***Звіт***

*Програмування інтелектуальних інформаційних систем*

Лабораторна робота №3

“Алгоритм Negamax, Negamax з альфа-бета відсіканням та Negascout”

Студентки групи ІП-01

Галько Міли Вячеславівни

**Лабораторна робота №3**

“Алгоритм Negamax, Negamax з альфа-бета відсіканням та Negascout”

**Завдання:**

1. Реалізувати алгоритм Negamax (можна вносячи зміни в алгоритм другої лабораторної, а можна реалізувавши нову гру, наприклад, хрестики-нолики або Гомоку).

2. Реалізувати алгоритм Negamax з альфа-бета відсіканням (можна вносячи зміни у алгоритм другої лабораторної).

3. Реалізувати алгоритм NegaScout.

**Виконання:**

Реалізуємо гру «Хрестики-нолики». Для цього створимо класи:

1. Field: утримає масив-поле, що складається з Cell (enum: Empty, X, O); має змогу перевірити на закінчення гри, отримати оцінку поля, спадкоємних станів
2. TicTacToeGame: реалізує гру (ходи гравців), для гравця ручного виконується обробка введення, для противника використовується клас INegaMax для виконання ходу.
3. INegaMax та спадкоємці-реалізації: NegaMaxStandard, NegaMaxAlphaBetaPruning, NegaScout.

**Код класів**

3) INegaMax та спадкоємці-реалізації:

public interface INegaMax  
**{** Field GetBestMove(Field field);  
**}**

public class NegaMaxStandard : INegaMax  
**{** private int Apply(Field board, int depth, int color)  
 {  
 if (depth == 9 || board.GameFinished(out \_))  
 {  
 return color \* board.GetScore(depth);  
 }  
  
 int **value** = int.**MinValue**;  
  
 foreach (var adjacent in board.GetAdjacents(color == -1))  
 {  
 **value** = Math.Max(**value**, -1 \* Apply(adjacent, depth + 1, color \* -1));  
 }  
  
 return **value**;  
 }  
  
  
 public Field GetBestMove(Field field)  
 {  
 var adjacents = field.GetAdjacents(true);  
 Field **bestField** = null;  
 var **bestScore** = Int32.**MinValue**;  
 foreach (var adj in adjacents)  
 {  
 var adjScore = -1\*Apply(adj, 1, 1);  
 if (**bestScore** < adjScore)  
 {  
 **bestScore** = adjScore;  
 **bestField** = adj;  
 }  
 }  
  
 return **bestField**;  
 }  
**}**

public class NegaMaxAlphaBetaPruning : INegaMax  
**{** private int Apply(Field board, int depth, int color, int alpha, int beta)  
 {  
 if (depth == 9 || board.GameFinished(out \_))  
 {  
 return color \* board.GetScore(depth);  
 }  
  
 int **value** = int.**MinValue**;  
 foreach (var adjacent in board.GetAdjacents(color == -1))  
 {  
 **value** = Math.Max(**value**, -1 \* Apply(adjacent, depth + 1, color \* -1, -1 \* beta, -1 \* **alpha**));  
   
 **alpha** = Math.Max(**alpha**, **value**);  
 if (**alpha** >= beta)  
 {  
 break;  
 }  
 }  
  
 return **value**;  
 }  
  
  
 public Field GetBestMove(Field field)  
 {  
 var adjacents = field.GetAdjacents(true);  
 Field **bestField** = null;  
 var **bestScore** = Int32.**MinValue**;  
 foreach (var adj in adjacents)  
 {  
 var adjScore = -1\*Apply(adj, 1, 1,int.**MinValue**, int.**MaxValue**);  
 if (**bestScore** < adjScore)  
 {  
 **bestScore** = adjScore;  
 **bestField** = adj;  
 }  
 }  
  
 return **bestField**;  
 }  
**}**

public class NegaScout : INegaMax  
**{** private int Apply(Field board, int depth, int color, int alpha, int beta)  
 {  
 if (depth == 9 || board.GameFinished(out \_))  
 {  
 return color \* board.GetScore(depth);  
 }  
   
 int **b** = beta;  
 var adjacents = board.GetAdjacents(color == -1);  
 for (var **i** = 0; **i** < adjacents.Count; **i**++)  
 {  
 int **value** = -1 \* Apply(adjacents[**i**], depth + 1, -1 \* color, -1 \* **b**, -1 \* **alpha**);  
 if (**value** > **alpha** && **value** < beta && **i** != 0)  
 {  
 **value** = -1 \* Apply(adjacents[**i**], depth + 1, -1 \* color, -1 \* beta, -1 \* **alpha**);  
 }  
  
  
 **alpha** = Math.Max(**alpha**, **value**);  
 if (**alpha** >= beta)  
 {  
 break;  
 }  
  
 **b** = **alpha** + 1;  
 }  
  
 return **alpha**;  
 }  
  
 public Field GetBestMove(Field field)  
 {  
 var adjacents = field.GetAdjacents(true);  
 Field **bestField** = null;  
 var **bestScore** = Int32.**MinValue**;  
 foreach (var adj in adjacents)  
 {  
 var adjScore = -1\*Apply(adj, 1, 1,int.**MinValue**, int.**MaxValue**);  
 if (**bestScore** < adjScore)  
 {  
 **bestScore** = adjScore;  
 **bestField** = adj;  
 }  
 }  
  
 return **bestField**;  
 }  
**}**

2) TicTacToeGame:

public class TicTacToeGame  
**{** private Field \_field;  
 private bool \_playerTurn;  
 private INegaMax \_negaMax;  
  
 public TicTacToeGame(bool playerTurn, INegaMax negaMax)  
 {  
 \_playerTurn = playerTurn;  
 \_negaMax = negaMax;  
 \_field = new Field(\_playerTurn ? Cell.**X** : Cell.**O**);  
 }  
  
 public void Play()  
 {  
 Console.WriteLine(\_field);  
 Cell **winner** = Cell.Empty;  
 while (!\_field.GameFinished(out **winner**))  
 {  
 if (\_playerTurn)  
 {  
 PlayerMove();  
 }  
 else  
 {  
 ComputerMove();  
 }  
   
 \_playerTurn = !\_playerTurn;  
 Console.Clear();  
 Console.WriteLine(\_field);  
 }  
  
 Console.WriteLine($"Winner is {**winner**}");  
 }  
  
 private void ComputerMove()  
 {  
 \_field = \_negaMax.GetBestMove(\_field);  
 }  
  
 private void PlayerMove()  
 {  
 int i;  
 int j;  
 do  
 {  
 Console.Write("I: ");  
 i = InputValidate();  
   
 Console.Write("\nJ: ");  
 j = InputValidate();  
 } while (\_field[i, j] != Cell.**Empty**);  
   
 \_field[i, j] = \_field.Player;  
 }  
  
 private static int InputValidate()  
 {  
 int i;  
 while (!(int.TryParse(Console.ReadKey().KeyChar.ToString(), out i) && i is > 0 and < 4))  
 {  
 }  
  
 return i -1;  
 }  
**}**

1. Field та enum Cell:

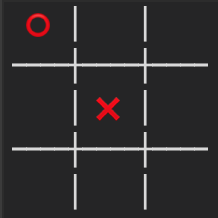
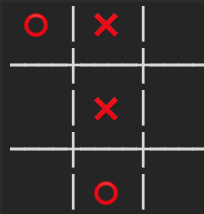
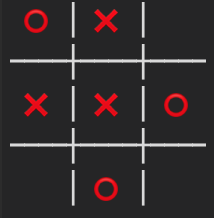
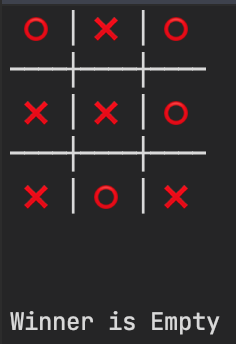
public enum Cell  
{  
 **Empty**,  
 **X**,  
 **O**};  
  
public class Field : ICloneable  
**{** public Cell this[int i, int j]  
 {  
 get => \_field[i, j];  
 set => \_field[i, j] = value;  
 }  
  
 public Field(Cell player)  
 {  
 \_player = player;  
 \_computer = player == Cell.**X** ? Cell.**O** : Cell.**X**;  
 \_field = new Cell[3, 3];  
 }  
  
 public Field(Cell[,] field, Cell player, Cell computer)  
 {  
 \_field = field;  
 \_player = player;  
 \_computer = computer;  
 }  
  
  
 public Cell Player => \_player;  
 private Cell \_player;  
 private Cell \_computer;  
 private Cell[,] \_field;  
  
 public bool GameFinished(out Cell winner)  
 {  
 for (int **i** = 0; **i** < 3; **i**++)  
 {  
 // Check rows and columns  
 if ((\_field[**i**, 0] == \_field[**i**, 1]  
 && \_field[**i**, 1] == \_field[**i**, 2]  
 && \_field[**i**, 0] != Cell.**Empty**))  
 {  
 winner = \_field[**i**, 0];  
 return true;  
 }  
   
 if ((\_field[0, **i**] == \_field[1, **i**]  
 && \_field[1, **i**] == \_field[2, **i**]  
 && \_field[0, **i**] != Cell.**Empty**))  
 {  
 winner = \_field[0, **i**];  
 return true;  
 }  
 }  
  
 //Check diagonals  
 if (\_field[1, 1] != Cell.**Empty** &&  
 (\_field[0, 0] == \_field[1, 1]  
 && \_field[1, 1] == \_field[2, 2]  
 ||  
 \_field[0, 2] == \_field[1, 1]  
 && \_field[1, 1] == \_field[2, 0]  
 ))  
 {  
 winner = \_field[1, 1];  
 return true;  
 }  
  
 //Check move possibility  
 winner = Cell.**Empty**;  
 foreach (var cell in \_field)  
 {  
 if (cell == Cell.**Empty**)  
 {  
 return false;  
 }  
 }  
  
 return true;  
 }  
  
 public int GetScore(int depth)  
 {  
 if (!GameFinished(out Cell winner) || winner == Cell.**Empty**)  
 {  
 return 0;  
 }  
  
 return winner == \_player ? 500 / depth : -500 / depth;  
 }  
  
 public List<Field> GetAdjacents(bool computerMove)  
 {  
 var newStates = new List<Field>();  
 for (int **i** = 0; **i** < 3; **i**++)  
 {  
 for (int **j** = 0; **j** < 3; **j**++)  
 {  
 if (\_field[**i**, **j**] == Cell.**Empty**)  
 {  
 var state = Clone() as Field;  
 state[**i**, **j**] = computerMove ? \_computer : \_player;  
 newStates.Add(state);  
 }  
 }  
 }  
  
 return newStates;  
 }  
  
 public override string ToString()  
 {  
 var sb = new StringBuilder();  
 for (int **i** = 0; **i** < 3; **i**++)  
 {  
 for (int **j** = 0; **j** < 3; **j**++)  
 {  
 if (\_field[**i**, **j**] == Cell.**Empty**)  
 {  
 sb.Append(" ");  
 }  
 else if (\_field[**i**, **j**] == Cell.**X**)  
 {  
 sb.Append(" ❌ ");  
 }  
 else  
 {  
 sb.Append(" ⭕ ");  
 }  
  
 if (**j** == 0 || **j** == 1)  
 {  
 sb.Append("│");  
 }  
 }  
 sb.AppendLine();  
 if (**i** == 0 || **i** == 1)  
 {  
 sb.AppendLine("────┼────┼────");  
 }  
 else  
 {  
 sb.AppendLine();  
 }  
 }  
  
 return sb.ToString();  
 }  
  
 public object Clone()  
 {  
 Field cloned = new Field(\_field.Clone() as Cell[,], \_player, \_computer);  
 return cloned;  
 }  
**}**

0) Program:

new TicTacToeGame(true,new NegaMaxStandard()).Play();

**Результат при «нічия»:**

Ми ходимо «х»:

**Результат при «програш»:**

