

Uniwersytet Morski w Gdyni



Projekt

Napisanie programu pozwalającego na stworzenie histogramu z danych dostępnych w pliku tekstowym

Oliwia Milewska
1 rok Informatyka gr. Lab. 4
Nr albumu: 47542
Wydział Elektryczny

1. Teoria

Do napisania programu używamy środowiska .NET Framework stworzonego przez firmę Microsoft, obejmujące środowisko uruchomieniowe (Common Language Runtime – CLR) oraz biblioteki klas. Jako kompilator został użyty program Visual Studio, również stworzony przez firmę Microsoft. Poprzez Visual Studio mamy możliwość tworzenia aplikacji komputerowych, aplikacji mobilnych, stron internetowych oraz usług internetowych. Kompilator wspiera 36 różnych języków programowania oraz w wersji Community jest on darmowy. Do napisania projektu został użyty język wysokopoziomowy C#. Jest to obiektowy język programowania, który jest ścisłe zintegrowany z platformą .NET.

Głównym zadaniem programu jest wczytanie danych z pliku tekstowego, zawierające średnie oceny produktu, a następnie przeanalizowanie ich i stworzenie na ich podstawie histogramu. Aplikacja ma być przyjazna dla użytkownika, czyli charakteryzować się estetyczną szatą graficzną oraz funkcjonalnością. Powinna zapobiegać niechcianym działaniom użytkownika i umieć reagować na napotkane wyjątki programistyczne.

Tworzenie aplikacji zaczynamy od formularza zwanego *Form1*. To na nim możemy dodawać graficzne komponenty tj. *button*, *textBox*, *label*, *checkBox*, *radioButton*, *chart*, które znajdują się po lewej stronie w oknie *Przybornik*. Komponent *button* to nic innego jak przycisk, który wyświetla się na ekranie i może zostać naciśnięty przez użytkownika co wywoła zaprogramowany przez nas algorytm. *TextBox*'ie jest polem tekstowym, w którym możemy umieszczać tekst lub też sam użytkownik może w nim go wprowadzić. W komponencie *label* również zostaje umieszczony tekst ale użytkownik nie ma bezpośrednio wpływu na jego zawartość, jedynie pośrednią np. poprzez naciśnięcie przycisku, które spowoduje zmianę we właściwości *text* w *label*. *CheckBox* jest to pole wielokrotnego wyboru. Po naciśnięciu przez użytkownika pole zmienia się na odhaczone lub nie po czym może nastąpić zaprogramowany algorytm. Komponent *radioButton* również jest polem wyboru ale za to nie wielokrotnego. Gdy używamy ich kilku w naszej aplikacji mamy możliwość zaznaczenia tylko jednego pola. Komponent *chart* służy do tworzenia różnego rodzaju wykresów. Między innymi słupkowe, kołowe i wiele innych. Przy jego programowaniu, oprócz rodzaju wykresu, mamy możliwość, regulowania rozmiaru, zmiany koloru poszczególnych elementów, dodania legendy, opisów, kilku serii danych, tła oraz tego skąd będzie on pobierał potrzebne mu dane.

Wyżej wymienione komponenty są używane w formularzach, zostają one na nie dodane. Efekty takich działań widzimy na *Form1.cs [Projekt]*. Nasz kod, wykonujący się przy różnych działaniach użytkownika, edytujemy w *Form1.cs*. To tam znajduje się kod funkcji zdarzeń. W formularzu *Form1.Designer.cs* znajduje się kod generowany przez Projektanta formularzy systemu Windows, przez co mamy dokładny wgląd w kod tworzący użyte przez nas komponenty. To w tym miejscu są

one programowane od samego początku a my nie musimy się martwić o ich pierwotny wygląd, rozmiar, nazwę, położenie na ekranie oraz inne podstawowe właściwości.

Po prawej stronie programu Visual Studio znajduje się okno eksploratora rozwiązań, które wyświetla wszystkie elementy w projekcie i ułatwia przechodzenie do nich oraz okno właściwości, gdzie możemy zmieniać właściwości elementów znajdujących się na formularzu jak i właściwości samego formularza.

Aby program mógł wczytać dane z pliku podanego przez użytkownika wykorzystuje się klasę *OpenFileDialog*. Otwiera ona okno dialogowe gdzie użytkownik może wybrać plik, który ma zostać wczytany. Dodatkowo klasa umożliwia dodania filtra dzięki, któremu użytkownikowi pojawią się jedynie pliki z takim rozszerzeniem jaki został ustalony.

Do przekonwertowania danych z pliku tekstowego (domyślnie typu string) program używa metody *Parse*. Dzięki niej zmieniamy typ danych na double.

Do zapisu przygotowanych danych do stworzenia wykresu program używa klasy *SaveFileDialog* oraz *BinaryWriter*. Użytkownik sam wybiera miejsce gdzie zapisze się plik oraz nadaje mu nazwę. Podobnie do klasy *OpenFileDialog*, w klasie *SaveFileDialog* możemy ustawić filtr aby plik zapisywał się z konkretnym rozszerzeniem, w przypadku tego programu jest to .txt.

Najistotniejszą częścią programu jest komponent *chart* (używany w drugim formularzu), który umożliwia stworzenie histogramu. Do właściwości *Series* ładujemy wyniki obliczeń ilości ocen z poszczególnych przedziałów. Za pomocą *Color* możemy edytować kolor słupków na wykresie. Dodatkowo dzięki właściwości *Legends*, *Titles* oraz *Series.isValueShownAsLabel* program wpływa na opis wykresu. Umożliwiają wyświetlenie się legendy, tytułu wykresu oraz wartości nad słupkami będące dokładną reprezentacją ilości ocen produktu w danym przedziale.

Podczas wykonywania poszczególnych etapów m.in. zapisywania danych, odczytu danych, program za pomocą *MessageBox' informuje użytkownika o powodzeniu lub błędzie podczas wykonywania kolejnych czynności. *MessageBox* wyświetla okno komunikatu, które przedstawia komunikat dla użytkownika. Jest to modalne okno, które blokuje inne akcje w aplikacji do czasu aż użytkownik go nie zamknie. *MessageBox* może zawierać tekst, przyciski, symbole, które można zaprogramować dzięki odpowiedniemu kodu.*

Do zabezpieczenia wyjątków program korzysta z metody *try/catch*. W bloku *try* umieszczony został kod, który w wyniku swojego działania może wywołać wyjątek. Wyjątek ten łapiemy w bloku *catch*, a następnie dowolnie go obsługujemy. W bloku *finally*, umieszczać instrukcję, które mają się wykonać niezależnie od tego, czy wyjątek został złapany, czy też nie. Jeśli pominiemy blok *catch*, wtedy blok *finally* jest wymagany, w przeciwnym wypadku jest on niewymagany. W programie zostały użyte bloki *try* i *catch*.

2.Zadanie

Program otwiera pliki tekstowe z zapisanymi ocenami produktu od 0 do 5. Liczby zapisane są w postaci zmienoprzecinkowej. Program wydziela wartości liczbowe i wyświetla je w okienku tekstu. Następnie analizuje je i wyznacza ilość ocen z danego przedziału 0-1, 1-2, 2-3, 3-4, 4-5. Program umożliwia zapisanie tych wyników do pliku tekstowego. Następnie użytkownik może modyfikować wygląd histogramu. Wybrać, które dane mają zostać wyświetcone, w jakim kolorze ma być wykres, jaki przedział ocen ma się wyświetlić i czy na histogramie ma być widoczna legenda. Po kliknięciu przycisku *Generuj wykres*, w osobnym okienku wyświetla się gotowy wykres.

Na ocenę 5:

- otwieranie pliku tekstowego z ocenami (np. `ReadAllText`), każdorazowo otwierając okno dialogowe umożliwiające wskazanie pliku (`openFileDialog`)
- parsowanie tekstu (`TryParse`) i wyodrębnienie wartości liczbowych do oddzielnej tablicy
- obliczanie histogramu własnoręcznie napisanym algorytmem
- zapisywanie wyników obliczeń do pliku tekstowego (np. `WriteLine`, `saveFileDialog`)
- wyświetlanie danych na wykresie (`Chart`), w którym można dokonać przynajmniej 3 modyfikacji
- zadbanie o obsługę wyjątków (`try...catch`) związanym z IO
- blokowanie niektórych pól tylko do odczytu oraz funkcjonalności zanim użytkownik ma prawo je wykonać (np. poprzez zablokowanie przycisku otwierającego wykres/obliczającego i zapisującego histogram, zanim zostaną wczytane dane)
- wyświetlanie komunikatów (`MessageBox.Show`) informujących o stanie IO (powodzeniu lub niepowodzeniu podczas otwarcia/zapisu pliku)
- wyświetlanie wykresu w osobnym oknie

Kod:

```
namespace projekt_sem2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        int[] tab1 = new int[5];
        int[] tab2 = new int[5];
        int[] tab3 = new int[5];
        public static double p01, p12, p23, p34, p45;
        int click = 0;
        public static int kolor;
        OpenFileDialog dane = new OpenFileDialog();
        String przetworzone;
```

Fragment 1. Zmienne globalne

Na początek zostały zadeklarowane kilka zmiennych globalnych, które będę używane w kilku metodach w programie.

```
private void button1_Click(object sender, EventArgs e)
{
    //////////////// Odczyt pliku
    dane.Filter = "Pliki tekstowe (*.txt)";
    if (dane.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        textBox1.Text = File.ReadAllText(dane.FileName);
        button2.Enabled = true;
        MessageBox.Show("Otwarto: " + dane.SafeFileName, "Sukces", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        button1.Enabled = false;
        button2.Enabled = true;
        button3.Enabled = false;
        button4.Enabled = false;
    }
    else
    {
        MessageBox.Show("Coś poszło nie tak :c", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Fragment 2. Kod wykonujący się po wciśnięciu przycisku Otwórz plik z danymi

Metoda `button1_Click` służy do odczytu danych z pliku tekstowego wybranego przez użytkownika. Program korzysta tutaj z komponentu `OpenFileDialog` o nazwie `dane`, który został zadeklarowany wcześniej. W linijce 32 widzimy, że został włączony filtr i jedyne pliki jakie będzie można wczytać będą miały rozszerzenie .txt. Jeżeli wszystko pójdzie dobrze na ekranie w miejscu gdzie jest komponent `textBox1` wyświetli się zawartość podanego pliku. Wyświetli się również `MessageBox` informujący o powodzeniu lub niepowodzeniu wczytania pliku. Pod instrukcją warunkową znajdują się komendy, które określają, które przyciski będą teraz dostępne dla użytkownika.

- button1 - Otwórz plik z danymi, niedostępny
- button2 - Przygotuj dane, dostępny
- button3 - Generuj wykres, niedostępny
- button4 - Zapisz dane, niedostępny

```
private void button2_Click(object sender, EventArgs e)
{
    button4.Enabled = true;
    button2.Enabled = false;
    button1.Enabled = true;
    ////////////////// Załadowanie danych z pliku do tablicy int'ów
    string[] liczbyString = new string[100];
    double[] liczbyDouble = new double[100];
    try
    {
        liczbyString = File.ReadAllLines(dane.FileName);
        for (int i = 0; i < 100; i++)
        {
            liczbyDouble[i] = Double.Parse(liczbyString[i]);
        }
    }
```

Fragment 3. Konwersja danych ze string na double

Metoda *button2_Click* służy do przygotowania załadowanych danych w sposób, który pozwoli je następnie przedstawić na histogramie. Znowu zostaje zmieniona dostępność niektórych przycisków. Zadeklarowane zostały dwie tablice, jedna typu String, druga typu double. Aby obsłużyć wyjątki używam bloku *try/catch*. W środku części *try* do tablicy *liczbyString* wpisuje zawartość wczytanego pliku. Następnie za pomocą pętli parsuje dane na double i wpisuje je do tablicy *liczbyDouble*.

```
/////////////////// Policzenie ilości elementów w przedziałach
for (int i = 0; i < 100; i++)
{
    if (click == 0)
    {
        if (liczbyDouble[i] >= 0 && liczbyDouble[i] <= 1)
        {
            tab1[0] += 1;
        }
        if (liczbyDouble[i] > 1 && liczbyDouble[i] <= 2)
        {
            tab1[1] += 1;
        }
        if (liczbyDouble[i] > 2 && liczbyDouble[i] <= 3)
        {
            tab1[2] += 1;
        }
        if (liczbyDouble[i] > 3 && liczbyDouble[i] <= 4)
        {
            tab1[3] += 1;
        }
        if (liczbyDouble[i] > 4 && liczbyDouble[i] <= 5)
        {
            tab1[4] += 1;
```

```

        }
    }
    if (click == 1)
    {
        if (liczbyDouble[i] >= 0 && liczbyDouble[i] <= 1)
        {
            tab2[0] += 1;
        }
        if (liczbyDouble[i] > 1 && liczbyDouble[i] <= 2)
        {
            tab2[1] += 1;
        }
        if (liczbyDouble[i] > 2 && liczbyDouble[i] <= 3)
        {
            tab2[2] += 1;
        }
        if (liczbyDouble[i] > 3 && liczbyDouble[i] <= 4)
        {
            tab2[3] += 1;
        }
        if (liczbyDouble[i] > 4 && liczbyDouble[i] <= 5)
        {
            tab2[4] += 1;
        }
    }
    if (click == 2)
    {
        if (liczbyDouble[i] >= 0 && liczbyDouble[i] <= 1)
        {
            tab3[0] += 1;
        }
        if (liczbyDouble[i] > 1 && liczbyDouble[i] <= 2)
        {
            tab3[1] += 1;
        }
        if (liczbyDouble[i] > 2 && liczbyDouble[i] <= 3)
        {
            tab3[2] += 1;
        }
        if (liczbyDouble[i] > 3 && liczbyDouble[i] <= 4)
        {
            tab3[3] += 1;
        }
        if (liczbyDouble[i] > 4 && liczbyDouble[i] <= 5)
        {
            tab3[4] += 1;
        }
        button1.Enabled = false;
    }
}

```

Fragment 4. Zliczenie ilości ocen dla przedziałów 0-1, 1-2, 2-3, 3-4, 4-5

W kolejnej pętli zaczynamy zliczać ilość ocen należących do danego przedziału. Program posługuje się tutaj wcześniej zadeklarowanymi tablicami pięcioelementowymi typu int. Tablica o nazwie *tab1* odpowiada za dane dodane jako pierwsze. Zmienna *click* przechowuje informacje ile danych zostało już dodane. Na samym początku została zadeklarowana jako równą zero dlatego pierwszy warunek

wykona się gdy *click*=0. Indeks zerowy tablicy *tab1* odpowiada przedziałowi 0-1, indeks pierwszy 1-2, indeks drugi 2-3 itd. W zależności od tego w jakim przedziale znajduje się liczba pod indeksem *i* w tablicy *liczbyDouble*, zwiększy się o jeden wartość tablicy *tab1* pod konkretnym indeksem.

Tak samo postępujemy w przypadku dla drugiej i trzeciej serii danych. Program działa wtedy na tablicach *tab2* i *tab3*.

```
click++;
    p01 = tab1[0] + tab2[0] + tab3[0];
    p12 = tab1[1] + tab2[1] + tab3[1];
    p23 = tab1[2] + tab2[2] + tab3[2];
    p34 = tab1[3] + tab2[3] + tab3[3];
    p45 = tab1[4] + tab2[4] + tab3[4];
}
catch
{
    p01 = 0;
    p12 = 0;
    p23 = 0;
    p34 = 0;
    p45 = 0;
    MessageBox.Show("Coś poszło nie tak :c", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
przetworzone = "Rozkład ocen: \n 0-1: " + p01 + " \n 1-2: " + p12 + " \n 2-3: " + p23 + " \n 3-4: " + p34 + " \n 4-5: " +
p45 + "";
textBox1.Text = przetworzone;
```

Fragment 5. Zmienne *pXY* oraz blok *catch*

Po wyjściu z pętli zwiększa się zmienienna *click*, aby program widział ile już zostało wczytyanych danych. Do zmiennych *pXY* (XY- konkretne przedziały 0-1, 1-2 itp.) wpisujemy wartości z tablic. W przypadku gdy gdzieś po drodze w bloku try nie zadziała, program automatycznie zacznie wykonywać instrukcje zapisane w bloku catch. W tym przypadku nie zwiększy się zmienienna *click* (bo żadne dane nie zostaną wczytane), wartości *pXY* zostaną ustalone z powrotem na 0 i wyświetli się komunikat o błędzie. W zmiennej *przetworzone* (stworzonej wcześniej) zapiszemy tekst, który następnie wyświetli się w *textBox1*.

////////// Guziczki serii, ilości danych

```
if (click == 1)
{
    checkBox1.Enabled = true;
    checkBox2.Enabled = false;
    checkBox3.Enabled = false;
}
if (click == 2)
{
    checkBox1.Enabled = true;
    checkBox2.Enabled = true;
    checkBox3.Enabled = false;
}
if (click == 3)
{
    checkBox1.Enabled = true;
    checkBox2.Enabled = true;
    checkBox3.Enabled = true;
```

```
}
```

Fragment 6. Ustawianie dostępności przycisków odpowiadających za ilość danych wyświetlanego na histogramie

Po wyjściu z bloku *try/catch* mamy trzy instrukcje warunkowe, które odpowiadają za to jakie przyciski służące do modyfikacji ilości danych na wykresie będą dostępne dla użytkownika. Jeżeli zostały załadowane tylko jedne dane to *click=1* więc dostępny dla użytkownika będzie tylko *checkBox1*. Jeżeli załadujemy dwa razy dane to *click=2* i dla użytkownika będzie dostępny *checkBox1* oraz *checkBox2*. Analogicznie dla przypadku gdy wczytaliśmy trzy serie danych.

```
private void button4_Click(object sender, EventArgs e)
{
    ///////////////////// Zapisanie do pliku z wynikami i wyświetlenie na ekranie
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Plik tekstowy | *.txt";
    if (sfd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        BinaryWriter bw = new BinaryWriter(File.Create(sfd.FileName));
        bw.Write(przetworzone);
        bw.Dispose();
        MessageBox.Show("Udało się zapisać plik! ", "Sukces", MessageBoxButtons.OK, MessageBoxIcon.Information);
        button4.Enabled = false;
        button1.Enabled = false;
        button3.Enabled = true;
    }
}
```

Fragment 7. Zapisanie danych

Metoda *button4_Click* odpowiada za zapisanie już przygotowanych danych do oddzielnego pliku tekstowego. Używam do tego komponentu *SaveFileDialog* oraz *BinaryWriter*. Program ustawia filtr aby plik mógł zostać zapisany jedynie z rozszerzeniem .txt. Do *bw* wpisujemy wartość przechowaną w zmiennej *przetworzone*. Następnie wyświetla się *messageBox* informujący użytkownika o zapisaniu plik. Na koniec zmienia się dostępność przycisków *button4*, *button1* na niedostępny a *button3* (*Generuj wykres*) zmieni się na dostępny.

```
private void checkBox4_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox4.Checked == false)
    {
        checkBox5.Checked = true;
        checkBox4.Checked = false;
    }
    else
    {
        checkBox5.Checked = false;
        checkBox4.Checked = true;
    }
}

private void checkBox5_CheckedChanged(object sender, EventArgs e)
```

```

{
    if (checkBox5.Checked == false)
    {
        checkBox4.Checked = true;
        checkBox5.Checked = false;
    }
    else
    {
        checkBox4.Checked = false;
        checkBox5.Checked = true;
    }
}

```

Fragment 8. CheckBox'y odpowiadające za opis wykresu

Komponentów *checkbox* może być wiele zaznaczonych lub też żaden. Akurat dla *checkbox'a* 4 i 5, które odpowiadają za włączenie lub wyłączenie opisu histogramu jest potrzebne aby jeden z nich był zawsze odhaczony. Algorytm widoczny na zdjęciu właśnie to umożliwia.

```

private void radioButton1_CheckedChanged(Object sender, EventArgs e)
{
    if (radioButton1.Checked == true)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)
        {
            kolor = colorDialog1.Color.FromArgb();
        }
    }
}

```

Fragment 9. Wybieranie niestandardowego koloru wykresu

W przypadku, gdy użytkownik chce wybrać niestandardowy kolor słupków na wykresie (przyciska *radioButton1*) zostanie wyświetlony komponent *colorDialog*. Użytkownik wybiera kolor a następnie po zatwierdzeniu do zmiennej *kolor* przypisujemy informacje jaki był to kolor.

```

private void button3_Click(object sender, EventArgs e)
{
    //////////////// Przetworzenie danych
    if (checkBox1.Checked == true && checkBox2.Checked == false && checkBox3.Checked == false)
    {
        p01 = tab1[0];
        p12 = tab1[1];
        p23 = tab1[2];
        p34 = tab1[3];
        p45 = tab1[4];
    }
    if (checkBox1.Checked == false && checkBox2.Checked == true && checkBox3.Checked == false)
    {
        p01 = tab2[0];
        p12 = tab2[1];
        p23 = tab2[2];
        p34 = tab2[3];
    }
}

```

```

    p45 = tab2[4];
}
if (checkBox1.Checked == false && checkBox2.Checked == false && checkBox3.Checked == true)
{
    p01 = tab3[0];
    p12 = tab3[1];
    p23 = tab3[2];
    p34 = tab3[3];
    p45 = tab3[4];
}
if (checkBox1.Checked == true && checkBox2.Checked == true && checkBox3.Checked == false)
{
    p01 = tab1[0] + tab2[0];
    p12 = tab1[1] + tab2[1];
    p23 = tab1[2] + tab2[2];
    p34 = tab1[3] + tab2[3];
    p45 = tab1[4] + tab2[4];
}
if (checkBox1.Checked == false && checkBox2.Checked == true && checkBox3.Checked == true)
{
    p01 = tab2[0] + tab3[0];
    p12 = tab2[1] + tab3[1];
    p23 = tab2[2] + tab3[2];
    p34 = tab2[3] + tab3[3];
    p45 = tab2[4] + tab3[4];
}
if (checkBox1.Checked == true && checkBox2.Checked == false && checkBox3.Checked == true)
{
    p01 = tab1[0] + tab3[0];
    p12 = tab1[1] + tab3[1];
    p23 = tab1[2] + tab3[2];
    p34 = tab1[3] + tab3[3];
    p45 = tab1[4] + tab3[4];
}
if (checkBox1.Checked == true && checkBox2.Checked == true && checkBox3.Checked == true)
{
    p01 = tab1[0] + tab2[0] + tab3[0];
    p12 = tab1[1] + tab2[1] + tab3[1];
    p23 = tab1[2] + tab2[2] + tab3[2];
    p34 = tab1[3] + tab2[3] + tab3[3];
    p45 = tab1[4] + tab2[4] + tab3[4];
}

```

Fragment 10. Ustawienia wartości pXY potrzebne do wygenerowania wykresu

Po kliknięciu w button3 (*Generuj wykres*) program sprawdza, które *checkbox'y* odpowiadające za wyświetlenie serii danych są zaznaczone i na tej podstawie do zmiennych *pXY* wpisuje wartości z tablic, które przechowują informacje o ilości ocen produktu z konkretnej serii danych.

```

///Zabezpieczenie aby były zaznaczone chociaż po jednym checkbox'ie
if ((checkBox1.Checked == true || checkBox2.Checked == true || checkBox3.Checked == true) &&
    (checkBox4.Checked == true || checkBox5.Checked == true) &&
    (checkBox6.Checked == true || checkBox7.Checked == true || checkBox8.Checked == true || 
    checkBox9.Checked == true || checkBox10.Checked == true))
{
    Form2 wykres = new Form2();
    wykres.ShowDialog();
}

```

```

        }
        else if (checkBox1.Checked != true && checkBox2.Checked != true && checkBox3.Checked != true)
        {
            MessageBox.Show("Musisz zaznaczyć chociaż jedne dane.", "Zaznacz dane", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
        else if (checkBox4.Checked != true && checkBox5.Checked != true)
        {
            MessageBox.Show("Musisz zaznaczyć czy chesz aby wykres posiadał legendę i opis.", "Zaznacz dane",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("Musisz zaznaczyć jaki zakres ocen ma się wyświetlić.", "Zaznacz dane", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
    }
}

```

Fragment 11. Wyświetlenie histogramu oraz zabezpieczenie checkBox'ów

Po ustawieniu wartości p_{XY} , program sprawdza czy w każdej kategorii (ile danych, jakie przedziały, czy ma być opis wykresu) jest zaznaczony chociaż jeden *checkbox*. Jeżeli warunek jest spełniony zostaje wyświetlony *Form2*, na którym jest histogram. W przeciwnym wypadku wyświetla się stosowny komunikat, w zależności od tego w jakiej kategorii nie został spełniony warunek zaznaczenia.

```

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    DialogResult odp = MessageBox.Show("Czy na pewno chcesz wyjść z
programu?", "Exit", MessageBoxButtons.YesNo, MessageBoxIcon.Hand);
    if(odp == DialogResult.No)
    {
        e.Cancel = true;
    }
}

```

Fragment 12. Zabezpieczenie przed niechcianym zamknięciem programu

Gdy użytkownik zechce zamknąć program, wyświetlony zostanie *MessageBox*, który zażąda potwierdzenia tej czynności. Jeżeli zostanie zaznaczone TAK program się zamknie.

```

public static System.Windows.Forms.CheckBox checkBox6;
public static System.Windows.Forms.CheckBox checkBox7;
public static System.Windows.Forms.CheckBox checkBox8;
public static System.Windows.Forms.CheckBox checkBox9;
public static System.Windows.Forms.CheckBox checkBox10;
public static System.Windows.Forms.RadioButton radioButton1;
public static System.Windows.Forms.CheckBox checkBox1;
public static System.Windows.Forms.CheckBox checkBox2;
public static System.Windows.Forms.CheckBox checkBox3;
public static System.Windows.Forms.CheckBox checkBox4;
public static System.Windows.Forms.CheckBox checkBox5;
public static System.Windows.Forms.RadioButton radioButton8;
public static System.Windows.Forms.RadioButton radioButton9;
public static System.Windows.Forms.RadioButton radioButton10;

```

```
public static System.Windows.Forms.RadioButton radioButton11;
public static System.Windows.Forms.RadioButton radioButton12;
public static System.Windows.Forms.RadioButton radioButton13;
```

Fragment 13. Statyczne komponenty

Aby umożliwić użycie komponentów *checkBox* i *radioButton* w *Form2* zostały one zmienione na static w *Form1.Designer.cs*

```
namespace projekt_sem2
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();

            if (Form1.checkBox6.Checked == true)
            {
                chart1.Series[0].Points.AddXY("0-1", Form1.p01);
            }
            if (Form1.checkBox7.Checked == true)
            {
                chart1.Series[0].Points.AddXY("1-2", Form1.p12);
            }
            if (Form1.checkBox8.Checked == true)
            {
                chart1.Series[0].Points.AddXY("2-3", Form1.p23);
            }
            if (Form1.checkBox9.Checked == true)
            {
                chart1.Series[0].Points.AddXY("3-4", Form1.p34);
            }
            if (Form1.checkBox10.Checked == true)
            {
                chart1.Series[0].Points.AddXY("4-5", Form1.p45);
            }
        }
    }
}
```

Fragment 14. Załadowanie wyników

Przechodzimy teraz do drugiego formularza (*Form2*). Za pomocą instrukcji warunkowych sprawdzamy, które przedziały zostały zaznaczone i mają się wyświetlić na wykresie. Następnie do odpowiednich słupków wpisujemy wartości *pXY*.

```
if (Form1.radioButton8.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(250, 0, 250);
}
if (Form1.radioButton9.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(0, 255, 255);
}
if (Form1.radioButton10.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(255, 215, 0);
}
if (Form1.radioButton11.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(255, 20, 147);
```

```

}
if (Form1.radioButton12.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(220, 20, 60);
}
if (Form1.radioButton13.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(0, 255, 0);
}
if (Form1.radioButton1.Checked == true)
{
    chart1.Series[0].Color = Color.FromArgb(Form1.kolor);
}

```

Fragment 15. Załadowanie kolorów

Program sprawdza, który *checkbox*'y odpowiadający za kolor jest zaznaczony a następnie ustawia ten kolor dla słupków na wykresie.

```

if (Form1.checkBox4.Checked == true)
{
    chart1.Legends[0].Enabled = true;
    chart1.Series[0].IsValueShownAsLabel = true;
    chart1.Titles[0].Visible = true;
}
if (Form1.checkBox5.Checked == true)
{
    chart1.Legends[0].Enabled = false;
    chart1.Series[0].IsValueShownAsLabel = false;
    chart1.Titles[0].Visible = false;
}

```

Fragment 16. Ustawienia legendy

Program sprawdza czy użytkownika zaznaczył, aby opis legendy był włączony. W zależności od rezultatu sprawdzania algorytm wyłącza lub włącza poszczególne właściwości komponentu.

```

private void button1_Click(object sender, EventArgs e)
{
    this.Close();
}

```

Fragment 17. Zamknięcie formularza

Na formularza znajduje się również przycisk Zamknij (*button1*) , który umożliwia zamknięcie *Form2*.

Rezultat:

Histogram

Otwórz plik z danymi

Edytuj wygląd histogramu:

Rodzaj wyświetlanych danych:

dane 1 dane 2 dane 3

Kolor słupków:

Fuksja Limonka
 Aqua Karmazynowy Niestandardowy
 Złoty Różowy

Wyświetl przedziały ocen:

0-1 1-2 2-3 3-4 4-5

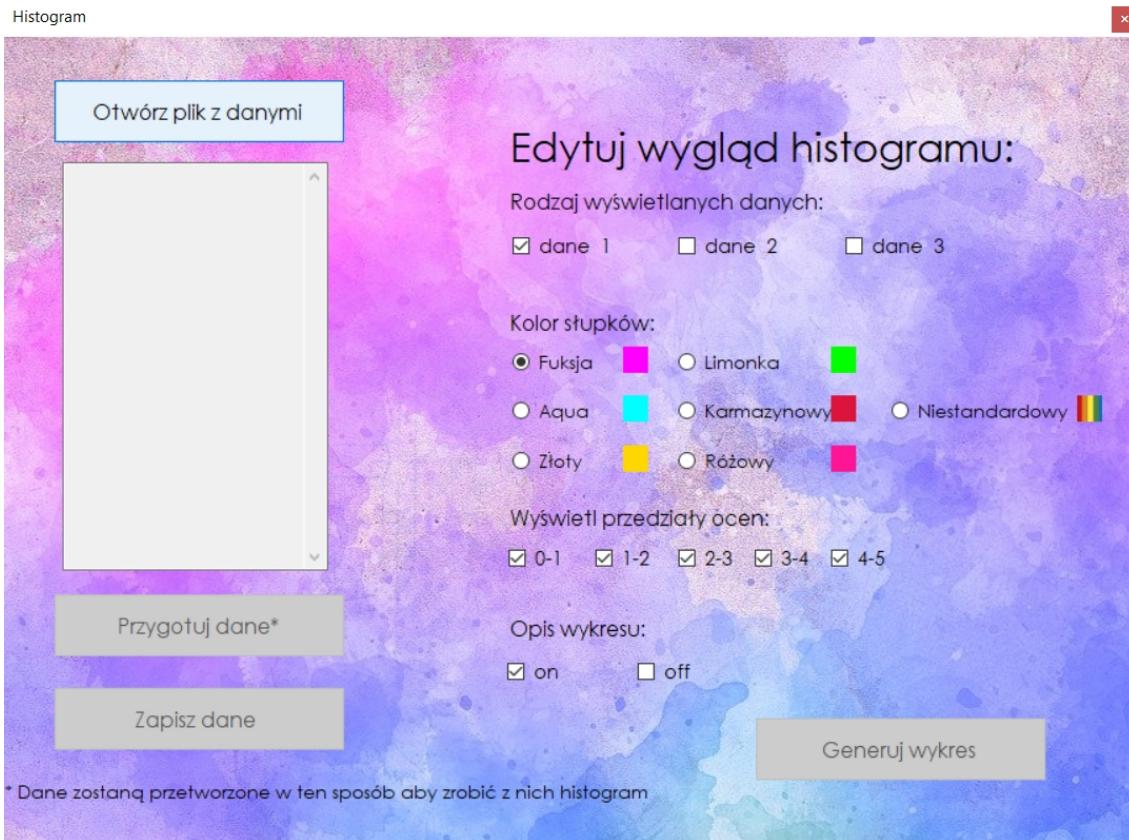
Opis wykresu:

on off

Zapisz dane

Generuj wykres

* Dane zostaną przetworzone w ten sposób aby zrobić z nich histogram



Zdj 1. Program tuż po otwarciu

Histogram

Otwórz plik z danymi

Edytuj wygląd histogramu:

Rodzaj wyświetlanych danych:

dane 1 dane 2 dane 3

Sukces

i Otwarto: oceny_produktu1.txt

X

Limonka Karmazynowy Niestandardowy

OK

wyświetl przedziały ocen:

0-1 1-2 2-3 3-4 4-5

Opis wykresu:

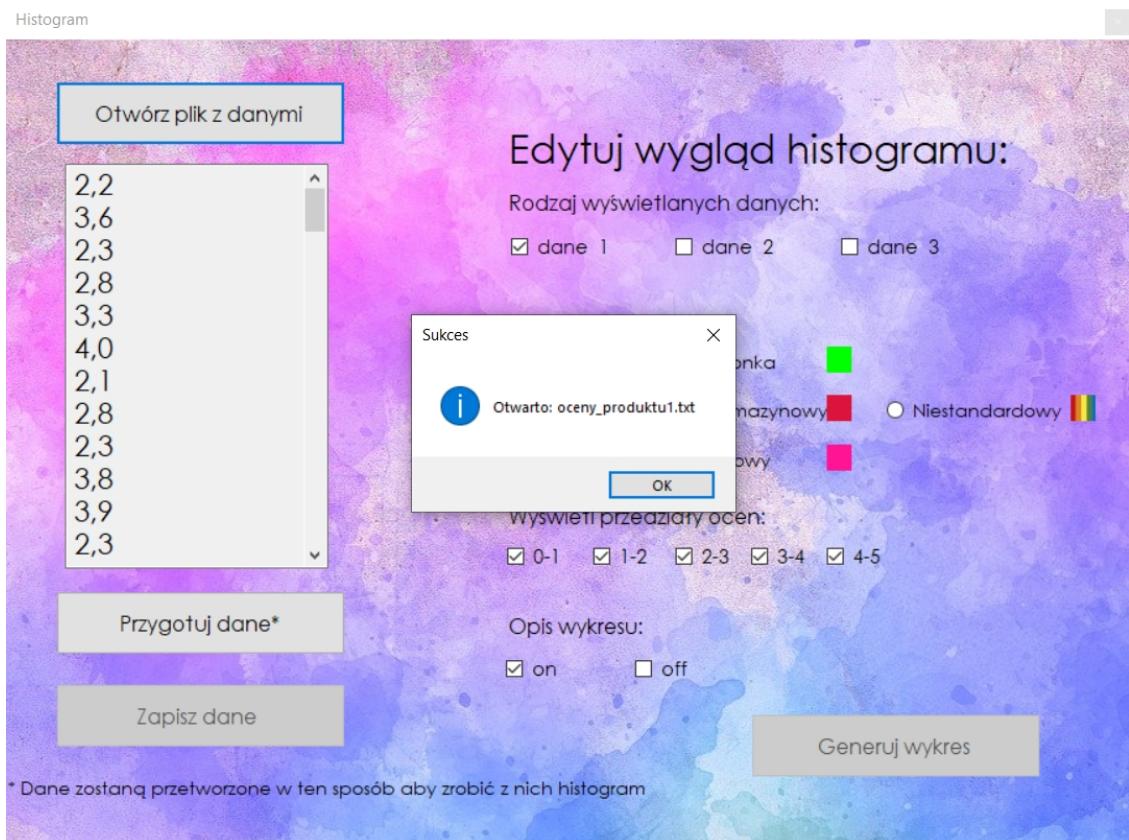
on off

Przygotuj dane*

Zapisz dane

Generuj wykres

* Dane zostaną przetworzone w ten sposób aby zrobić z nich histogram



Zdj 2. Wczytanie pierwszych danych

Otwórz plik z danymi

Rozkład ocen:

0-1: 15 1-2: 16
2-3: 25 3-4: 20
4-5: 24

dane 1 dane 2 dane 3

Kolor słupków:

- Fuksja █ Limonka █
- Aqua █ Karmazynowy █ Niestandardowy ███
- Złoty █ Różowy █

Wyświetl przedziały ocen:

0-1 1-2 2-3 3-4 4-5

Opis wykresu:

on off

Zapisz dane

Generuj wykres

* Dane zostaną przetworzone w ten sposób aby zrobić z nich histogram

Zdj 3. Przygotowanie pierwszych danych

Otwórz plik z danymi

Rozkład ocen:

0-1: 36 1-2: 45
2-3: 56 3-4: 76
4-5: 87

dane 1 dane 2 dane 3

Kolor słupków:

- Fuksja █ Limonka █
- Aqua █ Karmazynowy █ Niestandardowy ███
- Złoty █ Różowy █

Wyświetl przedziały ocen:

0-1 1-2 2-3 3-4 4-5

Opis wykresu:

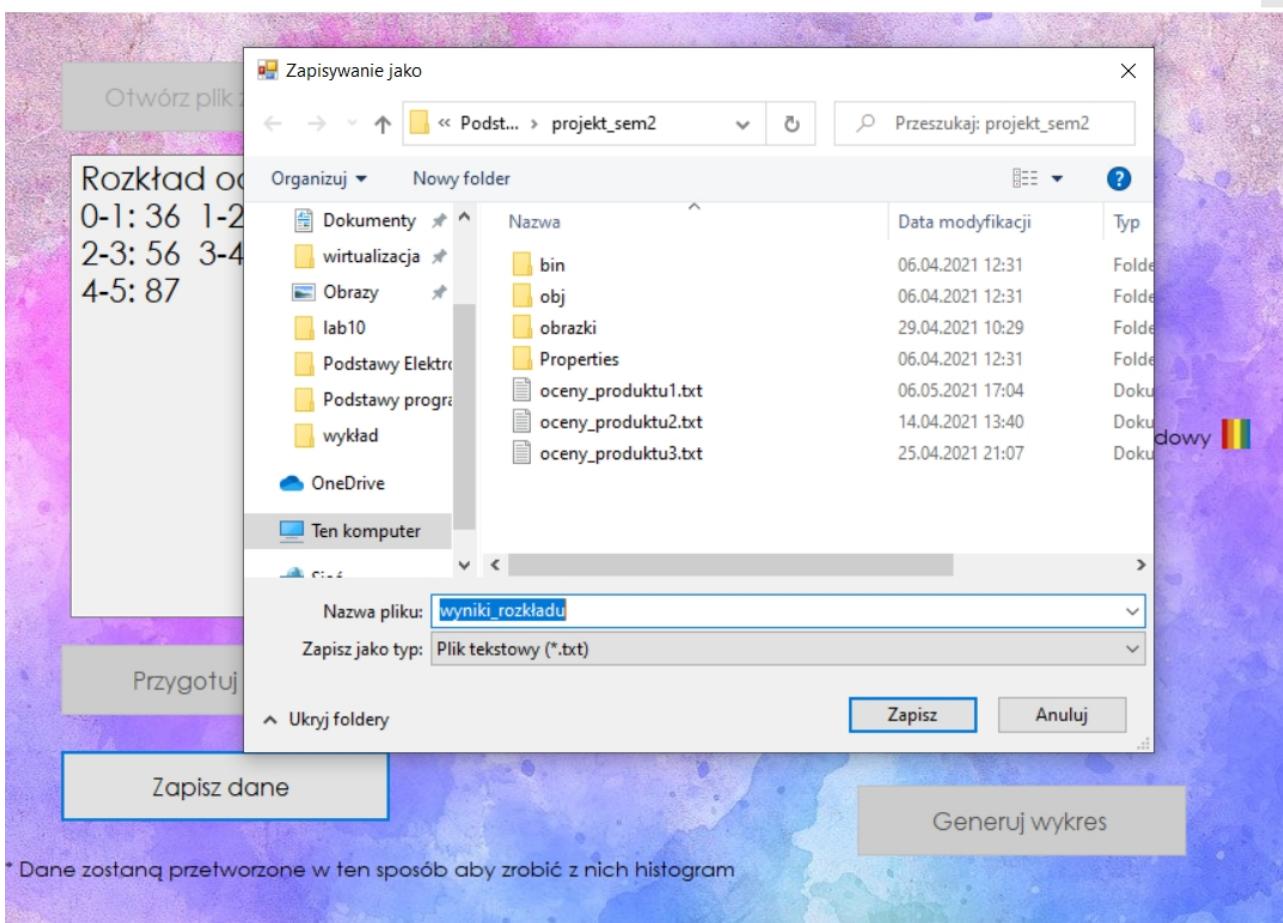
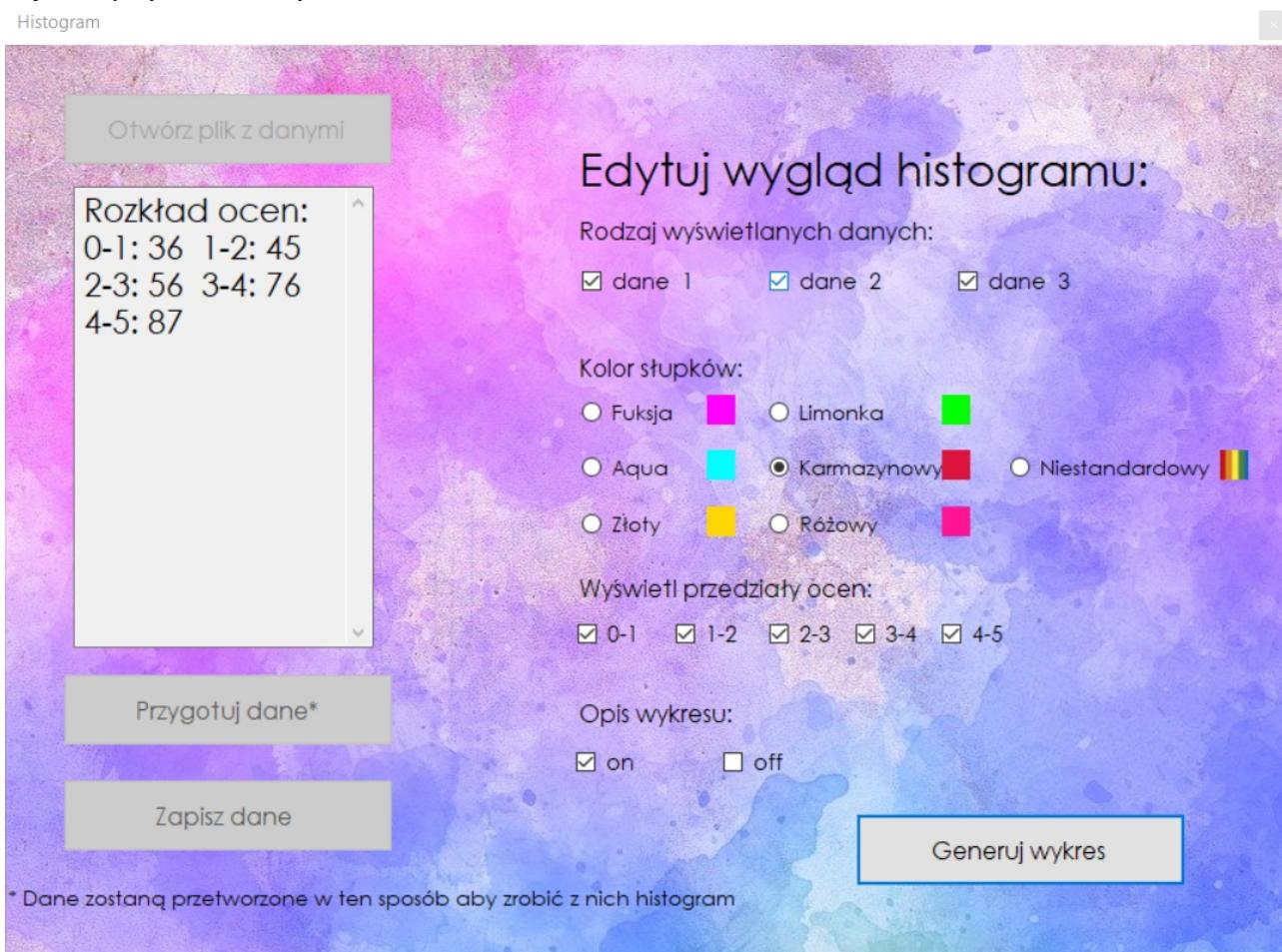
on off

Zapisz dane

Generuj wykres

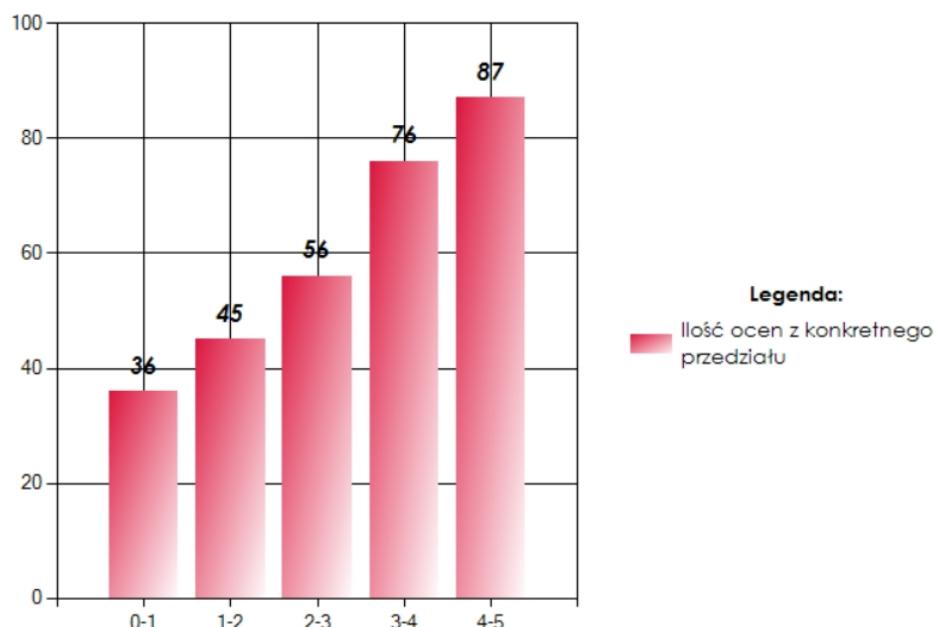
* Dane zostaną przetworzone w ten sposób aby zrobić z nich histogram

Zdj 4. Rezultat dodania pozostałych danych

**Zdj 5. Zapisywanie danych****Zdj 6. Edycja wyglądu histogramu (zmiana koloru na karmazynowy, wybór wszystkich danych)**



Histogram ocen produktu



Zamknij

Zdj 7. Wygląd histogramu po wciśnięciu przycisku Generuj wykres

Histogram



Otwórz plik z danymi

Rozkład ocen:
 0-1: 36 1-2: 45
 2-3: 56 3-4: 76
 4-5: 87

Edytuj wygląd histogramu:

Rodzaj wyświetlanych danych:

 dane 1 dane 2 dane 3

Kolor słupków:

- | | | |
|--|-----------------------------------|--------------------------------------|
| <input type="radio"/> Fuksja | <input type="radio"/> Limonka | <input type="radio"/> Niestandardowy |
| <input type="radio"/> Aqua | <input type="radio"/> Karmazynowy | <input type="radio"/> |
| <input checked="" type="radio"/> Złoty | <input type="radio"/> Różowy | <input type="radio"/> |

Wyświetl przedziały ocen:

 0-1 1-2 2-3 3-4 4-5

Przygotuj dane*

Opis wykresu:

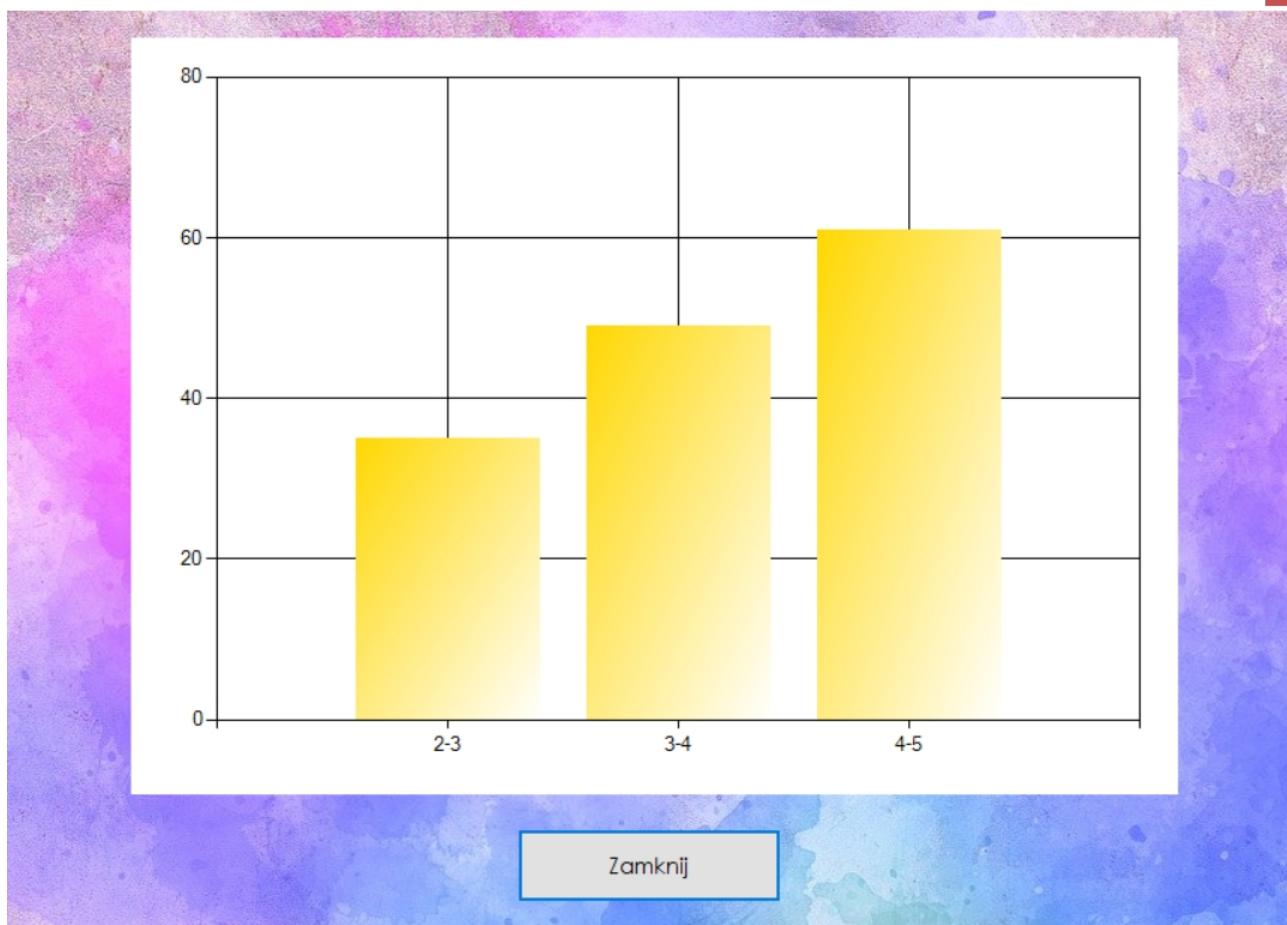
 on off

Zapisz dane

Generuj wykres

* Dane zostaną przetworzone w ten sposób aby zrobić z nich histogram

Zdj 8. Zmiana wyglądu histogramu



Zdj 9. Wygląd wykresu po zmianie wyglądu

Histogram

Otwórz plik z danymi

Rozkład ocen:
0-1: 36 1-2: 45
2-3: 56 3-4: 76
4-5: 87

Edytuj wygląd histogramu:

Rodzaj wyświetlanych danych:

dane 1 dane 2 dane 3

Kolor słupków:

Zaznacz dane

Musisz zaznaczyć chociaż jedne dane.

OK

Limonka Karmazynowy Różowy Niestandardowy

Wybierz zakresy ocen:

0-1 1-2 2-3 3-4 4-5

Przygotuj dane*

Opis wykresu:

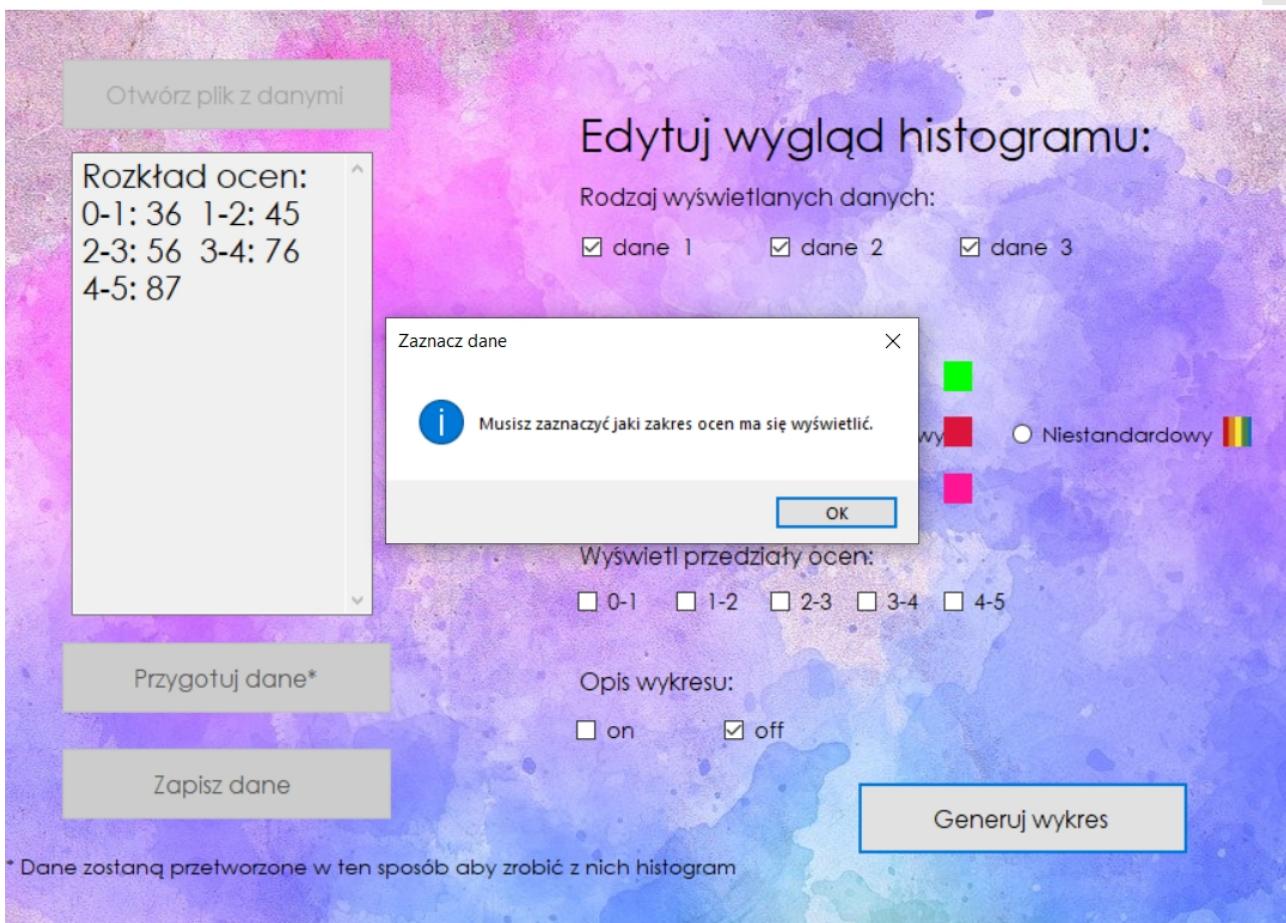
on off

Zapisz dane

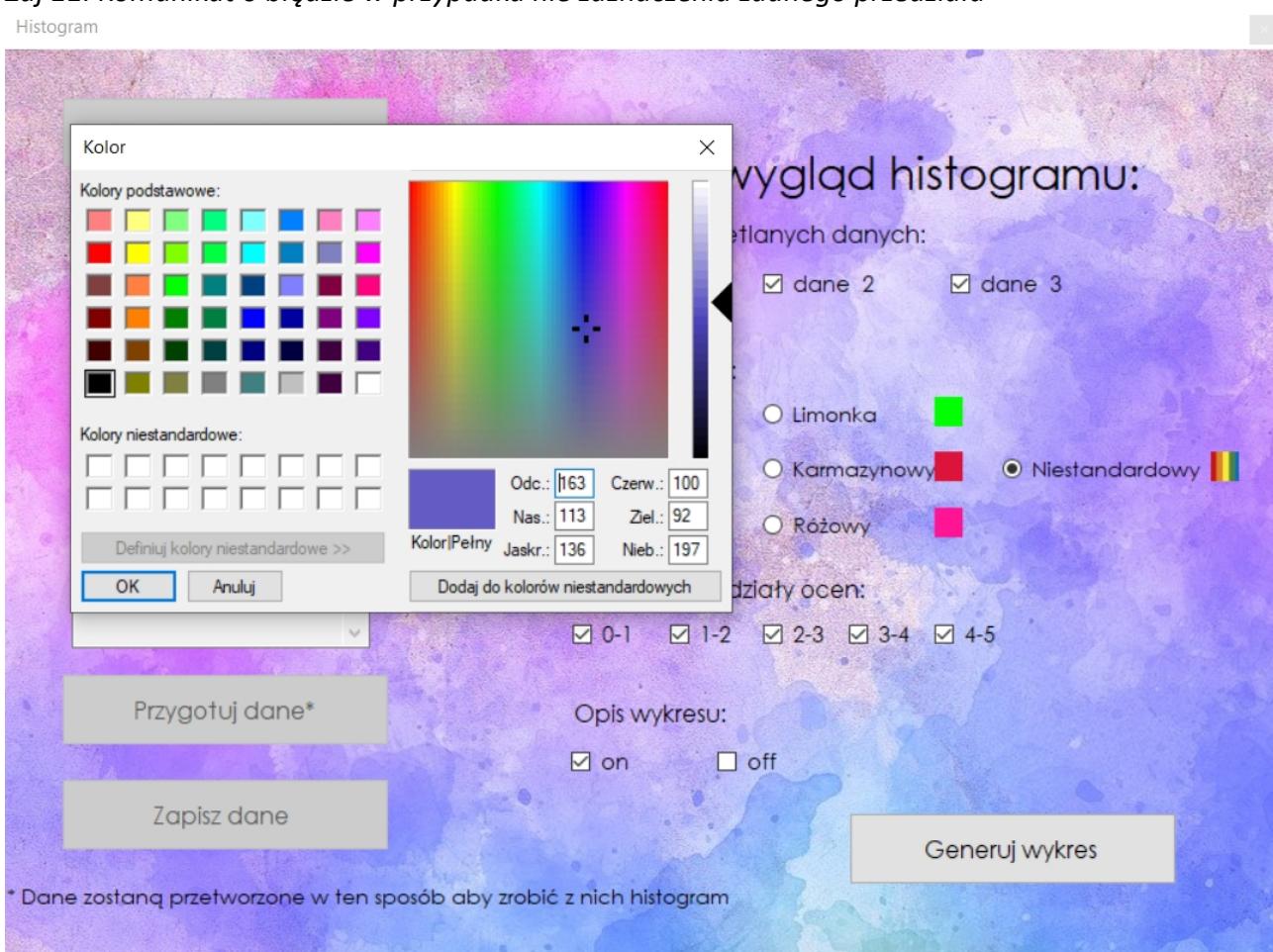
Generuj wykres

* Dane zostaną przetworzone w ten sposób aby zrobić z nich histogram

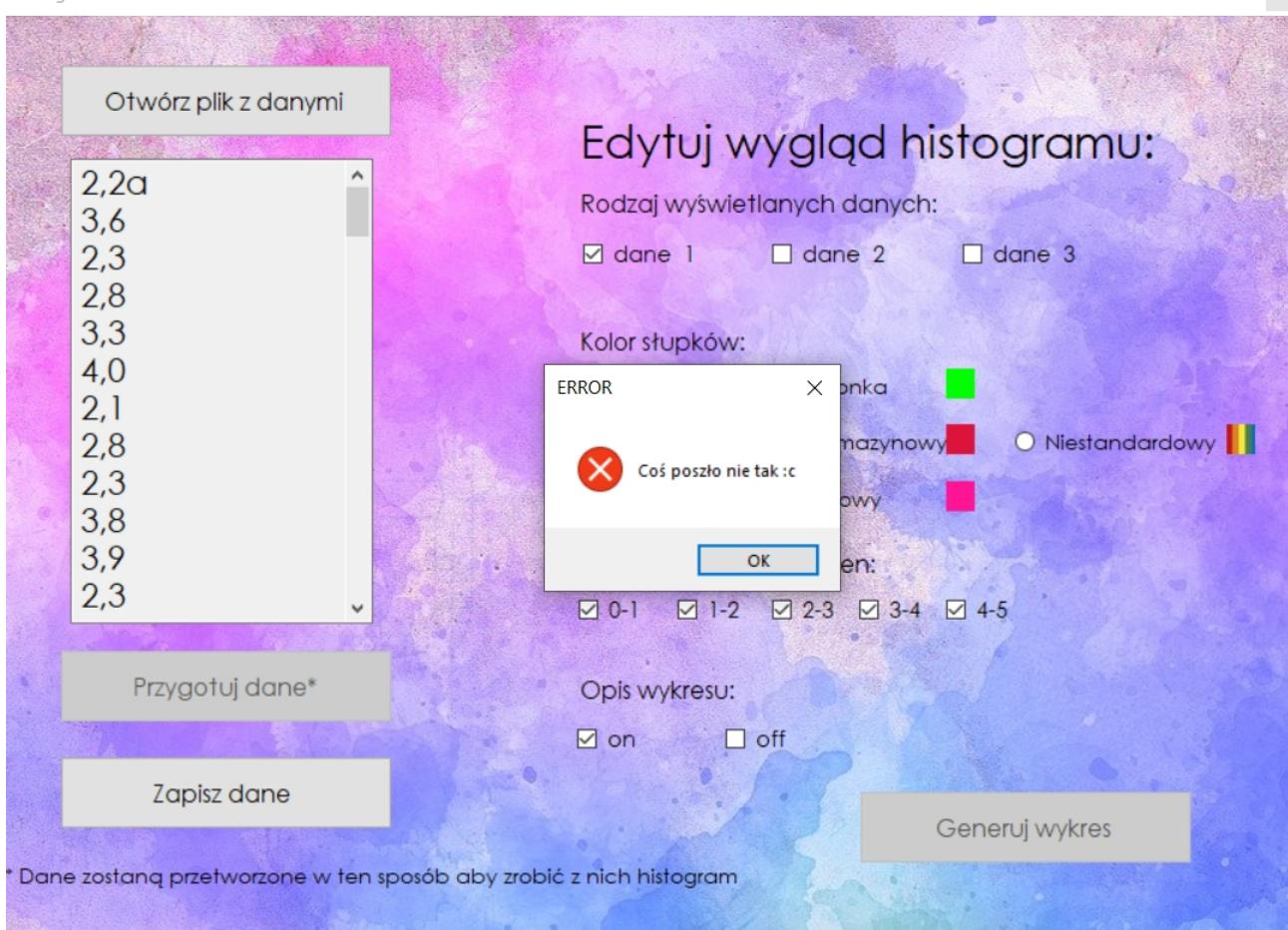
Zdj 10. Komunikat o błędzie w przypadku nie zaznaczenia ani jednych danych



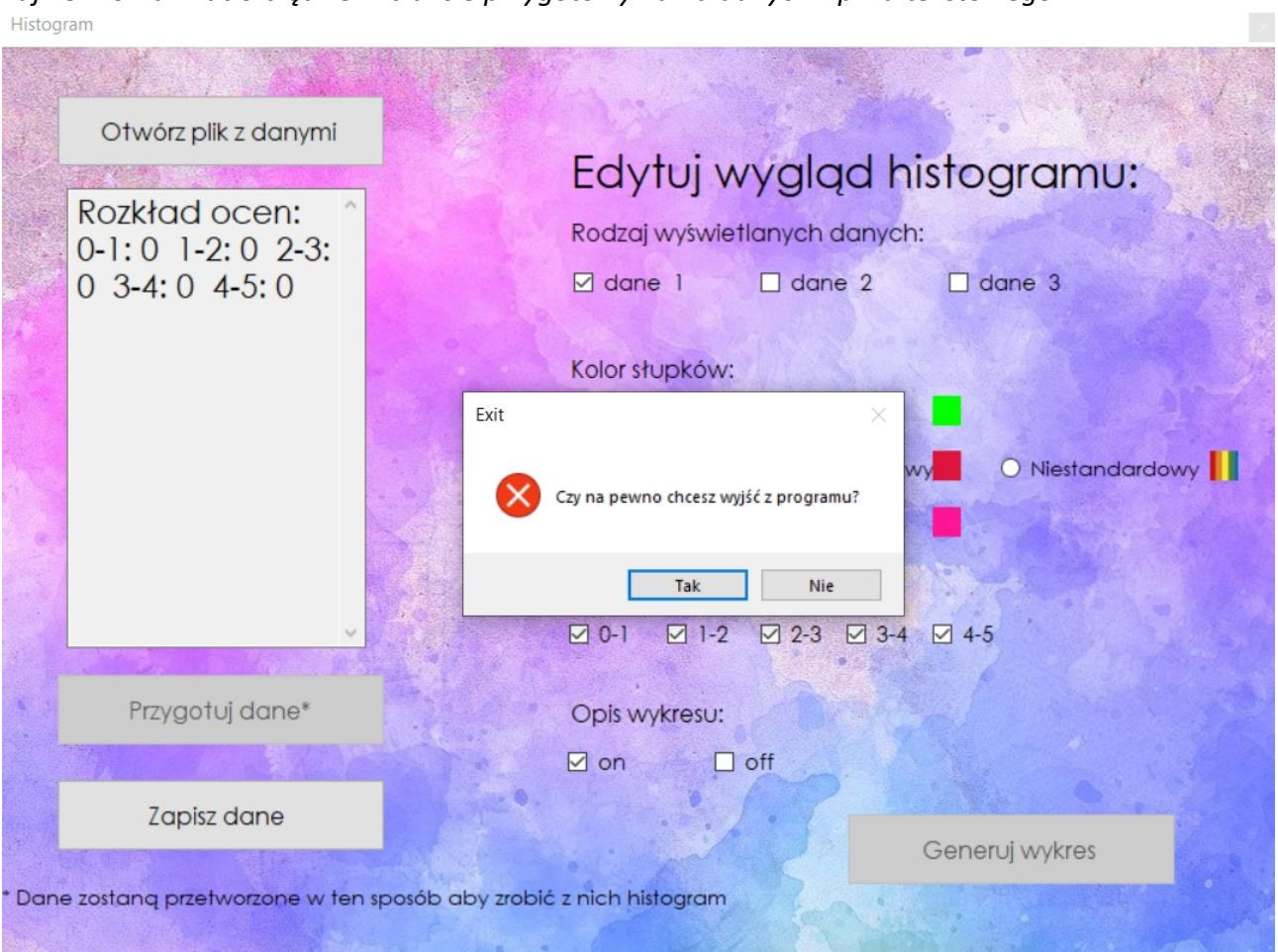
Zdj 11. Komunikat o błędzie w przypadku nie zaznaczenia żadnego przedziału



Zdj 12. Wybór niestandardowego koloru przy edycji wyglądu histogramu



Zdj 13. Komunikat o błędzie w trakcie przygotowywania danych z pliku tekstowego



Zdj 14. Program upewnia się, że użytkownik chce, po kliknięciu w X, wyjść z programu

3.Podsumowanie

Program został napisany przy pomocy kompilatora Visual Studio, obsługującego środowisko .NET Framework, w języku wysokopoziomowym C#.

Aplikacja umożliwia wczytanie danych z trzech plików tekstowych. Wygaszając dostępność przycisku *Otwórz plik z danymi*, uniemożliwiona została możliwość dodania następnych. Wczytane dane zostają przetworzone, a w przypadku błędu użytkownik zostaje powiadomiony poprzez komunikat w *MessageBox'ie*. Dane, z których powstaje wykres są zapisywane do notatnika w dowolne miejsce wskazane przez użytkownika. Umożliwia to klasa *SaveFileDialog*, która wyświetla okienko dialogowe.

Wygląd histogramu jest edytowany w pierwszym formularzu. Dzięki *checkbox'om*, użytkownik może modyfikować liczbę wyświetlanych danych, wyświetlany przedział ocen oraz opis wykresu. Przy wyborze danych oraz przedziałów istnieje możliwość zaznaczenia kilku pól naraz. W przypadku opisu wykresu, *checkbox'y* musiały zostać specjalnie zaprogramowane aby dopuścić zaznaczenia tylko jednego z nich. Jeżeli w konkretnej kategorii żadne pole nie jest zaznaczone a użytkownik wciśnie przycisk *Generuj wykres*, zostaje wyświetlony komunikat w formie *MessageBox'a*, w którym jest informacja, gdzie nie zostało zaznaczone chociaż jedno pole. Do modyfikacji koloru histogramu został użyty komponent *radioButton* umożliwiający wybranie tylko jednego pola. Przy wyborze przez użytkownika pola koloru *niestandardowego* za pomocą klasy *ColorDialog*, wyświetlany jest edytor, gdzie dostępne są standardowe odcienie oraz możliwość wybrania koloru z palety barw.

Tak o to przygotowany histogram, po naciśnięciu przycisku *Generuj wykres*, zostaje wyświetlony w drugim oknie. Na formularzu znajduje się również przycisk *Zamknij*, dzięki czemu możemy wrócić do pierwszego okna i ponownie edytować wygląd wykresu bez powtarzania wszystkich wcześniejszych kroków.

Przy rozpoczęciu procesu zamknięcia programu, naciśnięciu **x** w górnym prawym rogu, ukazuje się *MessageBox*, który pyta czy na pewno chcemy zamknąć program. Po zatwierdzeniu aplikacja się wyłącza, a w przypadku zaznaczenia opcji *nie* następuje powrót do programu.

Podsumowując, program spełnia wszelkie wymogi podane w kryteriach oceniania. Jest wygodny w użytkowaniu, intuicyjny oraz posiada ciekawą szatę graficzną. Wszelkie niepożądane sytuacje wywołane przez użytkownika zostały przewidziane i rozwiązane odpowiednimi algorytmami. Użytkownik nie może wczytać więcej niż trzech danych, dodać kolejne dane uprzednio ich nie porządkując, zapisać wyniki z innym rozszerzeniem niż *.txt* i nie zaznaczyć ani jednego pola w danej kategorii przy modyfikacji wyglądu wykresu oraz zapobiega przypadkowemu zamknięciu programu. Wszystko to przy pomocy odpowiednich algorytmów, metod *try/catch* oraz *MessageBox'ów*.

Literatura:

- ◆ MessageBox
<https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.messagebox?view=net-5.0>
- ◆ Podstawy programowania- Aplikacje GUI
<http://ktm.am.gdynia.pl/~pik/prokom/pliki/pp-w0B.pdf>
- ◆ Zapis i odczyt pliku w C#
http://zuig.el.pcz.czest.pl/jackrat/progob/c_io.htm
- ◆ Działania na plikach
<http://www.cs.put.poznan.pl/anstroinski/data/uploads/mipo/materials/mipo-struktury-i-dzialania-na-plikach-w-c.pdf>
- ◆ Obsługa plików tekstowych
https://4programmers.net/C_sharp/Wprowadzenie/Rozdzia%C5%82_12#obsługa-plikow-tekstowych
- ◆ Dziedziczenie w C#
<http://www.csharpowezmagania.pl/dziedziczenie-w-jezyku-c/>
- ◆ Klasa
OpenFileDialog
<https://docs.microsoft.com/pl-pl/dotnet/api/microsoft.win32.openfiledialog?view=net-5.0>
- ◆ Działania na kilku formularzach
https://www.youtube.com/watch?v=o0X_paV5Eoc
- ◆ Zmiana kolorów na wykresie
<https://stackoverflow.com/questions/26729843/c-sharp-change-color-in-chart>
- ◆ Komponent *chart*
https://www.youtube.com/watch?v=VPHbzv_WRAk&list=WL&index=17
- ◆ Metoda *Close*
<https://docs.microsoft.com/pl-pl/dotnet/api/system.windows.window.close?view=net-5.0>

- ◆ Użycie *ColorDialog*

<https://www.youtube.com/watch?v=loRRYrKj3f0>

- ◆ *ColorDialog C#*

<https://www.dotnetperls.com/colordialog>

- ◆ C# *ColorDialog Box*

<http://csharp.net-informations.com/gui/cs-color-dialog-box.htm>