Assessment 3
Question 1

Software Agents and Multi-Agent Systems

CS551K

Dr. Wamberto Vasconcelos

Yassin Dinana

52094480

16th February 2021

one day extension

Question 1:

A- Agents interaction is an important aspect of multi-agent systems. The best way to easily understand the communication between agents is using the AUML diagram, which is based on the Unified Modelling Language, the diagram represents and explains the system as well as its agents, roles, and different actions.

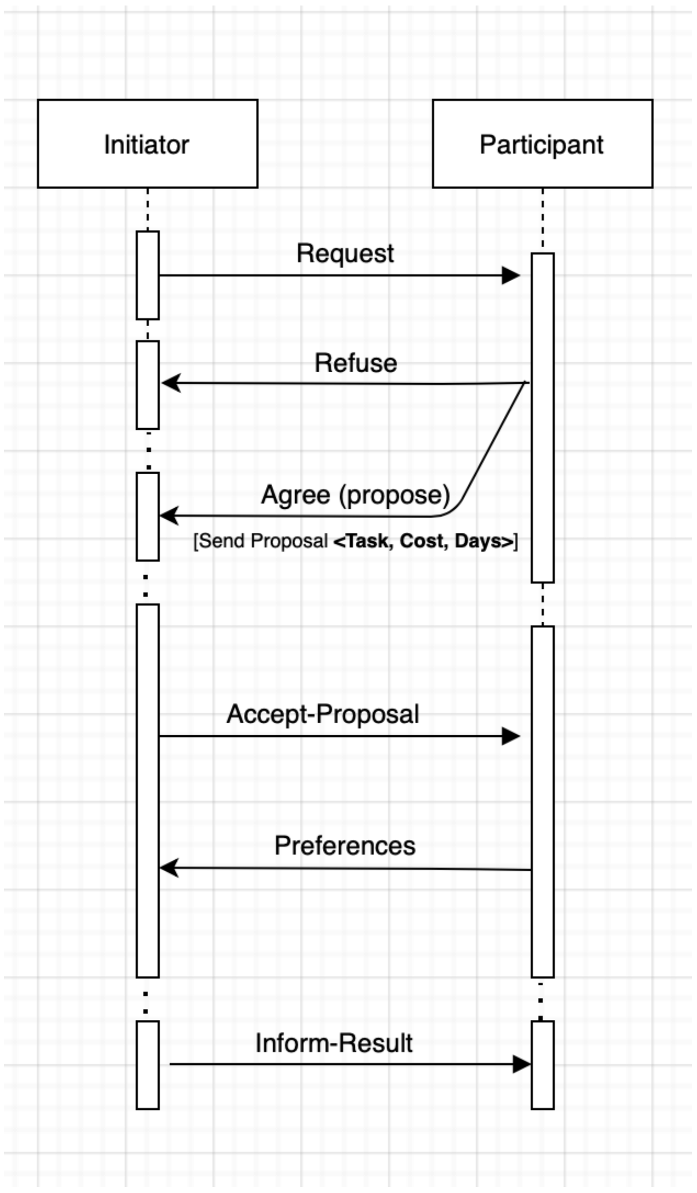


Figure 1: AUML diagram

Looking at the figure 1 above, it shows the AUML diagram represented for the given system. It can be seen that there are two different actors, presented as the Initiator and the Participant. The AUML diagram above, describes all the actions that the system interprets. There is only one initiator and the participant class represent 12 of them. The Initiator start off by sending a request for proposals to the participants as a first step, the participants then select to either accept the request or refuse it; if the participants decide to accept the request, each participant send his proposals in a form of (Task, Cost, Days).

After the initiator receives all the proposals, it ignores all the refused ones and accepts all the received ones by informing that is has accepted all the proposals and the initiator sends back all the proposals to the participants for voting.

The participants have now received all the proposals and start their voting, each participant then sends back his preferences to the initiator. Finally, after the initiator has received all the preferences, it computes the winners and losers then informs all the participants with the final result.

B-

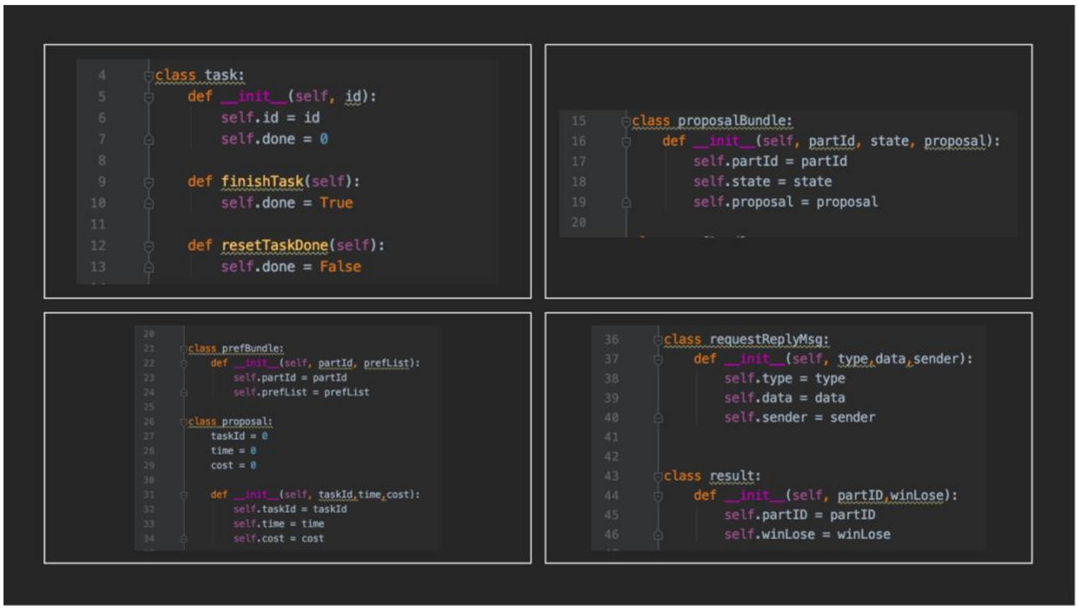
```
procedure Initiator(myID, ParticipantsIDs, TaskID, VotingTechnique)
    Initialise Request # Request Call For Proposal (message)
    Initialise ReceivedProposals
    ParticipantCount = 0 # Empty List equals to zero.
    for P ∈ Participants:
        send(myID, P, RequestForProposal)
    end for
    while ParticipantCount < Participants do:
        receive(participantID , myID, Proposal) #Receive Proposals
        PCount = PCount + 1 #Increase the count by +1
        if Proposal = refuse then:
            null            #Ignore if reply is refused.
        else:
            Proposals 🡐 Proposals ∪{<P, Reply>} # Store reply in proposals.
            AllReceivedProposals = AllReceivedProposals ∪ ProposalAndID
        end if
    end while
    for P ∈ ParticipantsIDs:
        send(myID, P, Proposals)
    end for
    ProCount = 0
    while ProCount < NumberOf(Proposals) do:
        receiveB(participantID , myID, Preferences)
        PCount = PCount + 1
        PreferencesAndIDs = (Preferences, participantID)
        VotingBox = VotingBox ∪ ProposalsPreferences # append vote to ballotbox
    end while
    [winnersID, losersIDs] = TallyTheVote(VotingBox, VotingTechnique)
    send(myID, winnerID, inform: win). #Inform results - winners
    NumberOfLosers = 0
    for Loser ∈ losersIDs :
        send(myID, Loser, inform: losers) #Inform results - Losers
    end for
```

Yassin Dinana
52094480

C-

```
1: procedure Participant(myID, Initiator, request, ReadyMadeProposals,
SocialChoiceFunction)

2: Proposal = SelectProposal(ReadyMadeProposals)

3: while True do:

4:      receive(InitiatorID, myID, request)

5: case-of: Message

6: inform:winner

7:      execute(TaskID)

8:      send(myID, InitiatorID, ‘inform:done’)

9:      inform:rejected

11: RequestForProposal (Task)

12: send(myID, InitiatorID, Proposal)

13:     AllReceivedProposals

14: AllVotes = AllVotes_Proposals (AllReceivedProposals, SocialChoiceFunction

15: send(myID, InitiatorID, Votes)

16:                   end case-of

17:             end while

18:     end procedure
```

D- This part of the question will include the Python implementation of the system using the round-robin fashion. The section includes code snippets along with explanation of the program and the outputs.



```
class task:
    def __init__(self, id):
        self.id = id
        self.done = 0

    def finishTask(self):
        self.done = True

    def resetTaskDone(self):
        self.done = False
```

```
class proposalBundle:
    def __init__(self, partId, state, proposal):
        self.partId = partId
        self.state = state
        self.proposal = proposal
```

```
class prefBundle:
    def __init__(self, partId, prefList):
        self.partId = partId
        self.prefList = prefList

class proposal:
    taskId = 0
    time = 0
    cost = 0

    def __init__(self, taskId,time,cost):
        self.taskId = taskId
        self.time = time
        self.cost = cost
```

```
class requestReplyMsg:
    def __init__(self, type,data,sender):
        self.type = type
        self.data = data
        self.sender = sender


class result:
    def __init__(self, partID,winLose):
        self.partID = partID
        self.winLose = winLose
```

Figure 2: Initializing Classes

Looking at figure 2 above, different classes are initialized in the beginning of the program which defines actions of the system. The first-class *Task*, takes parameter "ID", which defines that each task sent by the initiator to the participant will have an ID. Two functions are defined in the class where they will be used to flag when a task is done and to reset it.

The second class presented is known as *ProposalBundle*, where it includes the Participants Ids, as the system includes 12 participants where each participant will have a different ID from 0 to 11. It also takes the state of the reply from the participant to the initiator which will show if the request for proposal is accepted or refused. Finally, it includes the proposal itself if the state is accepted.

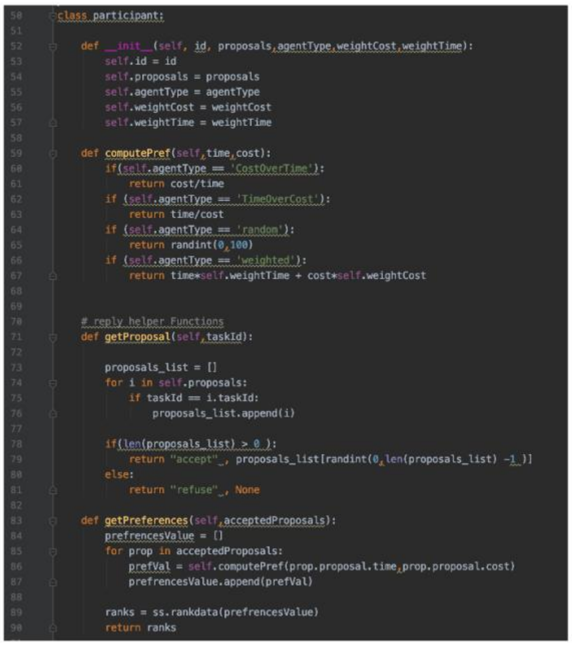
As the participants will have to send their preferences to the lord initiator, the class *prefBundle* will be responsible for the preferences where it includes the parameters ParticipantID in order to know which participant sent the proposal and the PreferenceList including all preferences.

The proposals come in a form of (TaskID, Days/Time, Cost). The class *Proposals* is responsible for the form of the proposals where the taskID, time, and cost are all initialized to zero.

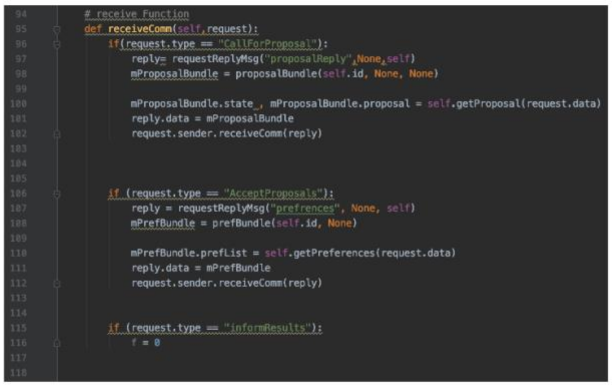
The given class *requestReplyMsg*, represents the state of the message, it shows the message type which definies how the proposal was computed using weight, cost over time, time over cost, or randomly. It also shows the data, which is the proposal, and finally it shows who is the sender.

The final class initialized represents the final result, it states the participantsID and defines the winner and losers' scenarios.

1- ***Participants Class:***



```python
class participant:

    def __init__(self, id, proposals,agentType,weightCost,weightTime):
        self.id = id
        self.proposals = proposals
        self.agentType = agentType
        self.weightCost = weightCost
        self.weightTime = weightTime

    def computePref(self,time,cost):
        if(self.agentType == 'CostOverTime'):
            return cost/time
        if (self.agentType == 'TimeOverCost'):
            return time/cost
        if (self.agentType == 'random'):
            return randint(0,100)
        if (self.agentType == 'weighted'):
            return time*self.weightTime + cost*self.weightCost


    # reply helper Functions
    def getProposal(self,taskId):

        proposals_list = []
        for i in self.proposals:
            if taskId == i.taskId:
                proposals_list.append(i)

        if(len(proposals_list) > 0 ):
            return "accept" , proposals_list[randint(0,len(proposals_list) -1 )]
        else:
            return "refuse" , None

    def getPreferences(self,acceptedProposals):
        prefrencesValue = []
        for prop in acceptedProposals:
            prefVal = self.computePref(prop.proposal.time,prop.proposal.cost)
            prefrencesValue.append(prefVal)

        ranks = ss.rankdata(prefrencesValue)
        return ranks
```



```python
    # receive Function
    def receiveComm(self,request):
        if(request.type == "CallForProposal"):
            reply= requestReplyMsg("proposalReply",None,self)
            mProposalBundle = proposalBundle(self.id, None, None)

            mProposalBundle.state , mProposalBundle.proposal = self.getProposal(request.data)
            reply.data = mProposalBundle
            request.sender.receiveComm(reply)



        if (request.type == "AcceptProposals"):
            reply = requestReplyMsg("prefrences", None, self)
            mPrefBundle = prefBundle(self.id, None)

            mPrefBundle.prefList = self.getPreferences(request.data)
            reply.data = mPrefBundle
            request.sender.receiveComm(reply)


        if (request.type == "informResults"):
            f = 0

```

Figure 3: Participants Class

Figure 3 above shows the participants class, where the class takes different attributes which will be used later on such as the participantID, proposals, agentType, weightCost, and weightTime, all of those attributes are used to get the proposals and compute the preferences.

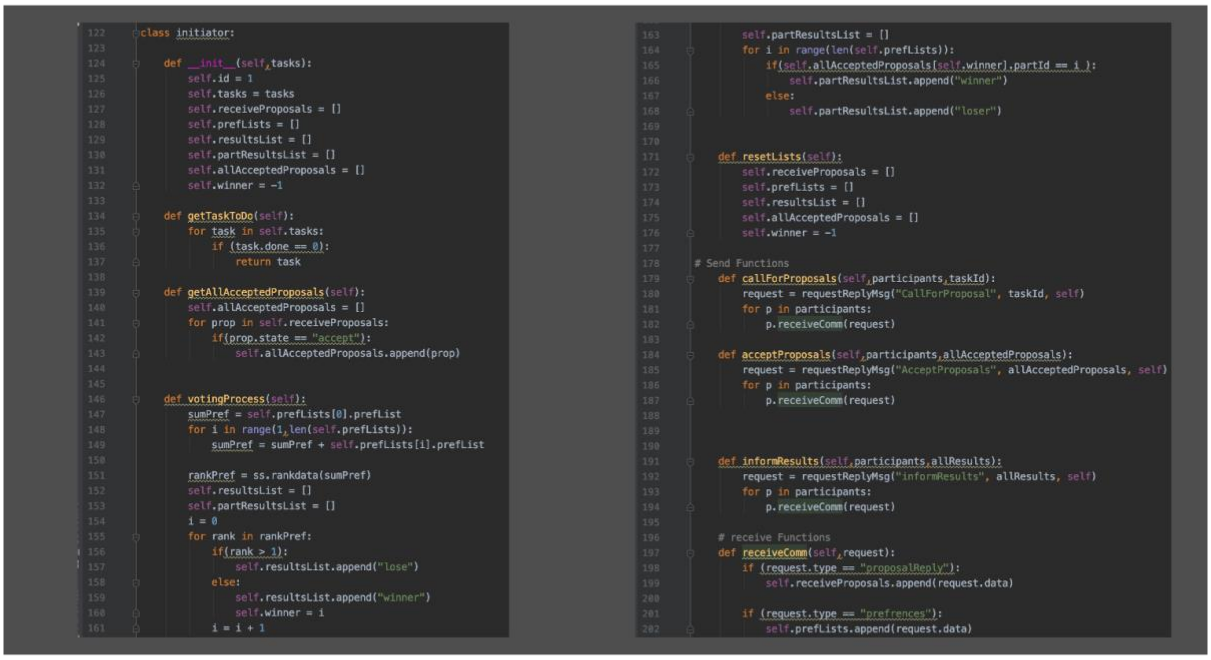
By looking at the first function in the class known as computePref is used to compute the participants preferences using different methods such as Cost over Time, Time over Cost, randomly, and weights. Where the cost over time returns the division equation of dividing the cost over the amount of days (time), same for the time over cost. When it comes to computing the preference randomly, this is done using the Random library in Python.

In order for the participants to get the proposals, an empty list is created calling the proposals function defined in figure 2. A for loop is then created to check if the proposal has been accepted, if yes, then store / append it in the proposals list, if not, it refuses it. It then checks if the proposals list is > 0 (meaning proposals can be found in the list) then it returns accept, if the list is zero, it returns None.

The participants then get all the proposals and ranks them in the getPreference function. Where all the preference values are ranked using the SciPy Stats Python library, it ranks all the days and cost.

Looking at figure 3, in the right image, the function receiveComm is created to build the communication between the participant and the initiator, where it can be seen the request "Call for Proposal" is built, this function uses the proposal bundle function in figure 2, where it requests the CFP and received a reply with the sender and data info. The same function all states that if the request time is “Accepted” then the participant will reply with the preference by sending it to the initiator.

2- ***Initiator Class:***



```
class initiator:

    def __init__(self,tasks):
        self.id = 1
        self.tasks = tasks
        self.receiveProposals = []
        self.prefLists = []
        self.resultsList = []
        self.partResultsList = []
        self.allAcceptedProposals = []
        self.winner = -1

    def getTaskToDo(self):
        for task in self.tasks:
            if (task.done == 0):
                return task

    def getAllAcceptedProposals(self):
        self.allAcceptedProposals = []
        for prop in self.receiveProposals:
            if(prop.state == "accept"):
                self.allAcceptedProposals.append(prop)


    def votingProcess(self):
        sumPref = self.prefLists[0].prefList
        for i in range(1,len(self.prefLists)):
            sumPref = sumPref + self.prefLists[i].prefList

        rankPref = ss.rankdata(sumPref)
        self.resultsList = []
        self.partResultsList = []
        i = 0
        for rank in rankPref:
            if(rank > 1):
                self.resultsList.append("lose")
            else:
                self.resultsList.append("winner")
                self.winner = i
            i = i + 1
```

```
        self.partResultsList = []
        for i in range(len(self.prefLists)):
            if(self.allAcceptedProposals[self.winner].partId == i ):
                self.partResultsList.append("winner")
            else:
                self.partResultsList.append("loser")


    def resetLists(self):
        self.receiveProposals = []
        self.prefLists = []
        self.resultsList = []
        self.allAcceptedProposals = []
        self.winner = -1

# Send Functions
    def callForProposals(self,participants,taskId):
        request = requestReplyMsg("CallForProposal", taskId, self)
        for p in participants:
            p.receiveComm(request)

    def acceptProposals(self,participants,allAcceptedProposals):
        request = requestReplyMsg("AcceptProposals", allAcceptedProposals, self)
        for p in participants:
            p.receiveComm(request)



    def informResults(self,participants,allResults):
        request = requestReplyMsg("informResults", allResults, self)
        for p in participants:
            p.receiveComm(request)

    # receive Functions
    def receiveComm(self,request):
        if (request.type == "proposalReply"):
            self.receiveProposals.append(request.data)

        if (request.type == "prefrences"):
            self.prefLists.append(request.data)
```

Figure 4: Initiator Class

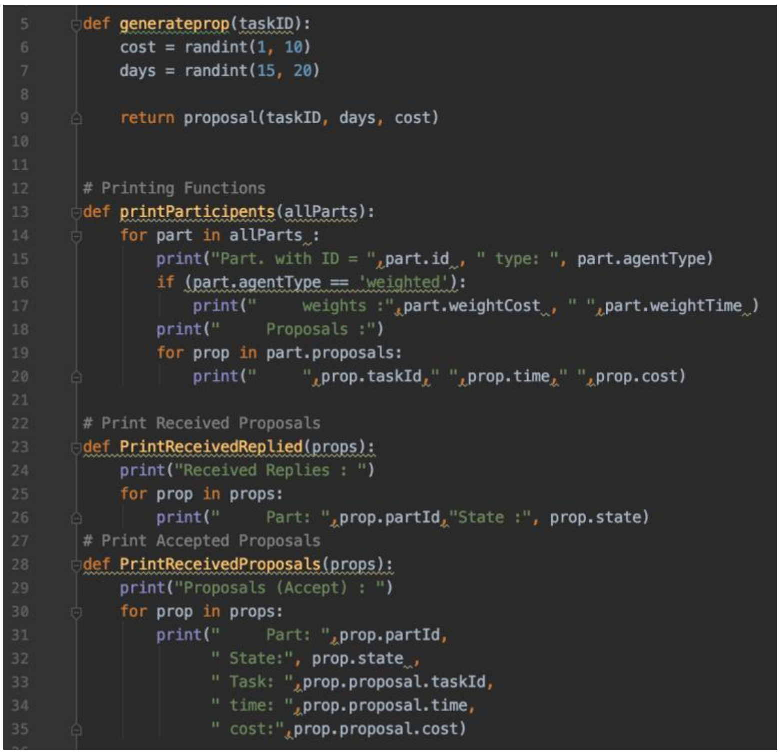
Figure 4 above represents the initiator class, where it also includes different functions such as get all accepted proposals, where a for loop is created to receive all proposals, as stated before, the state of the proposals is either “Accepted” or “Refused” by the participants, if the state is accepted, the created list is appended with the accepted proposal. If the proposal is refused, it is ignored.

After all the accepted proposals are gathered and the preferences are received from the participants, the preferences are ranked using the SciPy stats library, where the preferences with the lowest cost and days will be presented as the winner, all others will be presented as losers in the output console. A function called resetLists is then created where is it used to reset the lists after the results have been informed.

As this is the initiator class, as it can be seen in figure 4, it includes different send and receive functions. Where the send functions include calling for proposals, accepting the proposals, and informing the results.

The receive function include the proposal reply (whether the proposal is accepted or refused) and the preferences sent by the participants and received by the initiator.

E- A main.py file is created, where it calls the python file including all the classes and it includes the print function as well as generate the output.



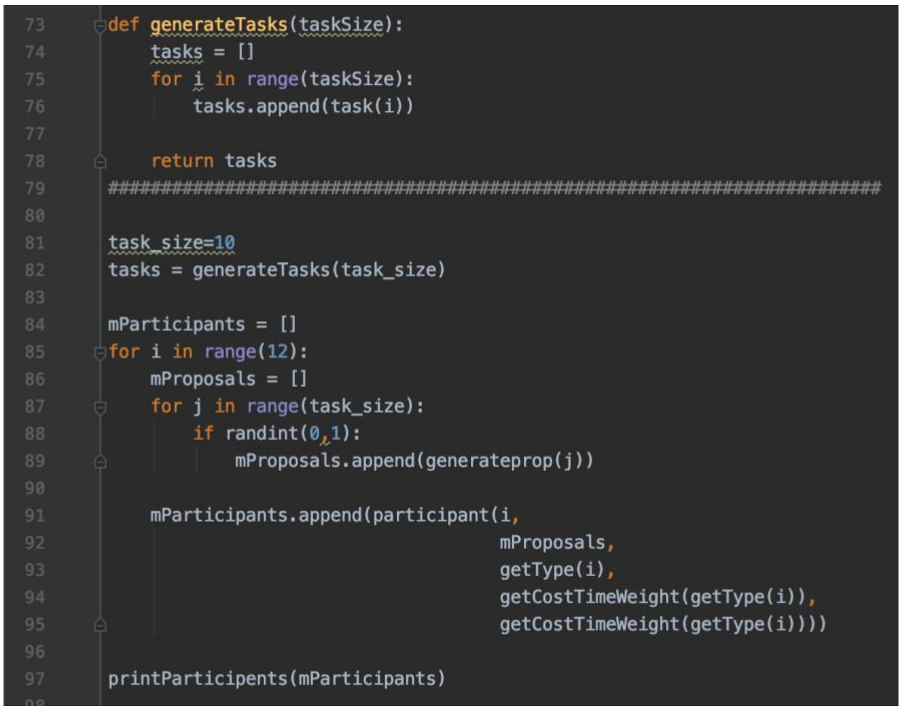
```
def generateprop(taskID):
    cost = randint(1, 10)
    days = randint(15, 20)

    return proposal(taskID, days, cost)


# Printing Functions
def printParticipents(allParts):
    for part in allParts :
        print("Part. with ID = ",part.id , " type: ", part.agentType)
        if (part.agentType == 'weighted'):
            print("     weights :",part.weightCost , " ",part.weightTime )
        print("     Proposals :")
        for prop in part.proposals:
            print("     ",prop.taskId," ",prop.time," ",prop.cost)

# Print Received Proposals
def PrintReceivedReplied(props):
    print("Received Replies : ")
    for prop in props:
        print("     Part: ",prop.partId,"State :", prop.state)
# Print Accepted Proposals
def PrintReceivedProposals(props):
    print("Proposals (Accept) : ")
    for prop in props:
        print("     Part: ",prop.partId,
              " State:", prop.state ,
              " Task: ",prop.proposal.taskId,
              " time: ",prop.proposal.time,
              " cost:",prop.proposal.cost)
```

Figure 5: Generating proposals and printing functions

Looking at figure 5 above, where the code snippet is from the main.py file. The first function generates random proposals using the *random* python library, where is generates random cost integers between 1 & 10 as well as random days integers between 15 & 20. This function returns the proposal(taskID, days, cost) mentioned in figure 2 classes initiating. Different printing functions are then initiated, such as printing all participants data including the participants ID, their proposal, and the type of their proposal. This means that the population of participants is divided into 25% cost over time, 25% time over cost, 25% random , 25% using weighting.

Another printing function is initiated to print the received replies, where it includes the state meaning if the reply is “Accepted” or “Refused” and it also prints the participantID sending this specific reply.

It can also be seen that another function is built to print all the received and accepted proposals, this is done by printing the participantID who sent this proposal, the state of the proposal and finally the content of the proposal such as taskID, time, and cost.



```python
def generateTasks(taskSize):
    tasks = []
    for i in range(taskSize):
        tasks.append(task(i))

    return tasks
##########################################################################

task_size=10
tasks = generateTasks(task_size)

mParticipants = []
for i in range(12):
    mProposals = []
    for j in range(task_size):
        if randint(0,1):
            mProposals.append(generateprop(j))

    mParticipants.append(participant(i,
                                     mProposals,
                                     getType(i),
                                     getCostTimeWeight(getType(i)),
                                     getCostTimeWeight(getType(i))))

printParticipents(mParticipants)
```

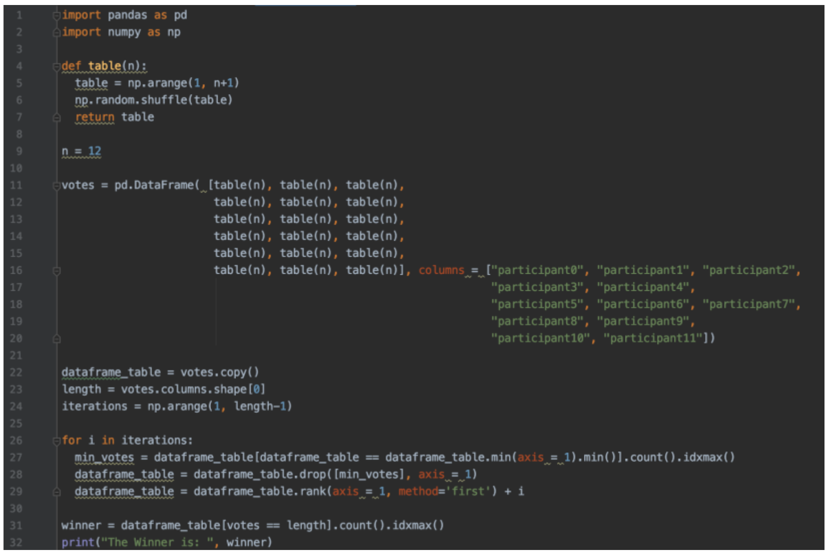
Figure 6: Generating tasks

The main.py file also includes a function which generate tasks and adds them to an empty list, figure 6 above shows the process. The task size is generated as well as the weighted cost and time computed of each proposal. It will be visibly printed in the output of the program.

In order to visualize the output of the program, the python file including all the classes is called in the main file; when running the main.py file it generates the output of both files. The next section shows the output of the system.

3- Run-Off Voting Technique:

The above code snippets and voting methods included plurality voting, where the participant preference with the lowest cost and days wins. Another voting method presented is the Run-Off methods which keep looping on the results, where the lowest preference (who gets the least votes) is dropped from the data frame and the participants who voted for it are then asked who they would vote for if their first preference was not available. Figure 7 below shows the implementation of the Run-Off Voting algorithm.



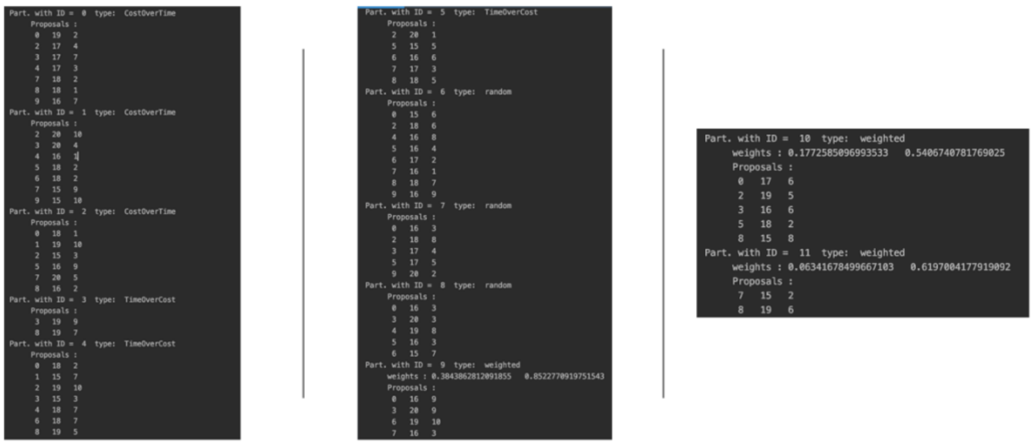
```python
import pandas as pd
import numpy as np

def table(n):
  table = np.arange(1, n+1)
  np.random.shuffle(table)
  return table

n = 12

votes = pd.DataFrame( [table(n), table(n), table(n),
                       table(n), table(n), table(n),
                       table(n), table(n), table(n),
                       table(n), table(n), table(n),
                       table(n), table(n), table(n),
                       table(n), table(n), table(n)], columns = ["participant0", "participant1", "participant2",
                                                                 "participant3", "participant4",
                                                                 "participant5", "participant6", "participant7",
                                                                 "participant8", "participant9",
                                                                 "participant10", "participant11"])

dataframe_table = votes.copy()
length = votes.columns.shape[0]
iterations = np.arange(1, length-1)

for i in iterations:
  min_votes = dataframe_table[dataframe_table == dataframe_table.min(axis = 1).min()].count().idxmax()
  dataframe_table = dataframe_table.drop([min_votes], axis = 1)
  dataframe_table = dataframe_table.rank(axis = 1, method='first') + i

winner = dataframe_table[votes == length].count().idxmax()
print("The Winner is: ", winner)
```

Figure 7: Run-Off Voting Implementation

In order to visualize the output of the program, the python file including all the classes is called in the main file; when running the main.py file it generates the output of both files. The next section shows the output of the system.

4- Output:

The output generated after running the main.py file discussed above, the first step of the output can be seen in the figure below.



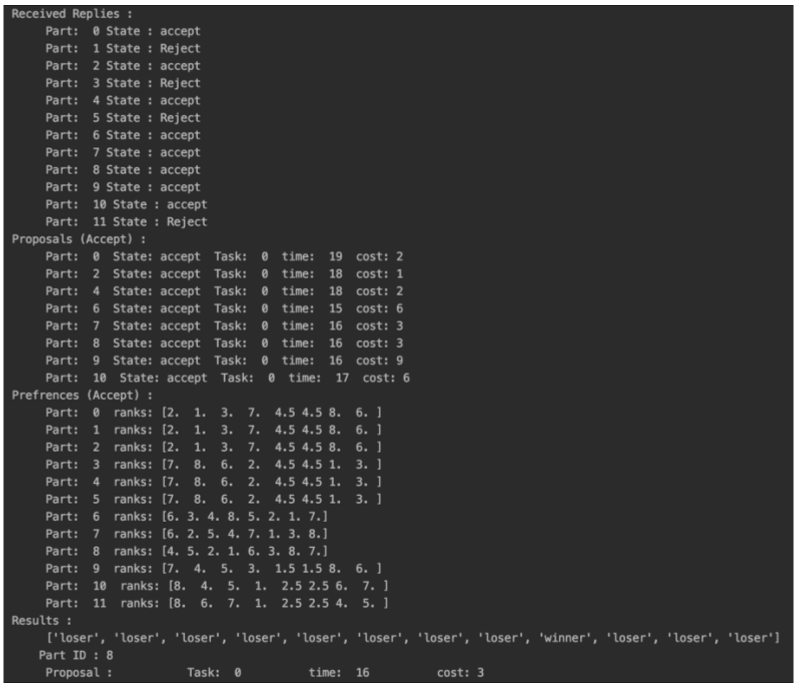
```
Part. with ID =  0  type:  CostOverTime
      Proposals :
       0   19   2
       2   17   4
       3   17   7
       4   17   3
       7   18   2
       8   18   1
       9   16   7
Part. with ID =  1  type:  CostOverTime
      Proposals :
       2   20   10
       3   20   4
       4   16   1
       5   18   2
       6   18   2
       7   15   9
       9   15   10
Part. with ID =  2  type:  CostOverTime
      Proposals :
       0   18   1
       1   19   10
       2   15   3
       5   16   9
       7   20   5
       8   16   2
Part. with ID =  3  type:  TimeOverCost
      Proposals :
       3   19   9
       8   19   7
Part. with ID =  4  type:  TimeOverCost
      Proposals :
       0   18   2
       1   15   7
       2   19   10
       3   15   3
       4   18   7
       6   18   7
       8   19   5
```

```
Part. with ID =  5  type:  TimeOverCost
      Proposals :
       2   20   1
       5   15   5
       6   16   6
       7   17   3
       8   18   5
Part. with ID =  6  type:  random
      Proposals :
       0   15   6
       2   18   6
       4   16   8
       5   16   4
       6   17   2
       7   16   1
       8   18   7
       9   16   9
Part. with ID =  7  type:  random
      Proposals :
       0   16   3
       2   18   8
       3   17   4
       5   17   5
       9   20   2
Part. with ID =  8  type:  random
      Proposals :
       0   16   3
       3   20   3
       4   19   8
       5   16   3
       6   15   7
Part. with ID =  9  type:  weighted
      weights : 0.3843862812091855   0.8522770919751543
      Proposals :
       0   16   9
       3   20   9
       6   19   10
       7   16   3
```

```
Part. with ID =  10  type:  weighted
      weights : 0.1772585096993533   0.5406740781769025
      Proposals :
       0   17   6
       2   19   5
       3   16   6
       5   18   2
       8   15   8
Part. with ID =  11  type:  weighted
      weights : 0.06341678499667103   0.6197004177919092
      Proposals :
       7   15   2
       8   19   6
```

Figure 8: Proposals sent by all participants

Looking at figure 8 above, it can be seen that we have 12 different participants represented between 0 and 11. It can be seen that each participant has a different ID and sends different proposals to the initiator in the form of (TaskID, Days, Cost). It can also be visualised the type used to generate the proposal, such as cost over time, time over cost, or randomly generated using the random python library.

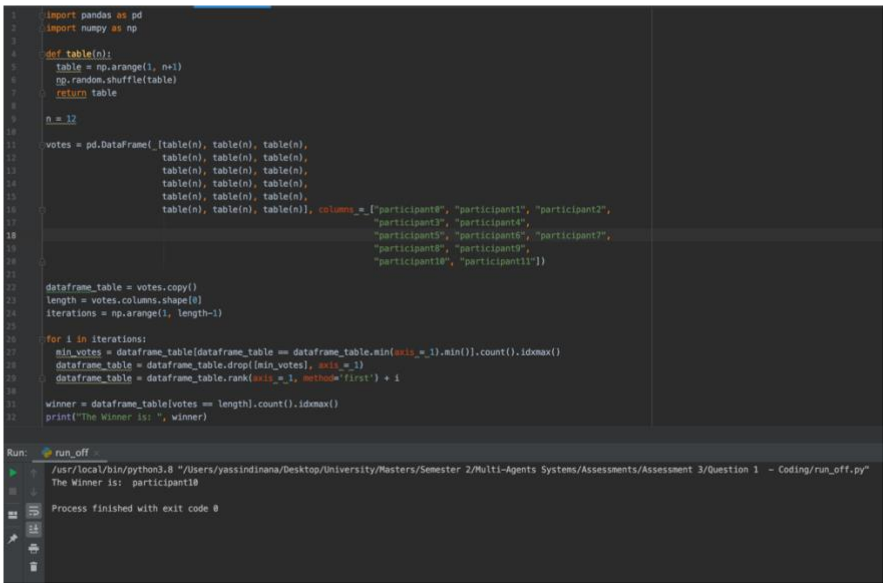


```
Received Replies :
     Part:  0 State : accept
     Part:  1 State : Reject
     Part:  2 State : accept
     Part:  3 State : Reject
     Part:  4 State : accept
     Part:  5 State : Reject
     Part:  6 State : accept
     Part:  7 State : accept
     Part:  8 State : accept
     Part:  9 State : accept
     Part:  10 State : accept
     Part:  11 State : Reject
Proposals (Accept) :
     Part:  0  State: accept  Task:  0  time:  19  cost: 2
     Part:  2  State: accept  Task:  0  time:  18  cost: 1
     Part:  4  State: accept  Task:  0  time:  18  cost: 2
     Part:  6  State: accept  Task:  0  time:  15  cost: 6
     Part:  7  State: accept  Task:  0  time:  16  cost: 3
     Part:  8  State: accept  Task:  0  time:  16  cost: 3
     Part:  9  State: accept  Task:  0  time:  16  cost: 9
     Part:  10  State: accept  Task:  0  time:  17  cost: 6
Prefrences (Accept) :
     Part:  0  ranks: [2.  1.  3.  7.  4.5 4.5 8.  6. ]
     Part:  1  ranks: [2.  1.  3.  7.  4.5 4.5 8.  6. ]
     Part:  2  ranks: [2.  1.  3.  7.  4.5 4.5 8.  6. ]
     Part:  3  ranks: [7.  8.  6.  2.  4.5 4.5 1.  3. ]
     Part:  4  ranks: [7.  8.  6.  2.  4.5 4.5 1.  3. ]
     Part:  5  ranks: [7.  8.  6.  2.  4.5 4.5 1.  3. ]
     Part:  6  ranks: [6. 3. 4. 8. 5. 2. 1. 7.]
     Part:  7  ranks: [6. 2. 5. 4. 7. 1. 3. 8.]
     Part:  8  ranks: [4. 5. 2. 1. 6. 3. 8. 7.]
     Part:  9  ranks: [7.  4.  5.  3.  1.5 1.5 8.  6. ]
     Part:  10  ranks: [8.  4.  5.  1.  2.5 2.5 6.  7. ]
     Part:  11  ranks: [8.  6.  7.  1.  2.5 2.5 4.  5. ]
Results :
     ['loser', 'loser', 'loser', 'loser', 'loser', 'loser', 'loser', 'loser', 'winner', 'loser', 'loser', 'loser']
    Part ID : 8
     Proposal :          Task:  0         time:  16         cost: 3
```

Figure 9: Proposals state, preferences, and results

Figure 9 above shows the proposals state whether its accepted or rejected, it can be seen that out of the 12 proposals, 8 are accepted and 4 are rejected. The accepted proposals are then analysed, and the rejected ones are omitted. The participant then sends all their preferences to the initiator. The initiator then ranks all the proposals data (TaskID, Days, Cost) where the proposal with the lowest data wins.

The lord initiator then sends the results to the participants where there is only one winner and all others are losers. In the example output in figure 9, it can be seen that the participant with ID number 8 has won, where he can finish the task in only 16 days with a cost of 3.



```python
import pandas as pd
import numpy as np

def table(n):
  table = np.arange(1, n+1)
  np.random.shuffle(table)
  return table

n = 12

votes = pd.DataFrame( [table(n), table(n), table(n),
                      table(n), table(n), table(n),
                      table(n), table(n), table(n),
                      table(n), table(n), table(n),
                      table(n), table(n), table(n),
                      table(n), table(n), table(n)], columns = ["participant0", "participant1", "participant2",
                                                                "participant3", "participant4",
                                                                "participant5", "participant6", "participant7",
                                                                "participant8", "participant9",
                                                                "participant10", "participant11"])

dataframe_table = votes.copy()
length = votes.columns.shape[0]
iterations = np.arange(1, length-1)

for i in iterations:
  min_votes = dataframe_table[dataframe_table == dataframe_table.min(axis = 1).min()].count().idxmax()
  dataframe_table = dataframe_table.drop([min_votes], axis = 1)
  dataframe_table = dataframe_table.rank(axis = 1, method='first') + i

winner = dataframe_table[votes == length].count().idxmax()
print("The Winner is: ", winner)
```

```
/usr/local/bin/python3.8 "/Users/yassindinana/Desktop/University/Masters/Semester 2/Multi-Agents Systems/Assessments/Assessment 3/Question 1  - Coding/run_off.py"
The Winner is:  participant10

Process finished with exit code 0
```

Figure 10: Output of the run-off algorithm

By looking at figure 10 above, it references the Run-Off algorithm implemented and explained in figure 7. There are 12 participants which are integrated in a table. The for loop seen in figure 10 iterates and computes the output by dropping the participants with the lowest number of votes and then iterating the vote to another proposal.

By looking at the output of the algorithm, it can be seen that the winner after the run-off voting technique is participant 10.