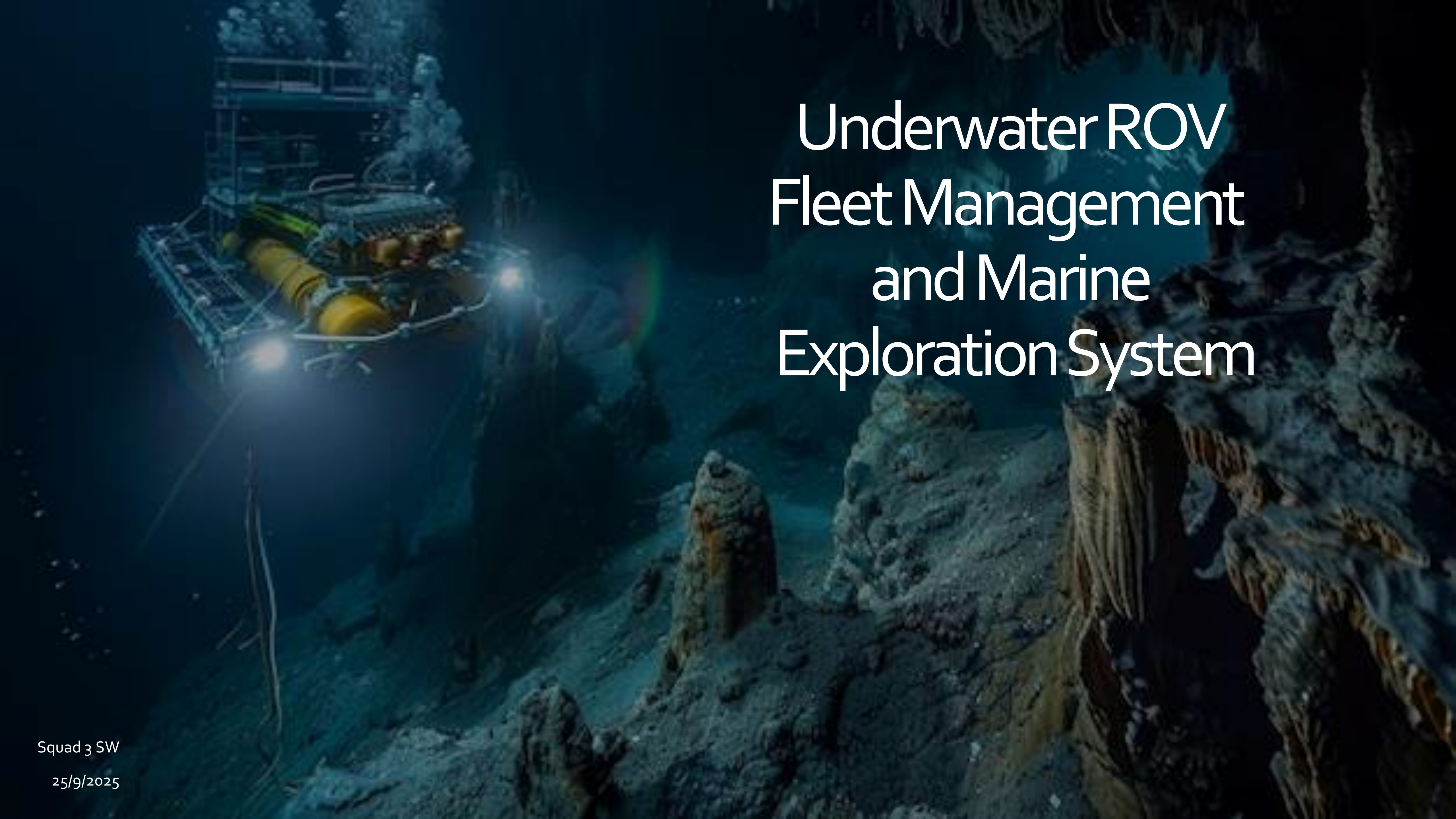


# Underwater ROV Fleet Management and Marine Exploration System

Squad 3 SW

25/9/2025

# Project Overview

We “Squad 3 Software Team” excited to introduce our latest project: **the Marine Mission Control System (MMCS).**

This innovative simulation is designed to manage a fleet of Remotely Operated Vehicles (ROVs) for various underwater tasks, including exploration, data collection, and environmental monitoring.

Our goal is to create an efficient and user-friendly CLI-based simulation that allows users to coordinate multiple types of ROVs. By leveraging object-oriented programming (OOP) and concurrency, we aim to create a robust system that can handle real-time mission execution.

# introduction

**in recent years**, the importance of marine research has grown tremendously, and ROVs play a crucial role in tasks such as exploration, data collection, and environmental monitoring. Our MMCS serves as a central hub that simulates the operations of a fleet of ROVs, allowing users to experience the complexities of underwater missions firsthand.

Throughout our presentation, we will explore how our system is designed to effectively manage multiple ROVs, showcasing their distinct capabilities and behaviors. We'll demonstrate how our command-line interface enables user interaction, allowing for a dynamic and immersive simulation experience.

Moreover, we will highlight the technical challenges we faced—such as ensuring OOP and concurrency—while also emphasizing how these challenges led to a robust and scalable solution.

Join us as we dive deeper into the features and functionalities of MMCS, illustrating not just the technology behind it, but also its potential applications in marine exploration. Let's embark on this journey together!

# The Project

# Code Structure

- ## Import:

The code begins by importing necessary libraries, including ABC for creating abstract base classes, threading for concurrency, random and time for simulating delays and randomness, and datetime for timestamping.

- ## Classes:

The code includes 5 classes as follow:

Base Class

- **(ROV Class)**
  It's an abstract base class that defines the common attributes and methods for all types of ROVs.

ROV Subclasses

- **(ExplorationROV)**:
  Inherits from ROV and implements the execute_mission() method to handle exploration missions.
- **(SamplingROV)**:
  Inherits from ROV and implements the execute_mission() method for sampling tasks.
- **(MaintenanceROV)**:
  Inherits from ROV and implements the execute_mission() method for maintenance activities

Main control class

- **(MMCS)**
  The MMCS class manages the fleet of ROVs and oversees mission assignments and execution.

- ## Main Function:

The main() function serves as the entry point for the program, handling user input and orchestrating the initialization and simulation of the MMCS.

# Main Function

At the beginning a welcomed message will be appear to the user!
The program starts executing from the main() function and the user will be asked to enter number of ROVs, number of missions and type of each ROV.
After that by calling "MMCS Class" it will Initializes the MMCS instance with the specified number of ROVs and calls initialize_rov_fleet() to create the ROV instances based on user input
,where this class is responsible for managing ROVs , assigning mission to them .

the __init__ method of the MMCS class initializes key attributes to manage a fleet of ROVs and their missions. It sets up lists to store ROV instances and missions, tracks the total number of ROVs, initializes a mission ID counter, counts unexecuted missions for each ROV, and prepares to track the total required missions. Finally, it calls a method to populate the fleet with ROV instances based on user input, establishing the foundation for the simulation's functionality.

# Main control class

**(MMCS)**

Depending on user input type of each ROV, it creates instances of the appropriate ROV subclass and stores them in the self.rovs list using initialize_rov_fleet method

Then generate_mission(mission_type) method creates a mission dictionary with attributes like mission ID, type, target location, and status.
Then by calling assign_missions(num_missions) method from the main ()function randomly these method selects ROVs and assigns missions random to them based on their type.

After that,
By calling start_simulation() method from the main ()function initiates the simulation by creating a thread for each ROV.
Each thread runs concurrently, allowing multiple ROVs to operate simultaneously.

At the end,
terminate_simulation() method is being called from the main ()function to print a summary of the results.
It calls the display_summary() method to present the final status of each ROV and the total number of missions completed.

# Base Class

This class defines common attributes and methods that all ROV subclasses (like ExplorationROV, SamplingROV, and MaintenanceROV) will inherit by using Constructor **(__init__ method)** it initializes common attributes that will be shared among all ROV subclasses

When a subclass is instantiated, it calls the ROV constructor using e.g:**super().__init__(rov_id, 'Exploration').** This ensures that all the common attributes (like rov_id, rov_type, status, battery_level, etc.) are initialized correctly for the specific type of ROV.

# SubClasses

e.g :(ExplorationROV)

The ExplorationROV class defines a specialized type of ROV that can receive and execute exploration missions, handling navigation, battery consumption, and status updates throughout the mission lifecycle.

super().__init__(rov_id, 'Exploration'): Calls the initializer of the parent ROV class, passing the rov_id and setting the rov_type to 'Exploration'. This initializes the common attributes (like status, battery_level, etc.) defined in the ROV class.

execute_mission(self) method defines how the ExplorationROV carries out its missions.

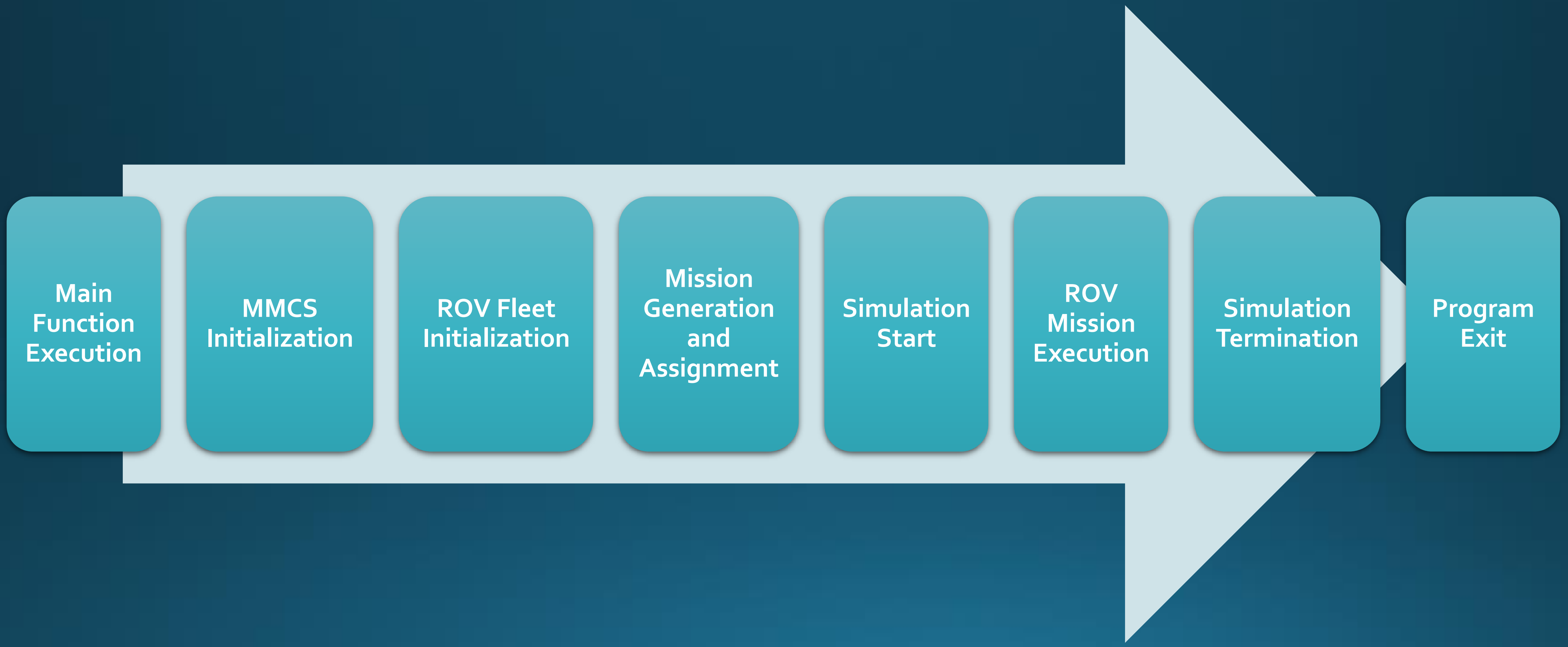
Calls the self.navigate_to method to navigate to the mission's target location, which is retrieved from the mission dictionary.

Calling send_status_update() method each time we need to update status to reflect the new one.

Calling self.check_battery() method to Checks the battery level at the end of each completed mission and updates the status if it's low (35% or less).

The same for (SamplingROV)(MaintenanceROV)

# How the code flow?



# output

- at the beginning the user asked to enter number of ROVs and number of missions as shown:

```
Welcome to the Underwater ROV Fleet Management Simulator!
Please enter the number of ROVs: 3
Please enter the number of missions to complete: 5
Enter type for ROV 1 (1 - Exploration, 2 - Sampling, 3 - Maintenance): 2
Enter type for ROV 2 (1 - Exploration, 2 - Sampling, 3 - Maintenance): 3
Enter type for ROV 3 (1 - Exploration, 2 - Sampling, 3 - Maintenance): 1
ROV ROV_3 received mission 1001
[21:42:48] ROV_3 assigned Mission 1001: Exploration at (28, 4)
ROV ROV_2 received mission 1002
[21:42:48] ROV_2 assigned Mission 1002: Maintenance at (2, 21)
ROV ROV_3 received mission 1003
[21:42:48] ROV_3 assigned Mission 1003: Exploration at (97, 88)
ROV ROV_1 received mission 1004
[21:42:48] ROV_1 assigned Mission 1004: Sampling at (2, 56)
ROV ROV_1 received mission 1005
[21:42:48] ROV_1 assigned Mission 1005: Sampling at (40, 44)
[Status] ROV ROV_1: In Mission, Battery: 100%
ROV ROV_1 is navigating to (2, 56)...
[Status] ROV ROV_2: In Mission, Battery: 100%
[Status] ROV ROV_3: In Mission, Battery: 100%
```

Before starting the simulation ,Battery for each ROV is Fully charged

```
ROV ROV_2 is navigating to (2, 21)...
ROV ROV_3 is navigating to (28, 4)...
ROV ROV_2 arrived at (2, 21)
ROV ROV_2 is performing maintenance at (2, 21)...
ROV ROV_1 arrived at (2, 56)
ROV ROV_1 is collecting sample at (2, 56)...
ROV ROV_3 arrived at (28, 4)
ROV ROV_3 is mapping the area at (28, 4)...
ROV ROV_2 completed maintenance at (2, 21)
[Status] ROV ROV_2: Idle, Battery: 76%
ROV ROV_1 completed collecting sample at (2, 56)
[Status] ROV ROV_1: Idle, Battery: 78%
[Status] ROV ROV_1: In Mission, Battery: 78%
ROV ROV_1 is navigating to (40, 44)...
ROV ROV_3 completed mapping at (28, 4)
[Status] ROV ROV_3: Idle, Battery: 83%
ROV ROV_1 arrived at (40, 44)
ROV ROV_1 is collecting sample at (40, 44)...
[Status] ROV ROV_3: In Mission, Battery: 83%
ROV ROV_3 is navigating to (97, 88)...
ROV ROV_3 arrived at (97, 88)
ROV ROV_3 is mapping the area at (97, 88)...
ROV ROV_1 completed collecting sample at (40, 44)
[Status] ROV ROV_1: Idle, Battery: 65%
ROV ROV_3 completed mapping at (97, 88)
[Status] ROV ROV_3: Idle, Battery: 70%
[Simulation Ended]

Final Summary:
- ROV_1 Status: Idle, Battery Level: 65%
- ROV_2 Status: Idle, Battery Level: 76%
- ROV_3 Status: Idle, Battery Level: 70%
- Total Missions Completed: 5
```

- During the simulation, the status of each ROV is continuously updated to inform the user about the current mission being executed, its target location, battery level, and also whether the mission has been accomplished

- At the end we inform the user with the final battery level for each ROV

# output

EXCEPTION CASE

If the battery of any ROV is ≤35%, the user will receive a warning.
If it falls to ≤29%, the ROV will stop operating. The user will be informed that the ROV cannot continue, and the final summary will show the number of missions that have not been executed as shown :

```
ROV ROV_1 is running low on battery. Current level: 34%
[Status] ROV ROV_1: In Mission, Battery: 34%
ROV ROV_1 is navigating to (3, 85)...
ROV ROV_1 arrived at (3, 85)
ROV ROV_1 is mapping the area at (3, 85)...
ROV ROV_2 completed maintenance at (2, 74)
[Status] ROV ROV_2: Idle, Battery: 16%
ROV ROV_2 is running low on battery. Current level: 16%
ROV ROV_2 cannot continue. Battery is too low.
ROV ROV_1 completed mapping at (3, 85)
[Status] ROV ROV_1: Idle, Battery: 11%
ROV ROV_1 is running low on battery. Current level: 11%
ROV ROV_1 cannot continue. Battery is too low.
[Simulation Ended]

Final Summary:
- ROV_1 Status: Low Battery, Battery Level: 11%, Unexecuted Missions: 1
- ROV_2 Status: Low Battery, Battery Level: 16%, Unexecuted Missions: 2
- Total Missions Required: 11
- Total Missions Completed: 8
```

# Task division

**Mazen el deep** : Base class (ROV)
**Asmaa abo shady** : main class (MMCS)
**perihane Hossam** : subclass(ExplorationROV) & presentation
**kenzy Mohamed** : subclass (SamplingROV)
**Youssef Mohamed** : subclass (MaintenanceROV) & documentation

# Challenges

**Using threading**

made it difficult to ensure that shared resources (like mission updates) were accessed safely.

**Using inheritance**

Facing a difficulty with using Inheritance through out the code

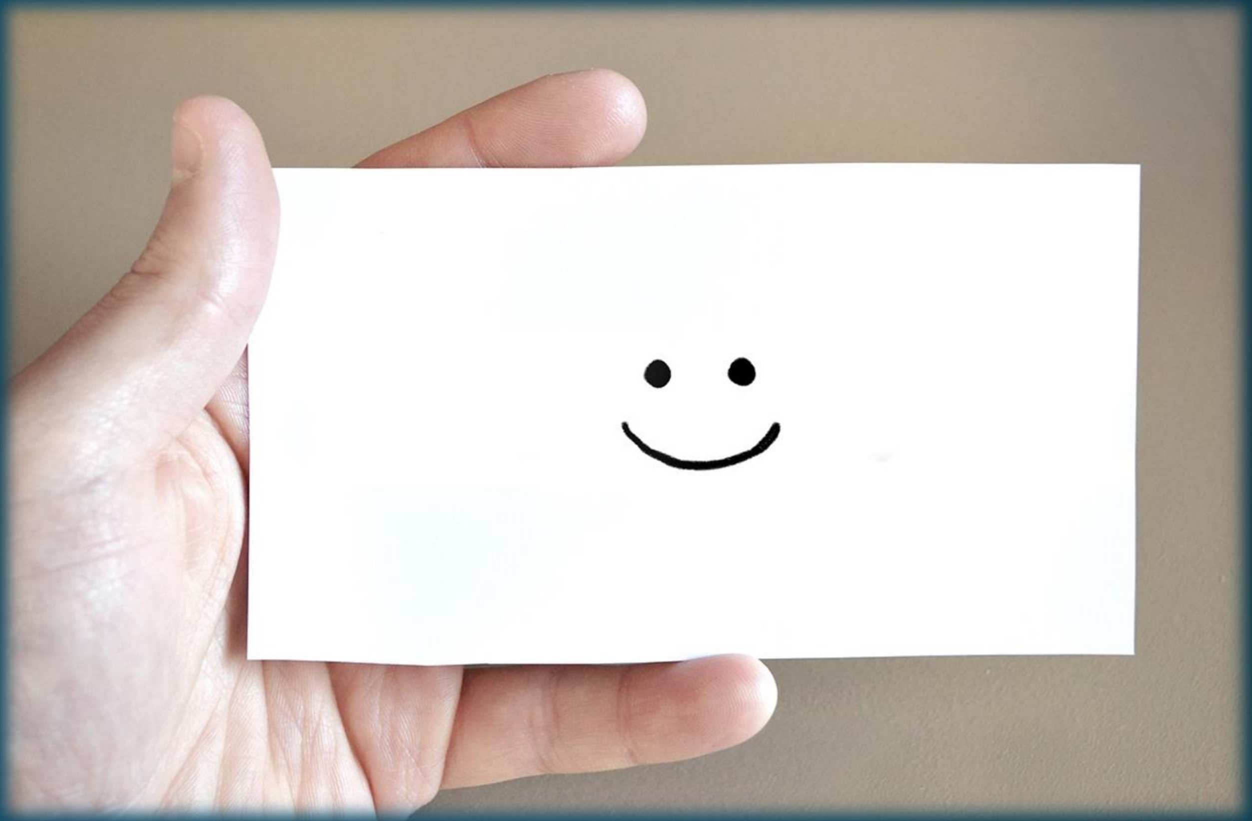
**Debugging problem**

multithreaded applications is challenging because errors can be inconsistent, making it tough to pinpoint issues.

**Time management**

Coordinating schedules and managing deadlines between us was challenging.

# Lesson learned



**Understanding thread**

Importance of understanding how to manage Thread Safety

**Importance of Clear Base Classes**

Establishing a well-defined base class and uniform interfaces for subclasses enhances code to performs readability and maintainability.

**Time management**

How to manage time among team members and Ensuring that everyone is aligned on deadlines and responsibilities.

**Important of comments**

Working on a code project within a team must consider using a readable and understandable comments for teach important key features