Problem statement: USING gRPC TO CREATE CLIENT SERVER MODEL TO SIMULATE MISSILE STRIKE ON BATTLEFIELD.

Group Details:

1. BUNTY AGARWAL (2023H1030090P)
2. APURAV DESHMUKH (2023H1030096P)

Overview

In this project, we have developed a simulation of a battlefield scenario where soldiers and a commander utilize a client-server model implemented using gRPC (Google Remote Procedure Call) to simulate missile strikes and the defense actions of soldiers. The simulation includes missile attacks, soldier reactions, casualty tracking, and outcome assessment.

PROGRAM DESIGN:

Client Server Model: The program follows a client-server model communication is facilitated using gRPC, enabling message passing between components.

Components: The program consists of components such as the Commander, soldiers, Battlefield Grid, and Missile Attacks. These components interact to simulate the battlefield scenario.

Why gRPC?

1. A gRPC messages are serialized with ProtoBuf, a light weight message format. A gRPC message is always smaller than an equivalent JSON message.
2. gRPC has both unidirectional and bidirectional streaming support.

Tech Stack:

1. RPC Framework - gRPC
2. Programming Language: Python
3. Version Control: Git

Implementation Details:

Project Structure:

1) `‘server.py’` – **Contains the Global State , Print the 2-D matrix with Battle Field status where it display the information of Soldiers that are alive , Soldiers those are dead , then movement of the soldiers those are under the threat accordingly update the coordinates of the soldiers. If Commander is dead it will select the new commander randomly from the remaining soldier.**

2) `‘client.py’` – **Client.py will take user input for the parameters number of soldier , size of the field , duration of battle and missile interval. It will tell the missile has been launched and lastly show whether the battle has been won or lost.**

3) `‘messages.proto’`- **Defines the message formats using protocol buffers.**

4) `‘output.log’`- **Logs simulation output.**

## Instructions to Run the Code :

Step 1: pip install grpcio

Step 2: pip install grpc-tools

Step 3: pip install colorama (for printing coloured text on terminal)

Command to genarete the protobuf file :

Step 3: python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. soldier.proto

Command to run server and client file (NOTE: Please run Server command before running client command) :

Step 4: python server.py

Step 5: python client.py

After executing the client.py file it will ask for the inputs from the user , example as shown below :

```
bunty@LAPTOP-B1F5FA3J:/mnt/f/grpc/Test$ python3 client.py

Enter number of soldiers : 10

Enter size of the field : 10
```

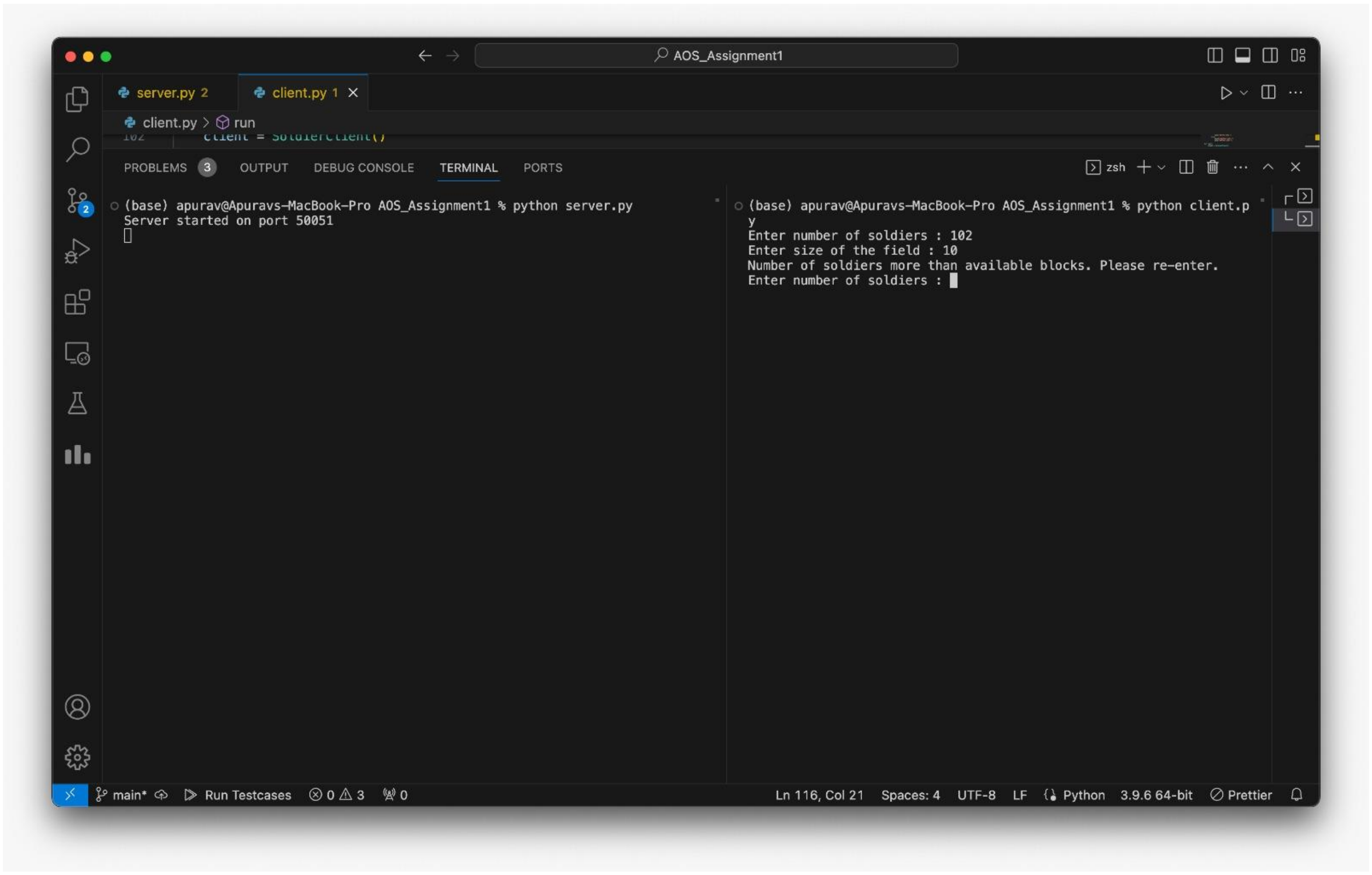
Enter duration of battle T : 30

Enter missile interval t : 5

Now the user can enter the programmable inputs accordingly.

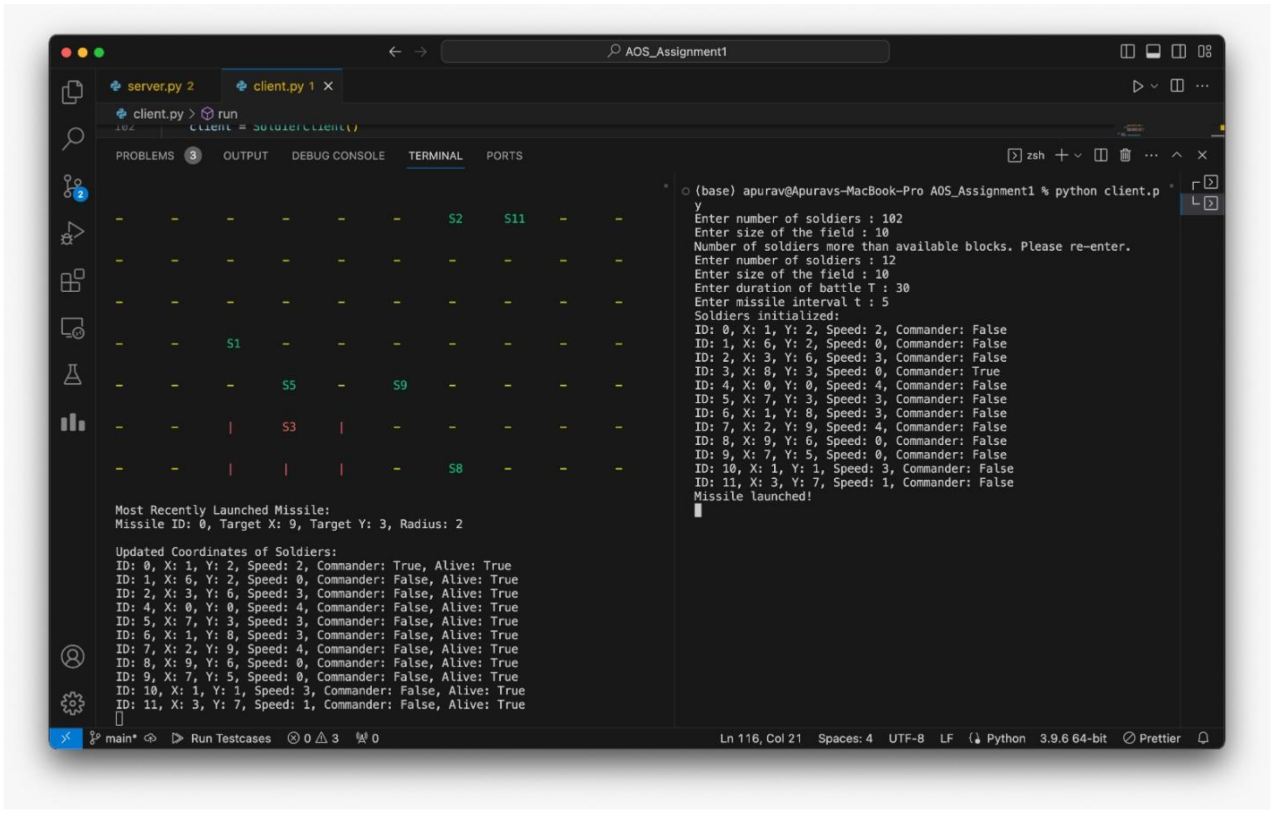
GITHUB LINK: https://github.com/ApuravDeshmukh2309/AOS_Assignment1.git

Below is test case :



```
(base) apurav@Apuravs-MacBook-Pro AOS_Assignment1 % python server.py
Server started on port 50051
```

```
(base) apurav@Apuravs-MacBook-Pro AOS_Assignment1 % python client.p
y
Enter number of soldiers : 102
Enter size of the field : 10
Number of soldiers more than available blocks. Please re-enter.
Enter number of soldiers :
```



```
-    -    -    -    -    -    S2   S11  -    -

-    -    -    -    -    -    -    -    -    -

-    -    -    -    -    -    -    -    -    -

-    -    S1   -    -    -    -    -    -    -

-    -    -    S5   -    S9   -    -    -    -

-    -    |    S3   |    -    -    -    -    -

-    -    |    |    |    -    S8   -    -    -

Most Recently Launched Missile:
Missile ID: 0, Target X: 9, Target Y: 3, Radius: 2

Updated Coordinates of Soldiers:
ID: 0, X: 1, Y: 2, Speed: 2, Commander: True, Alive: True
ID: 1, X: 6, Y: 2, Speed: 0, Commander: False, Alive: True
ID: 2, X: 3, Y: 6, Speed: 3, Commander: False, Alive: True
ID: 4, X: 0, Y: 0, Speed: 4, Commander: False, Alive: True
ID: 5, X: 7, Y: 3, Speed: 3, Commander: False, Alive: True
ID: 6, X: 1, Y: 8, Speed: 3, Commander: False, Alive: True
ID: 7, X: 2, Y: 9, Speed: 4, Commander: False, Alive: True
ID: 8, X: 9, Y: 6, Speed: 0, Commander: False, Alive: True
ID: 9, X: 7, Y: 5, Speed: 0, Commander: False, Alive: True
ID: 10, X: 1, Y: 1, Speed: 3, Commander: False, Alive: True
ID: 11, X: 3, Y: 7, Speed: 1, Commander: False, Alive: True
```

```
(base) apurav@Apuravs-MacBook-Pro AOS_Assignment1 % python client.p
y
Enter number of soldiers : 102
Enter size of the field : 10
Number of soldiers more than available blocks. Please re-enter.
Enter number of soldiers : 12
Enter size of the field : 10
Enter duration of battle T : 30
Enter missile interval t : 5
Soldiers initialized:
ID: 0, X: 1, Y: 2, Speed: 2, Commander: False
ID: 1, X: 6, Y: 2, Speed: 0, Commander: False
ID: 2, X: 3, Y: 6, Speed: 3, Commander: False
ID: 3, X: 8, Y: 3, Speed: 0, Commander: True
ID: 4, X: 0, Y: 0, Speed: 4, Commander: False
ID: 5, X: 7, Y: 3, Speed: 3, Commander: False
ID: 6, X: 1, Y: 8, Speed: 3, Commander: False
ID: 7, X: 2, Y: 9, Speed: 4, Commander: False
ID: 8, X: 9, Y: 6, Speed: 0, Commander: False
ID: 9, X: 7, Y: 5, Speed: 0, Commander: False
ID: 10, X: 1, Y: 1, Speed: 3, Commander: False
ID: 11, X: 3, Y: 7, Speed: 1, Commander: False
Missile launched!
```

Conclusion

This program design document provides a clear overview of the Missile Strike Simulation's architecture, implementation details, and instructions for running the server and clients. By following these instructions, you can effectively run and verify the code.

This design document focuses on running the code by providing detailed instructions for executing the server and client components. Adjust the document as needed to match your specific project structure and requirements.