GUI Lab: Agent Assignment 2

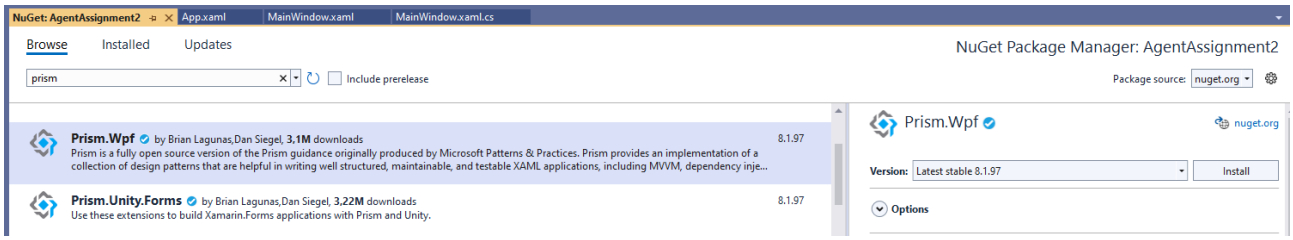
Kommentarer til løsningsforslag

Lab 1

Opretter et nyt projekt (WPF app) og opretter mapper i Visual Studio for Models, ViewModel og Views. Flytter MainWindow til Views mappen og husker at ændre url i filen App.xaml:

```
StartupUri="/Views/MainWindow.xaml"
```

Jeg bruger "Manage NuGet Packages" til at installere **Prism.WPF** (og den inkluderer Prism.Core).



Flytter koden fra AgentAssignment1 over til dette projekt, men placere koden i de relevante mapper. Sikre mig at koden bygger og virker, før end jeg begynder transformationen til brug af BindableBase.

(Dette kan jeg desværre endnu ikke i VS 2022: Installerer Prism Template Pack som en Extension til VS. Genstarter Visual Studio for at få installationen gennemført.)

Ændre MainWindowViewModel til at arve fra BindableBase. For at koden stadig skal kunne compilere så beholder jeg interfacet INotifyPropertyChanged indtil jeg har omkodet al brug af det:

```
public class MainWindowViewModel : BindableBase, INotifyPropertyChanged
```

(Virker ikke i VS 2022: Bruger kode snippet propp (tryk Tab 2 gange) til at tilføje skabelonen for en property, som benytter BindableBase's `SetProperty` funktion).

Laver en ny implementering af `CurrentAgent`. Sletter den gamle udgave af CurrentAgent, og tester at den nye virker.

Gentager denne fremgangsmåde for alle properties som benytter notify. Og kan så slette implementering-en af INotifyPropertyChanged, da den nu ikke længere benyttes.

Lab 2

Den store udfordring i denne opgave er anvendelsen af DelegateCommands. Når man bruger de angivne kode snippets (se slides), så er det let at lave strukturen for en property som returnerer en kommando. Men i selve kommandhandleren (execute) skal der bruges en anden teknik end ved implementeringen i code-behind filen, da man i en viewmodel ikke har direkte adgang til controllerne via variable.

F.eks. ved implementeringen af PreviusCommandExecute har vi brug for at ændre Listbox'ens SelectedIndex. Da vi ikke direkte kan nå ListBox kontrollen, så laves der en property CurrentIndex, som ListBox kontrollen i XAML kan binde til dens SelectedIndex:

```
SelectedIndex="{Binding Path=CurrentIndex}"
```

Jeg bruger snippet cmdfull, da kommandoen skal kun skal enables når CurrentIndex er større end 0.
Hver gang værdien for CurrentIndex ændres skal det checkes om PreviusCommand skal være enablet, hvilket angives med `.ObservesProperty(() => CurrentIndex)`

Den endelige implementering:

```
        private DelegateCommand? _previusCommand;
        public DelegateCommand PreviusCommand =>
            _previusCommand ?? (_previusCommand = new DelegateCommand(ExecutePreviusCommand,
CanExecutePreviusCommand)
            .ObservesProperty(() => CurrentIndex));

        void ExecutePreviusCommand()
        {
            if (CurrentIndex > 0)
                --CurrentIndex;
        }

        bool CanExecutePreviusCommand()
        {
            if (CurrentIndex > 0)
                return true;
            else
                return false;
        }
```

Og binder til den i XAML:

```
    <MenuItem Header="_Previous" Command="{Binding PreviusCommand}"/>
    <Button Height="auto"
                    ToolTip="Move to previous agent"
                    Content="&lt;"
                    Command="{Binding Path=PreviusCommand}"
                    />
```

I MainViewViewModel-konstruktoren kunne man evt. vælge kun at indsætter data, hvis der bygges til DEBUG mode ved brug af kompilerdirektivet #if.
Jeg kunne også have valgt kun at indsætte data i design-mode:

```
  public MainWindowViewModel()
  {
    if ((bool)(DesignerProperties.IsInDesignModeProperty.GetMetadata(
                typeof(DependencyObject)).DefaultValue))
    {
       // In Design mode
       Agents.Add(new Agent("001", "Nina", "Assassination", "UpperVolta"));
       Agents.Add(new Agent("007", "James Bond", "Martinis", "North Korea"));
    }
  }
```

Lab 3

Fjerner koden hvor jeg sætter DataContext i xaml og indsætter i stedet koden:

```
        xmlns:vm="clr-namespace:AgentAssignment2.ViewModels"
        mc:Ignorable="d"
        prism:ViewModelLocator.AutoWireViewModel="True"
```

```
Title="Lab 3" Height="450" Width="800">
```

Bemærk at denne kode kræver Nuget package Prism.WPF.

Lab 4

Indsætter et DockPanel unde om mit Grid, og tilføjer så menuen med som dockes til Top:

```
<DockPanel>
  <Menu DockPanel.Dock="Top">
      <MenuItem Header="_File" >
          <MenuItem Header="E_xit" Command="{Binding CloseAppCommand}"/>
      </MenuItem>
```

Under menuen indsættes Toolbar:

```
<ToolBar DockPanel.Dock="Top" Height="auto"
         FontSize="20" FontWeight="Bold"
         Background="AliceBlue">

    <Button Height="auto"
            ToolTip="Move to previous agent"
            Content="&lt;"
            Command="{Binding PreviusCommand}"
            />
```

File – Exit kan implementeres i ViewModel med:

```
Application.Current.MainWindow.Close();
```

Og så tilføjer jeg "?" på nogle referencetype, for at sige til kompileren, at det er ok, at de kan have værdien null. Dette går jeg for ikke at få nogle warnings når jeg bygger programmet.

Lab 5

Indsætter en statusbar som i koden indsættes under toolbaren, men som på brugergrænsefladen dockes til bunden af vinduet.

```
    </ToolBar>
    <StatusBar DockPanel.Dock="Bottom" VerticalAlignment="Bottom"
Background="AliceBlue">
    <StatusBarItem Content="Agents Count:" />
    <StatusBarItem Name="sbiAgentsCount" Width="40"
                   Content="{Binding Path=Agents.Count}" />
```

Som ur bruges en DispatcherTimer som initialiseres i viewModelens konstruktør. Denne timer opdaterer en hjælpeklasse Clock, som eksponerer Date og Time som properties vinduet kan binde til. Her er det vigtigt, at de giver eventen PropertyChanged – til det lader jer Clock nedarve fra BindableBase.

```
public class Clock : BindableBase
    {

        string? date;
        public string? Date
        {
            get { return date; }
            private set { SetProperty(ref date, value); }
        }
```