

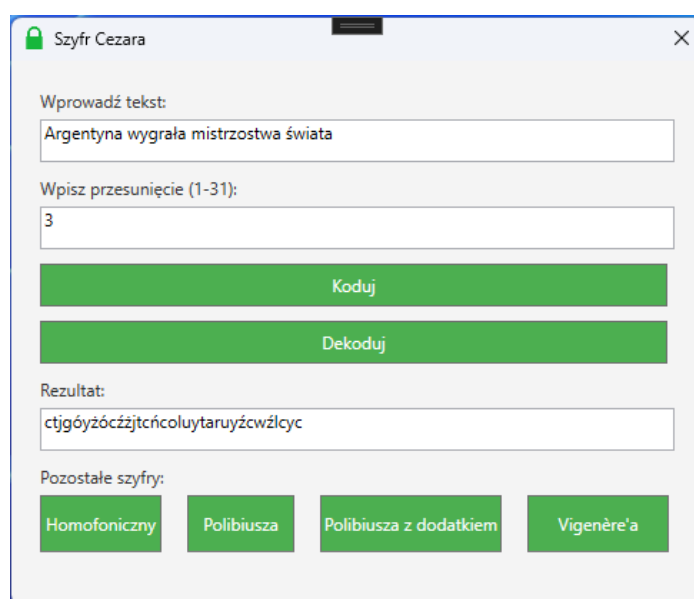
Projekt zaliczeniowy - Szyfry
Bezpieczeństwo systemów
informatycznych

Adam Nowak

Link do aplikacji na github:
<https://github.com/AdamNowak02/Szyfry>

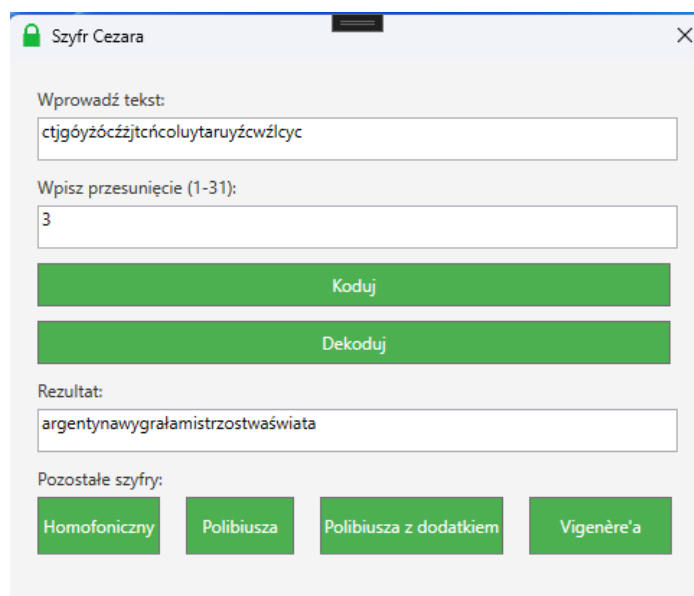
Szyfr Cezara

Polega na przesunięciu każdej z liter o wybraną ilość miejsc w alfabecie np. litera „a” przesunięta o trzy miejsca daje nam „c” (programy uwzględniają kompletny alfabet polski).



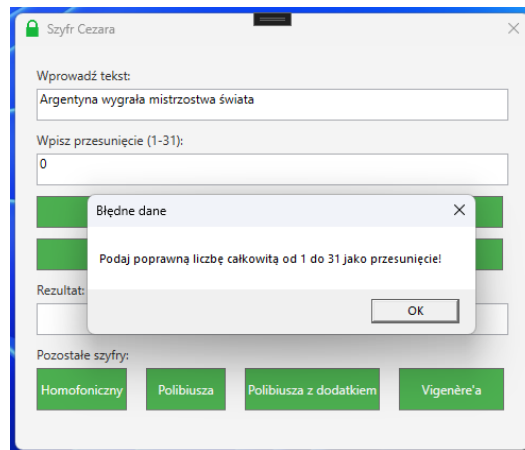
The screenshot shows a web application titled "Szyfr Cezara". It has a light blue header with a lock icon and a close button. The main content area is white. It contains a text input field labeled "Wprowadź tekst:" with the value "Argentyna wygrała mistrzostwa świata". Below it is a numeric input field labeled "Wpisz przesunięcie (1-31):" with the value "3". There are two green buttons: "Koduj" and "Dekoduj". Below these is a text output field labeled "Rezultat:" containing the encoded text "ctjgóyzóćzjtñrcoluytaruyźcwłzyc". At the bottom, there are four green buttons labeled "Homofoniczny", "Polibiusza", "Polibiusza z dodatkiem", and "Vigenère'a".

Program usuwa spacje w celu utrudnienia deszyfracji. Po skopiowaniu rezultatu i naciśnięciu „Dekoduj” otrzymujemy nasz początkowy tekst.



This screenshot shows the same application after clicking the "Dekoduj" button. The "Wprowadź tekst:" field now contains the encoded text "ctjgóyzóćzjtñrcoluytaruyźcwłzyc". The "Rezultat:" field now displays the decoded text "argentynawygrałamistrzostwaświata", where the spaces have been removed. The other elements, including the shift value of 3 and the buttons, remain the same.

Oczywiście wprowadzenie w przesunięciu innej wartości niż liczby z zakresu 1-31 spowoduje wyświetlenie stosownego komunikatu.



Oto fragment kodu z MainWindow.xaml.cs odpowiadający za kodowanie oraz dekodowanie w tym oknie.

```
public partial class MainWindow : Window
{
    private static readonly string PolishAlphabet = "aąbcćdeęfghijklłmnńoóprśstuwyzżź";

    public MainWindow()
    {
        InitializeComponent();

        PlainTextTextBox.SetValue(InputMethod.IsInputMethodEnabledProperty, false);
        ShiftTextBox.SetValue(InputMethod.IsInputMethodEnabledProperty, false);
        ResultTextBox.SetValue(InputMethod.IsInputMethodEnabledProperty, false);
    }

    private void EncryptButton_Click(object sender, RoutedEventArgs e)
    {
        if (int.TryParse(ShiftTextBox.Text, out int shift) && shift >= 1 && shift <= 31)
        {
            ResultTextBox.Text = Encrypt(PlainTextTextBox.Text.ToLower(), shift).Replace(" ", "");
        }
        else
        {
            MessageBox.Show("Podaj poprawną liczbę całkowitą od 1 do 31 jako przesunięcie!", "Błędne dane");
        }
    }

    private void DecryptButton_Click(object sender, RoutedEventArgs e)
    {
        if (int.TryParse(ShiftTextBox.Text, out int shift) && shift >= 1 && shift <= 31)
        {
            ResultTextBox.Text = Decrypt(PlainTextTextBox.Text.ToLower(), shift).Replace(" ", "");
        }
        else
        {
            MessageBox.Show("Podaj poprawną liczbę całkowitą od 1 do 31 jako przesunięcie!", "Błędne dane");
        }
    }

    private string Encrypt(string text, int shift)
    {
        char[] result = text.ToCharArray();

        for (int i = 0; i < result.Length; i++)
        {
```

```

        if (char.IsLetter(result[i]))
        {
            int index = (PolishAlphabet.IndexOf(result[i]) + shift) % PolishAlphabet.Length;
            result[i] = PolishAlphabet[index];
        }
    }

    return new string(result);
}

private string Decrypt(string text, int shift)
{
    return Encrypt(text, PolishAlphabet.Length - shift);
}
}

```

Szyfr Homofoniczny

Polega na zastąpieniu danej litery zestawem znaków. W zależności od częstotliwości występowania litery zmienia się ilość zastępujących ją znaków (w tym przypadku liczb). Za literę „a” jest dziewięć zamienników a za „ą” już tylko jeden ponieważ występuje rzadko. Poniżej fragment kodu dla zobrazowania.

```

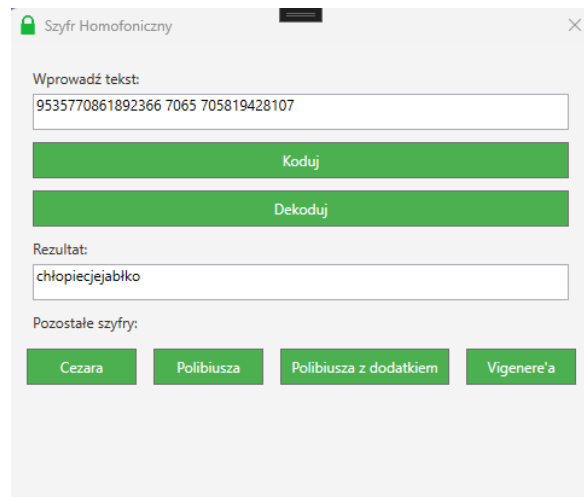
//zadeklaruj słownik z literami alfabetu
homophonicMapping.Add('A', new List<string> { "92", "48", "18", "04", "17", "82", "34", "58", "41" });
homophonicMapping.Add('A', new List<string> { "45" });
homophonicMapping.Add('B', new List<string> { "19" });
homophonicMapping.Add('C', new List<string> { "80", "73", "95", "66" });

```

Taka kombinacja powoduje, że jeden i ten sam tekst zwykle będzie kodowany jako inny ciąg liczb np.:

The image shows two side-by-side windows of a software application titled "Szyfr Homofoniczny". Both windows have a text input field labeled "Wprowadź tekst:" containing the text "Chłopiec je jabłko". Below the input field are two green buttons: "Koduj" and "Dekoduj". Underneath these buttons is a text field labeled "Rezultat:". In the left window, the result is "9535774961106573 7044 701819429807". In the right window, the result is "8035775461157695 7062 284119427807". At the bottom of each window, there is a section labeled "Pozostałe szyfry:" with four green buttons: "Cezara", "Polibiusza", "Polibiusza z dodatkiem", and "Vigenere'a".

Teraz dekodujemy otrzymany kod i otrzymujemy wprowadzony tekst (ponownie usunięte spacje oraz duże litery). Dekodowanie działa bez różnicy dla obu ciągów znaków. Dekoder wie które liczby mogą zastępować daną literę.



Oto fragment kodu z Window1.xaml.cs odpowiadający za kodowanie oraz dekodowanie w tym oknie.

```
public partial class Window1 : Window
{
    private Dictionary<char, List<string>> homophonicMapping;

    public Window1()
    {
        InitializeComponent();
        InitializeHomophonicMapping();
    }

    private void InitializeHomophonicMapping()
    {
        homophonicMapping = new Dictionary<char, List<string>>();

        //zadeklaruj słownik z literami alfabetu
        homophonicMapping.Add('A', new List<string> { "92", "48", "18", "04", "17", "82", "34", "58", "41" });
        homophonicMapping.Add('A', new List<string> { "45" });
        homophonicMapping.Add('B', new List<string> { "19" });
        homophonicMapping.Add('C', new List<string> { "80", "73", "95", "66" });
        homophonicMapping.Add('C', new List<string> { "12" });
        homophonicMapping.Add('D', new List<string> { "21", "68", "40" });
        homophonicMapping.Add('E', new List<string> { "23", "76", "62", "57", "01", "56", "44", "65" });
        homophonicMapping.Add('E', new List<string> { "50" });
        homophonicMapping.Add('F', new List<string> { "32" });
        homophonicMapping.Add('G', new List<string> { "29" });
        homophonicMapping.Add('H', new List<string> { "35" });
        homophonicMapping.Add('I', new List<string> { "89", "10", "93", "15", "63", "72", "11", "24" });
        homophonicMapping.Add('J', new List<string> { "28", "70" });
        homophonicMapping.Add('K', new List<string> { "78", "98", "81", "69" });
        homophonicMapping.Add('L', new List<string> { "26", "88" });
        homophonicMapping.Add('k', new List<string> { "42", "77" });
        homophonicMapping.Add('M', new List<string> { "97", "13", "36" });
        homophonicMapping.Add('N', new List<string> { "03", "14", "75", "74", "71", "55" });
        homophonicMapping.Add('Ń', new List<string> { "25" });
        homophonicMapping.Add('O', new List<string> { "08", "53", "07", "94", "49", "09", "54", "38" });
        homophonicMapping.Add('Ó', new List<string> { "47" });
        homophonicMapping.Add('P', new List<string> { "20", "64", "61" });
        homophonicMapping.Add('Q', new List<string> { "59" });
        homophonicMapping.Add('R', new List<string> { "33", "37", "84", "46", "86" });
        homophonicMapping.Add('S', new List<string> { "31", "52", "60", "83" });
        homophonicMapping.Add('Ś', new List<string> { "16" });
        homophonicMapping.Add('T', new List<string> { "85", "22", "67", "91" });
        homophonicMapping.Add('U', new List<string> { "90", "79", "74" });
        homophonicMapping.Add('V', new List<string> { "43" });
        homophonicMapping.Add('W', new List<string> { "30", "51", "96", "87", "05" });
        homophonicMapping.Add('X', new List<string> { "27" });
        homophonicMapping.Add('Y', new List<string> { "06", "99", "16", "02" });
        homophonicMapping.Add('Z', new List<string> { "59", "28", "75", "92", "70", "43" });
        homophonicMapping.Add('Ż', new List<string> { "39" });
        homophonicMapping.Add('Ż', new List<string> { "58", "76" });
    }
}
```

```

private void EncryptButton_Click(object sender, RoutedEventArgs e)
{
    string plainText = PlainTextTextBox.Text.ToUpper();
    string encryptedText = Encrypt(plainText);
    ResultTextBox.Text = encryptedText;
}

private void DecryptButton_Click(object sender, RoutedEventArgs e)
{
    string encryptedText = ResultTextBox.Text;
    string decryptedText = Decrypt(encryptedText);
    ResultTextBox.Text = decryptedText;
}

private string Encrypt(string plainText)
{
    StringBuilder encryptedText = new StringBuilder();

    foreach (char c in plainText)
    {
        if (homophonicMapping.ContainsKey(c))
        {
            List<string> codes = homophonicMapping[c];
            int randomIndex = new Random().Next(codes.Count);
            encryptedText.Append(codes[randomIndex]);
        }
        else
        {
            encryptedText.Append(c);
        }
    }

    return encryptedText.ToString();
}

private string Decrypt(string encryptedText)
{
    StringBuilder decryptedText = new StringBuilder();

    // Usuń ewentualne spacje z zakodowanego tekstu
    encryptedText = encryptedText.Replace(" ", "");

    // Odkoduj tekst
    for (int i = 0; i < encryptedText.Length; i++)
    {
        char c = encryptedText[i];

        // Sprawdź, czy znak jest w mapowaniu homofonicznym
        foreach (var entry in homophonicMapping)
        {
            if (entry.Value.Contains(encryptedText.Substring(i, Math.Min(2, encryptedText.Length - i))))
            {
                decryptedText.Append(entry.Key.ToString().ToLower());
                i++; // Przeskocz do następnego znaku kodu
                break;
            }
            else if (entry.Key == c)
            {
                decryptedText.Append(c);
                break;
            }
        }
    }

    return decryptedText.ToString();
}
}

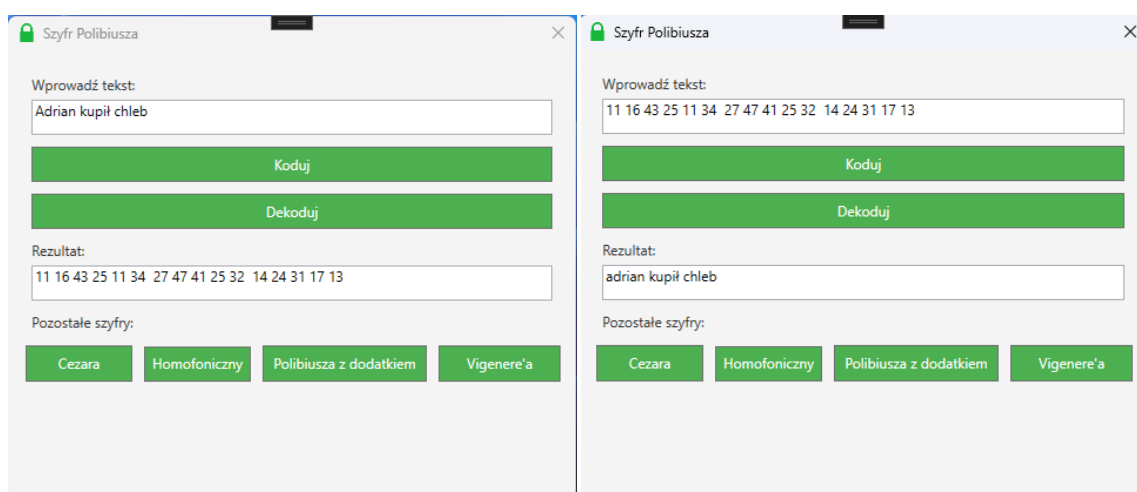
```

Szyfr Polibiusza

Polega na zastąpieniu danej litery jej współrzędnymi z macierzy. Kolejno najpierw wiersz a następnie kolumna. Np. „a”=11, „ą”=12, „ę”=21 itd. Oto fragment kodu z macierzą dla zobrazowania.

```
public partial class Window2 : Window
{
    private char[,] polibiuszTable = {
        {'a', 'a', 'b', 'c', 'ć', 'd', 'e'},
        {'e', 'f', 'g', 'h', 'i', 'j', 'k'},
        {'l', 'ł', 'm', 'n', 'ń', 'o', 'ó'},
        {'p', 'q', 'r', 's', 'ś', 't', 'u'},
        {'v', 'w', 'x', 'y', 'z', 'ż', 'ź'}
    };
};
```

Przykładowe zdanie po kodowaniu oraz dekodowaniu



Oto fragment kodu z Window2.xaml.cs odpowiadający za kodowanie oraz dekodowanie w tym oknie.

```
public partial class Window2 : Window
{
    private char[,] polibiuszTable = {
        {'a', 'a', 'b', 'c', 'ć', 'd', 'e'},
        {'e', 'f', 'g', 'h', 'i', 'j', 'k'},
        {'l', 'ł', 'm', 'n', 'ń', 'o', 'ó'},
        {'p', 'q', 'r', 's', 'ś', 't', 'u'},
        {'v', 'w', 'x', 'y', 'z', 'ż', 'ź'}
    };
};

public Window2()
{
    InitializeComponent();
}

private void EncryptButton_Click(object sender, RoutedEventArgs e)
{
    string plainText = PlainTextTextBox.Text.ToLower();
    string encryptedText = Encrypt(plainText);
    ResultTextBox.Text = encryptedText;
}

private void DecryptButton_Click(object sender, RoutedEventArgs e)
{
    string encryptedText = ResultTextBox.Text;
    string decryptedText = Decrypt(encryptedText);
    ResultTextBox.Text = decryptedText;
}
```

```

    }

    private string Encrypt(string input)
    {
        input = input.ToLower();
        string encryptedMessage = "";

        foreach (char character in input)
        {
            if (character == ' ')
            {
                encryptedMessage += ' ';
                continue;
            }

            for (int i = 0; i < polibiuszTable.GetLength(0); i++)
            {
                for (int j = 0; j < polibiuszTable.GetLength(1); j++)
                {
                    if (polibiuszTable[i, j] == character)
                    {
                        encryptedMessage += (i + 1).ToString() + (j + 1).ToString() + ' ';
                        break;
                    }
                }
            }
        }

        return encryptedMessage.Trim();
    }

    private string Decrypt(string input)
    {
        string decryptedMessage = "";

        string[] pairs = input.Split(' ');

        foreach (string pair in pairs)
        {
            if (pair == "")
            {
                decryptedMessage += ' ';
                continue;
            }

            int row = int.Parse(pair[0].ToString()) - 1;
            int col = int.Parse(pair[1].ToString()) - 1;

            decryptedMessage += polibiuszTable[row, col];
        }

        return decryptedMessage;
    }
}

```

Szyfr Polibiusza z dodatkiem arytmetycznym

Działa on na tej samej zasadzie co Szyfr Polibiusza natomiast dodatkiem jest to, że do uzyskanej liczby dodawana jest liczba dwa a następnie liczba podnoszona jest do kwadratu. Na przykładzie liczby „a”. Liczba z macierzy to 11, dodajemy 2 i otrzymujemy 13, podnosimy do kwadratu i otrzymujemy 169. Takim sposobem litera „a” kodowana jest jako 169. Pozostałe liczby analogicznie

Oto zdanie takie jak w powyższym szyfrze lecz zakodowane oraz zdekodowane wraz z dodatkiem arytmetycznym.

Wprowadź tekst:
Adrian kupił chleb

Koduj

Dekoduj

Rezultat:
169 324 2025 729 169 1296 841 2401 1849 729 1156 256 676 1089 361 225

Pozostałe szyfry:
Cezara Homofoniczny Polibiusza Vigenere'a

Wprowadź tekst:
169 324 2025 729 169 1296 841 2401 1849 729 1156 256 676 1089 361 225

Koduj

Dekoduj

Rezultat:
adrian kupił chleb

Pozostałe szyfry:
Cezara Homofoniczny Polibiusza Vigenere'a

Względem zwykłego kodu Szyfru Polibiusza różnica jest niewielka w funkcji Encrypt oraz Decrypt kolejno:

```
...
if (polibiuszTable[i, j] == character)
{
    // Dodaj 2, a następnie podnieś do kwadratu
    int encryptedValue = (i + 1) * 10 + (j + 1) + 2;
    encryptedMessage += (encryptedValue * encryptedValue).ToString() + ' ';
    break;
    //
}
...
```

```
if (pair == "")
{
    decryptedMessage += ' ';
    continue;
}

// Wydobywanie liczby z pary, odejmowanie 2 i pierwiastkowanie kwadratowe
int decryptedValue = (int)Math.Sqrt(int.Parse(pair)) - 2;
int row = decryptedValue / 10 - 1;
int col = decryptedValue % 10 - 1;
//
decryptedMessage += polibiuszTable[row, col];
}
```

Szyfr Vigenère'a

Działanie tego szyfru oparte jest na danej tablicy (u nas wraz z polskimi znakami):

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

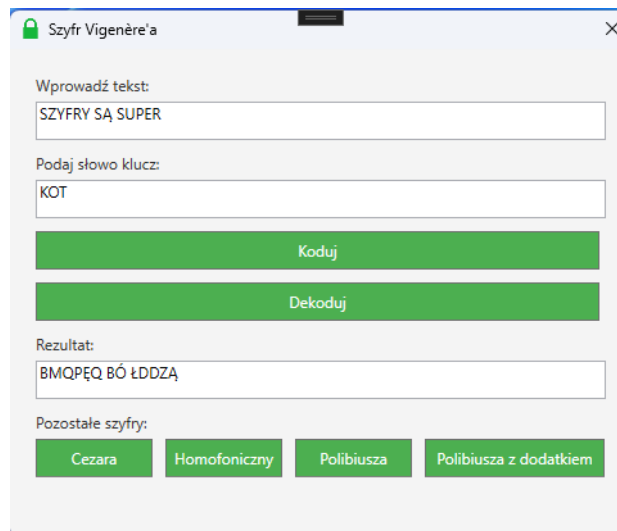
Każdy wiersz odpowiada szyfrowi cezara, w pierwszym wierszu przesunięcie wynosi zero, w drugim jeden, w trzecim dwa itd. Przypuśćmy, że chcemy zaszyfrować tekst „Szyfry są super”. W każdym przypadku potrzebujemy jeszcze słowo klucz „przez które” będziemy kodowali. Do naszego przykładu weźmy słowo „kot”. Zauważamy, że słowo klucz jest krótsze niż nasz tekst do zakodowania więc w tym celu powielamy je i używamy wielokrotnie. Dla zobrazowania przykładu:

SZYFRY SĄ SUPER
KOTKOT KO TKOTK

Teraz szukamy przecięcia w tablicy tych liter tzn. u nas kolejno S oraz K, następnie Z oraz O (brane jako początki wierszy/kolumn). Nie ma znaczenia, która z liter będzie brana jako kolumna a która jako wiersz. Efekt szyfrowania zawsze będzie ten sam.

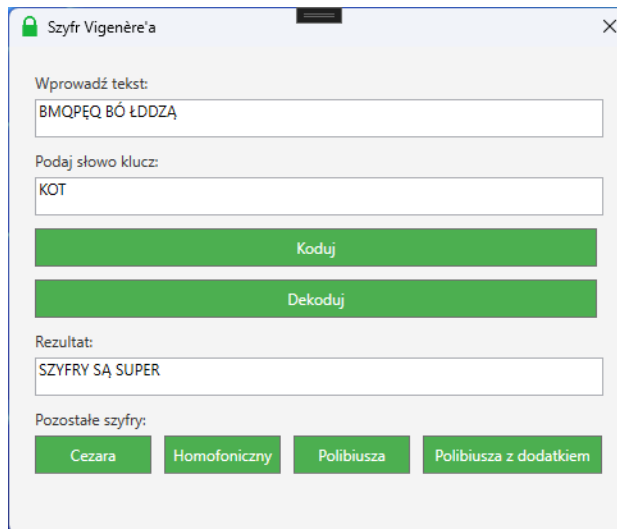
Przykład z aplikacji:

Należy zwrócić uwagę, że w powyższej tabeli przecięcie S oraz K da nam C, u nas daje B ponieważ mamy zaimplementowane polskie znaki co modyfikuje tablicę.



The screenshot shows a window titled "Szyfr Vigenère'a". It has two input fields: "Wprowadź tekst:" containing "SZYFRY SA SUPER" and "Podaj słowo klucz:" containing "KOT". Below these are two green buttons: "Koduj" and "Dekoduj". The "Dekoduj" button is highlighted. Below the buttons is a "Rezultat:" field containing "BMQPĘQ BÓ ŁDDZA". At the bottom, there are four buttons: "Cezara", "Homofoniczny", "Polibiusza", and "Polibiusza z dodatkiem".

Musimy pamiętać aby przy dekodowaniu posłużyć się tym samym słowem kluczowym



The screenshot shows the same window as before, but now the "Wprowadź tekst:" field contains "BMQPĘQ BÓ ŁDDZA" and the "Rezultat:" field contains "SZYFRY SA SUPER". The "Dekoduj" button is still highlighted.

Oto fragment kodu z Window4.xaml.cs odpowiadający za kodowanie oraz dekodowanie w tym oknie.

```
public partial class Window4 : Window
{
    private readonly Dictionary<char, int> PolishAlphabetMap = new Dictionary<char, int>
    {
        {'A', 0}, {'A', 1}, {'B', 2}, {'C', 3}, {'Ć', 4}, {'D', 5},
        {'E', 6}, {'Ę', 7}, {'F', 8}, {'G', 9}, {'H', 10}, {'I', 11},
        {'J', 12}, {'K', 13}, {'L', 14}, {'ł', 15}, {'M', 16}, {'N', 17},
        {'Ń', 18}, {'O', 19}, {'Ó', 20}, {'P', 21}, {'Q', 22}, {'R', 23},
        {'S', 24}, {'Ś', 25}, {'T', 26}, {'U', 27}, {'V', 28}, {'W', 29},
        {'X', 30}, {'Y', 31}, {'Z', 32}, {'Ż', 33}, {'ź', 34}
    };

    public Window4()
    {
        InitializeComponent();
    }
}
```

```

private void Button_Click(object sender, RoutedEventArgs e)
{
    string plainText = PlainTextTextBox.Text.ToUpper();
    string key = ShiftTextBox.Text.ToUpper();

    string encryptedText = Encrypt(plainText, key);
    ResultTextBox.Text = encryptedText;
}

private void Button_Click_1(object sender, RoutedEventArgs e)
{
    string encryptedText = ResultTextBox.Text.ToUpper();
    string key = ShiftTextBox.Text.ToUpper();

    string decryptedText = Decrypt(encryptedText, key);
    ResultTextBox.Text = decryptedText;
}

private string Encrypt(string input, string key)
{
    StringBuilder encryptedMessage = new StringBuilder();
    int keyIndex = 0;

    foreach (char character in input)
    {
        if (PolishAlphabetMap.ContainsKey(character))
        {
            int shift = PolishAlphabetMap[key[keyIndex]];
            char encryptedChar = GetPolishAlphabetChar((PolishAlphabetMap[character] + shift) % 35);
            encryptedMessage.Append(encryptedChar);

            keyIndex = (keyIndex + 1) % key.Length;
        }
        else
        {
            encryptedMessage.Append(character);
        }
    }

    return encryptedMessage.ToString();
}

private string Decrypt(string input, string key)
{
    StringBuilder decryptedMessage = new StringBuilder();
    int keyIndex = 0;

    foreach (char character in input)
    {
        if (PolishAlphabetMap.ContainsKey(character))
        {
            int shift = PolishAlphabetMap[key[keyIndex]];
            char decryptedChar = GetPolishAlphabetChar((PolishAlphabetMap[character] - shift + 35) % 35);
            decryptedMessage.Append(decryptedChar);

            keyIndex = (keyIndex + 1) % key.Length;
        }
        else
        {
            decryptedMessage.Append(character);
        }
    }

    return decryptedMessage.ToString();
}

private char GetPolishAlphabetChar(int index)
{
    foreach (var pair in PolishAlphabetMap)
    {
        if (pair.Value == index)
        {
            return pair.Key;
        }
    }

    return ' '; // Handle the case where the character is not found
}

```