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

Курсовая работа

Игра «Точки и квадраты»

по дисциплине

Алгоритмы и структуры данных

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Капранов С. Н.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В.А.

19-ИВТ-3

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

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

Нижний Новгород 2020

Оглавление

[Правила игры 4](#_Toc59293960)

[Инструкция пользователя 5](#_Toc59293961)

[Инструкция программиста 6](#_Toc59293962)

[Оценочная функция 7](#_Toc59293963)

[Описание оценочной функции 7](#_Toc59293964)

[Пример расчёта оценочной функции 7](#_Toc59293965)

[Программная реализация оценочной функции 8](#_Toc59293966)

[Алгоритм минимакса 9](#_Toc59293967)

[Описание вычислений 9](#_Toc59293968)

[Пример расчёта 10](#_Toc59293969)

[Программная реализация 11](#_Toc59293970)

[Функция альфа бета отсечения 12](#_Toc59293971)

[Описание функции альфа бета отсечения 12](#_Toc59293972)

[Программная реализация 13](#_Toc59293973)

[Код Tree.h 17](#_Toc59293974)

[Код основного файла 19](#_Toc59293975)

[Функция game\_is\_over() 19](#_Toc59293976)

[Функция check\_cell() 20](#_Toc59293977)

[Функция is\_square\_up () 23](#_Toc59293978)

[Функция is\_square\_down () 24](#_Toc59293979)

[Функция is\_square\_right () 25](#_Toc59293980)

[Функция is\_square\_left () 26](#_Toc59293981)

[Функция is\_square () 27](#_Toc59293982)

[Функция generation\_move() 29](#_Toc59293983)

[Функция make\_tree() 32](#_Toc59293984)

[Функция get\_assessement() 34](#_Toc59293985)

[Функция alpha\_betta() 35](#_Toc59293986)

[Функция choose\_best\_move() 37](#_Toc59293987)

[Функция Print\_field() 38](#_Toc59293988)

[Функция create\_root() 39](#_Toc59293989)

[Функция re\_create\_root() 40](#_Toc59293990)

[Функция read\_human\_move() 41](#_Toc59293991)

[Функция game() 43](#_Toc59293992)

[Функция get\_result() 44](#_Toc59293993)

[Функция main() 45](#_Toc59293994)

[Результаты работы программы 46](#_Toc59293995)

[Ввод некорректных данных 46](#_Toc59293996)

[Корректная работа программы 47](#_Toc59293997)

# Правила игры

Игровое поле представляет из себя n рядов по n точек. Играющие соединяют соседние по вертикали или горизонтали точки отрезками. Ходы делаются по очереди, но если игрок замкнул своим ходом один или два единичных квадратика, то он закрашивает эти квадраты своим цветом, и он обязан продолжать делать ходы, пока не сделает ход, не замыкающий квадратик, или до конца игры. Когда проведены все отрезки, подсчитываются квадратики, которые замкнул каждый игрок. Выигрывает тот, кто занял больше точек.

# Инструкция пользователя

Программа разработана для игры в «Точки и квадраты». Исходное поле представляет собой поле точек, в крайних двух противоположных углах располагаются точки компьютера и Ваша: в левом нижем компьютера, в правом верхнем Ваша.

Цель игры заключается в захвате незанятых точек. Побеждает тот, кто соберёт больше.

Если игрок собирает квадрат, то его противник пропускает ход.

Соединять можно любую занятую Вами точку с точкой, не принадлежащей противнику и расположенной рядом с данной (сверху, снизу, справа или слева).

Координаты точки – 2 значения от 0 до 9.

# Инструкция программиста

Файл Tree.h содержит структуру узла дерева и функции для работы с ней. Поля структуры:

struct Tree\_node {

Tree\_node\* child[N]; //Массив указателей на сыновей

Tree\_node\* parent; //Указатель на родителя

Tree\_node\* next\_brother; //Указатель на следующего брата

int field [10][10] ; //Состояние поля игры

int connections[10][10][4]; //Массив клеток, соединённых с данной

int count\_connections[10][10];//Количество клеток, соединённых с данной

int count\_child; //Количество сыновей

int number; //Номер узла

int height; //Высота узла

int assessment; //Оценка текущего состояния игры

int alpha; //Параметр для альфа-бета отсечения

int betta; //Параметр для альфа-бета отсечения

int count\_check\_child; //Количество просмотренных сыновей при

//поиске лучшего хода

bool is\_check; //Был ли полностью рассмотрен узел т.е.

//рассмотрены все его сыновья

bool is\_square; //Показывает был ли образован квадрат на

//этом ходу

};

Функции:

1. Tree\_node\* newNode //Позволяет создавать узел с переданными

//параметрами

1. void push(]); //Позволяет добавлять сына узлу с

//переданными номером

1. void deletion(); //Позволяет освободить динамическую

//память выделенную под дерево

Код основного файла содержит функции логики игры:

1. bool game\_is\_over(); //Проверяет завершенность игры
2. bool check\_cell(); //Проверяет можно ли занять клетку
3. void is\_square(); //Проверяет был ли образован квадрат

//на данном ходу

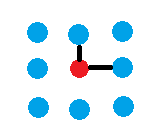
1. void generation\_move(); //Генерирует ход
2. void make\_tree(); //Строит дерево возможных ходов
3. void get\_assessment(); //Оценочная функция
4. void alpha\_betta(); //Алгоритм альфа бета отсечения
5. void choose\_best\_move(); //Выбирает лучший ход
6. void Print\_field(); //Выводит поле игры
7. Tree\_node\* creat\_root(); //Создаёт начальный корень дерева
8. Tree\_node \* re\_create\_root();//Создаёт корень дерева
9. void read\_human\_move(); //Считывает ход игрока
10. Tree\_node\* game(); //Осуществляет игровой процесс
11. void get\_result(); //Анализ результатов игры

# Оценочная функция

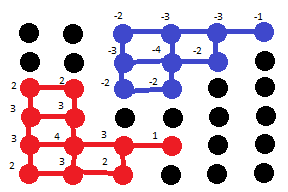
## Описание оценочной функции

Каждая точка на игровом поле имеет свою стоимость, её значение может быть отрицательным или положительным, в зависимости от принадлежности игроку. Если точка принадлежит компьютеру, то её стоимость положительная, если человеку – отрицательная. Если точка не принадлежит игроку, то её стоимость равна нулю. Результат оценочной функции – сумма стоимостей каждой точки.

Стоимость точки определяется её связями с соседними "‎союзными "‎ точками: модуль значения стоимости точки равен количеству её связей.

Пример: Красная точка принадлежит компьютеру и имеет 2 связи с соседними точками. Её стоимость – 2.

## Пример расчёта оценочной функции



1. Рассчитываем стоимость каждой клетки.
2. Вычисляем значение оценочной функции.

F=2+2+3+3+3+4+3+1+2+3+2-2-3-3-1-3-4-2-2-2=6

## Программная реализация оценочной функции

/\*Оценочная функция

Параметры:

1)Указатель на узел дерева, поля которого:

1. int field[10][10]- текущее состояние поля
2. int count\_connections[10][10]- количество связей клетки
3. assessment - оценка поля

Принцип работы:

1)Просматриваются значения клеток текущего поля

2)Если значение клетки равно 1, то она принадлежит компьютеру. К результату прибавляется количество связей этой клетки

3)Если значение клетки равно 2, то она принадлежит человеку. К результату вычитается количество связей этой клетки

\*/

void get\_assessment(Tree\_node\* PNode) {

int res = 0;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

if (PNode->field[i][j] == 1) {

res = res + PNode->count\_connections[i][j];

}

if (PNode->field[i][j] == 2) {

res = res - PNode->count\_connections[i][j];

}

}

}

PNode->assessment = res;

}

# Алгоритм минимакса

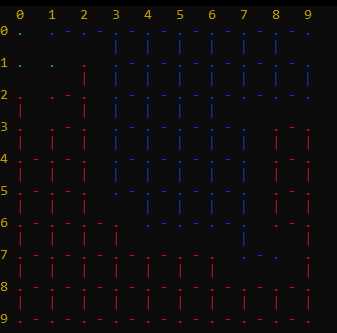
## Описание вычислений

Алгоритм Минимакса — правило принятия решений, используемое в теории игр, теории принятия решений, исследовании операций, статистике и философии для минимизации возможных потерь их тех, которые лицу, принимающему решение, нельзя предотвратить при развитии событий по наихудшему сценарию.

Строится дерево ходов (решений) заданной глубины. После постройки дерева получаем результаты оценочной функции для каждого листа (состояния поля) и начинаем двигаться вверх, к корню дерева. На уровне MIN выбирается минимальное значение дочерних узлов и транслируется на данный уровень, а на уровне MAX максимальное значение дочерних узлов. Последними операциями выбора и трансляции получаем значение для корня дерева. Как только получено значение корня дерева делаем ход согласно той ветви, которая привела к данному результату. Если к одинаковому значению корня приводят несколько ветвей дерева, то направления движения выбирается произвольно.

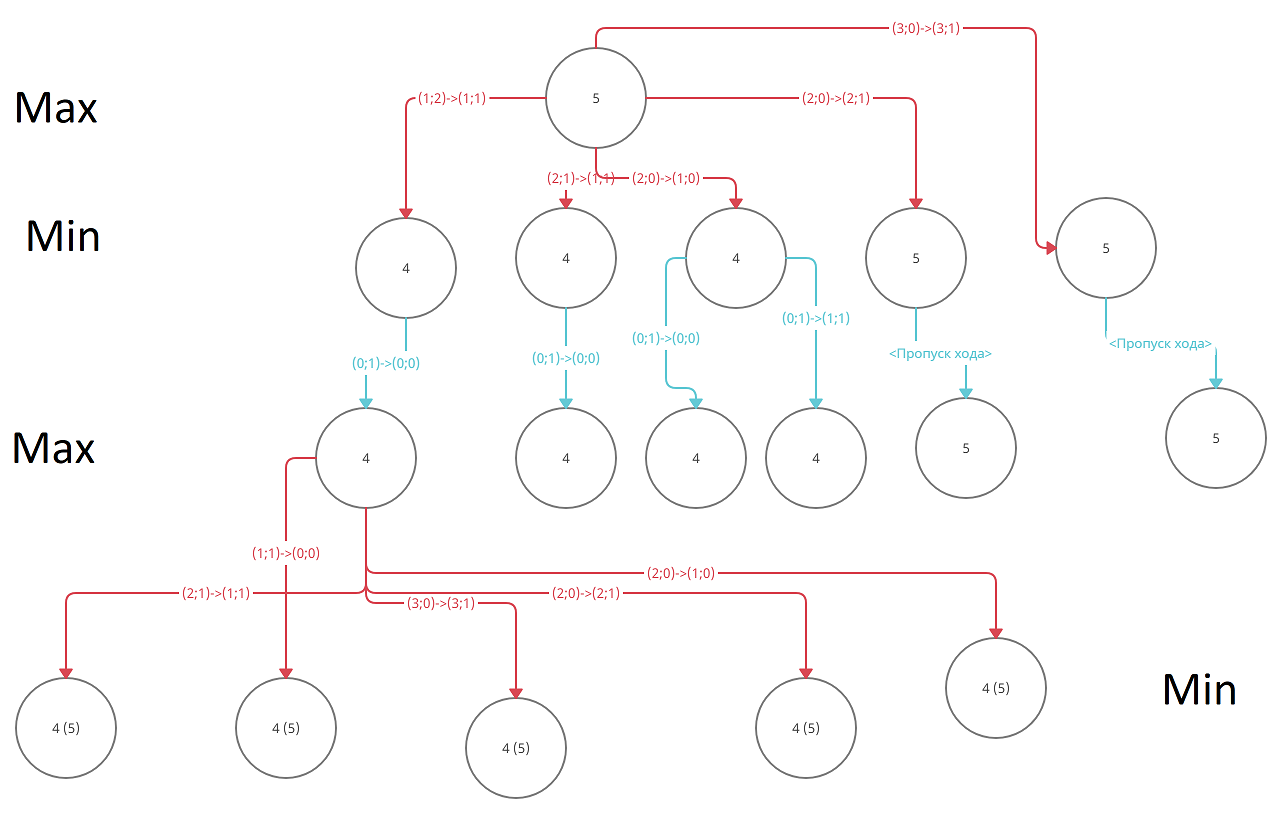
## Пример расчёта

Рассмотрим состояние поля, представленное на скриншоте 1. Ход делает компьютер (MAX – красные точки).



Скриншот 1

Для данного состояния возможно построение следующего дерева минимакса.



Замечание: Последний уровень у всех вершин одинаковый, для отсутствия нагромождения он построено только для одной вершины.

В дереве красным цветом отмечены ходы компьютер, синим ходы игрока.

## Программная реализация

/\*Функция минимакса

Параметры:

1)Указатель на узел дерева

Принцип работы:

1)Если узел является листом, то высчитываем его оценку

2)Если проверены не все сыновья узла, то применяем функция к непроверенным сыновьям

4)Вычисляем значение проверенных сыновей

5)Если проверены все сыновья, то узел проверен

6)Если узел не корень:

6.1) Если родитель находится на чётном уровне (минимума) и его значение больше оценки текущего узла, то заменяем оценку

6.2) Если родитель находится на нечётном уровне (максимума и его значение меньше оценки текущего узла, то заменяем оценку

\*/

void Min\_max(Tree\_node\* PNode) {

if (PNode->height == 3) {

get\_assessment(PNode);

}

if (PNode->count\_check\_child != PNode->count\_child) {

for (int i = PNode->count\_check\_child;

i <PNode->count\_child; i++)

{

Min\_max(PNode->child[i]);

}

}

for (int i = 0; i < PNode->count\_child; i++) {

if (PNode->child[i]->is\_check == true) {

PNode->count\_check\_child++;

}

}

if (PNode->count\_check\_child == PNode->count\_check\_child) {

PNode->is\_check = true;

}

if (PNode->height != 1) {

if (PNode->parent->height % 2 == 0) {

if (PNode->parent-> assessment > PNode->assessment)

PNode->parent-> assessment = PNode->assessment;

}

if (PNode->parent->height % 2 == 1) {

if (PNode->parent-> assessment < PNode->assessment) {

PNode->parent-> assessment = PNode->assessment;

}

}

if (PNode->next\_brother != nullptr) {

PNode = PNode->next\_brother;

}

else {

PNode = PNode->parent;

}

}

}

# Функция альфа бета отсечения

## Описание функции альфа бета отсечения

Алгоритм альфа бета отсечения состоит из двух функций: нахождения наилучшей оценки для вершины дерева и выбора хода, если у корня есть несколько сыновей с одинаковой оценкой.

В первой функции осуществляется обход всех вершин и определение их оценки. Оценка листа определяется изходя из оценочной функции, оценка остальных листов определяется по правилу:

1)Если проверены не все сыновья узла, то применяем функция к непроверенным сыновьям

2)Если узел не является корнем, то копируем значения alpha и betta родителя узла

3)Вычисляем значение проверенных сыновей

4)Если проверены все сыновья, то узел проверен

5)Если узел не корень:

5.1) Если родитель находится на чётном уровне (минимума), то корректируем значение betta родителя узла

5.2) Если родитель находится на нечётном уровне (максимума), то корректируем значение alpha родителя узла

5.3) Если у родителя узла alpha <betta:

5.3.1) Значение оценки родителя равно оценке текущего узла

5.3.2) Если у узла есть следующий брат, то переходим к нему

5.3.3) Переходим к родителю текущего узла

5.4) Если у родителя узла alpha> betta:

5.4.1) Переходим к родителю текущего узла

5.4.2) Количество проверенных сыновей = количество сыновей

5.4.3) Узел проверен

Вторая функция копирует положение игрового поля первого сына корня с оценкой, равной оценке вершины, если встречается ещё один сын с такой оценкой, то с вероятностью 50% заменяется поле вершины.

## Программная реализация

/\*Функция альфа бета отсечения

Параметры:

1)Указатель на узел дерева

Принцип работы:

1)Если узел является листом, то высчитываем его оценку

2)Если проверены не все сыновья узла, то применяем функция к непроверенным сыновьям

3)Если узел не является корнем, то копируем значения alpha и betta родителя узла

4)Вычисляем значение проверенных сыновей

5)Если проверены все сыновья, то узел проверен

6)Если узел не корень:

6.1) Если родитель находится на чётном уровне (минимума), то корректируем значение betta родителя узла

6.2) Если родитель находится на нечётном уровне (максимума), то корректируем значение alpha родителя узла

6.3) Если у родителя узла alpha <betta:

6.3.1) Значение оценки родителя равно оценке текущего узла

6.3.2) Если у узла есть следующий брат, то переходим к нему

6.3.3) Переходим к родителю текущего узла

6.4) Если у родителя узла alpha> betta:

6.4.1) Переходим к родителю текущего узла

6.4.2) Количество проверенных сыновей = количество сыновей

6.4.3) Узел проверен

\*/

void alpha\_betta(Tree\_node\* PNode) {

if (PNode->height == 5) {

get\_assessment(PNode);

}

if (PNode->count\_check\_child != PNode->count\_child) {

for (int i = PNode->count\_check\_child;

i <PNode->count\_child; i++)

{

alpha\_betta(PNode->child[i]);

}

}

if (PNode->height != 1) {

PNode->alpha = PNode->parent->alpha;

PNode->betta = PNode->parent->betta;

}

for (int i = 0; i < PNode->count\_child; i++) {

if (PNode->child[i]->is\_check == true) {

PNode->count\_check\_child++;

}

}

if (PNode->count\_check\_child == PNode->count\_check\_child) {

PNode->is\_check = true;

}

if (PNode->height != 1) {

if (PNode->parent->height % 2 == 1) {

if (PNode->parent->betta > PNode->assessment)

PNode->parent->betta = PNode->assessment;

}

if (PNode->parent->height % 2 == 0) {

if (PNode->parent->alpha < PNode->assessment) {

PNode->parent->alpha = PNode->assessment;

}

}

if (PNode->parent->alpha < PNode->parent->betta) {

PNode->parent->assessment = PNode->assessment;

if (PNode->next\_brother != nullptr) {

PNode = PNode->next\_brother;

}

else {

PNode = PNode->parent;

}

}

else {

PNode = PNode->parent;

PNode->count\_check\_child = PNode->count\_child;

PNode->is\_check = true;

}

}

}

/\*Функция поиска лучшего хода

Параметры:

1)Указатель на корень дерева

Принцип работы:

1)Проверяем оценку сыновей корня

2)Если значение оценки сына равно оценке корня, то меняем значение поля корня на значение поля текущего сына

3)Если встречается ещё один сын с оценкой, равной оценке корня, то с вероятностью 50%(проверяем остаток случайно сгенерированного числа от деления на 2) заменяем значение поля корня

\*/

void choose\_best\_move(Tree\_node\* PNode) {

int tmp = 0;

srand(unsigned int(time(0)));

for (int k = 0; k < PNode->count\_child; k++) {

if (PNode->child[k]->assessment == PNode->assessment) {

tmp++;

if (tmp == 1) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->field[i][j] =

PNode->child[k]->field[i][j];

PNode->count\_connections[i][j] =

PNode->child[k]->

count\_connections[i][j];

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

PNode->connections[i][j][l] =

PNode->child[k]->

connections[i][j][l];

}

}

}

if (PNode->child[k]->is\_square == true) {

PNode->is\_square = true;

}

}

else {

int r = rand() % 2;

if (r == 0) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->field[i][j] =

PNode->child[k]

->field[i][j];

PNode->count\_connections[i][j] =

PNode->child[k]->

count\_connections[i][j];

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

PNode->connections[i][j][l] =

PNode->child[k]->

connections[i][j][l];

}

}

}

if (PNode->child[k]->is\_square == true) {

PNode->is\_square = true;

}

}

}

}

}

}

# Код Tree.h

#pragma once

#ifndef \_MyTree\_H\_

#define \_MyTree\_H\_

#include<iostream>

#define N 30 //Определение максимального количества

//сыновей

using namespace std;

// Класс узла дерева

struct Tree\_node {

Tree\_node\* child[N]; //Массив указателей на сыновей

Tree\_node\* parent; //Указатель на родителя

Tree\_node\* next\_brother; //Указатель на следующего брата

int field [10][10] ; //Состояние поля игры

int connections[10][10][4]; //Массив клеток, соединённых клеток

//с данной

int count\_connections[10][10]; //Колличество клеток, соединённых

//с данной

int count\_child; //Количество сыновей

int number; //Номер узла

int height; //Высота узла

int assessment; //Оценка текущего состояния игры

int alpha; //Параметр для альфа-бетта отсечения

int betta; //Параметр для альфа-бетта отсечения

int count\_check\_child; //Колличество просмотренных сыновей

//при поиске лучшего хода

bool is\_check; //Был ли полностью рассмотрен узел

//т.е. рассмотрены все его сыновья

bool is\_square; //Показывет был ли образован квадрат

//на этом ходу

Tree\_node() {

for (int i = 0; i < 10; i++){

for (int j = 0; j < 10; j++){

field[i][j] = 0;

count\_connections[i][j] = 0;

connections[i][j][0] = -1;

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

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

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

}

}

number = 0;

height = 0;

count\_child = 0;

count\_check\_child = 0;

alpha = INT16\_MIN;

betta = INT16\_MAX;

assessment = INT16\_MIN;

parent = nullptr;

next\_brother = nullptr;

is\_square = false;

is\_check = false;

}

};

int num = 0; //Счётчик узлов

/\*Функция для создания узла

Параметры функции:

1)Состояние поля

Результат - указатель на узел

\*/

Tree\_node\* newNode(int value[][10] ) {

num++;

Tree\_node\* temp = new Tree\_node;

for (int i = 0; i < 10; i++){

for (int j = 0; j < 10; j++){

temp->field[i][j] = value[i][j];

}

}

temp->number = num;

return temp;

}

/\* Функция для добавления узла в дерево по индексу родителя

Параметры функции:

1)Указатель на корень дерева

2)Индекс родительского узла

3)Значение узла

\*/

void push(Tree\_node\* PNode, int index, int key[][10]) {

if (PNode->number == index) {

if (PNode->parent == nullptr) {

PNode->height = 1;

}

PNode->count\_child++;

PNode->child[PNode->count\_child-1] = newNode(key);

PNode->child[PNode->count\_child-1]->parent = PNode;

PNode->child[PNode->count\_child-1]->height = PNode->height + 1;

if (PNode->count\_child > 1) {

PNode->child[PNode->count\_child - 2]->next\_brother =

PNode->child[PNode->count\_child - 1];

}

}

for (int i = 0; i < PNode->count\_child; i++) {

push(PNode->child[i], index, key);

}

}

/\*Функция удаления дерева

Параметры функции:

1)Указатель на корень дерева

\*/

void deletion(Tree\_node\* PNode) {

for (int i = 0; i < PNode->count\_child; i++) {

deletion(PNode->child[i]);

}

delete PNode;

num = 0;

}

#endif

# Код основного файла

#include<iostream>

#include <cstdlib> // для функций rand() и srand()

#include <ctime> // для функции time()

#include"Tree.h"

using namespace std;

## Функция game\_is\_over()

/\*Функция проверки завершенности игры

Параметры функии:

1)Указатель на узел дерева

Результат - значние логического типа: истина или ложь

Принцип работы:

1)Нулевая гипетеза- игра не закончена.

2)Если в поле встречается "0"-незанятая клетка, то гипотеза опровергается.

\*/

bool game\_is\_over(Tree\_node\* PNode) {

bool res = true;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

if (PNode->field[i][j] == 0) {

res = false;

}

}

}

return res;

}

## Функция check\_cell()

/\*Функция проверки клетки

Параметры:

1)Указатель на узел дерева

2)Значение строки текущей клетки

3)Значение столбца текущей клетки

4)Значение целого типа: 1 или 2- значение человека или компьютера

5)Значение типа char: s или с - направление смещения: по строке или

столбцу

6)Значение целого типа: 1 или -1 - значение смещения

Принцип работы:

1)Копируем значение текущего поля

2)Смотрим значние параметра direction, если он равен s, то смещаемся

на значение параметра step по строке, если directon равен c, то смещаеся по столбцу

3)Если значение рассматриваемой клетки равно переданному и заданная

клетка не с ней имееет связь, то добавляем сына текущему узлу

\*/

bool check\_cell(Tree\_node\* PNode, int string, int column, const int value, char direction, const int step) {

bool flag=false;

int field[10][10];

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

field[i][j] = PNode->field[i][j];

}

}

if (direction == 'c') {

if (field[string + step][column] == 0) {

field[string + step][column] = value;

}

if (field[string + step][column] == value) {

bool res = false;

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

if (PNode->connections[string][column][i] ==

(string + step) \* 10 + column) {

res = true;

}

}

if (res == false) {

flag = true;

push(PNode, PNode->number, field);

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->child[PNode->count\_child - 1]

->count\_connections[i][j] =

PNode->count\_connections[i][j];

PNode->child[PNode->count\_child - 1]

->connections[i][j][0] =

PNode->connections[i][j][0];

PNode->child[PNode->count\_child - 1]

->connections[i][j][1] =

PNode->connections[i][j][1];

PNode->child[PNode->count\_child - 1]

->connections[i][j][2] =

PNode->connections[i][j][2];

PNode->child[PNode->count\_child - 1]

->connections[i][j][3] =

PNode->connections[i][j][3];

}

}

PNode->child[PNode->count\_child - 1]->

connections[string + step][column][PNode->child[PNode

->count\_child - 1]->

count\_connections[string + step][column]] =

string \* 10 + column;

PNode->child[PNode->count\_child - 1]->

connections[string][column][PNode->child[PNode

->count\_child - 1]->count\_connections[string][column]] =

(string + step) \* 10 + column;

PNode->child[PNode->count\_child - 1]->

count\_connections[string][column]++;

PNode->child[PNode->count\_child - 1]->

count\_connections[string + step][column]++;

}

}

}

if (direction == 's') {

if (field[string][column + step] == 0) {

field[string][column + step] = value;

}

if (field[string][column + step] == value) {

bool res = false;

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

if (PNode->connections[string][column][i] ==

string \* 10 + column + step) {

res = true;

}

}

if (res == false) {

flag = true;

push(PNode, PNode->number, field);

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->child[PNode->count\_child - 1]->

count\_connections[i][j] =

PNode->count\_connections[i][j];

PNode->child[PNode->count\_child - 1]->

connections[i][j][0] =

PNode->connections[i][j][0];

PNode->child[PNode->count\_child - 1]->

connections[i][j][1] =

PNode->connections[i][j][1];

PNode->child[PNode->count\_child - 1]->

connections[i][j][2] =

PNode->connections[i][j][2];

PNode->child[PNode->count\_child - 1]->

connections[i][j][3] =

PNode->connections[i][j][3];

}

}

PNode->child[PNode->count\_child - 1]

->connections[string][column + step][PNode->child[PNode

->count\_child - 1]->count\_connections[string]

[column + step]] = string \* 10 + column;

PNode->child[PNode->count\_child - 1]->

connections[string][column][PNode->child[PNode

->count\_child - 1]->count\_connections[string][column]] =

string \* 10 + column + step;

PNode->child[PNode->count\_child - 1]->

count\_connections[string][column]++;

PNode->child[PNode->count\_child - 1]->

count\_connections[string][column + step]++;

}

}

}

return flag;

}

## Функция is\_square\_up ()

/\*Функция проверки того, что на текущем ходу был сформирован квадрат из текущих клеток и клеток, находящихся над ними

Параметры функции:

1)Указатель на узел дерева

2)Номер строки первой клетки

3)Номер столбца первой клетки

4)Номер строки второй клетки

5)Номер столбца второй клетки

Принцип работы:

1)Если у первой клетки есть связь с верхней, то переменная flag\_first=true

2)Если у второй клетки есть связь с верхней, то переменная flag\_second=true

3)Если клетки соеденены с верхними, то проверяем связь верхних клеток между собой

\*/

bool is\_square\_up(Tree\_node\* PNode, int string\_first, int column\_first, int string\_second, int column\_second) {

bool flag\_first = false;

bool flag\_second = false;

bool flag\_up = false;

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

if (PNode->connections[string\_first][column\_first][i] ==

(string\_first - 1) \* 10 + column\_first) {

flag\_first = true;

}

if (PNode->connections[string\_second][column\_second][i] ==

(string\_second - 1) \* 10 + column\_second) {

flag\_second = true;

}

}

if (flag\_second == true && flag\_first == true) {

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

if (PNode->connections[string\_first - 1][column\_first][i] ==

(string\_second - 1) \* 10 + column\_second) {

flag\_up = true;

}

}

}

return flag\_up;

}

## Функция is\_square\_down ()

/\*Функция проверки того, что на текущем ходу был сформирован квадрат из текущих клеток и клеток, находящихся под ними

Параметры функции:

1)Указатель на узел дерева

2)Номер строки первой клетки

3)Номер столбца первой клетки

4)Номер строки второй клетки

5)Номер столбца второй клетки

Принцип работы:

1)Если у первой клетки есть связь с нижней, то переменная flag\_first=true

2)Если у второй клетки есть связь с нижней, то переменная flag\_second=true

3)Если клетки соеденены с нижними, то проверяем связь нижних клеток между собой

\*/

bool is\_square\_down(Tree\_node\* PNode, int string\_first, int column\_first, int string\_second, int column\_second) {

bool flag\_first = false;

bool flag\_second = false;

bool flag\_down = false;

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

if (PNode->connections[string\_first][column\_first][i] ==

(string\_first + 1) \* 10 + column\_first) {

flag\_first = true;

}

if (PNode->connections[string\_second][column\_second][i] ==

(string\_second + 1) \* 10 + column\_second) {

flag\_second = true;

}

}

if (flag\_second == true && flag\_first == true) {

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

if (PNode->connections[string\_first + 1][column\_first][i] ==

(string\_second + 1) \* 10 + column\_second) {

flag\_down = true;

}

}

}

return flag\_down;

}

## Функция is\_square\_right ()

/\*Функция проверки того, что на текущем ходу был сформирован квадрат из текущих клеток и клеток, находящихся справа

Параметры функции:

1)Указатель на узел дерева

2)Номер строки первой клетки

3)Номер столбца первой клетки

4)Номер строки второй клетки

5)Номер столбца второй клетки

Принцип работы:

1)Если у первой клетки есть связь с правой, то переменная flag\_first=true

2)Если у второй клетки есть связь с правой, то переменная flag\_second=true

3)Если клетки соеденены с правыми, то проверяем связь правых клеток между собой

\*/

bool is\_square\_right(Tree\_node\* PNode, int string\_first, int column\_first, int string\_second, int column\_second) {

bool flag\_first = false;

bool flag\_second = false;

bool flag\_right = false;

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

if (PNode->connections[string\_first][column\_first][i] ==

string\_first \* 10 + column\_first + 1) {

flag\_first = true;

}

if (PNode->connections[string\_second][column\_second][i] ==

string\_second \* 10 + column\_second + 1) {

flag\_second = true;

}

}

if (flag\_second == true && flag\_first == true) {

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

if (PNode->connections[string\_first][column\_first + 1][i] ==

string\_second \* 10 + column\_second + 1) {

flag\_right = true;

}

}

}

return flag\_right;

}

## Функция is\_square\_left ()

/\*Функция проверки того, что на текущем ходу был сформирован квадрат из текущих клеток и клеток, находящихся слева

Параметры функции:

1)Указатель на узел дерева

2)Номер строки первой клетки

3)Номер столбца первой клетки

4)Номер строки второй клетки

5)Номер столбца второй клетки

Принцип работы:

1)Если у первой клетки есть связь с левой, то переменная flag\_first=true

2)Если у второй клетки есть связь с левой, то переменная flag\_second=true

3)Если клетки соеденены с леваыми, то проверяем связь левых клеток между собой

\*/

bool is\_square\_left(Tree\_node\* PNode, int string\_first, int column\_first, int string\_second, int column\_second) {

bool flag\_first = false;

bool flag\_second = false;

bool flag\_left = false;

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

if (PNode->connections[string\_first][column\_first][i] ==

string\_first \* 10 + column\_first - 1) {

flag\_first = true;

}

if (PNode->connections[string\_second][column\_second][i] ==

string\_second \* 10 + column\_second - 1) {

flag\_second = true;

}

}

if (flag\_second == true && flag\_first == true) {

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

if (PNode->connections[string\_first][column\_first - 1][i] ==

string\_second \* 10 + column\_second - 1) {

flag\_left = true;

}

}

}

return flag\_left;

}

## Функция is\_square ()

/\*Функция проверки того, что на текущем ходу был сформирован квадрат

Параметры функции:

1)Указатель на узел дерева

2)Номер строки первой клетки

3)Номер столбца первой клетки

4)Номер строки второй клетки

5)Номер столбца второй клетки

Принцип работы:

1)Смотрим номер строки.Если клетки расположены на одной строке:

1.1)Если она последняя,то вызываем функцию is\_square\_up()

1.2)Если она первая,то вызываем функцию is\_square\_down()

1.3)Если строка не крайняя,то вызывем функции is\_square\_up() и is\_square\_down()

2)Смотрим номер столбца. Если клетки в одном столбце:

2.1)Если он последний,то вызываем функцию is\_square\_left()

2.2)Если он первый,то вызываем функцию is\_square\_right()

2.3)Если столбец не крайний, то вызывем функции is\_square\_left() и is\_square\_right()

\*/

void is\_square(Tree\_node\* PNode, int string\_first, int column\_first, int string\_second, int column\_second) {

bool flag = false;

if (string\_first == string\_second) {

if (string\_first == 9) {

flag = is\_square\_up(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) {PNode->is\_square = true;}

}

if (string\_first == 0) {

flag = is\_square\_down(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

}

if (string\_first > 0 && string\_first < 9) {

flag = is\_square\_up(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

flag = is\_square\_down(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

}

}

if (column\_first == column\_second) {

if (column\_first == 9) {

flag = is\_square\_left(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

}

if (column\_first == 0) {

flag = is\_square\_right(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

}

if (column\_first > 0 && column\_first < 9) {

flag = is\_square\_left(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

flag = is\_square\_right(PNode, string\_first, column\_first,

string\_second, column\_second);

if (flag == true) { PNode->is\_square = true; }

}

}

}

## Функция generation\_move()

/\*Функция генерации хода

Параметры функии:

1)Указатель на узел дерева

2)Значение целого типа: 1, если нужно сгенерировать ход для компьютера,

2 - для человека.

Принцип работы:

1)Просматриваем все значения клеток текущего игрового поля

2)Если значение равно переданному, то рассматриваем его позицию.

3)В зависимости от позиции равссматриваем соседнии клетки

4)Ели узел был добвлен, проверяем был ли собрана квадрат на данном ходу

\*/

void generation\_move(Tree\_node\* PNode, const int value) {

bool flag;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

if (PNode->field[i][j] == value) {

if (i == 0 && j == 0) {

flag=check\_cell(PNode, i, j, value, 's', 1);

if (flag == true) {is\_square(PNode->child[PNode->

count\_child-1], i, j,

i, j + 1);}

flag=check\_cell(PNode, i, j, value, 'c', 1);

if (flag == true) {is\_square(PNode->child[PNode->

count\_child-1], i, j,

i+1, j );}

}

if (i == 0 && j == 9) {

flag=check\_cell(PNode, i, j, value, 's', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child-1], i, j,

i, j - 1);}

flag=check\_cell(PNode, i, j, value, 'c', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child-1], i, j, i + 1, j);}

}

if (i == 9 && j == 0) {

flag=check\_cell(PNode, i, j, value, 'c', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i, j, i - 1, j);}

flag=check\_cell(PNode, i, j, value, 's', 1);

if (flag == true) { is\_square(PNode->child[PNode-> count\_child - 1], i,

j, i, j + 1);}

}

if (i == 9 && j == 9) {

flag=check\_cell(PNode, i, j, value, 'c', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i, j, i - 1, j);}

check\_cell(PNode, i, j, value, 's', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i, j, i, j - 1); }

}

if (i == 0 && j > 0 && j < 9) {

flag=check\_cell(PNode, i, j, value, 's', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i, j, i, j - 1); }

flag=check\_cell(PNode, i, j, value, 's', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i, j + 1); }

flag=check\_cell(PNode, i, j, value, 'c', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i + 1, j); }

}

if (i == 9 && j > 0 && j < 9) {

flag=check\_cell(PNode, i, j, value, 's', -1);

if (flag == true) { is\_square(PNode->child[PNode-> count\_child - 1], i,

j, i, j - 1); }

flag=check\_cell(PNode, i, j, value, 's', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i, j + 1); }

flag=check\_cell(PNode, i, j, value, 'c', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i - 1, j); }

}

if (j == 0 && i != 0 && i != 9) {

flag=check\_cell(PNode, i, j, value, 'c', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i, j, i - 1, j); }

flag=check\_cell(PNode, i, j, value, 'c', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i + 1, j); }

flag=check\_cell(PNode, i, j, value, 's', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i, j+1); }

}

if (j == 9 && i != 0 && i != 9) {

flag = check\_cell(PNode, i, j, value, 'c', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i - 1, j); }

flag = check\_cell(PNode, i, j, value, 'c', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i, j, i + 1, j); }

flag = check\_cell(PNode, i, j, value, 's', -1);

if (flag == true) { is\_square(PNode->child[PNode-> count\_child - 1], i,

j, i, j - 1); }

}

if (i != 0 && i != 9 && j != 0 && j != 9) {

flag = check\_cell(PNode, i, j, value, 's', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i, j - 1); }

flag = check\_cell(PNode, i, j, value, 's', 1);

if (flag == true) { is\_square(PNode->child[PNode-> count\_child - 1], i,

j, i, j + 1); }

flag = check\_cell(PNode, i, j, value, 'c', -1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i-1, j); }

flag = check\_cell(PNode, i, j, value, 'c', 1);

if (flag == true) { is\_square(PNode->child[PNode->

count\_child - 1], i,

j, i+1, j); }

}

}

}

}

}

## Функция make\_tree()

/\*Функция постоения дерева возможных ходов

Параметры:

1)Указатель на корень дерева

Принцип работы:

1)Функция повторяется, пока высота дерева не равна 5

2)Если высота текущего дерева нечетная, то следующий ход компьютера

2.1)Если на предыдущем ходу был собран квадрат, то добавляем сына с текущем состоянием поля т.к. человек пропускает ход

2.2)Если на предыдущем ходу не был собран квадрат, то вызываем функцию generation\_move с параметром 2.

3)Если высота текущего дерева четная, то следующий ход человека

3.1)Если на предыдущем ходу был собран квадрат, то добавляем сына с текущем состоянием поля т.к. компьютер пропускает ход

3.2)Если на предыдущем ходу не был собран квадрат, то вызываем функцию generation\_move с параметром 1.

\*/

void make\_tree(Tree\_node\* PNode) {

if (PNode->height % 2 == 1) {

if (PNode->is\_square == false) {

generation\_move(PNode, 1);

}

else {

push(PNode, PNode->number, PNode->field);

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->child[0]->count\_connections[i][j] =

PNode->count\_connections[i][j];

PNode->child[0]->connections[i][j][0] =

PNode->connections[i][j][0];

PNode->child[0]->connections[i][j][1] =

PNode->connections[i][j][1];

PNode->child[0]->connections[i][j][2] =

PNode->connections[i][j][2];

PNode->child[0]->connections[i][j][3] =

PNode->connections[i][j][3];

}

}

}

}

if (PNode->height % 2 == 0) {

if (PNode->is\_square == false) {

generation\_move(PNode, 2);

}

else {

push(PNode, PNode->number, PNode->field);

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->child[0]->count\_connections[i][j] =

PNode->count\_connections[i][j];

PNode->child[0]->connections[i][j][0] =

PNode->connections[i][j][0];

PNode->child[0]->connections[i][j][1] =

PNode->connections[i][j][1];

PNode->child[0]->connections[i][j][2] =

PNode->connections[i][j][2];

PNode->child[0]->connections[i][j][3] =

PNode->connections[i][j][3];

}

}

}

}

if (PNode->height <= 3) {

for (int i = 0; i < PNode->count\_child; i++) {

make\_tree(PNode->child[i]);

}

}

}

## Функция get\_assessement()

/\*Оценочная функция

Параметры:

1)Указатель на узел дерева

Принцип работы:

1)Просматриваются значения клеток текущего поля

2)Если значение клетки равно 1, то она принадлежит компьютеру. К результату прибавляется количество связей этой клетки

3)Если значение клетки равно 2, то она принадлежит человеку. К результату вычитается количество связей этой клетки

\*/

void get\_assessment(Tree\_node\* PNode) {

int res = 0;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

if (PNode->field[i][j] == 1) {

res = res + PNode->count\_connections[i][j];

}

if (PNode->field[i][j] == 2) {

res = res - PNode->count\_connections[i][j];

}

}

}

PNode->assessment = res;

}

## Функция alpha\_betta()

/\*Функция альфа бетта отсечения

Параметры:

1)Указатель на узел дерева

Принцип работы:

1)Если узел является листом, то высчитываем его оценку

2)Если проверены не все сыновья узла, то применяем функция к непроверенным сыновьям

3)Если узел не является корнем, то копируем значения alpha и betta родителя узла

4)Вычисялем значение проверенных сыновей

5)Если проверены все сыновья, то узел проверен

6)Если узел не корень:

6.1)Если родитель находится на чётном уровне (минимума), то корректируем значение betta родителя узла

6.2)Если родитель находится на нечётном уровне (максимума), то корректируем значение alpha родителя узла

6.3)Если у родителя узла alpha<betta:

6.3.1)Значение оценки родителя равно оценке текущего узла

6.3.2)Если у узла есть следующий брат, то переходим к нему

6.3.3)Переходим к родителю текущего узла

6.4)Если у родителя узла alpha>betta:

6.4.1)Переходим к родителю текущего узла

6.4.2)Количетсво проверенных сыновей = количество сыновей

6.4.3)Узел проверен

\*/

void alpha\_betta(Tree\_node\* PNode) {

if (PNode->height == 5) {

get\_assessment(PNode);

}

if (PNode->count\_check\_child != PNode->count\_child) {

for (int i = PNode->count\_check\_child; i <PNode->count\_child;

i++){

alpha\_betta(PNode->child[i]);

}

}

if (PNode->height != 1) {

PNode->alpha = PNode->parent->alpha;

PNode->betta = PNode->parent->betta;

}

for (int i = 0; i < PNode->count\_child; i++) {

if (PNode->child[i]->is\_check == true) {

PNode->count\_check\_child++;

}

}

if (PNode->count\_check\_child == PNode->count\_check\_child) {

PNode->is\_check = true;

}

if (PNode->height != 1) {

if (PNode->parent->height % 2 == 1) {

if (PNode->parent->betta > PNode->assessment)

PNode->parent->betta = PNode->assessment;

}

if (PNode->parent->height % 2 == 0) {

if (PNode->parent->alpha < PNode->assessment) {

PNode->parent->alpha = PNode->assessment;

}

}

if (PNode->parent->alpha < PNode->parent->betta) {

PNode->parent->assessment = PNode->assessment;

if (PNode->next\_brother != nullptr) {

PNode = PNode->next\_brother;

}

else {

PNode = PNode->parent;

}

}

else {

PNode = PNode->parent;

PNode->count\_check\_child = PNode->count\_child;

PNode->is\_check = true;

}

}

}

## Функция choose\_best\_move()

/\*Функция поиска лучшего хода

Параметры:

1)Указатель на корень дерева

Принцип работы:

1)Проверяем оценку сыновей корня

2)Если значение оценки сына равно оценке корня, то меняем значение поля корня на значение поля текущего сына

3)Если встречается ещё один сын с оценкой, равной оценке корня, то с вероятностью 50% (проверяем остаток случайно сгенерированного числа от деления на 2) заменяем значение поля корня

\*/

void choose\_best\_move(Tree\_node\* PNode) {

int tmp = 0;

srand(unsigned int(time(0)));

for (int k = 0; k < PNode->count\_child; k++) {

if (PNode->child[k]->assessment == PNode->assessment) {

tmp++;

if (tmp == 1) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->field[i][j] =

PNode->child[k]->field[i][j];

PNode->count\_connections[i][j] =

PNode->child[k]->count\_connections[i][j];

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

PNode->connections[i][j][l] =

PNode->child[k]->connections[i][j][l];

}

}

}

if (PNode->child[k]->is\_square == true) {

PNode->is\_square = true;}

}else {

int r = rand() % 2;

if (r == 0) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

PNode->field[i][j] =

PNode->child[k]->field[i][j];

PNode->count\_connections[i][j] =

PNode->child[k]->count\_connections[i][j];

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

PNode->connections[i][j][l] =

PNode->child[k]->

connections[i][j][l];

}

}

}

if (PNode->child[k]->is\_square == true) {

PNode->is\_square = true; }

}

}

}

}

}

## Функция Print\_field()

/\*Функция вывода теущего поля

Параметры:

1)Указатель на узел дерева

Принцип работы:

1)Если значение клетки равно 1, то выводим красную точку,2 - синию, 0-белую

2)Если у клетки есть связь с правой, то выводим "-" соответствующего цвета, иначе " "

3)Если у клетки есть связь с нижней, то выводим "|"соответствующего цвета " "

\*/

void Print\_field(Tree\_node\* PNode) {

cout << " \u001B[33m 0 1 2 3 4 5 6 7 8 9\n";

for (int i = 0; i < 10; i++) {

if (i > 0) {

for (int j = 0; j < 10; j++) {

bool f = false;

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

if (PNode->connections[i][j][k] ==(i - 1) \* 10 + j) {

f = true;

}

}

if (f == true) {

if (PNode->field[i][j] == 1) {

cout << "\u001B[31m | ";

}

else {

cout << "\u001B[34m | ";

}

}

else {cout << " ";}

}

cout << "\n";

}

cout << "\u001B[33m" << i << " ";

for (int j = 0; j < 10; j++) {

if (PNode->field[i][j] == 1) {cout << "\u001B[31m.";}

if (PNode->field[i][j] == 2) {cout << "\u001B[34m.";}

if (PNode->field[i][j] == 0) {cout << "\u001B[32m.";}

bool f = false;

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

if (PNode->connections[i][j][k] == i \* 10 + j + 1){

f = true;

}

}

if (f == true) {

if (PNode->field[i][j] == 1) {cout << "\u001B[31m - ";}

else {cout << "\u001B[34m - ";}

}

else {cout << " ";}

}

cout << "\n";

}

cout << "\n";

}

## Функция create\_root()

/\*Функция создания корня дерева

Параметры функции:

1)Матрица состояния текущего игровго поля

Принцип работы:

1)Создание новго узла с переданным значением

2)Высота узла равна 1

\*/

Tree\_node\* creat\_root(int start\_field[10][10]) {

Tree\_node\* root = newNode(start\_field);

root->height = 1;

return root;

}

## Функция re\_create\_root()

/\*Функция пересоздания корня

Параметры:

1)Указатель на корень

Принцип работы:

1)Копирование состояния поля, связей клеток и их количества во временные переменные

2)Удаление дерева по указателю на корень

3)Создание нового корня, по переданному указателю

4)Сохранение значний переданного корня

\*/

Tree\_node \* re\_create\_root(Tree\_node\* root) {

int field[10][10];

int count\_connections[10][10];

int connections[10][10][4];

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

field[i][j] = root->field[i][j];

count\_connections[i][j] = root->count\_connections[i][j];

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

connections[i][j][k] = root->connections[i][j][k];

}

}

}

deletion(root);

root = creat\_root(field);

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

root->count\_connections[i][j] = count\_connections[i][j];

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

root->connections[i][j][k] = connections[i][j][k];

}

}

}

return root;

}

## Функция read\_human\_move()

/\*Функция считывания хода человека

Параметры функии:

1)Указатель на узел дерева

Принцип работы:

1)Считываются номера строки и столбца начальной клетки.

2)Если они выходят за диапозон [0;9], то сообщается об ошибке, функция перезапускается.

3)Считываются номера строки и столбца целевой клетки.

4)Если они выходят за диапозон [0;9], то сообщается об ошибке, функция перезапускается.

5)Если целевая и начльная клетка не соседние, то сообщается об ошибке, функция перезапускается.

6)Значение поля в целевой клетке заполняется значением "2".

7)К количествам связей начальной о целевой клеток добаляется единица.

8)Значения строк и столбцов начальной и целевой клеток записываются в соответствующие массивы соединённых клеток.

\*/

void read\_human\_move(Tree\_node\* PNode) {

int start\_string = 0, start\_column = 0, finish\_string = 0, finish\_column = 0;

cout << "\u001B[37mВведите строку начальной клетки : ";

cin >> start\_string;

cout << "Введите столбец начальной клетки : ";

cin >> start\_column;

if (start\_column < 10 && start\_column >= 0 && start\_string < 10 &&

start\_string >= 0) {

if (PNode->field[start\_string][start\_column] == 2) {

cout << "Введите строку целевой клетки : ";

cin >> finish\_string;

cout << "Введите столбец целевой клетки : ";

cin >> finish\_column;

if (finish\_column < 10 && finish\_column >= 0 &&

finish\_string < 10 && finish\_string >= 0) {

if (PNode->field[finish\_string][finish\_column] != 1) {

if (abs(start\_string - finish\_string) == 1 &&

start\_column == finish\_column ||

abs(start\_column - finish\_column) == 1 &&

start\_string == finish\_string)

{

PNode->field

[finish\_string][finish\_column]=2;

PNode->count\_connections

[start\_string][start\_column]++;

PNode->count\_connections

[finish\_string][finish\_column]++;

int i = 0;

while (PNode->connections

[start\_string][start\_column][i]

!= -1) {

i++;

}

PNode->connections

[start\_string][start\_column][i] =

finish\_string \* 10 + finish\_column;

i = 0;

while (PNode->connections

[finish\_string][finish\_column][i]

!= -1) {

i++;

}

PNode->connections

[finish\_string][finish\_column][i] =

start\_string \* 10 + start\_column;

is\_square(PNode, start\_string,

start\_column, finish\_string,

finish\_column);

}

else {

cout << "\u001B[31m\tЭту клетку нельзя"

<< "соеденить\n"

<< "\tОт начальной клетки до нее не

<< достать\n\n";

read\_human\_move(PNode);

}

}

else {

cout << "\u001B[31m\tЭту клетку нельзя соеденить\n"

<< "\tОна принадлежит сопернику\n\n";

read\_human\_move(PNode);

}

}

else {

cout << "\u001B[31m\tЦелевая клетка вне допустимого"

<<"диапозона\n"

<< "\tЗначния строки и столбца должны лежать в"

<<"интервале от 0 до 9\n\n";

read\_human\_move(PNode);

}

}

else {

cout << "\u001B[31m\tЭта клетка не принадлежит вам\n\n";

read\_human\_move(PNode);

}

}

else {

cout << "\u001B[31m\tНачальная клетка вне допустимого" <<"диапозона\n"

<< "\tЗначния строки и столбца должны лежать в интервале от 0 до"

<<"9\n\n";

read\_human\_move(PNode);

}

}

## Функция game()

/\*Функция игры

Параметры:

1)Указатель на узел дерева

Принцип работы:

1)Функция повторяется пока игра не окончена

2)i- показатель того, что компьютер собрал квадрат

3)Если компьютером был собран квадрат,то выводим соответствующую надпись

4)Выводим поле игры

4)Считываем ход человека

5)Если он собрал квадрат - повторяем пункты 4)-5)

6)Строим дерево возможных ходов

7)Производим алгоритм альфа-бетта отсечения

8)Выбираем лучший ход

9)Если был собран квадрат- повторяем пункты 6)-9)

\*/

Tree\_node\* game(Tree\_node\* root) {

int i = 0;

while (game\_is\_over(root) == false) {

system("cls");

if (i > 0) {

cout << "\u001B[35mКомпьютер собрал квадрат \n";

i = 0;

}

Print\_field(root);

read\_human\_move(root);

while (root->is\_square == true) {

system("cls");

Print\_field(root);

cout << "\u001B[35mВы собрали квадрат\n";

root->is\_square = false;

read\_human\_move(root);

}

make\_tree(root);

alpha\_betta(root);

choose\_best\_move(root);

while (root->is\_square == true) {

root = re\_create\_root(root);

root->is\_square = false;

make\_tree(root);

alpha\_betta(root);

choose\_best\_move(root);

i++;

}

root = re\_create\_root(root);

}

return root;

}

## Функция get\_result()

/\*Функция вывода результата игры

Параметры:

1)Указатель на корень дерева

Принцип работы:

1)Подсчёт баллов игрока и компьютера

2)Вывод результата

\*/

void get\_result(Tree\_node\* root) {

int score\_c = 0, score\_h = 0;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

if (root->field[i][j] == 1) { score\_c++; }

if (root->field[i][j] == 2) { score\_h++; }

}

}

if (score\_h > score\_c) {

cout << "Вы выграли! Ваш счёт " << score\_h

<< " Счёт компьютера " << score\_c << "\n";

}

if (score\_h < score\_c) {

cout << "Вы проиграли( Ваш счёт " << score\_h

<< " Счёт компьютера " << score\_c << "\n";

}

if (score\_h == score\_c) {

cout << "Ничья! Ваш счёт " << score\_h

<< " Счёт компьютера " << score\_c << "\n";

}

}

## Функция main()

int main() {

setlocale(LC\_ALL, "ru"); //Включение русского языка

int field[10][10];

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

field[i][j] = 0;

}

}

field[9][0] = 1;

field[0][9] = 2;

Tree\_node\* root;

root = creat\_root(field);

root = game(root);

Print\_field(root);

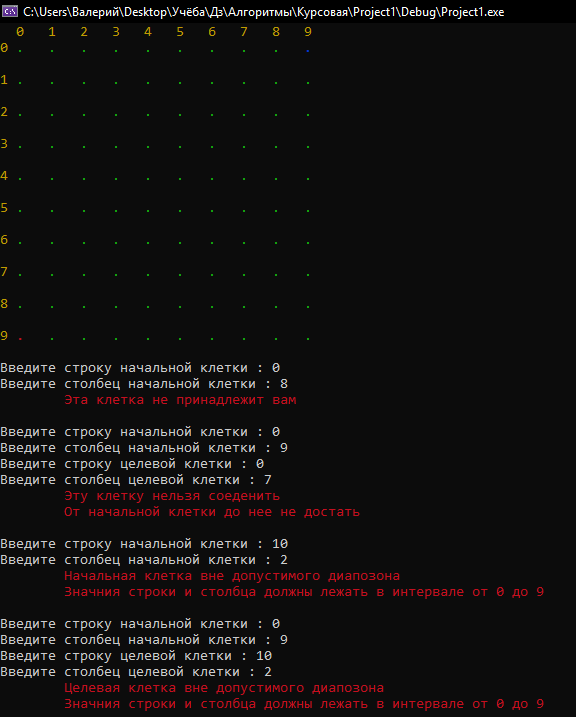
get\_result(root);

return 0;

}

# Результаты работы программы

## Ввод некорректных данных



## Корректная работа программы









