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

15. Написать программу играющую в игру ОТЕЛЛО (РЕВЕРСИ)

(наименование темы проекта или работы)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине

Алгоритмы и структуры данных  
 (наименование дисциплины)

РУКОВОДИТЕЛЬ:

Капранов С.Н.

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

(подпись)

СТУДЕНТ:

Погонов Д. И.

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

(подпись)

20-ИВТ-2

(шифр группы)

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

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

Нижний Новгород 2021

**Задание.** Написать программу играющую в игру Отелло (Реверси). Реализовать бота, делающего ходы за 2-ого игрока, на основе алгоритма минимакс с использованием Альфа-Бетта отсечения.

**Правила игры**

В игре используется квадратная доска размером 8 × 8 клеток (все клетки могут быть одного цвета) и 64 специальные [фишки](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D1%88%D0%BA%D0%B0), окрашенные с разных сторон в контрастные цвета, например, в белый и чёрный. Один из игроков играет белыми, другой — чёрными. Делая ход, игрок ставит фишку на клетку доски «своим» цветом вверх.

В начале игры в центр доски выставляются 4 фишки: чёрные на *d5* и *e4,* белые на *d4* и *e5.*

* Первый ход делают белые. Далее игроки ходят по очереди.
* Делая ход, игрок должен поставить свою фишку на одну из клеток доски таким образом, чтобы между этой поставленной фишкой и одной из имеющихся уже на доске фишек его цвета находился непрерывный ряд фишек соперника, горизонтальный, вертикальный или диагональный (другими словами, чтобы непрерывный ряд фишек соперника оказался «закрыт» фишками игрока с двух сторон). Все фишки соперника, входящие в «закрытый» на этом ходу ряд, переворачиваются на другую сторону (меняют цвет) и переходят к ходившему игроку.
* Если в результате одного хода «закрывается» одновременно более одного ряда фишек противника, то переворачиваются все фишки, оказавшиеся на всех «закрытых» рядах.
* Игрок вправе выбирать любой из возможных для него ходов. Если игрок имеет возможные ходы, он не может отказаться от хода. Если игрок не имеет допустимых ходов, то ход передаётся сопернику.
* Игра прекращается, когда на доску выставлены все фишки или когда ни один из игроков не может сделать хода. По окончании игры проводится подсчёт фишек каждого цвета, и игрок, чьих фишек на доске выставлено больше, объявляется победителем. В случае равенства количества фишек засчитывается ничья.

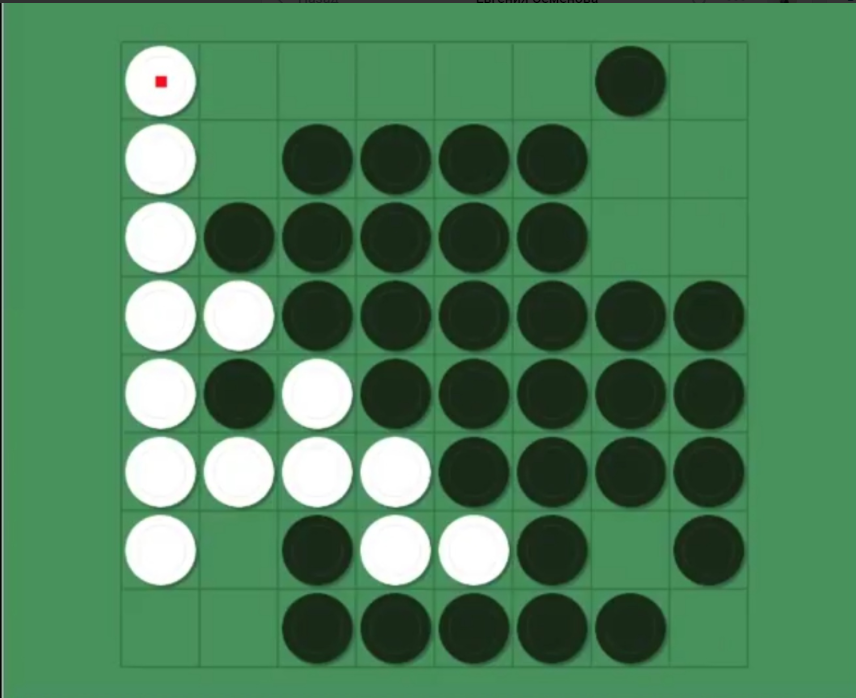
**Оценочная функция**

Нужно написать алгоритм оценивающий ситуацию на игровом поле.

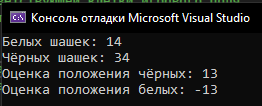
Данный алгоритм работает на основе разности количества белых и чёрных фишек. Но в игре «Реверси» очень важны угловые клетки, если фишка занимает эту клетку, она называется “Стабильной шашкой”, которую нельзя перевернуть. Поэтому фишка, стоящая в углу, в данном алгоритме стоит 8 обычных фишек (данное значение может измениться после тестирования полного алгоритма “бота”).

**Пример расчёта:**

**Игровая ситуация:**

****

**Расчёт:**

****

**Алгоритм Альфа-Бетта**

Функция Альфа-Бетта осуществляет поиск лучшего хода для компьютера, рассчитывая дерево всех возможных вариантов развития игры на определённое количество ходов вперёд. Уровень max – ход компьютера. Среди всех потомков выбирается тот, у которого максимально выигрышная позиция. Уровень min – ход человека. Среди всех потомков выбирается тот, у которого максимально проигрышная позиция. Таким образом гарантируется наиболее оптимальное решение с минимальным риском. Так же в функции используется Альфа-Бетта отсечение, которое отбрасывает невозможные варианты ходов, минимизируя дерево возможных ходов.

**Расчёт Альфа-Бетта для конкретной позиции**

**Расчёт:**

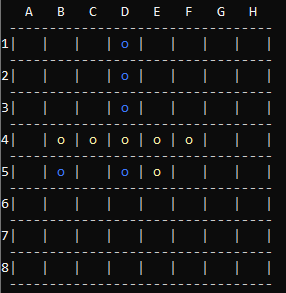
Для удобства использования функции, я решил сделать так, чтобы из функции возвращались координаты поля, ход в которое будет наилучшем решением.

**Вывод функции minimax –** Целое двузначное число, первый символ – i т.е. строка, второй символ – j т.е. столбец­­­­

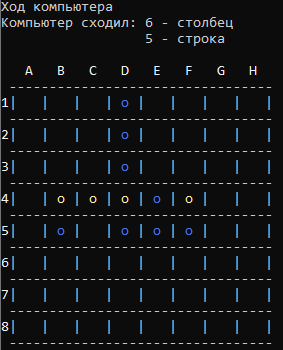
Жёлтые фишки - человек

Синие фишки – компьютер

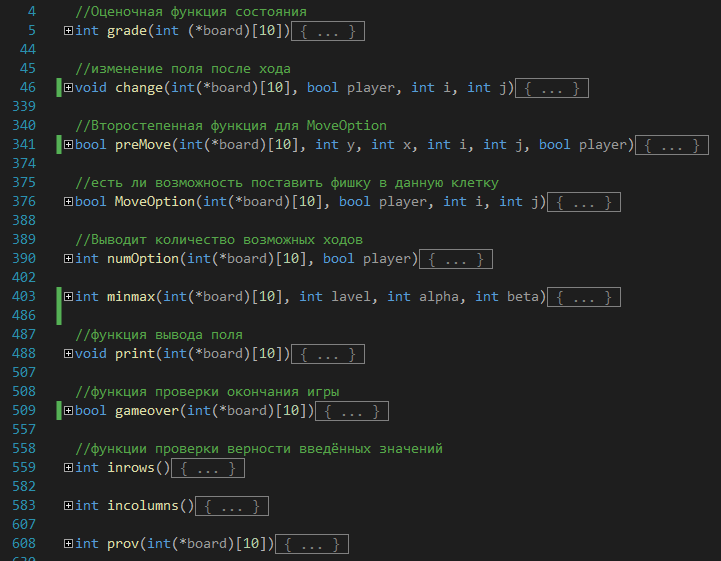
**Игровая ситуация:**

****

**Решение Альфа-Бетта:**

****

**Список используемых функций**



**Исходный код программы**

#include <iostream>

//Оценочная функция состояния

int grade(int (\*board)[10]) //в функцию передаётся массив, в

{ //каждой ячейке которого состояние

int result {}; //соответствующей клетки игрового поля

int white {}; //0 - клетка пустая, 1-белая фишка,

int black {}; //2 - чёрная фишка

int k = 7; //коэффициент важности угла

for (int i = 1; i < 9; i++) //подсчёт кол-ва фишек

{

for (int j = 1; j < 9; j++)

{

if (board[i][j] == 1) white++;

if (board[i][j] == 2) black++;

}

}

if (white == 0) return 999;

if (black == 0) return -999;

if (board[1][1] != 0) //проверяем, заняты ли углы доски

{

if (board[1][1] == 1) white += k;

else black += k; //соответствующему игроку добавляем k

}

if (board[1][8] != 0)

{

if (board[1][8] == 1) white += k;

else black += k;

}

if (board[8][1] != 0)

{

if (board[8][1] == 1) white += k;

else black += k;

}

if (board[8][8] != 0)

{

if (board[8][8] == 1) white += k;

else black += k;

}

result = black - white; //расчёт разности фишек

return result;

}

//изменение поля после хода

void change(int(\*board)[10], bool player, int i, int j)

{ //i, j - координаты точки, в которую ставится шашка

int ii = I; //для временного хранения положения

int jj = j;

if (player == 0) //если ход делает белый игрок

{

board[i][j] = 1;

if (board[i + 1][j] == 2)

{

i += 1;

while (board[i][j] == 2) i++; //пока фишки противника

//стоят в ряд

if (i < 9 && i > 0 && board[i][j] == 1) //если мы не вышли

{ //за поле и данная фишка наша

for (int n = ii + 1; n <= i; n++)

{

board[n][j] = 1;

}

}

i = ii;

j = jj;

}

if (board[i - 1][j] == 2)

{

i -= 1;

while (board[i][j] == 2) i--;

if (i < 9 && i > 0 && board[i][j] == 1)

{

for (int n = ii - 1; n >= i; n--)

{

board[n][j] = 1;

}

}

i = ii;

j = jj;

}

if (board[i][j + 1] == 2)

{

j += 1;

while (board[i][j] == 2) j++;

if (j < 9 && j > 0 && board[i][j] == 1)

{

for (int m = jj + 1; m <= j; m++)

{

board[i][m] = 1;

}

}

i = ii;

j = jj;

}

if (board[i][j - 1] == 2)

{

j -= 1;

while (board[i][j] == 2) j--;

if (j < 9 && j > 0 && board[i][j] == 1)

{

for (int m = jj - 1; m >= j; m--)

{

board[i][m] = 1;

}

}

i = ii;

j = jj;

}

if (board[i + 1][j + 1] == 2)

{

i += 1;

j += 1;

while (board[i][j] == 2)

{

i++;

j++;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 1)

{

int m = jj + 1;

for (int n = ii + 1; n <= i; n++)

{

board[n][m] = 1;

m++;

}

}

i = ii;

j = jj;

}

if (board[i - 1][j + 1] == 2)

{

i -= 1;

j += 1;

while (board[i][j] == 2)

{

i--;

j++;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 1)

{

int m = jj + 1;

for (int n = ii - 1; n >= i; n--)

{

board[n][m] = 1;

m++;

}

}

i = ii;

j = jj;

}

if (board[i + 1][j - 1] == 2)

{

i += 1;

j -= 1;

while (board[i][j] == 2)

{

i++;

j--;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 1)

{

int m = jj - 1;

for (int n = ii + 1; n <= i; n++)

{

board[n][m] = 1;

m--;

}

}

i = ii;

j = jj;

}

if (board[i - 1][j - 1] == 2)

{

i -= 1;

j -= 1;

while (board[i][j] == 2)

{

i--;

j--;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 1)

{

int m = jj - 1;

for (int n = ii - 1; n >= i; n--)

{

board[n][m] = 1;

m--;

}

}

i = ii;

j = jj;

}

}

else //если ход делает чёрный игрок

{

board[i][j] = 2;

if (board[i + 1][j] == 1)

{

i += 1;

while (board[i][j] == 1) i++; //пока фишки противника

//стоят в ряд

if (i < 9 && i > 0 && board[i][j] == 2) //если мы не вышли

{ //за поле и данная фишка наша

for (int n = ii + 1; n <= i; n++)

{

board[n][j] = 2;

}

}

i = ii;

j = jj;

}

if (board[i - 1][j] == 1)

{

i -= 1;

while (board[i][j] == 2) i--;

if (i < 9 && i > 0 && board[i][j] == 2)

{

for (int n = ii - 1; n >= i; n--)

{

board[n][j] = 2;

}

}

i = ii;

j = jj;

}

if (board[i][j + 1] == 1)

{

j += 1;

while (board[i][j] == 1) j++;

if (j < 9 && j > 0 && board[i][j] == 2)

{

for (int m = jj + 1; m <= j; m++)

{

board[i][m] = 2;

}

}

i = ii;

j = jj;

}

if (board[i][j - 1] == 1)

{

j -= 1;

while (board[i][j] == 1) j--;

if (j < 9 && j > 0 && board[i][j] == 2)

{

for (int m = jj - 1; m >= j; m--)

{

board[i][m] = 2;

}

}

i = ii;

j = jj;

}

if (board[i + 1][j + 1] == 1)

{

i += 1;

j += 1;

while (board[i][j] == 1)

{

i++;

j++;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 2)

{

int m = jj + 1;

for (int n = ii + 1; n <= i; n++)

{

board[n][m] = 2;

m++;

}

}

i = ii;

j = jj;

}

if (board[i - 1][j + 1] == 1)

{

i -= 1;

j += 1;

while (board[i][j] == 1)

{

i--;

j++;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 2)

{

int m = jj + 1;

for (int n = ii - 1; n >= i; n--)

{

board[n][m] = 2;

m++;

}

}

i = ii;

j = jj;

}

if (board[i + 1][j - 1] == 1)

{

i += 1;

j -= 1;

while (board[i][j] == 1)

{

i++;

j--;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 2)

{

int m = jj - 1;

for (int n = ii + 1; n <= i; n++)

{

board[n][m] = 2;

m--;

}

}

i = ii;

j = jj;

}

if (board[i - 1][j - 1] == 1)

{

i -= 1;

j -= 1;

while (board[i][j] == 1)

{

i--;

j--;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 2)

{

int m = jj - 1;

for (int n = ii - 1; n >= i; n--)

{

board[n][m] = 2;

m--;

}

}

i = ii;

j = jj;

}

}

}

//Второстепенная функция для MoveOption

bool preMove(int(\*board)[10], int y, int x, int i, int j, bool player) //проверяет одно направление от шашки

{ //x, y - задают направление

bool result{};

if (player == 0)

{

if (board[i + y][j + x] == 2)

{

i += y;

j += x;

while (board[i][j] == 2)

{

i += y;

j += x;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 1)

result = 1;

}

}

else

{

if (board[i + y][j + x] == 1)

{

i += y;

j += x;

while (board[i][j] == 1)

{

i += y;

j += x;

}

if (j < 9 && j > 0 && i < 9 && i > 0 && board[i][j] == 2)

result = 1;

}

}

return result;

}

//есть ли возможность поставить фишку в данную клетку

bool MoveOption(int(\*board)[10], bool player, int i, int j)

{ //если клетка свободна и по правилам игры в неё можно сходить

if (board[i][j] == 0 && preMove(board, 0, 1, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, 0, -1, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, 1, 1, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, 1, 0, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, 1, -1, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, -1, 1, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, -1, 0, i, j, player) == 1) return 1;

if (board[i][j] == 0 && preMove(board, -1, -1, i, j, player) == 1) return 1;

return 0;

}

//Выводит количество возможных ходов

int numOption(int(\*board)[10], bool player)

{

int count{};

for (int i = 1; i < 9; i++)

{

for (int j = 1; j < 9; j++)

{

if (MoveOption(board, player, i, j) == 1) count++;

}

}

return count;

}

int minmax(int(\*board)[10], int lavel, int alpha, int beta)

{

if (numOption(board, lavel % 2) == 0 || lavel == 6)

return grade(board); //Если ходов нет или мы дошли до 5

//уровня, то возвращаем оценку

if (lavel % 2 == 1) //Уровень max

{

if (lavel == 1) //если находимся на первом уровне

{

int\* arrG = new int[64]; //создаём массив значений g

for (int i = 0; i < 64; i++) //заполнение дин. массива

arrG[i] = -999;

int k{}; //счётчик для дин. массива

for (int i = 1; i < 9; i++)

{

for (int j = 1; j < 9; j++)

{

if (MoveOption(board, lavel % 2, i, j) == 1)

{

int cboard[10][10];

memcpy(cboard, board, sizeof(int) \* 100);

change(cboard, lavel % 2, i, j);

arrG[k] = i\*(-100);//записываем координаты

//каждого хода

arrG[k+1] = j\*(-100); //чтобы понять, какой

//конкретно ход дал наилучшее значение

arrG[k+2] = minmax(cboard, lavel + 1, alpha, beta);

k += 3;

}

}

}

int max{-99};

int ii{};

for (int i = 0; i < 64; i++)

{

if (arrG[i] > max)

{

max = arrG[i];

ii = i;

}

}

int result{};

result = (arrG[ii - 1] / (-10)) + (arrG[ii - 2] / (-100));

//результат - двузначное число, первый символ которого j, второй - i

delete[] arrG;

return result;

}

else

{

int g{ -9999 };

for (int i = 1; i < 9; i++)

{ //проходим по всем клеткам поля

for (int j = 1; j < 9; j++)

{ //если нашли ту, в которой можно сделать ход

if (MoveOption(board, lavel % 2, i, j) == 1)

{

int cboard[10][10];

memcpy(cboard, board, sizeof(int) \* 100);

change(cboard, lavel % 2, i, j) //делаем

//"мнимый" ход

g = std::max(g, (minmax(cboard, lavel + 1, alpha, beta))); //вызываем рекурсию

alpha = std::max(g, alpha);

if (g < beta) return g;

}

}

}

return g;

}

}

else //Уровень min

{

int g{ 9999 };

for (int i = 1; i < 9; i++)

{ //проходим по всем клеткам поля

for (int j = 1; j < 9; j++)

{ //если нашли ту, в которой можно сделать ход

if (MoveOption(board, lavel % 2, i, j) == 1)

{

int cboard[10][10];

memcpy(cboard, board, sizeof(int) \* 100);

change(cboard, lavel % 2, i, j); //делаем

//"мнимый" ход

g = std::min(g, (minmax(cboard, lavel + 1, alpha, beta))); //вызываем рекурсию

beta = std::min(g, beta);

if (g > alpha) return g;

}

}

}

return g;

}

}

//функция вывода поля

void print(int(\*board)[10])

{

std::cout << "\n\x1b[1m A B C D E F G H \x1b[0m \n";

std::cout << " ---------------------------------\n";

for (int i = 1; i < 9; i++)

{

std::cout << "\x1b[1m" << i << "\x1b[0m|";

for (int j = 1; j < 9; j++)

{

if (board[i][j] == 0) std::cout << " |";

else

{

if (board[i][j] == 1) std::cout << " \x1b[93mo\x1b[0m |";

else std::cout << " \x1b[94mo\x1b[0m |";

}

}

std::cout << "\n ---------------------------------\n";

}

}

//функция проверки окончания игры

bool gameover(int(\*board)[10])

{

int pc{}; //счётчики

int man{};

int voidCell{};

for (int i = 1; i < 9; i++)

{

for (int j = 1; j < 9; j++)

{

if (board[i][j] == 0) voidCell++;

if (board[i][j] == 1) man++;

if (board[i][j] == 2) pc++;

}

}

if (numOption(board, 0) == 0 && numOption(board, 0) == 1)

{

std::cout << "Ходов больше нет. ";

if (man > pc) std::cout << "\x1b[93m Человек выиграл! \x1b[0m\n";

if (man < pc) std::cout << "\x1b[94m Компьютер выиграл! \x1b[0m\n";

if (man == pc) std::cout << " Конец игры. Бевая ничья!! \n";

return 1;

}

if (pc == 0 || man == 0 || voidCell == 0)

{

if (pc == 0)

{

std::cout << "\x1b[93m Конец игры. Человек выиграл!! \x1b[0m\n";

exit(1);

}

if (man == 0)

{

std::cout << "\x1b[94m Конец игры. Компьютер выиграл!! \x1b[0m\n";

exit(1);

}

if (voidCell == 0)

{

if (man > pc)

{

std::cout << "\x1b[93m Конец игры. Человек выиграл!!\x1b[0m\n";

exit(1);

}

if (man < pc)

{

std::cout << "\x1b[94m Конец игры.Компьютер выиграл!\x1b[0m\n";

exit(1);

}

if (man == pc)

{

std::cout << "\x1b[94m Конец игры. Боевая ничья!! \x1b[0m\n";

exit(1);

}

}

return 1;

}

else return 0;

}

//функции проверки верности введённых значений

int inrows()

{

int num{}; //поле ввода

int k{}; //счётчик

while (k < 3)

{

std::cout << "Введите номер строки (1-8): ";

std::cin >> num;

if (!std::cin.good() || num < 1 || num>8)

{

if (k == 2)

{

std::cout << "Слишком много попыток ввода, завершение работы игры!\n";

exit(-1);

}

std::cout << "Ошибка. Введён неверный параметр, попробуйте

ввести значение ещё раз." << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),

'\n');

k++;

}

else return num;

}

}

int incolumns()

{

char col{}; //поле ввода

int k{}; //счётчик

while (k < 3)

{

std::cout << "Введите букву столбца (A-H): ";

std::cin >> col;

col = tolower(col); //переводим в нижний регистр

if (!std::cin.good() || ((int)col - 96) < 1 || ((int)col - 96) > 8)

{

if (k == 2)

{

std::cout << "\x1b[31mСлишком много попыток ввода, завершение работы игры!\x1b[0m\n";

exit(-1);

}

std::cout << "\x1b[31mОшибка. Введён неверный параметр,

попробуйте ввести значение ещё раз.\x1b[0m" << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),

'\n');

k++;

}

else return (int)col - 96;

}

}

int prov(int(\*board)[10])

{

int rows{};

int columns{};

int k{};

while (k < 3)

{

if (k == 2)

{

std::cout << "\x1b[31mСлишком много попыток ввода, завершение работы игры!\x1b[0m\n";

exit(-1);

}

columns = incolumns();

rows = inrows();

if (MoveOption(board, 0, rows, columns) == 0)

{

std::cout << "\x1b[31mОшибка. В данную клетку нельзя сходить.

Попробуйте ещё раз.\x1b[0m" << std::endl;

k++;

}

else return (columns \* 10) + rows;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int board[10][10]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 2, 0, 0, 0, 0,

0, 0, 0, 0, 2, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

bool player1 = 0; //человек

bool player2 = 1; //компьютер

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

std::cout << " | Nizhniy Novgorod Technical University |\n";

std::cout << " | Study course work. |\n";

std::cout << " | Performed student 20-IVT-2 Pogonov D. |\n";

std::cout << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

std::cout << "Игра Реверси (Отелло)\n";

std::cout << "\x1b[93mo\x1b[0m - фишки человека\n";

std::cout << "\x1b[94mo\x1b[0m - фишки компьютера\n";

print(board);

while (gameover(board) == 0)

{

std::cout << "Ход человека\n";

int res{};

if (numOption(board, player1) > 0)

{

res = prov(board);

change(board, player1, res % 10, res / 10);

print(board);

}

else std::cout << "У человека нет ходов\n";

gameover(board);

std::cout << "Ход компьютера\n";

res = minmax(board, 1, 99999, -99999);

std::cout << "Компьютер сходил: " << res / 10 << " - столбец\n";

std::cout << " " << res % 10 << " - строка\n";

change(board, player2, res % 10, res / 10);

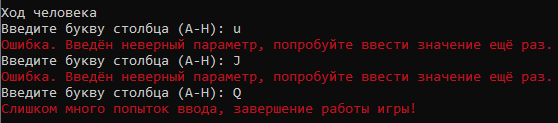
print(board);

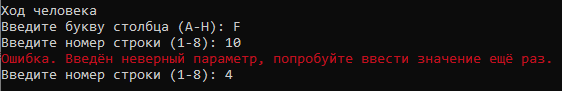
}

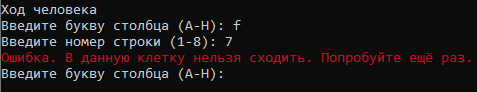
return 1;

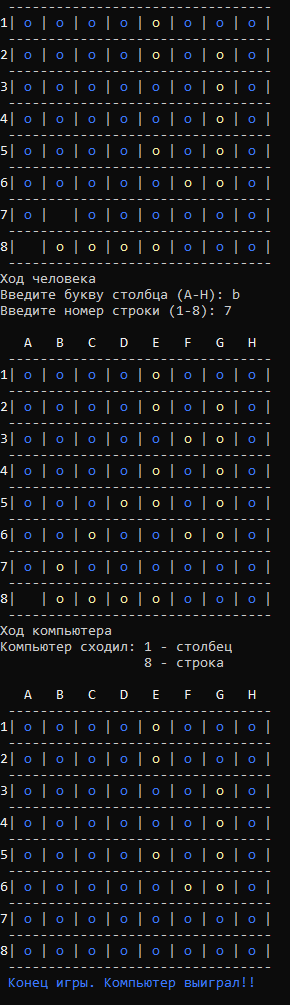
}

**Результаты работы**

****

****

****

****

Обыграть созданный алгоритм не получилось.

**Список литературы**

1. https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B2%D0%B5%D1%80%D1%81%D0%B8