# 40.317 Design Exercise

This assignment is due at 11:59 p.m. on Monday 13th July, so you should have sufficient time to complete it *excluding* your Week 7 gap week. Individual submissions are allowed, however this assignment is best completed by a group of up to 4 people. Be sure to list the names of all group members in your submission.

Your task is to reverse engineer the "little language" within the trading game demonstrated in Lecture 1. You will propose a complete set of commands which the server (the "exchange") knows how to handle, along with a reasonably detailed description of each command.

Every request to the server is a one-line string, and every response from the server is also a one-line string (which might have a complicated format if it contains lots of data). You do not need to invent a numeric code for every possible type of error, as we discussed in Lecture 11.

To help ensure your success, you are being provided with the following:

- An appendix listing the categories of required server commands.
- An appendix containing a complete example submission, based on the simple holdings manager application we worked with throughout the first half of the class.
- A screen shot of the folder containing the server's source code, showing the names of the actual Python files which implement the server. Each of these files typically contains the definition of one class, so their names alone provide insight into how the server works.
- Screen shots of the GUI which each team runs.
- Screen shots of the GUI which the game's administrator runs.

*Suggestion:* Begin this assignment by deducing the class variables and methods needed for both order objects and trade objects. You do *not* need to submit your OO designs for these two classes, but you will benefit from figuring them out. Here are the most basic characteristics of these two classes:

- Every order has an Order ID number. Order IDs are completely unique across all teams, i.e. never re-used.
- Similarly, every trade has a completely unique Trade ID number.
- An order has a type, which can be either Buy or Sell.
- An order has a status, which can be one of five possible values: Live, Submitted, Filled, Cancelled, or Cancellable.
- A trade has two orders, "ours" and "theirs".
- A trade has a status, which can be one of three possible values: Unmatched, Matched, or Error.

## Rubric

- Up to 10 points will be deducted for not including commands which are necessary for the game to operate.
- Up to 10 points will be deducted for including commands which are impossible, or not necessary for the game to operate.
    - In particular, *you will lose the full 10 points if you include a server command called "launch_server".* "launch_server" cannot possibly be a part of the server's interface, because the server is not yet running to receive it.
- Up to 15 points will be deducted for insufficient detail in your description of the return values.
    - Your descriptions of the return values must be so detailed that a programmer could use them to write a client.

## Appendix: Trading Game Command Categories

The commands which the server has to support can be grouped into the following categories:

- *Setup*
  Commands which add players and teams before play begins.
- *Informational, across teams*
  Commands which retrieve information about the overall state of the game, i.e. commands someone could use to build a dashboard for all players to see. Such information includes:
    - The status of the market, i.e. open or closed
    - All players and/or all teams
    - The most recent trade price and/or a history of every trade price since the market opened
    - All submitted orders / cancelled orders / filled orders
    - The current "performance metrics" of all teams, e.g. P&L and numbers of matched trades / unmatched trades / error trades
- *Informational, team-specific*
  Commands which retrieve all the information a team might ever need or want to know, such as:
    - Their live orders
    - Their matched / unmatched / error trades
    - Their performance metrics, as listed above
- *Changes of state, across teams*
  Commands which affect the entire game:
    - Open and close the market
    - Clear every team's trade history and P&L in between multiple rounds of trading (if desired)
- *Changes of state, team specific*
    - Cancel a (cancellable) live order
    - Submit half of an agreed trade for matching (the most important command, and the hardest to implement)
- *Miscellaneous*
  Help and shutdown commands.

## Appendix: An Example Submission

Here is a short, complete example based on the simple holdings manager we have been working with throughout the first half of the class. It illustrates the level of detail your submission should contain. For each command, you should provide the following types of information:

- Arguments
- Validations
- Return value
- Internal behaviour

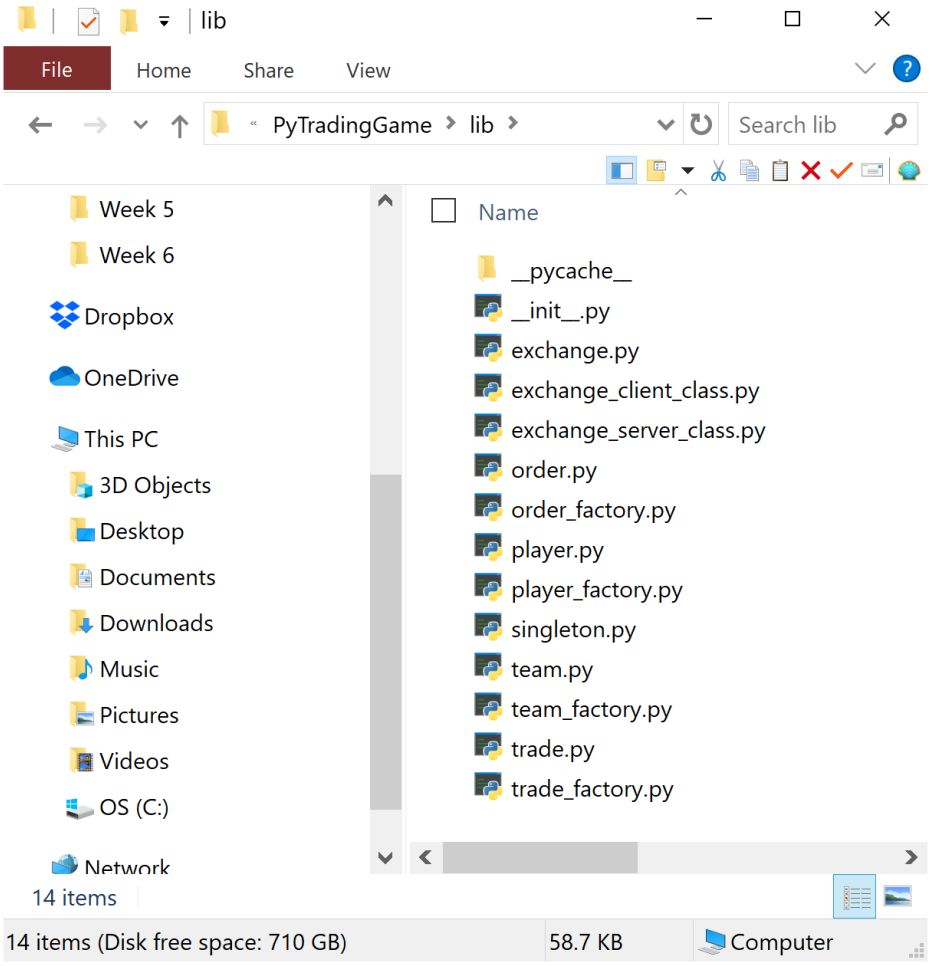
You do not need to list all of these types if some of them are N/A. Also, since the trading game's commands can be grouped into categories, you might want to include the category in a third column.

| Command | Description |
|---|---|
| buy | Arguments:<br>• Number of shares to buy (qty)<br>• Purchase price per share (amt)<br>Validations:<br>• There must be two arguments.<br>• qty must be an integer, and must be positive.<br>• amt must be a number, and must be positive.<br>• qty * amt must be <= amount of cash on hand.<br>Returns: On success, the string "[OK] Purchased". On failure, the string "[ERROR] <msg>", where <msg> is a short description of the error.<br>Behaviour:<br>• Adds qty to the number of shares on hand.<br>• Subtracts qty * amt from the cash on hand. |
| deposit_cash | Arguments: Amount to deposit (amt)<br>Validations:<br>• There must be one argument.<br>• The argument must be a positive number.<br>Returns: On success, the string "[OK] Deposited". On failure, the string "[ERROR] <msg>", where <msg> is a short description of the error.<br>Behaviour:<br>1) Rounds amt to two decimal places.<br>2) Adds the rounded amount to cash on hand. |
| exit, quit | Behaviour: Shuts down the client, leaving the server running. |
| get_cash_balance | Returns: A string of the form "[OK] <n>", where <n> is the current cash balance rounded to two decimal places. |
| get_share_balance | Returns: A string of the form "[OK] <n>", where <n> is the current share balance (an integer). |
| help | Returns: A string of the form "[OK] Supported commands: <cmd1>, <cmd2>, ..., <cmd<n>>", where <cmd<i>> is one of the commands in this table, including "help" itself. |

| | |
|---|---|
| sell | Arguments:<br>• Number of shares to sell (qty)<br>• Sale price per share (amt)<br>Validations:<br>• There must be two arguments.<br>• qty must be an integer, and must be positive.<br>• amt must be a number, and must be positive.<br>• qty must be <= number of shares on hand.<br>Returns: On success, the string "[OK] Sold". On failure, the string "[ERROR] <msg>", where <msg> is a short description of the error.<br>Behaviour:<br>• Subtracts qty from the shares on hand.<br>• Adds qty * amt to the cash on hand. |
| shutdown_server | Behaviour: Sends a shutdown request to the server.<br>Returns: The string "[OK] Server shutting down". |
| <others> | Returns: The string "[ERROR] Unknown command" for any command not listed above. |

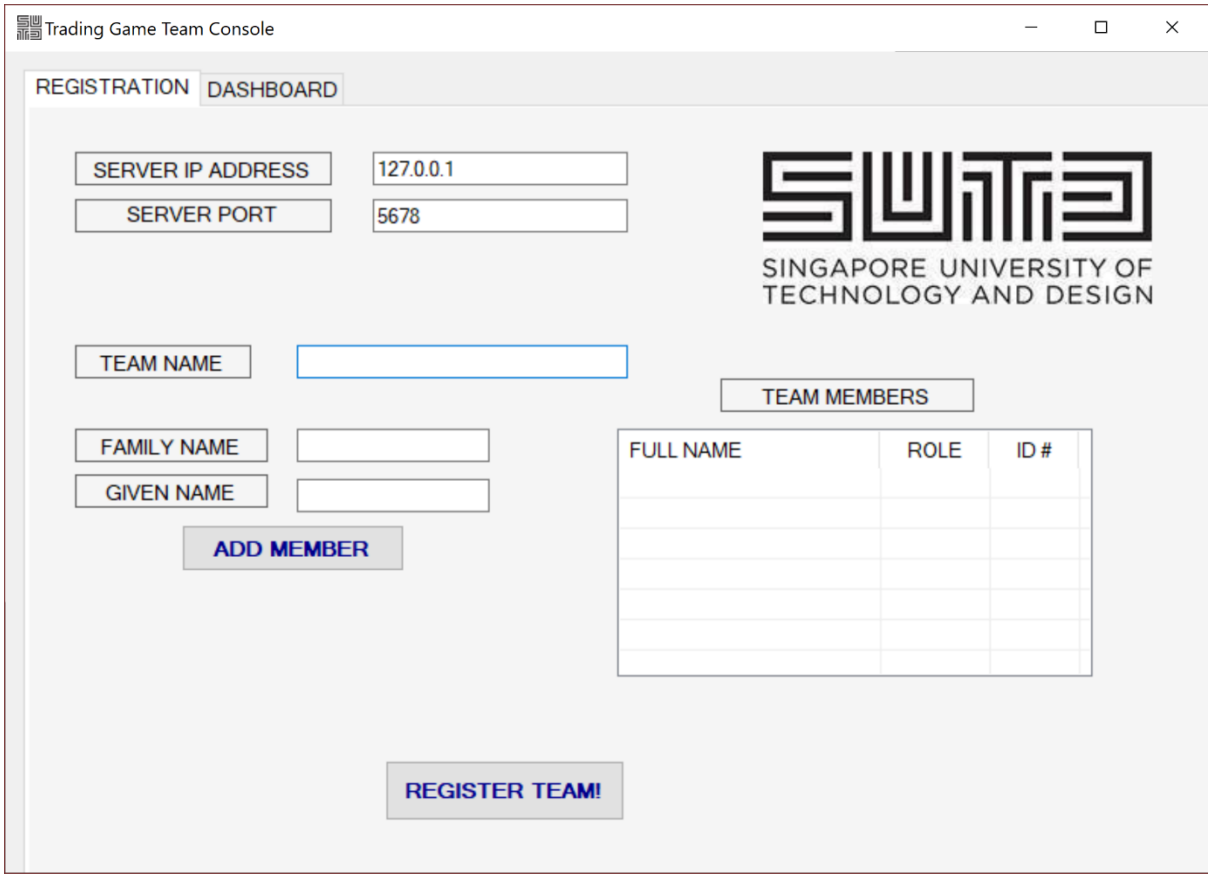
# Appendix: Screen Shots

The contents of the folder containing the server's source code:

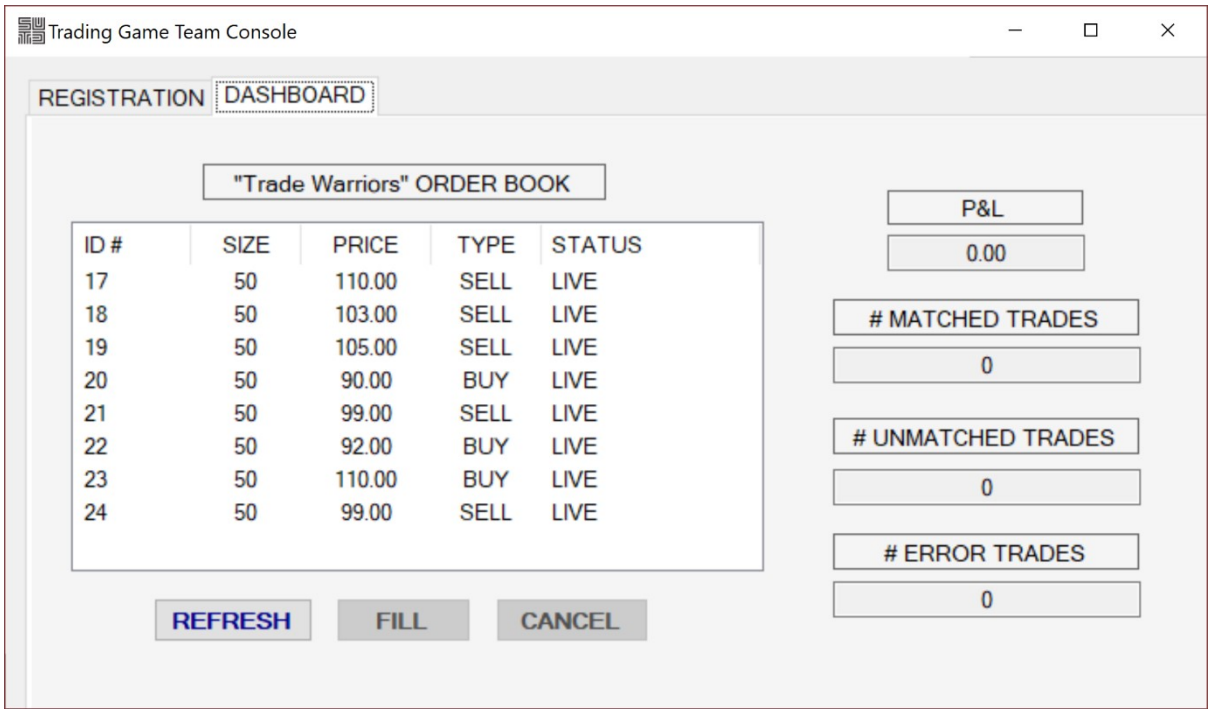


There are two GUIs:  a team GUI run by each team, and an admin GUI run by the game's administrator.

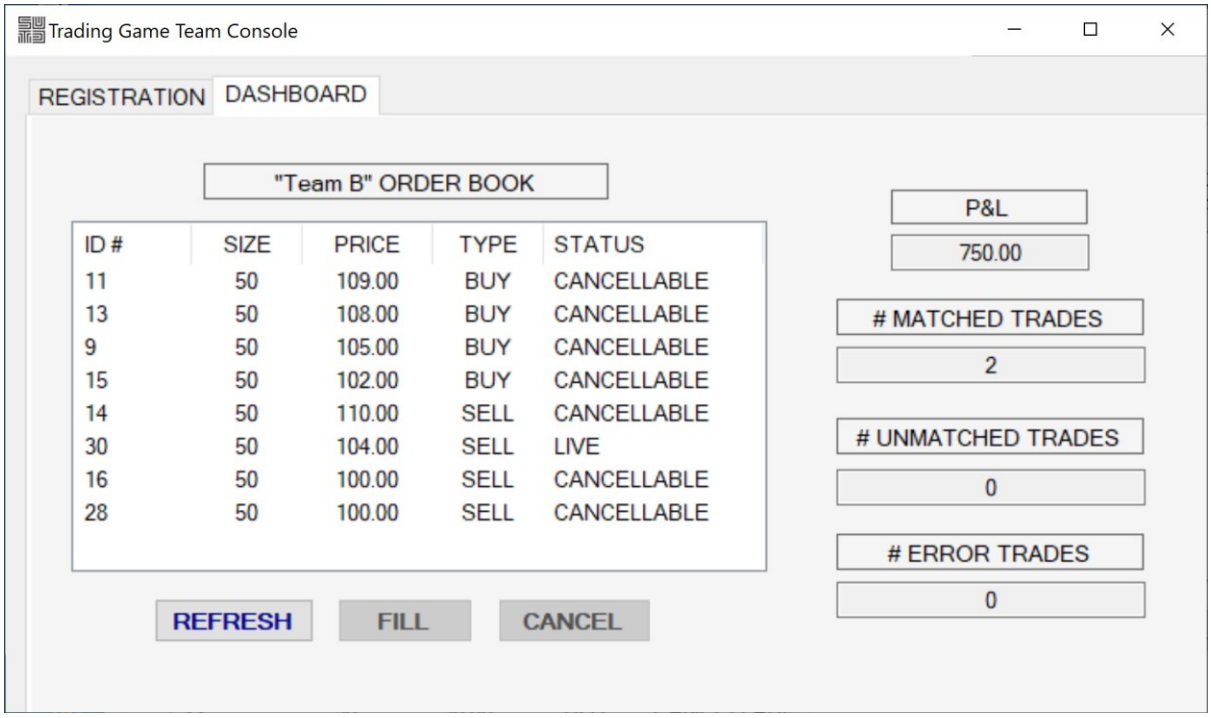
The team GUI in its initial state, i.e. prior to team registration:



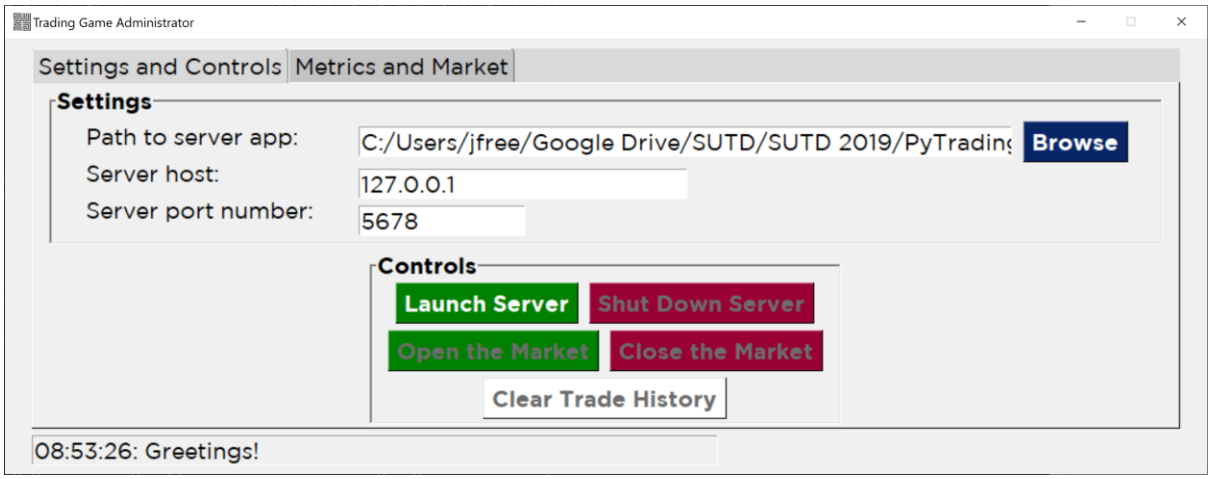

The team GUI after team registration and market opening:



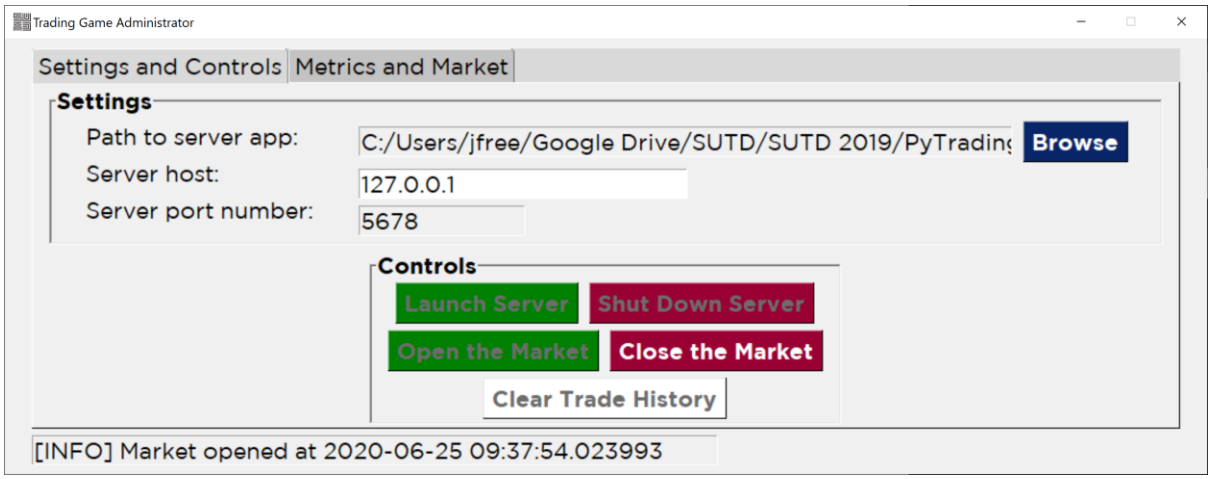
The team GUI after completing a couple of trades with other teams:



The admin GUI in its initial state:




The admin GUI after launching the server and opening the market:




The admin GUI after the teams have completed a few trades with each other: