

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра информационной безопасности**

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Программирование»**  
**Тема: Игра «Крестики-нолики»**

Студент гр. 0361

\_\_\_\_\_

Бушуев Д.И.

Преподаватель

\_\_\_\_\_

Халиуллин Р.А.

Санкт-Петербург

2020

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент Бушуев Д.И.

Группа 0361

Тема работы: игра «Крестики-нолики»

Исходные данные: Разработать на языке программирования С или С++ (по выбору студента) приложение для игры в «крестики-нолики». Игра должна проводиться на игровом поле классического размера (3 на 3), в режиме «пользователь против приложения». Право первого хода может предоставляться пользователю, приложению или распределяться случайным образом, по выбору студента. Приложение должно анализировать текущую ситуацию на игровом поле и выбирать ход таким образом, чтобы стремиться к выигрышу партии. Если по результатам анализа найдено несколько возможных ходов, то приложение должно выбирать ход из нескольких возможных ходов с помощью генератора псевдослучайных чисел. Приложение должно иметь консольный или графический интерфейс, по выбору студента. Интерфейс приложения должен быть интуитивно понятным и содержать подсказки для пользователя. В исходном коде приложения должны быть реализованы проверки аргументов реализованных функций и проверки возвращаемых функциями значений (для всех функций, как сторонних, так и реализованных). Приложение должно корректно обрабатывать ошибки, в том числе ошибки ввода/вывода, выделения/освобождения памяти и т. д.

Содержание пояснительной записки: Введение, теоретическая часть, определение, история игры крестики-нолики, классический вариант 3x3, правила игры, анализ, реализация программы, основные сведения о программном обеспечении, реализованные функции, результаты тестирования программы, заключение, список использованных источников, приложение 1 – руководство пользователя, приложение 2 –блок-схема алгоритма, приложение 3 – исходный код программы.

Предполагаемый объем пояснительной записки: Не менее 15 страниц.

Дата выдачи задания: 04.10.2020

Дата сдачи реферата: 24.12.2020

Дата защиты реферата: 30.12.2020

Студент

\_\_\_\_\_

Бушуев Д.И.

Преподаватель

\_\_\_\_\_

Халиуллин Р.А.

## **АННОТАЦИЯ**

Задачей курсовой работы является создание приложения для игры в «Крестики-нолики» на языке программирования C. Приложение должно иметь консольный интерфейс, возможность выбора постановки хода в одну из ячеек. Приложение должно корректно обрабатывать все ошибки (ошибки ввода/вывода, выделения/освобождения памяти).

## **SUMMARY**

The objective of the course work is to create an application for playing "Tic-Tac-Toe" in the C programming language. The application must have a console interface, the ability to select a move to one of the cells. The application must correctly handle all errors (I / O errors, memory allocation / deallocation).

## СОДЕРЖАНИЕ

	Введение	6
1.	Теоретическая часть	7
1.1.	Определение	7
1.2.	История игры «Крестики-нолики»	7
1.3.	Классический вариант 3x3	7
1.3.1	Правила игры	7
1.3.2	Анализ	7
1.3.3	Дерево игровых ситуаций	7
2.	Реализация программы	8
2.1.	Основные сведения о программном обеспечении	8
2.2.	Реализованные функции	8
3.	Результаты тестирования программы	12
	Заключение	14
	Список использованных источников	15
	Приложение 1. Руководство пользователя	16
	Приложение 2. Блок-схема алгоритма	18
	Приложение 3. Исходный код программы	19

## **ВВЕДЕНИЕ**

Основной задачей курсовой работы является создание приложения для игры в «Крестики-нолики» на языке программирования C. Это приложение нужно для реализации корректной обработки компьютером выигрышной стратегии.

Для успешного выполнения курсовой работы мне потребовалось воспользоваться источниками информации, найденными мной в сети Интернет. После написания приложения потребовалось протестировать работу и проверить ее с помощью одной из сред программирования. В результате выполнения курсовой работы были получены навыки работы с различными видами команд на языке C, а также методы оптимизированного ведения кода.

## **1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

### **1.1. Определение**

Крестики-нолики— логическая игра между двумя противниками на квадратном поле 3 на 3 клетки или большего размера (вплоть до «бесконечного поля»). Один из игроков играет «крестиками», второй — «ноликами».

### **1.2. История игры крестиков-ноликов**

В России изначально игра «Крестики-нолики» называлась по-другому, а именно «Херики-оники». Ее называли так вплоть до орфографической реформы 1918 года. В то время она была самой древней и самой популярной игрой. Происхождение этого развлечения точно не известно.

### **1.3. Классический вариант 3х3**

В классическом варианте игры поле состоит из 9 ячеек, составляющих квадрат 3х3.

#### **1.3.1. Правила игры**

Игроки по очереди ставят на свободные клетки поля 3х3 знаки (один всегда крестики, другой всегда нолики). Первый, выстроивший в ряд 3 своих фигуры по вертикали, горизонтали или диагонали, выигрывает.

#### **1.3.2. Анализ**

Для каждой из сторон общеизвестны алгоритмы, которые гарантируют ничью при любой игре противника, а при его ошибке позволяют выиграть.

Ниже приведены некоторые из таких стратегий. Считается, что игрок всегда соблюдает два правила, имеющие приоритет над всеми остальными:

1. Если игрок может немедленно выиграть, он это делает.
2. Если игрок не может немедленно выиграть, но его противник мог бы немедленно выиграть, сделав ход в какую-то клетку, игрок сам делает ход в эту клетку, предотвращая немедленный проигрыш.

#### **1.3.3 Дерево игровых ситуаций**

Дерево игровых ситуаций для игры крестики-нолики, где игрок за «крестики» ходит первым и поступает по приведенному выше алгоритму, а игрок за «нолики» может поступать как угодно состоит из 50-ти узлов.

## **2. РЕАЛИЗАЦИЯ ПРОГРАММЫ**

### **2.1. Основные сведения о программном обеспечении**

Курсовая работа выполнена мной на языке программирования C. Использовалась операционная система Windows и среда разработки Visual Studio Community 2019 версии 16.8.2 с установленным компонентом «Разработка классических приложений на C++», компилятор — Microsoft Visual C++.

### **2.2. Реализованные функции**

#### **1) Функция main**

Функция main выводит на экран инструкцию по работе с программой, пустое поле для игры (с помощью вызова функции visualize()), а затем в цикле do {...} while (...) поочередно вызывает другие функции, указанные ниже. Исходный код функции находится в файле code.c.

Объявление функции:

```
int main();
```

Тип функции: int.

Аргументы функции: функция аргументов не принимает.

Возвращаемое значение:

- 0 – Если функция выполнена успешно;

Автор функции: Иконников Степан и Бушуев Данил

#### **2) Функция visualize**

Функция visualize выводит на экран игровое поле. Исходный код функции находится в файле code.c.

Объявление функции:

```
void visualize();
```

Тип функции: void.

Аргументы функции: функция аргументов не принимает.

Возвращаемое значение: функция значений не возвращает.

Автор функции: Иконников Степан и Бушуев Данил

#### **3) Функция Alive\_move**



Функция `Alive_more` запрашивает ход игрока и ставит крестик в клетку с введенными координатами. Если входные данные будут некорректными, функция выведет сообщение об ошибке и попросит пользователя повторить ход, обратившись к себе же. Исходный код функции находится в файле `code.c`.

Объявление функции:

```
void Alive_move();
```

Тип функции: `void`.

Аргументы функции: функция аргументов не принимает.

Возвращаемое значение: функция аргументов не возвращает.

Автор функции: Иконников Степан и Бушуев Данил

#### 4) Функция `char win_check`

Функция `win_check` проверяет наличие выигрышных позиций на доске. Исходный код функции находится в файле `code.c`.

Объявление функции:

```
char win_check();
```

Тип функции: `char`.

Аргументы функции: Функция аргументов не принимает.

Возвращаемое значение:

- 'X' — если найдена выигрышная позиция для крестиков.
- 'O' — если найдена выигрышная позиция для ноликов.
- ' ' (пробел) — если выигрышных ситуаций не найдено

Автор функции: Иконников Степан и Бушуев Данил

#### 5) Функция `panic`

Функция `panic()` проверяет игровое поле на наличие предвыигрышных позиций для программы и игрока. Следует учесть, что в функции сначала проверяются пред выигрышные позиции программы, а только затем предвыигрышные позиции игрока, поэтому в случае, когда расположение элементов на поле будет таково, что программа может либо выиграть, либо заблокировать победный ход игрока, то функция вернет значение,

соответствующее выигрышному ходу. Исходный код функции находится в файле code.c.

Объявление функции:

```
int panic();
```

Тип функции: int.

Аргументы функции: Функция не принимает аргументов.

Возвращаемое значение:

- 0 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][1] И [0][2]); ([1][1] И [2][2]); ([1][0] И [2][0]);
- 1 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([1][1] И [2][1]); ([0][0] И [0][2]);
- 2 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][0] И [0][1]); ([2][0] И [1][1]); ([1][2] И [2][2]);
- 3 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][0] И [2][0]); ([1][1] И [1][2]);
- 5 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][2] И [2][2]); ([1][0] И [1][1]);
- 6 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][0] И [1][0]); ([0][2] И [1][1]); ([2][1] И [2][2]);
- 7 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][1] И [1][1]); ([2][0] И [2][2]);
- 8 – если только нулями или только крестами заняты клетки с координатами из следующего списка: ([0][0] И [1][1]); ([2][0] И [2][1]); ([0][2] И [1][2]);
- 10 – если не встретилось ни одной из вышеперечисленных ситуаций.

Автор функции: Иконников Степан и Бушуев Данил

6) Функция AI\_move

Функция `AI_move` принимает ход программы и проверяет игровое поле на ничью. Реализовать проверку на ничью было удобно в ходе компьютера т.к. последний ход в игре совершается именно программой. В случае, если найдена ничья программа выведет сообщение о ничьей и завершит выполнение программы. Исходный код функции находится в файле `code.c`.

Объявление функции:

```
void AI_move();
```

Тип функции: `void`.

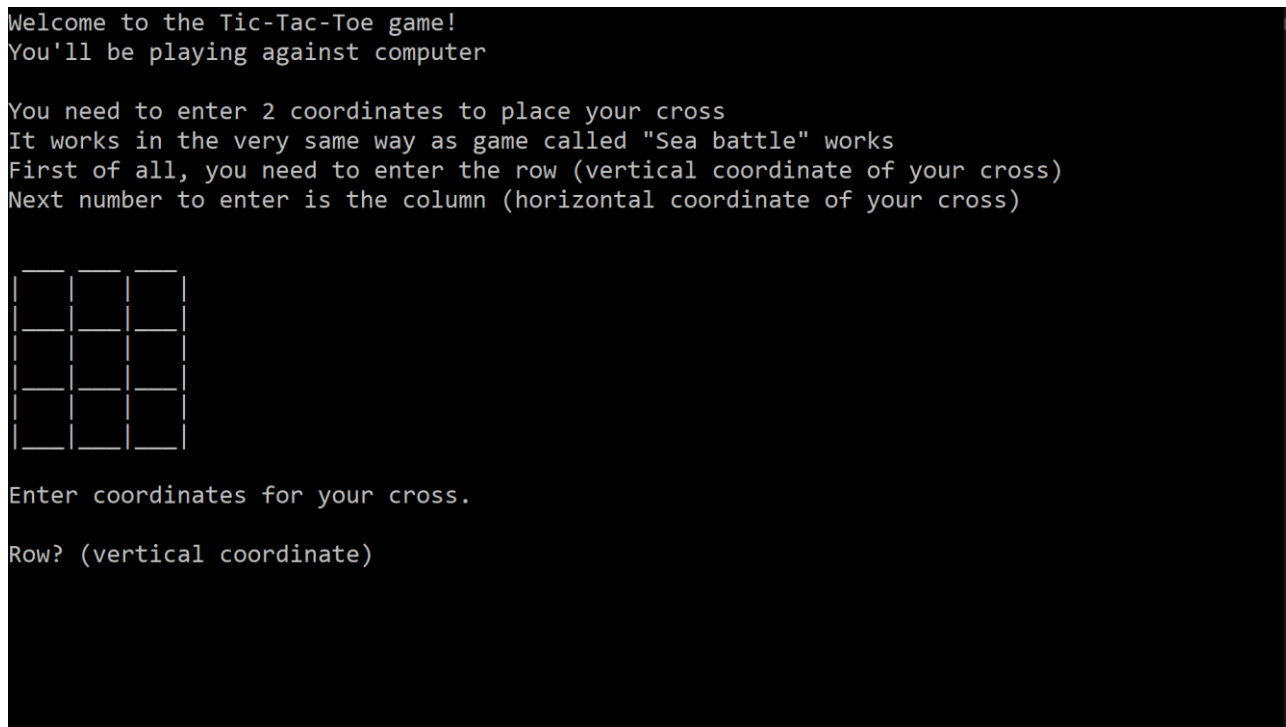
Аргументы функции: Функция аргументов не принимает

Возвращаемое значение: Функция значений не возвращает

Автор функции: Иконников Степан и Бушуев Данил

### 3. РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ ПРОГРАММЫ

При запуске программы, на экран выводится сообщение о том, что первый ход предоставляется игроку, нужно ввести координаты точки на поле, сначала номер строки (1-3), а потом номер столбца (1-3), изображено на рисунке 1.



```
Welcome to the Tic-Tac-Toe game!
You'll be playing against computer

You need to enter 2 coordinates to place your cross
It works in the very same way as game called "Sea battle" works
First of all, you need to enter the row (vertical coordinate of your cross)
Next number to enter is the column (horizontal coordinate of your cross)

|_|_|_|
|_|_|_|
|_|_|_|
|_|_|_|

Enter coordinates for your cross.
Row? (vertical coordinate)
```

Рисунок 1 – Запуск программы

От пользователя требуется ввести цифру от 1 до 3 – номер строки. А затем требуется ввести цифру от 1 до 3 – номер столбца. Если же, игрок введет координаты не из промежутка от 1 до 3, любой другой символ или уже занятую ячейку, программа попросит ввести ход корректно, изображено на рисунке 2.

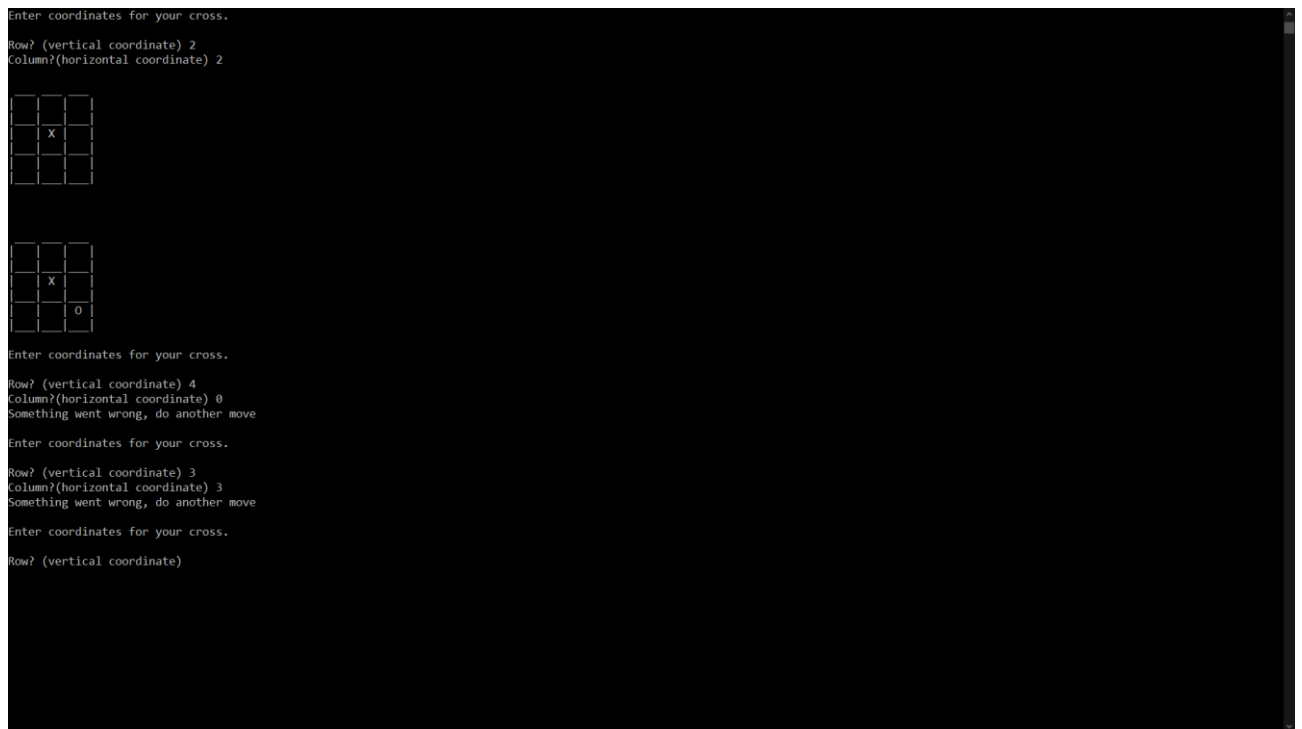


Рисунок 2 – Пример некорректного ввода координат ячейки поля

После того как игрок введет координаты нужной ему точки. Поле с отображенным ходом выведется на экран. Ниже выведется поле с ходом компьютера, изображено на рисунке 3.

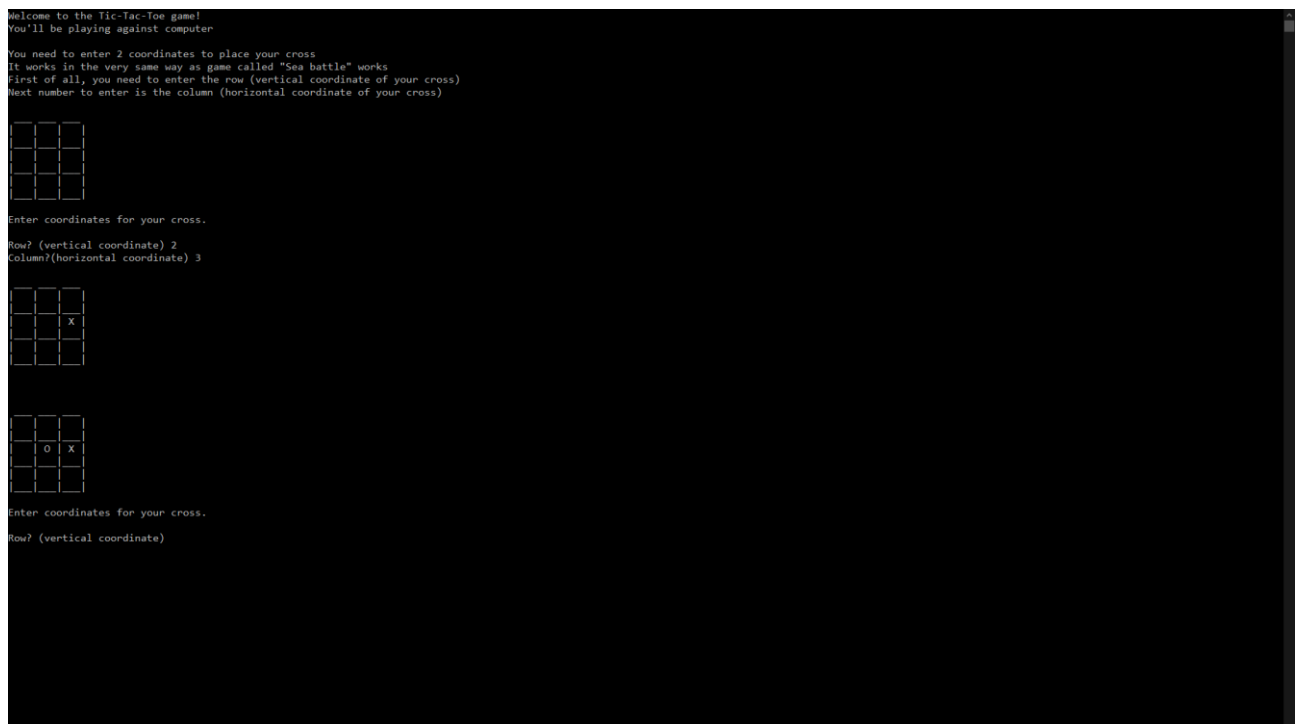


Рисунок 3 – Пример ввода координат пользователем

Далее игрок должен будет ввести координаты следующей точки, и так до момента пока игра не завершится победой, ничьей или проигрышем игрока. На рисунках 4 и 5 показан результат игры в ничью и проигрыш игрока соответственно.



Рисунок 4 – Пример результата игры в ничью



Рисунок 5 – Пример результата игры, в которой игрок проигрывает

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной курсовой работы на языке программирования С был написана игра «Крестики-нолики», которая имеет консольный интерфейс. Программа может считывать команды пользователя с клавиатуры посредством ввода координат нужной игроку точки, выводить на экран результат введенного хода и результат ответного хода компьютера. Реализована корректная обработка ошибок, которые могут возникнуть в ходе выполнения программы

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Allis, L. V. Searching for solutions in games and artificial intelligence, – Ph.D. Thesis, University of Limburg, Maastricht., 1994. — 225 с.
2. Керниган Б., Ритчи. Д. Язык программирования Си. — Санкт-Петербург: Невский диалект, 2000. — 352 с.
3. Standard C Library Functions Table, By Name [Электронный ресурс]. – URL: [https://www.ibm.com/support/knowledgecenter/ssw\\_ibm\\_i\\_74/rtref/stalib.htm](https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rtref/stalib.htm) (дата обращения: 21.12.2020).
4. The Open Group Technical Standard Base Specifications [Электронный ресурс]. – URL: <https://pubs.opengroup.org/onlinepubs/9699919799/> (дата обращения: 13.12.2020).
5. C/C++ Refences – All C Functions [Электронный ресурс]. – URL: [https://doc.bccnsoft.com/docs/cppreference\\_en/all\\_c\\_functions.html](https://doc.bccnsoft.com/docs/cppreference_en/all_c_functions.html) (дата обращения: 13.12.2020).
6. Свободная энциклопедия [Электронный ресурс] – URL: <https://en.wikipedia.org/wiki/Tic-tac-toe> (дата обращения: 19.12.2020)



## ПРИЛОЖЕНИЕ 1

### РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа «Крестики-Нолики».

Программа считывает с клавиатуры координаты клетки, в которую необходимо поместить крестик (ход игрока), а затем отвечает на ход игрока своим ходом (ход компьютера). Перед началом игры программа выведет инструкцию на английском языке. Ходы совершаются по очереди, право первого хода дано игроку.

Минимальные системные требования:

- Процессор: как минимум 1ГГц или SoC;
- ОЗУ: 1 Гб (для 32-разрядных систем) или 2 Гб (для 64-разрядных систем);
- Место на жестком диске: 16 Гб (для 32-разрядных систем) или 20 Гб (для 64-разрядных систем);
- Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0;
- Дисплей: 800 x 600;
- Операционная система: Windows 10.
- Среда разработки, например Visual Studio Community последней версии.

Установка программы.

Установка не требуется, нужно запустить исполняемый файл code.exe, скопировав его в любую директорию.

Запуск программы.

Запустите файл code.exe.

Работа с программой.

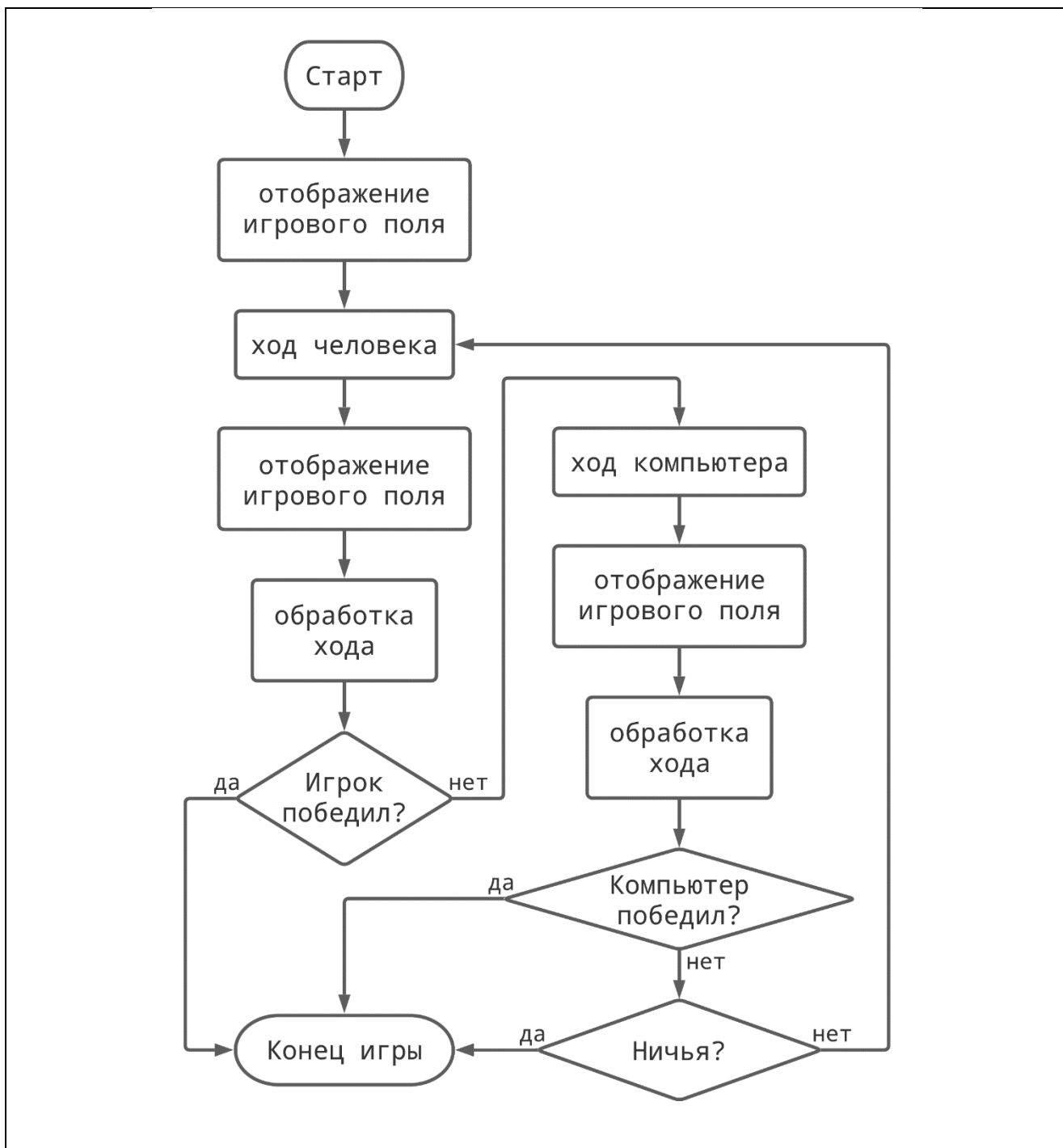
Программа выведет инструкцию на английском языке, затем выведет пустое поле и каждый Ваш ход (ход человека) будет запрашивать координаты ячейки, в которую Вы хотите походить, запросив сначала вертикальное положение клетки (row), а затем горизонтальное (column). После каждого запроса вы должны ввести координату - число от 1 до 3.

Если на середине своего хода (после ввода вертикального положения клетки) Вы поняли, что ошиблись – введите в качестве горизонтальной координаты любой символ, не являющийся числом от 1 до 3, программа выдаст сообщение об ошибке и попросит Вас повторить ход.

В конце игры программа выведет сообщение об окончании игры и ее результат.

## ПРИЛОЖЕНИЕ 2

### БЛОК СХЕМА АЛГОРИТМА



					<i>Приложение 2. Блок-схема алгоритма</i>		
Изм.	Лис.	№ докум	Подпись	Дата			
Разраб	Бушурев Д.И.					Литера	Лист
Пров	Халиуллин Р.А					у	1
Реценз	Халиуллин Р.А						
Н.Контр.	Халиуллин Р.А						
Утв	Халиуллин Р.А						

## ПРИЛОЖЕНИЕ 3

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

#define ABYSS ' '
char board[3][3] = { {ABYSS, ABYSS, ABYSS},{ABYSS, ABYSS, ABYSS},{ABYSS,
ABYSS, ABYSS} };

void AI_move(), Alive_move(), visualize();
char win_check();

int main()
{
    char win;
    printf("Welcome to the Tic-Tac-Toe game!\nYou'll be playing against
computer\n\nYou need to enter 2 coordinates to place your cross\nIt works
in the very same way as game called \"Sea battle\" works\nFirst of all,
you need to enter the row (vertical coordinate of your cross)\nNext
number to enter is the column (horizontal coordinate of your cross)\n");
    win = ABYSS;
    visualize();
    do {

        Alive_move();
        visualize();
        win = win_check();
        if (win != ABYSS) break;
        AI_move();
        visualize();
        win = win_check();
    } while (win == ABYSS);
    if (win == 'X') printf("You won\n");
    else printf("You lost\n");
    visualize();
    return 0;
}

int panic()
{
    char* s;
    s = (char*)board;
    int mirage[] = { 0,0,0,0,0,0,0,0,0 };
    for (int i = 0; i < 9; i++)
    {
        if (s[i] == 'X')
```

```

        {
            mirage[i] = 1;
        }
    }

    printf("\n");
    if ((s[1] == '0' && s[2] == '0' && !mirage[0]) || (s[4] == '0' &&
s[8] == '0' && !mirage[0]) || (s[3] == '0' && s[6] == '0' && !mirage[0]))
    { return 0; }
    if ((s[0] == '0' && s[2] == '0' && !mirage[1]) || (s[4] == '0' &&
s[7] == '0' && !mirage[1])) { return 1; }
    if ((s[0] == '0' && s[1] == '0' && !mirage[2]) || (s[6] == '0' &&
s[4] == '0' && !mirage[2]) || (s[8] == '0' && s[5] == '0' && !mirage[0]))
    { return 2; }
    if ((s[4] == '0' && s[5] == '0' && !mirage[3]) || (s[0] == '0' &&
s[6] == '0' && !mirage[3])) { return 3; }
    if ((s[3] == '0' && s[4] == '0' && !mirage[5]) || (s[2] == '0' &&
s[8] == '0' && !mirage[5])) { return 5; }
    if ((s[0] == '0' && s[3] == '0' && !mirage[6]) || (s[2] == '0' &&
s[4] == '0' && !mirage[6]) || (s[7] == '0' && s[8] == '0' && !mirage[6]))
    { return 6; }
    if ((s[6] == '0' && s[8] == '0' && !mirage[7]) || (s[1] == '0' &&
s[4] == '0' && !mirage[7])) { return 7; }
    if ((s[6] == '0' && s[7] == '0' && !mirage[8]) || (s[0] == '0' &&
s[4] == '0' && !mirage[8]) || (s[2] == '0' && s[5] == '0' && !mirage[8]))
    { return 8; }
    if ((mirage[1] && mirage[2] && s[0] == ABYSS) || (mirage[3] &&
mirage[6] && s[0] == ABYSS) || (mirage[4] && mirage[8] && s[0] == ABYSS))
    { return 0; }
    if ((mirage[0] && mirage[2] && s[1] == ABYSS) || (mirage[4] &&
mirage[7] && s[1] == ABYSS)) { return 1; }
    if ((mirage[0] && mirage[1] && s[2] == ABYSS) || (mirage[6] &&
mirage[4] && s[2] == ABYSS) || (mirage[8] && mirage[5] && s[0] == ABYSS))
    { return 2; }
    if ((mirage[4] && mirage[5] && s[3] == ABYSS) || (mirage[0] &&
mirage[6] && s[3] == ABYSS)) { return 3; }
    if ((mirage[3] && mirage[4] && s[5] == ABYSS) || (mirage[2] &&
mirage[8] && s[5] == ABYSS)) { return 5; }
    if ((mirage[0] && mirage[3] && s[6] == ABYSS) || (mirage[2] &&
mirage[4] && s[6] == ABYSS) || (mirage[7] && mirage[8] && s[6] == ABYSS))
    { return 6; }
    if ((mirage[6] && mirage[8] && s[7] == ABYSS) || (mirage[1] &&
mirage[4] && s[7] == ABYSS)) { return 7; }
    if ((mirage[6] && mirage[7] && s[8] == ABYSS) || (mirage[0] &&
mirage[4] && s[8] == ABYSS) || (mirage[2] && mirage[5] && s[8] == ABYSS))
    { return 8; }
    return 10;
}

void Alive_move()
{

```

```

int x, y;
char coordsX[10], coordsY[10];
printf("Enter coordinates for your cross.\n\n");
printf("Row? (vertical coordinate) ");
scanf("%s", coordsX);
coordsX[strlen(coordsX)] = '\0';
if (strlen(coordsX) != 1 || !isdigit(coordsX[0]))
{
    printf("Something went wrong, do another move\n\n");
    Alive_move();
    return;
}
else
{
    x = atoi(coordsX) - 1;
}
printf("Column?(horizontal coordinate) ");
scanf("%s", coordsY);
if (strlen(coordsY) != 1 || !isdigit(coordsY[0]))
{
    printf("Something went wrong, do another move\n\n");
    Alive_move();
    return;
}
else {
    y = atoi(coordsY) - 1;
}

if (x < 0 || y < 0 || x>2 || y>2 || board[x][y] != ABYSS ||
board[x][y] == 'X')
{
    printf("Something went wrong, do another move\n\n");
    Alive_move();
}
else
{
    board[x][y] = 'X';
    return;
}
}

void AI_move()
{
    int d9;
    int i, stime;
    long ltime;
    char* s;
    s = (char*)board;
    int boardScan = panic();
    if (boardScan < 10 && s[boardScan] == ABYSS)
    {

```

```

        s[boardScan] = '0';
        return;
    }

    if (s[4] == ABYSS) {
        s[4] = '0';
        return;
    }
    if (s[4] != ABYSS) {
        ltime = time(NULL);
        stime = (unsigned int)ltime / 2;
        srand(stime);
        d9 = rand() % 9;

        if (s[d9] != ABYSS || d9 % 2 != 0) {
            while (d9 % 2 != 0 || s[d9] != ABYSS) { if (s[0] == ABYSS ||
s[2] == ABYSS || s[4] == ABYSS || s[6] == ABYSS || s[8] == ABYSS) { d9 =
rand() % 9; } else { break; } }
        }
        if (s[d9] != ABYSS || d9 % 2 == 0) {
            while (d9 % 2 == 0 || s[d9] != ABYSS) { if (s[0] != ABYSS &&
s[2] != ABYSS && s[4] != ABYSS && s[6] != ABYSS && s[8] != ABYSS) { if
(s[1] == ABYSS || s[3] == ABYSS || s[5] == ABYSS || s[7] == ABYSS) { d9 =
rand() % 9; } else { break; } } } else { break; } }
        }

        if (s[d9] == ABYSS) {
            s[d9] = '0';
            return;
        }

        for (i = 0; *s != ABYSS && i < 9; ++i) { s++; }
        if (i == 9)
        {
            printf("Draw\n");
            visualize();
            exit(0);
        }

        return;
    }
}

void visualize()
{
    int t;
    printf("\n ____ _\n");
    for (t = 0; t < 3; t++)
    {

```

```

        printf("| %c | %c | %c |\n", board[t][0], board[t][1],
board[t][2]);
        printf("|__|__|__|\n");
    }
    printf("\n");
}

char win_check()
{
    int i;
    char* s;
    for (i = 0; i < 3; i++) {
        s = &board[i][0];
        if (*s == *(s + 1) && *(s + 1) == *(s + 2)) return *s;
    }
    for (i = 0; i < 3; i++) {
        s = &board[0][i];
        if (*s == *(s + 3) && *(s + 3) == *(s + 6)) return *s;
    }

    if (board[0][0] == board[1][1] && board[1][1] == board[2][2]) {
return board[0][0]; }
    if (board[0][2] == board[1][1] && board[1][1] == board[2][0]) {
return board[0][2]; }
    return ABYSS;
}

```