Министерство цифрового развития, связи и массовых коммуникаций
«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 «Информатика и вычислительная техника»
профиль «Программное обеспечение средств вычислительной техники и автоматизированных систем»

РГР

по дисциплине «Программирование»

Вариант №25

Выполнил:

Студент гр. ИП-311 ___/Подкорытова А.В./

«__» _____2024г.

Проверил:

Старший преподаватель кафедры ПМиК _____/Агалаков А.А./

«___» _____2024 г.

Новосибирск, 2024 г.

Оглавление

Задание

25. Написать программу для игры с компьютером в крестики-нолики. Программа не должна проигрывать.

Теория

В данной работе самым главным является алгоритм поиска наилучшего хода. В программе был использован алгоритм альфа-бета отсечения (англ. Alpha-beta pruning). Данный алгоритм является улучшенной версией алгоритма Минимакс (англ. minimax).

Минимакс. Ключевая идея алгоритма применительно к играм состоит в поиске наилучшего хода. На рис. 1 изображена ситуация, в которой игра в «крестики-нолики» близится к завершению, а ход предстоит сделать игроку X. Алгоритм рассматривает три доступных хода, построив тем самым дерево рекурсии. Ветвь считается законченной, если ее листовой узел выполняет условия окончания игры, данное условие является «крайним случаем» рекурсии. Таким образом, условие окончания игры выполняется, если метка одного из игроков последовательно занимает три клетки по горизонтали, вертикали или диагонали либо игровое поле не содержит свободных клеток. В данном случае вершиной дерева является состояние игровой доски, которую требуется заполнить в контексте алгоритма единственным ходом игрока X, называемым максимизирующим. Каждый из возможных вариантов порождает новые гипотетические ситуации, результат которых зависит от хода игрока O, называемого минимизирующим. Для принятия оптимального решения

алгоритму необходимо оценивать возможные ходы. Описав условия окончания игры и определив ключевые понятия, оптимальной оценкой победы максимизирующего игрока следует задать значение, равное единице, минимизирующего — обратной величиной, а случай ничьей - нулем. Оценивание и возврат результатов происходит снизу вверх, т. е. от «листьев» к «корню», причем максимизирующий игрок ожидает получить максимально возможную оценку, а минимизирующий – наоборот.

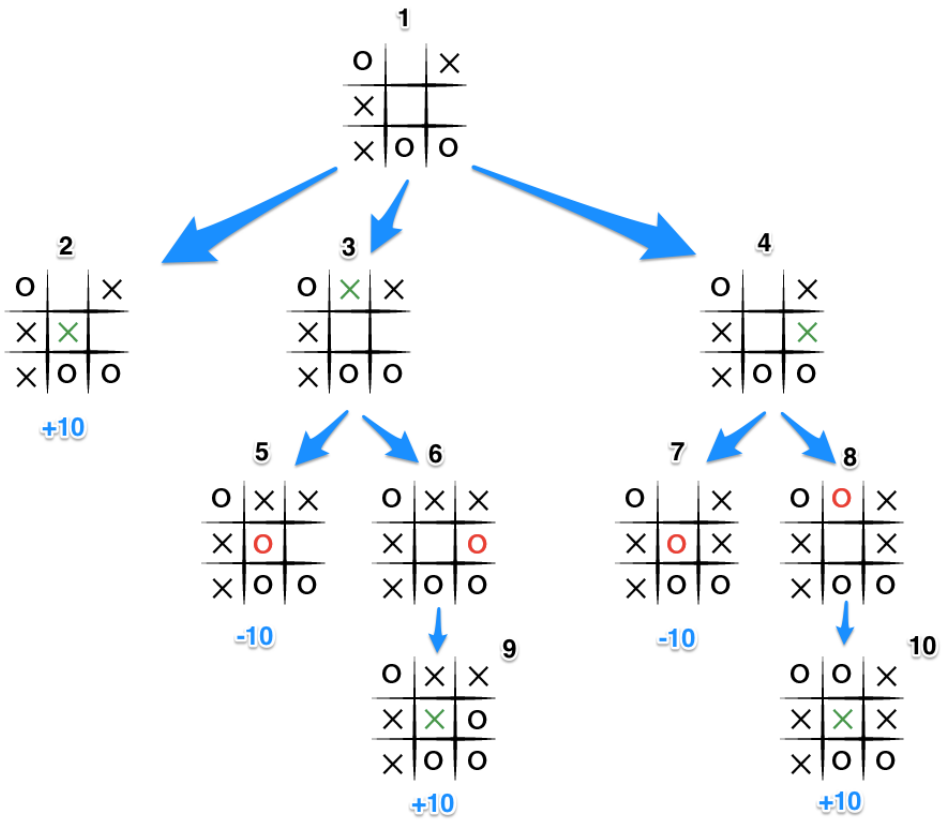


Рисунок 1. – Принцип работы алгоритма Минимакс.

Альфа-бета-отсечение. По своей сути он является методом «ветвей и границ». Основная его идея заключается в том, чтобы после «прохода» по одной из ветвей дерева решений «отсекать» ветви, заведомо не имеющие оптимального решения, относительно коэффициентов альфа и бета, полученных на первом проходе. Альфа — это минимально возможная оценка,

которую может получить максимизирующий игрок, инициализируется значением минус бесконечности и наоборот. Если в одном из узлов ветви значения альфа больше либо равно бета, найден ход, который гарантирует победу максимизирующему игроку. Если значения этих параметров в текущем узле не улучшаются, ветвь не следует рассматривать, поскольку все ее «потомки», как и она сама, не содержат оптимального решения. Таким образом, описанный метод позволяет находить оптимальное решение, заметно сократив при этом число рекурсивных вызовов.

В данной программе есть выбор уровня сложности от 1 до 10, цифры соответствуют максимальной глубине просмотра вариантов.

Ход работы

1. Написана функция Check_Win, проверяющая окончание игры и возвращающая 0 в случае завершения игры, 1 в случае выигрыша игрока, 2 в случае ничьи и 3 в случае выигрыша компьютера.
2. Реализация функции Board_Initializer, которая инициализирует игровое поле размером 3x3.
3. Реализация функции Print_Board, которая выводит игровое поле размером 3x3 в консоли или терминале.
4. Реализация функции Welcome, которая является приветствует игрока при запуске программы и включает в себя выбор действий: 's', 'q' и 'r'.
   ‘s’ – запуск самой игры
   ‘q’ – выход из программы
   ‘r’ – правила игры
5. Реализация функции get_user_move, с помощью которой игрок может выбрать позицию на поле.

6. Реализация функции evaluate_board, которая проверяет строки, столбцы и диагонали и оценивает результаты на игровом поле.
7. Реализация функции find_best_move с использованием алгоритма альфа-бета, которая ищет лучший ход для компьютера.
8. Реализация функции difficulty, позволяющая игроку выбрать уровень сложности игры.
9. Реализация функции main, в начале которой в которой происходит вызов функций Board_Initializer, Welcome, difficulty и Print_Board. Затем задаётся игровой цикл через while, в котором вызываются функции get_user_move, Check_Win, find_best_move и Print_Board. После с помощью условных операторов выводится результат игры.

Демонстрация работы

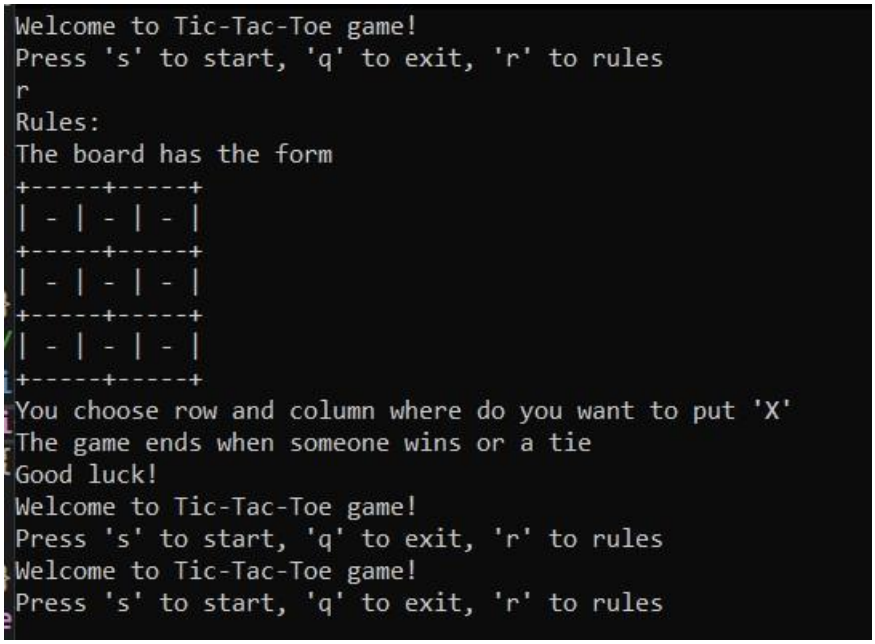
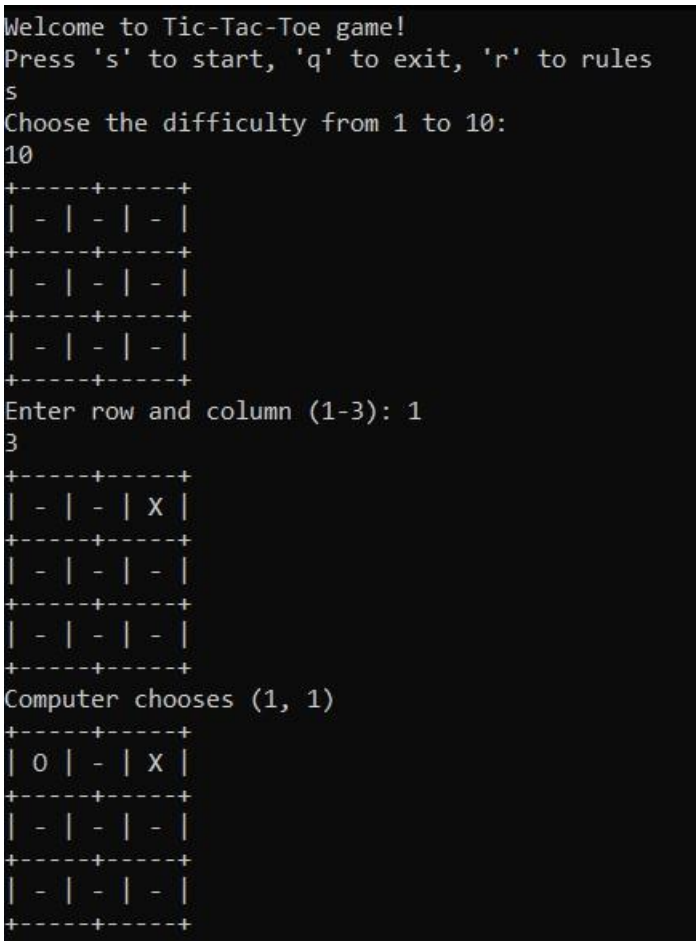
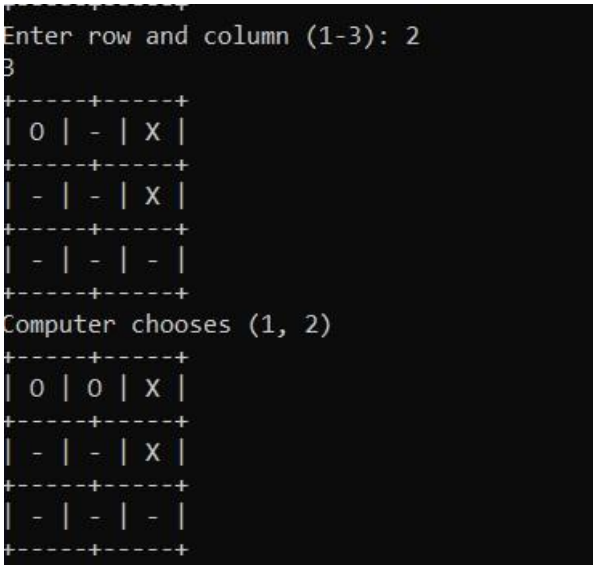
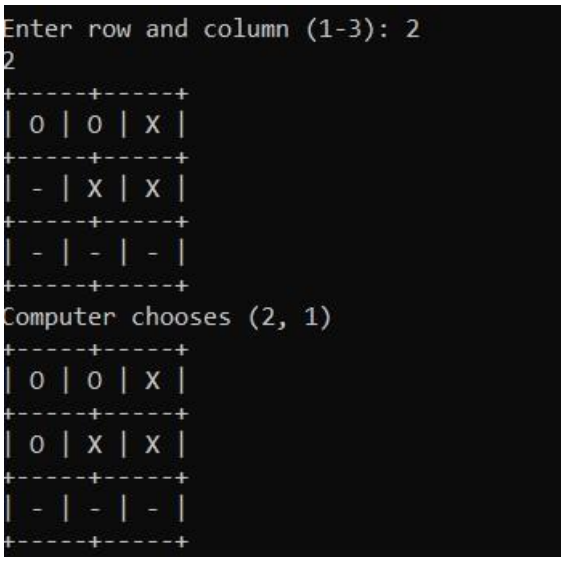


Рисунок 2. – Вывод правил игры


Welcome to Tic-Tac-Toe game!
Press 's' to start, 'q' to exit, 'r' to rules
s
Choose the difficulty from 1 to 10:
10
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
Enter row and column (1-3): 1
3
+-----+-----+
| - | - | X |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
Computer chooses (1, 1)
+-----+-----+
| O | - | X |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+


Enter row and column (1-3): 2
3
+-----+-----+
| O | - | X |
+-----+-----+
| - | - | X |
+-----+-----+
| - | - | - |
+-----+-----+
Computer chooses (1, 2)
+-----+-----+
| O | O | X |
+-----+-----+
| - | - | X |
+-----+-----+
| - | - | - |
+-----+-----+



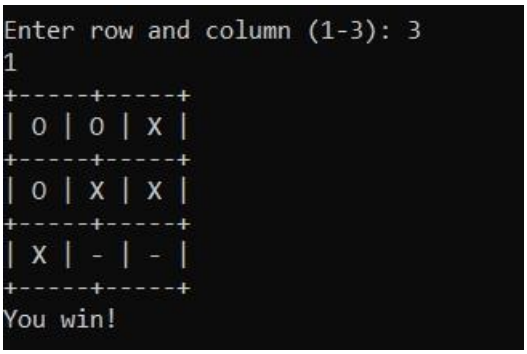


Рисунок 3. – Победа игрока

```
Welcome to Tic-Tac-Toe game!
Press 's' to start, 'q' to exit, 'r' to rules
s
Choose the difficulty from 1 to 10:
2
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
Enter row and column (1-3): 1
3
+-----+-----+
| - | - | X |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
Computer chooses (1, 1)
+-----+-----+
| O | - | X |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
Enter row and column (1-3): 1
2
+-----+-----+
| O | X | X |
+-----+-----+
| - | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
Computer chooses (2, 1)
+-----+-----+
| O | X | X |
+-----+-----+
| O | - | - |
+-----+-----+
| - | - | - |
+-----+-----+
```

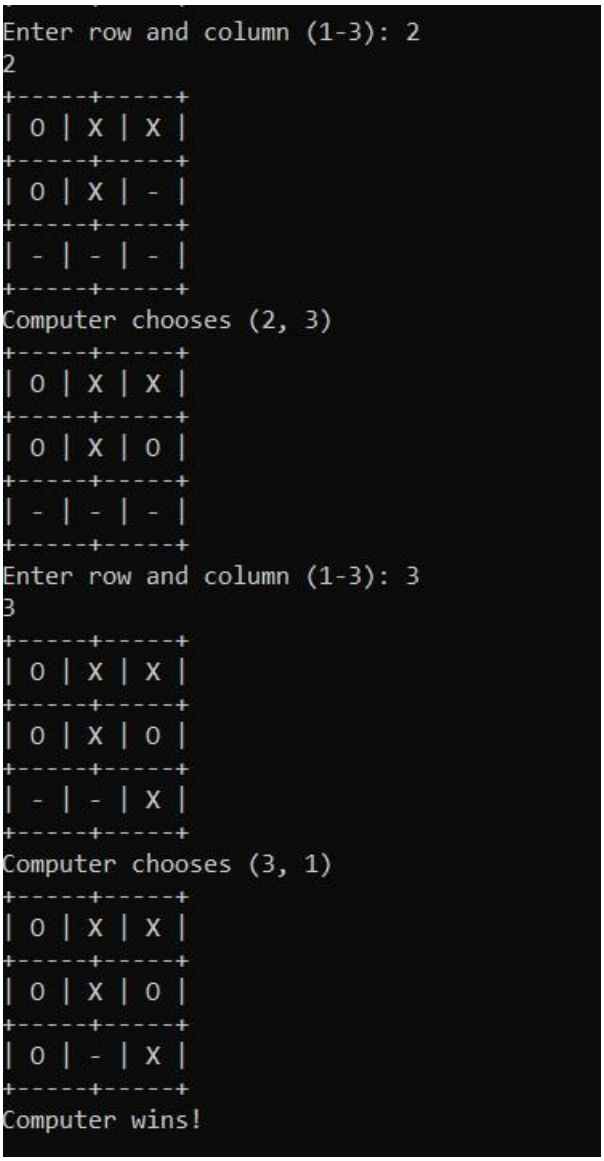
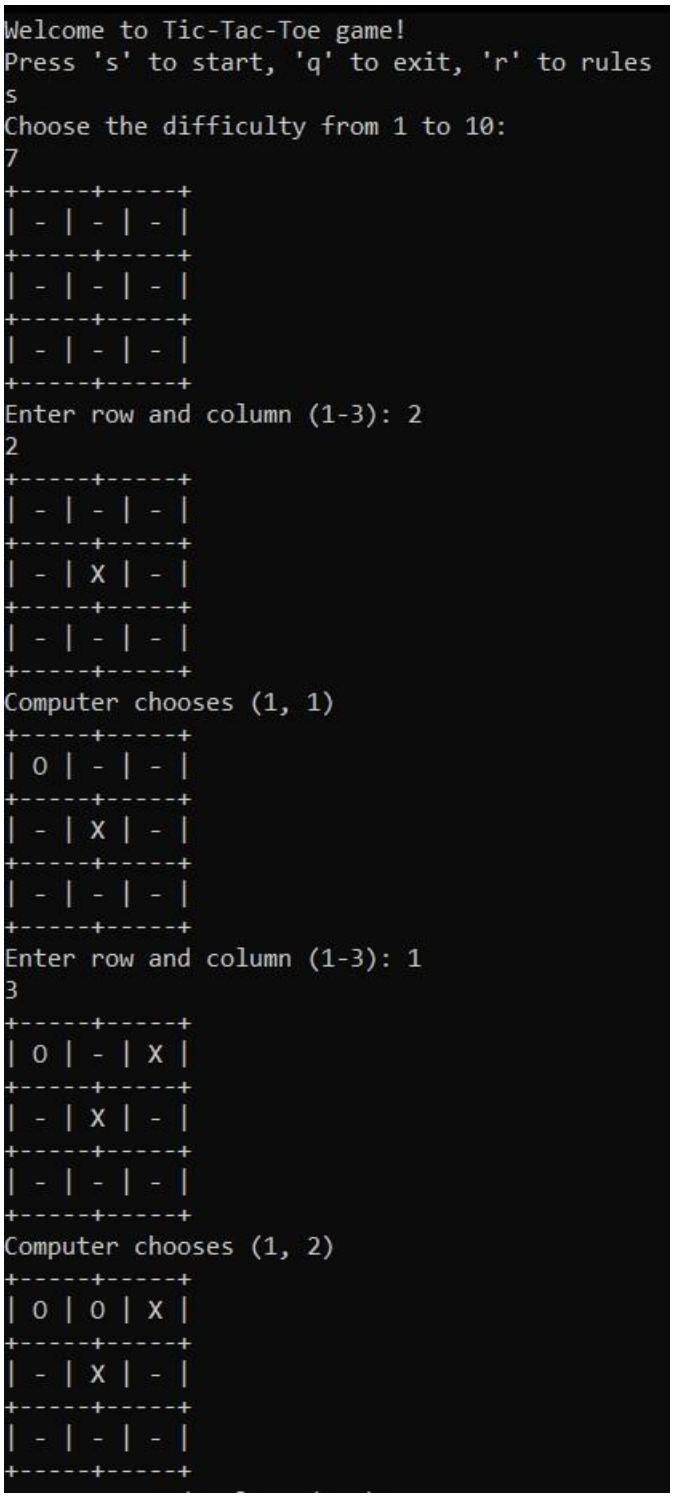


Рисунок 4. – Победа компьютера


Welcome to Tic-Tac-Toe game!
Press 's' to start, 'q' to exit, 'r' to rules
s
Choose the difficulty from 1 to 10:
7
Enter row and column (1-3): 2
2
| - | - | - |
-	X	-
Computer chooses (1, 1)
O	-	-
-	X	-
Enter row and column (1-3): 1
3
O	-	X
-	X	-
Computer chooses (1, 2)
O	O	X
-	X	-

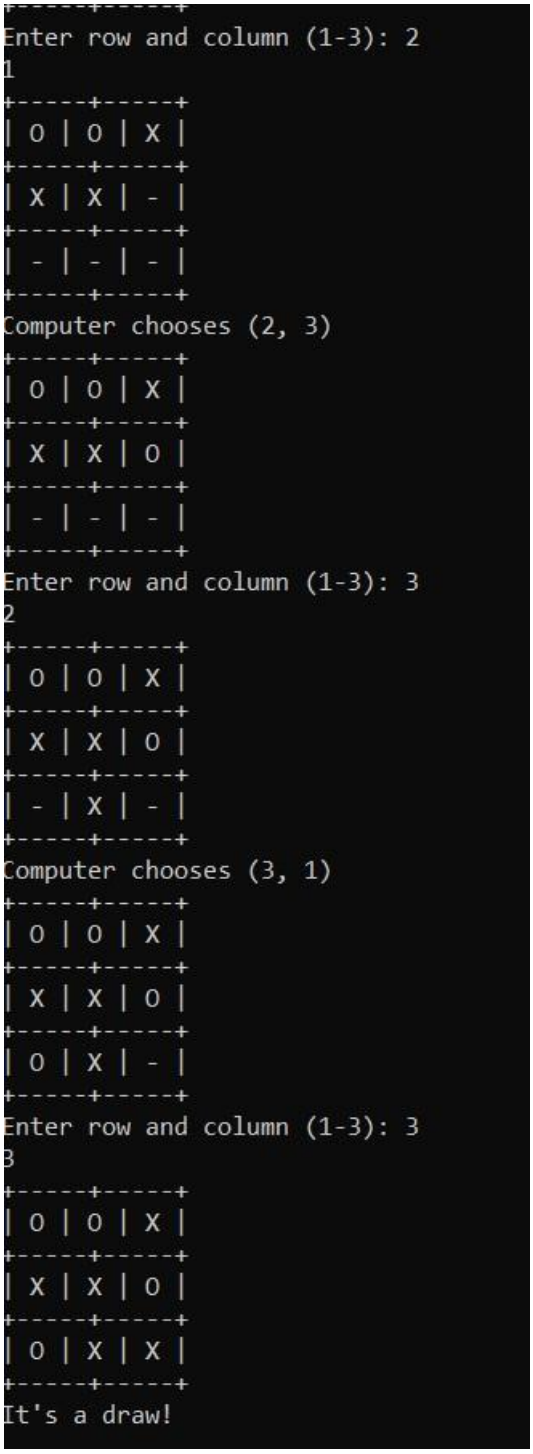


Рисунок 5. – Ничья

Вывод

В ходе проделанной работы была разработана программа для игры в крестики-нолики с использованием алгоритма альфа-бета. Этот алгоритм позволяет находить оптимальные ходы для компьютера, чтобы он не проигрывал в игре. Реализация программы на практике показала, что компьютер способен противостоять игроку и достигать результата в ничью или выигрыша. Таким образом, программа успешно выполняет поставленную задачу и демонстрирует высокий уровень эффективности при игре в крестики-нолики.

Литература

1. Рябко Б. Я., Фионов А. Н. Криптографические методы защиты информации: учебное пособие для вузов. М.: Горячая линия–Телеком, 2005. 229 с.

2. Blake I., Seroussi G., Smart N. Elliptic Curves in Cryptography. Cambridge University Press, 2002. 204 p.

3. История России: учебник / А. С. Орлов, В. А. Георгиев, Н. Г. Георгиева, Т. А. Сивохина. 2-е изд., перераб. и доп. М.: Проспект, 2004. 514 с.

4. Экономика: учебник / под ред. А. С. Булатова. 3-е изд., перераб. и доп. М.: Экономистъ, 2003. 894 с.

5. Рябко Б. Я., Фионов А. Н. Эффективный метод адаптивного арифметического кодирования для источников с большими алфавитами // Проблемы передачи информации. 1999. Т. 35, № 4. С. 1–14.

6. Лэтчфорд Е. У. С Белой армией в Сибири [Электронный ресурс] // Восточный фронт армии адмирала А. В. Колчака: [сайт]. [2004]. URL: `http://east-front.narod.ru/memo/latchford.htm` (дата обращения: 23.08.2007).

Приложение

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <limits.h>
    #include <math.h>

    #define min(i, j) (((i) < (j)) ? (i) : (j))
    #define max(i, j) (((i) > (j)) ? (i) : (j))

    #define BOARD_SIZE 3

    //функция проверки окончания игры
    //возвращает 0, если игра не завершена, 1 если
игрок выиграл, 2 если ничья, и 3 если компьютер
выиграет
    int Check_Win(char board[BOARD_SIZE][BOARD_SIZE])
    {
        int i, j;
        for (i = 0; i < BOARD_SIZE; i++) //проверка
строк
        {
            if (board[i][0] == board[i][1] &&
board[i][1] == board[i][2])
            {
                if (board[i][0] == 'X') return 1;
                else if (board[i][0] == 'O') return 3;
            }
        }
        for (j = 0; j < BOARD_SIZE; j++) //проверка
столбцов
        {
            if (board[0][j] == board[1][j] &&
board[1][j] == board[2][j])
            {
                if (board[0][j] == 'X') return 1;
                else if (board[0][j] == 'O') return 3;
            }
        }
        if (board[0][0] == board[1][1] && board[1][1]
== board[2][2])
```

```
        {
            if (board[0][0] == 'X') return 1;
            else if (board[0][0] == 'O') return 3;
        }
        if (board[0][2] == board[1][1] && board[1][1]
== board[2][0])
        {
            if (board[0][2] == 'X') return 1;
            else if (board[0][2] == 'O') return 3;
        }
        for (i = 0; i < BOARD_SIZE; i++) //проверка на
ничью
        {
            for (j = 0; j < BOARD_SIZE; j++)
            {
                if (board[i][j] != 'X' && board[i][j]
!= 'O') return 0;
            }
        }
        return 2;
    }

    void Board_Initializer(char board[][BOARD_SIZE])
    {
        char index = '-';
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                board[i][j] = index;
            }
        }
    }

    void Print_Board(char board[][BOARD_SIZE])
    {
        printf("+-----+-----+\n");
        printf("|");
        for (int i = 0; i < BOARD_SIZE; i++)
        {
```

```
            for (int j = 0; j < BOARD_SIZE; j++)
            {
                printf(" %c |", board[i][j]);
            }
            puts("");
            printf("+-----+-----+\n");
            if (i < BOARD_SIZE - 1)
            {
                printf("|");
            }
        }
    }

    //приветствие
    int Welcome(char board[][BOARD_SIZE])
    {
        char choise = 0;
        while (choise != 's')
        {
            printf("Welcome to Tic-Tac-Toe
game!\nPress 's' to start, 'q' to exit, 'r' to
rules\n");
            scanf_s("%c", &choise);
            if (choise == 'q')
            {
                exit(0);
            }
            else if (choise == 'r')
            {
                puts("Rules:\nThe board has the
form");
                Print_Board(board);
                puts("You choose row and column where
do you want to put 'X'");
                puts("The game ends when someone wins
or a tie");
                puts("Good luck!");
            }
        }
    }
```

```
    //выбор позиции игрока
    void get_user_move(char board[][BOARD_SIZE], int*
row, int* col)
    {
        int valid_move = 0;
        while (!valid_move)
        {
            printf("Enter row and column (1-%d): ",
BOARD_SIZE);
            scanf_s("%d %d", row, col);
            (*row)--;
            (*col)--;
            if (*row < 0 || *row >= BOARD_SIZE || *col
< 0 || *col >= BOARD_SIZE)
            {
                printf("Invalid move. Please try
again.\n");

            }
            else if (board[*row][*col] == 'X' ||
board[*row][*col] == 'O')
            {
                printf("That square is already taken.
Please try again.\n");
            }
            else valid_move = 1;
        }

    }

    ////функция оценки результатов на поле
    int evalute_board(char board[][BOARD_SIZE])
    {
        int i, j, score = 0;
        for (i = 0; i < BOARD_SIZE; i++) //проверка
строк
        {
            if (board[i][0] == board[i][1] &&
board[i][1] == board[i][2])
```

```
            {
                if (board[i][0] == 'O') score += 10;
                else if (board[i][0] == 'X') score -=
10;
            }
        }
        for (j = 0; j < BOARD_SIZE; j++) //проверка
столбцов
        {
            if (board[0][j] == board[1][j] &&
board[1][j] == board[2][j])
            {
                if (board[0][j] == 'O') score += 10;
                else if (board[0][j] == 'X') score -=
10;
            }
        }
        if (board[0][0] == board[1][1] && board[1][1]
== board[2][2]) //проверка диагоналей
        {
            if (board[0][0] == 'O') score += 10;
            else if (board[0][0] == 'X') score -= 10;
        }
        if (board[0][2] == board[1][1] && board[1][1]
== board[2][0])
        {
            if (board[0][2] == 'O') score += 10;
            else if (board[0][2] == 'X') score -= 10;
        }
        return score;
    }

    //Функция поиска лучшего хода для компьютера с
использованием алгоритма альфа-бета.
    int find_best_move(char board[][BOARD_SIZE], int
depth, int alpha, int beta, int is_max, int MAX_DEPTH)
    {
        int i, j, score, best_score, best_row = -1,
best_col = -1;
        int game_result = Check_Win(board);
```

```
        if (depth == 0 || game_result != 0)
        {
            return evalute_board(board);
        }
        if (is_max)
        {
            best_score = INT_MIN;
            for (i = 0; i < BOARD_SIZE; i++)
            {
                for (j = 0; j < BOARD_SIZE; j++)
                {
                    if (board[i][j] != 'X' &&
board[i][j] != 'O')
                    {
                        board[i][j] = 'O';
                        score = find_best_move(board,
depth - 1, alpha, beta, ~is_max, MAX_DEPTH); //~is_max
= !is_max;
                        board[i][j] = '-';
                        if (score > best_score)
                        {
                            best_score = score;
                            best_row = i;
                            best_col = j;
                        }
                        alpha = max(alpha,
best_score);
                        if (alpha >= beta) break;
                    }
                }
                if (alpha >= beta) break;
            }
            if (depth == MAX_DEPTH)
            {
                printf("Computer chooses (%d, %d)\n",
best_row + 1, best_col + 1);
                board[best_row][best_col] = 'O';
            }
            return best_score;
        }
```

```
        else
        {
            best_score = INT_MAX;
            for (i = 0; i < BOARD_SIZE; i++)
            {
                for (j = 0; j < BOARD_SIZE; j++)
                {
                    if (board[i][j] != 'X' &&
board[i][j] != 'O')
                    {
                        board[i][j] = 'X';
                        score = find_best_move(board,
depth - 1, alpha, beta, ~is_max, MAX_DEPTH); //~is_max
= !is_max;
                        board[i][j] = '-';
                        if (score < best_score)
                        {
                            best_score = score;
                        }
                        beta = min(beta, best_score);
                        if (alpha >= beta) break;
                    }
                }
                if (alpha >= beta) break;
            }
            return best_score;
        }
    }

    //выбор уровня сложности игры
    void difficulty(int* MAX_DEPTH)
    {
        printf("Choose the difficulty from 1 to
10:\n");
        scanf_s("%d", MAX_DEPTH);
        if (*MAX_DEPTH < 1 || *MAX_DEPTH > 10)
        {
            printf("Difficulty must be form 1 to 10,
try again\n");
            difficulty(MAX_DEPTH);
```

```
        }
        else *MAX_DEPTH = *MAX_DEPTH;
    }

    int main()
    {
        int row, col, MAX_DEPTH;
        char board[BOARD_SIZE][BOARD_SIZE];
        Board_Initializer(board);
        Welcome(board);
        difficulty(&MAX_DEPTH);

        Print_Board(board);

        while (1) //игровой цикл
        {
            get_user_move(board, &row, &col);
//очередь пользователя
            board[row][col] = 'X';
            Print_Board(board);
            if (Check_Win(board) != 0) break;
            find_best_move(board, MAX_DEPTH, INT_MIN,
INT_MAX, 1, MAX_DEPTH); //очередь компьютера
            Print_Board(board);
            if (Check_Win(board) != 0) break;
        }
        //игра окончена
        int game_result = Check_Win(board);
        if (game_result == 1)
        {
            printf("You win!\n");

        }
        else if (game_result == 3)
        {
            printf("Computer wins!\n");
        }
        else printf("It's a draw!\n");
        return 0;
    }
```