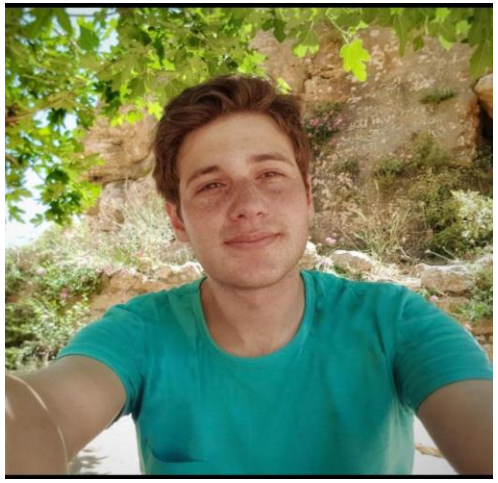
Flight Control Android Application

SUBMITTED BY:

YANA SIDANYCH

ITAY YAAKOV



About The Project

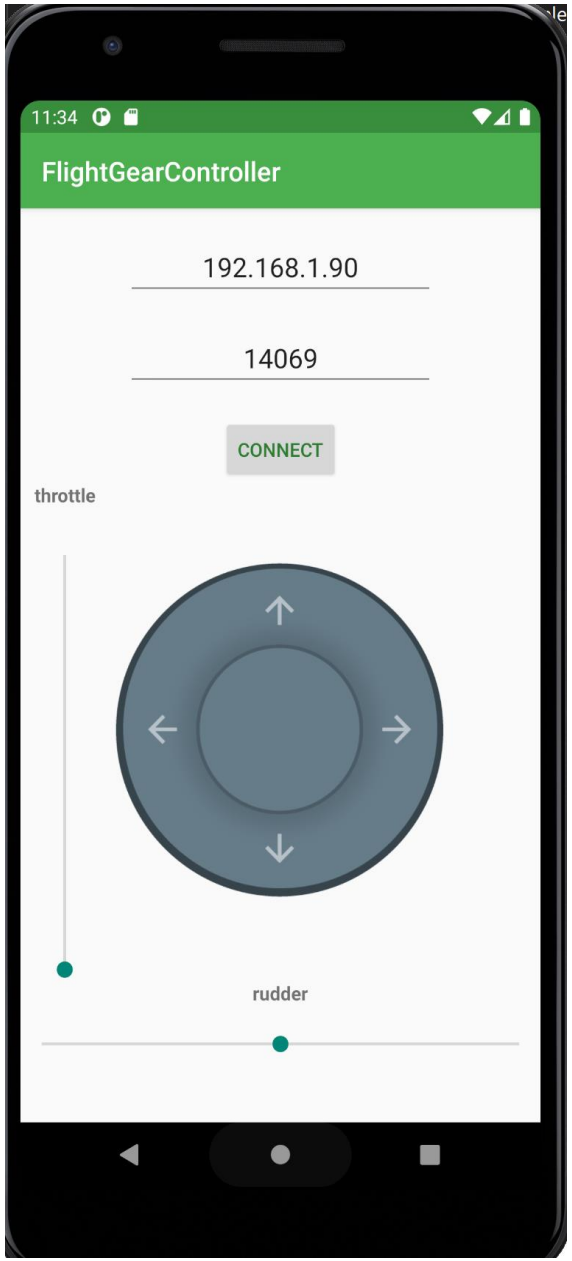
- We created a joystick app that allows the use to control a plane inside the flight gear simulator running on a different machine.
- The app was written for android devices using Kotlin.
- The app was created as a part of our Advanced Programming course:

  Course number:89211

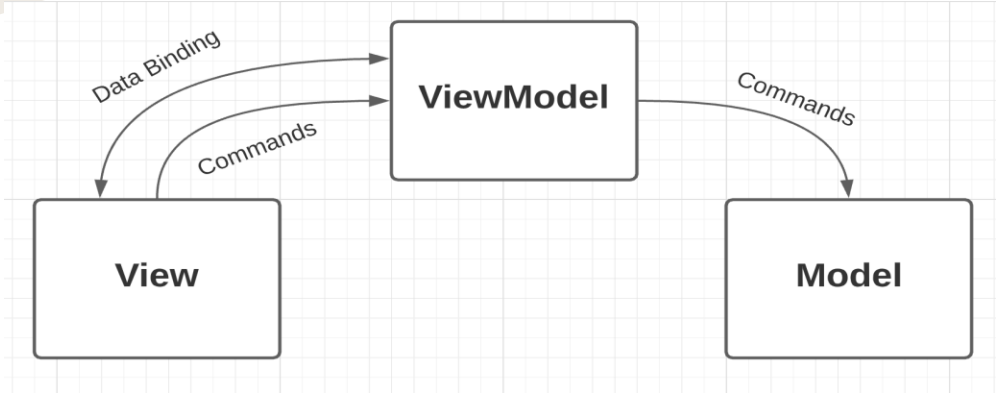
  Lecturer: Eliyahu Khalastchi

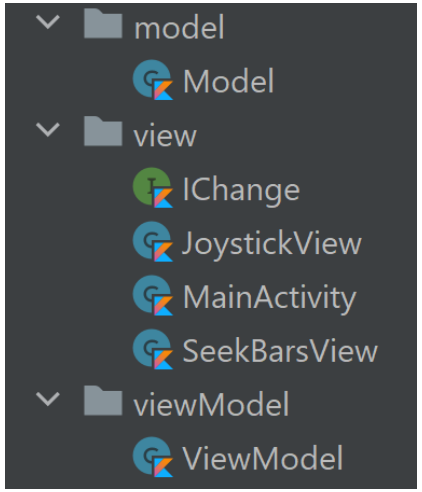
Our User Interface

- The app has one screen, which contains the two main parts of the app.
- First, we have the connection section, that allows the user to connect to the flight gear simulator.
- Second, we have the joystick itself that allows the user to control the plane, from takeoff to the landing.



The MVVM Architecture





- As the title says, in our app we used the MVVM architecture.
- The code was organized in three packages: model, view and viewModel.
- The view package contains the three main views, the joystick, the throttle and rudder bars and the mainActivity. They are responsible for showing and managing all the GUI components. The of the views uses data binding to update the viewModel when a variable is changed.
- The model is responsible for connecting to the simulator and sending the flight commands
- When a bound variable is notified as changed by the view, the bound variable in the viewModel also changes and the viewModel is responsible of sending the corresponding command to the model.

The MVVM Architecture – Code Examples

- Lets demonstrate the data binding between the ip and port GUI EditText elements in the view, to the ip and port variables in the viewModel:
- `android:text="@={viewModel.ip}"` `android:text="@={viewModel.port}"` From: activity_main.xml
- In these two lines of code you can see how the text atrribute of the GUI EditText elements is bound to the ip and port variables that exists in the viewModel.kt file.
- You can see that the viewModel variable is defiend in these code lines from activity_main.xml:

```
<variable
    name="model"
    type="com.example.flightgearcontroller.model.Model" />

<variable
    name="viewModel"
    type="com.example.flightgearcontroller.viewModel.ViewModel" />
```

- The model variable is also defined here and in a similar way and the isConnected variable from the model.kt file is bound to the connect button text and color attributes:

```
android:text="@{model.isConnected() ? @string/button_text_off : @string/button_text_on}"
android:textColor="@{model.isConnected() ? @color/md_red_800 : @color/md_green_800}"
```

The MVVM Architecture – Code Examples

- For the joystick and seek bars GUI things work a little bit different.
- The main activity holds both of this views as global variables. Each of this views contains an object from type IChange interface. In this interface there is only one function, onChange, receiving two variables, name of the value changed, and the value it changed to.
- When this two GUI components are created inside the mainActivity.kt file, the onChange function is implemented and is set to trigger the viewModel sendCommands function. The mainActivity view holds the viewModel as a global variable.
- Now the mainActivity view can respond to changes in the joystick and seek bars value, when the user is touching them.

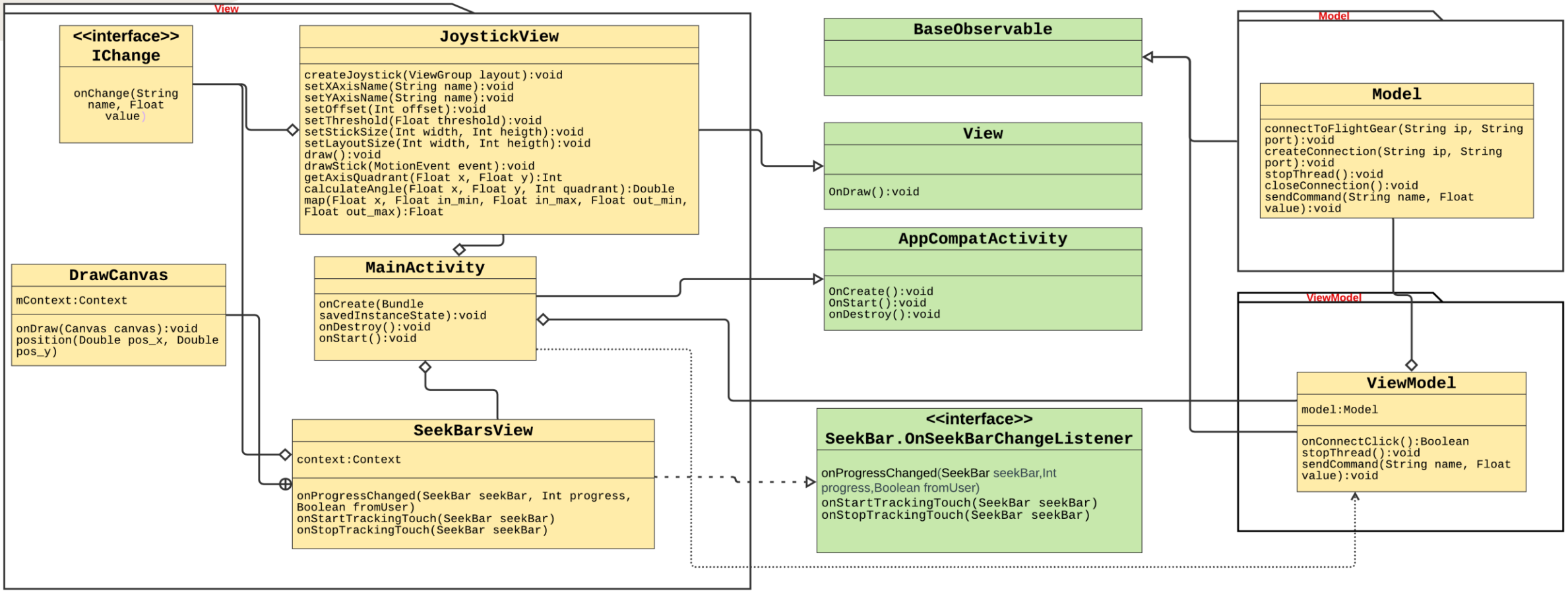
```
public var sendUpdateEvent: IChange? = null
```

```
// Interface for strategy design pattern
fun interface IChange {
    fun onChange(name: String, value: Float)
}
```

```
// create sliders object
seekBars = SeekBarsView(this)
// listen to values change of sliders
seekBars!!.sendUpdateEvent = IChange { name, value ->
    viewModel!!.sendCommand(name, value)
}
```

```
when (seekBar) {
    throttleSeekBar -> sendUpdateEvent?.onChange("throttle", progressF)
    rudderSeekBar -> sendUpdateEvent?.onChange("rudder", progressF)
}
```

UML Class Diagram



Clarification:

Green Class/Interface - Kotlin provided class

Dotted line - data binding

Thank you for listening :)

ENJOY THE FLIGHT !