

Functional Design



Functional
Design

Table of Contents

Introduction	3
Description of Application	4
The Pilot Application Flow	4
Ground Control Flow	5
Wireframes	6
Requirements.....	10
The Pilot.	10
Business	10
User	10
System.....	10
Ground Control	11
Business	11
User	11
System.....	12
Use case diagrams	13
Pilot UI	13
Ground Control UI.....	15
Quality Assurance	18

Introduction

The software that will be expanded within the entirety of this document was created to solve the problem of controlling a moon rover and record all necessary data, as well as providing a user with an interface to control the device. The team was responsible for managing the interpretation and communication of data between the frog and the software. The goal at the end of this project was to provide a product capable of managing and communicating with the frog based on all requirements specified.

Within this document, the reader will find a descriptive and detailed report of a project to create a User Interface for a Moon Rover environment within Unity. This document is one of two parts, the other being the Technical Design Document, between all these documents you will find every detail described in its entirety.

To summarize the created User Interface, it was created with the Java Swing library to give the user a visual representation of all the controls and actions the frog is capable of. Visualizations of the data from the frog through maps is also included and provided. All important data is also committed to a database with the end of each action.

The architecture of the software consists of a standard Java program with all necessary libraries. These libraries are mostly dependent on the provided Sasa Communications library. This specific library handles all the communication between the different components. The sending and receiving of data were for the team to decide how to interpret and solve.

The entirety of this project was version and managed within Gitlab. Using this tool helped the team to manage and collaborate with each other. Each being able to work independently and push code within a specific time frame. The provided SCRUM layout by Gitlab was also used to link features with user stories and this was split accordingly between the entire team.

By the end of this document the reader should have a clear understanding of the software, as well as the inner workings and architecture.

Description of Application

The Pilot Application Flow

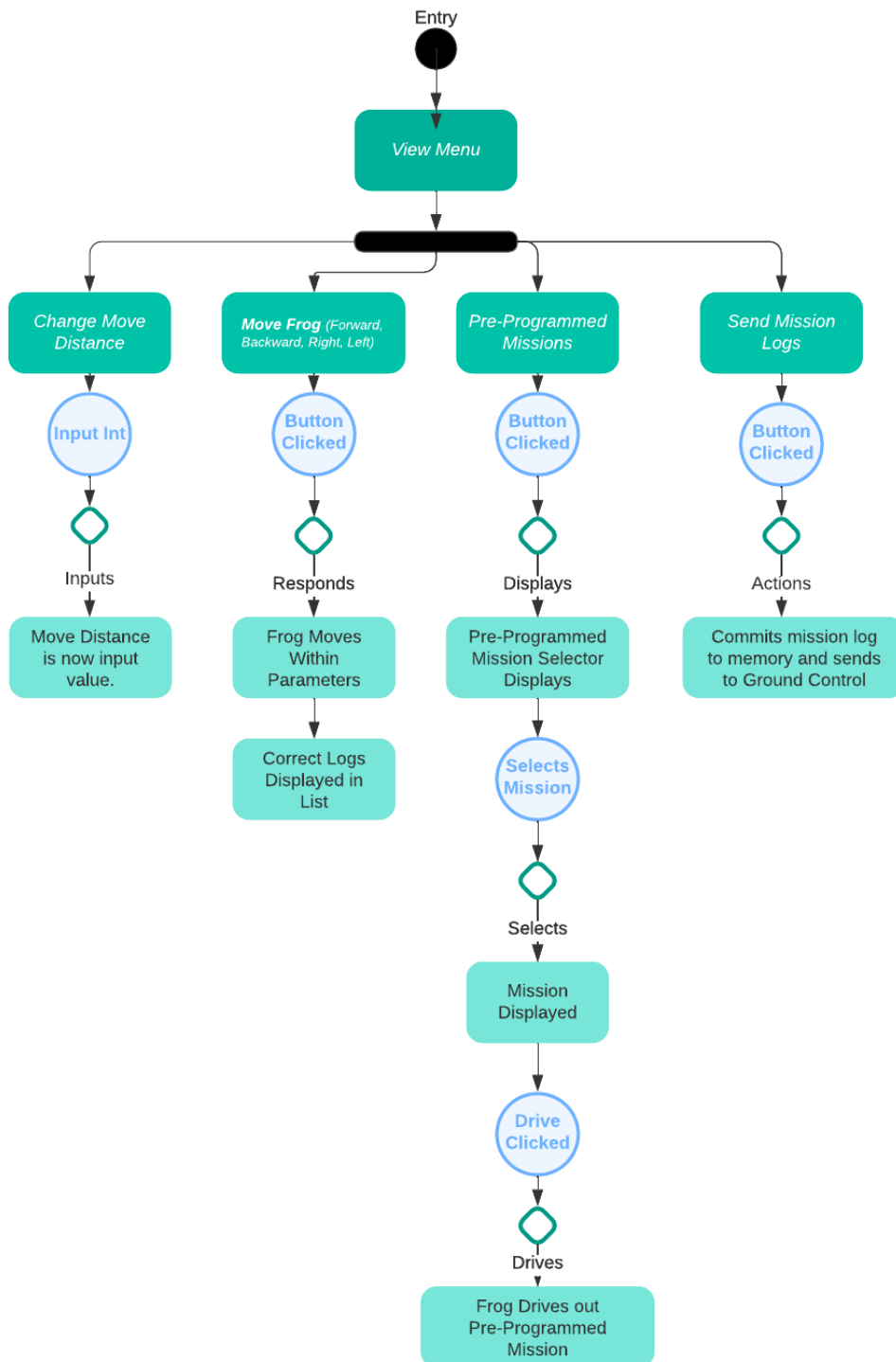
These Diagrams displayed, such as the diagram to the left, will give an idea of the possible actions and the expected flow of the application.

This application flow diagram shown is with regards to The Pilot UI and all possible actions the user can take.

The diagram starts with the Main Menu of the UI, this being the **Change Move Distance, Move Frog Controls, Pre-Programmed Missions and Send Mission Logs**, it then takes into account the actions and re-actions the UI takes based on what the user clicks.

Flow of Application

The Pilot | Functional Design



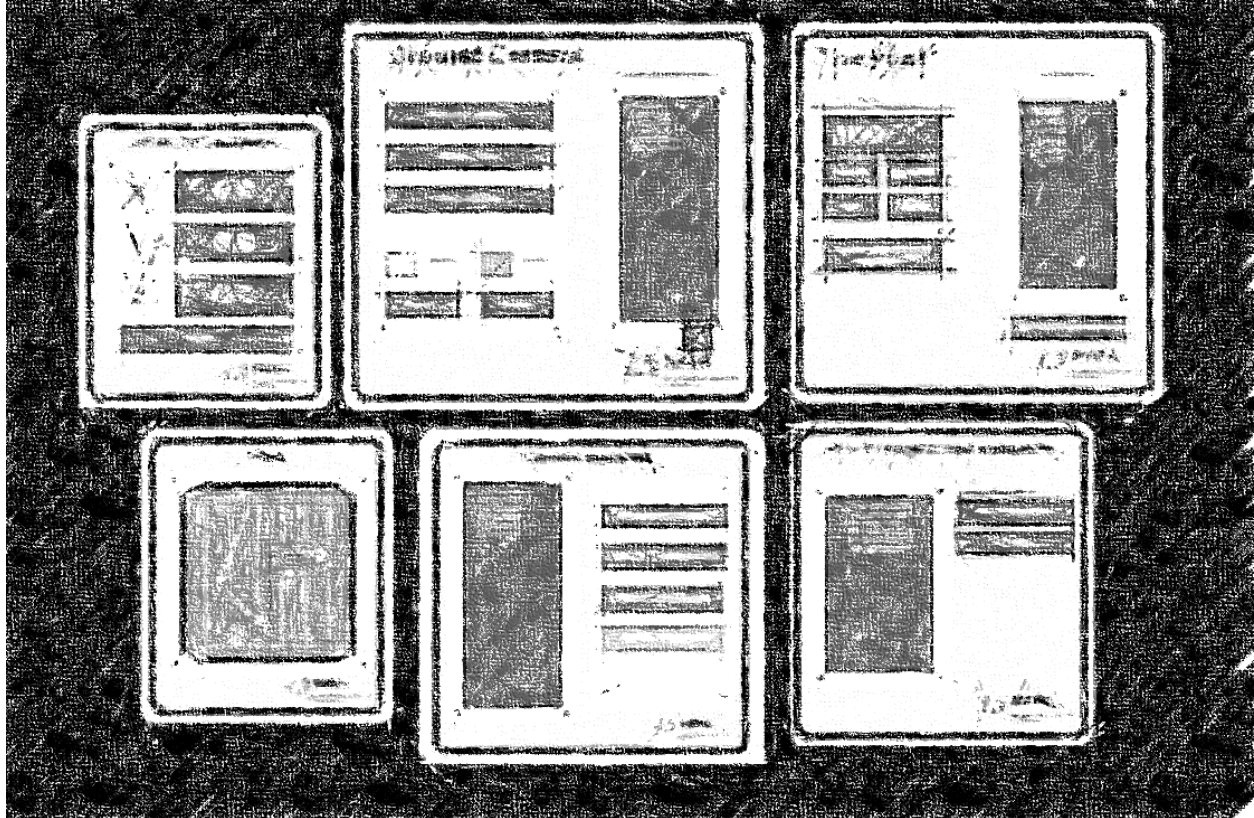
Ground Control Flow



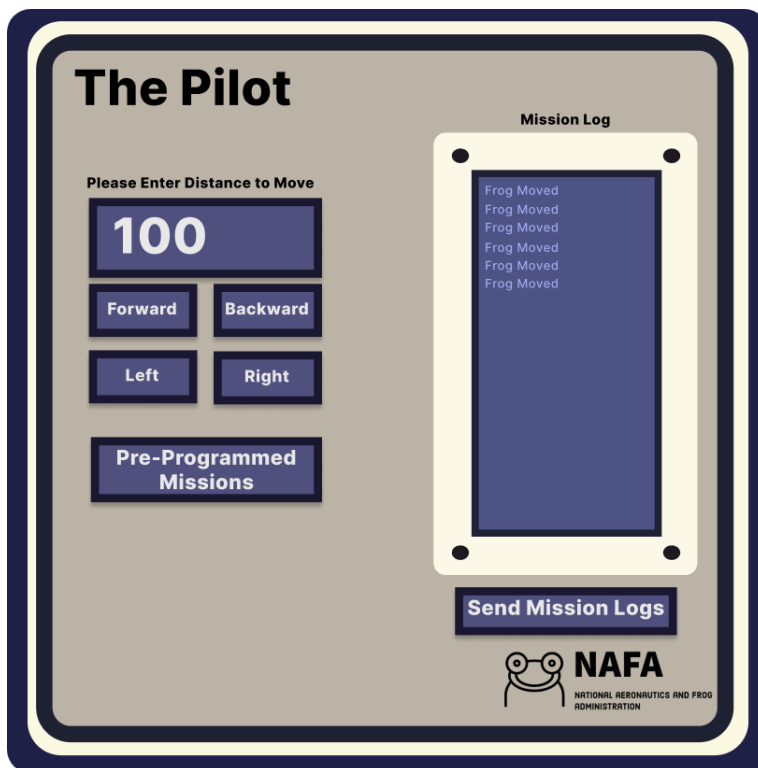
This Diagram is the application flow of the Ground Control. Following the same concept of the previous diagram, it starts with the main menu of the Ground Control and gives the following subset flows through the action that the button takes.

Wireframes

The Wireframes created for the Ground Control and Pilot consists of six different pages or sets of User Interface. The UI is predominantly split between Ground Control and The Pilot. Within this section of the document, design choices and functionality of these screens will each be explained.



The Pilot



This is the main User Interface for The Pilot. All Controls will be set up from here.

This wireframe consists of the main controls of the **Frog** from the **Pilot**. It allows the user to enter in the distance that would be required to travel. After entering this in, the user is then able to move forward, backward, right or left.

This UI also contains a button for "Pre-Programmed Missions". This will open a separate window when clicked. This window will allow the user to select a pre-programmed mission as specified from the **Ground Control**.

The list located on the right of the UI is the mission logs. All the actions the **frog**

takes will be recorded and displayed here. Whether that is current location or commands inputted from the pre-programmed missions. It will also be possible to send these mission logs to the ground control.



This will be the UI of the popup window from clicking pre-programmed missions. It will allow you to select from a list of missions, set by the **Ground Control** and then proceed to tell the **Frog** which actions to take based on what was recorded.

Once drive is clicked, the background algorithms will take over and drive the **Frog** to the desired location. All these actions will be recorded within the Mission Log.

The Ground Control

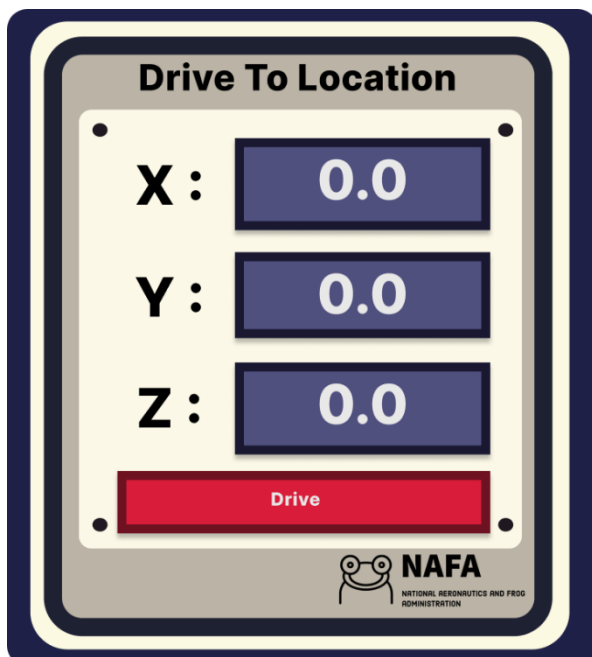


This will be considered as the main User Interface for the **Ground Control** and all actions are capable from here.

On left container, there is an array of buttons with different actions and screens. The first of these being a "Drive to Location" button. This button will allow the user to summon a screen with which to give the **Frog** drive commands.

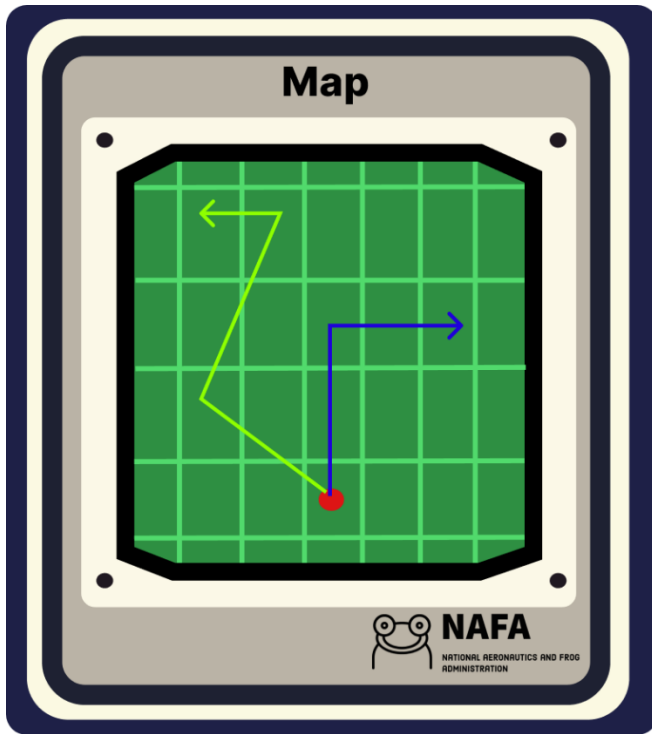
The next button is a "Display Map". This will start up a screen with the a map of the terrain and the **Frog's** current location.

The "Create Mission" button will summon a window with a separate UI to be able to send another Pre-Programmed mission to the **Pilot**. The **Pilot** will then be able to choose and select any number of these.



At the bottom of this left section with the **Ground Control** UI, the user will find two checkboxes and two buttons. These are one in the same. To be able to import and export Collision points, the user needs to have the "Collisions" checkbox selected. Similar with the Missions checkbox. This will allow the user to import or export missions or collisions.

This screen on the left is the "Drive To Location" screen. It allows for the user to input an X, Y and Z co-ordinate to drive the **Frog** to. Once these locations have been correctly inputted, the **Frog** will then adhere to the command and do its best to travel to the desired location. All algorithms for avoidance and automatic driving will be implemented for this type of action.



This is the map available to the user from within the **Ground Control**. It will contain the current location of the **Frog** and display a ground map to give as accurate of a representation of the **Frogs** location.

It should be able to show the collision points and routes the **Frog** should take.



This is the User Interface for "Create Mission". It allows for you to enter the mission's name, a command for the frog to follow and then to add this command. When "Add Command" is clicked, the list on the left is the updated with appropriate command.

"Save Mission" will populate the "Pre-Programmed" Missions on the Pilot with all of these commands allow the **Pilot** to select one to follow.

This concludes all the screens that will be accessible within our UI. All of which provided allows for full control of the **Frog** and a communication interface. Most actions will be recorded within the Database.

Requirements

The Pilot.

Business

No.	Requirement	MSCROW
F-TP-B-01	The pilot and ground control need to communicate using the SASA library	Must
F-TP-B-02	The pilot and ground control both store information in their own respective database	Must
F-TP-B-03	The application has a UI made using SWING	Must
F-TP-B-04	Users only needs to run the application to be able to control the frog and store and export the received data from the frog.	Should
F-TP-B-05	The pilot and Ground control must be written in java	Must

User

No.	Requirement	MSCROW
F-TP-U-01	Drive manually using onscreen buttons	Must
F-TP-U-02	Display mission logs	Should
F-TP-U-03	Display preprogrammed missions	Should
NF-TP-U-04	UI is easy to use and buttons explain their functionality properly	Should

System

No.	Requirement	MSCROW
F-TP-S-01	Execute pre-programmed missions	Must
F-TP-S-02	Send mission log	Must
F-TP-S-03	Import data such as missions and collision points	Must
F-TP-S-04	export data such as missions and collision points	Must

F-TP-S-05	Execute preplanned missions	Must
F-TP-S-06	Drive to Location	Must
F-TP-S-07	Display (collision and position) maps	Must
F-TP-S-08	Update mission logs	Must

Ground Control

Business

No.	Requirement	MSCROW
F-GC-B-01	The pilot and ground control need to communicate using the SASA library	Must
F-GC-B-02	The pilot and ground control both store information in their own respective database	Must
F-GC-B-03	The application has a UI made using SWING	Must
F-GC-B-04	Users only needs to run the application to be able to control the frog and store and export the received data from the frog.	Should
F- GC-B-05	The pilot and Ground control must be written in java	Must

User

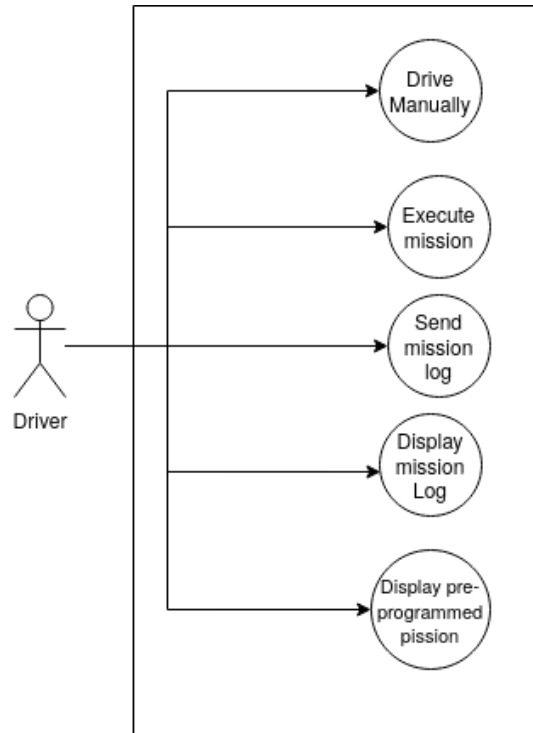
No.	Requirement	MSCROW
F-GC-U-01	Display map	Must
F-GC-U-02	Display frog location on the map	Must
F-GC-U-03	Display mission logs	Must
F-GC-U-04	Drive to location.	Must
F-GC-U-05	Update mission logs	Must

System

No.	Requirement	MSCROW
F-GC-S-01	Create missions for the frog	Must
F-GC-S-02	Drive to location	Must
F-GC-S-03	Import missions	Must
F-GC-S-04	Export missions	Must
F-GC-S-05	Import collision points	Must
F-GC-S-06	export collision points	Must

Use case diagrams

Pilot UI



Title	Drive Manually
ID	UC1
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	<ol style="list-style-type: none">1. User selects the drive manually option2. A textbox with a button is presented3. User enters a command and presses the submit button4. Rover executes the command<ol style="list-style-type: none">4.1. If rover radar has detected a collision point the Frog will avoid it or stop
Postcondition	Rover has executed a command

Title	Execute Missions
ID	UC2
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	<ol style="list-style-type: none">1. User selects the executable missions option2. User is presented with a drop-down menu and a button3. User selects one of the uploaded missions from the Ground Control4. User presses the submit button<ol style="list-style-type: none">4.1. Rover executes the list of commands one by one

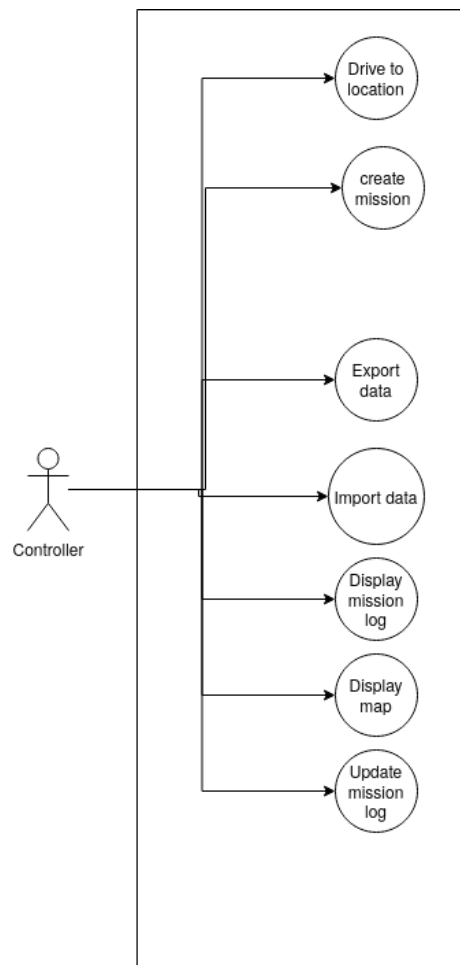
	4.2. The Frog will avoid collisions for every executed command
Postcondition	Rover has executed a Preprogrammed mission

Title	Send Mission Log
ID	UC4
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. User selects the Send Mission log option 2. User is presented with a complete history of all the executed drive commands so far and a button 3. User presses the send button 4. The mission log is sent to the Ground Control and the local Mission Log DB is cleared
Postcondition	Rover sends a Mission log

Title	Display mission log
ID	UC5
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. User is viewing the main frog page
Postcondition	Mission logs list is updated

Title	Display pre-programmed missions
ID	UC6
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. User selects the preprogrammed mission's option on the main pilot ui. 2. User is displayed with a list of all missions and is able to select one to execute.
Postcondition	Users are presented with a list of the pre-programmed missions which they can select from.

Ground Control UI



Title	Drive to location
ID	UC6
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. User selects give destination option 2. User is presented with 3 text columns and a send button <ol style="list-style-type: none"> 2.1. Each text column is mark (X,Y,Z) 3. User fills up the coordinates and presses the submit button
Postcondition	The System sends coordinates to the Frog

Title	Create preprogrammed missions
ID	UC7
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. User selects Create a mission option 2. User is presented with 2 text field and 2 button and a drop-down menu <ol style="list-style-type: none"> 2.1. The first text field is to give the mission a name 2.2. The next text field is to write down the first command

	<ol style="list-style-type: none"> 2.3. The first button is to add new text field for the following command 2.4. The drop-down menu is to select the map this mission is meant for 2.5. The last button is to submit the mission 3. The mission is sent to The Frog 4. The same mission is saved locally
Postcondition	Operator sends mission to the rover

Title	Display Map
ID	UC8
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. Operator selects map option 2. User is presented with the lastly requested map <ol style="list-style-type: none"> 2.1. The map is a white board with red dots marking the collision points 2.2. The current position of the rover is also displayed 3. The user has update map button to request from the rover the new collision points it has detected 4. The recorded missions are also displayed with lines
Postcondition	Operator can see the current location of the rover the available missions and the collision points on the map

Title	Export data
ID	UC9
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1. Operator selects export data option from the menu 2. User is presented with 2 options: Export Missions and Export collision-points 3. User selects one of the 2 options 4. The data is exported into SCV-file
Postcondition	Operator exports required data

Title	Import data
ID	UC10
Actor	Operator
Precondition	Operator has control of the Frog UI

Flow	<ol style="list-style-type: none"> 1. Operator selects import data option from the menu 2. User is presented with 2 options: Import Missions and Import collision-points 3. User selects one of the 2 options 4. The user is presented with a text field to file up the SCV-file path 5. User presses the import button <ol style="list-style-type: none"> 5.1. If the path is wrong user is presented with a message
Postcondition	Operator imports required data

Title	Display mission logs
ID	UC11
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	1.Users are presented with a list of mission logs on the main Ground control screen
Postcondition	

Title	Update mission log
ID	UC12
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	<ol style="list-style-type: none"> 1.Operator selects the update button under the mission logs list. 2.Operator is presented with a new list of mission logs.
Postcondition	Operator imports required data

Quality Assurance

Test Plan

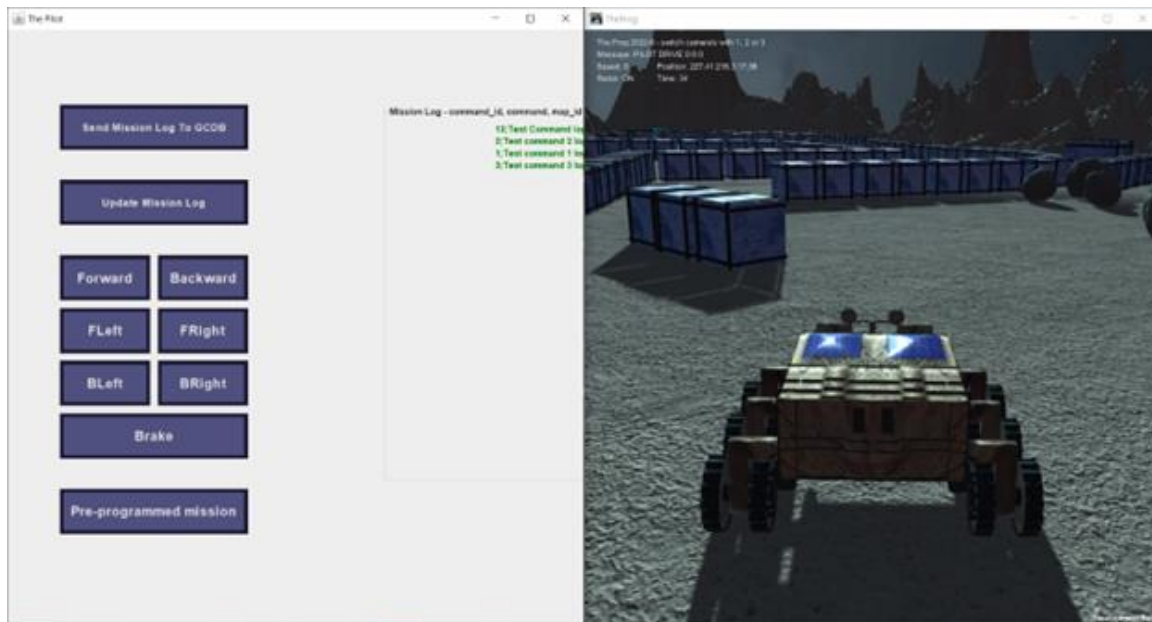
Before finalizing the project, testing will be carried out to ensure that each needed requirement for this project is integrated and functional. The primary test that will be carried out is the acceptance test and if possible other types of testing will also be performed. During the testing, the requirements will be looked at and from an end-user perspective, the testing will be served. As the user performs the test on the project and checks each integrated requirement function as intended then the test for the requirement is noted down as a pass. In the case the performed test on a requirement is a fail, it will be noted down what is wrong and fixed and the test will be performed again to ensure the fixes is a pass.

Testing

Below details regarding testing done to each requirement can be found alongside screenshots.

F-TP-U-01

The testing for the manual control for the Pilot is carried out through the following actions. The pilot is booted up and then the frog establishes a connection. Once the pilot is running, is it checked if options for manual driving are present in the UI, which during the testing it is. This then allows the testing to continue to the functionality of the manual control. One of the buttons is clicked and is checked if the frog receives any commands and does what the button intends. During the testing, it is seen that the button functions as intended which concludes the test to be a pass for the requirement.





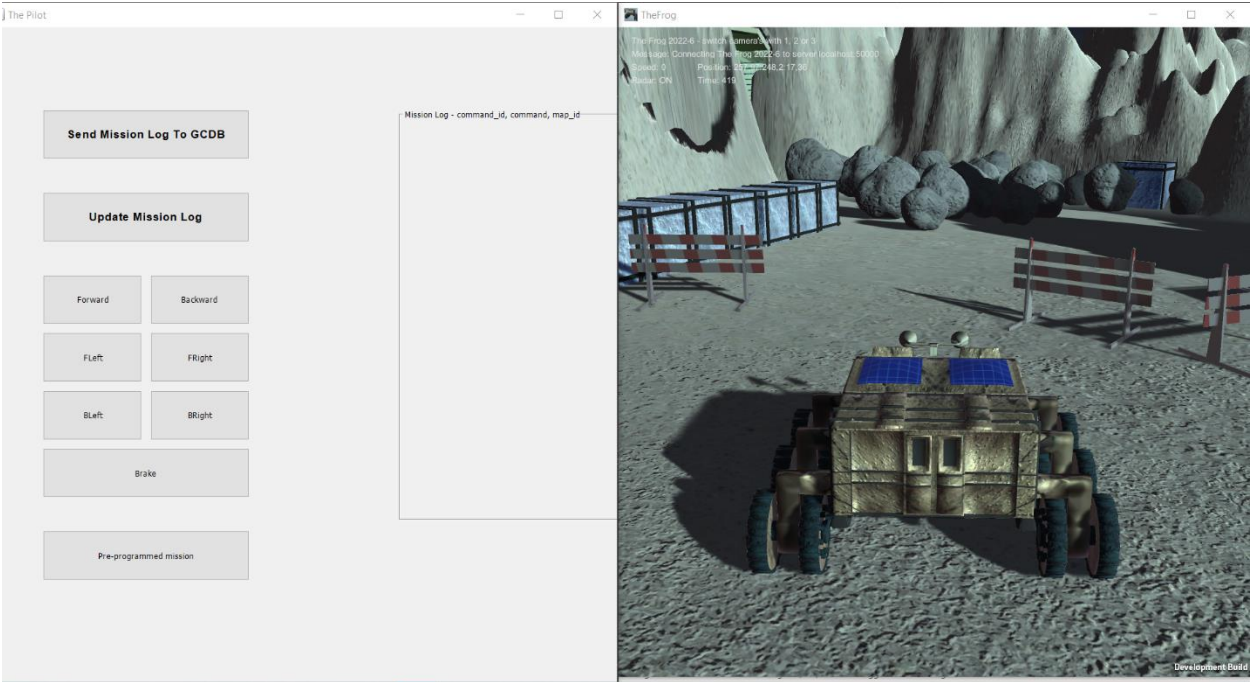
F-TP-U-04 and F-TP-S-06

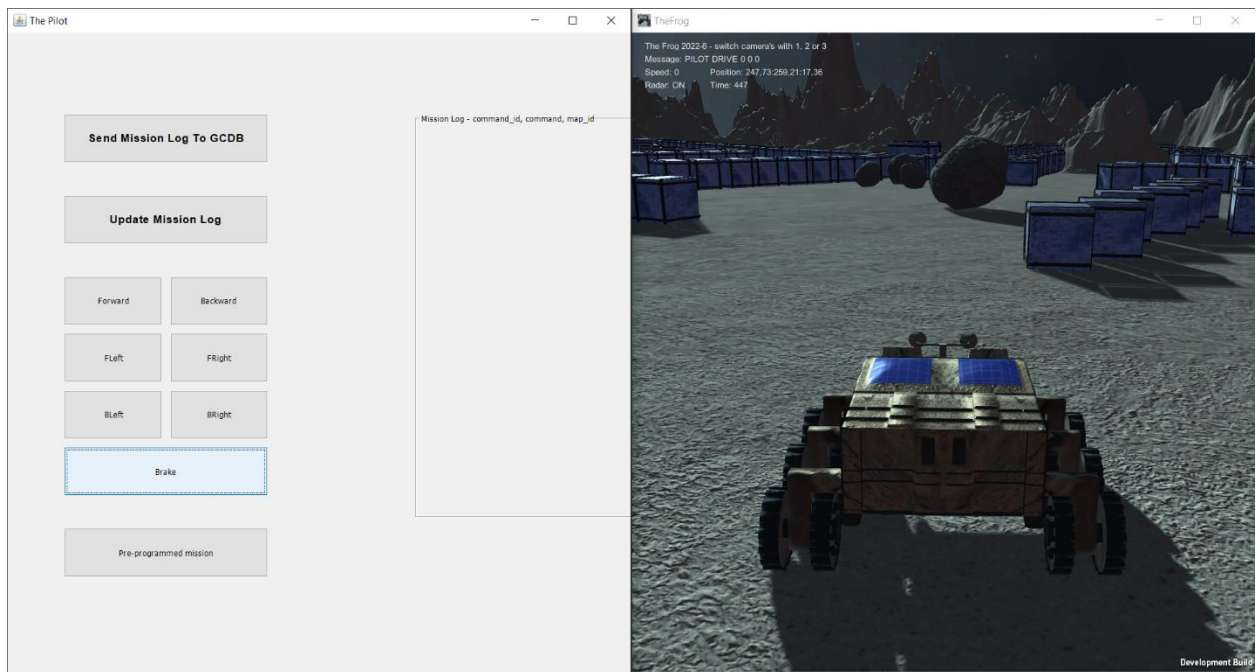
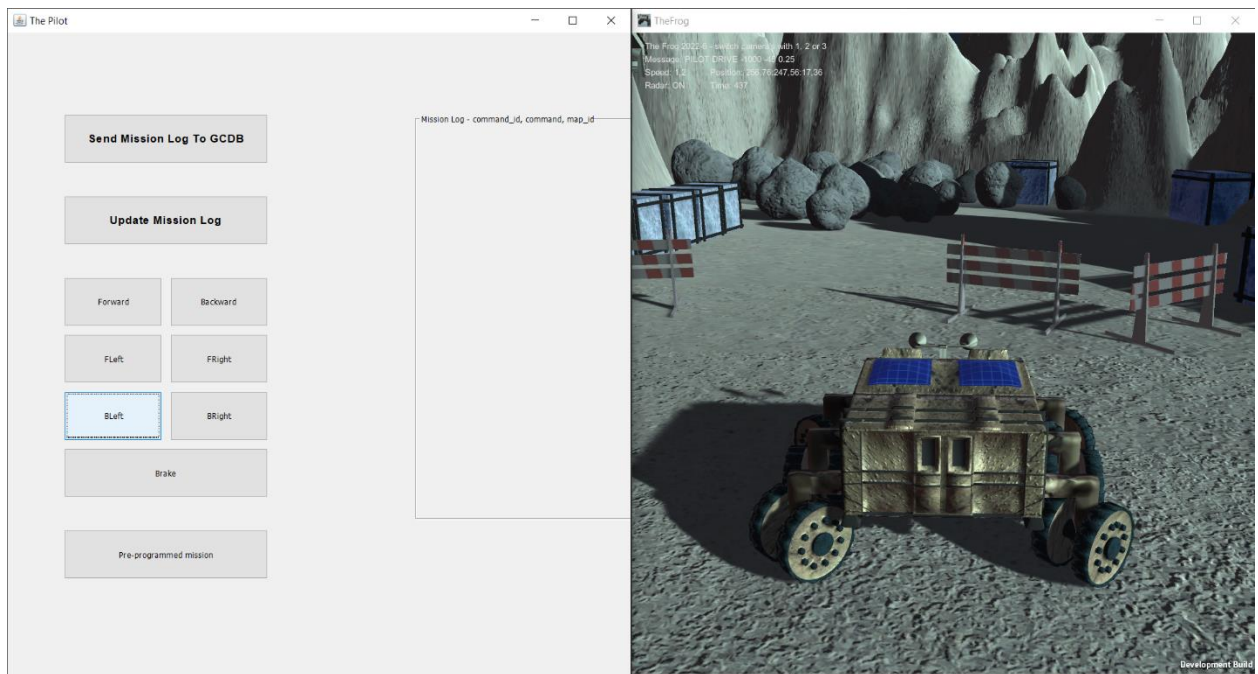
During this testing, the execution of the pre-programmed mission is tested. The following actions were taken in the testing of the feature. The Pre-Programmed mission button is clicked which should open a new window with a list of mission names that is present within the database alongside a drive button and text field that displays the name of the mission that is selected. In the case of the test, this is observed as intended. Then a working mission is selected, and the drive button is clicked which then checked if the frog receives each command found in the database that is associated with the selected mission. To validate this, the console is checked to which commands were sent to the frog, and then the database is checked to see if the commands are associated with the correct mission and is observed that the feature functions as intended.

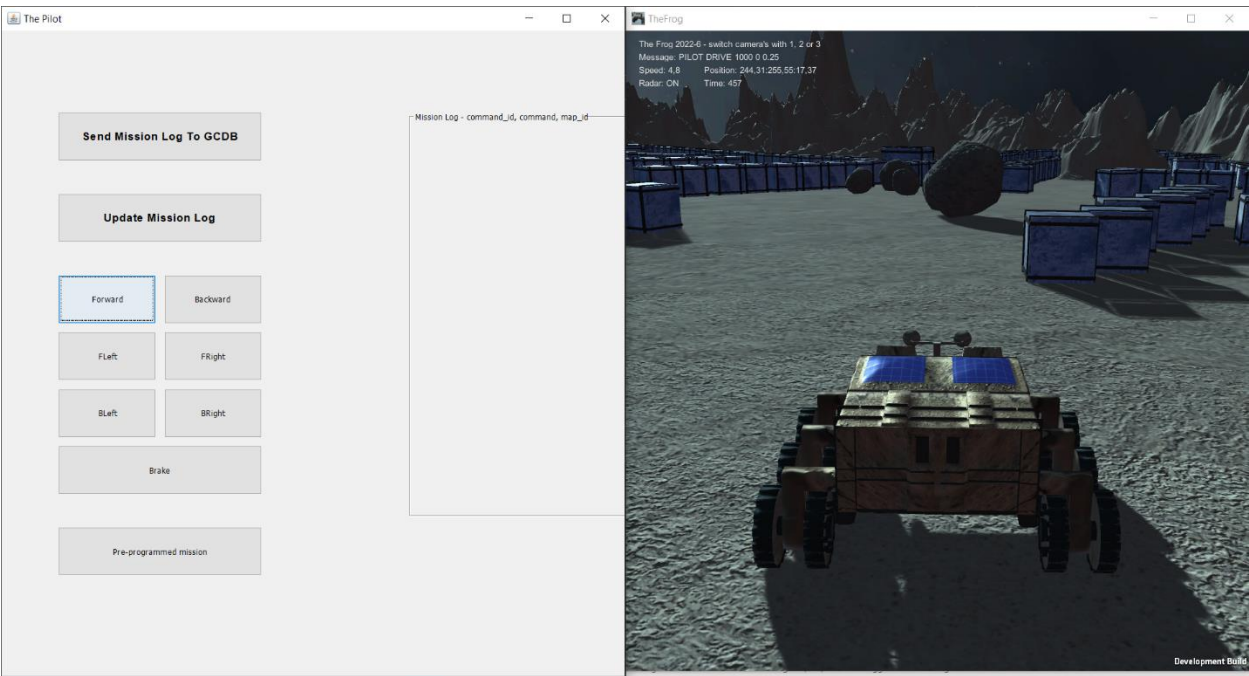
7	94	pilot drive 900 45 12	3
8	95	pilot drive 500 0 12	3
9	96	pilot drive 400 -15 2	3
10	97	pilot drive 500 0 7	3
11	98	pilot drive 400 15 4	3
12	99	pilot drive 500 0 3	3




F-TP-S-02

The following test that will be carried out will cover the requirement requiring sending commands to be logged in mission logs. First, it is ensured there is no data within the database for mission logs. Second, multiple commands are sent to the frog through manual driving. And lastly, the database is checked to if data is stored in the database. In which the requirement functions as intended and concludes the test as a pass.



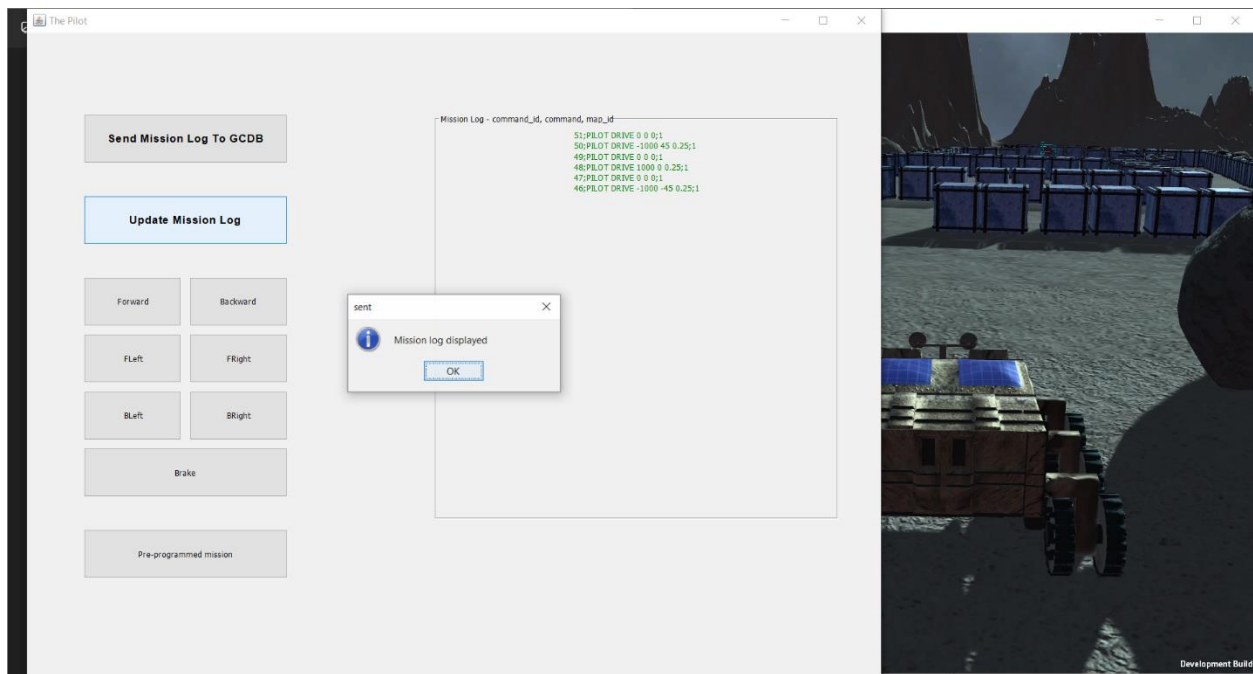




	 id ↕	 command ↕	 map_id ↕
1	46	PILOT DRIVE -1000 -45 0.25	1
2	47	PILOT DRIVE 0 0 0	1
3	48	PILOT DRIVE 1000 0 0.25	1
4	49	PILOT DRIVE 0 0 0	1
5	50	PILOT DRIVE -1000 45 0.25	1
6	51	PILOT DRIVE 0 0 0	1

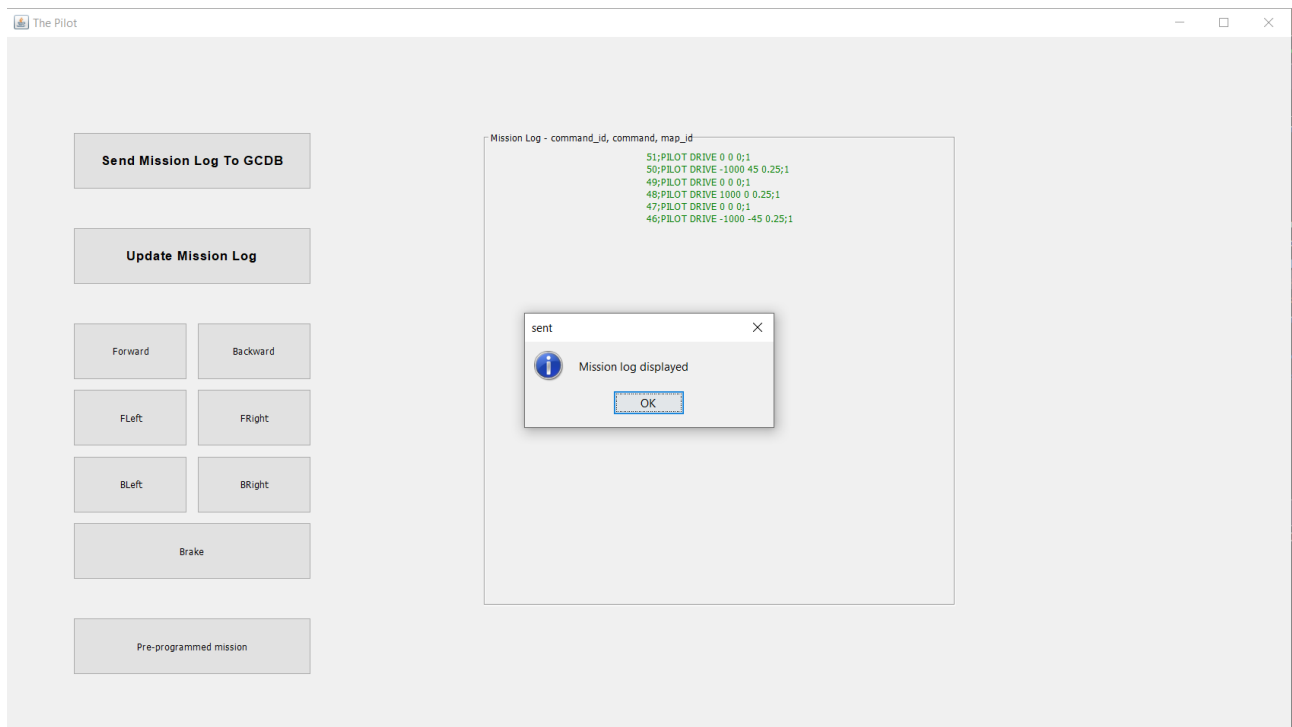
F-TP-S-08

The following test will be carried out to test the requirement, the update mission logs button is clicked to update the mission log display with any data logged to the database. In which the logged data is displayed on the mission log display. In which the requirement functions as intended as the test concludes as a pass.



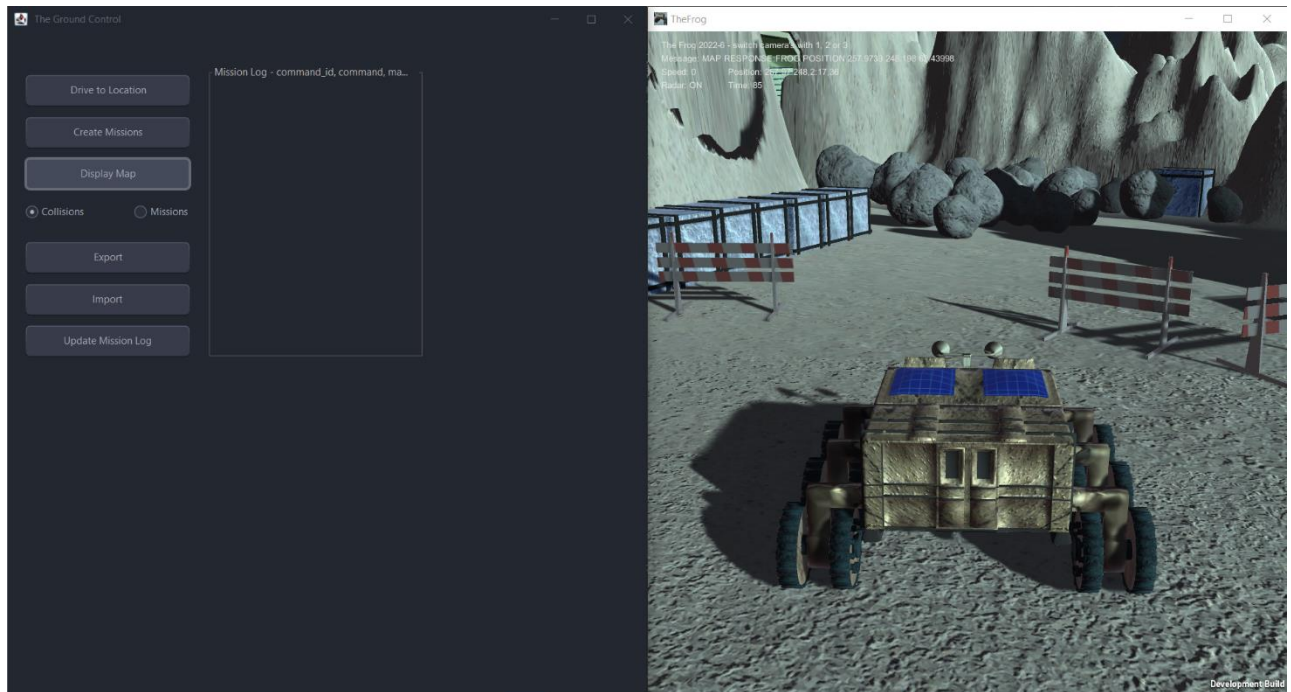
F-TP-U-02

The following test will be carried out to test the requirement, the UI is loaded up and is checked if any mission log data present in the database is loaded into the display at start. In which the requirement functions as intended as the test concludes as a pass.

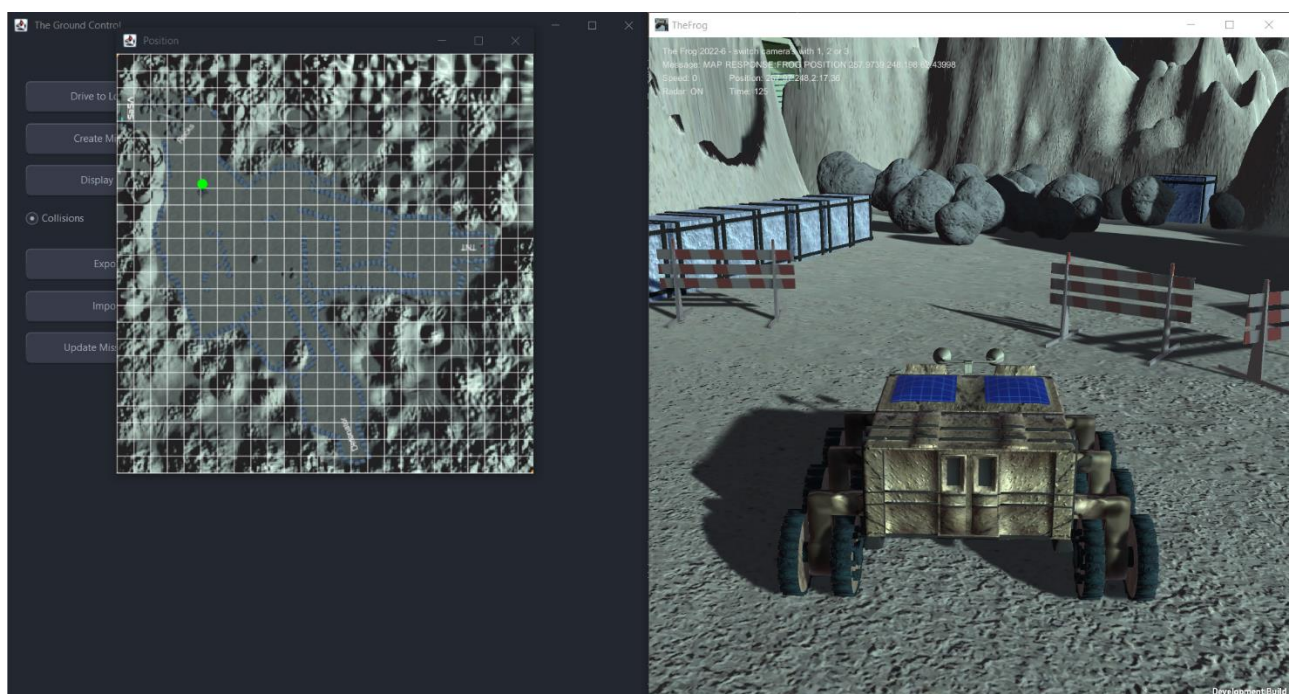


F-GC-U-01

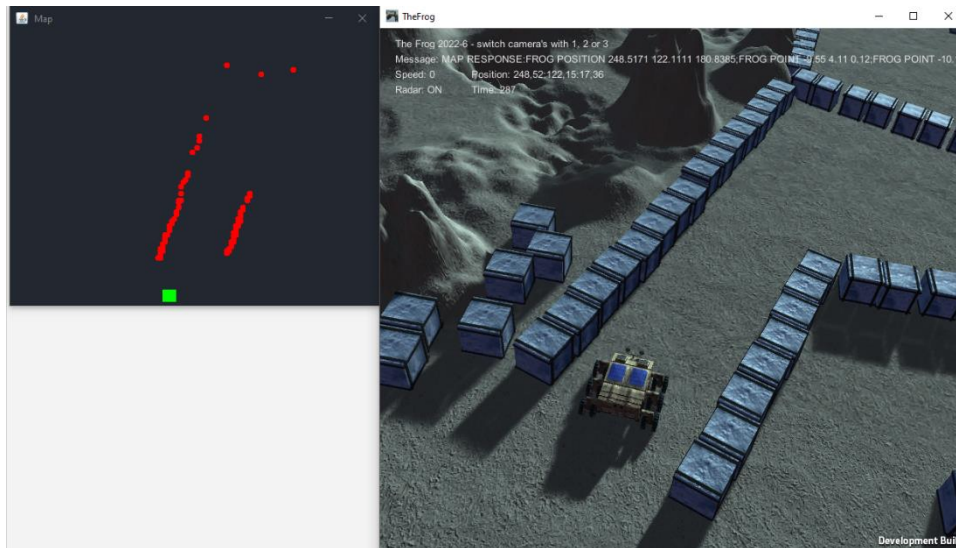
The following test will be carried out to test the following requirement. The ground control is booted up and then the frog is booted up. Display map is clicked on the ground control UI. In which a map should be displayed. Once the map is displayed it is then compared with the frog location. In which the requirement functions as intended as the test concludes as a pass.



This is the resulting screen when “Display Map” is clicked. This window is an external one showing the current position within the green dot.



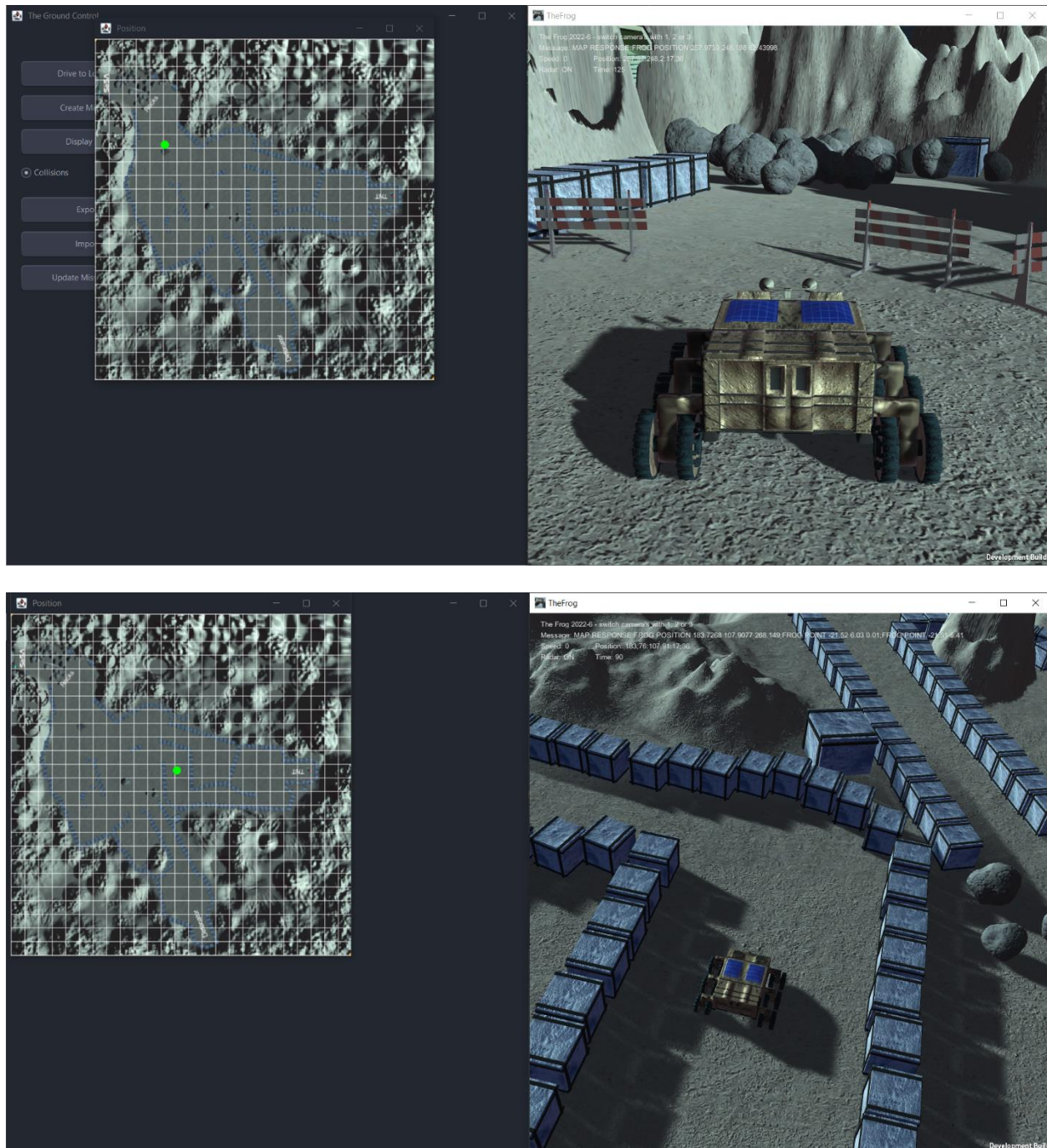
The collision map displays the collision points scanned by The Frog's radar. It is (reasonably) demonstrated and considered acceptable in the below screenshot that the map of the collision points functioned properly as expected, judging from the general shape of the corridors on the map and TheFrog.exe



F-GC-U-02

The following test will be carried out to test the following requirement. The following steps are taken during the test. First, the ground control is booted up and then the frog is booted up. Second, the display map is clicked on the ground control UI. In which a map should be displayed. Once the map is displayed it is then compared with the frog location. Third, the frog is moved to a different position and the map is reloaded. Lastly, once the map is reloaded, the displayed location is compared to the frog's new location. Which is found the displayed location on the map matches the location of the frog. The

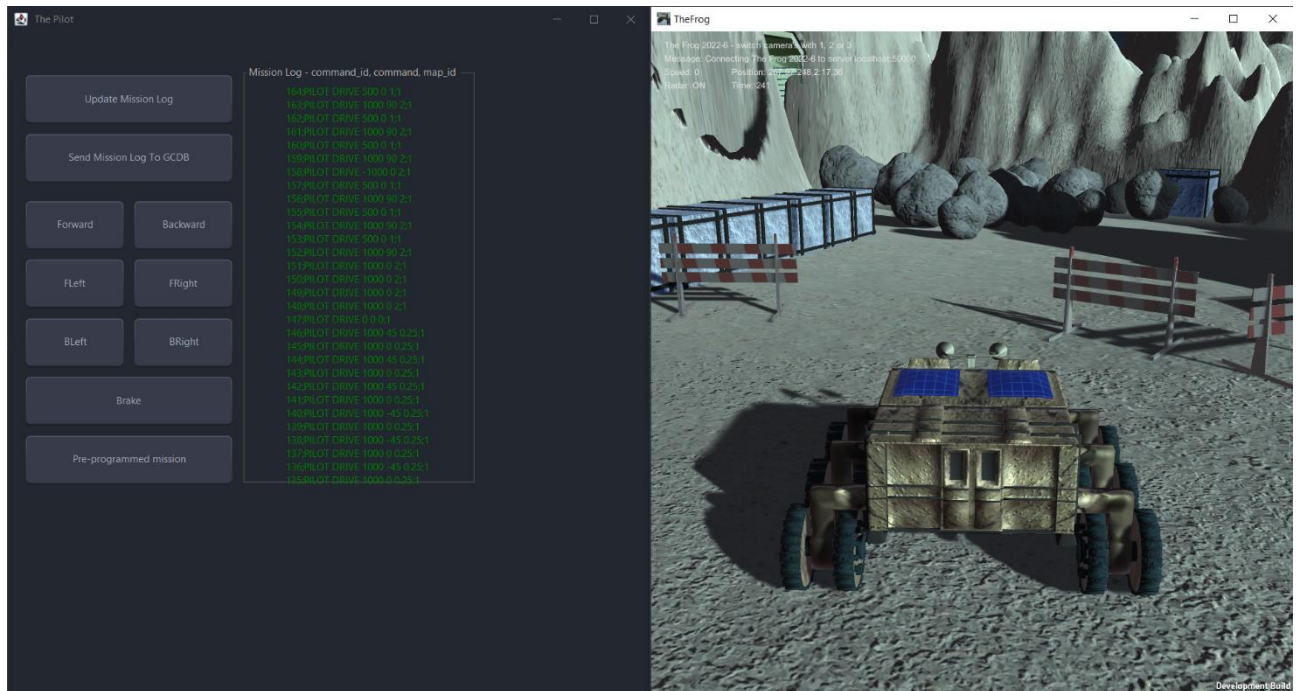
requirement functions as intended as the test concludes as a pass.

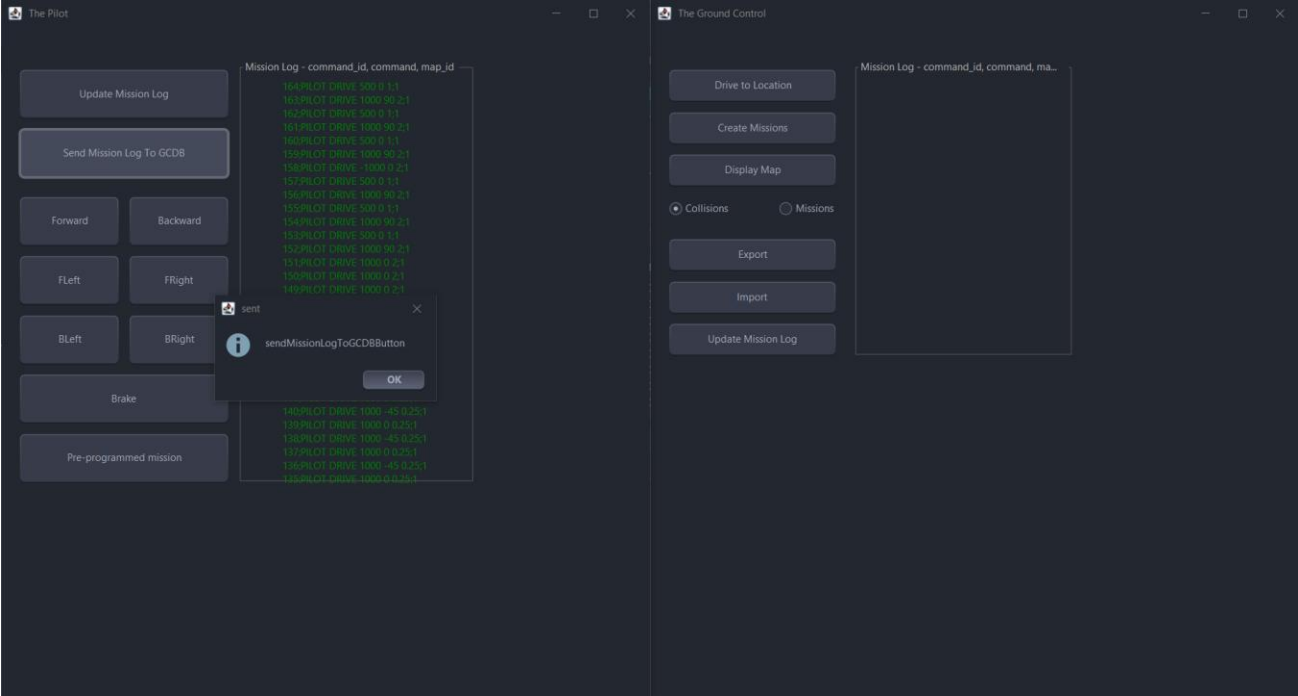
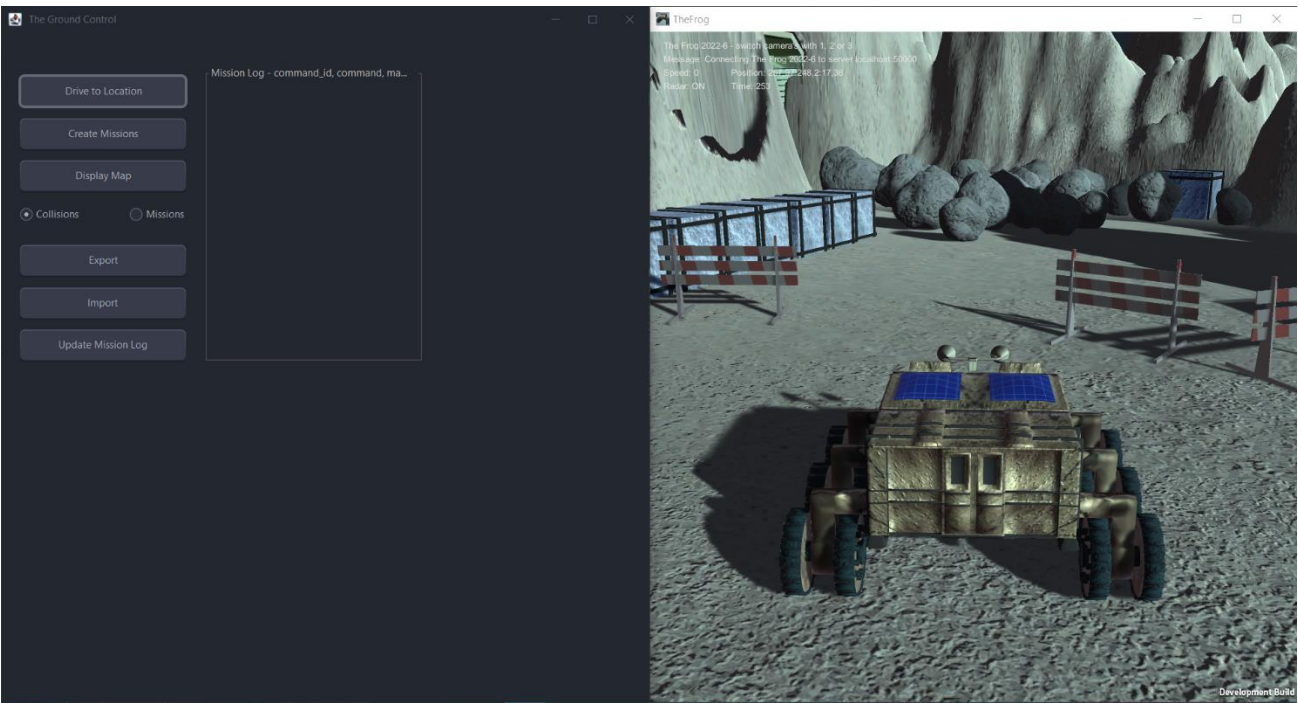


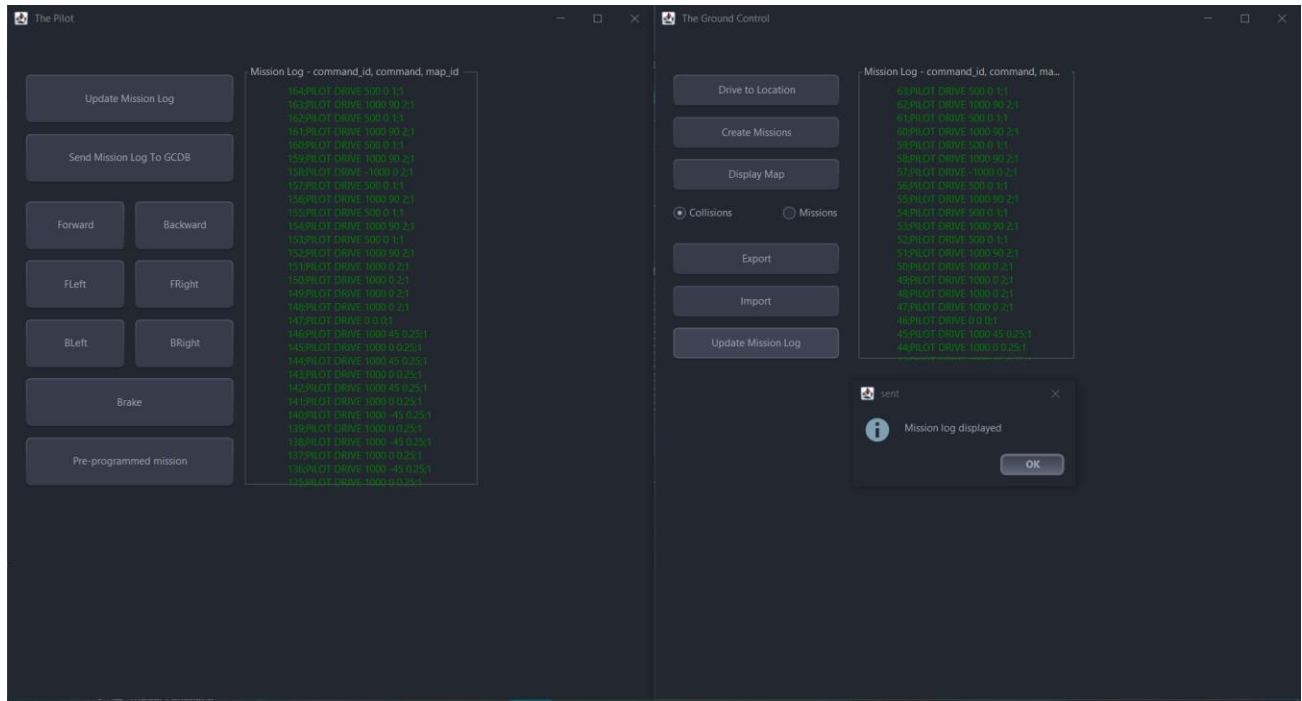
F-GC-U-03 and F-GC-U-05

The following test will be carried out to test the following requirement. The following steps are taken during the test. First, the pilot is booted up which automatically executes the frog. Second, the Ground Control is booted up. Third, it is checked if the pilot database contains any mission log data. This is present through the mission log data. Fourth, send mission log to ground control database is clicked. Lastly, the update mission log button is clicked on Ground Control UI which is then seen data from the

pilot is displayed in the ground control mission log. In which the requirement functions as intended as the test concludes as a pass.

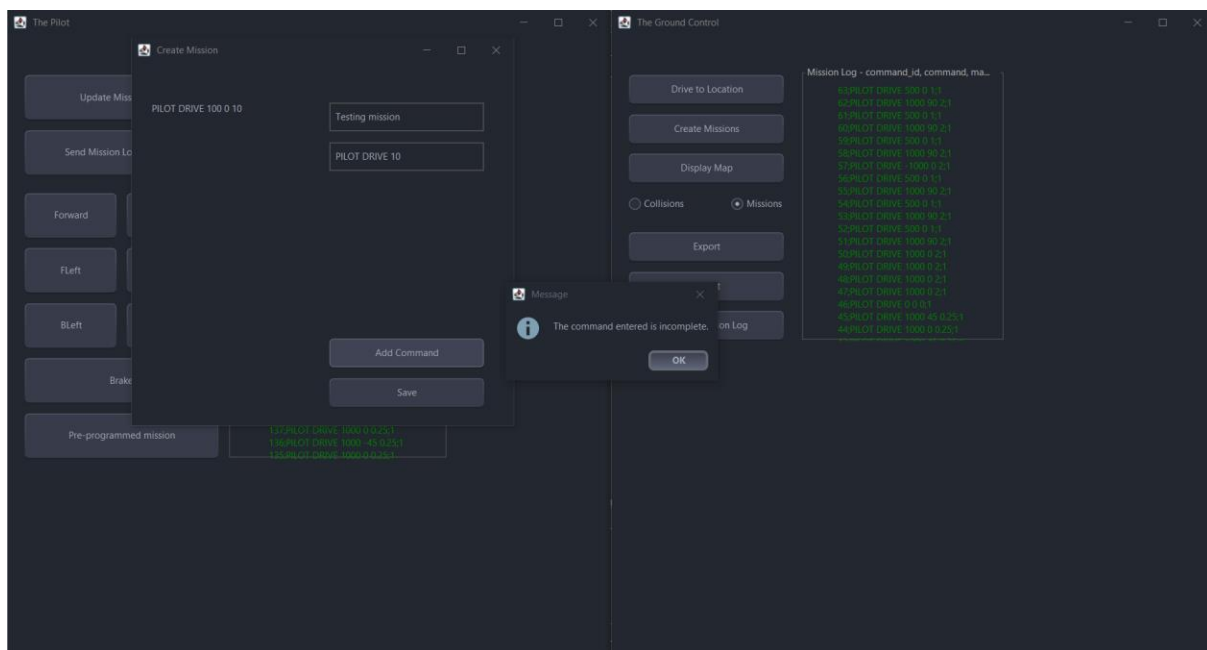
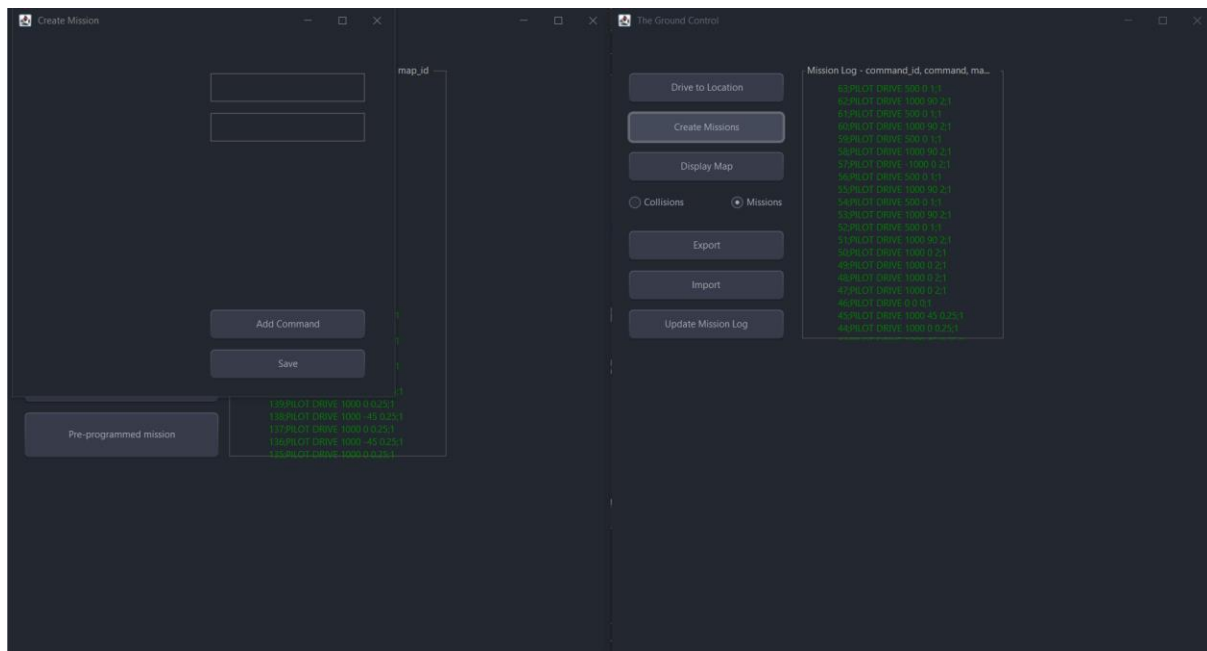


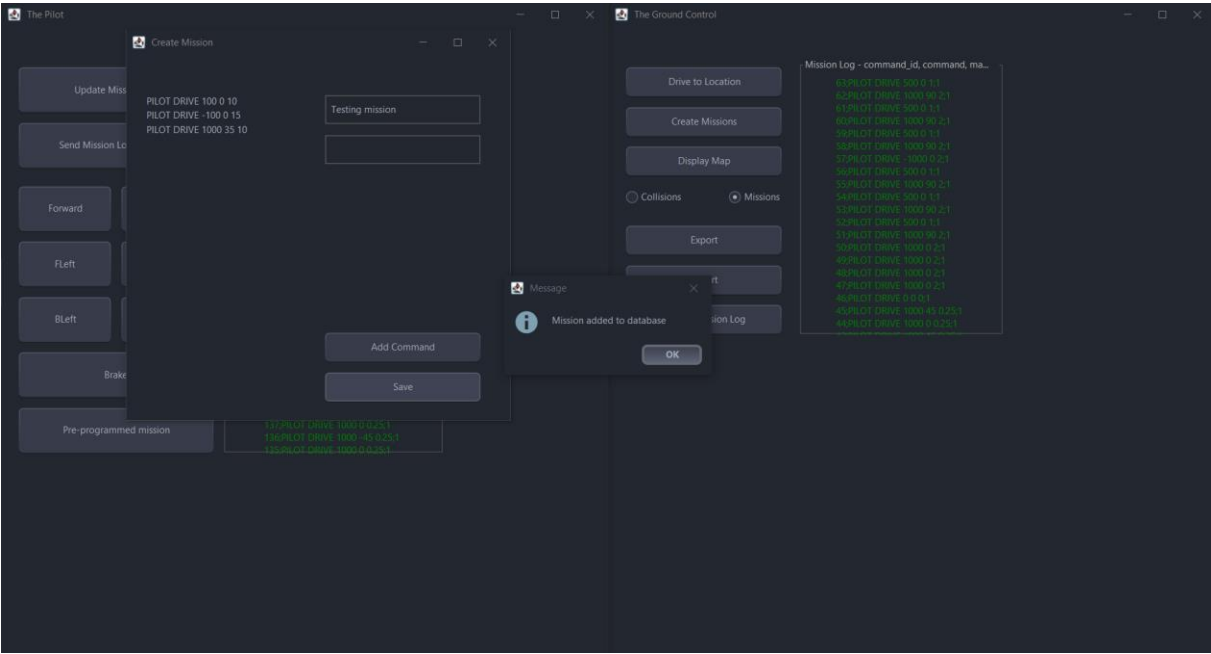
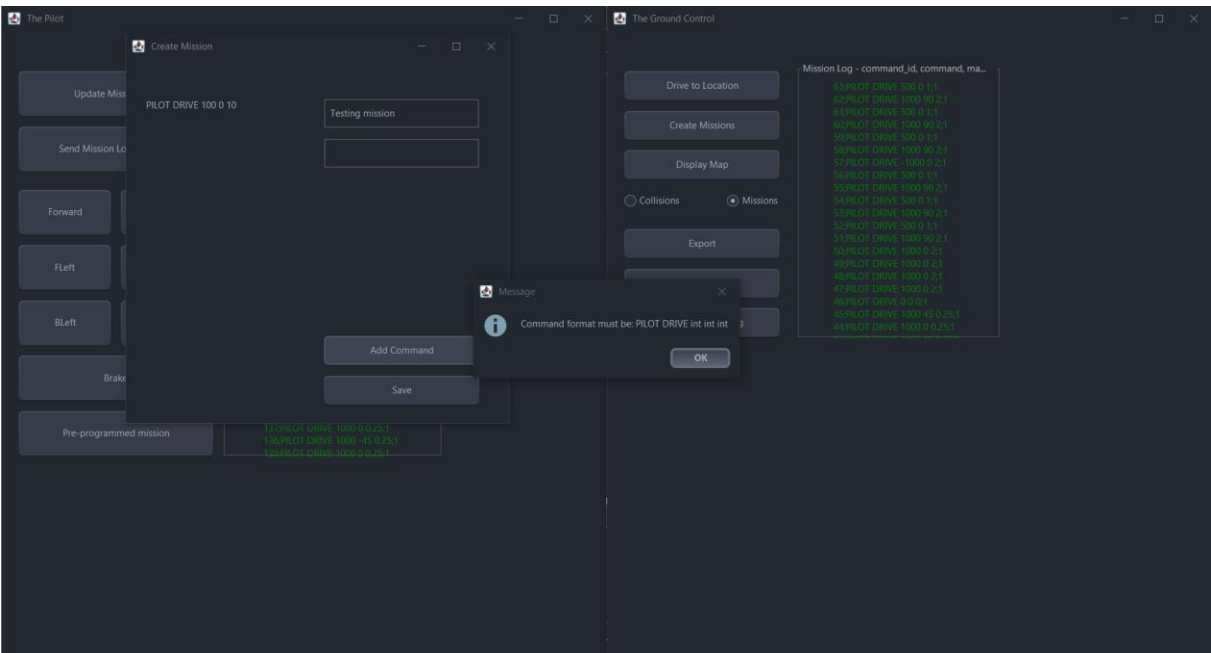


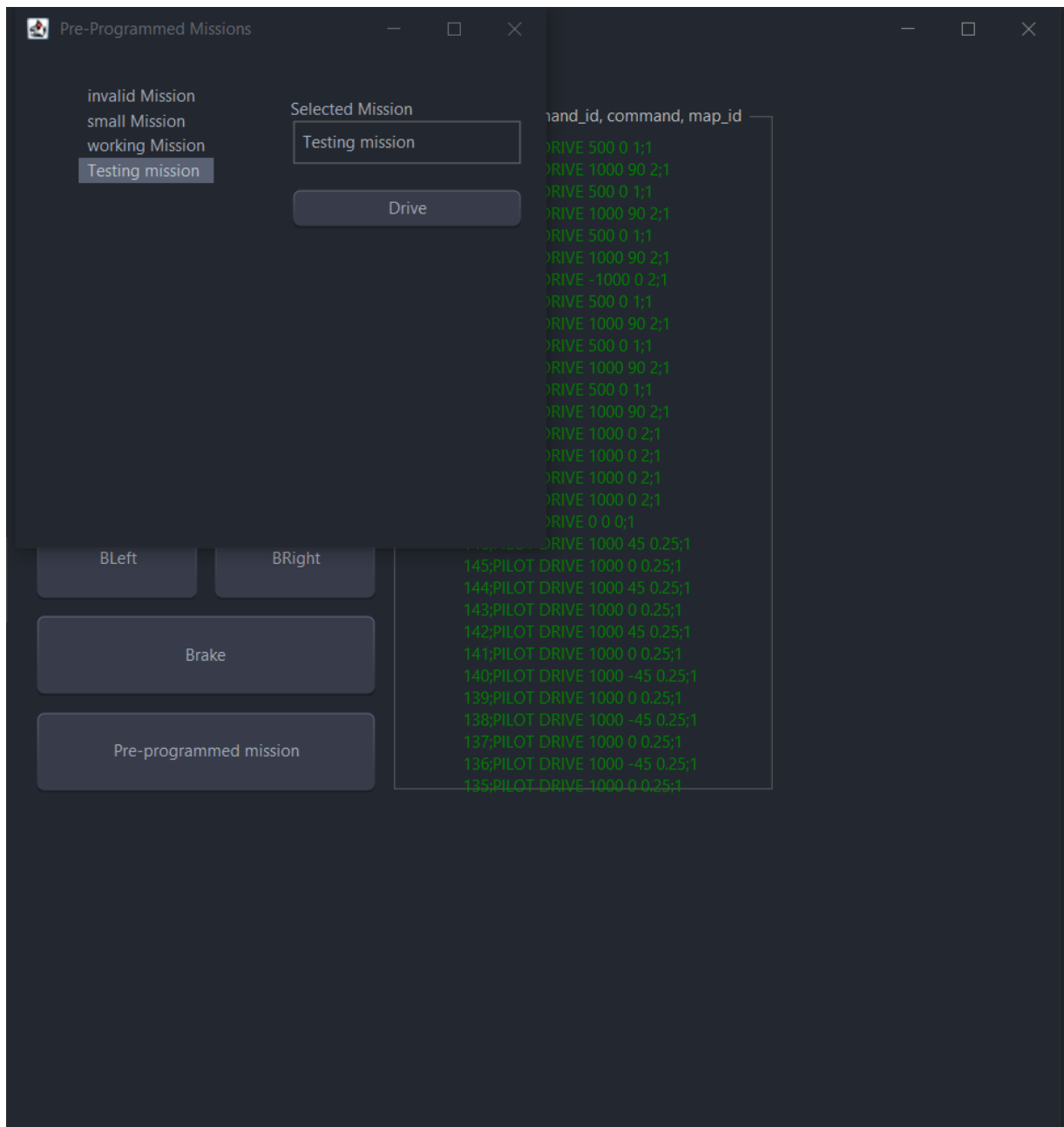


F-GC-S-01

The following test will be carried out to test the following requirement. The following steps are taken during the test. First, both the pilot and ground control UI is launched. Second, through the ground control UI mission is created alongside commands. As commands were being added it was also tested to see if any validation is performed, in this case, it is seen it was being performed. Lastly, the mission is saved and checked if it will appear on the pre-programmed mission in the pilot UI. It is seen the created mission is sent to the pilot database and stored. In which the requirement functions as intended as the test concludes as a pass.

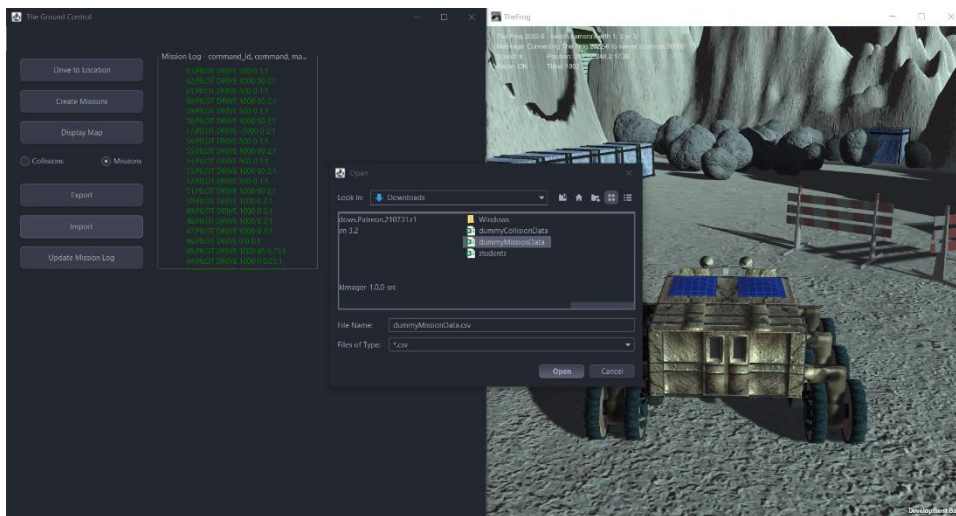
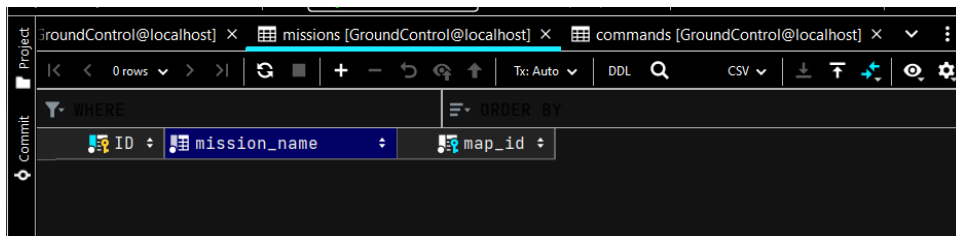






F-GC-S-03, F-GC-S-04, F-GC-S-05, F-GC-S-06

The following test will be carried out to test the following requirement. The following actions are taken during the testing. Before initial testing it is ensured that the database contains no data. First, importing mission & importing collision point functions is tested. To test these functions dummy data is created. Second, missions are selected in the Ground Control UI and then the import button is clicked. Third, file containing dummy mission data is selected and imported. In which the database is checked and compared with the file to ensure correct data is imported. And lastly, the export buttons are clicked to test the export function. Once the file has been exported, data is compared with the initial file and with the database.



Testing result: The importing and exporting of missions and collision points are tested using two .csv dummy file: dummyCollisionData.csv (containing dummy coordinates) and dummyMissionData.csv (containing dummy missions). The original testing state of the database is empty (tables “coordinates”, “missions”, “commands”).

```
mission;command
test;drive 100 10 10
test;drive 100 10 10
test;drive 100 10 10
test;drive 100 10 10
test;drive 100 10 10
test;drive 100 10 10
test2;drive 100 10 10
```

^ Collapse 1 KB

```
x;y;z
23;43;54
67;67;78
```

1 KB

First, we import the missions and the coordinates using the Ground Control UI. Then we query the database using PgAdmin4: the tables indeed contain the expected information

The “coordinates” table indeed contains 2 entries shown in the dummy csv

	id [PK] bigint	type coordinate_type	x double precision	y double precision	z double precision	map_id bigint
1	3562	collision		23	43	54
2	3563	collision		67	67	78

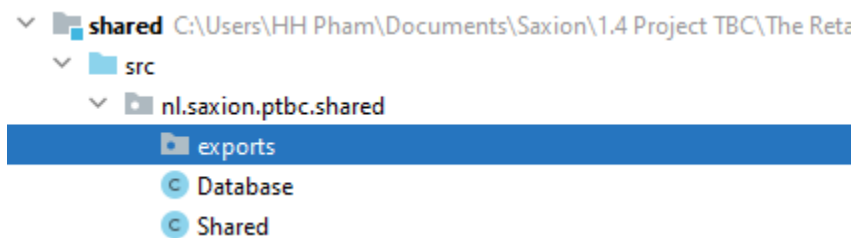
The “missions” table indeed contains 2 missions that can be found in the dummy csv: “test” and “test2”

	ID [PK] bigint	mission_name text	map_id bigint
1	3	test	1
2	4	test2	1

And finally, the “commands” table indeed contains all commands that can be found in the dummy csv:

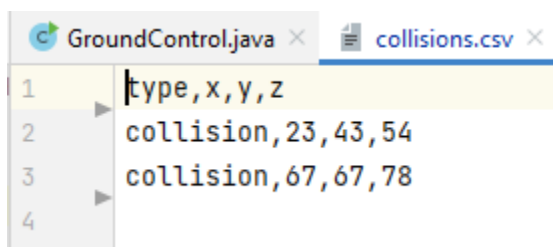
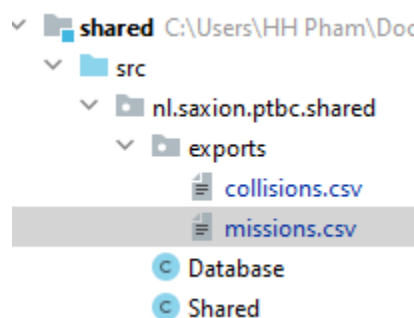
	id [PK] bigint	command text	mission_id bigint
1	8	drive 100 10 10	3
2	9	drive 100 10 10	3
3	10	drive 100 10 10	3
4	11	drive 100 10 10	3
5	12	drive 100 10 10	3
6	13	drive 100 10 10	3
7	14	drive 100 10 10	4

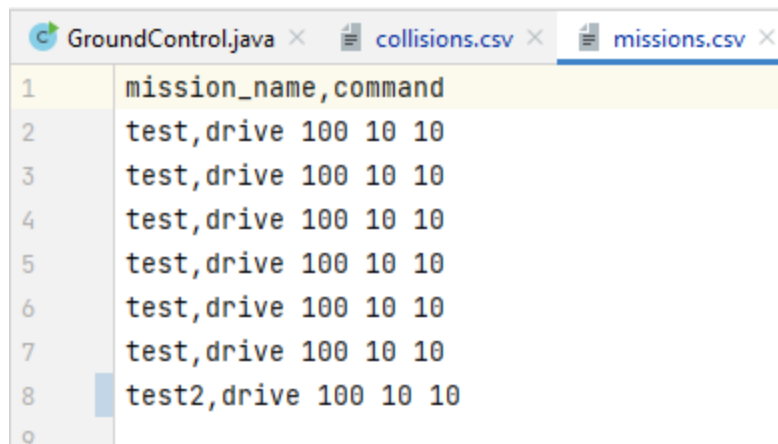
Now we try to export the collision points and the missions that is currently exist database. For testing purposes, the existing database is the same as was imported earlier.



CSV files to be exported to has the destination of the “exports” folder in the “shared” folder of the project repository. Currently the “exports” folder is empty.

Now we click the export button on the Ground Control UI to export the missions and the collision points. 2 new CSV files are generated, and indeed contain the same data as in the database:





```
1 mission_name,command
2 test,drive 100 10 10
3 test,drive 100 10 10
4 test,drive 100 10 10
5 test,drive 100 10 10
6 test,drive 100 10 10
7 test,drive 100 10 10
8 test2,drive 100 10 10
9
```

These function works as intended and was demonstrated acceptable.