Assignment 2

Team number: 17
Team members

Name	**Student Nr.**	**Email**
Bas Dijkstra	2629858	b.f.dijkstra@student.vu.nl
Krasen Todorov	2735134	k.todorov2@student.vu.nl
Daniel Roos	2550957	d.a3.roos@student.vu.nl
Victor Retamal Guiberteau	2741820	v.retamalguiberteau@student.vu.nl

Implemented feature

ID	**Short name**	**Description**
F1	Commands	On the main screen of the software, the user will have the option to activate the following function: - Load GPX
F2	GPX file handling	The GPX file will be loaded and the data will be ready to be read by the other functions. It will generate an object with the possibility to add more information as a sport. The file extension is .gpx

Class diagram

Author(s): Bas Dijkstra

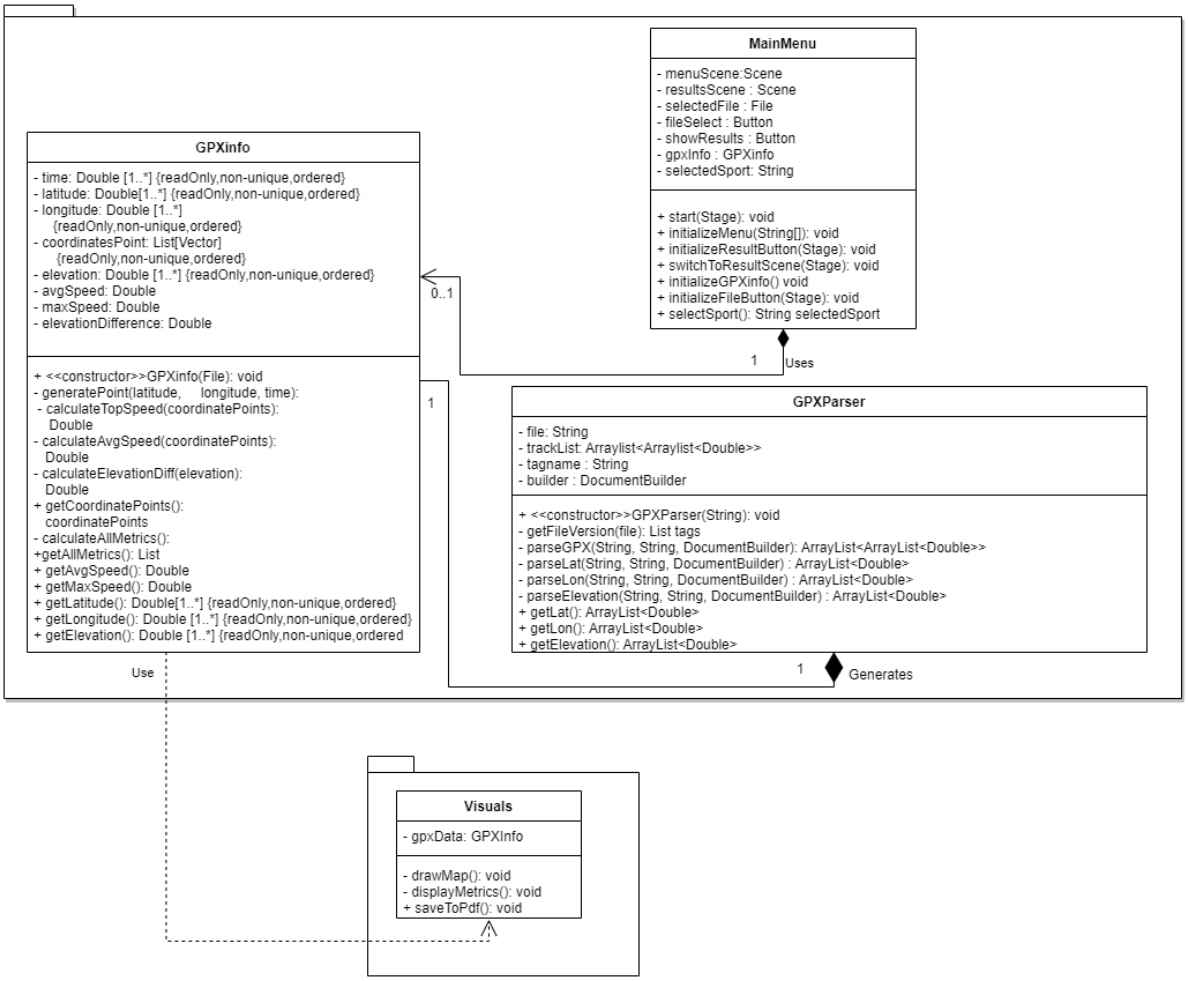

Figure 1. Class Diagram

MainMenu

The main menu is mainly used for displaying the UI. It also has functions to get the file path for the GPX file and switch between scenes.

Attributes

- menuScene: Contains information on the main menu screen
- resultsScene: Contains information on the screen to is switched to once the GPX is parsed
- gpxInfo: Contains all information on the parsed GPX data
- selectedFile: Contains the select GPX file, and it's class path
- selectedSport: Contains information on which sport the user has chosen

Operations

- initializeMenu: Function to visualise the main menu and display it
- switchToResultScene: Function to switch from the main menu to the results, which is displayed after the GPX file is parsed
- initializeGPXinfo: Used to start the GPX parser and create a GPXinfo object
- selectSport: Function to allow the user to select a sport. Will be stored in selectedSport.

Association

- GPXinfo: The main menu initialises the GPXinfo object once the file has been parsed

GPXinfo

The GPXinfo class contains all necessary information extracted from the GPX file input.

Attributes

- latitude: Information on all the latitude points in the data
- longitude: Information on all the longitude points in the data
- elevation: Information on all the elevation points in the data
- Time: information on all the time points in the data

Operations

- GPXinfo: Constructor, this calls the parser and sends it the location of the file. The parser then returns all the info of the GPX file

Association

- GPXparser: It calls the GPXparser in the constructor. The GPXparser only exists once the GPXinfo object has been created.

GPXParser

The GPX parser is used to parse the input GPX file and return a GPXinfo object.

Attributes

- file: This is where the class path of the GPX file is located
- tagName: this string is used in the parsing of the file

- builder: This DocumentBuilder is used in parsing the string, it has some useful features for parsing

Operations

- getLat: This function calls the parseLat function and then returns a list of all latitudes contained in the GPX file
- getLon: This function calls the parseLon function and then returns a list of all longitudes contained in the GPX file
- getElevation: This function calls the parseElevation function and then returns a list of all elevations contained in the GPX file

Visuals

The visuals class is used to create all visual information, graphs and the map, from the GPXinfo data.

Attributes

- gpxData: Contains all GPX info to create the visual objects

Operations

- drawMap: Uses the library **GMapsFX** functions to draw a map using the GPX data
- displayMetrics: Uses the library **Xchart** functions to draw some graphs using the GPX data
- saveToPDF: Saves a summary report as PDF data using the **Apache PDFBox** library

Object diagram

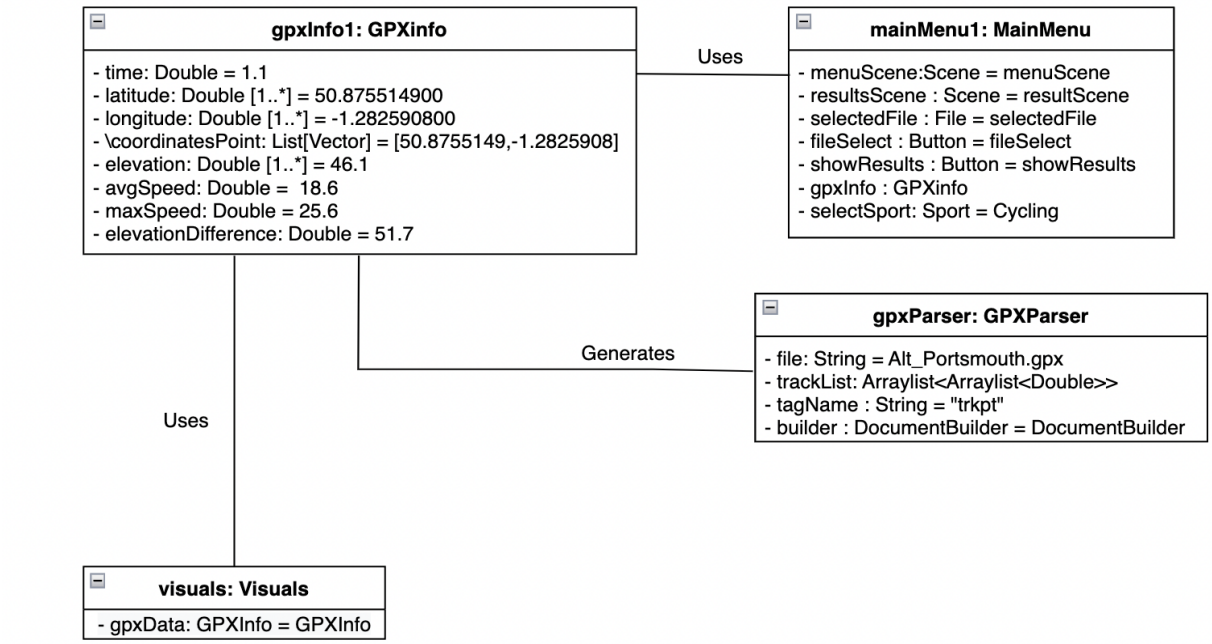

Figure 2. Object Diagram

The image above represents an Object diagram of a classical run of the system. It shows the process of how the objects transition through runtime. When executing the tool, an instance of MainMenu is being created, following the selecting of a sport (Cycling) and the location of a valid GPX file has been sent by the constructor. After that, the GPXparser is being initialised and all the gpx data that is contained within the file is being used in order to calculate the specific metrics of the chosen sport and create the visual objects.

- *mainMenu1: MainMenu* - The attribute of type **Sport**'s value changes to **Cycling** after the user has selected the sport. After the location of the GPX file has been sent by the constructor the **GPXinfo** object is being initialised

- *gpxInfo1: GPXinfo* - The attribute **time** has the new value **1.1** assigned using the data from the gpx file. The attributes: **latitude, longitude, coordinatesPoint, elevation** also have values assigned matching the info data accordingly. The rest of the attributes: **avgSpeed, maxSpeed, elevationDifference** receive new values after the calculation of the metrics is executed.

- *gpxParser: GPXparser* - "Alt_Portsmouth.gpx" value is being assigned to the String **file** after the class path of the GPX file has been located. The string "trkpt" which is used in the parsing of the file is being assigned to **tagName.**

- *visuals: Visuals* - at this stage all the GPX data that is contained in **gpxData** is used and visualised as a graph on a GPS map and made available for the user to download as PDF

State machine diagrams

Author(s): Daniel Roos

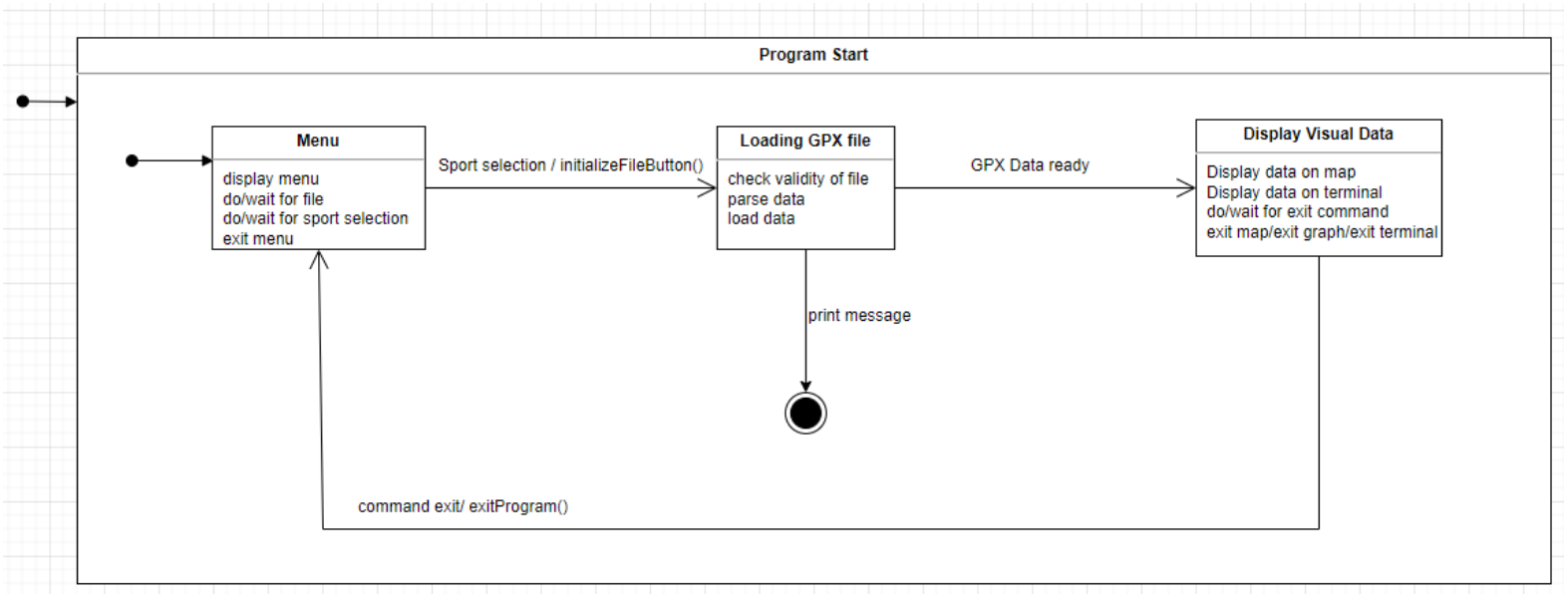

Figure 3. State Machine Diagram

The figure above demonstrates the state diagram of the program. The program references the whole program which encapsulates all classes and objects. The initialization of the program starts with a menu being displayed to the user. All values are given to the program through a GPXdata file meaning that at this state there are no values being calculated or initialised.

Firstly, after initialization the Menu will prompt users for a file which will be limited to .gpx files. The program will then use Menu to ask the user a sport selection, once all this information is given by the user the menu will close.

The selection and file will be transferred over to the next state the program reaches, which is Loading GPX file. At this state the program checks the validity of the file, if the file is valid it will parse the data and load the data ready for it to be displayed on one of the final states.

The last state is Display Visual data; this state is reached once the Loading GPX file state has the data ready. At this state the program displays the data the user has provided on a map and a terminal for any extra data. This state is purely visual and has had all calculations done by the previous state. The only option for the user is now to exit the program with a button, this will loop the program back to the Menu state initialising itself ready for a new file.

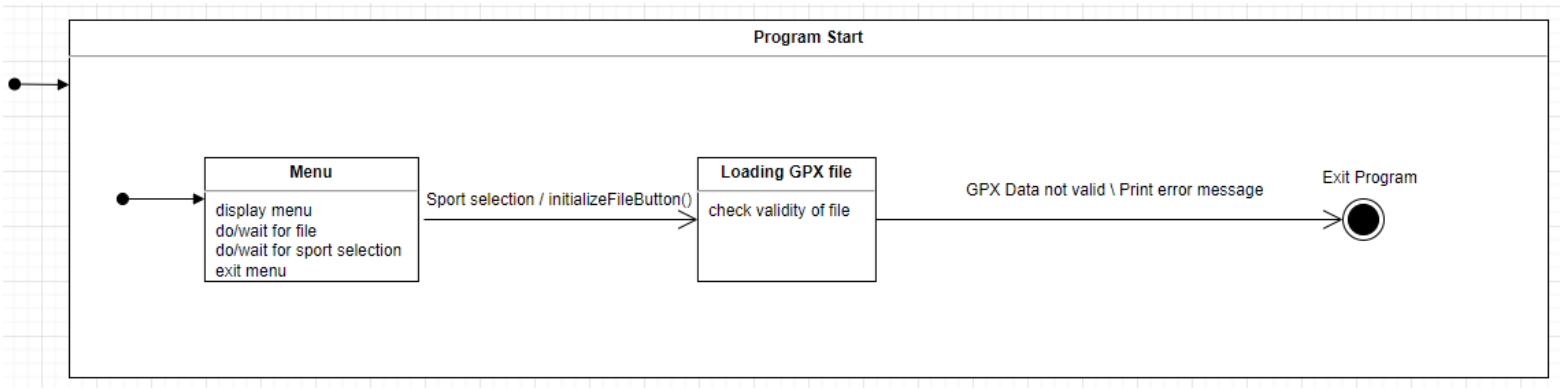

Figure 4. State Machine Diagram: Program start

In this state diagram the program is started and the program enters the Menu states and prompts users for a file and to make a selection for a sport. After this is done the program initialises and starts the load up the GPX file given by the user.

The program will now enter the Loading GPX file state but the validity of the file is checked first. At this state the program has determined that the given file is not valid, a non-GPX file or a corrupt file as an example. The program will catch the exception and print out an error message to the user. And lastly the program will leave the Loading GPX file state and close the program completely

Sequence diagrams

Author(s): Victor Retamal

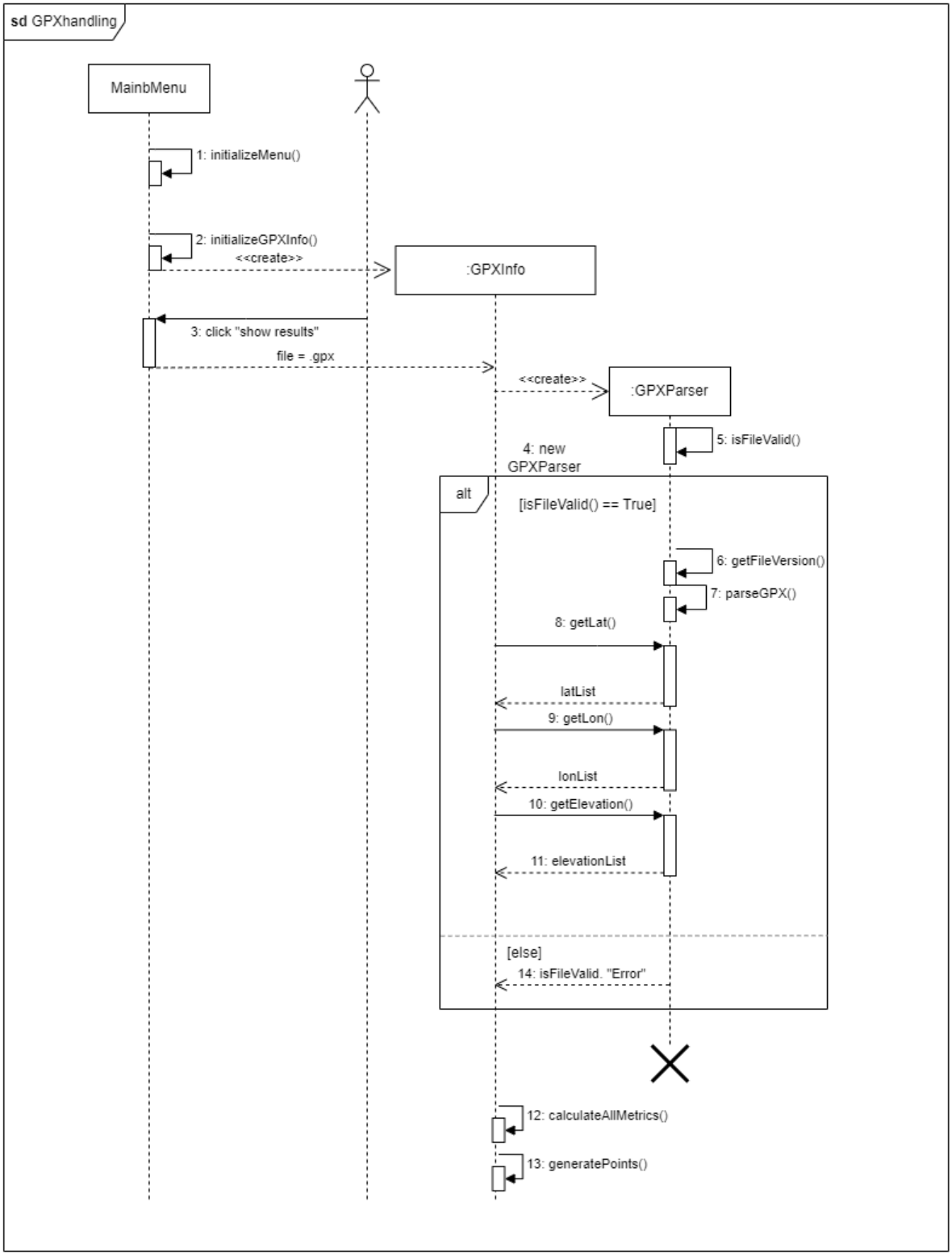

Figure 5. Sequence Diagram: GPX File Handling

Sequence Diagram 1. GPX File Handling.

The diagram above (Figure 5) shows the initialization of a gpx file when a new program starts. When the software is executed, an instance of **MainMenu** is created. In the initialization of the menu *initializeMenu()*, buttons ("select file" and "show results") and UI are created. An instance of **GPXInfo** is created with the function *initializeGPXInfo()*. The click of the button "show results" by the user triggers the creation of an instance of **GPXParser**. The first trigger action in the parser will be to filter for a valid .gpx file. If the file is not correct an error is displayed and the program is exited. If the file is a valid gpx file *isValidFile()*, the parser will check for the version of gpx to extract the tag where the information of the route is stored *getFileVersion()*. With a valid file and the tag extracted, we parse latitude, longitude, elevation and any extra metric if present such as time *parseGPX()*. When parsing is done, the instance gets destroyed. After the parsing metrics and coordinate points are calculated and stored in the **GPXInfo** instance with *calculateAllMetrics()* and *generatePoints().*

Sequence Diagram 2. GPX File Handling.

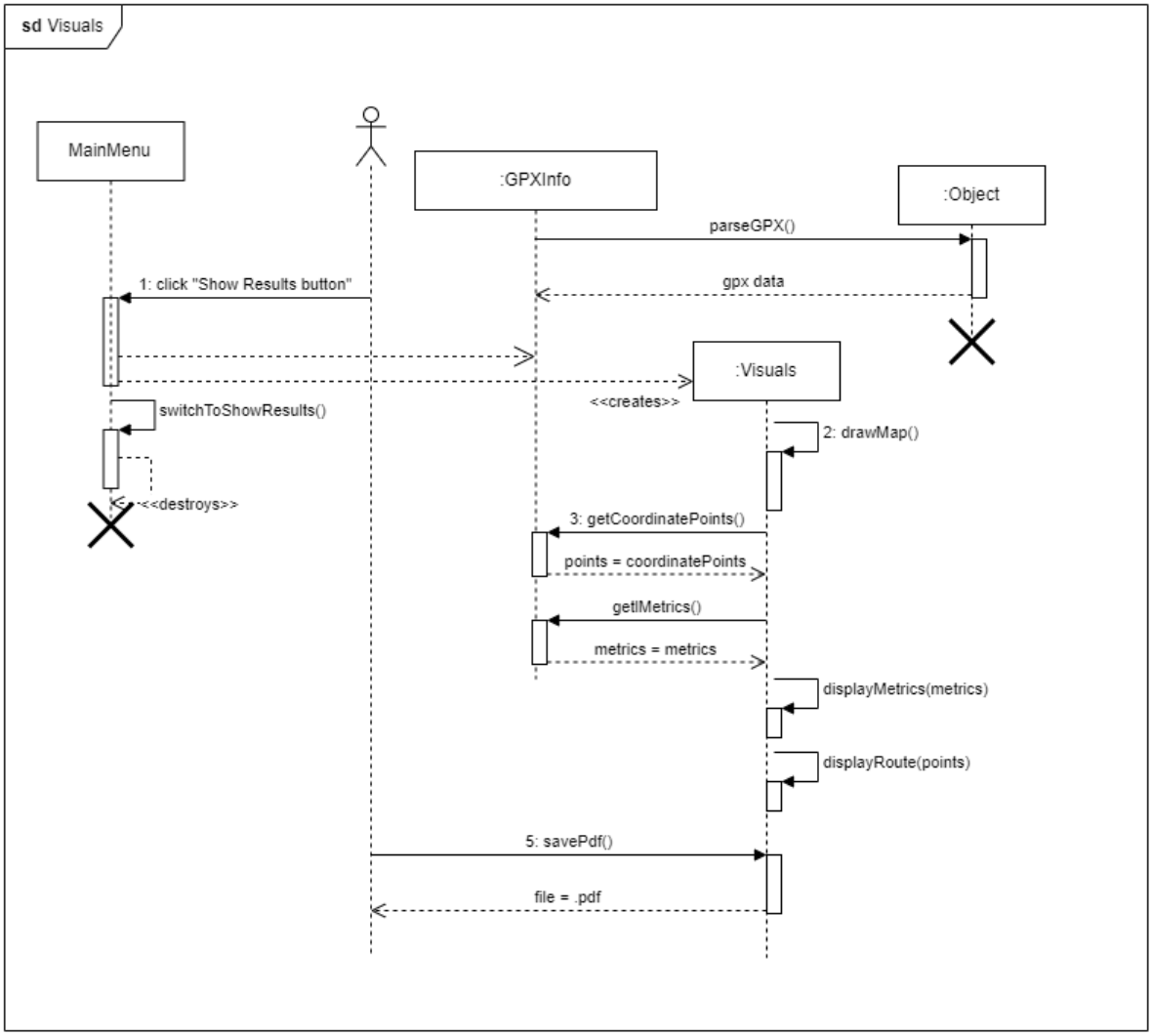

Figure 6. Sequence Diagram: Visualization

The Visualization management and display of the metrics is done by the object **Visuals.** After the users click the button “show results” the UI for the main menu gets destroyed and the map for the display of the data is generated by *drawMap().* In order to get the data needed for the visualization, the functions *getCoordinatePoints()* and *getMetrics()* are called, storing the variables in the **Visuals** instance. Two display functions are triggered after the data is obtained. *displayMetrics(metrics)* and *displayRoute(points)* , these two functions will display information utilising the added library *GMapsFX* in a local version of Google Maps. After all the metrics and the route are displayed, the user has the option to export a snapshot of the **Visuals** instance UI with the map to a pdf file. This is done by the method *savePDF()*

Implementation

Author(s): Bas Dijkstra, Victor Retamal

Implementing from the UML models

To start off, we modelled a very basic class diagram. This was to get everyone on the same page on the start of the implementation. Then we decided it would be best to start implementing a little bit, before we went on to model the rest of our diagrams.
We implemented the basics of the GPX parser and the UI for assignment 2. We then used this to model the rest of our diagrams (state diagram, object diagram etc), and finish the class diagram for our current implementation.
Finally, we finished off the parts of the implementation we deemed necessary at the moment. We will improve our current project and add the other features in the next few weeks.

Key Solutions

At the moment, we have not run into complex issues on the design of our implementation. Our main menu now contains both the code for the main menu screen as well as the code for the result screen, we might want to change that in the feature if our code gets too complex.
For the current assignment, we did not implement a graphic visualisation for our gpx file yet, but we will change that in the feature. At the moment we are just displaying the info of the GPX file in the result screen (elevation, longitude and latitude).

Location of main

The main class to run the project is Main.java (src/main/java/softwaredesign/main.java)

Location of jar

The Fat Jar file is located at:
(SDProject/out/artifacts/software_design_vu_2020_jar/software-design-vu-2020.jar)

Demo video

https://youtu.be/NYhEK38knGo

Time logs

Team number	17		
Member	**Activity**	**Week number**	**Hours**
Group	Meeting	3	1
Bas	Programming	3	2
Group	Meeting	4	1
Bas	Programming	4	4
Bas	Making diagram	4	1
Daniel	Making diagram	4	2
Krasen	UML Diagram	4	4
Victor	Programming	4	4
Group	Meeting	5	1
Bas	Writing report	5	3
Bas	Programming	5	3
Daniel	Making diagram	5	3
Krasen	Writing report	5	4
Group	Meeting	5	2
Victor	UML Diagram	5	5
Victor	Writing report	5	2
			42