МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Игра Уголки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование темы проекта или работы)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Алгоритмы и структуры данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

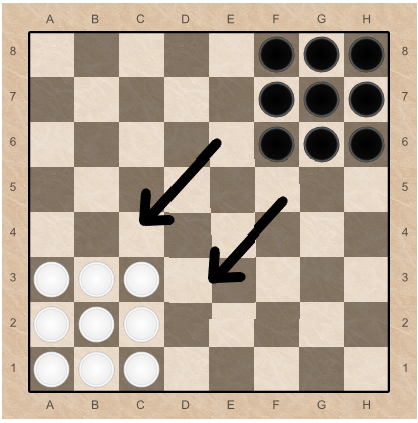
**Текст задачи:**

Цель игры переставить шашки из своего угла (дома) в угол соперника раньше, чем он построит свои в Вашем углу. Ходы делаются по очереди. Движение возможно по горизонталям и вертикалям только в направлениях к углу противника. За один ход передвигается не более одной шашки. Шашка может либо передвигаться на одно поле, либо перепрыгивать через одну или несколько шашек.

Игра заканчивается, когда один из игроков построил все свои шашки в доме противника. Все шашки необходимо вывести из своего дома не позднее чем за 3\*'количество шашек' ходов - иначе будет засчитан проигрыш.

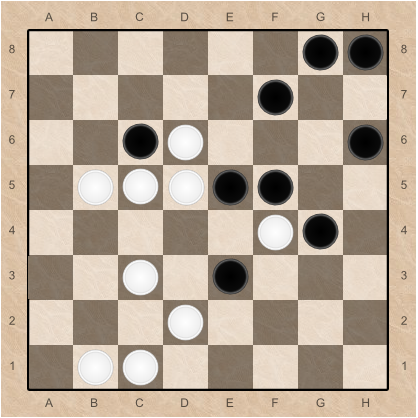
**Описание оценочной функции:**

Основная задача итогового алгоритма состоит в нахождении такого хода, который наиболее далеко продвинет фишку компьютера за один ход как можно ближе к финишу. Примем игру компьютера за игру за чёрных т.е. он будет ходить сверху вниз.

Для того, чтобы оценить состояние игры в текущий момент будем подсчитывать, насколько далеко фишка может продвинуться за n-ное количество ходов.

В данной игре важна такая позиция, при которой наша фишка может передвинуться не на одну клетку влево или вниз, а перепрыгивая через одну клетку, даже при перепрыгивании через одну фишку (неважно, чужую или собственную) т.к. это ход заменяет целых 3 обычных хода, а при нахождении такого хода, при перепрыгивании 2-х и фишек подряд, то вероятность победы гораздо возрастёт.

**Пример расчёта оценочной функции:**

Для нахождения оценки воспользуемся правилом: ход на одну клетку - 1 очко, ход через фишку – 3 очка.

Просчитаем для всех чёрных фишек оценки, которые покажут, насколько максимально далеко каждая фишка может продвинутся за 4 хода (4-величина, взятая рандомно, просматриваемое количество нужно будет подбирать в зависимости от уровня сложности).

Фишка на H8: 10 (перепрыгнуть чёрную, ещё раз чёрную, потом один обычный ход влево, перепрыгнуть через чёрную), аналогично, Фишка на G8: 9, Фишка на F7: 8, Фишка на C6: 8, Фишка на H6: 6, Фишка на E5: 8, Фишка на F5: 10, Фишка на G4: 10, Фишка на E3: 6.

Суммарная оценка соответственно будет равна 75(для белых 68) т.е. у чёрных фишек позиции лучше, а максимальная оценка будет равна 10. Именно фишки с этим числовым значением с определённой вероятностью могут продвинуться к финишу как можно ближе.

**Описание алгоритма Альфа-Бета:**

Альфа-бета отсечение основано на той идее, что анализ

некоторых ходов можно прекратить досрочно, игнорируя результат

их показаний

Эта стратегия оптимизации использует две дополнительные

переменные alpha и beta, где:

- alpha — текущее максимальное значение, меньше которого игрок

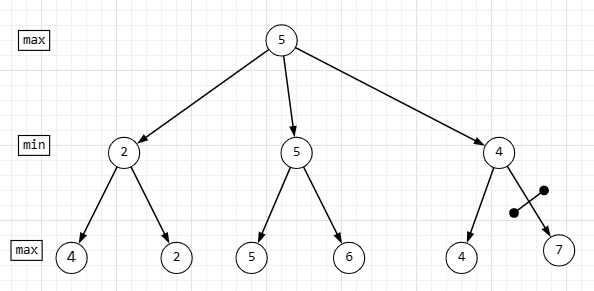
максимизации (компьютер) никогда не выберет (изначально -1000)

- beta — текущее минимальное значение, больше которого игрок минимизации (человек) никогда не выберет (изначально + 1000)

Возвращается оценка состояния, которая получается при ходе из ветки и сравнивается с другими поддеревьями. С помощью алгоритма значения листьев поднимаются к корню. В результате работы алгоритма мы получим позицию с наилучшим количеством очков для компьютера на данном ходе, при этом для каждого отдельного случая мы сделаем просчет следующих возможных ходов в зависимости от максимального уровня рекурсии.

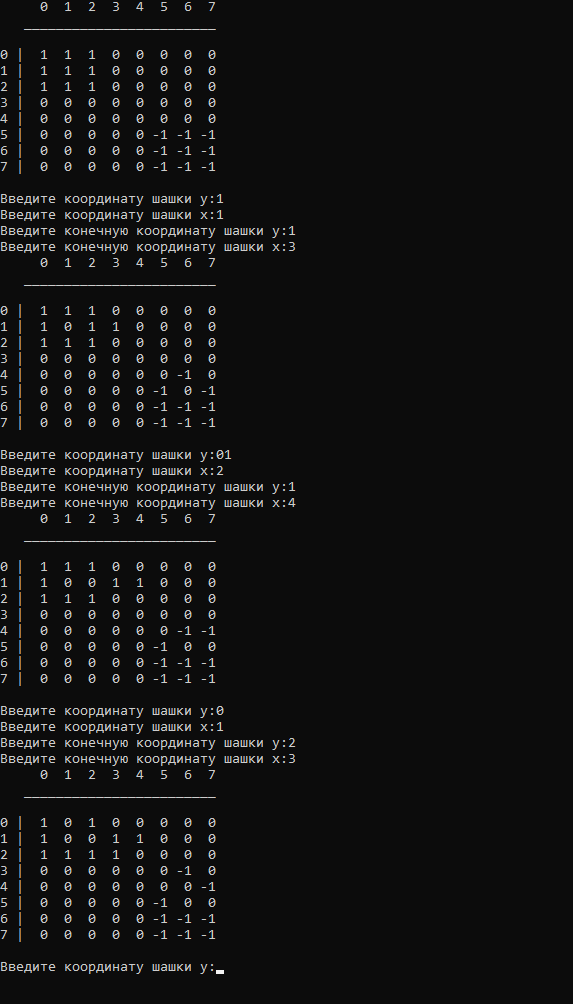
Функция scorebot-оценочная функция, используемая алгоритмом. Структура задана с помощью бинарного дерева из-за особенностей задачи (т.е. робот может пойти или вниз, или влево). Сначала, с помощью функции treewrite записываются значения листьев дерева игры. А уже с помощью функции alfabeta ищется и записывается по алгоритмы нужное нам значение.

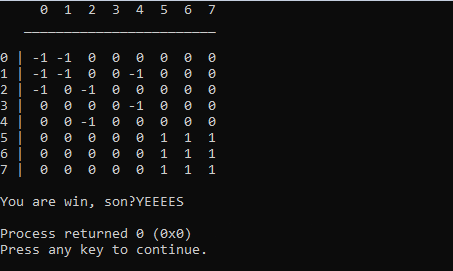
**Пример расчета Альфа-Бета:**

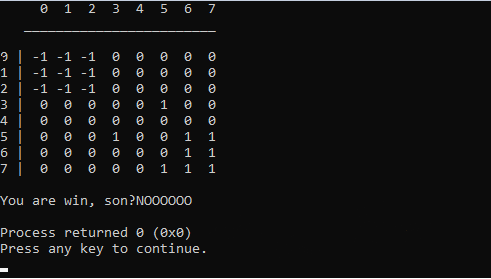
****

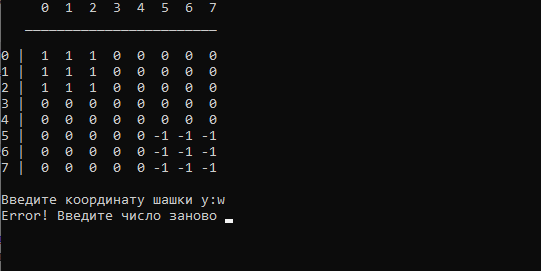
Рассмотрим данный граф. Начнём с самой левой ветви. Алгоритм наверх поднимет число 4, при этом сделав бета=4, дальше алгоритм идёт к цифре 2, соответственно меняя значение бета=2 и поднимая её вверх, взамен цифры 4. После этого, цифра 2 поднимается в корень. Далее рассматривается средняя ветка корня, при этом значение альфа=2. Доходим до цифры 5, поднимаем её, и теперь альфа=2, а бета=5. Соответственно алгоритм заходит в правую ветвь, содержащую 6. При поднятии её наверх возникнет случай, когда альфа> бета, соответственно это число мы оставляем. Поднимаем число 5 наверх к корню. Соответвенно в корне альфа=5, бета=1000. После идём в правую ветвь с альфой=5, встречается цифра 4. Опять появляется случай альфа> бета т.е. 5>4 на уровне минимума, следовательно дальнейшую ветвь с цифрой 7 мы рассматривать не будем.

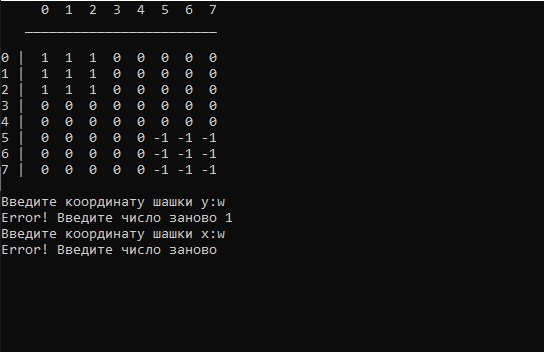
**Пример работы программы:**

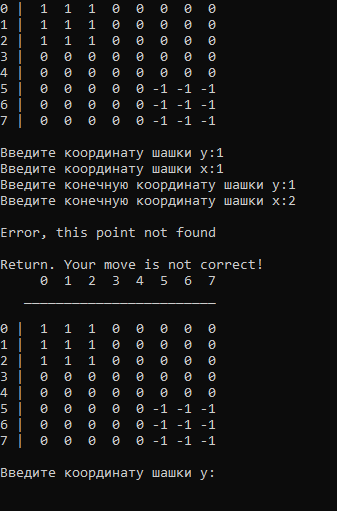
****

****

****

****

****

****

**Список файлов проекта с описанием содержания что в них содержится**

**(классы, функции, и т.д.):**

Проект содержит 1 файл main.cpp.

Функции проекта:

* Scorebot – оценочная функция
* tree\_element – структура для создания и хранения дерева.
* create\_tree\_element - функция для создания дерева.
* insert\_into\_tree - функция для добавления элементов в дерево.
* print\_tree1 - функция вывода дерева.
* Zero - функция заполнения листьев дерева значением из оценочной функции.
* AlphaBeta - функция выполняющая алгоритм альфа-бета, пример которого находится выше.
* minmax - функция выполняющая алгоритм минмакса.
* Xod1, Xod2, Xod3, Xod4- функции находящие оценки в зависимости от вариаций ходов.
* Chod1, Chod2, Chod3, Chod4 - функция выполняющие ход компьютера.
* Chek – функция проверки правильности хода игрока.

**Текст программы:**

#include <iostream>

#include <iomanip>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Оценочная функция scorebot //

int scorebot(int row, int col,int\*\* mass, int n, int ki, int kg){

int f1=n;

if(row>0 and mass[row-1][col]==0){ // Ход на одну влево

f1=f1+1; // Прибавление к числу оценки +1

ki=ki+1;

if(ki<4) {

f1=scorebot(row-1, col,mass,f1,ki,kg); // Вызов данной функции для позиции слева

}

}

else{

if(row>1 and mass[row-2][col]==0){ // Ход через фишку влево

f1=f1+3;

ki=ki+1;

if(ki<4) {

f1=scorebot(row-2,col,mass,f1,ki,kg); // Вызов данной функции рекурсивно для позиции через фишку слева

}

}

}

int f2=n;

if(col>0 and mass[row][col-1]==0){ // Ход на одну вниз

f2=f2+1; // Прибавление к числу оценки +1

kg=kg+1;

if(kg<4) {

f2=scorebot(row,col-1,mass,f2,ki,kg); // Вызов данной фунции рекурсивно для позции вниз

}

}

else{

if(col>1 and mass[row][col-2]==0){ // Ход на через фишку вниз

f2=f2+3;

kg=kg+1;

if(kg<4) {

f2=scorebot(row,col-2,mass,f2,ki,kg); // Вызов данной функции для позиции через фишку вниз

}

}

}

if(f2>=f1){ // Блок выбора наибольшего числа оценки

return f2;

}

if(f2<f1){

return f1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct tree\_element{

int value;

struct tree\_element\* left;

struct tree\_element\* right;

} tree\_element;

tree\_element\* create\_tree\_element(int i){

tree\_element\* elem=new tree\_element;

elem->value=i;

elem->left=NULL;

elem->right=NULL;

return elem;

}

void insert\_into\_tree(tree\_element\* root, tree\_element\* elem){

if(elem->value < root->value){

if(root->left==NULL){

root->left=elem;

}

else{

insert\_into\_tree(root->left,elem);

}

}

else{

if(root->right==NULL){

root->right=elem;

}

else{

insert\_into\_tree(root->right,elem);

}

}

}

void print\_tree1(tree\_element\* cur\_elem){

if(cur\_elem->left!=NULL){

print\_tree1(cur\_elem->left);

}

std::cout<<"V= "<<cur\_elem->value<<std::endl;

if(cur\_elem->right!=NULL){

print\_tree1(cur\_elem->right);

}

}

void Zero(tree\_element\* cur\_elem, int\* mass){

if(cur\_elem->left!=NULL){

Zero(cur\_elem->left, mass);

}

if(cur\_elem->left==NULL and cur\_elem->right==NULL){

for(int i=0;i<4;i++){

if(mass[i]!=0){

cur\_elem->value=mass[i];

mass[i]=0;

}

}

}

if(cur\_elem->right!=NULL){

Zero(cur\_elem->right,mass);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int minmax(tree\_element\* tree, int koef){

int readln1=0,readln2=0;

if (tree->left!=NULL or tree->right!=NULL) { //Пока не встретится пустые дети

readln1=minmax(tree->left,koef+1); //Рекурсивная функция для левого поддерева

readln2=minmax(tree->right,koef+1); //Рекурсивная функция для правого поддерева

if(koef%2!=0){

if(readln1>=readln2){

tree -> value=readln2;

}

else{

tree -> value=readln1;

}

}

if(koef%2==0){

if(readln1>=readln2){

tree -> value=readln1;

}

else{

tree -> value=readln2;

}

}

}

if(koef>=0){

return tree-> value;

}

}

int AlphaBeta(tree\_element\* tree, int koef, int alзрa, int beta){

if (koef%2!=0) // koef принадлежит уровню max

{

g := -1000;

cur\_pos = FirstChild(pos);

while ((tree->left!=NULL) && (g < beta))

{

g = max(g, AlphaBeta(tree, koef + 1, alpha, beta));

alpha = max(g,alpha);

cur\_pos = NextBrother(cur\_pos);

}

}

else // koef принадлежит уровню min

{

g := +1000;

cur\_pos = FirstChild(pos);

while (tree->left!=NULL) && (g > alpha)

{

g = min(g, AlphaBeta(cur\_pos, koef + 1, alpha, beta));

beta = min(g,beta);

cur\_pos = NextBrother(cur\_pos);

}

}

return g;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Xod1(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(i-1>=0 ){

if(mass[i-1][j]==0){

if(i-2>=0 and mass[i-2][j]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j]=-1;

flag=false;

sum=scorebot(i-2,j,mass,0,0,0);

}

if(i-3>=0 and mass[i-2][j]!=0 and mass[i-3][j]==0 and flag==true){

mass[i][j]=0;

mass[i-3][j]=-1;

flag=false;

sum=scorebot(i-3,j,mass,0,0,0)+2;

}

}

if(i-2>=0 and mass[i-1][j]!=0 and mass[i-2][j]==0 and flag==true){

if(i-3>=0 and mass[i-3][j]==0){

mass[i][j]=0;

mass[i-3][j]=-1;

flag=false;

sum=scorebot(i-3,j,mass,0,0,0)+2;

}

if(i-4>=0 and mass[i-3][j]!=0 and mass[i-4][j]==0 and flag==true){

mass[i][j]=0;

mass[i-4][j]=-1;

flag=false;

sum=scorebot(i-4,j,mass,0,0,0)+6;

}

}

}

return sum;

}

int Xod2(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(i-1>=0 ){

if(mass[i-1][j]==0){

if(j-1>=0 and mass[i-1][j-1]==0 and flag==true){

mass[i][j]=0;

mass[i-1][j-1]=-1;

flag=false;

sum=scorebot(i-1,j-1,mass,0,0,0);

}

if(j-2>=0 and mass[i-1][j-1]!=0 and mass[i-1][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-1][j-2]=-1;

flag=false;

sum=scorebot(i-1,j-2,mass,0,0,0);

}

}

if(i-2>=0 and mass[i-1][j]!=0 and mass[i-2][j]==0){

if(j-1>=0 and mass[i-2][j-1]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-1]=-1;

flag=false;

sum=scorebot(i-2,j-1,mass,0,0,0);

}

if(j-2>=0 and mass[i-2][j-1]!=0 and mass[i-2][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-2]=-1;

flag=false;

sum=scorebot(i-2,j-2,mass,0,0,0);

}

}

}

return sum;

}

int Xod3(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(j-1>=0 ){

if(mass[i][j-1]==0){

if(i-1>=0 and mass[i-1][j-1]==0 and flag==true){

mass[i][j]=0;

mass[i-1][j-1]=-1;

flag=false;

sum=scorebot(i-1,j-1,mass,0,0,0);

}

if(i-2>=0 and mass[i-1][j-1]!=0 and mass[i-2][j-1]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-1]=-1;

flag=false;

sum=scorebot(i-2,j-1,mass,0,0,0);

}

}

if(j-2>=0 and mass[i][j-1]!=0 and mass[i][j-2]==0){

if(i-1>=0 and mass[i-1][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-1][j-2]=-1;

flag=false;

sum=scorebot(i-1,j-2,mass,0,0,0);

}

if(i-2>=0 and mass[i-1][j-2]!=0 and mass[i-2][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-2]=-1;

flag=false;

sum=scorebot(i-2,j-2,mass,0,0,0);

}

}

}

return sum;

}

int Xod4(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(j-1>=0 ){

if(mass[i][j-1]==0){

if(j-2>=0 and mass[i][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i][j-2]=-1;

flag=false;

sum=scorebot(i,j-2,mass,0,0,0);

}

if(j-3>=0 and mass[i][j-2]!=0 and mass[i][j-3]==0 and flag==true){

mass[i][j]=0;

mass[i][j-3]=-1;

flag=false;

sum=scorebot(i,j-3,mass,0,0,0)+2;

}

}

if(j-2>=0 and mass[i][j-1]!=0 and mass[i][j-2]==0){

if(j-3>=0 and mass[i][j-3]==0 and flag==true){

mass[i][j]=0;

mass[i][j-3]=-1;

flag=false;

sum=scorebot(i,j-3,mass,0,0,0)+2;

}

if(j-4>=0 and mass[i][j-4]!=0 and mass[i][j-4]==0 and flag==true){

mass[i][j]=0;

mass[i][j-4]=-1;

flag=false;

sum=scorebot(i,j-4,mass,0,0,0)+6;

}

}

}

return sum;

}

void Chod1(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(i-1>=0 ){

if(mass[i-1][j]==0 and flag==true){

mass[i][j]=0;

mass[i-1][j]=-1;

flag=false;

}

if(i-2>=0 and mass[i-1][j]!=0 and mass[i-2][j]==0 and flag==true){

if(i-4>=0 and mass[i-3][j]!=0 and mass[i-4][j]==0 and flag==true){

mass[i][j]=0;

mass[i-4][j]=-1;

flag=false;

}

else if(j-2>=0 and mass[i-2][j-1]!=0 and mass[i-2][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-2]=-1;

flag=false;

}

else{

mass[i][j]=0;

mass[i-2][j]=-1;

flag=false;

}

}

}

}

void Chod2(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(i-1>=0 ){

if(mass[i-1][j]==0 and flag==true){

mass[i][j]=0;

mass[i-1][j]=-1;

flag=false;

}

if(i-2>=0 and mass[i-1][j]!=0 and mass[i-2][j]==0 and flag==true){

if(i-4>=0 and mass[i-3][j]!=0 and mass[i-4][j]==0 and flag==true){

mass[i][j]=0;

mass[i-4][j]=-1;

flag=false;

}

else if(j-2>=0 and mass[i-2][j-1]!=0 and mass[i-2][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-2]=-1;

flag=false;

}

else{

mass[i][j]=0;

mass[i-2][j]=-1;

flag=false;

}

}

}

}

void Chod3(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(j-1>=0 ){

if(mass[i][j-1]==0 and flag==true){

mass[i][j]=0;

mass[i][j-1]=-1;

flag=false;

}

if(j-2>=0 and mass[i][j-1]!=0 and mass[i][j-2]==0 and flag==true){

if(j-4>=0 and mass[i][j-3]!=0 and mass[i][j-4]==0 and flag==true){

mass[i][j]=0;

mass[i][j-4]=-1;

flag=false;

}

else if(i-2>=0 and mass[i-1][j-2]!=0 and mass[i-2][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-2]=-1;

flag=false;

}

else{

mass[i][j]=0;

mass[i][j-2]=-1;

flag=false;

}

}

}

}

void Chod4(int\*\* mass,int i,int j){

int sum=0;

bool flag=true;

if(j-1>=0 ){

if(mass[i][j-1]==0 and flag==true){

mass[i][j]=0;

mass[i][j-1]=-1;

flag=false;

}

if(j-2>=0 and mass[i][j-1]!=0 and mass[i][j-2]==0 and flag==true){

if(j-4>=0 and mass[i][j-3]!=0 and mass[i][j-4]==0 and flag==true){

mass[i][j]=0;

mass[i][j-4]=-1;

flag=false;

}

else if(i-2>=0 and mass[i-1][j-2]!=0 and mass[i-2][j-2]==0 and flag==true){

mass[i][j]=0;

mass[i-2][j-2]=-1;

flag=false;

}

else{

mass[i][j]=0;

mass[i][j-2]=-1;

flag=false;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Chek(int\*\* mass, int x1,int y1, int x2, int y2){

bool qwerty=false;

if(mass[x1+1][y1]==0){

if(x1+1==x2 and y1==y2){

qwerty=true;

}

}

if(mass[x1][y1+1]==0){

if(x1==x2 and y1+1==y2){

qwerty=true;

}

}

if(mass[x1+1][y1]!=0 and mass[x1+2][y1]==0){

if(mass[x1+3][y1]!=0 and mass[x1+4][y1]==0){

if(x1+4==x2 and y1==y2){

qwerty=true;

}

}

else if(mass[x1+2][y1+1]!=0 and mass[x1+2][y1+2]==0){

if(x1+2==x2 and y1+2==y2){

qwerty=true;

}

}

else{

if(x1+2==x2 and y1==y2){

qwerty=true;

}

}

}

if(mass[x1][y1+1]!=0 and mass[x1][y1+2]==0){

if(mass[x1][y1+3]!=0 and mass[x1][y1+4]==0){

if(x1==x2 and y1+4==y2){

qwerty=true;

}

}

else if(mass[x1+1][y1+2]!=0 and mass[x1+2][y1+2]==0){

if(x1+2==x2 and y1+2==y2){

qwerty=true;

}

}

else{

if(x1==x2 and y1+2==y2){

qwerty=true;

}

}

}

return qwerty;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int DIM1 = 8;

const int DIM2 = 8;

int main()

{

tree\_element\* root=create\_tree\_element(0);

tree\_element\* el1=create\_tree\_element(-2);

insert\_into\_tree(root,el1);

tree\_element\* el2=create\_tree\_element(-3);

insert\_into\_tree(root,el2);

tree\_element\* el3=create\_tree\_element(-1);

insert\_into\_tree(root,el3);

tree\_element\* el4=create\_tree\_element(2);

insert\_into\_tree(root,el4);

tree\_element\* el5=create\_tree\_element(3);

insert\_into\_tree(root,el5);

tree\_element\* el6=create\_tree\_element(1);

insert\_into\_tree(root,el6);

bool proverka=true;

int maxMM=0;

int maxMAX=0;

int numberPh=0, numberCh=0;

setlocale(LC\_ALL, "Russian");

int \*\*area=new int \*[DIM1];

for(int i=0;i<DIM1;i++){

area[i]=new int [DIM2];

}

int \*\*Carea=new int \*[DIM1];

for(int i=0;i<DIM1;i++){

Carea[i]=new int [DIM2];

}

int \*OF=new int [4];

int Finish[4][9];

for(int i=0;i<4;i++){

for (int j=0;j<9;j++){

Finish[i][j]=0;

}

}

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM1;j++){

area[i][j]=0;

}

}

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM2;j++){

if(i>=5 and j>=5){

area[i][j]=-1;

}

if(i<3 and j<3){

area[i][j]=1;

}

}

}

int x1,y1,x2,y2;

int kol=0,k=0;

while((area[5][5]!=1 and area[5][6]!=1 and area[5][7]!=1 and area[6][5]!=1 and area[6][6]!=1 and area[6][7]!=1 and area[7][5]!=1 and area[7][6]!=1 and area[7][7]!=1) or (area[0][0]!=-1 and area[0][1]!=-1 and area[0][2]!=-1 and area[1][0]!=-1 and area[1][1]!=-1 and area[1][2]!=-1 and area[2][0]!=-1 and area[2][1]!=-1 and area[2][2]!=-1)){

kol=0;

k=0;

std::cout<<" 0 1 2 3 4 5 6 7"<<std::endl;

std::cout<<" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<std::endl;

std::cout<<std::endl;

for(int i=0;i<DIM1;i++){

std::cout<<i<<" |";

for (int j=0;j<DIM1;j++){

std::cout<<std::setw(3)<<area[i][j];

}

std::cout<<std::endl;

}

std::cout<<std::endl;

std::cout<<"Введите координату шашки y:";

while(!(std::cin>>x1) || (std::cin.peek()!='\n')){

std::cin.clear();

while(std::cin.get()!='\n');

std::cout<<"Error! Введите число заново ";

}

std::cout<<"Введите координату шашки x:";

while(!(std::cin>>y1) || (std::cin.peek()!='\n')){

std::cin.clear();

while(std::cin.get()!='\n');

std::cout<<"Error! Введите число заново ";

}

if(x1>=0 and x1<8 and y1>=0 and y1<8){

if(area[x1][y1]==1){

std::cout<<"Введите конечную координату шашки y:";

while(!(std::cin>>x2) || (std::cin.peek()!='\n')){

std::cin.clear();

while(std::cin.get()!='\n');

std::cout<<"Error! Введите число заново ";

}

std::cout<<"Введите конечную координату шашки x:";

while(!(std::cin>>y2) || (std::cin.peek()!='\n')){

std::cin.clear();

while(std::cin.get()!='\n');

std::cout<<"Error! Введите число заново ";

}

if(x2>=0 and x2<8 and y2>=0 and y2<8 and area[x2][y2]==0){

proverka=Chek(area,x1,y1,x2,y2);

if(proverka==true){

area[x1][y1]=0;

area[x2][y2]=1;

}

}

else{

std::cout<<std::endl<<"Error, this point not found"<<std::endl;

proverka=false;

}

}

else{

std::cout<<std::endl<<"Error, this is not your sashka"<<std::endl;

proverka=false;

}

}

else{

std::cout<<std::endl<<"Error, this point not found"<<std::endl;

proverka=false;

}

if(proverka!=false){

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM1;j++){

Carea[i][j]=0;

}

}

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM1;j++){

Carea[i][j]=area[i][j];

}

}

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM1;j++){

if (area[i][j]==-1){

//std::cout<<"Shashka: "<<i<<" "<<j<<std::endl;

OF[0]=Xod1(Carea,i,j);

//std::cout<<"OF[1] "<<OF[0]<<std::endl;

OF[1]=Xod2(Carea,i,j);

//std::cout<<"OF[2] "<<OF[1]<<std::endl;

OF[2]=Xod3(Carea,i,j);

//std::cout<<"OF[3] "<<OF[2]<<std::endl;

OF[3]=Xod4(Carea,i,j);

//std::cout<<"OF[4] "<<OF[3]<<std::endl;

if(OF[0]<OF[1]){

if(OF[2]>=OF[3]){

if(OF[3]>=OF[0]){

maxMM=OF[3];

}

if(OF[3]<OF[0]){

maxMM=OF[0];

}

}

if(OF[2]<OF[3]){

if(OF[2]>=OF[0]){

maxMM=OF[2];

}

if(OF[2]<OF[0]){

maxMM=OF[0];

}

}

}

if(OF[0]>=OF[1]){

if(OF[2]>=OF[3]){

if(OF[3]>=OF[1]){

maxMM=OF[3];

}

if(OF[3]<OF[1]){

maxMM=OF[1];

}

}

if(OF[2]<OF[3]){

if(OF[2]>=OF[1]){

maxMM=OF[2];

}

if(OF[2]<OF[1]){

maxMM=OF[1];

}

}

}

//std::cout<<"minmax "<<maxMM<<std::endl;

if(maxMAX<maxMM){

maxMAX=maxMM;

}

Finish[0][k]=OF[0];

//std::cout<<"Finish[0][k] "<<Finish[0][k]<<std::endl;

Finish[1][k]=OF[1];

//std::cout<<"Finish[1][k] "<<Finish[1][k]<<std::endl;

Finish[2][k]=OF[2];

//std::cout<<"Finish[2][k] "<<Finish[2][k]<<std::endl;

Finish[3][k]=OF[3];

//std::cout<<"Finish[3][k] "<<Finish[3][k]<<std::endl;

k=k+1;

}

}

}

//std::cout<<"maxMAX "<<maxMAX<<std::endl;

for(int i=0;i<4;i++){

for (int j=0;j<9;j++){

//std::cout<<Finish[i][j]<<" ";

}

//std::cout<<std::endl;

}

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM1;j++){

if(Finish[i][j]==maxMAX){

numberPh=j;

//std::cout<<"numberPh "<<numberPh<<std::endl;

numberCh=i;

//std::cout<<"numberCh "<<numberCh<<std::endl;

maxMAX=-1000;

}

}

}

k=0;

for(int i=0;i<DIM1;i++){

for (int j=0;j<DIM1;j++){

if(area[i][j]==-1){

if(k==numberPh){

if(numberCh==0){

//std::cout<<"Chod1 "<<std::endl;

Chod1(area,i,j);

}

if(numberCh==1){

//std::cout<<"Chod2 "<<std::endl;

Chod2(area,i,j);

}

if(numberCh==2){

//std::cout<<"Chod3 "<<std::endl;

Chod3(area,i,j);

}

if(numberCh==3){

//std::cout<<"Chod4 "<<std::endl;

Chod4(area,i,j);

}

}

k=k+1;

}

}

}

}

else{

std::cout<<"Return"<<std::endl;

}

}

std::cout<<"Победила дружба)))))))))))))))"<<std::endl;

return 0;

}