Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут прикладної математики та фундаментальних наук

Кафедра прикладної математики

Звіт

до лабораторної роботи №6

з дисципліни “Математичні основи штучного інтелекту”

Виконала:

студентка групи ПМ-33

Стецюк Анастасія

Прийняв:

Доцент кафедри ІМФН Пабирівський В.В

Львів 2024

Алгоритм гри на основі дерева рішень

Завдання: Розробити програмну реалізацію гри «хрестики-нулики» на основі дерева рішень.

Етапи виконання завдання:

1. Вивчити із використанням запропонованих літературних джерел зміст методу та застосування дерева прийняття рішень.
2. Розробити структуру даних для зберігання дерева рішень.
3. Розробити дерево рішень для гри «хрестики-нулики».
4. Реалізувати допоміжну функцію для вибору комп'ютером рішення з дерева рішень.
5. Реалізувати допоміжні функції для візуалізація поля гри «хрестики-нулики» та взаємодії з користувачем через маніпулятор «миша» (може бути використано довільний інструмент візуальної розробки, наприклад, MS Visual Studio та Win Forms).
6. Безпосередньо реалізувати гру «хрестики-нулики» для гри людини з комп'ютером (забезпечити безпрограшну гру комп'ютера).

Виконання

Реалізація програми на мові JavaScript в середовищі VS Code:

```
var theboard = new Board();
var ai = new AI('O');
var selected, opponent;

$(document).ready(function(){
    $('button').click(function(){
        $(this).addClass('seed');
        selected = $(this).text();
        opponent = selected === 'X' ? 'O' : 'X';
        ai = new AI('X');
        $('button').attr('disabled', 'true').removeClass('hover');

        if(opponent === 'X') {
            //Make the computer's move
            var move = ai.getBestMove(theboard);
            var moveStr = move.join('');
```

```
            $('#'+moveStr).text('X').addClass('selected').unbind( "click" );
            theboard.makeMove('X', move);
        }

        //Handles clicking on a spot, only active after selecting which piece to play
    $('.spot').click(handleClicks);
    });
});

function handleClicks() {
        if($('#result').text())
            $('#result').empty();

        //Make the players move.
        $(this).text(selected);
        $(this).addClass('selected').unbind( "click" );
        var spotid = $(this).attr('id');
        spotid = spotid.split('');
        theboard.makeMove(selected, [spotid[0], spotid[1]]);

        //Ensure that it is not time for gameOver
        var winner = theboard.checkForWin();
        if(winner || theboard.isFull())
            gameOver(winner);
        else {
            //Make the computer's move
            var move = ai.getBestMove(theboard);
            console.log(move);
            var moveStr = move.join('');
            $('#'+moveStr).text(opponent).addClass('selected').unbind( "click" );
            theboard.makeMove(opponent, move);

            //Check again for game over.
            winner = theboard.checkForWin();
```

```
            if(winner || theboard.isFull())
                gameOver(winner);
        }
}

function gameOver(winner) {
    //Notify the player of the status
    if(winner)
        $('#result').text(winner +' won the game!');
    else
        $('#result').text('The game was a draw!');

    //Wait a second to show the move, then reset
    setTimeout(function(){
        //Reset the game
        $('table').empty();
        $('table').append("<tr id='row0' class='row'><td id='00' class='spot'>
</td><td id='01' class='spot'> </td><td id='02' class='spot'> </td></tr><tr id='row1'
class='row'><td id='10' class='spot'> </td><td id='11' class='spot'> </td><td id='12'
class='spot'> </td></tr><tr id='row2' class='row'><td id='20' class='spot'> </td><td
id='21' class='spot'> </td><td id='22' class='spot'> </td></tr>");
        theboard = new Board();
        $('.spot').click(handleClicks);

        if(opponent === 'X') {
            //Make the computer's move
            var move = ai.getBestMove(theboard);
            var moveStr = move.join('');
            $('#'+moveStr).text(opponent).addClass('selected').unbind( "click" );
            theboard.makeMove(opponent, move);
        }
    }, 1000);
}
```

```
//Implementation of the game AI
function AI(seed) {
    this.marker = seed;
    this.opponent = seed == 'X' ? 'O' : 'X';
    this.max = 10;
    this.min = -10;

    this.minimax = function(board, player) {
        var bestScore = -10,
                currScore = 0,
                moves = board.getAvailableMoves();

        //Base case for finding leaf nodes
        if(board.turnCnt >= 9 || board.checkForWin() || !moves)
            return this.evaluate(board);

        //Maximize
        if(player === this.marker) {
            bestScore = this.min;
            for(var move in moves) {
                var newBoard = board.clone();
                newBoard.makeMove(this.marker, moves[move]);
                currScore = this.minimax(newBoard, this.opponent);
                if(currScore > bestScore) {
                    bestScore = currScore;
                }
            }
            return bestScore;
        }

        //Minimize
        if(player === this.opponent) {
            bestScore = this.max;
            for(var move in moves) {
```

```
            var newBoard = board.clone();
            newBoard.makeMove(this.opponent, moves[move]);
            currScore = this.minimax(newBoard, this.marker);
            if(currScore < bestScore) {
                bestScore = currScore;
            }
        }
        return bestScore;
    }
};

//Gets the best move for this board configuration
this.getBestMove = function(board) {
    var bestScore = this.min;
    var currScore;
    var bestMove = null;
    var moves = board.getAvailableMoves();
    var corners = [[0, 0], [0, 2], [2, 0], [2, 2]];
    //Prunes a few options for the first few states
    if(board.turnCnt === 0)
        return [1, 1];
    else if(board.turnCnt === 1 && board.gamestate[1][1] === '')
        return [1, 1];
    else if(board.turnCnt === 1)
        return corners[Math.floor(Math.random() * 4)];

    for(var move in moves) {
        var newBoard = board.clone();
        newBoard.makeMove(this.marker, moves[move]);
        currScore = this.minimax(newBoard, this.opponent);
        console.log('Current score: ' + currScore);
        console.log('Current move: ' + moves[move]);
        if(currScore > bestScore) {
            bestScore = currScore;
```

```
                bestMove = moves[move];
            }
        }
        return bestMove;
    };

    //Evaluates the score for the board passed by checking each line
    this.evaluate = function(board) {
        var score = 0;

        score += this.evaluateLine(board, 0, 0, 0, 1, 0, 2);  // row 0
        score += this.evaluateLine(board, 1, 0, 1, 1, 1, 2);  // row 1
        score += this.evaluateLine(board, 2, 0, 2, 1, 2, 2);  // row 2
        score += this.evaluateLine(board, 0, 0, 1, 0, 2, 0);  // col 0
        score += this.evaluateLine(board, 0, 1, 1, 1, 2, 1);  // col 1
        score += this.evaluateLine(board, 0, 2, 1, 2, 2, 2);  // col 2
        score += this.evaluateLine(board, 0, 0, 1, 1, 2, 2);  // diagonal
        score += this.evaluateLine(board, 0, 2, 1, 1, 2, 0);  // alternate diagonal

        return score;
    };

    //Scores the line by checking each cell for our marker, 1 point for 1, 10 point
for 2, 100 for 3, opposite for opponent marker
    this.evaluateLine = function(board, r1, c1, r2, c2, r3, c3) {
        var score = 0;

        //First cell
        if(board.gamestate[r1][c1] === this.marker)
            score = 1;
        else if(board.gamestate[r1][c1] === this.opponent)
            score = -1;

        //Second cell
```

```
        if(board.gamestate[r2][c2] === this.marker){
            if(score == 1) //Cell 1 was my marker
                score = 10;
            else if (score === -1) // Cell 1 was my opponent
                return 0;
            else //Cell 1 was empty
                score = 1;
        }
        else if(board.gamestate[r2][c2] === this.opponent){
            if(score == -1) //Cell 1 was opponent
                score = -10;
            else if (score === 1) // Cell 1 was my marker
                return 0;
            else //Cell 1 was empty
                score = -1;
        }

        //Final cell
        if(board.gamestate[r3][c3] === this.marker){
            if(score > 1) //Cell 1 and or 2 was my marker
                score *= 10;
            else if (score < 0) // Cell 1 and or 2 was my opponent
                return 0;
            else //Cell 1 and 2 are empty
                score = 1;
        }
        else if(board.gamestate[r3][c3] === this.opponent){
            if(score < 0) //Cell 1 and or 2 was my opponent
                score *= 10;
            else if (score > 1) // Cell 1 and or 2 was my marker
                return 0;
            else //Cell 1 and 2 are empty
                score = -1;
        }
```

```
        return score;
    };
}

//Implementation of the board object
function Board() {
    this.turnCnt = 0;
    this.gamestate = [['','',''],
                                        ['','',''],
                                        ['','','']];
    //Returns the open positions on the board as an array of points as [row, column]
or [y, x]
    this.getAvailableMoves = function() {
        var moves = [];

        for(var row in this.gamestate)
            for(var col in this.gamestate[row])
                if(this.gamestate[row][col] === '')
                    moves.push([row, col]);

        return moves;
    };

    this.clone =    function() {
        var newBoard = new Board();

        //Copy over the positions of X's and O's and the turn number to the cloned
board
        for(var row = 0; row < 3; row++)
            for(var col = 0; col < 3; col++)
                newBoard.gamestate[row][col] = this.gamestate[row][col];
        newBoard.turnCnt = this.turnCnt;

        return newBoard;
```

```
    };

    //Will take in the player making the move as well as an [y, x] array of where to
place the player's marker
    this.makeMove = function(player, point) {
        var row = parseInt(point[0]);
        var col = parseInt(point[1]);
        this.gamestate[row][col] = player;
        this.turnCnt++;
    };

    this.isFull = function() {
        return this.turnCnt === 9;
    };

    this.checkForWin = function() {
        var boardState = this.gamestate;
        var winner;

        //checking the diagonals
        if(boardState[1][1] !== '' &&
             ((boardState[0][0] === boardState[1][1]
                 && boardState[2][2] === boardState[1][1])
                || (boardState[0][2] === boardState[1][1]
                        && boardState[2][0] === boardState[1][1]))) {
            winner = boardState[1][1];
            return winner;
        }
        else {
            //Checking the horizontals
            for(var row in boardState) {
                if(boardState[row][0] !== '' &&
                     boardState[row][0] === boardState[row][1]
                     && boardState[row][2] === boardState[row][1]) {
```

```
                    winner = boardState[row][0];
                    return winner;
                }
            }
            //Verticals
            for(var col in boardState) {
                if(boardState[0][col] !== '' &&
                     boardState[0][col] === boardState[1][col]
                     && boardState[1][col] === boardState[2][col]) {
                    winner = boardState[0][col];
                    return winner;
                }
            }
        }
    };
}
```

Файл на мові HTML для візуалізації та взаємодії з користувачем через маніпулятор «миша»:

```
<!-- <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="tiktaktor.css">
    <script src="tiktaktoe.js"></script>

    <title>Document</title>
</head>
<body>
    <div class="mainboard">
 <div class="field"> </div>
 <div class="field"> </div>
 <div class="field"> </div>
 <div class="field"> </div>
```

```
 <div class="field"> </div>
 <div class="field"> </div>
 <div class="field"> </div>
 <div class="field"> </div>
 <div class="field"> </div>
    </div>
<div class="lastscreenwin">
<div class="wintextmessage"></div>
<button class="playagain-btn">Play again</button>
</div>


</body>
</html> -->
<!DOCTYPE html>
<html>
    <head>
      <link rel="stylesheet" href="tiktaktor.css">
        <link href='https://fonts.googleapis.com/css?family=Arimo|Montserrat'
rel='stylesheet' type='text/css'>
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

        <script src="tiktaktoe.js"></script>
    </head>

    <body>
        <h1>Tic-Tac-Toe by Anastasiia</h1>
        <div id='selection'>
            Play as:
            <button class='hover'>X</button>
            <button class='hover'>O</button>
        </div>
        <div id='result'>
```

```
        </div>
        <table>
            <tr id='row0' class='row'>
                <td id='00' class='spot'> </td>
                <td id='01' class='spot'> </td>
                <td id='02' class='spot'> </td>
            </tr>
            <tr id='row1' class='row'>
                <td id='10' class='spot'> </td>
                <td id='11' class='spot'> </td>
                <td id='12' class='spot'> </td>
            </tr>
            <tr id='row2' class='row'>
                <td id='20' class='spot'> </td>
                <td id='21' class='spot'> </td>
                <td id='22' class='spot'> </td>
            </tr>
        </table>
    </body>
</html>
```

CSS – файл для стилизації зовнішнього вигляду проєкту:

```
/* *, *::after, *::before{
    box-sizing: borderbox;
}
body{
    margin:0;
}

:root{
    --cell-size:100px;
    --mark-size:cal(var(--cel-size) * .9);
}
```

```
.mainboard{
    width: 100vh;
    height:100vh;
    display:grid;
    justify-content: center;
    align-content: center;
    grid-template-columns: repeat(3 auto);
    justify-items: center;
    align-items: center;
}
.field{
    width:100px;
    height: 100px;
    border: 1px solid black;
    display: flex;
    justify-content: center;
    align-items: center;
    position: relative;
}
.field:first-child,
.field:nth-child(2),
.field:nth-child(3) {
    border-left:none;
}

.cell:nth-child(3n + 1) {
    border-left: none;
}

.cell:nth-child(3n + 3) {
    border-right: none;
}

.field:last-child,
```

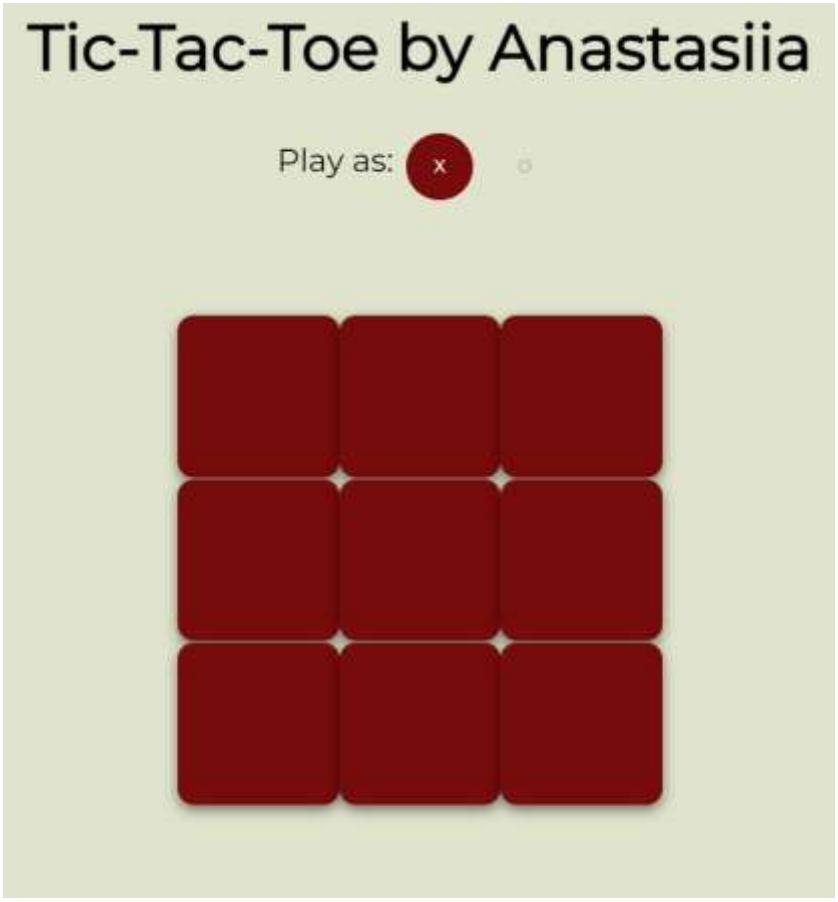
```
.field:last-child(8),
.field:last-child(7){
    border-bottom: none;
} */
body {
    background-color: #e0e4cc;
    font-family: "Montserrat", sans-serif;
    margin: 0;
    padding: 0;
    text-align: center;
}

h1 {
    font-size: 3em;
}

table {
    width: 75%;
    margin:0 auto;
    /* display: flex;
    justify-content: center;
    align-items: center; */
    margin-top: 20px;
    font-family: "Arimo", sans-serif;
    font-size: 70px;
}

td {
    display: inline-block;
    width: 80px;
    height: 80px;
    line-height: 80px;
    background-color: #760c0c ;
    color: white;
```

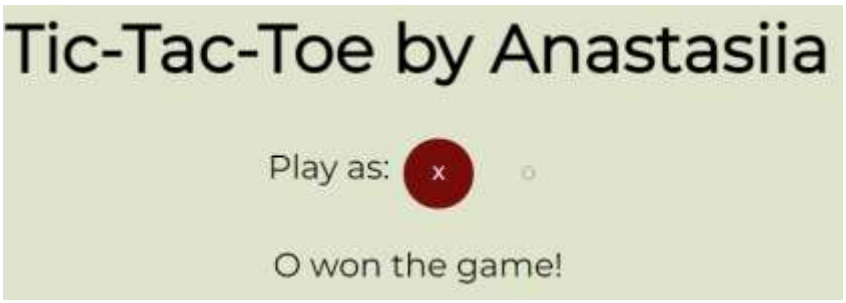
```
    border: none;
    border-radius: 10%;
    box-shadow: 0 0.03em 0.08em rgba(0, 0, 0, 0.5);
    margin: 8px;
    transition: background-color 0.2s ease;

}

td:hover {
    cursor: pointer;
    background-color: #F2D6CE;
}

.selected {
    background-color: #F2D6CE;
}

tr {
    list-style-type: none;
    width: 500px;
}

#selection {
    height: 60px;
    font-size: 1.5em;
}

.spot{
    /* display: flex;
    justify-content: center;
    align-items: center; */
      margin: 0 auto;
}
```

```css
button {
    border: none;
    background: transparent;
    width: 50px;
    height: 50px;
    margin: 0.3em;
    border-radius: 50%;
}

.hover:hover {
    background: #760c0c;
    color: white;
}

:active {
    outline: none;
}

:focus {
    outline: none;
}

.seed {
    background: #760c0c;
    color: white;
}

#result {
    padding-top: 20px;
    height: 40px;
    font-size: 1.5em;
}

@media only screen and (min-device-width: 1000px) {
```

```
    td {
        width: 120px;
        height: 120px;
        line-height: 120px;
        font-size: 120px;
    }
}
```

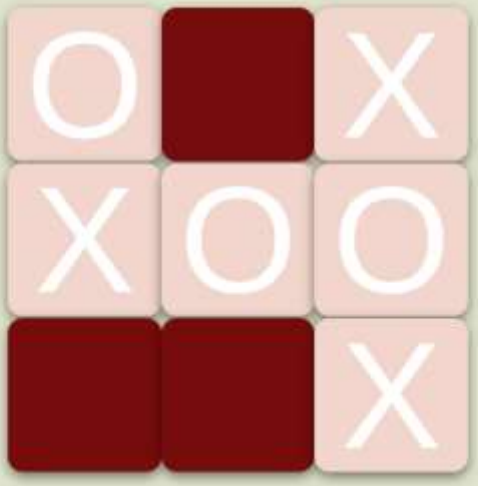
Результати:





Tic-Tac-Toe by Anastasiia

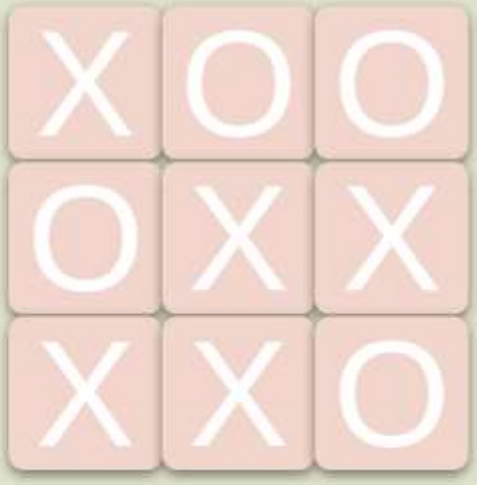
Play as: x o



Tic-Tac-Toe by Anastasiia

Play as: x o

The game was a draw!



Висновки:

Під час цієї лабораторної роботи ми розробили програмну реалізацію гри "хрестики-нулики" з використанням дерева рішень. Дерево рішень використовується для аналізу можливих ходів та їх наслідків у грі, що дозволяє програмі вибирати найоптимальніший хід в кожній ситуації.

У процесі розробки ми використовували мову програмування та відповідні алгоритми для створення гри "хрестики-нулики". Ми побудували дерево рішень, яке враховує всі можливі комбінації ходів гравців та програвача, оцінили їх виграшність та вибрали найкращий хід для кожної ситуації.

Перевірка роботи програми показала, що вона здатна грати в "хрестики-нулики" на високому рівні, уникати поразок та намагатися досягти перемоги. Дерево рішень дозволяє програмі робити відповідальні та обдумані ходи в кожній ситуації, що робить гру цікавою та викликає відчуття конкуренції.

Отримані результати свідчать про ефективність використання дерева рішень у програмній реалізації гри "хрестики-нулики". Такий підхід може бути застосований для розробки інших ігор та алгоритмів прийняття рішень, де важливо враховувати всі можливі варіанти та їх наслідки для досягнення оптимального результату.

У майбутньому можна розглянути можливість розширення функціональності програми, додавши підтримку для графічного інтерфейсу користувача, а також розробивши алгоритми для покращення штучного інтелекту та збільшення складності гри.