Sprint 1 Report

Main objectives of the Sprint

For the first sprint our main objective was to understand the task we were presented and design a solution for the client. In addition, our secondary objective was to familiarize ourselves with the environment in which we must develop our solution, as in the Sasa communication library, Swing and JDBC and understand how everything works together. Another of our main objectives was also to develop the framework within we will be working on. We did this by organizing our git repository and our OneDrive folder in which we will be sharing code and documents and by developing the database structure in PostgreSQL and connecting it to our code. For our final goal of this sprint, we decided that we want to start working on some of the simpler requirements like the manual control system for the Frog, creating preprogrammed missions in Ground Control and executing those missions in the Pilot.

Design decisions explanation

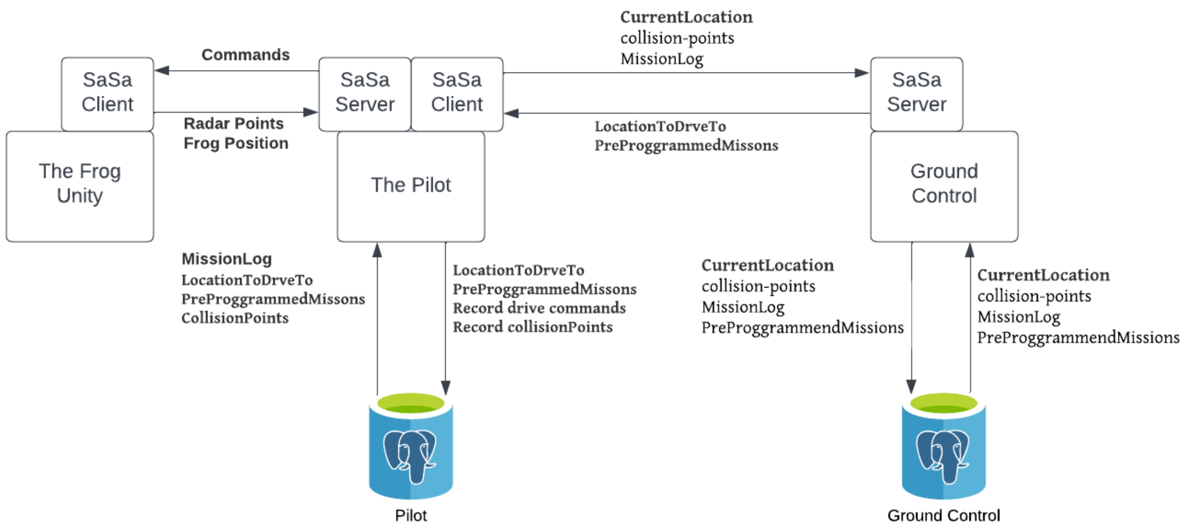
Because we didn't had an opportunity to speak to the client during this sprint we made a lot of decisions both with regards to the UI and how the end user will be using the system and with regards to the dataflow and how the system will function.

To start this off we had a lot of discussions when it came down to the manual control of the Pilot and how the user was supposed to control the Frog. Initially ideas like using keyboard or buttons and sliders on the screen were floating around but at the end we decided that it will be for the best to create a simple text box with a send button so the user can simply send commands directly to the rover. The reason why we choose this design is because of the nature of the commends itself, by pressing a button u can execute a command but there is no way of saying for how long for example the rover should move. In the future I we think of better solution we might implement it for more details consult the Use case description UC1.

When developing the preprogrammed missions we thought that in order to cover the requirements for those in both applications it would be for the best if a user creates a mission in the ground control where he/she will be able to manually add commands to the frog and from there the system will save those missions in the local DB and in the same time send this mission to the Pilot application from which a user can select this mission to be executed. We described these processes in more detail in UC2 and UC7.

For the dataflow between the rover itself, the Pilot and the Ground control with regards to the Sasa library we decided that the rover is our client, and he accepts commands and sends responses to the Pilot application which will be a Sasa server and then the Pilot will send response the Ground control server and the Pilot is going to functions as a client. It is a bit complicated this way and we could have made the pilot only a server and the ground control another Sasa client, but we stopped at this solution since for us it is the most logical from Ground control perspective. In Figure 1.1 you can see more clearly how the whole communication between servers and clients will be happening.

Figure1.1



For the mission log we decided that the pilot will be saving each executed command on its DB and once the user chooses to send that mission log to the Ground control the commands will be sent, and the database will be cleared. This is to prevent submitting the same data again and again. We could have made it so that both DBs have a complete mission log not just the Ground Control but for us this made the most sense.

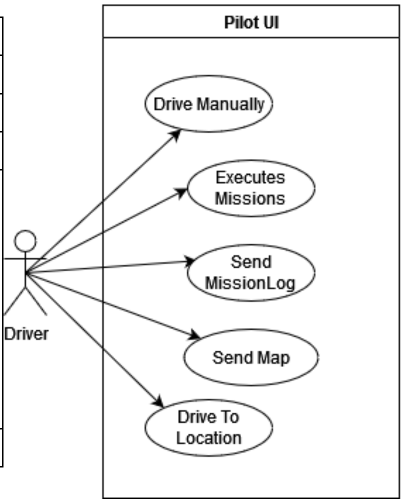
For the Autopilot drive to location, we decided that it won’t be necessary to save the coordinates locally on the Ground Control UC6 but directly send them to the Pilot DB and from there the Pilot user will have a list of locations UC5 it could command the rover to go automatically.

For the map we decided that the Ground Control will have to manually request updates, and, on the map, it will be displayed the collision points, the path of the missions and the current position of the rover UC8. In the Pilot map the user will only be able to see the collision points UC3. Further down the line we might add the rover location and path of missions.

Use case Diagrams

1. Pilot

Title	Drive Manually
ID	UC1
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	1. User selects the drive manually option 2. A textbox with a button is presented 3. User enters a command and presses the submit button 4. Rover executes the command 4.1. If rover radar has detected a collision point the Frog will avoid it or stop
Postcondition	Rover has executed a command



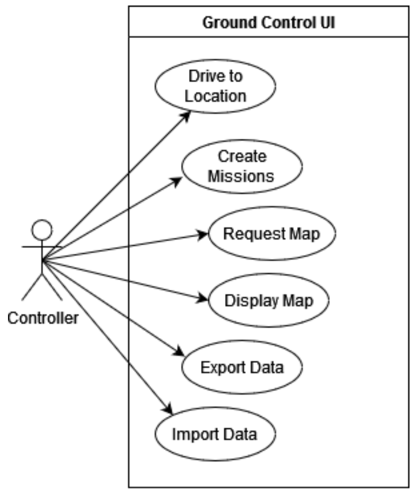
Title	Execute Missions
ID	UC2
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	1. User selects the executable missions option 2. User is presented with a drop-down menu and a button 3. User selects one of the uploaded missions from the Ground Control 4. User presses the submit button 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 map
ID	UC3
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	1. User selects the send map option 2. User is presented with a white board where all the collision points seen so far have been marked with red and a send button 3. User selects the send button 4. The same white board is sent to the user
Postcondition	Rover sends a map of collision points it has detected so far

Title	Send Mission Log
ID	UC4
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	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	Drive to location
ID	UC5
Actor	Driver
Precondition	Driver has control of the Frog UI
Flow	1. User selects Drive to location option 2. Rover retrieves the most recent coordinates Ground Control has send to the Frog 3. Rover automatically starts executing commands to reach the destination 3.1. Rover avoids collisions on every command
Postcondition	Rover automatically drives to its destination

2. Ground Control UI



Title	Drive to location
ID	UC6
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	1. User selects give destination option 2. User is presented with 3 text columns and a send button 2.1. Each text column is mark (X,Y,Z) 3. User filles 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	1. User selects Create a mission option 2. User is presented with 2 text field and 2 button and a drop-down menu 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 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

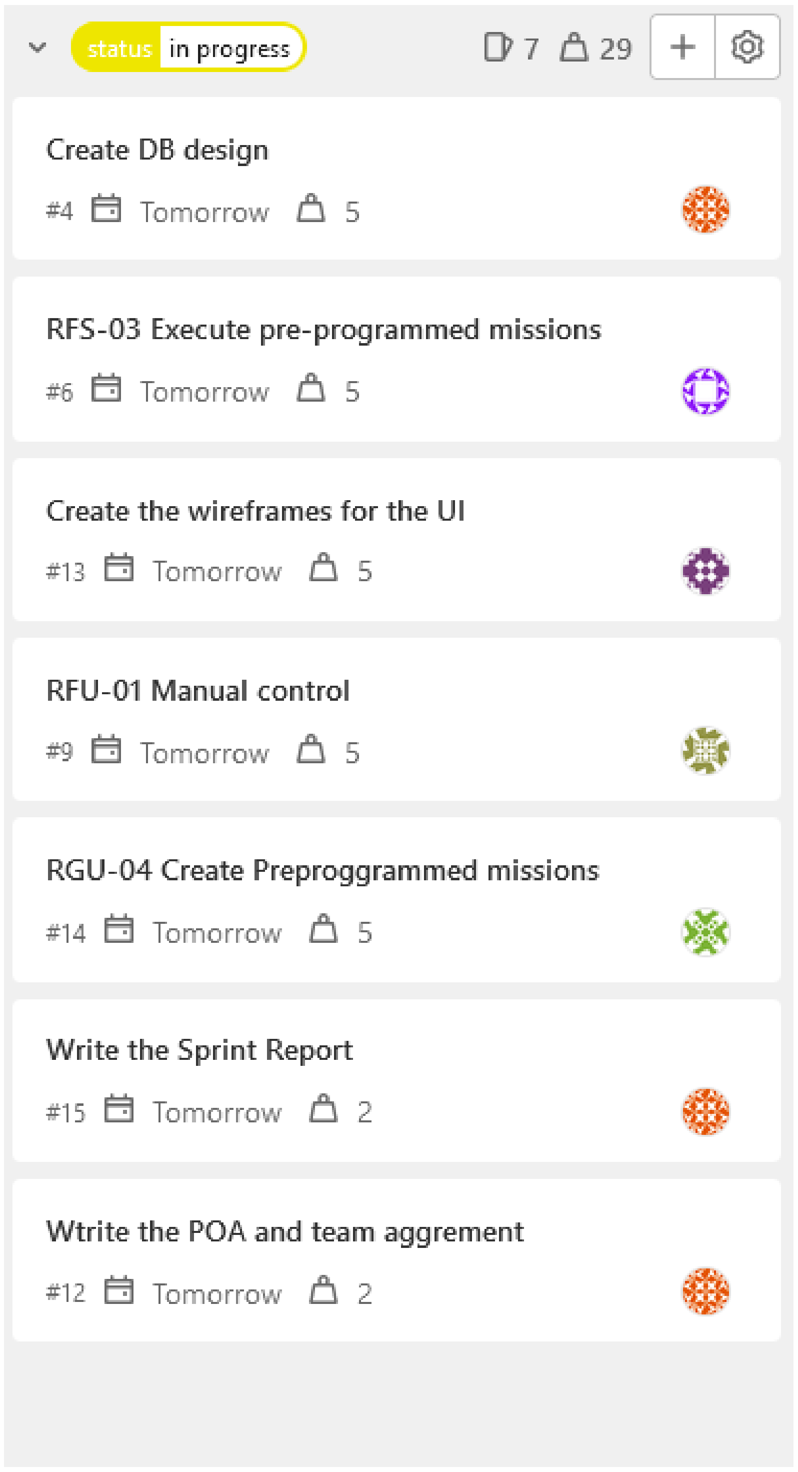
Title	Request/Display Map
ID	UC8
Actor	Operator
Precondition	Operator has control of the Frog UI
Flow	1. Operator selects map option 2. User is presented with the lastly requested map 2.1. The map is a white board with red dots marking the collision points 2.2. The current position of the rover is also 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	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	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 5.1. If the path is wrong user is presented with a message
Postcondition	Operator imports required data

Sprint Backlog

For more detail on the individual stories, you can consult our Gitlab repository.



status in progress 7 29

Create DB design
#4 Tomorrow 5

RFS-03 Execute pre-programmed missions
#6 Tomorrow 5

Create the wireframes for the UI
#13 Tomorrow 5

RFU-01 Manual control
#9 Tomorrow 5

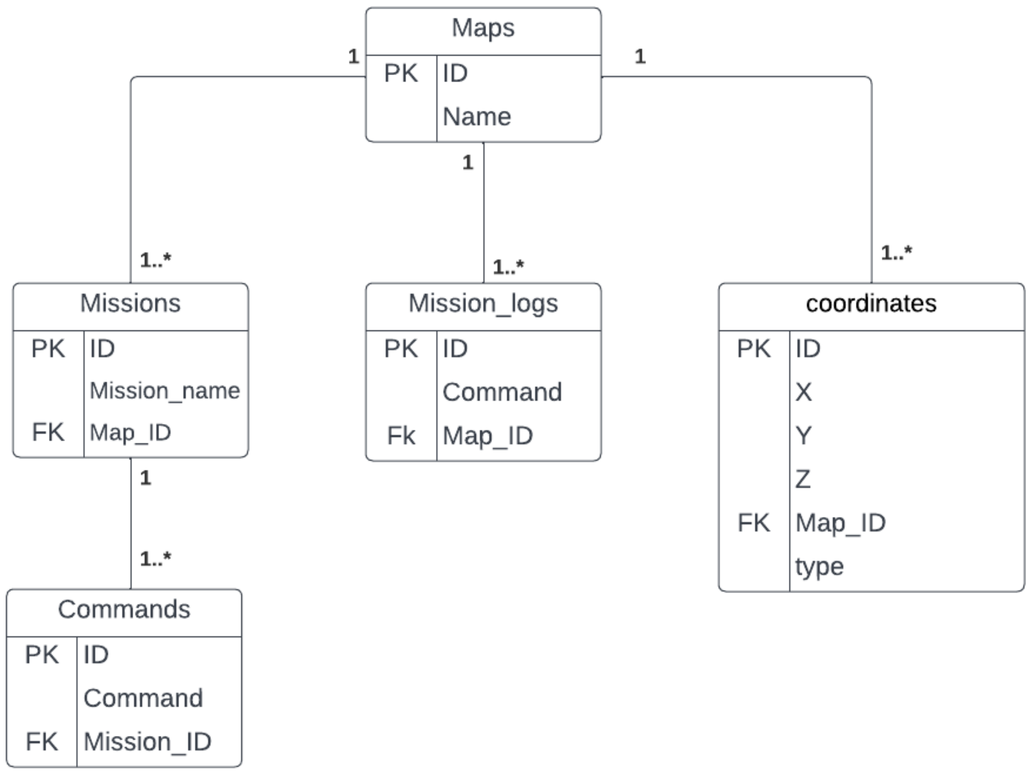
RGU-04 Create Preproggrammed missions
#14 Tomorrow 5

Write the Sprint Report
#15 Tomorrow 2

Wtrite the POA and team aggrement
#12 Tomorrow 2

Database design

The diagram for both DB is almost identical with one exemption. The "coordinate_type" in "Ground Control" DB has only 2 types: "collision" which is for the collision points, "location" which is for the current location of the rover, while Pilot has one more type called "endpoint", and this is for the location that Ground Control will send to the pilot and the rover will have to automatically go to that location. Coordinate type is an Enum so technically speaking both databases can have all 3 types, but we should not store the "location" type in the Ground Control DB. The reason why we design it this way is to store data more efficiently since for all 3 types of coordinates we will have the same data (X, Y, Z integers). As far as why the diagrams are identical, well, we put a lot of thought into this, and this is the only way we see to design the DB's and meet the requirements. The structures might look the same but the information they will contain will be different.



Demo

The Demo can be found in our group repository on the main branch you can view it here.

Sprint Retrospective

Veci Hodja

Q: What went well?

A: Tasks where able to split up per person easily.

Q: What went not so well?

A: Confusion from lack of proper documentation.

Q: What can we improve?

A: Communication between teammates.

Viviana Radu

Q: What went well?

A: Team members being available for offering help despite short notice.

Q: What went not so well?

A: Confusion about project requirements.

Q: What can we improve?

A: Increase frequency of whole group meetings (online and offline).

Sebastian van der Westhuizen

Q: What went well?

A: All group members were present and understood the main goal of what would be talked about before the meeting.

Q: What went not so well?

A: Finished products were not presentable by every group member, this not being a large issue but creating confusion about where members of the group are when it comes to the entire project. Understanding what everyone has done and is doing gives a bigger picture and allows members to compensate for one another. Like a big ant hive.

Q: What can we improve?

A: Work progression of specific tasks can be updated per project.

Omar Zaarour El Keddeh

Q: What went well?

A: Conducting weekly meetings.

Q: What went not so well?

A: Confusion about the requirements.

Q: What can we improve?

A: Talk more with one another.

Yani Yanev

Q: What went well?

A: We were able to formulate some plan of how to develop the applications.

Q: What went not so well?

A: Confusion surrounding what is required of us is the big one but also some group mates not being able to complete their tasks on time.

Q: What can we improve?

A: More communication between groupmate and helping each other with our tasks.