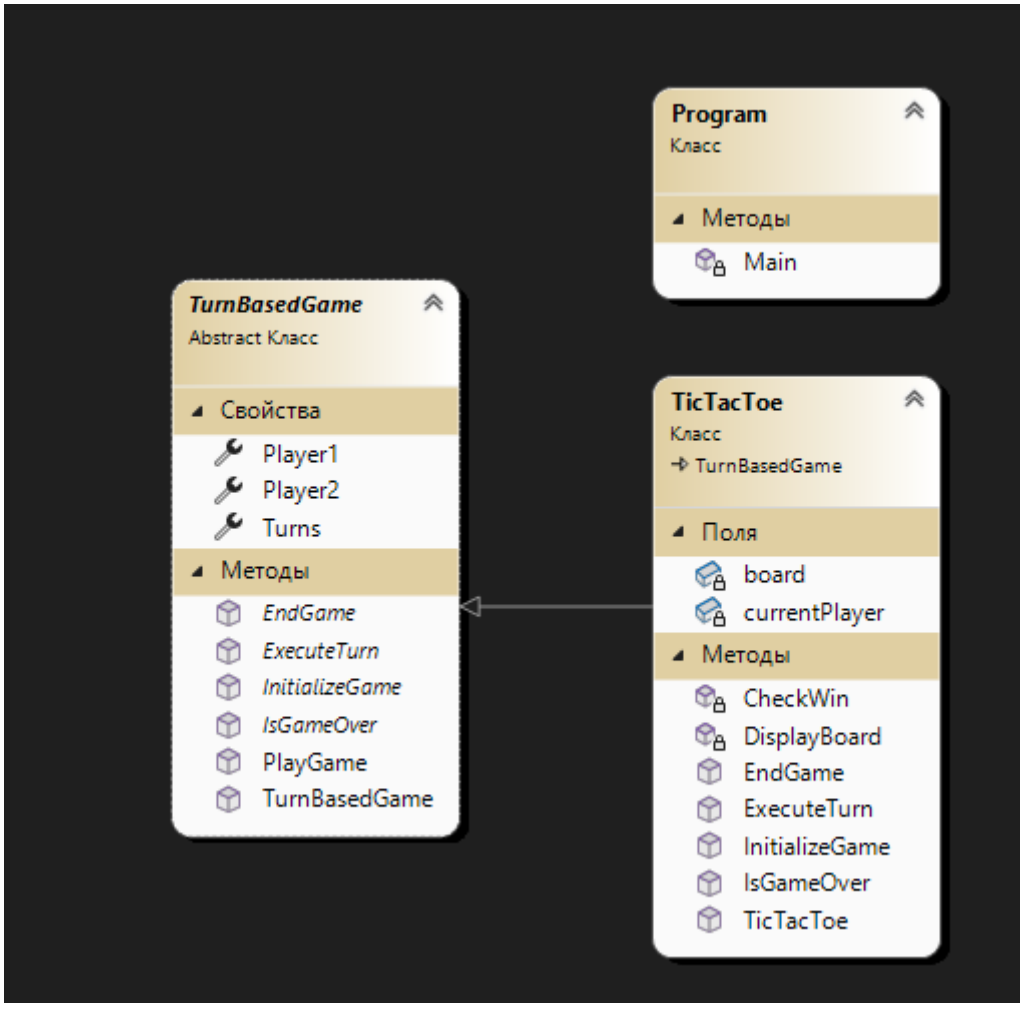
Отчет по использованию паттернов в проекте

1. Пошаговая игра (Паттерн «Шаблонный метод»)

1.1 Описание предметной области

Пошаговая игра реализует игровую логику, в которой игроки совершают ходы по очереди. Алгоритм игры включает инициализацию, выполнение хода, проверку условий завершения и завершение игры. Данный процесс формализован с помощью паттерна "Шаблонный метод".

1.2 UML-диаграмма классов



1.3 Краткий вывод по использованию паттерна

Паттерн "Шаблонный метод" позволил задать общую структуру игры, оставляя реализацию конкретных шагов (инициализация, выполнение хода, проверка завершения) для подклассов. Это упрощает расширяемость кода и обеспечивает единый алгоритм игры.

1.4 Листинг кода

Абстрактный класс TurnBasedGame (Шаблонный метод)

```
namespace Task1
{
    internal abstract class TurnBasedGame
    {
        public string Player1 { get; set; }
        public string Player2 { get; set; }
        public int Turns { get; set; }

        public TurnBasedGame(string player1, string player2)
        {
            Player1 = player1;
            Player2 = player2;
            Turns = 0;
        }

        public abstract void InitializeGame();
        public abstract void ExecuteTurn();
        public abstract bool IsGameOver();
        public abstract void EndGame();

        public void PlayGame()
        {
            InitializeGame();
            while (!IsGameOver())
            {
                ExecuteTurn();
            }
            EndGame();
        }
    }

}
```

Класс TicTacToe

```
namespace Task1
{
    internal class TicTacToe : TurnBasedGame
    {
        private char[,] board;
        private char currentPlayer;

        public TicTacToe(string player1, string player2) : base(player1, player2)
        {
            board = new char[3, 3];
            currentPlayer = 'X';
        }

        public override void InitializeGame()
        {
            Console.WriteLine("Игра \"Крестики-нолики\" началась!");
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    board[i, j] = '-';
                }
            }
        }

        public override void ExecuteTurn()
        {
            Console.WriteLine($"Ход игрока {currentPlayer}");
            DisplayBoard();
```

```
            Console.Write("Введите строку (0-2): ");
            int row = int.Parse(Console.ReadLine());
            Console.Write("Введите столбец (0-2): ");
            int col = int.Parse(Console.ReadLine());

            if (board[row, col] == '-')
            {
                board[row, col] = currentPlayer;
                currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
                Turns++;
            }
            else
            {
                Console.WriteLine("Клетка занята, попробуйте снова.");
            }
        }

        public override bool IsGameOver()
        {
            return CheckWin() || Turns >= 9;
        }

        public override void EndGame()
        {
            DisplayBoard();
            if (CheckWin())
            {
                char winner = (currentPlayer == 'X') ? 'O' : 'X';
                Console.WriteLine($"Игрок {winner} победил!");
            }
            else
            {
                Console.WriteLine("Ничья!");
            }
        }

        private void DisplayBoard()
        {
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    Console.Write(board[i, j] + " ");
                }
                Console.WriteLine();
            }
        }

        private bool CheckWin()
        {
            for (int i = 0; i < 3; i++)
            {
                if (board[i, 0] != '-' && board[i, 0] == board[i, 1] && board[i, 1]
== board[i, 2]) return true;
                if (board[0, i] != '-' && board[0, i] == board[1, i] && board[1, i]
== board[2, i]) return true;
            }
            if (board[0, 0] != '-' && board[0, 0] == board[1, 1] && board[1, 1] ==
board[2, 2]) return true;
            if (board[0, 2] != '-' && board[0, 2] == board[1, 1] && board[1, 1] ==
board[2, 0]) return true;
            return false;
        }
    }
```

```
}
```

Класс Program

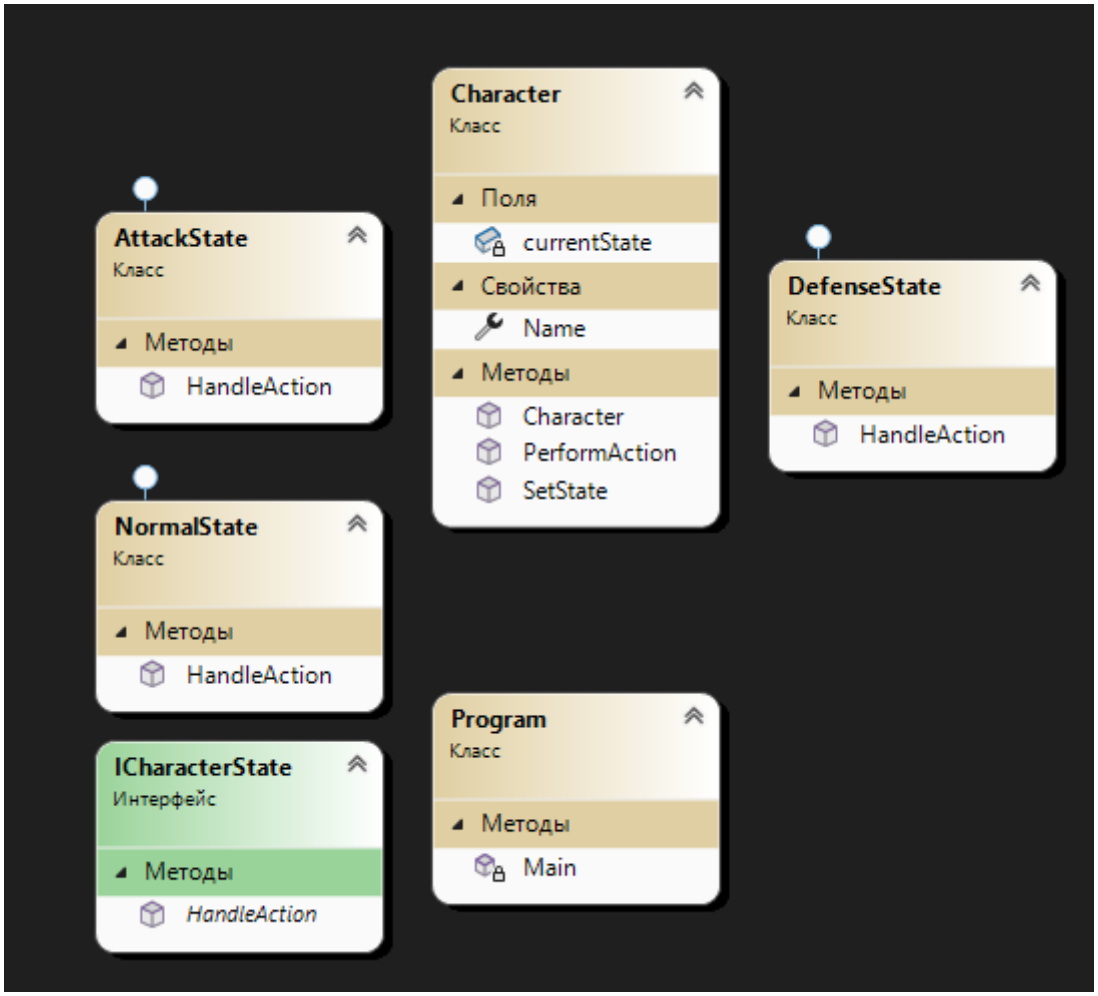
```
internal class Program
 {
     static void Main(string[] args)
     {
         TurnBasedGame game = new TicTacToe("Игрок 1", "Игрок 2");
         game.PlayGame();
     }

 }
```

2. Состояния персонажа (Паттерн «Состояние»)

2.1 Описание предметной области

Игровой персонаж может находиться в разных состояниях (например, атака, защита, нормальное состояние). В зависимости от состояния меняется его поведение. Паттерн "Состояние" позволяет динамически изменять состояние объекта и его поведение без изменения основного кода персонажа.

2.2 UML-диаграмма классов



2.3 Краткий вывод по использованию паттерна

Паттерн "Состояние" позволил избежать множества условных операторов в коде и предоставил гибкий способ управления поведением персонажа. Добавление новых состояний возможно без модификации существующего кода.

2.4 Листинг кода

Интерфейс ICharacterState

```
namespace Task2
{
    internal interface ICharacterState
    {
        void HandleAction(Character character);
    }

}
```

Класс Character

```
namespace Task2
{
    internal class Character
    {
        private ICharacterState currentState;
        public string Name { get; set; }

        public Character(string name)
        {
            Name = name;
            currentState = new NormalState(); // Начальное состояние – обычное
        }

        public void SetState(ICharacterState state)
        {
            currentState = state;
        }

        public void PerformAction()
        {
            currentState.HandleAction(this);
        }
    }

}
```

Классы состояний

```
namespace Task2
{
    internal class DefenseState : ICharacterState
    {
        public void HandleAction(Character character)
        {
            Console.WriteLine($"{character.Name} поднимает щит и защищается!");
        }
    }
```

```
}


namespace Task2
{
    internal class AttackState : ICharacterState
    {
        public void HandleAction(Character character)
        {
            Console.WriteLine($"{character.Name} атакует врага мечом!");
        }
    }
}

namespace Task2
{
    internal class NormalState : ICharacterState
    {
        public void HandleAction(Character character)
        {
            Console.WriteLine($"{character.Name} стоит и осматривается.");
        }

    }
}
```

Класс Program

```
internal class Program
{
    static void Main(string[] args)
    {
        Game game = new Game();
        GameInvoker invoker = new GameInvoker();

        while (true)
        {
            Console.WriteLine("\nВыберите действие:");
            Console.WriteLine("1 - Сделать ход");
            Console.WriteLine("2 - Отменить последний ход");
            Console.WriteLine("3 - Показать историю ходов");
            Console.WriteLine("0 - Выйти");

            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    Console.Write("Введите описание хода: ");
                    string move = Console.ReadLine();
                    ICommand command = new MoveCommand(game, move);
                    invoker.ExecuteCommand(command);
                    break;

                case "2":
                    invoker.UndoLastCommand();
                    break;

                case "3":
                    game.ShowMoves();
                    break;

                case "0":
                    return;
```

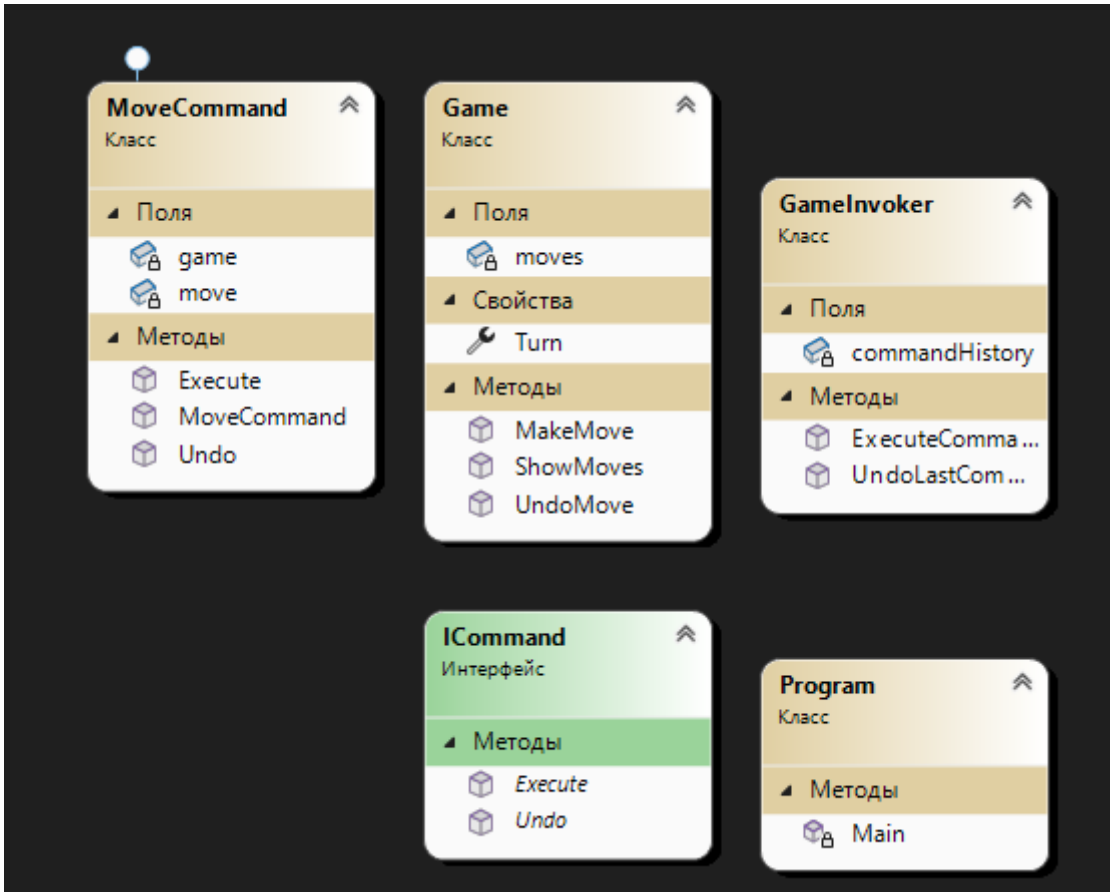
```
                default:
                    Console.WriteLine("Некорректный ввод. Попробуйте снова.");
                    break;
            }
        }
    }
}
```

3. Настольная игра (Паттерн «Команда»)

3.1 Описание предметной области

Настольная игра поддерживает команды для выполнения ходов, отмены предыдущего хода и отмены действий. Паттерн "Команда" позволяет инкапсулировать команды в объекты, что упрощает управление ими.

3.2 UML-диаграмма классов



3.3 Краткий вывод по использованию паттерна

Применение паттерна "Команда" упростило реализацию отмены и повторного выполнения ходов, а также обеспечило удобное управление действиями игроков.

3.4 Листинг кода

Интерфейс ICommand

```
namespace Task3
{
    internal interface ICommand
    {
        void Execute();
        void Undo();
    }

}
```

Класс Game

```
namespace Task3
{
    internal class Game
    {
        private List<string> moves = new List<string>();
        public int Turn { get; private set; } = 1;

        public void MakeMove(string move)
        {
            moves.Add($"Ход {Turn}: {move}");
            Console.WriteLine($"Выполнен ход {Turn}: {move}");
            Turn++;
        }

        public void UndoMove()
        {
            if (moves.Count > 0)
            {
                string lastMove = moves[^1];
                moves.RemoveAt(moves.Count - 1);
                Turn--;
                Console.WriteLine($"Отменён {lastMove}");
            }
            else
            {
                Console.WriteLine("Нет ходов для отмены.");
            }
        }

        public void ShowMoves()
        {
            Console.WriteLine("\nИстория ходов:");
            if (moves.Count == 0)
                Console.WriteLine("Пока нет ходов.");
            else
                moves.ForEach(Console.WriteLine);
        }
    }

}
```

Класс MoveCommand

```
namespace Task3
{
    internal class MoveCommand : ICommand
```

```
    {
        private Game game;
        private string move;

        public MoveCommand(Game game, string move)
        {
            this.game = game;
            this.move = move;
        }

        public void Execute()
        {
            game.MakeMove(move);
        }

        public void Undo()
        {
            game.UndoMove();
        }
    }

}
```

Класс GameInvoker

```
namespace Task3
{
    internal class GameInvoker
    {
        private Stack<ICommand> commandHistory = new Stack<ICommand>();

        public void ExecuteCommand(ICommand command)
        {
            command.Execute();
            commandHistory.Push(command);
        }

        public void UndoLastCommand()
        {
            if (commandHistory.Count > 0)
            {
                ICommand lastCommand = commandHistory.Pop();
                lastCommand.Undo();
            }
            else
            {
                Console.WriteLine("Нет команд для отмены.");
            }
        }
    }
}
```

Класс Program

```
  internal class Program
  {
      static void Main(string[] args)
      {
          Game game = new Game();
          GameInvoker invoker = new GameInvoker();

          while (true)
          {
              Console.WriteLine("\nВыберите действие:");
```

```
            Console.WriteLine("1 - Сделать ход");
            Console.WriteLine("2 - Отменить последний ход");
            Console.WriteLine("3 - Показать историю ходов");
            Console.WriteLine("0 - Выйти");

            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    Console.Write("Введите описание хода: ");
                    string move = Console.ReadLine();
                    ICommand command = new MoveCommand(game, move);
                    invoker.ExecuteCommand(command);
                    break;

                case "2":
                    invoker.UndoLastCommand();
                    break;

                case "3":
                    game.ShowMoves();
                    break;

                case "0":
                    return;

                default:
                    Console.WriteLine("Некорректный ввод. Попробуйте снова.");
                    break;
            }
        }
    }

}
```