Julita Borkowska

242817

**MVVM Light Toolkit**

**Przewodnik “krok po kroku”**

W celu lepszego zrozumienia elementów MVVM Light Toolkit przedstawionych w prezentacji, zostanie poniżej „krok po kroku” napisany program losujący sekretną liczbę, której odgadnięcie będzie zadaniem dla użytkownika.

Zaczynamy od instalacji dodatku:

Otwieramy Visual Studio, wybieramy Tools -> Extensions&Updates -> Online -> wpisujemy „MVVM Light Toolkit” i pobieramy, a następnie instalujemy.

Następnie tworzymy nowy projekt i wybieramy MVVMLight (WPF451).

Po stworzeniu projektu mamy już wygenerowane dwie klasy: MainViewModel.cs oraz ViewModelLocator.cs oraz pliki XAML: App.xaml i MainWindow.xaml.

Klasa MainViewModel odpowiada za przechowywanie danych i odpowiednie reagowanie na zachowania użytkownika.

Dodaję właściwość:

- SecretNumber (typu int) – reprezentuje ona liczbę, która należy odgadnąć;

- Suggestion (typu string) - w niej zapisywane będą sugestie dotyczące podanej przez użytkownika liczby;

- IsInProgress (typu bool) – określa, czy użytkownik jest w danej chwili w trakcie zgadywania;

- IsSolved (typu bool) – określa, czy sekretna liczba została odgadnięta;

Właściwości IsInProgress oraz IsSolved posłużą do pokazywania i ukrywania odpowiednich komunikatów).

We wszystkich powyższych podczas ustawiania wartości zostanie wywołana funkcja RaisePropertyChanged() z klasy bazowej (ViewModelBase), która opakowuje funkcjonalność interfejsu INotifyPropertyChanged. Interfejs ten ma zadanie informować klienta, że wartość właściwości (obiektu) uległa zmianie. Dzięki temu możemy korzystać z DataBindingu: przy każdej zmianie wartości zmiennej (zarówno w kodzie jak i przez użytkownika), jej wartość będzie na bieżąco aktualizowana.

Aby oszczędzić sobie konieczności każdorazowego wpisywania powtarzalnego kodu, najlepiej jest skorzystać ze snippetów (czyli właśnie takich gotowych fragmentów kodu). W celu ich zastosowania należy w nowej linii wpisać mvvminpc, a następnie nacisnąć TAB – pojawi się pożądany fragment kodu - konieczne jest już tylko podanie nazwy.

#region SecretNumber Property

public const string SecretNumberPropertyName = "SecretNumber";

private int \_secretNumber = 0;

public int SecretNumber

{

 get

 {

 return \_secretNumber;

 }

set

 {

 if (\_secretNumber == value)

{

 return;

}

\_secretNumber = value;

RaisePropertyChanged(SecretNumberPropertyName);

 }

}

#endregion

#region Suggestion

public const string SuggestionPropertyName = "Suggestion";

private string \_suggestion = "";

public string Suggestion

{

get

 {

 return \_suggestion;

}

 set

 {

if (\_suggestion == value)

 {

return;

 }

 \_suggestion = value;

 RaisePropertyChanged(SuggestionPropertyName);

}

}

#endregion

#region IsInProgress Property

public const string IsInProgressPropertyName = "IsInProgress";

private bool \_isInProgress = false;

public bool IsInProgress

{

get

 {

return \_isInProgress;

 }

 set

 {

 if (\_isInProgress == value)

 {

 return;

 }

 \_isInProgress = value;

 RaisePropertyChanged(IsInProgressPropertyName);

 }

}

#endregion

#region IsSolved Property

public const string IsSolvedPropertyName = "IsSolved";

private bool \_isSolved = false;

public bool IsSolved

{

get

{

 return \_isSolved;

 }

 set

 {

 if (\_isSolved == value)

 {

 return;

 }

 \_isSolved = value;

 RaisePropertyChanged(IsSolvedPropertyName);

 }

}

#endregion

**Tworzenie komend**

Zostaną teraz stworzone dwie komendy:

- Start – mająca za zadanie wylosować nową liczbę do odgadnięcia oraz stawienie właściwości określających stan zabawy na odpowiednie wartości logiczne;

- CheckGuess – mającą za zadanie sprawdzenie, czy podana przez użytkownika liczba jest prawidłowa oraz ewentualne podanie wskazówek, jeśli nie.

Za reprezentowanie komend w MVVM Light Toolkit odpowiada klasa RelayCommand. Opakowuje ona interfejs ICommand, dając jednocześnie wygodny sposób na tworzenie nowych komend (nie trzeba więc zagłębiać się w ich implementację). Istnieje również generyczna wersja RelayCommand, która pozwala na przekazywanie parametru do komendy.

Przy tworzeniu kodu dla komendy Start możemy skorzystać ze snippetu mvvmrelay, a dla komendy CheckGuess – mvvmrelaygeneric.

#region Start

private RelayCommand \_startCommand;

public RelayCommand Start

{

get

 {

 if (\_startCommand == null)

 {

 \_startCommand = new RelayCommand(

 () =>

 {

 SecretNumber = rand.Next(10) + 1;

 IsSolved = false;

 IsInProgress = true;

 Suggestion = "";

 });

}

 return \_startCommand;

}

}

#endregion

#region CheckGuess Command

private RelayCommand<int> \_checkGuess;

public RelayCommand<int> CheckGuess

{

get

 {

 if(\_checkGuess == null)

 \_checkGuess = new RelayCommand<int>(

x =>

 {

 if (x == \_secretNumber)

 {

 IsInProgress = false;

 IsSolved = true;

 }

 else if (x < \_secretNumber)

 Suggestion = "Liczba jest za mala.";

 else if (x > \_secretNumber)

 Suggestion = "Liczba jest za duza.";

 });

 return \_checkGuess;

}

}

#endregion

**Interfejs użytkownika**

Aby możliwe było zgadywanie liczby, podawanie wskazówek oraz wyniku zgadywania stworzono interfejs widoczny na poniższym obrazku:

Generuje go poniższy kod XAML, umieszczony w pliku MainWindow.xaml:

<Window x:Class="projekt05.MainWindow"

 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

 xmlns:local="clr-namespace:projekt05"

 xmlns:ignore="http://www.ignore.com"

 mc:Ignorable="d ignore"

 Height="300"

 Width="300"

 Title="MVVM Light Application"

 DataContext="{Binding Main, Source={StaticResource Locator}}">

 <Window.Resources>

 <BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>

 <local:StringToIntConverter x:Key="StringToIntConverter"/>

 </Window.Resources>

 <StackPanel>

 <StackPanel x:Name="spSecret" Orientation="Horizontal" Height="16" Visibility="{Binding IsSolved, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

 <TextBlock Height="16" TextWrapping="Wrap" Text="Gratulacje! Sekretna liczba to: " />

 <TextBlock x:Name="txblSecret" Height="16" TextAlignment="Center" Margin="20,0,0,0" TextWrapping="Wrap" Text="{Binding SecretNumber}"/>

 </StackPanel>

 <Button x:Name="btnStart" Content="Start" Margin="0,10,0,0" Command="{Binding Start, Mode=Default}"/>

 <StackPanel x:Name="spGuess" Visibility="{Binding IsInProgress, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

 <StackPanel Margin="0,10,0,0" Orientation="Horizontal">

 <TextBlock Margin="8,0,0,0" TextWrapping="Wrap" Text="Podaj liczbę: " />

 <TextBox x:Name="txbUserGuess" Height="24" TextAlignment="Center" Margin="25,0,0,0" TextWrapping="Wrap" Text="0" Width="50" HorizontalAlignment="Center" VerticalAlignment="Center"/>

 </StackPanel>

 <Button x:Name="btnCheck" Content="Sprawdź" Margin="0,10,0,0" Command="{Binding CheckGuess}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbUserGuess}"/>

 <TextBlock x:Name="txblSuggestion" Margin="0,15,0,0" TextWrapping="Wrap" TextAlignment="Center" Text="{Binding Suggestion}"/>

 </StackPanel>

 </StackPanel>

</Window>

Podczas analizy powyższego kodu szczególną uwagę należy zwrócić na linijki, w których pojawia się słowo „Binding”. Dotyczy ono bowiem wspomnianej na początku tego opracowania możliwości DataBindingu, czyli tworzenia odpowiednich powiązań między widokiem i obiektem ViewModel.

Jeśli chodzi o wiązanie właściwości lub komend, zastosowanie DataBinding jest bardzo proste:

<Button x:Name="btnStart" Content="Start" Command="{Binding Start}"/>

Podobnie jak dla przycisku Start rozpoczynającego grę wygląda to dla pól tekstowych txblSecret oraz txblSuggestion, które są powiązane z właściwościami SecretNumber oraz Suggestion.

Powiązanie jest nieco trudniejsze w przypadku przycisku, którym użytkownik sprawdza, czy odgadł sekretną liczbę:

<Button x:Name="btnCheck" Content="Sprawdź" Command="{Binding CheckGuess}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbUserGuess}"/>

Jak widzimy powyżej, w chwili naciśnięcia przycisku, Binding ustawia wykonywaną komendę na CheckGuess i podaje właściwość Text pola tekstowego txbUserGuess jako jej parametr. Problemem jest jednak to, iż właściwość Text jest typu string, a komenda CheckGuess przyjmuje jako parametr obiekt typu int. Konieczna jest więc konwersja pomiędzy oba typami. Stworzono do tego celu obiekt StringToIntConverter dokonujący zmiany typów w obie strony. Dziedziczy on po interfejsie IValueConverter i implementuje jego dwie metody: Convert oraz ConvertBack.

[ValueConversion(typeof(string), typeof(int))]

public class StringToIntConverter : IValueConverter

{

#region IValueConverter Members

 public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

 string str = value.ToString();

 if (string.IsNullOrEmpty(str))

 return 0;

 else

try

{

 int result = 0;

 int.TryParse(str, out result);

 return result;

 }

 catch (ArgumentException)

 {

 return 0;

 }

}

 public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

 {

 return value.ToString();

}

#endregion

}

Aby móc korzystać z nowo stworzonego konwertera, dodajemy go jako zasób obecnego widoku:

<Window.Resources>

 <local:StringToIntConverter x:Key="StringToIntConverter"/>

</Window.Resources>

Aby użytkownik nie dowiedział się jaka jest nasza sekretna liczba, należy prawidłowo ukryć oba panele. W tym celu należy powiązać ich właściwości Visibility z odpowiednimi właściwościami obiektu ViewModel: IsInProgress oraz IsSolved:

<StackPanel x:Name="spSecret" Visibility="{Binding IsSolved, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

<StackPanel x:Name="spGuess" Visibility="{Binding IsInProgress, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

Wykorzystany w powyższych fragmentach kodu konwerter BooleanToVisibilityConverter jest dostępny domyślnie, trzeba tylko dodać go jako zasób do obecnego widoku, podobnie jak uczyniono to z konwerterem StringToIntConverter.

**EventToCommand**

W kontrolkach WPF’a komendy podpinane są domyślnie do jednego z góry przewidzianego zdarzenia, np. dla przycisku jest to odpowiednik onClick. Nie ma możliwości podłączenia komendy do innych zdarzeń. Dzięki MVVM Light Toolkit, istnieje możliwość podłączenia komend z obiektu ViewModel do dowolnych zdarzeń kontrolki bez angażowania do tego jakiegokolwiek Code Behind – wszystko pozostaje więc w zgodzie z założeniami MVVM.

<Window x:Class="projekt05.MainWindow"

 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

 xmlns:vm="clr-namespace:projekt05"

 xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"

 xmlns:GalaSoft\_MvvmLight\_Command="http://www.galasoft.ch/mvvmlight"

 xmlns:ignore="http://www.ignore.com"

 mc:Ignorable="d ignore"

 Height="300"

 Width="300"

 Title="MVVM Light Application"

 DataContext="{Binding Main, Source={StaticResource Locator}}">

 <Window.Resources>

 <BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>

 <vm:StringToIntConverter x:Key="StringToIntConverter"/>

 </Window.Resources>

 <StackPanel>

 <StackPanel x:Name="spSecret" Orientation="Horizontal" Height="16" Visibility="{Binding IsSolved, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

 <TextBlock Height="16" TextWrapping="Wrap" Text="Gratulacje! Sekretna liczba to: " />

 <TextBlock x:Name="txblSecret" Height="16" TextAlignment="Center" Margin="20,0,0,0" TextWrapping="Wrap" Text="{Binding SecretNumber}"/>

 </StackPanel>

 <Button x:Name="btnStart" Content="Start" Margin="0,10,0,0" Command="{Binding Start, Mode=Default}"/>

 <StackPanel x:Name="spGuess" Visibility="{Binding IsInProgress, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

 <StackPanel Margin="0,10,0,0" Orientation="Horizontal">

 <TextBlock Margin="8,0,0,0" TextWrapping="Wrap" Text="Podaj liczbę: " />

 <TextBox x:Name="txbUserGuess" Height="24" TextAlignment="Center" Margin="25,0,0,0" TextWrapping="Wrap" Text="0" Width="50" HorizontalAlignment="Center" VerticalAlignment="Center">

 <i:Interaction.Triggers>

 <i:EventTrigger EventName="LostFocus">

 <GalaSoft\_MvvmLight\_Command:EventToCommand Command="{Binding CheckGuess, Mode=Default}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbUserGuess}"/>

 </i:EventTrigger>

 </i:Interaction.Triggers>

 </TextBox>

 </StackPanel>

 <Button x:Name="btnCheck" Content="Sprawdź" Margin="0,10,0,0" Command="{Binding CheckGuess}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbUserGuess}"/>

 <TextBlock x:Name="txblSuggestion" Margin="0,15,0,0" TextWrapping="Wrap" TextAlignment="Center" Text="{Binding Suggestion}"/>

 </StackPanel>

 </StackPanel>

</Window>

W przytoczonej powyżej zawartości pliku MainWindow.xaml zaznaczono na szaro dodane fragmenty. W przykładzie do zdarzenia LostFocus podpinane jest sprawdzenie, czy podana przez użytkownika liczba jest wylosowanym sekretnym numerem – umożliwia to sprawdzanie liczby zarówno za pomocą przycisku „Sprawdź”, jak i kliknięcia TAB na klawiaturze.

Do istniejącego TextBoxa odpowiedzialnego za wprowadzanie przez użytkownika liczby podpinamy nowy event trigger. Jest to trigger reagujący na zajście zdarzenia LostFocus – za pomocą jednego znacznika możemy podpiąć komendę do tego triggera (w sposób w jaki wcześniej podpięliśmy ją do przycisku).

**Messenger**

Z założenia obiekt Messenger został dodany do MVVM Light Toolkit jako narzędzie do wszechstronnej komunikacji i to nie tylko pomiędzy obiektami ViewModel, ale między dowolnymi klasami. Idea opiera się na statycznym obiekcie Messenger, który udostępnia mechanizmy do wysyłania komunikatów oraz rejestrowania akcji reagujących na konkretne komunikaty.

Celem zademonstrowania działania obiektu Messenger, stworzony zostanie nowy obiekt ViewModel, który będzie panelem opcji programu, posiadającym jednak tylko jedną opcję – mianowicie wybór maksymalnego zakresu wartości, z jakiego można wylosować sekretną liczbę.

Po zatwierdzeniu zmian zaktualizowany zostanie maksymalny zakres liczb, a dzięki DataBinding informacja o aktualnym maksymalnym zakresie zostanie odświeżona.

Na początek należy dodać w projekcie nowy folder View i stworzyć w nim nowy interfejs użytkownika OptionsView.xaml (klikamy na projekcie prawym klawiszem i wybieramy Add -> User control):

<UserControl x:Class="projekt05.View.OptionsView"

 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

 mc:Ignorable="d"

 xmlns:local="clr-namespace:projekt05"

 d:DesignHeight="300" d:DesignWidth="300">

 <UserControl.Resources>

 <local:StringToIntConverter x:Key="StringToIntConverter"/>

 </UserControl.Resources>

 <StackPanel>

 <StackPanel Orientation="Horizontal" Margin="0,10,0,0">

 <TextBlock TextWrapping="Wrap" Margin="5,0,0,0" HorizontalAlignment="Center" VerticalAlignment="Center" Text="Maksymalna liczba z zakresu: "></TextBlock>

 <TextBox HorizontalAlignment="Center" TextWrapping="Wrap" Margin="10,0,0,0" VerticalAlignment="Center" Name="txbMaxNumber" Text="10"></TextBox>

 </StackPanel>

 <Button Content="Zapisz opcje" Margin="0,10,0,0" VerticalAlignment="Center" Command="{Binding SaveOptions}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbMaxNumber}"/>

 </StackPanel>

</UserControl>

Następnie tworzymy odpowiadający temu widokowi obiekt ViewModel o nazwie OptionsViewModel.cs (dziedziczący oczywiście po ViewModelBase). Dodajemy jedną prostą komendę SaveOptions, która zostanie podpięta do przycisku „Zapisz opcje”:

public class OptionsViewModel : ViewModelBase

{

 #region Save Options Command

 private RelayCommand<int> \_saveCommand;

 public RelayCommand<int> SaveOptions

 {

 get

 {

 if (\_saveCommand == null)

 \_saveCommand = new RelayCommand<int>(

 x =>

 {

 Messenger.Default.Send<ChangeOptionsMessage>(new ChangeOptionsMessage(x));

 });

 return \_saveCommand;

 }

 }

 #endregion

 public OptionsViewModel()

 {

 }

}

Jej jedynym przeznaczeniem jest, aby za pomocą obiektu Messenger przesłać dalej otrzymany argument. Wynika on z DataBindingu samego przycisku i jest po prostu zawartością pola tekstowego odpowiadającego za maksymalną liczbę (odpowiednio przekonwertowaną na liczbę całkowitą):

<Button Content="Zapisz opcje" Command="{Binding SaveOptions}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbMaxNumber}"/>

Przesyłany komunikat jest typu ChangeOptionsMessage, który to jest naszym własnym typem komunikatu – dzięki temu możemy dostosować go tak, aby zawierał tylko takie informacje, jakich potrzebujemy w aktualnym scenariuszu. Dla wygody dziedziczy on po typie MessageBase z MVVM Light Toolkit i zawiera jedną dodatkową właściwość przechowującą maksymalną liczbę do wylosowania.

Dodajemy więc do projektu folder Messages, a w nim klasę ChangeOptionsMessages.cs:

public class ChangeOptionsMessage : MessageBase

{

 public int MaxRange { get; set; }

 public ChangeOptionsMessage(int \_maxRange)

 {

 MaxRange = \_maxRange;

 }

}

Teraz musimy jeszcze odebrać wysłany komunikat w głównym oknie. Zmieniamy więc plik MainViewModel.cs dodając właściwość MaxRange, rejestrując w konstruktorze akcję reagującą na wysłanie komunikatu typu ChangeOptionsMessage oraz pisząc prosta funkcję obsługującą komunikat HandleChangeOptions() (zmienione elementy zostały zaznaczone na szaro):

public class MainViewModel : ViewModelBase

{

 private Random rand = new Random();

#region SecretNumberProperty

#region Suggestion Property

#region IsInProgress Property

#region MaxRange Property

 public const string MaxRangePropertyName = "MaxRange";

 private int \_maxRange = 10;

 public int MaxRange

 {

 get

 {

 return \_maxRange;

 }

 set

 {

 if (\_maxRange == value)

 return;

 //var oldValue = \_maxRange;

 \_maxRange = value;

 // Update bindings, no broadcast

 RaisePropertyChanged(MaxRangePropertyName);

 }

 }

 #endregion

 #region IsSolved

 #region Start

 #region CheckGuess Command

 public MainViewModel()//IDataService dataService)

 {

 Messenger.Default.Register<Messages.ChangeOptionsMessage>(this, HandleChangeOptions);

 if (IsInDesignMode)

 {

 IsSolved = true;

 IsInProgress = true;

 }

 else

 {

 // Code runs "for real": Connect to service, etc...

 }

 private void initGuess()

 {

 SecretNumber = rand.Next(MaxRange) + 1;

 IsSolved = false;

 IsInProgress = true;

 Suggestion = "";

 }

 private void HandleChangeOptions(Messages.ChangeOptionsMessage m)

 {

 this.MaxRange = m.MaxRange;

 initGuess();

 }

 }

}

Funkcja initGuess() rozpoczyna turę zgadywania z już zmienionym zakresem.

Na koniec pozostaje nam jeszcze wprowadzenie drobnych zmian w pliku MainView.xaml, tak, aby możliwe było zobaczenie nowych elementów widoku (zmienione elementy zaznaczono na szaro):

<Window x:Class="projekt05.MainWindow"

 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

 xmlns:local="clr-namespace:projekt05"

 xmlns:localView="clr-namespace:projekt05.View"

 xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"

 xmlns:GalaSoft\_MvvmLight\_Command="http://www.galasoft.ch/mvvmlight"

 xmlns:ignore="http://www.ignore.com"

 mc:Ignorable="d ignore"

 Height="300"

 Width="300"

 Title="MVVM Light Application"

 DataContext="{Binding Main, Source={StaticResource Locator}}">

 <Window.Resources>

 <BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>

 <local:StringToIntConverter x:Key="StringToIntConverter"/>

 </Window.Resources>

 <DockPanel HorizontalAlignment="Left" LastChildFill="False" Margin="5,0">

 <StackPanel DockPanel.Dock="Top">

 <StackPanel x:Name="spSecret" Orientation="Horizontal" Height="16" Visibility="{Binding IsSolved, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

 <TextBlock Height="16" TextWrapping="Wrap" Text="Gratulacje! Sekretna liczba to: " />

 <TextBlock x:Name="txblSecret" Height="16" TextAlignment="Center" Margin="20,0,0,0" TextWrapping="Wrap" Text="{Binding SecretNumber}"/>

 </StackPanel>

 <Button x:Name="btnStart" Content="Start" Margin="0,10,0,0" Command="{Binding Start, Mode=Default}"/>

 <StackPanel x:Name="spGuess" Visibility="{Binding IsInProgress, Converter={StaticResource BooleanToVisibilityConverter}, Mode=OneWay}">

 <StackPanel Margin="0,10,0,0" Orientation="Horizontal">

 <TextBlock Margin="8,0,0,0" TextWrapping="Wrap" Text="Podaj liczbę: " />

 <TextBox x:Name="txbUserGuess" Height="24" TextAlignment="Center" Margin="25,0,0,0" TextWrapping="Wrap" Text="0" Width="50" HorizontalAlignment="Center" VerticalAlignment="Center">

 <i:Interaction.Triggers>

 <i:EventTrigger EventName="LostFocus">

 <GalaSoft\_MvvmLight\_Command:EventToCommand Command="{Binding CheckGuess, Mode=Default}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbUserGuess}"/>

 </i:EventTrigger>

 </i:Interaction.Triggers>

 </TextBox>

 </StackPanel>

 <Button x:Name="btnCheck" Content="Sprawdź" Margin="0,10,0,0" Command="{Binding CheckGuess}" CommandParameter="{Binding Text, Converter={StaticResource StringToIntConverter}, ElementName=txbUserGuess}"/>

 <TextBlock x:Name="txblSuggestion" Margin="0,15,0,0" TextWrapping="Wrap" TextAlignment="Center" Text="{Binding Suggestion}"/>

 </StackPanel>

 </StackPanel>

 <localView:OptionsView VerticalAlignment="Bottom" DockPanel.Dock="Bottom" Margin="0,0,0,10"/>

 </DockPanel>

</Window>

Po wprowadzeniu powyższych zmian interfejs użytkownika prezentuje się następująco:

**Przewodnik przygotowano na podstawie:**

**[1]** http://cieplok.blogspot.com/2010/09/podstawy-mvvm-light-toolkit-czesc-1.html

**[2]** http://cieplok.blogspot.com/2010/10/podstawy-mvvm-light-toolkit-czesc-2.html