



PB98-121965

Animation of Traffic through Roundabouts

By John Larson III

Bryan Pearce

Per Garder

HomeWork 8 - Your Name Here

LAYOUT PLAN

Allowable Gap in Seconds

Accelerations in g's

Yield Point

Draw Segments

Show Output Data

Show Delay Data

Single Step

SetUp Run Stop Clear

Elapsed Time Seconds

Input Data



Exhibit C

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.	
4. Title and Subtitle Animation of Traffic Through Roundabouts		5. Report Date January 14, 1998	
		6. Performing Organization Code	
7. Author(s) John Larson III, Bryan Pearce, Per Garder		8. Performing Organization Report No.	
9. Performing Organization Name and Address Department of Civil Engineering University of Maine Orono, ME 04469		10. Work Unit No. (TRAIIS)	
		11. Contract or Grant No. DTRS95-G-0001	
12. Sponsoring Agency Name and Address New England (Region One) UTC Massachusetts Institute of Technology 77 Massachusetts Avenue, Room 1-235 Cambridge, MA 02139		13. Type of Report and Period Covered Final Report 9/1/96 - 12/31/97	
		14. Sponsoring Agency Code	
15. Supplementary Notes Supported by a grant from the US Department of Transportation, University Transportation Centers Program			
16. Abstract This report describes work done on a roundabout animation program during 1997. The roundabout animation program began as an undergraduate class project and has evolved to its current state. The program is based on the principle of an autonomous agent. The cars are programmed to speed up, to slow down, and to enter the roundabout based on an acceptable gap length. That is, the gap between themselves and the cars around them. The actual gap is compared to an allowed gap that is based on vehicle speed and assumed driver response time. If necessary, the vehicle speed is adjusted. The cars travel through the roundabout following a randomly assigned path. Traffic flow values may be input into the program manually during initialization. During simulation, the cars enter and exit randomly based on these values. After the simulation, traffic count data and average delay data may be displayed.			
17. Key Words Roundabout, Traffic Circle, Simulation, Animation, Autonomous Agent, Delay, Computer Simulation		18. Distribution Statement	
19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages 22 + Appendix	22. Price \$25,000

Abstract:

This report describes work done on a roundabout animation program during 1997. The roundabout animation program began as an undergraduate class project and has evolved to its current state. The program is based on the principle of an autonomous agent. The cars are programmed to speed up, to slow down, and to enter the roundabout based on an acceptable gap length. That is, the gap between themselves and the cars around them. The actual gap is compared to an allowed gap that is based on vehicle speed and assumed driver response time. If necessary, the vehicle speed is adjusted. The cars travel through the roundabout following a randomly assigned path. Traffic flow values may be input into the program manually during initialization. During simulation, the cars enter and exit randomly based on these values. After the simulation, traffic count data and average delay data may be displayed.

Preface:

About two years ago, the students of CIE 115, a course in programming for first year students in civil engineering at the University of Maine, set about the task of writing code that would simulate the motion of traffic in a roundabout. The program was to be based on the theory of an autonomous agent. That is, the cars would be modeled as if they had drivers inside them controlling the actions of the car. The class was split up into groups so the students could better devise a plan of attack. My group created a program with some nice features, but it was far from perfect. It occasionally drew lines where it was not supposed to draw lines, and cars frequently crashed into one another. That fall I was a little amazed to be asked to write a report about the project we had done the previous semester. Unfortunately, at the time I was so involved in other activities that I could not accept the job. During the spring, Bryan Pearce, the professor of CIE 115, wondered if I might become the student assistant for the course. This time I could and did accept. For a semester, I helped the other students with their programming and eventually with their own versions of simulated traffic circles.

When I was looking for a summer job, I asked Professor Pearce if he knew of anyone who might be willing to hire me. He said, yes. He said that he and Professor Per Garder would employ me to continue the traffic circle project over the summer. So here I am; finally working on that report, and hopefully taking the circle a few steps further than when I last left it.

John Larson

Table of contents

Abstract	2
Preface	3
Introduction	5
House Rules	6
Coordinate System	7
Setup	7
The Code – The House Rules in Depth	8
- General Declarations	8
- Initial Code	10
- PathCalculations	11
- Timer	12
- AddCars	13
- AdjustSpeeds	17
- MoveCars	20
Conclusion	22
Appendix A – The Setup Program	A-1
Appendix B – The Code	B-1

Introduction:

When traffic volumes at an intersection increase to a point where the travel time through it becomes long, or they increase to a point where the intersection becomes unsafe, something should be done. Usually in the United States, the solution is to use a traffic light. However, this does not always solve the delay problem. If done right, the traffic circle can be a more efficient and safer solution.¹²³⁴

Our program simulates cars travelling through a traffic circle. Since actual traffic data changes from day to day, and from rush hour to nighttime, we have designed the program so the user can edit the traffic flow data. The program simulates the traffic flow by calculating the movement in time steps using a computer object called a Timer. This device repeats the code written within it until the timer is turned off. With each repetition, or timestep, a certain amount of "model time" passes. The variable deltaT holds the value of this time step, for example 0.5 seconds. The total model time that the program runs can be changed as well as the time that data starts recording. The traffic simulator will show the cars moving along their appropriate paths with relatively few "crashes". The user can also view the "traffic counts" generated by the program, these can be converted to vehicles per hour (when the simulation is complete), and the average time that the cars are in the simulation. Other options allow the user to see the actual paths the cars are moving along, and to have the timer operate one step at a time.

The program development started with the traffic circle as an octagon, centered in the window. From there, it has progressed to a circle with entrances and exits. We have now simulated the recently constructed Gorham Traffic Circle. (Figure 1.) The setup for the Gorham Circle was cumbersome and we hope that in the future the setup can be simplified so any circle can be easily simulated.

¹ Retting, Richard, 1996. Urban Motor Vehicle Crashes and Potential Countermeasures. Transportation Quarterly 50/3:19-31.

² Schoon, C.C., and J. Van Minnen, 1993. Accidents on Roundabouts. R-93-16 SWOV - Stichting Wetenschappelijk Onderzoek Verkeersveiligheid. The Netherlands.

³ Ourston, Leif, 1994. Nonconforming Traffic Circle Becomes Modern Roundabout. Leif ourston and Associates, Santa Barbara, California, 93111.

⁴ Jorgensen, Else, and N. O. Jorgensen, 1994. Safety of 82 Danish Roundabouts. Report 4 - IVTB, Danish Technical University.

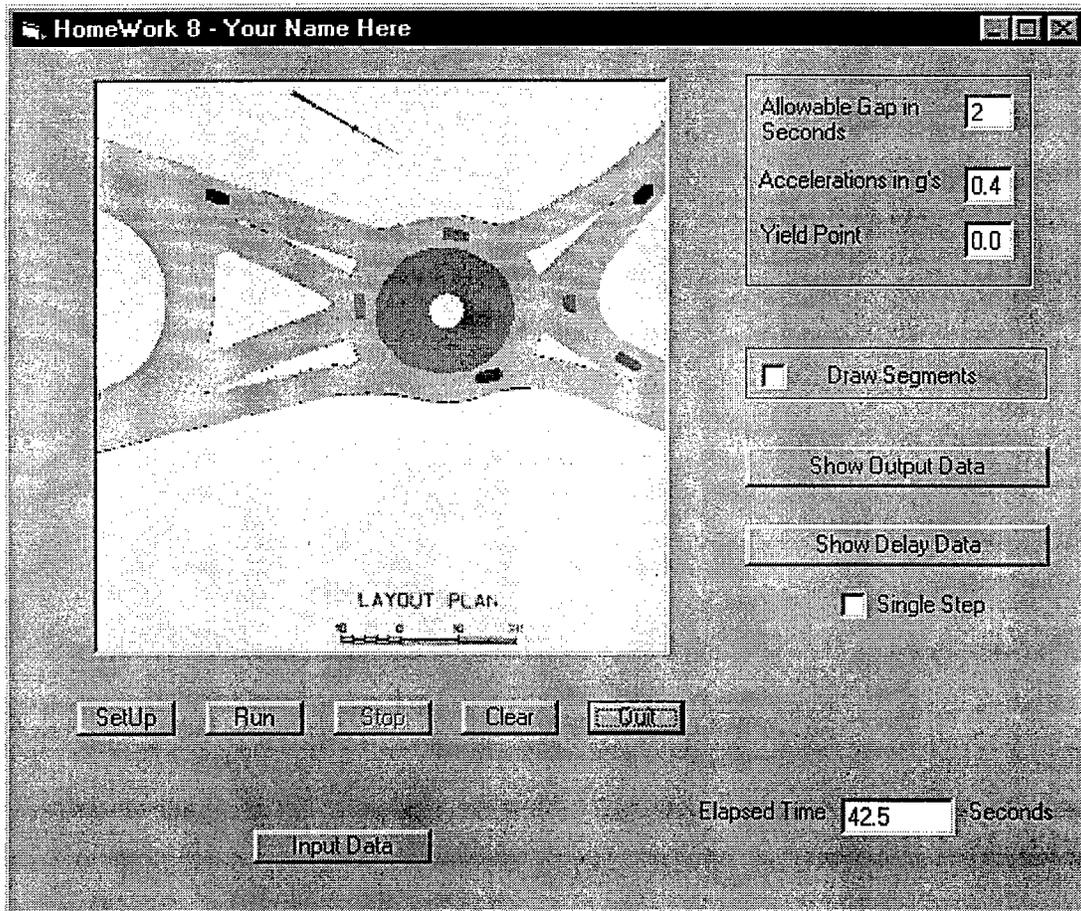


Figure 1 – The Main Form

As in the beginning, we still employ the concept of autonomous agents to run the cars. When the program is initially set up, each car (the agent) is assigned a set of characteristics that help to define how it should act as it travels through the traffic circle. The car then follows the traffic laws, or the computer code, according to those characteristics.

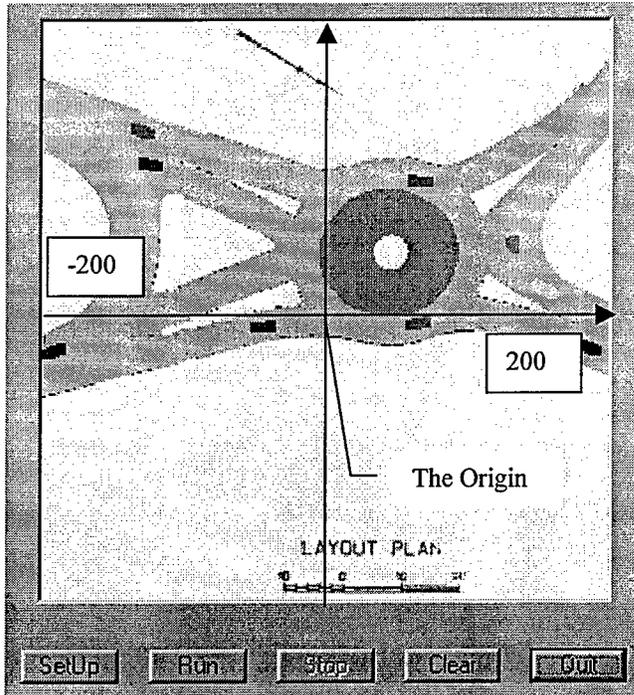
House Rules:

Prior to writing the program, we defined a set of rules that would describe how the cars behave. First, how could we keep the cars from crashing into one another? How is it done in real life situations? Drivers adjust their speed to match that of the car in front of them. Therefore, they will decelerate as soon as they feel they are in danger of hitting that car. Different drivers will do this at different times, depending on how fast they are going.

We developed a system such that if a car follows another too closely, then the car in back, or the backcar, will slow.

Next, we needed to decide how the cars would enter the simulation, and where they would enter the simulation. Based on traffic volumes the cars are randomly entered into the simulation. This also depends on the physical space available, that is, a car may not occupy

an already occupied space. Once the cars enter, how do they know where to go? In real life, a driver usually knows his or her destination, we randomly assign each car a specific exit, based on traffic volume. This will be explained further in AddCars.



Coordinate System:

The coordinate system used is standard Cartesian, ranging from -200 to 200 feet in both directions (see Figure 2). We previously used a modified polar coordinate system with

Figure 2- The Coordinate System the round circle that was centered in the middle of the screen. However, the conversion to the Gorham traffic circle, dictated that we revert to the more general Cartesian system.

Setup:

With the polar coordinate version, setting up the traffic circle was fairly easy. Write the code to draw a bunch of segments and/or arcs a certain way, then set their properties, such as

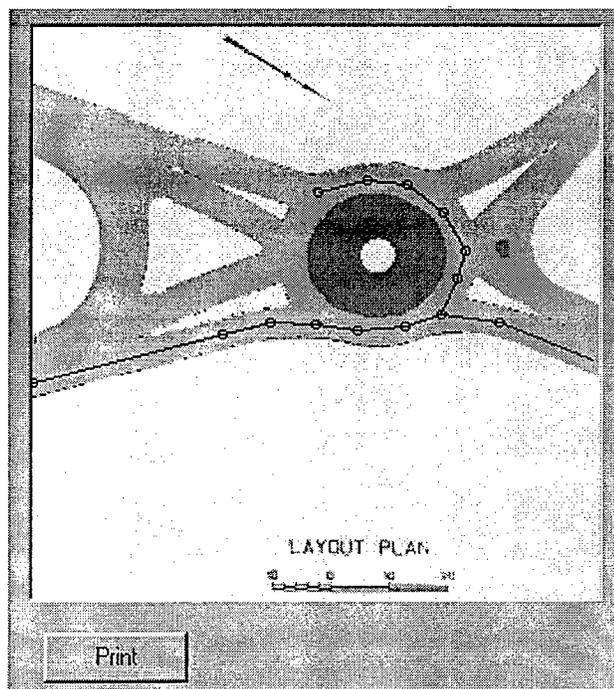


Figure 3 – Set Up of Segments

the end points. When these segments have to be projected on top of an image of an existing traffic circle, the task is not as simple. To work around this problem, a separate program was created (see Appendix A). The program allows one to click on the places where the segments' endpoints are desired. The program then prints out the x-y coordinates for those points (Figure 3). From there, we had to "hardwire everything" so the traffic circle would run properly. That is, we had to manually write all the code that set the properties. Unfortunately, this is not the ideal set up solution; one goal of future versions will be to overcome this problem.

Code – The House Rules in Depth:

General Declarations –

Before we get too involved in the explanation of the code, we should describe what variables are being used and why. Green annotation is original to the code; blue annotation has been added for clarification.

```
Private Type Point 'this is a variable type defined by us to hold the x and y
coordinates of the endpoints of the segments
```

```
    x As Single 'the x coordinate
```

```
    y As Single 'the y coordinate
```

```
End Type
```

```
Private Type segment 'another variable type we made up this defines the paths
```

```
    carsIn(20) As Integer 'list of cars in each segment – last car first
```

```
    totCars As Integer 'number of cars in a segment
```

```
    endPt As Integer 'index of segment ending point
```

```
    length As Single 'length of segment in feet!!!!!!!!!!!!!!!!!!!!!!
```

```
    leftSegs(2) As Integer 'when entering the circle look for cars on the two
segments to the left
```

```
    nextSegL As Integer 'index of next segment continuing in circle - "0" if none
```

```
    nextSegR As Integer 'index of next segment leaving circle - "0" if none
```

```
    startPt As Integer 'index of beginning point of segment
```

```
    slope As Single 'the slope of the segment
```

```
End Type
```

Private Type Car 'this type defines the properties of the cars
 active As Boolean 'if a car is being used or not
 color As Long 'the car's color
 deSpeed As Single 'desired speed or how fast the car "wants" to go
 length As Single 'the car's length
 width As Single 'the car's width
 location As Single 'location in the segment, ratio of position to length, it has
 a value between zero and one, with zero being the beginning of the segment
 new As Boolean 'if this is true the program will know not to "erase" the car
 after the first time step
 nextseg As Integer 'next segment car is headed for, "-1" for an exit segment
 and "0" if not assigned yet
 segment As Integer 'number of the segment
 speed As Single 'actual speed
 exit As Integer 'assigned exit segment
 begintime As Single 'time the car enters
 entrance As Integer 'segment the car enters on

End Type

Private myPts(50) As Point 'array of points
 Private mySegs(50) As segment 'array of segments
 Private myCars(100) As Car 'array of cars
 Private oldFront(100) As Point, oldBack(100) As Point 'these hold the old positions
 of the cars
 Private numCars As Integer 'number of cars
 Private numSegs As Integer 'number of segments
 Private deltaT As Single, yieldPt As Single 'the time that passes each timestep, the
 point where the cars "look" to enter the circle
 Private black As Long 'the color black
 Private carFront As Point, carBack As Point 'endpoints of the cars
 Dim TimeSteps As Long 'number of timesteps
 Dim Leftt(13 To 16) As Single 'these determine where cars exit
 Dim Straight(13 To 16) As Single

Dim Right(13 To 16) As Single

Dim PutCarInNow(13 To 16) As Single 'the probability that a car will enter

Dim counter(4, 4) As Integer 'keeps track of the mean delay time

Dim frontcar As Integer, backcar As Integer 'these six variables are used to control speed, the car in front, the car in question

Dim thisseg As Integer, nextseg As Integer 'the segment backcar is on, the segment it is going to

Dim gap As Single, allowedGap As Single 'the gap between the two cars, and minimum gap allowed between them

The MSFlexGrid

	1	2	3	4	
1	South Bound	60	60	60	0
2	East Bound	60	60	60	0
3	North Bound	60	110	60	0
4	West Bound	60	60	60	0

You can change the data in the grid by entering the row and column numbers and the new data in the appropriate boxes, then clicking on the update grid button. Below enter the run length in minutes, and the length of time before data will be recorded in minutes. When done, click O.K.

Run for Minutes
Exclude the First Minutes

O.K.

Figure 4 - frmInput

These of course are not all of the variables used. They are, however, the major ones. The others shall be described as needed.

Initial Code –

When the program first starts up, only one button is active. That is the Input Data button on the main form (see Figure 1.). When this button is clicked a separate form

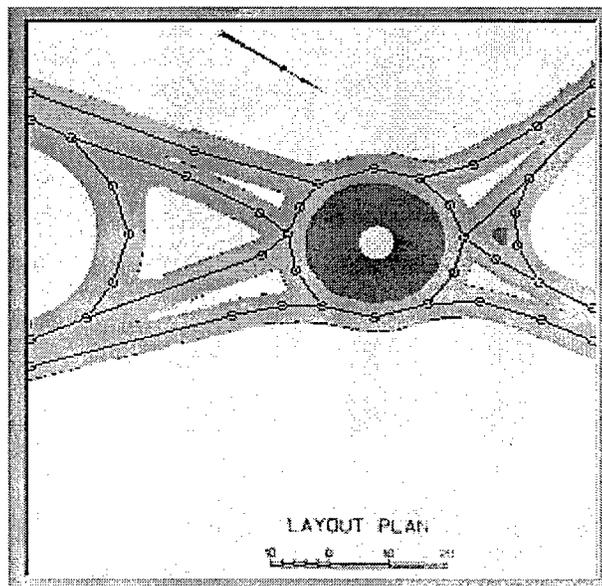


Figure 5 – The Segments

comes up that allows the controlling data for the program to be edited. This new form, frmInput (see Figure 4), is set up with an MSFlexGrid control, and holds the vehicles per hour data for each path. A method for editing this data, and a way to change the time for which the program will run, or runtime, and the time at which the output data starts recording are also included on this form. The default data is entered under the form_load event, that is, when the form loads into memory.

Once this is finished, the Setup button becomes active (Figure 1.). The code under (or associated with) this button is straightforward. It initializes variables and properties to be used later, and it calls procedures that setup the cars, points, and segments.

CarSetup loops through all the cars in the myCars array and makes sure that the active property is set to false. It also sets the dimensions, color and desired speed for each car. The New property is set to true so the program knows that this is the first time each car will be drawn.

In PointSetup the program opens the file point.txt and reads in the x and y coordinates for each of the points. Editing point.txt in Notepad, or Wordpad, can change the points' coordinates. EndPointSetup "hardwires" the indexes of the points that define each segment. SegSetup sets the length and slope properties of each segment, and sets the leftSegs, nextSegL and nextSegR properties. These last two variables help set up the direction in which the traffic will move. They are set up according to the direction of movement. NextSegR will be the segment on the right, if there is a choice. If there is no choice, then nextSegR will have the value of zero and nextSegL will be the next segment. LeftSegs are used to see if there is room to enter the actual circle. If there is a car approaching the circle and there is another car within the circle to the left of the intersection, on one of the LeftSegs, then there is no room to enter and the first car must wait. SegSetup will also draw the segments and points over the background if the checkbox on the main form tells it to do so (see Figure 5).

PathCalculations –

The Setup button also calls the procedure PathCalculations. This procedure implements the logic that will decide where the cars will enter the circle, when they will enter the circle, and where they will go once they do. PathCalculations reads the data from the MSFlexGrid on frmInput (Figure 4), and for each entrance, it calculates the probability that a

car would turn left or right, go straight, or make a U-turn. It will also calculate the probability of cars entering each entrance each timestep.

In computing the probabilities, PathCalculations gives values to three variables for each entrance: Leftt, Straight, and Right. Right is always calculated first. Its value is the simple probability that a car will turn right if it starts on the appropriate entrance. Next, the value for Straight is set. This is the sum of the simple probability of straight going cars and the value of Right. Last, the simple probability of left turning cars is added to Straight to get the value for Leftt. (Leftt is spelled with two *t*'s to avoid internal conflicts within Visual Basic.) I will explain why their values are found this way when I explain AddCars. The last thing done in PathCalculations is to originate the PutCarInNow variable for each entrance. The value for these variables is the probability that a car would enter on the appropriate entrance in any timestep.

Timer –

Once the code for the Setup button is finished, the Run button becomes active. Clicking this button enables the Timer, enables the Stop button, and disables itself. The Stop button simply does the opposite of the Run button. However, the Timer becomes the central nervous system of the whole program (see Figure 6). It calls

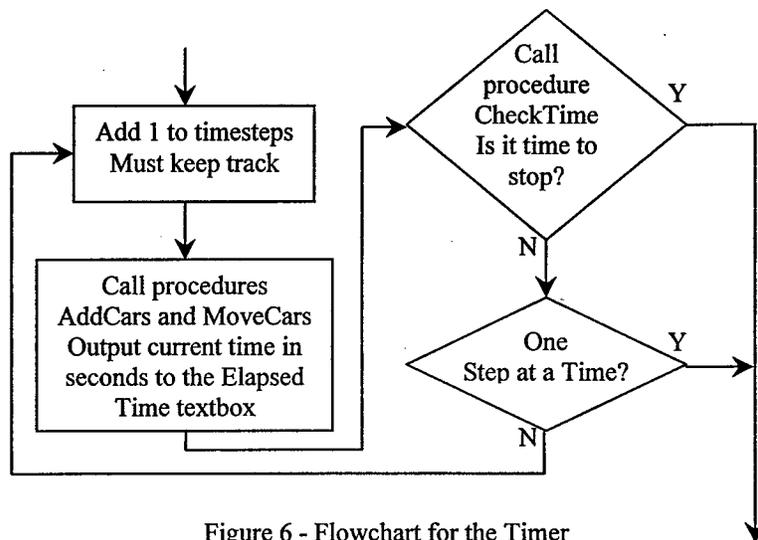


Figure 6 - Flowchart for the Timer

the two major components of the animation control, AddCars and MoveCars. If the Single Step checkbox is checked then the Timer will turn itself off and turn the Run button back on. This will continue until the box is unchecked. In addition, the Timer calls the procedure CheckTime. This checks to see if it is time to stop the animation and relevant computations, and will do so if it is.

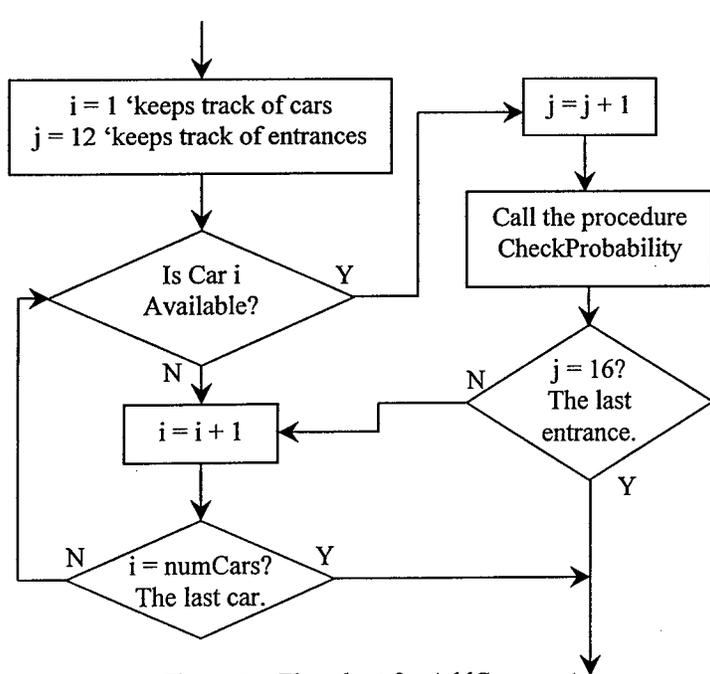


Figure 7 – Flowchart for AddCars

AddCars –

AddCars searches all the cars being used until it finds the first four that are inactive (see Figure 7). Each car is given a chance to enter one of the four entrances. CheckProbability determines if a car will enter or not (Figure 8). It does this by first returning a random number.

If this number is less than the appropriate PutCarInNow variable, then CheckProbability will let the car enter and the procedure EnterNow will be called.

Before it does anything else EnterNow checks to see if there is the “physical” space on the entrance segment to add a car. In other words, if there is no car on the entrance segment or the car that is there is beyond a certain distance, then a car can be added. Otherwise, no car will enter this segment on this timestep.

When the car is allowed on, EnterNow makes the car active and sets its location to the beginning of the entrance segment. Next, the car’s destination is made known using the FindExit procedure. This is where the other variables defined in PathCalculations come into play.

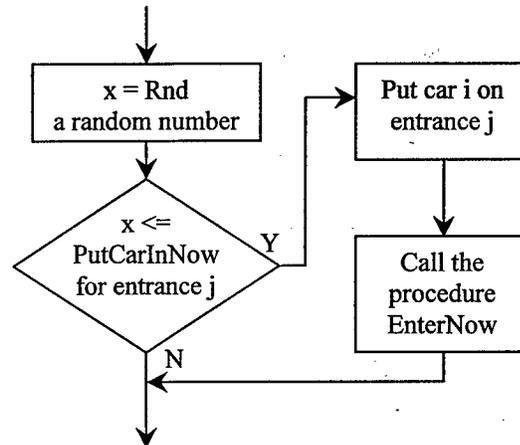


Figure 8 – Flowchart for CheckProbability

FindExit is based on the logic that the sum of all the probabilities of a car taking one of the possible paths is one. Remember when finding the values for Right, Straight, and Leftt we kept adding the previous variable to the correct probability to get the next. Here is why. If we use a random number generator to get a number between zero and one, we can use these variables to determine which direction the car will turn. Here’s a section of the code with an example.

segin = myCars(i).segment ‘the entrance the car is on.

```

x = Rnd 'a random number between 0 and almost 1, let's say 0.754832
If x < Right(segin) 'defaulted to 0.3333... - hmm, not less than this
    Then direction = "right"
ElseIf x < Straight(segin) 'defaulted to 0.666... - or this either
    Then direction = "straight"
ElseIf x < Leftt(segin) 'defaulted to 1.00 - ah here we are
    Then direction = "left"
Else 'with the default data there are no u-turns
    direction = "back"
End If

```

The sample car will be turning left. This part has been set up so the exits will be randomly picked, but also they will depend on the vehicles per hour that should follow the paths. x is found using the Rnd function, a random number generator. If it is less than Right the car will turn right; less than Straight, straight; less than Leftt, left. If, however, the random number is greater than Leftt the car will make a u-turn through the circle.

Once the direction is found, the index of the exit segment needs to be assigned to the car's exit property. (See Figure 9 for indexes of the Segments.) "Case statements" that are based on the entrance on which the car is, that are nested within "case statements" that are based on the direction of the car, can do this. This method may seem a bit cumbersome and unwieldy; however, it was the simplest and the most straightforward way to be done. Nested case statements are used quite often throughout the program, because of the decisions that need to be made that rely on two variables.

EnterNow calls the FindNextSegment procedure after executing FindExit. This procedure goes through several *if* statements in a series, sometimes referred to as a *block If* statement. First, we check to see if there are any segments beyond the segment that the car in

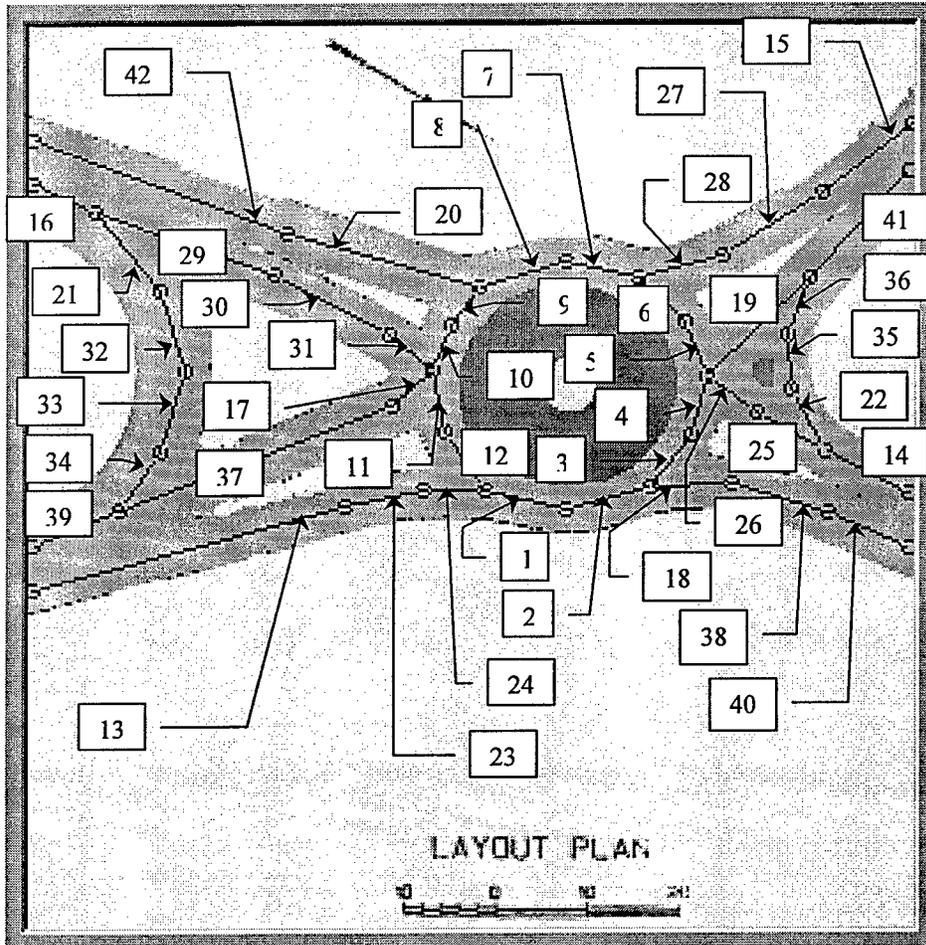


Figure 9 – Indexes of Segments

question is on.

In this case, there has to be at least one, since a car just entered the circle, but if there are no segments then the car's nextseg property is set to negative one. That way the program will know what to do and remove the car from the simulation.

Next, we want to know if there is a choice of going

left or right. If not, the segment's nextSegL property will become the car's nextseg property. However, if there is a choice, we need to know if this is where the car will turn. If the car exits here then nextseg becomes the nextSegR property; otherwise, it is nextSegL. Interested readers may want to look at the code:

```
segin = myCars(i).segment 'segment the car, i, is in Let's say we're on entrance 14
(see Figure 8)
```

```
If mySegs(segin).nextSegL = 0 Then 'we are on the exit ramp But we're not, we're
on segment 14
```

```
myCars(i).nextseg = -1
```

```
ElseIf mySegs(segin).nextSegR = 0 Then 'we turn left here
```

```

myCars(i).nextseg = mySegs(segIn).nextSegL
ElseIf myCars(i).exit = mySegs(segIn).nextSegR Then 'but we said we wanted to go
left (Page 12) through the circle, this is not our exit.
myCars(i).nextseg = mySegs(segIn).nextSegR
Else 'so nextseg becomes segment 25 (Figure 9)
myCars(i).nextseg = mySegs(segIn).nextSegL
End If

```

This procedure has worked very well, and has been only slightly modified since the spring course in 1996. The most important of these was in changing how the cars decided to turn off the circle. In the original homework program, the decision was made by probability within FindNextSegment. Now the exit is already known, a priori, all we need to do is compare nextSegR with the exit.

Once we have found the next segment for which we are heading, we need to fix the carsIn array and the totCars property for the entrance segment. When updating the array, we move all of the cars in the array to the next highest slot in the array and put our car in slot

one. After this, we must increase the totCars, or total cars, property by one.

Now that we know where the car has entered and where it will

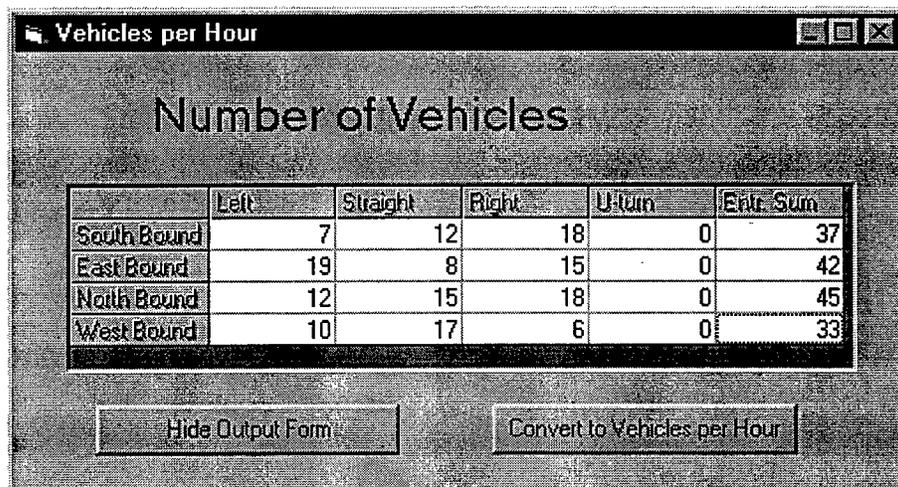


Figure 10 - frmVPH

exit, we can update the grid on frmVPH (see Figure 10). EnterNow calls the procedure UpdateOutput to accomplish this task. Of course, UpdateOutput will do nothing unless the elapsed time has reached the time at which data starts recording. When it is time to record data, the procedure enters another nested case statement (this one relies on the car's entrance and exit) to determine which cell in the grid to increment by one.

Before the car moves, we must adjust the car's speed so it will not be likely to crash into the car in front of it. To do this, we must call the procedure AdjustSpeeds twice. This

may seem like the car will be over-accelerating; however, it really is not. By calling AdjustSpeeds twice, we are, in actuality, adjusting the speed of the car along the long approach to the circle, the approach that would normally be beyond what is represented in the simulation.

The last two steps EnterNow performs are to set the car's begintime property and the car's entrance property. These will be used later in the procedure FindDelayTime.

AdjustSpeeds –

First, we need to find the values for the variables we will be using. In AdjustSpeeds, we set values to the variables: accel, taken from the acceleration text box on frmMain; backcar, the car being controlled (conceptually the car in which we are riding); thisseg, the segment backcar is on; and nextseg, the segment to where backcar is heading. At this point, AdjustSpeeds enters a Do loop in order to find a value for j. This is the index of the backcar's place in the carsIn array for thisseg. j becomes useful when trying to find the value of frontcar, the car in front of backcar. Here we set the values of lookLeftSeg and lookLeftCar. They are used if the car is approaching an intersection. What happens then will be explained later.

Next, the Boolean variable flag is set to false. This variable will let the program know if we had to slow backcar because it was getting too close to frontcar. Now, we find the value of allowedGap, using the equation: $\text{allowedGap} = \text{CSng}(\text{txtGap.Text}) * \text{myCars}(\text{backcar}).\text{speed} ^ 1.5 + 2 * \text{myCars}(\text{backcar}).\text{length}$. The CSng(txtGap.Text) is the value taken from the text box on frmMain. MyCars(backcar).speed is backcar's speed. It is taken to the power of 1.5 to make allowedGap small when the speed is low and large when the speed is high. The length of backcar is entered into the equation to account for the fact the gaps are measured from the centers of the cars. This length is multiplied by two as a safety factor.

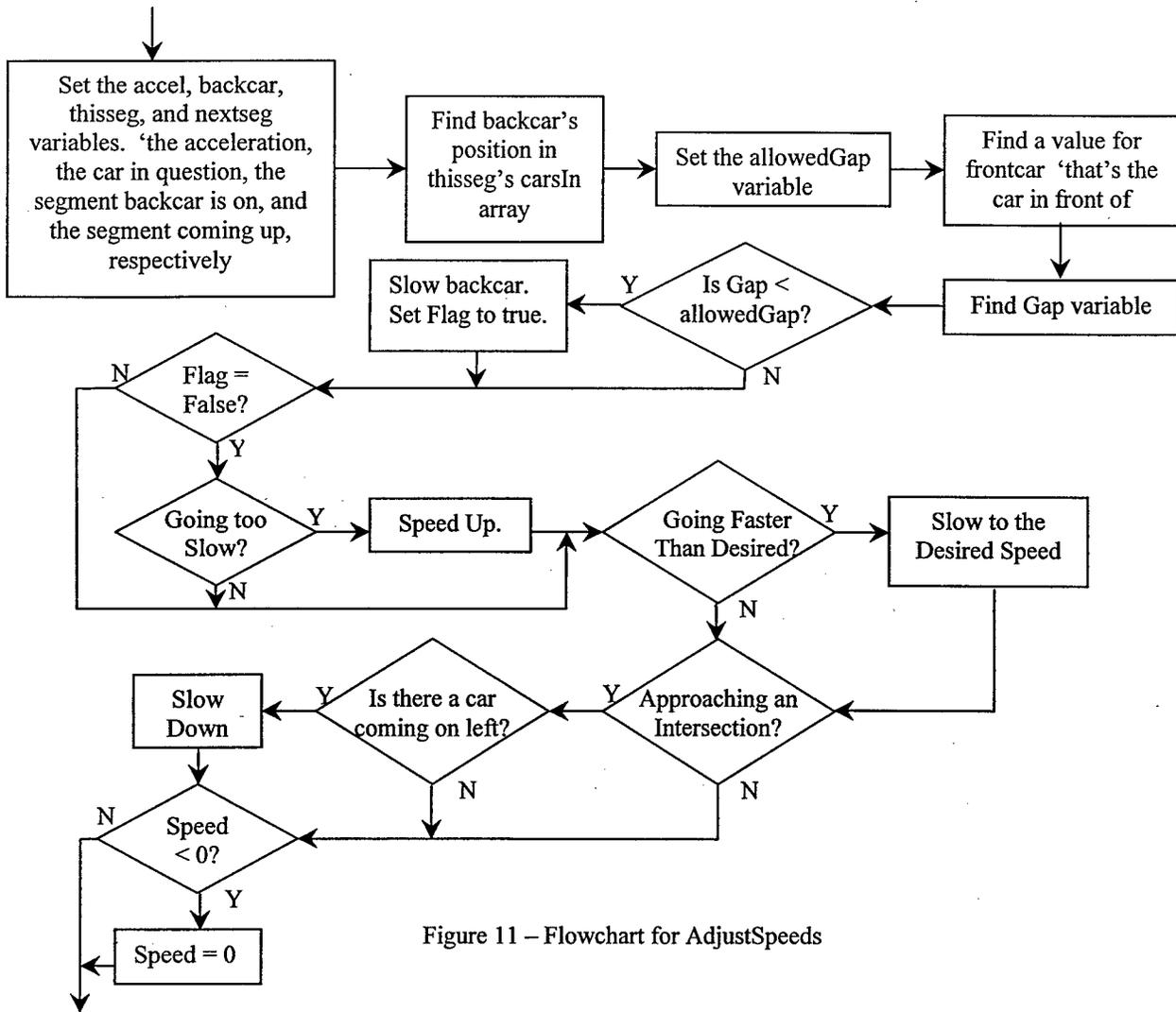


Figure 11 – Flowchart for AdjustSpeeds

Once these variables are defined, we can find the value for frontcar. First, we check thisseg. Because of the way the carsIn array is set up, we can see if frontcar is on thisseg by adding one to j and then checking that spot in the array for a number other than zero. If we get a number then that will be frontcar, and Gap's value will be the difference of frontcar's and backcar's location multiplied by thisseg's length. If there is no car in front of backcar on thisseg, we look to see if nextseg is negative one, meaning that the car will be leaving the simulation. If this is the case nothing need be done and we can move on. However, if this is not the case, we need to find out how far ahead the next car is. We do this by calling the aptly named procedure FindFrontCar.

FindFrontCar uses the same variables we have already defined in AdjustSpeeds. It first initializes the Gap variable with the length of thisseg that backcar has not yet traveled. Then it uses a Do loop to find frontcar. Let us look at the code:

gap = (1 - myCars(backcar).location) * mySegs(thisseg).length 'the remaining length of thisseg

Do 'the program will loop back to here

frontcar = mySegs(nextseg).carsIn(1) 'the last car on nextseg

If frontcar > 0 Then 'is there a car on nextseg?

 gap = gap + myCars(frontcar).location * mySegs(nextseg).length 'add the length of nextseg traveled by frontcar if there is.

 Exit Do

Else

 gap = gap + mySegs(nextseg).length 'add the entire length of nextseg if there is not.

End If

thisseg = nextseg 'if frontcar wasn't found we need to find the next nextseg

If mySegs(thisseg).nextSegR = 0 Then 'this part is just like FindNextSegment

 nextseg = mySegs(thisseg).nextSegL

ElseIf myCars(backcar).exit = mySegs(thisseg).nextSegR Then

 nextseg = mySegs(thisseg).nextSegR

Else

 nextseg = mySegs(thisseg).nextSegL

End If

If nextseg = 0 Then gap = allowedGap + gap 'just to make sure

Loop Until gap >= allowedGap Or thisseg >= 39 'keep looking until it doesn't matter

The second to last line was added because the cars were slowing prematurely and just before they got on the last segment before leaving the simulation. As they grew close to the beginning of the last segment, the gap would only be a small amount plus the length of the last segment. Thus, Gap would be smaller than allowedGap, and the car would slow.

FindFrontCar was created to find an accurate value for Gap. With this value, we can now adjust backcar's speed if need be. If Gap is less than the allowedGap, we need to slow backcar. So, to do this we subtract the product of accel, the acceleration in g's; 32.2 ft/sec², g; and the time that has passed since last we checked, deltaT from backcar's speed. Then, we set flag to true. Next, we check backcar's speed against the rest of our initial rules. If flag were false at this point and backcar were going slower than its desired speed, or deSpeed,

then it should speed up. In this case, we add the aforementioned product to backcar's speed. However in any case, if backcar should be going faster than its deSpeed, then, since it should not be going that much faster, we set its speed equal to its deSpeed. Next, if the car is approaching an intersection (i.e. the main traffic circle and one of its entrances), it needs to look left and see if there is a car coming. In the code it does this by first checking the value of lookLeftCar. If that is not zero, then it checks to see if lookLeftCar's location is high enough that it will be in the way of entering (or close enough to crash into). If so, then backcar will slow the same way it did before. The last thing AdjustSpeeds checks for is to see if backcar has slowed so much that it has started going backwards. In other words, has backcar's speed gone below zero? If it has then its speed is set to zero, as if it were waiting in a queue.

MoveCars –

MoveCars has been split into two procedures: SwitchSegments, which moves the cars along, calls adjust speeds and switches the segments the cars are on when necessary; and DrawCars, which draws and erases the cars as they move around the traffic circle.

SwitchSegments loops through all the “active” cars, the cars with their

active property set to true. It calls AdjustSpeeds for every one, and advances the cars along the segments. It does the latter using the equation: $myCars(i).location = myCars(i).location + myCars(i).speed * \Delta T / segLength$. The location increment is the distance the car would have traveled in one timestep. The car's location has a value between zero and one, no matter how long the segment is. It is merely a ratio of how far the car has gone to how far it could go. (see Figure 12) Therefore, when we multiply the car's speed with the elapsed time we get the distance the car has actually gone. To relate that to the cars location value we need to divide by the length of thisseg, segLength. (Thisseg and nextseg are used here as well.)

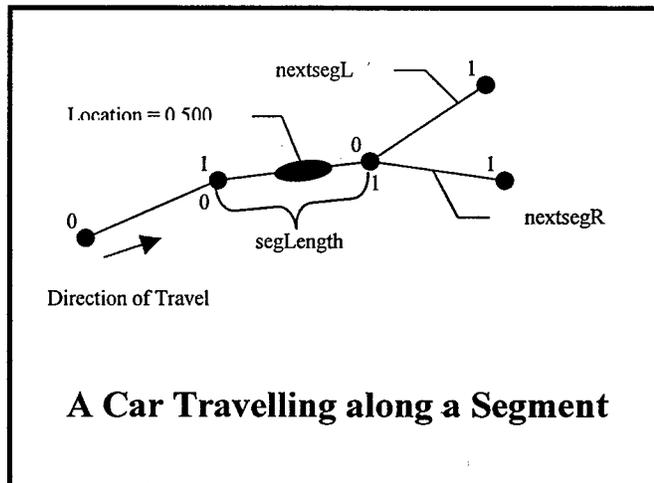


Figure 12 - Location

If the car's location becomes larger than one, then the car has moved onto the next segment, and it must be treated accordingly. First the carsIn array and the totCars property for thisseg must be updated. When that is done, we check to see if the car is leaving the simulation, in which case nextseg equals negative one. If it is, then make the car inactive, set

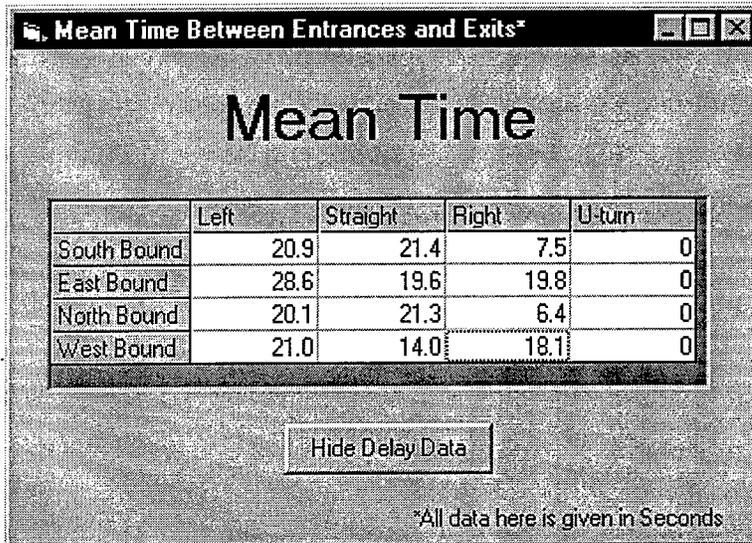


Figure 13 - frmDelayTime

its new property to true, and erase it so it does not leave a "blip" on the screen. We also call the procedure FindDelayTime at this point. This uses a nested case statement to find the proper cell in the MSFlexGrid control. Once found, it will update the average delay (see Figure 13). If the car is not leaving

the simulation, then we must update the carsIn array and the totCars property of nextseg. Then, we call FindNextSegment to find the new nextseg, and correct the car's location so it will fit its new segment.

The animation is the apparent motion of our drawings. In actuality the drawings are not moving. In the traffic circle program, the cars, are being drawn and erased and drawn again in a different place, creating the illusion of motion. Initially, the cars were just round dots. When we extended them into lines, we had trouble orientating them. We came up with the following rules for orienting the "cars." When on the round circle, to find the angle of the car, add 90 degrees to the angle of the car's location on the circle. When on a straight line, or a curved line for that matter, the angle of the car is the arctangent of the slope of the line at the point where the car is. The endpoints of the car could then be found using trigonometry.

DrawCars loops through all the "active cars", and will find their new x and y coordinates, measured at the center of the car. It does this by finding the endpoints of the segment the car is on and interpolating the car's coordinates using the coordinates of the endpoints and the car's location. It will then find the angle of the car, CarAngle, and the car's endpoints as described above. After setting the drawwidth to the width of the car, it will draw the car using Visual Basic's line command. If the car's new property is not set to

true then DrawCars will erase the old drawing of the car at its old position. If it were set to true then DrawCars would do nothing except set the new property to false, because there is no old drawing to erase. Lastly, DrawCars will store the coordinates of the car's endpoints so it can erase it during the next time step.

Conclusion:

The simulation/animation program presented here is based on a homework problem assigned to civil engineering students in an introductory computer class at University of Maine. The initial model development was partially funded by a grant from the New England UTC program through the project "Visual Aids in the Public Participation Process." The program is based on the principle of an autonomous agent—the motor vehicle—programmed to speed up or slow down in order to keep its desired velocity—which is randomly chosen at or below the maximum permissible speed—and keep a minimum gap to the preceding vehicle. The autonomous vehicle doesn't enter the circulating lane unless an acceptable gap is provided. Vehicles are assigned paths through the roundabout based on quarter-hour volumes for each possible path. Travel times—and delays—are calculated for each vehicle and then averaged over paths and time periods. Vehicle positions are continuously shown on the screen in real time or in scaled time. Build-ups of queues for varying degrees of saturation are directly illustrated on the screen. This animation can be a valuable help when comparing different design proposals.

The modifications made within this project have added a sense of realism to the simulation. This is exemplified with superimposing the movements on the geometric outline of the recently constructed roundabout in Gorham, Maine, rather than on a 'perfect' circle with four spokes in orthogonal positions as in the original homework. Also, a method of inputting traffic flow data was added and new output modules were developed. The output now includes average simulated travel time through the roundabout. However, the program has not yet been validated against its real life counterpart. Therefore, simulated travel times and delays may differ significantly from those actually experienced at the Gorham roundabout.

The simulation/animation model can be modified to any geometric design with respect to number of approach legs, approach angles, circle diameter, skews, etc., but only one-lane approaches and one-lane circulating roadways can be simulated at this time. And changing the geometric layout takes, at present, many hours. The model could be automated with respect to geometric set-up, potentially making it more attractive to clients. A strength of the present program is the simplicity of its execution once it is set up, and that simplicity should be guarded. Many existing animation programs—such as NetSim—lack user-friendliness.

The program can be improved with respect to numerous aspects. And before it is used in any commercial way, it needs to be calibrated and validated against real data. A thorough quality control with respect to reliability is needed prior to this, ensuring that the model actually follows all specified rules. Once the model is proven reliable and validated it will be a valuable tool in illustrating the effect of choice of junction type as well as how detailed geometric layout influences queue lengths and travel times.



Appendix A:

```
Dim numPoints As Integer
Dim saveX(1000) As Single
Dim saveY(1000) As Single
Dim numLines As Integer
Dim lineStart(500) As Integer
Dim lineEnd(500) As Integer
Dim lineThickness(500) As Integer
```

```
Private Sub Command1_Click()
Printer.EndDoc
End Sub
```

```
Private Sub Form_Load()
Picture1.Scale (-200, 200)-(200, -200)
numPoints = 0
numLines = 1
End Sub
```

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Picture1.Circle (X, Y), 3
numPoints = numPoints + 1
saveX(numPoints) = X
saveY(numPoints) = Y
lineStart(numLines) = numPoints
Else
Picture1.Line -(X, Y)
Picture1.Circle (X, Y), 3
numPoints = numPoints + 1
saveX(numPoints) = X
saveY(numPoints) = Y
lineEnd(numLines) = numPoints
numLines = numLines + 1
lineStart(numLines) = numPoints
End If
Debug.Print X, Y, numPoints
Printer.Print X, Y, numPoints
End Sub
```

```
Private Sub Picture1_Paint()
Dim i As Integer
For i = 1 To numLines - 1
Picture1.Circle (saveX(lineStart(i)), saveY(lineStart(i))), 3
Picture1.Line -(saveX(lineEnd(i)), saveY(lineEnd(i)))
Picture1.Circle (saveX(lineEnd(numLines - 1)), saveY(lineEnd(numLines - 1))), 3
Next i
End Sub
```



Appendix B:

```
frmMain
Option Explicit
Private Type Point
    x As Single
    y As Single
End Type
```

```
Private Type segment
    carsIn(20) As Integer 'list of cars in the segment
    totCars As Integer 'number of cars in segment
    endPt As Integer 'index of ending point
    length As Single 'length of segment in feet
    leftSegs(2) As Integer 'when entering the circle look for cars
    nextSegL As Integer 'index of next segment continuing in circle - "0" if none
    nextSegR As Integer 'index of next segment leaving circle - "0" if none
    startPt As Integer 'index of beginning point
    slope As Single
End Type
```

```
Private Type Car
    active As Boolean 'if a car is being used or not
    color As Long
    deSpeed As Single 'desired speed
    length As Single
    width As Single
    location As Single 'location in the segment
    new As Boolean
    nextseg As Integer 'next segment car is headed for
    "'-1" for an exit segment and "0" if not assigned yet
    segment As Integer 'number of the segment
    speed As Single 'actual speed
    exit As Integer 'assigned exit segment
    begintime As Single 'time enters
    entrance As Integer 'segment enters on
End Type
```

```
Private myPts(50) As Point 'array of points
Private mySegs(50) As segment 'array of segments
Private myCars(100) As Car 'array of cars
Private oldFront(100) As Point, oldBack(100) As Point
Private numCars As Integer 'number of cars
Private numSegs As Integer 'number of segments
Private deltaT As Single, yieldPt As Single
Private black As Long
Private carFront As Point, carBack As Point 'endpoints of the cars
Dim TimeSteps As Long 'number of timesteps
Dim Leftt(13 To 16) As Single 'these determine where cars exit
Dim Straight(13 To 16) As Single
Dim Right(13 To 16) As Single
Dim PutCarInNow(13 To 16) As Single 'the probability that a car will enter
Dim counter(4, 4) As Integer 'keeps track of the mean delay time
Dim frontcar As Integer, backcar As Integer 'these six variables are used to control speed
Dim thisseg As Integer, nextseg As Integer
Dim gap As Single, allowedGap As Single
```



```

Private Sub cmdClear_Click()
Dim i As Integer, j As Integer
pctPix.Cls
For i = 1 To 4
    frmVPH.grdVPH.Row = i
    For j = 1 To 5
        frmVPH.grdVPH.Col = j
        frmVPH.grdVPH.Text = CStr(0)
    Next j
Next i

For i = 1 To 4
    For j = 1 To 4
        counter(i, j) = 1
    Next j
Next i

For i = 1 To 4
    frmDelayTime.grdDelays.Row = i
    For j = 1 To 4
        frmDelayTime.grdDelays.Col = j
        frmDelayTime.grdDelays.Text = CStr(0)
    Next j
Next i

frmVPH.cmdVPH.Enabled = False
End Sub

Private Sub cmdDelay_Click()
frmDelayTime.Show
End Sub

Private Sub cmdInput_Click()
cmdSetUp.Enabled = True
frmInput.Show
End Sub

Private Sub cmdOutput_Click()
frmVPH.Show
End Sub

Private Sub cmdQuit_Click()
End
End Sub

Private Sub cmdRun_Click()
Timer1.Enabled = True
cmdStop.Enabled = True
cmdRun.Enabled = False
End Sub

Private Sub cmdSetUp_Click()
Dim x As Single, i As Integer, j As Integer, k As Integer
cmdRun.Enabled = True

```



```

cmdClear.Enabled = True
Randomize
black = RGB(0, 0, 0)
Erase mySegs
Erase myCars
yieldPt = CSng(txtyieldPt.Text)
pctPix.Scale (-200, 200)-(-200, -200) 'set user scale
pctPix.DrawMode = 10 'Not xor - same as the clock problem
pctPix.DrawStyle = 0 'Solid line
PointSetup
EndPointSetup
SegSetup
numCars = 100 'number of cars in array
deltaT = 0.5 'time step
PathCalculations
CarSetup
TimeSteps = 0 'Initialize this variable
For i = 1 To 4 'initialize these too
    For j = 1 To 4
        counter(i, j) = 1
    Next j
Next i
End Sub
Private Sub cmdStop_Click()
Timer1.Enabled = False
cmdRun.Enabled = True
cmdStop.Enabled = False
End Sub
Private Sub Form_Load()
cmdSetUp.Enabled = False
cmdRun.Enabled = False
cmdStop.Enabled = False
cmdClear.Enabled = False
End Sub

Private Sub Timer1_Timer()
TimeSteps = TimeSteps + 1 'count the timesteps
AddCars
MoveCars
txtElapsedTime.Text = deltaT * TimeSteps
CheckTime 'check to see if its time to stop
If chkSingleStep.Value = 1 Then
    Timer1.Enabled = False
    cmdRun.Enabled = True
End If
End Sub

Public Sub MoveCars()
SwitchSegments
DrawCars
End Sub

Public Sub AdjustSpeeds(i As Integer)
Dim j As Integer, k As Integer
Dim accel As Single, flag As Boolean

```



```

Dim lookLeftSeg As Integer, lookLeftCar As Integer
accel = CSng(txtAcceleration.Text)
backcar = i 'car we are in
thisseg = myCars(backcar).segment
nextseg = myCars(backcar).nextseg
j = 0
k = 0
Do Until k = i
    j = j + 1
    k = mySegs(thisseg).carsIn(j)
Loop
lookLeftSeg = mySegs(thisseg).leftSegs(1)
lookLeftCar = mySegs(lookLeftSeg).carsIn(mySegs(lookLeftSeg).totCars)
flag = False
allowedGap = CSng(txtGap.Text) * myCars(backcar).speed ^ 1.5 _
    + 2 * myCars(backcar).length
If mySegs(thisseg).carsIn(j + 1) <> 0 Then 'there is a car in front on this segment
    frontcar = mySegs(thisseg).carsIn(j + 1)
    gap = (myCars(frontcar).location - myCars(backcar).location) _
        * mySegs(myCars(thisseg).segment).length
ElseIf nextseg = -1 Then
    'Do nothing since car is on its way out
Else
    FindFrontCar (i) 'need to find out many segments ahead the next car is
End If
If gap < allowedGap Then
    myCars(backcar).speed = myCars(backcar).speed - accel * 32.2 * deltaT
    flag = True
End If
If (flag = False) And (myCars(backcar).speed < myCars(backcar).deSpeed) Then 'speed up
    myCars(backcar).speed = myCars(backcar).speed + accel * 32.2 * deltaT
End If
If (myCars(backcar).speed > myCars(backcar).deSpeed) Then 'slow down
    myCars(backcar).speed = myCars(backcar).deSpeed
End If
If lookLeftCar <> 0 Then
    If (myCars(lookLeftCar).location > ((mySegs(myCars(lookLeftCar).segment).length -
    21.3) / mySegs(myCars(lookLeftCar).segment).length)) And (myCars(backcar).location >
    yieldPt) Then
        myCars(backcar).speed = myCars(backcar).speed - accel * 32.2 * deltaT
    End If
End If
If myCars(backcar).speed < 0 Then myCars(backcar).speed = 0
End Sub

Public Sub FindNextSegment(i As Integer)
'If you are in this subroutine then you are trying to figure out
'where the car i is going next.
Dim segin As Integer
segin = myCars(i).segment 'segment car i is in
If mySegs(segin).nextSegL = 0 Then 'we are on the exit ramp
    myCars(i).nextseg = -1
ElseIf mySegs(segin).nextSegR = 0 Then
    myCars(i).nextseg = mySegs(segin).nextSegL
ElseIf myCars(i).exit = mySegs(segin).nextSegR Then

```



```

        myCars(i).nextseg = mySegs(segin).nextSegR
    Else
        myCars(i).nextseg = mySegs(segin).nextSegL
    End If
End Sub
Public Sub AddCars()
Dim i As Integer, j As Integer
j = 12
For i = 1 To numCars
    If myCars(i).active = False Then 'we can add it somewhere
        j = j + 1
        Call CheckProbability(i, j) 'to see if its time to enter a car
        If j = 16 Then Exit For
    End If
Next i
End Sub

```

```

Public Sub FindExit(i As Integer)
Dim segin As Integer
Dim x As Single
Dim direction As String 'where going?
segin = myCars(i).segment
x = Rnd
If x < Right(segin) Then
    direction = "right"
ElseIf x < Straight(segin) Then
    direction = "straight"
ElseIf x < Leftt(segin) Then
    direction = "left"
Else
    direction = "back"
End If
Select Case direction
    Case "left"
        Select Case segin
            Case 13
                myCars(i).exit = 20
            Case 14
                myCars(i).exit = 17
            Case 15
                myCars(i).exit = 18
            Case 16
                myCars(i).exit = 19
        End Select
    Case "straight"
        Select Case segin
            Case 13
                myCars(i).exit = 19
            Case 14
                myCars(i).exit = 20
            Case 15
                myCars(i).exit = 17
            Case 16
                myCars(i).exit = 18
        End Select

```



```

Case "right"
  Select Case segin
    Case 13
      myCars(i).exit = 18
    Case 14
      myCars(i).exit = 22
    Case 15
      myCars(i).exit = 20
    Case 16
      myCars(i).exit = 21
  End Select
Case "back"
  Select Case segin
    Case 13
      myCars(i).exit = 17
    Case 14
      myCars(i).exit = 18
    Case 15
      myCars(i).exit = 19
    Case 16
      myCars(i).exit = 20
  End Select
End Select
End Sub

Public Sub UpdateOutput(i As Integer)
Dim segin As Integer, cExit As Integer, j As Integer, k As Integer, esum As Integer, exsum
As Integer
If (deltaT * TimeSteps) >= (frmInput.txtETime.Text * 60) Then
  segin = myCars(i).segment
  cExit = myCars(i).exit
  Select Case segin
    Case 14
      frmVPH.grdVPH.Row = 1
      Select Case cExit
        Case 18
          frmVPH.grdVPH.Col = 4
          frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 22
          frmVPH.grdVPH.Col = 3
          frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 20
          frmVPH.grdVPH.Col = 2
          frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 17
          frmVPH.grdVPH.Col = 1
          frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
      End Select
    Case 15
      frmVPH.grdVPH.Row = 2
      Select Case cExit
        Case 18
          frmVPH.grdVPH.Col = 1
          frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 19

```



```

        frmVPH.grdVPH.Col = 4
        frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
    Case 20
        frmVPH.grdVPH.Col = 3
        frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
    Case 17
        frmVPH.grdVPH.Col = 2
        frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
End Select
Case 16
    frmVPH.grdVPH.Row = 3
    Select Case cExit
        Case 18
            frmVPH.grdVPH.Col = 2
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 19
            frmVPH.grdVPH.Col = 1
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 20
            frmVPH.grdVPH.Col = 4
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 21
            frmVPH.grdVPH.Col = 3
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
    End Select
Case 13
    frmVPH.grdVPH.Row = 4
    Select Case cExit
        Case 18
            frmVPH.grdVPH.Col = 3
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 19
            frmVPH.grdVPH.Col = 2
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 20
            frmVPH.grdVPH.Col = 1
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
        Case 17
            frmVPH.grdVPH.Col = 4
            frmVPH.grdVPH.Text = CStr(CInt(frmVPH.grdVPH.Text) + 1)
    End Select
End Select
For k = 1 To 4
    esum = 0
    frmVPH.grdVPH.Row = k
    For j = 1 To 4
        frmVPH.grdVPH.Col = j
        esum = esum + CInt(frmVPH.grdVPH.Text)
    Next j
    frmVPH.grdVPH.Col = 5
    frmVPH.grdVPH.Text = esum
Next k
End If
End Sub

```



```

Public Sub PathCalculations()
Dim i As Integer
Dim sum(13 To 16) As Single
'traffic from segment 14
sum(14) = 0
frmInput.grdInput.Row = 1
For i = 1 To 4
    frmInput.grdInput.Col = i
    sum(14) = sum(14) + CSng(frmInput.grdInput.Text)
Next i
frmInput.grdInput.Col = 3
Right(14) = CSng(frmInput.grdInput.Text) / sum(14)
frmInput.grdInput.Col = 2
Straight(14) = Right(14) + CSng(frmInput.grdInput.Text) / sum(14)
frmInput.grdInput.Col = 1
Leftt(14) = Straight(14) + CSng(frmInput.grdInput.Text) / sum(14)
'traffic from segment 15
sum(15) = 0
frmInput.grdInput.Row = 2
For i = 1 To 4
    frmInput.grdInput.Col = i
    sum(15) = sum(15) + CSng(frmInput.grdInput.Text)
Next i
frmInput.grdInput.Col = 3
Right(15) = CSng(frmInput.grdInput.Text) / sum(15)
frmInput.grdInput.Col = 2
Straight(15) = Right(15) + CSng(frmInput.grdInput.Text) / sum(15)
frmInput.grdInput.Col = 1
Leftt(15) = Straight(15) + CSng(frmInput.grdInput.Text) / sum(15)
'traffic from segment 16
sum(16) = 0
frmInput.grdInput.Row = 3
For i = 1 To 4
    frmInput.grdInput.Col = i
    sum(16) = sum(16) + CSng(frmInput.grdInput.Text)
Next i
frmInput.grdInput.Col = 3
Right(16) = CSng(frmInput.grdInput.Text) / sum(16)
frmInput.grdInput.Col = 2
Straight(16) = Right(16) + CSng(frmInput.grdInput.Text) / sum(16)
frmInput.grdInput.Col = 1
Leftt(16) = Straight(16) + CSng(frmInput.grdInput.Text) / sum(16)
'traffic from segment 13
sum(13) = 0
frmInput.grdInput.Row = 4
For i = 1 To 4
    frmInput.grdInput.Col = i
    sum(13) = sum(13) + CSng(frmInput.grdInput.Text)
Next i
frmInput.grdInput.Col = 3
Right(13) = CSng(frmInput.grdInput.Text) / sum(13)
frmInput.grdInput.Col = 2
Straight(13) = Right(13) + CSng(frmInput.grdInput.Text) / sum(13)
frmInput.grdInput.Col = 1
Leftt(13) = Straight(13) + CSng(frmInput.grdInput.Text) / sum(13)

```



```

'entrance data
'the following code is trying to make the data more accurately reflect the input data
For i = 13 To 16
    PutCarInNow(i) = (sum(i) / 3600) * deltaT
    'seconds per timestep * cars per hour / seconds per hour = cars per timestep
Next i
End Sub
Public Sub FindDelayTime(i As Integer)
Dim ave As Single, t As Single, sum As Single
If (deltaT * TimeSteps) >= (60 * CSng(frmInput.txtETime.Text)) Then
    t = (TimeSteps * deltaT) - myCars(i).begintime
    Select Case myCars(i).entrance
        Case 14
            frmDelayTime.grdDelays.Row = 1
            Select Case myCars(i).exit
                Case 18
                    frmDelayTime.grdDelays.Col = 4
                    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
                    sum = sum + t
                    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
                    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
                    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
                Case 22
                    frmDelayTime.grdDelays.Col = 3
                    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
                    sum = sum + t
                    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
                    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
                    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
                Case 20
                    frmDelayTime.grdDelays.Col = 2
                    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
                    sum = sum + t
                    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
                    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
                    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
                Case 17
                    frmDelayTime.grdDelays.Col = 1
                    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
                    sum = sum + t
                    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
                    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
                    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
            End Select
        End Select
    End If
End Sub

```



```

End Select
Case 15
frmDelayTime.grdDelays.Row = 2
Select Case myCars(i).exit
Case 18
frmDelayTime.grdDelays.Col = 1
sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
sum = sum + t
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
frmDelayTime.grdDelays.Text = Format(ave, "##.0")
Case 19
frmDelayTime.grdDelays.Col = 4
sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
sum = sum + t
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
frmDelayTime.grdDelays.Text = Format(ave, "##.0")
Case 20
frmDelayTime.grdDelays.Col = 3
sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
sum = sum + t
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
frmDelayTime.grdDelays.Text = Format(ave, "##.0")
Case 17
frmDelayTime.grdDelays.Col = 2
sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
sum = sum + t
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
frmDelayTime.grdDelays.Text = Format(ave, "##.0")
End Select
Case 16
frmDelayTime.grdDelays.Row = 3
Select Case myCars(i).exit
Case 18
frmDelayTime.grdDelays.Col = 2
sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
sum = sum + t
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1

```



```

    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    Case 19
    frmDelayTime.grdDelays.Col = 1
    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
    sum = sum + t
    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    Case 20
    frmDelayTime.grdDelays.Col = 4
    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
    sum = sum + t
    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    Case 21
    frmDelayTime.grdDelays.Col = 3
    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
    sum = sum + t
    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    End Select
    Case 13
    frmDelayTime.grdDelays.Row = 4
    Select Case myCars(i).exit
    Case 18
    frmDelayTime.grdDelays.Col = 3
    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
    sum = sum + t
    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
    ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
    frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    Case 19
    frmDelayTime.grdDelays.Col = 2
    sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
    sum = sum + t
    counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1

```



```

        ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
        frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    Case 20
        frmDelayTime.grdDelays.Col = 1
        sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
        sum = sum + t
        counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
        ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
        frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    Case 17
        frmDelayTime.grdDelays.Col = 4
        sum = CSng(frmDelayTime.grdDelays.Text) *
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col)
        sum = sum + t
        counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) =
counter(frmDelayTime.grdDelays.Row, frmDelayTime.grdDelays.Col) + 1
        ave = sum / counter(frmDelayTime.grdDelays.Row,
frmDelayTime.grdDelays.Col)
        frmDelayTime.grdDelays.Text = Format(ave, "##.0")
    End Select
End Select
End If
End Sub
Public Sub CheckTime()
If (deltaT * TimeSteps) >= (frmInput.txtRTime.Text * 60) Then
    Timer1.Enabled = False
    cmdStop.Enabled = False
    cmdRun.Enabled = True
    frmVPH.cmdVPH.Enabled = True
End If
End Sub

Public Sub SegSetup()
Dim i As Integer, j As Integer, numpoints As Integer
numSegs = 42
numpoints = 40
For i = 1 To numSegs
    If cbxShowSegs.Value = 1 Then
        pctPix.Line (myPts(mySegs(i).startPt).x, myPts(mySegs(i).startPt).y)-
(myPts(mySegs(i).endPt).x, myPts(mySegs(i).endPt).y), black 'draw lines
    End If
    With mySegs(i)
        .length = ((myPts(mySegs(i).startPt).x - myPts(mySegs(i).endPt).x) ^ 2 +
(myPts(mySegs(i).startPt).y - myPts(mySegs(i).endPt).y) ^ 2) ^ 0.5
        .slope = (myPts(mySegs(i).startPt).y - myPts(mySegs(i).endPt).y) /
(myPts(mySegs(i).startPt).x - myPts(mySegs(i).endPt).x)
    End With
Next i
If cbxShowSegs.Value = 1 Then
    For i = 1 To numpoints
        pctPix.Circle (myPts(i).x, myPts(i).y), 3
    
```



```

Next i
End If
For i = 1 To 12
  j = i + 1: If i = 12 Then j = 1
  mySegs(i).nextSegL = j
Next i
mySegs(2).nextSegR = 18
mySegs(4).nextSegR = 19
mySegs(8).nextSegR = 20
mySegs(10).nextSegR = 17
With mySegs(13)
  .nextSegL = 23
  .nextSegR = 0
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(14)
  .nextSegL = 25
  .nextSegR = 22
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(15)
  .nextSegL = 27
  .nextSegR = 0
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(16)
  .nextSegL = 29
  .nextSegR = 21
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(17)
  .nextSegL = 37
  .nextSegR = 0
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(18)
  .nextSegL = 38
  .nextSegR = 0
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(19)
  .nextSegL = 41
  .nextSegR = 0
  .leftSegs(1) = 0
  .leftSegs(2) = 0
End With
With mySegs(20)
  .nextSegL = 42
  .nextSegR = 0

```



```

    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(21)
    .nextSegL = 32
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(22)
    .nextSegL = 35
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(23)
    .nextSegL = 24
    .nextSegR = 0
    .leftSegs(1) = 12
    .leftSegs(2) = 11
End With
With mySegs(24)
    .nextSegL = 1
    .nextSegR = 0
    .leftSegs(1) = 12
    .leftSegs(2) = 11
End With
With mySegs(25)
    .nextSegL = 26
    .nextSegR = 0
    .leftSegs(1) = 4
    .leftSegs(2) = 3
End With
With mySegs(26)
    .nextSegL = 5
    .nextSegR = 0
    .leftSegs(1) = 4
    .leftSegs(2) = 3
End With
With mySegs(27)
    .nextSegL = 28
    .nextSegR = 0
    .leftSegs(1) = 6
    .leftSegs(2) = 5
End With
With mySegs(28)
    .nextSegL = 7
    .nextSegR = 0
    .leftSegs(1) = 6
    .leftSegs(2) = 5
End With
With mySegs(29)
    .nextSegL = 30
    .nextSegR = 0
    .leftSegs(1) = 0

```



```

    .leftSegs(2) = 0
End With
With mySegs(30)
    .nextSegL = 31
    .nextSegR = 0
    .leftSegs(1) = 10
    .leftSegs(2) = 9
End With
With mySegs(31)
    .nextSegL = 11
    .nextSegR = 0
    .leftSegs(1) = 10
    .leftSegs(2) = 9
End With
With mySegs(32)
    .nextSegL = 33
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(33)
    .nextSegL = 34
    .nextSegR = 0
    .leftSegs(1) = 37
    .leftSegs(2) = 17
End With
With mySegs(34)
    .nextSegL = 39
    .nextSegR = 0
    .leftSegs(1) = 37
    .leftSegs(2) = 17
End With
With mySegs(35)
    .nextSegL = 36
    .nextSegR = 0
    .leftSegs(1) = 19
    .leftSegs(2) = 4
End With
With mySegs(36)
    .nextSegL = 41
    .nextSegR = 0
    .leftSegs(1) = 19
    .leftSegs(2) = 4
End With
With mySegs(37)
    .nextSegL = 39
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(38)
    .nextSegL = 40
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0

```



```

End With
With mySegs(39)
    .nextSegL = 0
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(40)
    .nextSegL = 0
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(41)
    .nextSegL = 0
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
With mySegs(42)
    .nextSegL = 0
    .nextSegR = 0
    .leftSegs(1) = 0
    .leftSegs(2) = 0
End With
End Sub
Public Sub CarSetup()
Dim i As Integer
For i = 1 To 100
    With myCars(i)
        .active = False
        .length = 10# + 2 * Rnd
        .color = RGB(255 * Rnd, 255 * Rnd, 255 * Rnd)
        .speed = 25 + 20 * Rnd
        .deSpeed = 15 + 20 * Rnd
        .new = True
        .width = 5 + Rnd
    End With
Next i
End Sub

Public Sub EnterNow(i As Integer)
Dim y As Single, j As Integer, segin As Integer, lastCar As Integer
segin = myCars(i).segment
lastCar = mySegs(segin).carsIn(1)
'Find location of last car on segment
If lastCar = 0 Or myCars(lastCar).location >= (30.7 / mySegs(segin).length) Then
    myCars(i).active = True
    myCars(i).location = 0.1
    FindExit (i)
    FindNextSegment (i)
    For j = 19 To 2 Step -1
        mySegs(segin).carsIn(j) = mySegs(segin).carsIn(j - 1)
    Next j
    mySegs(segin).carsIn(1) = i

```



```

    mySegs(segIn).totCars = mySegs(segIn).totCars + 1
    UpdateOutput (i)
    AdjustSpeeds (i)
    AdjustSpeeds (i)
    myCars(i).begintime = deltaT * TimeSteps
    myCars(i).entrance = segIn
End If
End Sub

Public Sub DrawCars()
Dim begx As Single, begy As Single, endx As Single, endy As Single
Dim i As Integer, newx As Single, newy As Single
Dim seg As Integer, CarAngle As Single
For i = 1 To numCars
    If myCars(i).active = True Then
        seg = myCars(i).segment
        begx = myPts(mySegs(seg).startPt).x
        begy = myPts(mySegs(seg).startPt).y
        endx = myPts(mySegs(seg).endPt).x
        endy = myPts(mySegs(seg).endPt).y
        newx = begx + (endx - begx) * myCars(i).location
        newy = begy + (endy - begy) * myCars(i).location
        CarAngle = Atn(mySegs(seg).slope)
        carFront.x = newx + (myCars(i).length / 2) * Cos(CarAngle)
        carFront.y = newy + (myCars(i).length / 2) * Sin(CarAngle)
        carBack.x = newx - (myCars(i).length / 2) * Cos(CarAngle)
        carBack.y = newy - (myCars(i).length / 2) * Sin(CarAngle)
        pctPix.DrawWidth = myCars(i).width
        If myCars(i).new = False Then
            pctPix.Line (oldFront(i).x, oldFront(i).y)-(oldBack(i).x, oldBack(i).y),
myCars(i).color 'Erase old cars
        Else
            myCars(i).new = False
        End If
        pctPix.Line (carFront.x, carFront.y)-(carBack.x, carBack.y), myCars(i).color 'Draw
new ones
        oldFront(i).x = carFront.x 'remember location to erase on next time step
        oldFront(i).y = carFront.y
        oldBack(i).x = carBack.x
        oldBack(i).y = carBack.y
    End If
Next i
End Sub

Public Sub SwitchSegments()
Dim i As Integer, j As Integer, segLength As Single
Dim thisSegg As Integer, nextSegg As Integer
For i = 1 To numCars
    If myCars(i).active = True Then
        'move car -- distance = rate * time
        AdjustSpeeds (i)
        thisSegg = myCars(i).segment
        nextSegg = myCars(i).nextseg
        segLength = mySegs(thisSegg).length
        myCars(i).location = myCars(i).location + myCars(i).speed * deltaT / segLength
        If myCars(i).location > 1 Then ' exit or move car into the next segment

```



```

mySegs(thisSegg).carsIn(mySegs(thisSegg).totCars) = 0
mySegs(thisSegg).totCars = mySegs(thisSegg).totCars - 1
If nextSegg <> -1 Then
    mySegs(nextSegg).totCars = mySegs(nextSegg).totCars + 1
    For j = 19 To 2 Step -1
        mySegs(nextSegg).carsIn(j) = mySegs(nextSegg).carsIn(j - 1)
    Next j 'make space in carsIn of the next segment for the car
    mySegs(nextSegg).carsIn(1) = i 'shift current car into nextSegg
    myCars(i).segment = myCars(i).nextseg
    FindNextSegment (i) 'i is the index of the current car
    'set location in next segment
    myCars(i).location = (myCars(i).location - 1#) * mySegs(thisSegg).length /
mySegs(nextSegg).length
Else
    myCars(i).active = False 'turn car off.
    myCars(i).new = True
    pctPix.FillColor = myCars(i).color 'set car color
    pctPix.DrawWidth = myCars(i).width
    pctPix.Line (oldFront(i).x, oldFront(i).y)-(oldBack(i).x, oldBack(i).y),
myCars(i).color 'erase for last time
    FindDelayTime (i)
End If
End If
Next i
End Sub
Public Sub FindFrontCar(i As Integer)
backcar = i
thisseg = myCars(backcar).segment
nextseg = myCars(backcar).nextseg
gap = (1 - myCars(backcar).location) * mySegs(thisseg).length
Do
    frontcar = mySegs(nextseg).carsIn(1)
    If frontcar <> 0 Then
        gap = gap + myCars(frontcar).location * mySegs(nextseg).length
        Exit Do
    Else
        gap = gap + mySegs(nextseg).length
    End If
    thisseg = nextseg
    If mySegs(thisseg).nextSegR = 0 Then
        nextseg = mySegs(thisseg).nextSegL
    ElseIf myCars(backcar).exit = mySegs(thisseg).nextSegR Then
        nextseg = mySegs(thisseg).nextSegR
    Else
        nextseg = mySegs(thisseg).nextSegL
    End If
    If nextseg = 0 Then gap = allowedGap + gap
Loop Until gap >= allowedGap Or thisseg >= 39
End Sub
Public Sub PointSetup()
Dim i As Integer
i = 0
Open "c:\stuff\gorham circle\point.txt" For Input As #1
Do While Not EOF(1)

```



```
    i = i + 1
    Input #1, myPts(i).x, myPts(i).y
Loop
Close #1
End Sub
```

```
Public Sub EndPointSetup()
Dim i As Integer, j As Integer
For i = 1 To 12
    j = i + 1: If i = 12 Then j = 1
    mySegs(i).startPt = i
    mySegs(i).endPt = j
Next i
With mySegs(13)
    .startPt = 25
    .endPt = 24
End With
With mySegs(14)
    .startPt = 14
    .endPt = 13
End With
With mySegs(15)
    .startPt = 23
    .endPt = 22
End With
With mySegs(16)
    .startPt = 34
    .endPt = 33
End With
With mySegs(17)
    .startPt = 11
    .endPt = 35
End With
With mySegs(18)
    .startPt = 3
    .endPt = 28
End With
With mySegs(19)
    .startPt = 5
    .endPt = 17
End With
With mySegs(20)
    .startPt = 9
    .endPt = 26
End With
With mySegs(21)
    .startPt = 33
    .endPt = 38
End With
With mySegs(22)
    .startPt = 13
    .endPt = 15
End With
With mySegs(23)
    .startPt = 24
```



.endPt = 20
End With
With mySegs(24)
.startPt = 20
.endPt = 1
End With
With mySegs(25)
.startPt = 13
.endPt = 19
End With
With mySegs(26)
.startPt = 19
.endPt = 5
End With
With mySegs(27)
.startPt = 22
.endPt = 21
End With
With mySegs(28)
.startPt = 21
.endPt = 7
End With
With mySegs(29)
.startPt = 33
.endPt = 32
End With
With mySegs(30)
.startPt = 32
.endPt = 31
End With
With mySegs(31)
.startPt = 31
.endPt = 11
End With
With mySegs(32)
.startPt = 38
.endPt = 39
End With
With mySegs(33)
.startPt = 39
.endPt = 40
End With
With mySegs(34)
.startPt = 40
.endPt = 36
End With
With mySegs(35)
.startPt = 15
.endPt = 16
End With
With mySegs(36)
.startPt = 16
.endPt = 17
End With
With mySegs(37)



```

        .startPt = 35
        .endPt = 36
    End With
    With mySegs(38)
        .startPt = 28
        .endPt = 29
    End With
    With mySegs(39)
        .startPt = 36
        .endPt = 37
    End With
    With mySegs(40)
        .startPt = 29
        .endPt = 30
    End With
    With mySegs(41)
        .startPt = 17
        .endPt = 18
    End With
    With mySegs(42)
        .startPt = 26
        .endPt = 27
    End With
End Sub

```

```

Public Sub CheckProbability(i As Integer, j As Integer)
    Dim x As Single
    x = Rnd 'randomly decide when to enter
    If x <= PutCarInNow(j) Then 'it's time to put a car in
        myCars(i).segment = j
        EnterNow (i)
    End If
End Sub
Private Sub cmdOK_Click()
    frmInput.Hide
End Sub
Private Sub cmdUpdate_Click()
    grdInput.Row = CInt(txtRow.Text)
    grdInput.Col = CInt(txtCol.Text)
    grdInput.Text = txtData.Text
End Sub

```

```

frmInput
Private Sub Form_Load()
    Dim i As Integer, x As Integer
    grdInput.Row = 0
    grdInput.Col = 1
    grdInput.Text = "Left"
    grdInput.Col = 2
    grdInput.Text = "Straight"
    grdInput.Col = 3
    grdInput.Text = "Right"
    grdInput.Col = 4
    grdInput.Text = "U-turn"
    grdInput.Col = 0

```



```

grdInput.ColWidth(0) = grdInput.ColWidth(0) * 1.1
grdInput.Row = 1
grdInput.Text = "South Bound" 'segment 14
grdInput.Row = 2
grdInput.Text = "East Bound" 'segment 15
grdInput.Row = 3
grdInput.Text = "North Bound" 'segment 16
grdInput.Row = 4
grdInput.Text = "West Bound" 'segment 13
For i = 1 To 4
    grdInput.Row = i
    For x = 1 To 4
        grdInput.Col = x
        If x = 4 Then
            grdInput.Text = 0
        Else
            grdInput.Text = 60
        End If
    Next x
Next i
End Sub

```

```

FrmVPH
Private Sub cmdhide_Click()
frmVPH.Hide
End Sub

```

```

Private Sub cmdVPH_Click()
Dim i As Integer, j As Integer
Label10.Caption = "Vehicles per Hour"
For i = 1 To 4
    grdVPH.Row = i
    For j = 1 To 5
        grdVPH.Col = j
        If i = 5 And j = 5 Then Exit For
        grdVPH.Text = Format((CSng(grdVPH.Text) * 60) / (CSng(frmInput.txtRTime.Text)
- CSng(frmInput.txtETime.Text)), "###.0")
    Next j
Next i
cmdVPH.Enabled = False
End Sub

```

```

Private Sub Form_Load()
Dim i As Integer, x As Integer
grdVPH.Row = 0
grdVPH.Col = 1
grdVPH.Text = "Left"
grdVPH.Col = 2
grdVPH.Text = "Straight"
grdVPH.Col = 3
grdVPH.Text = "Right"
grdVPH.Col = 4
grdVPH.Text = "U-turn"
grdVPH.Col = grdVPH.Col + 1
grdVPH.Text = "Entr. Sum"

```



```

grdVPH.Col = 0
grdVPH.ColWidth(0) = grdVPH.ColWidth(0) * 1.1
grdVPH.Row = 1
grdVPH.Text = "South Bound" 'segment 14
grdVPH.Row = 2
grdVPH.Text = "East Bound" 'segment 15
grdVPH.Row = 3
grdVPH.Text = "North Bound" 'segment 16
grdVPH.Row = 4
grdVPH.Text = "West Bound" 'segment 13
For i = 1 To 4
    frmVPH.grdVPH.Row = i
    For j = 1 To 5
        frmVPH.grdVPH.Col = j
        frmVPH.grdVPH.Text = CStr(0)
    Next j
Next i
End Sub

```

```

frmDelayTime
Private Sub cmdhide_Click()
frmDelayTime.Hide
End Sub
Private Sub Form_Load()
Dim i As Integer, j As Integer
grdDelays.Row = 0
grdDelays.Col = 1
grdDelays.Text = "Left"
grdDelays.Col = 2
grdDelays.Text = "Straight"
grdDelays.Col = 3
grdDelays.Text = "Right"
grdDelays.Col = 4
grdDelays.Text = "U-turn"
grdDelays.Col = 0
grdDelays.ColWidth(0) = grdDelays.ColWidth(0) * 1.1
grdDelays.Row = 1
grdDelays.Text = "South Bound" 'segment 14
grdDelays.Row = 2
grdDelays.Text = "East Bound" 'segment 15
grdDelays.Row = 3
grdDelays.Text = "North Bound" 'segment 16
grdDelays.Row = 4
grdDelays.Text = "West Bound" 'segment 13
For i = 1 To 4
    grdDelays.Row = i
    For j = 1 To 4
        grdDelays.Col = j
        grdDelays.Text = CStr(0)
    Next j
Next i
End Sub

```

