Specyfikacja Aplikacji Szyfru Poligjusza

Autor: 14546

1. Wprowadzenie

   Aplikacja szyfru Poligjusza to wszechstronne narzędzie zaprojektowane do szyfrowania i deszyfrowania tekstu przy użyciu algorytmu szyfru Poligjusza. Aplikacja ta umożliwia użytkownikom wprowadzanie tekstu, wybieranie klucza szyfrowania lub deszyfrowania oraz wykonywanie odpowiednich operacji na tekście.

2. Funkcje Aplikacji

   2.1 Szyfrowanie

   - Użytkownicy mogą wprowadzić tekst do zaszyfrowania w polu "Input".
   - Po wybraniu klucza i kliknięciu przycisku "Run Encryption", tekst zostaje zaszyfrowany przy użyciu algorytmu szyfru Poligjusza.
   - Zaszyfrowany tekst jest następnie wyświetlany w polu "Output”

   2.2 Deszyfrowanie

   - Użytkownicy mogą wprowadzić zaszyfrowany tekst w polu " Input " w celu deszyfrowania.
   - Po wybraniu klucza i kliknięciu przycisku "Run decryption", zaszyfrowany tekst jest deszyfrowany i wyświetlany w polu " Output ".

   2.3 Zmiana Trybu

   - Użytkownicy mogą płynnie przełączać się między trybami szyfrowania i deszyfrowania, przełączając przyciski typu " Run decryption " i " Run Encryption ".
   - Zmiana trybu dynamicznie aktualizuje etykiety pól tekstowych i przycisków, odzwierciedlając wybraną operację.

   2.4 Obsługa Błędów

   - Aplikacja z łatwością radzi sobie z sytuacjami, w których użytkownicy nie podają tekstu zawierającego poprawne polskie litery podczas szyfrowania. W takich przypadkach te znaki są pomijane podczas szyfrowania.
   - Wielkość liter nie ma znaczenia; wszystkie znaki są konwertowane na małe litery dla spójności.
   - Aplikacja skutecznie radzi sobie z przypadkami, gdy użytkownicy nie podają tekstu lub klucza przed próbą szyfrowania lub deszyfrowania. W takich sytuacjach aplikacja wyświetla komunikat o błędzie i nie kontynuuje procesu.

   2.5 Dodatkowe Kompikacje w Szyfrowaniu i Deszyfrowaniu

- Aplikacja wprowadza dodatkową złożoność zarówno w procesie szyfrowania, jak i deszyfrowania, za pomocą metod MoreComplicatedEN i MoreComplicatedDE.
- MoreComplicatedEN: Ta statyczna metoda wprowadza dodatkowe kroki szyfrowania poprzez dostosowanie wartości pewnych znaków w zaszyfrowanym wyniku.
- MoreComplicatedDE: Podobnie ta statyczna metoda jest stosowana podczas deszyfrowania, aby odwrócić dodatkowe kroki zastosowane podczas szyfrowania, zapewniając dokładne wyniki deszyfrowania.

3. Interfejs Użytkownika

Graficzny interfejs użytkownika (GUI) aplikacji jest starannie zaprojektowany, zawierając poniższe kluczowe elementy:

- Pole tekstowe "Input": Przeznaczone do wprowadzania tekstu do zaszyfrowania lub deszyfrowania.
- Pole tekstowe "Output": Wyświetla wynik procesu szyfrowania lub deszyfrowania.
- Przyciski " Run Encryption " lub " Run decryption ": Aktywują proces szyfrowania lub deszyfrowania w zależności od wybranej opcji.

4. Kod aplikacji

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Text;
using static Polybius_cipher.Form1;
using static System.Net.Mime.MediaTypeNames;

namespace Polybius_cipher
{
    // The main form class
    public partial class Form1 : Form
    {
        // TableLayoutPanel for the Polybius cipher grid
        TableLayoutPanel Tabel = new TableLayoutPanel();

        // Class representing elements in the Polybius cipher grid
        public class PolyList
        {
            public char id { get; set; }
            public int XAx { get; set; }
            public int YAx { get; set; }
            public int BigID;

            public PolyList(char Id, int xAx, int yAx)
            {
                id = Id;
                XAx = xAx;
```

```
        YAx = yAx;
        BigID = XAx * 10 + YAx;
    }
}

// Dictionary to check if a letter is already in the Polybius grid
public Dictionary<char, int> checkletter = new Dictionary<char, int>();

// List to store Polybius grid elements
public List<PolyList> PolybiusList = new List<PolyList>();

// Flag to indicate errors in the Polybius grid
public bool ErrorList = false;

// Constructor for the main form
public Form1()
{
    InitializeComponent();
    StworzSiatke(5, 7); // Create the Polybius cipher grid
}

// Method to create the Polybius cipher grid
private void StworzSiatke(int liczbaWierszy, int liczbaKolumn)
{
    // Configure the TableLayoutPanel
    Tabel.Dock = DockStyle.Fill;
    Tabel.CellBorderStyle = TableLayoutPanelCellBorderStyle.Single;

    // Create rows and columns in the grid
    for (int i = 0; i < liczbaWierszy; i++)
    {
        Tabel.RowStyles.Add(new RowStyle(SizeType.AutoSize));

        for (int j = 0; j < liczbaKolumn; j++)
        {
            Tabel.ColumnStyles.Add(new ColumnStyle(SizeType.AutoSize));
            TextBox poleTekstowe = new TextBox();

            poleTekstowe.Text = "-";
            poleTekstowe.Width = 20;

            Tabel.Controls.Add(poleTekstowe, j, i);
        }
    }

    panel1.Controls.Add(Tabel); // Add the grid to the form
}

// Method to read and validate the Polybius grid
private void Czytaj()
{
    PolybiusList.Clear();

    for (int i = 0; i <= 6; i++)
    {
        for (int j = 0; j <= 4; j++)
        {
            TextBox? pt = Tabel.GetControlFromPosition(i, j) as TextBox;

            string wartosc = pt.Text;
            char finalwartosc = wartosc[0];

            // Check if the letter is already in the grid
```

```
                    if (checkletter.ContainsKey((char)wartosc[0]))
                    {
                        ErrorList = false;
                    }
                    else
                        checkletter.TryAdd(finalwartosc, 1);

                    PolybiusList.Add(new PolyList((char)wartosc[0], j + 1, i +
1));
                }
            }

            // Check if the grid contains the correct number of unique characters
            if (checkletter.Count != 35)
            {
                ErrorList = false;
                MessageBox.Show("character array contains errors", "Alert",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                return;
            }
        }

        // Method to encrypt the input using the Polybius cipher
        private void encrypt()
        {
            Czytaj();

            if (ErrorList == false)
            {
                return;
            }

            string InputTxT = InputBox.Text.ToLower();
            InputTxT = CheckStringEn(InputTxT);

            List<char> Output = new List<char>();

            foreach (char letter in InputTxT)
            {
                PolyList znalezionaKlasa = PolybiusList.FirstOrDefault(klasa =>
klasa.id == letter);

                Output.Add((znalezionaKlasa.XAx).ToString()[0]);
                Output.Add((znalezionaKlasa.YAx).ToString()[0]);
            }

            string FinalOutput = string.Join("", Output);

            FinalOutput = MoreComplicatedEN(FinalOutput);

            OutputBox.Text = FinalOutput;
        }

        // Event handler for the encryption button click
        private void button1_Click(object sender, EventArgs e)
        {
            encrypt();
        }

        // Method to decrypt the input using the Polybius cipher
        private void decrypt()
        {
            Czytaj();
```

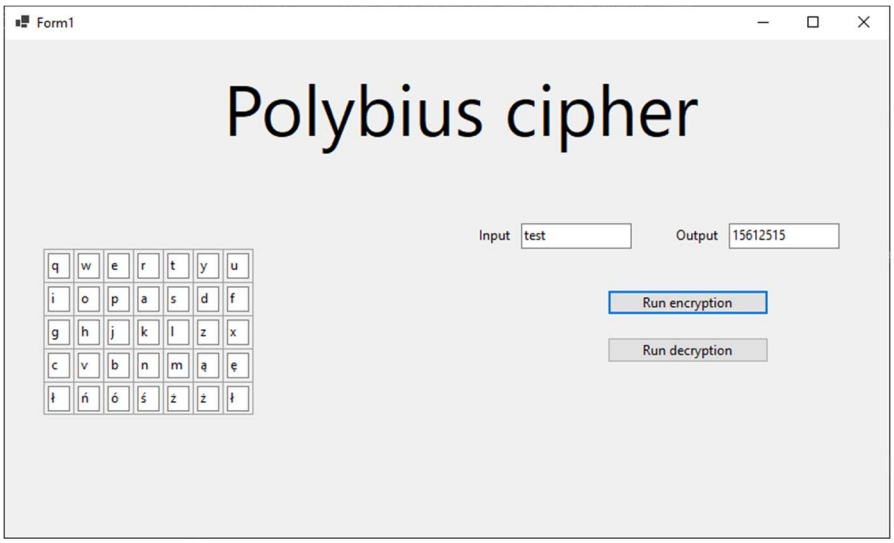
```
            if (ErrorList == false)
            {
                return;
            }

            string TxtToDecrypt = InputBox.Text.ToLower();
            TxtToDecrypt = CheckStringDe(TxtToDecrypt);
            TxtToDecrypt = MoreComplicatedDE(TxtToDecrypt);

            char[] charInput = TxtToDecrypt.ToCharArray();

            int length = charInput.Length;

            int[,] TxtCord = new int[length / 2, 2];

            for (int i = 0; i < length; i += 2)
            {
                TxtCord[i / 2, 0] = charInput[i] - '0';
                TxtCord[i / 2, 1] = charInput[i + 1] - '0';
            }

            char[] output = new char[length / 2];

            for (int i = 0; i < output.Length; i++)
            {
                output[i] = FindIdByBigID(PolybiusList, TxtCord[i, 0] * 10 +
TxtCord[i, 1]);
            }
            OutputBox.Text = new string(output);
        }

        // Method to find Polybius grid element by BigID
        public char FindIdByBigID(List<PolyList> polyList, int targetBigID)
        {
            PolyList foundObject = polyList.Find(obj => obj.BigID ==
targetBigID);

            return foundObject != null ? foundObject.id : '\0';
        }

        // Event handler for the decryption button click
        private void Run_decipher_Click(object sender, EventArgs e)
        {
            decrypt();
        }

        private void label4_Click(object sender, EventArgs e) { }

        // Static method to filter valid characters for encryption
        static string CheckStringEn(string input)
        {
            string charList = "aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż";
            char[] allowedChars = charList.ToCharArray();
            char[] resultArray = new char[input.Length];
            int resultIndex = 0;

            foreach (char character in input)
            {
                if (Array.IndexOf(allowedChars, character) != -1)
                {
                    resultArray[resultIndex++] = character;
                }
            }
```

```
    return new string(resultArray, 0, resultIndex);
}

// Static method to filter valid characters for decryption
static string CheckStringDe(string input)
{
    string charList = "0123456789";
    char[] allowedChars = charList.ToCharArray();
    char[] resultArray = new char[input.Length];
    int resultIndex = 0;

    foreach (char character in input)
    {
        if (Array.IndexOf(allowedChars, character) != -1)
        {
            resultArray[resultIndex++] = character;
        }
    }

    return new string(resultArray, 0, resultIndex);
}

// Static method for additional complication during decryption
static string MoreComplicatedDE(string input)
{
    char[] resultArray = input.ToCharArray();
    if (input.Length > 2)
    {
        int a = resultArray[2] - '0';
        int b = resultArray[3] - '0';

        a = (a + 2) % 7;
        b = (b + 2) % 7;

        resultArray[2] = (char)(a + '0'); ;
        resultArray[3] = (char)(b + '0'); ;

        return new string(resultArray, 0, input.Length);
    }
    else return input;
}

// Static method for additional complication during encryption
static string MoreComplicatedEN(string input)
{
    char[] resultArray = input.ToCharArray();
    if (input.Length > 2)
    {
        int a = resultArray[2] - '0';
        int b = resultArray[3] - '0';

        a = (a - 2) % 7;
        b = (b - 2) % 7;

        if (a < 0)
        {
            a = 7 + a;
        }
        if (b < 0)
        {
            b = 7 + b;
        }
```

```
                resultArray[2] = (char)(a + '0'); ;
                resultArray[3] = (char)(b + '0'); ;

                return new string(resultArray, 0, input.Length);
            }
            else return input;
        }
    }
}
```

1. Przykłady działania

1.1.1. Szyfrowanie:



1.1.2. Deszyfrowanie: