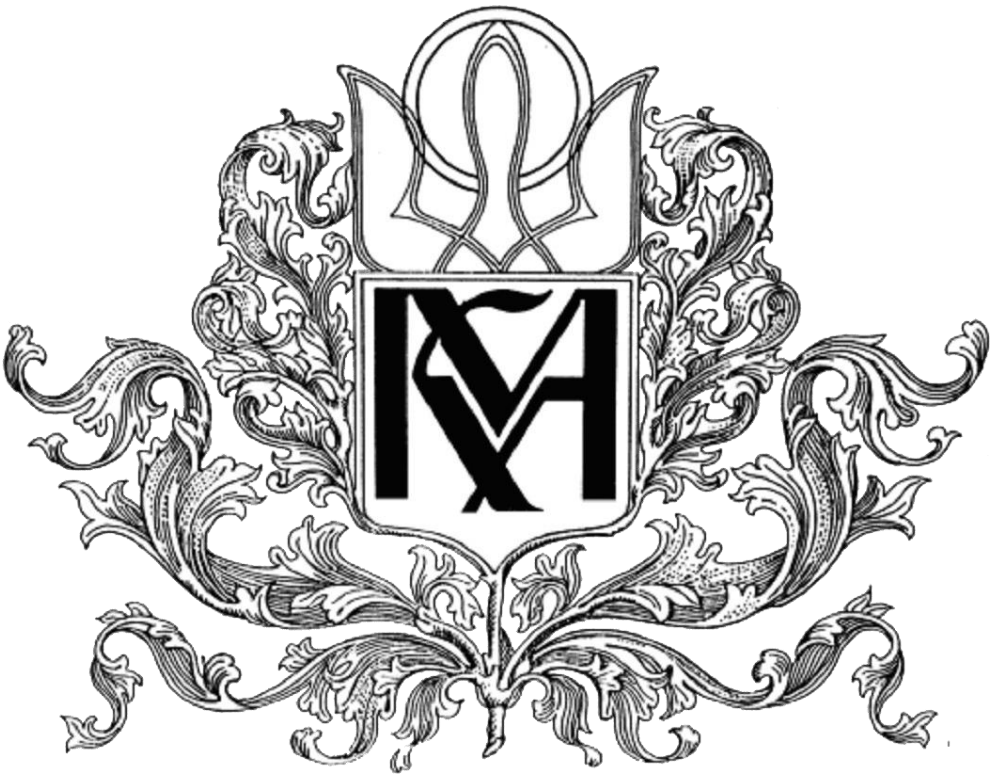
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики



# Створення інтерактивної системи побудови суб'єктивної шкали корисності на прикладі грошових доходів (Програмна реалізація)

**Текстова частина до курсової роботи**
**за спеціальністю „Прикладна математика"**

Керівник курсової роботи
Михалевич В.М.

_____
(підпис)
"____" _____ 2020 р.
Виконав студент Заморський І. В.
"____" _____ 2020 р.

**Київ 2020**

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра математики

ЗАТВЕРДЖУЮ

Зав.кафедри математики, проф., д.ф.-м.н.

_____Б. В. Олійник (підпис)

„____”_____2020 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на курсову роботу

студента Заморського І. В. факультету інформатики 4-го курсу

ТЕМА “ Створення інтерактивної системи побудови суб’єктивної шкали корисності на прикладі грошових доходів (Програмна реалізація)”

Зміст ТЧ до курсової роботи:

1) Вступ
2) Постановка задачі та існуючі рішення
3) Загальний розбір поняття корисності
4) Програмна реалізація системи побудови шкали корисності
5) Огляд та порівняння засобів розробки
6) Висновки
7) Список використаної літератури

10) Додатки

Дата видачі „___” _____ 2020 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

**Календарний план виконання курсової роботи**

**Тема: “ Створення інтерактивної системи побудови суб’єктивної шкали корисності на прикладі грошових доходів (Програмна реалізація)”**

**Календарний план виконання роботи:**

| № | Назва етапу курсового роботи | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на курсову роботу | 27.09.2019 | |
| 2 | Огляд літератури за темою роботи | 10.11.2019 | |
| 3 | Дослідження аналогів | 01.01.2020 | |
| 4 | Конфігурації та можливості застосування | 27.01.2020 | |
| 5 | Тестування застосування | 13.03.2020 | |
| 6 | Виконання аналізу отриманих результатів | 01.04.2020 | |
| 7 | Написання теоретичної частини | 10.04.2020 | |
| 8 | Створення слайдів для доповіді | 16.04.2020 | |

Студент Заморський Ігор Володимирович

“_____”_____ Михалевич В. М.

**Зміст**

**Вступ**

Теорія очікуваної корисності  являється потужним механізмом в сфері прийняття рішень на протязі багатьох десятиліть починаючи з середини XX століття. Вона часто використовується в фінансовій та економічній теорії для передбачень, також в управлінських дисциплінах для приписів і психології – для опису. Модель очікуваної корисності часто використовується в багатьох дослідженнях, та має багато математичних пояснень і модифікацій.

В цій роботі я хотів би зупинитись на програмній реалізації експериментальної системи для визначення суб'єктивної шкали корисності певного індивіда. Математична частина теорії корисності досить чітко описується в багатьох книгах, присвячених цьому, отже я приведу лише деякі деталі з них для загального розуміння усієї теми.

Найголовніша частина роботи – проробка правильного алгоритму для визначення суб'єктивних  поглядів індивіда на ризик та деякого його ставлення до грошових витрат або доходів.

Яке практичне використання такої системи?

- Визначення функції корисності в цілях аналізу індивіда при різних тестах, включаючи психологічні, або ж тестуванні на співбесідах.
- Визначення функції корисності в особистих цілях індивіда, для самоаналізу.

**Актуальність теми:** не зважаючи на те, що це досить добре вивчена та багато де використана теорія, різні впровадження її в сфери життя людей допоможуть краще розуміти, що їм потрібно.

**Мета і завдання дослідження:** Створити програму, здатну на графічне визначення суб'єктивної функції корисності індивіда, а також надання пояснення для розуміння результатів виконання.

**Цінність теоретичної та практичної частини:** програма дає можливість максимально просто зобразити функцію корисності окремого індивіда, проаналізувати їх і зрозуміти для себе щось нове.

## Розділ 1: Постановка задачі та існуючі рішення

### 1.1 Постановка задачі

1. Переглянути інформацію стосовно існуючих реалізацій схожих алгоритмів та різні дослідження в цій сфері.
2. Створити гнучкий та якісний алгоритм по опитуванню суб'єкта з ціллю отримання інформації про його суб'єктивні погляди на грошові доходи та витрати.
3. На основі готового алгоритму створити інтерактивний клієнт програми для визначення суб'єктивної функції корисності користувача.
4. По результату роботи програми доповнити алгоритм.
5. Порівняти готову програму з іншими доступними варіантами.
6. Зробити висновки що до результатів виконання програми в різних ситуаціях.

### 1.2 Математична постановка задачі

Математична реалізація полягає в імітації деякої гри з користувачем. За правилами гри, він отримає певну суму коштів ( $x_i, i \in [1, 24]$ ) з імовірністю ($p_i, i \in [1, 24]$ )( заданою для кожного кроку окремо), і суму ($y_i, i \in [1, 24]$ ) з імовірністю $(1 - p_i)$.

$x_i, y_i, p_i$ належать деякому імовірнісному розподілу $X_i$. Перед грою гравець має визначити таку суму $r_i$, $i \in [1, 24]$ , яку він згодний отримати замість участі у вказаній грі ( тобто, «продати» гру). Далі генерується випадкове значення $Y_i$. Якщо вибрана сума $r_i$ менша, чим випадково генероване значення $Y_i$ , гравець отримує ці кошти в розмірі $Y_i$, інакше грає у вказану вище гру з можливим виграшем, який належить до розподілу $X_i$.

### 1.3 Існуючі рішення

На даний момент існуючих програмних рішень для алгоритмів такого типу мені знайти не вдалось.

## Розділ 2: Загальний розбір поняття очікуваної корисності

### 2.1 Перші вчені та Санкт-Петербурзький парадокс

Перші формування поняття корисності були зроблені вченими Габріелем Крамером та Даніелем Бернуллі, які хотіли пояснити природу Санкт-Петербурзького парадоксу. Суть парадоксу в тому, що люди готові заплатити порівняно невелику суму в грі, математичне очікування виграшу в якій нескінченно велике. Суть гри досить проста – підкидається монета («правильна», де шанс появи обох сторін монети однаковий і дорівнює 1/2) доти , поки не випаде «реверс» (або ж «аверс», тобто одна із сторін). Виграш залежить від кількості підкидань монети. Якщо було здійснено n кидків, виграш складатиме $2^n$ грошових одиниць. Звідси видно, що очікуваний виграш в даній грі – нескінченний, адже $\sum_{n=1}^{\infty}(\frac{1}{2})^n\, 2^n = \infty$.

Так як більшість людей готові заплатити лише невелику суму грошей за нескінченно великий можливий виграш, Бернуллі припустив що люди максимізують не очікуваний виграш, а очікувану корисність. Він не ставив перед собою задачі підрахувати корисність, але як теорія все це виглядало схожим на правду. Доведення того, що принцип максимізації середньої корисності є критерієм раціональності рішень, які приймаються, запропонували лише Джон фон Нейман та Оскар Моргенштерн. Вони запропонували 5 аксіом, далі модифікованих Джейкобом Маршаком, які забезпечують існування корисності в цілому для будь яких результатів будь яких процесів.

### 2.2 Різні підходи до імовірності

Досить важливим при визначенні корисності є підхід до поняття імовірності. Є декілька основних концепцій, які в цілому відрізняються простотою та поглядами вченого, який її запропонував.

Найпростіша із концепцій, яка належить П’єру Лапласу , полягає в тому, що імовірність це відношення кількості сприятливих результатів деякої елементарної події до всіх результатів.

Концепція Бернуллі складена дещо по іншому. Він визначив ймовірність як особисте ставлення людини до вірогідності одного із результатів деякої події. Також він вважав, що мистецтво «вгадувати» полягає в уточненні людиною деяких невідомих ймовірностей, досліджуючи об’єктивні частоти.

Школа Джона Мейнарда Кейнса і Гарольда Джеффріса стверджувала, що множина емпіричних даних знаходиться в відношенні до істинності деякої гіпотези. Імовірність визначає силу цього зв’язку.

### 2.3 Підходи до використання моделей очікуваної корисності

Всі підходи до використання теорії очікуваної корисності можна розділити на 4 основні групи: Описовий, Передбачуваний, Пояснювальний та Розпорядчий.

Описовий підхід використовується для моделювання процесів прийняття рішень, пов’язаних з ризиком.

Передбачуваний або позитивістський підхід в основному використовується в економічній і фінансовій сфері. Найголовніше в моделі в такому випадку – точність передбачень, які вона робить на основі існуючої інформації.

Пояснювальний підхід повинен впорядкувати данні та виявляти оптимальну модель людської поведінки.

Розпорядчий підхід націлений покращити сферу прийняття рішень людини, так як в більшості поведінка людини є не оптимальною, а суб’єктивною. Мається на увазі, що в такому випадку модель дає поради людині стосовно складних ситуацій вибору.

### 2.4 Експеримент ДеҐрута, Маршака та Беккера

Для дослідження суб’єктивності вибору людини та її здатність до навчання або ж адаптації був проведений експеримент, в основі якого лежить послідовні пропозиції зіграти в деяку гру. Принцип, використаний цими вченими лежить в основі моєї програми для визначення суб’єктивної шкали корисності людини, отже буде розгорнуто описаний далі в курсовій роботі.

## Розділ 3: Програмна реалізація моделі системи побудови шкали корисності

### 3.1 Опис алгоритму

Алгоритм складається з 24 кроків. На ножному з кроків користувачу пропонується наступна гра. Він отримає певну суму коштів ( $x_i, i \in [1, 24]$ ) з імовірністю ($p_i, i \in [1, 24]$ )( заданою для кожного кроку окремо), і суму ($y_i, i \in [1, 24]$ ) з імовірністю $(1 - p_i)$.

$x_i, y_i, p_i$ належать деякому імовірнісному розподілу $X_i$. Перед грою гравець має визначити таку суму $r_i$, $i \in [1, 24]$ , яку він згодний отримати замість участі у вказаній грі ( тобто, «продати» гру). Далі генерується випадкове значення $Y_i$. Якщо вибрана сума $r_i$ менша, чим випадково генероване значення $Y_i$ , гравець отримує ці кошти в розмірі $Y_i$, інакше грає у вказану вище гру з можливим виграшем, який належить до розподілу $X_i$.

За допомогою визначення сум $r_i$, які користувач готовий отримати замість участі в грі, та знаючи наперед корисність вказаної суми можна досить просто зобразити на графіку відхилення від норми. Норма в даному контексті є такою сумою $r_i$, що $U(r_i) = E[U(X_i)]$. Відхилення від норми і зображає суб'єктивність вибору людини в сенсі грошових доходів, так як максимальна середня корисність це є в деякому сенсі об'єктивність.

На вхід роботи алгоритму подаються межі гри - $a$ та $b$. Відповідно до цього, всі значення $x_i, y_i$ (можливих виграшів з розподілу $X_i$)будуть менші, ніж $a$, та більші, ніж $b$. Також в цьому діапазоні на всіх кроках роботи алгоритму визначається випадкове значення $Y_i$. В такому випадку очевидно, що корисність $U(a) = 0, U(\mathrm{b}) = 1$. Відштовхуючись від цього, на кожному кроці алгоритму визначається корисність для вибраної гравцем суми грошей $r_i$. Далі буде описаний кожний крок роботи алгоритму в вигляді таблиці. Вибір випадкового значення $Y_i$ не змінює діапазону впродовж всієї роботи алгоритму.

| № | x | y | p | U(r) | № | x | y | p | U(r) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | a | b | 1/2 | 1/2 | 13 | $r_4$ | b | 1/2 | 7/8 |
| 2 | a | b | 3/4 | 1/4 | 14 | $r_7$ | $r_8$ | 3/4 | 43/64 |
| 3 | a | b | 1/4 | 3/4 | 15 | $r_5$ | $r_4$ | 1/2 | 1/2 |
| 4 | $r_1$ | b | 1/2 | 3/4 | 16 | $r_9$ | b | 1/2 | 13/16 |
| 5 | a | $r_1$ | 1/2 | 1/4 | 17 | $r_6$ | $r_7$ | 3/4 | 7/16 |
| 6 | $r_2$ | $r_3$ | 3/4 | 3/8 | 18 | $r_9$ | $r_{13}$ | 1/2 | 3/4 |
| 7 | $r_2$ | $r_3$ | 1/4 | 5/8 | 19 | $r_{11}$ | $r_8$ | 3/4 | 5/8 |
| 8 | b | $r_3$ | 1/4 | 13/16 | 20 | a | $r_{13}$ | 1/2 | 7/16 |
| 9 | $r_5$ | b | 1/2 | 5/8 | 21 | $r_1$ | $r_4$ | 1/2 | 5/8 |
| 10 | $r_5$ | $r_1$ | 1/2 | 5/8 | 22 | $r_{11}$ | $r_8$ | 1/4 | 3/4 |
| 11 | a | $r_3$ | 1/4 | 9/16 | 23 | A | $r_4$ | 1/2 | 3/8 |
| 12 | $r_2$ | b | 3/4 | 7/16 | 24 | $r_{11}$ | b | 3/4 | 43/64 |

Значення в таблиці формулюються по простим формулам корисності, відштовхуючись від корисності меж та попередніх значень. В даному випадку 24 – достатня точність для правильного відображення вибору користувача.

Наступний і останній крок – побудова функції корисності методом інтерполяції по вказаним точкам.

Алгоритм повинен пройти хоча б дві-три ітерації для більш чіткого результату.

Також, в таблиці видно, що є значення, для яких корисність – однакова. Звідси слідує, що при правильному виборі та максимізації середньої корисності значення в точках з однаковою корисністю повинні співпадати. Це перевіряється вкінці виконання і також може бути зображено на графіку результату.

### 3.2 Архітектура програми

Програма будується навколо головного класу MainWindow, який відповідає за відображення інформації користувачу, більшість логіки роботи міститься саме в ньому. Детальний опис полів та методів буде наданий в наступному пункті розділу 3.

Програма працює, базуючись на понятті «етапу виконання». Для кожного етапу є свій опис та налаштування, які задаються в файлі формату Json та десеріалізуються через допоміжний клас Settings.

Ключовий алгоритм виконання починається з отримання даних в форматі Json та їх перетворення в екземпляри класу Stage, кожен з яких описує один із кроків виконання. Згідно вказаного порядку програма отримує інформацію, яке вікно потрібно зобразити на екрані користувачу.

Всього існує 5 етапів виконання:

1. Prepare (підготовка) - являється першим етапом виконання та не несе якогось логічного значення. Також його можна описати як стартовий етап.
2. Settings (налаштування) - це етап виконання, на якому користувач згідно вказаних інструкцій вибирає межі для подальшої гри.
3. XDefine (визначення змінних) - головний етап програми, на якому користувач в різних межах виконує вказані інструкцією умови.
4. Results (результати) - вивід результатів виконання програми для користувача.
5. Exit (вихід) - завершення роботи програми.

### 3.3 Опис головних класів

Клас **Stage**

```
public class Stage
    {
        public string MainText;
        public UtilityStage UtilityStage;
        public Stage Next;

    }
    public enum UtilityStage
    {
        Preparing,
```

```
        Settings,
        Xdefine,
        Result,
        Exit
             }
```

## Поля класу

- MainText - зберігає текст інструкцію для кожного із етапів виконання.
- UtilityStage - містить інформацію про поточний етап виконання програми.

Клас **Settings**

```
public static class Settings
    {
        private const string FilePath = @"D:\Рабочий стол\Курсова\Kursova\Utils\Settings.json";

        public static Stage Get()
        {
            AllSteps info = new AllSteps();
            try
            {
                using (StreamReader sr = new StreamReader(FilePath, Encoding.GetEncoding(1251)))
                {
                    string text = sr.ReadToEnd();
                    Console.WriteLine(text);

                    info = JsonConvert.DeserializeObject<AllSteps>(text);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Stage current = new Stage();
            Stage first = current;
            foreach (var step in info.steps)
            {
                current.MainText = step.text;
                current.UtilityStage = ParseStage(step.programStage);
                current.Next = new Stage();
                current = current.Next;
            }
            return first;
        }

        public static UtilityStage ParseStage(string stage)
        {
            switch (stage)
            {
                case "prepare":
```

```
                return UtilityStage.Preparing;
            case "settings":
                return UtilityStage.Settings;
            case "xdefine":
                return UtilityStage.Xdefine;
            case "result":
                return UtilityStage.Result;
            case "exit":
                return UtilityStage.Exit;
            default:
                return UtilityStage.Preparing;
        }
    }


    public class AllSteps
    {
        public Step[] steps { get; set; }
    }

    public class Step
    {
        public string text { get; set; }
        public string programStage { get; set; }
    }
}
```

### Поля класу

- FilePath – константа шляху до файлу з налаштуваннями.

Основна логіка десеріалізації міститься в статичному методі Get(), де отримується текст файлу, та згідно з класами AllSteps та Step перетворює отриману інформацію в екземпляри класу Stage. Усі отримані екземпляри передаються посиланням на перший із них, так як кожен містить в собі посилання на наступний. Так реалізується архітектура по типу одностороннього LinkedList.

### Клас MainWindow

```
public partial class MainWindow : Window
  {
    public Stage Current;

    public int Max = 1000;
    public int Min = 0;
    public const int Accuracy = 24;

    public Dictionary<float, int> XValues;
    public Dictionary<int, int> AuxValues;
```

```
public Dictionary<float, int> XMistake;
public int XdefineStage;
public float CurrentUtility;
public IEnumerator<Borders> ProgBorders;

public SeriesCollection Data { get; set; }
public float[] Utilities { get; set; }
public Func<int, string> YFormatter { get; set; }

public MainWindow()
{
  InitializeComponent();
  Start();
}

public void Start()
{
  HideAllWindows();
  Current = Utils.Settings.Get();
  MainText.Text = Current.MainText;
}

private void NextStage(object sender, RoutedEventArgs e)
{
  HideAllWindows();
  Current = Current.Next;
  MainText.Text = Current.MainText;
  switch (Current.UtilityStage)
  {
    case UtilityStage.Preparing:
      break;
    case UtilityStage.Settings:
      Settings.Visibility = Visibility.Visible;
      NextButton.Visibility = Visibility.Hidden;
      break;
    case UtilityStage.Xdefine:
      ModifySlider();
      break;
    case UtilityStage.Result:
      ShowResults();
      break;
    case UtilityStage.Exit:
      Close();
      break;
  }
}


public void SetValues(object sender, RoutedEventArgs e)
{
  int.TryParse(MinValue.Text, out Min);
  int.TryParse(MaxValue.Text, out Max);
```

```
  if (Max == 0)
  {
    MessageBox.Show("Спершу введіть данні!");
    return;
  }
  XdefineStage = 1;
  XValues = new Dictionary<float, int>();
  AuxValues = new Dictionary<int, int>();
  XMistake = new Dictionary<float, int>();
  ProgBorders = new BordersProvider(Accuracy, Max, Min).NextBorders(AuxValues);
  XValues[1] = Max;
  XValues[0] = Min;
  XMistake[0] = 0;
  XMistake[1] = 0;
  ProgBorders.MoveNext();
  NextStage(sender, e);
}

public void SetXValue(object sender, RoutedEventArgs e)
{
  var answer = Round((int)ExactSlider.Value);
  if (XValues.Keys.Contains(CurrentUtility))
  {
    var mistake = Math.Abs(answer - XValues[CurrentUtility]);
    XMistake[CurrentUtility] = mistake;
    XValues[CurrentUtility] = Math.Max(answer, XValues[CurrentUtility]);
  }
  else
    XValues[CurrentUtility] = answer;
  AuxValues[XdefineStage] = answer;
  XdefineStage++;
  ProgBorders.MoveNext();
  if (XdefineStage > Accuracy)
    NextStage(sender, e);
  else
    ModifySlider();
}

public void ModifySlider()
{
  Slider.Visibility = Visibility.Visible;
  if (ProgBorders.Current.Min < ProgBorders.Current.Max)
  {
    ExactSlider.Minimum = ProgBorders.Current.Min;
    ExactSlider.Value = ProgBorders.Current.Min;
    MinForSlider.Content = ProgBorders.Current.Min;
    ExactSlider.Maximum = ProgBorders.Current.Max;
    MaxForSlider.Content = ProgBorders.Current.Max;
    MinProb.Content = ProgBorders.Current.Probability;
    MaxProb.Content = 1 - ProgBorders.Current.Probability;
  }
  else
```

```
  {
    ExactSlider.Minimum = ProgBorders.Current.Max;
    ExactSlider.Value = ProgBorders.Current.Max;
    MinForSlider.Content = ProgBorders.Current.Max;
    ExactSlider.Maximum = ProgBorders.Current.Min;
    MaxForSlider.Content = ProgBorders.Current.Min;
    MinProb.Content = 1 - ProgBorders.Current.Probability;
    MaxProb.Content = ProgBorders.Current.Probability;
  }
  CurrentUtility = ProgBorders.Current.Utility;
  SliderValue.Content = 0;

}

public void SliderValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
  SliderValue.Content = Round((int)ExactSlider.Value);
}

public void ShowResults()
{
  HideAllWindows();
  Results.Visibility = Visibility.Visible;
  List<ObservablePoint> points = new List<ObservablePoint>();
  foreach (var key in XValues.Keys)
    points.Add(new ObservablePoint(XValues[key], key));
  points = points.OrderBy(x => x.Y).ToList();
  List<ObservablePoint> mistakes = new List<ObservablePoint>();
  foreach (var key in XMistake.Keys)
    mistakes.Add(new ObservablePoint(XValues[key] - XMistake[key], key));
  mistakes = mistakes.OrderBy(x => x.Y).ToList();
  Data = new SeriesCollection
        {
            new LineSeries
            {
                Title="Витрати",
                Values = new ChartValues<ObservablePoint>(points)
            },
            new LineSeries
            {
                Title="Похибки",
                Values = new ChartValues<ObservablePoint>(mistakes)
            }
        };
  Utilities = XValues.Keys.ToArray();
  YFormatter = value => value.ToString();
  DataContext = this;
}

public void HideAllWindows()
{
  Settings.Visibility = Visibility.Hidden;
```

```
      Slider.Visibility = Visibility.Hidden;
      Results.Visibility = Visibility.Hidden;
      SliderValue.Content = 0;
    }

    public int Round(int value)
    {
      double aux = value / 10;
      return (int)(Math.Round(aux) * 10);
    }
}
```

**Поля класу**

- Current – поточний етап виконання програми.
- Max – верхня межа тестування.
- Min – нижня межа тестування.
- XValues – словник збереження даних, введених користувачем, де ключ – корисність для деякої суми грошей.
- AuxValues – словник збереження даних, введених користувачем, де ключ – крок виконання програми, на якому були ці дані отримані.
- XMistake – словник збереження похибок, тобто різниці в значеннях, вибраних користувачем, для кроків з однаковою корисністю.
- XdefineStage – поточний крок виконання етапу програми XDefine.
- CurrentUtility – корисність на даному кроці виконання.
- ProgBorders – енумератор для отримання меж гри на кожному з кроців алгоритму.
- Data – екземпляр класу SeriesCollection для відображення отриманих даних на графіку.
- Utilities – масив для відображення осі корисності на графіку.
- YFormatter – перетворювач значень типу int в значення типу float для відображення осі грошових доходів на графіку.

Програма бере початок в конструкторі вказаного класу, де викликає методи InitializeComponent та Start. В методі Start отримується інформація про етапи виконання програми та записується в поле Current.

Метод NextStage переводить програму на наступний етап виконання , перезаписуючи поле Current.

Метод SetValues відповідає за встановлення налаштувань та ініціалізацію більшості полів виходячи з вибраних меж гри. Також створює екземпляр класу BordersProvider для отримання потрібних меж на кожному кроці виконання програми. Він являється одним із методів, де викликається метод NextStage.

Метод SetXValue служить для запису поточної відповіді користувача та переходу на наступний крок алгоритму або ж наступний етап програми.

Метод ModifySlider змінює данні на слайдері для взаємодії користувачем.

Метод ShowResults виводить результати виконання програми у виді графіку.

**Клас BordersProvider**

```
public class BordersProvider
    {

      private int MaxValue;
      private int MinValue;

      public BordersProvider(int maxValue, int minValue)
      {

        MaxValue = maxValue;
        MinValue = minValue;
      }

      public IEnumerator<Borders> NextBorders(Dictionary<int, int> values)
      {

        var borders = GetBorders(values);
        foreach (var border in borders) yield return border;
      }

      private IEnumerable<Borders> GetBorders(Dictionary<int, int> values)
      {
        yield return new Borders { Min = MinValue, Max = MaxValue, Probability = 0.5f, Utility = 0.5f };
        yield return new Borders { Min = MinValue, Max = MaxValue, Probability = 0.75f, Utility = 0.25f };
        yield return new Borders { Min = MinValue, Max = MaxValue, Probability = 0.25f, Utility = 0.75f };
        yield return new Borders { Min = values[1], Max = MaxValue, Probability = 0.5f, Utility = 0.75f };
        yield return new Borders { Min = MinValue, Max = values[1], Probability = 0.5f, Utility = 0.25f };
```

```
        yield return new Borders { Min = values[2], Max = values[3], Probability = 0.75f, Utility = 0.375f
};
        yield return new Borders { Min = values[2], Max = values[3], Probability = 0.25f, Utility = 0.625f
};
        yield return new Borders { Min = MaxValue, Max = values[3], Probability = 0.25f, Utility = 0.8125f
};
        yield return new Borders { Min = values[5], Max = MaxValue, Probability = 0.5f, Utility = 0.625f
};
        yield return new Borders { Min = values[5], Max = values[1], Probability = 0.5f, Utility = 0.375f
};
        yield return new Borders { Min = MinValue, Max = values[3], Probability = 0.25f, Utility = 0.5625f
};
        yield return new Borders { Min = values[2], Max = MaxValue, Probability = 0.75f, Utility = 0.4375f
};
        yield return new Borders { Min = values[4], Max = MaxValue, Probability = 0.5f, Utility = 0.875f
};
        yield return new Borders { Min = values[7], Max = values[8], Probability = 0.75f, Utility =
0.671875f };
        yield return new Borders { Min = values[5], Max = values[4], Probability = 0.5f, Utility = 0.5f };
        yield return new Borders { Min = values[9], Max = MaxValue, Probability = 0.5f, Utility = 0.8125f
};
        yield return new Borders { Min = values[6], Max = values[7], Probability = 0.75f, Utility =
0.4375f };
        yield return new Borders { Min = values[9], Max = values[13], Probability = 0.5f, Utility = 0.75f
};
        yield return new Borders { Min = values[11], Max = values[8], Probability = 0.75f, Utility =
0.625f };
        yield return new Borders { Min = MinValue, Max = values[13], Probability = 0.5f, Utility = 0.4375f
};
        yield return new Borders { Min = values[1], Max = values[4], Probability = 0.5f, Utility = 0.625f
};
        yield return new Borders { Min = values[11], Max = values[8], Probability = 0.25f, Utility = 0.75f
};
        yield return new Borders { Min = MinValue, Max = values[4], Probability = 0.5f, Utility = 0.375f
};
        yield return new Borders { Min = values[11], Max = MaxValue, Probability = 0.75f, Utility =
0.671875f };

      }
    }


    public struct Borders
    {
      public int Max;
      public int Min;
      public float Probability;
      public float Utility;
    }
}
```

**Поля класу**

- MaxValue – верхня межа гри
- MinValue – нижня межа гри

Метод NextBorders повертає енумератор для меж гри. Це зроблено для зручності отримання даних на кожному кроці алгоритму за допомогою використання методу інтерфейсу IEnumerator MoveNext.

Як структура для передачі даних використовується структура Borders.

**Поля Borders**

- Max – поточний максимум для гри.
- Min – поточний мінімум для гри.
- Probability – в контексті гри імовірність отримання грошової суми, вказаної в полі Min.
- Utility – корисність на поточному кроці виконання.

### 3.4 Порівняння з аналогами

Нажаль, програм для порівняння з моєю я знайти не зміг, але без порівнянь можу сказати, що отримана архітектурна структура є досить гнучкою та має можливості до розширення. За допомогою зміни методу NextBorders класу BordersProvider можна досить легко змінювати логіку розрахунку меж гри та їх ймовірностей. Також завдяки поділу програми на етапи, їх кількість, опис та суть можна змінювати не сильно втручаючись до всієї програми.

Також завдяки досить простому та зрозумілому інтерфейсу програми вона є зрозумілою для користувача. В будь-якому випадку, якщо розглядати, наприклад, не досить зрозумілі пояснення для етапів виконання людина, яка не уміє програмувати, може змінити їх в файлі налаштувань.

## Розділ 4:  Огляд та порівняння засобів розробки

### 4.1 Інструменти розробки C#

Мова програмування C# була вибрана мною через зручність розробки простих клієнтських програм та гнучкість самої мови, можливість досить просто реалізувати складні архітектури та їх подальшу стабільну роботу. Також об’єктно-орієнтований стиль мови досить добре підходить для виконання поставленої задачі.

### 4.2 Інструмент розробки WPF

WPF, як на мене, являється прекрасним вибором в контексті програми для Windows. Він має досить простий набір базових елементів для взаємодії з користувачем, та простий алгоритм написання логіки для роботи програми. Також я зупинив вибір на ньому через його безпосередній зв'язок з мовою програмування C#.

**Висновок**

У висновку можна сказати, що поставлене завдання, а саме програмна реалізація алгоритму побудови суб'єктивної функції корисності користувача було виконане та, як результат, отримана робоча версія програми для визначення та зображення шкали корисності користувача. Найбільше часу зайняв алгоритм побудови та його програмна реалізація, проробка класів та тестування програми. Поняття дослідження шкали корисності користувача є більш об'ємним, ніж зазначено в даній роботі, але як один із варіантів реалізації ця програма є хорошим результатом роботи.

**Список використаної літератури**


1. ДеГрут М. «Оптимальні статистичні рішення» .
   Eng. DeGroot M. «Optimal statistical decisions» ст. 92-124
2. Шумейкер П. «Моделі очікуваної корисності. Різновиди, підходи, результати та межі можливостей ».
   Eng. Schoemaker P. «The Expected Utility Model: Its Variants, Purposes, Evidence and Limitations».
3. Фрідмен М. та Севідж Л. Дж. «Аналіз корисності при виборі серед альтернатив, в яких можливий ризик» , «Теорія поведінки та попиту споживачів. Віхи економічної теорії».
   Eng. M. Friedman and L. J. Savage «The utility analysis of choices involving risk».
4. Беккер Г., ДеГроот М., Маршак Я. – «Вимірювання корисності послідовним методом однієї відповіді».
   Eng. Becker G. , DeGroot M, Marschak J. «Measuring utility by a single-response sequential method».