17.06.2016

PODSTAWY TELEINFORMATYKI - DOKUMENTACJA

Podłączenie dodatkowego ekranu przez sieć IP

Autorzy:
Tomasz Ładowski 116859
Przemysław Kułaga 116866

Spis Treści

1. Wstęp

Celem naszego projektu było stworzenie aplikacji zapewniającej funkcjonalność podłączenia dodatkowego ekranu przez sieć IP. Z jednego z komputerów wysyła się obraz pulpitu, który odbiera drugi komputer przez sieć IP poprzez protokół TCP. Na początku naszej pracy źle zrozumieliśmy temat naszego projektu, który w rzeczywistości okazał się o wiele bardziej złożony. Poprzez nasz błąd straciliśmy wiele czasu na początku próbując zrealizować samo przechwytywanie głównego pulpitu jednego komputera, a następnie przesyłanie go na ekran drugiego komputera. W rezultacie nie udało nam się dokładnie zrealizować właściwego tematu naszego projektu. W zamian zaprogramowaliśmy aplikację, która tworzy dodatkowy pulpit dla użytkownika systemu Windows, a następnie za pomocą stworzonej wcześniej aplikacji przesyłamy go na docelową stację roboczą

1.1. Podobne aplikacje

Po dłuższym przeszukaniu Internetu nie udało nam się znaleźć podobnych aplikacji (na system Windows) do programu, który był celem naszego projektu.

W przypadku systemu operacyjnego Linux istnieje już podobne rozwiązanie kwestii podłączenia dodatkowego ekranu poprzez IP dzięki specyfikacji Wifi-Display znanej jako Miracast.

1.2. Dlaczego wybraliśmy ten temat?

Głównym powodem, dla którego wybraliśmy ten temat była chęć wykorzystania stworzonej aplikacji do celów prywatnych. Posiadamy w domach stare komputery i laptopy, które z powodzeniem mogą służyć jako dodatkowy ekran dla naszego komputera. Niewielkie zapotrzebowanie na moc obliczeniową mogłoby uczynić te urządzenia niezwykle przydatnymi.

Dodatkowo chcieliśmy poprawić swoje umiejętności w pisaniu aplikacji sieciowych wykorzystujących Sockety do połączeń internetowych i przesyłania danych w różnych formatach między klientem a serwerem w technologii C# Windows Forms.

2. Cele projektu

2.1 Osiągnięte

2.1.1 Przesyłanie ekranu

- Nawiązanie połączenia z odbierającym obraz - wykorzystanie biblioteki systemowej do tworzenia Socketów
- Utworzenie strumienia przesyłu danych do odbierającego komputera, w celu możliwości ciągłego wysyłania obrazów do komputera odbierającego dane
- Wybór adresu IP, na który będą przesyłane dane - w programie do odbierania danych wyświetli się automatycznie adres IP, który następnie należy podać użytkownikowi, który chce wysyłać nam swój pulpit
- Wybór portu, na którym będzie się przesyłać dane - użytkownik może wybrać port, do którego dane mają być wysyłane
- Tworzenie Bitmapy z aktualnego obrazu pulpitu - program wykonuje zrzut ekranu o rozmiarach rozdzielczości ekranu komputera wysyłającego dane
- Przesyłanie utworzonej wcześniej Bitmapy, co jeden tick timera - timer można ustawić na różne prędkości wysyłania danych, Bitmapa jest kodowana za pomocą BinaryFormattera a następnie przesyłana do stacji docelowej
- Wyświetlanie adresu IP komputera, który chce odbierać obraz - w celu podania go stacji, która ma zamiar wysłać dane
- Wybór portu, na którym obraz będzie odbierany - użytkownik może wybrać dowolny port, na którym będzie odbierał dane od stacji wysyłającej
- Nawiązanie połączenia z wysyłającym obraz - użytkownik czeka, aż ktoś nie będzie chciał wysyłać obrazu na jego adres IP i podany wcześniej port
- Utworzenie strumienia do odbierania danych od wysyłającego komputera, aby móc ciągle odbierać obraz od stacji nadającej
- Utworzenie wątku odpowiedzialnego za nasłuchiwanie i oczekiwanie na połączenie
- Utworzenie wątku odpowiedzialnego za odbieranie danych, kiedy połączenie zostało już nawiązane
- Odbieranie Bitmapy od wysyłającego i dekodowanie jej za pomocą BinaryFormatter do formatu obrazu

- Wyświetlanie odebranego obrazu w pełnym ekranie lub oknie - przed odbieraniem obrazu użytkownikowi ukaże się dodatkowe okno, w którym obraz będzie wyświetlany i przycisk do włączenia lub wyłączenia trybu pełnoekranowego
- Zatrzymanie połączenia - kiedy wyłączymy ekran do odbierania obrazu, wątki odpowiedzialne za odbieranie obrazu i nasłuchiwanie zakończy się

2.1.2 Tworzenie pulpitów

- Utworzenie nowego pulpitu - wykorzystanie funkcji Desktop API udostępnionych przez Microsoft do stworzenia nowego wirtualnego pulpitu dla systemów Windows
- Uruchomienie procesów na nowym pulpicie - uruchomienie procesu “explorer.exe” dla nowo powstałego pulpitu w celu inicjacji paska start oraz całego interfejsu przyjaznego użytkownikowi
- Przełączanie pulpitów - możliwość przełączania użytkownika między pulpitami

2.2 Nieosiągnięte

- Przesyłanie grafiki w trybie rozszerzonego obrazu - jedną z możliwości rozwiązania problemu byłoby napisanie sterownika monitora bądź stworzenie tak zwanej wtyczki “dummy VGA”, która wpięta do karty graficznej zostaje odczytana jako monitor, który następnie można by przesłać na inny komputer

3. Metodyka konstruowania systemu

3.1 Metodyka pracy zespołowej

Przy realizacji projektu wykorzystaliśmy metodykę scrum dostosowaną na potrzeby warunków studenckich. Ze względu na dużą liczbę projektów w tym semestrze ograniczyliśmy się na początku do sprintów 2-tygodniowych. Niestety błędna interpretacja tematu zmusiła nas do zredukowania czasu trwania sprintów do 1 tygodnia.

3.2 Środki implementacji

- C# WPF - zastosowany do pierwszych prób implementacji naszego projektu, zrezygnowaliśmy z tej technologii ze względu na szereg problemów wynikających z jej korzystania
- C# Windows Forms - zastosowany do właściwej implementacji naszego projektu, dzięki prostocie wykorzystania i tworzenia interfejsu graficznego pozwolił nam w prostszy sposób zaprogramować elementy projektu

3.3 Metodyka modelowania

Przy modelowaniu naszego projektu korzystaliśmy z następujących diagramów UML:

- przypadków użycia
- aktywności
- klas

3.4 Wykorzystane środowiska programistyczne

- Microsoft Visual Studio 2015 Enterprise - zintegrowane środowisko programistyczne firmy Microsoft. Jest używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika, w tym aplikacje Windows Forms, WPF, Web Sites, Web Applications i inne. Aplikacje mogą być pisane na platformy: Microsoft Windows, Windows Phone, Windows CE, .NET Framework, Microsoft Silverlight oraz konsole XBOX. W przypadku naszego projektu użyliśmy go do napisania 3 programów z interfejsem graficznym w technologii Windows Forms. Elementy, takie jak przegląd klas i integracja z GitHubem bardzo ułatwiły nam pracę i implementację naszego projektu.

3.5 Wykorzystane narzędzia

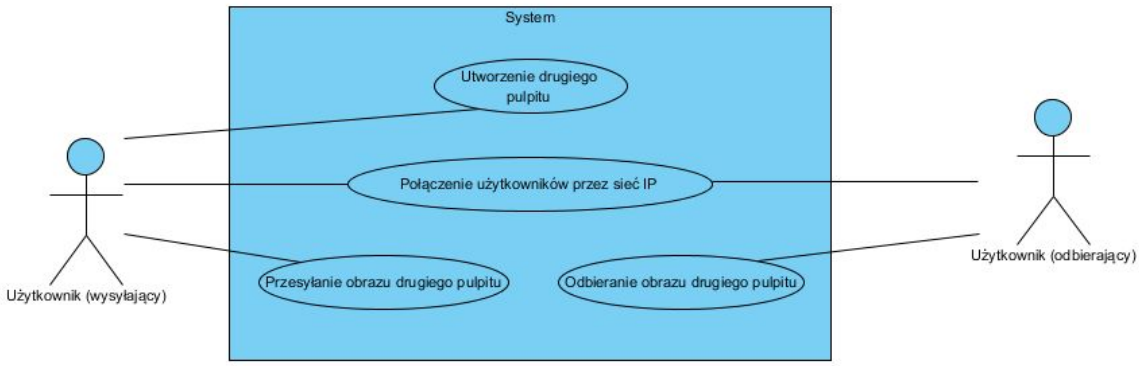
- GitHub - hostingowy serwis internetowy przeznaczony dla projektów programistycznych wykorzystujących system kontroli wersji Git. Umożliwił nam on dostęp do kodu na obu komputerach, na których pracowaliśmy oraz pozwolił zobaczyć zmiany w kodzie, przez co mogliśmy łatwo analizować historię tworzenia projektu.

4. Model systemu (Diagramy)

Przy modelowaniu naszego systemu korzystaliśmy z następujących diagramów UML:

4.1 Przypadków użycia

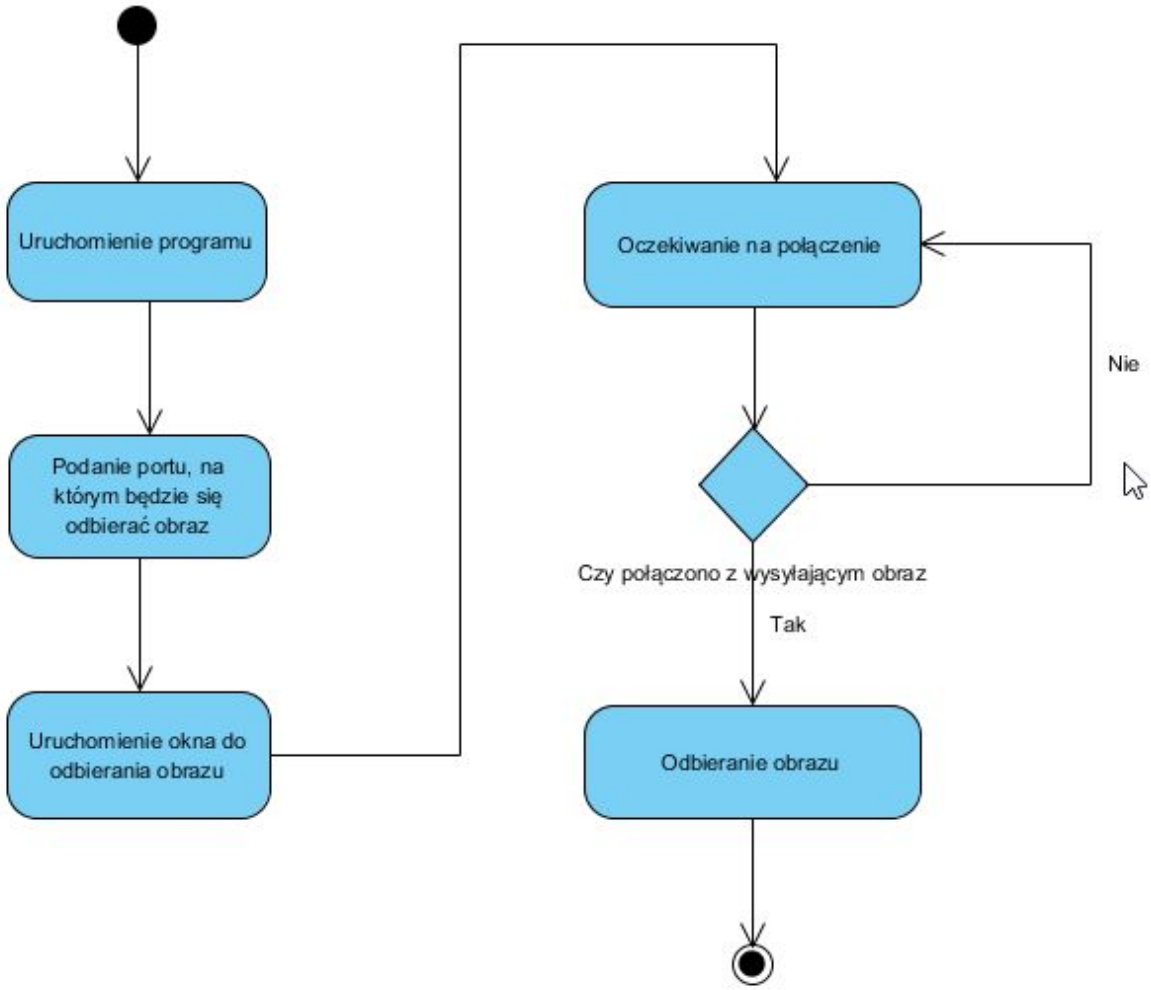
Diagram przedstawiony na rysunku 1.



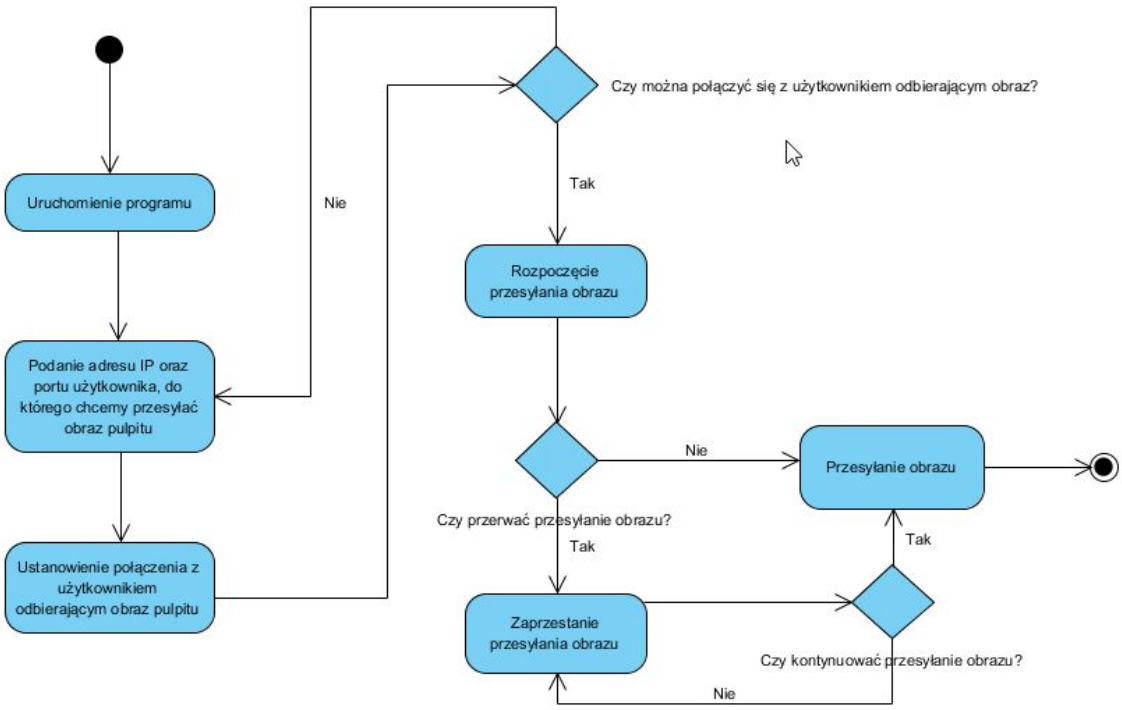
Rysunek 1: Diagram przypadków użycia dla systemu przesyłania obrazu pulpitu

4.2 Aktywności

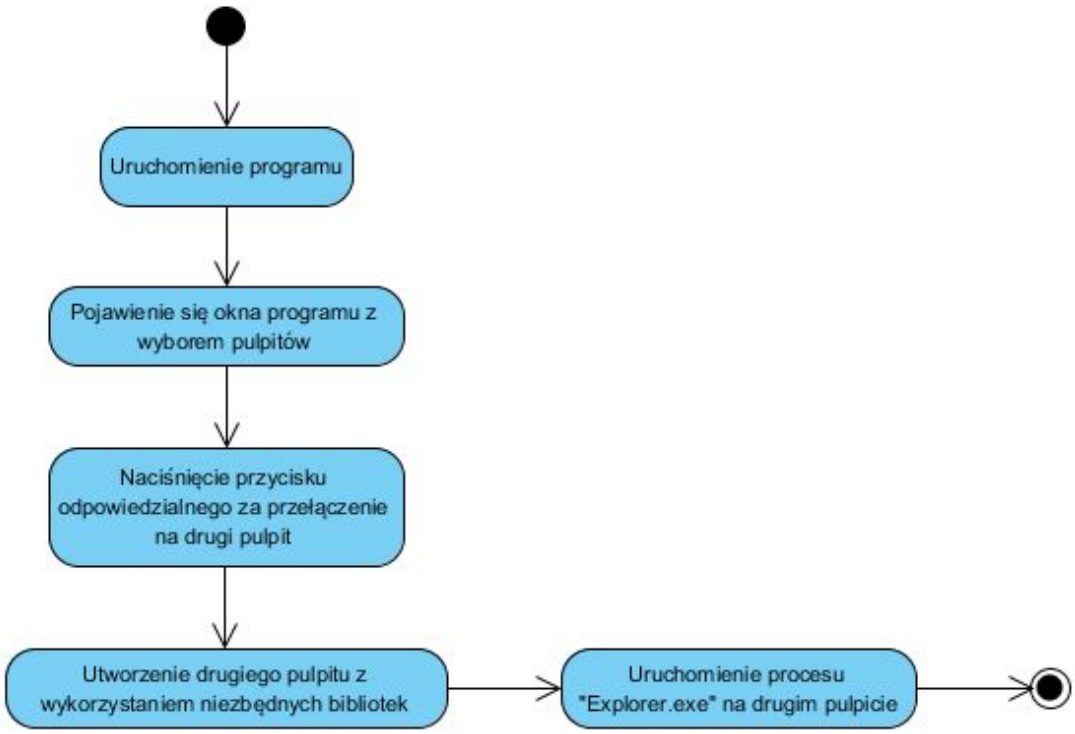
Diagramy przedstawione na rysunkach 2, 3 oraz 4.



Rysunek 2: Diagram aktywności dla programu odbierającego obraz



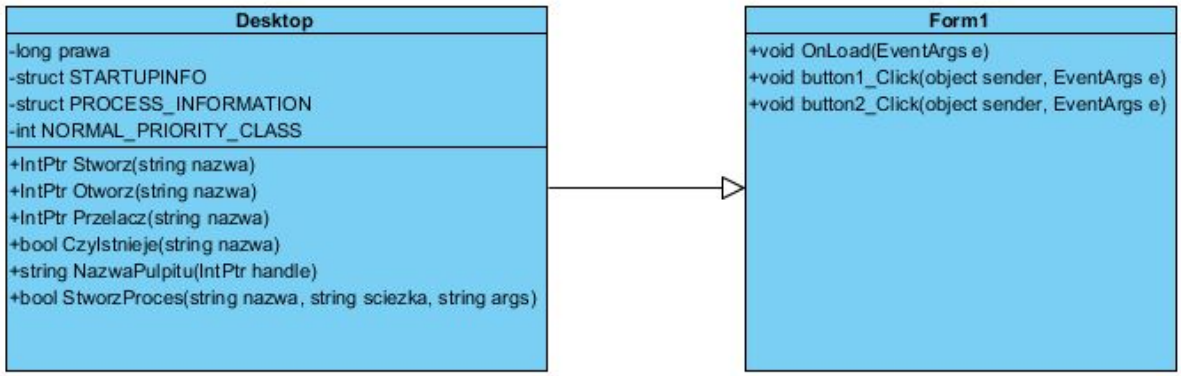
Rysunek 3: Diagram aktywności dla programu przesyłającego obraz pulpitu



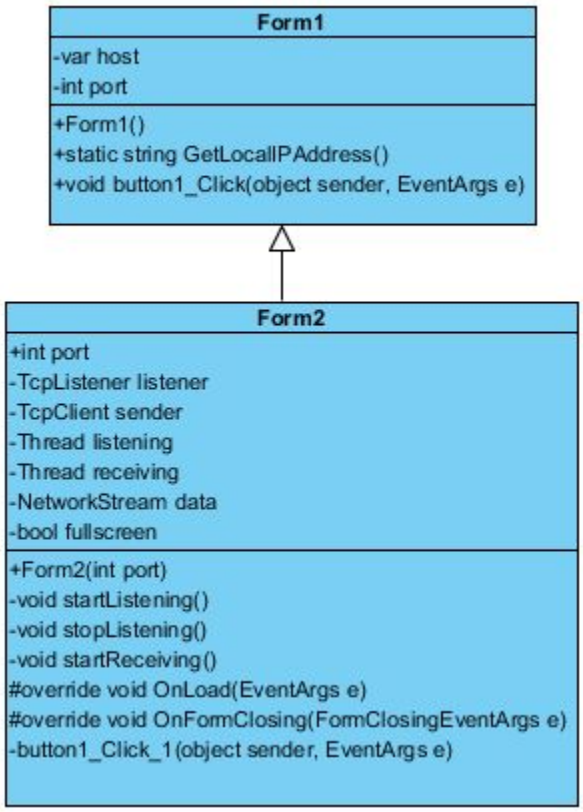
Rysunek 4: Diagram aktywności dla programu tworzącego nowy pulpit

4.3 Klas

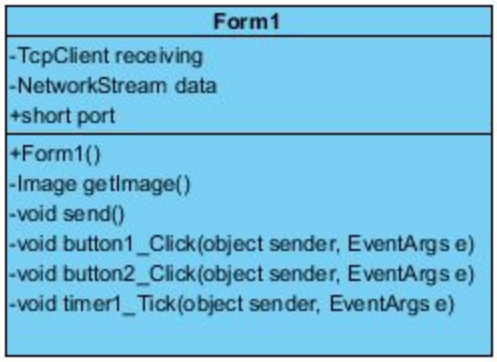
Diagramy przedstawione na rysunkach 5, 6 oraz 7.



Rysunek 5: Diagram klas dla programu tworzącego drugi pulpit



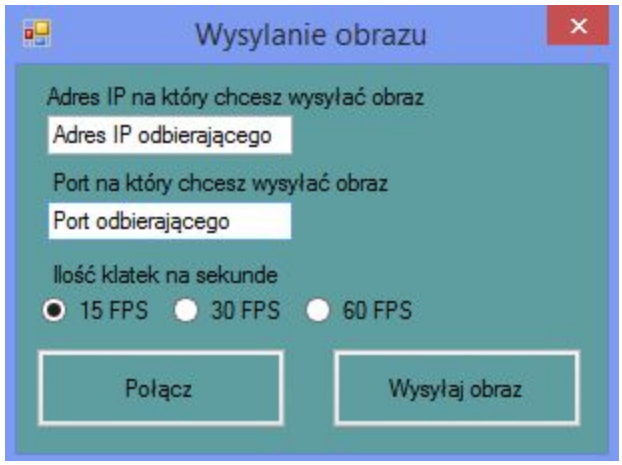
Rysunek 6: Diagram klas dla programu odbierającego obraz pulpitu



Rysunek 7: Diagram klas dla programu wysyłającego obraz pulpitu

5. Implementacja

5.1 Program do wysyłania obrazu pulpitu



Rysunek 8: Design programu wysyłającego obraz pulpitu

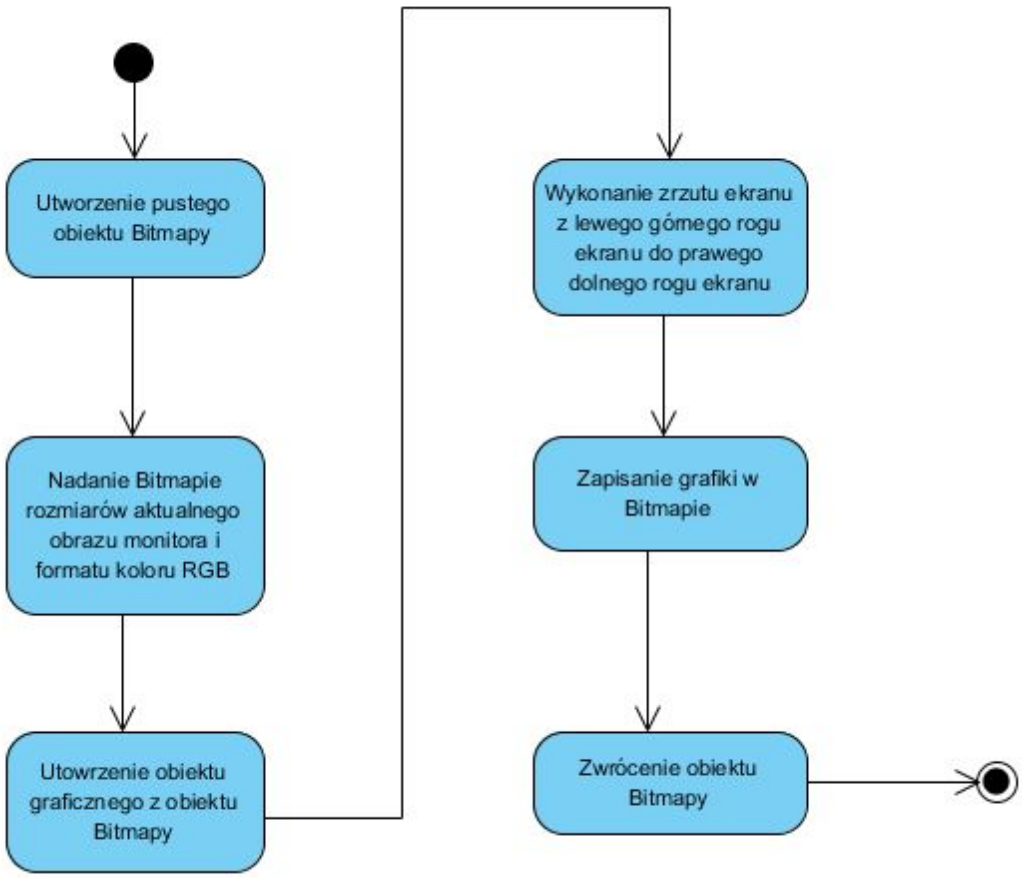
5.1.1 Funkcja getImage()

Służy do wykonania zrzutu ekranu pulpitu, na którym uruchomiony jest program.

```
private Image getImage()
      {
         Bitmap screenshot = null;

         foreach (Screen screen in Screen.AllScreens)
         {
            screenshot = new Bitmap(screen.Bounds.Width,
               screen.Bounds.Height,
             System.Drawing.Imaging.PixelFormat.Format32bppArgb);
```

```
            Graphics gfxScreenshot =
Graphics.FromImage(screenshot);
            gfxScreenshot.CopyFromScreen(
                screen.Bounds.X,
                screen.Bounds.Y,
                0,
                0,
                screen.Bounds.Size,
                CopyPixelOperation.SourceCopy);
        }
```



Rysunek 9: Diagram aktywności dla funkcji getImage()

5.1.2 Funkcja send()

Otwiera stream do wysyłania obrazu i zmienia wcześniej wykonany zrzut ekranu na bity

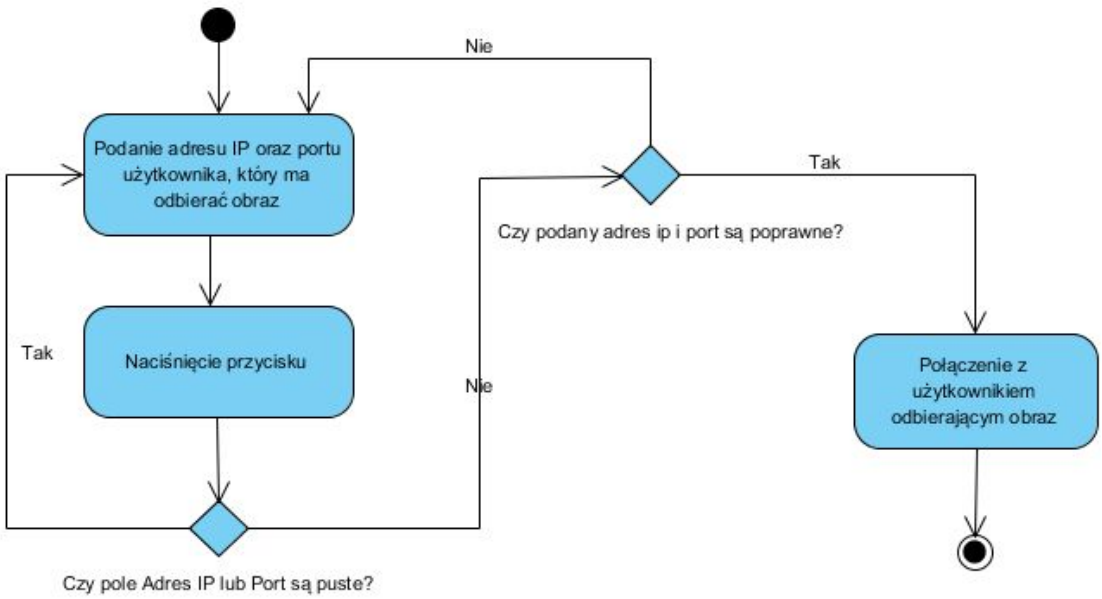
```
private void send()
      {
         BinaryFormatter bf = new BinaryFormatter();
         data = receiving.GetStream();
         bf.Serialize(data, getImage());
      }
```

Rysunek 10: Diagram aktywności dla funkcji send()

5.1.3 Przycisk “Połącz”

Realizuje połączenie z użytkownikiem odbierającym obraz pulpitu

```
private void button1_Click(object sender, EventArgs e) //polaczenie
        {
            if (textBox1.Text != "" && textBox2.Text != "")
            {
                try
                {
                    port = short.Parse(textBox2.Text);
                    receiving.Connect(textBox1.Text, port);
                    MessageBox.Show("Połączono!");
                }
                catch (Exception)
                {
                    MessageBox.Show("Nie udało się połączyć");
                }
            }
            else
            {
                MessageBox.Show("Podaj adres IP i port");
            }
        }
```

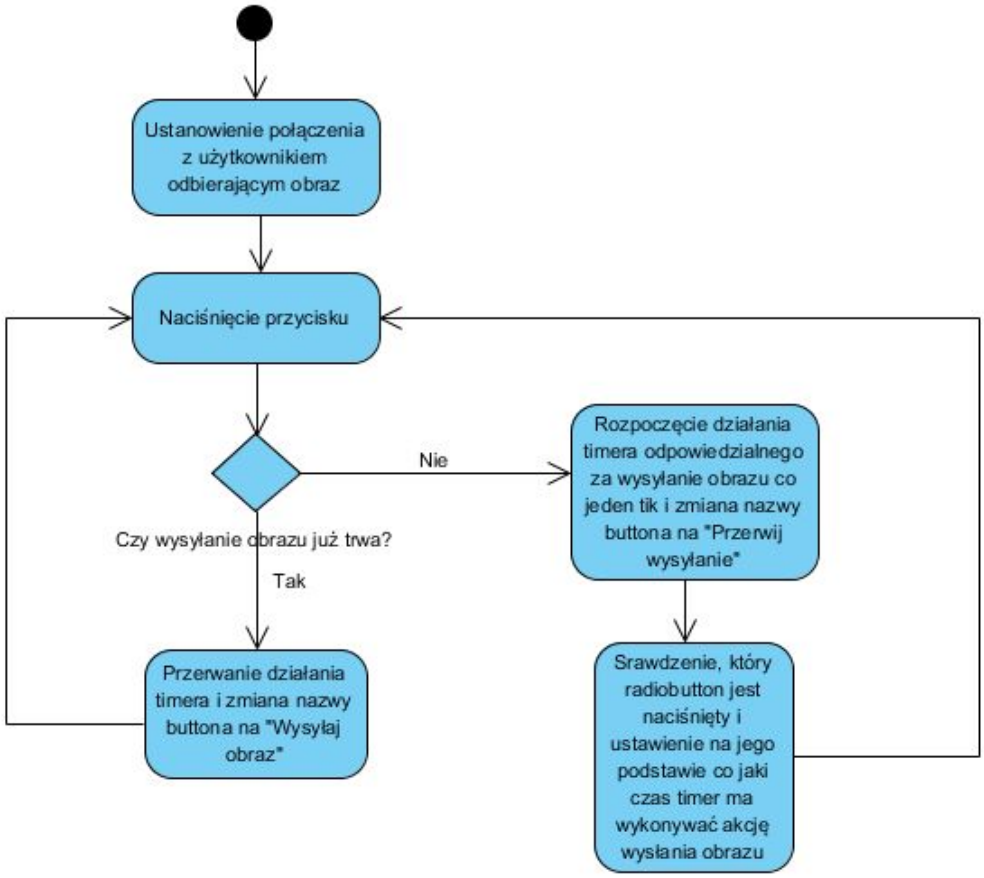


Rysunek 11: Diagram aktywności dla przycisku odpowiedzialnego za połączenie

5.1.4 Przycisk “Wysyłaj obraz”

Przycisk odpowiedzialny za proces rozpoczęcia wysyłania obrazu, przerwania wysyłania obrazu i kontynuacji przesyłania obrazu do użytkownika, który ten obraz odbiera z wybraną prędkością przesyłania danych

```
private void button2_Click(object sender, EventArgs e) //wyslij
obraz
      {
         if (button2.Text.StartsWith("Wys"))
         {
            timer1.Start();
            button2.Text = "Przerwij wysyłanie";
         }
         else
         {
            timer1.Stop();
            button2.Text = "Wysyłaj obraz";
         }
         if (radioButton1.Checked)
            timer1.Interval = 65;
         if (radioButton2.Checked)
            timer1.Interval = 32;
         if (radioButton3.Checked)
            timer1.Interval = 16;
      }
```

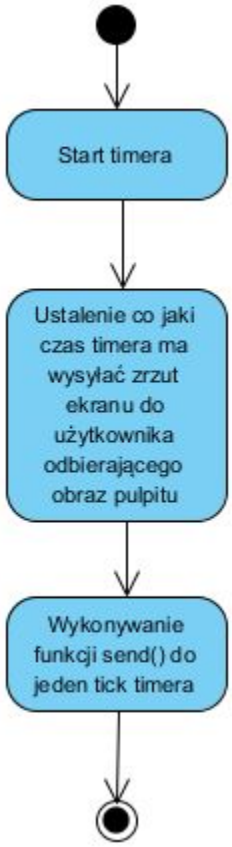


Rysunek 12: Diagram aktywności dla przycisku odpowiedzialnego za wysyłanie obrazu

5.1.5 Działanie timera

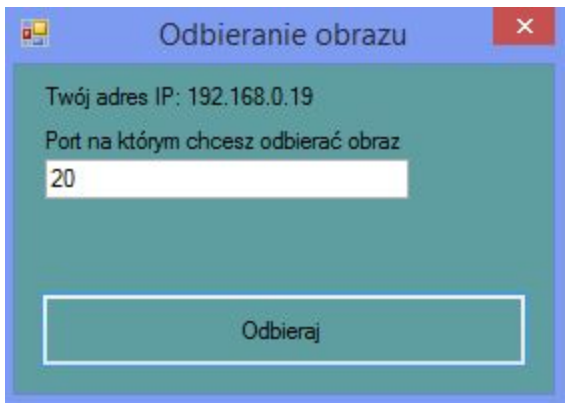
Timer co jeden tick, którego wartość jest ustalana przywyborze FPSów wykonuje funkcję send() i przesyła dane do użytkownika odbierającego obraz

```
private void timer1_Tick(object sender, EventArgs e)
      {
         try
         {
            send();
         }
         catch (Exception)
         {
            MessageBox.Show("Błąd wysyłania");
         }
      }
```



Rysunek 13: Diagram aktywności dla działania timera

5.2 Program do odbierania obrazu



Rysunek 14: Design pierwszego okna programu odpowiedzialnego za odbieranie obrazu

Rysunek 15: Design okna do wyświetlania odebranego obrazu

5.2.1 Funkcja GetLocalIPAddress()

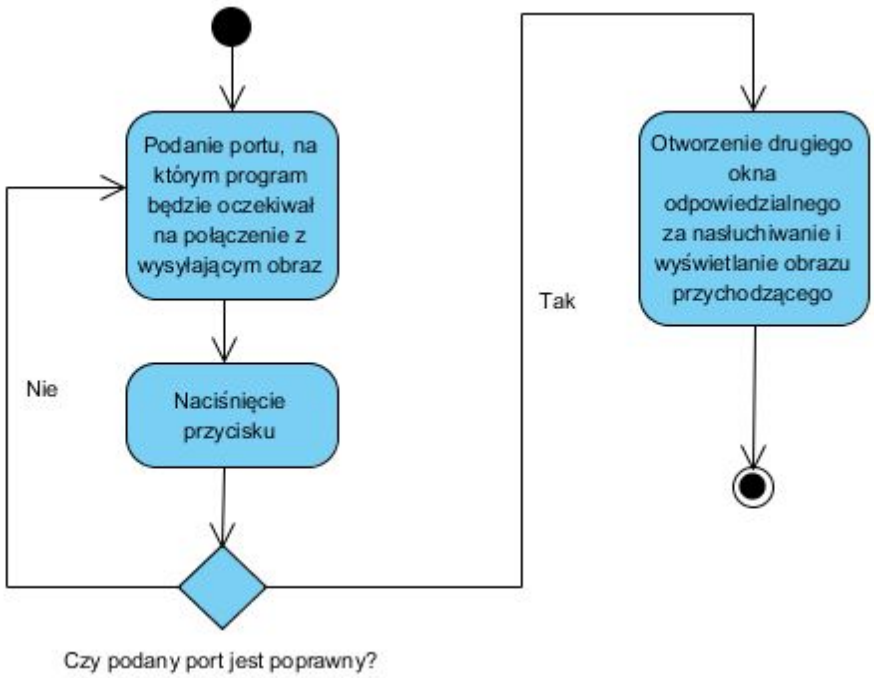
Funkcja, która zwraca nasz lokalny adres IP, który następnie będzie wyświetlony w 1 oknie, aby osoba, która wysyła nam obraz mogła poznać szybciej nasz adres

```
public static string GetLocalIPAddress()
        {
            var host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (var ip in host.AddressList)
            {
                if (ip.AddressFamily == AddressFamily.InterNetwork)
                {
                    return ip.ToString();
                }
            }
            throw new Exception("Local IP Address Not Found!");
        }
```

5.2.2 Przycisk “Odbieraj”

Służy do otworzenia drugiego okna odpowiedzialnego za nasłuchiwanie na połączenie i wyświetlanie odebranego obrazu, przekazuje mu też numer portu, na którym ma nasłuchiwać

```
private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            short port = short.Parse(textBox1.Text);
            new Form2(port).Show();
         }
         catch (Exception)
         {
            MessageBox.Show("Błędny port");
         }
      }
```

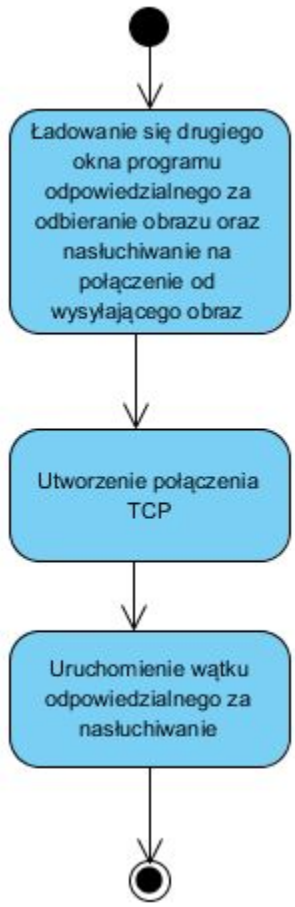


Rysunek 16: Diagram aktywności dla przycisku “Odbieraj”

5.2.3 Funkcja OnLoad(EventArgs e)

Funkcja, która uruchamia się przy włączaniu drugiego okna programu po naciśnięciu w pierwszym programie przycisku “Odbieraj”. Rozpoczyna nasłuchiwanie na przekazanym wcześniej numerze portu i oczekuje połączenia od dowolnego adresu IP użytkownika wysyłającego obraz.

```
protected override void OnLoad(EventArgs e)
      {
         base.OnLoad(e);
         listener = new TcpListener(IPAddress.Any, port);
         listening.Start();
      }
```



Rysunek 17: Diagram aktywności dla funkcji uruchamianej przy włączaniu okna odpowiedzialnego za nasłuchiwanie i odbieranie obrazu

5.2.4 Funkcja OnFormClosing(FormClosingEventArgs e)

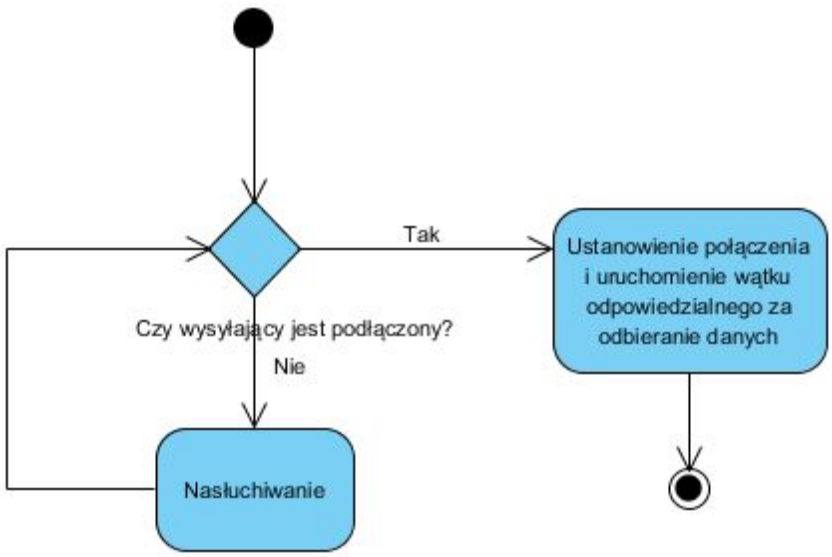
Funkcja uruchamiana przy wyłączaniu drugiego okna programu, zatrzymuje wątek nasłuchiwania

```
protected override void OnFormClosing(FormClosingEventArgs e)
        {
            base.OnFormClosing(e);
            stopListening();
        }
```

5.2.5 Funkcja startListening()

Funkcja odpowiedzialna za rozpoczęcie nasłuchiwania i włączenia wątku odbierania danych od użytkownika wysyłającego obraz

```
private void startListening()
        {
            while (!sender.Connected)
            {
                listener.Start();
                sender = listener.AcceptTcpClient();
            }
            receiving.Start();
        }
```



Rysunek 18: Diagram aktywności dla funkcji startListening()

5.2.6 Funkcja stopListening()

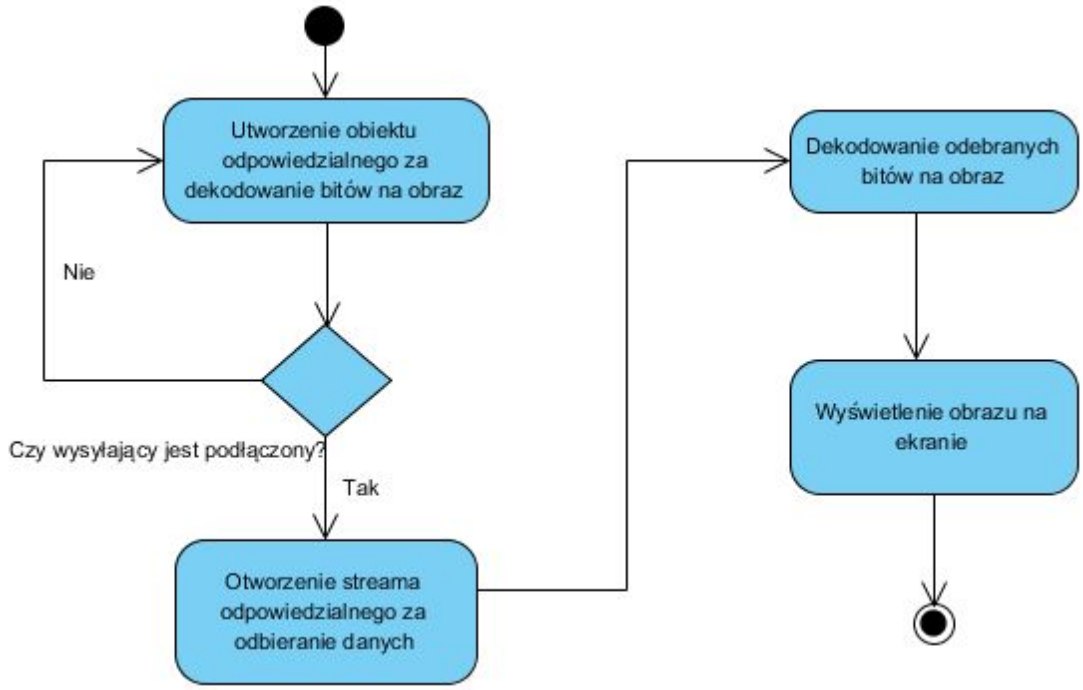
Funkcja, która wyłącza wątki odpowiedzialne za odbieranie i nasłuchiwanie

```
private void stopListening()
      {
         listener.Stop();
         sender = null;
         if (listening.IsAlive)
            listening.Abort();
         if (receiving.IsAlive)
            receiving.Abort();
      }
```

5.2.7 Funkcja startReceiving()

Funkcja odpowiedzialna za odbieranie strumienia bitów i konwersję na obraz oraz wyświetlenie tego obrazu na ekranie

```
private void startReceiving()
      {
         BinaryFormatter bf = new BinaryFormatter();
         while (sender.Connected)
         {
            data = sender.GetStream();
            pictureBox1.Image = (Image)bf.Deserialize(data);
         }
      }
```



Rysunek 19: Diagram aktywności dla funkcji startReceiving()

5.2.8 Przycisk “Pełny ekran”

Odpowiada za przełączanie drugiego okna na tryb pełnoekranowy

```
private void button1_Click_1(object sender, EventArgs e)
      {
         if (!fullscreen)
         {
            this.WindowState = FormWindowState.Normal;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.None;
            this.Bounds = Screen.PrimaryScreen.Bounds;
            fullscreen = true;
         }
         else
         {
            this.WindowState = FormWindowState.Maximized;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.Sizable;
            fullscreen = false;
         }
      }
```

5.3 Program do tworzenia dodatkowego pulpitu

Rysunek 20: Design programu tworzącego pulpit

5.3.1 Funkcja Stworz(string nazwa)

Służy do utworzenia nowego pulpitu o określonej nazwie korzystając z funkcji CreateDesktop zawartej w bibliotece user32.dll.

```
public static IntPtr Stworz(string nazwa)
      {
         return CreateDesktop(nazwa, IntPtr.Zero, IntPtr.Zero, 0, prawa,
IntPtr.Zero);
      }
```

5.3.2 Funkcja Otworz(string nazwa)

Funkcja otwiera wcześniej utworzony pulpit o określonej nazwie korzystając z funkcji OpenDesktop zawartej w bibliotece user32.dll.

```
public static IntPtr Otworz(string nazwa)
      {
         return OpenDesktop(nazwa, 0, true, prawa);
      }
```

5.3.3 Funkcja Przelacz(string nazwa)

Służy do przełączania się między pulpitami o zadanych nazwach korzystając z funkcji SwitchDesktop zawartej w bibliotece user32.dll.

```
public static bool Przelacz(string nazwa)
      {
         if (Otworz(nazwa) == IntPtr.Zero)
         {
            return false;
         }
         return SwitchDesktop(Otworz(nazwa));
      }
```

5.3.4 Funkcja CzyIsnieje(string nazwa)

Sprawdza czy istnieje pulpit o podanej nazwie.

```
public static bool CzyIstnieje(string nazwa)
      {
         if (Otworz(nazwa) == IntPtr.Zero)
         {
            return false;
         }
         return true;
      }
```

5.3.5 Funkcja NazwaPulpitu(IntPtr handle)

Funkcja zwracająca nazwę pulpitu, w razie niepowodzenia zwraca pusty string.

```
public static string NazwaPulpitu(IntPtr handle)
      {
         if (handle == IntPtr.Zero)
         {
            return string.Empty;
         }

         int dlugosc = 0;
         GetUserObjectInformation(handle, 2, IntPtr.Zero, 0, ref
dlugosc);
```

```
        IntPtr wskaznik = Marshal.AllocHGlobal(dlugosc);
        bool wynik = GetUserObjectInformation(handle, 2, wskaznik,
dlugosc, ref dlugosc);
        string nazwa = Marshal.PtrToStringAnsi(wskaznik);
        Marshal.FreeHGlobal(wskaznik);

        if (!wynik)
        {
            return string.Empty;
        }
        return nazwa;
    }
```

5.3.6 Funkcja StworzProces(string nazwa, string sciezka, string args)

Służy do uruchamiania procesów na wygenerowanym pulpicie (w szczególności do wystartowania “explorer.exe” odpowiedzialnego np. za pasek start).

```
public static bool StworzProces(string nazwa, string sciezka, string
args)
    {
        try
        {
            PROCESS_INFORMATION informacje = new
PROCESS_INFORMATION();
            STARTUPINFO startupInfo = new STARTUPINFO();
            startupInfo.cb = Marshal.SizeOf(startupInfo);
            startupInfo.lpDesktop = nazwa;
            string sciezka2 = string.Format("\"{0}\" {1}", sciezka, args);
            bool result = CreateProcess(sciezka, sciezka2, IntPtr.Zero,
IntPtr.Zero, true,
                NORMAL_PRIORITY_CLASS, IntPtr.Zero, null, ref
startupInfo, ref informacje);

            return result;
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex.ToString());
            return false;
        }
    }
```

5.3.7 Przyciski “Pulpit 1” oraz “Pulpit 2”

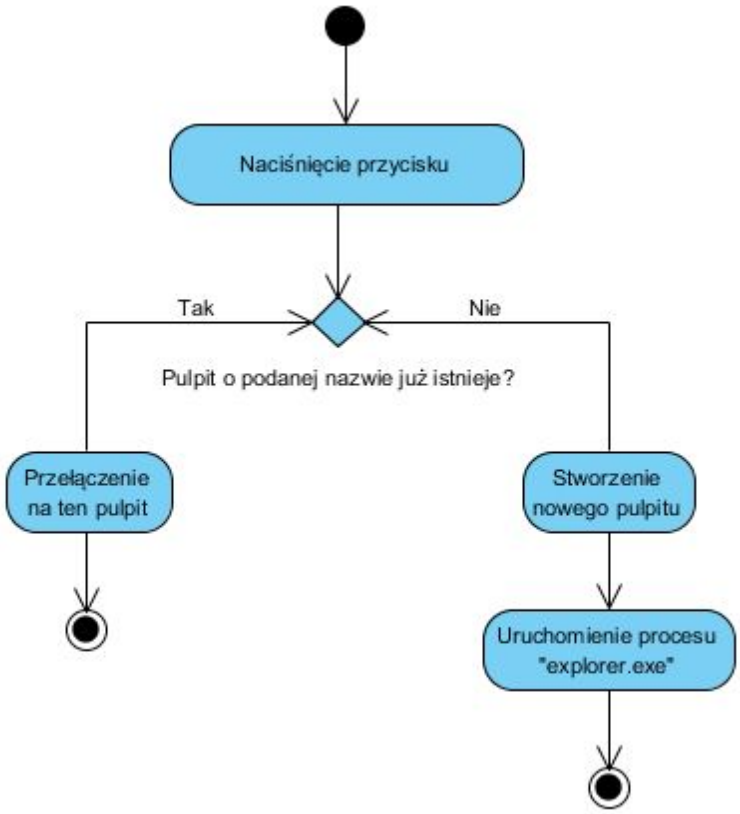
Odpowiada za procedury tworzenia oraz przełączania pulpitów.

```
private void button2_Click(object sender, EventArgs e)
      {
         string nazwa = "Nowy_Pulpit_Drugi";

         if (!Desktop.CzyIstnieje(nazwa))
         {
            Desktop.Stworz(nazwa);

            Desktop.StworzProces(nazwa,
Application.ExecutablePath, "start");
         }

         Desktop.Przelacz(nazwa);
      }
```

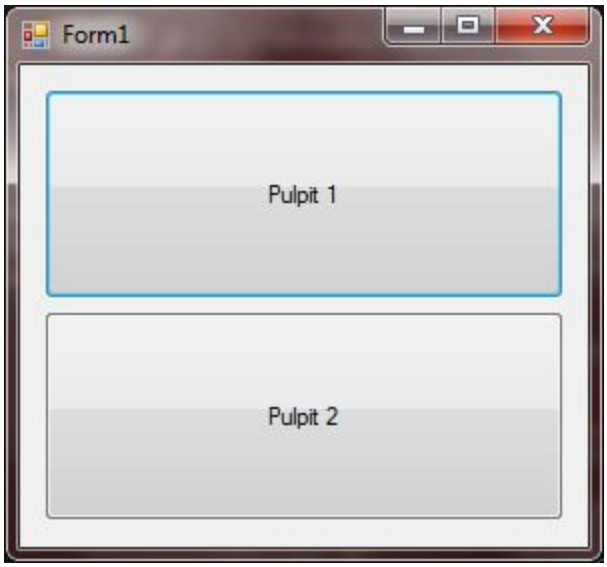


Rysunek 21: Diagram aktywności dla przycisku “Pulpit 2”

6. Użytkowanie

Przykładowa sesja użytkownika:

6.1 Utworzenie dodatkowego pulpitu



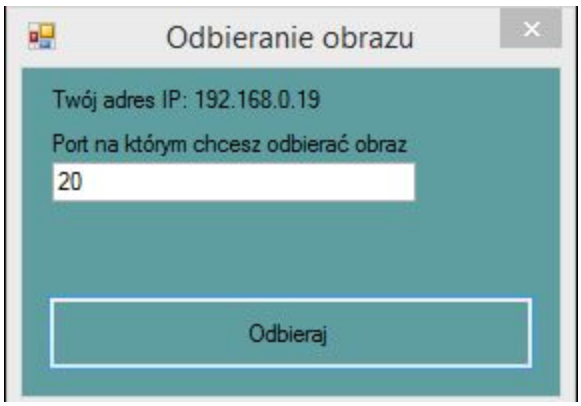
Rysunek 22: Naciśnięcie przycisku “Pulpit 2”

Rysunek 23: Uruchomienie się nowego pulpitu

6.2 Zestawienie połączenia

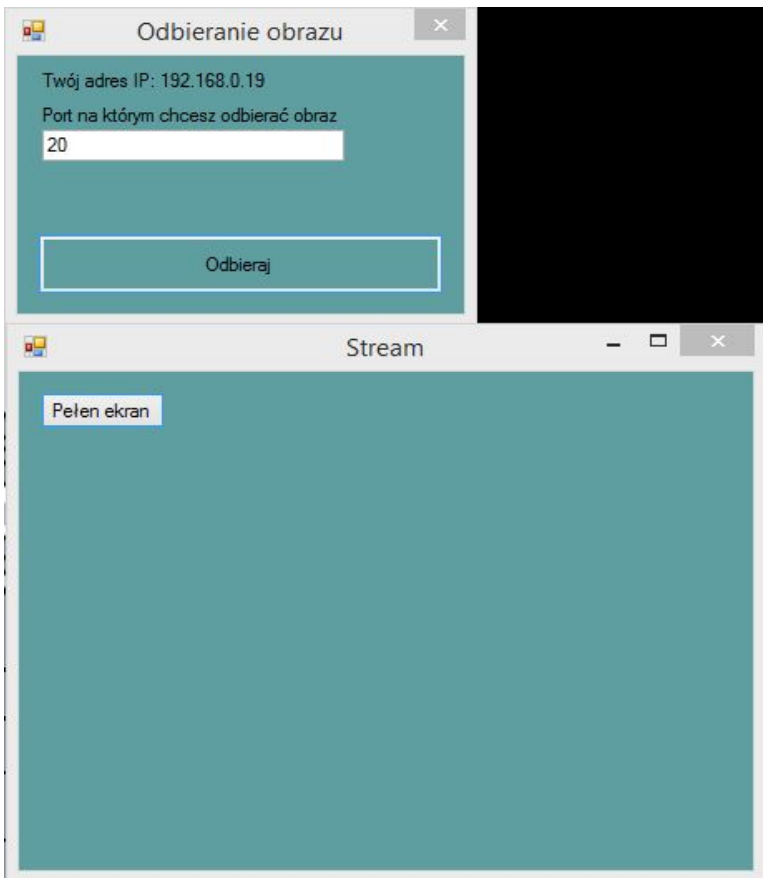
6.2.1 Zestawienie połączenia odbierającego obraz

- Wpisanie portu, na którym chce się oczekiwać na połączenie



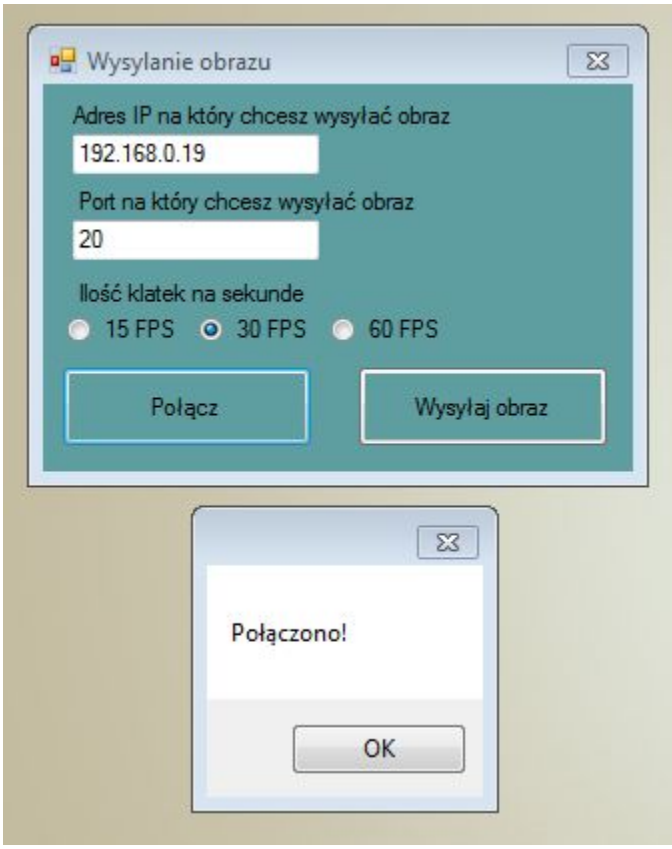
Rysunek 24: Wpisanie portu

- Kliknięcie przycisku odbieraj i oczekiwanie na wysyłającego obraz



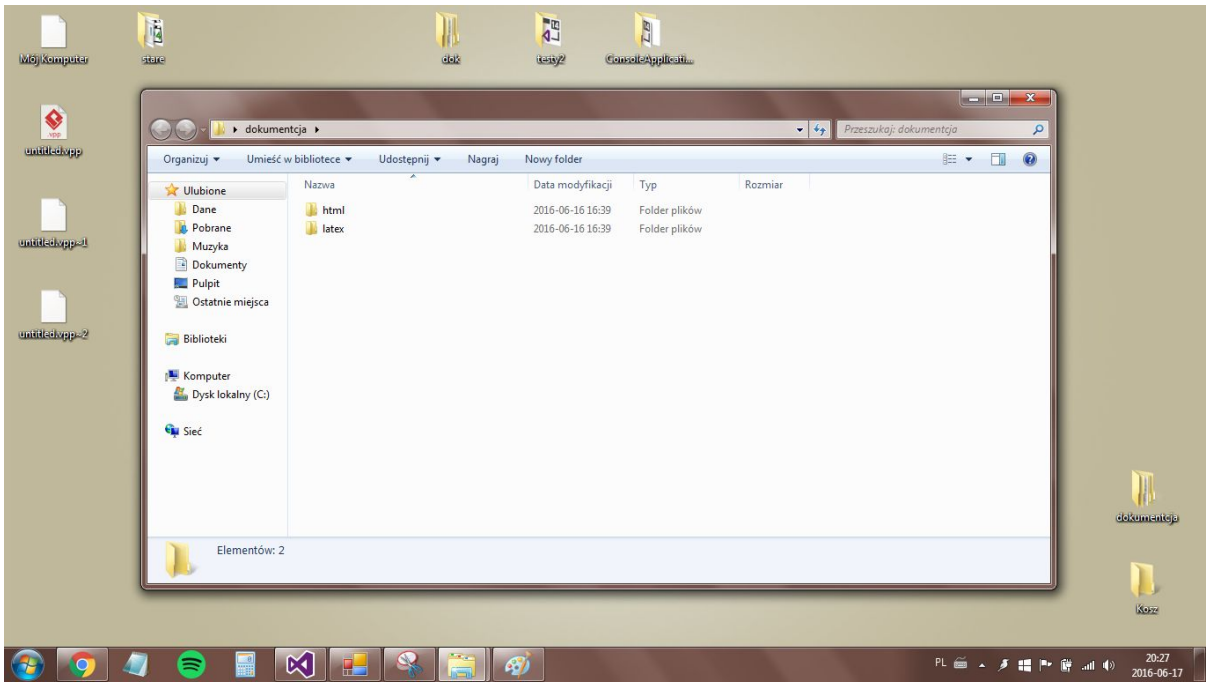
Rysunek 25: Oczekiwanie na połączenie z wysyłającym

6.2.2 Zestawienie połączenia wysyłającego obraz

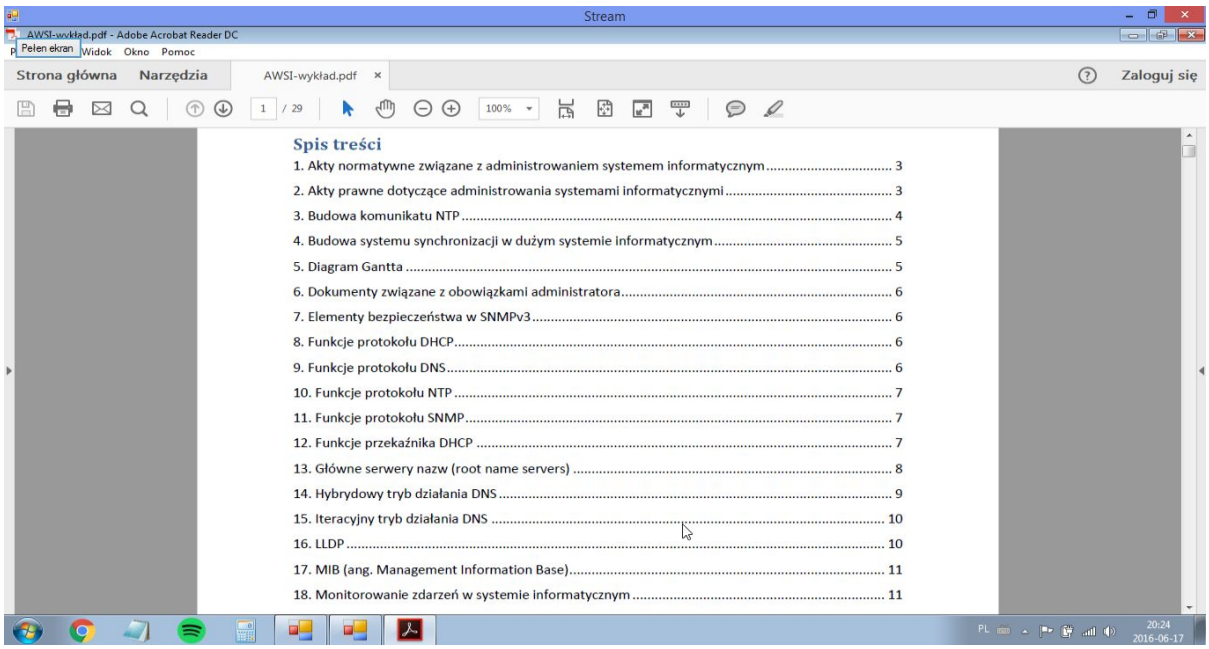


Rysunek 26: Wpisanie adresu IP oraz portu użytkownika odbierającego obraz i zestawienie połączenia

6.3 Korzystanie z dwóch pulpitów



Rysunek 27: Pulpit pierwszy



Rysunek 28: Pulpit drugi

7. Testy

Naszą aplikację poddaliśmy testom dotyczącym stabilności działania (zostały zastosowane mechanizmy informujące użytkownika o wykrytych błędach), funkcjonowania na różnych wersjach systemu operacyjnego Windows czy wielokrotnemu tworzeniu i przełączaniu utworzonych wcześniej pulpitów.

8. Podsumowanie

8.1 Uwagi ogólne

Przy tworzeniu naszego projektu utrwaliliśmy wiedzę na temat tworzenia aplikacji opartych na gniazdkach, zyskaliśmy więcej doświadczenia dotyczącego pracy w grupie czy dowiedzieliśmy się o możliwości tworzenia wielu pulpitów dla systemu Windows.

8.2 Możliwe kierunki rozwoju aplikacji

Projekt wciąż można rozbudować o:

- tworzenie większej ilości pulpitów
- zarządzanie drugim pulpitem bez każdorazowego przełączania się na niego (trudne do zrealizowania)
- płynniejsze przesyłanie obrazu
- wybór rozdzielczości wysyłanego ekranu poprzez interfejs użytkownika

8.3 Praca zespołowa

Dzięki zastosowaniu techniki Scrum przy niesprzyjających warunkach akademickich, gdzie wykonuje się kilka projektów w tym samym czasie nauczyliśmy się:

- efektywniej zarządzać czasem
- podziału większych zadań na kilka mniejszych
- słuchania drugiej osoby oraz przedstawiania jej własnego zdania
- wyróżniania zadań krytycznych (niezbędnych do funkcjonowania aplikacji) oraz skupieniu większej uwagi na nich

9. Wykorzystane materiały

- http://4programmers.net/
- http://virtualdesktop.codeplex.com/
- http://stackoverflow.com
- poradniki na YouTube