Курсов Проект

По

Обектно Ориентирано програмиране

I – ва част

**Изготвил: Илиян Костов**

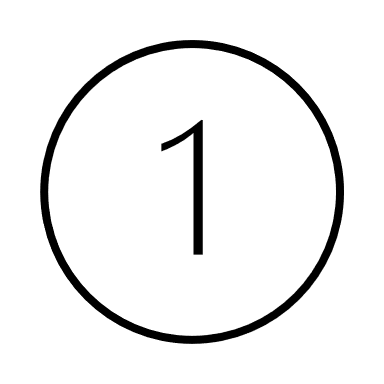
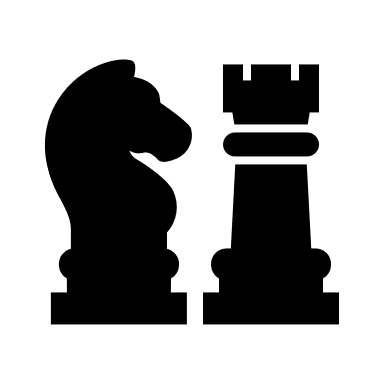
**Факултетен № : 19621609**

**Специалност: СИТ**

**Курс: 2**

**Група : 4**

**Подгрупа: б**



Задание : Шах



Проект 2

ШАХ

(изготвил: Александър Шумаков)

Задачата ви е да напишете програма, която позволява да се провежда игра на

шах между двама играчи.

За целта, създайте следните класове и интерфейси със съответните

функционалности:

Фигура

Фигура (Figure) : абстрактен клас, който ще се разширява от различните

видове фигури в играта:

● Пешка (Pawn) : може да се придвижва само едно поле напред (две ако

досега не се е движила); може да атакува фигури, които са на 1

квадратче по диагонал от нея

● Топ (Rook) : може да се придвижва хоризонтално или вертикално до

достъпните квадратчета и да атакува фигури, намиращи се по тях.

● Кон (Knight) : може да се придвижва Г-образно по дъската и да атакува

фигури, намиращи се на квадратчетата, на които може да стъпи.

● Офицер (Bishop) : може да се придвижва по диагонал и да атакува

фигури на съответните позиции.

● Царица (Queen) : може да се придвижва като Топ или като Офицер; да се

реализира чрез наследяване на тези 2 класа;

● Цар (King) : може да се придвижва и атакува само фигури само в

непосредствено съседство;

Всички фигури да притежават информация за собственика им и информация за

противниковите фигури, които са взели по време на играта (коя фигура, на коя

позиция, в кой ход).

Да се реализират следните функции:

● printStats - извежда информация за всички взети фигури

● move - приема като параметри координатите, на които искаме да

преместим фигурата, и връща t rue , ако фигурата може да се премести

на конкретна позиция, и false в противен случай.

Да се реализират нужните за тези имплементации помощни функции и да се

използват помощни променливи по собствена преценка.

Дъска

7

Дъска (Board): клас, който ще държи информацията за разположението на

фигурите по дъската в даден момент. Да се имплементира с нужната

функционалност по собствена преценка.

Игра

Игра (Game) : клас, който ще представлява клиентски интерфейс за играта.

Интерфейсът трябва поддържа следните команди, въведени от клавиатурата:

● move x1 y1 x2 y2 – при възможност мести фигурата от позиция (x1, y1)

на позиция (x2, y2);

● print - извежда на екрана игровата дъска

● stats x1 y1 – при възможност извежда информация за фигурата на

позиция (x1, y1);

● undo – връща играта в състоянието преди последния ход.

При невъзможност да се изпълни някоя от функционалностите, да се

извежда интуитивно съобщение, което пояснява на потребителя защо не

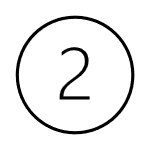
може да се изпълни дадената операция и да му позволи да въведе нова

команда.

Потребителите трябва да могат да въвеждат команди редувайки се и

командите трябва да отговарят по функционалност на съответният

потребител, например “move” не може да се изпълни на чужда фигура.



Кратко описание на програмата



**Предназначение:**

Програмата позволява да се провежда игра на шах между двама играчи .

Предоставени са фигури за шах за всеки от двата играча със съответният им цвят, на които играчите могат да взаимодействат с тях .

Вложени са 4 опции за двамата от играчите, правейки ги активни със съответният им ред :

1. Движение на фигура от позиция х,у към х,у
2. Извеждане на информация за дадена фигура от позиция х, у
3. Извеждане на всички взети фигури
4. Извеждане на игровата дъска в текущия момент
5. Връщане на движение на фигура

**Данни, с които работи програмата**

Програмата работи с входните данни от потребителя, главно от тип 2 цели числа ( integer ) - позиция х ( колона ) и у ( ред ) на дадената фигура , за която потребителя може да я придвижи или да се изведе информация за фигурата от дадена позиция, както и избирането на листа от опции.

Програмата също работи с единичен символ ( char ) за потвърждаване или отказ за връщане на движение на фигура , която потребителят може да е решил да се откаже от самото и движение.

Информацията за фигурите, които са взети от противниковия опонент се запазват в текстов файл .

**Възможни грешки и защита включена в програмата**

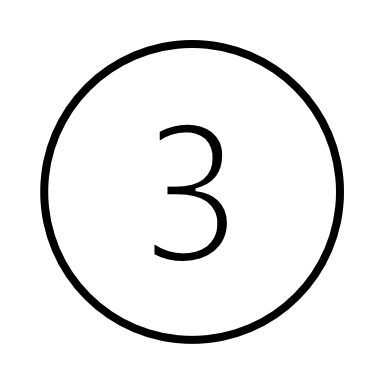
В програмата са вложени проверки при правилно въвеждане от потребителя данни , като :

* В случай , че потребителят иска да използва фигурите на опонента си
* Потребителят се опитва да премести фигура в невалидна позиция ( в зависимост от вида на фигурата )
* Потребителят е избрал позиция, на която позиция няма фигура.

**Възможности за развитие на програмата**

Програмата може допълнително да се разшири и да направи направлението на потребителя по – удобно, примери :

* Позициите да се заменят с единични символи ( A , B C .. )
* Нова опция , която запазва състоянието на играта, в случай че потребителите искат да излязат от играта и да я довършат в бъдеще време .
* Преместване на опцията за връщане на състоянието на играта в по – удобна за потребителя позиция .

Обща структура на програмата



Програмата е структурирана в един основен клас , който се наследява от множество други класове

**Клас Figure**

Основният абстрактен клас , на която всеки вид фигура трябва да го наследява .

В нея е вмъкната чисто виртуална функция за правилно движение на фигура и за връщане на иконата ( тип character) .

Функция за връщане на цвета на фигурата .

**Член функции**

virtual bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) =0; - чисто виртуална функция за правилно движение на фигура

const char GetColor(){return Color;} - функция , която връща цвета на фигурата

virtual const char getIcon()=0; - чисто виртуална функция , която връща иконата на фигурата

**Член променливи**

char Color; - цвят на фигурата

**Клас Pawn**

Клас , който наследява абстрактния клас Figure и имплементира функцията move по един или два квадрата напреде и възможност да атакува по диагонал.

В нея е вмъкната функция за връщане на иконата ( тип character) .

**Член функции**

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col); - функция за правилно движение на фигура

const char getIcon(); функция , която връща иконата на фигурата

**Член променливи**

bool flag; - променлива , за проверка дали пешката е извършила първото си движение

**Клас Rook**

Клас , който виртуално наследява абстрактния клас Figure и имплементира функцията move по само вертикал или хоризонтал .

В нея е вмъкната функция за връщане на иконата ( тип character) .

**Член функции**

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col); - функция за правилно движение на фигура

const char getIcon(); функция , която връща иконата на фигурата

**Клас Knight**

Клас , който наследява абстрактния клас Figure и имплементира функцията move по един хоризонтален ред и два вертикални или обратното .

В нея е вмъкната функция за връщане на иконата ( тип character) .

**Член функции**

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col); - функция за правилно движение на фигура

const char getIcon(); функция , която връща иконата на фигурата

**Клас Bishop**

Клас , който виртуално наследява абстрактния клас Figure и имплементира функцията move по вертикал .

В нея е вмъкната функция за връщане на иконата ( тип character) .

**Член функции**

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col); - функция за правилно движение на фигура

const char getIcon(); функция , която връща иконата на фигурата

**Клас Queen**

Клас , който наследява клас Knight и клас Bishop и имплементира функцията move по имплементацията на двата класа .

В нея е вмъкната функция за връщане на иконата ( тип character) .

**Член функции**

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col); - функция за правилно движение на фигура

const char getIcon(); функция , която връща иконата на фигурата

**Клас King**

Клас , който наследява абстрактния клас Figure и имплементира функцията move по един прав или вертикален ред .

В нея е вмъкната функция за връщане на иконата ( тип character) .

**Член функции**

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col); - функция за правилно движение на фигура

const char getIcon(); функция , която връща иконата на фигурата

**Клас Board**

Клас , който записва информацията на взетите фигури във файл и ги извежда в конзолата .

Функция за извеждане на информация за всички фигури в текущия момент на извикването и .

**Член функции**

static void WriteFileStats\_taken(int x,int y) – статична функция , която записва информацията на взетите фигури във файл

static void printStats\_taken() – статична функция , която извежда информацията на взетите фигури от файл.

static void printStats\_board(); - статична функция , която извежда информация за всичките фигури на дъската в текущия момент .

**Клас Game**

Клас , който представлява клиентски интерфейс за двамата играчи .

В нея е имплементирана функция , която инициализира ( разпределя памет ) всички фигури със съответният им вид и цвят и извежда игровата дъска , както и меню от опции за движение на фигура, извеждане на информация за дадена фигура , извеждане на взети фигури и извеждане на информация за всички фигури в текущия момент .

Функцията очаква въвеждане на опция от потребителя, за да обработи и изведе желаното от потребителя резултат.

Функция , за движение на фигура в паметта и промяна на състоянието на дъската .

Функция за проверка за край на игра .

**Член функции**

void print() – функция която извежда игровата дъска и меню от опции

static void stats(int x,int y) – статична функция , която извежда информация за фигура , въведена от потребителя позиция .

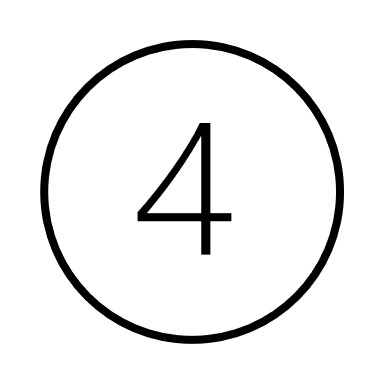
bool Game\_move(int curr\_x,int curr\_y,int dest\_x,int dest\_y) – функция , която проверява за успешност за правилно движение на фигурата от текущата позиция към бъдещата , при успешност връща истина и премества фигурата . Вътре във функцията е имплементирана опция за прекратяване на движението, в случай че потребителят се отказал да я извърши .

void Start\_game() – функция , която първо е стартирана от програмата и променя редът на играчите , както и взема данните от тях и извиква съответните функции за обработката им .

bool GameOver() – функция, която проверява всички възможни движения за играчите , за установяване за край на играта.

**Член функции**

char Player\_Turn; - променлива , която се използва за смяна на ред към следващ играч .

Тестови резултат

Картина, която съдържа текст

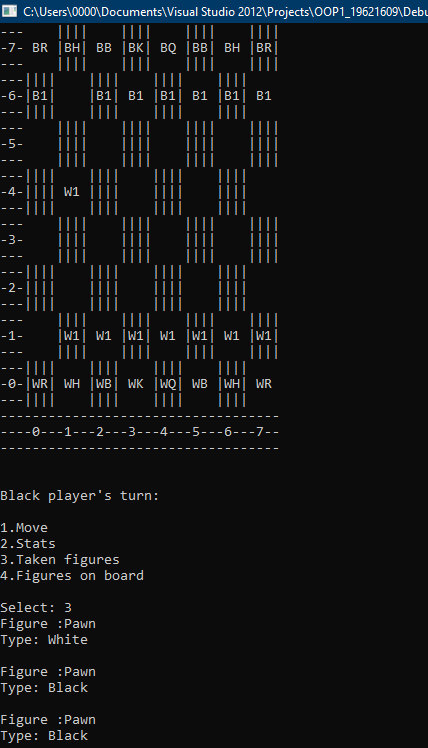
Описанието е генерирано автоматичноКартина, която съдържа текст

Описанието е генерирано автоматично

Картина, която съдържа текст

Описанието е генерирано автоматичноКартина, която съдържа текст

Описанието е генерирано автоматично



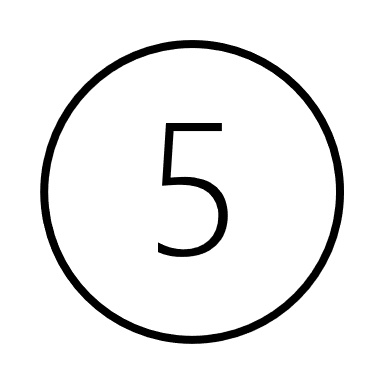
Картина, която съдържа текст

Описанието е генерирано автоматично

Картина, която съдържа текст

Описанието е генерирано автоматичноКартина, която съдържа текст

Описанието е генерирано автоматично

Сорс код



//Проект : Шах

// от : Илиян Костов

// Ф. н : 19621609

// Специалност: СИТ

// Група : 4

// Курс : 2

#include<iostream>

#include<stdio.h>

#include<string>

#include<typeinfo>

#include<exception>

#include<fstream>

#include<locale>

using namespace std;

const string file\_name\_taken = "Taken\_figures.txt";

const string file\_name\_board= "Board\_figures.txt";

static string temp="";

static string temp\_board="";

static bool flag=false;

static bool flag\_board=false;

class Figure{

char Color;

public:

Figure(){}

Figure(char argColor) : Color(argColor){}

virtual const char getIcon()=0;

const char GetColor(){return Color;}

virtual bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) =0;

};

static Figure\* board[8][8];

class Pawn: public Figure {

bool flag;

const char getIcon() { return '1'; }

public:

Pawn(char argColor) : Figure(argColor){flag=true;}

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) {

int moves=1;

if (flag) moves=2;

if (board[dest\_row][dest\_col] == 0 || board[dest\_row][dest\_col]->GetColor() != board[curr\_row][curr\_col]->GetColor() ) {

switch(GetColor()) {

bool flag3;

case 'W':

flag3=(dest\_row==curr\_row+1 && (dest\_col==curr\_col-1 || dest\_col==curr\_col+1) );

if (board[dest\_row][dest\_col] != 0) {

if ( flag3 && board[dest\_row][dest\_col]->GetColor() != GetColor()) return true;

}

else if (board[dest\_row][dest\_col]==0) {

if ((dest\_row==curr\_row+1 || dest\_row==curr\_row +moves ) && dest\_col==curr\_col) { flag=false;return true; }

}

break;

case 'B':

flag3=(dest\_row==curr\_row-1 && (dest\_col==curr\_col-1 || dest\_col==curr\_col+1) );

if (board[dest\_row][dest\_col]!=0) {

if ( flag3 && board[dest\_row][dest\_col]->GetColor() != GetColor()) return true;

}

else if (board[dest\_row][dest\_col]==0) {

if ((dest\_row==curr\_row-1 || dest\_row==curr\_row-moves)&& dest\_col==curr\_col) {flag=false;return true;}

}

break;

}

}

return false;

}

};

class Rook : virtual public Figure {

const char getIcon() { return 'R';}

public :

Rook(){}

Rook(char argColor) : Figure(argColor){}

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) {

if (board[dest\_row][dest\_col] == 0 || board[dest\_row][dest\_col]->GetColor() != GetColor()) {

if ( (dest\_row==curr\_row+1 || dest\_col==curr\_col+1) || (dest\_row==curr\_row-1 || dest\_col==curr\_col-1) && board[dest\_row][dest\_col]->GetColor() !=GetColor()) return true;

if (curr\_row==dest\_row) { // ако ще мърда вертикално

int dx=curr\_col<dest\_col ? 1: -1;

for(int j=curr\_col+dx;j!=dest\_col;j+=dx) {

if (board[curr\_row][j]!=0 ) return false;

}

}////////rows

else if (curr\_col== dest\_col) {

int dy=curr\_row<dest\_row ? 1:-1;

for(int j=curr\_row+dy;j!=dest\_row;j+=dy) {

if (board[j][curr\_col]!=0) return false;

}

}///////cols

if (curr\_row != dest\_row && curr\_col != dest\_col) return false;

return true;

}

else return false;

}

};

class Knight : public Figure {

const char getIcon() { return 'H';}

public :

Knight(char argColor) : Figure(argColor){}

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) {

//ако позицията е празна или позията е заета от противник да

if (board[dest\_row][dest\_col] == 0 || board[dest\_row][dest\_col]->GetColor() != GetColor()) {

if (dest\_row== curr\_row+1 || dest\_row==curr\_row-1) {

if (dest\_col==curr\_col+2 || dest\_col==curr\_col-2) return true;

}

else if (dest\_row == curr\_row+2 || dest\_row==curr\_row-2) {

if (dest\_col==curr\_col+1 || dest\_col==curr\_col-1) return true;

}

}

return false;

}

};

class Bishop : virtual public Figure {

const char getIcon() { return 'B';}

public :

Bishop(){}

Bishop(char argColor) : Figure(argColor){}

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) {

if (dest\_col - curr\_col == dest\_row - curr\_row || dest\_col - curr\_col == curr\_row - dest\_row ) {

int moved\_Row = dest\_row - curr\_row > 0 ? 1: -1;

int moved\_Col = dest\_col - curr\_col > 0 ? 1: -1;

int i,j;

for(i=curr\_row+moved\_Row,j=curr\_col+moved\_Col;i!=dest\_row,j!=dest\_col;i+=moved\_Row,j+=moved\_Col) {

if ( board[i][j]!=0 ) { return false;}

}

if (board[dest\_row][dest\_col]!=0) {if (board[dest\_row][dest\_col]->GetColor()!=GetColor()) return true;

else return false;

}

return true;

}

return false;

}

};

class Queen : public Rook ,public Bishop {

const char getIcon() { return 'Q';}

public :

Queen(char argColor) : Rook::Figure(argColor) {}

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) {

if (Rook::move(curr\_row,curr\_col,dest\_row,dest\_col) || Bishop::move(curr\_row,curr\_col,dest\_row,dest\_col))

return true;

return false;

}

};

class King : public Figure {

const char getIcon() { return 'K';}

public :

King(char argColor) : Figure(argColor){}

bool move(int curr\_row,int curr\_col,int dest\_row,int dest\_col) {

bool check\_row = dest\_row - curr\_row >=-1 && dest\_row - curr\_row <= 1;

bool check\_col= dest\_col - curr\_col >= -1 && dest\_col - curr\_col <= 1;

if ( check\_row && check\_col && board[dest\_row][dest\_col]!=0 ) { if (board[dest\_row][dest\_col]->GetColor() !=GetColor()) return true;}

else if (check\_row && check\_col && board[dest\_row][dest\_col]==0 ) return true;

return false;

}

};

class Board {

public:

static void WriteFileStats\_taken(int x,int y) {

ofstream ofile;

ofile.open(file\_name\_taken,ios::binary | ios::out | ios::app);

if (!ofile.is\_open()) cout<<"\nCan't open file!\n";

else {

ofile<<temp;

ofile<<"\n";

}

ofile.close();

}

static void printStats\_taken() {

ifstream ifile;

ifile.open(file\_name\_taken,ios::binary | ios::in);

if (!ifile.is\_open()) cout<<"\nCan't open file!\n";

else {

for(string i ;getline(ifile,i);) {

cout<<i<<endl;

}

}

ifile.close();

}

static void printStats\_board();

Board() {}

};

class Game {

char Player\_Turn;

public:

Game() {

//Syzdavane na dyska

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

board[i][j]=0;

//първо всичко с нули

}

}

//Cherni Figuri Black

for(int j=0;j<8;j++) {

board[6][j]=new Pawn('B');

}

board[7][0] = new Rook('B');

board[7][1] = new Knight('B');

board[7][2] = new Bishop('B');

board[7][3] = new King('B');

board[7][4] = new Queen('B');

board[7][5] = new Bishop('B');

board[7][6] = new Knight('B');

board[7][7] = new Rook('B');

//Beli Figuri White

for(int j=0;j<8;j++) {

board[1][j]=new Pawn('W');

}

board[0][0] = new Rook('W');

board[0][1] = new Knight('W');

board[0][2] = new Bishop('W');

board[0][3] = new King('W');

board[0][4] = new Queen('W');

board[0][5] = new Bishop('W');

board[0][6] = new Knight('W');

board[0][7] = new Rook('W');

}

void print() {

const int kiSquareWidth = 4;

const int kiSquareHeight = 3;

for (int i = 0; i < 8\*3; ++i) {

int Sq\_Row = i/3 ;

if (i % 3 == 1) {

cout << '-' << (char)('0' + 7 - Sq\_Row) << '-';

} else {

cout << "---";

}

for (int j = 0; j < 8\*4; ++j) {

int Sq\_Col = j/4;

if (((i % 3) == 1) && ((j % 4) == 1 || (j % 4) == 2) && board[7-Sq\_Row][Sq\_Col] != 0) {

if ((j % 4) == 1) {

cout << board[7-Sq\_Row][Sq\_Col]->GetColor();

} else {

cout << board[7-Sq\_Row][Sq\_Col]->getIcon();

}

} else {

if ((Sq\_Row + Sq\_Col) % 2 == 1) {

cout << '|';

} else {

cout << ' ';

}

}

}

cout << endl;

}

for (int i = 0; i < 3; ++i) {

if (i % 3 == 1) {

cout << "---";

for (int j = 0; j < 8\*4; ++j) {

int Sq\_Col = j/4;

if ((j % 4) == 1) {

cout << (Sq\_Col + 0);

} else {

cout << '-';

}

}

cout << endl;

} else {

for (int k = 1; k < 9\*4; ++k) {

cout << '-';

}

cout << endl;

}

}

}

static void stats(int x,int y) {

if (board[x][y] !=0) {

Figure\* getFigure=board[x][y];

string Name,Type="White";

switch(getFigure->getIcon()) {

case '1' : Name="Pawn";

break;

case 'R': Name="Rook";

break;

case 'H': Name="Knight";

break;

case 'B': Name="Bishop";

break;

case 'Q': Name="Queen";

break;

case 'K': Name="King";

break;

}

if (getFigure->GetColor() =='B') Type="Black";

if (flag) {temp="Figure :"+Name+"\nType: "+Type+"\n";flag=false;}

cout<<"Figure :"<<Name<<"\nType: "<<Type+"\n"<<"Position x: "<<x<<"\nPosition y: "<<y<<"\n";

}

else {cout<<"There is no figure in position: "<<x<<" , "<<y<<endl;}

}

bool Game\_move(int curr\_x,int curr\_y,int dest\_x,int dest\_y) {

if (board[curr\_x][curr\_y]!=0) {

if (board[curr\_x][curr\_y]->move(curr\_x,curr\_y,dest\_x,dest\_y)) {

char choice;

cout<<"\nCorrect movement!\n\n\nUndo ?\n(Y/N): ";

cin>>choice;

if (choice == 'Y' || choice=='y') return false;

else {

if (board[dest\_x][dest\_y]!=0) {flag=true;cout<<"Taken figure:\n\n"; stats(dest\_x,dest\_y);Board::WriteFileStats\_taken(dest\_x,dest\_y);}

board[dest\_x][dest\_y]=board[curr\_x][curr\_y];

board[curr\_x][curr\_y]=NULL;

return true;}

}

else{ cout<<"\nCannot make that move!"<<endl;

return false;}

}

else {cout<<"\nThere is no figure in position: "<<curr\_x<<" ,"<<curr\_y<<endl;return false;}

}

void Start\_game(){

Player\_Turn='W';

int x,y;

int x2,y2;

int choice;

string strtemp;

do {

system("cls");

print();

if (Player\_Turn == 'W') strtemp="White";

else strtemp="Black";

cout<<"\n\n"<<strtemp<<" player's turn: ";

cout<<"\n\n1.Move\n2.Stats\n3.Taken figures\n4.Figures on board\n\nSelect: ";

cin>>choice;

switch (choice) {

case 1:

cout<<"\nChoose position: ";

cout<<"\npos x: ";

cin>>x;

cout<<"\npos y: ";

cin>>y;

cout<<"\n\nMove to position:pos x: ";

cin>>x2;

cout<<"\npos y: ";

cin>>y2;

if (board[x][y] != 0 ) {

if (board[x][y]->GetColor() == Player\_Turn) {

if (Game\_move(x,y,x2,y2)) {

if (Player\_Turn=='W') Player\_Turn='B';

else Player\_Turn='W';

}

system("pause");

}else {cout<<"\nInvalid figure!\n";system("pause");}

}else {cout<<"\nThere is no figure in position: "<<x<<","<<y<<endl;system("pause");}

break;

case 2:

cout<<"\nChoose position: ";

cout<<"\npos x: ";

cin>>x;

cout<<"\npos y: ";

cin>>y;

stats(x,y);

system("pause");

break;

case 3:

if (temp!="")

Board::printStats\_taken();

else cout<<"\nThere are no taken figures!\n";

system("pause");

break;

case 4:

Board::printStats\_board();

system("pause");

break;

default:

cout<<"\nInvalid choice!";

system("pause");

break;

}

}while(GameOver());

}

bool GameOver(){

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

if (board[i][j] !=0) {

if (board[i][j]->GetColor() == Player\_Turn) {

for(int k=0;k<8;k++) {

for(int m=0;m<8;m++) {

if (board[i][j]->move(i,j,k,m)) return true;

}

}

}

}

}

}

cout<<"\nNo avalible moves left!\n";

if (Player\_Turn == 'B') cout<<"\nWhite player wins!\n";

else cout<<"\nBlack player wins!\n";

system("pause");

return false;

}

};

void Board::printStats\_board() {

cout<<"\n\nFigures in board :\n\n";

for(int i=0;i<8;i++) {

for(int j=0;j<8;j++) {

if (board[i][j]!=0){ Game::stats(i,j);cout<<"\n\n";}

}

}

}

int main()

{

setlocale(LC\_ALL,"bulgarian");

Game gm;

gm.Start\_game();

system("pause");

return 0;

}