МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Практикум №5

з курсу «Основи розробки програмного забезпечення на платформі Microsoft.NET»

на тему: «Шаблони проектування.
Поведінкові шаблони»

Викладач:
Крамар Ю.М.

Виконав:
студент 2 курсу
групи ІП-15 ФІОТ
Костін В.А.

Київ-2023

Комп‘ютерний практикум № 5.

Тема: шаблони проектування, поведінкові шаблони.

Мета: ознайомитися з основними шаблонами проектування, навчитися застосовувати їх при проектуванні і розробці ПЗ.

Постановка задачі комп‘ютерного практикуму № 5

При виконанні комп‘ютерного практикуму необхідно виконати наступні дії:

1) Вивчити поведінкові патерни. Знати загальну характеристику та призначення кожного з них, особливості реалізації кожного з поведінкових патернів та випадки їх застосування.

2) Реалізувати задачу згідно варіанту, запропонованого нижче у вигляді консольного застосування на мові C#. Розробити інтерфейси та класи з застосування одного або декількох патернів. Повністю реалізувати методи, пов‘язані з реалізацією обраного патерну.

3) Повністю описати архітектуру проекту (призначення методів та класів), особливості реалізації обраного патерну. Для кожного патерну необхідно вказати основні класи та їх призначення.

4) Навести UML-діаграму класів

5) Звіт повинен містити:

a. обґрунтування обраного патерну (чому саме він);

b. опис архітектури проекту (призначення методів та класів);

c. UML-діаграму класів

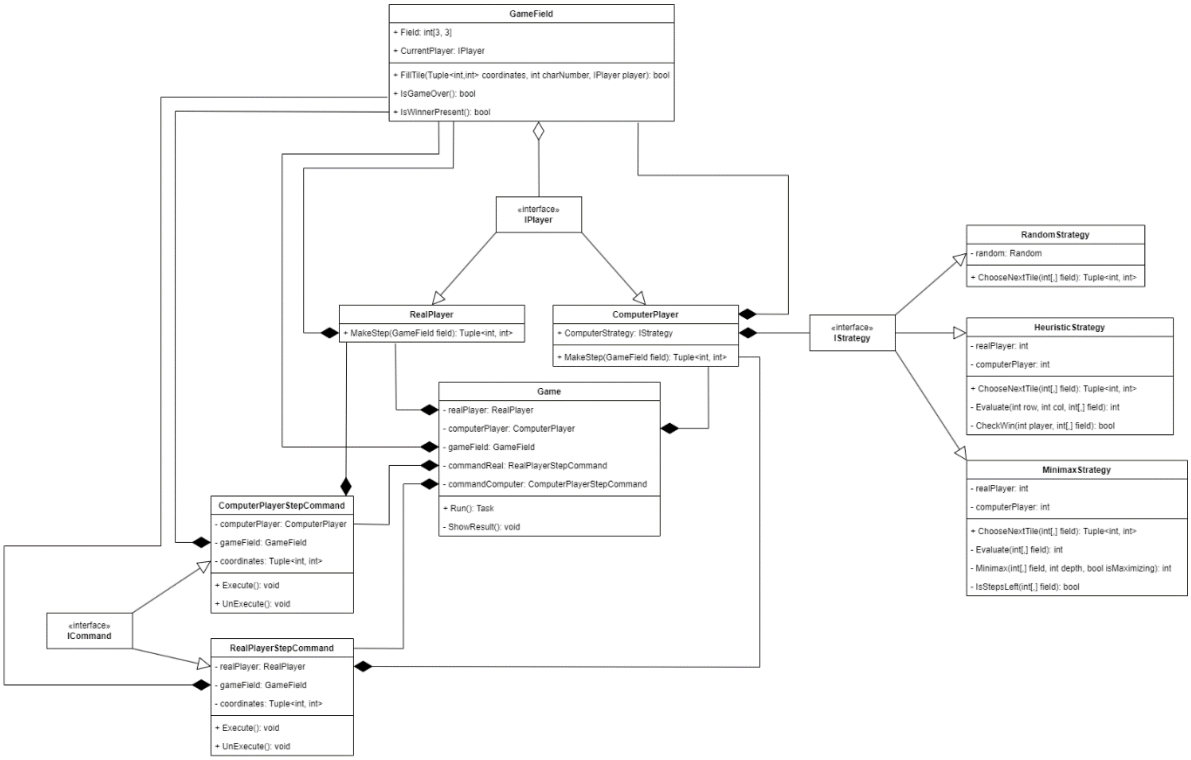
d. особливості реалізації обраного патерну

e. текст програмного коду

f. скріншоти результатів виконання.

2) Реалізувати алгоритм гри «хрестики-нулики». Реалізувати можливість «взяти назад хід».

UML-діаграма класів:



Архітектура програми

Назва класу: GameField

Призначення: поле для гри в “хрестики-нулики”.

Опис властивостей:

int[3, 3] Field – матриця, яка є імітацією поля. Значення 0 певного елемента вказує на те, що клітинка вільна, 1 – в клітинку зробив хід перший гравець, а 2 – другий гравець.

IPlayer CurrentPlayer – гравець, який зробив хід останнім.

Опис методів:

bool FillTile(Tuple<int, int> coordinates, int charNumber, IPlayer player) – метод для заповнення гравцем клітинки поля певним символом. На вхід приймає координати поля, номер символу(0 - порожній символ, 1 - X, 2 - O), гравець який робить хід. Повертає True - якщо заповнення клітинки пройшло вдало, інакше - false.

bool IsGameOver() – метод для перевірки закінчення гри.

bool IsWinnerPresent() – метод що перевіряє чи є в грі переможець.

Назва класу: RealPlayer

Призначення: реальний гравець у гру “хрестики-нулики”. Реальний гравець робить хід “хрестиками”.

Опис методів:

Tuple<int, int> MakeStep(GameField field) – метод для ходу реального гравця(запит координат для ходу у користувача).

Назва класу: ComputerPlayer

Призначення: комп’ютерний гравець у гру “хрестики-нулики”, який грає за обраною “стратегією”. Комп'ютер робить хід “нуликами”.

Опис властивостей:

IStrategy ComputerStrategy – “стратегія” гри комп’ютера у гру “христики-нулики”.

Опис методів:

Tuple<int, int> MakeStep(GameField field) – метод для ходу комп’ютерного гравця за обраною “стратегією”.

Назва класу: RandomStrategy

Призначення: стратегія комп'ютера для гри "хрестики-нулики" із застосуванням методу обирання рандомної вільної клітинки.

Опис методів:

Tuple<int, int> ChooseNextTile(int[,] field) – метод для ходу обирання клітинки наступного ходу на рандом.

Назва класу: HeuristicStrategy

Призначення: Стратегія комп'ютера для гри "хрестики-нулики" із застосуванням евристичного алгоритму з обмеженим пошуком.

Опис методів:

Tuple<int, int> ChooseNextTile(int[,] field) – метод для ходу обирання клітинки наступного ходу з використанням евристичного алгоритму.

Назва класу: MinimaxStrategy

Призначення: Стратегія комп'ютера для гри "хрестики-нулики" із застосуванням алгоритму Minimax.

Опис методів:

Tuple<int, int> ChooseNextTile(int[,] field) – метод для ходу обирання клітинки наступного ходу з використанням алгоритму Minimax.

Назва класу: RealPlayerStepCommand

Призначення: “Команда” ходу реального гравця.

Опис методів:

void Execute() – метод для виконання ходу реально гравця.

void UnExecute() – метод для виконання “ходу назад” реального гравця.

Назва класу: ComputerPlayerStepCommand

Призначення: “Команда” ходу комп’ютерного гравця.

Опис методів:

void Execute() – метод для виконання ходу комп’ютерного гравця.

void UnExecute() – метод для виконання “ходу назад” комп’ютерного гравця.

Назва класу: Game

Призначення: гра в “хрестики-нулики”.

Опис методів:

Task Run() – метод для початку гри в “хрестики нулики”.

Шаблони

В данній лабораторній роботі було використано два поведінкових паттерни: команда та стратегія.

Використання паттерну “Команда” обумовлено необхідністю реалізувати функціонал повернення ходу гравця. Тому в команді були реалізовані методи як для виконання ходу, так і для ходу назад.

Для реалізації функціоналу обирання складності гри користувачем доцільно реалізувати паттерн “Стратегія”. Головна мета цього шаблону в нашій програмі – це розробка різних алгоритмів гри комп'ютера у гру “хрестики-нулики” та використання певного алгоритму залежно від обраного рівня слкадності.

Реалізація шаблонів

Паттерн “Команда” був реалізований за допомогою створення двох класів зі спільним інтерфейсом ICommand: RealPlayerStepCommand та ComputerPlayerStepCommand –, які реалізують логіку ходу реального та комп’ютерного гравця відповідно. В командах реалізовано два методи: Execute() та UnExecute() – для виконнання ходу та відміни ходу відповідно.

Паттерн “Стратегія” реалізований за допомогою створення спільного для всіх алгоритмів(стратегій) інтерфейсу IStrategy. В класі ComputerPlayer створена властивість типу IStrategy для подальшої ініціалізації цього поля обраним алгоритмом. Всього реалізовано три алгоритми: RandomStrategy – легкий рівень, HeuristicStrategy – середній рівень, MinimaxStrategy – складний рівень. Після обрання рівня складності користувачем комп’ютерно гравцеві присвоюється обраний алгоритм для подальшої гри.

Код програми

Код програми можна подивитись на репозиторії за посиланням:

https://github.com/VadimkaKostin/.Net

GameField.cs

```
public class GameField
    {
        //Матриця що задає поле
        private int[,] _field = new int[3, 3] { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0,
0 } };
```

```
        public int[,] Field { get { return _field; } }

        //Властивість, яка вказує який гравець зробив хід останнім
        public IPlayer CurrentPlayer { get; set; }

        /// <summary>
        /// Метод для заповнення гравцем клітинки поля певним символом.
        /// </summary>
        /// <param name="coordinates">Координати поля.</param>
        /// <param name="charNumber">Номер символу(0 - порожній символ, 1 - X, 2
- O).</param>
        /// <param name="player">Гравець який робить хід.</param>
        /// <returns>True - якщо заповнення клітинки пройшло вдало, інакше -
false.</returns>
        public bool FillTile(Tuple<int, int> coordinates, int charNumber, IPlayer
player)
        {
            //Валідація індексів
            if((coordinates.Item1 > 2 || coordinates.Item1 < 0) ||
(coordinates.Item2 > 2 || coordinates.Item2 < 0))
                return false;

            //Якщо гравець намагається помітити вже зайняту клітинку, хід
завершується невдало
            if (charNumber != 0 && (_field[coordinates.Item1, coordinates.Item2]
== 1 || _field[coordinates.Item1, coordinates.Item2] == 2))
                return false;

            _field[coordinates.Item1, coordinates.Item2] = charNumber;

            CurrentPlayer = player;

            return true;
        }

        /// <summary>
        /// Метод для перевірки закінчення гри.
        /// </summary>
        /// <returns>True - якщо гра завершена, в іншому випадку -
false.</returns>
        public bool IsGameOver()
        {
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    if (_field[i, j] == 0)
                        return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Метод що перевіряє чи є в грі переможець.
        /// </summary>
        /// <returns>True - якщо гра завершена перемогою одного з гравців, в
іншому випадку - false.</returns>
        public bool IsWinnerPresent()
        {
            return _field[0, 0] == _field[0, 1] && _field[0, 1] == _field[0, 2]
&& _field[0, 0] != 0 ||
```

```
                    _field[1, 0] == _field[1, 1] && _field[0, 1] == _field[1, 2]
&& _field[1, 0] != 0 ||
                    _field[2, 0] == _field[2, 1] && _field[2, 1] == _field[2, 2]
&& _field[2, 0] != 0 ||
                    _field[0, 0] == _field[1, 0] && _field[1, 0] == _field[2, 0]
&& _field[0, 0] != 0 ||
                    _field[0, 1] == _field[1, 1] && _field[1, 1] == _field[2, 1]
&& _field[0, 1] != 0 ||
                    _field[0, 2] == _field[1, 2] && _field[1, 2] == _field[2, 2]
&& _field[0, 2] != 0 ||
                    _field[0, 0] == _field[1, 1] && _field[1, 1] == _field[2, 2]
&& _field[0, 0] != 0 ||
                    _field[0, 2] == _field[1, 1] && _field[1, 1] == _field[2, 0]
&& _field[0, 2] != 0;
        }

        public override string ToString()
        {
            var sb = new StringBuilder();

            for(int i = 0; i < 3; i++)
            {
                sb.Append("-------------------\n");
                sb.Append("|     |     |     |\n|");
                for(int j = 0; j < 3; j++)
                {
                    char tile = ' ';

                    switch (_field[i,j])
                    {
                        case 1:
                            tile = 'X';
                            break;
                        case 2:
                            tile = 'O';
                            break;
                    }

                    sb.Append($"  {tile}  |");
                }
                sb.Append("\n|     |     |     |\n");
            }
            sb.Append("-------------------\n");

            return sb.ToString();
        }
    }
```

RealPlayer.cs

```
public class RealPlayer : IPlayer
    {
        public Tuple<int, int> MakeStep(GameField field)
        {
            int row = -1, col = -1;

            while (true)
            {
                Console.Write("\nУведіть координати клітинки для ходу: ");

                try
                {
                    string? str = Console.ReadLine();
```

```
                    string[] args = str.Split(' ');
                    row = int.Parse(args[0]);
                    col = int.Parse(args[1]);
                }
                catch(Exception)
                {
                    Console.WriteLine("Неправильний формат координат");
                    continue;
                }

                break;
            }

            return Tuple.Create(--row, --col);
        }
    }
```

ComputerPlayer.cs

```
public class ComputerPlayer : IPlayer
    {
        public IStrategy ComputerStrategy { get; set; }

        public Tuple<int, int> MakeStep(GameField field)
        {
            return ComputerStrategy.ChooseNextTile(field.Field);
        }
    }
```

RandomStrategy.cs

```
public class RandomStrategy : IStrategy
    {
        private readonly Random _random = new Random();
        public Tuple<int, int> ChooseNextTile(int[,] field)
        {
            int row, col;

            do
            {
                row = _random.Next(0, 3);
                col = _random.Next(0, 3);
            } while (field[row, col] != 0);

            return Tuple.Create(row, col);
        }
    }
```

HeuristicStrategy.cs

```
public class HeuristicStrategy : IStrategy
    {
        private int realPlayer = 1;
        private int computerPlayer = 2;

        public Tuple<int, int> ChooseNextTile(int[,] field)
        {
            Tuple<int, int> bestMove = null;
            int bestScore = int.MinValue;

            for (int row = 0; row < 3; row++)
            {
                for (int col = 0; col < 3; col++)
```

```
                {
                    if (field[row, col] == 0)
                    {
                        int score = Evaluate(row, col, field);
                        if (score > bestScore)
                        {
                            bestScore = score;
                            bestMove = Tuple.Create(row, col);
                        }
                    }
                }
            }

            return bestMove;
        }

        /// <summary>
        /// Метод для оцінювання ходу комп'ютера.
        /// </summary>
        /// <param name="row">Рядок ходу комп'ютеру.</param>
        /// <param name="col">Ствопецю ходу комп'ютера.</param>
        /// <param name="field">Поле для оцінювання.</param>
        /// <returns>100 - якщо комп'ютер вийграє, 50 - якщо він блокує хід
гравця, інакше - 0.</returns>
        private int Evaluate(int row, int col, int[,] field)
        {
            int score = 0;

            // Виграшні комбінації
            int[,] winPositions = new int[,]
            {
                { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 },
                { 0, 3, 6 }, { 1, 4, 7 }, { 2, 5, 8 },
                { 0, 4, 8 }, { 2, 4, 6 }
            };

            field[row, col] = computerPlayer;
            if (CheckWin(computerPlayer, field))
            {
                score = 100;
            }
            else
            {
                field[row, col] = realPlayer;
                if (CheckWin(realPlayer, field))
                {
                    score = 50;
                }
            }

            field[row, col] = 0;
            return score;
        }

        /// <summary>
        /// Метод для перевірки чи є переможна комбінація для гравця на полі.
        /// </summary>
        /// <param name="player">Номер гравця(1 - реальний грок, 2 -
комп'ютер).</param>
        /// <param name="field">Поле для перевірки.</param>
        /// <returns>True - якщо вийграшна комбінація існує, інакше -
false.</returns>
        private bool CheckWin(int player, int[,] field)
```

```
        {
            // Виграшні комбінації
            int[,] winPositions = new int[,]
            {
                { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 },
                { 0, 3, 6 }, { 1, 4, 7 }, { 2, 5, 8 },
                { 0, 4, 8 }, { 2, 4, 6 }
            };

            for (int i = 0; i < 8; i++)
            {
                int pos1 = winPositions[i, 0],
                    pos2 = winPositions[i, 1],
                    pos3 = winPositions[i, 2];

                if (field[pos1 / 3, pos1 % 3] == player &&
                    field[pos2 / 3, pos2 % 3] == player &&
                    field[pos3 / 3, pos3 % 3] == player)
                {
                    return true;
                }
            }

            return false;
        }
    }
```

MinimaxStrategy.cs

```
public class MinimaxStrategy : IStrategy
    {
        private int realPlayer = 1;
        private int computerPlayer = 2;

        public Tuple<int, int> ChooseNextTile(int[,] field)
        {
            int bestScore = int.MinValue;
            Tuple<int, int> bestMove = null;

            for (int row = 0; row < 3; row++)
            {
                for (int col = 0; col < 3; col++)
                {
                    if (field[row, col] == 0)
                    {
                        field[row, col] = computerPlayer;
                        int score = Minimax(field, 0, false);
                        field[row, col] = 0;

                        if (score > bestScore)
                        {
                            bestScore = score;
                            bestMove = Tuple.Create(row, col);
                        }
                    }
                }
            }

            return bestMove;
        }

        /// <summary>
        /// Метод для розразунку оцінки ходу.
```

```
        /// </summary>
        /// <param name="field">Поле для розразунку оцінки.</param>
        /// <returns>10 - якщо перемагає комп'ютер, -10 - якщо перемагає
користувач, 0 - якщо нічия.</returns>
        private int Evaluate(int[,] field)
        {
            // Перевірка на рядки
            for (int row = 0; row < 3; row++)
            {
                if (field[row, 0] == field[row, 1] && field[row, 1] == field[row,
2])
                {
                    if (field[row, 0] == computerPlayer)
                    {
                        return 10;
                    }
                    else if (field[row, 0] == realPlayer)
                    {
                        return -10;
                    }
                }
            }

            // Перевірка на стовпці
            for (int col = 0; col < 3; col++)
            {
                if (field[0, col] == field[1, col] && field[1, col] == field[2,
col])
                {
                    if (field[0, col] == computerPlayer)
                    {
                        return 10;
                    }
                    else if (field[0, col] == realPlayer)
                    {
                        return -10;
                    }
                }
            }

            // Перевірка на діагоналі
            if ((field[0, 0] == field[1, 1] && field[1, 1] == field[2, 2]) ||
                (field[0, 2] == field[1, 1] && field[1, 1] == field[2, 0]))
            {
                if (field[1, 1] == computerPlayer)
                {
                    return 10;
                }
                else if (field[1, 1] == realPlayer)
                {
                    return -10;
                }
            }

            return 0;
        }

        /// <summary>
        /// Метод який реалізує рекурсивний алгоритм Minimax, за допомогою якого
можна зробити рекурсивне
        /// дослідження всіх сценаріїв розвитку подій та обрати найкращий.
        /// </summary>
        /// <param name="field">Поле для дослідження.</param>
```

```
        /// <param name="depth">Глибина занурення.</param>
        /// <param name="isMaximizing">Прапорець максимізації(На різних рівнях в
нас можде бути
        /// або максимізація ходу комп'ютера, або мінімізація ходу реального
гравця.</param>
        /// <returns>Оцінка ходу.</returns>
        private int Minimax(int[,] field, int depth, bool isMaximizing)
        {
            int score = Evaluate(field);

            if (score == 10 || score == -10)
            {
                return score;
            }

            if (!IsStepsLeft(field))
            {
                return 0;
            }

            if (isMaximizing)
            {
                int bestScore = int.MinValue;

                for (int row = 0; row < 3; row++)
                {
                    for (int col = 0; col < 3; col++)
                    {
                        if (field[row, col] == 0)
                        {
                            field[row, col] = computerPlayer;
                            bestScore = Math.Max(bestScore, Minimax(field, depth
+ 1, !isMaximizing));
                            field[row, col] = 0;
                        }
                    }
                }

                return bestScore;
            }
            else
            {
                int bestScore = int.MaxValue;

                for (int row = 0; row < 3; row++)
                {
                    for (int col = 0; col < 3; col++)
                    {
                        if (field[row, col] == 0)
                        {
                            field[row, col] = realPlayer;
                            bestScore = Math.Min(bestScore, Minimax(field, depth
+ 1, !isMaximizing));
                            field[row, col] = 0;
                        }
                    }
                }

                return bestScore;
            }
        }

        /// <summary>
```

```
        /// Метод для перевірки чи незаповненні вже всі клітинки поля.
        /// </summary>
        /// <param name="field">Поле для перевірки.</param>
        /// <returns>True - якщо всі клітинки поля заповненні, інакше -
false.</returns>
        private bool IsStepsLeft(int[,] field)
        {
            for (int row = 0; row < 3; row++)
            {
                for (int col = 0; col < 3; col++)
                {
                    if (field[row, col] == 0)
                    {
                        return true;
                    }
                }
            }
            return false;
        }
    }
```

RealPlayerStepCommand.cs

```
public class RealPlayerStepCommand : ICommand
    {
        private RealPlayer _realPlayer;
        private GameField _gameField;

        private Tuple<int, int> _coordinates;

        public RealPlayerStepCommand(RealPlayer realPlayer, GameField gameField)
        {
            _realPlayer = realPlayer;
            _gameField = gameField;
        }

        public void Execute()
        {
            do
            {
                _coordinates = this._realPlayer.MakeStep(this._gameField);
            }
            while (!this._gameField.FillTile(_coordinates, 1, _realPlayer));
        }

        public void UnExecute()
        {
            this._gameField.FillTile(_coordinates, 0, _realPlayer);
        }
    }
```

ComputerPlayerStepCommand.cs

```
public class ComputerPlayerStepCommand : ICommand
    {
        private ComputerPlayer _computerPlayer;
        private GameField _gameField;

        private Tuple<int, int> _coordinates;

        public ComputerPlayerStepCommand(ComputerPlayer computerPlayer, GameField
gameField)
        {
```

```
            _computerPlayer = computerPlayer;
            _gameField = gameField;
        }

        public void Execute()
        {
            _coordinates = this._computerPlayer.MakeStep(this._gameField);

            this._gameField.FillTile(_coordinates, 2, _computerPlayer);
        }

        public void UnExecute()
        {
            this._gameField.FillTile(_coordinates, 0, _computerPlayer);
        }
    }
```

Game.cs

```
public class Game
    {
        private RealPlayer _realPlayer;
        private ComputerPlayer _computerPlayer;

        private GameField _gameField;

        private RealPlayerStepCommand _commandReal;
        private ComputerPlayerStepCommand _commandComputer;

        public Game(RealPlayer realPlayer, ComputerPlayer computerPlayer)
        {
            _realPlayer = realPlayer;
            _computerPlayer = computerPlayer;

            _gameField = new GameField();
        }

        public async Task Run()
        {
            Console.WriteLine(this._gameField);

            while (true)
            {
                //Крок реального гравця
                _commandReal = new RealPlayerStepCommand(_realPlayer,
_gameField);
                _commandReal.Execute();

                Console.Clear();

                Console.WriteLine(this._gameField);


                if (_gameField.IsWinnerPresent() || _gameField.IsGameOver())
break;

                //Крок комп'ютера
                Console.WriteLine("Хід комп'ютера...");

                await Task.Delay(2000);

                _commandComputer = new ComputerPlayerStepCommand(_computerPlayer,
_gameField);
```

```
                _commandComputer.Execute();

                Console.Clear();

                Console.WriteLine(this._gameField);

                if (_gameField.IsWinnerPresent() || _gameField.IsGameOver())
break;

                //Хід назад
                Console.Write("\nЗробити хід назад(1/0): ");

                int choise = 0;

                try
                {
                    choise = int.Parse(Console.ReadLine());
                }
                catch (Exception) { }

                if (choise == 1)
                {
                    _commandReal.UnExecute();
                    _commandComputer.UnExecute();
                }

                Console.Clear();

                Console.WriteLine(this._gameField);
            }

            this.ShowResult();
        }
        private void ShowResult()
        {
            if (_gameField.IsWinnerPresent())
            {
                if (_gameField.CurrentPlayer == _realPlayer)
                {
                    Console.BackgroundColor = ConsoleColor.Green;
                    Console.WriteLine("\nВи перемогли!");
                    Console.BackgroundColor = ConsoleColor.Black;
                }
                else
                {
                    Console.BackgroundColor = ConsoleColor.Red;
                    Console.WriteLine("\nКомп'ютер переміг!");
                    Console.BackgroundColor = ConsoleColor.Black;
                }

                return;
            }

            Console.WriteLine("Гра завершена, переможця немає!");
        }
    }
```

Program.cs

```
public static class Program
    {
        public static async Task Main(string[] args)
        {
```

```
            Console.OutputEncoding = Encoding.UTF8;

            Console.Write("Оберіть складність гри\n1 - легка\n2 - середня\n3 -
висока\n\nСкладінсть: ");

            int hardLevel = int.Parse(Console.ReadLine());

            Console.Clear();

            RealPlayer realPlayer = new RealPlayer();

            ComputerPlayer computerPlayer = new ComputerPlayer();

            switch(hardLevel)
            {
                case 1:
                    computerPlayer.ComputerStrategy = new RandomStrategy();
                    break;
                case 2:
                    computerPlayer.ComputerStrategy = new HeuristicStrategy();
                    break;
                case 3:
                    computerPlayer.ComputerStrategy = new MinimaxStrategy();
                    break;
            }

            Game game = new Game(realPlayer, computerPlayer);

            await game.Run();
        }
    }
```

Результат роботи програми

Під час роботи програми користувач обирає складінсть гри:

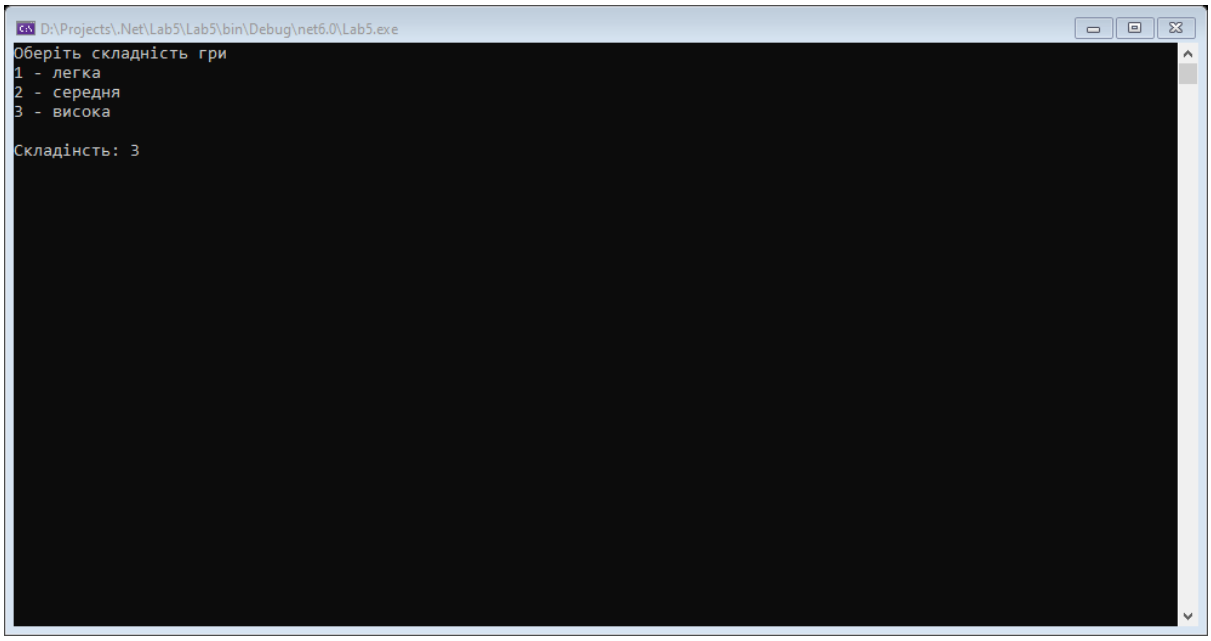


Рисунок 1.1 – введення користувачем номеру складності гри.

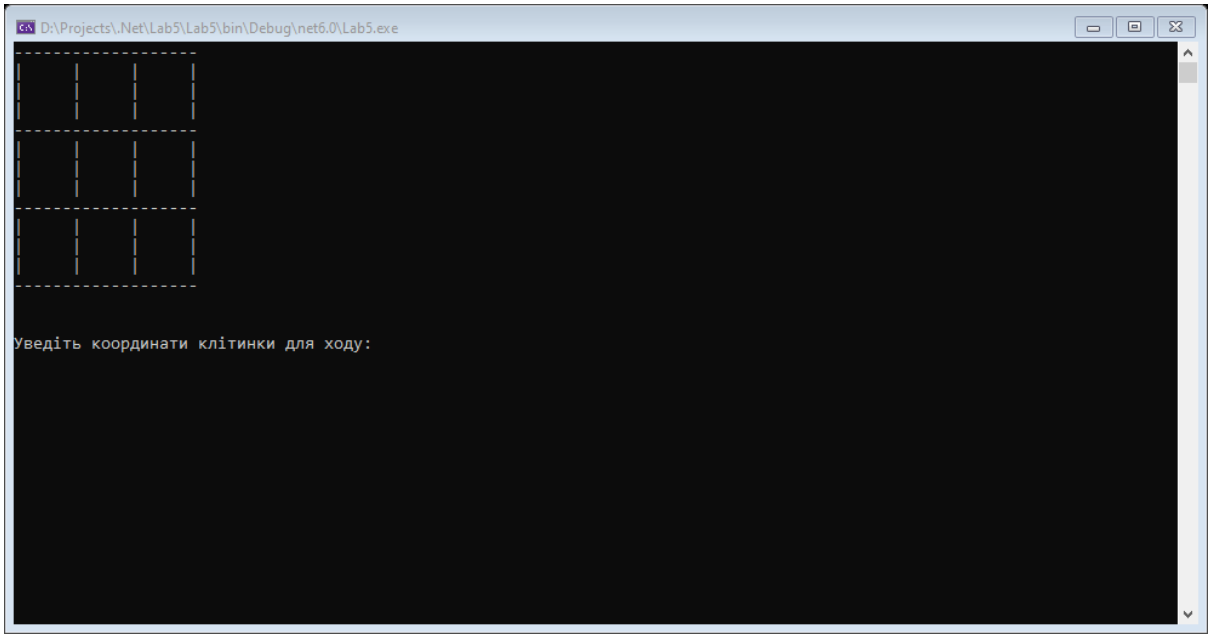


Рисунок 1.2 – початок гри, перший хід за реальним гравцем.

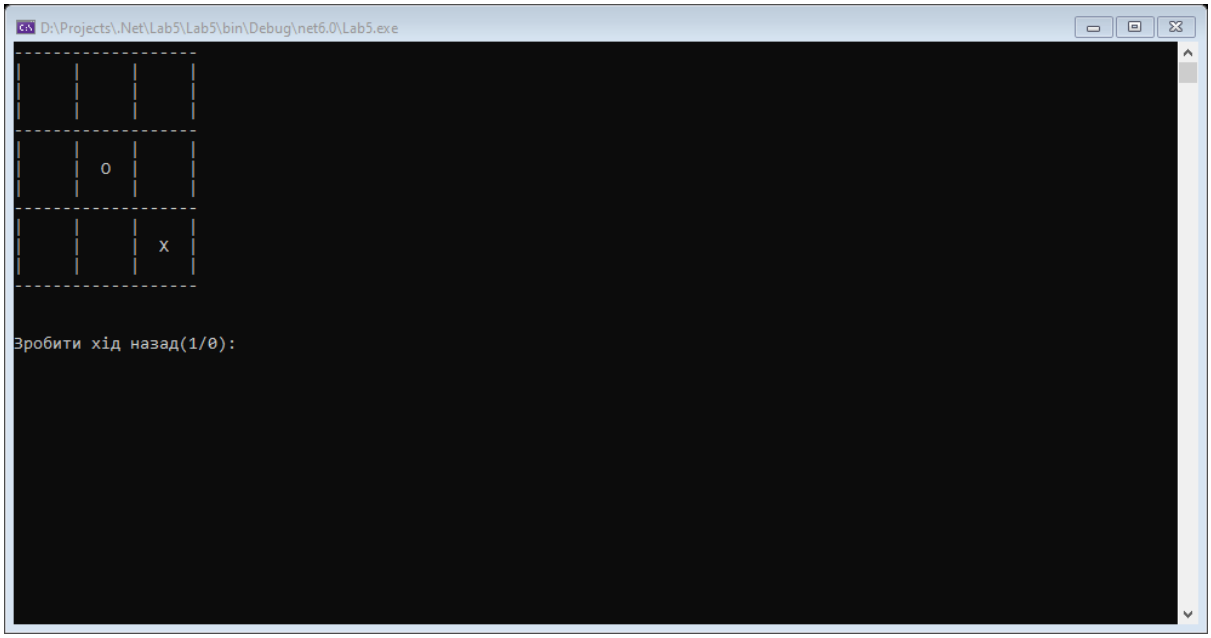


Рисунок 1.3 – після ходу реального гравця хрестиками свій хід виконує комп’ютер, користувачеві пропонують зробити хід назад.

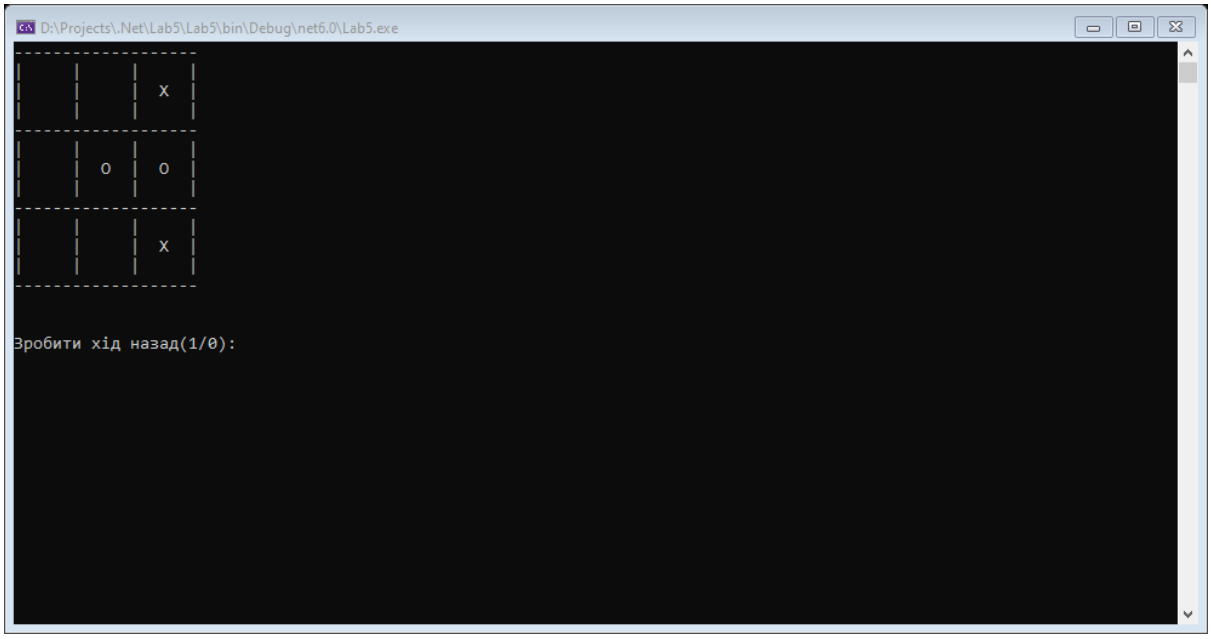


Рисунок 1.4 – при продовженні гри, комп’ютер за допомгою алгоритму Minimax буде блокувати спроби реального гравця перемогти.

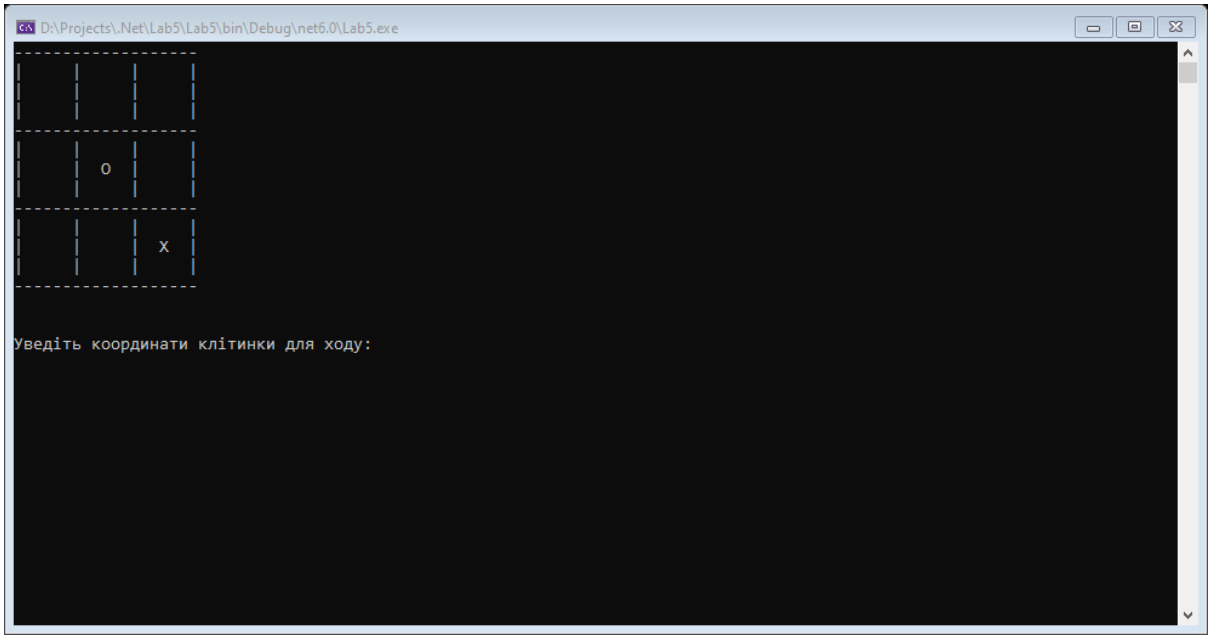


Рисунок 1.5 – при відмінні ходу реальний гравець і комп’ютер роблять хід назад.

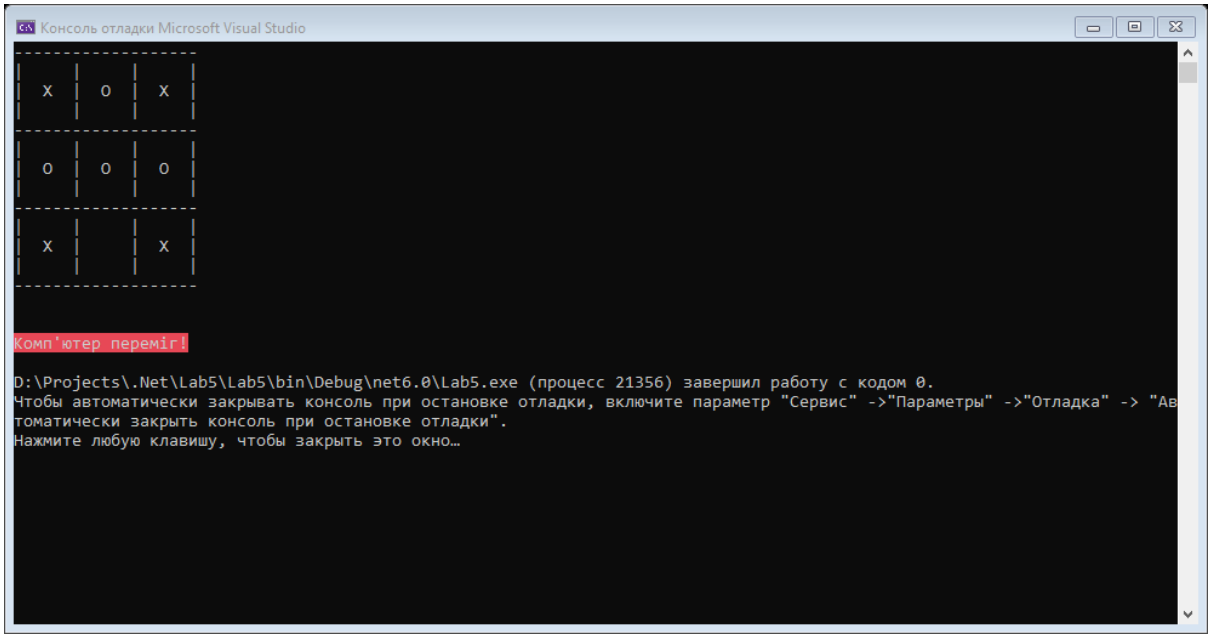


Рисунок 1.6 – комп’ютер перемагає.

Висновок

Протягом п’ятого комп’ютерного практикуму ми ознайомитися зі поведінковими шаблонами. Був розроблений консольний застосунок-гра у “хрестики-нулики”. Під час розробки застосунку були реалізовані паттерни “Команда” та “Стратегія”. Необхідність в першому паттерні полягає у реалізації функціоналу виконання “ходу назад”. Другий паттерн розроблений для реалізації можливості обирати рівень складності гри користувачем.