МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | **Факультет прикладной математики и информатики**

Кафедра теоретической и прикладной информатики

Лабораторная работа № 1

по дисциплине «Интеллектуальные системы»

СТРАТЕГИИ РЕШЕНИЯ ЗАДАЧ И ПРОГРАММИРОВАНИЕ ИГР

Бригада 1 ХАЙДАЕВ К.Е.

Группа ПМИ-82 ЗЯБЛИЦЕВА У.П.

Вариант 1

Преподаватели ФАДДЕЕНКОВ А.В.

Новосибирск, 2022

1 Задание

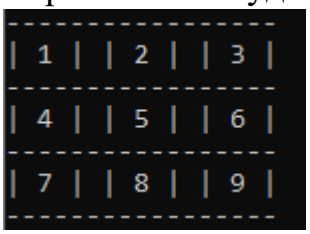
1. Сформулируйте задачу.
2. Разработайте адекватную структуру данных, максимально учитывающую специфику предметной области задачи. Обоснуйте выбор структуры. В случае представления задачи с помощью пространства состояний нарисуйте несколько первых уровней графа переходов. В случае сведения задач к подзадачам – несколько уровней И/ИЛИ дерева.
3. Реализуйте формальное описание проблемы, снабдив программу достаточным количеством средств ввода-вывода для наглядного отображения результатов.

Вариант 1. Реализовать игру в крестики-нолики для поля 3Х3.

2 Ход работы

Реализуем консольную игру крестики-нолики. Предоставим игроку выбор за кого играть за X или 0. Реализуем ИИ с помощью алгоритма минимакс с альфа-бета отсечениями.

Игровое поле будет массивом из 9 элементов:



Для реализации нам потребуется 3 класса: Node, Tree, Game и класс Program, содержащий точку входа.

Node – 1 элемент дерева решений, содержащий в себе игровое поле, массив других элементов дерева, вес данного элемента и сделанный ход.
Tree – дерево содержащее Node root, т.е. корень дерева решений, алгоритм минимакс, оценочную функцию и проверку на терминальное состояние.
Game – будет начальное игровое поле, а затем реализовывать ходы игрока и ИИ.

Самая интересная функция - `public int minimax(Node currentNode,int depth,int alpha,int beta, int AIorHuman,bool A_Fpruning=true)` – функция минимакс
Параметры:
currentNode – текущий элемент дерева;
depth – глубина;
alpha – максимальное значение для альфа;
beta – максимальное значение beta;
AIorHuman – ход человека или ИИ;

A_Fpruning – если true, то производятся alpha-beta отсечения, если false то нет (по умолчанию true).

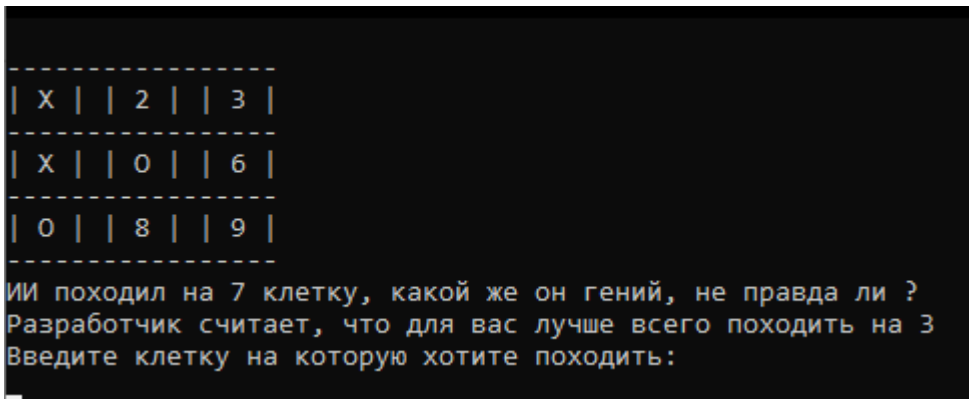
В начале вызова функция проверяет терминальное ли состояние поля на данный момент, если да, то возвращает вес данного состояния. Если нет, то создает новых потомков для данного состояния. Затем происходит рекурсивный вызов этой функции каждым из потомков, а результат выполнения функции записывается в переменную weight.

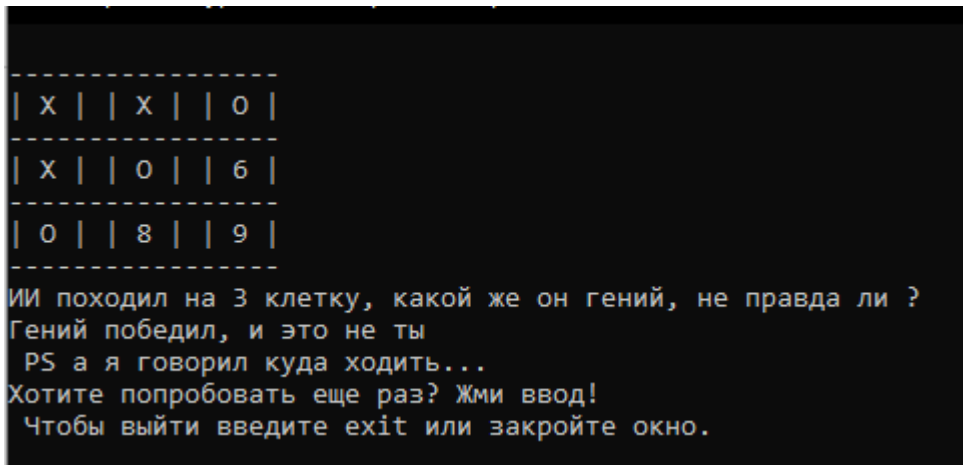
Функция оценки - int HeuristicEval(Node node, int AIorHuman)
Параметры
node – лист дерева решений (победа, поражение или ничья);
AIorHuman – для кого было построено дерево решений.
При победе, того для кого мы рассматривали данную ситуацию возвращает 100/(глубину+1). При поражении -100(глубину+1). При ничьей возвращаем 0. Таким образом, чем меньше ходов приводят нас к победе/поражению тем больший/меньший вес имеет данный ход.

3 Тесты

```
Введите 0 - если хотите играть за 0, введите X - если хотите играть за X
X

----------------
| 1 | | 2 | | 3 |
----------------
| 4 | | 5 | | 6 |
----------------
| 7 | | 8 | | 9 |
----------------

Разработчик считает, что для вас лучше всего походить на 1
Введите клетку на которую хотите походить:
```

```

-----------------
| X | | 2 | | 3 |
-----------------
| 4 | | O | | 6 |
-----------------
| 7 | | 8 | | 9 |
-----------------
ИИ походил на 5 клетку, какой же он гений, не правда ли ?
Разработчик считает, что для вас лучше всего походить на 2
Введите клетку на которую хотите походить:
```





4 Код программы

Program.cs

```
// See https://aka.ms/new-console-template for more information
using lab1_alpha_beta_algorithm_X_0_v3;


    string read;
do
{
    int AI = 10;
    int human = -10;
    while (true)
    {
        Console.WriteLine("Введите 0 - если хотите играть за 0, введите X - если хо-
тите играть за X");
        read = Console.ReadLine();
        if (read == "X" || read == "x")
        {
            AI = -10;
            human = 10;
            break;
        }
        else if (read == "0" || read == "o" || read == "O")
        {
            AI = 10;
            human = -10;
            break;
        }
    }

    Game game = new Game(AI, human);
```

```
    //Console.WriteLine(game);
    while (true)
    {
        if (AI == 10)
        {
            game.MoveAI();
            if (game.IsEnd())
            {
                Console.Clear();
                Console.WriteLine(game);
               // Console.ReadLine();
                break;
            }
            Console.WriteLine(game);
            Tree tree = new Tree(game.gameBoard, human, AI);
            Console.WriteLine("Разработчик считает, что для вас лучше всего походить
на " + (tree.BestMove() + 1));
            while (true)
            {
                Console.WriteLine("Введите клетку на которую хотите походить: ");
                read = Console.ReadLine();
                int tmp = 0;
                if (int.TryParse(read, out tmp))
                {
                    if (game.MoveHuman(int.Parse(read)))
                        break;
                    else
                        Console.WriteLine("Данная клетка занята или не существует");
                }
                else
                    Console.WriteLine("Некоректный ввод");
            }
            if (game.IsEnd())
            {
                Console.Clear();
                Console.WriteLine(game);
               // Console.ReadLine();
                break;
            }
            Console.Clear();
            // Console.WriteLine(game);
        }
        else
        {
            Console.WriteLine(game);
            Tree tree = new Tree(game.gameBoard, human, AI);
            Console.WriteLine("Разработчик считает, что для вас лучше всего походить
на " + (tree.BestMove() + 1));
            while (true)
            {
                Console.WriteLine("Введите клетку на которую хотите походить: ");
                read = Console.ReadLine();
                int tmp = 0;
                if (int.TryParse(read, out tmp))
                {
                    if (game.MoveHuman(int.Parse(read)))
                        break;
                    else
                        Console.WriteLine("Данная клетка занята или не существует");
                }
                else
                    Console.WriteLine("Некоректный ввод");
            }
            if (game.IsEnd())
            {
```

```
                Console.Clear();
                Console.WriteLine(game);
               // Console.ReadLine();
                break;
            }

            //Console.WriteLine(game);

            game.MoveAI();
            if (game.IsEnd())
            {
                Console.Clear();
                Console.WriteLine(game);
                //Console.ReadLine();
                break;
            }
            Console.Clear();
            //Console.WriteLine(game);
        }

    }
    Console.WriteLine("Хотите попробовать еще раз? Жми ввод!\n Чтобы выйти введите
exit или закройте окно.");
    read = Console.ReadLine();
    Console.Clear();
} while (read!="exit");
```

Tree.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab1_alpha_beta_algorithm_X_0_v3
{

    internal class Node
    {
        /// <summary>
        /// {1,2,3,4,5,6,7,8,9}
        /// </summary>
        public List<int> gameBoard;
        /// <summary>
        /// дети
        /// </summary>
        public List<Node> nodes;
        /// <summary>
        /// вес
        /// </summary>
        public int weight;
        /// <summary>
        /// глубина
        /// </summary>
        public int depth;
        /// <summary>
        /// 10 or -10
        /// </summary>
        int player;
        /// <summary>
        /// куда походил
        /// </summary>
        public int currentTurn;

        /// <summary>
        /// создаем копию gameBoard
        /// </summary>
        /// <param name="gameBoard"></param>
        public Node(List<int> gameBoard)
        {
            this.gameBoard = new List<int>();
            foreach (var item in gameBoard)
            {
                this.gameBoard.Add(item);
            }
            nodes = new List<Node>();
        }
        public Node(List<int> gameBoard,int currentTurn,int depth=0)
        {
```

```
        this.gameBoard = new List<int>();
        foreach (var item in gameBoard)
        {
            this.gameBoard.Add(item);
        }
        this.currentTurn = currentTurn;
        this.depth = depth;
        nodes = new List<Node>();
    }
    public override string ToString()
    {
        string res = "";

        int i = 0;
        foreach (var item in gameBoard)
        {
            i++;
            if (i == 4 || i == 1 || i == 7)
                res += "\n" + "-----------------\n";
            res += "| ";
            if (item == 10)
                res += "X";
            else if (item == -10)
                res += "O";
            else
                res += item.ToString();
            res += " | ";
        }
        res += "\n" + "-----------------\n";
        res += "Ход=" + currentTurn + " Глубина=" + depth + " Вес=" + weight;

        return res;
    }
}
internal class Tree
{
    public Node root;
    /// <summary>
    /// X=10 0=-10
    /// </summary>
    int AI;
    int human;

    public int countNodes = 0;
    public Tree(List<int> gameBoard,int AI, int human)
    {
        root = new Node(gameBoard);
        countNodes++;
        this.AI = AI;
        this.human = human;
        minimax(root, 1, int.MinValue, int.MaxValue, AI);
    }
```

```
    public int minimax(Node currentNode,int depth,int alpha,int beta, int AIorHuman,bool A_Fpruning=true)
    {
      //если это лист, то возвращаем оценку конечного состояния
      if ( IsTerminated(currentNode, AIorHuman))
      {
        //то есть вес нашего листа
        currentNode.weight = HeuristicEval(currentNode, AIorHuman);
        return currentNode.weight;
      }

      int eval;
      List<int> board = new List<int>(currentNode.gameBoard);
      //Делаем все ходы по очереди, создаем новые листья для дерева
      Node temp;
      foreach (var turn in GetEmptyIndices(currentNode))
      {
        board[turn] = AIorHuman;
        temp = new Node(board, turn + 1,depth);
        countNodes++;
        currentNode.nodes.Add(temp);
        if (IsTerminated(temp, AIorHuman))
          break;
        board[turn] = turn + 1;
      }
      //ход ИИ
      if (AIorHuman == AI)
      {
        int max = int.MinValue;
        foreach (var child in currentNode.nodes)
        {
          //это есть вес данного поля
          eval = minimax(child, depth+1, alpha, beta, -AIorHuman,A_Fpruning);
          child.depth = depth;
          child.weight = eval;

          max = Max(max, eval);
          if (A_Fpruning)
          {
            alpha = Max(alpha, eval);
            //отсекаем
            if (beta <= alpha)
              break;
          }


        }
        //вернем лучший вес
        return max;
      }
      //Ход человека
      else
      {
```

```
        int min=int.MaxValue;
        foreach (var child in currentNode.nodes)
        {
            eval = minimax(child, depth +1, alpha, beta, -AIorHuman,A_Fpruning);

            child.weight = eval;

            min = Min(min, eval);
            if (A_Fpruning)
            {
                beta = Min(beta, eval);
                //отсекаем
                if (beta <= alpha)
                    break;
            }
        }
        return min;
    }
}
int Max(int n1,int n2)
{
    if (n1 > n2)
        return n1;
    else
        return n2;
}
int Min(int n1, int n2)
{
    if (n1 < n2)
        return n1;
    else
        return n2;
}
int HeuristicEval(Node node, int AIorHuman)
{
    if (Win(node, AI)||Win(node,-AI))
    {
        if (AIorHuman == AI)
            return -100 / (node.depth+1);
        else
            return 100 / (node.depth+1);
    }
    else if (GetEmptyIndices(node).Count == 0)
        return 0;
    //может изменим оценку тут как-то
    else
        return 0;
}
public static bool IsTerminated(Node node,int AIorHuman )
{
    if (Win(node, AIorHuman) || Win(node, -AIorHuman) || GetEmptyIndices(node).Count == 0)
        return true;
    else
```

```
        return false;
    }
    public static bool Win(List<int> gameBoard, int AIorHuman)
    {
       int p = AIorHuman;
       if ((gameBoard[0] == p && gameBoard[1] == p && gameBoard[2] == p) ||
         (gameBoard[3] == p && gameBoard[4] == p && gameBoard[5] == p) ||
         (gameBoard[6] == p && gameBoard[7] == p && gameBoard[8] == p) ||

         (gameBoard[0] == p && gameBoard[3] == p && gameBoard[6] == p) ||
         (gameBoard[1] == p && gameBoard[4] == p && gameBoard[7] == p) ||
         (gameBoard[2] == p && gameBoard[5] == p && gameBoard[8] == p) ||

         (gameBoard[0] == p && gameBoard[4] == p && gameBoard[8] == p) ||
         (gameBoard[2] == p && gameBoard[4] == p && gameBoard[6] == p))
          return true;
       else
          return false;

    }
    public static bool Win(Node node, int AIorHuman)
    {
       List<int> gameBoard = node.gameBoard;
       int p = AIorHuman;
       if ((gameBoard[0] == p && gameBoard[1] == p && gameBoard[2] == p) ||
         (gameBoard[3] == p && gameBoard[4] == p && gameBoard[5] == p) ||
         (gameBoard[6] == p && gameBoard[7] == p && gameBoard[8] == p) ||

         (gameBoard[0] == p && gameBoard[3] == p && gameBoard[6] == p) ||
         (gameBoard[1] == p && gameBoard[4] == p && gameBoard[7] == p) ||
         (gameBoard[2] == p && gameBoard[5] == p && gameBoard[8] == p) ||

         (gameBoard[0] == p && gameBoard[4] == p && gameBoard[8] == p) ||
         (gameBoard[2] == p && gameBoard[4] == p && gameBoard[6] == p))
          return true;
       else
          return false;

    }

    public static List<int> GetEmptyIndices(Node node)
    {
       List<int> res = new List<int>();
       foreach (var item in node.gameBoard)
       {
          if (item != -10 && item != 10)
             res.Add(item - 1);
       }
       return res;
    }
    public static List<int> GetEmptyIndices(List<int> gameBoard)
    {
       List<int> res = new List<int>();
```

```
            foreach (var item in gameBoard)
            {
                if (item != -10 && item != 10)
                    res.Add(item - 1);
            }
            return res;
        }


        public int BestMove()
        {
            int max = root.nodes.Max(t => t.weight);
            foreach (var item in root.nodes)
            {
                if (item.weight == max)
                    return item.currentTurn-1;
            }
            return 0;
        }
        public override string ToString()
        {
            string res = "";
            FullBypassTree(root, ref res);
            res += "\nВсего ходов=" + countNodes;
            return res;
        }
        void FullBypassTree(Node start, ref string res)
        {
            //if (start.depth == 9)
            {
                res += start.ToString();
            }

            foreach (var item in start.nodes)
            {
                FullBypassTree(item, ref res);
            }
        }
    }
}
```

Game.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab1_alpha_beta_algorithm_X_0_v3
{
    internal class Game
    {
        public List<int> gameBoard;

        int whoWin = -1;
        int AI;
        int Human;
        public int whoMove = 10;
        public int currentTurn = -1;
        /// <summary>
        /// X=10 0=-10
        /// </summary>
        /// <param name="AI">10 or -10</param>
        /// <param name="Human">10 or -10</param>
        public Game(int AI, int Human)
        {
            gameBoard = new List<int>() { 1,2,3,4,5,6,7,8,9};
            this.AI = AI;
            this.Human = Human;
        }

        public bool IsEnd()
        {
            if (Tree.Win(gameBoard, AI))
                whoWin = AI;
            else if (Tree.Win(gameBoard, Human))
                whoWin = Human;
            else if (Tree.GetEmptyIndices(gameBoard).Count == 0)
                whoWin = 0;
            else
                return false;
            return true;

        }

        public void MoveAI()
        {
            Tree tree = new Tree(gameBoard, AI, Human);
            currentTurn = tree.BestMove()+1;
            gameBoard[currentTurn-1] = AI;
            whoMove = Human;
        }

        public bool MoveHuman(int move)
        {
            bool res=false;
            foreach (var item in Tree.GetEmptyIndices(gameBoard))
            {
                if (move == item+1)
                {
                    res = true; break;
                }
            }
            if (!res)
                return false;
            gameBoard[move - 1] = Human;
```

```
            currentTurn = move;
            whoMove = AI;
            return res;
        }

        public override string ToString()
        {
            string res = "";

            int i = 0;
            foreach (var item in gameBoard)
            {
                i++;
                if (i == 4 || i == 1 || i == 7)
                    res += "\n" + "-----------------\n";
                res += "| ";
                if (item == 10)
                    res += "X";
                else if (item == -10)
                    res += "O";
                else
                    res += item.ToString();
                res += " | ";
            }
            res += "\n" + "-----------------\n";
            if(currentTurn!=-1)
            if (whoMove == AI)
                res += "Вы походили на " + currentTurn + " клетку";
            else
                res += "ИИ походил на " + currentTurn + " клетку, какой же он гений,
не правда ли ?";

            if (whoWin == 0)
                res += "\nЛегкая игра, ничья, но даже не думай о ней в следующий
раз!";
            else if (whoWin == AI)
                res += "\nГений победил, и это не ты\n PS а я говорил куда ходить...";
            else if (whoWin == Human)
                res += "\nВ этот раз тебе повезло, сдаюсь";

            return res;
        }
    }
}
```