ECE 567 : Software Engineering I

Minor Project: Problem #2

October, 2021

Ian Herrighty
Gregory Giovannini
Eric Schreiber
Dawn Park

Abstract:

This report details the development of a mathematical software program to calculate customer ("rider") wait times for a ride-share service, Lyfter. The program plots rider and driver positions in the area of service, efficiently pairs riders and drivers, and then calculates wait times for all riders. The program can be run in multiple iterations with varying number of drivers so that Lyfter can calculate wait times vs. number of available drivers.

Development of the program utilized an agile process model executed in two sprints. The first sprint established a working prototype, and the second sprint refined distributions and program execution to provide more accurate data.

The developed program provides the means to find wait times for 100 riders given a user-defined number of drivers, and a defined service area and location distribution functions. The development team concludes that the program could be improved in the future by utilizing road-based distance calculations and finding wait times in a dynamic model (e.g. where riders/drivers re-populate over time).

Problem Statement (#2):

Lyfter is an app-based personal vehicle system (i.e., ride-hailing), and they approach your team for help in designing a system that allows them to efficiently match riders to drivers. They want your team to help them claim "the wait time is never more than 5 minutes", and thus in determining how many drivers they should recruit if they anticipate 100 riders at any instant of time.

Development Plan:

Our team will utilize the agile development model to produce a product to meet the customer's needs efficiently and to manage change. Development will be conducted in two sprints:

Sprint 1: 6 OCT 2021 to 16 OCT 2021
Sprint 1 will develop a basic prototype to demonstrate basic functionality.
Sprint 2: 17 OCT 2021 to 25 OCT 2021
Sprint 2 will produce a shippable product.

The Project Team consists of four students from Rutgers University's School of Graduate Studies. To ensure productive collaboration and equity of learning value, all team members have contributed to each Software Development Process; however, for accountability and documentation purposes, each team member assumes lead responsibilities for the following:

Ian Herrighty: Software Specification
Eric Schreiber: Integration and Tools
Greg Giovannini: Design and Implementation
Dawn Park: Test and evaluation

The product will be developed in Python 3.7. Configuration will be managed by using Github.com.

Testing will be conducted using Virtual Environments in both Windows 10 and Linux 20.04 Operating Systems.

Key Terms / Definitions:

Rider: a customer wishing to use Lyfter for transportation from **start point** to **destination**

Driver: Lyfter personnel providing vehicle transportation

Wait time: The time a rider is waiting for the driver to arrive.

SPRINT 1

Requirements Engineering:

The following customer requirements are extracted from the problem statement. These are considered the high level requirements the software must satisfy:

Req #	Requirement Text	Requirement Rationale
CUS-1	The system shall match riders to drivers	Verbatim from problem statement
CUS-2	The system shall determine rider wait time as a function of the number of riders and drivers	Customer objective is to determine number of drivers needed for a wait time <5min with 100 riders

The problem statement does not contain enough detail to proceed directly to specification and design. Therefore, the project team proposes the following ground rules and assumptions to the customer in order to bound/guide the design.

GR #	Ground Rule	Ground Rule Rationale
GR-1	Lyfter’s area of service is defined as the square bounded by US National Grid Coordinates 18T WK 4681 and 18T WK 5085	No information provided by customer.
GR-2	At t=0 100 riders will be distributed in the area of service	Provided by customer

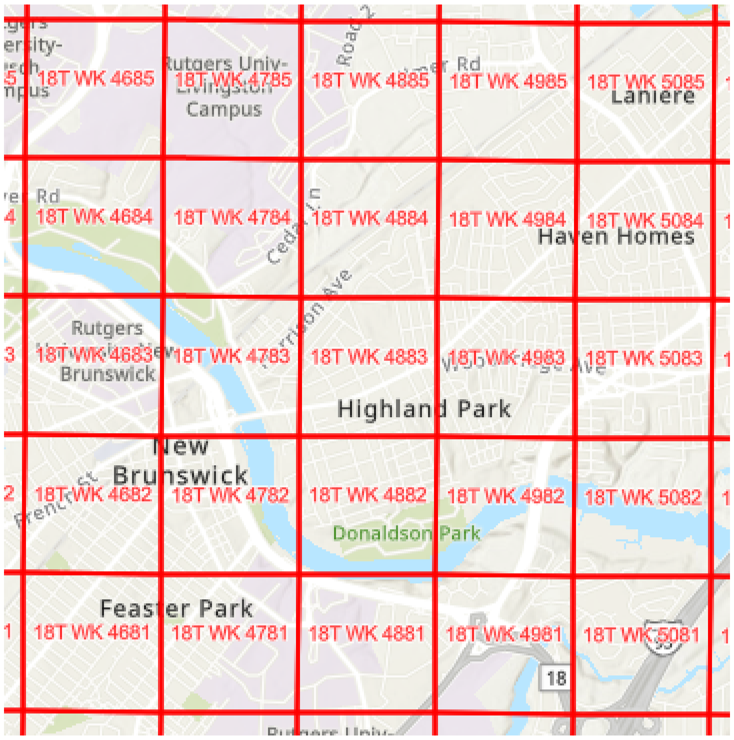


Figure 1: Area of Service for simulations

AS #	Assumption	Assumption Rationale
AS-1	Rider position distribution is assumed to be random.	Simplicity for first sprint.
AS-2	Driver position distribution is assumed to be random.	Simplicity for first sprint.
AS-3	Drivers are assumed to move at velocity = 15 kilometers per hour	Average of urban traffic, reduced to compensate for “straight” distance vs over-road distance

System Requirements:

System requirements have been developed based on the Problem Statement and in consideration of the aforementioned Ground Rules and Assumptions. System Requirements are tabulated and include several attributes which are updated throughout the sprint:

- **Requirement Number**: A unique identifier for the requirement for traceability
- **Requirement Text:** The requirement statement.
- **Requirement Rationale:** The justification for why the requirement is written (informs requirement validation).
- **Validation (Method):** The validation status at the end of Sprint 1 (Passed or Failed) and the method of requirement validation. Additional notes may be included in italics.
- **Verification (Method):** The verification status at the end of Sprint 1 (Passed/Failed/Not tested) and the method of requirement validation. Additional notes may be included in italics.

Req #	Requirement Text	Requirement Rationale	Validation (Method)	Verification (Method)
REQ-1	*System Inputs*	N/A (section title)	N/A	N/A
REQ-2	The program shall accept the number of drivers as a user-defined input, N	For purpose of determining wait time by number of drivers	Passed (peer review)	Passed (demonstration)
REQ-3	N values less than 100 shall be invalid	Less than 100 drivers available will result in wait times exceeding 5 minutes (driver must complete trip before serving second rider). Future sprints may "spawn" new drivers as a function of time.	Passed (peer review)	Not tested
REQ-4	Non-integer N values shall be invalid	N value represents number of drivers	Passed (peer review)	Not tested
REQ-5	The program shall define N drivers from driver_1 to driver_N	Derived requirement for data formatting	Passed (peer review)	Passed (system test)

REQ-6	The program shall define 100 riders rider_1 to rider_100	Customer expects 100 riders at any point in time	Passed (peer review)	Passed (unit test)
REQ-7	*Rider distribution*	N/A (section title)	N/A	N/A
REQ-8	The program shall generate a random easterly grid coordinate between 4600 and 5000 for each rider, rider_[number]_X	This will generate a random 10m grid square location for each rider	Passed (peer review)	Passed (unit test)
REQ-9	The program shall generate a random northerly grid coordinate between 8100 and 8500 for each rider, rider_[number]_Y	This will generate a random 10m grid square location for each rider	Passed (peer review)	Passed (unit test)
REQ-10	*Driver distribution*	N/A (section title)	N/A	N/A
REQ-11	The program shall generate a random easterly grid coordinate between 4600 and 5000 for each driver, driver_[number]_X	This will generate a random 10m grid square location for each driver	Passed (peer review)	Passed (unit test)
REQ-12	The program shall generate a random northerly grid coordinate between 8100 and 8500 for each driver, driver_[number]_Y	This will generate a random 10m grid square location for each driver	Passed (peer review)	Passed (unit test)
REQ-13	*Wait time computation*	N/A (section title)	N/A	N/A
REQ-14	The program shall calculate the distance between all drivers and all riders	In order to pair drivers/riders, distances must be known	Passed (peer review)	Failed (system test, analysis) *Root cause analysis of REQ-16 test failure determined inconsistent units used in distance calculation*
REQ-15	The program shall find the closest driver for each rider	Necessary to pair riders/drivers	Passed (peer review)	Passed (system test)

REQ-16	Where a driver is closest for multiple riders, the program shall pair the driver with the closest rider	Drivers and riders must be one-to-one paired	Passed (peer review)	Passed (system test)
REQ-17	The program shall pair one driver to one rider	Customer model of service	Passed (peer review)	Passed (system test)
REQ-18	The program shall calculate wait time	This is the result Lyfter wants to assess	Passed (peer review)	Failed (system test) *Testing revealed erroneously low wait times*
REQ-19	*Data processing*	N/A (section title)	N/A	N/A
REQ-20	The program shall generate a bar chart of number of drivers serviced by corresponding time interval (1 minute intervals)	Visual representation of data	Failed (peer review) *Sprint 2 will change requirement*	Not tested
REQ-21	The program shall display fraction of drivers serviced in under 5 minutes	Visual representation of data	Failed (peer review) *Sprint 2 will change requirement*	Not tested
REQ-22	The program shall be capable of graphically displaying maximum rider wait time as a function of driver quantity	Visual representation of data	Passed (peer review)	Passed (system test)

SYSTEM MODEL

A simple activity diagram for the program is shown in Figure 1. There is only one decision gate to determine whether the user input is valid.

If the program receives an N value less than 100 drivers, this will be considered an invalid input that will result in terminating the program. See requirements REQ-4 and REQ-5.

If the program receives a valid N value it will proceed to the identified activities in sequence until the program is complete. No additional user input or decision gates are utilized during Sprint 1.

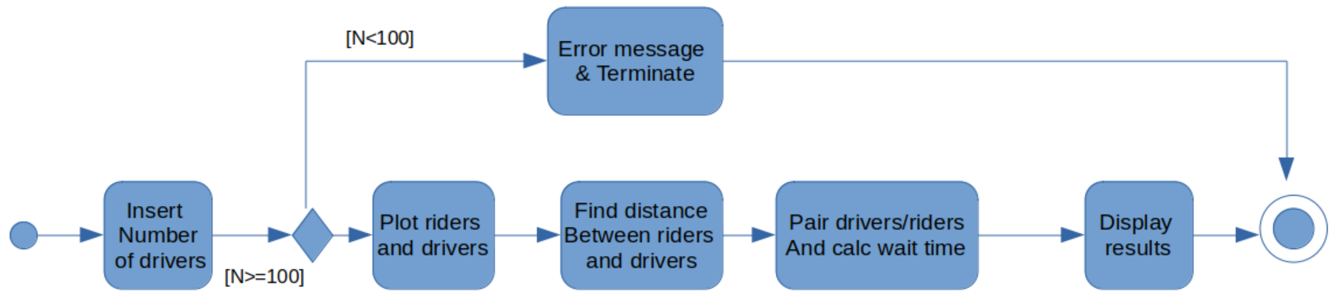

[N<100]
Error message & Terminate
Insert Number of drivers
[N>=100]
Plot riders and drivers
Find distance Between riders and drivers
Pair drivers/riders And calc wait time
Display results

Figure 2: Lyfter Activity Diagram

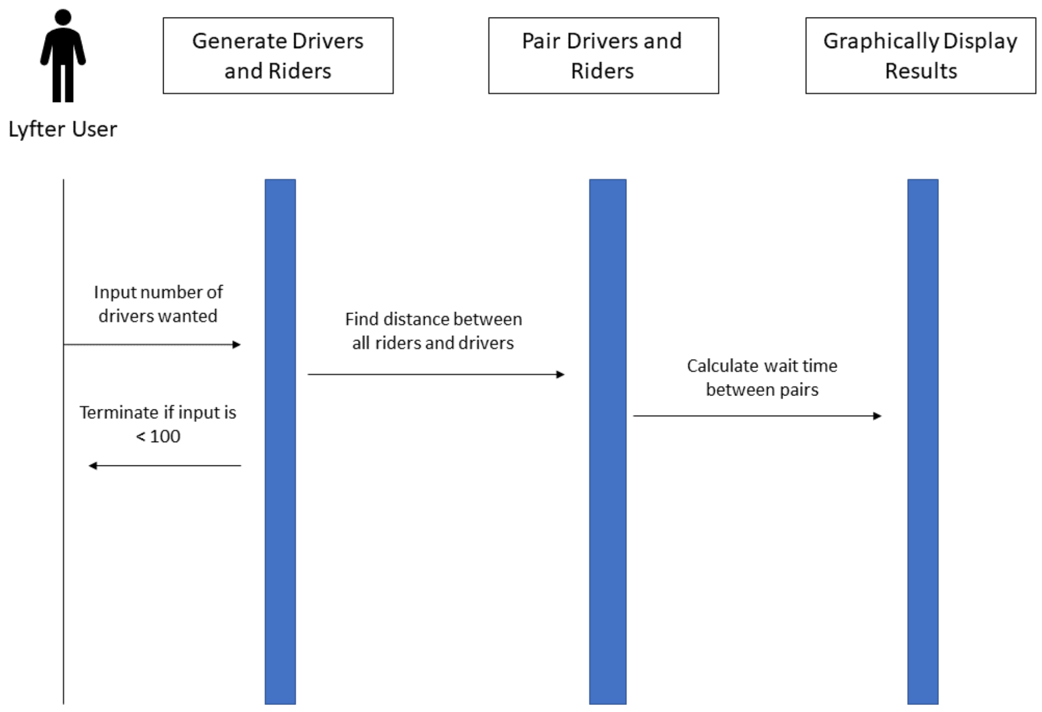
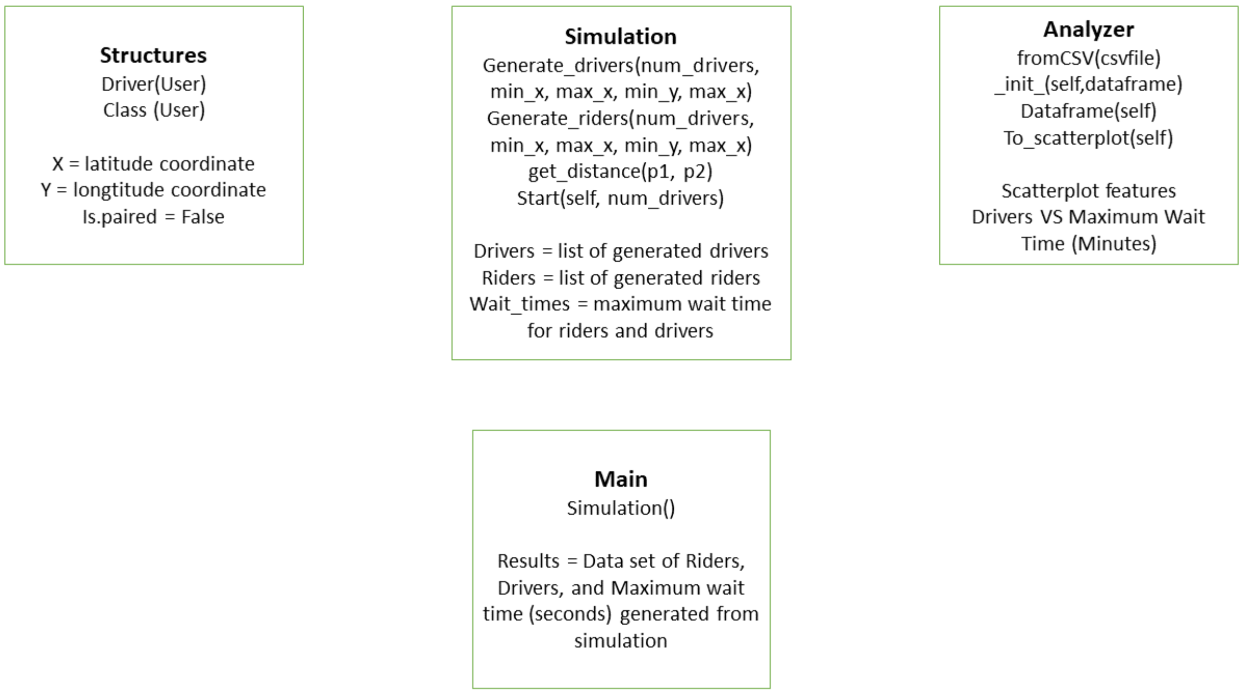

Lyfter User
Generate Drivers and Riders
Pair Drivers and Riders
Graphically Display Results
Input number of drivers wanted
Terminate if input is < 100
Find distance between all riders and drivers
Calculate wait time between pairs

Figure 3: Lyfter Sequence Diagram

SOFTWARE DESIGN

During Sprint 1 the software design solution was organized into the following files:

- main.py : Main program for executing the operation based on user input and providing the desired output. The main program is the user interface for the product.
- constants.py : defines constants which will be used for calculations
- structures.py : defines classes for drivers and riders
- simulation.py : Creates lists / arrays for drivers and riders, and contains the mathematical equations necessary to compute distances, pair riders/drivers, and determine wait times.
- analyze.py : Graphically displays program results
- tests.py : Used for testing the program



Structures
Driver(User)
Class (User)
X = latitude coordinate
Y = longtitude coordinate
Is.paired = False
Simulation
Generate_drivers(num_drivers, min_x, max_x, min_y, max_x)
Generate_riders(num_drivers, min_x, max_x, min_y, max_x)
get_distance(p1, p2)
Start(self, num_drivers)
Drivers = list of generated drivers
Riders = list of generated riders
Wait_times = maximum wait time for riders and drivers
Analyzer
fromCSV(csvfile)
init(self,dataframe)
Dataframe(self)
To_scatterplot(self)
Scatterplot features
Drivers VS Maximum Wait Time (Minutes)
Main
Simulation()
Results = Data set of Riders, Drivers, and Maximum wait time (seconds) generated from simulation

Figure 4: Lyfter Software Architecture Diagram

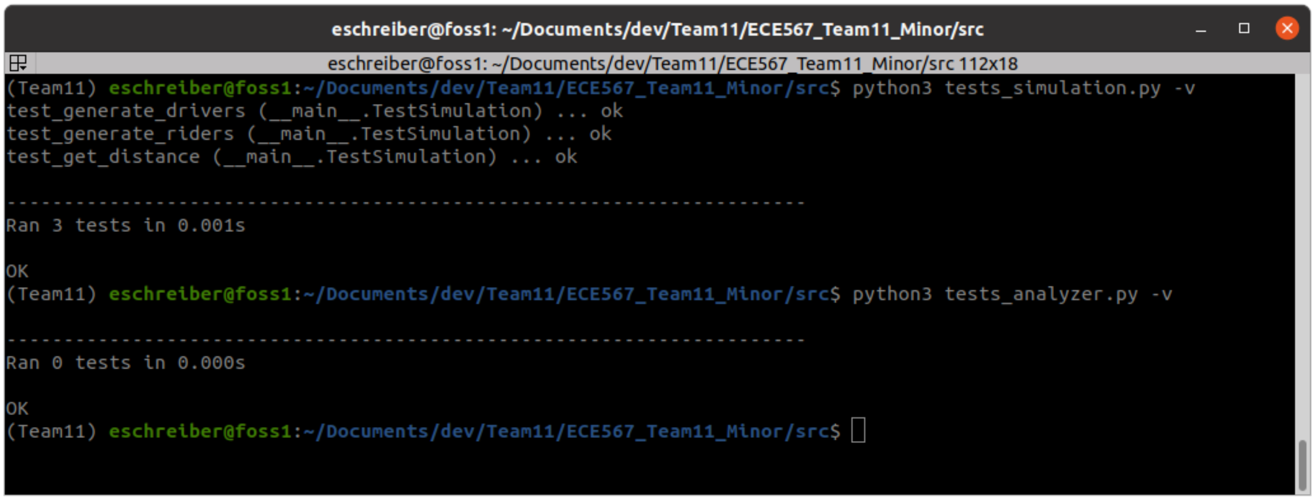
Simulation.py contains the most significant logic for meeting the customer's requirements, and is where the majority of design effort during Sprint 1 was focused. The code meets the system requirements and follows the activity diagram (figure 1) by implementing the following:

1. Constants, classes, and python libraries are imported to the program
2. Lists are generated for driver and rider positions (random float lat and long)
3. Distances are calculated between all drivers and all riders and placed in an array
4. Drivers and riders are paired by finding the pair with minimum distance
5. Wait time is determined by average velocity
6. The program returns maximum wait times

TESTING

Unit testing:
Three unit tests were devised to test the TestSimulation class. Each test corresponds to a test of one method:

1. test_generate_drivers: tests that the number of generated drivers corresponds to the number requested and that each driver's location is within the bounding rectangle.
2. test_generate_riders: tests that the number of generated riders corresponds to the number requested and that each rider's location is within the bounding rectangle.
3. test_get_distance: tests the distance function correctly calculates the diagonal distance of the bounding rectangle


eschreiber@foss1: ~/Documents/dev/Team11/ECE567_Team11_Minor/src
eschreiber@foss1: ~/Documents/dev/Team11/ECE567_Team11_Minor/src 112x18

```
(Team11) eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$ python3 tests_simulation.py -v
test_generate_drivers (__main__.TestSimulation) ... ok
test_generate_riders (__main__.TestSimulation) ... ok
test_get_distance (__main__.TestSimulation) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
(Team11) eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$ python3 tests_analyzer.py -v

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
(Team11) eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$ 
```

Figure 5: Simulation Unit Test Screen Capture

A placeholder was created to test the Analyzer class, but no tests were defined in this sprint.

System Testing:
A system test was devised to determine the number N of randomly placed drivers required to limit the maximum pick up time to less than 5 minutes for 100 randomly placed riders within a bounding rectangle of dimensions 4 kilometers by 4 kilometers. The simulation was run for 100 iterations over a range of 100 to 150 drivers. The maximum pick up time was noted for each iteration and the values were plotted for each value N. Results shown in Figure 2.

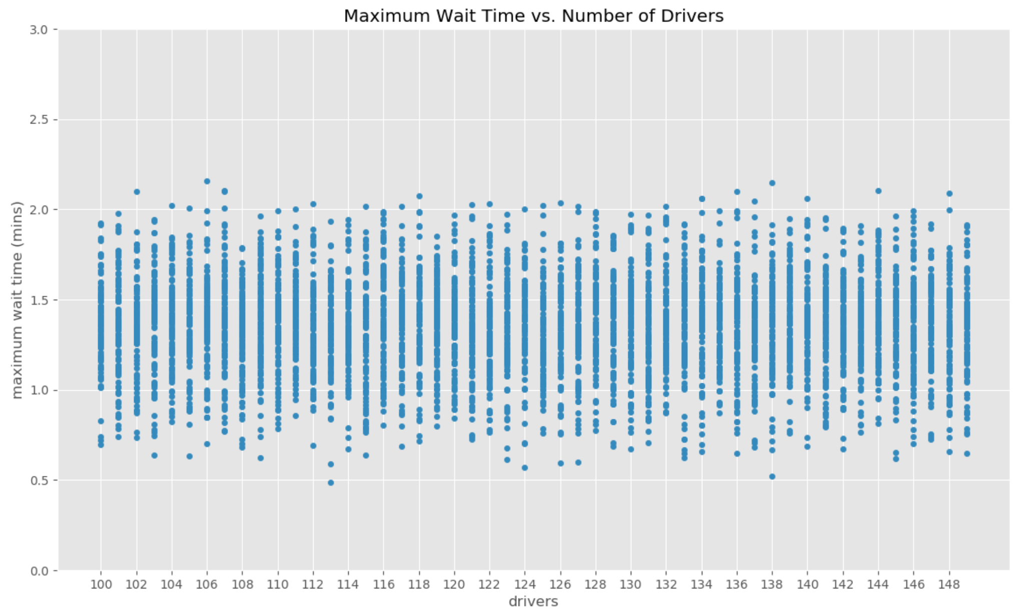


Figure 6: Maximum Rider Wait Time vs. Number of Drivers (20 iterations each)

The graph shows a nearly constant maximum pickup time of approximately 2 minutes for the entire range of drivers 100 to 150, which was not anticipated. A misunderstanding of the bounding rectangle units of distance was discovered on review - the bounding box is intended to be a 4 kilometer square but a 400 meter square was used in simulation. This will be fixed in the next iteration.

SPRINT 1 CONCLUSION

Sprint 1 concluded on October, 16, 2021 and produced a working program which provides a successful foundation for further development.

Testing conducted during Sprint 1 identified shortcomings in results and functionality which may be resolved with these improvements:

1. An error occurred during coding which resulted in erroneously low wait times (distance calculated in 1m increments vs 10m). This was detected via system testing and will be resolved in Sprint 2.
2. Analysis.py should provide greater data resolution so that testing can confirm correct functionality at lower levels of detail.
3. The program should model a more realistic rider position distribution than “random”
4. The product should consider actively-controlled driver distribution models which produce lower wait times.

SPRINT 2

Requirements Engineering:

For sprint 2 there are NO CHANGES to the Customer requirements or Ground Rules, and Assumptions. The Customer requirements, Ground Rules, and Assumptions from Sprint 1 apply to Sprint 2 but are not duplicated in this section.

Software Design:

For sprint 2, we added a new file and updated pre-existing ones for the software design solution:

- csvutils.py: Writes csv files and records their locations
- analyze.py: Added a class function, 'LocationsAnalyzer' that graphically displays location results. The initial 'Analyzer" class function only graphically displayed drivers and maximum wait times.
- simulation.py: Added a class method, 'validate_constants' that logically validates the inputs in the constants.py file in case the user experiments with changing some of the assumed values.
- constants.py: Added the number of Drivers as a range, specified by MIN_DRIVERS (the minimum number of Drivers), MAX_DRIVERS (the maximum number of Drivers) and STEP_DRIVERS (the amount of Drivers to increment by after each NUM_ITERATIONS iterations). NUM_ITERATIONS determines how many times each number of Drivers shall be tested.

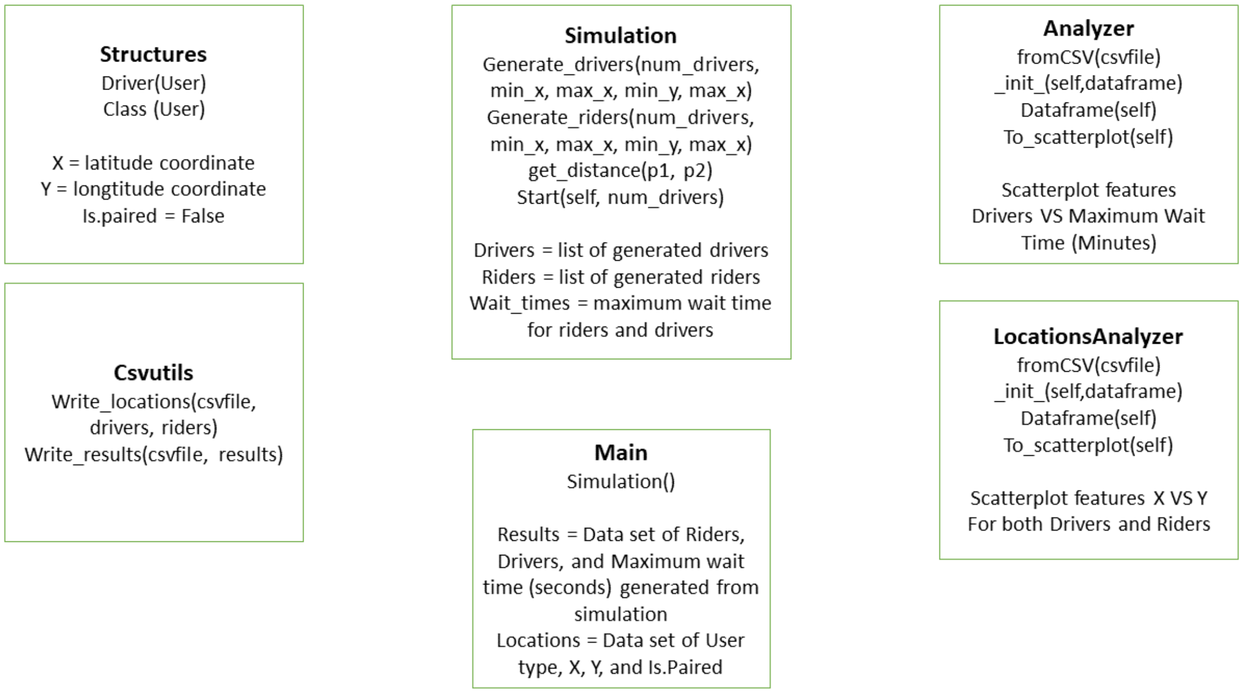

Structures
Driver(User)
Class (User)
X = latitude coordinate
Y = longtitude coordinate
Is.paired = False
Csvutils
Write_locations(csvfile, drivers, riders)
Write_results(csvfile, results)
Simulation
Generate_drivers(num_drivers, min_x, max_x, min_y, max_x)
Generate_riders(num_drivers, min_x, max_x, min_y, max_x)
get_distance(p1, p2)
Start(self, num_drivers)
Drivers = list of generated drivers
Riders = list of generated riders
Wait_times = maximum wait time for riders and drivers
Main
Simulation()
Results = Data set of Riders, Drivers, and Maximum wait time (seconds) generated from simulation
Locations = Data set of User type, X, Y, and Is.Paired
Analyzer
fromCSV(csvfile)
init(self,dataframe)
Dataframe(self)
To_scatterplot(self)
Scatterplot features Drivers VS Maximum Wait Time (Minutes)
LocationsAnalyzer
fromCSV(csvfile)
init(self,dataframe)
Dataframe(self)
To_scatterplot(self)
Scatterplot features X VS Y For both Drivers and Riders

Figure 7: Lyfter Software Architecture (Sprint 2)

System Requirements:

System requirements have been developed based on the Problem Statement and in consideration of the aforementioned Ground Rules and Assumptions. System Requirements are tabulated and include several attributes which are updated throughout the sprint:

- **Requirement Number**: A unique identifier for the requirement for traceability
- **Requirement Text:** The requirement statement.
- **Requirement Rationale:** The justification for why the requirement is written (informs requirement validation).
- **Validation (Method):** The validation status at the end of Sprint 2 (Passed or Failed) and the method of requirement validation. Additional notes may be included in italics.
- **Verification (Method):** The verification status at the end of Sprint 2 (Passed/Failed/Not tested) and the method of requirement validation. Additional notes may be included in italics.

For Sprint 2 the requirements are refined in order to implement the improvements identified in the conclusion of Sprint 1. Requirement ID tags are unchanged for previously existing requirements to ensure traceability.

The new and modified requirements address the improvements from Sprint 1:

- *The program should model a more realistic rider position distribution than "random":*
    - Rider location determination is changed from "random" to "normal" distribution models. This is considered to be more representative of rider distribution in an urban environment.
    - The center of the distribution is the center of the service area with standard deviation of 300m
- *The product should consider actively-controlled driver distribution models which produce lower wait times.*
    - The program will distribute drivers in "quadrants" of responsibility
    - This represents a model where lyfter assigns drivers designated areas of responsibility to assure drivers are spread out to service the city center and outlying areas within a 5 minute wait time

Req #	Requirement Text	Requirement Rationale	Validation (Method)	Verification (Method)
REQ-1	*System Inputs*	N/A (section title)	N/A	N/A

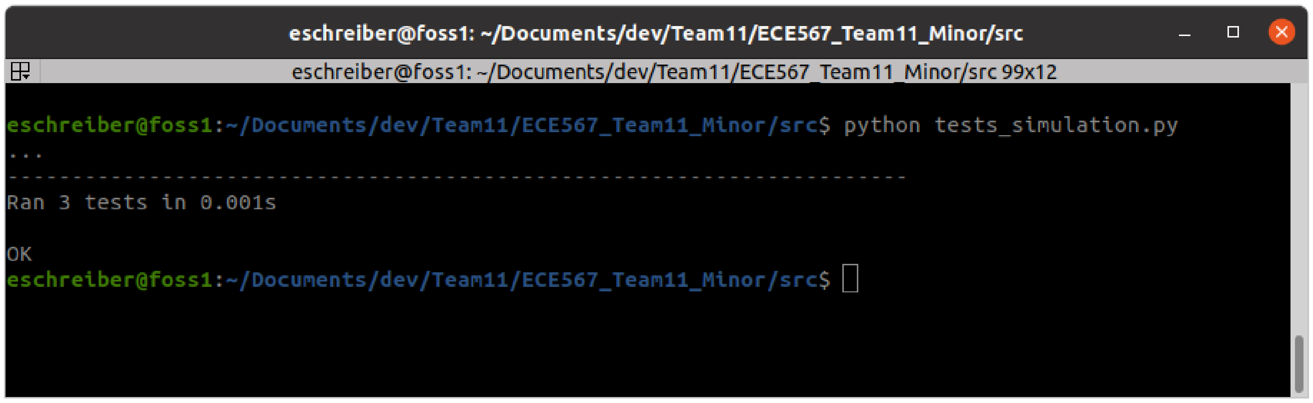
REQ-2	The program shall accept the number of drivers as a user-defined input, N	For purpose of determining wait time by number of drivers	See sprint 1	Passed (regression testing)
REQ-3	N values less than 100 shall be invalid	Less than 100 drivers available will result in wait times exceeding 5 minutes (driver must complete trip before serving second rider). Future sprints may “spawn” new drivers as a function of time.	See sprint 1	Passed (regression testing)
REQ-4	Non-integer N values shall be invalid	N value represents number of drivers	See sprint 1	Passed (regression testing)
REQ-5	The program shall define N drivers from driver_1 to driver_N	N/A (section title)	N/A	N/A
REQ-6	The program shall define 100 riders rider_1 to rider_100	Customer expects 100 riders at any point in time	See sprint 1	Passed (regression testing)
REQ-7	*Rider distribution*	N/A (section title)	N/A	N/A
REQ-8	The program shall use a normal distribution to generate a grid coordinate between 4600 and 5000 for each rider, rider_[number]_X	A normal distribution will more-closely model rider locations in an urban environment.	Passed (peer review)	Passed (system test)
REQ-9	The program shall use a normal distribution to generate a grid coordinate between 8100 and 8500 for each rider, rider_[number]_Y	A normal distribution will more-closely model rider locations in an urban environment.	Passed (peer review)	Passed (system test)
REQ-23	The center of the normal distribution shall be grid coordinate 4800, 8300.	This is the center of the defined area of service.	Passed (peer review)	Passed (system test)

REQ-24	The standard deviation of the rider normal distribution shall be 300m	This value will mean that ~95% of riders will be located within 600m of the service area center.	Passed (peer review)	Passed (system test)
REQ-10	*Driver distribution*	N/A (section title)	N/A	N/A
REQ-11	The program shall use a normal distribution to generate a grid coordinate between 4600 and 5000 for each driver, driver_[number]_X	This will generate a random 10m grid square location for each driver	Passed (peer review)	Passed (system test)
REQ-12	The program shall use a normal distribution to generate a grid coordinate between 8100 and 8500 for each driver, driver_[number]_Y	This will generate a random 10m grid square location for each driver	Passed (peer review)	Passed (system test)
REQ-25	The standard deviation of the driver normal distribution shall be 200m	This value will result in ~95% of drivers will be within 400m of the center of their quadrant	Passed (peer review)	Passed (system test)
REQ-26	25% of drivers shall be distributed around center coordinate 4700, 8200	This is the center of the south west quadrant	Passed (peer review)	Passed (system test)
REQ-27	25% of drivers shall be distributed around center coordinate 4700, 8400	This is the center of the north west quadrant	Passed (peer review)	Passed (system test)
REQ-28	25% of drivers shall be distributed around center coordinate 4900, 8200	This is the center of the south east quadrant	Passed (peer review)	Passed (system test)
REQ-29	25% of drivers shall be distributed around center coordinate 4900, 8400	This is the center of the north east quadrant	Passed (peer review)	Passed (system test)
REQ-13	*Wait time computation*	N/A (section title)	N/A	N/A
REQ-14	The program shall calculate the distance between all drivers and all riders	In order to pair drivers/riders, distances must be known	See sprint 1	Passed (regression testing)

REQ-15	The program shall find the closest driver for each rider	Necessary to pair riders/drivers	See sprint 1	Passed (regression testing)
REQ-16	Where a driver is closest for multiple riders, the program shall pair the driver with the closest rider	Drivers and riders must be one-to-one paired	See sprint 1	Passed (regression testing)
REQ-17	The program shall pair one driver to one rider	Customer model of service	See sprint 1	Passed (regression testing)
REQ-18	The program shall calculate wait time	This is the result Lyfter wants to assess	See sprint 1	Passed (regression testing)
REQ-19	*Data processing*	N/A (section title)	N/A	N/A
REQ-20	[DELETED]	N/A	N/A	N/A
REQ-21	[DELETED]	N/A	N/A	N/A
REQ-22	The program shall be capable of graphically displaying maximum rider wait time as a function of driver quantity	Visual representation of data	See sprint 1	Passed (regression testing)

TESTING

Unit testing:
The three simulation tests propagated forward from Sprint 1 pass:


eschreiber@foss1: ~/Documents/dev/Team11/ECE567_Team11_Minor/src
eschreiber@foss1: ~/Documents/dev/Team11/ECE567_Team11_Minor/src 99x12

```
eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$ python tests_simulation.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$ 
```

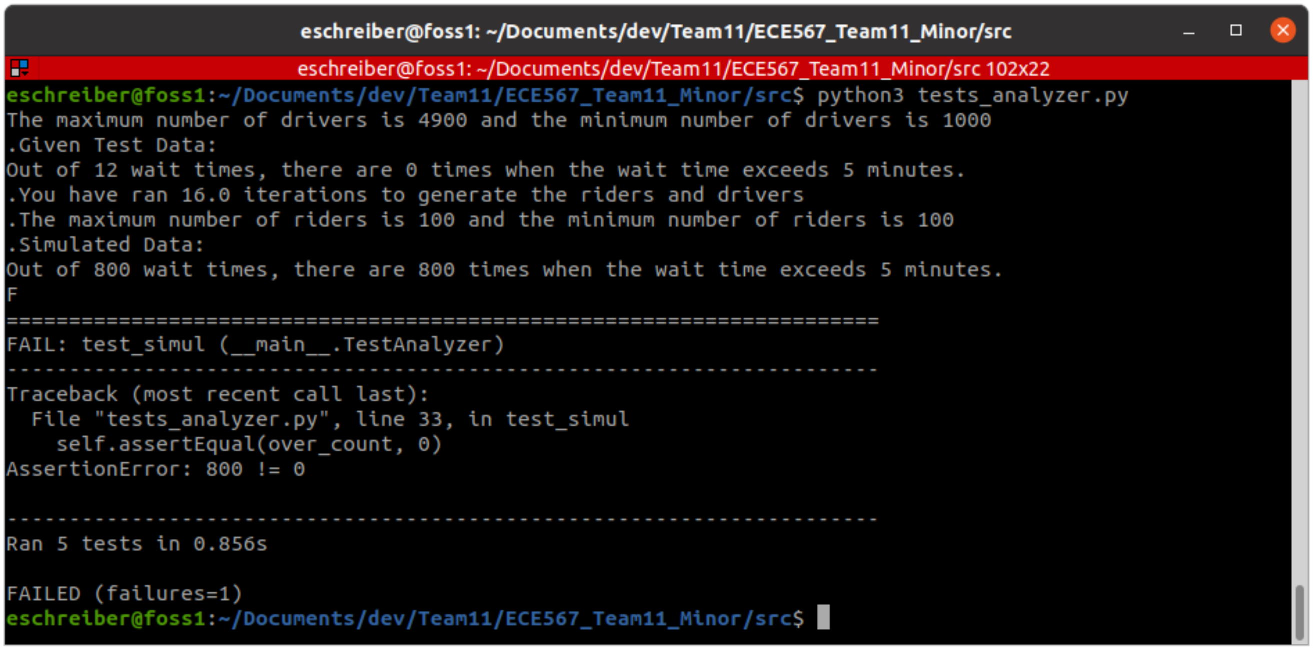
Figure 8: Simulation Unit Test Screen Capture

5 analyzer tests are created in sprint 2:

1. test_fromCSV: tests the the ResultsAnalyzer correctly loads data from a test data file (test_data.csv) and determines that the number of maximum drive times exceeding 5 minutes is 0.
2. test_simul: tests the the ResultsAnalyzer correctly loads data from a results data file (results.csv) and determines that the number of maximum drive times exceeding 5 minutes is 0. results.csv is generated by a simulation run of 20 iterations each at driver counts of 1000 to 4900 in steps of 100 (800 total iterations) and fixed size of 100 riders.
3. test_drivers: tests that the ResultsAnalyzer correctly determines minimum of 1000 drivers and maximum or 4900 drivers in results.csv
4. test_riders: tests that ResultsAnalyzer correctly determines 100 riders in each iteration in results.csv.
5. test_iterate: tests that ResultsAnalyzer correctly determines 800 iterations of tests in results.csv.

4 of 5 tests pass (screen capture below). test_simul fails as a 5 minute maximum wait time was not found in all (or any) of the simulation iterations from results.csv.

The failed test_simul unit test helps to flag a failed system test. It's failure confirms that our simulation to identify a range of drivers where maximum wait time does not exceed 5 minutes is unsuccessful. This is discussed further in the next section.



```
eschreiber@foss1: ~/Documents/dev/Team11/ECE567_Team11_Minor/src
eschreiber@foss1: ~/Documents/dev/Team11/ECE567_Team11_Minor/src 102x22
eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$ python3 tests_analyzer.py
The maximum number of drivers is 4900 and the minimum number of drivers is 1000
.Given Test Data:
Out of 12 wait times, there are 0 times when the wait time exceeds 5 minutes.
.You have ran 16.0 iterations to generate the riders and drivers
.The maximum number of riders is 100 and the minimum number of riders is 100
.Simulated Data:
Out of 800 wait times, there are 800 times when the wait time exceeds 5 minutes.
F
======================================================================
FAIL: test_simul (__main__.TestAnalyzer)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "tests_analyzer.py", line 33, in test_simul
    self.assertEqual(over_count, 0)
AssertionError: 800 != 0

----------------------------------------------------------------------
Ran 5 tests in 0.856s

FAILED (failures=1)
eschreiber@foss1:~/Documents/dev/Team11/ECE567_Team11_Minor/src$
```

Figure 9: Analyzer Unit Test Screen Capture

System Testing:
For Sprint 2, rider and driver locations were generated using the uniform distribution from Sprint 1 and the normal distribution placement added in Sprint 2 (REQ-7, REQ-10), with drivers placed in 4 quadrants (REQ-26 - REQ-29). For each distribution a simulation was run placing 100 riders and drivers in a range from 1000 to 4900 in steps of 100. Twenty iterations were run at each driver step.

Plots of maximum rider wait time are plotted for each iteration for both cases. The first graph below shows the results for the uniformly distributed placement and the second graph for the normally distributed placement. In both cases, maximum waiting time exceeds 5 minutes for every iteration. This is unexpected. Still more unexpected, the maximum wait time does not appear to decrease as the number of drivers increase, at least for the tested range of drivers.

The graphs do show differences in the variability of waiting times for the two distributions. The normal distribution times are confined to almost entirely between 8 and 12 minutes. For the uniform distribution, the spread is wider, mainly between 8 and 20 minutes. Whether this is due primarily to the difference in rider or driver placement, or a combination of both will be left for a future sprint investigation.

One improvement that the graphs confirm - the bounding box bug from Sprint 1 is fixed (REQ-14); the maximum wait times are now consistent with a bounding area that exceeds the distance that can be travelled in 5 minutes (4 x 4 kilometer instead of 400 x 400 meters).

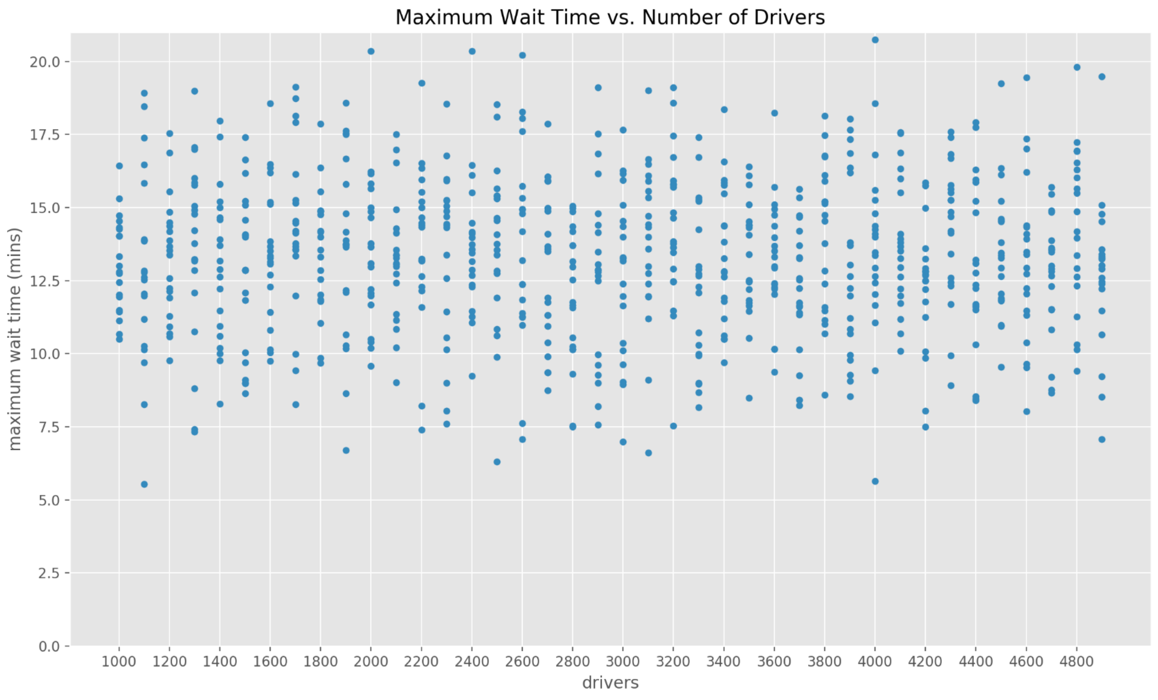


Figure 10: Maximum Rider Wait Time vs. Number of Drivers (20 iterations each), uniform distribution placement

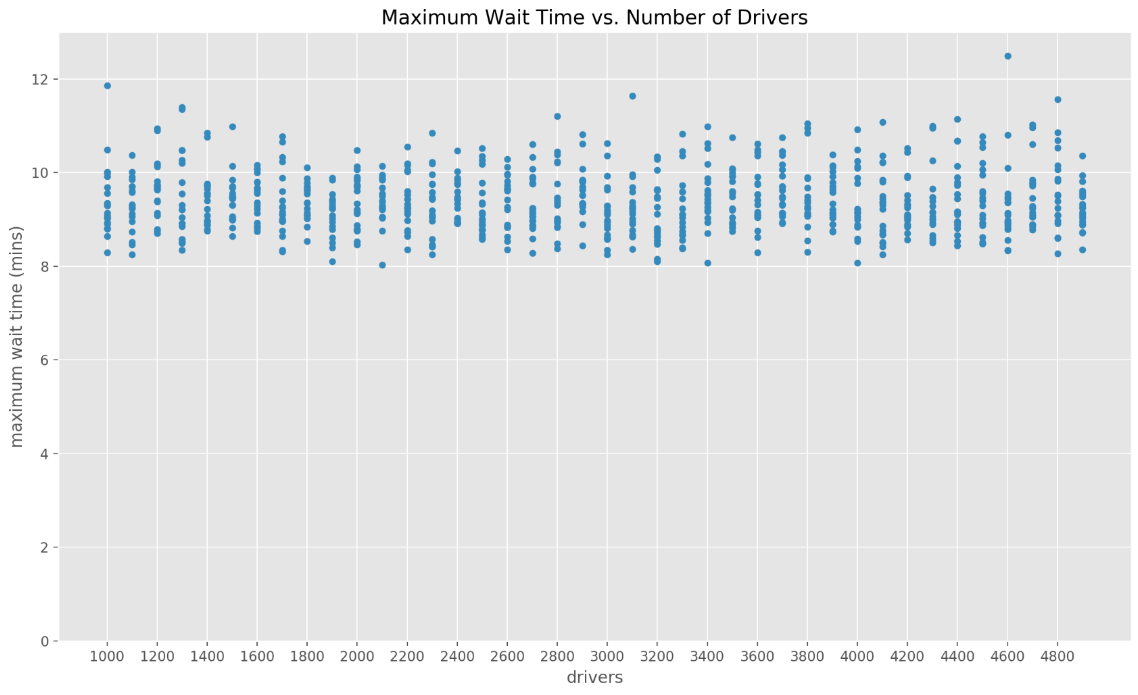


Figure 11: Maximum Rider Wait Time vs. Number of Drivers (20 iterations each), normal distribution placement

An additional locations plot is also added in Sprint 2 to confirm that driver and rider locations are, in fact, randomly distributed throughout the bounded 4 x 4 kilometer area of service as predicted. One graph was generated for a single iteration for each number of drivers and each placement distribution (*sample_data/uniform and sample_data/normal*). The figures below show the case for 1000 drivers and 100 riders for both distributions..

The locations are consistent with those expected, with the uniform placement randomized over the entire square and the normal placement with riders concentrated at center and the drivers evenly distributed between quadrants. More analysis will be required in a future sprint to determine the disconnect between expected and actual maximum wait times.

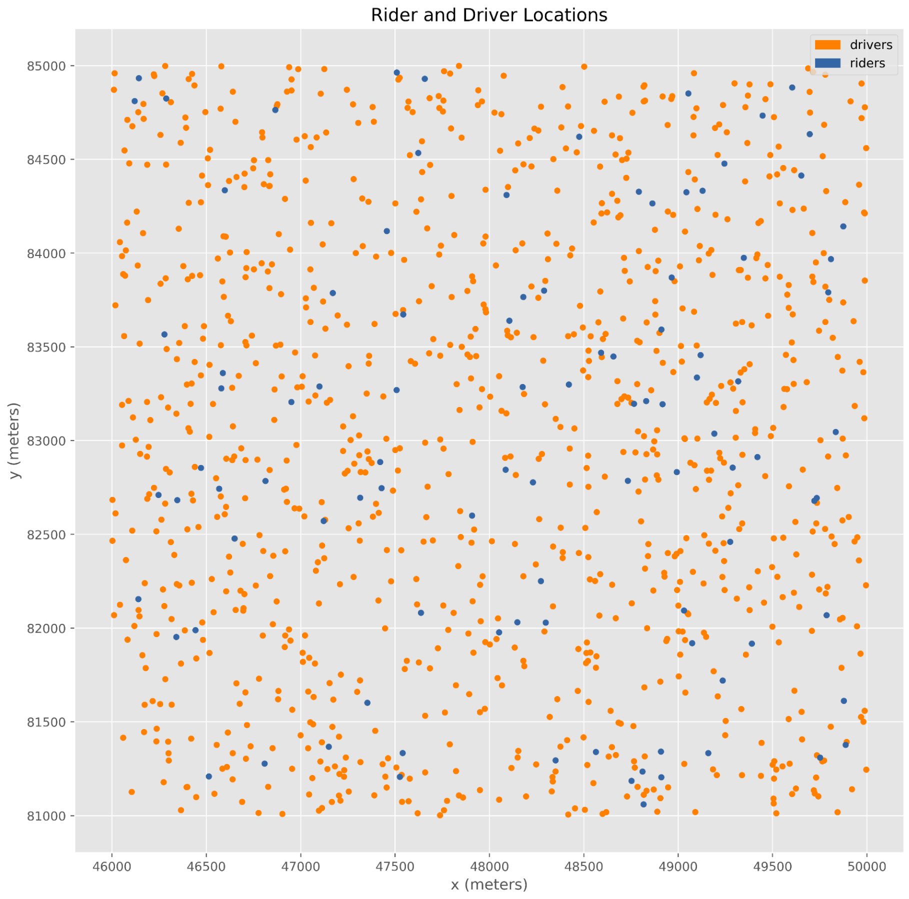


Figure 12: Rider and Driver location placement for 1000 drivers and 100 riders case, uniform distribution placeme

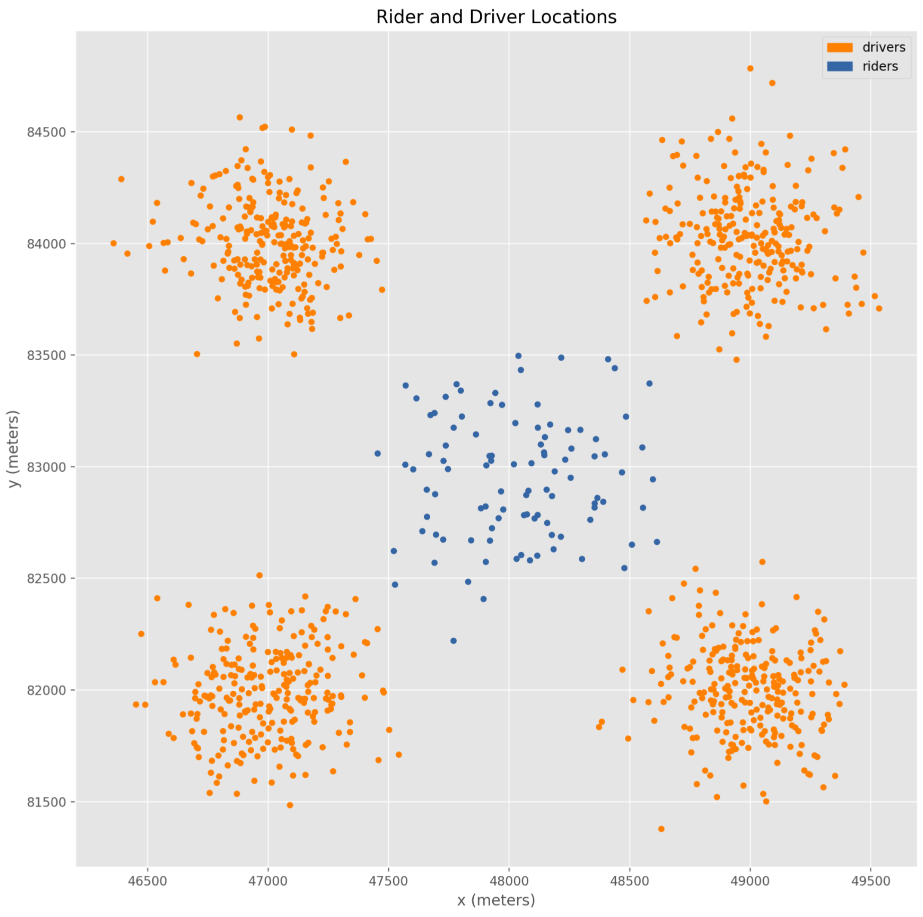


Figure 13: Rider and Driver location placement for 1000 drivers and 100 riders case, normal distribution placement, with 4 quadrant partition for drivers

SPRINT 2 CONCLUSION

Sprint 2 concluded on October, 25, 2021 and successfully implemented refinements to the driver and rider distribution functions and the graphical outputs/test functions. Sprint 2 is the final sprint for this development effort.

Testing conducted during Sprint 2 identified shortcomings and unexpected behaviour which was resolved by a re-evaluation of the existing code after the official end of the sprint. The logic for pairing Riders and Drivers, carried over from Sprint 1, contained a bug that resulted in unexpectedly longer wait times for larger number of drivers. The bugged code paired Drivers with Riders rather than the other way around – thus, only the first 100 Drivers would get paired with a Rider, even if the other remaining drivers were closer to them. This was fixed after the official end of Sprint 2 by reversing the logic to pair Riders with Drivers, and the expected behavior was identified as a correlation between number of drivers and lower max wait time.

If this project were to continue, the following recommendations would be made:

1. The relationship between rider wait time and number of drivers is not as strongly linear as expected. This could be due to one or more of the following:
    a. The chosen service area is too small for the number of drivers and riders distributed. The drivers and riders are already close together at initial deployment, and increasing the number of drivers has diminishing returns.
    b. The velocity is too high for the service area, resulting in small differences in wait times
    c. There is another error in the program which was not uncovered during testing.
2. The program uses a “brute force” approach to calculating distances and pairing drivers with riders. This is inefficient and there may be opportunities for improvement.
3. The program is “static” in that it only pairs drivers and riders available at one instant in time. Lyfter’s operations would be better modeled by a program that could consider time in the following ways:
    a. New riders populate over time
    b. Drivers become available after completing a trip
    c. Wait times factor in road distance and traffic conditions rather than linear distance.
4. Driver and rider distributions should be refined based on real data (e.g. Lyfter’s records)