Software Test Documentation: Test Results

for

A GUI-based Software Package for the Simulation of Queuing Networks

Prepared by

Fatma Erem Aksoy – 2315075
Ece Erseven – 2385383
Ceazar Hatokah – 2456119

Middle East Technical University Northern Cyprus Campus

Computer Engineering

Supervised by Enver Ever

06/04/2023

Contents

1 Introduction

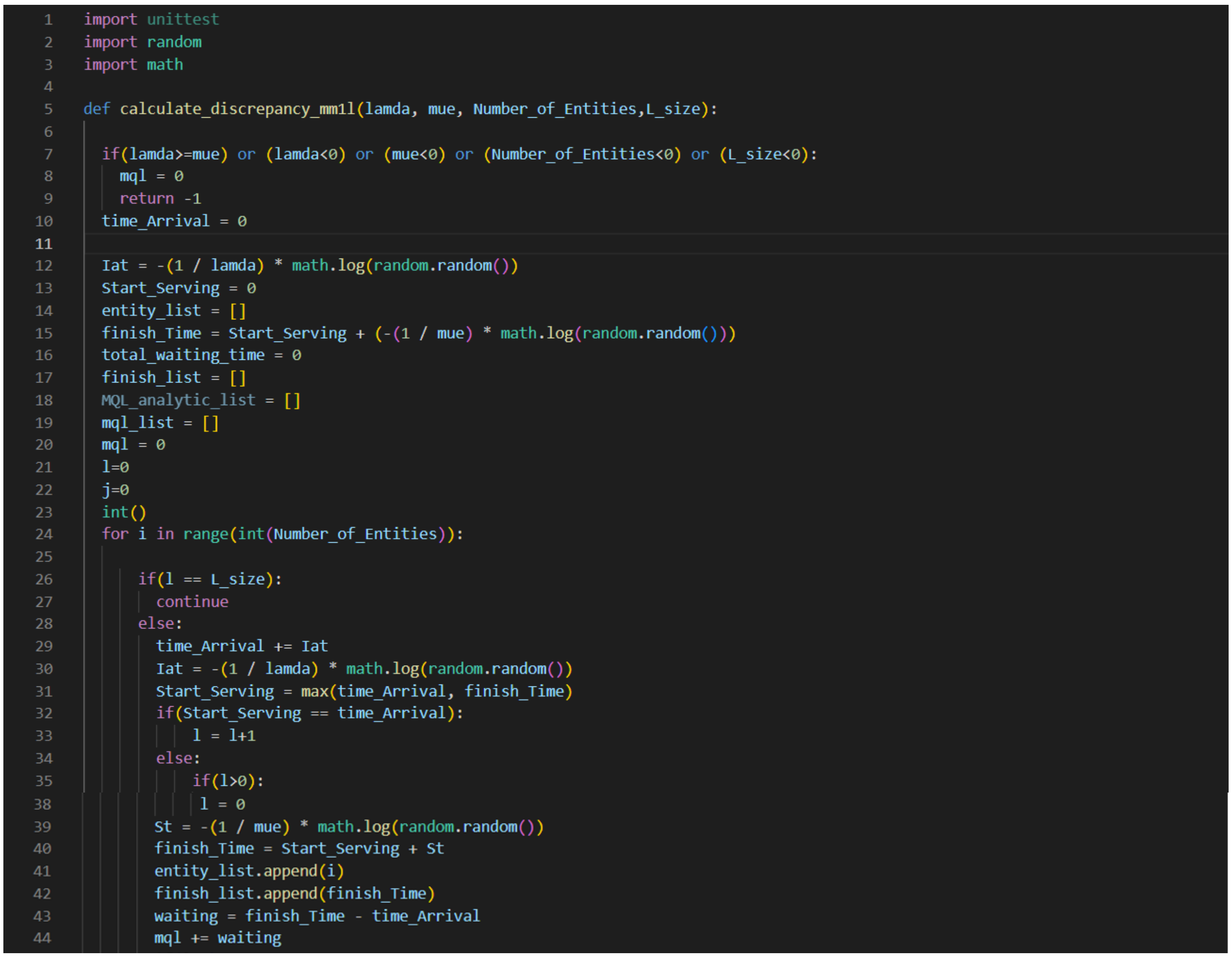
This report presents the test results of the GUI-based Software Package for the Simulation of Queueing Networks. The software product is available as a web application and builds queue simulations by supplying a GUI interface to the user. The web application redirects to the selection page when the user clicks the "Create Simulation" button on the homepage. The selection page represents all available simulation types: M/M/1, M/M/1/L, M/M/C, M/M/C/L. After selecting the simulation type, the user is redirected to the drag-and-drop page, and GUI allows users to drag the server, queue, and connection from the toolbox and drop on the canvas. The dropped items (server and queue) have clickable functionality. The user is asked to enter the inputs based on the selected simulation type. For example, for M/M/1 and M/M/C simulations, the user must enter the number of entities and lambda (average arrival rate) for the queue component. For M/M/1/L, in addition to the number of entities and lambda user is asked to enter the queue size value, and for M/M/C/L simulation user is asked to enter the limit value (the number of jobs in the system). For all simulation types, the user must enter mue (average service rate) for the server component when clicking on it; for multiple server cases, the user must enter the mue value for each server by clicking on the server components separately. While taking inputs from the user, the software also checks if the input parameters are all provided and in appropriate ranges. After providing all parameters, the user clicks on the run button, and the related simulation with the chosen simulation type is generated. During the testing period, we conducted Unit, Integration, System, Performance, and User Testing as planned in the test plan. Regarding Unit testing, we verified the correctness and functionality of simulation-related code elements individually. Regarding Integration testing, we confirmed the connection between different pages in the web application. We also tested the integration of the front-end and back-end components to ensure the inputs provided by the user successfully can be accessed and used by the simulation code. We performed the System testing to ensure that the entire software functions as intended and meets the specified requirements by checking all aspects of the software to ensure it satisfies the desired expectations. Considering Performance testing, we verified the performance of generating simulations under different workloads. In terms of user testing, we used the alpha testing technique by creating a questionnaire and delivering it to participants who are from the computer engineering department. Regarding the questionnaire answers, we verified that all the components, including the drag-and-drop feature, home page, and help page, are easy to use and understand. To sum up, we have completed all the tests specified in the test plan to confirm the quality and reliability of the GUI-based Simulation of Queuing software.

2 Test Procedures, Cases and Results

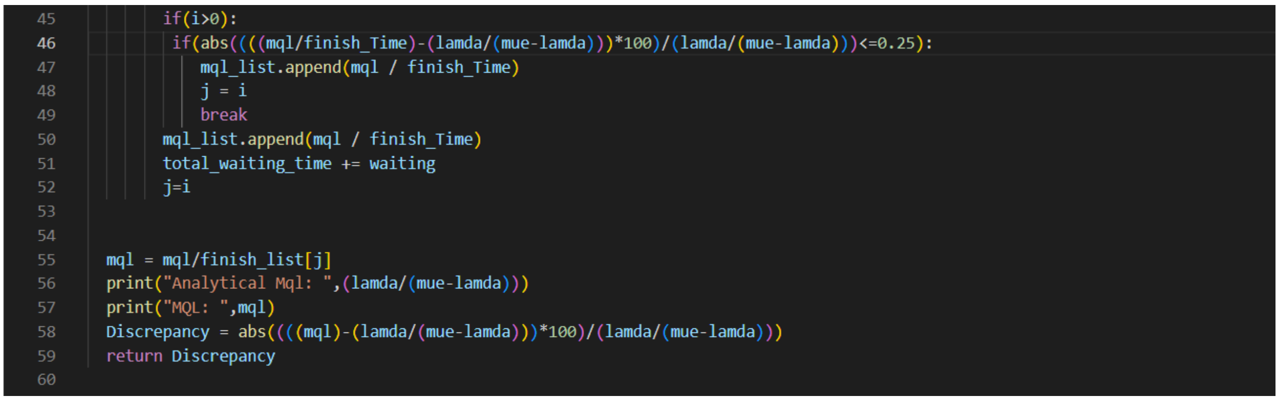
2.1 Unit Testing

2.1.1 Test Procedure

For Unit Testing, plotting of the graphs for M/M/1, M/M/1/L, M/M/C and M/M/C/L simulations were tested by running the html files for them in a browser and checking the plotting of the graphs; but for the statistics and formulas, they were tested on PyUnit and expected values are decided by using analytical approach which calculates average number of customers in the system, analytical mql calculated, for M/M/1, M/M/1/L, M/M/C and M/M/C/L. Erem created the test cases and performed the validation checks, and Ceazar did the other parts of the testing listed below in the next section, 2.1.2. A sample code we used for M/M/1/L Unit Testing in PyUnit is provided below, Figure 1, and this code calculates the mql value and prints it. When the user enters the parameter inputs, the inputs are checked; if there is a negative input it will print an error message, otherwise it will calculate the mql value. This code is used for all test cases regarding M/M/1/L.



```
import unittest
import random
import math

def calculate_discrepancy_mm1l(lamda, mue, Number_of_Entities,L_size):

    if(lamda>=mue) or (lamda<0) or (mue<0) or (Number_of_Entities<0) or (L_size<0):
        mql = 0
        return -1
    time_Arrival = 0

    Iat = -(1 / lamda) * math.log(random.random())
    Start_Serving = 0
    entity_list = []
    finish_Time = Start_Serving + (-(1 / mue) * math.log(random.random()))
    total_waiting_time = 0
    finish_list = []
    MQL_analytic_list = []
    mql_list = []
    mql = 0
    l=0
    j=0
    int()
    for i in range(int(Number_of_Entities)):

        if(l == L_size):
            continue
        else:
            time_Arrival += Iat
            Iat = -(1 / lamda) * math.log(random.random())
            Start_Serving = max(time_Arrival, finish_Time)
            if(Start_Serving == time_Arrival):
                l = l+1
            else:
                if(l>0):
                    l = 0
            St = -(1 / mue) * math.log(random.random())
            finish_Time = Start_Serving + St
            entity_list.append(i)
            finish_list.append(finish_Time)
            waiting = finish_Time - time_Arrival
            mql += waiting
```



```
            if(i>0):
             if(abs((((mql/finish_Time)-(lamda/(mue-lamda)))*100)/(lamda/(mue-lamda)))<=0.25):
                mql_list.append(mql / finish_Time)
                j = i
                break
            mql_list.append(mql / finish_Time)
            total_waiting_time += waiting
            j=i


    mql = mql/finish_list[j]
    print("Analytical Mql: ",(lamda/(mue-lamda)))
    print("MQL: ",mql)
    Discrepancy = abs((((mql)-(lamda/(mue-lamda)))*100)/(lamda/(mue-lamda)))
    return Discrepancy
```

Figure 1: Sample Unit Testing code used for M/M/1/L Simulation in PyUnit

2.1.2 Test cases

Test cases we used for Unit testing are listed below:

ID	***Description***	***Steps***	***Test Data***	***Pre-condition***	***Expected Output***	***Passed /Failed***
1	*Enter values for the input data of the servers and queues in the canvas and run the simulation for M/M/1.*	*1. Select the type of simulation you want to run 2. Drag and drop items on the canvas. 3. Enter data for queues and servers dragged (Mue for server; lamda and number of entities for the queue) 4. Run simulation*	*Lamda (Arrival rate) = 2 Mue (Service rate) = 5 Number Of Entities = 10,000*	-	*Line Graph converging towards the end of number of entities with Mql = 0.67*	*Passed*
2	*Enter values for the input data of the server and queue in the canvas and run the simulation for M/M/1.*	*1. Select the type of simulation you want to run 2. Drag and drop items on the canvas. 3. Enter data for queues and servers dragged (Mue for server; lamda and number of entities for the queue) 4. Run simulation*	*Lamda (Arrival rate) = 2 Mue (Service rate) = 5 Number Of Entities = 1,000,000*	-	*Line Graph converging towards the end of number of entities with Mql = 0.67*	*Passed*
3	*Enter values for the input data of the server and queue in the canvas and run the simulation for M/M/1.*	*1. Select the type of simulation you want to run 2. Drag and drop items on the canvas. 3. Enter data for queues and servers dragged (Mue for server; lamda and number of entities for the queue) 4. Run simulation*	*Lamda (Arrival rate) = 2 Mue (Service rate) = 1 Number Of Entities = 10,000*	-	*Print an error message stating that the Lambda value should be less than Mue*	*Passed*
4	*Enter values for the input data of the server and queue in the canvas and run the simulation for M/M/1/L.*	*1. Select the type of simulation you want to run 2. Drag and drop items on the canvas. 3. Enter data for queue and server dragged (Mue for server; lamda, number of entities and queue size for the queue) 4. Run simulation*	*Lamda (Arrival rate) = 2 Mue (Service rate) = 4 Number Of Entities = 10,000 Queue size = 500*	-	*Line Graph converging towards with Mql = 1 (Not necessarily traversing through all of the entities)*	*Passed*

5	*Enter values for the input data of the servers and queues in the canvas and run the simulation for M/M/1/L.*	*1. Select the type of simulation you want to run* *2. Drag and drops items on the canvas.* *3. Enter data for queues and servers dragged (Mue for server; lamda, number of entities and queue size for the queue)* *4. Run simulation*	*Lamda (Arrival rate) = -3* *Mue (Service rate) = 5* *Number Of Entities = 10,000* *Queue size = 1000*	**-**	*Print an error message stating that the Lambda value cannot be a negative number*	*Passed*
6	*Enter values for the input data of the servers and queues in the canvas and run the simulation for M/M/C.*	*1. Select the type of simulation you want to run* *2. Drag and drops items on the canvas.* *3. Enter data for queues and servers dragged (Mue value for each server; lamda and number of entities for the queue)* *4. Run simulation*	*Lamda (Arrival rate) = 2* *Mue (Service rate) = 5 (entered same value for 2 servers)* *Number Of Entities = 10,000*	**-**	*Line Graph with Mql = 0.416 value for 2 servers*	*Passed*
7	*Enter values for the input data of the servers and queues in the canvas and run the simulation for M/M/C.*	*1. Select the type of simulation you want to run* *2. Drag and drops items on the canvas.* *3. Enter data for queues and servers dragged (Mue value for each server; lamda and number of entities for the queue)* *4. Run simulation*	*Lamda (Arrival rate) = 2* *Mue (Service rate) = 5,8,6 (entered values for 3 different servers)* *Number Of Entities = 5,000*	**-**	*Line Graph with Mql = 0.4 value for 3 servers*	*Passed*
8	*Enter values for the input data of the servers and queues in the canvas and run the simulation for M/M/C/L.*	*1. Select the type of simulation you want to run* *2. Drag and drops items on the canvas.* *3. Enter data for queues and servers dragged (Mue value for each server; lamda, number of entities and limit, for the jobs in the system at a time, for the queue)* *4. Run simulation*	*Lamda (Arrival rate) = 2* *Mue (Service rate) = 5 (entered same value for 2 servers)* *Number Of Entities = 10,000, Limit=50*	**-**	*Line Graph with Mql = 1.02 value for 2 servers*	*Passed*
9	*Enter values for the input data of the servers and queues in the canvas and run the simulation for M/M/C/L.*	*1. Select the type of simulation you want to run* *2. Drag and drops items on the canvas.* *3. Enter data for queues and servers dragged (Mue value for each server; lamda, number of entities and limit, for the jobs in the system at a time, for the queue)* *4. Run simulation*	*Lamda (Arrival rate) = 2* *Mue (Service rate) = 5,8,6 (entered values for 3 different servers)* *Number Of Entities = 5,000, Limit = 50*	**-**	*Line Graph with Mql = 1 value for 3 servers*	*Passed*

Table 1: Sample Cases for Unit Testing

2.1.3 Test Results

Test case 1: A line graph converging with value Mql= 0.67 as it's reaching towards the end of number of entities, Passed.

Test case 2: A line graph converging with value Mql= 0.67 as it's reaching towards the end of number of entities, Passed.

Test case 3: An alert notification displayed on the screen stating that Lamda should be less than Mue, Passed.

Test case 4: A line graph converging with value Mql= 1, for M/M/1/L as soon as we get the expected mql value in one of the entities, we plot the line graph (Not necessarily reaching to the end of the entities), Passed.

Test case 5: An alert message stating that Lambda is negative, Passed.

Test case 6: A line graph with value Mql= 0.416, for 2 servers M/M/C simulation. It also displays the Mql value for 1 server, so the change can be seen from the graph, Passed.

Test case 7: A line graph with value Mql=0.4, for 3 servers M/M/C simulation. It also displays the Mql value for other servers till it reaches to 3, so the change can be seen from the graph, Passed.

Test case 8: A line graph with value Mql=1.02, for 2 servers M/M/C simulation. It also displays the Mql value for 1 server, so the change can be seen from the graph, Passed.

Test case 9: A line graph with value Mql=1, for 3 servers M/M/C simulation. It also displays the Mql value for other servers till it reaches to 3, so the change can be seen from the graph, Passed.

While executing the simulation cases, a validation check is done for the inputs; if one of the inputs is negative, then an error message is displayed or if arrival rate is greater than service rate then an appropriate message is displayed again. Another condition is that the number of entities needs to be greater than 1024, otherwise it gives an error message. If the validation is successful, then simulation runs and plots the line graph. In all these cases tested, no failure happened. Some screenshots can be found in 2.1.4 for further proof.

2.1.4 Test Log

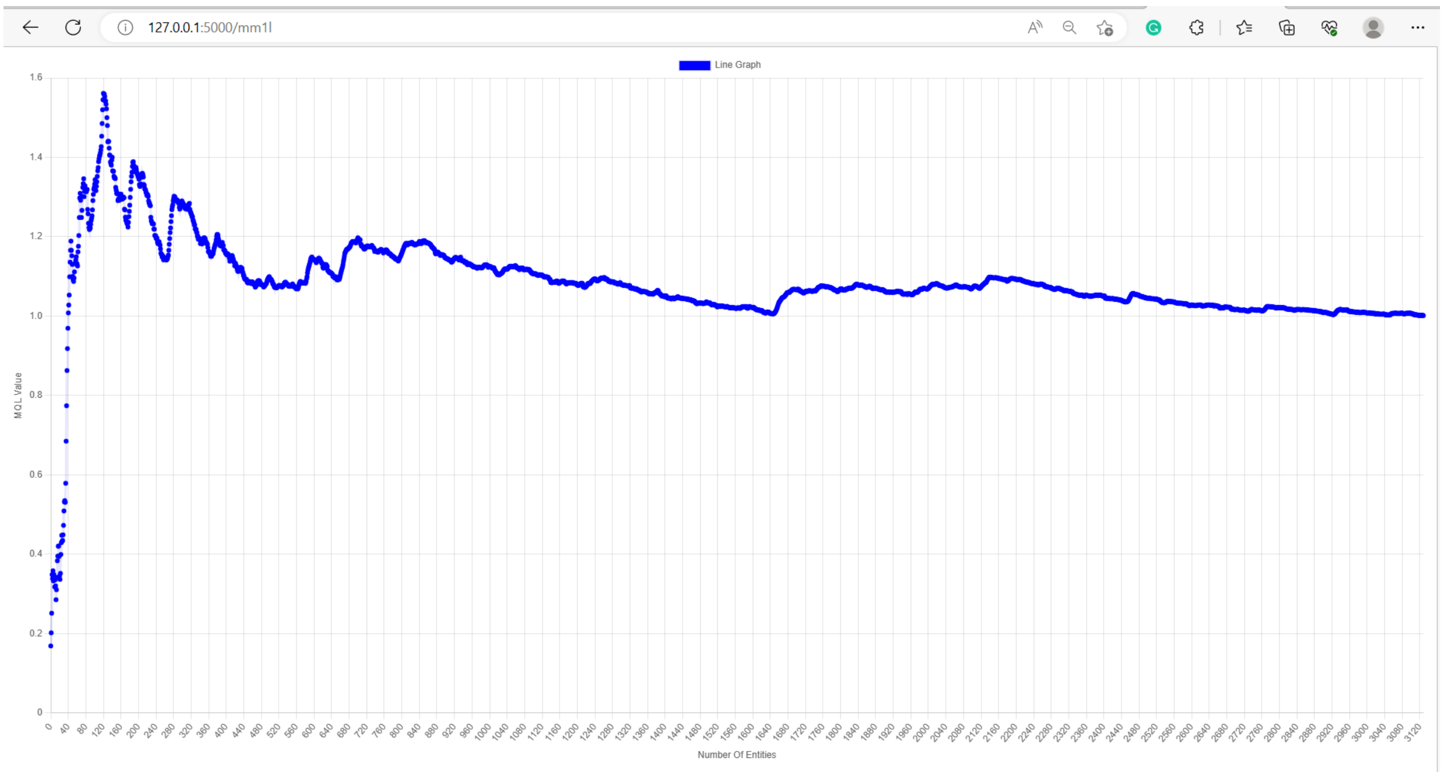
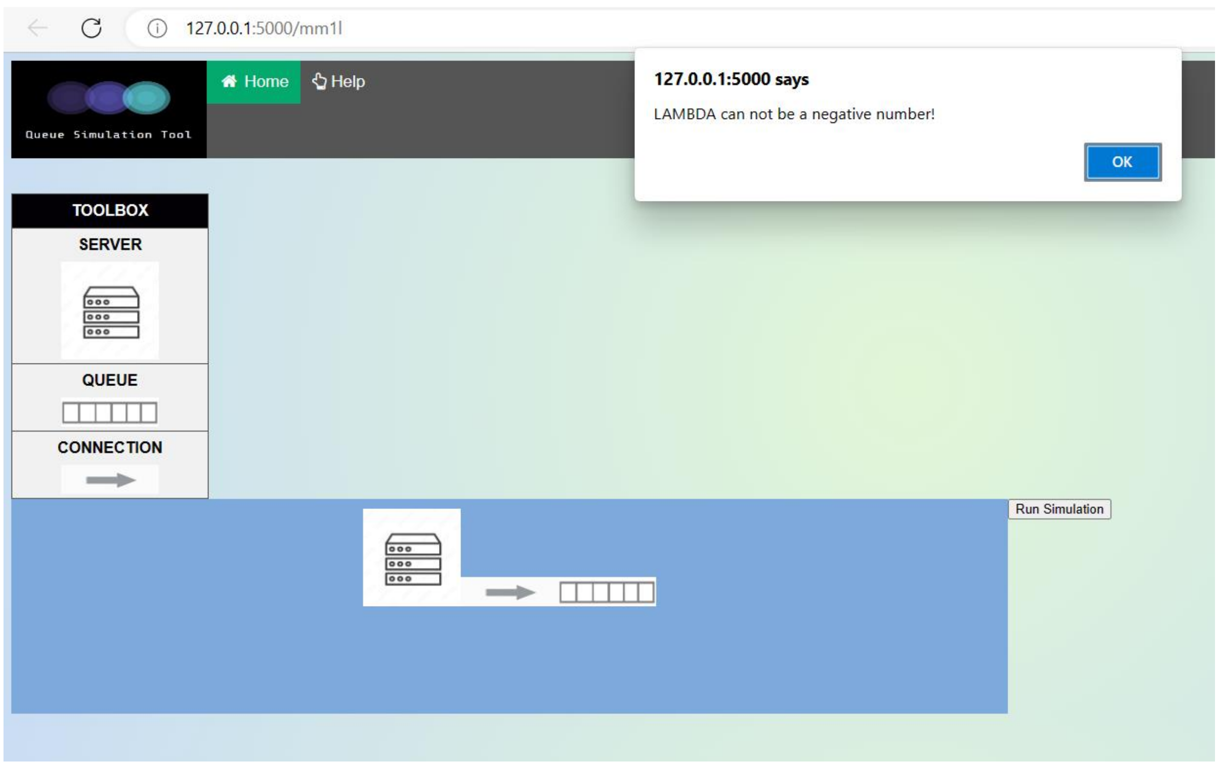


Figure 2: Case 4 (successful validation and plotting the line graph)



127.0.0.1:5000 says

LAMBDA can not be a negative number!

Figure 3: Case 8 (one of the inputs, Lambda, is negative)

2.2 Integration Testing

2.2.1 Test Procedure

Integration testing has been performed by using the bottom-up technique. We first tested each file/module separately and then did the testing for the whole program. We used Python Flask for the integration of the GUI part and the simulation part. We have .html and .js files for the GUI part and .py and .js files for the simulation part of the project. We performed all the test cases manually as we needed to check the different values and errors separately for each specified case. For the GUI part, we tested if all the web pages were connected to each other successfully and if we could switch between pages without any problem. For the simulation part, we tested if the values entered for each test case were saved correctly and if the corresponding graphs were plotted correctly with these inputs. We have tested nine graph-related files (five .js and four .html files) for this part. In addition, some of these files are related to both sides, so their testing was performed after completing separate tests for each part. For instance, the files containing drag-and-drop-related features where the values needed to be taken from the user in the GUI part and sent to the main .py file to be checked for validations and then to the corresponding .js file to plot the graph.

Ece performed the GUI part and drag-and-drop-related cases (1&2&3), and Erem performed the rest of the testing which are related to the simulation part (saving values and plotting graphs-cases 4&5&6).

2.2.2 Test cases

Test cases we performed for this section are listed as:

ID	***Description***	***Steps***	***Test Data***	***Pre-condition***	***Expected Output***	***Passed /Failed***
1	*Connection of each web page to each other*	*1*	*-*	*-*	*Being able to switch between web pages*	*Passed*
2	*Drag-and-drop feature*	*1*	*Server*	*-*	*Being able to see the Server image on the blue canvas*	*Passed*
		2	*Queue*	*-*	*Being able to see the Queue image on the blue canvas*	*Passed*
3	*Entering values for Server(s) and Queue*	*1*	*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*	*Selection of M/M/1 simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*

		2	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Selection of M/M/1/L simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Selection of M/M/C simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*
		4	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Selection of M/M/C/L simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*
4	*Accessing values for Server(s) and Queue in the main.py file*	*1*	*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 3-step 1*	*Printing the entered values correctly in the main.py console*	*Passed*
		2	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Verifying test case 3-step 2*	*Printing the entered values correctly in the main.py console*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 3-step 3*	*Printing the entered values correctly in the main.py console*	*Passed*
		4	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Verifying test case 3-step 4*	*Printing the entered values correctly in the main.py console*	*Passed*
5	*Accessing values for Server(s) and Queue in the corresponding .js files for each graph*	*1*	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Verifying test case 4-step 2*	*Printing the entered values correctly inside the mm1L_graph.js*	*Passed*
		2	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 4-step 3*	*Printing the entered values correctly inside the mmc_graph.js*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Verifying test case 4-step 4*	*Printing the entered values correctly inside the mmcL_graph.js*	*Passed*
6	*Plotting the correct graphs with the values entered for each case*	*1*	*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 5-step 1*	*Plotting an MM1 graph as expected*	*Passed*
		2	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Verifying test case 5-step 2*	*Plotting an MM1L graph as expected*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 5-step 3*	*Plotting an MMC graph as expected*	*Passed*
		4	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Verifying test case 5-step 4*	*Plotting an MMCL graph as expected*	*Passed*

Table 2: Sample Cases for Integration Testing

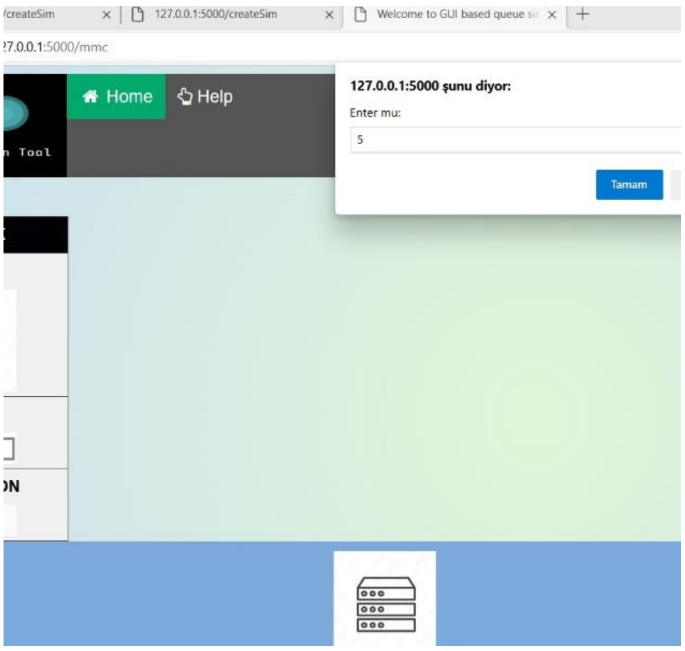
2.2.3. Test Results

All the test cases provided in the table above are passed the integration testing and how they are tested are provided with more details onn the next section, 2.2.4, with their corresponding screenshots as well.

2.2.4 Test Log

One of the cases tested is provied with detailed steps below:

The sample test case for an M/M/C simulation with the correct inputs and its graph plotted:

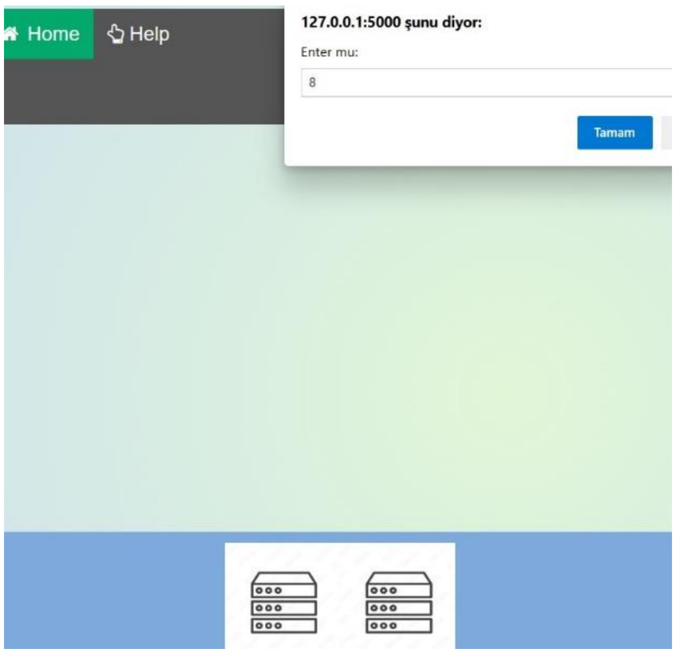


127.0.0.1:5000 şunu diyor:

Enter mu:

5

Figure 4: Entering the values, Mue value for Server 0

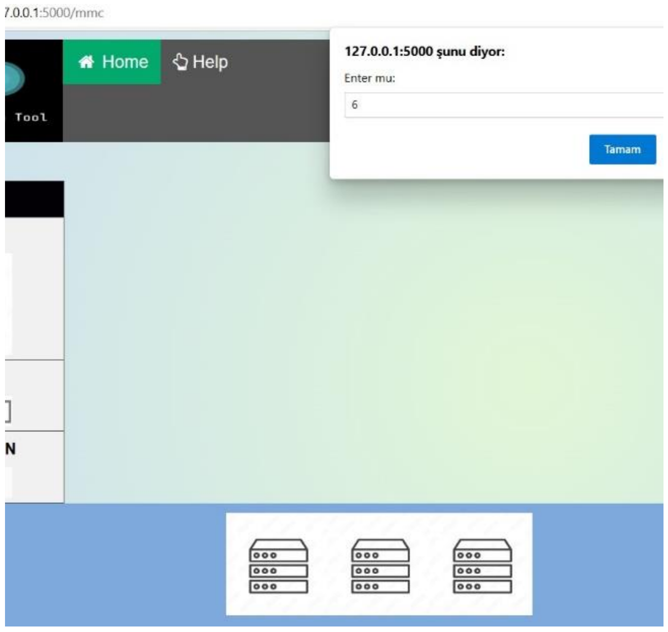


127.0.0.1:5000 şunu diyor:

Enter mu:

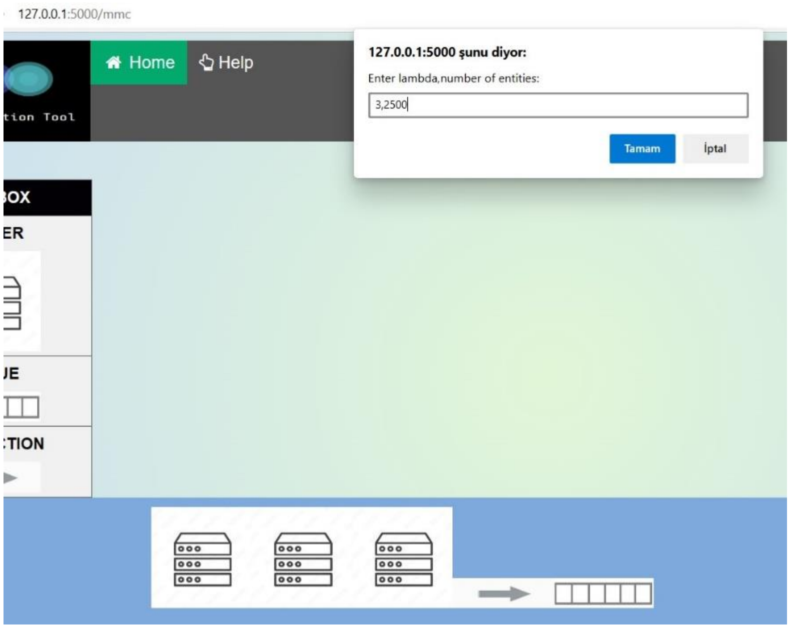
8

Figure 5: Entering the values, Mue value for Server 1



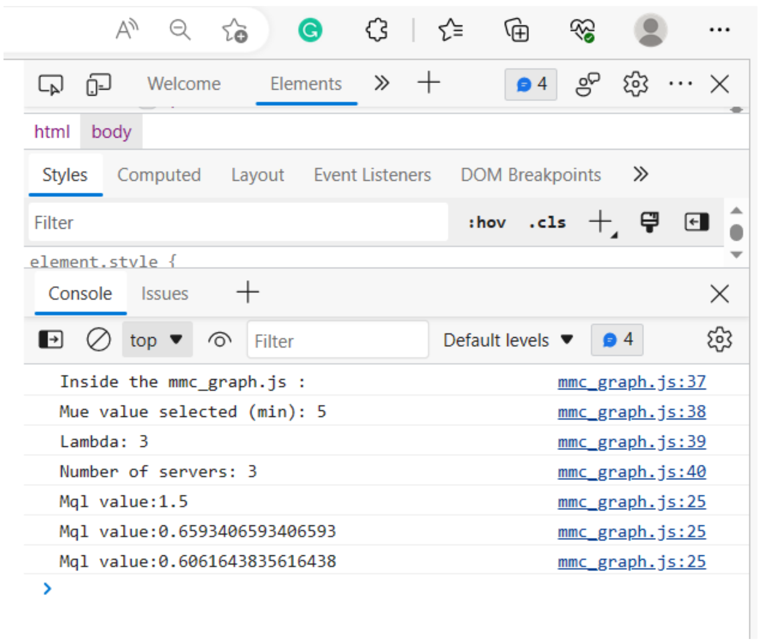
127.0.0.1:5000 şunu diyor:
Enter mu:
6
Tamam

Figure 6: Entering the values, Mue value for Server 2



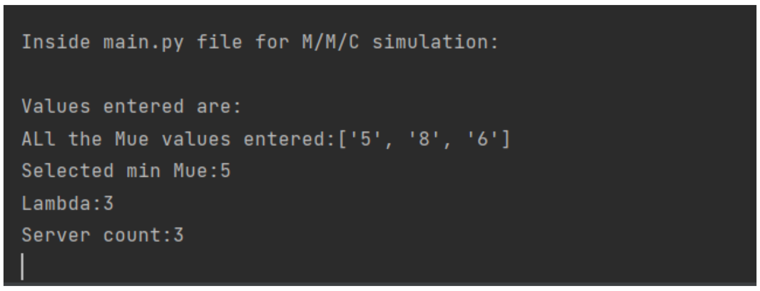
127.0.0.1:5000 şunu diyor:
Enter lambda,number of entities:
3,2500
Tamam İptal

Figure 7: Entering the values, Lambda and Number of Entities



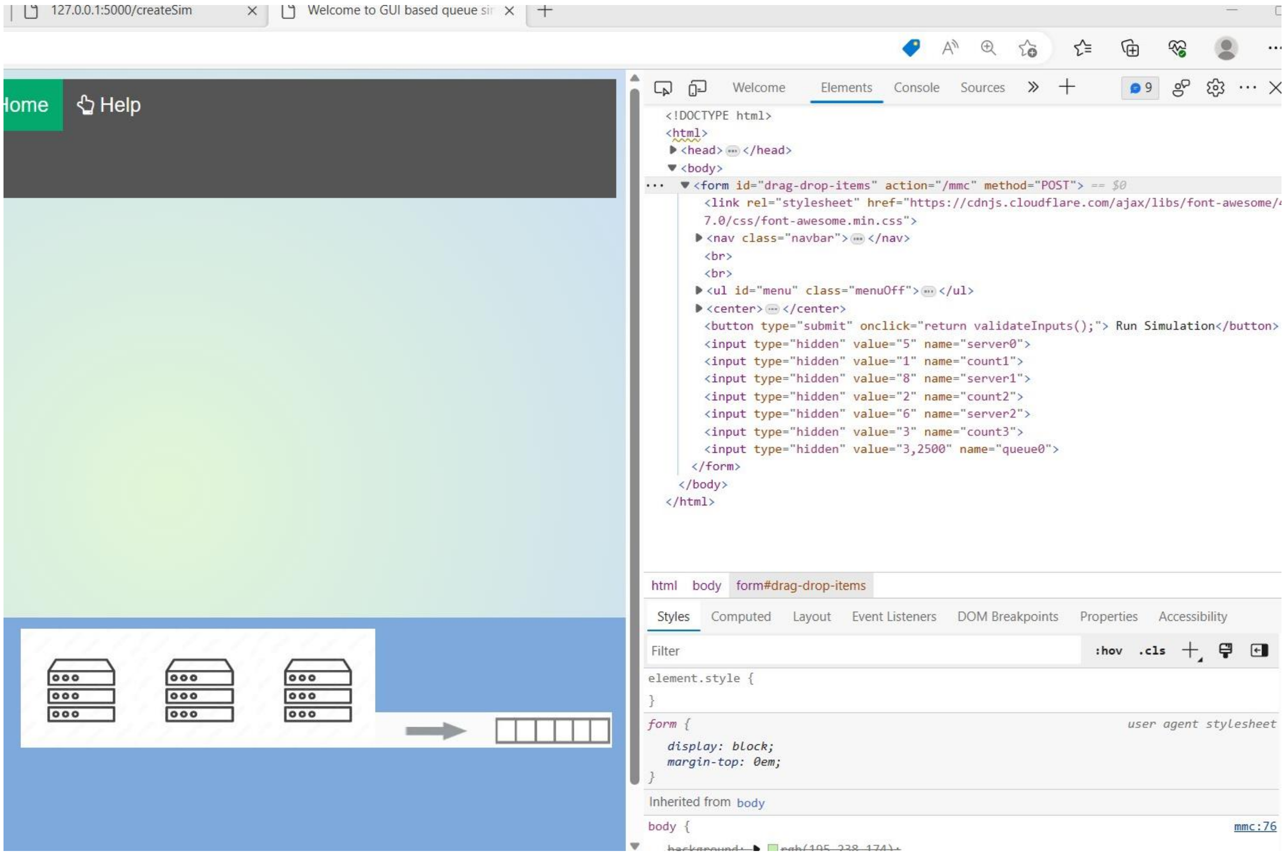
```
Inside the mmc_graph.js :                    mmc_graph.js:37
Mue value selected (min): 5                  mmc_graph.js:38
Lambda: 3                                    mmc_graph.js:39
Number of servers: 3                         mmc_graph.js:40
Mql value:1.5                                mmc_graph.js:25
Mql value:0.6593406593406593                 mmc_graph.js:25
Mql value:0.6061643835616438                 mmc_graph.js:25
```

Figure 8: Values saved inside the mmc_graph.js file



```
Inside main.py file for M/M/C simulation:

Values entered are:
ALL the Mue values entered:['5', '8', '6']
Selected min Mue:5
Lambda:3
Server count:3
```

Figure 9: Values saved inside the main.py file



```
<!DOCTYPE html>
<html>
 ▶<head> ⋯ </head>
 ▼<body>
 ···  ▼<form id="drag-drop-items" action="/mmc" method="POST"> == $0
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/
        7.0/css/font-awesome.min.css">
       ▶<nav class="navbar"> ⋯ </nav>
        <br>
        <br>
       ▶<ul id="menu" class="menuOff"> ⋯ </ul>
       ▶<center> ⋯ </center>
        <button type="submit" onclick="return validateInputs();"> Run Simulation</button>
        <input type="hidden" value="5" name="server0">
        <input type="hidden" value="1" name="count1">
        <input type="hidden" value="8" name="server1">
        <input type="hidden" value="2" name="count2">
        <input type="hidden" value="6" name="server2">
        <input type="hidden" value="3" name="count3">
        <input type="hidden" value="3,2500" name="queue0">
      </form>
    </body>
</html>
```

```
element.style {
}
form {                                   user agent stylesheet
    display: block;
    margin-top: 0em;
}
Inherited from body
body {                                                  mmc:76
```

Figure 10: Values entered saved in the web page

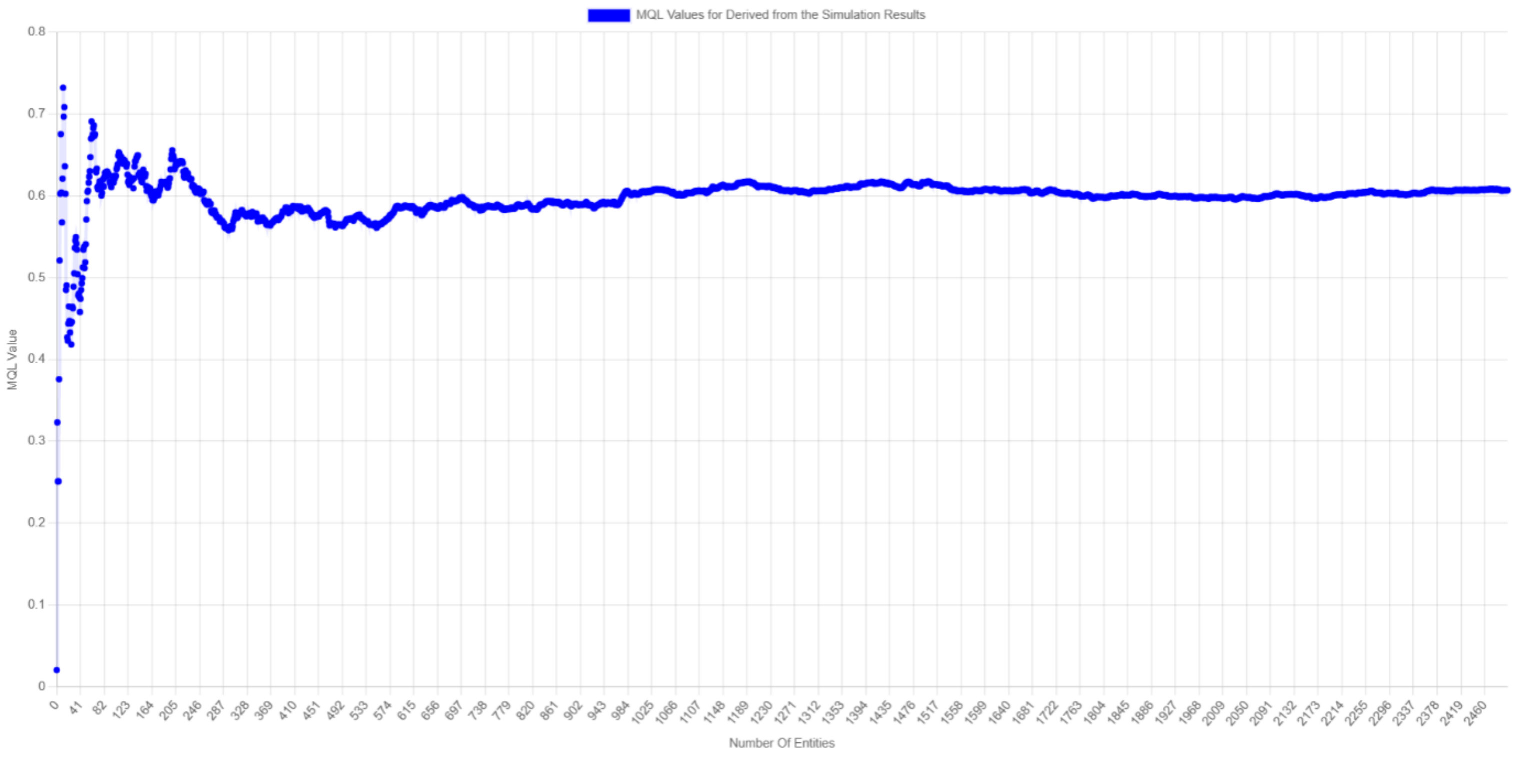


Figure 11: Graph plotted for the MQL value calulated according to the entered values (the MQL values can be verified with Figure 7 outputs as well)

2.3 System Testing

2.3.1 Test Procedure

We have used scenario-based system testing to perform system testing for this project. Several scenarios were created to split between the group members, which were tested to see if the whole system worked flawlessly in harmony. We manually tested these scenarios by following the steps provided below. Ece tested cases 1&2&3, Erem tested cases 4&5& 6.

2.3.2 Test Scenarios

The cases tested are as listed below:

1. Home -> Create Simulation -> M/M/1 -> Entering values (*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*)->Run Simulation

2. Home -> Create Simulation -> M/M/1/L -> Entering values (*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*)->Run Simulation

3. Home -> Help -> Creating a new simulation -> Create Simulation -> M/M/1 -> Entering values (*Server(mue)=6, Queue(lambda, # of entities)=2, 3700*)->Run Simulation

4. Home -> Contact Us -> Home -> Create Simulation -> M/M/C -> Entering values (*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*)->Run Simulation

5. Home -> Help -> Create Simulation -> M/M/C/L -> Entering values (*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*)->Run Simulation

6. Home -> Create Simulation -> M/M/C/L -> Entering values (*Server0(mue)=5, Server1(mue)=3, Server2(mue)=11, Server3(mue)=4, Queue(lambda, # of entities,limit)=2, 10000,8*)->Run Simulation

2.3.3 Test Results

All the test scenarios given in section 2.3.2 are passed the system testing without any minor/major errors and the screenshots are provided for one of those cases in the section 2.3.4.

2.3.4 Test Log

Case 3:

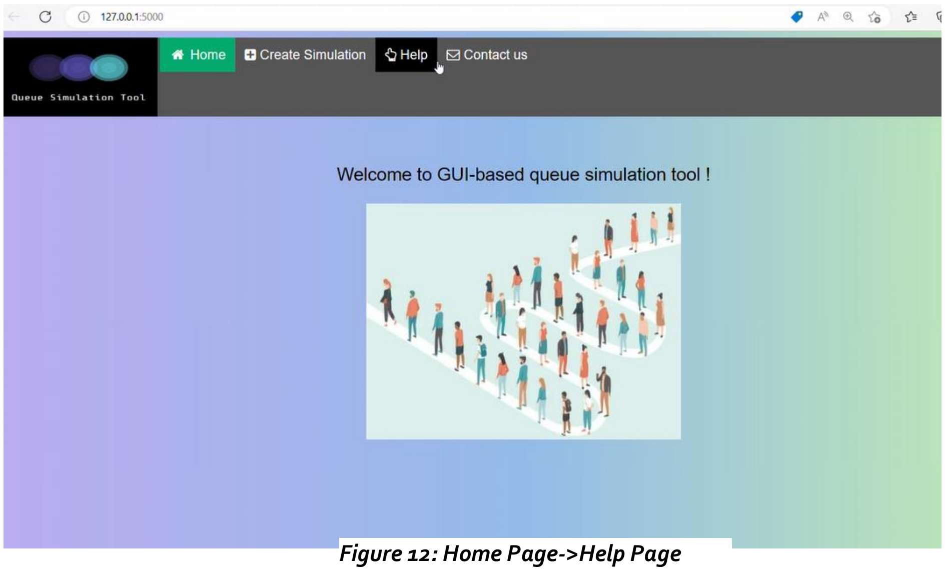


Figure 12: Home Page->Help Page

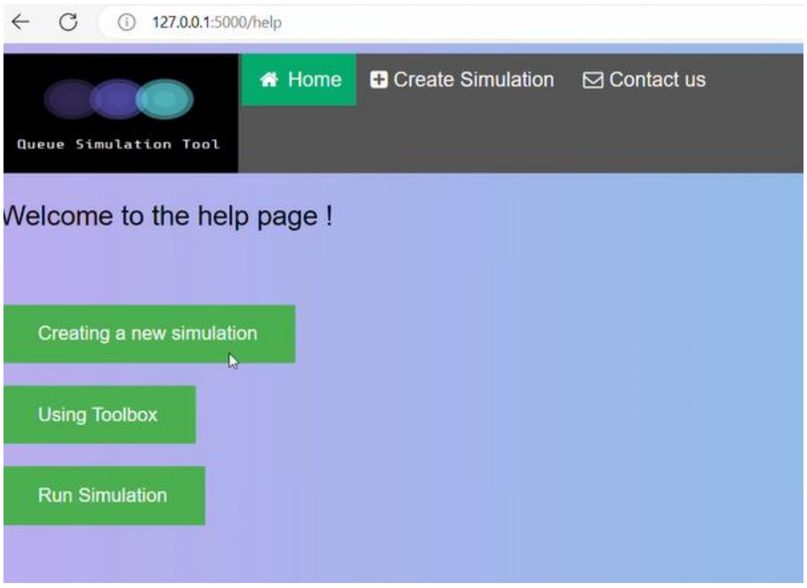


Figure 13: Help Page->Creating a new simulation

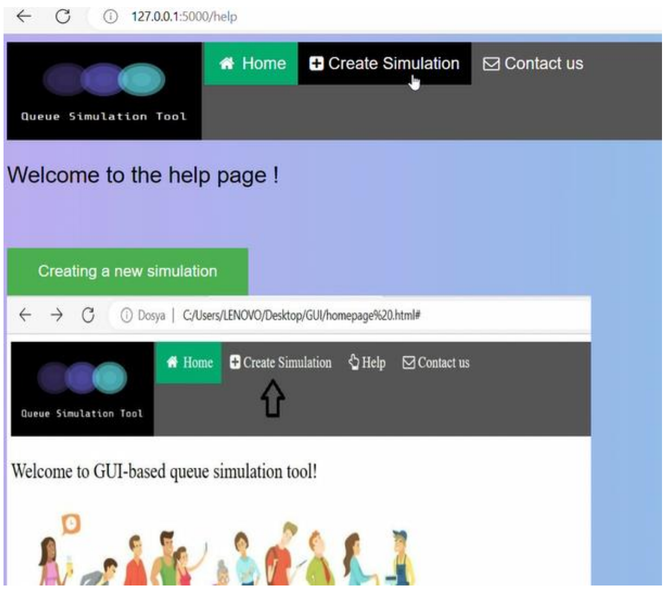

127.0.0.1:5000/help
Home
Create Simulation
Contact us
Queue Simulation Tool
Welcome to the help page !
Creating a new simulation
Dosya | C:/Users/LENOVO/Desktop/GUI/homepage%20.html#
Home
Create Simulation
Help
Contact us
Welcome to GUI-based queue simulation tool!

Figure 14: Creating a new simulation->Create Simulation

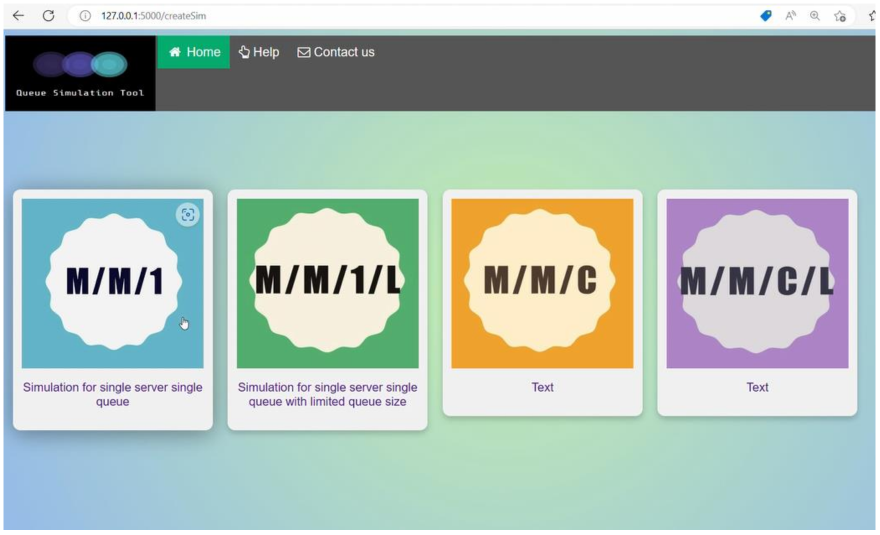
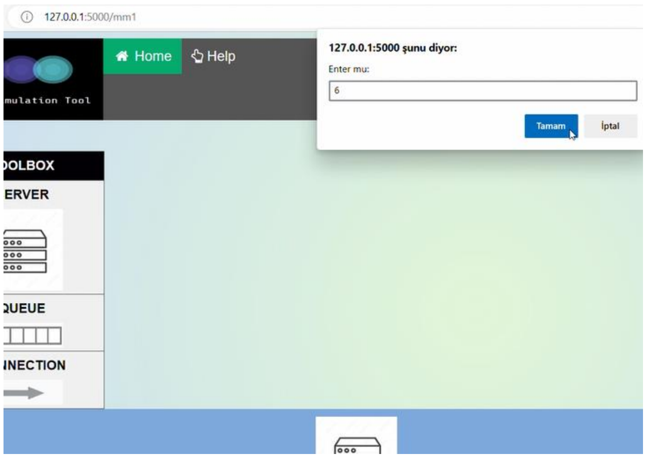

127.0.0.1:5000/createSim
Home
Help
Contact us
Queue Simulation Tool
M/M/1
Simulation for single server single queue
M/M/1/L
Simulation for single server single queue with limited queue size
M/M/C
Text
M/M/C/L
Text

Figure 15: Create Simulation->M/M/1



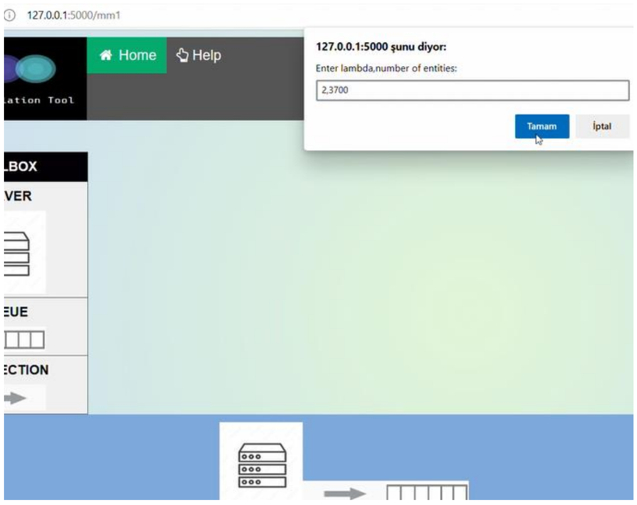
127.0.0.1:5000 şunu diyor:

Enter mu:

6

Tamam İptal

Figure 16: Entering values, Mu for Server



127.0.0.1:5000 şunu diyor:

Enter lambda,number of entities:

2,3700

Tamam İptal

Figure 17: Entering values, Lambda and Number of Entities for Queue

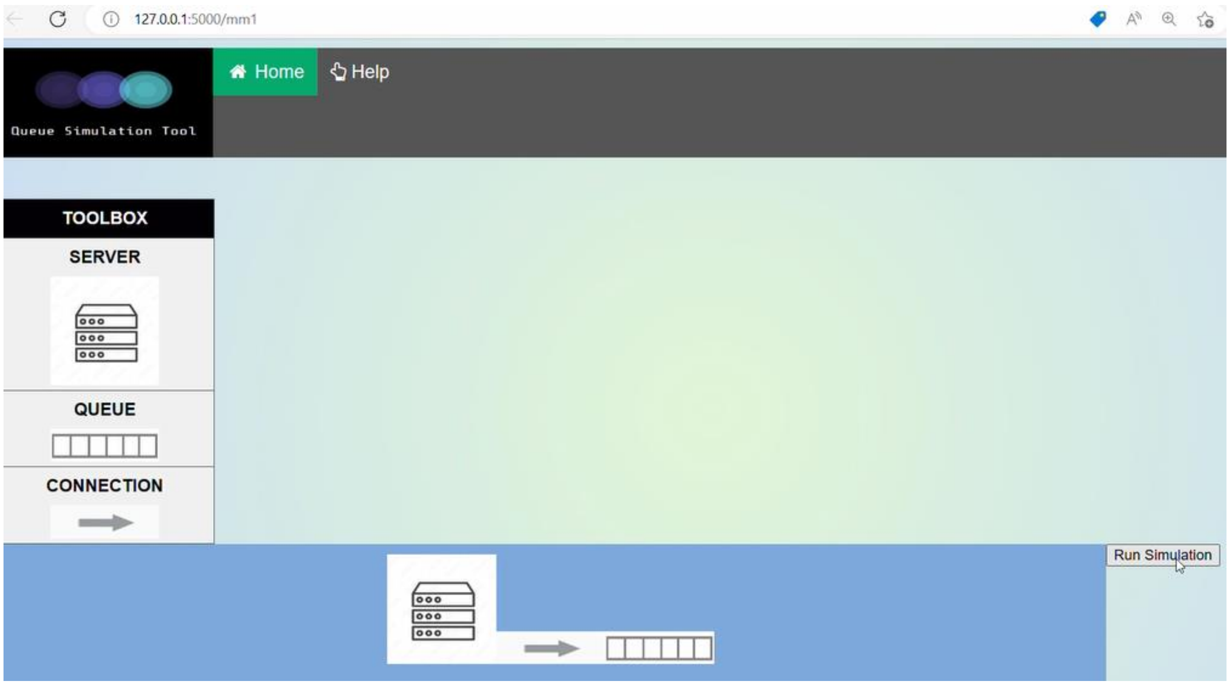


Figure 18: Run Simulation

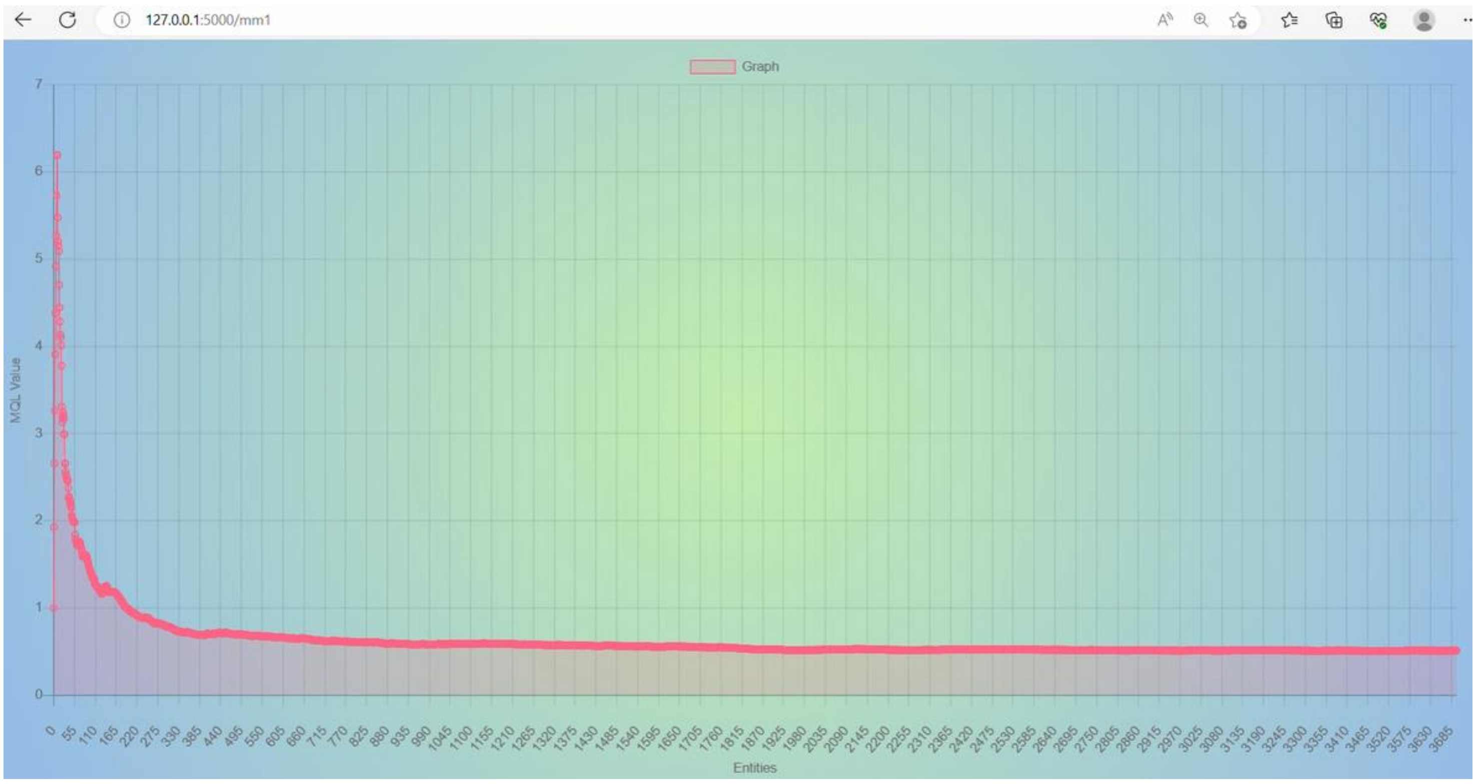


Figure 19: Plotted Graph of the Entered Values

2.4 Performance Testing

2.4.1 Test Procedure

For performance testing, we used stress testing technique to see how the system behaves with a huge amount of data, in addition to that, we want the system to be very accurate in terms of results, the expected result is calculated from the analytical approach (for M/M/1=(lamda/mue)/(1 - (lamda/mue)) and for M/M/1/L = lamda/(mue – lamda)), and then discrepancy of each entity is calculated by absolute of **[** (mql - lamda/(mue – lamda) x 100) / lamda/(mue – lamda) **]**. Discrepancy should be maximum = 0.25 so we can get very accurate results. PyUnit has been used to test the performance of the simulation with very large number of entities, and to check the discrepancy to see if passes the criteria or fails. Ceazar and Erem were responsible for performance testing.

2.4.2 Test cases

The cases tested for performance testing are listed as:

ID	*Description*	*Steps*	*Test Data*	*Pre-condition*	*Expected Output*	*Passed/ Failed*
1	*Test the data on Python by entering input data for M/M/1.*	*1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 10,000*	-	*Discrepancy <= 0.25*	*Passed*
2	*Test the data on Python by entering input data for M/M/1.*	*1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 100,000*	-	*Discrepancy <= 0.25*	*Failed*
3	*Test the data on Python by entering input data for M/M/1.*	*1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 1,000,000*	-	*Discrepancy <= 0.25*	*Passed*
4	*Test the data on Python by entering input data for M/M/1.*	*1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 2,000,000*	-	*Discrepancy <= 0.25*	*Passed*

5	*Test the data on Python by entering input data for M/M/1/L.*	*1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 10,000* *Queue limit = 500*	-	*Discrepancy <= 0.25*	*Passed*
6	*Test the data on Python by entering input data for M/M/1/L.*	*1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 100,000* *Queue limit = 500*	-	*Discrepancy <= 0.25*	*Passed*
7	*Test the data on Python by entering input data for M/M/1/L.*	*1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 1,000,000* *Queue limit = 500*	-	*Discrepancy <= 0.25*	*Passed*
8	*Test the data on Python by entering input data for M/M/1/L.*	*1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 2,000,000* *Queue limit = 500*	-	*Discrepancy <= 0.25*	*Passed*

Table 3: Sample Cases for Performance Testing

In Figure 20, the speed results for M/M/1 simulation, stress testing, are shown in a graph generated by using the values in Table 4. How the execution time changes under huge amount of data, number of entities when other parameters are fixed, can be seen from the graph.

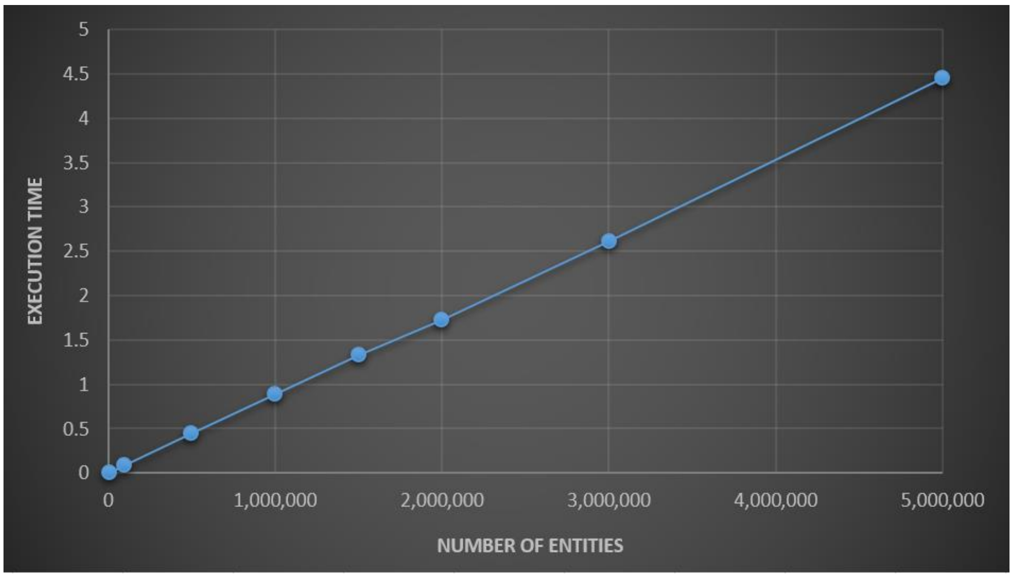


Figure 20: Performance Testing Speed Results

Number of Entities	***Execution Time (in sec)***
10,000	0.009
100,000	0.084
500,000	0.442
1,000,000	0.889
1,500,000	1.334
2,000,000	1.728
3,000,000	2.605
5,000,000	4.452

Table 4: M/M/1 Simulation Sample Speed Results

2.4.3 Test Results

During the execution of the system performance testing, we do a validation check for the input data as well similar to the unit testing. In addition to that, we check the value of the discrepancy and depending on that, we decide if the test fails or passes.

Case 1: Discrepancy is less than 0.25, Passed

Case 2: Fail error message, Failed since discrepancy was greater than 0.25

Case 3: Discrepancy is less than 0.25, Passed

Case 4: Discrepancy is less than 0.25, Passed

Case 5: Discrepancy is less than 0.25, Passed

Case 6: Discrepancy is less than 0.25, Passed

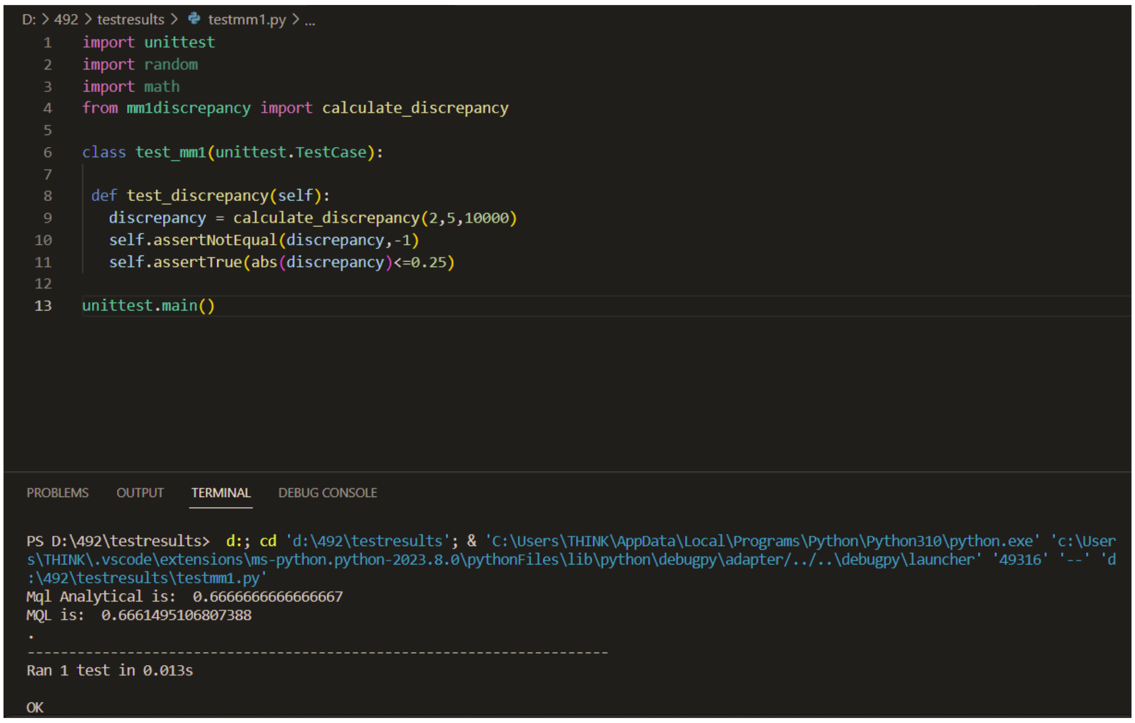
Case 7: Discrepancy is less than 0.25, Passed

Case 8: Discrepancy is less than 0.25, Passed

2.4.4 *Test Log*

Screenshots for two of the cases tested are provided below:

Case 1:



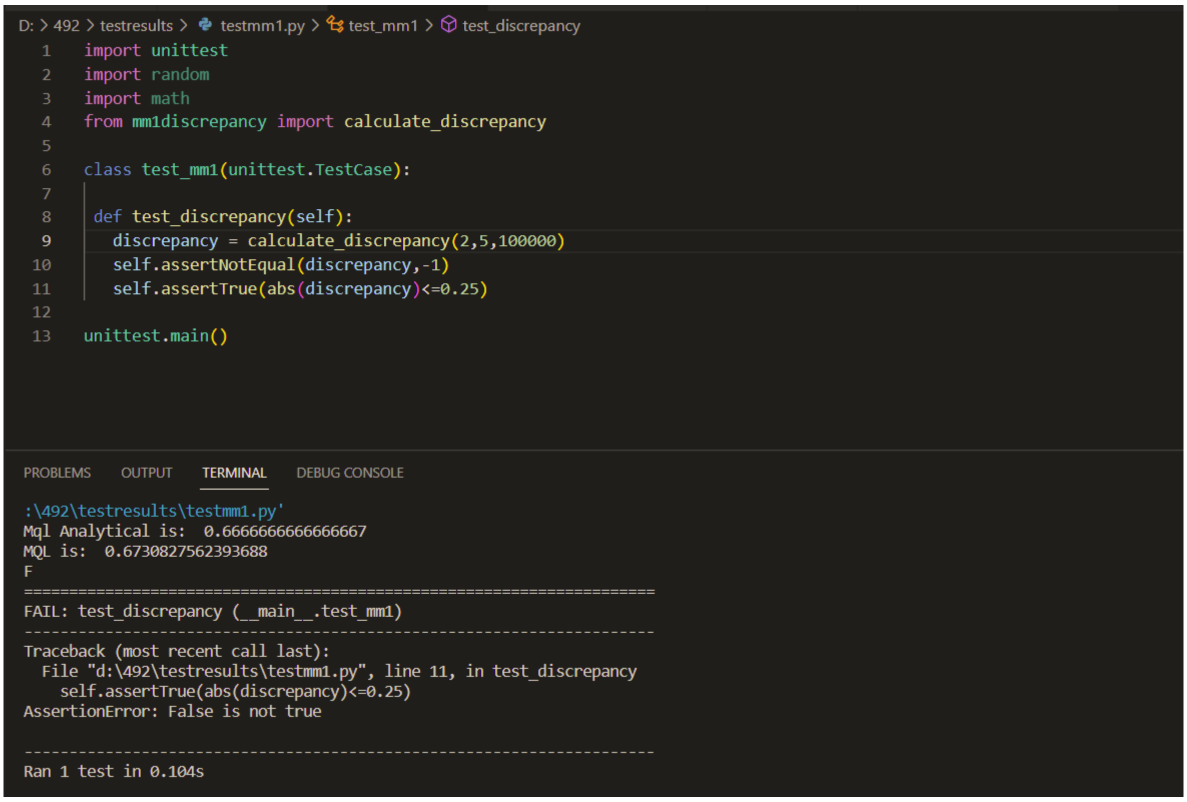
D: > 492 > testresults > testmm1.py > ...

```python
import unittest
import random
import math
from mm1discrepancy import calculate_discrepancy

class test_mm1(unittest.TestCase):

    def test_discrepancy(self):
        discrepancy = calculate_discrepancy(2,5,10000)
        self.assertNotEqual(discrepancy,-1)
        self.assertTrue(abs(discrepancy)<=0.25)

unittest.main()
```

```
PS D:\492\testresults>  d:; cd 'd:\492\testresults'; & 'C:\Users\THINK\AppData\Local\Programs\Python\Python310\python.exe' 'c:\User
s\THINK\.vscode\extensions\ms-python.python-2023.8.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49316' '--' 'd
:\492\testresults\testmm1.py'
Mql Analytical is:  0.6666666666666667
MQL is:  0.6661495106807388
.
----------------------------------------------------------------------
Ran 1 test in 0.013s

OK
```

Figure 21: Case 1 Tested - Passed

Case 2:


D: > 492 > testresults > testmm1.py > test_mm1 > test_discrepancy

```python
import unittest
import random
import math
from mm1discrepancy import calculate_discrepancy

class test_mm1(unittest.TestCase):

 def test_discrepancy(self):
   discrepancy = calculate_discrepancy(2,5,100000)
   self.assertNotEqual(discrepancy,-1)
   self.assertTrue(abs(discrepancy)<=0.25)

unittest.main()
```

```
:\492\testresults\testmm1.py'
Mql Analytical is:  0.6666666666666667
MQL is:  0.6730827562393688
F
======================================================================
FAIL: test_discrepancy (__main__.test_mm1)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "d:\492\testresults\testmm1.py", line 11, in test_discrepancy
    self.assertTrue(abs(discrepancy)<=0.25)
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.104s
```

Figure 22: Case 2 Tested - Failed

2.5 User Testing

2.5.1 Test Procedure

To conduct user testing, we selected the alpha testing method. In order to perform the user testing, we delivered the System Usability Scale (SUS) questionnaire to a selected group of participants with pre-existing Computer Engineering backgrounds. By selecting participants from the Computer Engineering department, we desired to receive feedback from individuals familiar with the concepts and requirements related to our software tool. The questionnaire covered the general user experience and various aspects of the software, including complexity, consistency, and integration of the system's functions. The participants are asked to rate the statements on a 5-point Likert scale, ranging from "Strongly Disagree" to "Strongly Agree." Their objective answers supported us in measuring their perceptions of the software's usability and identifying the parts of the system that need improvement. Using the SUS questionnaire, we aimed to get standardized feedback on our software's usability from the participants familiar with Computer Engineering. This enabled us to recognize our software's strengths and weaknesses and complete the necessary changes to improve usability and satisfy better user experiences.

2.5.2 Expectations

We expect that user testing will help us understand if there is a problematic case regarding the web application's usability. This test will provide an opportunity to evaluate the usefulness and user-friendliness of the software. By observing users' experiences with their interaction with the graphical user interface (GUI), we can determine whether the design elements, layout, and navigation are easy to use and meet the user's expectations. User testing will also allow us to validate whether the software functions are well-integrated and consistent. Regarding user feedback, we can verify that the software performs as desired and delivers the intended functionality in an easy-to-use manner. Since we aim to ensure that the software meets users' expectations and provides a satisfying experience, the users' suggestions and comments will help us improve the software product and solve any problems.

2.5.3 Test Results

To calculate the SUS score for each participant, 1 is subtracted from odd-numbered questions' scores, and the even-numbered questions' scores are subtracted from 5 (to prevent negative scores). Then, the sum of the scores per question is calculated. To convert the scale out of 100, the sum is multiplied by 2.5. These calculations are performed for each participant who answered the questionnaire, in order the explain in detail, the calculations for two participants are provided as follows:

User 1:

Question 1:	3 – 1 = 2	Question 2:	5 – 1 = 4
Question 3:	5 – 1 =4	Question 4:	5 – 2 = 3
Question 5:	5 – 1 =4	Question 6:	5 – 1 = 4
Question 7:	5 – 1 =4	Question 8:	5 – 2 = 3
Question 9:	5 – 1 =4	Question 10:	5 – 1 = 4

Sum of the scores = 36

Sum is multiplied by 2.5:

36 * 2.5 = 90

User 2:

Question 1: 5 – 1 = 4

Question 2: 5 – 1 = 4

Question 3: 5 – 1 =4

Question 4: 5 – 2 = 3

Question 5: 5 – 1 =4

Question 6: 5 – 2 = 3

Question 7: 4 – 1 =3

Question 8: 5 – 1 = 4

Question 9: 5 – 1 =4

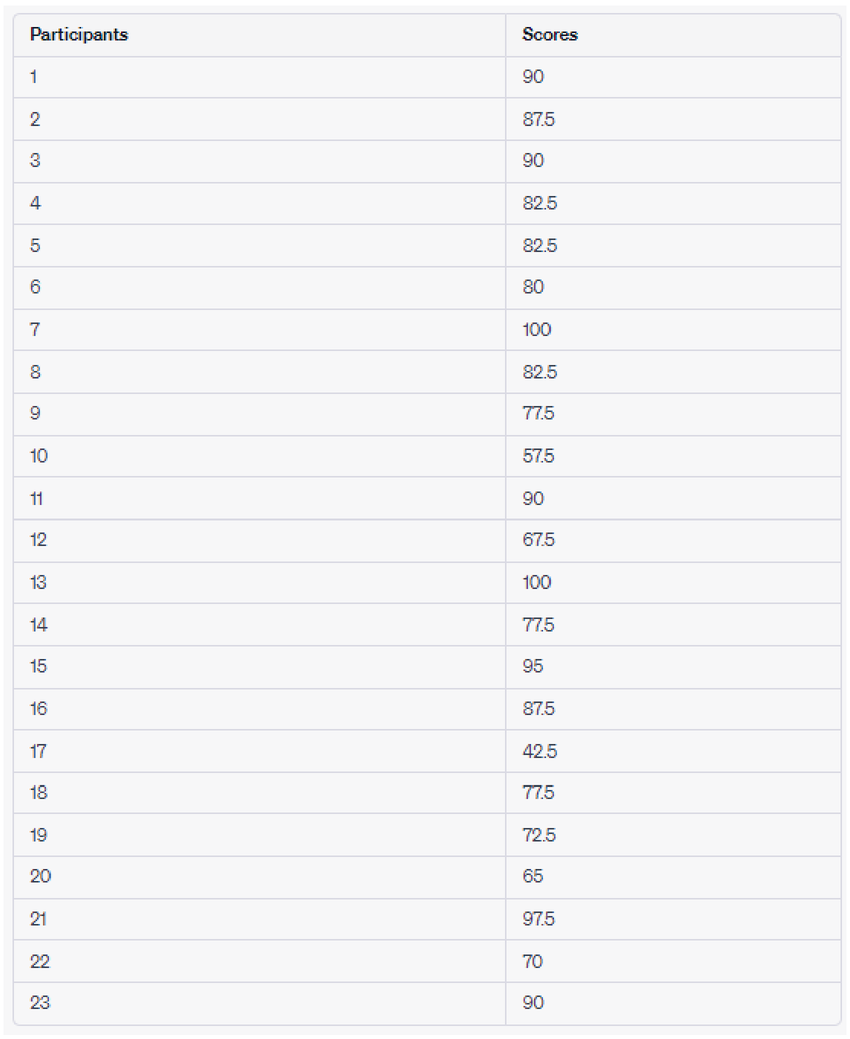
Question 10: 5 – 3 = 2

Sum of the scores = 35

Sum is multiplied by 2.5:

35 * 2.5 = 87.5

The table in the following figure displays each participant score individually:



Participants	Scores
1	90
2	87.5
3	90
4	82.5
5	82.5
6	80
7	100
8	82.5
9	77.5
10	57.5
11	90
12	67.5
13	100
14	77.5
15	95
16	87.5
17	42.5
18	77.5
19	72.5
20	65
21	97.5
22	70
23	90

Figure 23: Participant Scores

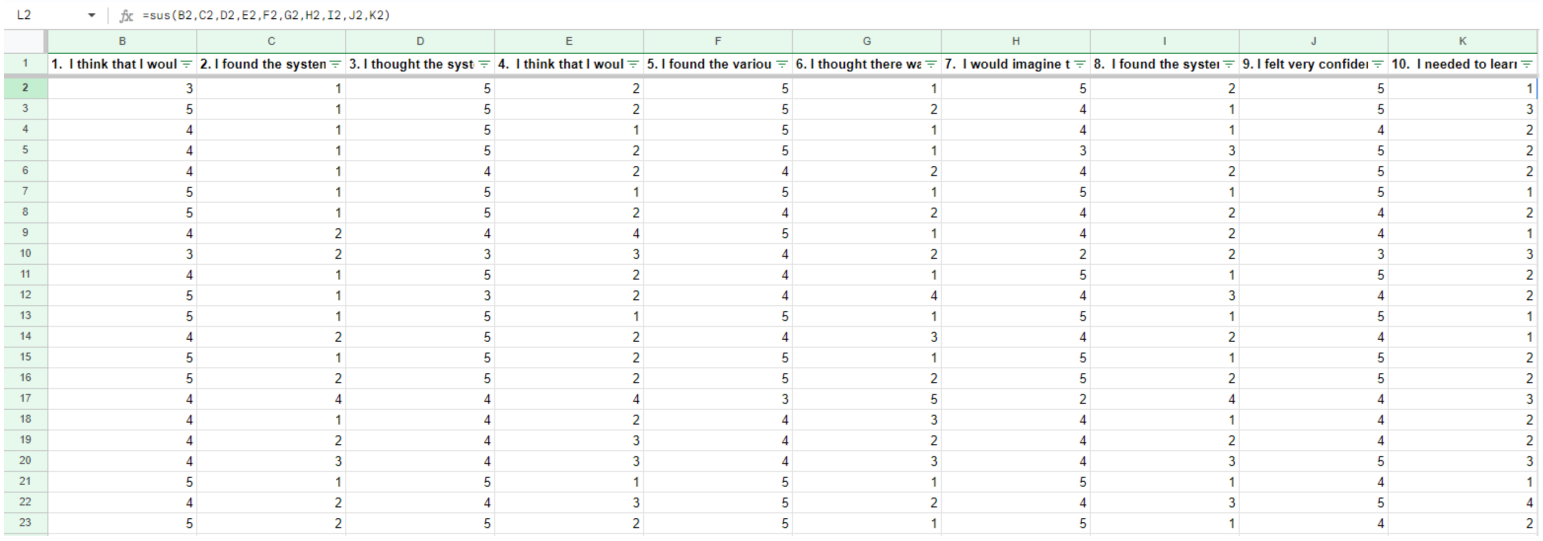
To find the final score, sum is computed and divided by the number of users that answered the questionnaire to find the average:

(90 + 87.5 + 90 + 82.5 + 82.5 + 80 + 100 + 82.5 + 77.5 + 57.5 + 90 + 67.5 + 100 + 77.5 + 95 + 87.5 + 42.5 + 77.5 + 72.5 + 65 + 97.5 + 70 + 90) / 23 = 80

The resulting score is calculated as 80. According to an article published in usability.gov (n.d), a SUS score greater than 68 is considered above the average, so it can be concluded that the SUS result of the simulation software tool is above the average. As a result, the simulation software meets the expectations of the users in many aspects, and the user testing is successful.

2.5.4 *Test Log*

Total of 23 participant answered the SUS questionnaire. The results are transferred into the Google tables as follows:



L2 =sus(B2,C2,D2,E2,F2,G2,H2,I2,J2,K2)

	B	C	D	E	F	G	H	I	J	K
1	1. I think that I woul	2. I found the systen	3. I thought the syst	4. I think that I woul	5. I found the variou	6. I thought there wa	7. I would imagine t	8. I found the systei	9. I felt very confidei	10. I needed to leari
2	3	1	5	2	5	1	5	2	5	1
3	5	1	5	2	5	2	4	1	5	3
4	4	1	5	1	5	1	4	1	4	2
5	4	1	5	2	5	1	3	3	5	2
6	4	1	4	2	4	2	4	2	5	2
7	5	1	5	1	5	1	5	1	5	1
8	5	1	5	2	4	2	4	2	4	2
9	4	2	4	4	5	1	4	2	4	1
10	3	2	3	3	4	2	2	2	3	3
11	4	1	5	2	4	1	5	1	5	2
12	5	1	3	2	4	4	4	3	4	2
13	5	1	5	1	5	1	5	1	5	1
14	4	2	5	2	4	3	4	2	4	1
15	5	1	5	2	5	1	5	1	5	2
16	5	2	5	2	5	2	5	2	5	2
17	4	4	4	4	3	5	2	4	4	3
18	4	1	4	2	4	3	4	1	4	2
19	4	2	4	3	4	2	4	2	4	2
20	4	3	4	3	4	3	4	3	5	3
21	5	1	5	1	5	1	5	1	4	1
22	4	2	4	3	5	2	4	3	5	4
23	5	2	5	2	5	1	5	1	4	2

Figure 24: SUS Questionnaire Results

The questionnaire link is: https://forms.gle/yWEtntULhtSVBgdv8

3 Test Tools

To perform user testing, we created a questionnaire using Google Forms. First, we selected an empty form and added the title, explanation, and questions. Then we add a header image that displays the name of our software simulation tool. While adding questions, we selected multiple-choice and text as answer types as multiple choice. To have information about questions and view the design of the questionnaire form, you can visit the link provided in section 2.5.4. Another tool we used is PyUnit, which is a Python framework used to do mainly unit testing. It was used to test functionalities for M/M/1, M/M/1/L, M/M/C, M/M/C/L. Based on the results we pass or fail, we test the system performance accuracy as well. Several test cases were tested with PyUnit and sections 2.2.1 and 2.4.1 can be checked for further reference.

4 Test Incidents

We didn't face any major issues while performing any of the testing. The cases for all the testing types are completed successfully, so we don't need to make any further improvements to the project. As a minor issue in the performance testing, it can be seen from the Case 2, see Figure21, that the discrepancy check is failed as it is larger than 0.25. To fix this problem, the margin of the discrepancy accuracy can be fixed to a higher value or the value entered for number of entities can be increased, 100,000 is entered in the failed case. Related sections can be checked to see the test cases performed and their results for each testing type.

5 Summary

5.1 Summary of the Tests Passed

Unit Testing:

- Test Case #1: All inputs were validated for M/M/1, and the plotted graph was successful with the expected convergence and mql value.
- Test Case #2: All inputs were validated for M/M/1, and the plotted graph was successful with the expected convergence and mql value.
- Test Case #3: All inputs were validated for M/M/1, and the plotted graph was successful with the expected convergence and mql value.

- Test Case #4: All inputs were validated for M/M/1/L, and the plotted graph was successful with the expected convergence and mql value.
- Test Case #5: All inputs were validated for M/M/1/L, and the plotted graph was successful with the expected convergence and mql value.

Integration Testing:

- Test Case #1: Connection of each page was validated by successfully switching between all the pages.
- Test Case #2: Drag-and-drop feature was validated by successfully taking the items from the Toolbox and putting them into the canvas.
- Test Case #3: All the input values entered for the selected Queue and Server(s) were validated successfully.
- Test Case #4: All the input values entered for the selected Queue and Server(s) were validated by checking the values in the main.py file from the PyCharm console.
- Test Case #5: All the input values entered for the selected Queue and Server(s) were validated by checking the displayed values in the corresponding .js files.
- Test Case #6: All the graphs plotted were validated by checking the outputs in the graphs according to the inputs entered for those cases, and the all the graphs were successfully displayed as expected.

System Testing:

- All the sample scenarios created and tested from the beginning of the program, Home Page, till the end, corresponding graph plotting, were successful and the steps for each of them are provided in 2.3.2.

Performance Testing:

- Test Case #1: All inputs were validated for M/M/1 and the discrepancy value was less than 0.25.
- Test Case #3: All inputs were validated for M/M/1 and the discrepancy value was less than 0.25.
- Test Case #4: All inputs were validated for M/M/1 and the discrepancy value was less than 0.25.
- Test Case #5: All inputs were validated for M/M/1/L and the discrepancy value was less than 0.25.

- Test Case #6: All inputs were validated for M/M/1/L and the discrepancy value was less than 0.25.
- Test Case #7: All inputs were validated for M/M/1/L and the discrepancy value was less than 0.25.
- Test Case #8: All inputs were validated for M/M/1/L and the discrepancy value was less than 0.25.

5.2 Summary of the Tests Failed

Performance Testing:

- Test Case #2, discrepancy was greater than 0.25.
  - ➔ This test item can be improved by either fixing the value of the margin of the discrepancy accuracy to a higher value or increasing the value entered for number of entities.

No other failure occurred in any of the other testing types performed.

6 References

Assistant Secretary for Public Affairs. (2013, September 6). *System usability scale (SUS)*. Usability.gov. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Running PyUnit tests and selenium tests created with PyUnit. (2023). https://support.smartbear.com/testcomplete/docs/working-with/integration/unit-test-frameworks/pyunit.html#:~:text=PyUnit%20is%20a%20standard%20unit,the%20tests%20to%20be%20run.