

Sprawozdanie z zadania 1 - Kalkulator z zegarem

1) Założenie projektu

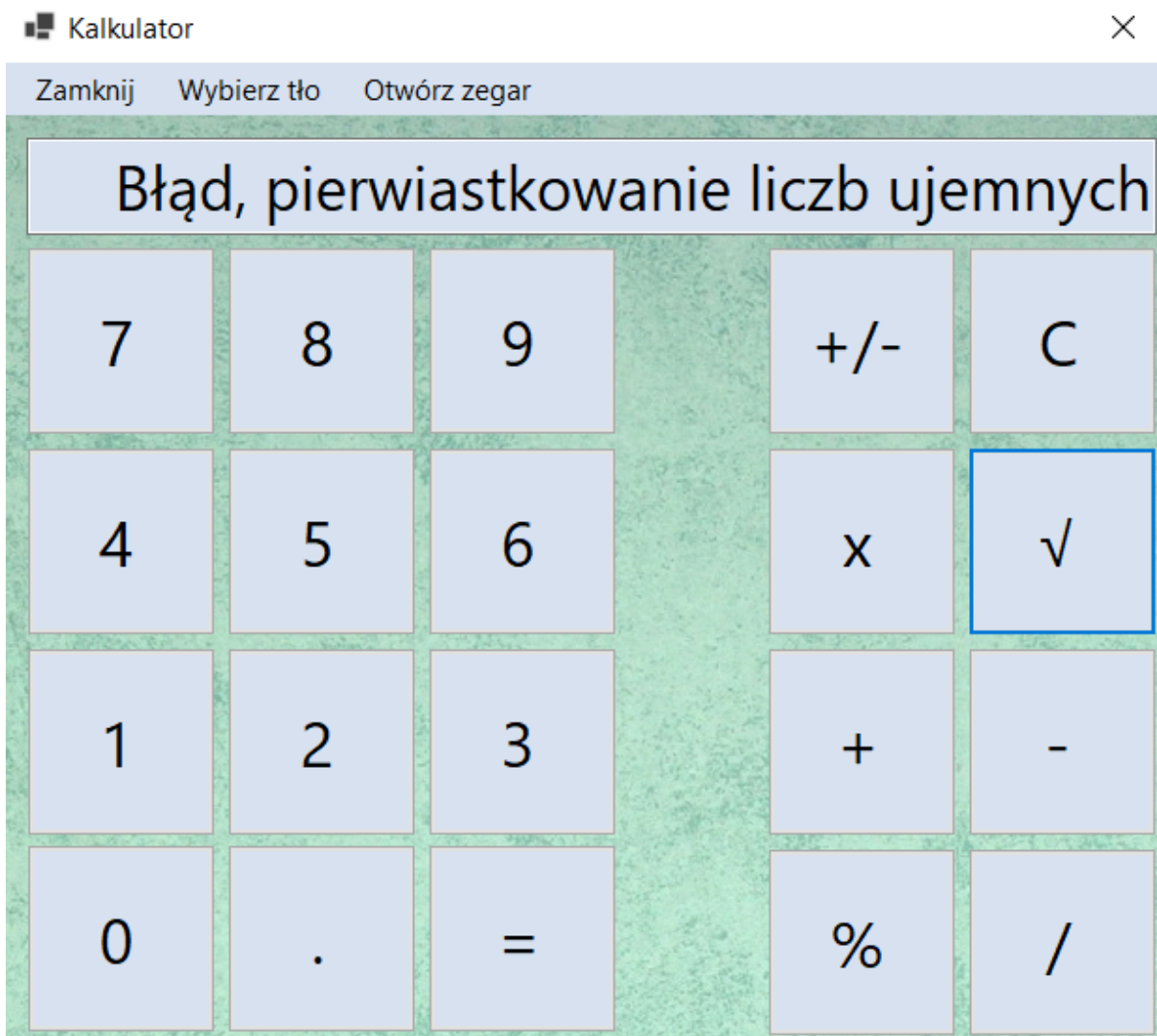
Celem projektu było wykorzystanie dowolnego języka programowania dla komputerów w standardzie PC do napisania aplikacji wykorzystującej technikę tworzenia graficznego interfejsu użytkownika. Interfejs miał w sposób interaktywny wykorzystywać do komunikacji z operatorem monitor ekranowy, klawiaturę oraz myszkę.

Do wykonania tego zadania zdecydowaliśmy się wykonać aplikację kalkulatora z zegarem w języku C#, korzystając z platformy .NET.

2) Kalkulator

W założeniach kalkulator miał przede wszystkim poprawnie rozwiązywać najbardziej podstawowe operacje na liczbach zarówno na liczbach całkowitych, jak i na liczbach wymiernych. Ponadto miał mieć możliwość interakcji poprzez zarówno monitor ekranowy jak i klawiaturę. Dodatkową funkcjonalnością jaką kalkulator miał spełniać w założeniach jest zmienianie tła, czyli tak zwanej "skórki".

Wszystkie założenia udało się spełnić, kalkulator reaguje na wpisanie niepoprawnych działań takich jak dzielenie przez zero, bądź pierwiastkowanie liczb ujemnych, wyświetlając na ekranie komunikat o stosownym błędzie. Sterowanie klawiaturą działa bezbłędnie. Oprócz oczywistych przypisań klawiszy zdecydowaliśmy się, że pierwiastkowanie będzie odpowiadało znakowi backtick (`) na klawiaturze, a zmiana znaku odpowiada znakowi zapytania (?) na klawiaturze, a czyszczenie ekranu odpowiada literze "c" na klawiaturze.



Rys.1 Wyświetlacz w momencie próby pierwiastkowania liczby ujemnej.

```

1 odwołanie
private void P_zero_Click(object sender, EventArgs e)
{
    if(textBox1.Text == "0" || textBox1.Text.Contains("Błąd")) // usuwa napis z błędem
    {
        textBox1.Text = "0"; // warunek uniemożliwiający wpisywanie nieskończonych zer
    }
    else
    {
        textBox1.Text += "0";
    }
}

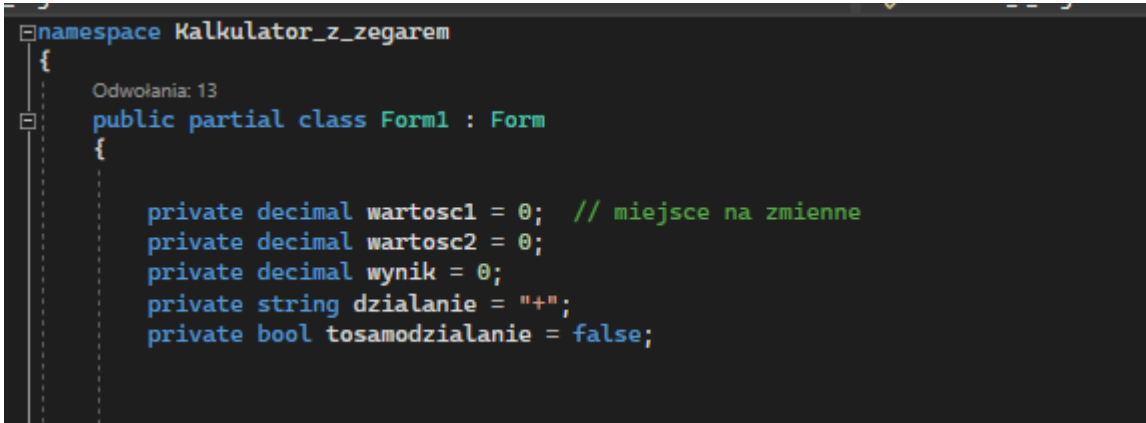
1 odwołanie
private void P_kropka_Click(object sender, EventArgs e)
{
    if (!textBox1.Text.Contains(",") && !textBox1.Text.Contains("Błąd")) // warunek który uniemożliwia wpisanie wielu kropek
    {
        textBox1.Text += ",";
    }
}

1 odwołanie
private void P_jeden_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "0" || textBox1.Text.Contains("Błąd")) // warunek usuwający początkowe zero z wyświetlacza i ewentualny napis z błędem
    {
        textBox1.Text = "1";
    }
    else
    {
        textBox1.Text += "1";
    }
}

```

Rys. 2 Część kodu prezentująca wpisanie odpowiednio zera, przecinka i jedynki w kalkulator

Wykonywanie działań zostało zrealizowane za pomocą zmiennych “wartosc1”, “wartosc2”, “wynik”, “dzialanie”, “tosamodzialanie”, które zostały zdefiniowane na początku pliku Form1.cs.



```
namespace Kalkulator_z_zegarem
{
    Odwołania: 13
    public partial class Form1 : Form
    {
        private decimal wartosc1 = 0; // miejsce na zmienne
        private decimal wartosc2 = 0;
        private decimal wynik = 0;
        private string dzialanie = "+";
        private bool tosamodzialanie = false;
    }
}
```

Rys.3 Deklaracje zmiennych używanych w aplikacji kalkulator.

Wykonanie działania wygląda w następujący sposób:

- a) użytkownik wpisał pierwszą wartość i kliknął znak działania np. minus
- b) zapisanie wartości, która znajduje się na wyświetlaczu do zmiennej “wartosc1”
- c) wyczyszczenie wyświetlacza
- d) przypisanie znaku do zmiennej “dzialanie” odpowiedniej do wykonywanego działania
- e) użytkownik wpisuje drugą wartość i klika znak równości
- f) kod przechodzi do switcha, który posiada case dla każdego z możliwych działań. Przechodzi do odpowiedniego case’a i zapisuje do zmiennej “wartosc2” wartość znajdującą się na wyświetlaczu.
- g) wykonanie działania i przechowanie wyniku w zmiennej “wynik”.
- h) wyświetlenie na ekranie wyniku.

Oczywiście przedstawiony algorytm jest uproszczony i nie wspomina o wielu warunkach, które umożliwiają, np. zmienienie działania po wcześniejszym naciśnięciu znaku, bądź co się stanie, gdy na wyświetlaczu zamiast wartości znajdzie się komunikat o błędzie. Wszystkie takie sytuacje zostały przez nas wyłapane i odpowiednio dobraliśmy warunki, aby korzystanie z kalkulatora było łatwe i bezproblemowe.

```

1 odwołanie
private void P_minus_Click(object sender, EventArgs e)
{
    if (!textBox1.Text.Contains("Błąd"))
    {
        if (!string.IsNullOrEmpty(textBox1.Text)) // warunek który umożliwia zmianę działania (można klikać kilka razy klawisz działania i będzie zapisany najnowszy)
        {
            wartosc1 = decimal.Parse(textBox1.Text);
            textBox1.Clear();
        }
        dzialanie = "-";
        tosamodzialanie = false;
    }
}

1 odwołanie
private void P_plus_Click(object sender, EventArgs e)

1 odwołanie
private void P_razy_Click(object sender, EventArgs e)

1 odwołanie
private void P_dzielenie_Click(object sender, EventArgs e)

1 odwołanie
private void P_procent_Click(object sender, EventArgs e) // reszta z dzielenia

1 odwołanie
private void P_pierwiastek_Click(object sender, EventArgs e)

1 odwołanie
private void P_rownosc_Click(object sender, EventArgs e)
{
    switch (dzialanie)
    {
        case "-":
            if (tosamodzialanie == false && !string.IsNullOrEmpty(textBox1.Text))
            {
                wartosc2 = decimal.Parse(textBox1.Text);
            }
            wynik = wartosc1 - wartosc2;
            textBox1.Text = wynik.ToString();
            wartosc1 = wynik;
            tosamodzialanie = true;
            break;
        case "+":

```

Rys. 4 Przykład wykonywania działania odejmowania.

Wyjątkowym działaniem, który nie podlega opisanemu wyżej algorytmowi jest działanie pierwiastkowania, ponieważ realizuje się go jedynie na jednej wartości. Wykonywane jest ono od razu po naciśnięciu symbolu pierwiastka, bez potrzeby klikania znaku równości.

```

1 odwołanie
private void P_pierwiastek_Click(object sender, EventArgs e)
{
    if (!textBox1.Text.Contains("Błąd") && !string.IsNullOrEmpty(textBox1.Text))
    {
        wartosc1 = (decimal)double.Parse(textBox1.Text);
        textBox1.Clear();
        if (wartosc1 >= 0)
        {
            wynik = (decimal)Math.Sqrt((double)wartosc1);
            textBox1.Text = wynik.ToString();
        }
        else
        {
            textBox1.Text = "Błąd, pierwiastkowanie liczb ujemnych";
        }
    }
}

```

Rys.5 Pierwiastkowanie

Ciekawym warunkiem, który wymagał od nas dodanie zmiennej “tosamodzialanie” jest sytuacja, w której użytkownik będzie chciał kilkakrotnie wykonać to samo działanie na następnych otrzymanych wynikach. Przykład input’u: “2”, “+”, “3”, “=”, “=”, “=”. Chcieliśmy, żeby w takiej sytuacji wykonało się działanie 2+3 i następnie przy kolejnym naciśnięciu znaku równości do wyniku dodawała się kolejna trójka itd. Jednak zaimplementowany algorytm nie był w stanie sobie z tym poradzić i otrzymywaliśmy błędne wyniki. Ostatecznie dodanie zmiennej “tosamodzialanie” umożliwiło nam dodanie dwóch warunków (jeden dla działań przemiennych i jeden dla działań nieprzemiennych), które rozwiązały problem. Dla działań przemiennych takich jak dodawanie program po pierwszym wykonaniu przypisuje do zmiennej “wartosc1” zawartość zmiennej “wartosc2” i przy każdym następnym wywołaniu takiego działania “wartosc1” pozostaje taka sama. Z kolei dla działań nieprzemiennych takich jak odejmowanie “wartosc2” pozostaje taka sama po pierwszym wykonaniu działania.

```
switch (dzialanie)
{
    case "-":
        if (tosamodzialanie == false && !string.IsNullOrEmpty(textBox1.Text))
        {
            wartosc2 = decimal.Parse(textBox1.Text);
        }
        wynik = wartosc1 - wartosc2;
        textBox1.Text = wynik.ToString();
        wartosc1 = wynik;
        tosamodzialanie = true;

        break;
    case "+":
        if (!string.IsNullOrEmpty(textBox1.Text)) // warunek ktory umozliwia klikniecie dzialanie i od razu znaku rownosc
        {
            wartosc2 = decimal.Parse(textBox1.Text);
        }
        wynik = wartosc1 + wartosc2;
        textBox1.Text = wynik.ToString();
        if (tosamodzialanie == false)
        {
            wartosc1 = wartosc2;
            tosamodzialanie = true;
        }
        break;
}
```

Rys. 6 Przykład warunków wykorzystujących zmienną “tosamodzialanie”

Do zmieniania wyglądu naszego kalkulatora, zamykania aplikacji oraz do otwierania zegara posłużyliśmy się paskiem menu “menustrip”, na którym znajdują się wszystkie 3 wymienione funkcje. Są dostępne 3 różne tła do wyboru.

```
1 odwołanie
private void zieloneToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("C:\\Users\\Mati\\source\\repos\\OSK\\Kalkulator\\Kalkulator_z zegarem\\zdjęcia\\zielony.jpeg");
}

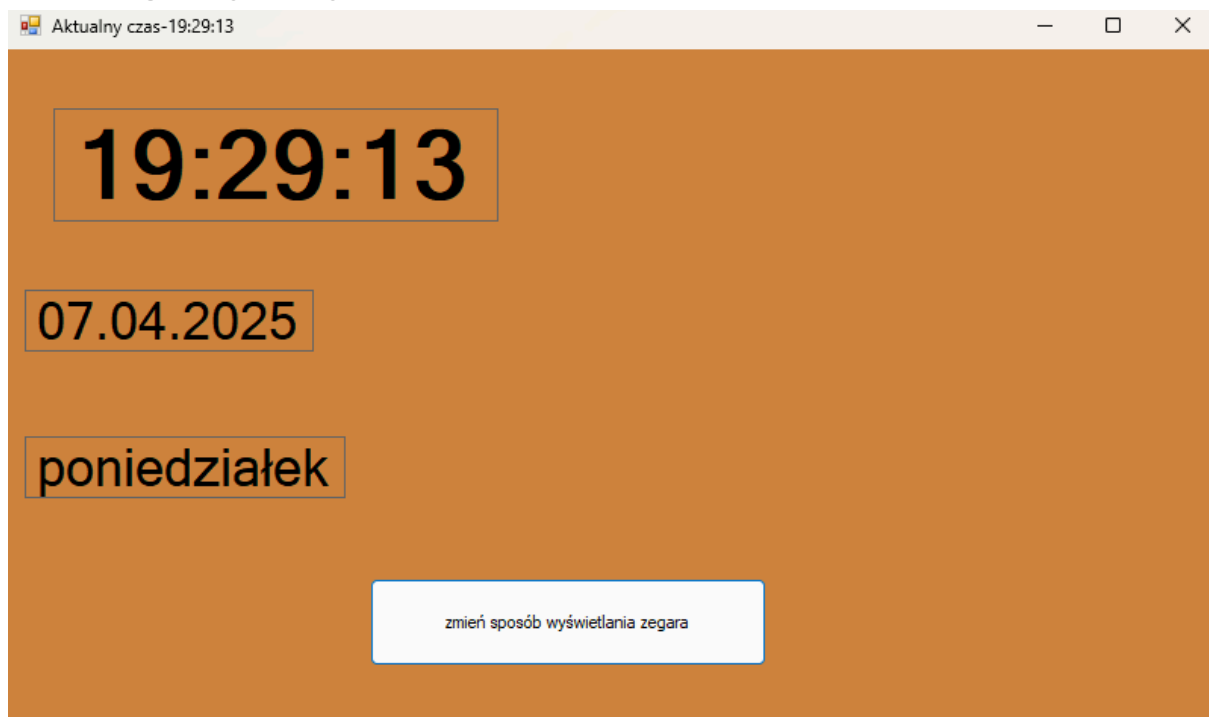
1 odwołanie
private void niebieskieToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("C:\\Users\\Mati\\source\\repos\\OSK\\Kalkulator\\Kalkulator_z zegarem\\zdjęcia\\niebieskie.jpg");
}

1 odwołanie
private void różoweToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("C:\\Users\\Mati\\source\\repos\\OSK\\Kalkulator\\Kalkulator_z zegarem\\zdjęcia\\różowe.jpg");
}
```

Rys.7 Zmiana wyglądu tła kalkulatora

3) Zegar

a) zegar cyfrowy



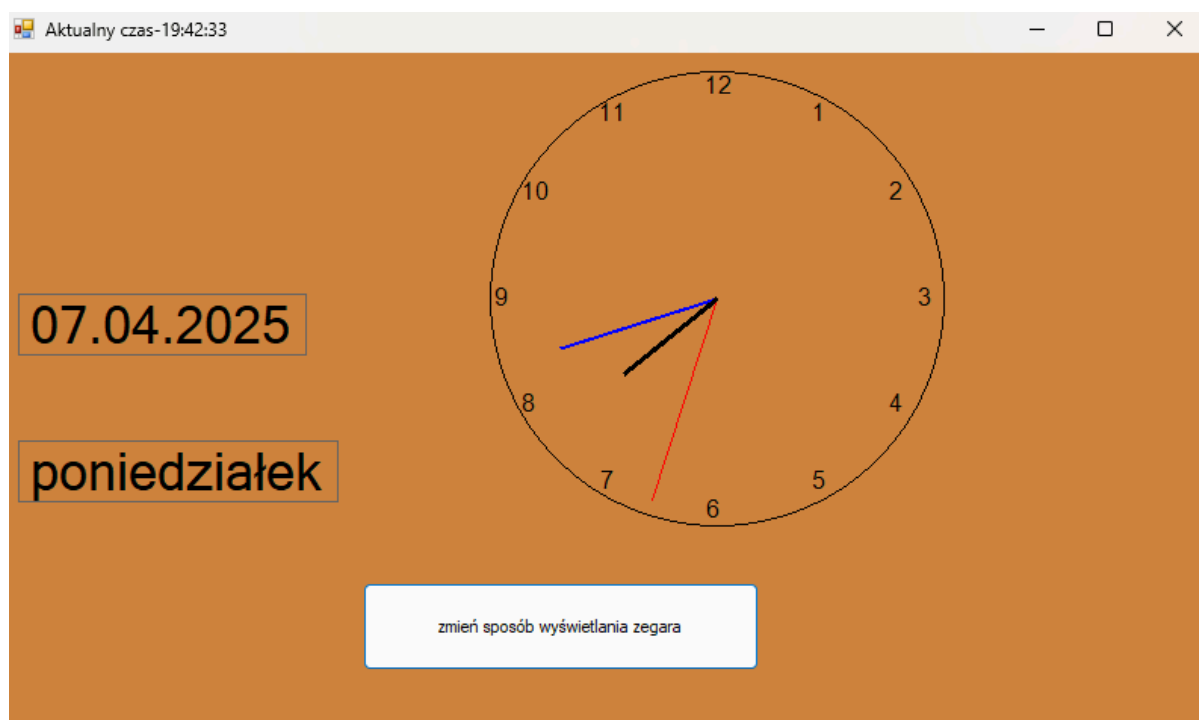
Rys.8 Wygląd aplikacji z wyświetlaniem zegara cyfrowego. Label wyświetlający godzinę nazywa się godzina_minuta, label wyświetlający datę nazywa się data, label wyświetlający dzień tygodnia nazywa się dzien.

Aplikacja zegara ma na celu wyświetlać aktualny czas. W tym przypadku została wzbogacona o funkcję wyświetlania aktualnej daty i dnia tygodnia. Każdy z tych elementów jest strukturą "label". W aplikacji funkcjonuje także "timer" z sekundowym interwałem. Co sekundę każdy "label" pobiera odpowiednio przypisaną informację o obecnej dacie, godzinie i dniu, wykorzystując strukturę i funkcję `DateTime.Now.ToString()`. Wewnątrz nawiasów tej funkcji zostały odpowiednio wpisane skróty literowe potrzebne do wyświetlania godziny, minuty, sekundy, dnia tygodnia i daty.

```
1 odwołanie
private void timer1_Tick(object sender, EventArgs e)
{
    godzina_minuta.Text = DateTime.Now.ToString("HH:mm:ss");
    data.Text = DateTime.Now.ToString("dd.MM.yyyy");
    dzien.Text = DateTime.Now.ToString("dddd");
}
```

Rys 9. Kod realizujący funkcję pobrania informacji o dacie, godzinie i dniu tygodnia.

b) zegar analogowy



Rys.10 Wygląd aplikacji z wyświetlaniem zegara analogowego.

Zegar analogowy jest wyświetlany wewnątrz struktury PictureBox o rozmiarze 302x302. Na początku działania programu tworzona jest bitmapa oraz deklarowana jest klasa Graphics g, aby móc rysować po bitmapie. Dodatkowo ustalana jest na początku długość wskazówek zegara (secHand, minHand, hourHand), oraz współrzędne środka tarczy zegara(cx,cy).

```
int cx;
int cy;
//Timer t = new Timer();
1 odwołanie
public Form1()
{
    InitializeComponent();
}

Bitmap bmp;
Graphics g;
int secHand = 140;
int minHand = 110;
int hourHand = 80;
int width = 300;
int height = 300;

1 odwołanie
private void Form1_Load(object sender, EventArgs e)
{
    //deklaracja rozpoczęcia pracy timera, ustawienie kordów bitmapy i środka zegara
    timer1.Start();
    bmp = new Bitmap(width + 1, height + 1);
    cx = width / 2;
    cy = height / 2;
```

Rys.11 Deklaracja bitmapy, Graphics g, długości wskazówek zegara i środka tarczy

Zegar analogowy co sekundę pobiera aktualny czas, wykorzystując strukturę DateTime.Now i przypisuje godzinę, minutę i sekundę do zmiennych hh,mm,ss. Następnie rysuje tarczę zegara przy pomocy struktury i funkcji g.DrawEllipse, wylicza współrzędne godzin i podpisuje je na tarczy w pętli for, wykonując obliczenia trygonometryczne w funkcji HourNumberCoord i funkcję g.DrawString(). Następnie zegar wylicza współrzędne punktów potrzebnych do narysowania wskazówek przy pomocy funkcji g.DrawLine. Po narysowaniu wskazówek bitmapa jest wczytywana do PictureBoxa, a następnie czyszczona, aby bitmapy nie nakładały się na siebie. Dodatkowo aplikacja wyświetla aktualny czas cyfrowy zamiast nazwy Form2, poprzez modyfikację struktury this.Text.


```

g = Graphics.FromImage bmp;
//pobranie czasu
int ss = DateTime.Now.Second;
int mm = DateTime.Now.Minute;
int hh = DateTime.Now.Hour;

int[] handcoord = new int[2];
//czyszczenie mapy i wypełnianie jej kolorem, każdy tik zegara czyści pictureBoxa i rysuje na nowo tarcze ze wskazówkami
if(kolor == "pomaranczowy")
{
    g.Clear(Color.Peru);
}
else if (kolor == "niebieski")
{
    g.Clear(Color.LightBlue);
}
else if (kolor == "biały")
{
    g.Clear(Color.White);
}
// g.Clear(Color.Peru);

//rysowanie tarczy
g.DrawEllipse(new Pen(Color.Black, 1f), 0, 0, 300, 300);

Font font = new Font("Arial", 12);
Brush brush = Brushes.Black;

```

```

*/
for (int i = 1; i <= 12; i++)
{
    int[] pos = hourNumberCoord(i, secHand); // secHand to promień
    g.DrawString(i.ToString(), font, brush, new PointF(pos[0] - 10, pos[1] - 10));
}

//wskazowka sekund
handcoord = msCoord(ss, secHand);
g.DrawLine(new Pen(Color.Red, 1f), new Point(cx, cy), new Point(handcoord[0], handcoord[1]));
//wskazowka minut
handcoord = msCoord(mm, minHand);
g.DrawLine(new Pen(Color.Blue, 2f), new Point(cx, cy), new Point(handcoord[0], handcoord[1]));
//wskazowka godzin
handcoord = hrCoord(hh % 12, mm, hourHand);
g.DrawLine(new Pen(Color.Black, 3f), new Point(cx, cy), new Point(handcoord[0], handcoord[1]));
//załadowanie bitmapy do pictureBoxa
tarcza.Image = bmp;
this.Text = "Aktualny czas-" + hh + ":" + mm + ":" + ss;
//zwolnienie pamieci
g.Dispose();
}

```

```

1 odwołanie
private int[] hourNumberCoord(int hour, int radius)
{
    int[] coord = new int[2];

    // 1 godzina to 30 stopni
    int val = hour * 30;

    if (val >= 0 && val <= 180)
    {
        coord[0] = cx + (int)(radius * Math.Sin(Math.PI * val / 180));
        coord[1] = cy - (int)(radius * Math.Cos(Math.PI * val / 180));
    }
    else
    {
        coord[0] = cx - (int)(radius * -Math.Sin(Math.PI * val / 180));
        coord[1] = cy - (int)(radius * Math.Cos(Math.PI * val / 180));
    }

    return coord;
}

```

Rys.12 realizacja rysowania tarczy i wskazówek, wyliczanie współrzędnych godzin na tarczy

Aby poprawnie narysować wskazówki, zegar wykorzystuje funkcję `msCoord` zwracającą dwuelementową tablicę do obliczania współrzędnych końcówki wskazówki sekundy i minuty, oraz funkcję `hrCoord` do obliczania współrzędnych końcówki wskazówki godziny. Do funkcji `msCoord` wchodzi zmienna `val`, która jest aktualną sekundą lub minutą. Zmienna ta jest mnożona przez 6, aby móc reprezentować ją w stopniach. Następnie sprawdzane jest w jakiej połowie tarczy będzie się znajdować współrzędna (przyjętą bazą jest: godzina 12-0 stopni, godzina 3- 90 stopni itd.). Później wykonywane są obliczenia trygonometryczne w celu obliczenia współrzędnych końcówki. Wykorzystywana jest tu zmienna `hlen`, która przechowuje długość danej wskazówki. Po wykonaniu obliczeń zwracana jest tablica przechowująca obie współrzędne. Przy rysowaniu wskazówek (Rys.12) wykorzystujemy zmienne `ss` i `mm`, które przechowują aktualną sekundę i minutę.

```
// kordy na wskazowki minuty i sekundy
Odwołania: 2
private int[] msCoord(int val, int hlen)
{
    int[] coord = new int[2];
    //kazda sekunda to 6 stopni
    val *= 6;

    if(val >= 0 && val <= 180)
    {
        coord[0] = cx + (int)(hlen * Math.Sin(Math.PI * val / 180));
        coord[1] = cy - (int)(hlen * Math.Cos(Math.PI * val / 180));
    }
    else
    {
        coord[0] = cx - (int)(hlen * Math.Sin(Math.PI * val / 180));
        coord[1] = cy - (int)(hlen * Math.Cos(Math.PI * val / 180));
    }
    return coord;
}
```

Rys. 13 Obliczanie współrzędnych wskazówki sekundy i minuty

Obliczanie współrzędnych końcówki wskazówki godziny jest identyczne do obliczania współrzędnych końcówki wskazówki minuty i sekundy. Jedyną różnicą jest sposób obliczania wartości zmiennej `val`. Przy jej obliczaniu wykorzystujemy dwie wczytywane zmienne `mval` i `hval`, które reprezentują aktualną minutę i godzinę. Wykorzystujemy tutaj fakt, że zmiana czasu o minutę odpowiada przesunięciu się wskazówki o 0.5 stopnia, a zmiana godziny odpowiada przesunięciu o 30 stopni. Przy

rysowaniu wskazówek(Rys.12) wykorzystujemy zmienną hh, która przechowuje aktualną godzinę. Tę godzinę wcześniej konwertujemy na standard 12-godzinny.

```
//kordy na wskazowke godziny
1 odwołanie
private int[] hrCoord(int hval, int mval, int hlen)
{
    int[] coord = new int[2];

    //1 godzina to 30 stopni, 1 minuta to 0.5 stopni
    int val = (int)((hval * 30) + (mval * 0.5));

    if (val >= 0 && val <= 180)
    {
        coord[0] = cx + (int)(hlen * Math.Sin(Math.PI * val / 180));
        coord[1] = cy - (int)(hlen * Math.Cos(Math.PI * val / 180));
    }
    else
    {
        coord[0] = cx - (int)(hlen * -Math.Sin(Math.PI * val / 180));
        coord[1] = cy - (int)(hlen * Math.Cos(Math.PI * val / 180));
    }
    return coord;
}
```

Rys 14.Obliczanie współrzędnych wskazówki godziny

c) przełączanie między zegarami

Aby móc wyświetlać tylko jeden zegar i przełączać się między nimi, przy załadowaniu aplikacji wyłączania jest widoczność zegara analogowego przy pomocy linijki `tarcza.Visible=false`. Następnie w funkcji realizującej kliknięcie przycisku przełączania zegarów zmieniana jest wartość zmiennej boolowskiej odpowiedzialnej za widoczność obydwu zegarów. Dzięki temu wyświetlany jest jednocześnie tylko jeden zegar.

```
1 odwołanie
private void Form1_Load(object sender, EventArgs e)
{
    //deklaracja rozpoczecia pracy timera, ustawienie kordów bitmapy i środka zegara
    timer1.Start();
    bmp = new Bitmap(width + 1, height + 1);
    cx = width / 2;
    cy = height / 2;
    tarcza.Visible = false;
}
```

```

1 odwołanie
private void button1_Click(object sender, EventArgs e)
{
    godzina_minuta.Visible = !godzina_minuta.Visible;
    data.Visible = !data.Visible;
    dzien.Visible = !dzien.Visible;
    tarcza.Visible = !tarcza.Visible;
}

```

Rys.15 Realizacja przełączania się między zegarami

4) Integracja kalkulatora z zegarem

Przełączanie między kalkulatorem, a zegarem odbywa się za pomocą paska menu. W momencie wyboru zegara z pozycji kalkulatora (bądź na odwrót) program chowa poprzednie okienko, otwiera nowe, z wybranym narzędziem i na koniec zamyka wszystkie poprzednio otwarte okienka. Zamknięcie wszystkich poprzednich okienek działa jako zabezpieczenie przed możliwymi spadkami wydajności przy otwarciu zbyt wielu okien na raz. Oprócz tego zachowaliśmy zmienną `loo`, która również służyła do zabezpieczenia przed otwarciem więcej niż jednego okna.

```

1 odwołanie
private void otwórzZegarToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (Form2.loo == 0) // zabezpieczenie przed otwarciem więcej niż jednego okna
    {
        this.zegar = new Form2();
        this.Hide();

        Form1.loo--;

        zegar.ShowDialog();
        foreach (Form f in Application.OpenForms.Cast<Form>().ToList())
        {
            if (f != zegar) f.Close(); // Zamykamy inne okna, ale NIE nowo otwarte
        }
    }
}

```

Rys. 8 Otwarcie nowego okna.

5) Podsumowanie rezultatów

Wszystkie założenia, które były wymagane od nas przez polecenie udało nam się pomyślnie zrealizować. Wykorzystaliśmy do komunikacji, nie tylko monitor i myszkę, ale również klawiaturę. Zawarliśmy konfigurację aplikacji zarówno w kalkulatorze, jak i w zegarze. Skorzystaliśmy z większości elementów wprowadzonych na zajęciach laboratoryjnych z OSK. Dodaliśmy od siebie parę elementów w celu polepszenia komfortu użytkownika (np. zmienna "tosamodzialanie" w kalkulatorze).