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.	2

Section No.	05
Submission Date	February 23rd, 2025
Due Date	February 23rd, 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

The objective of this lab was to use gRPC in Python to simulate the communication of rovers (clients) with ground control (server) which was an extension of lab 1. Each rover must implement the 5 methods below:

1. Get map (e.g., “The 2D land array”)
2. Get a stream of commands (e.g., “RMLMMMMMDLMMRMD”)
3. Get a mine serial number
4. Let the server know if they have executed all commands successfully, or not (a mine might explode with a wrong series of commands)
5. Share a mine PIN with the server

The lab consisted of 3 main parts. First, Proto Buffer Files were implemented as it is required for the gRPC communication. Next, the server was created which uses the 2D map files used in lab 1, the rover command files containing the commands given to each rover and a mine serial number file. Finally the client was created, which simulates the rovers. Each client initializes by retrieving and storing the operational map from the server, then processes a stream of commands for navigation and interaction with obstacles like mines, retrieves necessary mine serial numbers for disarming, and communicates back to the server regarding the success or failure of command execution and mine disarming tasks.

2. Part 1: Proto Buffer Files

During this part of the lab, buffer.proto file seen in *Figure 1* was created which is a protocol buffer. This is essential for defining the structured data types for this lab's gRPC-based communication between the client and server. The buffer.proto file was meticulously includes service definitions such as GroundControlService, which comprised methods like GetMap and GetCommands. These methods allowed data to be exchanged across the network, such as sending map details and rover commands.

```
1   syntax = "proto3";
2
3   package rover;
4
5   //Ground Control service definition
6   service GroundControl{
7     rpc getMap(mapRequest) returns (mapReply){}
8     rpc getCommands(commandRequest) returns (commandReply){}
9     rpc getMineSerialNumber(mineSerialNumberRequest) returns (mineSerialNumberReply){}
10    rpc getSuccess(successRequest) returns (successReply){}
11    rpc getMinePin(minePinRequest) returns (minePinReply){}
12  }
13
14  message mapRequest{
15    string request = 1;
16  }
```

```
17
18  message mapReply{
19    string map = 1;
20    string rows = 2;
21    string column = 3;
22  }
23
24  message commandRequest{
25    string roverNum = 1;
26  }
27
28  message commandReply{
29    string moves = 1;
30  }
31
32  message mineSerialNumberRequest{
33    string roverPos = 1;
34  }
35
36  message mineSerialNumberReply{
37    string serialNum = 1;
38  }
39
40  message successRequest{
41    string status = 1;
42  }
43
44  message successReply{
45    string response = 1;
46  }
47
48  message minePinRequest{
49    string mineNum = 1;
50  }
51
52  message minePinReply{
53    string pin = 1;
54  }
```

***Figure 1**: buffer.proto code*

This file then needed to be initialized using the terminal which then generated buffer_pb2.py and buffer_pb2_grpc.py. These two files enabled communication between the server and client.

3. Part 2: Server

This part of the lab required the implementation of the gRPC server seen in *Figure 2* which acts as 'ground control' for managing communication with the rovers. Using the proto file created in part 1 and seen in Figure 1, the server was able to handle necessary operations such as fetching and serving 2D map data from text files, retrieving command sequences for individual rovers from separate command files, and accessing a mine serial number from a mines.txt file. Additionally, the server was designed to handle requests for mine serial numbers and to display if the rover was successfully executed.

```python
from concurrent import futures
import logging

import grpc
import requests
import json

# import the protobuf files for gRPC service and message definitions
import buffer_pb2
import buffer_pb2_grpc


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 *
        # open the map file in read mode
        mapFile = open('map1.txt', 'r')
        global rows
        global columns
        # read the first line to get the map dimensions
        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 = ""
        # concatenate rest of the lines to get the full map contents
        for line in mapFile:
            mapConents += line

        print(mapConents)
        # return the map contents and dimensions as a gRPC message
        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()  # Directly parse the JSON
        move = apiContent['data']['moves']

        print("Commands: " + move)
        # return the commands as a gRPC message
        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()  # Assuming one serial number per line
            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")


# function to configure and start the gRPC server
def serve():  1 usage  new *
    # 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("Server started. Listening on port {}".format(port))

    # Additional info could be included here such as server's IP address if needed
    # However, in most local cases, it's localhost or 127.0.0.1
    try:
        server.wait_for_termination()
    except KeyboardInterrupt:
        print("Server stopping...")
        server.stop(0)
        print("Server stopped successfully.")


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

Figure 2: *gRPC_server.py Code showing the server implementation*

4. Part 3: Client

The final step of this lab required the implementation of the gRPC client seen in *Figure 3*, which shows the rovers receiving commands and interacting with the server. The client was designed to initiate requests to the ground control server for operational data such as reading 2D maps and specific navigation commands. Each rover client is identified by a unique number which retrieves a map and a sequence of commands. Each rover then executes these commands sequentially to simulate a rover's navigation and disarming mines.

```python
import hashlib
import logging
import random

# import the protobuf files for gRPC service and message definitions
import grpc
import buffer_pb2
import buffer_pb2_grpc

# 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 variables to track rover's state
    global roverDirection
    global startingPosition
    global roverHealth
    global routeArray
    global rows
    global columns
    global moves
    global 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'

    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
```

```
57
58      # process & execute received commands
59      for j in range(len(moves)):
60          # check current command needs to move forward and if so execute the forward movement function
61          if moves[j] == 'M':
62              forward()
63          # check if command is to move right, and adjust the rover's direction
64          elif moves[j] == 'R':
65              roverDirection = right(roverDirection)
66          # check if command is to move left, and adjust the rover's direction
67          elif moves[j] == 'L':
68              roverDirection = left(roverDirection)
69          # check if the command is to dig at the current position
70          elif moves[j] == 'D':
71              try:
72                  # If the current position has a mine ('1'), start the digging process
73                  if 0 <= startingPosition[1] < len(routeArray) and 0 <= startingPosition[0] < len(routeArray[startingPosition[1]]):
74                      if routeArray[startingPosition[1]][startingPosition[0]] == 1:
75                          print("Start Digging")
76
```

```
76
77                          # start a gRPC connection to get the mine's serial number
78                          with grpc.insecure_channel('localhost:1500') as channel:
79                              stub = buffer_pb2_grpc.GroundControlStub(channel)
80                              response3 = stub.getMineSerialNumber(buffer_pb2.mineSerialNumberRequest(
81                                  roverPos=(str(startingPosition[0]) + " " + str(startingPosition[1]))))
82
83                          print(response3)
84                          # disarm the mine by generating a hash key and checking if it's valid
85                          rand = random.randint( a: 100,  b: 999)
86                          tempKey = str(rand) + str(routeArray[startingPosition[1]][startingPosition[0]])
87                          print("Temporary key = ", tempKey)
88                          hashKey = hashlib.sha256(tempKey.encode()).hexdigest()
89                          print("Hash key = " + hashKey)
90                          while hashKey[0] != '0':
91                              print("Invalid PIN Detected, Attempting to Retry with a Different PIN")
92                              rand = random.randint( a: 100,  b: 999)
93                              tempKey = str(rand) + str(routeArray[startingPosition[1]][startingPosition[0]])
94                              # print("Temporary key = " + tempKey)
95                              hashKey = hashlib.sha256(tempKey.encode()).hexdigest()
96                              print("Hash key = " + hashKey)
97                          print("Valid Print Detected, Disarming Mine")
98
99                          # send valid pin back to the server to disarm the mine
100                         with grpc.insecure_channel('localhost:1500') as channel:
101                             stub = buffer_pb2_grpc.GroundControlStub(channel)
102                             response4 = stub.getMinePin(buffer_pb2.minePinRequest(mineNum=hashKey))
103                     else:
104                         print("Detecting Mine ... No mine")
105                 else:
106                     print("Position: Out of map bounds")
107             except IndexError:
108                 print("Error in Index")
109
110             # print the rover's current direction, position, and health state after each command
111             print("Direction: " + roverDirection)
112             print("Current Position: ", startingPosition)
113             print("Current State: " + roverHealth)
114             print("Movement Completed\n\n")
```

```
        # if rover encounters a mine without a dig command, it is destroyed
        elif routeArray[startingPosition[0]][startingPosition[1]] != '0':
            print("Mine Located! No Digging Attempt Made, Resulting in Explosion!")
            roverHealth = "Dead"
            break

    # 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"))

# move the rover forward based on its current direction and position
def forward():  1 usage  new *
    try:
        if roverDirection == 'North':
            if startingPosition[1] == 0:
                print("Position: Boundary Reached ... Command Ignored")
                return
            if routeArray[startingPosition[1] - 1][startingPosition[0]] == '1':
                print("Collision Detected ... Rover Destroyed")
                roverHealth = 'Dead'
                return roverHealth

            startingPosition[1] = startingPosition[1] - 1
            routeArray[startingPosition[1]][startingPosition[0]] = "*"

        elif roverDirection == 'South':

            if startingPosition[1] == rows:
                print("Position: Boundary Reached ... Command Ignored")
                return
            if routeArray[startingPosition[1] + 1][startingPosition[0]] == '1':
                print("Collision Detected ... Rover Destroyed")
                roverHealth = 'Dead'
                return roverHealth
            startingPosition[1] = startingPosition[1] + 1
            routeArray[startingPosition[1]][startingPosition[0]] = "*"

        elif roverDirection == 'East':
            if startingPosition[0] == columns:
                print("Position: Boundary Reached ... Command Ignored")
                return
            if routeArray[startingPosition[1]][startingPosition[0] + 1] == '1':
```

```
                print("Collision Detected ... Rover Destroyed")
                roverHealth = 'Dead'
                return roverHealth
            startingPosition[0] = startingPosition[0] + 1
            routeArray[startingPosition[1]][startingPosition[0]] = "*"

        elif roverDirection == 'West':
            if startingPosition[0] == 0:
                print("Position: Boundary Reached ... Command Ignored")
                return
            if routeArray[startingPosition[1]][startingPosition[0] - 1] == '1':
                print("Collision Detected ... Rover Destroyed")
                roverHealth = 'Dead'
                return roverHealth
            startingPosition[0] = startingPosition[0] - 1
            routeArray[startingPosition[1]][startingPosition[0]] = "*"
    except IndexError:
        print("Position: Out of map bounds")

# function to turn the rover left
def left(roverDirection):  1 usage  new *
    if roverDirection == 'North':
        roverDirection = 'West'
```


```
                print("Collision Detected ... Rover Destroyed")
                roverHealth = 'Dead'
                return roverHealth
            startingPosition[0] = startingPosition[0] + 1
            routeArray[startingPosition[1]][startingPosition[0]] = "*"

        elif roverDirection == 'West':
            if startingPosition[0] == 0:
                print("Position: Boundary Reached ... Command Ignored")
                return
            if routeArray[startingPosition[1]][startingPosition[0] - 1] == '1':
                print("Collision Detected ... Rover Destroyed")
                roverHealth = 'Dead'
                return roverHealth
            startingPosition[0] = startingPosition[0] - 1
            routeArray[startingPosition[1]][startingPosition[0]] = "*"
    except IndexError:
        print("Position: Out of map bounds")

# function to turn the rover left
def left(roverDirection):  1 usage  new *
    if roverDirection == 'North':
        roverDirection = 'West'
    elif roverDirection == 'East':
        roverDirection = 'North'
    elif roverDirection == 'South':
        roverDirection = 'East'
    elif roverDirection == 'West':
        roverDirection = 'South'
    return roverDirection
```

```
198
199  # function to turn the rover right
200  def right(roverDirection):  1 usage  new *
201      if roverDirection == 'North':
202          roverDirection = 'East'
203      elif roverDirection == 'East':
204          roverDirection = 'South'
205      elif roverDirection == 'South':
206          roverDirection = 'West'
207      elif roverDirection == 'West':
208          roverDirection = 'North'
209      return roverDirection
210
211
212  if __name__ == '__main__':
213      logging.basicConfig()
214      run()
215
```

Figure 3: *gRPC_client.py code showcasing how the rovers move*

5. Conclusion

In conclusion, this lab was successfully completed. When the server is run, the console outputs the port the server is using. Then once the client is run, it displays the map and prompts the user to enter a rover number. Once the user types in a valid rover number, all the movements for that rover are outputted in thorough detail in the console. Once the rover has completed all its moves, the server then outputs a message stating that the rover is completed. This can be seen in *Figure 4* and *Figure 5*. Overall this lab was completed successfully.

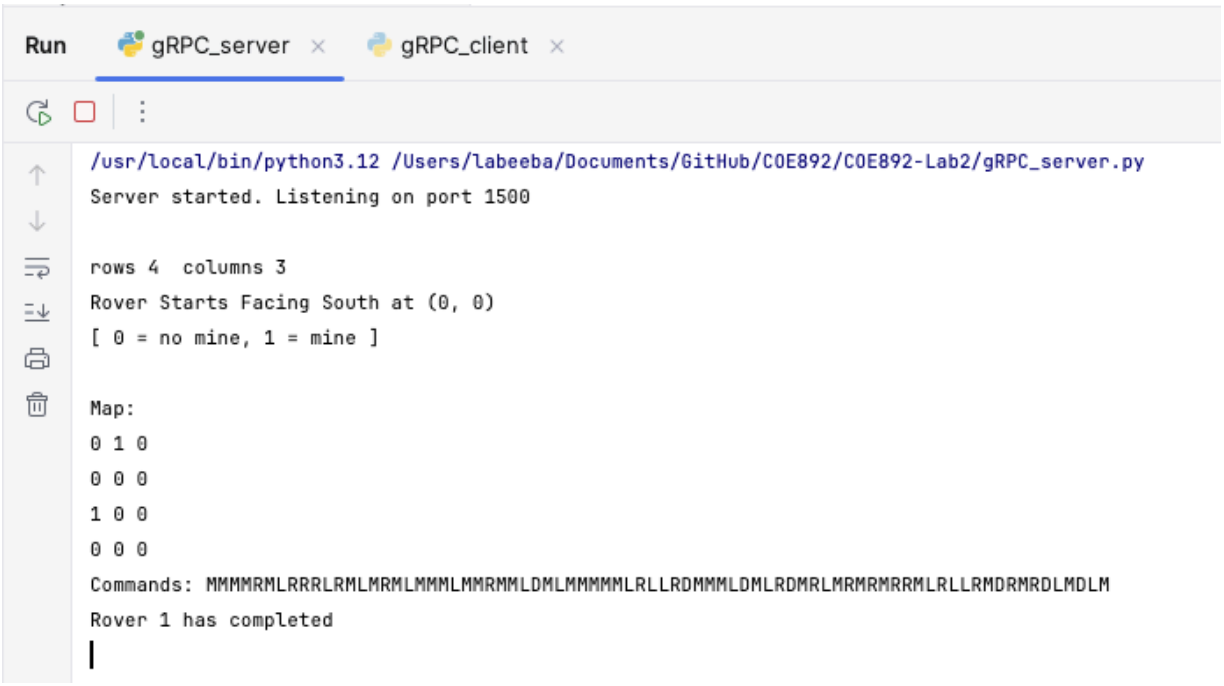
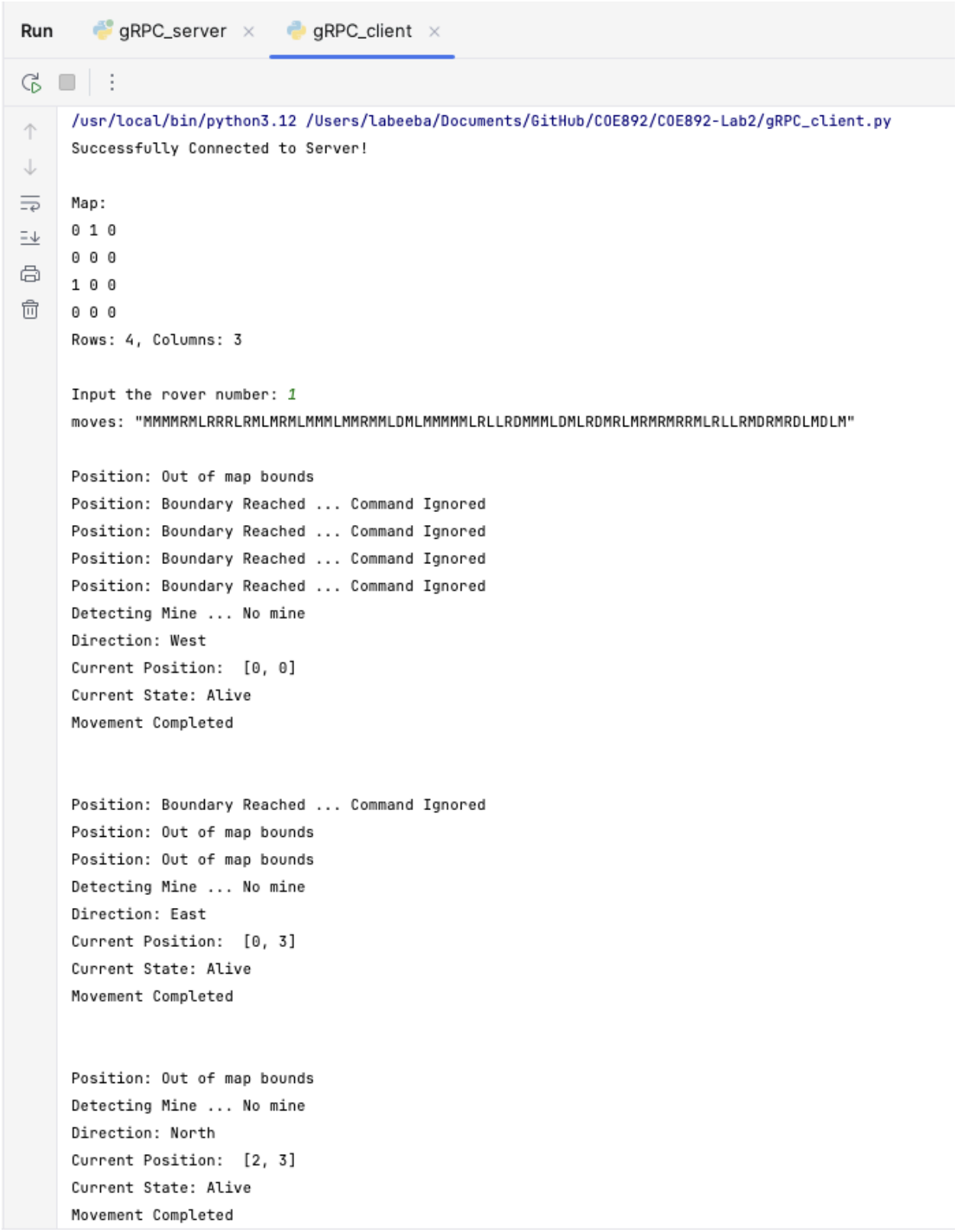


Figure 4: *Server Console*


Run
gRPC_server
gRPC_client
/usr/local/bin/python3.12 /Users/labeeba/Documents/GitHub/COE892/COE892-Lab2/gRPC_client.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"
Position: Out of map bounds
Position: Boundary Reached ... Command Ignored
Position: Boundary Reached ... Command Ignored
Position: Boundary Reached ... Command Ignored
Position: Boundary Reached ... Command Ignored
Detecting Mine ... No mine
Direction: West
Current Position: [0, 0]
Current State: Alive
Movement Completed
Position: Boundary Reached ... Command Ignored
Position: Out of map bounds
Position: Out of map bounds
Detecting Mine ... No mine
Direction: East
Current Position: [0, 3]
Current State: Alive
Movement Completed
Position: Out of map bounds
Detecting Mine ... No mine
Direction: North
Current Position: [2, 3]
Current State: Alive
Movement Completed

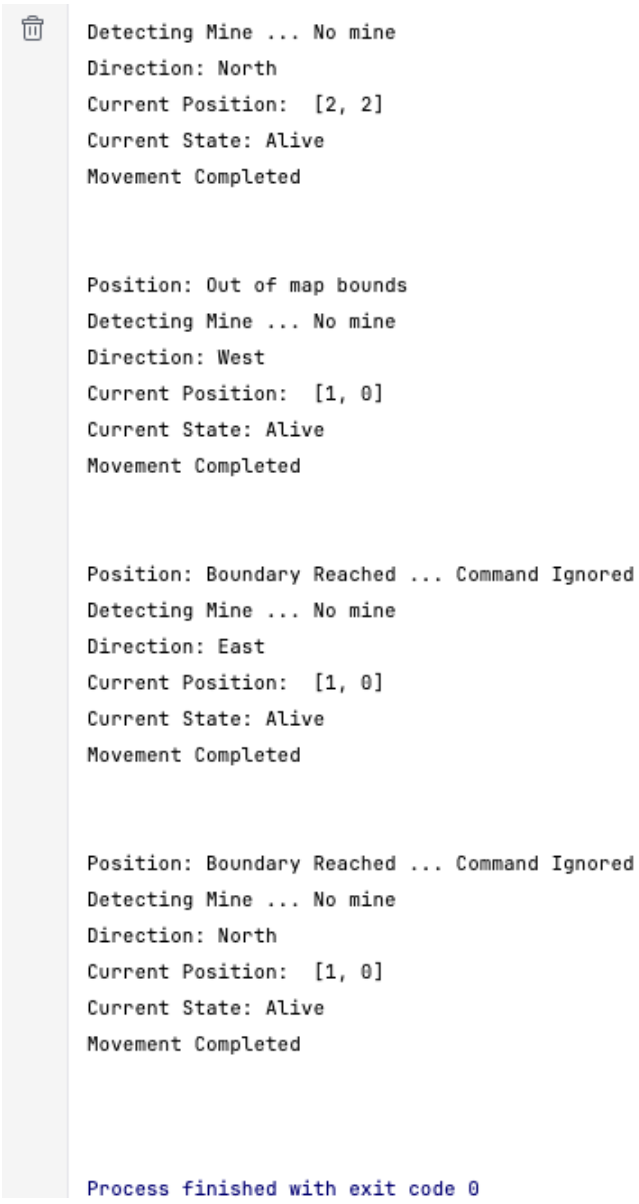


Figure 5: *Client Console*

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.