Lyfter - Sprint 2

Group 20

A. Requirements Engineering

(Sprint 1 requirements still hold)

Additional Functional Requirements:

- Carpooling/driver can pick up multiple riders in a single car (within a 5 minute wait time)
- Require the least number of drivers

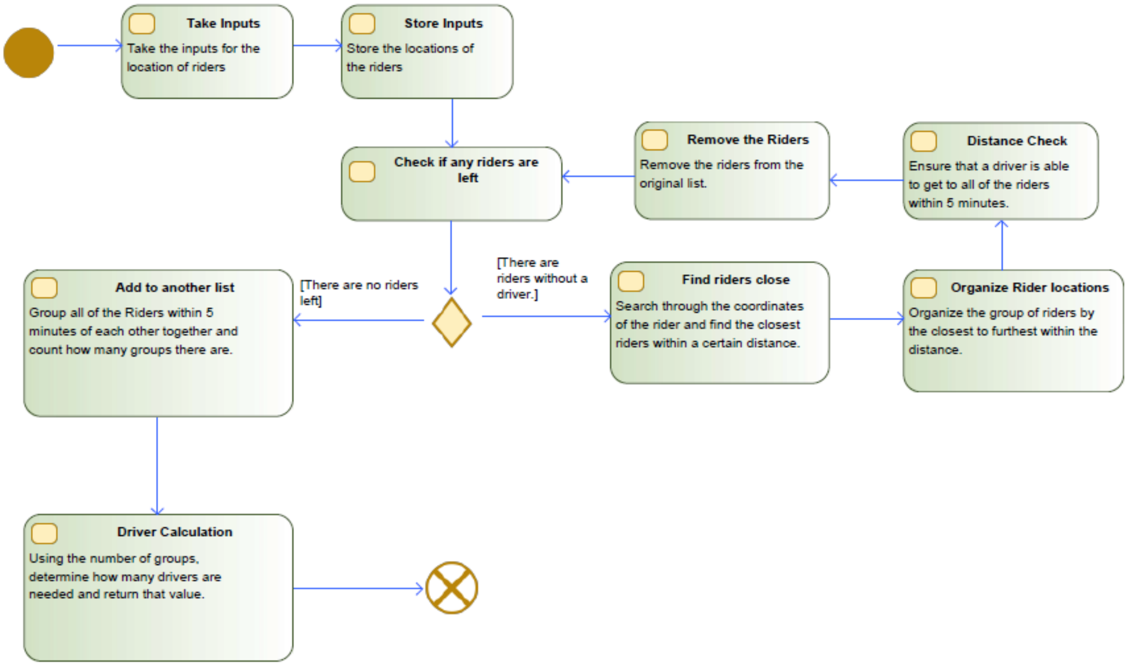
B. System Modeling

Grouping Riders Model:

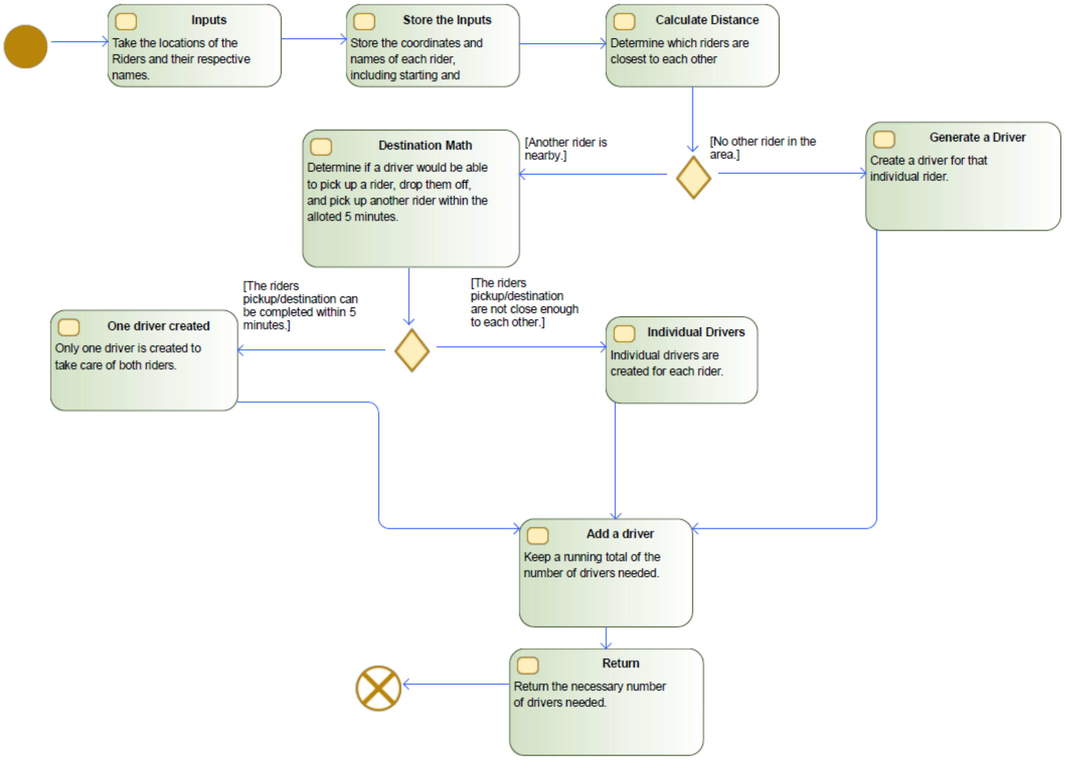
								🟪 R6	🟪 R7
							🟪 R10	🟪 R9	🟪 R8
	🟦 R5	🟦 R1							
				🟥 R3	🟥	🟥 R4			
				🟥	🟥 R2	🟥			

C. Architectural Design

Carpooling Activity Diagram

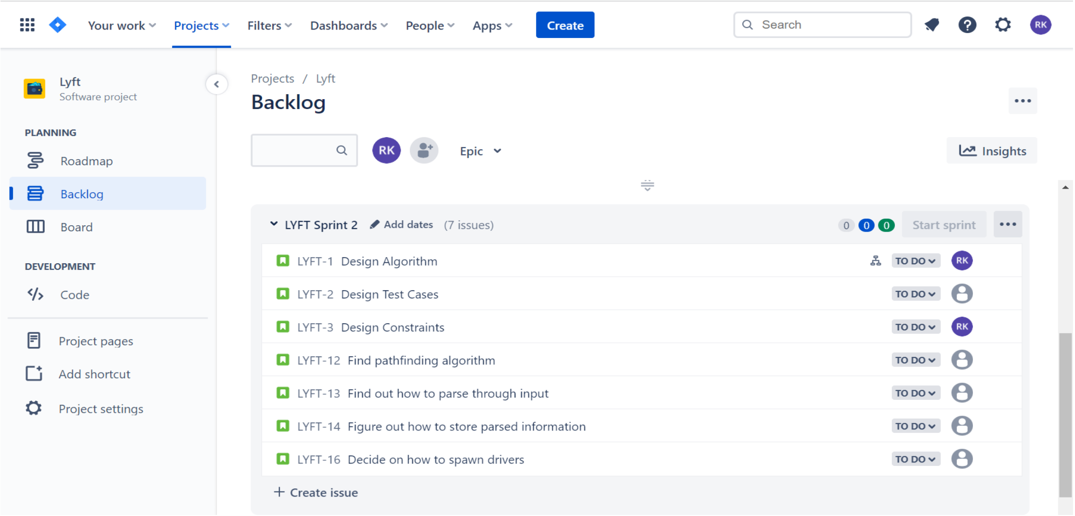

Take Inputs
Take the inputs for the location of riders
Store Inputs
Store the locations of the riders
Check if any riders are left
Remove the Riders
Remove the riders from the original list.
Distance Check
Ensure that a driver is able to get to all of the riders within 5 minutes.
Add to another list
Group all of the Riders within 5 minutes of each other together and count how many groups there are.
[There are no riders left]
[There are riders without a driver.]
Find riders close
Search through the coordinates of the rider and find the closest riders within a certain distance.
Organize Rider locations
Organize the group of riders by the closest to furthest within the distance.
Driver Calculation
Using the number of groups, determine how many drivers are needed and return that value.

One Driver to One Rider Activity Diagram


Inputs
Take the locations of the Riders and their respective names.
Store the Inputs
Store the coordinates and names of each rider, including starting and
Calculate Distance
Determine which riders are closest to each other
Destination Math
Determine if a driver would be able to pick up a rider, drop them off, and pick up another rider within the alloted 5 minutes.
[Another rider is nearby.]
[No other rider in the area.]
Generate a Driver
Create a driver for that individual rider.
[The riders pickup/destination can be completed within 5 minutes.]
[The riders pickup/destination are not close enough to each other.]
One driver created
Only one driver is created to take care of both riders.
Individual Drivers
Individual drivers are created for each rider.
Add a driver
Keep a running total of the number of drivers needed.
Return
Return the necessary number of drivers needed.

D. Design and Implementation

Jira Product Backlog:



Projects / Lyft

Backlog

LYFT Sprint 2 Add dates (7 issues) 0 0 0

LYFT-1 Design Algorithm	TO DO	RK
LYFT-2 Design Test Cases	TO DO	
LYFT-3 Design Constraints	TO DO	RK
LYFT-12 Find pathfinding algorithm	TO DO	
LYFT-13 Find out how to parse through input	TO DO	
LYFT-14 Figure out how to store parsed information	TO DO	
LYFT-16 Decide on how to spawn drivers	TO DO	

In order to find the best optimized solution we gave the option to select whether car pooling or private car method was desired. The program will then prompt which user text file containing all of the randomized coordinates of riders.

We implemented a grouping algorithm to carpool and we made different rider txt files to test different cases. There are drivers spawning next to a rider so that the driver can reach the group of riders in 5 minutes or less through carpooling. We also have a 1 to 1 algorithm for private riding that needs the starting and ending points for each rider and if the driver can pick up and drop off a rider and still get to another rider in 5 minutes or less, the number of drivers can be optimized.

Code in the github https://github.com/radical-teach/minor-project-group-20 (most recent version)

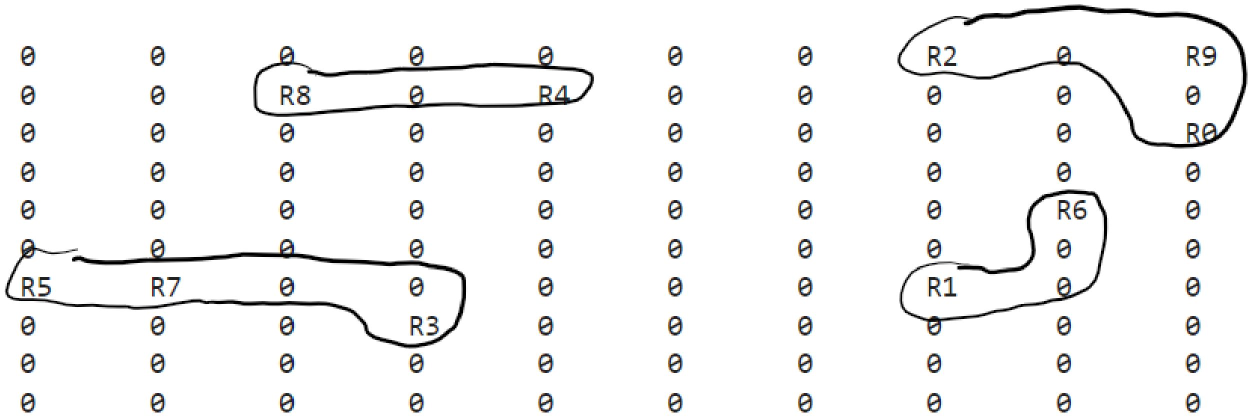
E. Software Testing

Carpooling

Txt file	# of users	Drawing	Expected Output	Actual Output
testTen1.txt	10	See Output3	4	4
testTen2.txt	10	See Output4	5	5

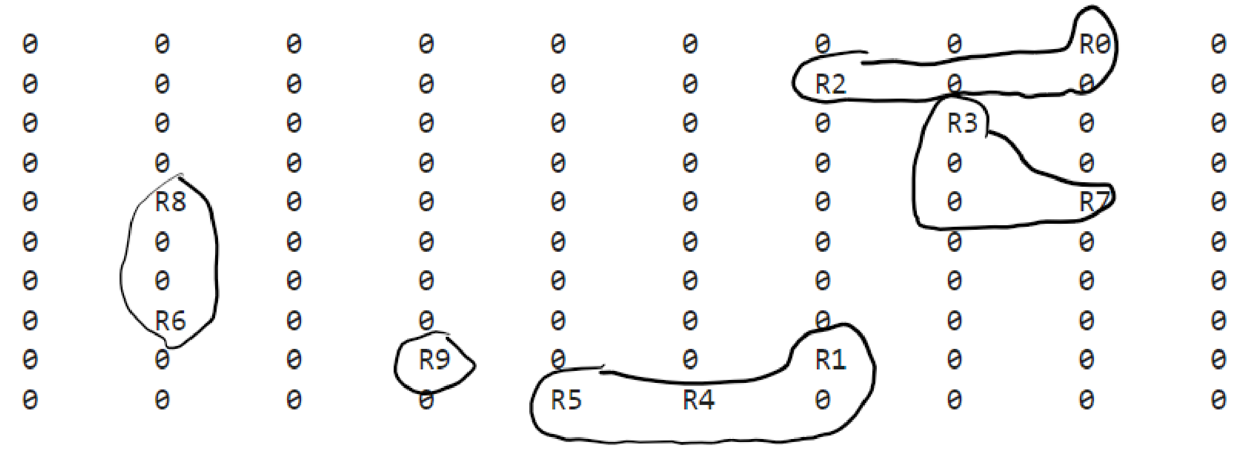
testFifty.txt	50		37	37
testHundred1.txt	100		83	83
testHundred2.txt	100		84	84

Output3:



```
0    0    0    0    0    0    0    R2   0    R9
0    0    R8   0    R4   0    0    0    0    0
0    0    0    0    0    0    0    0    0    R0
0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    R6   0
0    0    0    0    0    0    0    0    0    0
R5   R7   0    0    0    0    0    R1   0    0
0    0    0    R3   0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0
```

Output4:

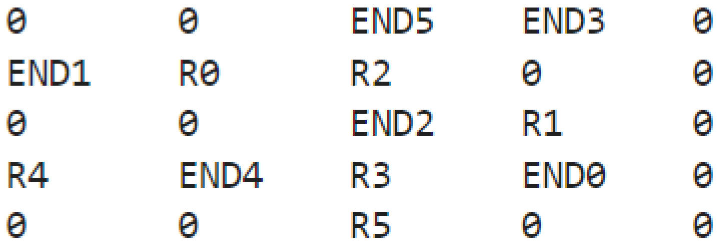


```
0    0    0    0    0    0    0    0    R0   0
0    0    0    0    0    0    R2   0    0    0
0    0    0    0    0    0    0    R3   0    0
0    0    0    0    0    0    0    0    0    0
0    R8   0    0    0    0    0    0    R7   0
0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0
0    R6   0    0    0    0    0    0    0    0
0    0    0    R9   0    0    R1   0    0    0
0    0    0    0    R5   R4   0    0    0    0
```

One Rider to One Driver

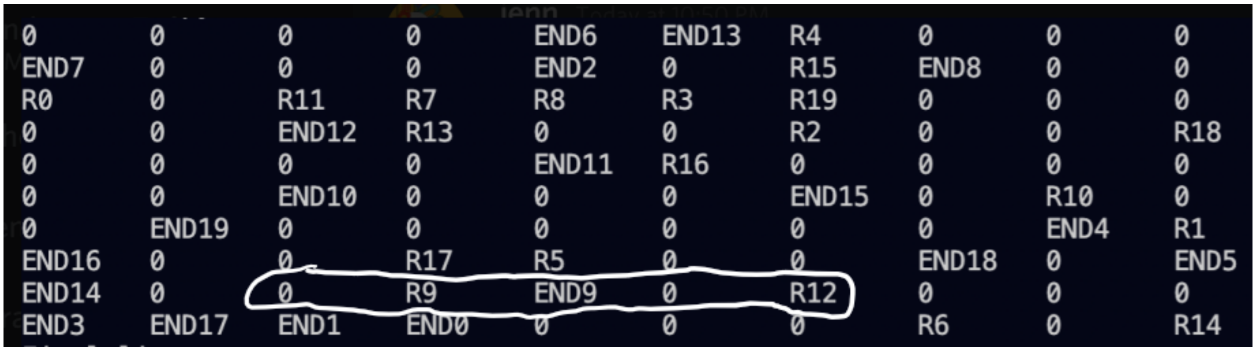
Txt file	# of users	Drawing	Expected Output	Actual Output
user.txt	5	See Output8	4	4
usersidk.txt	20	See Output9	19	19

Output8:



```
0         0         END5      END3      0
END1      R0        R2        0         0
0         0         END2      R1        0
R4        END4      R3        END0      0
0         0         R5        0         0
```

Output9:



```
0       0       0       0       END6    END13   R4      0       0       0
END7    0       0       0       END2    0       R15     END8    0       0
R0      0       R11     R7      R8      R3      R19     0       0       0
0       0       END12   R13     0       0       R2      0       0       R18
0       0       0       0       END11   R16     0       0       0       0
0       0       END10   0       0       0       END15   0       R10     0
0       END19   0       0       0       0       0       0       END4    R1
END16   0       0       R17     R5      0       0       END18   0       END5
END14   0       0       R9      END9    0       R12     0       0       0
END3    END17   END1    END0    0       0       0       R6      0       R14
```

F. Evaluation

Verification:

In order to verify the code, multiple tests were ran with as many variables as possible being changed. Some of the variables that were changed include the number of riders, locations of riders, and the size of the grid. Testing that would cause errors was also completed, ensuring that the expected error would occur.

Validation:

In order to fully validate the code, the requirements were consistently being compared to the results and the system. Ensuring that the requirements were all being fulfilled was one of the core parts of each Scrum, with a much more detailed version being done at the beginning and end of each Sprint.