Wyższa Szkoła Bankowa w Poznaniu

Wydział Finansów i Bankowości

Studia stacjonarne I stopnia – Informatyka

Implementacja algorytmu kryptograficznego

Szyfr Cezara

Dokumentacja projektu

Wykonawca: **Patryk Wysocki**

Przedmiot: **Bezpieczeństwo w systemach i sieciach komputerowych**

Grupa: **zIinz_3_K31**

Rok akad.: **2021/2022**

Prowadzący: dr inż. Izabela Janicka-Lipska

1. Cel projektu

Celem projektu jest zapoznanie się z algorytmem kryptograficznym oraz jego implementacja programistyczna i przedstawienie finalnego rozwiązania.

2. Stosowane narzędzia programistyczne

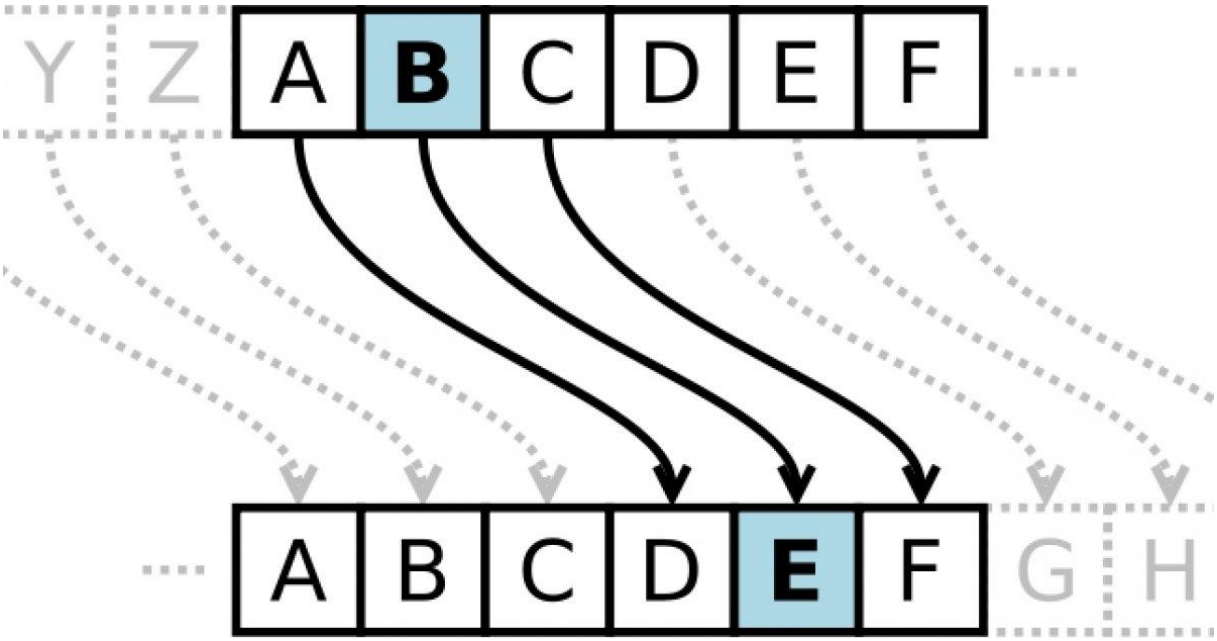
- Visual Studio 2019

3. Opis algorytmu

Szyfr Cezara to prosty szyfr podstawieniowy, który zastępuje każdą literę w tekście jawnym inną literą alfabetu. Nazwa tego hasła pochodzi od pseudonimu Gajusz Juliusz Cezar (100 p.n.e.-44 p.n.e).

Juliusz Cezar szyfrował litery na wiele różnych sposobów, w tym przepisywanie tekstu lub pisanie tekstu po łacinie za pomocą liter greckich. Starożytni pisarze zademonstrowali użycie tego prostego kodu zastępczego, który wymagał przesunięcia tylko jednego lub trzech znaków.

Szyfr Cezara jest jednym z najprostszych algorytmów szyfrujących. Każda litera w tekście jawnym jest zastępowana inną literą, oddzieloną od niej ustaloną liczbą pozycji liter (zawsze w tym samym kierunku). Jeśli algorytm wskazuje pozycję poza ostatnią literą alfabetu, przejdzie na początek alfabetu.



Algorytm można przedstawić w formie matematycznej:

En(x) = (x+n) mod 26 – schemat opisujący szyfrowanie

Dn(x) = (x-n) mod 26 – schemat opisujący deszyfrację

gdzie:

n to wybrane przesunięcie liter, z kolei 26 to liczba liter w alfabecie łacińskim (dla innych alfabetów należy oczywiście użyć innej liczby), „mod” to operacja reszty z dzielenia

Szyfr Cezara można łatwo złamać. Wystarczy sprawdzić wszystkie 26 możliwych (łacińskich) przesunięć liter, aby znaleźć szukany tekst jawny.

4. Opis implementacji

Implementacja w języku C# (z wykorzystaniem interfejsu graficznego)

Stworzenie tablic niezbędnych do operacji matematycznych.

```
public static char[] alfabet =
  {'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', };
```

```
public static char[] alfabet_m =
  {'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', };
```

* Aplikacja operuje na zmiennych tablicach dla dużych i małych liter.

```
public static char[] alfabet_reverse = new char[26];
public static char[] alfabet_reverse_m = new char[26];
```

*Tablice niezbędne do procesu deszyfracji – odwołanie w dalszej części dokumentacji.

(odwrócone tablice dużych i małych liter)

```
public static void Odwrocenie_tablic()
{
    for (int i = 0; i < alfabet.Length; i++)
    {
        alfabet_reverse[i] = alfabet[i];
        alfabet_reverse_m[i] = alfabet_m[i];
    }
    Array.Reverse(alfabet_reverse);
    Array.Reverse(alfabet_reverse_m);
    if (alfabet_reverse[0] == 'Z')
        MessageBox.Show("Wczytano dane");
}
```

*Schemat pozwalający wykonać odwrócenie tablicy alfabetu (osobno dla małych i dużych liter)

```
public static void Code(string tekst_wejsciowy, int kod) // szyfrowanie
{
    Array.Clear(szyfr, 0, szyfr.Length);
    char[] tekst;
    tekst = tekst_wejsciowy.ToCharArray(); // tekst wpisany przez użytkonika
    int point = 0;

    for (int i = 0; i < tekst.Length; i++) // filtrowanie znaków w tekście użytkownika
    {
        if (tekst[i] != ' ') // sprawdzanie czy znak z tekstu nie jest spacją
        {
            for (int j = 0; j < alfabet.Length; j++) // szukanie w alfabecie znaku pasującego do znaku z tekstu (DUŻE LITERY)
            {
                if (tekst[i] == alfabet[j]) // jeśli znak pasuje to trzeba go przesunąć o kod
                {
                    //     C = (n + k) mod 26,
                    //gdzie k jest kluczem szyfrowania, n jest numerem litery, którą szyfrujemy, a C jest numerem litery po zaszyfrowaniu.
                    szyfr[point] = alfabet[(j + kod) % 26];
                    point++;
                }
            }
            for (int j = 0; j < alfabet_m.Length; j++) // szukanie w alfabecie znaku pasującego do znaku z tekstu (MAŁE LITERY)
            {
                if (tekst[i] == alfabet_m[j]) // jeśli znak pasuje to trzeba go przesunąć o kod
                {
                    //     C = (n + k) mod 26,
                    //gdzie k jest kluczem szyfrowania, n jest numerem litery, którą szyfrujemy, a C jest numerem litery po zaszyfrowaniu.
                    szyfr[point] = alfabet_m[(j + kod) % 26];
                    point++;
                }
            }

        }
        else if (tekst[i] == ' ') // jeśli jest jest spacją
        {
            szyfr[i] = ' '; // przypisuje spację do ciągu szyfru
            point++;
        }
        else
        {
            MessageBox.Show("Nieprawidłowy znak wprowadzony do szyfrowania");
        }
    }

}
```

*Obiekt odpowiadający za szyfrowani tekstu jawnego, przy zdefiniowanym kluczu (podanym przez użytkownika)

-Warunki sprawdzające wielkość liter i występowanie spacji w ciągu znaków.

-Wykorzystanie wzoru matematycznego dla długości alfabetu równej 26 znaków.

-Modulo (reszta z dzielenia) jest o kluczowym znaczeniu dla algorytmu.

Zapobiega przekroczeniu granic tablicy i pozwala na obsługę klucza o dużej wartości.

-Komentarze wprowadzone w kodzie (zielony tekst) opisują działanie obiektu, opisane w sposób czytelny dla autora (mogą wystąpić „skróty myślowe”)

```
public static void Decode(string tekst_wejsciowy, int kod) //deszyfracja - zbudowana na identycznym algorytmie, jednak odwróconej tablizy
{
    Array.Clear(deszyfr, 0, deszyfr.Length);
    char[] tekst;
    tekst = tekst_wejsciowy.ToCharArray(); // tekst wpisany przez użytkonika
    int point = 0;

    for (int i = 0; i < tekst.Length; i++) // filtrowanie znaków w tekście użytkownika
    {
        if (tekst[i] != ' ') // sprawdzanie czy znak z tekstu nie jest spacją
        {
            for (int j = 0; j < alfabet_reverse.Length; j++) // szukanie w alfabecie znaku pasującego do znaku z tekstu (DUŻE LITERY)
            {
                if (tekst[i] == alfabet_reverse[j]) // jeśli znak pasuje to trzeba go przesunąć o kod
                {
                    deszyfr[point] = alfabet_reverse[(j + kod) % 26];
                    point++;
                }
            }
            for (int j = 0; j < alfabet_reverse_m.Length; j++) // szukanie w alfabecie znaku pasującego do znaku z tekstu (MAŁE LITERY)
            {
                if (tekst[i] == alfabet_reverse_m[j]) // jeśli znak pasuje to trzeba go przesunąć o kod
                {
                    deszyfr[point] = alfabet_reverse_m[(j + kod) % 26];
                    point++;
                }
            }

        }
        else if (tekst[i] == ' ') // jeśli jest jest spacją
        {
            deszyfr[i] = ' '; // przypisuje spację do ciągu szyfru
            point++;
        }
        else
        {
            MessageBox.Show("Nieprawidłowy znak wprowadzony do odszyfrowania");
        }
    }

}
```

*Obiekt odpowiadający za deszyfrację, wykorzystujący wprowadzony szyfr i klucz podane przez innego użytkownika.

Jeśli klucz został poprawnie wpisany – użytkownik otrzyma tekst jawny przed szyfrowaniem.

Obiekt zbudowany jest w sposób identyczny jak w poprzednim przypadku.

Niezbędnym elementem jest odwrócona tablica alfabetów, pozwalająca na deszyfrację (w prawo o klucz) – efektem jest uzyskanie tekstu jawnego.

Zastosowanie takiego rozwiązania pozwala na wykorzystanie klucza o niezmienionej wartości.

```
private void do_code_Click(object sender, EventArgs e)
{
    try
    {
        int numVal = Int32.Parse(code.Text);
        if (trackBar.Value == numVal)
        {
            Obliczenia.Code(text_to_code.Text, trackBar.Value);
            text_code.Text = new string(Obliczenia.szyfr);
        }
        else
        {
            Obliczenia.Code(text_to_code.Text, numVal);
            text_code.Text = new string(Obliczenia.szyfr);
        }
    }
    catch (FormatException)
    {
        MessageBox.Show("Błędny kod");
    }
}
```

*Przycisk w aplikacji odpowiadający za szyfrowanie tekstu.

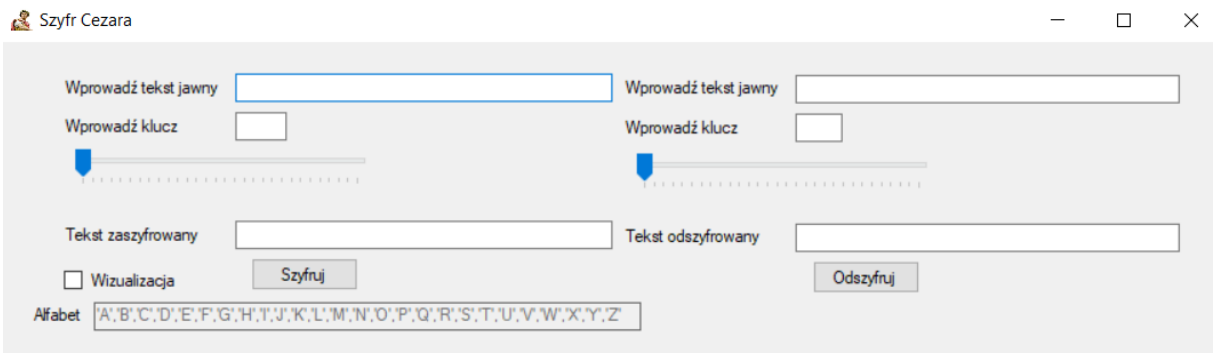
Zabezpieczenie kodu przed błędami użytkownika związanymi z błędnym wpisaniem klucza (przykładowo – użytkownik próbuje użyć innego znaku niż cyfra).

Głównym zadaniem jest wyświetlenie oczekiwanego efektu działania w aplikacji.

```
private void do_decode_Click(object sender, EventArgs e)
{
    try
    {
        int numVal = Int32.Parse(decode.Text);
        if (trackBar1.Value == numVal)
        {
            Obliczenia.Decode(text_to_decode.Text, trackBar1.Value);
            text_decode.Text = new string(Obliczenia.deszyfr);
        }
        else
        {
            Obliczenia.Decode(text_to_decode.Text, numVal);
            text_decode.Text = new string(Obliczenia.deszyfr);
        }
    }
    catch (FormatException)
    {
        MessageBox.Show("Błędny kod");
    }
}
```

*Obiekt opisujący procedurę użycia funkcji deszyfrującej tekst.

Działanie tożsame z poprzednim przykładem.



*Wygląd graficzny aplikacji.

-Podział na część odpowiadającą za szyfrowanie i odszyfrowanie tekstu.

-Dostęp do „TrackBar” w celu łatwiejszego i szybszego definiowania klucza.

-Widok dostępnych liter alfabetu.

Wizualizacja

```
public static string[] znaki = new string[((2 * alfabet.Length))]; // tablica wykorzystana przy wizualizacji
```

*Przypisanie tablicy znaków (liter), niezbędnej przy wizualizacji.

Podwojona wielkość tablicy (w stosunku to wielkości tablicy alfabetu) pozwala na uniknięcie problemu związanego z przekroczeniem indeksu tablicy znaków.

```
public static void Wczytaj_w() // wczytanie danych do tablicy pomocniczej przy wizualizacji
{
    for (int i = 0; i < alfabet.Length; i++)
    {
        znaki[i] = alfabet[i].ToString();
    }
    int point = 0;
    for (int i = alfabet.Length; i < ((2 * alfabet.Length)); i++)
    {
        znaki[i] = alfabet[point].ToString();
        point++;
    }
    MessageBox.Show("Wczytano dane");
}
```

*Obiekt odpowiada za wygenerowanie wartości w tablicy znaków, jako powtórzona dwukrotnie tablica alfabetu (duże litery) – obsługa małych liter nie jest wymagana w wizualizacji.

```
public void Wizualizacja()
{
    char[] tekst;
    char[] szyfr;
    tekst = text_to_code.Text.ToCharArray();
    szyfr = text_code.Text.ToCharArray();
    for (int i = 0; i < tekst.Length; i++)
    {
        if ((tekst[i] != ' ') && (szyfr[i] != ' '))
        {
            Przesun(tekst[i].ToString(), szyfr[i].ToString());
            MessageBox.Show("Znak  " + tekst[i].ToString() + "  został zaszyfrowany jako:  " + szyfr[i].ToString());
            Reset_color();
        }
    }
}
```

*Wizualizacja wykorzystuje zdefiniowane niżej funkcje takie jak Przesun() i Reset_color().

Początkowo definiuje i przypisuje tekst jawny i szyfr do zmiennych tablicowych.

Weryfikuje wystąpienie spacji i przekazuje dalszą część do funkcji Przesun().

Po wykonaniu operacji wyświetla gotowy wynik przesunięcia liter i uruchamia procedurę Reset_color().

```
public void Reset_color()
{
    p1.BackColor = System.Drawing.SystemColors.Info;
    p2.BackColor = System.Drawing.SystemColors.Info;
    p3.BackColor = System.Drawing.SystemColors.Info;
    p4.BackColor = System.Drawing.SystemColors.Info;
    p5.BackColor = System.Drawing.SystemColors.Info;
    p6.BackColor = System.Drawing.SystemColors.Info;
    p7.BackColor = System.Drawing.SystemColors.Info;
    p8.BackColor = System.Drawing.SystemColors.Info;
    p9.BackColor = System.Drawing.SystemColors.Info;
    p10.BackColor = System.Drawing.SystemColors.Info;
    p11.BackColor = System.Drawing.SystemColors.Info;
    p12.BackColor = System.Drawing.SystemColors.Info;
    p13.BackColor = System.Drawing.SystemColors.Info;
    p14.BackColor = System.Drawing.SystemColors.Info;
    p15.BackColor = System.Drawing.SystemColors.Info;
    p16.BackColor = System.Drawing.SystemColors.Info;
    p17.BackColor = System.Drawing.SystemColors.Info;
    p18.BackColor = System.Drawing.SystemColors.Info;
    p19.BackColor = System.Drawing.SystemColors.Info;
    p20.BackColor = System.Drawing.SystemColors.Info;
    p21.BackColor = System.Drawing.SystemColors.Info;
    p22.BackColor = System.Drawing.SystemColors.Info;
    p23.BackColor = System.Drawing.SystemColors.Info;
    p24.BackColor = System.Drawing.SystemColors.Info;
    p25.BackColor = System.Drawing.SystemColors.Info;
    p26.BackColor = System.Drawing.SystemColors.Info;
}
```

*Funkcja Reset_color() odpowiada za końcowy efekt przywrócenia wizualizacji do stanu początkowego.

```
public async void Przesun(string a, string b)
{
    a = a.ToUpper();
    b = b.ToUpper();
    try
    {
        if (a == b)
        {
            Pokaz(a, 0);
        }
        else
        {
            for (int i = 0; i < Obliczenia.znaki.Length; i++)
            {
                if (Obliczenia.znaki[i] == a)
                {
                    Pokaz(Obliczenia.znaki[i], 0);
                    while (true)
                    {
                        Pokaz(Obliczenia.znaki[i++], 0);
                        await Task.Delay(200);
                        if (Obliczenia.znaki[i] == b)
                        {
                            Pokaz(Obliczenia.znaki[i], 1);
                            break;
                        }
                    }
                }
            }
        }
    }
    catch (IndexOutOfRangeException)
    {
    }

}
```

*Funkcja Przesun() odpowiada za graficzny efekt przesunięcia znaków o zdefiniowany klucz.

Pobiera odpowiednie znaki z tekstu jawnego i szyfru, dla których ma wyświetlić przejścia znaków.

Pomija w tym przypadku klucze o dużej wartości i wyświetla ostatnie przejście litery tekstu jawnego w odpowiednie miejsce litery szyfru.

Funkcja działa asynchronicznie, co wymaga od użytkownika cierpliwej obserwacji przejść i zakończenie działania po wykonaniu wszystkich operacji.

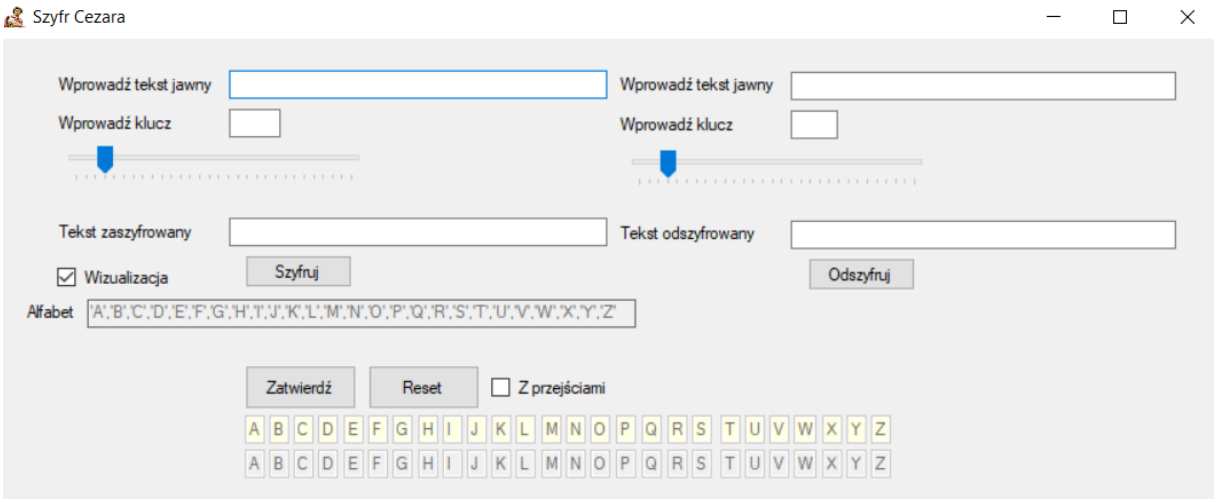
Wykorzystanie zdefiniowanej funkcji Pokaz() (opisanej niżej).

```
public void Pokaz(string znak, int i)
{
    znak = znak.ToUpper();
    switch (znak)
    {
        case "A":
            {
                if (i == 0)
                {
                    p1.BackColor = System.Drawing.Color.Red;
                }
                else
                {
                    p1.BackColor = System.Drawing.Color.Green;
                }
                break;
            }
        case "B":
            {
                if (i == 0)
                {
                    p2.BackColor = System.Drawing.Color.Red;
                }
                else
                {
                    p2.BackColor = System.Drawing.Color.Green;
                }
                break;
            }
        case "C":
            {
                if (i == 0)
                {
                    p3.BackColor = System.Drawing.Color.Red;
                }
                else
                {
                    p3.BackColor = System.Drawing.Color.Green;
                }
                break;
            }
```

*Funkcja Pokaz() początkowo konwertuje wprowadzany znak jako dużą literę (pominięcie obsługi małych liter jest celowe)

Wymagane jest również podanie liczby. Jeśli będzie to „0” – funkcja przypisuje dla konkretnego pola kolor zielony, jeśli inna – czerwony.

Zrzut ekranu pokazuje początkowe znaki, w dalszej części zostały zdefiniowane pozostałe litery alfabetu.



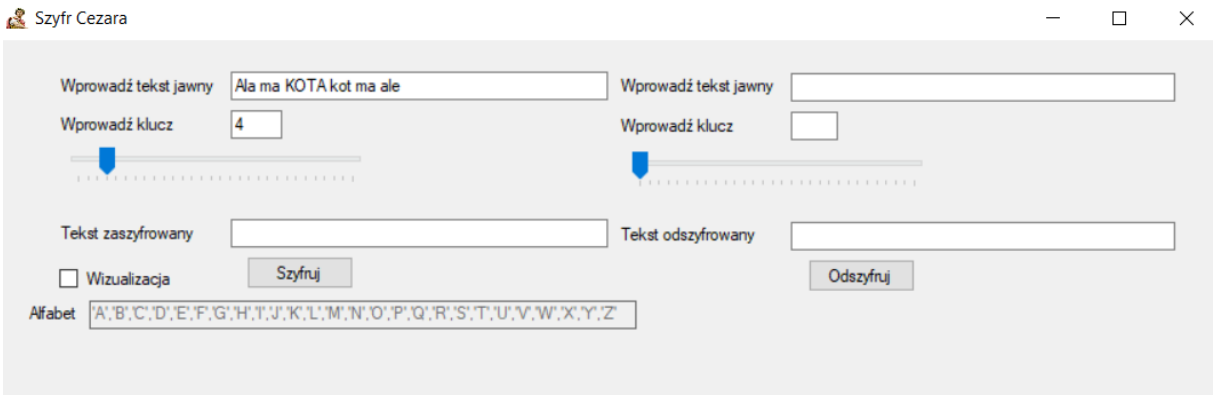
*Wgląd w procedurę wizualizacji dostępny jest po wybraniu takiej opcji w aplikacji.

Przycisk „Zatwierdź” pozwala na uruchomienie wizualizacji.

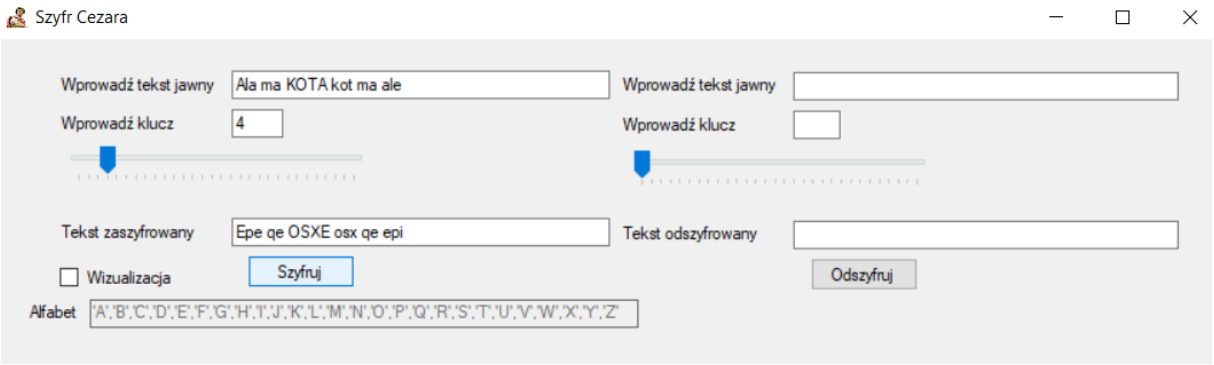
Przycisk „Reset” przywraca wygląd wizualizacji do stanu początkowego (Przydatne w momencie sprawdzania kilku szyfrów, jeden po drugim).

Opcja „Z przejściami” pokazuje cykliczne przejście znaków. Brak zaznaczenia tej opcji wyświetla gotowy wynik, łącznie z komunikatem dla użytkownika o wykonaniu zadanej operacji.

5. Wyniki testowania zaimplementowanej aplikacji

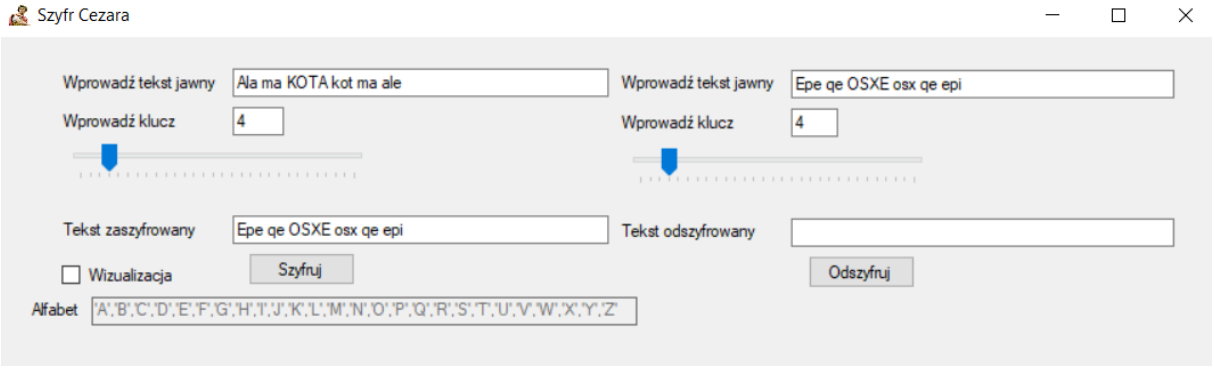


*Wprowadzenie tekstu jawnego oraz klucza.

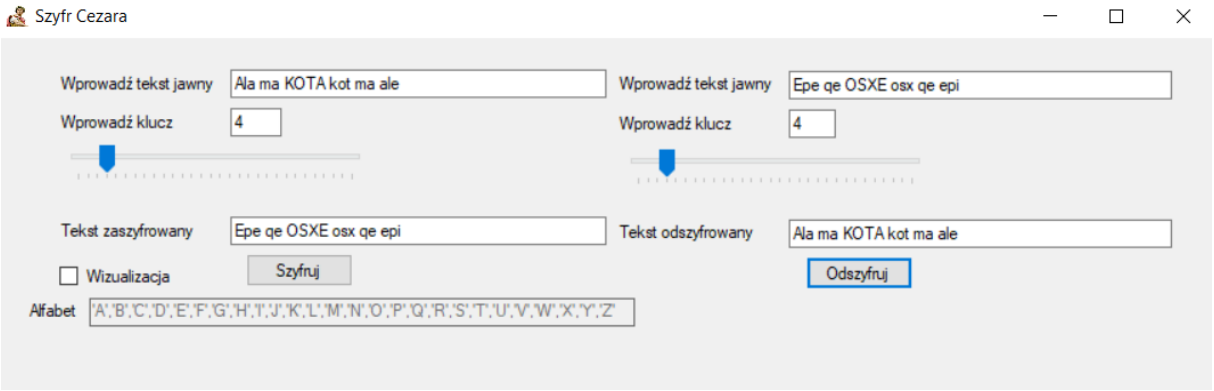


*Użycie przycisku odpowiadającego za szyfrowanie i uzyskanie tekstu zaszyfrowanego na ekranie aplikacji

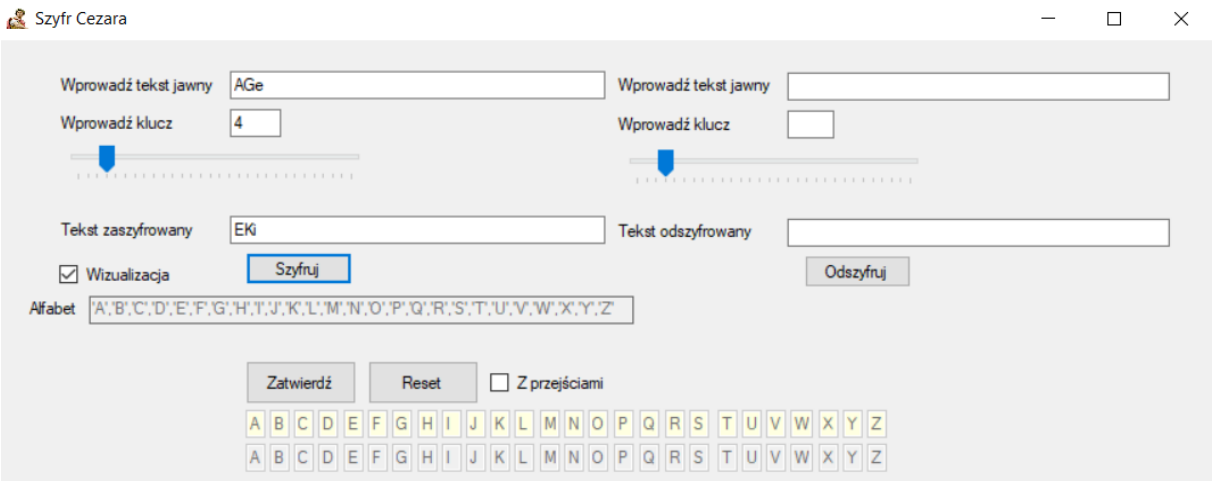
Efektem jest uzyskanie tekstu w odpowiadającym formacie, tj. odpowiednie uwzględnienie spacji i wielkości liter.



*Wprowadzenie tekstu zaszyfrowanego przez użytkownika nr.1 w miejsce tekstu jawnego dla użytkownika nr.2 oraz odpowiedniego klucza (co ważne, użytkownicy mogą posługiwać się taką samą wartością klucza).



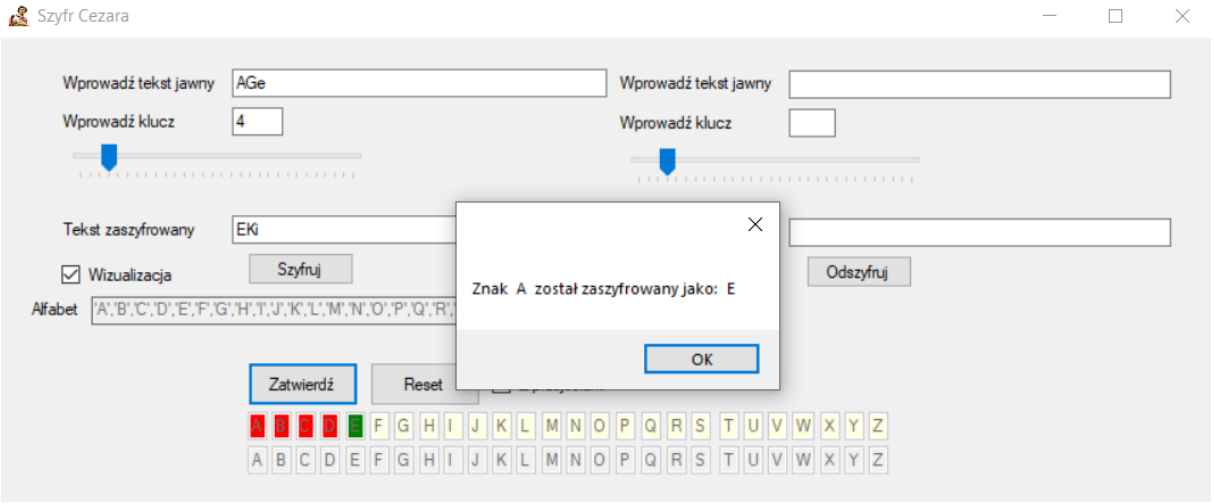
Użytkownik nr.2 używając przycisku odpowiadającego za odszyfrowanie tekstu, uzyskuje tożsamy tekst z tekstem jawnym dla użytkownika nr.1.



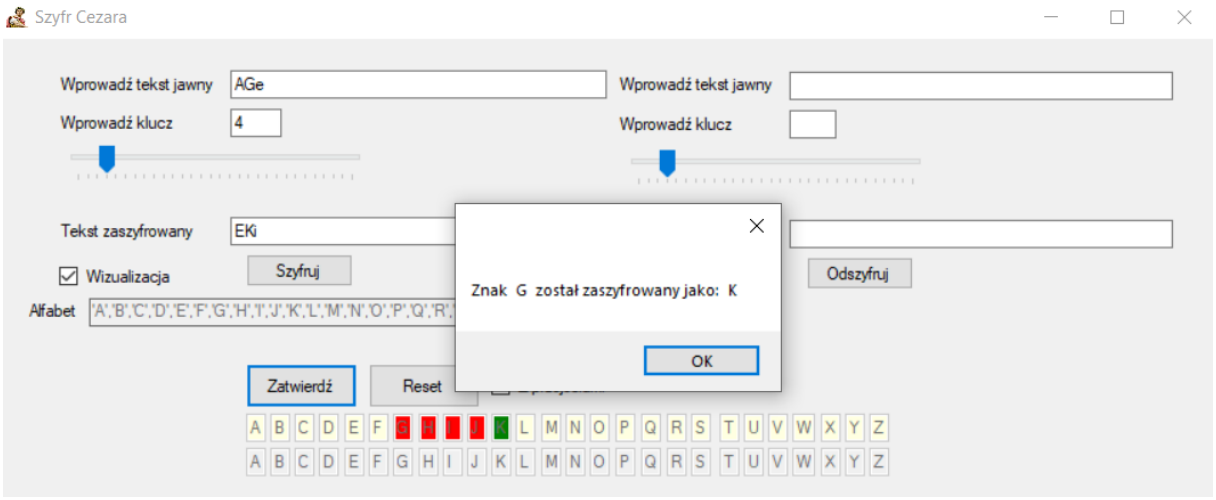
*Zaznaczając opcję wizualizacji użytkownik otrzymuje dostęp części aplikacji, odpowiadającej za te działania.

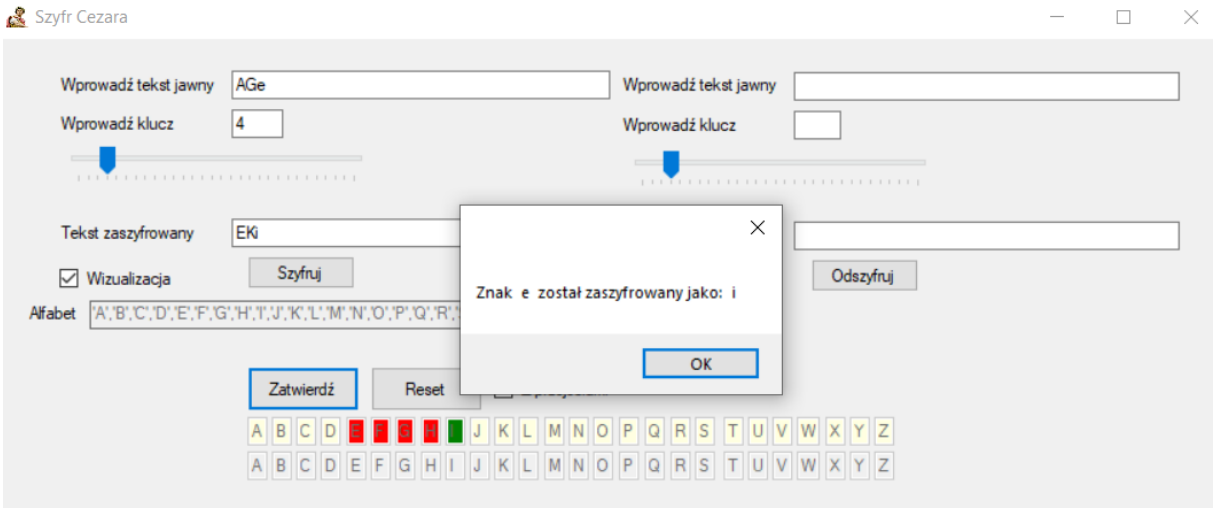
Ciągi znaków zostały zmienione na inne, na potrzeby prezentacji procedury.

Użytkownik ma opcję wyboru wizualizacji z przejściami, tj. znaki przesuwają się cyklicznie, bądź uproszczonego wygląd, gdzie efekt wyświetlany jest bez przejść.



*Pierwszy znak tekstu jawnego („A”) został przesunięty o 4 pozycje (według klucza) w prawo, co daje pierwszy znak szyfru („E).





*Przedstawienie pozostałych znaków tekstu jawnego i szyfru.

W wizualizacji operacje odbywają się na dużych literach, jednak w komunikacje dla użytkownika otrzymujemy informację o faktycznej wielkości litery.

6. Uwagi i wnioski

Algorytm przedstawiający Szyfr Cezara jest prostym działaniem matematyczny, jednak podczas badania zagadnienia i prób wykonania aplikacji w inny sposób okazał się niezbędny.

Kluczową operacją jest działanie modulo pozwalające utrzymać index tablicy w jej granicach.

Aplikacja została zaprojektowana w układzie pozwalającym użytkownikowi na szybką i łatwą obsługę.

Operacje wykonywane w aplikacji nie obciążają procesora, jak i karty graficznej.

Istnieje możliwość uruchomienia programu na sprzęcie komputerowym niższej klasy.

7. Literatura

http://www.algorytm.org/kryptografia/szyfr-cezara.html

http://www.crypto-it.net/pl/proste/szyfr-cezara.html

http://www.algorytm.edu.pl/algorytmy-maturalne/szyfr-cezara.html

https://mateuszrus.pl/szyfr-cezara/