Simulation guide

Background

This project develops and tests a decentralised replanning system for a drone swarm. The goal is to make sure the mission continues even if some drones fail. In the simulation, the drones search a grid and use a market system to buy or exchange sections. When a drone gets a fault, the tasks are automatically reassigned. Faulty drones will return home, land, or act as relays, and the healthy drones will take over their sections. The simulation also includes collision avoidance, lawn-mower search patterns, dynamic retasking, and a GUI where you can inject faults.

Use

To run the simulation, follow the instructions in the simulation folder in this GitHub repository: https://github.com/MDU-C2/Intelligent-Drone-Swarm.git

Once you run the Python file gui.py, you will see something like Figure 1 below (without all the colouring, of course).

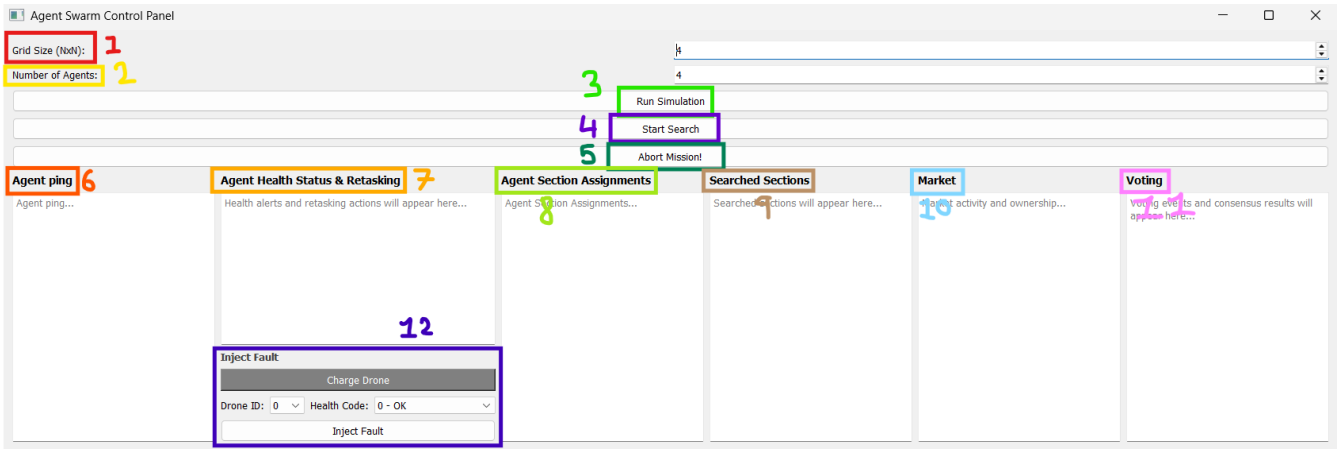

Figure 1: The Graphical User Interface (GUI) created for the drone swarm simulation.

Following the numbering in figure 1:

1. **Grid Size (NxN)**: In this part you can control the size of the search area. So, if you choose 4, then the search area will be a 4x4 grid of 16 sections, where each section is a predefined area. You can select from a 2x2 to a 10x10 search area.
2. **Number of Agents**: This part is where you choose the number of drones/agents. The maximum number of drones you can choose changes with the grid size. The idea is that there should be at least one section for each drone.
3. **Run Simulation**: Once this button is pressed, the PyBullet simulation will open with information of the grid size and number of agents.
4. **Start Search**: Once this button is pressed, the search will begin, and you will also be able to see information directly on the 5 displays (6, 8, 9, 10).

5. **Abort Mission**: This button can be pressed at any time after the search has started. Once pressed, all drones will fly back to the home position (where they started off from).
6. **Agent ping**: This display shows pings (pulses) from all the drones. The agents ping every 5 seconds, and the ping contains information about the position (x, y, z) and the time.
7. **Agent Health Status & Retasking**: This display shows the health status of a drone and the action it will take. Remember that this display only shows information when a fault is injected.
8. **Agent Section Assignments**: This display will show which drone owns which section.
9. **Searched Sections**: This display will show which sections are searched and unsearched.
10. **Market**: This display will show the available sections for sale, the transactions of agents buying the sections, and how many points each drone has.
11. **Voting**: This display will only be shown once an agent has found the subject. After the agent has found the subject, the three closest agents will fly over to confirm, and their decision/vote will be displayed here.
12. **Inject Fault**: This part is where you can inject a fault into a drone. You choose the Drone ID of the drone you want to inject the fault into, then you choose the Health Code which in this case has only 6 faults, 1-6. Once you have chosen the health code, press on the button Inject Fault to inject the fault. The fault will be displayed on the Agent Health Status & Retasking display. The agent will, depending on the fault, return home, emergency land, or fly higher up to be used as a relay. If the fault is “LOW_BATTERY” then the drone will return home and once there, the button “Charge Drone” (it has been changed to Charge Agent now, old image) will turn green and can be pressed to charge the agent. Once charged, it will fly away to the search area to continue searching on its section. But if the charge button is not pressed within a minute, then the section the drone owned will be available in the market for other drones to buy. And if the button is pressed after a minute, then the drone will still fly over to the search area and buy a new available section and start searching.

Structure of the code

The simulation contains a total of 11 python files and uses at least 2 other files from the gym-pybullet-drones that defines the physics and PID movements of the drones. Figure 2 below shows how all the files are connected to the main file. For more information about the code refer to the GitHub repository.

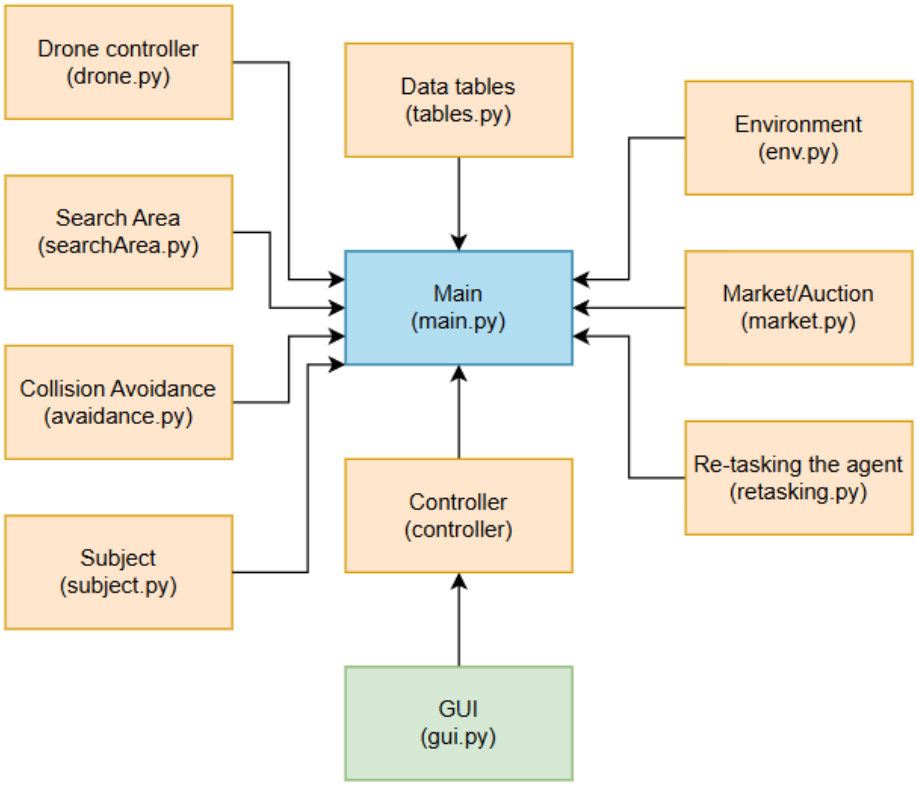

Figure 2: How all the files for the simulation are connected.