

SACHA BARBER

F# in Finance, Final Exam Project

Contents

1.	The General Idea	3
2.	The Overall Structure	4
3.	What Does The Demo App Do?	5
3.1	The Algorithm	5
3.2	The Gene Sequence	6
3.3	The Initial Population	7
3.4	The Population Count	7
3.5	How Is The Population Scored	7
3.6	How Is A New Population Created	7
4.	Deep Dive Into Demo App	8
4.1	BioCSharp.Interfaces	8
4.2	BioCSharp	8
4.3	SachaBarberFSharpGeneticAlgorithm	9
4.3.1	XAML Type Provider	9
4.3.2	Initialisation	10
4.3.3	The App Startup	10
4.3.4	ViewModels General Approach	10
4.3.5	MainWindowViewModel	10
4.3.6	Charting	11
5.	Conclusion	11

1. The General Idea

For my final piece of course work for the F# In Finance course, I decided to try and include a broad cross section of the techniques included in the course. To this end I have tried to include the following areas:

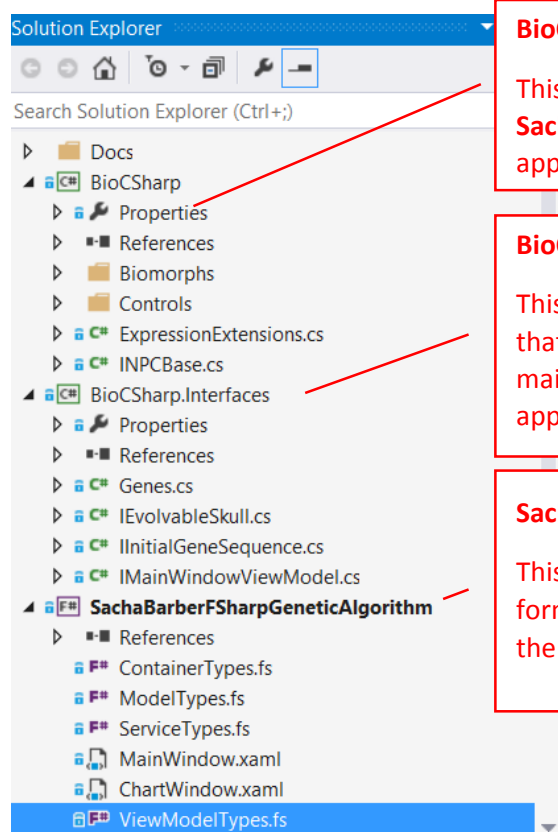
- Deedle usage
- CSVTypeProvider usage
- Other type provider usage
- F# charting
- Functional F#
- OOP F#
- Interop with C#

I think I have done fairly ok in my usage of these things, and I have also included a few other things for fun such as:

- Reactive Extensions (RX)
- F# Async workflow
- Inversion Of Control (IOC) container
- Log4net
- WPF app

2. The Overall Structure

When you open the Visual Studio solution (VS2013) you should see something like this:



The screenshot shows the Visual Studio Solution Explorer for a project named 'SachaBarberFSharpGeneticAlgorithm'. The project is a solution containing three main components: 'BioCSharp', 'BioCSharp.Interfaces', and 'SachaBarberFSharpGeneticAlgorithm'. 'BioCSharp' is a C# DLL project with subfolders for Properties, References, Biomorphs, Controls, ExpressionExtensions.cs, and INPCBase.cs. 'BioCSharp.Interfaces' is a C# DLL project with subfolders for Properties, References, Genes.cs, IEvolvableSkull.cs, IInitialGeneSequence.cs, and IMainWindowViewModel.cs. 'SachaBarberFSharpGeneticAlgorithm' is an F# application project with subfolders for References, ContainerTypes.fs, ModelTypes.fs, ServiceTypes.fs, MainWindow.xaml, ChartWindow.xaml, and ViewModelTypes.fs. Red arrows point from the project names in the Solution Explorer to the corresponding text boxes on the right.

BioCSharp

This is a C# DLL that is used from the main F# **SachaBarberFSharpGeneticAlgorithm** application

BioCSharp.Interfaces

This is C# DLL that provides common interfaces that are used by both the **BioCSharp** DLL and the main F# **SachaBarberFSharpGeneticAlgorithm** application

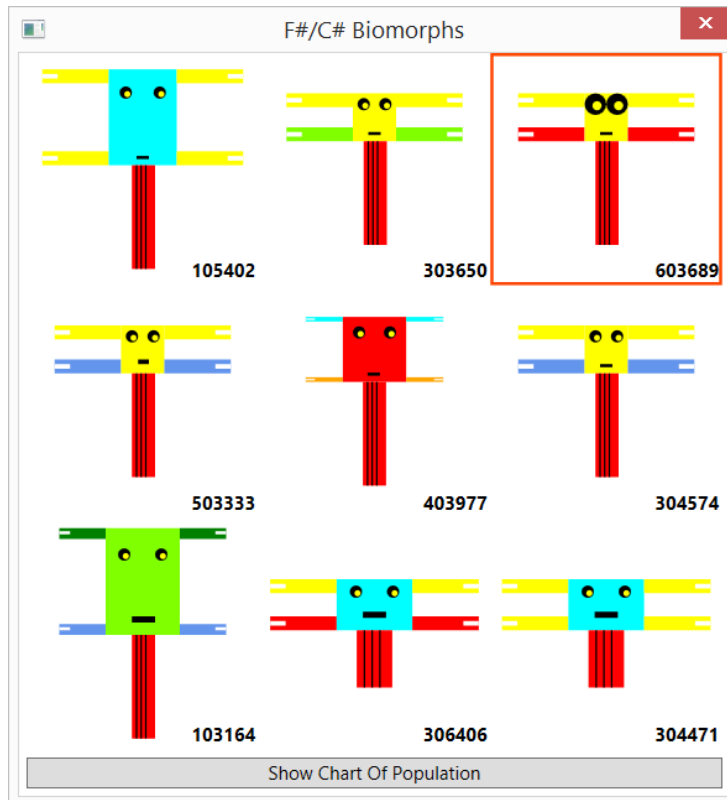
SachaBarberFSharpGeneticAlgorithm

This is main F# application. It is this one that forms the main basis for my submission. This is the one that has the main window in it.

3. What Does The Demo App Do?

The demo app I have decided to create is a Genetic Algorithm that will try and create an elite population of cartoon'ish skulls.

The running app will look something like this:



Where it can be seen that we have a population of cartoon'ish skulls each with different bones/head/eyes

3.1 The Algorithm

The basic operation of the Microbial GA training is as follows:

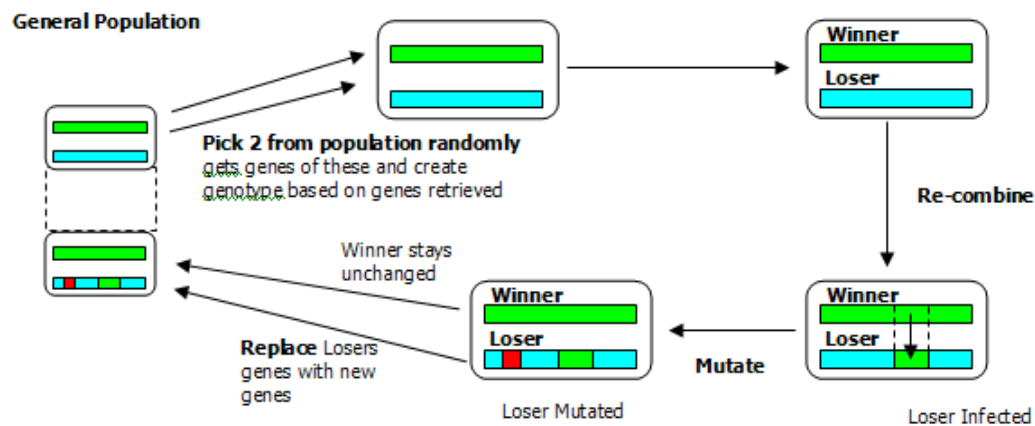
- Pick two genotypes at random
- Compare Scores (Fitness) to come up with a Winner and Loser
- Go along genotype, at each locus (Point)

That is:

- With some probability (randomness), copy from Winner to Loser (overwrite)
- With some probability (randomness), mutate that locus of the Loser

So only the Loser gets changed, which gives a version of Elitism for free, this ensures that the best in the breed remains in the population.

That's it. That is the complete algorithm. This diagram may help to further illustrate things:



But there are some essential issues to be aware of, when playing with GAs:

1. The genotype will be different for a different problem domain
2. The Fitness function will be different for a different problem domain

These two items must be developed again, whenever a new problem is specified.

For example, if we wanted to find a person's favourite pizza toppings, the genotype and fitness would be different from that which is used for this article's problem domain.

These two essential elements of a GA (for this article's problem domain) are specified below.

3.2 The Gene Sequence

There are a total of 9 genes describing the genotype of the organism, which are held in a static class in the BioCSharp.Interfaces.Genes class.

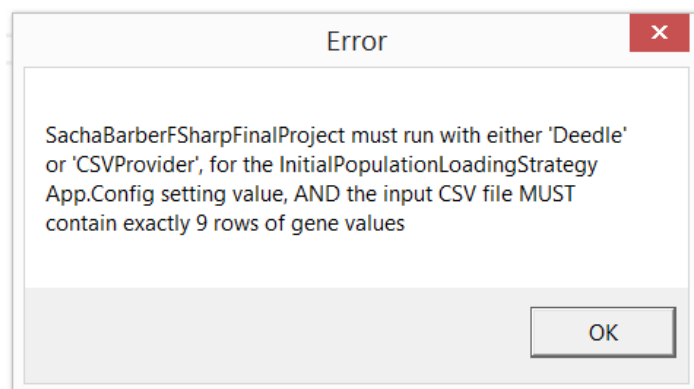
```
public static class Genes
{
    // the individual genes are at fixed array positions. So simply declare this nice
    //description names. To hold the array positions, so the array positions may be
    //referenced to later as GenotypeWidths[LEFT_EYE_GENE] etc etc
    public static int FACE_GENE = 0;
    public static int LEFT_EYE_GENE = 1;
    public static int RIGHT_EYE_GENE = 2;
    public static int NOSE_GENE = 3;
    public static int LEFTBONE_TOP_GENE = 4;
    public static int RIGHTBONE_TOP_GENE = 5;
    public static int TEETH_GENE = 6;
    public static int LEFTBONE_BOTTOM_GENE = 7;
    public static int RIGHTBONE_BOTTOM_GENE = 8;
}
```

3.3 The Initial Population

The initial population is meant to come from a CSV file called "InitialPopulation.csv" within the main **SachaBarberFSharpGeneticAlgorithm** F# project. Depending on the App.Config "InitialPopulationLoadingStrategy" value, this CSV file will be read with Deedle or the CSVTypeProvider for F#

3.4 The Population Count

The initial population is expected to be EXACTLY 9, so if the user tries to amend the count of the "InitialPopulation.csv" file within the main **SachaBarberFSharpGeneticAlgorithm** F# project, which should result in an error being shown something like this:



3.5 How Is The Population Scored

Each organism of the population is scored, such that an overall dominant one may be found. So how does the scoring work? Well for this simple GA, we treat each gene roughly as some rectangle type shape that has a width/height/colour. So we simply add up all the overall areas of the gene sequences for the organism, and add on any bonuses it may hold.

Bonuses are available for:

- Preferred eye colour
- Preferred gene colour

3.6 How Is A New Population Created

There is a cross over point picked, and up to that point some of the genes will come from the parent, and after that point will come from the other parent. Then some random genes are mutated (have new values assigned). This is how a new population is created in this demo app

4. Deep Dive Into Demo App

This section will talk about the individual projects/items within the projects in a bit more detail, though for the best understanding you will still need to read the code

4.1 BioCSharp.Interfaces

This project is a C# Dll, that is used by both the **BioCSharp C#** project and the main **SachaBarberFSharpGeneticAlgorithm F#** project.

The actual interfaces/classes this project contains are listed below:

File name	Description
Genes.cs	Static class holding the int values representing the gene numbers
IEvolvableSkull.cs	Defines interface to dictate what an organism in the population can do
IInitialGeneSequence.cs	Defines interface to dictate what initial gene sequence will look like
IMainWindowViewModel.cs	Defines interface that will be implemented by MainWindowViewModel

4.2 BioCSharp

This project is a C# Dll, that is used by the main **SachaBarberFSharpGeneticAlgorithm F#** project. This project provides a couple of thing, that are used via interop between F#/C#

The actual interfaces/classes this project contains are listed below:

File name	Description
Biomorphs/EvolvableSkullComparer.cs	A simple IComparer implementation for EvolvableSkullViewModel types
Biomorphs/EvolvableSkullViewModel.cs	A view model to represent one organism of the population
Biomorphs/RandomInitialGeneSequence.cs	Random population initialiser
Controls/BiomorphItem.cs	A control to represent the drawing of the EvolvableSkullViewModel item(s)
Controls/BiomorphItemsControl.cs	A specialized XAML ItemsControl
ExpressionExtensions.cs	Set of useful expression tree extension methods
INPCBase.cs	Useful base class for XAML view models

4.3 SachaBarberFSharpGeneticAlgorithm

This is the main F#

The actual interfaces/classes this project contains are listed below:

File name	Description
ContainerTypes.fs	F# IOC Container, written by Phil Trelford
ModelTypes.fs	F# record type representing population item
ServiceTypes.fs	F# classes representing view model services
MainWindow.xaml	MainWindow which is loaded by F# XAML type provider. This shows the population at any given moment in time
ChartWindow.xaml	ChartWindow, this is used to show a F# chart of the active population
ViewModelTypes.fs	Viewmodel types, and ICommand implementation
Initialisation	This contains the code to load the initial population either from reading CSV file in from Deedle or CSVTypeProvider
App.xaml	Specified the main window to use for the F# application
App.fs	The main entry point, which will read the App.Config and load the population using the chosen loader
App.config	Contains config for Log4net and also on the currently chosen population initialiser
InitialPopulation.csv	The CSV file to read the initial population from

4.3.1 XAML Type Provider

I came across a way to get better support for XAML by way of this NuGet package :

<https://www.nuget.org/packages/FSharpX.TypeProviders.Xaml/>

Which offers better F# XAML support. You can read more about this here:

<https://sergeytihon.wordpress.com/2013/04/27/wpfx-mvvm-with-xaml-type-provider/>

4.3.2 Initialisation

See Initialisation.fs

The App.Config contains a value “InitialPopulationLoadingStrategy” which can either be used to read in the CSV file values into a Sequence of Person records using Deedle or using a CSVTypeProvider

4.3.3 The App Startup

See App.fs

Several things happen at startup.

1. Log4net is configured for logging for the entire app
2. The App.Config file is read to see how the CSV file should be read to initialise the population
3. When the population is read in, it is published out using an application service that makes use of RX
4. The app is then run, which in turn creates a new MainWindowViewModel from the IOC container used. The MainWindowViewModel subscribes to the application service via RX, and then loads the initial population from that published data

4.3.4 ViewModels General Approach

See ViewModelTypes.fs

For the MainWindowViewModel, I have developed an attached DependencyProperty which is set in the XAML of the Window, which expected to be set to the Type of the ViewModel required. When set the ViewModel type is located in the F# IOC container by Phil Trelford, and used as the DataContext for the Window.

4.3.5 MainWindowViewModel

See ViewModelTypes.fs

The MainWindowViewModel is where a lot of the action occurs. This ViewModel carries out the following tasks

- Listens for the initial population from the RX application service
- Implements the IMainWindowViewModel, which means it sorts the hosted population
- Creates a new population from dominant

- Runs a RX timer, which in turn runs a F# async workflow to score/sort the population
- Exposes an ICommand to launch a F# charting window to show chart of the current population

4.3.6 Charting

See ChartWindow.xaml / ViewModelTypes.fs

The ChartWindow hosts a WindowsFormHost to allow F#/WPF and F# Charting to play nicely together. Which you can read more about at this post:

<http://fsharp.github.io/FSharp.Charting/UsingChartsInWPF.html>

5. Conclusion

I have had quite a bit of fun working on this final piece of course work. I think I have done reasonable job of trying to showcase some of the stuff we learnt on the course in this demo app. One area where I do think F# is severely crippled compared to other .NET languages is in the tooling. Even forgetting how much of a pain it is to try and create a standard WPF Window, just consider something basic like try to re-order a file in the Visual Studio IDE. You have to locate the file and move it up/down one at a time.



I understand why we have this, it is to make use think about our types a bit more clearly before we start. This mechanism though, really? This takes ages and I think it is absolutely retarded personally.

That said I have really enjoyed working with F#, and creating what I would consider to be a proper application, even if it is a small one. I just wish the tooling was a lot better.