Journal of Artificial Intelligence and Technology, 2022, 2, 111-122
https://doi.org/10.37965/jait.2022.0106

ISTP
RESEARCH ARTICLE

Traffic Dataset and Dynamic Routing Algorithm in Traffic Simulation

Zhemin Zhang,[1] Gennaro De Luca,[2] Brian Archambault,[1] Juan M. Chavez,[3] and Brandon Rice[4]
[1]School of Computing and Augmented Intelligence, Arizona State University, Tempe, USA
[2]Polytechnic School, Arizona State University, Mesa, USA
[3]Andersen Corporation, Arizona, USA
[4]Holcombe Department of Electrical and Computer Engineering, Clemson University, South Carolina, USA

(Received 07 April 2022; Revised 11 May 2022; Accepted 12 May 2022; Published online 12 May 2022)

Abstract: The purpose of this research is to create a simulated environment for teaching algorithms, big data processing, and machine learning. The environment is similar to Google Maps, with the capacity of finding the fastest path between two points in dynamic traffic situations. However, the system is significantly simplified for educational purposes. Students can choose different traffic patterns and program a car to navigate through the traffic dynamically based on the changing traffic. The environments used in the project are Visual IoT/Robotics Programming Language Environment (VIPLE) and a traffic simulator developed in the Unity game engine. This paper focuses on creating realistic traffic data for the traffic simulator and implementing dynamic routing algorithms in VIPLE. The traffic data are generated from the recorded real traffic data published on the Arizona Maricopa County website. Based on the generated traffic data, VIPLE programs are developed to implement the traffic simulation with support for dynamic changing data.

Keywords: computer science education; dynamic routing; traffic dataset; path planning; traffic simulation

I. INTRODUCTION

The purpose of the overall project is to create a simulated environment similar to Google Maps, where live traffic data are fed into the map system that can be dynamically updated to display the quickest path to a particular destination. Google's system is one of the best navigation solutions, but it is also too complex for introductory computer science students to create similar systems for learning purposes. A simplified traffic simulator environment was implemented, so that students could learn to program a virtual car with dynamically changing traffic. The main motivations of this paper are the presentation of a dataset creation process based on real traffic data and a corresponding dynamic routing algorithm for educational purposes.

The environment used in the project is ASU Visual IoT/Robotics Programming Language Environment (VIPLE) [1]. VIPLE offers all the functions necessary for novice programmers to learn programming concepts while experienced computer science students are able to program complex IoT devices, robots, and machine learning (ML) [1–3]. Various simulators have been implemented in VIPLE, including both desktop simulators and web simulators [2]. Traffic simulation is one of the functions that has recently been added into the VIPLE environment. VIPLE is particularly suited to this project as it provides strong support for orchestration and workflow in robotic applications [1,4].

This paper focuses on the traffic simulator, which is a desktop simulator, developed using the Unity game engine. The simulator provides an aerial view layout of a city grid and contains several different subareas of the city. Noncontrollable cars are colored yellow and are spawned by the simulator at any position on the road with randomized driving directions. These cars are used to represent traffic and can be given specific driving patterns according to supplied data. The red car can be programmed, for example, using VIPLE, and allows users to develop both manual and autonomous driving applications.

To provide a strong educational experience for computer science students, a number of student teams have engaged in developing the Unity traffic simulator. This project focuses on dynamic traffic routing on the traffic simulator using traffic data generated from real Maricopa County traffic data from 2020. This work aims to help students visualize their pathfinding algorithm using VIPLE and the traffic simulator. The data employed in this project are generated according to manually specified patterns from the original Maricopa County data. A self-sufficient approach can be implemented using an online ML application. Section VI provides more information on this topic.

Section II provides a review of the previous work that provides the foundation for this project. Section III details the outline of this project. Section IV outlines the method and implementation including a description of the generation of traffic data, operation of the traffic simulator, and a dynamic routing algorithm. Section V discusses and presents the results followed by a discussion of the contribution. Section VI concludes the paper with a discussion on potential future work.

II. RELATED WORK

This section presents related studies that have been done by Arizona State University faculty and students within the same overall project, as well as work done by researchers and practitioners from other organizations.

Corresponding author: Gennaro De Luca, (e-mail: Gennaro.Deluca@asu.edu)

A. TRAFFIC SIMULATION

According to the definition from Wikipedia, "traffic simulation is the mathematical modeling of transportation systems through the application of computer software and graphic presentation to better help understand, plan, design, and operate transportation systems" [5]. Today, many traffic simulators exist, some of them are proprietary systems and others are open-source systems. For example, Anylogic [6] and TransModeler [7] are commercial software. Simulation of Urban Mobility (SUMO) [8], CityFlow [9], and OpenTrafficSim [10] are open-source simulators.

The software used as the main reference is SUMO, which has been developed by the Institute of Transportation Research at German Aerospace Center since the year 2000 [11]. The goal of SUMO is for traffic management and traffic forecasting [12], whereas the VIPLE simulator is intended for educational purposes focusing on programming logic and visualizing graphic algorithms. SUMO's road network is similar to the VIPLE traffic simulator in that both use a graph data structure for the map. The roads are represented as edges and intersections are represented as vertices (nodes) [13]. SUMO uses a microscopic traffic simulation model which means that each vehicle on the street is simulated individually [14,15]. ASU VIPLE can be considered as using a macroscopic model, since it simulates the overall traffic situation, including traffic density and spawning rate, rather than the specific details of each car. Krauß discussed the traffic flow of SUMO in [16]. SUMO uses a microscopic traffic model in which traffic flow is also at the microscopic scale. The maximum safe velocity and acceleration for each car are important factors to be considered in traffic flow [16–18]. VIPLE's traffic simulator does not consider these factors from the microscopic scale, focusing instead solely on the macroscopic scale. As a result, simulation and analysis become simpler at the cost of lower realism. Behrisch et al. introduce research topics related to SUMO vehicle-to-infrastructure communication, route choice, and traffic light algorithms in [11]. These elements have a certain similarity with VIPLE's traffic simulator but with different approaches and goals.

B. TRAFFIC DATA GENERATION

Traffic data generation plays an essential role in this project and other research and applications. However, missing data is commonly encountered and typically causes problems in traffic data generation. Much research has been dedicated to solving data imputation problems. For example, bidirectional recurrent neural networks can be used to generate missing data in consecutive time periods [19]. Convolutional neural networks are models to compute the spatiotemporal missing traffic data [20,21]. Graph neural networks also can be used in spatiotemporal missing data. Zheng et al. proposed a graph multi-attention network to predict future traffic conditions [22]. In addition, a graph attention convolutional network was proposed by Ye et al. whose study suggested to "incorporate graph attention mechanism to learn spatial correlation of the traffic data," then "temporal convolutional layers are stacked to extract relations in time-series after graph attention layers" [23]. A recent study summarized the previous works on missing patterns and missing traffic data imputation and addresses two limitations on the traffic data imputation in the previous works. The solution from that research is to combine the bidirectional recurrent layer with a graph-based convolution layer to model the spatiotemporal dependencies [24]. To facilitate visualization and presentation of the techniques and data, we employed a different approach as discussed below. However, these techniques can be used to enhance our approach in managing incomplete data.

C. VIPLE

VIPLE is a graphical programming language environment developed in our previous work. A VIPLE program "is represented as a workflow diagram, and the components are defined as activities or Web services in the diagram" [25]. These programs can be mathematically modeled using the Pi-Calculus, enabling automated semantic verification of decentralized IoT and robotics applications written using VIPLE [3,25]. The ability to perform such verification on student programs provides additional benefit to this system as an educational platform. Figure 1 shows the basic activities in VIPLE.

In addition to the basic activities, VIPLE provides various services to facilitate different types of teaching and research projects. Figure 2 presents two more sets of services that are used in this paper. The first set of services includes general services, including input/output services (Simple Dialog, Print Line, Text to Speech, and Random), event services (Key Press Event, Key Release Event, Custom Event, and Timer), and RESTful services. The second set includes Robot/IoT services. Communication types between VIPLE and Robot/IoT devices include Wi-Fi (TCP or WebSockets), Bluetooth, and USB interfaces. JavaScript Object Notation (JSON) is the data format VIPLE and Robot/IoT devices use to send or receive messages, though custom interfaces can be defined. These features provide the foundation for the current paper to develop a dynamic path planning algorithm in the VIPLE environment. ASU VIPLE can be downloaded from this site: http://venus.sod.asu.edu/VIPLE.

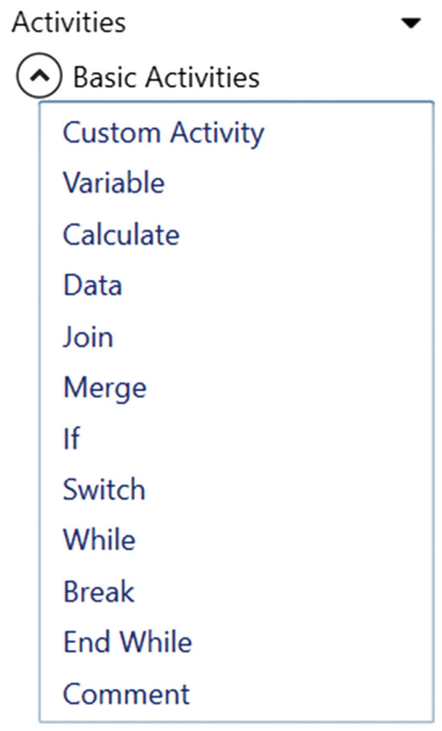
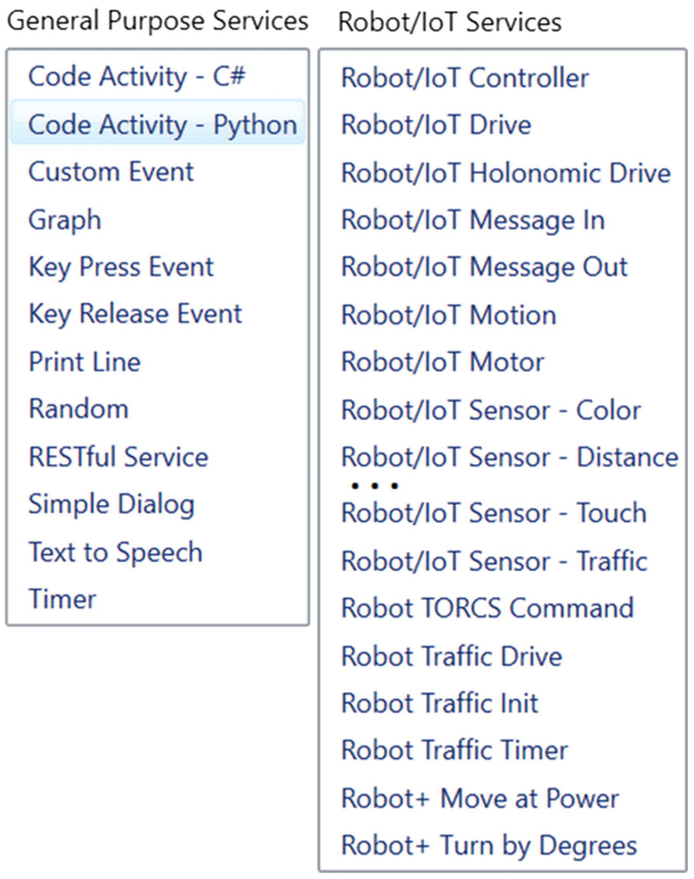


Fig. 1. VIPLE Basic Activities [25].



General Purpose Services	Robot/IoT Services
Code Activity - C#	Robot/IoT Controller
Code Activity - Python	Robot/IoT Drive
Custom Event	Robot/IoT Holonomic Drive
Graph	Robot/IoT Message In
Key Press Event	Robot/IoT Message Out
Key Release Event	Robot/IoT Motion
Print Line	Robot/IoT Motor
Random	Robot/IoT Sensor - Color
RESTful Service	Robot/IoT Sensor - Distance
Simple Dialog	• • •
Text to Speech	Robot/IoT Sensor - Touch
Timer	Robot/IoT Sensor - Traffic
	Robot TORCS Command
	Robot Traffic Drive
	Robot Traffic Init
	Robot Traffic Timer
	Robot+ Move at Power
	Robot+ Turn by Degrees

Fig. 2. More Services from VIPLE.

III. DESIGN

This section presents the overall design of the project, which includes the ideas and methods used.

For this project, real traffic data were employed in VIPLE's traffic simulator from the local area, Tempe, Arizona, located in Maricopa County [26]. Although this traffic dataset is incomplete, it provides records of partially complete hourly traffic data for individual intersections. These data records facilitate the application of the data to the VIPLE traffic simulator.

With the dataset selected, the next step is to generate traffic data. Because of the incompleteness of the traffic dataset from the Maricopa government website, the data generation was based on partially complete data, according to the hourly traffic trend in previous research. Specifics of this data generation are presented in Section IV.B. This data generation step was completed manually for each road and each time interval. Due to the large values of the generated traffic data, the dataset could not fit into the traffic simulator at the original scale. It was therefore scaled down before integration. Subsequently, the roads were mapped to the traffic simulator according to the appropriate geographical position in Maricopa County. This generated traffic data, a complete and scaled dataset using the values from real traffic data, was incorporated into the traffic simulator through the JSON data format and the Unity platform where the traffic simulator was developed. Lastly, the path planning algorithm, Dijkstra's algorithm, was implemented in the VIPLE environment using a min-heap data structure. In a classroom environment, various shortest path algorithms may be substituted. Dijkstra's algorithm was arbitrarily chosen for this work to provide a baseline demonstration of the work.

The experiment for this project is to run Dijkstra's algorithm with the city's simulated dynamically changing traffic patterns. The global path for the VIPLE programmable red car that defines the navigation path to the destination continually updates at each new intersection, demonstrating the quickest route according to the distance and traffic congestion on the roads at the given time.

IV. IMPLEMENTATION

This section illustrates the implementation process in detail, including generating traffic data, incorporating traffic data into the simulator, and implementing a dynamic version of Dijkstra's path planning algorithm in VIPLE.

A. TRAFFIC SIMULATOR OPERATION

This section introduces some functionalities and features of the traffic simulator that were used in this project.

1) DATA STRUCTURE OF THE TRAFFIC SIMULATOR MAP. As shown in Fig. 3, the traffic simulator is arranged in a grid. As a result, the current simulator only supports streets that match this layout, though further work could support more types of roads. The entire traffic simulator map is stored as a graph data structure. Each road is stored as an edge, and intersections are represented as vertices. The adjacency list is formed by the neighboring intersections of each intersection. The weight of each segment of road is represented as a combination of the Manhattan distance and number of cars on the road. The integration of this information into the simulator is detailed in Section IV.D.

2) COMMUNICATION WITH A VIPLE PROGRAM. To enable communication between VIPLE and the traffic simulator, the Robot/IoT Controller service needed to be configured properly. Figure 4 presents the configuration applied for this project. Robot/

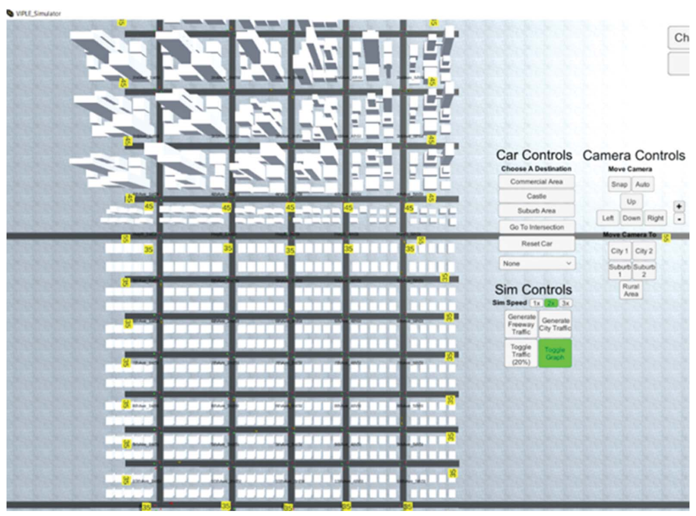


Fig. 3. Simulator Grid.

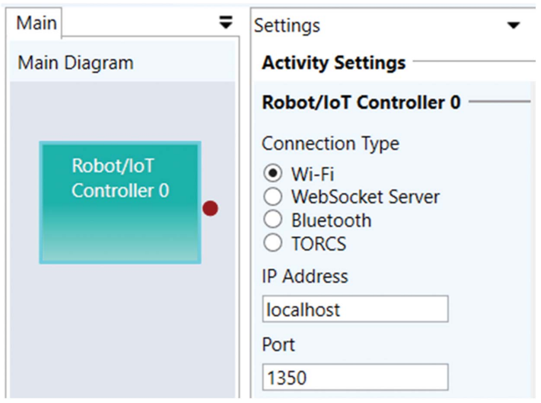

Main
Main Diagram
Settings
Activity Settings
Robot/IoT Controller 0
Connection Type
(●) Wi-Fi
( ) WebSocket Server
( ) Bluetooth
( ) TORCS
IP Address
localhost
Port
1350

Fig. 4. Robot/IoT Controller Configuration.

IoT Message In is a service that can receive custom JSON messages from the traffic simulator by listening on the socket defined in the Robot/IoT Controller configured earlier. Robot/IoT Message Out is used to transmit JSON messages through the socket to the traffic simulator. The traffic simulator receives the JSON messages and responds accordingly.

3) THE TRAFFIC SIMULATOR'S CONTROL INTERFACE. The main control buttons are shown in Fig. 5. The Car Controls section's buttons take the red car to the designated area of the map. The dropdown list includes different traffic patterns that control the number of yellow cars on the map, corresponding to the traffic density set for the simulator. Camera Controls help to navigate the traffic simulator to different areas. The WASD keys may also be used to pan the map north, west, south, and east, respectively, Z can be used to zoom in, and X can be used to zoom out. Toggle Traffic from the Sim Controls section toggles the yellow cars to spawn at 0%, 20%, or 50% spawn rates. Toggle Graph shows each intersection's name on the graph.

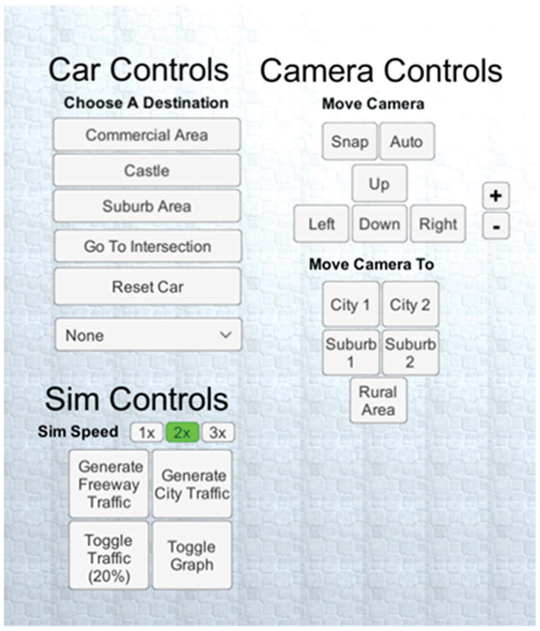


Fig. 5. Control Menu.

B. MARICOPA COUNTY TRAFFIC DATA GENERATION

The purpose of this section is to present the Maricopa County traffic data, describe how to overcome the incomplete data problem to generate traffic data, and the approach used to match the real traffic data to the VIPLE traffic simulator.

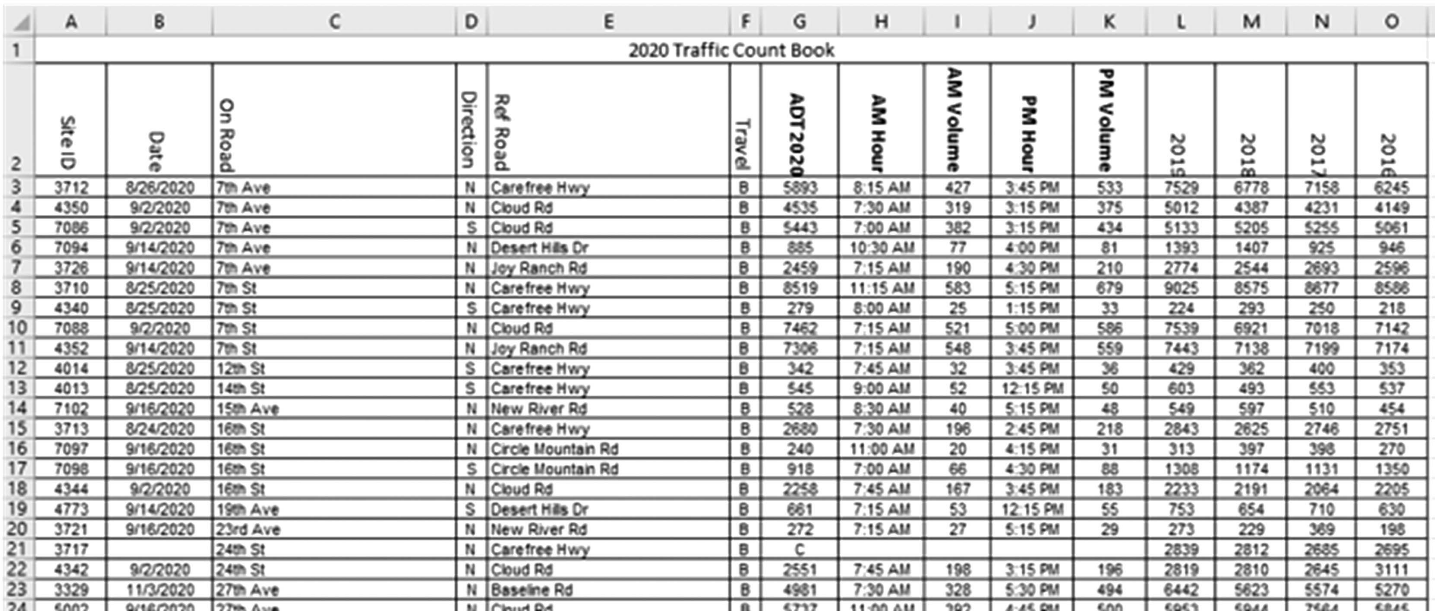
The data used in this project are from the Maricopa County government website. Fig. 6 shows a small sample of the data. This dataset contains 1378 records of various intersections' traffic information at different times of day. For example, the record in the 3rd row of the table shows that on August 26, 2020, at the intersection of Carefree Hwy and 7th Ave, 5893 cars drove north on 7th Ave through the intersection. At 8:15, 427 cars passed through this intersection heading north, while 533 cars did the same at 15:45.

Considering many major roads in Maricopa County follow a grid layout, the traffic simulator was created using an 11×5 grid as shown in Fig. 3. This structure was chosen to enable the application of local traffic patterns. According to this 11×5 grid in the simulator, eleven roads run in the east and west (E/W) direction, and five roads run in the north and south (N/S) direction. Maricopa County traffic data needed to be chosen and matched to the 11×5 simulator grid. However, a direct translation was impossible as the Maricopa County government website is not complete in the following ways. First, the dates to record Maricopa County traffic were not complete. For this traffic dataset, most of the records were for weekdays. Second, hourly traffic in a day was incomplete. For example, one intersection may only have contained traffic records at 8am and 4pm for that day. Third, one intersection may only have had traffic records for one direction. For example, the dataset only recorded the number of cars going north at the 7thAve and Carefree Hwy intersection but omitted traffic records for the south, east, and west directions.

With acknowledgement of the incompleteness of the dataset, the next step was to clean and process the traffic dataset from the Maricopa County website, mainly using the pandas Python library. First, rows containing empty cells were removed. By removing these rows, the traffic records were reduced from 1378 rows to 1278 rows. The data in the columns of Date, AM Hour, and PM Hour were originally represented as strings. In order to do computation on time in the future, the to_datetime function from pandas was used to convert the Date, AM Hour, and PM Hour into a date and time format recognized in Python.

Once the traffic data were cleaned, selecting proper traffic data records became an essential task. Eleven roads running east and west and five roads running north and south were chosen from Maricopa County traffic data to match to the traffic simulator. The first attempt included selecting major streets from the local area. However, this approach was not successful due to the incompleteness of the data because one intersection only had traffic records for one direction.

To resolve this issue, we selected roads and intersections from the dataset which had as much complete information as possible. Then, based on the records from the dataset, we generated traffic data according to hourly traffic trends in a given day [27]. According to the dataset, the west area of Maricopa County included more complete data records. Thus, roads were chosen mainly from the west area of Maricopa County. N/S roads included 355th Ave, Jackrabbit Tr, El Mirage, 107th Ave, 99th Ave and E/W roads included Thunderbird Blvd, Peoria Ave, Camelback Rd, Indian School Rd, Van Buren St, Buckeye Rd, Lower Buckeye Rd,

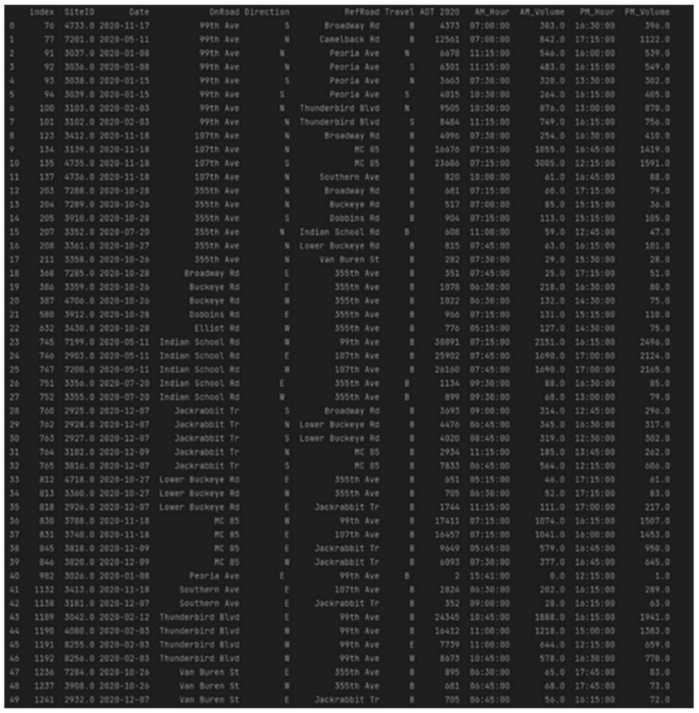


2020 Traffic Count Book

	Site ID	Date	On Road	Direction	Ref Road	Travel	ADT 2020	AM Hour	AM Volume	PM Hour	PM Volume	2019	2018	2017	2016
3	3712	8/26/2020	7th Ave	N	Carefree Hwy	B	5893	8:15 AM	427	3:45 PM	533	7529	6778	7158	6245
4	4350	9/2/2020	7th Ave	N	Cloud Rd	B	4535	7:30 AM	319	3:15 PM	375	5012	4387	4231	4149
5	7086	9/2/2020	7th Ave	S	Cloud Rd	B	5443	7:00 AM	382	3:15 PM	434	5133	5205	5255	5061
6	7094	9/14/2020	7th Ave	N	Desert Hills Dr	B	885	10:30 AM	77	4:00 PM	81	1393	1407	925	946
7	3726	9/14/2020	7th Ave	N	Joy Ranch Rd	B	2459	7:15 AM	190	4:30 PM	210	2774	2544	2693	2596
8	3710	8/25/2020	7th St	N	Carefree Hwy	B	8519	11:15 AM	583	5:15 PM	679	9025	8575	8677	8586
9	4340	8/25/2020	7th St	S	Carefree Hwy	B	279	8:00 AM	25	1:15 PM	33	224	293	250	218
10	7088	9/2/2020	7th St	N	Cloud Rd	B	7462	7:15 AM	521	5:00 PM	586	7539	6921	7018	7142
11	4352	9/14/2020	7th St	N	Joy Ranch Rd	B	7306	7:15 AM	548	3:45 PM	559	7443	7138	7199	7174
12	4014	8/25/2020	12th St	S	Carefree Hwy	B	342	7:45 AM	32	3:45 PM	36	429	362	400	353
13	4013	8/25/2020	14th St	S	Carefree Hwy	B	545	9:00 AM	52	12:15 PM	50	603	493	553	537
14	7102	9/16/2020	15th Ave	N	New River Rd	B	528	8:30 AM	40	5:15 PM	48	549	597	510	454
15	3713	8/24/2020	16th St	N	Carefree Hwy	B	2680	7:30 AM	196	2:45 PM	218	2843	2625	2746	2751
16	7097	9/16/2020	16th St	N	Circle Mountain Rd	B	240	11:00 AM	20	4:15 PM	31	313	397	398	270
17	7098	9/16/2020	16th St	S	Circle Mountain Rd	B	918	7:00 AM	66	4:30 PM	88	1308	1174	1131	1350
18	4344	9/2/2020	16th St	N	Cloud Rd	B	2258	7:45 AM	167	3:45 PM	183	2233	2191	2064	2205
19	4773	9/14/2020	19th Ave	S	Desert Hills Dr	B	661	7:15 AM	53	12:15 PM	55	753	654	710	630
20	3721	9/16/2020	23rd Ave	N	New River Rd	B	272	7:15 AM	27	5:15 PM	29	273	229	369	198
21	3717		24th St	N	Carefree Hwy	B	C					2839	2812	2685	2695
22	4342	9/2/2020	24th St	N	Cloud Rd	B	2551	7:45 AM	198	3:15 PM	196	2819	2810	2645	3111
23	3329	11/3/2020	27th Ave	N	Baseline Rd	B	4981	7:30 AM	328	5:30 PM	494	6442	5623	5574	5270
24	5002	9/16/2020	27th Ave	N	Cloud Rd	B	5737	11:00 AM	392	4:45 PM	600	5953	5944	7564	5845

Fig. 6. Maricopa County Traffic Data.

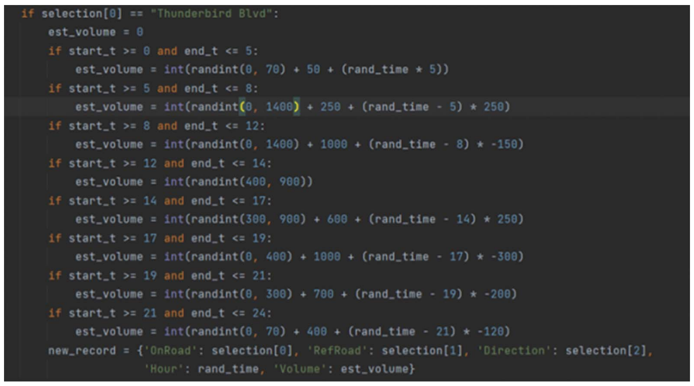
MC 85, Broadway Rd, Southern Ave, Baseline Rd, Dobbins Rd, Elliot Rd. The isin function was used from pandas to select the records that have roads running in the N/S direction. These were designated as On Road. E/W roads were set as Ref Road. Other records had roads running in the E/W direction delineated as On Road and roads running in the N/S direction that were Ref Road. The records after processing are shown in Fig. 7.



```
    index  SiteID       Date          OnRoad Direction          RefRoad Travel  ADT 2020   AM_Hour  AM_Volume   PM_Hour  PM_Volume
0      76  4733.0 2020-11-17        99th Ave         S      Broadway Rd      B      4373  07:00:00      303.0  16:30:00      396.0
1      77  7201.0 2020-05-11        99th Ave         N     Camelback Rd      B     12561  07:00:00      842.0  17:15:00     1122.0
2      91  3037.0 2020-01-08        99th Ave        N        Peoria Ave     N      6678  11:15:00      546.0  16:00:00      539.0
3      92  3036.0 2020-01-08        99th Ave         N       Peoria Ave      S      6301  11:15:00      483.0  16:15:00      549.0
4      93  3038.0 2020-01-15        99th Ave         S       Peoria Ave      N      3663  07:30:00      320.0  13:30:00      302.0
5      94  3039.0 2020-01-15        99th Ave        S        Peoria Ave     S      4015  10:30:00      264.0  16:15:00      405.0
6     100  3103.0 2020-02-03        99th Ave        N  Thunderbird Blvd     N      9505  10:30:00      876.0  13:00:00      870.0
7     101  3102.0 2020-02-03        99th Ave         N Thunderbird Blvd      S      8484  11:15:00      749.0  16:15:00      756.0
8     123  3412.0 2020-11-18       107th Ave         N      Broadway Rd      B      4096  07:30:00      254.0  16:30:00      410.0
9     134  3139.0 2020-11-18       107th Ave         N            MC 85      B     16676  07:15:00     1055.0  16:45:00     1419.0
10    135  4735.0 2020-11-18       107th Ave         S            MC 85      B     23686  07:15:00     3005.0  12:15:00     1591.0
11    137  4736.0 2020-11-18       107th Ave         N     Southern Ave      B       820  10:00:00       61.0  16:45:00       88.0
12    203  7288.0 2020-10-28       355th Ave         N      Broadway Rd      B       681  07:15:00       60.0  17:15:00       79.0
13    204  7289.0 2020-10-26       355th Ave         N       Buckeye Rd      B       517  07:00:00       85.0  15:15:00       36.0
14    205  3910.0 2020-10-28       355th Ave         S       Dobbins Rd      B       904  07:15:00      113.0  15:15:00      105.0
15    207  3352.0 2020-07-20       355th Ave        N  Indian School Rd     B       608  11:00:00       59.0  12:45:00       47.0
16    208  3361.0 2020-10-27       355th Ave         N Lower Buckeye Rd      B       815  07:45:00       63.0  16:15:00      101.0
17    211  3358.0 2020-10-26       355th Ave         N     Van Buren St      B       282  07:30:00       29.0  15:30:00       28.0
18    368  7285.0 2020-10-28     Broadway Rd         E        355th Ave      B       351  07:45:00       25.0  17:15:00       51.0
19    386  3359.0 2020-10-26      Buckeye Rd         E        355th Ave      B      1078  06:30:00      218.0  16:30:00       80.0
20    387  4706.0 2020-10-26      Buckeye Rd         W        355th Ave      B      1022  06:30:00      132.0  14:30:00       75.0
21    580  3912.0 2020-10-28      Dobbins Rd         E        355th Ave      B       966  07:15:00      131.0  15:15:00      110.0
22    632  3430.0 2020-10-28       Elliot Rd         W        355th Ave      B       776  05:15:00      127.0  14:30:00       75.0
23    745  7199.0 2020-05-11 Indian School Rd        W         99th Ave      B     30891  07:15:00     2151.0  16:15:00     2496.0
24    746  2903.0 2020-05-11 Indian School Rd        E        107th Ave      B     25902  07:45:00     1690.0  17:00:00     2124.0
25    747  7200.0 2020-05-11 Indian School Rd        W        107th Ave      B     26160  07:45:00     1690.0  17:00:00     2165.0
26    751  3356.0 2020-07-20 Indian School Rd       E         355th Ave     B      1134  09:30:00       88.0  16:30:00       85.0
27    752  3355.0 2020-07-20 Indian School Rd       W         355th Ave     B       899  09:30:00       68.0  13:00:00       79.0
28    760  2925.0 2020-12-07   Jackrabbit Tr         S      Broadway Rd      B      3693  09:00:00      314.0  12:45:00      296.0
29    762  2928.0 2020-12-07   Jackrabbit Tr         N Lower Buckeye Rd      B      4476  06:45:00      345.0  16:30:00      317.0
30    763  2927.0 2020-12-07   Jackrabbit Tr         S Lower Buckeye Rd      B      4020  08:45:00      319.0  12:30:00      302.0
31    764  3182.0 2020-12-09   Jackrabbit Tr         N            MC 85      B      2934  11:15:00      185.0  13:45:00      262.0
32    765  3816.0 2020-12-07   Jackrabbit Tr         S            MC 85      B      7833  06:45:00      564.0  12:15:00      606.0
33    812  4718.0 2020-10-27 Lower Buckeye Rd        E        355th Ave      B       651  05:15:00       46.0  17:15:00       61.0
34    813  3360.0 2020-10-27 Lower Buckeye Rd        W        355th Ave      B       705  06:30:00       52.0  17:15:00       83.0
35    818  2926.0 2020-12-07 Lower Buckeye Rd        E    Jackrabbit Tr      B      1744  11:15:00      111.0  17:00:00      217.0
36    830  3788.0 2020-11-18           MC 85         W         99th Ave      B     17411  07:15:00     1074.0  16:15:00     1507.0
37    831  3740.0 2020-11-18           MC 85         E        107th Ave      B     16457  07:15:00     1041.0  16:00:00     1453.0
38    845  3818.0 2020-12-09           MC 85         E    Jackrabbit Tr      B      9649  05:45:00      579.0  16:45:00      950.0
39    846  3820.0 2020-12-09           MC 85         W    Jackrabbit Tr      B      6093  07:30:00      377.0  16:45:00      645.0
40    982  3026.0 2020-01-08      Peoria Ave        E          99th Ave     B         2  15:41:00        0.0  12:15:00        1.0
41   1132  3413.0 2020-11-18    Southern Ave         E        107th Ave      B      2824  06:30:00      202.0  16:15:00      289.0
42   1138  3181.0 2020-12-07    Southern Ave         E    Jackrabbit Tr      B       352  09:00:00       28.0  16:15:00       63.0
43   1189  3042.0 2020-02-12 Thunderbird Blvd        E         99th Ave      B     24345  10:45:00     1888.0  16:15:00     1941.0
44   1190  4080.0 2020-02-03 Thunderbird Blvd        W         99th Ave      B     16412  11:00:00     1218.0  15:00:00     1383.0
45   1191  8255.0 2020-02-03 Thunderbird Blvd        W         99th Ave      E      7739  11:00:00      644.0  12:15:00      659.0
46   1192  8256.0 2020-02-03 Thunderbird Blvd        W         99th Ave      W      8673  10:45:00      578.0  16:30:00      770.0
47   1236  7284.0 2020-10-26    Van Buren St         E        355th Ave      B       895  06:30:00       65.0  17:45:00       83.0
48   1237  3908.0 2020-10-26    Van Buren St         W        355th Ave      B       681  06:45:00       68.0  17:45:00       73.0
49   1241  2932.0 2020-12-07    Van Buren St         E    Jackrabbit Tr      B       705  06:45:00       56.0  16:15:00       72.0
```

Fig. 7. Selected Traffic Data from Arizona Maricopa County.

With the selected records from the dataset and hourly traffic graphs obtained using the same method as described by [27], traffic data generation proceeded. The next step of data generation was completed manually for each road for different time durations. The daily traffic trends were split into eight distinct time groups. The eight different groups are: 0:00–5:00, 5:00–8:00, 8:00–12:00, 12:00–14:00, 14:00–17:00, 17:00–19:00, 19:00–21:00, and 21:00–23:59. To manually generate a complete traffic dataset, we analyzed the existing Maricopa County traffic data for each road. Then, based on the existing traffic data, we applied traffic patterns as linear functions with a random component to obtain the whole day traffic data for a specific road. Figure 8 presents sample code illustrating how traffic data were generated for Thunderbird Blvd for eight different time periods in a day. First, as shown in Fig. 7, #43 to #47 are the records that have Thunderbird as On Road. Available times in records for Thunderbird are 10:45, 11:00, 12:15, 15:00, 16:15, and 16:30, which fall in the 8:00–12:00, 12:00–14:00, and 14:00–17:00 durations. It should be noted that



```
if selection[0] == "Thunderbird Blvd":
    est_volume = 0
    if start_t >= 0 and end_t <= 5:
        est_volume = int(randint(0, 70) + 50 + (rand_time * 5))
    if start_t >= 5 and end_t <= 8:
        est_volume = int(randint(0, 1400) + 250 + (rand_time - 5) * 250)
    if start_t >= 8 and end_t <= 12:
        est_volume = int(randint(0, 1400) + 1000 + (rand_time - 8) * -150)
    if start_t >= 12 and end_t <= 14:
        est_volume = int(randint(400, 900))
    if start_t >= 14 and end_t <= 17:
        est_volume = int(randint(300, 900) + 600 + (rand_time - 14) * 250)
    if start_t >= 17 and end_t <= 19:
        est_volume = int(randint(0, 400) + 1000 + (rand_time - 17) * -300)
    if start_t >= 19 and end_t <= 21:
        est_volume = int(randint(0, 300) + 700 + (rand_time - 19) * -200)
    if start_t >= 21 and end_t <= 24:
        est_volume = int(randint(0, 70) + 400 + (rand_time - 21) * -120)
    new_record = {'OnRoad': selection[0], 'RefRoad': selection[1], 'Direction': selection[2],
                  'Hour': rand_time, 'Volume': est_volume}
```

Fig. 8. Sample Code for Traffic Generation.

for manual traffic generation, the dates do not play an important role in this process because most of the dates are on weekdays and the goal for the traffic data generation is to generate hourly traffic for any of the weekdays. The highest number for 10:45, during the 8:00–12:00 duration, is 1888, and the lowest number is 578. The highest number for 11:00 is 1218, and the lowest number for 11:00 is 644. According to the traffic trend from Fig. 8, the traffic slightly decreased from 8:00 to 12:00 during the weekday. Thus, the linear function for 8:00–12:00 is shown in Equation 1 below, where rand_time is the time randomly generated between 8:00 and 12:00. Though this formula does not perfectly model the values in the real data, it provides similar ranges based on the ranges from the real dataset in each time range.

$$traffic = 1000 - 150 \cdot (rand_{time} - 8) \tag{1}$$

Then, we added randint (0, 1400) to the linear function because for the time duration between 8:00 and 12:00, the maximum difference for the same hour at 10:45 was 1888–578, which is 1310. A random number is used in the total traffic to take different traffic situations for different days into consideration. For the time duration omitted in the Maricopa County traffic data, the hourly traffic trend graph from Fig. 8 is applied to the linear function. For instance, the traffic for Thunderbird Blvd between 5:00 and 8:00 is missing. According to the existing traffic data from 8:00 to 12:00, the traffic at 8:00 is between 1000 and 2400. Also, the increase rate for 5:00–8:00 is faster. Thus, the linear function used is shown in Equation 2, with a random integer between (0, 1400). This formula demonstrates the same ideology as Equation 1 for a different time range.

$$traffic = 250 + (rand\_time - 5) * 250 \tag{2}$$

At 8:00, the traffic increased between 1000 and 2400 with 250 cars per hour. This process is repeated for other time durations in a day for Thunderbird Blvd. Then, other roads used the same idea to generate traffic data at different hours.

C. MAP MARICOPA COUNTY TRAFFIC DATA TO TRAFFIC SIMULATOR

Determining how the Maricopa County traffic data map to the traffic simulator was the next task. The mapping process was performed in two major steps. The first step was to map the roads from Maricopa County to the simulator geographically. The second step was to correspond the hourly traffic data for each road in the simulator.

The first step of the mapping process was straightforward because the chosen roads in Maricopa County were mapped to the same geographical position in the traffic simulator as much as possible. Figure 9 shows the mapping from the Maricopa County roads to the simulator roads. The yellow columns show the roads that run in the N/S direction, and the green rows show the roads that run in the E/W direction.

The second step of mapping was necessary because the real traffic data from Maricopa County were too large for the traffic simulator. To have traffic data that the simulator could process, we had to scale down the Maricopa County traffic volume. The traffic volume for each road in Maricopa County varies from 20 to 3000. To scale down, we divided all the original traffic data by 100. Then, we grouped the resulting values from 0 to 30 into ten intervals and assigned the appropriate volume value of each interval to each traffic simulator road as presented in Fig. 10.

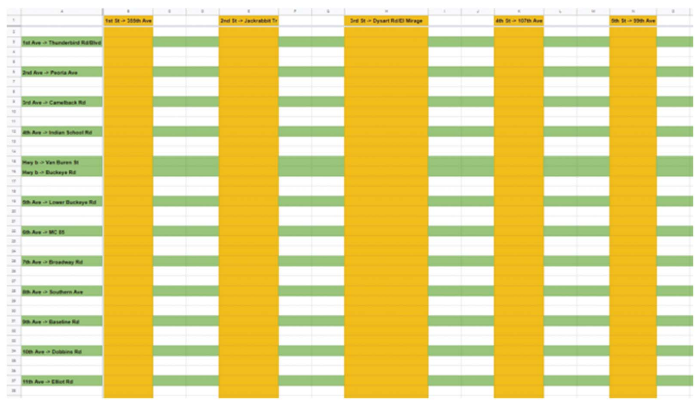

1st St -> 355th Ave
2nd St -> Jackrabbit Tr
3rd St -> Dysart Rd/El Mirage
4th St -> 107th Ave
5th St -> 59th Ave
1st Ave -> Thunderbird Rd/Blvd
2nd Ave -> Peoria Ave
3rd Ave -> Camelback Rd
4th Ave -> Indian School Rd
Hwy b -> Van Buren St
Hwy b -> Buckeye Rd
5th Ave -> Lower Buckeye Rd
6th Ave -> MC 85
7th Ave -> Broadway Rd
8th Ave -> Southern Ave
9th Ave -> Baseline Rd
10th Ave -> Dobbins Rd
11th Ave -> Elliot Rd

Fig. 9. Mapping between Maricopa Roads and Simulator Roads.

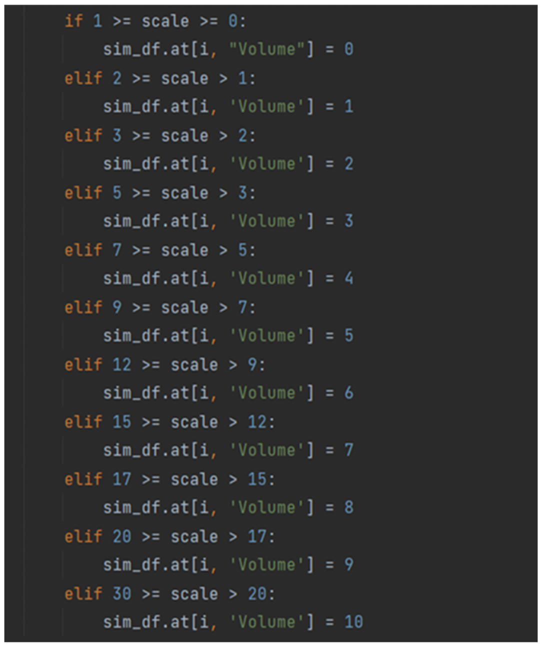

```
if 1 >= scale >= 0:
    sim_df.at[i, "Volume"] = 0
elif 2 >= scale > 1:
    sim_df.at[i, 'Volume'] = 1
elif 3 >= scale > 2:
    sim_df.at[i, 'Volume'] = 2
elif 5 >= scale > 3:
    sim_df.at[i, 'Volume'] = 3
elif 7 >= scale > 5:
    sim_df.at[i, 'Volume'] = 4
elif 9 >= scale > 7:
    sim_df.at[i, 'Volume'] = 5
elif 12 >= scale > 9:
    sim_df.at[i, 'Volume'] = 6
elif 15 >= scale > 12:
    sim_df.at[i, 'Volume'] = 7
elif 17 >= scale > 15:
    sim_df.at[i, 'Volume'] = 8
elif 20 >= scale > 17:
    sim_df.at[i, 'Volume'] = 9
elif 30 >= scale > 20:
    sim_df.at[i, 'Volume'] = 10
```

Fig. 10. Scale Down Traffic Data in Ten Intervals.

Figure 11 presents how the traffic data changes hourly as a whole for the scaled-down data.

Finally, since the traffic data generated in Python were in a pandas DataFrame object, which cannot be directly sent to the simulator, the traffic data needed to be changed to JSON format. A sample JSON file that includes 7:00–8:00 traffic data is shown in Fig. 12.

D. INCORPORATE GENERATED TRAFFIC DATA INTO TRAFFIC SIMULATOR

The traffic simulator map was stored as a graph data structure. Roads were represented as edges and intersections were represented as vertices of the graph. Neighboring intersections for each

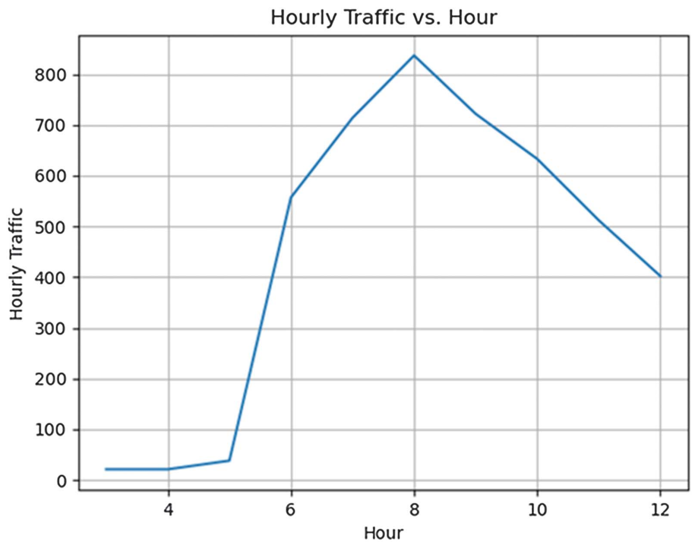
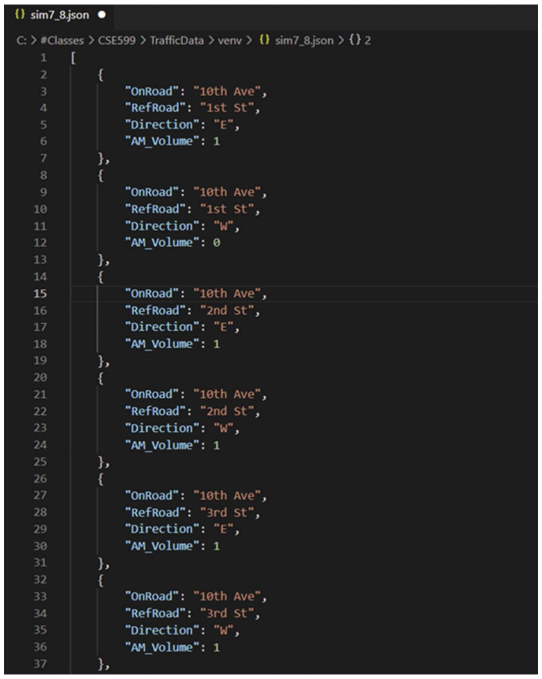


Fig. 11. Hourly Traffic for Simulator vs. Hour.

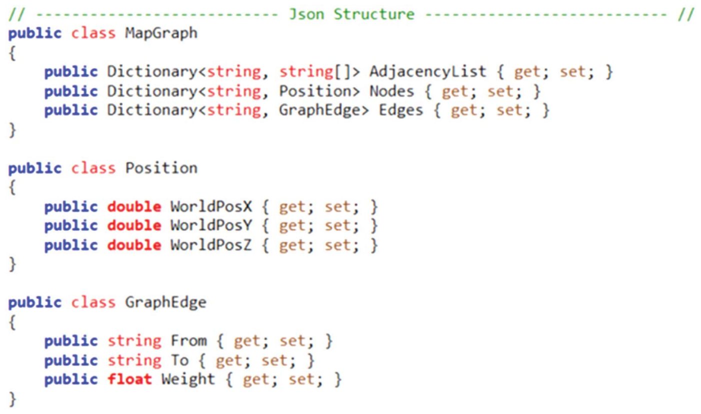


```
{} sim7_8.json ●
C: > #Classes > CSE599 > TrafficData > venv > {} sim7_8.json > {} 2
[
    {
        "OnRoad": "10th Ave",
        "RefRoad": "1st St",
        "Direction": "E",
        "AM_Volume": 1
    },
    {
        "OnRoad": "10th Ave",
        "RefRoad": "1st St",
        "Direction": "W",
        "AM_Volume": 0
    },
    {
        "OnRoad": "10th Ave",
        "RefRoad": "2nd St",
        "Direction": "E",
        "AM_Volume": 1
    },
    {
        "OnRoad": "10th Ave",
        "RefRoad": "2nd St",
        "Direction": "W",
        "AM_Volume": 1
    },
    {
        "OnRoad": "10th Ave",
        "RefRoad": "3rd St",
        "Direction": "E",
        "AM_Volume": 1
    },
    {
        "OnRoad": "10th Ave",
        "RefRoad": "3rd St",
        "Direction": "W",
        "AM_Volume": 1
    },
```

Fig. 12. Sample JSON File Showing Traffic Data Between 7:00 and 8:00.

vertex were included in the adjacency list. Code in Fig. 13 demonstrates the JSON structure sent from the simulator.

The senior capstone team at Arizona State University in charge of developing the traffic simulator helped create a preset dropdown list to select hourly traffic patterns in the simulator, as shown in Fig. 14. "Traffic from Viple" is the option that uses the Robot/IoT Message Out service in VIPLE to send an hourly JSON file to the simulator. For now, 4:00–5:00, 7:00–8:00, and 10:00–11:00 traffic patterns are chosen to be "Preset 1," "Preset 2," and "Preset 3" in the dropdown list.



```
// -------------------------- Json Structure -------------------------- //
public class MapGraph
{
    public Dictionary<string, string[]> AdjacencyList { get; set; }
    public Dictionary<string, Position> Nodes { get; set; }
    public Dictionary<string, GraphEdge> Edges { get; set; }
}

public class Position
{
    public double WorldPosX { get; set; }
    public double WorldPosY { get; set; }
    public double WorldPosZ { get; set; }
}

public class GraphEdge
{
    public string From { get; set; }
    public string To { get; set; }
    public float Weight { get; set; }
}
```

Fig. 13. JSON Structure of Traffic Simulator.

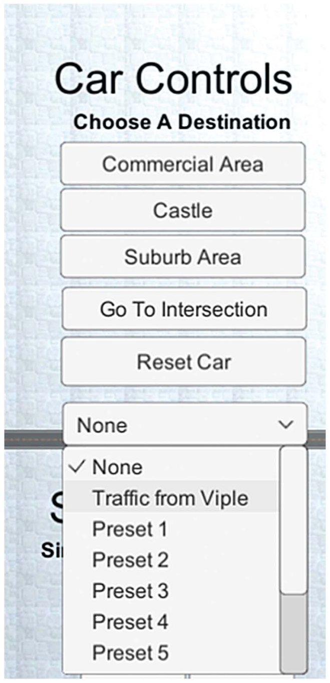


Fig. 14. Different Traffic Patterns in Dropdown List.

E. IMPLEMENT DYNAMIC DIJKSTRA'S ALGORITHM IN VIPLE

The VIPLE program received traffic from the simulator at every intersection and used the newest traffic information to reapply Dijkstra's algorithm to find the shortest path, so that the program would generate a path at every intersection that is shortest in terms of total distance and traffic congestion. The path was then sent to the traffic simulator to control the driving of the red car. The program has been divided into four parts as discussed below to demonstrate key concepts in detail.

1) RECEIVING MAP INFORMATION FROM THE SIMULATOR. The VIPLE code in Fig. 15 is used to receive the real-time map information from the traffic simulator. As shown in Fig. 12, the JSON structure received from the traffic simulator includes

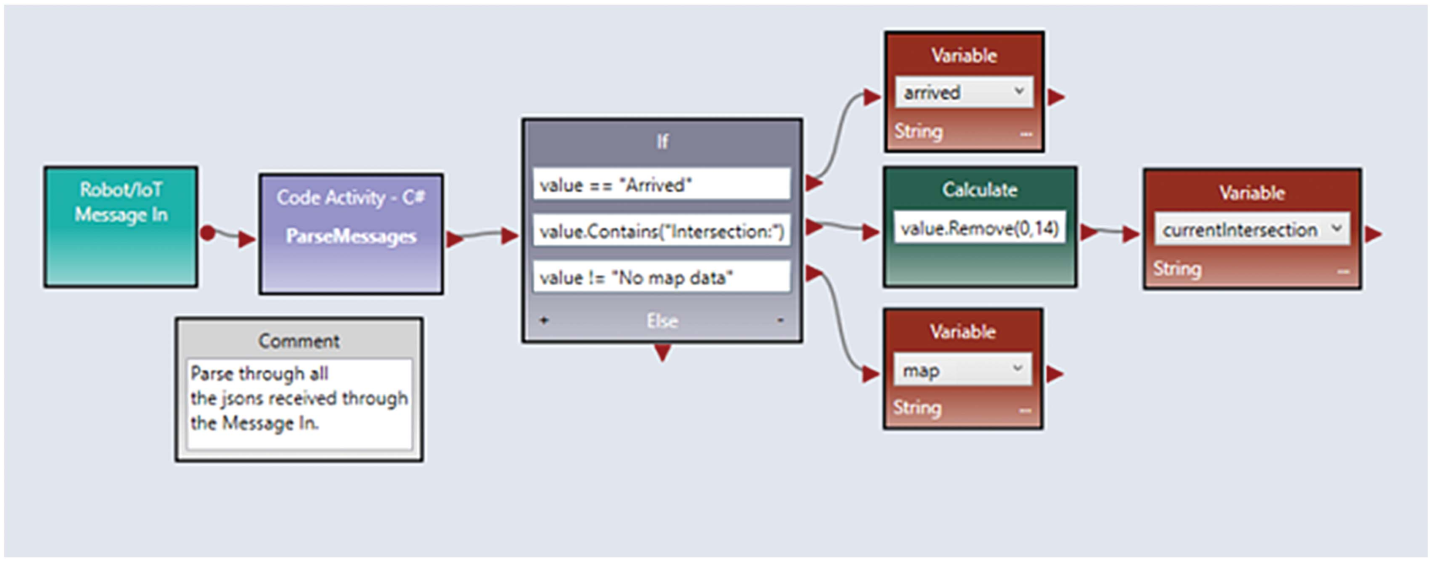


Fig. 15. Receive Map Information from the Simulator.

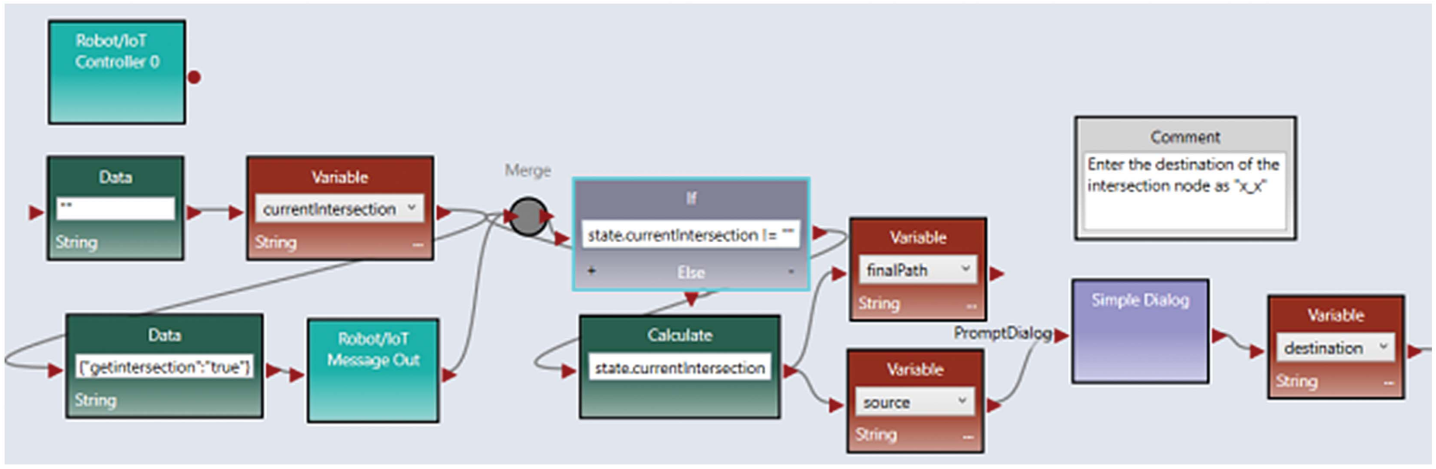


Fig. 16. VIPLE program for Starting Position and Destination.

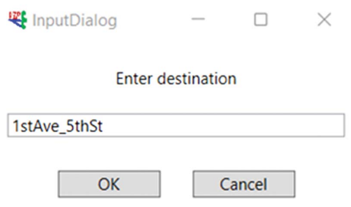


Fig. 17. Enter Destination Dialog.

information such as the edges, nodes, and position, each of which are static at every intersection. Additionally, the variable Weight changes over time because the weight takes distance and number of cars between two points into consideration.

2) RECEIVE STARTING POSITION FROM THE SIMULATOR AND DESTINATION FROM USER INPUT. The VIPLE program for this part is presented in Fig. 16. To receive the intersection where the red car is currently located, {"getintersection": "true"} is used to request the JSON information from the simulator. When the value has been received, it is stored in the source variable which represents the current intersection of the red car. Simple Dialog enables interaction from the user so they can enter the desired destination of the red car as shown in Fig. 17, then it stores it in the variable destination.

3) WHILE LOOP CALCULATING THE TRAFFIC AT EACH INTERSECTION. A while loop is used in the program to loop through all the intersections from the starting position of the red car to its destination. At every intersection, the program receives the up-to-date map information and the current intersection. Then, the program uses the new data in the Dijkstra's algorithm code to calculate a path with the current shortest distance and least traffic. This while loop is a key step to achieving the dynamic goal of the program. The basic structure is shown in Fig. 18.

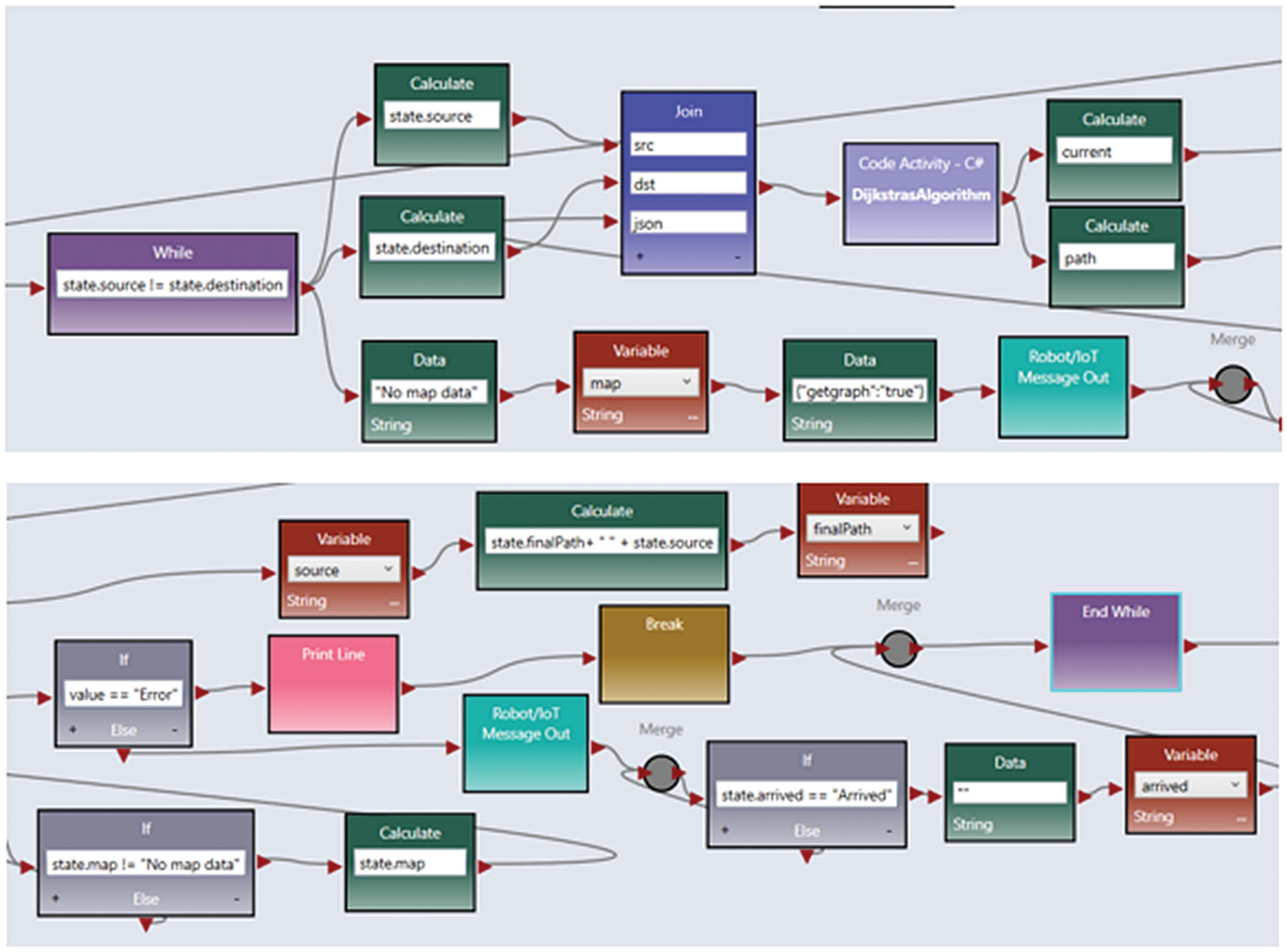


Fig. 18. While Loop.

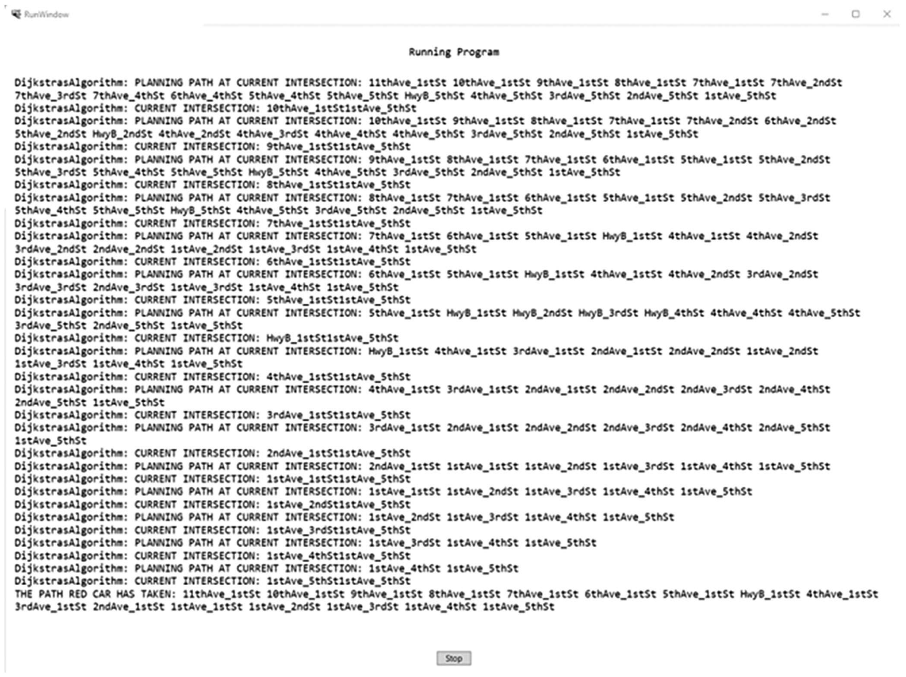
4) DIJKSTRA'S SHORTEST DISTANCE ALGORITHM. Dijkstra's algorithm, an algorithm to compute the shortest path based on a graph's edge weights, is implemented as a C# Code Activity in VIPLE. A min-heap is used as the data structure to store nodes, edges, weight, and the adjacency list information. The output of this C# code activity is stored in two variables. The first is the next intersection the red car will be at, which is output as current, then stored in variable source. The second one is output in path, which is the global path to the destination that Dijkstra's algorithm determined at the current intersection. As shown in Fig. 18, src, dst, and json are the inputs for Dijkstra's algorithm where src is the current intersection, dst is the destination intersection, and json is the JSON message that is received from the simulator at the current intersection. All three parts of the input are important parameters to be passed into the algorithm.

V. RESULTS AND DISCUSSION

This section presents the output of the implementation of Dijkstra's algorithm in VIPLE, and the corresponding routes shown in the traffic simulator. Figs. 19–21 are the outputs for the traffic pattern received from the simulator between 7:00 and 8:00. Figure 19shows the printed output of the planned global path at each intersection and the path the red car has taken. In particular, Fig. 19 highlights that the globally best path may change any number of times during execution due to changes in the traffic. The light blue path shown in Fig. 20 is the current planned global path. The dark blue path in Fig. 21 is the path that was actually taken by the red car to arrive at the destination, a history of where the car has traveled based on the dynamically updated global paths. The dark blue path corresponds to the printed output "THE PATH RED CAR HAS TAKEN" in Fig. 19.

VI. CONCLUSIONS AND FUTURE WORK

This paper presented a combination of generation of complete datasets from real traffic data, application of this traffic data in the traffic simulator, and dynamic routing algorithms using the generated traffic data. The generation of traffic data was based on real Maricopa County traffic data recorded in 2020. The hourly data for each road were translated manually. Then, the generated data were incorporated into the traffic simulator developed in the Unity game engine. Finally, routing algorithms were implemented in VIPLE by



```
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 11thAve_1stSt 10thAve_1stSt 9thAve_1stSt 8thAve_1stSt 7thAve_1stSt 7thAve_2ndSt
7thAve_3rdSt 7thAve_4thSt 6thAve_4thSt 5thAve_4thSt 5thAve_5thSt HwyB_5thSt 4thAve_5thSt 3rdAve_5thSt 2ndAve_5thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 10thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 10thAve_1stSt 9thAve_1stSt 8thAve_1stSt 7thAve_1stSt 7thAve_2ndSt 6thAve_2ndSt
5thAve_2ndSt HwyB_2ndSt 4thAve_2ndSt 4thAve_3rdSt 4thAve_4thSt 4thAve_5thSt 3rdAve_5thSt 2ndAve_5thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 9thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 9thAve_1stSt 8thAve_1stSt 7thAve_1stSt 6thAve_1stSt 5thAve_1stSt 5thAve_2ndSt
5thAve_3rdSt 5thAve_4thSt 5thAve_5thSt HwyB_5thSt 4thAve_5thSt 3rdAve_5thSt 2ndAve_5thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 8thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 8thAve_1stSt 7thAve_1stSt 6thAve_1stSt 5thAve_1stSt 5thAve_2ndSt 5thAve_3rdSt
5thAve_4thSt 5thAve_5thSt HwyB_5thSt 4thAve_5thSt 3rdAve_5thSt 2ndAve_5thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 7thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 7thAve_1stSt 6thAve_1stSt 5thAve_1stSt HwyB_1stSt 4thAve_1stSt 4thAve_2ndSt
3rdAve_2ndSt 2ndAve_2ndSt 1stAve_2ndSt 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 6thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 6thAve_1stSt 5thAve_1stSt HwyB_1stSt 4thAve_1stSt 4thAve_2ndSt 3rdAve_2ndSt
3rdAve_3rdSt 2ndAve_3rdSt 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 5thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 5thAve_1stSt HwyB_1stSt HwyB_2ndSt HwyB_3rdSt HwyB_4thSt 4thAve_4thSt 4thAve_5thSt
3rdAve_5thSt 2ndAve_5thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: HwyB_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: HwyB_1stSt 4thAve_1stSt 3rdAve_1stSt 2ndAve_1stSt 2ndAve_2ndSt 1stAve_2ndSt
1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 4thAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 4thAve_1stSt 3rdAve_1stSt 2ndAve_1stSt 2ndAve_2ndSt 2ndAve_3rdSt 2ndAve_4thSt
2ndAve_5thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 3rdAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 3rdAve_1stSt 2ndAve_1stSt 2ndAve_2ndSt 2ndAve_3rdSt 2ndAve_4thSt 2ndAve_5thSt
1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 2ndAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 2ndAve_1stSt 1stAve_1stSt 1stAve_2ndSt 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 1stAve_1stSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 1stAve_1stSt 1stAve_2ndSt 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 1stAve_2ndSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 1stAve_2ndSt 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 1stAve_3rdSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 1stAve_4thSt1stAve_5thSt
DijkstrasAlgorithm: PLANNING PATH AT CURRENT INTERSECTION: 1stAve_4thSt 1stAve_5thSt
DijkstrasAlgorithm: CURRENT INTERSECTION: 1stAve_5thSt1stAve_5thSt
THE PATH RED CAR HAS TAKEN: 11thAve_1stSt 10thAve_1stSt 9thAve_1stSt 8thAve_1stSt 7thAve_1stSt 6thAve_1stSt 5thAve_1stSt HwyB_1stSt 4thAve_1stSt
3rdAve_1stSt 2ndAve_1stSt 1stAve_1stSt 1stAve_2ndSt 1stAve_3rdSt 1stAve_4thSt 1stAve_5thSt
```

Fig. 19. Printed Output of the Dynamic Dijkstra's Algorithm.

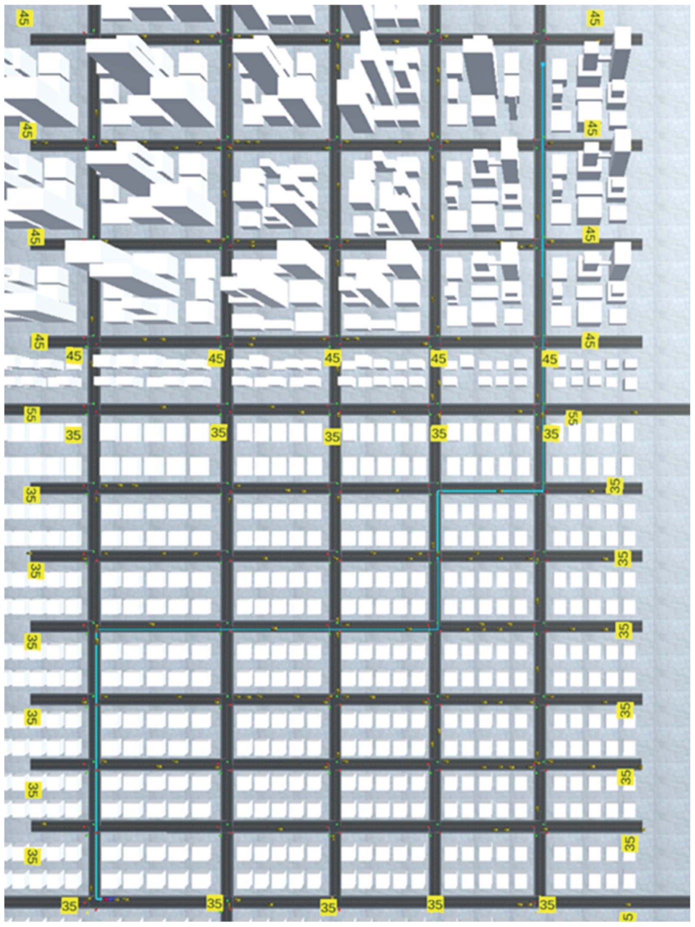


Fig. 20. Planning Path.

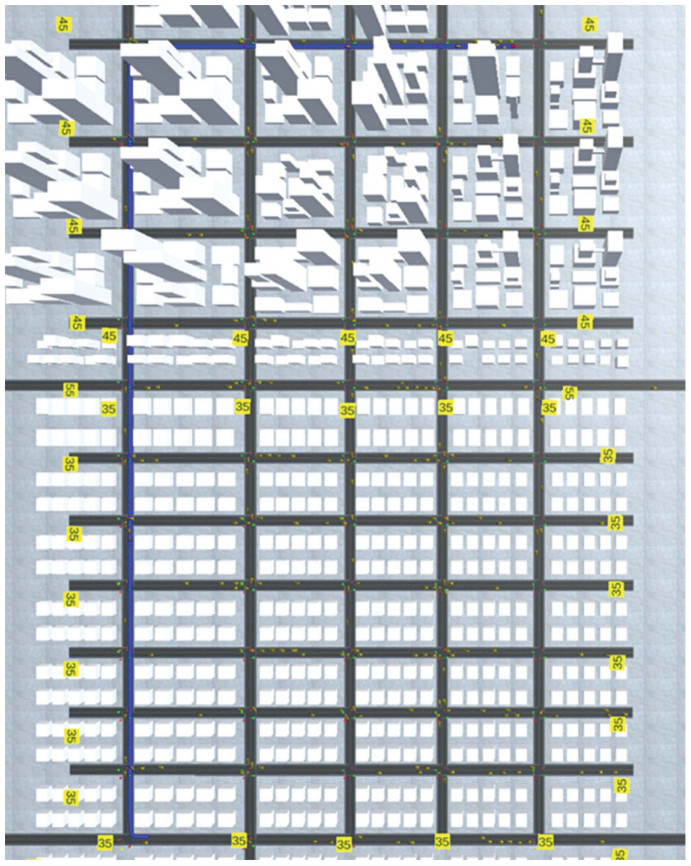


Fig. 21. Path Actually Taken.

using the dynamically changing traffic data in the simulator to update the red car's path. This work utilized the VIPLE traffic simulator. Furthermore, this work provided a visualization sample of dynamic routing algorithms for computer science education.

As discussed, real traffic generation in this work can be further improved in future projects. This part can be accomplished by collecting additional traffic data and applying more advanced ML techniques. One example of future research is to train a ML model to predict traffic on each road segment at certain times of the day based on the direction and distance from various city landmarks or popular areas, such as downtown buildings such as skyscrapers, amusement parks, and sports stadiums. Additionally, the same or a different model could be trained to incorporate automobile collisions, perhaps from a dataset with locations of the accident (such as on a freeway or a side street) or with traffic density. Then, the simulator could be updated to have car accidents occur, stalling traffic. These new traffic jams could then be used to verify the existing implementation of Dijkstra's algorithm and provide grounds for further research into more efficient dynamic path planning algorithms. For the traffic simulator, new functionalities can be incorporated to better simulate real traffic behavior and to show them on the map in real time to depict trends more accurately.

For a self-sustaining approach that uses present day data, an online ML application can be developed that automatically retrieves data from the traffic data website. Such an approach enables verification of student approaches using a variety of datasets and offers the possibility of performing research on modern traffic trends. For example, modern data can be analyzed to determine value of infrastructure changes.

ACKNOWLEDGEMENTS

This work was supported by several Arizona State University computer science and computer engineering senior capstone teams focused on maintaining and updating the traffic simulator. The team members include Isaiah Boblett, Nicholas Martinez, Adheesh Wadkar, Nathan Whitesides, Abdullah Masud, and Vuong Do. This research was overseen in part by Dr. Dijiang Huang and Dr. Yinong Chen.

FINANCIAL SUPPORT

This research is supported by the general fund of the IoT and Robotics Education Lab at Arizona State University.

REFERENCES

[1] Y. Chen and G. De Luca, "VIPLE: Visual IoT/robotics Programming Language environment for computer science education," *2016 IEEE Int. Parallel Distributed Process. Symp. Workshops (IPDPSW)*, 2016. DOI: 10.1109/ipdpsw.2016.55.

[2] G. De Luca, Z. Li, S. Mian, and Y. Chen, "Visual programming language environment for different IoT and robotics platforms in Computer Science Education," *CAAI Trans. Intell. Technol.*, vol. 3, no. 2, pp. 119–130, 2018. DOI: 10.1049/trit.2018.0016.

[3] G. De Luca and Y. Chen, "Semantic analysis of concurrent computing in decentralized IoT and robotics applications," in *2019 IEEE 14th International Symposium on Autonomous Decentralized System (ISADS). IEEE*, pp. 95–102, 2019.

[4] Q. Chen, Y. Chen, J. Zhu, G. De Luca, M. Zhang, and Y. Guo, "Traffic light and moving object detection for a guide-dog robot," *J. Eng.*, vol. 2020, no. 13, pp. 675–678, 2020.

[5] Wikipedia Foundation. *Traffic Simulation*. Wikipedia, Dec., 2020. Available: https://en.wikipedia.org/wiki/Traffic_simulation, Accessed on: Jan. 9, 2022

[6] Anylogic. Road traffic simulation software. AnyLogic Simulation Software, n.d. Available: https://www.anylogic.com/rod-traffic/, Accessed on: Jan. 9, 2022

[7] Caliper. Traffic simulation with TransModeler. Caliper Mapping Software and Transportation Software, n.d. Available: https://www.caliper.com/transmodeler/simulation.htm, Accessed on: Jan. 9, 2022

[8] SUMO. Simulation of Urban MObility. Eclipse SUMO – simulation of Urban mobility, n.d. Available: https://www.eclipse.org/sumo/, Accessed on: Jan. 9, 2022

[9] CityFlow. Welcome to CityFlow. CityFlow, n.d. Available: https://cityflow-project.github.io/, Accessed on: Jan. 9, 2022

[10] OpenTrafficSim. OpenTrafficSim Technical Manual. Atom, n.d. Available: https://opentrafficsim.org/manual/, Accessed on: Jan. 9, 2022

[11] M. Behrisch, J. Erdmann, and D. Krajzewicz, "SUMO–simulation of Urban Mobility: an overview," Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation. ThinkMind, 2011, pp. 23–28, 2011.

[12] D. Krajzewicz, M. Bonert, and P. Wagner, "The open source traffic simulation package SUMO," RoboCup 2006, 2006. Available:https://elib.dlr.de/46740/1/RoboCup2006_dkrajzew_SUMO.pdf

[13] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO – simulation of Urban mobility," *Adv. Syst. Meas.*, vol. 5, pp. 128–138, 2012.

[14] P. A. Lopez, E. Wiessner, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel and P. Wagner, "Microscopic traffic simulation using Sumo," *2018 21st Int. Conf. Intel. Transp. Syst. (ITSC)*, 2018. DOI: 10.1109/itsc.2018.8569938.

[15] J. Luo, X. Zeng, Y. Chen, and D. J. Sun, "Determining acceleration lane length on expressway weaving area using microscopic traffic simulation," in *Int. Symp. Intell. Trans. Smart City (ITASC) 2019 Proc.*, pp. 13–27, 2019. DOI: 10.1007/978-981-13-7542-2_2.

[16] S. Krauß (1998). Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics. Hauptabteilung Mobilität Und Systemtechnik Des DLR Köln, technique report 98-08. Retrieved from: https://sumo.dlr.de/pdf/KraussDiss.pdf

[17] E. Brockfeld and P. Wagner, "Calibration and validation of Microscopic Traffic Flow models," Traffic and Granular Flow '03, pp. 67–72, 2004. DOI: 10.1007/3-540-28091-x_6.

[18] E. Brockfeld and P. Wagner (2004). Testing and Benchmarking of Microscopic Traffic Flow Simulation Models. Proceedings of the 10th World Conference on Transport Research, Istanbul (Turkey), pp. 775–776. Retrieved from: https://elib.dlr.de/6709/1/WCTR2004_Brockfeld.pdf

[19] M. Berglund, T. Raiko, M. Honkala, L. Karkkainen, A. Vetek and J. Karhunen, Bidirectional recurrent neural networks as generative models," in *IEEE Conference on Neural Information Processing Systems. Neural Information Processing Systems Foundation*, vol. 1, pp. 856–864, 2015.

[20] R. Asadi and A. C. Regan, "A convolution recurrent autoencoder for spatio-temporal missing data imputation," *ArXiv*, 2019. abs/1904.12413.

[21] O. Benkraouda, B. T. Thodi, H. Yeo, M. Menendez, and S. E. Jabari, "Traffic data imputation using deep convolutional neural networks," *IEEE Access*, vol. 8, pp. 104740–104752, 2020. DOI: 10.1109/access.2020.2999662.

[22] C. Zheng, X. Fan, C. Wang and J. Qi, "GMAN: a graph multi-attention network for traffic prediction," *Proc. AAAI Conf. Artif.*

Intell., vol. 34, no. 01, pp. 1234–1241, 2020. DOI: 10.1609/aaai.v34i01.5477.

[23] Y. Ye, S. Zhang, and J. Yu, “Spatial-temporal traffic data imputation via graph attention convolutional network,” *Lecture Notes Comput. Sci.*, pp. 241–252, 2021. DOI: 10.1007/978-3-030-86362-3_20

[24] Y. Liang, Z. Zhao, and L. Sun, “Dynamic spatiotemporal graph convolutional neural networks for traffic data imputation with complex missing patterns,” ArXiv:2109.08357, 2021.

[25] G. De Luca and Y. Chen, “Visual IoT/robotics programming language in Pi-calculus,” *IEEE 13th Int. Symp. Auton. Decentralized Syst. (ISADS)*, 2017. DOI: 10.1109/isads.2017.32.

[26] Maricopa County. n.d.. Available: https://www.maricopa.gov/858/A, Accessed on: Jan. 9, 2022

[27] R. Venkatanarayana, B. L. Smith, and M. J. Demetsky, “Quantum-frequency algorithm for automated identification of traffic patterns,” *Transp. Res. Record: J. Transp. Res. Board*, vol. 2024, no. 1, pp. 8–17, 2007. DOI: 10.3141/2024-02.