Головоломка "Уголки"

//Создал Варченя И.В. гр.52492

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Position,

StdCtrls, Grids, Buttons, ComCtrls, ImgList, Menus;

type

TForm1 = class(TForm)

TurnLabel: TLabel;

CompMoveButton: TButton;

DepthComboBox: TComboBox;

ProgressBar: TProgressBar;

Start1: TBitBtn;

Start2: TBitBtn;

Rotate: TBitBtn;

DrawGrid: TDrawGrid;

LabelA: TLabel;

DeskImages: TImageList;

HistoryListBox: TListBox;

LabelB: TLabel;

LabelC: TLabel;

LabelD: TLabel;

LabelE: TLabel;

LabelF: TLabel;

LabelG: TLabel;

LabelH: TLabel;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

Label7: TLabel;

Label8: TLabel;

BreakButton: TButton;

btnStart3: TBitBtn;

mm1: TMainMenu;

N1: TMenuItem;

N2: TMenuItem;

N3: TMenuItem;

N11: TMenuItem;

N21: TMenuItem;

N31: TMenuItem;

procedure FormCreate(Sender: TObject); {создание формы}

procedure DrawGridDrawCell(Sender: TObject; ACol, ARow: Integer; {создание игрового поля}

Rect: TRect; State: TGridDrawState);

procedure DrawGridClick(Sender: TObject);

procedure RotateClick(Sender: TObject); {переворот игрового поля}

procedure AddHistoryString(Count: Integer; Turn: State; Move: TMove); {ведение истории ходов}

procedure CompMoveButtonClick(Sender: TObject); {авто ход}

procedure DrawGridExit(Sender: TObject); {работа с полем}

procedure FormClose(Sender: TObject; var Action: TCloseAction); {переход формы}

procedure BreakButtonClick(Sender: TObject); {прерывание ато хода}

procedure Start1Click(Sender: TObject); {выбор первого игрового поля}

procedure Start2Click(Sender: TObject); {выбор второго игрового поля}

procedure btnStart3Click(Sender: TObject); {выбор третьего игрового поля}

procedure N3Click(Sender: TObject); {Кнопки меню}

procedure N11Click(Sender: TObject);

procedure N21Click(Sender: TObject);

procedure N31Click(Sender: TObject);

procedure N2Click(Sender: TObject);

private

{ Private declarations }

IsFlipped: Boolean; {Переворачивание доски}

IsStartSelected: Boolean; {Выбрано ли откуда ходить}

ManualMove: TMove; {Ход}

MoveCount: Integer; {Число ходов}

{Для авто определения глубины просмотра}

LastCount: Integer; {примерное число возможных ходов}

CurrentDepth: Integer; {глубина просмотра для auto}

Procedure LabelLayout; {Расположить метки}

Procedure WinMessage; {Вывести сообщение о победе и сбросить позицию в ноль}

public

{ Public declarations }

end;

var

Form1: TForm1;

Pos: TPosition;

implementation

uses Unit2;

{$R \*.DFM}

//Стартовая позиция 1

var StartPosition1: TBoard=(

(White,White,White,Space,Space,Space,Space,Space),

(White,White,White,Space,Space,Space,Space,Space),

(White,White,White,Space,Space,Space,Space,Space),

(White,White,White,Space,Space,Space,Space,Space),

(Space,Space,Space,Space,Space,Black,Black,Black),

(Space,Space,Space,Space,Space,Black,Black,Black),

(Space,Space,Space,Space,Space,Black,Black,Black),

(Space,Space,Space,Space,Space,Black,Black,Black));

//Стартовая позиция 2

var StartPosition2: TBoard=(

(White,White,White,White,Space,Space,Space,Space),

(White,White,White,Space,Space,Space,Space,Space),

(White,White,Space,Space,Space,Space,Space,Space),

(White,Space,Space,Space,Space,Space,Space,Space),

(Space,Space,Space,Space,Space,Space,Space,Black),

(Space,Space,Space,Space,Space,Space,Black,Black),

(Space,Space,Space,Space,Space,Black,Black,Black),

(Space,Space,Space,Space,Black,Black,Black,Black));

//Стартовая позиция 3

var StartPosition3: TBoard=(

(White,White,White,Space,Space,Space,Space,Space),

(White,White,White,Space,Space,Space,Space,Space),

(White,White,White,Space,Space,Space,Space,Space),

(Space,Space,Space,Space,Space,Space,Space,Space),

(Space,Space,Space,Space,Space,Space,Space,Space),

(Space,Space,Space,Space,Space,Black,Black,Black),

(Space,Space,Space,Space,Space,Black,Black,Black),

(Space,Space,Space,Space,Space,Black,Black,Black));

//Стартовые настройки

procedure TForm1.FormCreate(Sender: TObject);

begin

Pos:=TPosition.Create;

Pos.SetInitialPosition(StartPosition1);

Pos.RestartGame;

IsFlipped:=False;

IsStartSelected:=False;

MoveCount:=0;

DepthComboBox.ItemIndex:=0;

LastCount:=15;

CurrentDepth:=4; DrawGrid.Width:=DrawGrid.DefaultColWidth\*DrawGrid.ColCount+(DrawGrid.ColCount+1)\*DrawGrid.GridLineWidth+2{border}; DrawGrid.Height:=DrawGrid.DefaultRowHeight\*DrawGrid.RowCount+(DrawGrid.RowCount+1)\*DrawGrid.GridLineWidth+2;

LabelLayout;

Randomize;

end;

//Процедура ведения истории ходов

procedure TForm1.AddHistoryString(Count: Integer; Turn: State; Move: TMove);

begin

if Turn=White then

HistoryListBox.Items.Add(Format('%.2d: %s%d-%s%d ',[Count div 2+1,Chr(Ord(Move.MoveFrom.X)+Ord('a')), Move.MoveFrom.Y, Chr(Ord(Move.MoveTo.X)+Ord('a')),Move.MoveTo.Y]))

else

HistoryListBox.Items[(Count-1) div 2]:=HistoryListBox.Items[(Count-1) div 2]+Format('%s%d-%s%d',[Chr(Ord(Move.MoveFrom.X)+Ord('a')),Move.MoveFrom.Y,Chr(Ord(Move.MoveTo.X)+Ord('a')),Move.MoveTo.Y]);

HistoryListBox.ItemIndex:=(Count-1) div 2;

HistoryListBox.ItemIndex:=-1;

end;

//Линия прогресса обработки хода

procedure ProgressHandler(Cur,Max:Integer);

begin

Form1.ProgressBar.Position:=(Cur\*100) div Max;

Form1.LastCount:=Max;

end;

//Если выбрать другую раскладку поля, то игра прервётся

procedure TForm1.DrawGridExit(Sender: TObject);

begin

IsStartSelected:=False;

TDrawGrid(Sender).Refresh;

end;

//Отображение состояния ячейки

procedure TForm1.DrawGridDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

var

BMP: TBitmap;

begin

BMP:=TBitmap.Create;

DeskImages.GetBitmap(2,BMP);

if not((odd(ACol)and odd(ARow)) or ((not odd(ACol)) and (not odd(ARow)))) then

begin

{Черная клетка}

if IsFlipped then

begin

case Pos.Get(HNum(7-ACol),ARow+1) of

Space:

DeskImages.GetBitmap(0,BMP);

Black:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(7-ACol)) and (ManualMove.MoveFrom.Y=ARow+1) then

DeskImages.GetBitmap(5,BMP)

else

DeskImages.GetBitmap(3,BMP);

White:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(7-ACol)) and (ManualMove.MoveFrom.Y=ARow+1) then

DeskImages.GetBitmap(4,BMP);

else

DeskImages.GetBitmap(2,BMP);

end

end

else

case Pos.Get(HNum(ACol),8-ARow) of

Space:

DeskImages.GetBitmap(0,BMP);

Black:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(ACol)) and (ManualMove.MoveFrom.Y=8-ARow) then

DeskImages.GetBitmap(5,BMP)

else

DeskImages.GetBitmap(3,BMP);

White:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(ACol)) and (ManualMove.MoveFrom.Y=8-ARow) then

DeskImages.GetBitmap(4,BMP)

else

DeskImages.GetBitmap(2,BMP);

end

end

else

begin

{Белая клетка}

if IsFlipped then

begin

case Pos.Get(HNum(7-ACol),ARow+1) of

Space:

DeskImages.GetBitmap(1,BMP);

Black:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(7-ACol)) and (ManualMove.MoveFrom.Y=ARow+1) then

DeskImages.GetBitmap(9,BMP)

else

DeskImages.GetBitmap(7,BMP);

White:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(7-ACol)) and (ManualMove.MoveFrom.Y=ARow+1) then

DeskImages.GetBitmap(8,BMP)

else

DeskImages.GetBitmap(6,BMP);

end;

end

else

begin

case Pos.Get(HNum(ACol),8-ARow) of

Space:

DeskImages.GetBitmap(1,BMP);

Black:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(ACol)) and (ManualMove.MoveFrom.Y=8-ARow) then

DeskImages.GetBitmap(9,BMP)

else

DeskImages.GetBitmap(7,BMP);

White:

if IsStartSelected and (ManualMove.MoveFrom.X=HNum(ACol)) and (ManualMove.MoveFrom.Y=8-ARow) then

DeskImages.GetBitmap(8,BMP)

else

DeskImages.GetBitmap(6,BMP);

end;

end;

end;

//выбор клетки

With Sender as TDrawGrid do

begin

canvas.Draw(Rect.Left, Rect.Top, BMP);

end;

BMP.Destroy;

end;

//Обработка щелчков мышкой на доске

procedure TForm1.DrawGridClick(Sender: TObject);

var

X,Y: Integer;

begin

{Первый щечок}

X:=TDrawGrid(Sender).Selection.Left+1;

Y:=TDrawGrid(Sender).Selection.Top+1;

if(IsFlipped) then

X:=9-X

else

Y:=9-Y;

if IsStartSelected then

begin

{Второй щелчок}

IsStartSelected:=False;

ManualMove.MoveTo.X:=HNum(X-1);

ManualMove.MoveTo.Y:=Y;

//Если ход возможен, то...

if Pos.IsMovePossible(ManualMove) then

begin

//Счетчик полуходов (один ход белыми/чёрными)

MoveCount:=MoveCount+1;

//Добавление в историю ходов

AddHistoryString(MoveCount,Pos.GetTurn,ManualMove);

//Ход

Pos.MakeMove(ManualMove);

//Замена очереди хода

Pos.ChangeTurn;

// Вывод надписи об очереди хода

if Pos.GetTurn=White then

TurnLabel.Caption:='White'

else

TurnLabel.Caption:='Black';

//Проверка на выигрыш, если да, то вывод сообщении об этом

if Pos.GetWinner<>Space then WinMessage;

end

//Если ход невозможен, то вывод сообщения об этом

else

begin

MessageBox(0,'Невозможный ход','Внимание!!!',MB\_OK);

end

end

else

begin

{Первый щелчек}

if Pos.Get(HNum(X-1),Y)=Pos.GetTurn then

begin

IsStartSelected:=True;

ManualMove.MoveFrom.X:=HNum(X-1);

ManualMove.MoveFrom.Y:=Y;

end;

end;

DrawGrid.Refresh;

end;

//Поворот доски

procedure TForm1.RotateClick(Sender: TObject);

begin

IsFlipped:=not IsFlipped;

DrawGrid.Refresh;

LabelLayout;

end;

//Автоход/ход компьютера

procedure TForm1.CompMoveButtonClick(Sender: TObject);

var

M: TMove;

D: Cardinal;

Hour,Min,Sec,MSec: Word;

T1,T2:TDateTime;

begin

//Отключение пользовательского интерфейса

TButton(Sender).Enabled:=False;

DrawGrid.Enabled:=False;

Start1.Enabled:=False;

Start2.Enabled:=False;

btnStart3.Enabled:=False;

Rotate.Enabled:=False;

DepthComboBox.Enabled:=False;

BreakButton.Enabled:=True;

//Определение глубины просмотра

If DepthComboBox.ItemIndex<>0 then

D:=DepthComboBox.ItemIndex

else

D:=CurrentDepth;

//Выбор лучшего хода

T1:=Time;

M:=Pos.GetBestMove(D,ProgressHandler);

T2:=Time;

if not((M.MoveFrom.X=M.MoveTo.X) and (M.MoveFrom.Y=M.MoveTo.Y)) then

begin

{Расчет хода был закончен, не был прерван.

Выбор глубины просмотра для следующего хода (при автоходе)}

If DepthComboBox.ItemIndex=0 then

begin

DecodeTime(T2-T1,Hour,Min,Sec,MSec);

if(Min\*60000+Sec\*1000+MSec)>20000 then

begin

if CurrentDepth<>1 then CurrentDepth:=CurrentDepth-1;

end

else

begin

if (Min\*60000+Sec\*1000+MSec)\*LastCount<20000 then CurrentDepth:=CurrentDepth+1;

end;

end;

//Счетчик полуходов

MoveCount:=MoveCount+1;

//Запись в историю

AddHistoryString(MoveCount,Pos.GetTurn,M);

Pos.MakeMove(M);

DrawGrid.Refresh;

//Выводим надписи об очереди хода

Pos.ChangeTurn;

if Pos.GetTurn=White then

TurnLabel.Caption:='White'

else

TurnLabel.Caption:='Black';

end;

//Включение пользовательского интерфейса

TButton(Sender).Enabled:=True;

DrawGrid.Enabled:=True;

Start1.Enabled:=True;

Start2.Enabled:=True;

btnStart3.Enabled:=True;

Rotate.Enabled:=True;

DepthComboBox.Enabled:=True;

BreakButton.Enabled:=False;

//Сброс полосы загрузки

ProgressBar.Position:=0;

//Проверка на выигрыш, если да, то вывод сообщении об этом

if Pos.GetWinner<>Space then WinMessage;

end;

//Кнопка выхода

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);

begin

Pos.Break;

end;

//Задание игровой доски/поля

Procedure TForm1.LabelLayout;

const

SP=3;

CW=5;

begin

//Горизонтальные обозначения

LabelA.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelA.Left:=DrawGrid.Left+0\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelA.Caption:='h' else LabelA.Caption:='a';

LabelB.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelB.Left:=DrawGrid.Left+1\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelB.Caption:='g' else LabelB.Caption:='b';

LabelC.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelC.Left:=DrawGrid.Left+2\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelC.Caption:='f' else LabelC.Caption:='c';

LabelD.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelD.Left:=DrawGrid.Left+3\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelD.Caption:='e' else LabelD.Caption:='d';

LabelE.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelE.Left:=DrawGrid.Left+4\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelE.Caption:='d' else LabelE.Caption:='e';

LabelF.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelF.Left:=DrawGrid.Left+5\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelF.Caption:='c' else LabelF.Caption:='f';

LabelG.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelG.Left:=DrawGrid.Left+6\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelG.Caption:='b' else LabelG.Caption:='g';

LabelH.Top:=DrawGrid.Top+DrawGrid.Height+SP;

LabelH.Left:=DrawGrid.Left+7\*(DrawGrid.DefaultColWidth + DrawGrid.GridLineWidth) + DrawGrid.DefaultColWidth div 2;

if isFlipped then LabelH.Caption:='a' else LabelH.Caption:='h';

//Вертикальные обозначения

Label1.Top:=DrawGrid.Top+7\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label1.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label1.Caption:='8' else Label1.Caption:='1';

Label2.Top:=DrawGrid.Top+6\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label2.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label2.Caption:='7' else Label2.Caption:='2'; Label3.Top:=DrawGrid.Top+5\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label3.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label3.Caption:='6' else Label3.Caption:='3';

Label4.Top:=DrawGrid.Top+4\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label4.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label4.Caption:='5' else Label4.Caption:='4';

Label5.Top:=DrawGrid.Top+3\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label5.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label5.Caption:='4' else Label5.Caption:='5'; Label6.Top:=DrawGrid.Top+2\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label6.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label6.Caption:='3' else Label6.Caption:='6';

Label7.Top:=DrawGrid.Top+1\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label7.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label7.Caption:='2' else Label7.Caption:='7';

Label8.Top:=DrawGrid.Top+0\*(DrawGrid.DefaultRowHeight+DrawGrid.GridLineWidth)+(DrawGrid.DefaultRowHeight div 2)-CW;

Label8.Left:=DrawGrid.Left+DrawGrid.Width+SP;

if isFlipped then Label8.Caption:='1' else Label8.Caption:='8';

end;

//Кнопка остоновки автохода

procedure TForm1.BreakButtonClick(Sender: TObject);

begin

Pos.Break;

end;

//Кнопка выбора первой формы поля

procedure TForm1.Start1Click(Sender: TObject);

begin

if MoveCount<>0 then

begin

if MessageBox(Form1.Handle,'Вы уверены, что хотите прервать игру?','Предупреждение',MB\_YESNO)=IDNO then

begin

exit;

end

end;

Pos.SetInitialPosition(StartPosition1);

//Сброс игры в начало

Pos.RestartGame;

MoveCount:=0;

HistoryListBox.Items.Clear;

Pos.ChangeTurn(White);

TurnLabel.Caption:='White';

IsFlipped:=False;

IsStartSelected:=False;

LastCount:=15;

CurrentDepth:=4;

LabelLayout;

DrawGrid.Refresh;

end;

//Кнопка выбора второй формы поля

procedure TForm1.Start2Click(Sender: TObject);

begin

if MoveCount<>0 then

begin

if MessageBox(Form1.Handle,'Вы уверены, что хотите прервать игру?','Предупреждение',MB\_YESNO)=IDNO then

begin

exit;

end

end;

Pos.SetInitialPosition(StartPosition2);

//Сброс игры в начало

Pos.RestartGame;

MoveCount:=0;

HistoryListBox.Items.Clear;

Pos.ChangeTurn(White);

TurnLabel.Caption:='White';

IsFlipped:=False;

IsStartSelected:=False;

LastCount:=15;

CurrentDepth:=4;

LabelLayout;

DrawGrid.Refresh;

end;

//Кнопка выбора третьей формы поля

procedure TForm1.btnStart3Click(Sender: TObject);

begin

if MoveCount<>0 then

begin

if MessageBox(Form1.Handle,'Вы уверены, что хотите прервать игру?','Предупреждение',MB\_YESNO)=IDNO then

begin

exit;

end

end;

Pos.SetInitialPosition(StartPosition3);

//Сброс игры в начало

Pos.RestartGame;

MoveCount:=0;

HistoryListBox.Items.Clear;

Pos.ChangeTurn(White);

TurnLabel.Caption:='White';

IsFlipped:=False;

IsStartSelected:=False;

LastCount:=15;

CurrentDepth:=4;

LabelLayout;

DrawGrid.Refresh;

end;

//Вывод собщения о победе

Procedure TForm1.WinMessage;

var

Winner: State;

S: String;

begin

Winner:=Pos.GetWinner;

case winner of

white:

S:='Белые победили!!!';

black:

S:='Черные победили!!!';

else

S:='Ничья!!!';

end;

MessageBox(Form1.Handle,PChar(S),'Обьявление победителя',MB\_OK);

//Сброс игры в начало

Pos.RestartGame;

MoveCount:=0;

HistoryListBox.Items.Clear;

Pos.ChangeTurn(White);

TurnLabel.Caption:='White';

IsFlipped:=False;

IsStartSelected:=False;

LastCount:=15;

CurrentDepth:=4;

LabelLayout;

DrawGrid.Refresh;

end;

//Обработка кнопок меню

//Кнопка "Выход"

procedure TForm1.N3Click(Sender: TObject);

begin

Close;

end;

//Кнопка выбора первого поля

procedure TForm1.N11Click(Sender: TObject);

begin

Start1Click(Sender);

end;

//Кнопка выбора второго поля

procedure TForm1.N21Click(Sender: TObject);

begin

Start2Click(Sender);

end;

//Кнопка выбора третьего поля

procedure TForm1.N31Click(Sender: TObject);

begin

btnStart3Click(Sender);

end;

//Кнопка "Помощь"

procedure TForm1.N2Click(Sender: TObject);

begin

Form1.Visible:=False;

Form2.Visible:=True;

end;

end.

unit Unit2;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, StdCtrls;

type

TForm2 = class(TForm)

img1: TImage;

btn1: TButton;

procedure btn1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form2: TForm2;

implementation

uses MFrame;

{$R \*.dfm}

procedure TForm2.btn1Click(Sender: TObject);

begin

Form2.Visible:=False;

Form1.Visible:=True;

end;

end.

{=====================================================================

Модуль описывает текущее положение на доске,

позволяет определять лучший ход

(класс TPosition)

======================================================================}

unit Position;

interface

uses forms, sysutils;

Type

{Процедура обратного вызова}

CallBackPosition=Procedure (Current,Amount:Integer);

{Описание каждой клетки}

State=(Black, White, Space);

{Нумерация по горизонтали и вертикали}

HNum=(a,b,c,d,e,f,g,h);

VNum=1..8;

{Позиция шашки}

TPlace=record

X: HNum;

Y: VNum;

end;

{Описание хода}

TMove=record

MoveFrom: TPlace;

MoveTo: TPlace;

end;

{Описание списка возможных ходов}

PListEntry=^TMoveList;

TMoveList=record

Move: TMove;

Next: PListEntry;

Prev: PListEntry;

end;

{Описание доски}

TBoard= Array[HNum, VNum] of State;

{Описание текущей позиции}

TPosition=class(TObject)

private

Board : TBoard; {Доска}

InitialPosition: TBoard; {Начальная позиция}

Turn: State; {Очередь хода}

Function GetMoveList():PListEntry; {Получить полный список возможных ходов в данной позиции}

Procedure DelMoveList(List: PListEntry); {Удалить список из памяти}

Function GoalFunc():Integer; {Целевая функция}

Function GoalFunc2():Integer;

function \_GetBestMove(Var BestMove: TMove; Depth: Cardinal; Progress:CallBackPosition=nil):Integer;

function \_GetBestMove2(Var BestMove: TMove; Depth: Cardinal; Progress:CallBackPosition=nil):Integer;

public

{Конструктор, деструктор}

constructor Create; overload;

constructor Create(Pattern: TPosition); overload;

destructor Destroy; override;

{Получить состояние клетки}

Function Get(X: HNum; Y: VNum):State;

{Получить очередь хода}

Function GetTurn: State;

{Получение копии позиции}

Procedure CopyFrom(Pattern: TPosition);

{Установка начальной позиции (необходимо для целевой функции) }

Procedure SetInitialPosition(Pos: TBoard);

{Меняет очередность хода}

Procedure ChangeTurn(T:State=Space);

{Сброс позиции в начальную}

Procedure RestartGame;

{Установить фигуру}

procedure SetPiece(X:HNum; Y:VNum; Color:State); overload;

procedure SetPiece(Pos:TPlace; Color:State); overload;

{Удалить фигуру}

procedure RemovePiece(X:HNum; Y:VNum); overload;

procedure RemovePiece(Pos:TPlace); overload;

{Произвести перемещение фигуры}

procedure MakeMove(X1:HNum; Y1:VNum; X2:HNum; Y2:VNum); overload;

procedure MakeMove(StartPos,EndPos: TPlace); overload;

procedure MakeMove(M: TMove); overload;

{Проверить возможность хода}

function IsMovePossible(X1:HNum; Y1:VNum; X2:HNum; Y2:VNum):Boolean; overload;

function IsMovePossible(StartPos,EndPos: TPlace):Boolean; overload;

function IsMovePossible(M: TMove):Boolean; overload;

{Определить лучший возможный ход}

function GetBestMove(Depth: Cardinal=5; Progress:CallBackPosition=nil): TMove;

{Определяет не победил ли кто случайно...}

function GetWinner():State;

{Прерывает поиск лучшего хода}

Procedure Break;

end;

implementation

var

BreakFlag: Boolean;

function max(a,b:cardinal):cardinal;

begin

if a>b then max:=a else max:=b;

end;

function min(a,b:cardinal):cardinal;

begin

if a>b then min:=b else min:=a;

end;

{Конструктор/деструктор}

{Подготовка к работе}

constructor TPosition.Create;

begin

inherited Create;

Turn:=White;

BreakFlag:=False;

end;

constructor TPosition.Create(Pattern: TPosition);

begin

inherited Create;

CopyFrom(Pattern);

BreakFlag:=False;

end;

{Удаление из динамической памяти}

destructor TPosition.Destroy;

begin

inherited Destroy;

end;

{Возвращает состояние заданной клеточки}

Function TPosition.Get(X: HNum; Y: VNum):State;

begin

Get:=Board[X,Y];

end;

{Возвращает состояние заданной клеточки}

Function TPosition.GetTurn:State;

begin

GetTurn:=Turn;

end;

{Стать точной копией объекта}

Procedure TPosition.CopyFrom(Pattern: TPosition);

begin

Board:=Pattern.Board;

InitialPosition:=Pattern.InitialPosition;

turn:=Pattern.Turn;

end;

{Установка начальной позиции (необходимо для целевой функции)}

Procedure TPosition.SetInitialPosition(Pos: TBoard);

var

i: VNum;

j: HNum;

begin

for i:= Low(i) to High(i) do

begin

for j:=Low(j) to High(j) do

begin

InitialPosition[j,i]:=Pos[j,i];

end

end

end;

{Процедура передает очередь хода противнику или указ. цвету}

Procedure TPosition.ChangeTurn(T:State);

begin

if T=Space then

begin

if turn=White then

turn:=Black

else

turn:=White;

end

else

begin

turn:=T;

end;

end;

{Сброс позиции в начальную}

Procedure TPosition.RestartGame;

var

i: VNum;

j: HNum;

begin

for i:= Low(i) to High(i) do

begin

for j:=Low(j) to High(j) do

begin

Board[j,i]:=InitialPosition[j,i];

end

end

end;

{Функция возвращает полный список возможных ходов}

Function TPosition.GetMoveList():PListEntry;

Type

TDirection=(right,up,left,down);

TTreeEntry=^TMovesTree;

TMovesTree=record

Disposition: TPlace;

Root: TTreeEntry;

SimpleStep: array [TDirection] of TTreeEntry;

JumpStep: array [TDirection] of TTreeEntry;

end;

{Процедура строит дерево возможных ходов для шашки}

procedure CreateTree(var PHead: TTreeEntry; Parent:TTreeEntry; X: HNum; Y: VNum; IsSimple: Boolean=True);

{Процедура инициализирует узел дерева заданными значениями}

Procedure Initialize(var PHead: TTreeEntry; Parent: TTreeEntry; X: HNum; Y: VNum);

var

i: TDirection;

begin

new(PHead);

PHead.Disposition.X:=X;

PHead.Disposition.Y:=Y;

PHead.Root:=Parent;

for i:=Right to down do PHead^.SimpleStep[i]:=nil;

for i:=Right to down do PHead^.JumpStep[i]:=nil;

end;

{Функция проверяет не были ли мы здесь раньше}

function IsFirstTime(PNode: TTreeEntry; X: HNum; Y: VNum):Boolean;

var

PCurNode: TTreeEntry;

begin

PCurNode:=PNode^.Root;

IsFirstTime:=True;

While PCurNode<> nil do

begin

if (PCurNode^.Disposition.X=X) and (PCurNode^.Disposition.Y=Y) then

begin

IsFirstTime:=False;

exit;

end;

PCurNode:=PCurNode^.Root;

end;

end;

{Тело процедуры Create Tree}

begin

{Инициализация }

Initialize(PHead,Parent,X,Y);

{Просмотр возможных продолжений хода}

if IsSimple then

begin

{Простые шаги обрабатываются только для первого шага

Рекурсия не вызывается, а просто создаются конечные узлы}

if (X<>High(X)) and (Board[Succ(X),Y] = Space) then

Initialize(PHead^.SimpleStep[Right],PHead,Succ(X),Y);

if (X<>Low(X)) and (Board[Pred(X),Y] = Space) then

Initialize(PHead^.SimpleStep[Left],PHead,Pred(X),Y);

if (Y<>High(Y)) and (Board[X,Succ(Y)] = Space) then

Initialize(PHead^.SimpleStep[Up], PHead, X, Succ(Y));

if (Y<>Low(Y)) and (Board[X,Pred(Y)] = Space) then

Initialize(PHead^.SimpleStep[Down],PHead,X,Pred(Y));

end;

{Обрабатывается для любого шага - прыжки

Рекурсивно вызывается CreateTree }

{Вправо}

if (X<>High(X)) and (Succ(X)<>High(x)) and (Board[Succ(X),Y]<>Space) and (Board[Succ(Succ(X)),Y]=Space) and IsFirstTime(PHead,Succ(Succ(X)),Y) then

CreateTree(PHead^.JumpStep[Right],PHead,Succ(Succ(X)),Y,False);

{Влево}

if (X<>Low(X)) and (Pred(X)<>Low(x)) and (Board[Pred(X),Y]<>Space) and (Board[Pred(Pred(X)),Y]=Space) and IsFirstTime(PHead,Pred(Pred(X)),Y) then

CreateTree(PHead^.JumpStep[Left],PHead,Pred(Pred(X)),Y,False);

{Вверх}

if (Y<>High(Y)) and (Succ(Y)<>High(Y)) and (Board[X,Succ(Y)]<>Space) and (Board[X,Succ(Succ(Y))]=Space) and IsFirstTime(PHead,X,Succ(Succ(Y))) then

CreateTree(PHead^.JumpStep[Up],PHead,X,Succ(Succ(Y)),False);

{Вниз}

if (Y<>Low(Y)) and (Pred(Y)<>Low(Y)) and (Board[X,Pred(Y)]<>Space) and (Board[X,Pred(Pred(Y))]=Space) and IsFirstTime(PHead,X,Pred(Pred(Y))) then

CreateTree(PHead^.JumpStep[Down],PHead,X,Pred(Pred(Y)),False);

end;

{Процедура освобождает память, выделенную дереву.}

procedure DeleteTree(Head: TTreeEntry);

var

i: TDirection;

begin

{Вначале удалим подчиненные узлы}

for i:=Right to Down do

begin

if Head^.SimpleStep[i]<>nil then DeleteTree(Head^.SimpleStep[i]);

if Head^.JumpStep[i]<>nil then DeleteTree(Head^.JumpStep[i]);

end;

{А теперь удалим сам узел}

dispose(Head);

end;

{Процедура на основание дерева ходов добавляет элементы в список ходов}

procedure TreeToList(Tree: TTreeEntry; Node: TTreeEntry; var List: PListEntry);

var

i: TDirection;

temp: PListEntry;

IsFirstTime: Boolean;

begin

{Проверить не является ли корнем, а если является, то пропустить добавление}

if Tree<>Node then

begin

{Проверить есть ли текущий ход в списке и если надо - добавить}

IsFirstTime:=True;

temp:=List;

while (temp<>nil) and (temp^.Move.MoveFrom.X=Tree^.Disposition.X) and (temp.Move.MoveFrom.Y=Tree.Disposition.Y) do

begin

if (Node.Disposition.X=temp.Move.MoveTo.X) and (Node.Disposition.Y=temp.Move.MoveTo.Y) then

begin

IsFirstTime:=False;

end;

temp:=temp.Next;

end;

if IsFirstTime then

begin

new(temp); {Создать новый элемент}

temp.Move.MoveFrom.X:=Tree.Disposition.X; {Подготовить к работе}

temp.Move.MoveFrom.Y:=Tree.Disposition.Y;

temp.Move.MoveTo.X:=Node.Disposition.X;

temp.Move.MoveTo.Y:=Node.Disposition.Y;

if List<> nil then List.Prev:=temp; {И внести в список первым}

temp.Next:=List;

temp.Prev:=nil;

list:=temp;

end;

end;

{Рекрусивно вызвать себя для всех "листьев" дерева не равных нулю}

for i:=Right to Down do

begin

if Node^.SimpleStep[i]<>nil then TreeToList(Tree,Node^.SimpleStep[i],List);

if Node^.JumpStep[i]<>nil then TreeToList(Tree,Node^.JumpStep[i],List);

end;

end;

{Локальные переменные}

var

PList: PListEntry; {Указатель на результирующий список}

PTree: TTreeEntry; {Указатель на "голову" дерева}

X: HNum;

Y: VNum;

begin

PList:=nil;

{Для каждой шашки текущего цвета}

for X:=a to h do

begin

for Y:=1 to 8 do

begin

if Board[X,Y]=Turn then

begin

{Построить дерево}

CreateTree(PTree,nil,X,Y);

{Добавить в список}

TreeToList(PTree,PTree,PList);

{Уничтожить дерево}

DeleteTree(PTree);

end;

end;

end;

{вернуть полученный список}

GetMoveList:=PList;

end;

{Процедура удаляет список из памяти}

Procedure TPosition.DelMoveList(List: PListEntry);

var

temp: PListEntry;

begin

while List<>nil do

begin

temp:=List.Next;

Dispose(List);

List:=temp;

end

end;

{Возвращает число, пропорциональное качеству позиции}

Function TPosition.GoalFunc():Integer;

var

i: HNum;

j: VNum;

WhiteR: Integer;

BlackR: Integer;

begin

{Подсчет расстояний}

WhiteR:=0;

BlackR:=0;

for i:=low(i) to high(i) do

begin

for j:=low(j) to high(j) do

begin

if board[i,j]=white then

begin

{Сумма расстояний до угла черных}

WhiteR:=WhiteR+integer(16-min(ord(i),7)-min(ord(j),7));

end;

if board[i,j]=black then

begin

{Сумма расстояний до угла белых}

BlackR:=BlackR+integer(max(ord(i),2)+max(ord(j),2));

end;

end;

end;

if turn=black then

result:=(WhiteR-BlackR)\*10+Random(10)

else

result:=(BlackR-WhiteR)\*10+Random(10);

end;

{Целевая функция для финальной части партии.

Возвращает число, пропорциональное качеству позиции}

Function TPosition.GoalFunc2():Integer;

var

i: HNum;

j: VNum;

WhiteR: Integer;

BlackR: Integer;

WhiteCount: Integer;

BlackCount: Integer;

begin

{Подсчет расстояний}

WhiteR:=0;

BlackR:=0;

WhiteCount:=0;

BlackCount:=0;

for i:=low(i) to high(i) do

begin

for j:=low(j) to high(j) do

begin

if board[i,j]=white then

begin

{Взвешенная Сумма расстояний до угла черных}

if (i<e) or (j<3) then

WhiteR:=WhiteR+(16-ord(i)-ord(j))\*5

else

WhiteR:=WhiteR+16-ord(i)-ord(j);

{Количество достигших своего места}

if InitialPosition[i,j]=Black then

begin

WhiteCount:=WhiteCount+1;

end;

end;

if board[i,j]=black then

begin

{Взвешенная Сумма расстояний до угла белых}

if (i>f) or (j>6) then

BlackR:=BlackR+(ord(i)+ord(j))\*5

else

BlackR:=BlackR+ord(i)+ord(j);

{Количество достигших своего места}

if InitialPosition[i,j]=White then

begin

BlackCount:=BlackCount+1;

end;

end;

end;

end;

if turn=black then

begin

result:=+BlackCount\*50-BlackR\*10+Random(10);

end

else

begin

result:=WhiteCount\*50-WhiteR\*10+Random(10);

end;

end;

{Установить шашку в заданную позицию}

procedure TPosition.SetPiece(X:HNum; Y:VNum; Color:State);

begin

Board[X,Y]:=Color;

end;

procedure TPosition.SetPiece(Pos:TPlace; Color:State);

begin

SetPiece(Pos.X,Pos.Y,Color);

end;

{Удалить шашку из заданной позиции}

procedure TPosition.RemovePiece(X:HNum; Y:VNum);

begin

Board[X,Y]:=Space;

end;

procedure TPosition.RemovePiece(Pos:TPlace);

begin

RemovePiece(Pos.X,Pos.Y);

end;

{Произвести ход. Проверка возможности не производится}

procedure TPosition.MakeMove(X1:HNum; Y1:VNum; X2:HNum; Y2:VNum);

begin

Board[X2,Y2]:=Board[X1,Y1];

Board[X1,Y1]:=Space;

end;

procedure TPosition.MakeMove(StartPos,EndPos: TPlace);

begin

MakeMove(StartPos.X,StartPos.Y,EndPos.X,EndPos.Y);

end;

procedure TPosition.MakeMove(M: TMove);

begin

MakeMove(M.MoveFrom.X, M.MoveFrom.Y, M.MoveTo.X, M.MoveTo.Y);

end;

{Проверка допустимости хода}

function TPosition.IsMovePossible(X1:HNum; Y1:VNum; X2:HNum; Y2:VNum):Boolean;

var

list: PListEntry;

CurMove: PListEntry;

begin

list:=GetMoveList();

CurMove:=list;

IsMovePossible:=False;

While CurMove<>nil do

begin

if (CurMove.Move.MoveFrom.X = X1) and (CurMove.Move.MoveFrom.Y = Y1) and (CurMove.Move.MoveTo.X = X2) and (CurMove.Move.MoveTo.Y = Y2) then IsMovePossible:=True;

CurMove:=CurMove.Next;

end;

DelMoveList(list);

end;

function TPosition.IsMovePossible(StartPos,EndPos: TPlace):Boolean;

begin

IsMovePossible:=IsMovePossible(StartPos.X, StartPos.Y, EndPos.X, EndPos.Y);

end;

function TPosition.IsMovePossible(M: TMove):Boolean;

begin

IsMovePossible:=IsMovePossible(M.MoveFrom, M.MoveTo);

end;

{Анализировать позицию и выбрать лучший ход}

function TPosition.GetBestMove(Depth: Cardinal; Progress:CallBackPosition): TMove;

const

Switch=50;

var

i:HNum;

J:VNum;

Count: Integer;

WhiteR,BlackR:Integer;

WhiteSp,BlackSp:Integer;

begin

{Сбросить флаг остановки}

BreakFlag:=False;

{Посчитать сумму расстояний до целевого угла}

WhiteR:=0;

BlackR:=0;

WhiteSp:=0;

BlackSp:=0;

Count:=0;

for i:=low(i) to high(i) do

begin

for j:=low(j) to high(j) do

begin

if board[i,j]=white then

begin

WhiteR:=WhiteR+16-ord(i)-ord(j);

Count:=Count+1;

end;

if board[i,j]=black then

begin

BlackR:=BlackR+ord(i)+ord(j);

end;

if(InitialPosition[i,j]=White) and (Board[i,j]<>Black) then BlackSp:=BlackSp+1;

if(InitialPosition[i,j]=Black) and (Board[i,j]<>White) then WhiteSp:=WhiteSp+1;

end;

end;

{Выбрать алгоритм вычисления следующего хода}

if ((turn=White) and (WhiteR>Count\*5)) or ((turn=Black) and (BlackR>Count\*5)) then

begin

{Выбор хода по правилам первой половины партии}

\_GetBestMove(Result,Depth,Progress);

end

else

begin

{Выбор хода по правилам второй половины партии}

if((turn=white) and (WhiteSp<=3)) or ((turn=black) and (BlackSp<=3)) then

\_GetBestMove2(Result,max(Depth,4),Progress)

else

\_GetBestMove2(Result,Depth,Progress);

end;

{Если расчет был прерван, то вернуть ход на месте}

if BreakFlag then

begin

Result.MoveFrom.X:=a;

Result.MoveTo.X:=a;

Result.MoveFrom.Y:=1;

Result.MoveTo.Y:=1;

end;

end;

{Функция выбирает лучший ход для первой стадии партии}

function TPosition.\_GetBestMove(Var BestMove: TMove; Depth: Cardinal; Progress:CallBackPosition=nil):Integer;

var

List: PListEntry;

CurMove: PListEntry;

P: TPosition;

BestScore, Score: Integer;

Move: TMove;

MovesCount,MoveNum: Integer;

PLst: PListEntry;

begin

if depth<>0 then

begin

{Создать список возможных ходов}

List:=GetMoveList();

{Если понадобится - посчитаем позже}

MovesCount:=0;

MoveNum:=0;

{Создать объект Position}

P:=TPosition.Create();

{Создать копию текущей позиции}

P.CopyFrom(Self);

{Инициализация цикла }

CurMove:=List;

BestScore:=Low(BestScore);

{Для каждого возможного хода: }

while CurMove<>nil do

begin

{Произвести ход }

P.MakeMove(CurMove.Move);

{Передать ход сопернику}

P.ChangeTurn;

{Вызвать GetBestMove для полученной позиции}

Score:=-P.\_GetBestMove(Move,Depth-1);

{Oпределиться лучший ли ход был}

if Score>BestScore then

begin

BestScore:=Score;

BestMove:=CurMove.Move;

end;

{Вернуть ход назад}

P.MakeMove(CurMove.Move.MoveTo,CurMove.Move.MoveFrom);

P.ChangeTurn;

{Вернуть сведения о состоянии просмотра}

if @Progress<>nil then

begin

{Посчитать общее количество возможных ходов, если это не было сделано раньше}

if MovesCount=0 then

begin

PLst:=List;

while PLst.Next<>nil do

begin

MovesCount:=MovesCount+1;

PLst:=PLst.Next;

end

end;

{Инкремент текущего}

MoveNum:=MoveNum+1;

{Вернуть текущий и общее количество}

Progress(MoveNum,MovesCount);

end;

{Позволить сообщениям пройти}

Application.ProcessMessages;

{Проверить необходимость прервать вычисления}

if BreakFlag then

begin

\_GetBestMove:=BestScore;

exit;

end;

{Следующий ход}

CurMove:=CurMove.Next;

end;

{Удалить список возможных ходов}

DelMoveList(List);

{Освободить объект}

P.Destroy;

{Вернуть оценку лучшего хода}

\_GetBestMove:=BestScore;

end

else {Прошли дерево просмотра до depth}

begin

{вызвать целевую функцию}

{вернуть значение целевой функции}

\_GetBestMove:=GoalFunc();

end;

end;

{Функция выбирает лучший ход для заключительной стадии партии}

function TPosition.\_GetBestMove2(Var BestMove: TMove; Depth: Cardinal; Progress:CallBackPosition=nil):Integer;

var

List: PListEntry;

CurMove: PListEntry;

P: TPosition;

BestScore, Score: Integer;

Move: TMove;

MovesCount,MoveNum: Integer;

PLst: PListEntry;

begin

{Проверить не конечная ли позиция}

if GetWinner=turn then

begin

{Нужно только при попытке хода из конечной позиции}

BestMove.MoveFrom.X:=a;

BestMove.MoveTo.X:=a;

BestMove.MoveFrom.Y:=1;

BestMove.MoveTo.Y:=1;

\_GetBestMove2:=100000+Depth;

exit;

end;

{Позиция не конечная, значит считаем дальше}

if depth<>0 then

begin

{Создать список возможных ходов}

List:=GetMoveList();

{Если понадобится - посчитаем позже}

MovesCount:=0;

MoveNum:=0;

{Создать объект Position}

P:=TPosition.Create();

{Создать копию текущей позиции}

P.CopyFrom(Self);

{Инициализация цикла }

CurMove:=List;

BestScore:=Low(BestScore);

{Для каждого возможного хода: }

while CurMove<>nil do

begin

{Произвести ход }

P.MakeMove(CurMove.Move);

{Вызвать GetBestMove для полученной позиции}

Score:=P.\_GetBestMove2(Move,Depth-1);

{Oпределиться лучший ли ход был}

if Score>BestScore then

begin

BestScore:=Score;

BestMove:=CurMove.Move;

end;

{Вернуть ход назад}

P.MakeMove(CurMove.Move.MoveTo,CurMove.Move.MoveFrom);

{Вернуть сведения о состоянии просмотра}

if @Progress<>nil then

begin

{Посчитать общее количество возможных ходов, если это не было сделано раньше}

if MovesCount=0 then

begin

PLst:=List;

while PLst.Next<>nil do

begin

MovesCount:=MovesCount+1;

PLst:=PLst.Next;

end

end;

{Инкремент текущего}

MoveNum:=MoveNum+1;

{Вернуть текущий и общее количество}

Progress(MoveNum,MovesCount);

end;

{Позволить сообщениям пройти}

Application.ProcessMessages;

{Проверить необходимость прервать вычисления}

if BreakFlag then

begin

\_GetBestMove2:=BestScore;

exit;

end;

{Следующий ход}

CurMove:=CurMove.Next;

end;

{Удалить список возможных ходов}

DelMoveList(List);

{Освободить объект}

P.Destroy;

{Вернуть оценку лучшего хода}

\_GetBestMove2:=BestScore;

end

else {Прошли дерево просмотра до depth}

begin

{вызвать целевую функцию}

{вернуть значение целевой функции}

\_GetBestMove2:=GoalFunc2();

end;

end;

{Установить флаг остановки}

procedure TPosition.break;

begin

BreakFlag:=True;

end;

{Функция проверяет не победил ли кто, и если да, то возвращает

его цвет. В противном случае - space}

function TPosition.GetWinner():State;

label

1,2,3,4;

var

NotWhite: Boolean;

NotBlack: Boolean;

i: HNum;

j: VNum;

begin

NotWhite:=False;

NotBlack:=False;

{Для черных}

for i:=Low(i) to High(i) do

begin

if InitialPosition[i,1]<>white then goto 2;

for j:=Low(j) to High(j) do

begin

if InitialPosition[i,j]=white then

begin

if Board[i,j]<>black then

begin

NotBlack:=True;

goto 2;

end;

end

else

begin

goto 1; {Cтроку досмотрели}

end

end;

1:{Просто прервали внутренний цикл};

end;

2:{Для белых};

for i:=High(i) downto Low(i) do

begin

if InitialPosition[i,8]<>black then goto 4;

for j:=High(j) downto Low(j) do

begin

if InitialPosition[i,j]=Black then

begin

if Board[i,j]<>White then

begin

NotWhite:=True;

goto 4;

end;

end

else

begin

goto 3; {Cтроку досмотрели}

end

end;

3:{Просто прервали внутренний цикл};

end;

4:{Посмотрим что получили};

if NotWhite and NotBlack then

begin

GetWinner:=Space;

end

else

begin

if NotWhite then GetWinner:=Black else GetWinner:=White;

end;

end;

end.