PresentationControl - sterowanie głosowe prezentacjami Microsoft PowerPoint za pomocą sensora Kinect

Magdalena Juchnowska gr. 7B 125601
Iwona Klimaszewska gr. 7B 125609

10 stycznia 2012

Spis treści

Część I
Wprowadzenie

1 O dokumencie

Istotnym aspektem naturalnego interfejsu użytkownika jest rozpoznawanie mowy. Doskonałym urządzeniem wejściowym dla aplikacji wykorzystujących wspomnianą funkcjonalność wydają się mikrofony sensora Kinect. Zestaw czterech mikrofonów zapewnia (w porównaniu z pojedynczym mikrofonem) znaczne ulepszenia związane m. in. z anulowaniem echa akustycznego czy też tłumieniem zakłóceń.

Niniejszy dokument stanowi specyfikację projektu realizowanego w ramach przedmiotu Oprogramowanie Systemowe na wydziale Elektroniki Telekomunikacji i Informatyki Politechniki Gdańskiej. Wykonana aplikacja demonstruje wykorzystanie Kinect SDK do rozpoznawania komend głosowych kontrolujących program PowerPoint.

2 Wymagania

Przed przystąpieniem do implementacji, należy się upewnić, czy na komputerze zainstalowano następujące komponenty:

- Microsotf Visual Studio 2010,
- Microsoft Office,
- Microsotf Kinect SDK, wersja 1.0 (beta 2),
- Microsoft Speech Platform - Software Development Kit (SDK), wersja 10.2 (edycja x86)
- Microsoft Speech Platform – Server Runtime, wersja 10.2 (edycja x86)
- Kinect for Windows Runtime Language Pack, wersja 0.9.

3 Funkcjonalność

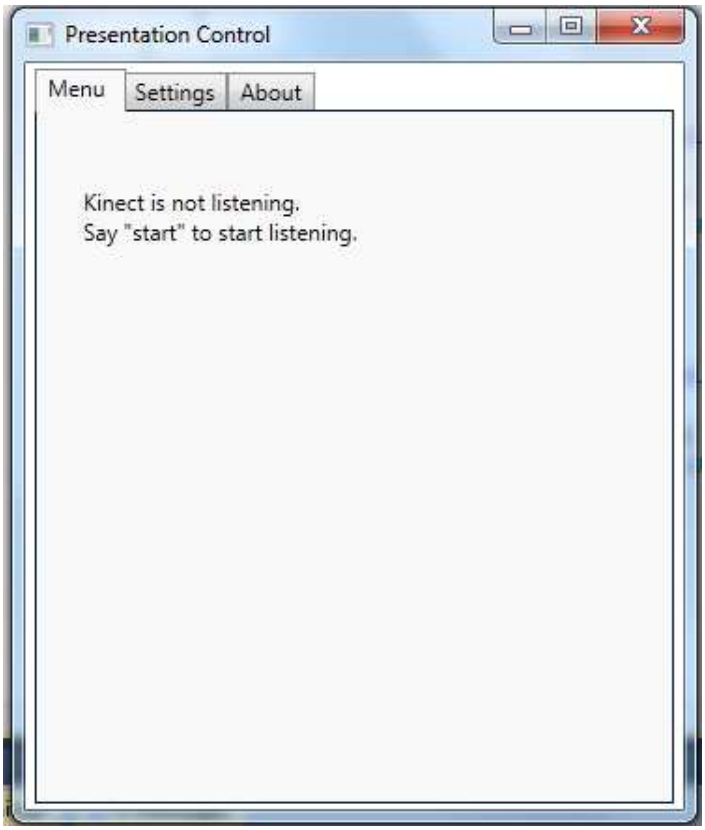
Aplikacje została stworzona w języku C# z wykorzystaniem interfejsu graficznego Windows Presentation Foundation (WPF) w środowisku Microsoft Visual Studio 2010.

Podstawowe funkcje:

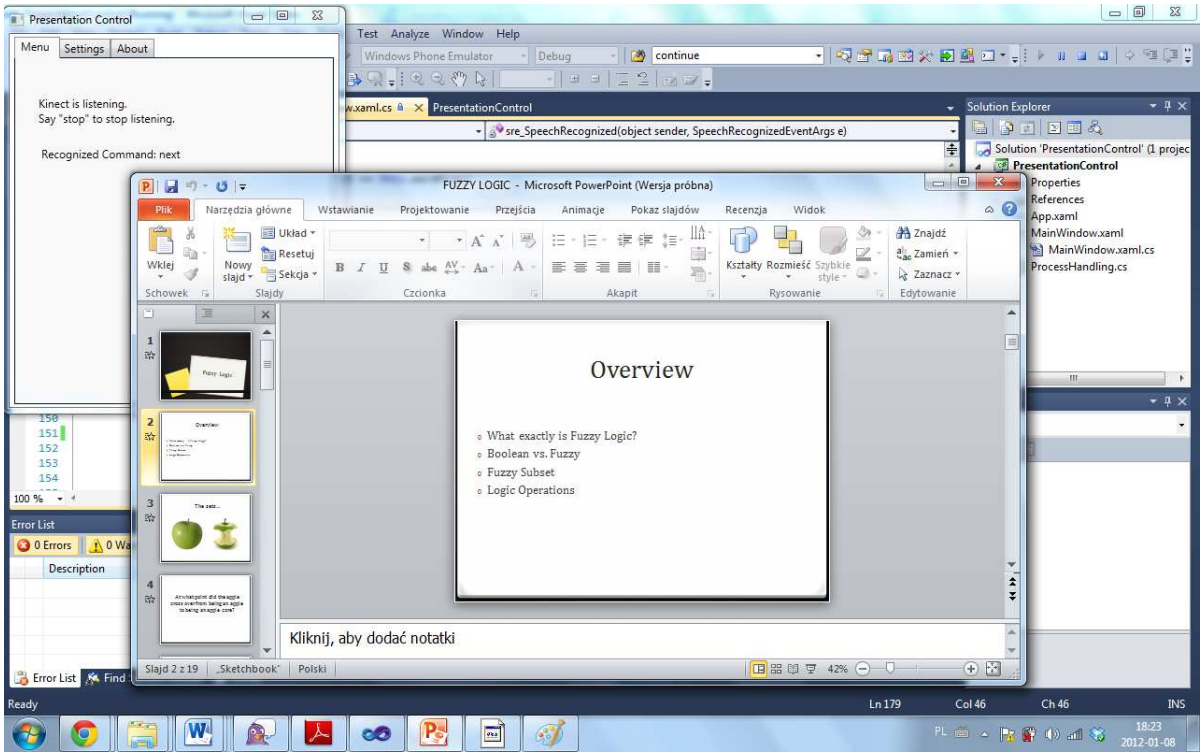
- rozpoznawanie komend głosowych:
  - polecenie włączające/wyłączające reagowanie na wypowiadane komendy - domyślnie *start/stop* (Screenshot 1),
  - polecenie uaktywniające okno Microsoft PowerPoint - domyślnie *powerpoint* (Screenshot 2),
  - polecenie uruchamiające pokaz slajdów - domyślnie *full screen*,
  - polecenia sterujące prezentacją programu Microsoft PowerPoint - domyślnie *previous/next*,

  - polecenie zamykające widok pełnoekranowy programu Microsoft PowerPoint - domyślnie *exit*,
  - polecenia ukrywające i pokazujące okno aplikacji - domyślnie *hide/show*.

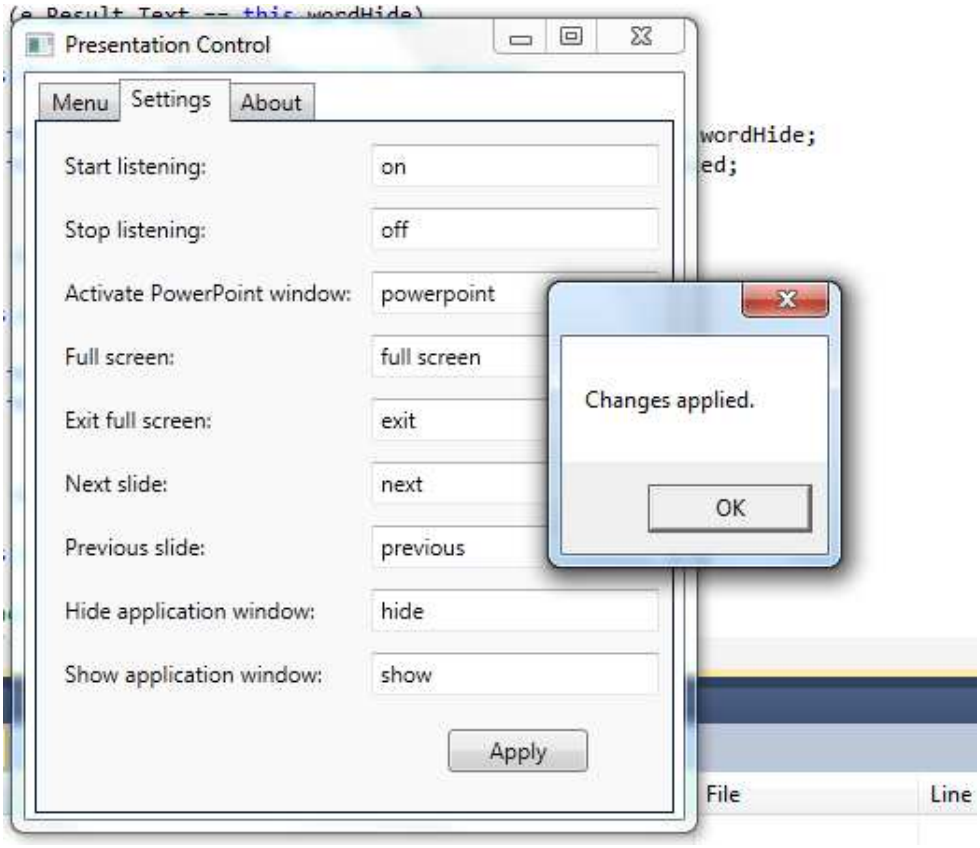
- definiowanie wyrażeń, które mogą zastąpić domyślnie ustawione polecenia głosowe (Screenshot 3).



Screenshot 1



Screenshot 2



Screenshot 3

Część II
Szczegóły dotyczące implementacji

4 Przebieg programu

Podstawowy przebieg programu można opisać następująco:

1. Stworzenie obiektu reprezentującego mikrofony sensora Kinect.
2. Stworzenie obiektu rozpoznającego mowę i wyspecyfikowanie gramatyki.
3. Reagowanie na zdefiniowane słowa.

5 Objaśnienia dotyczące kodu źródłowego

5.1 Klasa MainWindow

W klasie *MainWindow* zdefiniowane są główne funkcje aplikacji. Właściwą pracę aplikacji rozpoczyna wywołanie przy ładowaniu okna metody `StartSpeechRecognition`:

```
private void StartSpeechRecognition()
{
    t = new Thread(new ThreadStart(CaptureAudio));
    t.SetApartmentState(ApartmentState.MTA);
    t.Start();
}
```

Tworzy ona nowy wątek należący do modelu MTA (Multithreaded Apartments), ponieważ Audio API uruchamia DirectX Media Object w oddzielnym wątku. Metodą wywoływaną przy uruchomieniu wątku jest `CaptureAudio`.

```
private void CaptureAudio()
{
    this.source = new KinectAudioSource();
    this.source.FeatureMode = true;
    this.source.NoiseSuppression = true;
    this.source.AutomaticGainControl = false;
    this.source.SystemMode = SystemMode.OptibeamArrayOnly;
    . . .
```

Stworzony na początku obiekt `KinectAudioSource`, reprezentujący mikrofony sensora, wykorzystuje MSRKinectAudio DMO. Obiekt ten należy odpowiednio skonfigurować:

- *Feature mode* jest włączony, co pozwala na nadpisanie domyślnych ustawień właściwości DMO.
- *Noise Suppression* jest włączone - oznacza to tłumienie szumów.
- *Automatic gain control (AGC)* jest wyłączony (przy rozpoznawaniu mowy jest to konieczne).

- *System mode* jest ustawiony na `OptibeamArrayOnly`, co sprawia że mikrofony sensora działają jak pojedynczy mikrofon - wszystkie skupione są w kierunku źródła dźwięku.

Następnie tworzony jest silnik rozpoznawania mowy.

```
. . .
RecognizerInfo ri = SpeechRecognitionEngine.InstalledRecognizers().
Where(r => r.Id == RecognizerId).FirstOrDefault();
if (ri == null)
{
    return;
}
this.sre = new SpeechRecognitionEngine(ri.Id);
. . .
```

Metoda `InstalledRecognizers` zwraca listę silników rozpoznawania mowy dostępnych w systemie. Jako obiekt `RecognizerInfo` zwracany jest pierwszy z listy silnik i pobierając jego identyfikator tworzony jest silnik rozpoznawania mowy `SpeechRecognitionEngine`.

Kolejnym etapem jest wyspecyfikowanie słów, na które Kinect będzie reagował.

```
. . .
var words = new Choices();
words.Add(this.wordPowerpoint);
. . .
words.Add(this.wordShow);
var gb = new GrammarBuilder();
gb.Culture = ri.Culture;
gb.Append(words);
var g = new Grammar(gb);
sre.LoadGrammar(g);
this.sre.SpeechRecognized +=
    new EventHandler<SpeechRecognizedEventArgs>(sre_SpeechRecognized);
. . .
```

Obiekt `Choices` reprezentuje listę słów do rozpoznania, do której wyrazy mogą być dodawane poprzez wywołanie metody `Add` na tym obiekcie. Następnie tworzony jest obiekt `GrammarBuilder`, który pozwala skonstruować całą gramatykę (określamy kulturę oraz definiujemy jej elementy - wyżej zdefiniowane słowa - wywołując metodę `Append`). Na koniec ładujemy stworzoną według zdefiniowanych reguł gramatykę do silnika rozpoznawania mowy.

Zdarzenie `SpeechRecognized` występuje wtedy, gdy wypowiedziana komenda zostaje dopasowana do jednej ze zdefiniowanych wcześniej.

```
. . .
this.stream = this.source.Start();
this.sre.SetInputToAudioStream(this.stream, new SpeechAudioFormatInfo(
            EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));
while (shouldListen)
{
```

```
            sre.Recognize();
        }
    }
```

Przechwytywanie dźwięku z mikrofonów rozpoczyna się wraz z wywołaniem metody `Start` obiektu `KinectAudioSource`. Poprzez metodę `SetInputToAudioStream` specyfikowana jest charakterystyka źródła dźwięku. Metoda `Recognize` uruchamia pojedynczą operację rozpoznawania.

Obsługa zdarzenia rozpoznania komendy rozpoczyna się od sprawdzenia, czy poziom ufności (dopasowania) rozpoznania osiągnął pożądany próg (w naszym projekcie jest to 97%):

```
private void sre_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
{
    if (e.Result.Confidence > 0.97)
    {
        . . .
    }
}
```

Dopóki nie zostanie rozpoznane słowo kluczowe *start*, inne komendy są ignorowane. Po rozpoznaniu wywoływana jest metoda `StartListening` oraz stają się zawsze dostępne cztery komendy: *hide*, *show*, *powerpoint* oraz *stop* (która wywołuje metodę `StopListening`, wyłączającą rozpoznawanie). Po uaktywnieniu okna PowerPointa dostępne stają się cztery dodatkowe komendy: *full screen*, *next*, *previous* i *exit*, po rozpoznaniu których zostaje wywołana `CommandRecognized`. Symuluje ona naciśnięcie klawisza, wykorzystując metodę `Send` klasy `SendKeys`.

Jedną z opcji jest również zmiana domyślnie ustawionych komend, która jest realizowana poprzez wywołanie metody `ApplySettings`. Wywoływana jest w niej metoda `StartSpeechRecognition`, która na nowo definiuje ustawienia, ponieważ dodane słowo może nie znajdować się na liście zdefiniowanych w aktualnej gramatyce.

5.2 Klasa ProcessHandling

Klasa *ProcessHandling* umożliwia obsługę okien aplikacji. W aplikacji potrzebne będą pewne funkcje, znajdujące się w bibliotece `user32.dll`, dlatego należy je zaimportować:

```
[DllImport("user32.dll")]
static extern int GetForegroundWindow();

[DllImport("user32.dll")]
static extern bool SetForegroundWindow(IntPtr hWnd);

[DllImport("user32")]
static extern UInt32 GetWindowThreadProcessId(
    Int32 hWnd,out Int32 lpdwProcessId);
```

- `HWND GetForegroundWindow(void)` - zwraca uchwyt aktywnego okna aplikacji.

- `BOOL SetForegroundWindow(HWND hWnd)` - wysuwa na pierwszy plan wątek, który stworzył podane okno i aktywuje je.
- `DWORD GetWindowThreadProcessId(HWND hWnd, LPWDWORD lpdwProcessId)` - zwraca identyfikator wątku, który stworzył podane okno.

Stworzone zostały także dwie metody: `GetActiveWindowProcessName` oraz `SetActiveWindow`.

```
public static string GetActiveWindowProcessName()
{
    Int32 hwnd = GetForegroundWindow();
    string appProcessName =
        Process.GetProcessById(GetWindowProcessID(hwnd)).ProcessName;
    return appProcessName;
}
```

Metoda ta zwraca nazwę procesu aktywnego. Pobiera uchwyt aktywnego okna i na podstawie identyfikatora procesu uzyskuje nazwę.

```
public static void SetActiveWindow(string processName)
{
    Process[] processes = Process.GetProcessesByName(processName);
    foreach (Process p in processes)
    {
        IntPtr hWnd = p.MainWindowHandle;
        SetForegroundWindow(hWnd);
        }
}
```

Metoda ta na podstawie nazwy procesu ustawia okno aplikacji jako aktywne. Pobiera uchwyt procesu o danej nazwie i wysuwa okno na pierwszy plan.

5.3 Kod źródłowy

Pełny kod źródłowy aplikacji znajduje się w dodatku A.

Część III
Podsumowanie

6 Uwagi do implementacji

Silnik rozpoznawania mowy dla aktualnego SDK bazuje wyłącznie na języku angielskim i właśnie ten język (SR_MS_en-US_Kinect_10.0) zdefiniowano w omawianej aplikacji.
W zespole Microsoft trwają pracę nad możliwością rozpoznawania mowy dla innych kultur językowych.

W aplikacji zastosowano funkcję tłumienia zakłóceń - Noise Suppression, która przy niewielkim poziomie hałasu poprawia efektywność przechywtywania komend głosowych.
Działanie sensora, jeżeli w pomieszczeniu występują zakłócenia, może być niezadowalające. Kinect SDK Team zapewnia, że pracuje nad udoskonaleniem tej własności w wersji przeznaczonej dla zastosowań komercyjnych.

Akcja związana z danym poleceniem zostanie wykonana, jeżeli wyrażenie przechwycone przez urządzenie będzie w 97% zgodne ze zdefiniowanym. Zbyt niski poziom dopasowania mógłby powodować reakcje na za dużą liczbę podobnych bądź podobnie zaakcentowanych zwrotów, co wiązałoby się z niepoprawnym działaniem aplikacji. Poziom dopasowania 100% wskazany jest dla użytkowników biegle posługujących się językiem angielskim, aczkolwiek, również i w tym przypadku występuje problem błędnego rozpoznawania wypowiadanych komend.
Zespół odpowiedzialny za Kinect SDK zaleca używanie rozbudowanych wyrażeń, które pozwoli w pewnym stopniu na wyeliminowanie niektórych pomyłek przez dopasowanie losowo rozpoznanego słowa. Udoskonalenia rozpoznawania krótszych poleceń możemy spodziewać się w przyszłości.

7 Uwagi końcowe

Kinect rozpoznawany jest głównie ze względu na możliwość sterowania za pomocą gestów i ruchu ciała. W tym też obszarze spotkał się z szerokim gronem entuzjastów, którzy rozpoczęli testowanie usług dostarczanych przez kontroler jeszcze przed ukazaniem się pierwszych propozycji SDK.

Oprócz kamer rejestrujących użytkownika, Kinect wyposażony jest w zestaw czterech mikrofonów pozwalających na zastosowanie sensora do przechwytywania komend głosowych. W odróżnieniu od pojedynczego mikrofonu, urządzenia zakłada dostęp do zaawansowanych narzędzi odbierania, rozpoznawania i nagrywania dźwięków.

Niestety, można uznać to jedynie za zwiastun jego przyszłych możliwości. Nie zachwyca tłumienie zakłóceń, szybkość i jakość rozpoznawanie mowy, anulowanie echa akustycznego. Większość usług wydaje się być opatrzona etykietą wersji beta. W tym przypadku wydane SDK wymaga znacznego dopracowania. Niemniej jednak, nie można zarzucić - jest to genialnie pomyślane urządzenie, po którym w przyszłości możemy spodziewać się wielu usprawnień - i to nie tylko w dziedzinie gier komputerowych.

A Kod źródłowy

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Threading;
using System.IO;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Runtime.InteropServices;
using System.Diagnostics;
using Microsoft.Research.Kinect.Audio;
using Microsoft.Speech.Recognition;
using Microsoft.Speech.AudioFormat;

namespace PresentationControl
{
    public partial class MainWindow : Window
    {
        private Thread t;
        private const string RecognizerId = "SR_MS_en-US_Kinect_10.0";
        bool shouldListen = true;

        private string wordStart, wordStop, wordPowerpoint, wordFullscreen,
      wordExit, wordNext, wordPrevious, wordHide, wordShow;
        private KinectAudioSource source;
        private SpeechRecognitionEngine sre;
        private Stream stream;
        bool isControlling;
        public MainWindow()
        {
            InitializeComponent();
            this.Loaded += new RoutedEventHandler(Window_Loaded);
            this.Closed += new EventHandler(Window_Closed);
        }

        private void CaptureAudio()
        {
            this.source = new KinectAudioSource();
            this.source.FeatureMode = true;
            this.source.NoiseSuppression = true;
            this.source.AutomaticGainControl = false;
            this.source.SystemMode = SystemMode.OptibeamArrayOnly;
            RecognizerInfo ri = SpeechRecognitionEngine.InstalledRecognizers().
                Where(r => r.Id == RecognizerId).FirstOrDefault();
            if (ri == null)
            {
                return;
            }
            this.sre = new SpeechRecognitionEngine(ri.Id);

            var words = new Choices();

            words.Add(this.wordPowerpoint);
            words.Add(this.wordStart);
            words.Add(this.wordStop);
            words.Add(this.wordNext);
            words.Add(this.wordPrevious);
            words.Add(this.wordFullscreen);
            words.Add(this.wordExit);
            words.Add(this.wordHide);
            words.Add(this.wordShow);

            var gb = new GrammarBuilder();

            gb.Culture = ri.Culture;
```

```
    gb.Append(words);
    var g = new Grammar(gb);
    sre.LoadGrammar(g);
    this.sre.SpeechRecognized +=
new EventHandler<SpeechRecognizedEventArgs>(sre_SpeechRecognized);
    this.stream = this.source.Start();
    this.sre.SetInputToAudioStream(this.stream, new SpeechAudioFormatInfo(
        EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));
    while (shouldListen)
    {
        sre.Recognize();
    }
}

private void sre_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
{
    //jeżeli jest odpowiednio dokładnie rozpoznane słowo
    if (e.Result.Confidence > 0.97)
    {
        if (e.Result.Text == this.wordStart)
        {
            this.StartListening();
        }

        if (this.isControlling)
        {

            //zdobycie nazwy procesu aktywnej aplikacji
            string appProcessName = ProcessHandling.GetActiveWindowProcessName();
            //rozpoznawanie nawet gdy okno powerpointa nie jest aktywne
            if (e.Result.Text == this.wordPowerpoint)
            {
                this.Dispatcher.BeginInvoke((Action)delegate
                {
                    this.textBlock2.Text = "Recognized command: " + this.wordPowerpoint;
                });

                if (appProcessName.CompareTo("POWERPNT") != 0)
                {
                    ProcessHandling.SetActiveWindow("POWERPNT");
                }
            }
            else if (e.Result.Text == this.wordHide)
            {
                this.Dispatcher.BeginInvoke((Action)delegate
                {
                    this.textBlock2.Text = "Recognized command: " + this.wordHide;
                    this.WindowState = System.Windows.WindowState.Minimized;
                });
            }
            else if (e.Result.Text == this.wordShow)
            {
                this.Dispatcher.BeginInvoke((Action)delegate
                {
                    this.textBlock2.Text = "Recognized command: " + this.wordShow;
                    this.WindowState = System.Windows.WindowState.Normal;
                });
            }
            else if (e.Result.Text == this.wordStop)
            {
                this.StopListening();
            }
            //aktywne okno to powerpoint
            if (appProcessName.CompareTo("POWERPNT") == 0)
            {
                if (this.isControlling == true)
                {
                    if(e.Result.Text == this.wordNext)
                    {
                        CommandRecognized(this.wordNext, "{Right}");
                    }
                    else if(e.Result.Text == this.wordPrevious)
                    {
                        CommandRecognized(this.wordPrevious, "{Left}");
                    }
                    else if(e.Result.Text == this.wordFullscreen)
```

```
                    {
                        CommandRecognized(this.wordFullscreen, "{F5}");
                    }
                    else if(e.Result.Text == this.wordExit)
                    {
                        CommandRecognized(this.wordExit, "{ESC}");
                    }
                    else
                    {
                        this.Dispatcher.BeginInvoke((Action)delegate
                        {
                            textBlock2.Text = "";
                        });
                    }
                }
            }
        }
    }
}

private void StartListening()
{
    this.isControlling = true;
    this.Dispatcher.BeginInvoke((Action)delegate
    {

      this.textBlock1.Text = "Kinect is listening.\nSay \"" +
this.wordStop + "\" to stop listening.";

    });
}

private void StopListening()
{
    this.isControlling = false;
    this.Dispatcher.BeginInvoke((Action)delegate
    {

      this.textBlock1.Text = "Kinect is not listening.\nSay \"" +
  this.wordStart + "\" to start listening.";

    });
}

private void CommandRecognized(string word, string key)
{
    this.Dispatcher.BeginInvoke((Action)delegate
    {
        this.textBlock2.Text = "Recognized Command: " + word;
        System.Windows.Forms.SendKeys.SendWait(key);
        System.Windows.Forms.SendKeys.Flush();
    });
}

private void Window_Loaded(object sender, RoutedEventArgs e)
{

    this.textBox1.Text = this.wordStart = "start";
    this.textBox2.Text = this.wordStop = "stop";
    this.textBox3.Text = this.wordPowerpoint = "powerpoint";
    this.textBox4.Text = this.wordFullscreen = "full screen";
    this.textBox5.Text = this.wordExit = "exit";
    this.textBox6.Text = this.wordNext = "next";
    this.textBox7.Text = this.wordPrevious = "previous";
    this.textBox8.Text = this.wordHide = "hide";
    this.textBox9.Text = this.wordShow = "show";
    this.StopListening();
    StartSpeechRecognition();
}

private void StartSpeechRecognition()
{
    t = new Thread(new ThreadStart(CaptureAudio));
    t.SetApartmentState(ApartmentState.MTA);
    t.Start();
}
```

```
        private void ApplySettings(object sender, RoutedEventArgs e)
        {
            t.Abort();
            if (!this.ApplyWord(ref this.textBox1, ref this.wordStart)) return;
            if (!this.ApplyWord(ref this.textBox2, ref this.wordStop)) return;
            if (!this.ApplyWord(ref this.textBox3, ref this.wordPowerpoint)) return;
            if (!this.ApplyWord(ref this.textBox4, ref this.wordFullscreen)) return;
            if (!this.ApplyWord(ref this.textBox5, ref this.wordExit)) return;
            if (!this.ApplyWord(ref this.textBox6, ref this.wordNext)) return;
            if (!this.ApplyWord(ref this.textBox7, ref this.wordPrevious)) return;
            if (!this.ApplyWord(ref this.textBox8, ref this.wordHide)) return;
            if (!this.ApplyWord(ref this.textBox9, ref this.wordShow)) return;
            if (this.isControlling)
            {
                this.StartListening();
            }
            else
            {
                this.StopListening();
            }
            StartSpeechRecognition();
            MessageBox.Show("Changes applied.");
        }

        private bool ApplyWord(ref TextBox textBox, ref string word)
        {
            if (textBox.Text.Length == 0)
            {
                MessageBox.Show("Word " + word + " cannot be empty!");
                return false;
            }
            else
            {
                word = textBox.Text;
                return true;
            }
        }

        private void Window_Closed(object sender, EventArgs e)
        {
            this.isControlling = false;
        }
    }
}
```

ProcessHandling.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using System.Diagnostics;

namespace PresentationControl
{
    static class ProcessHandling
    {
        #region Private

        #region External Functions

        [DllImport("user32.dll")]
        static extern int GetForegroundWindow();

        [DllImport("user32.dll")]
        static extern bool SetForegroundWindow(IntPtr hWnd);

        [DllImport("user32")]
        static extern UInt32 GetWindowThreadProcessId(Int32 hWnd, out Int32 lpdwProcessId);

        #endregion

        #endregion

        #region Public

        /* Metoda zwracająca ID procesu */
        public static Int32 GetWindowProcessID(Int32 hwnd)
        {
            Int32 pid = 1;
            GetWindowThreadProcessId(hwnd, out pid);
            return pid;
        }

        /* Metoda zwracająca nazwę procesu aktywnej aplikacji */
        public static string GetActiveWindowProcessName()
        {
            Int32 hwnd = GetForegroundWindow();
            string appProcessName =
        Process.GetProcessById(GetWindowProcessID(hwnd)).ProcessName;
            return appProcessName;
        }

        /* Metoda ustawiająca aktywne okno */
        public static void SetActiveWindow(string processName)
        {
            Process[] processes = Process.GetProcessesByName(processName);

            foreach (Process p in processes)
            {
                IntPtr hWnd = p.MainWindowHandle;
                SetForegroundWindow(hWnd);
            }
        }

        #endregion
    }
}
```