Przygotowanie zdjęć do legitymacji studenckiej. Zespołowe przedsięwzięcie inżynierskie

Prowadzący: Antoni Ligęza

Spis treści

1
Zespołowe przedsięwzięcie

- Przygotowanie zdjęć do legitymacji studenckiej.
- Projekt jest odpowiedzią na uzasadnioną potrzebę istnienia narzędzia służącego do automatycznej edycji zdjęć, która ułatwi wydruk zdjęć do legitymacji

1.1. Członkowie zespołu z określeniem funkcji

1 Paweł Golonka - kierownik zespołu

2 Karol Liszka - programista C#

3 Bartosz Rusinek - tester aplikacji, autor pomocy, itp

1.2. Uzasadnienie potrzeby realizacji projektu

Projekt ma za zadanie pomóc nowym studentom w przygotowaniu zdjęć do legitymacji czy też do innego użytku. Utworzenie stosownej aplikacji pozwoli na ominięcie kosztów związanych z wykonaniem zdjęcia u profesjonalnego fotografa a także umożliwi łatwe przygotowanie zdjęcia do wysłania w formie elektronicznej. Narzędzie to pozwoli również na tworzenie gotowych bloków zdjęć w wybranej przez użytkownika wielkości.

1.3. Cele projektu

Zespołowe przedsięwzięcie inżynierskie obejmuje przygotowanie aplikacji dla Państwowej Wyższej Szkoły Zawodowej wspomagającej proces przygotowanie fotografii przeznaczonych do legitymacji studenckiej. Program ma pobierać zdjęcie w jednym z wymienionych formatów, a następnie wykadrować wybrany element oraz zapisać go do podanych rozmiarów. W dalszym etapie, jego zadaniem będzie utworzenie bloku zdjęć w oparciu o parametry podane przez użytkownika.

1.4. Grupy docelowe

Odbiorcą aplikacji jest PWSZ w Nowym Sączu - Instytut Techniczny a sama aplikacja dedykowana jest dla studentów.

1.5. Zakres projektu

Etapy, które zostaną zrealizowane aby uzyskać postawiony cel

1. Wywiad ze zleceniodawcą - zapoznanie się z oczekiwaniami co do aplikacji.
2. Przygotowanie środowiska do pracy z dokumentami LaTeXoraz C#.
3. Opracowanie programu w języku C#
4. Opracowanie plików pomocy i szaty graficznej programu.
5. Kompilacja raportów do formatu PDF, utworzonych w LaTeX.

1.6. Struktura podziału prac (zadań) - WBS

Hierarchiczna dekompozycja projektu na zadania i aktywności.

1. Wybranie tematu projektu.
2. Zebranie informacji w wywiadzie ze zleceniodawcą.
   (a) Informacje o celach programu.
   (b) Informacje na temat sposobu działania.
   (c) Informacje na temat wyniku który ma zostać zwrócony.
3. Organizacja pracy
   (a) Wybranie lidera grupy.
   (b) Wstępny podział obowiązków pośród członków grupy.
   (c) Określenie języka programowania oraz środowiska w którym program zostanie zaimplementowany.
4. Przygotowanie środowiska
   (a) Instalacja i konfiguracja środowiska LaTeX oraz edytora Texmaker.
   (b) Utworzenie i konfiguracja repozytorium.
   (c) Instalacja Microsoft Visual Studio 2013.
5. Budowa programu.
   (a) Wybór środowiska .NET, środowiska programistycznego oraz utworzenie w nim projektu.
   (b) Utworzenie interfejsu.
   (c) Zaimplementowanie wybranych metod.
   (d) Zdefiniowanie potrzebnych zmiennych.
   (e) Połączenie metod z elementami graficznymi programu.
   (f) Kompilacja programu.

6. Prace finalne.
   (a) Testowanie programu.
   (b) Poprawianie ewentualnych bugów.
   (c) Testy wtórne.

1.7. Diagram sieciowy

Diagram sieciowy ukazuje zależności czasowe, węzły (aktywności), krawędzie (zależności czasowe).

1.8. Harmonogram

1.8.1. Harmonogram prac poszczególnych członków zespołu

WBS, lub diagram Gantta.

1.9. Dokumentacja

Przygotowanie środowiska do równoległego opracowania dokumentacji projektu i realizacji przydzielonych zadań poszczególnym członkom zespołu projektowego.

1.9.1. Edycja plików dokumentacyjnych - każdy członek zespoły niezależnie

Każdy z członków zespołu edytuje swój plik LaTeX (czlonkowie/nrCzlonka/main.tex) i umieszcza w nim całość analiz i wyników, które pozwoliły mu zrealizować przydzielone zadanie. Wszystkie pliki graficzne, każdy niezależnie umieszcza w swoim katalogu (czlonkowie/nrCzlonka).
Pierwszą linia w pliku (czlonkowie/nrCzlonka/main.tex), zawiera imię i nazwisko opracowującego członka zespołu:

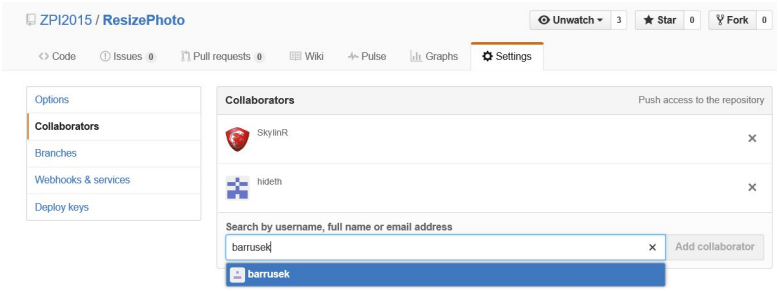
Każde działanie/zadanie należy DOKŁADNIE opisać podając w poleceniu \zadanieprojektowe cztery obowiązkowe dane:

- Rodzaj zadania [Przygotowanie przestrzeni do zespołowej pracy]
- Data rozpoczęcia [2014-11-01]
- Data zakończenia [2014-11-02]
- Aktualny status [zaplanowane do realizacji, w trakcie realizacji, zakończone]
- dokładny opis realizowanego zadania [powinien zawierać opis, rysunki, tabele, kody napisanych programów] Poniżej znajduje się przykładowy listing dla skróconych dwóch zadań:

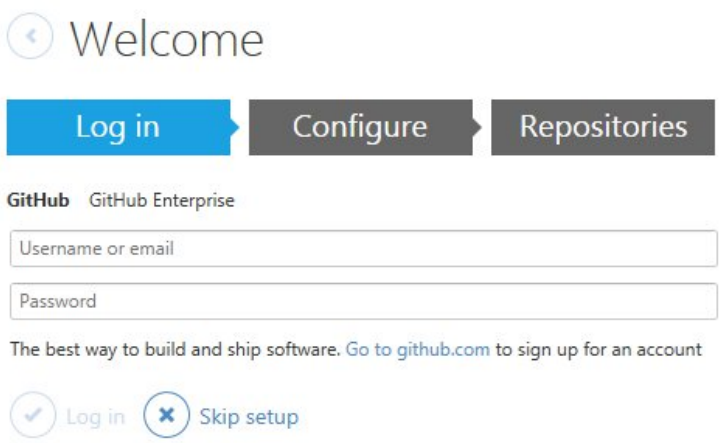
1.9.1.1. Obsługa SVN

Do obsługi repozytorium i wgrywania zmian przez poszczególnych członków zespołu zdecydowaliśmy się użyć rozwiązania Github. Zapewnia ono bezproblemową pracę przy tworzeniu dokumentacji.
Poniżej zamieszczam krótki opis jak wprowadzać aktualizacje na wcześniej utworzonym repozytorium

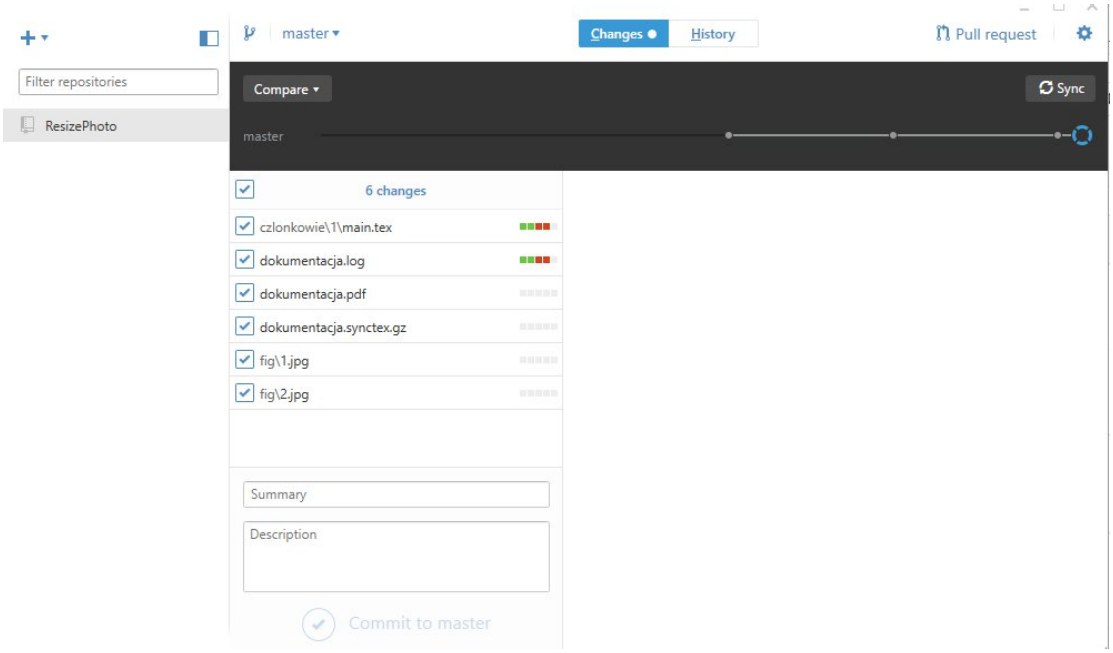
1. Pracę zaczynamy od dodania członków projektu jako współpracowników do repozytorium.



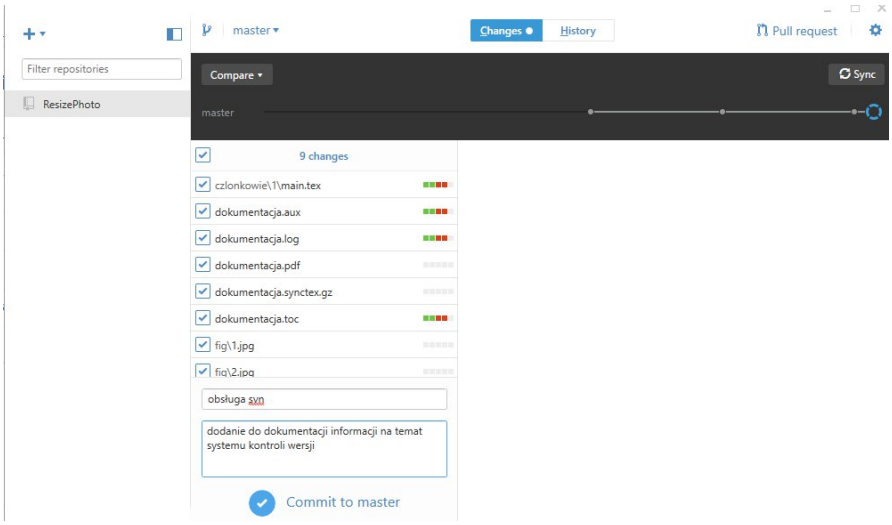
2. Następny krok to pobranie aplikacji klienckiej Github ze strony głównej i zalogowanie się na swoje konto.



3. Po dokonaniu jakichkolwiek zmian w plikach objętych repozytorium, zostają one wyszczególnione w menu głównym programu.



4. Ostatnim etapem jest wypełnienie danych związanych z uaktualnieniem i potwierdzenie commita.



2
Paweł Golonka

sdsd dsds

3
Karol Liszka

3.1. Zadanie projektowe

- **Rodzaj zadania:** Przygotowanie środowiska programistycznego. C#.
  Data rozpoczęcia: 2015.10.20
  Data zakończenia: 2015.10.22
  Aktualny status: Zakończone

- **Rodzaj zadania:** Tworzenie programu.
  Data rozpoczęcia: 2015.10.22
  Data zakończenia: 2015.11.15
  Aktualny status: Zakończone

- **Rodzaj zadania:** Poprawki do programu.
  Data rozpoczęcia: 2015.11.17
  Data zakończenia: 2015.12.01
  Aktualny status: Zakończone

3.2. Etapy budowania programu

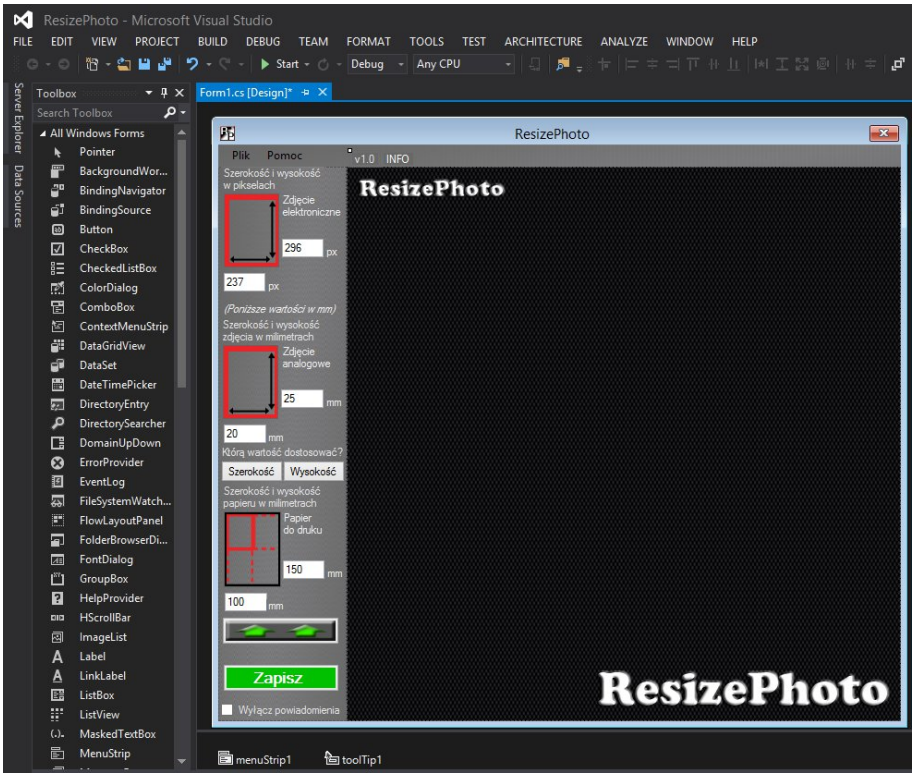
3.2.1. Projektowanie formy

Projekt budowania programu rozpoczął się od zaprojektowania okna z którym będzie miał styczność użytkownik. Okno jest jedynym elementem który użytkownik zobaczy, dla tego bardzo ważne jest aby było zrozumiałe i czytelne.
Poniżej okno programu:

3.2.2. Pisanie głównych funkcjonalności programu w języku C#

- **Funkcjonalność pól tekstowych. Pobieranie rozmiarów wyjściowych grafik. (textBox)**

```
private void TextSzer1_TextChanged(object sender, EventArgs e)
{
bool result1 = Int32.TryParse(TextSzer1.Text, out SzerPiks);
if (result1)
{
TextSzer1.ForeColor = Color.Green;
```



Rys. 3.1. Okno aplikacji

```
SzerPiks = Int32.Parse(TextSzer1.Text);
bool result2 = Int32.TryParse(TextWys1.Text, out WysPiks);
if (result2)
{
WysPiks = Int32.Parse(TextWys1.Text);
}
else TextWys1.Text = "0";
SzerZdj = SzerPiks;
SzerPiksOld = SzerPiks;
WysPiksOld = WysPiks;
konwert = true;

stosunekWH = (float)SzerPiks / (float)WysPiks;

/*SzerMM = 0;
WysMM = 0;
TextWys2.Text = WysMM.ToString();
TextSzer2.Text = SzerMM.ToString();*/

pictureBox1.Refresh();
if (SzerPiks > 0 && WysPiks> 0 && loaded)
{
WcisnietyX = 10;
WcisnietyY = 10;
Graphics g = pictureBox1.CreateGraphics();
Graphics g2 = pictureBox1.CreateGraphics();
Pen pen = new Pen(Color.Red, 2);
Pen pen2 = new Pen(Color.Yellow, 1);
```

```
MainRect.X = 10;
MainRect.Y = 10;

if (SzerPiksOld <= 550 && WysPiksOld <= 550)
{
MainRect.Width = (int)SzerPiksOld;
MainRect.Height = (int)WysPiksOld;
}
else if (SzerPiksOld > 550 || WysPiksOld > 550 &&
(SzerPiksOld < 1100 && WysPiksOld < 1100))
{
MainRect.Width = (int)(SzerPiksOld / 2);
MainRect.Height = (int)(WysPiksOld / 2);
}
else if (SzerPiksOld > 1100 || WysPiksOld > 1100)
{
MainRect.Width = (int)(SzerPiksOld / 4);
MainRect.Height = (int)(WysPiksOld / 4);
}

pictureBox1.Refresh();
g.DrawRectangle(pen, MainRect);



MiniRect.X = MainRect.X + MainRect.Width - 6;
MiniRect.Y = MainRect.Y + MainRect.Height - 6;
MiniRect.Width = 6;
MiniRect.Height = 6;


SolidBrush brush1 = new System.Drawing.SolidBrush
(System.Drawing.Color.Yellow);
g2.FillRectangle(brush1, MiniRect);
brush1.Dispose();
pen.Dispose();
g.Dispose();
g2.Dispose();
pen2.Dispose();


}

}
else
{
TextSzer1.ForeColor = Color.Red;
konwert = false;
}
labelInfo.Text =
"RZ:"+RealW.ToString()+"x"+RealH.ToString()+"|"+
"RZE:"+MainRect.Width.ToString()+"x"+MainRect.Height.ToString()+"␣"+
"PB:"+pictureBox1.Width.ToString()+"x"+pictureBox1.Height.ToString() +"|"+
"KOMP:"+pictureBox5.Width.ToString()+"x"+pictureBox5.Height.ToString()+"|"+
"RATIO:"+stosunekWH.ToString();
}
```

Powyższy kod służy do odczytywania w polu tekstowym od użytkownika Szerokości zdjęcia cyfrowego podawanej w pikselach. Na początku kod sprawdza czy wartość podana przez użytkownika jest cyfrą, następnie jeśli tak wykonuje szereg operacji przypisania tej wartości do odpowiednich zmiennych w celu wykorzystania ich w dalszej części kodu. Następnie odświeża obszar gdzie wyświetlana jest grafika, w celu narysowania nowego prostokąta do wycinania zdjęcia. Pozostała

część kodu to kosmetyczne ulepszenia, jak przykładowo zmniejszenie prostokąta do wycinania, jeśli jego rozmiar jest większy niż rozmiar okna aplikacji, tak aby zmieścił się w oknie. Na samym końcu aktualizujemy pasek stanu o zedytowane wartości// Pozostale pola tekstowe są zbudowane bardzo podobnie z małymi zmianami zależnymi od tego jakie informacje przyjmują.

- **Załadowanie grafiki do programu.**

```
        private void LoadPhoto(object sender, EventArgs e)
        {
        OpenFileDialog ofd = new OpenFileDialog();
        ofd.Filter = "jpg(*.jpg)|*.jpg|bmp (*.bmp)|*.bmp|
          png (*.png)|*.png|gif (*.gif)|*.gif";

        if (ofd.ShowDialog() == DialogResult.OK && ofd.FileName.Length > 0)
        {
        pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
        zaladowaneZdj = Image.FromFile(ofd.FileName);
        pictureBox1.Image = zaladowaneZdj;
        loaded = true;
        RealW = zaladowaneZdj.Width;
        RealH = zaladowaneZdj.Height;
        }

        labelInfo.Text =
        "RZ:"+RealW.ToString()+"x"+RealH.ToString()+"|"+
        "RZE:"+MainRect.Width.ToString()+"x"+MainRect.Height.ToString()+" "+
        "PB:"+pictureBox1.Width.ToString()+"x"+pictureBox1.Height.ToString() +"|"+
        "KOMP:"+pictureBox5.Width.ToString()+"x"+pictureBox5.Height.ToString()+"|"+
        "RATIO:"+stosunekWH.ToString();

        #region SKALOWANIE DO ROZMIARU ZDJECIA

        if (RealW > RealH)
        {
        int i;
        this.Height = 600+63;
        i= (int)((zaladowaneZdj.Width * 600) / zaladowaneZdj.Height);
        this.Width = i+155;
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
        firstklik = true;
        }
        else if (RealW < RealH)
        {
        int i;
        this.Height = 600+63;
        i= (int)((zaladowaneZdj.Width * 600) / zaladowaneZdj.Height);
        this.Width = i+155;
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
        firstklik = true;
        }
        else
        {
        this.Height = 600 + 63;
        this.Width = 600 + 155;
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
        firstklik = true;
        }
        #endregion
```

Powyższa część kodu odpowiada za przycisk do wczytywania zdjęć i ładowania do obiektu zwanego pictureBox. Na początku kodu otwieramy okno dialogowe ustalając rozszerzenie plików możliwych do wczytania (w obecnym przypadku konieczne

jest ustawienie filtru na pliki graficzne z rozszerzeniem takim jak bmp, jpg, png czy gif). Następnie kod ładuje wybrana grafikę do pictureBox pobierając przy tym jej prawdziwy wymiar. Na końcu kod aktualizuje pasek stanu o zmienione wartości oraz skaluje rozmiar okna odpowiednio do grafiki tak aby wypełniała ona całego pictureBox'a. Dodatkowo w kodzie został stworzony blok kodu za pomocą słowa kluczowego #region i #endregion. Jest to część typowo kosmetyczna ułatwiająca odczyt i edycję kodu.

- **Rysowanie prostokąta do wycinania grafiki (Rectangle)**

```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
currentPosMove.X = e.X;
currentPosMove.Y = e.Y;
if (MiniRect.Contains(currentPosMove) | skalowanie)
Cursor = Cursors.SizeNWSE;
else if (MiniRect.Contains(currentPosMove) == false &&
(MainRect.Contains(currentPosMove) | przeciaganie))
Cursor = Cursors.Hand;
else
Cursor = Cursors.Default;

Graphics g = pictureBox1.CreateGraphics();
Graphics g2= pictureBox1.CreateGraphics();
Pen pen = new Pen(Color.Red, 2);
Pen pen2 = new Pen(Color.Yellow, 1);
labelInfo.Text =
"RZ:"+RealW.ToString()+"x"+RealH.ToString()+"|"+
"RZE:"+MainRect.Width.ToString()+"x"+MainRect.Height.ToString()+"␣"+
"PB:"+pictureBox1.Width.ToString()+"x"+pictureBox1.Height.ToString() +"|"+
"KOMP:"+pictureBox5.Width.ToString()+"x"+pictureBox5.Height.ToString()+"|"+
"RATIO:"+stosunekWH.ToString();

if ((firstklik||isDown) && (przeciaganie || firstklik)
 && SzerPiks > 0 && WysPiks > 0&&loaded)
{

korner[0].X = e.X - (WcisnietyX - oldX);
korner[0].Y = e.Y - (WcisnietyY- oldY);

korner[1].X = e.X + ((WcisnietyX - oldX) + MainRect.Width);
korner[1].Y = e.Y - (WcisnietyY - oldY);

korner[2].X = e.X + ((WcisnietyX - oldX) + MainRect.Width);
korner[2].Y = e.Y + ((WcisnietyY - oldY) + MainRect.Height);
pictureBox1.Refresh();

korner[2].Y - korner[1].Y);
MainRect.X = e.X - (WcisnietyX - oldX);
MainRect.Y = e.Y - (WcisnietyY - oldY);
if (SzerPiksOld <= 550 && WysPiksOld <= 550)
{
MainRect.Width = SzerPiks;
MainRect.Height = WysPiks;
}
else if (SzerPiksOld > 550 || WysPiksOld > 550 &&
(SzerPiksOld < 1100 && WysPiksOld < 1100))
{
MainRect.Width = SzerPiks / 2;
MainRect.Height = WysPiks / 2;
}
else if (SzerPiksOld > 1100 || WysPiksOld > 1100)
{
MainRect.Width = SzerPiks / 4;
```

```
MainRect.Height = WysPiks / 4;
}
g.DrawRectangle(pen, MainRect);


PozX = korner[1].X - korner[0].X;
PozY = korner[2].Y - korner[1].Y;

MiniRect.X = MainRect.X + MainRect.Width - 6;
MiniRect.Y = MainRect.Y + MainRect.Height - 6;
MiniRect.Width = 6;
MiniRect.Height = 6;
SolidBrush brush1 = new System.Drawing.SolidBrush
(System.Drawing.Color.Yellow);
g2.FillRectangle(brush1, MiniRect);
MiniRect.Width = 10;
MiniRect.Height = 10;
brush1.Dispose();

pen.Dispose();
g.Dispose();
g2.Dispose();
pen2.Dispose();

}

if (isDown && skalowanie&&loaded)
{

pictureBox1.Refresh();

MainRect.X = korner[0].X;
MainRect.Y = korner[0].Y;
if (SzerPiksOld <= 550 && WysPiksOld <= 550)
{
MainRect.Width = SzerPiks;
MainRect.Height = WysPiks;
}
else if (SzerPiksOld > 550 || WysPiksOld > 550 &&
(SzerPiksOld < 1100 && WysPiksOld < 1100))
{
MainRect.Width = SzerPiks / 2;
MainRect.Height = WysPiks / 2;
}
else if (SzerPiksOld > 1100 || WysPiksOld > 1100)
{
MainRect.Width = SzerPiks / 4;
MainRect.Height = WysPiks / 4;
}
g.DrawRectangle(pen, MainRect);

if (e.X > PosW + 2 || e.Y > PosH + 2)
{
if (e.X > PosW + 2)
{
PosW = e.X;
SzerPiks += (int)(SzerPiksOld * 0.02);
PosH = e.Y;
WysPiks += (int)(WysPiksOld * 0.02);
}
if (e.Y > PosH + 2)
{
PosW = e.X;
SzerPiks += (int)(SzerPiksOld * 0.02);
PosH = e.Y;
WysPiks += (int)(WysPiksOld * 0.02);
```

```
}
}
else
{
if (e.X < PosW - 2)
{
PosW = e.X;
SzerPiks -= (int)(SzerPiksOld * 0.02);
PosH = e.Y;
WysPiks -= (int)(WysPiksOld * 0.02);
}
if (e.Y < PosH - 2)
{
PosW = e.X;
SzerPiks -= (int)(SzerPiksOld * 0.02);
PosH = e.Y;
WysPiks -= (int)(WysPiksOld * 0.02);
}
}
pen.Dispose();
g.Dispose();
}
}
```

Powyższy kod służy do rysowania czerwonego kwadratu (Rectangle) do wycinania kawałka grafiki, jednak podkreślam, że nie jest to jedyna funkcja która służy do rysowania rectangle. Zasługuje ona jednak na wyróżnienie ponieważ rysowanie rectangle głównie opiera się na tej funkcji.
Funkcja MouseMove odpowiada za poruszanie myszką w obiekcie pictureBox. Podczas wcisniętego prawego klawisza myszki poruszamy rectangle co oznacza ze cały czas jest rysowany na nowo w zaktualizowanych pozycjach, tak samo jest ze skalowaniem, tylko tutaj zamiast zmieniać pozycję zmieniamy rozmiar. Piewsza instrukcja warunkowa if sprawdza pozycję kursora i w zalezności od niej ustawia wygląd kursora do jego obecnej funkcjonalności. Następnie w kodzie tworzymy nowe obiekty niezbędne do rysowania takie jak Pen. Dalej aktualizujemy pasek stanu o zaktualizowane wartości. Kolejna instrukcja warunkowa sprawdza czy grafika jest załadowana i czy użytkownik kliknął w odpowiednim miejscu aby przesunąć rectangle. Jeśli warunki są spełnione czerwony prostokąt z powodzeniem się przemieszcza. Jeszcze następna instrukcja warunkowa sprawdza czy kursor jest w prawy dolnym rogu rectangle i czy jest wciśnięty prawy przycisk myszy. Jeśli warunki są spełnione skalowanie czerwonego prostokąta do wycinania jest możliwe. Dodatkowo w tej funkcji wykonywane są różnorakie obliczenia polegające na dopasowaniu stosunku podanego przez użytkownika do narysowanego rectangle w programie bądź też obliczaniu odległości myszki od początku narysowanego rectangle w celu dogodnego przesuwania prostokąta bez względu na to gdzie klikniemy myszką (z wykluczeniem punktu do skalowania).

- **Wycinanie elementu wskazanego przez Rectangle**

```
public Bitmap CropImage(Bitmap source, Rectangle section)
{
Bitmap bmp = new Bitmap(section.Width, section.Height);
Graphics g = Graphics.FromImage(bmp);
g.DrawImage(source, 0, 0, section, GraphicsUnit.Pixel);
return bmp;
}
```

Powyższa funkcja wycina element grafiki znajdujący się wewnątrz narysowanego rectangle.Na początku funkcja tworzy nowy obiekt typu Bitmap, o wymiarach narysowanego przez użytkownika rectangle. następnie wypełnia ten obiekt elementem grafiki na który wskazuje rectangle poprzez swoją pozycję oraz rozmiar. Na końcu funkcja zwraca grafikę w postaci bitmapy.

- **Zapis grafiki do pliku z rozszerzeniem graficznym.**

```
        private void SavePhotoToolStripMenuItem_Click(object sender, EventArgs e)
        {
        if (loaded)
        {
        SaveFileDialog sfd = new SaveFileDialog();
        sfd.Filter = "jpg (*.jpg)|*.jpg|bmp (*.bmp)|*.bmp| png(*.png)|*.png|
        gif (*.gif)|*.gif";
        if (sfd.ShowDialog() == DialogResult.OK && sfd.FileName.Length > 0)
        {
        grafika=CropImage(new Bitmap(pictureBox1.Image, pictureBox1.Size),MainRect);
        Bitmap Crop1 = new Bitmap(grafika, new Size(SzerZdj, WysZdj));
        #region zapisywanie cyfrowego
        try
        {
        Crop1.SetResolution(72.0f, 72.0f);
        Crop1.Save(sfd.FileName, ImageFormat.Jpeg);
        if (!powiadomienia)
        {
        MessageBox.Show("Zdjecie cyfrowe zostalo zapisane.", "Zapis",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
        }}
        catch (Exception)
        {
        MessageBox.Show("Zmien nazwe zapisywanego pliku.", "Blad01",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        }}}
        #endregion
        }
```

Podobnie jak w poprzednich przykładach, również tutaj podam jedną z przykładowych funkcji które wchodzą w skład programu. Funkcje działaja bardzo podobnie, różniąc się głównie wymiarami grafik jakie mają zapisać. Powyższa funkcja odpowiada za zapisanie pliku do zdjęcia cyfrowego(dwie pierwsze wartości podane w programie). Na początku oczywiście otwiera okno dialogowe w celu wskazania miejsca zapisu pliku oraz nazwy jaką chcemy nadać. Dzięki filtrowi mamy pewność, że zapisany plik będzie miał rozszerzenie zgodne z oczekiwaniami. Następnie jest wywoływana funkcja do wycinania grafiki. Wycięta grafika jest zapisywana do pliku za pomocą funkcji Save(). Jeśli zdjęcie zostało zapisane z powodzeniem, wyświeli się komunikat, jeśli jednak wystąpił błąd, program zwróci powiadomienie o błędzie i dodatkowo wyświeli wskazówkę jak ominąc błąd.

3.2.3. Czego nauczyłem się indywidualnie

- Wykorzystywać obiekt pictureBox do przechowywania grafiki,
- Używać część możliwości biblioteki drawing,
- Za pomocą Rectangle wycinać elementy graficzne,
- Wyłapywać błędy w trakcie działania programu i zwracać komunikaty o nich,

- Tworzyć dokumentację tworzonego programu,
- Utrzymywać kod w ładzie i porządku aby był czytelny i łatwy do dalszej edycji,
- Dostosowywać się do potrzeb i wymagań zleceniodawcy.

3.2.4. Czego nauczyłem się zespołowo

- Współpracować w zespole projektowym,
- Dotrzymywać terminów prowadzenia prac projektowych,
- Wykonywania zadan zleconych przez menadżera projektu.

4
Bartosz Rusinek

Bibliografia

[1] Balcerzak J., Pansiuk J.: *Wprowadzenie do kartografii matematycznej*, Warszawa, OWPW 2005.

[2] Barrett R. i inni: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods1*, wersja elektorniczna Mathematics http://www.siam.org/books.

[3] Bjork A., Dahlquist G.: *Numerical Methods in Scientific Computing*, Philadelphia, SIAM 2002.

[4] CCITT, *Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, Recommendation T.6, Volume VII, Fascicle VII.3, Terminal Equipment and Protocols for Telematic Services, The International Telegraph and Telephone Consultative Committee (CCITT)*, Geneva, CCITT 1985.

[5] Drwal G, i in., *Mathematica 4*, Gliwice, WPKJS 200.

[6] Gdowski B.: *Elementy geometrii różniczkowej w zadaniach*, Warszawa, PWN 1982.

[7] Januszewski J.: *Systemy satelitarne GPS, Galileo i inne*, Warszawa, PWN 2006.

[8] : Kiełbasiński A., Schwetlick H.: *Numeryczna algebra liniowa*, Warszawa, WNT 1992.

[9] Kincaid D.: *Analiza numeryczna*, Warszawa, WNT 2006.

[10] Levine J.: *Programowanie plików graficznych w C/C++*, New York, Wiley 1994.

[11] Longley P. i inni: *GIS teoria i praktyka*, Warszawa, PWN 2006.

[12] Open Geospatial Consortium Inc.: *OpenGIS Geography Markup Language (GML) Encoding Standard, Version: 3.2.1*, OGC 2007.

[13] Open Geospatial Consortium Inc.: *OpenGIS® Geography Markup Language (GML) Implementation Specification*, OGC 2004.

[14] Opera J.: *Geometria róniczkowa i jej zastosowania*, Warszawa, PWN 2002.

[15] Odlanicki-Poczobut M.: *Geodezja*, PPWK 1982.

[16] Li Y. i inni: *GML Topology Data Storage Schema Design*, Chiba University 2007.

[17] Li Y., Li J., Zhou S.: *GML Storage*, A Spatial Database Approach,ER (Workshops), str 55-66, 2004.

[18] Sayood K.: *Kompresja danych*, Warszawa, Rm 2002.

[19] *The Technical Instruction G-5, The Ground Cadastre and Buildings, The Main Surveying and Cartographic Bureau*, Warszawa 2003.