README FOR TASK ROUND OF KRSSG

DEBANJAN NASKAR, 23IE10015

I would like to express my gratitude towards team KRSSG for letting me work on these amazing, interesting topics throughout my summer break and making these dull evenings enjoyable.

Further I would like expand on each task and primarily how to run them. Libraries I have used will also be given.

TASK1

AIM: The goal was to build a client server architecture through ***multithreaded socket programming*** where any number of prisoners (or clients) could connect to the server and through random guesses get out of the prison (or program)

LANGUAGE: C++

OS: Windows 11

LIBRARIES USED:

- iostream
- WS2tcpip.h
- thread
- cstdlib
- with build dependency of ws2_32.lib (used for ***windows socket programming***)

INSTRUCTIONS:

Here you will find three files - Server, Prisoner Client and Count Client(bonus).

Firstly, with help of Visual Studio first build the solution of Server and then run it using local windows debugger.

Among the two the KRSSGtask1client and the KRSSGCountClient make sure you first run the KRSSGCountClient which is necessary for smooth running of the program.

In the same way with the help of the visual studio first build the solution of the count client and then run it using the local windows debugger.

Next build the solution of the KRSSGtask1client.sln. Now go to the **KRSSGtask1client>x64>Debug>KRSSGtask1client.exe** executable file.

Now ***run the number of clients you have previously entered in the count_client***. Wait for it to end the execution and provide you the escape order.

TASK2

AIM: Implementing a path planning algorithm on an image of 600x600 pixels. I have used the ***RRT* connect*** algorithm for path planning in my code. Next the path is traversed using PID by turtlesim node in ***ROS noetic***.

LANGUAGE: Python

OS: Ubuntu

LIBRARIES USED:

- cv2
- math
- random
- matplotlib
- rospy
- math

INSTRUCTIONS: Implementing a path planning algorithm on an image of 600x600 pixels. I have used the RRT* connect algorithm for path planning in my code. Next the path is traversed using PID by turtlesim node in ROS noetic. Here there are three files: spawnturt.py, goto.py and myconnect.py. Spawnturt.py spawns the turtle at the start position of the path. Goto.py traverse is the whole path coordinates and moves the turtle forward through PID. ***myconnect.py*** is the actual script which contents the whole algorithmic implementation of RRT Star connect.

Do these steps before running myconnect.py:

For running the ROS noetic, open your terminal and run >***roscore.***

Next you can choose to kill the turtle1 because the spawned turtle will be entirely different.

Command to do this: "***rosservice call /kill 'turtle1'***"

Now run myconnect.py. Few tips:

In lines: 13, 42 and 311, exchange the directories of the image you would want to generate a path in. You can ***easily alter the parameters within the first five line of the code***. Increasing the step size decreases the time required for getting the path. Increasing the search radius increases the rewiring radius of the path nodes. Increasing safety leads you to a safer path where it is ensured that no obstacles are there in the way. Ideal safety is half of the step parameter but that takes too much Max_iterations. So suitable step size and safety is given for a smooth path generating of Image1.png.

This runs the whole algorithm and then gives you a path from start to end point. As soon as you close the OpenCV window the turtle should run and the same implication can be seen in matplotlib window. if turtle misbehaves try to ***change parameters like tolerance, or coefficient of linear/angular velocity.*** Suitable parameters are given for a smooth path generation of Image1.png.

TASK3

AIM: Goal of this program was to train a neural network model through ***genetic algorithm*** in ***gym environment API*** provided by OpenAI. Main goal of this task force to design an efficient crossover method and mutation method. Then we had to ***show the fitness graph*** over generations.

OS: Ubuntu

LIBRARIES USED:

- gym
- NumPy
- matplotlib
- random

INSTRUCTIONS:

Only instruction to run this file is if the graph is not optimal, stuck in local maxima, that is does not achieve a good maximum (say 180) then you can run this code again and that should solve the problem. This is due to the ***entire random structure of the algorithm and the code overall.***

TASK4

AIM: Design multiplayer game architecture using ROS communication where each player has three components (monsters) and attacks each other through two modes 1 and 2. Here you will find three files **one for the server, and two for clients A and B respectively**.

OS: Ubuntu

LIBRARIES USED: rospy

INSTRUCTIONS:

Run the roscore

First run the server node.

Next open clients A and B in any order you would like. ***Player A will get chance to input their moves first*** and then Player B. This way the game will go on alternatively until winner is found. Most important precaution while running is game is that when using move 2 you should follow the exact format "2 <opponent name>". Here the ***opponent's name is case sensitive*** and the program will malfunction if you give any wrong inputs.

For any discrepancies, please let me know at debanjannaskar1@gmail.com

or drop a text at +91 9775332900