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

Звіт

з лабораторної роботи № 5 з дисципліни
«Проектування алгоритмів»

„Пошук в умовах протидії, ігри з повною інформацією”

Виконав(ла) *ІТ-01 Бардін В. Д.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Камінська П. А.* _____
(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи - вивчити основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Ознайомитися з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією.

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм альфа-бета-відсікань.

Реалізувати три рівні складності (легкий, середній, складний) + 1балл.

Зробити узагальнений висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти

№	**Варіант**
1	Баше https://ru.wikipedia.org/wiki/Баше_(игра)

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

```
internal static class Program
{
    private static void Main()
    {
        new BasheProcessor()
            .Start(new GameConfiguration(1, 3, 15), BasheDifficulty.Hard);
    }
}

public interface IBasheProcessor
{
    void Start(GameConfiguration config, BasheDifficulty difficulty =
BasheDifficulty.Medium);
}

public class BasheProcessor : IBasheProcessor
{
    public void Start(GameConfiguration config, BasheDifficulty difficulty =
BasheDifficulty.Easy)
    {
        IDifficultyStrategy handler = difficulty switch
        {
            BasheDifficulty.Easy => new EasyDifficultyStrategy(),
            BasheDifficulty.Medium => new MediumDifficultyStrategy(),
            BasheDifficulty.Hard => new HardDifficultyStrategy(),
            _ => throw new NotImplementedException()
        };

        new GameProcessor().Execute(config, handler);
    }
}

public interface IDifficultyStrategy
{
    int GetItemsAmount(GameConfiguration config, int itemsLeft);
}

public class EasyDifficultyStrategy : IDifficultyStrategy
{
    public int GetItemsAmount(GameConfiguration config, int itemsLeft)
    {
        var maxAmount = config.StepMax > itemsLeft ? itemsLeft : config.StepMax;
        for (var i = maxAmount; i >= config.StepMin; i--)
        {
            if (config.StepMax % i == 0)
                return i;
        }
        return config.StepMin;
    }
}
```

```
public class MediumDifficultyStrategy : IDifficultyStrategy
{
    public int GetItemsAmount(GameConfiguration config, int itemsLeft)
    {
        return AlphaBeta(2, config.StepMin, config.StepMax, itemsLeft);
    }
    private int AlphaBeta(int depth, int alpha, int beta, int itemsLeft) {
        if (depth == 0) return GetAmount(alpha, beta, itemsLeft);
        var moves = Enumerable.Range(Math.Abs(alpha), Math.Abs(beta)).ToList();
        for(int i = 0; i < moves.Count; i++) {
            itemsLeft -= moves[i];
            var eval = -AlphaBeta(depth-1, -beta, -alpha, -itemsLeft);
            itemsLeft += moves[i];
            if(eval >= beta)
                return beta;
            if(eval > alpha) {
                alpha = eval;
                if (depth == 1) {
                    return moves[i];
                }
            }
        }
        return alpha;
    }
    private int GetAmount(int min, int max, int itemsLeft)
    {
        if (itemsLeft <= max && itemsLeft >= min)
        {
            return itemsLeft;
        }
        var maxAmount = max > itemsLeft ? itemsLeft : max;
        for (var i = maxAmount; i >= min; i--)
        {
            if (max % i == 0)
                return i;
        }
        return min;
    }
}

public class HardDifficultyStrategy : IDifficultyStrategy
{
    public int GetItemsAmount(GameConfiguration config, int itemsLeft)
    {
        var maxAmount = config.StepMax > itemsLeft ? itemsLeft : config.StepMax;
        for (var i = maxAmount; i >= config.StepMin; i--)
        {
            if ((itemsLeft - i) % (config.StepMax + 1) == 0)
                return i;
        }
        return config.StepMin;
    }
}
```

```
internal static class ConsoleHelper
{
    internal static int GetUserInput(GameConfiguration config, int itemsLeft)
    {
        Console.WriteLine("It's your step now.");
        do
        {
            Console.Write($"Please enter a number from {config.StepMin} to
{config.StepMax}: ");
            var isNumber = int.TryParse(Console.ReadLine(), out var userInput);
            if (!isNumber) continue;
            var numNotInRange = !(userInput >= config.StepMin && userInput <=
config.StepMax);
            if (numNotInRange)
            {
                Console.WriteLine($"Your number should be in range:
[{config.StepMin}; {config.StepMax}]");
                continue;
            }
            var userInputValid = itemsLeft >= userInput;
            if (userInputValid) return userInput;

            Console.WriteLine("Sorry, you can't take more items that left. " +
                              $"Left {itemsLeft} {(itemsLeft != 1 ? "items" :
"item")}.");
        } while (true);
    }
}
```

```
public class GameProcessor
{
    private static readonly Random Randomizer = new();
    private int _itemsLeft;
    private bool _aiStep;
    public void Execute(GameConfiguration config, IDifficultyStrategy strategy)
    {
        _itemsLeft = config.ItemsAmount;
        _aiStep = Randomizer.Next(0, 2) == 0;
        while (_itemsLeft > 0)
        {
            if (!_aiStep)
            {
                // user makes a step
                var userAmount = ConsoleHelper.GetUserInput(config, _itemsLeft);
                _itemsLeft -= userAmount;
                Console.WriteLine($"You take {userAmount} items. Items left:
{_itemsLeft}");
                if (_itemsLeft == 0 || _itemsLeft < config.StepMin)
                {
                    Console.WriteLine("Congrats! You win!");
                    return;
                }
            }
            var aiAmount = strategy.GetItemsAmount(config, _itemsLeft);
            _itemsLeft -= aiAmount;
            Console.WriteLine($"AI takes: {aiAmount}. Items left {_itemsLeft}");
            if (_itemsLeft == 0 || _itemsLeft < config.StepMin)
            {
                Console.WriteLine("AI win!");
                return;
            }
            _aiStep = false;
        }
    }
}

public enum BasheDifficulty
{
    Easy,
    Medium,
    Hard,
}

public class GameConfiguration
{
    public int StepMin { get; init; }
    public int StepMax { get; init; }
    public int ItemsAmount { get; init; }

    public GameConfiguration()
    {
        StepMin = 1;
        StepMax = 3;
        ItemsAmount = 15;
    }
    public GameConfiguration(int stepMin, int stepMax, int itemsAmount)
    {
        StepMin = stepMin;
        StepMax = stepMax;
        ItemsAmount = itemsAmount;
    }
}
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
AI takes: 3. Items left 12
It's your step now.
Please enter a number from 1 to 3: 2
You take 2 items. Items left: 10
AI takes: 2. Items left 8
It's your step now.
Please enter a number from 1 to 3: 3
You take 3 items. Items left: 5
AI takes: 1. Items left 4
It's your step now.
Please enter a number from 1 to 3: 3
You take 3 items. Items left: 1
AI takes: 1. Items left 0
AI win!
```

Рисунок 3.1 – Перемога комп’ютера, на високому рівні складності

```
AI takes: 3. Items left 12
It's your step now.
Please enter a number from 1 to 3: 3
You take 3 items. Items left: 9
AI takes: 3. Items left 6
It's your step now.
Please enter a number from 1 to 3: 2
You take 2 items. Items left: 4
AI takes: 3. Items left 1
It's your step now.
Please enter a number from 1 to 3: 1
You take 1 items. Items left: 0
Congrats! You win!
```

Рисунок 3.2 – Поразка комп’ютера, на середньому рівні складності

ВИСНОВОК

В рамках даної лабораторної роботи було розроблено математичну гру Баше, а також рівні складності для неї. Для складного рівня використано оптимальну стратегію для цієї гри: залишок завжди має бути кратним M + 1, де M — це максимальна кількість предметів, які можна взяти за 1 хід. Середній рівень складності — реалізовано за допомогою алгоритму «Альфа-Бета Відсікання». Якщо алгоритм не дає результату, то застосовується покращення простого рівня складності: якщо є можливість забрати усі предмети за 1 хід — комп’ютер це зробить. Для найлегшого рівня складності обрано примітивний алгоритм, коли на кожному ході комп’ютер буде намагатися забрати число предметів, що кратне M.