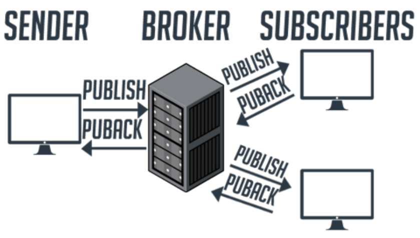
Introduction

The aim of this worksheet is to look at multi-agent coordination where agents “negotiate” for tasks via a shared message system.

Preliminary Exercise

The messaging system we will use is *mqtt*, which is commonly used for Internet of Things applications. It uses a server on the Internet called a *broker*, and agents called *senders* can send messages to that server on particular *channels* (aka *topics*) and other agents called *subscribers* subscribe to those channels to receive messages posted by the senders.

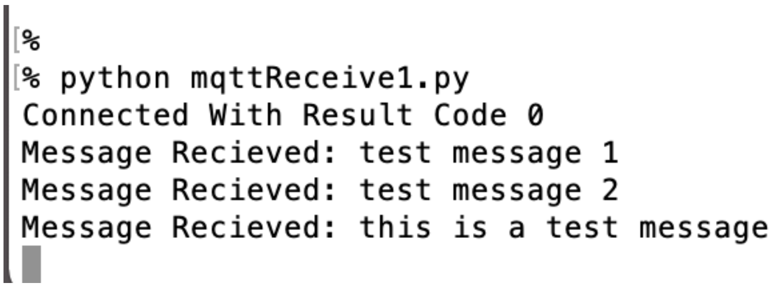


Before we start on this week’s main exercise, let’s do a preliminary piece of work to understand how mqtt works. Download the files *mqttReceive1.py* and *mqttSend.py* from the module Moodle page. You will first need to install the Paho mqtt library:

pip install paho-mqtt

…or however you install packages.

Have a look at *mqttReceive1.py*. This starts up a local mqtt client, connects to the public mqtt broker at test.mosquitto.org, sets up two callback functions that will be called when the client connects to the server (*on_connect*), and when a message is received (*on_message*), and then waits for messages to come in. The channel that we will be using for this class is called *COMP3071*. Run *mqttReceive1.py*. It will print a “connected” message to the console, then print messages as they are posted to the server. If other people are posting messages, you might start to see them arrive:



```
%
% python mqttReceive1.py
Connected With Result Code 0
Message Recieved: test message 1
Message Recieved: test message 2
Message Recieved: this is a test message
```

Now send a message yourself. Whilst leaving *mqttReceive1.py* running, also run *mqttSend1.py*. Check that the message that you send is added to the list of messages that are being received. Try changing your message. Perhaps work with another student to check that they receive the messages that you are sending. When you have finished, stop running *mqttReceive1.py*.

If you want to read more about mqtt, the tutorial at https://mntolia.com/mqtt-python-with-paho-mqtt-client/ is a good starting point.

Main Task

Download *distributedNegotiation.py* from the Moodle page. In the line

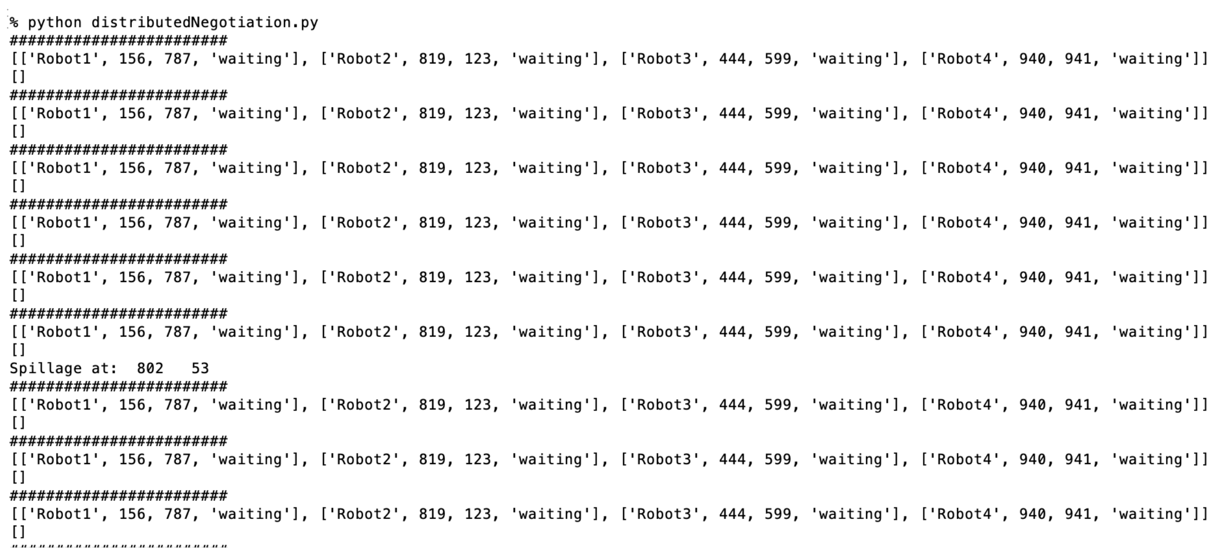
studentNumber = "ID00000000"

change *00000000* to your student number (or, any other arbitrary 8 digits).

The aim of this task is to demonstrate how a number of agents can coordinate by posting message requests to a shared space.

The scenario is that there are a number of robots, which respond to requests to clean up a spillage at a particular location in their space. When a spillage occurs, a request is posted to the mqtt broker, and the robots that are in that space calculate their location to it, and the one that is closest responds and moves to that spillage and takes a while cleaning it up.

Run *distributedNegotiation.py*. Every second, the main loop prints out the list of robots and their status (position, and whether they are waiting, bidding or cleaning), and a list of mqtt messages that have been received since the program started running. After a random period of time, a spillage occurs; at the moment, this is just represented by printing a message. So, you should see something like this:



```
% python distributedNegotiation.py
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
Spillage at:  802   53
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
########################
[['Robot1', 156, 787, 'waiting'], ['Robot2', 819, 123, 'waiting'], ['Robot3', 444, 599, 'waiting'], ['Robot4', 940, 941, 'waiting']]
[]
```

So far, not very interesting. The spillage occurs, but there is no message to say that it has happened, so the robots cannot respond.

Task 1. In the section headed *#does a spillage occur?* add a couple of lines that send a message to the mqtt server saying that a spillage has occurred. This is a string of the format:

cleanupNeeded,x,y,time,location

where x and y are the coordinates of the spillage, time is the current time (use *time.time()*), and location is the string in the variable *myLocation*. "cleanupNeeded" is a literal string. Note that the message is a string, so you will need to convert numbers to strings. So, an example message could be:

"cleanupNeeded,617,747,1649052710.59037,ID12345678"

create a string containing the message, and (by looking at mqttSend1.py for an example) publish it to the topic COMP3071_00000000 (where 00000000 is your student number).

If this is working correctly, you should see your message appear on the message list:

```
########################
[['Robot1', 719, 846, 'waiting'], ['Robot2', 90, 191, 'waiting'], ['Robot3', 276, 456, 'waiting'], ['Robot4', 502, 864, 'waiting']]
[]
Spillage at:  85   828
########################
[['Robot1', 719, 846, 'waiting'], ['Robot2', 90, 191, 'waiting'], ['Robot3', 276, 456, 'waiting'], ['Robot4', 502, 864, 'waiting']]
[['cleanupNeeded', 85.0, 828.0, 1649052859.281256, 'ID00000000']]
########################
```

It will crash on the following timestep, because you haven't added the code yet to handle the message, but don't worry about this.

Task 2. Now, add a couple of lines to the *#bots monitor traffic* section of the code. If a "cleanupNeeded" message is present on the mqtt broker, then any bots that are "waiting" will change their status to "bidding" for this; they will send a message to the server saying how far they are from the spillage. Write code to create and publish this message. This message is of the form:

biddingForCleanup,time-of-spillage,location-of-spillage,distance-to-spillage,robot name

you can get the time of spillage and the location from the current message ("m") and the distance by using the provided function *distance* to calculate the distance between the robot ("r") and the spillage message ("m"). A message will look something like this:

"biddingForCleanup,1649053424.131982,ID00000000,1016.3119599807925,Robot1"

This is a bid from the robot to clean up at that location.

If this is working correctly, you should see something like this:

```
########################
[['Robot1', 779, 937, 'waiting'], ['Robot2', 348, 637, 'waiting'], ['Robot3', 730, 798, 'waiting'], ['Robot4', 866, 861, 'waiting']]
[]
Spillage at:  786   573
########################
[['Robot1', 779, 937, 'waiting'], ['Robot2', 348, 637, 'waiting'], ['Robot3', 730, 798, 'waiting'], ['Robot4', 866, 861, 'waiting']]
[['cleanupNeeded', 786.0, 573.0, 1649053571.089066, 'ID00000000']]
########################
[['Robot1', 779, 937, 'bidding'], ['Robot2', 348, 637, 'bidding'], ['Robot3', 730, 798, 'bidding'], ['Robot4', 866, 861, 'bidding']]
[['cleanupNeeded', 786.0, 573.0, 1649053571.089066, 'ID00000000'],
 ['biddingForCleanup', 1649053571.089066, 'ID00000000', 364.06730147048364, 'Robot1'],
 ['biddingForCleanup', 1649053571.089066, 'ID00000000', 442.6511041441103, 'Robot2'],
 ['biddingForCleanup', 1649053571.089066, 'ID00000000', 231.8641843838759, 'Robot3'],
 ['biddingForCleanup', 1649053571.089066, 'ID00000000', 298.9046670763105, 'Robot4']]
```

i.e. the bids are now on the mqtt message list.

Task 3. Finally, we will add messages so that the robot that is nearest accepts the clean-up job, and the others withdrawn their bid. The code for this is in *#bots choose whether to take bid*. There is some code that compares the distance of each currently "bidding" bot to other "bidding" bots. The one that is closest accepts the bid, moves to the location, and begins cleaning. This is done by sending a message:

claimingBid,time-of-cleanup-request,location-id-of-cleanup-request,robot-name

e.g.

"claimingBid,1649054692.799127,ID00000000,Robot3"

This robot sets its status to "cleaning" and sets a time to finish the job.

The robots that are further away withdraw their bids by sending a similar message:

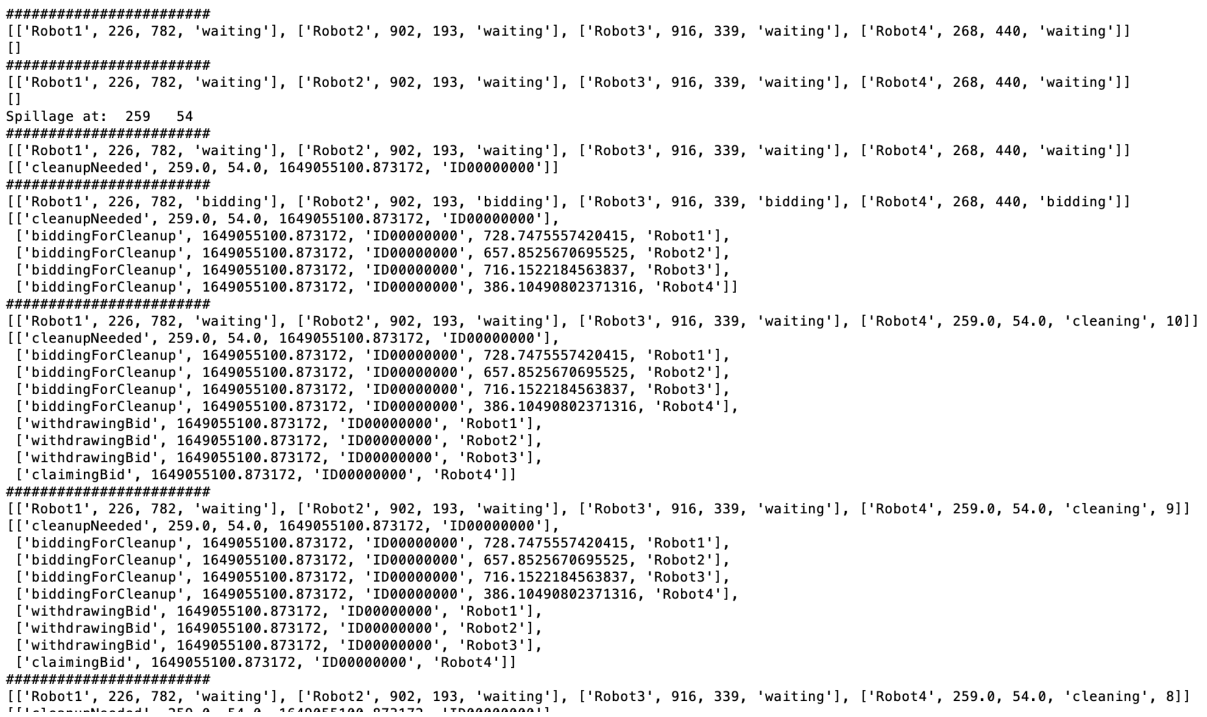
withdrawingBid,time-of-cleanup-request,location-id-of-cleanup-request,robot-name

e.g.

"withdrawingBid,1649054935.244881,ID00000000,Robot4"

and set their status back to "waiting".

If this is all working, you should see a series of messages like this:



```
########################
[['Robot1', 226, 782, 'waiting'], ['Robot2', 902, 193, 'waiting'], ['Robot3', 916, 339, 'waiting'], ['Robot4', 268, 440, 'waiting']]
[]
########################
[['Robot1', 226, 782, 'waiting'], ['Robot2', 902, 193, 'waiting'], ['Robot3', 916, 339, 'waiting'], ['Robot4', 268, 440, 'waiting']]
[]
Spillage at:  259   54
########################
[['Robot1', 226, 782, 'waiting'], ['Robot2', 902, 193, 'waiting'], ['Robot3', 916, 339, 'waiting'], ['Robot4', 268, 440, 'waiting']]
[['cleanupNeeded', 259.0, 54.0, 1649055100.873172, 'ID00000000']]
########################
[['Robot1', 226, 782, 'bidding'], ['Robot2', 902, 193, 'bidding'], ['Robot3', 916, 339, 'bidding'], ['Robot4', 268, 440, 'bidding']]
[['cleanupNeeded', 259.0, 54.0, 1649055100.873172, 'ID00000000'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 728.7475557420415, 'Robot1'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 657.8525670695525, 'Robot2'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 716.1522184563837, 'Robot3'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 386.10490802371316, 'Robot4']]
########################
[['Robot1', 226, 782, 'waiting'], ['Robot2', 902, 193, 'waiting'], ['Robot3', 916, 339, 'waiting'], ['Robot4', 259.0, 54.0, 'cleaning', 10]]
[['cleanupNeeded', 259.0, 54.0, 1649055100.873172, 'ID00000000'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 728.7475557420415, 'Robot1'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 657.8525670695525, 'Robot2'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 716.1522184563837, 'Robot3'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 386.10490802371316, 'Robot4'],
 ['withdrawingBid', 1649055100.873172, 'ID00000000', 'Robot1'],
 ['withdrawingBid', 1649055100.873172, 'ID00000000', 'Robot2'],
 ['withdrawingBid', 1649055100.873172, 'ID00000000', 'Robot3'],
 ['claimingBid', 1649055100.873172, 'ID00000000', 'Robot4']]
########################
[['Robot1', 226, 782, 'waiting'], ['Robot2', 902, 193, 'waiting'], ['Robot3', 916, 339, 'waiting'], ['Robot4', 259.0, 54.0, 'cleaning', 9]]
[['cleanupNeeded', 259.0, 54.0, 1649055100.873172, 'ID00000000'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 728.7475557420415, 'Robot1'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 657.8525670695525, 'Robot2'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 716.1522184563837, 'Robot3'],
 ['biddingForCleanup', 1649055100.873172, 'ID00000000', 386.10490802371316, 'Robot4'],
 ['withdrawingBid', 1649055100.873172, 'ID00000000', 'Robot1'],
 ['withdrawingBid', 1649055100.873172, 'ID00000000', 'Robot2'],
 ['withdrawingBid', 1649055100.873172, 'ID00000000', 'Robot3'],
 ['claimingBid', 1649055100.873172, 'ID00000000', 'Robot4']]
########################
[['Robot1', 226, 782, 'waiting'], ['Robot2', 902, 193, 'waiting'], ['Robot3', 916, 339, 'waiting'], ['Robot4', 259.0, 54.0, 'cleaning', 8]]
```

Finally, note that all of this is done in a distributed way. For simplicity, the bots draw their messages from a single list (mqttMessageList) but in theory, each bot could be operating on a separate computer and pull down the messages from the server separately.

Extensions

1. (simple) Add code so that a bot will not bid if it is over a certain distance from the spillage.

2. Add code so that multiple spillages can occur. Rather than having the countdown to a single spillage, create spillages at random times and add messages to the list asking for help with cleaning them. Robots that are currently cleaning will not be “waiting” and so will not bid for them. After they have finished waiting, they should be able to pick up any currently unclaimed requests. The main difficulty here is modifying the code in *#bots choose whether to take bid*, which assumes that there will be only one active spillage at any one time.

3. (difficult) Coordinate messages between different computers. Firstly, have robots at different positions but in with the same “location” variable run on different machines. Secondly, have each computer run a different “location”, and if a particular location has a large backlog of cleanupNeeded messages, post a new kind of message called *helpNeeded*, where robots from other locations move to the one where there is demand.

4. (very difficult) Integrate this with the robot vacuum simulation from the earlier classes.

Doing extensions 3 and 4 would make a good coursework project.