# Interop Application Review

## Lab Overview

In this lab session, you will explore an application that has a WPF front-end written in C# with its logic developed in F#. The application plays a more complex version of Monopoly than you observed earlier in this course, and allows the user to play a number of moves on one at a time, or in a burst fashion. It also tracks the number of times that the player has landed on any given position.

## Prerequisites

* Visual Studio 2015

## Time Estimate

* 30 minutes

## Table of Contents

[Interop Application Review 1](#_Toc434860015)

[Lab Overview 1](#_Toc434860016)

[Prerequisites 1](#_Toc434860017)

[Time Estimate 1](#_Toc434860018)

[Table of Contents 1](#_Toc434860019)

[Exercise 1: Overall Solution architecture 2](#_Toc434860020)

[Exercise 2: Exploring the F# code 2](#_Toc434860021)

[Exercise 3: Exploring interation points 2](#_Toc434860022)

## Exercise 1: Overall Solution architecture

In this exercise, we will familiarize ourselves with the solution and responsibilities of each project.

1. Open the solution. You will see two projects, one C# and one F#.
   1. The C# project contains the UI for the application – a single WPF page with an associated ViewModel. We use the Fody library in order to automatically implement INotifyPropertyChanged etc.
   2. The F# project contains the domain and business logic to simulate the Monopoly game.
2. Set the C# project as the Startup project and run the application.
3. You can play a “step-by-step” or opt to “auto play”, which will roll the dice a number of times automatically.
4. Set the F# as the Startup project and run the application.
5. You will see that it
6. We will be concentrating on the F# project, as well as touch points between the two projects.

## Exercise 2: Exploring the F# code

We’ll now review the F# project and understand how code and data has been organised.

1. Type.fs contains our data domain to build the Monopoly board.
2. Data.fs contains the actual board and card decks layout.
3. Controller.fs contains both a number of Record types which are used for events which expose what is happening in the game. It also contains the Controller class, which contains the entry point to the F# logic layer. Notice how we use List.scan to avoid the use of recursion, and to manage the state across turns, within the playGame function.
4. Test.fsx allows you to try out the F# in a script file. You can also see in this script file how F# has excellent support for working with events through the Event module.

Note that Controller is a proper .NET class, not a module. This is because .NET-compatible events are not supported in F# Modules.

## Exercise 3: Exploring interation points

We’ll now review how the C# and F# projects interoperate with one another, calling F# from C# across threads, whilst sharing data in classes, ViewModels and even Views.

1. Within Controller.fs, observe the OnMoved event – this is marked with the [<CLIEvent>] attribute so that it shows as a regular .NET event to C# and VB .NET. This event is fired every time the player moves to a new position on the board.
2. Find the two references to OnMoved within the C# project (you will need not be able to use Find All References). Both are within the two ViewModel Commands, AutoPlayCommand and StepByStepCommand
3. Notice how in the AutoPlayCommand we handle the event by looking up the appropriate ViewModel BoardPosition and Incrementing() the count of number of times it has been landed on.
4. The Execute method of the AutoPlayCommand simply sets up this event handler, and then on a background thread starts playing the game for 50,000 moves.
5. The StepByStepCommand works slightly differently – it records all the movements, and then gets an Enumerator from which to step forward through the history of moves on demand.
6. Notice that we perform a crude version of pattern matching in the Execute method in order to determine whether the movement event is a “Landed On” or a “Moved To” event.
7. On the BoardPosition ViewModel DTO, we expose the F#-created Position type directly to both Commands and potentially to the XAML view.