Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Number	COE892
Course Title	Distributed Cloud Computing
Semester/Year	Winter 2025
Instructor	Khalid Abdel Hafeez
TA name	Amirhossein Razavi

Lab No.	3

Section No.	05
Submission Date	March 9th, 2025
Due Date	March 9th, 2025

Student Name	**Student ID**	**Signature***
Maria Labeeba	500960386	M.L

Student Name Student ID Signature*

*By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct

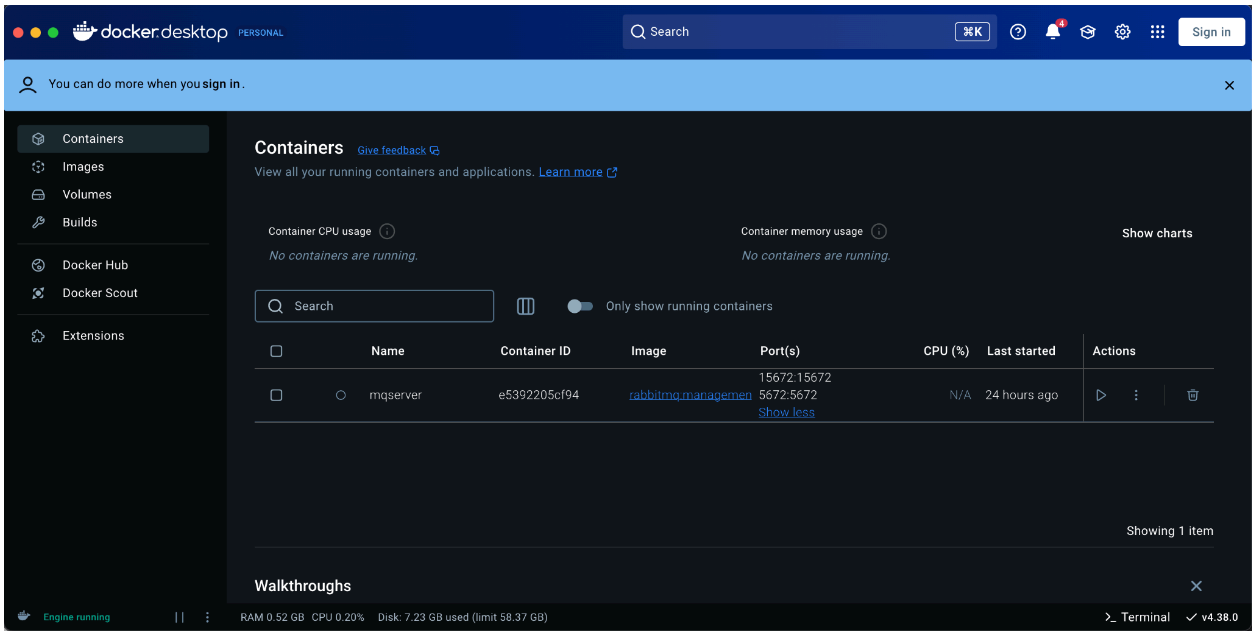
1. Introduction

This lab is a continuation of Lab 2 where we were required to write a gRPC client server program to allow rovers to detect and dispose of mines. The main objective of this lab is to use gRPC and RabbitMQ to allow communication between the clients (rover and de-miner) and the server (ground control).

The ground control relies on 3 files, the miners' serial number, the 2D map array, and the rovers commands list. The rovers are intended to explore the map and send a request to the gRPC server when a mine is found in order to retrieve the mine serial number. It will then publish a demining task using RabbitMQ, which uses the queues to get coordinates of found mines, their ID's as well as their serial numbers. The deminers will start disarming the discovered mines and will publish the pin of the mine over another RabbitMQ channel once the di-miner has finished.

2. Part 1: RabbitMQ Server

Before developing the program, the RabbitMQ server was first set up. This was done by downloading and installing the RabbitMQ server and clients from the given lab manual, as well as installing docker to launch the server. The following *Figure 1* is a screenshot displaying the Docker container named *mqserver* which was created from the rabbitmq image and maps the port 5672 of the machine to the port 5672 of the container.



You can do more when you sign in.

Containers Give feedback

View all your running containers and applications. Learn more

Container CPU usage: No containers are running.

Container memory usage: No containers are running.

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
	mqserver	e5392205cf94	rabbitmq:managemen	15672:15672 5672:5672 Show less	N/A	24 hours ago	

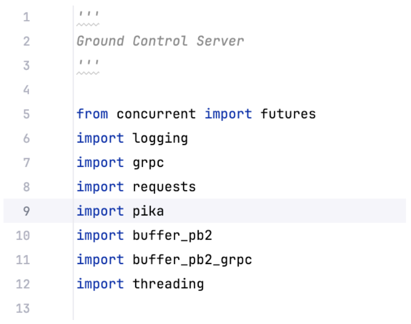
Showing 1 item

Engine running RAM 0.52 GB CPU 0.20% Disk: 7.23 GB used (limit 58.37 GB) v4.38.0

Figure 1: Docker container named *mqserver*

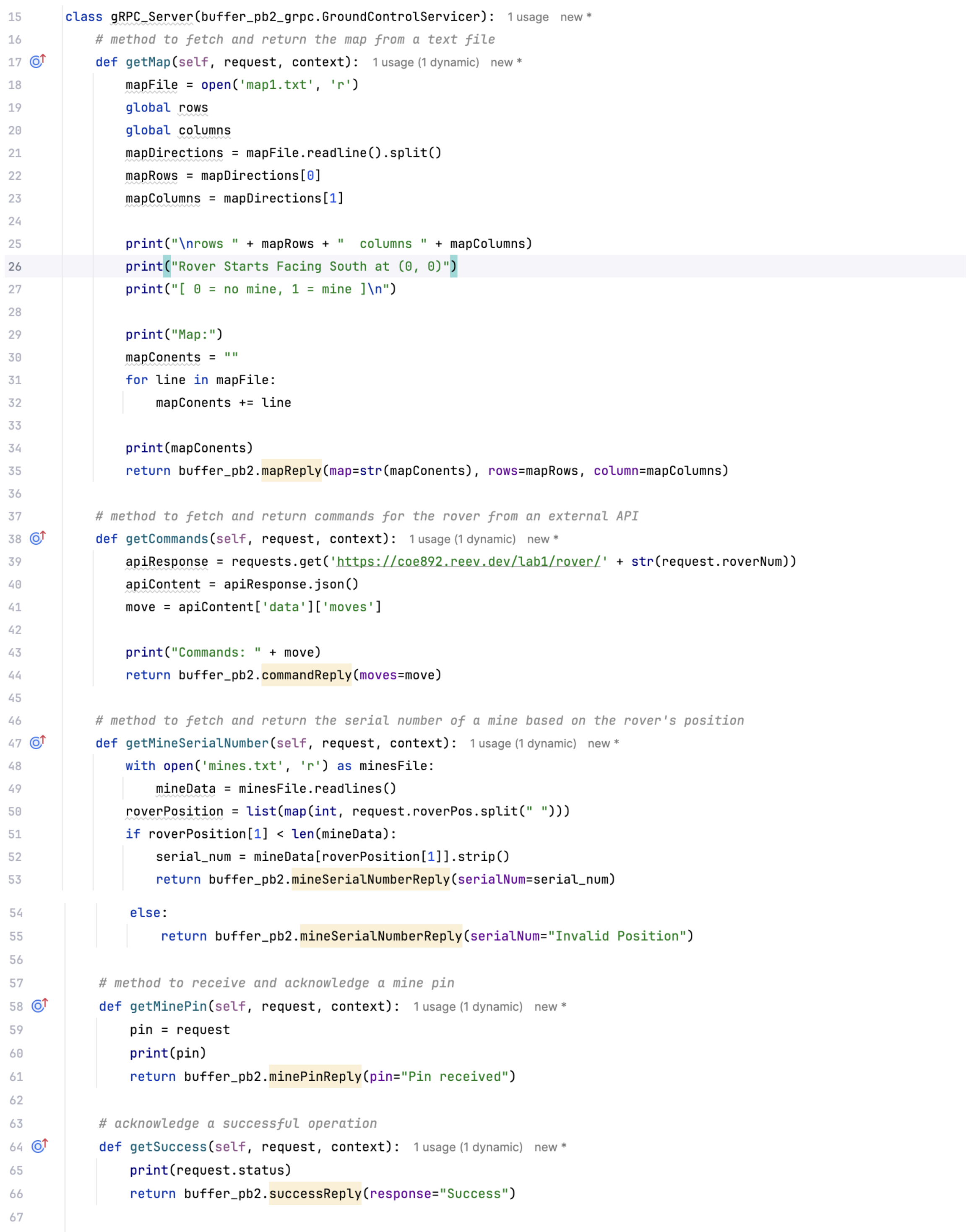
3. Part 2: Ground Control Server

Next the ground control server was created and the code can be seen below from *Figure 2 to Figure 4*. The server uses the 2D map files used in lab 1 and lab 2, the rover command files containing the commands given to each rover client as well as the mine serial number file used in lab 1 and lab 2. One key difference in the rover commands used for this lab is that the Dig, ‘D’ command will not be used since it will be replaced with the de-mining client. The server was created based on the .proto files which were created in part 1 of lab 2 (see Appendix A). The methods that were specifically used from lab 2 include; get map, get command stream and get mine serial number which can be seen in *Figure 3*. The server also subscribes the RabbitMQ server to the Diffused Mines Channel seen in *Figure 4* and then logs it to a file titled ‘defused_mines.log’.



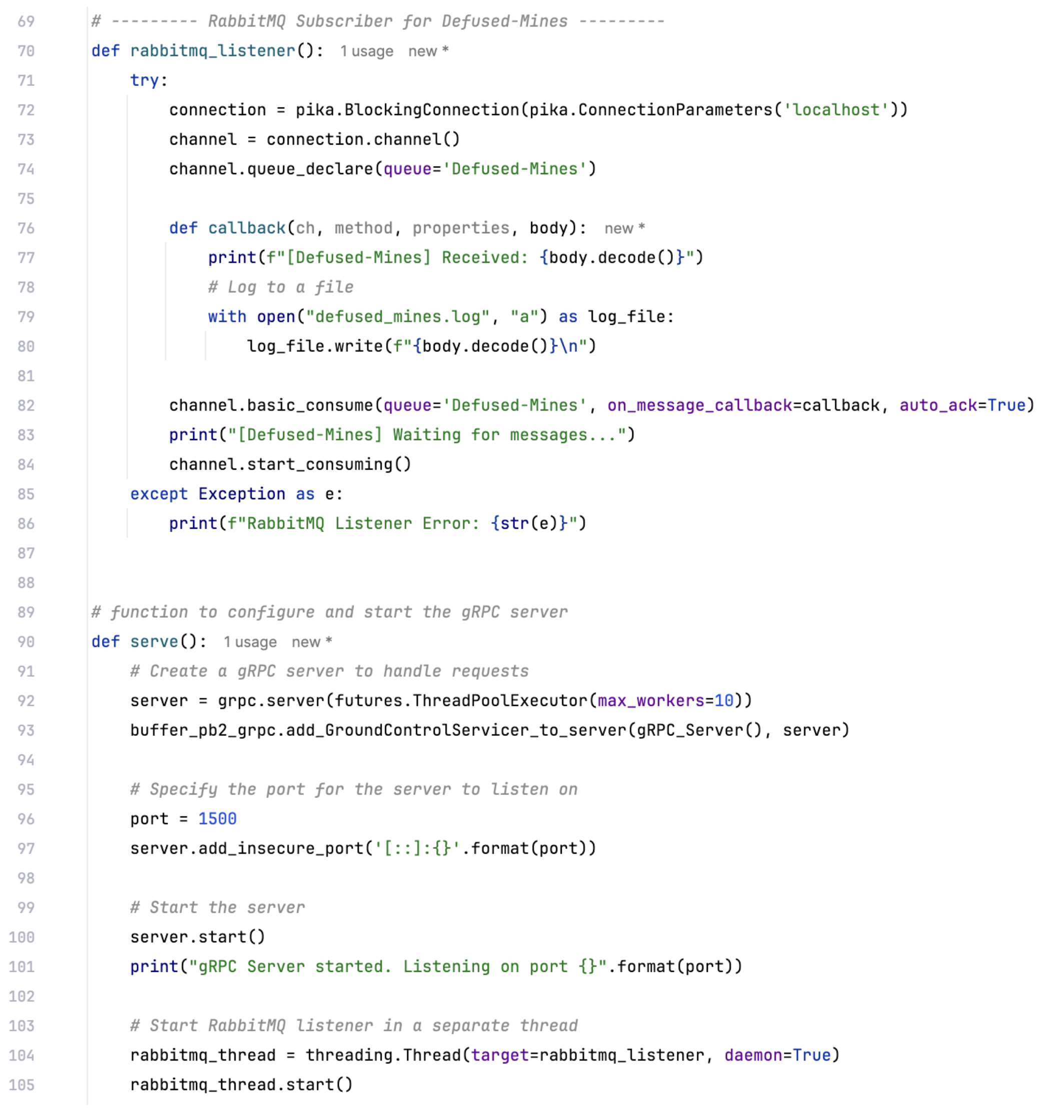
```python
'''
Ground Control Server
'''

from concurrent import futures
import logging
import grpc
import requests
import pika
import buffer_pb2
import buffer_pb2_grpc
import threading

```

Figure 2: *Imports for Ground Control Server*

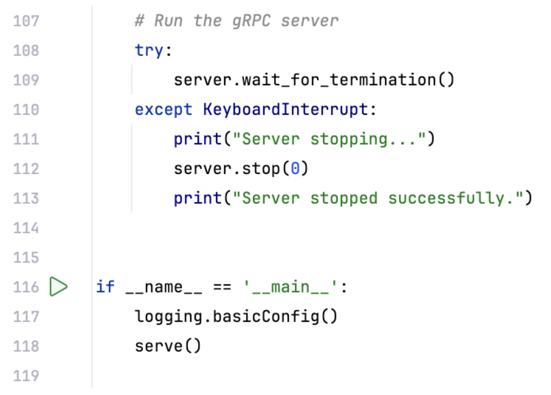


```python
class gRPC_Server(buffer_pb2_grpc.GroundControlServicer):  1 usage  new *
    # method to fetch and return the map from a text file
    def getMap(self, request, context):  1 usage (1 dynamic)  new *
        mapFile = open('map1.txt', 'r')
        global rows
        global columns
        mapDirections = mapFile.readline().split()
        mapRows = mapDirections[0]
        mapColumns = mapDirections[1]

        print("\nrows " + mapRows + "  columns " + mapColumns)
        print("Rover Starts Facing South at (0, 0)")
        print("[ 0 = no mine, 1 = mine ]\n")

        print("Map:")
        mapConents = ""
        for line in mapFile:
            mapConents += line

        print(mapConents)
        return buffer_pb2.mapReply(map=str(mapConents), rows=mapRows, column=mapColumns)

    # method to fetch and return commands for the rover from an external API
    def getCommands(self, request, context):  1 usage (1 dynamic)  new *
        apiResponse = requests.get('https://coe892.reev.dev/lab1/rover/' + str(request.roverNum))
        apiContent = apiResponse.json()
        move = apiContent['data']['moves']

        print("Commands: " + move)
        return buffer_pb2.commandReply(moves=move)

    # method to fetch and return the serial number of a mine based on the rover's position
    def getMineSerialNumber(self, request, context):  1 usage (1 dynamic)  new *
        with open('mines.txt', 'r') as minesFile:
            mineData = minesFile.readlines()
        roverPosition = list(map(int, request.roverPos.split(" ")))
        if roverPosition[1] < len(mineData):
            serial_num = mineData[roverPosition[1]].strip()
            return buffer_pb2.mineSerialNumberReply(serialNum=serial_num)
        else:
            return buffer_pb2.mineSerialNumberReply(serialNum="Invalid Position")

    # method to receive and acknowledge a mine pin
    def getMinePin(self, request, context):  1 usage (1 dynamic)  new *
        pin = request
        print(pin)
        return buffer_pb2.minePinReply(pin="Pin received")

    # acknowledge a successful operation
    def getSuccess(self, request, context):  1 usage (1 dynamic)  new *
        print(request.status)
        return buffer_pb2.successReply(response="Success")
```

Figure 3: *gRPC Server class containing the methods; getMap(). getCommands(), getMineSerialNumber(), getMinePin() and getSuccess()*

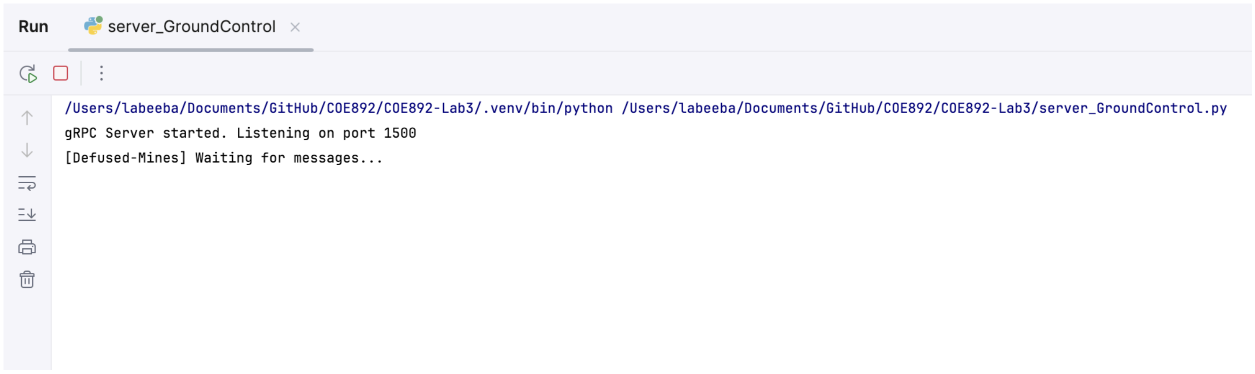


```python
# --------- RabbitMQ Subscriber for Defused-Mines ---------
def rabbitmq_listener():
    try:
        connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
        channel = connection.channel()
        channel.queue_declare(queue='Defused-Mines')

        def callback(ch, method, properties, body):
            print(f"[Defused-Mines] Received: {body.decode()}")
            # Log to a file
            with open("defused_mines.log", "a") as log_file:
                log_file.write(f"{body.decode()}\n")

        channel.basic_consume(queue='Defused-Mines', on_message_callback=callback, auto_ack=True)
        print("[Defused-Mines] Waiting for messages...")
        channel.start_consuming()
    except Exception as e:
        print(f"RabbitMQ Listener Error: {str(e)}")


# function to configure and start the gRPC server
def serve():
    # Create a gRPC server to handle requests
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    buffer_pb2_grpc.add_GroundControlServicer_to_server(gRPC_Server(), server)

    # Specify the port for the server to listen on
    port = 1500
    server.add_insecure_port('[::]:{}'.format(port))

    # Start the server
    server.start()
    print("gRPC Server started. Listening on port {}".format(port))

    # Start RabbitMQ listener in a separate thread
    rabbitmq_thread = threading.Thread(target=rabbitmq_listener, daemon=True)
    rabbitmq_thread.start()
```



```python
    # Run the gRPC server
    try:
        server.wait_for_termination()
    except KeyboardInterrupt:
        print("Server stopping...")
        server.stop(0)
        print("Server stopped successfully.")


if __name__ == '__main__':
    logging.basicConfig()
    serve()
```

Figure 4: *Methods to allow RabbitMQ server to subscribe to the* ***Diffused Mines Channel***

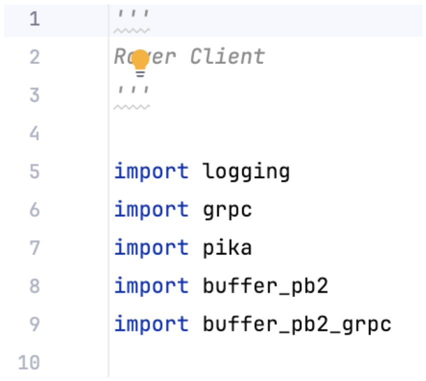

Run server_GroundControl

```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/server_GroundControl.py
gRPC Server started. Listening on port 1500
[Defused-Mines] Waiting for messages...
```

Figure 5: *Running the ground control server*

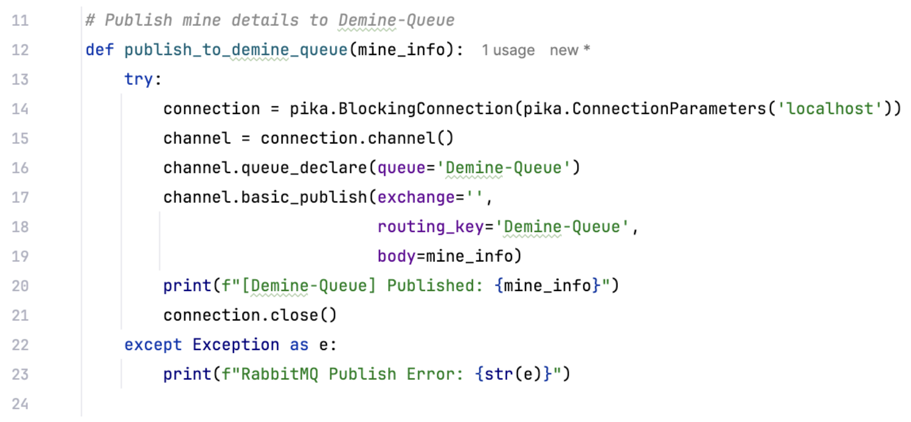
4. Part 3: Rover Client

Next, the rover client was created using the .proto files created in lab 2 and its corresponding code can be seen from *Figure 6 to Figure 9.* The file controls the movements and states for the rover as it navigates through the map as seen in *Figure 8 and Figure 9*. This file accepts a CLI (command line input) argument that indicates a rover number from 1 to 10. Before accepting the CLI argument, the client retrieves the map and stores it to a data structure as seen in *Figure 10*. Then it retrieves the rovers commands and starts navigating through the map sequentially. When the rover finds a mine, the client will get the mines serial number and will publish it along with the mines coordinates and ID to the Demine-Queue channel of RabbitMQ server. This process can be seen in the subsection **Results** and is done using the function *publish_to_demine_queue()* as seen in *Figure 7*.



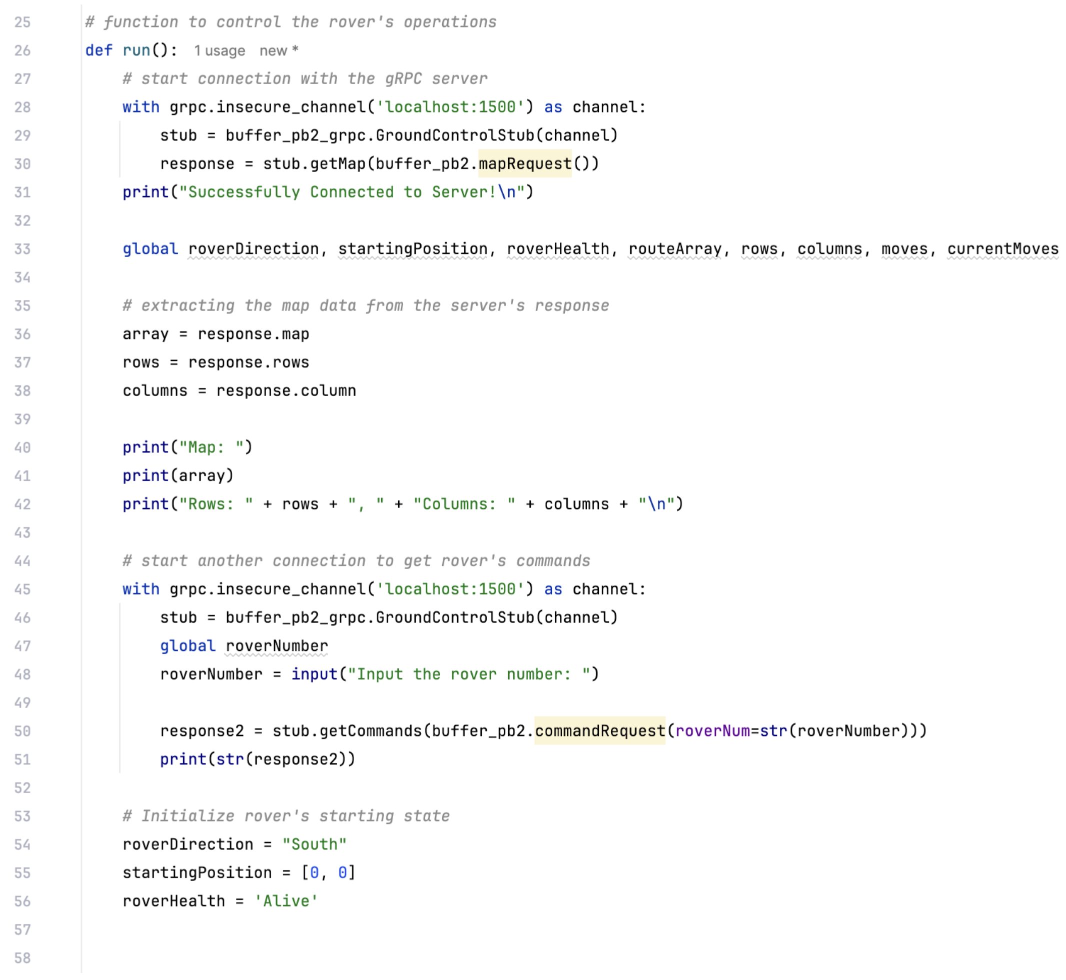
```
'''
Rover Client
'''

import logging
import grpc
import pika
import buffer_pb2
import buffer_pb2_grpc

```

Figure 6: *Imports for Rover Client*

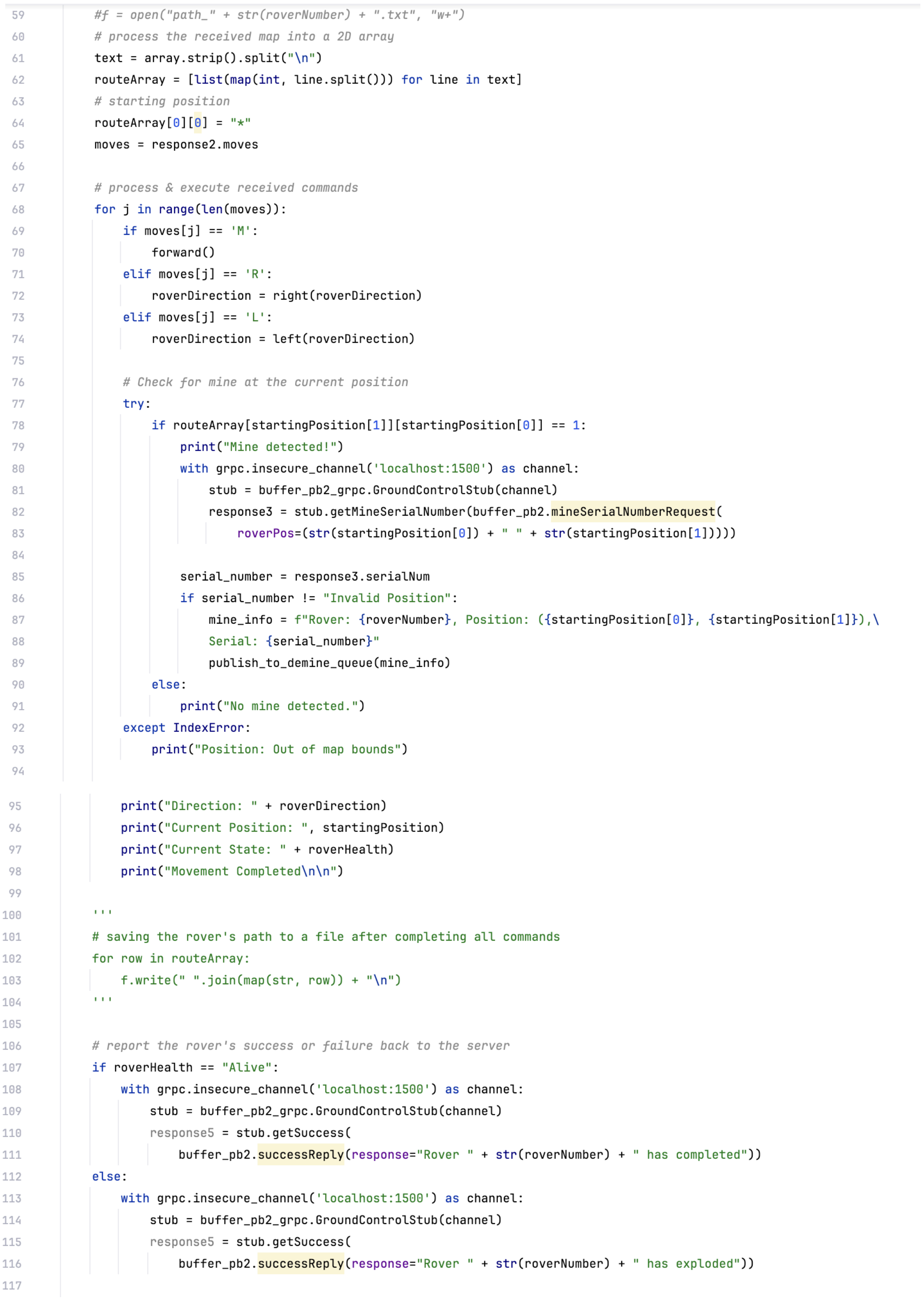


```
# Publish mine details to Demine-Queue
def publish_to_demine_queue(mine_info):  1 usage  new *
    try:
        connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
        channel = connection.channel()
        channel.queue_declare(queue='Demine-Queue')
        channel.basic_publish(exchange='',
                              routing_key='Demine-Queue',
                              body=mine_info)
        print(f"[Demine-Queue] Published: {mine_info}")
        connection.close()
    except Exception as e:
        print(f"RabbitMQ Publish Error: {str(e)}")

```

Figure 7: *Function to publish to the demine queue*

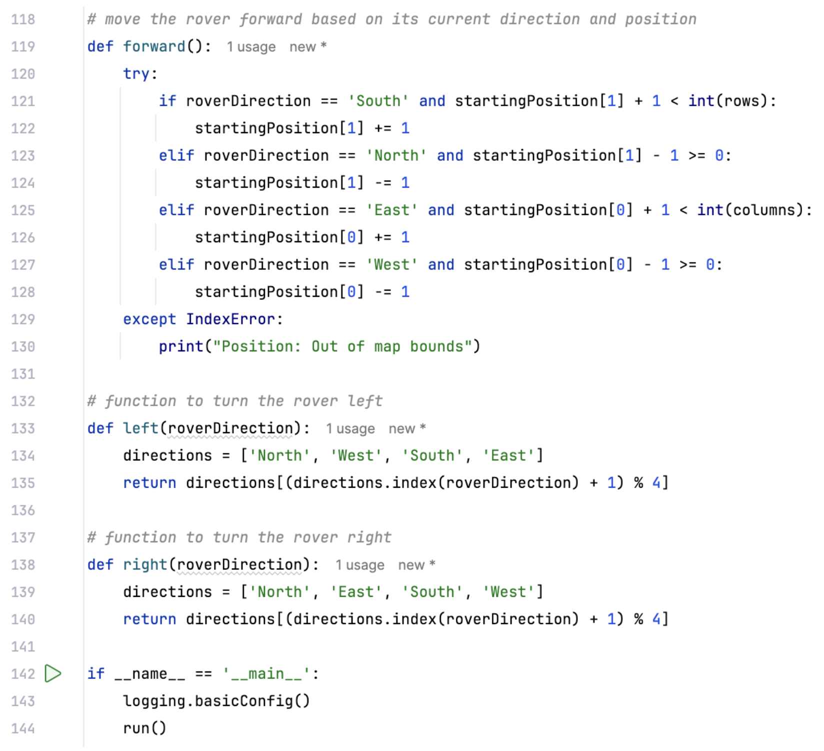


```
# function to control the rover's operations
def run():  1 usage  new *
    # start connection with the gRPC server
    with grpc.insecure_channel('localhost:1500') as channel:
        stub = buffer_pb2_grpc.GroundControlStub(channel)
        response = stub.getMap(buffer_pb2.mapRequest())
    print("Successfully Connected to Server!\n")

    global roverDirection, startingPosition, roverHealth, routeArray, rows, columns, moves, currentMoves

    # extracting the map data from the server's response
    array = response.map
    rows = response.rows
    columns = response.column

    print("Map: ")
    print(array)
    print("Rows: " + rows + ", " + "Columns: " + columns + "\n")

    # start another connection to get rover's commands
    with grpc.insecure_channel('localhost:1500') as channel:
        stub = buffer_pb2_grpc.GroundControlStub(channel)
        global roverNumber
        roverNumber = input("Input the rover number: ")

        response2 = stub.getCommands(buffer_pb2.commandRequest(roverNum=str(roverNumber)))
        print(str(response2))

    # Initialize rover's starting state
    roverDirection = "South"
    startingPosition = [0, 0]
    roverHealth = 'Alive'


```



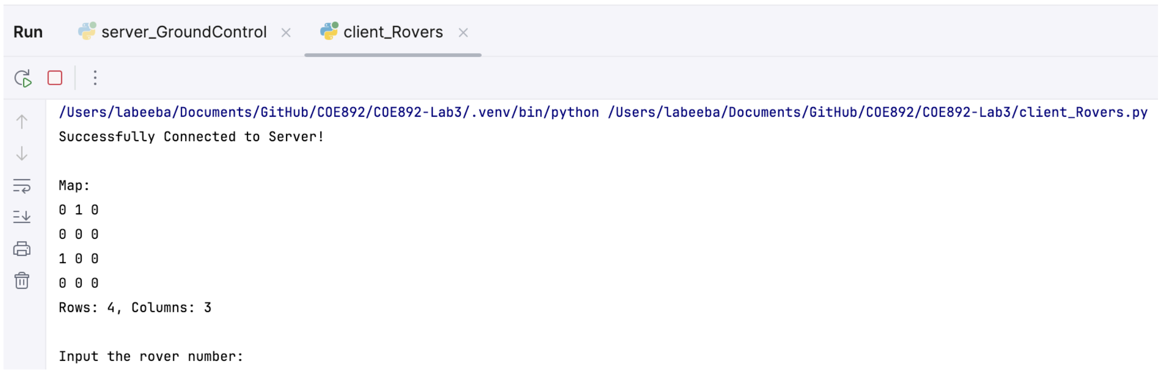
```python
    #f = open("path_" + str(roverNumber) + ".txt", "w+")
    # process the received map into a 2D array
    text = array.strip().split("\n")
    routeArray = [list(map(int, line.split())) for line in text]
    # starting position
    routeArray[0][0] = "*"
    moves = response2.moves

    # process & execute received commands
    for j in range(len(moves)):
        if moves[j] == 'M':
            forward()
        elif moves[j] == 'R':
            roverDirection = right(roverDirection)
        elif moves[j] == 'L':
            roverDirection = left(roverDirection)

        # Check for mine at the current position
        try:
            if routeArray[startingPosition[1]][startingPosition[0]] == 1:
                print("Mine detected!")
                with grpc.insecure_channel('localhost:1500') as channel:
                    stub = buffer_pb2_grpc.GroundControlStub(channel)
                    response3 = stub.getMineSerialNumber(buffer_pb2.mineSerialNumberRequest(
                        roverPos=(str(startingPosition[0]) + " " + str(startingPosition[1]))))

                serial_number = response3.serialNum
                if serial_number != "Invalid Position":
                    mine_info = f"Rover: {roverNumber}, Position: ({startingPosition[0]}, {startingPosition[1]}),\
                    Serial: {serial_number}"
                    publish_to_demine_queue(mine_info)
            else:
                print("No mine detected.")
        except IndexError:
            print("Position: Out of map bounds")

        print("Direction: " + roverDirection)
        print("Current Position: ", startingPosition)
        print("Current State: " + roverHealth)
        print("Movement Completed\n\n")

    '''
    # saving the rover's path to a file after completing all commands
    for row in routeArray:
        f.write(" ".join(map(str, row)) + "\n")
    '''

    # report the rover's success or failure back to the server
    if roverHealth == "Alive":
        with grpc.insecure_channel('localhost:1500') as channel:
            stub = buffer_pb2_grpc.GroundControlStub(channel)
            response5 = stub.getSuccess(
                buffer_pb2.successReply(response="Rover " + str(roverNumber) + " has completed"))
    else:
        with grpc.insecure_channel('localhost:1500') as channel:
            stub = buffer_pb2_grpc.GroundControlStub(channel)
            response5 = stub.getSuccess(
                buffer_pb2.successReply(response="Rover " + str(roverNumber) + " has exploded"))
```

Figure 8: *Function that controls the Rovers operations*



```python
# move the rover forward based on its current direction and position
def forward():  1 usage  new *
    try:
        if roverDirection == 'South' and startingPosition[1] + 1 < int(rows):
            startingPosition[1] += 1
        elif roverDirection == 'North' and startingPosition[1] - 1 >= 0:
            startingPosition[1] -= 1
        elif roverDirection == 'East' and startingPosition[0] + 1 < int(columns):
            startingPosition[0] += 1
        elif roverDirection == 'West' and startingPosition[0] - 1 >= 0:
            startingPosition[0] -= 1
    except IndexError:
        print("Position: Out of map bounds")

# function to turn the rover left
def left(roverDirection):  1 usage  new *
    directions = ['North', 'West', 'South', 'East']
    return directions[(directions.index(roverDirection) + 1) % 4]

# function to turn the rover right
def right(roverDirection):  1 usage  new *
    directions = ['North', 'East', 'South', 'West']
    return directions[(directions.index(roverDirection) + 1) % 4]

if __name__ == '__main__':
    logging.basicConfig()
    run()
```

Figure 9: *Functions that move the rover forward, turn left and right*

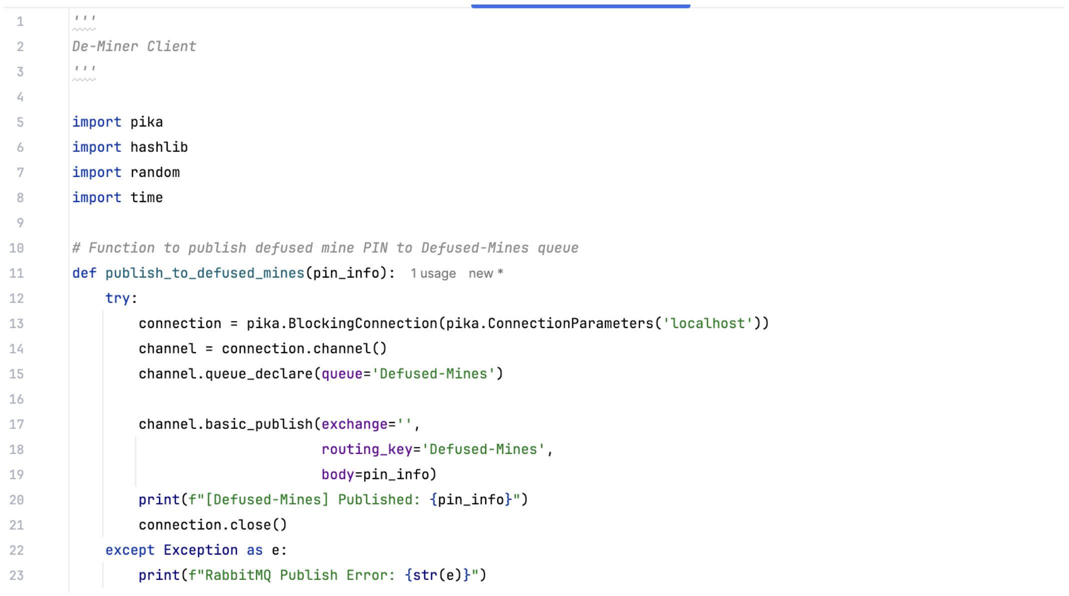


```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/client_Rovers.py
Successfully Connected to Server!

Map:
0 1 0
0 0 0
1 0 0
0 0 0
Rows: 4, Columns: 3

Input the rover number:
```

Figure 10: *Results of the rover client before a CLI argument is passed*

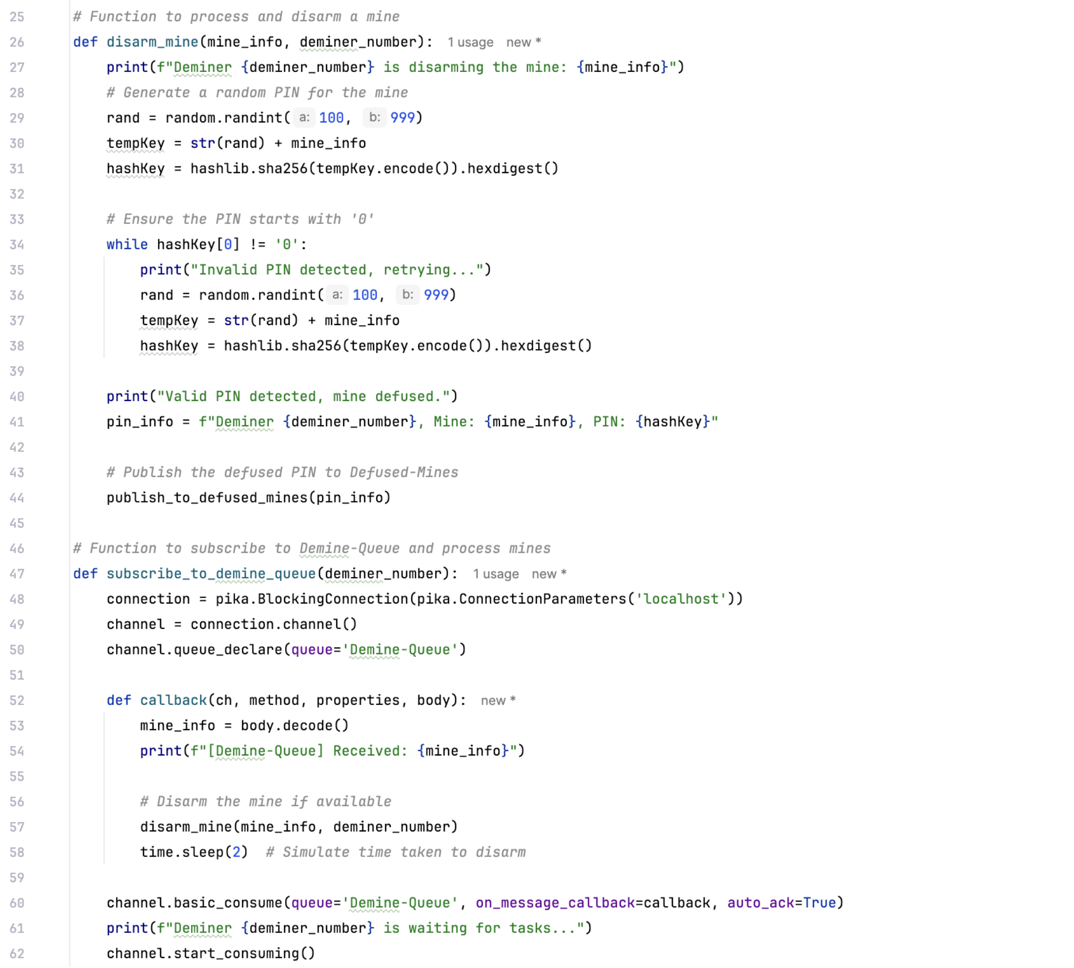
5. Part 4: De-Miner Client

Finally the de-miner client was implemented in a separate python file. The main purpose of the de-miner client is to replace the dig command and use RabbitMQ for messaging instead . It starts by subscribing to the Demine Queue to receive the mine information. When a mine is detected, it generates a random PIN, combines it with the mine info, and hashes it using SHA-256. If the hash does not start with 0, it retries until a valid PIN is generated. Once a valid PIN is found, it publishes the defused mine information, including the PIN, to the Defused Mines queue. The program supports two deminers, 1 or 2 and continuously listens for new tasks from the queue.



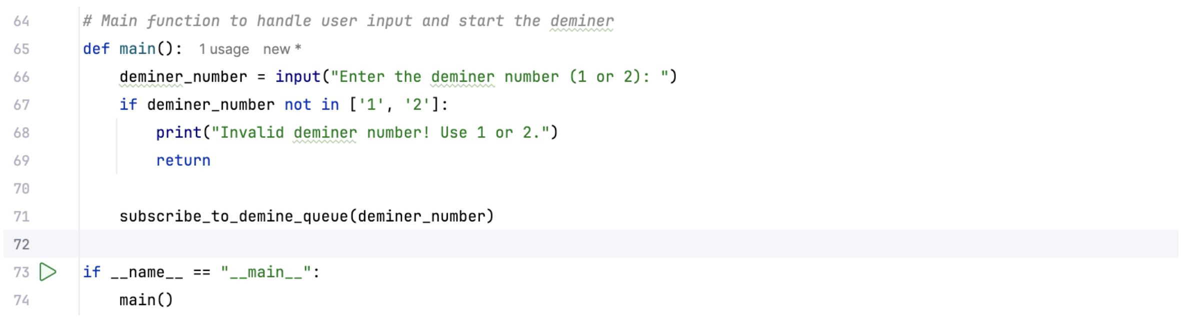
```python
'''
De-Miner Client
'''

import pika
import hashlib
import random
import time

# Function to publish defused mine PIN to Defused-Mines queue
def publish_to_defused_mines(pin_info):  1 usage  new *
    try:
        connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
        channel = connection.channel()
        channel.queue_declare(queue='Defused-Mines')

        channel.basic_publish(exchange='',
                              routing_key='Defused-Mines',
                              body=pin_info)
        print(f"[Defused-Mines] Published: {pin_info}")
        connection.close()
    except Exception as e:
        print(f"RabbitMQ Publish Error: {str(e)}")
```

Figure 11: *Imports and function to publish the defused mine pins to the defused-mine queue*



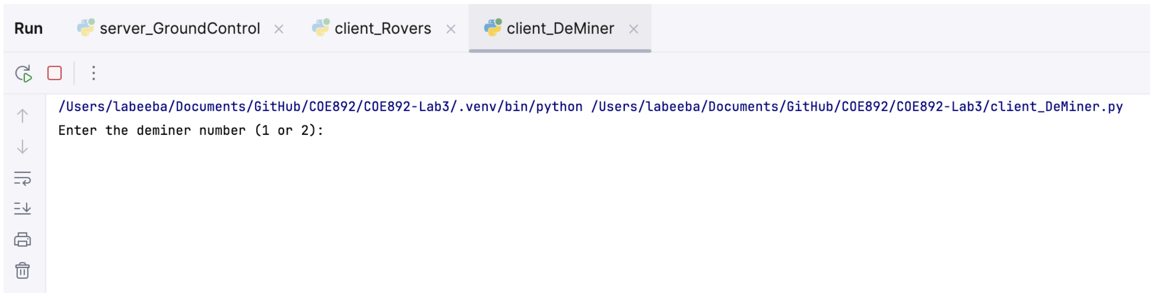
```python
# Function to process and disarm a mine
def disarm_mine(mine_info, deminer_number):  1 usage  new *
    print(f"Deminer {deminer_number} is disarming the mine: {mine_info}")
    # Generate a random PIN for the mine
    rand = random.randint( a: 100,  b: 999)
    tempKey = str(rand) + mine_info
    hashKey = hashlib.sha256(tempKey.encode()).hexdigest()

    # Ensure the PIN starts with '0'
    while hashKey[0] != '0':
        print("Invalid PIN detected, retrying...")
        rand = random.randint( a: 100,  b: 999)
        tempKey = str(rand) + mine_info
        hashKey = hashlib.sha256(tempKey.encode()).hexdigest()

    print("Valid PIN detected, mine defused.")
    pin_info = f"Deminer {deminer_number}, Mine: {mine_info}, PIN: {hashKey}"

    # Publish the defused PIN to Defused-Mines
    publish_to_defused_mines(pin_info)

# Function to subscribe to Demine-Queue and process mines
def subscribe_to_demine_queue(deminer_number):  1 usage  new *
    connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    channel = connection.channel()
    channel.queue_declare(queue='Demine-Queue')

    def callback(ch, method, properties, body):  new *
        mine_info = body.decode()
        print(f"[Demine-Queue] Received: {mine_info}")

        # Disarm the mine if available
        disarm_mine(mine_info, deminer_number)
        time.sleep(2)  # Simulate time taken to disarm

    channel.basic_consume(queue='Demine-Queue', on_message_callback=callback, auto_ack=True)
    print(f"Deminer {deminer_number} is waiting for tasks...")
    channel.start_consuming()
```

Figure 11: *Function to disarm a mine and subscribe to the demine queue respectively*



```
# Main function to handle user input and start the deminer
def main():  1 usage  new *
    deminer_number = input("Enter the deminer number (1 or 2): ")
    if deminer_number not in ['1', '2']:
        print("Invalid deminer number! Use 1 or 2.")
        return

    subscribe_to_demine_queue(deminer_number)

if __name__ == "__main__":
    main()
```

Figure 12: *Main function to handle user input and start the de-miner*



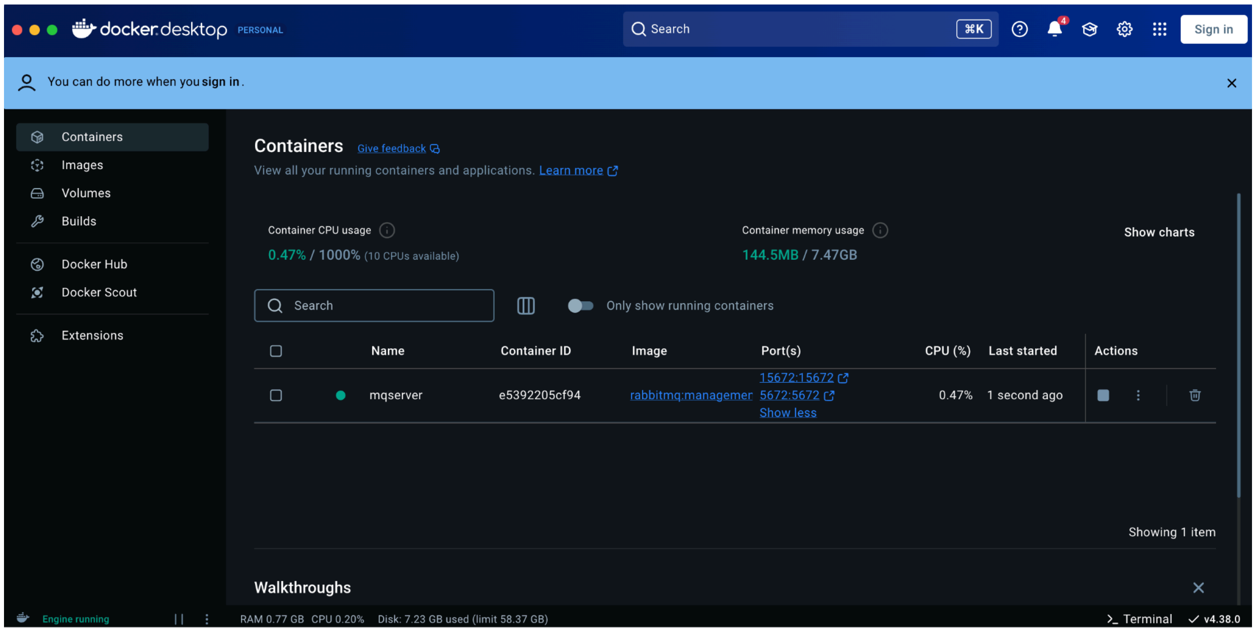
```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/client_DeMiner.py
Enter the deminer number (1 or 2):
```

Figure 13: *Console output when client_DeMiner is run*

6. Results

The following figures showcase the complete process of this lab using Rover 1's commands and De-Miner 1.

Step 1: Start the RabbitMQ Container



Containers
View all your running containers and applications. Learn more

Container CPU usage 0.47% / 1000% (10 CPUs available)
Container memory usage 144.5MB / 7.47GB

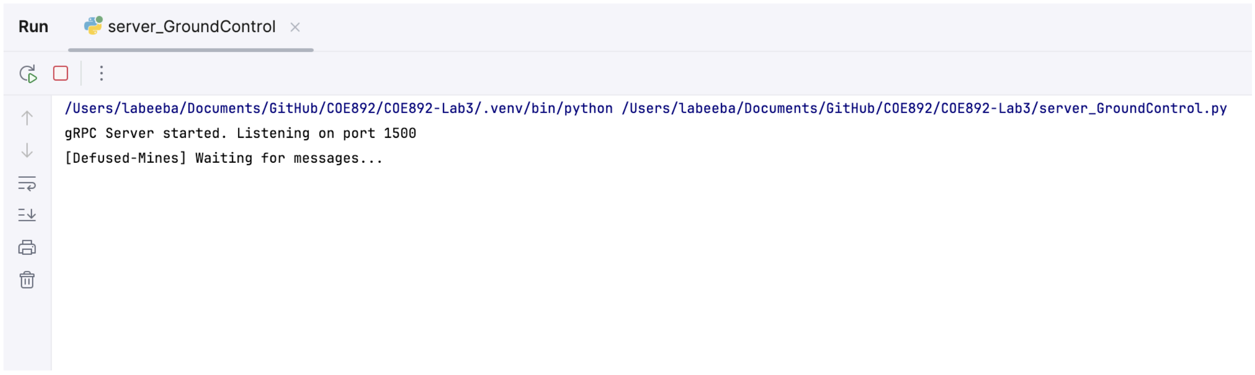
Name	Container ID	Image	Port(s)	CPU (%)	Last started
mqserver	e5392205cf94	rabbitmq:managemer	15672:15672 5672:5672 Show less	0.47%	1 second ago

Showing 1 item

RAM 0.77 GB CPU 0.20% Disk: 7.23 GB used (limit 58.37 GB)

Figure 14: *Starting the RabbitMQ container*

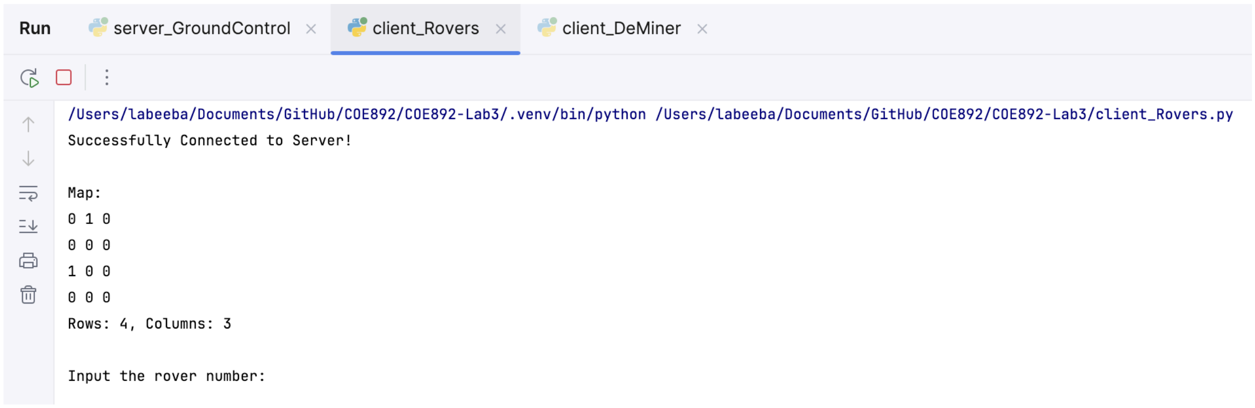
Step 2: Run Ground Control Server



```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/server_GroundControl.py
gRPC Server started. Listening on port 1500
[Defused-Mines] Waiting for messages...
```

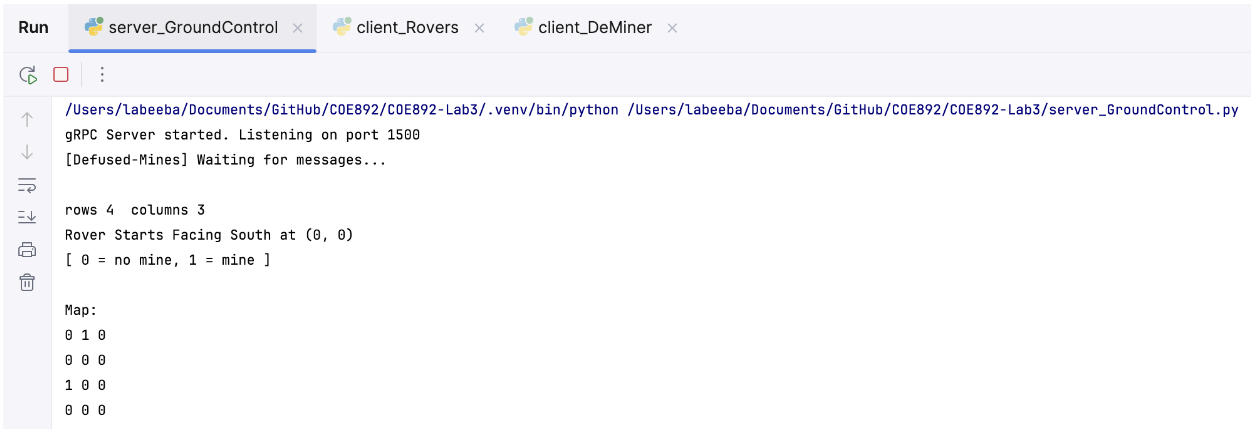
Figure 15: *Ground Control Server Console Output*

Step 3: Run Rover Client



```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/client_Rovers.py
Successfully Connected to Server!

Map:
0 1 0
0 0 0
1 0 0
0 0 0
Rows: 4, Columns: 3

Input the rover number:
```

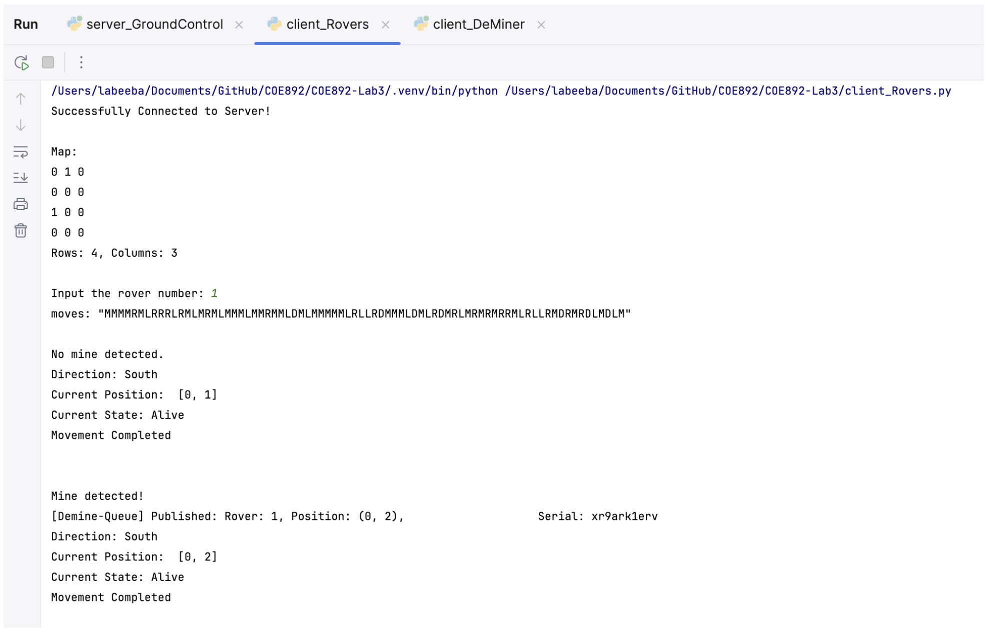
Figure 16: *Rover Client Console Output*



```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/server_GroundControl.py
gRPC Server started. Listening on port 1500
[Defused-Mines] Waiting for messages...

rows 4  columns 3
Rover Starts Facing South at (0, 0)
[ 0 = no mine, 1 = mine ]

Map:
0 1 0
0 0 0
1 0 0
0 0 0
```

Figure 17: *Ground Control Server Returns the map one the client is started*

Step 3: Input Rover Number = 1 in Client

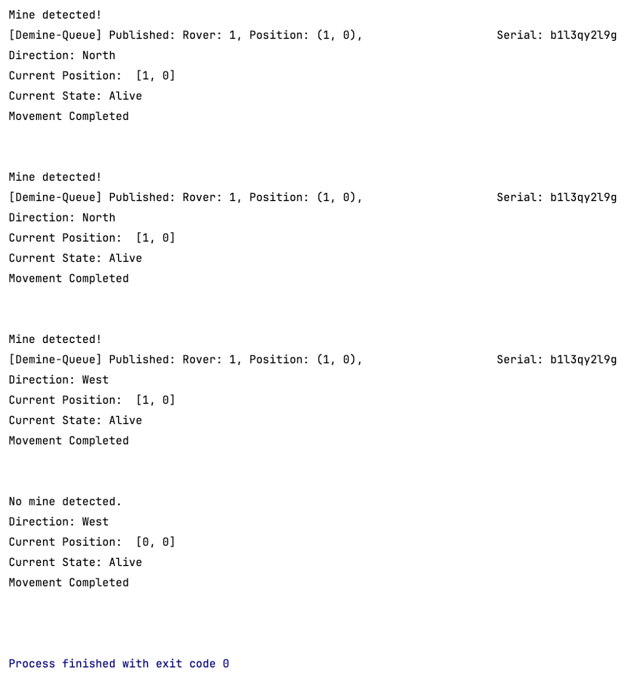


```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/client_Rovers.py
Successfully Connected to Server!

Map:
0 1 0
0 0 0
1 0 0
0 0 0
Rows: 4, Columns: 3

Input the rover number: 1
moves: "MMMMRMLRRRLRMLMRMLMMMLMMRMMLDMLMMMMMLRLLRDMMMLDMLRDMRLMRMRMRRMLRLLRMDRMRDLMDLM"

No mine detected.
Direction: South
Current Position:  [0, 1]
Current State: Alive
Movement Completed


Mine detected!
[Demine-Queue] Published: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv
Direction: South
Current Position:  [0, 2]
Current State: Alive
Movement Completed
```

▪

▪

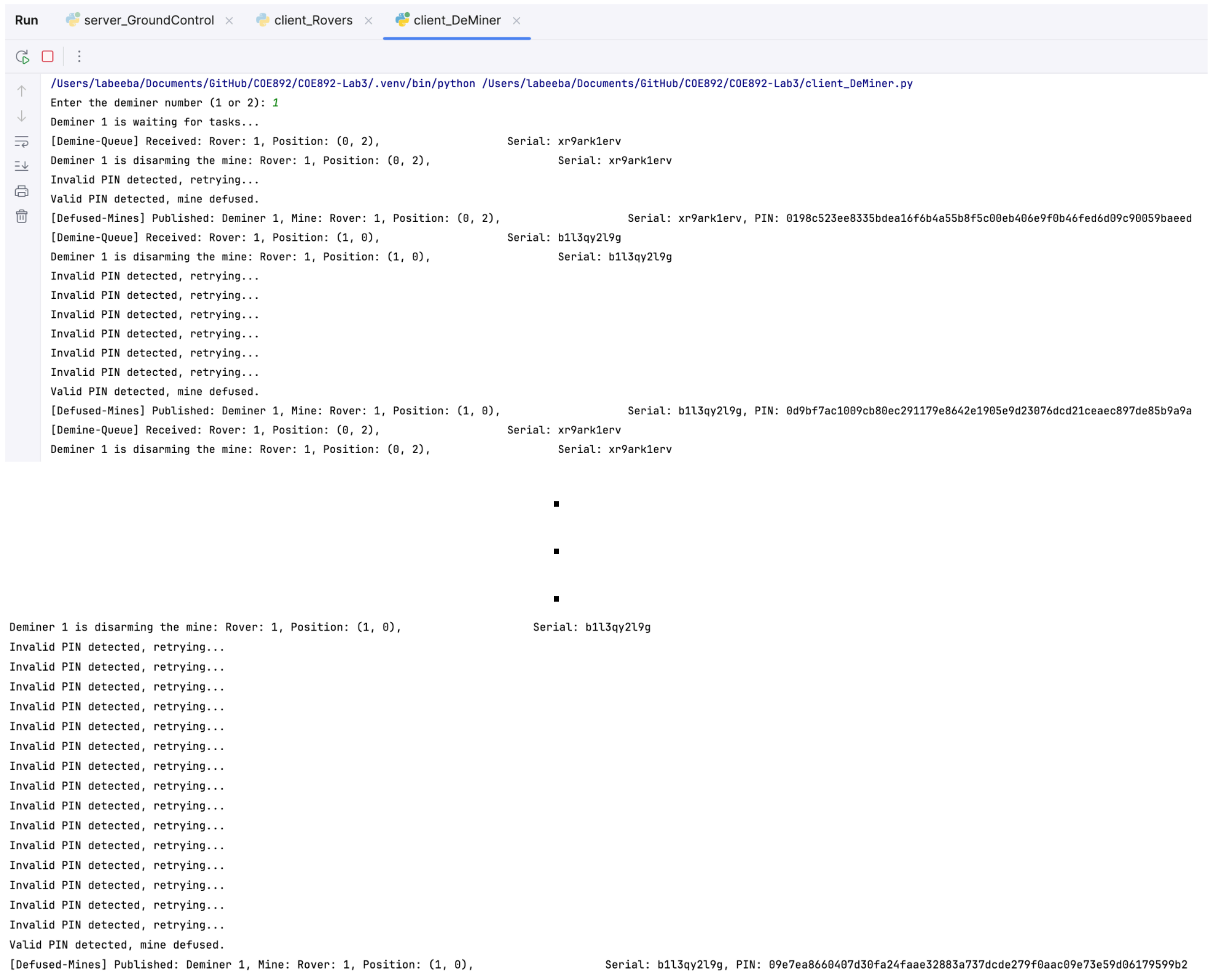
▪



```
Mine detected!
[Demine-Queue] Published: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g
Direction: North
Current Position:  [1, 0]
Current State: Alive
Movement Completed


Mine detected!
[Demine-Queue] Published: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g
Direction: North
Current Position:  [1, 0]
Current State: Alive
Movement Completed


Mine detected!
[Demine-Queue] Published: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g
Direction: West
Current Position:  [1, 0]
Current State: Alive
Movement Completed


No mine detected.
Direction: West
Current Position:  [0, 0]
Current State: Alive
Movement Completed



Process finished with exit code 0
```

Figure 18: *Rover 1 Process Complete*

Step 3: De-Miner Number = 1 in Deminer Client



```
/Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/.venv/bin/python /Users/labeeba/Documents/GitHub/COE892/COE892-Lab3/client_DeMiner.py
Enter the deminer number (1 or 2): 1
Deminer 1 is waiting for tasks...
[Demine-Queue] Received: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv
Deminer 1 is disarming the mine: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv
Invalid PIN detected, retrying...
Valid PIN detected, mine defused.
[Defused-Mines] Published: Deminer 1, Mine: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv, PIN: 0198c523ee8335bdea16f6b4a55b8f5c00eb406e9f0b46fed6d09c90059baeed
[Demine-Queue] Received: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g
Deminer 1 is disarming the mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Valid PIN detected, mine defused.
[Defused-Mines] Published: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0d9bf7ac1009cb80ec291179e8642e1905e9d23076dcd21ceaec897de85b9a9a
[Demine-Queue] Received: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv
Deminer 1 is disarming the mine: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv
```

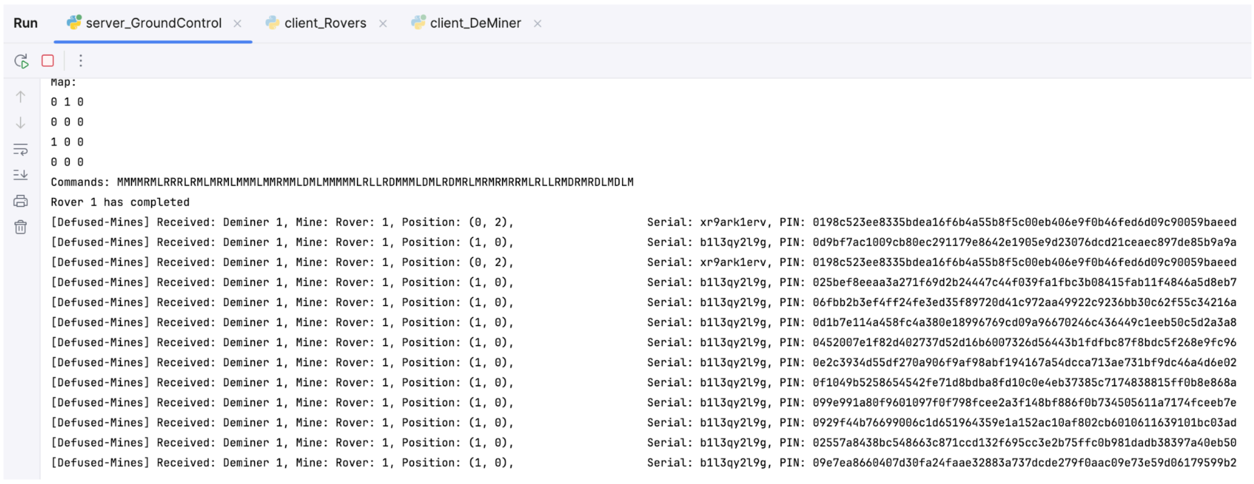
.

.

.

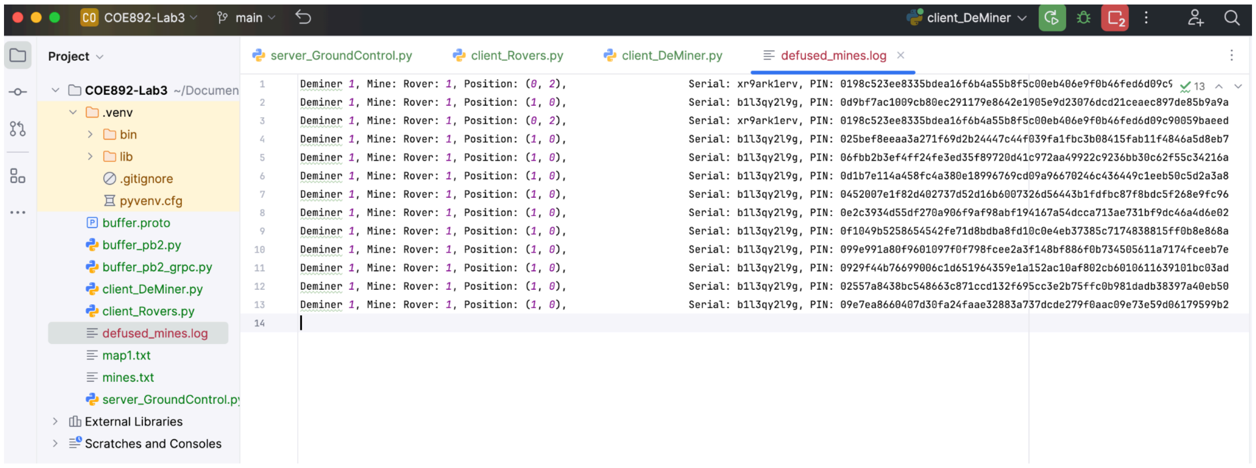
```
Deminer 1 is disarming the mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Invalid PIN detected, retrying...
Valid PIN detected, mine defused.
[Defused-Mines] Published: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 09e7ea8660407d30fa24faae32883a737dcde279f0aac09e73e59d06179599b2
```

Figure 19: *DeMiner 1 Process for Rover 1*



```
Map:
0 1 0
0 0 0
1 0 0
0 0 0
Commands: MMMMRMLRRRLRMLMRMLMMMLMMRMMLDMLMMMMMLRLLRDMMMLDMLRDMRLMRMRMRRMLRLLRMDRMRDLMDLM
Rover 1 has completed
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv, PIN: 0198c523ee8335bdea16f6b4a55b8f5c00eb406e9f0b46fed6d09c90059baeed
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0d9bf7ac1009cb80ec291179e8642e1905e9d23076dcd21ceaec897de85b9a9a
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (0, 2),                    Serial: xr9ark1erv, PIN: 0198c523ee8335bdea16f6b4a55b8f5c00eb406e9f0b46fed6d09c90059baeed
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 025bef8eeaa3a271f69d2b24447c44f039fa1fbc3b08415fab11f4846a5d8eb7
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 06fbb2b3ef4ff24fe3ed35f89720d41c972aa49922c9236bb30c62f55c34216a
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0d1b7e114a458fc4a380e18996769cd09a96670246c436449c1eeb50c5d2a3a8
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0452007e1f82d402737d52d16b6007326d56443b1fdfbc87f8bdc5f268e9fc96
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0e2c3934d55df270a906f9af98abf194167a54dcca713ae731bf9dc46a4d6e02
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0f1049b5258654542fe71d8bdba8fd10c0e4eb37385c7174838815ff0b8e868a
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 099e991a80f9601097f0f798fcee2a3f148bf886f0b734505611a7174fceeb7e
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 0929f44b76699006c1d651964359e1a152ac10af802cb6010611639101bc03ad
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 02557a8438bc548663c871ccd132f695cc3e2b75ffc0b981dadb38397a40eb50
[Defused-Mines] Received: Deminer 1, Mine: Rover: 1, Position: (1, 0),                    Serial: b1l3qy2l9g, PIN: 09e7ea8660407d30fa24faae32883a737dcde279f0aac09e73e59d06179599b2
```

Figure 20: *server_GroundControl Results when Deminer 1 ran*



```
Deminer 1, Mine: Rover: 1, Position: (0, 2),                     Serial: xr9ark1erv, PIN: 0198c523ee8335bdea16f6b4a55b8f5c00eb406e9f0b46fed6d09c
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 0d9bf7ac1009cb80ec291179e8642e1905e9d23076dcd21ceaec897de85b9a9a
Deminer 1, Mine: Rover: 1, Position: (0, 2),                     Serial: xr9ark1erv, PIN: 0198c523ee8335bdea16f6b4a55b8f5c00eb406e9f0b46fed6d09c90059baeed
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 025bef8eeaa3a271f69d2b24447c44f039fa1fbc3b08415fab11f4846a5d8eb7
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 06fbb2b3ef4ff24fe3ed35f89720d41c972aa49922c9236bb30c62f55c34216a
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 0d1b7e114a458fc4a380e18996769cd09a96670246c436449c1eeb50c5d2a3a8
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 0452007e1f82d402737d52d16b6007326d56443b1fdfbc87f8bdc5f268e9fc96
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 0e2c3934d55df270a906f9af98abf194167a54dcca713ae731bf9dc46a4d6e02
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 0f1049b5258654542fe71d8bdba8fd10c0e4eb37385c7174838815ff0b8e868a
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 099e991a80f9601097f0f798fcee2a3f148bf886f0b734505611a7174fceeb7e
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 0929f44b76699006c1d651964359e1a152ac10af802cb6010611639101bc03ad
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 02557a8438bc548663c871ccd132f695cc3e2b75ffc0b981dadb38397a40eb50
Deminer 1, Mine: Rover: 1, Position: (1, 0),                     Serial: b1l3qy2l9g, PIN: 09e7ea8660407d30fa24faae32883a737dcde279f0aac09e73e59d06179599b2
```

Figure 20: *De-Miner results stored in defused_mines.log*

7. Conclusion

In this lab, a distributed mine detection and disposal system using gRPC and RabbitMQ was successfully implemented. By creating separate clients for rovers and deminers and a server for ground control, effective inter-process communication through message queues was demonstrated. The rovers navigated through a map, identified mines, and communicated their locations using RabbitMQ. The deminers then disarmed the mines and published the results back to ground control. Overall this lab was successfully completed.

References

[1] M. Jaseemuddin et al., "COE892 Lab Manual," *Dept. Elect., Comput., and Biomed. Eng., Ryerson Univ.*, Toronto, ON, Canada, pp. 1-16, Winter 2024.