Flight Simulator

- In this project, we built an application that allows flight data to be displayed and analyzed on a dedicated simulator
- Throughout the project, the design template that was used is: MVVM (details below).
- Alongside the application, algorithms have been developed to detect anomalies in a way that can be used in conjunction with the application as a plugin.

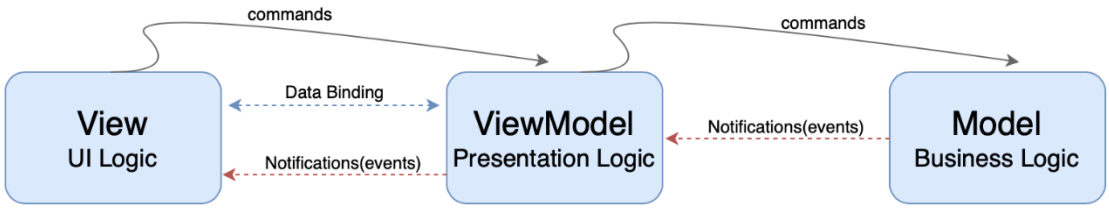
The main classes in the project:

- *FlightDetectorModel*
- *View's Model*:
  - *DataInfoVM*
  - *GraphVM*
  - *NavigatorStateVM*
  - *ProgressBarVM*
- *View's*:
  - *DataInfoView*
  - *GraphView*
  - *NavigatorStateView*
  - *ProgressBarView*
- *AnomalyDetector*
- *Tools Classes*:
  - *MostCorrelativeFinder*
  - *Transmitter*
- *Windows*:
  - *MainWindow*
  - *DashboardWindow*
  - *AnomalyDetectorLoaderWindow*

Using the MVVM design template:

The MVVM template is an architectural template in software engineering and aims to create order in UI applications and maintain programming principles.

Throughout the project we have used this template according to the following architecture:



Model

The project has one main model: $FlightDetectorModel$, which holds information data in which the main boot for the application takes the place:

- Opening and closing of the broadcast in front of the dedicated simulator
- Receiving data
- Updating the various View Model's while receiving new data. (Done by the $NotifyPropertyChanged$ function

ViewModel & View

For each significant user story we implemented ViewModel & View:

- Each ViewModel maintains an instance of the central model in order to receive updates on data changes.
- Each View holds an instance of the appropriate ViewModel to perform the data presentation.
- The data is transferred for the purpose of presenting them by Data Binding.

The idea is a complete separation between the View and the Model. The communication between them is through Commands mechanisms for transmitting commands and messages.

user story's

There are 4 main User Stories in our project:

DataInfo

- *DataInfoVM* :
  A communication component between FlightDetectorModel and DataInfoView

- *DataInfoView*:
  The part responsible for graphics display of the data, such as:
  1. Flight attitude
  2. Flight speed
  3. Flight direction
  4. Yaw, Roll, Pitch indices

Graph

- *GraphVM* :
  A communication component between FlightDetectorModel and DataInfoView

- *GraphView*:
  The part responsible for graphics display of the data and the anomalies in the graph.
  Contains 3 main graphs - a graph for a selected property, a graph for the most correlative property of a selected property, a graph for investigating the common data of the properties that includes the linear regression line between them.

NavigatorState

- *NavigatorStateVM* :
  A communication component between the FlightDetectorModel
  To NavigatorStateView.

- *NavigatorStateView*:
  The part responsible for graphics display of the aircraft's main rudders in a joystick-like display.
- Contains a joystick-like steering wheel, which changes the movement of the aircraft according to the data entered from the file.

ProgressBar

- *ProgressBarVM*:
  A communication component between the FlightDetectorModel
  To ProgressBarView.

- *ProgressBarView*:

  The part responsible for the scroll bar graphics, used to schedule the simulator display and other functionality such as: stop, increase / decrease the playback speed, flight movie time.

Helping Classes

- *MostCorrelativeFinder* – Responsible for finding the most correlative pairs according to the original pearson value and regression lines.
- *Transmitter* - Responsible for opening the broadcast with the flight simulator and sending the data.

AnomalyDetector + dll

In user story 9 we were asked to give the user the option to load algorithms to detect anomalies by using dll at runtime.

Our implementation:
The user selects the dll file from an external folder which isn't exposed to the project , by this implementation the application gets a path to the dll file and the application uses a file called kernel32.dll which is in all windows.

According to the requirement of the project, we offer support any dll file that reveals the detect function.

In in particular dll for detecting anomalies by linear regression, minimum circuit (as we implemented in c ++ in advanced programming1).

The requirement from the uploaded file is that it will reveal a function called detect that returns an array of ints.

when at first place of the array is its length, the rest of the data about the anomalies is inside.

To differentiate between the anomalies between the columns itself the algorithm will put a flag in the array (-1).

When another anomaly detection algorithm is implemented it will reveal the detection method which will get a path to the anomaly file and the normal file, and return the anomalies in an int array- when its length is at the top.

So for any algorithm no matter how it is implemented we can dynamically load it and extract the data by the detect method.

Also, we added relevant controls which allow the user to select a dll file to base the analysis.

In addition, we added an option where the user can set the correlation threshold (between 0-1) for maximum flexibility, and full utilization of the algorithm capabilities.

* It is important to note that one of the threshold conditions for running dll to detect anomalies is uploading 2 csv file - one is correct and the other we want to explore accordingly we have restricted the user screen from being allowed to upload a file unless uploading the relevant files.

The class anomaly detector is extract the data from the dll , and send data to the model.

Windows

There are 3 windows:

***MainWindow* – Main window**

Options:

1. Upload a csv file for flight screening
2. Upload a normal csv file for learning and an anomaly csv file for research
3. Load an anomaly detection algorithm using a dynamically loaded dll file at runtime.

***DashboardWindow* - Application window**

Options:

1. Control of the flight scroll bar.
2. Flight analysis using the data presented in graphs and controls.
3. Load another anomaly detection algorithm - using a dynamically loaded dll file while running.
4. Back to main menu

- *AnomalyDetectorLoaderWindow*

  Window for loading an anomaly detection algorithm.