A Roundabout Animation

Bryan Pearce
Per Gärder
Travis Folsom
University of Maine

The New England University Transportation Center
A Roundabout Animation

Bryan Pearce
Per Gårder
Travis Folsom
University of Maine

Final Report

Year 10 (97/98)
Traffic Circle Animation
Project No. MEE10-2

Year 11 (98/99)
Data Collection for Traffic Circle Animation
Project No. ME11E-2
A Roundabout Animation

This report describes work done on a roundabout animation program during 1998 and 1999. The roundabout animation program began as an undergraduate class project and was presented in February 1998 in the New England University Transportation Center report “Animation of Traffic through Roundabouts.” This new report presents the work funded by the Year 10 (97/98) grant Traffic Circle Animation (Project No. MEE10-2) and Year 11 (98/99) grant Data Collection for Traffic Circle Animation (Project No. ME11E-2). Undergraduate students were involved in all of these modifications.

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.
Abstract

This report describes work done on a roundabout animation program during 1998 and 1999. The roundabout animation program began as an undergraduate class project and was presented in February 1998 in the New England University Transportation Center report “Animation of Traffic through Roundabouts.” This new report presents the work funded by the Year 10 (97/98) grant Traffic Circle Animation (Project No. MEE10-2) and Year 11 (98/99) grant Data Collection for Traffic Circle Animation (Project No. ME11E-2). Undergraduate students were involved in all of these modifications.

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.
Acknowledgement

We want to thank everybody involved in this project. This includes numerous undergraduate students who during class time suggested methods and procedures for how traffic through roundabouts can be simulated and animated. We particularly want to mention three undergraduate students, Joshua Coombs, John Larson, and Travis Folsom for their field observation, programming and documentation work.

We also want to acknowledge the New England University Transportation Center for funding this work. The original funding source is U.S. Department of Transportation.
A Roundabout Animation

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.1 2 3 4

This paper is an update of an ongoing project of the University of Maine Roundabout Model (UMRoM). This project started in 1996 as a homework assignment for CIE 115, Computers In Civil Engineering. It was then embraced by one of the students who helped it a few more steps in evolution. The original system has been retained with changes and additions to increase the realism of the modeled traffic.

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 to allow the user to edit the traffic flow data. The program simulates the traffic flow by calculating the vehicle movement in discrete time steps. With each repetition, or timestep, a certain amount of “model time” passes. A variable, deltaT, holds the value of this time step, for example 0.5 seconds. The simulated time and a data recording time can be set by the user. The traffic simulator will show the vehicles moving along their appropriate paths. The user can also view the “traffic counts” generated by the program. These counts can be converted to vehicles per hour when the simulation is complete), as well as its average time that the cars are in the simulation. Other options allow the user to view all paths, and to have the model 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 Roundabout5. The current version of UMRoM now includes a feature to easily allow the setup for any roundabout with six or less entrance approaches and six or less ‘exits’ in any order or combination.

The simulation employs the concept of autonomous agents. 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. Each vehicle then follows the traffic ‘laws,’ according to those characteristics.

House Rules

Prior to writing the program, a set of rules were defined that would describe how vehicles behave. First, how could we keep the vehicles 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.

Merging with traffic at a yield was studied and a system was advanced. We used gap and lag for the model. Lag is called into effect when a car has open road ahead of it and has a car that it might have to yield to. The time it will take for the car on the left to reach the intersection is lag. Gap is a time gap between two cars in the stream of traffic that we are yielding to. On average lag is smaller than gap. When the actual lag or gap is larger than desired the car at the yield will merge.

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 a standard Cartesian, ranging from -200 * scale feet to 200 * scale feet in both directions (see Figure 1). The scale is defined in the sister program used to prepare the simulation for any given roundabout.

Setup

In the previous version of UMROm the roundabout and its segments were “hard wired” into the program. That is each value was measured, by hand, and then typed into the code. This process was laborious as well as extremely time consuming. To make the model more useful a sister program was created which allows for rapid setup of any new roundabout (See Figure 2). The program allows the user to load any picture and then click on the picture to place the points that define the segments that the cars follow.
The roundabout setup program allows the user to accomplish many things from telling the program where the roads are to naming the roads. The user may load any standard picture of a roundabout into the program. From here, they are able to pick points over the roads in the picture. To make editing the setup smoother the user is also able to delete or move points. Points are defined by clicking with the mouse. A segment joins two points and has a direction. To make a segment, the user clicks on the first point of the segment and then the second. The first point of a segment would be defined by the simple idea that that point would be the first point in the segment that the car would pass over.

As with points, the program allows for the removal of segments.

Setups can be saved either in progress or when completed. All setups for the roundabout may be loaded thorough simple save and load dialogue boxes, similar to the ones used to open a file in Microsoft Word. More options are available to the user, such as being able to control the appearance of points, segments, and their labels. One may also make segments and/or points invisible.

Once the segments are placed, the program finds all the possible paths for the traffic flow. The program first finds segments that have a beginning point that does not coincide with the end-point of a different segment. Therefore, that segment must be an entrance into the system. From here, the routine takes that segment and finds another segment that has a starting point that is the same as the first end point. The algorithm will continue to cycle this way from segment to segment and keep track of each path found. When there are two segments branching from an end-point, a routine is used to find which segment is on the right. The path is copied and we continue on to the right segment. The right segment is the segment that then would be the exit of the roundabout (assuming right hand drive).
The process for finding the right segment is done using vector cross products. First, we define the two segments as vectors. Then if we cross the right with the left the resulting vector will be positive. Along with checking for two segments that start from one segment, two segment with a single endpoint (a yield) are also located. The program will again use cross products to find which segment is on the right. This is done to find which of the two segments have to yield.

Once a path is completed, our algorithm will find the next path from the same origin. This is where the copy of the last path is used. Starting from the segment on the left (we took the right last time), we continue looking for “next” segments as before. The program finds all paths until a 360° circuit of the roundabout has been completed. It will discard the redundant path just found move on to the next entrance segment. The cycle continues and stops once all the starting segments have been used.

Now that all the paths have been found the user can then review all of the paths and remove all of the invalid paths. An example of an invalid path is given in Figure 3. A driver must follow the (red) dotted arrow and not the path shown. The user can decide to remove this incorrect path. Now the operator of the program must click on the “set path names” button to name all of the entrances and exits. At this point the user is almost done, all that is left is for the user to set the scale of the drawing by clicking on two points of the map and stating how far apart they are. Once the process of setting up the roundabout is done the user clicks the “create roundabout files” button and the information is saved.

The Files

There are four files created by the setup program. They all have the name given by the user but they have different extensions. They are pth, pts, seg, and set. These abbreviations stand for path, points, segments, and segment setup respectively. The path file holds information of how many entrance and exit segments are in the simulation. In addition, this file contains the names of the entrance segments and exit segments. These will be the road and highway names appearing on a real simulation. The last information within this file is combinations of entrances and exits for each path. This data is used to retain the list of segments for each path.

The point file holds the name of the picture that is used for the map and the location of the point used in the simulation. A segment is defined by two “points.” These locations are in the Cartesian system as described earlier. The segment file is a list of the ordered numbers of the points and the associated segments. The scale is saved in the segment setup file. Along with scale, the .seg files holds additional information for every segment. The information is stored in the format NextSegL, NextSegR, followed by the
five segments called LeftSegs(1) to LeftSegs(5). These variables will be explained later in this paper.

**Code – The 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. For those who read this in color, *green italic 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  ‘The paths are several linked segments  
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  
startPt As Integer  ‘index of beginning point of segment  
length As Single  ‘length of segment in feet!!!!!!!!!!!!!!!!!!!!!!  
leftSegs(5) As Integer  ‘when entering the circle look for cars on the five 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  
End Type

Private Type Car  ‘this type defines the properties of the cars  
type As String  ‘Type of auto. car, bus or truck  
accel As Single  ‘car’s rate of acceleration  
active As Boolean  ‘if a car is being used or not  
color As Long  ‘the car’s color  
colort As String  ‘the name of 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 of car’s front in the segment  
locationB As Single  ‘location of car’s back in the segment  
new As Boolean  ‘if this is true the program will know not to “erase” the car after the first time step  
nextsegf As Integer  ‘next segment car front is headed for  
nextsegb As Integer  ‘next segment car back is headed for  
"-1” for an exit segment and "0" if not assigned yet  
segmentf As Integer  ‘number of the segment car’s front is on
segmentb As Integer 'number of the segment car's back is on
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
carspeedup As Integer 'for debugging
lag As Single 'lag data for entering circle
gap As Single 'Minimum accepted gap
Follow As Single 'Follow-up value
Tail As Single 'How close a car will get to the one in front of it
yield As Boolean 'Tells if the car is yielding
YieldTime As Single 'The time a car starts to yield, for measuring how
long a car yields
CarsYielded As Integer 'Number times the car has been through the yield
process
End Type

Private myPts(200) As Point 'array of points
Private mySegs(100) 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 PutCarInNow(6) As Single 'the probability that a car will enter
Dim counter(6, 6) As Integer 'keeps track of the mean delay time
Dim TimeHolder As Single 'keeps track of when to add a car
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 your car (backcar) and the car in front (frontcar), and minimum gap allowed between them
Dim numpoints As Integer, numseg As Integer 'number of points and segments
Public roundname As String 'Name of the roundabout files
Private Startn(6, 6) As Way 'An array of entrances and exits
Private startnname(6) As String 'The names of the entrances of the Roundabout
Private endname(6) As String 'The names of the exits of the roundabout
Public startnum As Integer 'Number of entrances of the roundabout
Public endnum As Integer 'Number of exits of the roundabout
Private Turn(6, 6) As Single
'dThis holds the calculated chance of a vehicle
making any given turn.'
Private scalenum As Single
'Scale of the roundabout files'
Private SpeedDif As Single
'Differences in Current car and car yielding to'
Private difference As Single
'Differences in the time it will take to get to'
contested intersection
'path to where the files are'
Private path As String
'For real time calculations'
Private Rtime As Single
'The current time of the system'
Private CurrentTime As Single

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 starts, a main menu is displayed (See Figure 4). The operator is able to choose whether to simulate a roundabout or prepare a roundabout for simulation. After choosing to simulate, a new form is displayed with only one button active. That is the Input Data button on the Simulation form (See Figure 5.). When this button is clicked, an open dialogue box is displayed to allow selection of the roundabout files. Once the roundabout is picked, the program gathers all the information about the roundabout by reading the appropriate .pth, .pts, .seg, and .set files.

The input data button is now active and when that is clicked a separate form comes up that allows the controlling data for the program to be edited. This new form, frmInput (see Figure 6),
uses a 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, as well as the time at which the output data starts recording are also included on this form. The user may also define the length of a time step, or how much time passes between movements of the cars. A "real time" choice has also been added; thus, the time that passes in the simulation is that same as the time that passes in the real world. 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 (See Figure 5). The code under (or associated with) this button is fairly straightforward. It initializes variables and properties to be used later, and it calls procedures that setup the points, and segments.

In PointSetup the program opens the file point.txt and reads in the x and y coordinates for each of the points. EndPointSetup "hardwires" the indexes of the points that define each segment. SegSetup sets the length property 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.

**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 (See Figure 6), and, for each entrance, it calculates the probability that a car would turn in a certain direction. It will also calculate the probability of cars entering each entrance at each timestep.
In computing the probabilities, PathCalculations uses an array that is the number of entrances by the number of exits. For example, if there were 2 entrances and 3 exits then the array would be 2 by 3. The "odds" for a path being taken given the entrance is calculated. The value for 2,1 for example is the simple probability that a car will turn at the first exit. The value of 2,2 is the simple probability of a car making the second turn plus the value for the first. The last value, or 2,3 is the simple probability of a car making the third turn and the value of the second turn, this would be one because this is the last turn.

This will be elaborated on in the explanation of AddCars. The last step 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 7). It calls 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 necessary.
AddCars —

AddCars searches all the cars being used until it finds the first four that are inactive (See Figure 8). Each car is given a chance to enter one of the four entrances. CheckProbability determines if a car will enter or not (See Figure 9). It does so 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. When a vehicle is “allowed in”, the CarSetup routine is called.

CarSetup—

This routine uses the current car in the myCars array and makes sure that the active property is set to true. It also sets the type, dimensions, color and desired speed for each car based on the values given in the traffic statistics window (Figure 10). The operator is allowed to make traffic that consists of any percentage of cars, trucks, and buses. Of course, these various vehicles may have different properties depending on their type. Numbers with a mean and standard deviation are randomly assigned based on a normal distribution. The New Property is set to true so the program knows that this is the first time each car will be drawn.

EnterNow makes the car active and sets its location to the beginning of the entrance segment or behind the last car on the segment if there are cars that are off the screen waiting to enter. Next, the car’s destination is made known using the FindExit procedure. This is where the other variables defined in PathCalculations are used.

FindExit is based on the logic that the sum of all the probabilities of a car taking one of any number of possible paths is one. Remember when finding the values for the array we kept adding the previous variable to the correct probability to get the next. 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. Below is a section of the code with an example.
j = myCars(i).segment  'the entrance the car is on., let's say 2
x = Rnd  'a random number between 0 and almost 1, let's say 0.754832
For k = 1 To endnum - 1  'endnum in this case will be 3. So k will be 1 then 2.
  If X < Turn(j, k) Then  'first time we are checking turn(2,1) which = .333
    'No .754832 is larger than .333
    'The second time we would be checking turn(2,2) which = .666 again < .754832
    myCars(i).exit = Startn(j, k).ExitSeg
    Exit For
  End If

A Roundabout Animation - 16
Next k
If k = endnum Then ‘Yes k = 3 and endnum = 3 therefore this is our turn.
   myCars(i).exit = Startn(j, k).ExitSeg ‘This sets the segment that the car
   ‘will take for a right turn.
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 turn(j,k) then it will take the exit for startin(j,k).

EnterNow calls the FindNextSegment procedure after executing FindExit. First, we check to see if there are any segments beyond the segment that the car fronts and backs in question are on (see Figure 11). If there are no segments then the car’s nextseg property is set to negative one. Thus, the program will 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).segmentf ‘segment the front of car i is in
    If segin = -1 Then ‘The front of our car is out of the simulation and ignore it.
       MyCars(i).nextsegf = -1
    ElseIf mySegs(segin).nextSegL = 0 Then ‘We are on the exit ramp
       myCars(i).nextsegf = -1
    ElseIf mySegs(segin).nextSegR = 0 Then ‘There is only one segment in front of
       this segment
       myCars(i).nextsegf = mySegs(segin).nextSegL
    ElseIf myCars(i).exit = mySegs(segin).nextSegR Then ‘This is the exit the car is
       looking for and will take the right turn.
       myCars(i).nextsegf = mySegs(segin).nextSegR
    Else ‘This is not the car’s exit and will stay in the roundabout.
       myCars(i).nextsegf = mySegs(segin).nextSegL
    End If

This routine is repeated for the back point of the car as well. This procedure has worked very well, and has been only slightly modified since the spring course in 1996.

---

A Car Travelling along a Segment

Figure 11 – Car-Fronts and Backs
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 update the carsIn array. The carsIn array is set up to hold 50 numbers but we have to have slots for both the fronts of cars and the backs of cars. So, the carsIn array will hold 25 cars. The odd carsIn (carsIn(1), carsIn(3), carsIn(5)...) hold the values of the back of the car that is in the spot. A zero means no car. The even carsIn (carsIn(2), carsIn(4), carsIn(6)...) hold the values of the front of a car that is in the spot. When updating the array, we move all of the cars in the array to the next two highest slot in the array for example (carsIn(1) goes to carsIn(3) and carsIn(2) goes to carsIn(4)) and put our car in slot one or two.

We now know where the car has entered and where it will exit, and can update the grid on frmVPH (see Figure 12). 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. We do this by calling the procedure AdjustSpeeds. (The crash rate at a modern roundabout is typically around one per 2 million entering vehicles. We disregard this small probability in our simulation, and assume that cars do not collide when estimating average delays.

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** –

In AdjustSpeeds, we set values to the variables: accel, taken from the acceleration property of the given car; backcar, car i or the car being controlled (conceptually the car

---

we are in which we are riding); thisseg, the segment backcar is on; and nextseg, the segment to where backcar is heading. Typical speed and acceleration values have been obtained within this project, and are also presented in separate publications. The logic for AdjustSpeeds is given in the flow chart labeled Figure 13.

Figure 13 – Flowchart for AdjustSpeeds

At this point, AdjustSpeeds enters a “do loop” in order to find a value for $j$. This is the index of the place of the front of the backcar 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.

The routine sets the speedup variable to a number from 1 to 4. A 1 means the car can speed up; a 2 means that the car has to slow down; a 3 means that the car yields to another car in the roundabout; finally a 4 means that the car is speeding up from a stopped position. Now, we find the value of allowedGap, using the equation: allowedGap = myCars(i).Tail * myCars(backcar).speed + 2. The myCars(i).Tail is the value taken from car statistics window. myCars(backcar).speed is backcar’s speed. The two is added because if the speed were 0 the allowed gap would be 0 and that would cause problems.

Once these variables are defined, we can find the value for the back of the 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. A non-zero number will indicate the back of the frontcar, and Gap’s value will be the difference of frontcar’s back location and backcar’s front location. 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 needs to 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 = mySegs(thisseg2).length - myCars(backcar).locationf ‘the remaining length of thisseg
Do ‘the program will loop back to here
frontcar = mySegs(nextseg2).CarsIn(1) ‘the rear of the last car on nextseg
frontcarfront = mySegs(nextseg2).CarsIn(2) ‘the front of the last car on nextseg
If frontcar <> 0 Then ‘is there a car on nextseg?
    gap = gap + myCars(frontcar).locationb ‘add the length of nextseg traveled by frontcar if there is one.
    Exit Do
Else
    gap = gap + mySegs(nextseg2).length ‘add the entire length of nextseg if there is not.
End If
thisseg2 = nextseg2 ‘if frontcar wasn’t found we need to find the next nextseg
If mySegs(thisseg2).nextSegR = 0 Then ‘this part is just like FindNextSegment
    nextseg2 = mySegs(thisseg2).nextSegL.
ElseIf myCars(backcar).exit = mySegs(thisseg2).nextSegR Then
    nextseg2 = mySegs(thisseg2).nextSegR
Else
    nextseg2 = mySegs(thisseg2).nextSegL
End If
If nextseg2 = 0 Then gap = 4 * allowedGap + gap 'to make sure that the gap will be large and there won't be abnormal slowing down right before exiting the simulation.
Loop Until gap > allowedGap Or thisseg2 >= 99 'keep looking until it doesn't matter

The second to last line was added because the cars were slowing prematurely, just before they got on the last segment before leaving the simulation. As they came 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. Typical values of critical gaps were observed and researched through studies of literature within this project and integrated into the code.

FindFrontCar was created to find an accurate value for Gap. With this value, we can now adjust backcar's speed if necessary. If Gap is less than the allowedGap, we need to slow backcar and speedup is set to 2. We also must check to see if there is a car to the left at a yield. If there is a car to the left we find how far away it is. If the car is back further than the acceptable lag then we will go faster if we can, speedup = 1. Lag is the time it takes for a car that one would have to yield to, to reach the intersection. When the car to the left is closer than the acceptable lag we will slow to a stop and yield to the other car, speedup = 3. When we are yielding we start looking for an acceptable gap so we can drive into the roundabout, speedup = 4.

Once the speedup is defined, we act accordingly. If speedup = 1, we will accelerate. To do this we add the product of accel, the acceleration in ft/sec^2 and the time that has passed since last we checked, deltaT to backcar's speed. Next, we check backcar's speed. if more than the desired speed, we set the speed to the desired speed. When speedup is 2, the car must slow because it is getting too close to the car in front of it. This is done by subtracting the product of modaccel, the acceleration in ft/sec^2 and the time that has passed since last we checked deltaT from backcar's speed. Modaccel is the modified acceleration of the car. This takes in account how close the two cars are and the difference in their speeds. Therefore, when one car is “coming up fast” to a slow car it will brake harder. Again, we check the new speed, if it is less than 0 it is set to 0. A speedup of 3 means that the car has to yield. This is the same processes of slowing down but the yield variable is set to true, to know that the car is yielding. On a speedup of a 4, the driver give it a little higher acceleration so the car will join traffic smoothly.

MoveCars –

The procedure MoveCars has been split into two sub-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).locationf = myCars(i).locationf + myCars(i).speed * deltaT, and: myCars(i).locationb = myCars(i).locationb + myCars(i).speed * deltaT. This moves both the front and rear of the car the distance the car would have traveled in one timestep. Therefore, when we multiply the car's speed with the elapsed time we get the distance the car has actually gone.
If either part of the car's location becomes larger than the length of the segment it is on, then the car has moved onto the next segment and must be treated accordingly. First, the carsIn array 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 the car is made inactive, New Property is set to be true, and it is erased so that 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 14). If the car is not leaving the simulation, then we must update the carsIn array of nextseg. After that, 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 orienting them. The current method of drawing the cars is by drawing a line from the front of the car to the back of the car. This helps to make the vehicle appear to be making smooth turns.

DrawCars loops through all the “active cars”, and will find their new x and y coordinates, measured at the two ends 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 segment, theta, and the car’s endpoints as described above. After setting the draw width to the width of the car, it will draw the car using Visual Basic’s line method. 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.

Figure 14 – Mean Time of Traffic

<table>
<thead>
<tr>
<th>Mean Time Between Entrances and Exits*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hop In Not</td>
</tr>
<tr>
<td>13.7</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>18.2</td>
</tr>
<tr>
<td>15.0</td>
</tr>
</tbody>
</table>

*All data here is given in Seconds
Conclusion

This simulation and animation program has now little resemblance to the initial homework problem. The program has a sense of realism with the ability to simulate any roundabout with various approach and exit combinations, rather than only a circle with four spokes in orthogonal positions. The new method of setting up a roundabout for simulation is swift and reasonably effortless. The improved version also allows for manipulating the traffic flow data in many ways not possible before. The composition of the traffic can be changed easily as well as how “drivers” behave, e.g., critical gap choice.

The program now shows us the average time that the cars spend traveling through the roundabout simulation. Traffic volumes are given upon the completion of a simulation. Thus, we can verify how many cars took what kind of turn, and make sure this number corresponds to the input value (actual or assumed flows).

This output gives us an idea of the control delay of a roundabout, and how this delay varies with entering volumes and accepted gaps. Accepted gaps, in turn, vary with geometric characteristics. However, the program has not yet been validated against its real life counterpart.

The model has been modified to include large vehicles such as trucks or buses that may encompass several model segments.
Appendix A – Simulation Code

Option Explicit
Private Type Point
  X As Single
  Y As Single
End Type

Private Type Way
  StartSeg As Integer
  ExitSeg As Integer
End Type

Private Type segment
  CarsIn(52) As Integer 'list of cars ends in the segment.
    'An Odd number means a back of a car,
    'An even number means a front of a car.
  totCars As Integer 'number of cars in segment
  endPt As Integer 'index of ending point
  length As Single 'length of segment in feet
  leftSegs(6) 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
End Type

Private Type Car
  type As String 'Type of auto, car, bus or truck
    accel As Single 'car's rate of acceleration
    decel As Single 'car's rate of braking
    active As Boolean 'if a car is being used or not
  color As Long
  colorr As String 'desired speed
  length As Single
  width As Single
  locationa As Single 'location of front in the segment
  locationb As Single 'location of back in the segment
  new As Boolean
  nextsegf As Integer 'next segment car front is headed for
  nextsegb As Integer 'next segment car back is headed for
  "0" if not assigned yet
  segmentf As Integer 'number of the segment front is on
  segmentb As Integer 'number of the segment back is on
  speed As Single 'actual speed
  exit As Integer 'assigned exit segment
  begintime As Single 'time enters
  entrance As Integer 'segment enters on
carspeedup As Integer 'for debugging
  lag As Single 'lag data for entering circle
gap As Single 'minimum accepted gap
  Follow As Single 'follow-up value
  Tail As Single 'how close a car will get to the one in front of it
  yield As Boolean 'tells if the car has already yielded
  YieldTime As Single 'the time a car starts to yield, for measuring how long a car yields
  CarsYielded As Integer 'Number times the car has been through the yield process
End Type

Private myPts(200) As Point 'array of points
Private mySegs(100) 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 deltaT As Single, YieldPt As Single 'Rate of timesteps and how far from intersection car yields
Private black As Long
Private carFront As Point, carBack As Point 'endpoints of the cars
Dim TimeSteps As Long 'number of timesteps
Dim PutCarInNow(6) As Single 'time it takes for the next car to enter
Dim counter(6, 6) As Integer 'keeps track of the mean delay time
Dim TimeHolder As Single 'keeps track of when to add a car
Dim frontcar As Integer, backcar As Integer 'these six variables are used to control speed
Dim thisseg As Integer, nextseg As Integer 'current segment car is in and next segment car will be in
Dim gap As Single, allowedGap As Single 'used to determine if a car needs to slow down
Dim numpoints As Integer, numseg As Integer 'number of points and segments
Public roundname As String 'Name of the roundabout files
Private Startn(6, 6) As Way 'An array of entrances and exits
Private startname(6) As String 'The names of the entrances of the Roundabout
Private endname(6) As String 'The names of the exits of the Roundabout
Public startnum As Integer 'Number of entrances of the roundabout
Public endnum As Integer  'Number of exits of the roundabout
Private Turn(6, 6) As Single  'Percentage of a car making certain turns
Private scalenum As Single  'Scale of the roundabout files
Private SpeedDif As Single  'Differences in Current car and car yielding to
Private difference As Single  'Differences in the time it will take to get to contested intersection
Private path As String  'path to where the files are
Private Rtime As Single  'For real time calculations
Private CurrentTime As Single  'The current time of the system

Private Sub OpenDia()

End Sub
Private Sub btnLoad_Click()

Dim i As Integer, j As Integer

roundname = ""
cdbLoad.FileName = App.path
OnError GoTo ErrHandler
cdbLoad.ShowOpen
roundname = cdbLoad.FileName
i = InStr(1, roundname, ".")
path = Left(roundname, i)

Open roundname For Input As #4
Input #4, startnum, endnum

For i = 1 To startnum
Input #4, startname(i)
Next i

For i = 1 To endnum
Input #4, endname(i)
Next i

For i = 1 To startnum
  For j = 1 To endnum
    Input #4, Startn(i, j).StartSeg, Startn(i, j).ExitSeg
  Next j
Next i

Close #4
cmdInput.Enabled = True
cmdSetUp.Enabled = False
cmdRun.Enabled = False
cmdStop.Enabled = False
cmdClear.Enabled = False
frmSim.pctPix.Picture = LoadPicture(App.Path + "\setup.jpg")

Call ResetGrids
Exit Sub

ErrHandler:
  'User pressed Cancel button.
  Exit Sub

End Sub
Private Sub cmdClear_Click()

Dim i As Integer, j As Integer

petPix.Cls
For i = 1 To startnum
  Form3.grdVPH.Row = i
  For j = 1 To endnum + 1
    Form3.grdVPH.Col = j
    Form3.grdVPH.Text = CStr(0)
  Next j
Next i

For i = 1 To 6
  For j = 1 To 6
    counter(i, j) = 1
  Next j
Next i

For i = 1 To startnum
  Form5.grdDelays.Row = i
  For j = 1 To endnum
    Form5.grdDelays.Col = j
    Form5.grdDelays.Text = CStr(0)
  Next j
Next i

For i = 0 To startnum - 1
  Form4.Label(i).Visible = False
Next i

For i = 0 To endnum - 1
  Form4.Label(c(i).Visible = False
Next i

Form3.cmdVPH.Enabled = False
cmdRun.Enabled = False
timeSteps = 0
currentTime = timeSteps
frmSim.pctPix.Picture = LoadPicture(App.Path + "\setup.jpg")

End Sub
Private Sub cmdDelay_Click()

Form5.Show

End Sub
Private Sub cmdInput_Click()

Call InputGrids
cmdSim.pctPix.Picture = LoadPicture(App.Path + "\setup.jpg")

End Sub
Private Sub cmdNewSetup_Click()
    frmMain_Menu.Show
    frmSim.Hide
End Sub
Private Sub cmdOutput_Click()
    Form3.Show
End Sub
Private Sub cmdQuit_Click()
End Sub
Private Sub cmdRun_Click()
    Rtime = Timer
    Timer1.Enabled = True
    cmdStop.Enabled = True
    cmdRun.Enabled = False
    cmdClear.Enabled = False
    frmDeBug.Show
    frmSegProp.Show
    frmCarsIn.Show
End Sub
Private Sub cmdStop_Click()
    Timer1.Enabled = False
    cmdRun.Enabled = True
    cmdStop.Enabled = False
    cmdClear.Enabled = True
End Sub
Private Sub Command1_Click()
    frmSim.Hide
    frmStats.Show
End Sub
Private Sub Form_Load()
    cmdSetUp.Enabled = False
    cmdRun.Enabled = False
    cmdStop.Enabled = False
    cmdClear.Enabled = False
    cmdInput.Enabled = False
End Sub
Public Sub InputGrids()
    cmdSetUp.Enabled = True
    Form4.Show
End Sub
Public Sub ResetGrids()
    Dim i As Integer, j As Integer
    Form4.gridInput.Rows = startnum + 1
    Form4.gridInput.Cols = endnum + 1
    Form4.gridInput.Col = 0
    For i = 1 To startnum
        Form4.gridInput.Row = i
        Form4.gridInput.Text = startname(i)
        Next i
    Form4.gridInput.Row = 0
    For i = 1 To endnum
        Form4.gridInput.Col = i
        Form4.gridInput.Text = endname(i)
        Next i
    For i = 1 To startnum
        Form4.gridInput.Row = i
        For j = 1 To endnum
            Form4.gridInput.Col = j
            Form4.gridInput.Text = CStr(0)
            Next j
        Next i
        For i = 0 To startnum - 1
            Form4.Label(i).Visible = True
            Next i
        For i = 0 To endnum - 1
            Form4.Labelc(i).Visible = True
            Next i
        Form5.gridDelays.Rows = startnum + 1
        Form5.gridDelays.Cols = endnum + 1
        For i = 1 To startnum
            Form5.gridDelays.Row = i
            For j = 1 To endnum
                Form5.gridDelays.Col = j
                Form5.gridDelays.Text = CStr(0)
                Next j
            Next i
            Form5.gridDelays.Col = 0
            For i = 1 To startnum
                Form5.gridDelays.Row = i
                Form5.gridDelays.Text = startname(i)
                Next i
            Form5.gridDelays.Row = 0
            For i = 1 To endnum
                Form5.gridDelays.Col = i
                Form5.gridDelays.Text = endname(i)
                Next i
            Form3.gridVPH.Rows = startnum + 1
            Form3.gridVPH.Cols = endnum + 2
            For i = 1 To startnum
                Form3.gridVPH.Row = i
                For j = 1 To endnum
                    Form3.gridVPH.Col = j
                    Form3.gridVPH.Text = CStr(0)
                    Next j
                Next i
                Form3.gridVPH.Col = 0
                For i = 1 To startnum
                    Form3.gridVPH.Row = i
                    Next i
A Roundabout Animation - 26
Form3.grdVPH.Text = startname(i)
Next i

Form3.grdVPH.Row = 0
For i = 1 To endnum
    Form3.grdVPH.Col = i
    Form3.grdVPH.Text = endname(i)
Next i

Form3.grdVPH.Col = i
Form3.grdVPH.Text = "Entr. Sum"

End Sub
Private Sub Form_Unload(Cancel As Integer)
End

End Sub
Private Sub CMDSETUP_CLICK()

Dim X As Single, i As Integer, j As Integer, k As Integer
Dim picname As String

cmdRun.Enabled = True
cmdClear.Enabled = True

Randomize
black = RGB(0, 0, 0)
Erase mySegs
Erase myCars

Open path + "pts" For Input As #1
Input #1, picname

Open path + "set" For Input As #3
Input #3, scalenum

frmSim.pctPix.Picture = LoadPicture(App.path + "v" + picname)

YieldPt = CSng(frmStats.txtYieldPt.Text)
pctPix.Scale (-200 * scalenum, 200 * scalenum)- (200 * scalenum, -200 * scalenum)
pctPix.DrawMode = 10
pctPix.DrawStyle = 0
pctPix.DrawWidth = 1

Call PointSetup
Call EndPointSetup
Call SegSetup

numCars = 100
deltaT = CSng(Form4.txtDeltaT)

Call PathCalculations
Call CarSetup

TimeSteps = 0

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

Next i

End Sub
Private Sub PointSetup()
Dim i As Integer

i = 0
Do While Not EOF(1)
    i = i + 1
    Input #1, myPts(i).X, myPts(i).Y
    myPts(i).X = myPts(i).X * scalenum
    myPts(i).Y = myPts(i).Y * scalenum
Loop

numpoints = i
Close #1

End Sub
Private Sub EndPointSetup()

Dim i As Integer

Open path + "seg" For Input As #2

Input #2, mySegs(i).startPt, mySegs(i).endPt

numseg = i
Close #2

End Sub
Private Sub SegSetup()

Dim i As Integer, j As Integer

For i = 1 To numseg
    If cbxShowSegs.Value = 1 Then
    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
        Input #3, .nextSegL, .nextSegR, .leftSegs(1), .leftSegs(2), .leftSegs(3), .leftSegs(4), .leftSegs(5), .leftSegs(6)
    End With
Next i

Close #3

If cbxShowSegs.Value = 1 Then
    For i = 1 To numpoints

```vbnet
    private Sub PathCalculations()
        Dim i As Integer, j As Integer
        Dim sum(6) As Single

        'new method for calculation of turns
        For i = 1 To startnum
            sum(i) = 0
            Form4.grdInput.Row = i
            For j = 1 To endnum
                Form4.grdInput.Col = j
                sum(i) = sum(i) + CSng(Form4.grdInput.Text)
            Next j
            For j = 1 To endnum - 1
                Form4.grdInput.Col = j + 1
                If sum(i) <= 0 Then
                    Turn(i, j) = (CSng(Form4.grdInput.Text) / sum(i)) + Turn(i, j - 1)
                Else
                    Turn(i, j) = 0
                End If
            Next j
        Next i
    End Sub

    Private Sub PctPix_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
        txtX.Text = X
        txtY.Text = Y
    End Sub

    private Sub timer1_timer()
        Dim ptime As Single, TimePass As Single, dummy As Single
        If Form4.chkRTTime.Value = 1 Then
            ptime = Timer
            TimePass = ptime - Rtime
            Rtime = Timer
            CurrentTime = CurrentTime + deltaT
        Else
            TimeSteps = TimeSteps + 1 'counts the steps
            CurrentTime = deltaT * TimeSteps
        End If
        dummy = CurrentTime
        txtElapsedTime.Text = Format(dummy, "###.###")
        If chkSingleStep.Value = 1 Then
            Timer1.Enabled = False
            cmdRun.Enabled = True
        End If
        Call AddCars
        Call MoveCars
        Call CheckTime
        If chkDeBug.Value = 1 Then
            Call DeBugPrint
            Call SegProp
        End If
        If chkCarsIn.Value = 1 Then
            Call CarsInInfo
        End If
        End Sub
        Private Sub CarsInInfo()
            Dim i As Integer, j As Integer
            frmCarsIn.petCarsIn.Cls
            For i = 1 To numseg
                frmCarsIn.petCarsIn.Print "seg "; i; " ";
                For j = 1 To 20
                    frmCarsIn.petCarsIn.Print mySegs(i).CarsIn(j)
                Next j
                frmCarsIn.petCarsIn.Print mySegs(i).CarsIn(j)
            Next i
            End Sub
        Private Sub AddCars()
            Dim i As Integer, j As Integer
            j = 0
            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 it is time to enter a car
                End If
                If j = startnum Then Exit For
            Next i
        End Sub
```

A Roundabout Animation - 28
If $X \leq \text{PutCarInNow}(j) \times \text{deltaT}$ Then it's time to put a car in

```
myCars(i).segmentf = Startn(j, 1).StartSeg
myCars(i).segmentfb = Startn(j, 1).StartSeg
myCars(i).entrance = Startn(j, 1).StartSeg
Call CarSetup(i)
Call EnterNow(i)
```
End If

End Sub

Public Sub EnterNow(i As Integer)

Dim Y As Single, j As Integer, k As Integer,
segin As Integer, lastCar As Integer

segin = myCars(i).segmentf
lastCar = mySegs(segin).CarsIn(2)

' Finds the location of last car on segment if their is no car in segment then the car starts at the beginning
If myCars(lastCar).locationf <= 15 Then
  myCars(i).active = True
  myCars(i).locationf = myCars(lastCar).locationf - 10
  ' Puts a car 5 feet behind the last car
  myCars(i).locationb = myCars(i).locationf - myCars(i).length
Else
  myCars(i).active = True
  myCars(i).locationb = 5
  myCars(i).locationb = myCars(i).locationf - myCars(i).length
End If

Call FindExit(i)
Call FindNextSegment(i)

For $k = 1$ To 2
  For $j = 50$ To 2 Step -1
    mySegs(segin).CarsIn(j) = mySegs(segin).CarsIn(j - 1)
  Next j
Next k

mySegs(segin).CarsIn(1) = i
mySegs(segin).CarsIn(2) = i
mySegs(segin).totCars = mySegs(segin).totCars + 1

Call AdjustSpeeds(i)
Call AdjustSpeeds(i)

```
TimeHolder = deltaT * TimeSteps
myCars(i).begintime = TimeHolder
myCars(i).entrance = segin
```
End Sub

Public Sub FindExit(i As Integer)

Dim segin As Integer
Dim X As Single
Dim j As Integer, k As Integer

X = Rnd

For $j = 1$ To startnum
  If Startn(j, 1).StartSeg = myCars(i).entrance
    Exit For
  Next j

For $k = 1$ To endnum - 1
  If X < Turn(j, k) Then
    myCars(i).exit = Startn(j, k).ExitSeg
    Exit For
  End If
Next k

If k = endnum Then
  myCars(i).exit = Startn(j, k).ExitSeg
End If

Call UpdateOutput(i, j, k)

End Sub

Public Sub FindNextSegment(i As Integer)

If you are in the subroutine then you are trying to figure out where the car i is going next.
Dim segin As Integer

segin = myCars(i).segmentf 'segment the front of car i is in

If segin = -1 Then
ElseIf mySegs(segin).nextSegL = 0 Then We are on the exit ramp
  myCars(i).nextsegf = -1
ElseIf mySegs(segin).nextSegR = 0 Then
  myCars(i).nextsegf = mySegs(segin).nextSegL
ElseIf myCars(i).exit = mySegs(segin).nextSegR Then
  myCars(i).nextsegf = mySegs(segin).nextSegR
Else
  myCars(i).nextsegf = mySegs(segin).nextSegL
End If

segin = myCars(i).segmentb

If mySegs(segin).nextSegL = 0 Then We are on the exit ramp
  myCars(i).nextsegb = -1
ElseIf mySegs(segin).nextSegR = 0 Then
  myCars(i).nextsegb = mySegs(segin).nextSegL
ElseIf myCars(i).exit = mySegs(segin).nextSegR Then
  myCars(i).nextsegb = mySegs(segin).nextSegR
Else
  myCars(i).nextsegb = mySegs(segin).nextSegL
```
End If

End Sub

Public Sub UpdateOutput(i As Integer, st As Integer, es As Integer)

Dim segin As Integer, j As Integer, k As Integer, esum As Integer
Dim exsum As Integer

If CurrentTime >= (Form4.txtETime.Text * 60) Then

    Form3.grdVPH.Row = st
    Form3.grdVPH.Col = en
    Form3.grdVPH.Text = CStr(CInt(Form3.grdVPH.Text) + 1)

    For k = 1 To startnum
        esum = 0
        Form3.grdVPH.Row = k
        For j = 1 To endnum
            Form3.grdVPH.Col = j
            esum = esum + CInt(Form3.grdVPH.Text)
            Next j
        End If

    End Sub

Public Sub AdjustSpeeds(i As Integer)

Dim j As Integer, k As Integer, g As Integer, h As Integer
Dim accel As Single, YieldPt As Single, YieldSeg As Integer
Dim lookLeftSeg(2) As Integer, lookLeftcar(2) As Integer
Dim speedup As Integer
Dim moddec As Single

speedup = 0
accel = myCars(i).accel
backcar = i "car we are in"
thisseg = myCars(backcar).segment
nextseg = myCars(backcar).nextseg
allowedGap = myCars(i).Tail * myCars(backcar).speed + 6
        gap = 0
        j = 0
        k = 0

    For j = 2 To 50 Step 2 'This routine will find where "our" car is.
        If i = mySegs(thisseg).CarsIn(j) Then Exit For
    Next j

    If j >= 52 Then Exit Sub

    If mySegs(thisseg).CarsIn(j + 1) <> 0 Then "our" car on this segment
        frontcar = mySegs(thisseg).CarsIn(j + 1)
        gap = ((myCars(frontcar).locationb - myCars(backcar).locationf))
    ElseIf mySegs(thisseg).CarsIn(j + 2) <> 0 Then "our" car on this segment
        this SHOULD NOT happen 'will mean a collision.
    ElseIf nextseg = -1 Then 'this is an exit segment but add to gap so there is no slow down
        gap = gap + 4 * allowedGap
    Else
        Call FindFrontCar(i) 'finds the next car in the path of the car we are "in"
    End If

    For g = 1 To 6 'This finds the first car on the left.
        If mySegs(mySegs(thisseg).leftSegs(g)).CarsIn(2) <> 0 Then
            lookLeftcar(1) = mySegs(mySegs(thisseg).leftSegs(g)).CarsIn(2)
            Exit For
        End If
    Next g

    If myCars(lookLeftcar(1)).segmentb <= 6 Then "second car on the left"
        For g = myCars(lookLeftcar(1)).segmentb To 6
            If h = 1 To 50 Step 2
                If mySegs(mySegs(thisseg).leftSegs(g)).CarsIn(h) = lookLeftcar(1) Then
                    Exit For
                End If
            Next h

            If h > 1 Then 'This will set the second car on the left
                h = h - 1
                lookLeftcar(2) = mySegs(mySegs(thisseg).leftSegs(g)).CarsIn(h)
                Exit For
            End If
        Next g

        If h = 1 Then 'This will set the second car on the left
            h = g
            For g = h To 6
                Exit For
            Next g
        End If
If my-
Segs(mySegs(thissseg).leftSegs(g)).CarsIn(2) <> 0 Then
    lookLeftcar(2) = my-
Segs(mySegs(thissseg).leftSegs(g)).CarsIn(2)
Exit For
End If
Next g
End If
Else
    lookLeftcar(2) = 0
End If
If lookLeftcar(1) = lookLeftcar(2) Then
    lookLeftcar(2) = 0
End If
YieldPt = frmStats.txtYieldPt
If mySegs(thissseg).leftSegs(1) <> 0 Then
    If mySegs(thissseg).endPt = my-
Segs(mySegs(thissseg).leftSegs(1)).endPt And
YieldPt < mySegs(thissseg).length Then
        Yieldseg = thissseg
    ElseIf mySegs(thissseg).endPt <> my-
Segs(mySegs(thissseg).leftSegs(1)).endPt And
YieldPt > mySegs(thissseg).length Then
        YieldPt = YieldPt - mySegs(nextseg).length
        Yieldseg = thissseg
    ElseIf mySegs(thissseg).endPt = my-
Segs(mySegs(thissseg).leftSegs(1)).endPt Then
        YieldPt = mySegs(thissseg).length
        Yieldseg = thissseg
    Else
        Yieldseg = nextseg
    End If
End If
'difference As Single
' Code that might not be needed now.
'
j < mySegs(thissseg).totCars Then 'Deal with cars in this segment first
' If Gap > allowedGap Then
    speedup = 1
' Else
    speedup = 2
' End If
If thisseg = Yieldseg And gap > allowedGap Then 'We must look to see if there is a need to yield
    speedup = YieldGap(YieldPt, Yieldseg, look-
Leftcar(1), lookLeftcar(2), thissseg)
ElseIf thisseg = Yieldseg And gap < allowedGap Then
    speedup = 3
    SpeedDiff = myCars(frontcar).speed - my-
Cars(backcar).speed
ElseIf myCars(backcar).yield = True And thisseg = Yieldseg Then 'It is time to start looking a gaps
    speedup = YieldGap(lookLeftcar(1), lookLeft-
car(2), thissseg)
Else if gap > allowedGap Then
    speedup = 1 'Give it the gas!
Else
    speedup = 2 'Slow it down bud!
    SpeedDiff = myCars(frontcar).speed - my-
Cars(backcar).speed
End If
If speedup = 0 Then speedup = 1
myCars(backcar).carspeedup = speedup
Select Case speedup
Case 1 'Speed up
    myCars(backcar).speed = my-
Cars(backcar).speed + accel * deltaT
    If myCars(backcar).speed > my-
Cars(backcar).deSpeed Then my-
Cars(backcar).speed = myCars(backcar).deSpeed
    If myCars(backcar).speed < 0 Then my-
Cars(backcar).speed = 0
    myCars(backcar).yield = False
Case 2 'slow down
    SpeedDiff = SpeedDiff * 2
    moddecel = myCars(backcar).decel + Abs(SpeedDiff)
    If moddecel < 12.88 Then moddecel = 12.88
    myCars(backcar).speed = my-
Cars(backcar).speed - moddecel * deltaT
    If myCars(backcar).speed > my-
Cars(backcar).deSpeed Then my-
Cars(backcar).speed = myCars(backcar).deSpeed
    If myCars(backcar).speed < 0 Then my-
Cars(backcar).speed = 0
    myCars(backcar).yield = True
Case 3 'slow down or stop at yield
    SpeedDiff = SpeedDiff * 2
    moddecel = myCars(backcar).decel + Abs(SpeedDiff)
    If moddecel < 12.88 Then moddecel = 12.88
    myCars(backcar).speed = my-
Cars(backcar).speed - moddecel * deltaT
    If myCars(backcar).speed < 0 Then my-
Cars(backcar).speed = 0
    myCars(backcar).yield = True
Case 4 'start from stop
    myCars(backcar).speed = my-
Cars(backcar).speed + 20 * accel * deltaT
    If myCars(backcar).speed > my-
Cars(backcar).deSpeed Then my-
Cars(backcar).speed = myCars(backcar).deSpeed
    myCars(backcar).yield = False
End Select
End Sub
Private Function YieldGap(lookLeftcar1 As Integer, lookLeftcar2 As Integer, thissseg As Integer) As Integer
Dim GapTime As Single, xflag As Boolean
Dim distance As Single
Dim LeftDis As Single, LeftTime As Single, RightTime As Single
Dim accel As Single
Dim i As Integer, j As Integer
Dim ModLag As Single 'modified lag time accepted
Dim ModGap As Single 'modified lag time accepted
Dim YldTimeLength As Single 'modified accel = myCars(backcar).accel
xflag = False

If lookLeftcar1 = 0 And gap >= allowedGap Then
  YieldGap = 4
  Exit Function
End If

For i = 1 To 6
  If mySegs(thisseg).leftSegs(i) = myCars(lookLeftcar1).segmentb Then
    j = i - 1
    Exit For
  End If
Next i

LeftDis = mySegs(mySegs(thisseg).leftSegs(j + 1)).length - myCars(lookLeftcar1).location

For i = j To 1 Step -1
  LeftDis = LeftDis + mySegs(mySegs(thisseg).leftSegs(i)).length
Next i

LeftTime = LeftDis / (myCars(lookLeftcar1).speed + 0.001)
RightTime = CSng((mySegs(myCars(backcar).segmentf).length - myCars(backcar).location) / (myCars(backcar).speed + 0.001))
difference = LeftTime - RightTime

If difference >= myCars(backcar).lag Then 'finds the time difference between when car on the left
  YieldGap = 4
  Exit Function
End If

Else 'finds the distance between the two left cars
  If myCars(lookLeftcar1).segmentb = myCars(lookLeftcar2).segmentf Then
    distance = myCars(lookLeftcar1).location - myCars(lookLeftcar2).location
  ElseIf lookLeftcar2 = 0 Then
    YieldGap = 3
  Exit Function
  Else
    distance = mySegs(myCars(lookLeftcar2).segmentf).length - myCars(lookLeftcar2).location
  End If

  If myCars(lookLeftcar1).segmentb = myCars(lookLeftcar2).segmentf Then
    distance = distance + myCars(lookLeftcar1).segmentb = i
  Else
    distance = distance + mySegs(i).length
  End If
Loop
End If

GapTime = distance / (myCars(lookLeftcar2).speed + 0.001)
End If

If GapTime >= myCars(backcar).gap And difference < myCars(backcar).lag Then
  YieldGap = 4
Else
  YieldGap = 3
  SpeedDiff = myCars(lookLeftcar1).speed - myCars(backcar).speed
End If

End Function

Private Function YieldLag(YieldPt As Single, Yieldseg As Integer, lookLeftcar1 As Integer, lookLeftcar2 As Integer, thisseg As Integer) As Integer

Dim LeftDis As Single, LeftTime As Single, RightTime As Single
Dim i As Integer, k As Integer

If (((mySegs(myCars(backcar).segmentf).length) - (myCars(backcar).location)) > YieldPt) Or
  lookLeftcar1 = 0 Then
  YieldLag = 1
  Exit Function
End If

LeftDis = CSng((mySegs(myCars(lookLeftcar1).segmentf).length - myCars(lookLeftcar1).location)

For i = 1 To 6
  If myCars(lookLeftcar1).segmentf = mySegs(thisseg).leftSegs(i) Then
    k = i - 1
    Exit For
  End If
Next i

For i = k To 1 Step -1
  LeftDis = LeftDis + mySegs(mySegs(thisseg).leftSegs(i)).length
Next i

LeftTime = LeftDis / (myCars(lookLeftcar1).speed + 0.001)
RightTime = CSng((mySegs(myCars(backcar).segmentf).length - myCars(backcar).locationf) / (myCars(backcar).speed + 0.001))
difference = LeftTime - RightTime

If difference >= myCars(backcar).lag Then
  YieldLag = 1
Else
  YieldLag = 3
  SpeedDiff = myCars(lookLeftcar1).speed - myCars(backcar).speed
  myCars(backcar).YieldTime = CurrentTime
End If

End Function
Private Sub DeBugPrint()
Dim i As Integer

For i = 1 To 15
  frmDeBug.msfgDeBug.Row = i
  If myCars(i).active = True Then

    With myCars(i)
      frmDeBug.msfgDeBug.Col = 1
      frmDeBug.msfgDeBug.Text = .type
      frmDeBug.msfgDeBug.Col = 2
      frmDeBug.msfgDeBug.Text = .color
      frmDeBug.msfgDeBug.Col = 3
      frmDeBug.msfgDeBug.Text = "ON"
    End With
    Select Case .carsspeedup
    Case 1

    Case 2
      frmDeBug.msfgDeBug.Col = 4
      frmDeBug.msfgDeBug.Text = .deSpeed
      frmDeBug.msfgDeBug.Col = 5
      frmDeBug.msfgDeBug.Text = .speed
      frmDeBug.msfgDeBug.Col = 6
      frmDeBug.msfgDeBug.Text = .segmentf
      frmDeBug.msfgDeBug.Col = 7
      frmDeBug.msfgDeBug.Text = .segmentb
      frmDeBug.msfgDeBug.Col = 8
      frmDeBug.msfgDeBug.Text = .locationf
      frmDeBug.msfgDeBug.Col = 9
      frmDeBug.msfgDeBug.Text = .locationb
      frmDeBug.msfgDeBug.Col = 10
      frmDeBug.msfgDeBug.Text = .length
      frmDeBug.msfgDeBug.Col = 11
      frmDeBug.msfgDeBug.Text = Format(Sqr(((oldFront(i).X - oldBack(i).X) ^ 2) + ((oldFront(i).Y - oldBack(i).Y) ^ 2)), "###0.00")

    End Select
  End If

End Sub
Private Sub MoveCars()

Call SwitchSegments
Call DrawCars

End Sub
Public Sub SwitchSegments()
Dim i As Integer, j As Integer, segLengthf As Single, segLengthb As Single
Dim thisSeggf As Integer, nextSeggf As Integer
Dim thisSeggb As Integer, nextSeggb As Integer
Dim num As Integer, k As Integer

For i = 1 To numCars
  If myCars(i).active = True Then move car -- distance = rate * time
    Call AdjustSpeeds(i)
    thisSeggf = myCars(i).segmentf
    nextSeggf = myCars(i).nextsegf
    thisSeggb = myCars(i).segmentb
    nextSeggb = myCars(i).nextsegb
    segLengthf = mySegs(thisSeggf).length
    segLengthb = mySegs(thisSeggb).length
    myCars(i).locationf = myCars(i).locationf + myCars(i).speed * deltaT
    myCars(i).locationb = myCars(i).locationb + myCars(i).speed * deltaT

    If myCars(i).locationf > segLengthf Then
      'exit or move car to next segment
      If nextSeggf <> -1 Then
        For k = 2 To 50 Step 2
          If mySegs(thisSeggf).CarsIn(k) = i
            Then
              dum = k
              Exit For
            End If
          Next k
        mySegs(thisSeggf).CarsIn(dum) = 0
        myCars(i).segmentf = myCars(i).nextsegf
      End If
      For k = 1 To 2
        If
      End If
    End If
  End If
Next i

End Sub

For j = 50 To 1 Step -1 ' This makes space in carSim of the next segment
mySegs(nextSegg).CarsIn(j) = mySegs(nextSegg).CarsIn(j - 1)
Next j
Next k
mySegs(nextSegg).CarsIn(2) = i
Call FindNextSegment(i) ' i is the index of the current car

myCars(i).locationf = (my-Cars(i).locationf - mySegs(thisSegg).length)
Else
myCars(i).locationf = my-Cars(i).locationf
End If
End If

If myCars(i).locationb > segLengthb Then ' exit or move car to next segment
For k = 1 To 49 Step 2
If mySegs(thisSeggb).CarsIn(k) = i
Then
dum = k
Exit For
End If
Next k
mySegs(thisSeggb).CarsIn(dum) = 0
If nextSeggb <> -1 Then
mySegs(nextSeggb).totCars = my-Segs(nextSeggb).totCars + 1
mySegs(nextSeggb).CarsIn(1) = i ' shift current car into next segg
myCars(i).segmentb = my-Cars(i).nextSeggb
Call FindNextSegment(i) ' i is the index of the current car
myCars(i).locationb = (my-Cars(i).locationb - mySegs(thisSeggb).length)
Else
For k = 1 To 49 Step 2
If mySegs(thisSeggb).CarsIn(k) = i
Then
dum = k
Next k
mySegs(thisSeggb).CarsIn(dum) = 0
mySegs(thisSeggb).CarsIn(dum + 1) = 0
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, old-Front(i).Y)-(oldBack(i).X, oldBack(i).Y), _

myCars(i).color ' erase for the last time
Call FindDelayTime(i)
End If
End If
End If

Next i
End Sub

Public Sub FindDelayTime(i As Integer)
Dim j As Integer, k As Integer
Dim ave As Single, t As Single, sum As Single
If CurrentTime >= (60 * 
CSng(Form4.txtETime.Text)) Then

  t = CurrentTime - myCars(i).begintime
  For j = 1 To startnum
    If Startn(i, 1),StartSeg = myCars(i).entrance
    Then Exit For
    Next j
  For k = 1 To endnum
    If Startn(i, k),ExitSeg = myCars(i).exit
    Then Exit For
    Next k

  Form5.grdDelays.Row = j
  Form5.grdDelays.Col = k
  sum = CSng(Form5.grdDelays.Text) * counter(Form5.grdDelays.Row, Form5.grdDelays.Col)
  sum = sum + t
counter(Form5.grdDelays.Row, Form5.grdDelays.Col) = counter(Form5.grdDelays.Row, Form5.grdDelays.Col) + 1
ave = sum / counter(Form5.grdDelays.Row, Form5.grdDelays.Col)

  Form5.grdDelays.Text = Format(ave, "##.0")
End If
End Sub

Public Sub CheckTime()
If (deltaT * TimeSteps) >=
(Format4.txtRTimes.Text * 60) Then
Timer1.Enabled = False
cmdStop.Enabled = False
cmdRun.Enabled = True
cmdClear.Enabled = True
Form3.cmdVPH.Enabled = True
End If

If CurrentTime >= CSng(Form4.txtRTimes.Text * 60) Then
  Timer1.Enabled = False
cmdStop.Enabled = False
cmdRun.Enabled = True
cmdClear.Enabled = True
Form3.cmdVPH.Enabled = True
End If

End Sub
Public Sub CarSetup(i As Integer)

Dim X As Single

X = Rnd

If X < CLng(frmStats.txtBus.Text) / 100 Then
    myCars(i).type = "BUS"
ElseIf X < (CLng(frmStats.txtBus.Text) + CLng(frmStats.txtTruck.Text)) / 100 Then
    myCars(i).type = "TRUCK"
Else
    myCars(i).type = "CAR"
End If

Randomize
With myCars(i)
    .active = False

    X = Rnd
    If X < 0.2 Then
        .color = vbRed
        .color = "RED"
    ElseIf X < 0.4 Then
        .color = vbBlue
        .color = "BLUE"
    ElseIf X < 0.6 Then
        .color = vbGreen
        .color = "GREEN"
    ElseIf X < 0.8 Then
        .color = vbBlack
        .color = "BLACK"
    Else
        .color = vbCyan
        .color = "CYAN"
    End If

    If .type = "CAR" Then
        .accel = NormalDis(frmStats.txtCAccel, frmStats.txtCAccelDev)
        .decel = NormalDis(frmStats.txtCDecel, frmStats.txtCDecelDev)
        .Follow = NormalDis(frmStats.txtCFollow, frmStats.txtCFollowDev)
        .gap = NormalDis(frmStats.txtCCGap, frmStats.txtCCGapDev)
        .lag = NormalDis(frmStats.txtCLag, frmStats.txtCLagDev)
        .length = frmStats.txtCarLength + 2 * Rnd

        .new = True
        .speed = NormalDis(frmStats.txtCSpeed, frmStats.txtCSpeedDev)
        .speed = .speed * (22 / 15)
        .deSpeed = .speed
    .Tail = NormalDis(frmStats.txtCTail, frmStats.txtCTailDev)
    .width = frmStats.txtCarWidth + Rnd
    ElseIf .type = "BUS" Then
        .accel = NormalDis(frmStats.txtBAccel, frmStats.txtBAccelDev)
        .decel = NormalDis(frmStats.txtBDecel, frmStats.txtBDecelDev)
        .Follow = NormalDis(frmStats.txtBFollow, frmStats.txtBFollowDev)
        .gap = NormalDis(frmStats.txtBCGap, frmStats.txtBCGapDev)
        .lag = NormalDis(frmStats.txtBLag, frmStats.txtBLagDev)
        .length = frmStats.txtBusLength + 2 * Rnd
        .new = True
        .speed = NormalDis(frmStats.txtBSpeed, frmStats.txtBSpeedDev)
        .speed = .speed * (22 / 15)
        .deSpeed = .speed
        .Tail = NormalDis(frmStats.txtBTail, frmStats.txtBTailDev)
        .width = frmStats.txtBusWidth + Rnd
    Else
        .accel = NormalDis(frmStats.txtTAccel, frmStats.txtTAccelDev)
        .decel = NormalDis(frmStats.txtTDecel, frmStats.txtTDecelDev)
        .Follow = NormalDis(frmStats.txtTFollow, frmStats.txtTFollowDev)
        .gap = NormalDis(frmStats.txtTCGap, frmStats.txtTCGapDev)
        .lag = NormalDis(frmStats.txtTLag, frmStats.txtTLagDev)
        .length = frmStats.txtTLength + 2 * Rnd

        .new = True
        .speed = NormalDis(frmStats.txtTSpeed, frmStats.txtTSpeedDev)
        .speed = .speed * (22 / 15)
        .deSpeed = .speed
        .Tail = NormalDis(frmStats.txtTTail, frmStats.txtTTailDev)
        .width = frmStats.txtTWidth + Rnd
    End If

    .yield = False

End With

For debugging

Print "**************
Print "**************
If chkDeBug.Value = 1 Then
    frmDeBug.Show
End If

End Sub
Private Function NormalDis(Mean As Single, StDev As Single) As Single
A Roundabout Animation - 35
Dim Pi As Double

Pi = 3.14159265358979
If StDev = 0 Then
    NormalDis = Mean
    Exit Function
Else
    If Rnd > 0.5 Then
        NormalDis = CSng(Mean + ((2 * StDev ^ 2) * (Log(Rnd * StDev * (2 * Pi) ^ 2))))
    Else
        NormalDis = CSng(Mean - ((2 * StDev ^ 2) * (Log(Rnd * StDev * (2 * Pi) ^ 2))))
    End If
End If

End Function
Public Sub DrawCars()

Dim fbgx As Double, fbgx As Double, fendx As Double, fendy As Double, bbegx As Double, bbegy As Double, bendx As Double, bendy As Double, X As Double

For i = 1 To numCars
    If chkCarEnds.Value = 1 Then
        If myCars(i).active = True Then
            segf = myCars(i).segment
            segb = myCars(i).segmentb
            fbgx = myPts(mySegs(segf).startPt).X
            fbgx = myPts(mySegs(segf).startPt).Y
            fendx = myPts(mySegs(segf).endPt).X
            fendy = myPts(mySegs(segf).endPt).Y
            bbegx = myPts(mySegs(segb).startPt).X
            bbegy = myPts(mySegs(segb).startPt).Y
            bendx = myPts(mySegs(segb).endPt).X
            bendy = myPts(mySegs(segb).endPt).Y

            If fbgx <> fendx Then
                If fbgx < fendx Then
                    X = (fendy - fbegy) / (fendx - fbgx)
                Else
                    X = (fendy - fbegy) / (fbegx - fendx)
                End If

                theta = Atn(X)
                fnewx = fbgx + myCars(i).location
                Cos(theta)
                fnewy = fbgx + myCars(i).location
                Sin(theta)
                carFront.X = fnewx
                carFront.Y = fnewy
                Else
                    X = (fendy - fbegy) / my-
                    Segs(segf).length
                    fnewx = fbgx

                    carFront.X = fnewx
                    carFront.Y = fnewy
                End If
            End If
            Else
                X = (bendy - bbegy) / (bendx - bbegx)
                If bbegx <> bendx Then
                    X = (bendy - bbegy) / (bbegx - bendx)
                End If

                theta = Atn(X)
                bnewx = bbegx + myCars(i).location
                * Cos(theta)
                bnewy = bbegy + myCars(i).location
                * Sin(theta)
                carBack.X = bnewx
                carBack.Y = bnewy
                Else
                    X = (bendy - bbegy) / my-
                    Segs(segb).length
                    bnewx = bbegx
                    bnewy = bbegy + myCars(i).location
                * X
                carBack.X = bnewx
                carBack.Y = bnewy
            End If

            If myCars(i).new = False Then
                oldPt.X = carFront.X, old-
                Front(i).Y, 5, myCars(i).color
                oldBack(i).X, oldBack(i).Y, 5, myCars(i).color " Erase old cars
                Else
                    myCars(i).new = False
                End If

                If chkCarEnds.Value = 1 Then
                    If myCars(i).active = True Then
                        segf = myCars(i).segment
                        segb = myCars(i).segmentb
                        fbgx = myPts(mySegs(segf).startPt).X
                        fbgx = myPts(mySegs(segf).startPt).Y
                        fendx = myPts(mySegs(segf).endPt).X
                        fendy = myPts(mySegs(segf).endPt).Y
                        bbegx = myPts(mySegs(segb).startPt).X
                        bbegy = myPts(mySegs(segb).startPt).Y
                        bendx = myPts(mySegs(segb).endPt).X
                        bendy = myPts(mySegs(segb).endPt).Y

                        If fbgx <> fendx Then
                            If fbgx < fendx Then
                                X = (fendy - fbegy) / (fendx - fbgx)
                            Else
                                X = (fendy - fbegy) / (fbegx - fendx)
                            End If

                            theta = Atn(X)
                            fnewx = fbgx + myCars(i).location
                            Cos(theta)
                            fnewy = fbgx + myCars(i).location
                            Sin(theta)
                            carFront.X = fnewx
                            carFront.Y = fnewy
                            Else
                                X = (fendy - fbegy) / my-
                                Segs(segf).length
                                fnewx = fbgx
                        End If
                End If

                Else
                    If myCars(i).active = True Then
                        segf = myCars(i).segment
                        segb = myCars(i).segmentb
                        fbgx = myPts(mySegs(segf).startPt).X
                        fbgx = myPts(mySegs(segf).startPt).Y
                        fendx = myPts(mySegs(segf).endPt).X
                        fendy = myPts(mySegs(segf).endPt).Y
                        bbegx = myPts(mySegs(segb).startPt).X
                        bbegy = myPts(mySegs(segb).startPt).Y
                        bendx = myPts(mySegs(segb).endPt).X
                        bendy = myPts(mySegs(segb).endPt).Y

                        If fbgx <> fendx Then
                            If fbgx < fendx Then
                                X = (fendy - fbegy) / (fendx - fbgx)
                            Else
                                X = (fendy - fbegy) / (fbegx - fendx)
                            End If

                            theta = Atn(X)
                            fnewx = fbgx + myCars(i).location
                            Cos(theta)
                            fnewy = fbgx + myCars(i).location
                            Sin(theta)
                            carFront.X = fnewx
                            carFront.Y = fnewy
                            Else
                                X = (fendy - fbegy) / my-
                                Segs(segf).length
                                fnewx = fbgx
                        End If
                End If

            End If

        End If
    End If
End Sub
If fbegx <> fendx Then
  X = (fendy - fbegy) / (fendx - fbegx)
  theta = Atn(X)
  If fbegx < fendx Then
    fnewx = fbegx + myCars(i).location
    * Cos(theta)
    fnewy = fbegy + myCars(i).location
    * Sin(theta)
  Else
    fnewx = fbegx - myCars(i).location
    * Cos(theta)
    fnewy = fbegy - myCars(i).location
    * Sin(theta)
  End If
End If

carFront.X = fnewx
carFront.Y = fnewy
Else
  X = (fendy - fbegy) / my-
  Segs(segf).length
  fnewx = fbegx
  fnewy = fbegy + myCars(i).location
  * X
  carFront.X = fnewx
carFront.Y = fnewy
End If

If bbegx <> bendx Then
  X = (bendy - bbegy) / (bendx - bbegx)
  theta = Atn(X)
  If bbegx < bendx Then
    bnewx = bbegx + my-
    Cars(i).location
    * Cos(theta)
    bnewy = bbegy + my-
    Cars(i).location
    * Sin(theta)
  Else
    bnewx = bbegx - my-
    Cars(i).location
    * Cos(theta)
    bnewy = bbegy - my-
    Cars(i).location
    * Sin(theta)
  End If
  carBack.X = bnewx
carBack.Y = bnewy
Else
  X = (bendy - bbegy) / my-
  Segs(seg).length
  bnewx = bbegx
  bnewy = bbegy + myCars(i).location
  * X
  carBack.X = bnewx
carBack.Y = bnewy
  If myCars(i).new = False Then
    pectPix.DrawWidth = myCars(i).width
    If myCars(i).new = False Then
      pectPix.Line (oldFront(i).X, old-
      Front(i).Y), (oldBack(i).X, oldBack(i).Y), my-
      Cars(i).color
      Erase old cars
    Else
      myCars(i).new = False
    End If
    pectPix.Line (carFront.X, carFront.Y)-
    (carBack.X, carBack.Y), myCar(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
End If
Next i
End Sub

Public Sub FindFrontCar(i As Integer)

  Dim thisseg2 As Integer, nextseg2 As Integer
  Dim frontcarfront As Integer 'used to okk for a
  front of cars
  backcar = i
  thisseg2 = myCars(backcar).segment
  nextseg2 = myCars(backcar).nextseg
  gap = mySegs(thisseg2).length - my-
  Cars(backcar).location
  Do
    frontcar = mySegs(nextseg2).CarsIn(1)
    frontcarfront = mySegs(nextseg2).CarsIn(2)
    If frontcar <> 0 Then
      gap = gap + myCars(frontcar).locationb
      Exit Do
    ElseIf frontcarfront <> 0 Then
      Exit Do 'This stops gap where the next
      segment starts.
      This is because of a car merging.
      Else
        gap = gap + mySegs(nextseg2).length
        End If
        thisseg2 = nextseg2
        If mySegs(thisseg2).nextSegR = 0 Then
          nextseg2 = mySegs(thisseg2).nextSegL.
          ElseIf myCars(backcar).exit = my-
          Segs(thisseg2).nextSegR Then
            nextseg2 = mySegs(thisseg2).nextSegR
            Else
          nextseg2 = mySegs(thisseg2).nextSegL
          End If
          If nextseg2 = 0 Then gap = 4 * allowedGap +
          gap
          Loop Until gap > allowedGap Or thisseg2 >= 99
          End Sub
          Begin VB.Form frmDeBug

          Private Sub Form_Load()
            Dim i As Integer
            msgDeBug.Row = 0
msgDeBug.Col = 0
msgDeBug.Text = "Car #"
msgDeBug.Col = 1
msgDeBug.Text = "Type"
msgDeBug.Col = 2
msgDeBug.Text = "Color"
msgDeBug.Col = 3
msgDeBug.Text = "Active"
msgDeBug.Col = 4
msgDeBug.Text = "DeSpeed"
msgDeBug.Col = 5
msgDeBug.Text = "Speed"
msgDeBug.Col = 6
msgDeBug.Text = "Segf"
msgDeBug.Col = 7
msgDeBug.Text = "Segb"
msgDeBug.Col = 8
msgDeBug.Text = "LoCF"
msgDeBug.Col = 9
msgDeBug.Text = "LoCB"
msgDeBug.Col = 10
msgDeBug.Text = "Length"
msgDeBug.Col = 11
msgDeBug.Text = "Drawn Len"
msgDeBug.Col = 0
For i = 0 To 15
  msgDeBug.Row = i
  msgDeBug.Text = i
Next i
End Sub

Private Sub txtCar_Change()
  txtCar.Text = 100 - txtBus.Text - txtTruck.Text
End Sub

Private Sub txtTruck_Change()
  If txtTruck.Text = "" Then txtTruck.Text = 0
  txtCar.Text = 100 - txtBus.Text - txtTruck.Text
End Sub

Attribute VB_Name = "frmOpt"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub Command1_Click()
  frmOpt.Hide
  frmNewPts.Enabled = True
  frmNewPts.Show
End Sub

Private Sub Form_Terminate()
  frmOpt.Hide
  frmNewPts.Enabled = True
  frmNewPts.Show
End Sub

Private Sub Form_Unload(Cancel As Integer)
  frmOpt.Hide
  frmNewPts.Enabled = True
  frmNewPts.Show
End Sub

Attribute VB_Name = "frmNewPts"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Private Type Point
  X As Single
  Y As Single
End Type

Private Type path
  segment(40) As Integer
totalseg As Integer
End Type

Private Type Routs
  path(7) As path
  numpath As Integer
End Type

Private Type Way
  StartSeg As Integer
  ExitSeg As Integer
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(6) 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 Integer
    right As Integer
    straight As Integer
    ut As Integer
    direction As String
End Type

Private BI As Long, Rd As Long, Gn As Long, Be As Long, Wt As Long
Private ptN As Integer, first As Integer, second
As Integer 'is point counter
Private segN As Integer
Private myPts(100) As Point 'array of points
Private mySegs(100) As segment 'array of segments
Private myStarts(10) As Integer, myStartCount As Integer
Private myEnds(10) As Integer, myEndCount As Integer
Private totalPaths As Integer, GrandTotal As Integer
Private picName As String, path As String
Private starts(5) As Integer 'Holds up to 5 starting segment values
Private drive(6) As Routs 'Paths of traffic
Private Stoppoint As Integer 'Helps track of splits
Private startnum As Integer 'Number of starting segments
Private startname(6) As String 'Name of starting segment
Private endnum As Integer 'Number of ending segments
Private endname(6) As String 'Name of ending segment
Private exit(42) As Integer 'Holds the exit numbers
Private Start(6, 6) As Way
Private e As Integer, d As Integer, c As Integer, w As Integer, pathnum As Integer
Dim SetScale As Boolean, scalarum As Single
Dim x1 As Single, x2 As Single, y1 As Single, y2 As Single
Dim Wrong(42) As Integer
Dim RoundFile As String 'Path to the saved files

Private Sub btnBack_Click()
    frmNewPts.Hide
    frmMainMenu.Show
End Sub
Private Sub btnCreate_Click()

    Dim X As Integer 'code for the creation of the SegSet file.
    File is used for .nextSegR, .nextSegL,
    .leftSegs(1), .leftSegs(2)
    'current state WORKING.
    If scalarum = 0 Then
        X = MsgBox("You must set the scale before doing this.", vbOKOnly, "Error")
    Exit Sub
End If

Call save
Call MakeSetSet
Call MakePaths

End Sub
Private Sub MakeSetSet()

    Dim i As Integer, j As Integer, k As Integer
    Dim r As Integer, s As Integer, t As Integer, z As Integer
    Dim ans As String
    Dim a1 As Integer, b1 As Integer, a2 As Integer, b2 As Integer

    Cls
    Call Redraw

    For i = 1 To 6
        For j = 1 To drive(i).numPath
            For k = 1 To drive(i).path(j).totalSeg - 1
                If my-
                    Segs(drive(i).path(j).segment(k)).endPt = my-
                    Segs(drive(i).path(j).segment(k + 1)).startPt
                    Then
                        r = 0
                        For s = 1 To 42
                            If exit(s) =
                                drive(i).path(j).segment(k + 1) Then
                                    r = r + 1
                                    Next s
                        If r = 1 Then
                            my-
                            Segs(drive(i).path(j).segment(k)).nextSegR =
                            drive(i).path(j).segment(k + 1)
                            ElseIf r = 0 Then
                                my-
                                Segs(drive(i).path(j).segment(k)).nextSegL =
                                drive(i).path(j).segment(k + 1)
                            End If
                    End If
                End If
            End For
        Next j
    Next i
If Wrong(t) =
   drive(i).path(j).segment(k) Then
      Call yield(i, j, k, r)
      Exit For
   ElseIf Wrong(t) = r Then
      Call yield2(i, j, k, r)
      Exit For
   ElseIf Wrong(t) = 0 Then
      a1 =
      myPts(mySegs(r).startPt).X -
      myPts(mySegs(r).endPt).X
      a2 =
      myPts(mySegs(r).startPt).Y -
      myPts(mySegs(r).endPt).Y
      b1 =
      myPts(mySegs(drive(i).path(j).segment(k)).startPt).X -
      myPts(mySegs(drive(i).path(j).segment(k)).endPt).X
      b2 =
      myPts(mySegs(drive(i).path(j).segment(k)).startPt).Y -
      myPts(mySegs(drive(i).path(j).segment(k)).endPt).Y
      If (b1 * a2) - (b2 * a1) > 0
      Then
         Wrong(t) = r
         Call yield2(i, j, k, r)
         Exit For
      ElseIf (b1 * a2) - (b2 * a1) < 0
      Then
         Wrong(t) =
         drive(i).path(j).segment(k)
         Call yield(i, j, k, r)
         Exit For
      End If
   End If
Next t
End If
Next r
Next k
Next j
Next i

Call FindLefts  ' calls the code that will find
the 6 leftsegs

Open roundfile + "set" For Output As #3
Write #3, scalenum

For i = 1 To segN
   With mySegs(i)
      Write #3, .nextSegL, .nextSegR, .leftSegs(1),
      .leftSegs(2), .leftSegs(3), .leftSegs(4),
      .leftSegs(5), .leftSegs(6)
   End With
Next i
End If
Next j
Next i

End Sub

Private Sub btnOptions_Click()

frmNewPts.Enabled = False
frmOpt.Show

End Sub

Private Sub btnRemove_Click()

'Allows user to remove a desired path.
Current state WORKING.

Dim i As Integer, j As Integer, k As Integer, r As Integer
Dim pickpath As Integer, count As Integer

pickpath = txtRemove.Text

If Grandtotal = 0 Then
cmdShowPath.Enabled = False
btnRemove.Enabled = False
i = MsgBox("No more paths available.", vbOKOnly, "Error")
Exit Sub
ElseIf pickpath > Grandtotal Then
i = MsgBox("There is not a path with that number.", vbOKOnly, "Error")
Exit Sub
End If

For i = 1 To 6
count = count + drive(i).numpath
If count >= pickpath Then
pickpath = pickpath - (count - drive(i).numpath)
 Exit For
End If
Next i

For j = pickpath To drive(i).numpath - 1
 For k = 1 To drive(i).path(j) + 1.totalseg
 drive(i).path(j).segment(k) = drive(i).path(j + 1).segment(k)
 Next k
 If drive(i).path(j).totalseg > drive(i).path(j + 1).totalseg Then
 For k = drive(i).path(j + 1).totalseg To drive(i).path(j).totalseg
 drive(i).path(j).segment(k) = 0
 Next k
 End If
 Next j

j = drive(i).numpath
For k = 1 To drive(i).path(j).totalseg

Next k

drive(i).numpath = drive(i).numpath - 1

Grandtotal = Grandtotal - 1
lblPath.Caption = Grandtotal

End Sub

Private Sub Clear()

'clears all segments, points, paths.

Erase myPts()
Erase mySegs
Erase drive
Erase exits
Grandtotal = 0
segN = 0
ptN = 0
lblPath.Caption = Grandtotal
Redraw

End Sub

Private Sub btnScale_Click()

Dim X As Integer

petPix.Cls
X = MsgBox("Pick two points that you know the distance of on the drawing.", vbOKOnly, "Set Scale")
SetScale = True
x1 = 1000
x2 = 1000
y1 = 0
y2 = 0

frmNewPts.btnCreate.Enabled = True
End Sub

Private Sub cmdClear_Click()

Call Clear

End Sub

Private Sub cmdFindPaths_Click()

Dim i As Integer, j As Integer

'New code for the ultimate finding of all possible paths.

Call FindStarts
Call FindPaths
lblPath.Caption = Grandtotal
i = MsgBox("Done. ", vbOKOnly)
btnDir.Enabled = True
Call CountEnds
cmdShowPath.Enabled = True
btnRemove.Enabled = True

End Sub

Private Sub FindPaths()
code to finds the paths in a circle.
Current state is sorta WORKING.

Dim doneflag As Boolean, Turn As Boolean
Dim i As Integer, k As Integer, j As Integer, l As Integer, t As Integer, s As Integer
Dim counter As Integer, r As Integer, q As Integer, counterX As Integer, z As Integer
Dim answer As String
Dim dummy As Rout
Dim a1 As Integer, b1 As Integer, a2 As Integer, b2 As Integer

r = 0
Grandtotal = 0
doneflag = False
Turn = False

For i = 1 To startnum
    k = 0
    Do
        k = k + 1
        If k = 1 Then Turn = False
        j = 1
        drive(i).path(k).segment(j) = starts(i)
doneflag = False
        If Turn = True Then
            l = Stoppoint - 1
            j = j + 1
            Turn = False
        End If
    End Do
    l = j
    j = j + 1
    counter = 0
    For t = 1 To segN
        If mySegs(drive(i).path(k).segment(l)).endPt = mySegs(t).startPt Then
            counter = counter + 1
            If counter = 1 Then
                drive(i).path(k).segment(j) = t
            ElseIf counter <> 0 Then
                For s = 1 To j - 1
                    drive(i).path(k + 1).segment(s) = drive(i).path(k).segment(s)
                Next s
                Stoppoint = j
                Turn = True
                For q = 1 To 42
                    If exits(q) = t Then
                        drive(i).path(k + 1).segment(j) = drive(i).path(k).segment(j)
                    End If
                Next q
            End If
        End If
    End For
    a1 = myPts(mySegs(t).startPt).X - myPts(mySegs(t).endPt).X
    a2 = myPts(mySegs(t).startPt).Y - myPts(mySegs(t).endPt).Y
    b1 = myPts(mySegs(drive(i).path(k).segment(j)).startPt).X - myPts(mySegs(drive(i).path(k).segment(j)).endPt).X
    b2 = myPts(mySegs(drive(i).path(k).segment(j)).startPt).Y - myPts(mySegs(drive(i).path(k).segment(j)).endPt).Y
    If (b1 * a2) - (b2 * a1) > 0 Then
        exits(r) = drive(i).path(k).segment(j)
        drive(i).path(k + 1).segment(j) = t
    ElseIf (b1 * a2) - (b2 * a1) < 0 Then
        exits(r) = t
        drive(i).path(k + 1).segment(j) = drive(i).path(k).segment(j)
    End If
End If
Exit For
End If
Next t
If counter = 0 Then doneflag = True
Loop While doneflag = False
j = j - 1
drive(i).path(k).totalseg = j
For s = 1 To j
    counterX = 0
    For q = 1 To j
        If mySegs(drive(i).path(k).segment(s)).endPt = mySegs(drive(i).path(k).segment(q)).endPt Then
            counterX = counterX + 1
        End If
    Next q
    If counterX > 0 Then
        exit(q) = true
        drive(i).path(k).segment(j) = drive(i).path(k).segment(j)
    End If
Next s
For s = 1 To 42
    If exits(s) = t Then
        drive(i).path(k + 1).segment(j) = drive(i).path(k).segment(j)
    End If
Next s

A Roundabout Animation - 43
Next q
  If counterx > 1 Then
    drive(i).numpath = k - 1
    Grandtotal = Grandtotal +
  End If
  Next s
End Sub

Private Sub FindStarts()
  'code to find the starting segs for all paths.
  'Current state is WORKING.

Dim i As Integer, j As Integer, k As Integer
Dim TotalStart As Integer 'total of segs with this start point.

k = 0
For i = 1 To segN
  TotalStart = 0
  For j = 1 To segN
    If mySegs(i).startPt = mySegs(j).endPt
      Then TotalStart = TotalStart + 1
  End If
  Next j
  If TotalStart = 0 Then
    k = k + 1
    starts(k) = i
  End If
  Next i

startnum = k

For i = 1 To k
  If starts(i) = 0 Then startnum = startnum - 1
  Next i
End Sub

Private Sub CountEnds()

Dim i As Integer, j As Integer, k As Integer
Dim ends(6) As Integer

denum = 0

For i = 1 To 6
  For j = 1 To drive(i).numpath
    If k = 1 To 6
      If ends(k) =
        drive(i).path(j).segment(drive(i).path(j).totalseg)
        Then
        Exit For
        ElseIf ends(k) = 0 Then
        endnum = endnum + 1
        End If
        Next k
  End Sub

Private Sub cmdLoadPic_Click()

Dim i As Integer, t As Integer, f As Integer, o As Integer
Dim roundname As String

On Error GoTo ErrorHandler
frmNewPts.cdbCircleDialogTitle = "Choose Background Diagram"
frmNewPts.cdbCircle.FilterIndex = 1
frmNewPts.cdbCircle.FileName = App.path
frmNewPts.cdbCircle.ShowOpen
roundname = cdbCircle.FileName
frmNewPts.pctPix.Picture = LoadPicture(roundname)
i = InStr(1, roundname, ".")
roundfile = Left(roundname, i)
Call SetCap

picname = roundname

do
  t = 0
  t = InStr(1, picname, ",")
  picname = Right(picname, Len(picname) - t)
Loop While t <> 0
Exit Sub
End Sub

ErrorHandler:
  'User pressed Cancel button.
  Exit Sub
End Sub

Private Sub cmdQuit_Click()

End Sub

Private Sub cmdReadExistingSetup_Click()

Dim i As Integer, roundname As String
On Error GoTo ErrorHandler
Call Clear

roundname = ""
cdbOpen.FileName = App.path
cdbOpen.ShowOpen
roundname = cdbOpen.FileName
i = InStr(1, roundname, ",")
roundfile = Left(roundname, i)
Open roundfile + "pts" For Input As #1
Open roundfile + "seg" For Input As #2
Input #1, picname
frmNewPts.pctPix.Picture = LoadPicture(App.path + "," + picname)

End Sub
i = 0
Do While Not EOF(1)
    i = i + 1
    Input #1, myPts(i).X, myPts(i).Y
Loop
ptN = i
i = 0
Do While Not EOF(2)
    i = i + 1
    Input #2, mySegs(i).startPt, mySegs(i).endPt
Loop
segN = i
Close #1: Close #2
Call Redraw
Call SetCap
Exit Sub

ErrorHandler:
    Exit Sub
End Sub
Private Sub cmdSaveSetup_Click()
    Call save
End Sub
Private Sub save()
Dim i As Integer, roundname As String
On Error GoTo ErrorHandler
    cdbOpen.ShowSave
    roundname = cdbOpen.FileName
    i = InStr(1, roundname, ",")
    roundfile = Left(roundname, i)
Call SetCap
    Open roundfile + ",pts" For Output As #1
    Open roundfile + ",seg" For Output As #2
    Write #1, "pmcname"
    Write #1, myPts(i).X, myPts(i).Y
    Next i
    For i = 1 To segN
        Write #1, mySegs(i).startPt, mySegs(i).endPt
    Next i
    Close #1: Close #2
Exit Sub
ErrorHandler:
    Exit Sub
End Sub
Private Sub SetCap()
Dim i As Integer
Dim roundnamex As String
roundnamex = roundfile
    Do
        i = 0
        i = InStr(1, roundnamex, ";")
        roundnamex = Right(roundnamex, Len(roundnamex) - i)
    Loop While i <> 0
    roundnamex = Left(roundnamex, Len(roundnamex) - 1)
    frmNewPts.Caption = ";Roundabout setup:" + roundnamex
End Sub
Private Sub cmdShowPath_Click()
    'Shows any requested path, mostly for debugging.
    'current state WORKING
    Dim i As Integer, j As Integer, k As Integer
    Dim pickpath As Integer, count As Integer
    pickpath = txtShow.Text
    For i = 1 To 6
        count = count + drive(i).numpath
        If count >= pickpath Then
            pickpath = pickpath - (count - drive(i).numpath)
            Exit For
    End If
Next i
    For j = 1 To 7
        If j = pickpath Then
            For k = 1 To drive(i).path(j).totalseg
                If frmOpt.optRedPath.Value = True Then
                    frmNewPts.pctPix.ForeColor = Red
                If frmOpt.optBluePath.Value = True Then
                    frmNewPts.pctPix.ForeColor = Be
                If frmOpt.optGreenPath.Value = True Then
                    frmNewPts.pctPix.ForeColor = Bl
                If frmOpt.optBlackPath.Value = True Then
                    frmNewPts.pctPix.ForeColor = Black
                Next k
                (myPts(mySegs(drive(i).path(j).segment(k))).startPt).X =
                (myPts(mySegs(drive(i).path(j).segment(k))).startPt).Y =
                (myPts(mySegs(drive(i).path(j).segment(k))).endPt).X =
                (myPts(mySegs(drive(i).path(j).segment(k))).endPt).Y =
                If k = 1 Then
                    pctPix.CurrentX =
                End If
                myPts(mySegs(drive(i).path(j).segment(k))).startPt).X =
                myPts(mySegs(drive(i).path(j).segment(k))).endPt).X =
                myPts(mySegs(drive(i).path(j).segment(k))).startPt).Y =
                myPts(mySegs(drive(i).path(j).segment(k))).endPt).Y Then
            Next k
    Next i
pctPix.CurrentY = myPts(mySegs(drive(i).path(j).segment(k)).startPt).Y + 2
Else
  pctPix.CurrentY = myPts(mySegs(drive(i).path(j).segment(k)).endPt).Y + 2
End If
pctPix.Print "Enter"
ElseIf k = drive(i).path(j).totalseg Then
  pctPix.CurrentX = (myPts(mySegs(drive(i).path(j).segment(k)).startPt).X + myPts(mySegs(drive(i).path(j).segment(k)).endPt).X) / 2
  If myPts(mySegs(drive(i).path(j).segment(k)).startPt).Y > myPts(mySegs(drive(i).path(j).segment(k)).endPt).Y Then
    pctPix.CurrentY = myPts(mySegs(drive(i).path(j).segment(k)).startPt).Y + 2
  Else
    pctPix.CurrentY = myPts(mySegs(drive(i).path(j).segment(k)).endPt).Y + 2
    End If
  End If
Next k
Exit For
End If
Next j

End Sub
Private Sub btRedraw_Click()

Call Redraw
End Sub
Private Sub Form_Load()

frmNewPts.width = 7350
frmNewPts.pctPix.Scale (-200, 200)-(200, -200)
set user scale
BL = RGB(0, 0, 0)
Rd = RGB(255, 0, 0)
Gn = RGB(0, 255, 0)
Be = RGB(0, 255, 255)
Wt = RGB(255, 255, 255)
frmNewPts.pctPix.DrawMode = 10 'Not xor - same as the clock problem
frmNewPts.pctPix.DrawStyle = 0 'Solid line
ptN = 0
End Sub
Private Sub Redraw()

Dim i As Integer
frmNewPts.pctPix.Cls

If frmOpt.chkPoint.Value = 1 Then
  If frmOpt.optGreenPoint.Value = True Then
    frmNewPts.pctPix.ForeColor = Gn
  End If
  If frmOpt.optBluePoint.Value = True Then
    frmNewPts.pctPix.ForeColor = Rd
  End If
  If frmOpt.optBlackpoint.Value = True Then
    frmNewPts.pctPix.ForeColor = Be
  End If
  If frmOpt.optRedPoint.Value = True Then
    frmNewPts.pctPix.ForeColor = Bl
  End If
  For i = 1 To ptN
  Next i
End If

If frmOpt.chkPNum.Value = 1 Then
  If frmOpt.optGreenPNum.Value = True Then
    frmNewPts.pctPix.ForeColor = Gn
  End If
  If frmOpt.optBluePNum.Value = True Then
    frmNewPts.pctPix.ForeColor = Rd
  End If
  If frmOpt.optBlackPNum.Value = True Then
    frmNewPts.pctPix.ForeColor = Be
  End If
  If frmOpt.optRedPNum.Value = True Then
    frmNewPts.pctPix.ForeColor = Bl
  End If
  For i = 1 To ptN
    frmNewPts.pctPix.CurrentY = myPts(i).Y
    frmNewPts.pctPix.Print i
  Next i
End If

If frmOpt.chkSeg.Value = 1 Then
  If frmOpt.optBlackSeg.Value = True Then
    frmNewPts.pctPix.ForeColor = Be
  End If
  If frmOpt.optRedSeg.Value = True Then
    frmNewPts.pctPix.ForeColor = Bl
  End If
  If frmOpt.optBlueSeg.Value = True Then
    frmNewPts.pctPix.ForeColor = Rd
  End If
  If frmOpt.optGreenSeg.Value = True Then
    frmNewPts.pctPix.ForeColor = Gn
  End If
  For i = 1 To segN
    pctPix.Line (myPts(mySegs(i).startPt).X, myPts(mySegs(i).startPt).Y) - _
                 (myPts(mySegs(i).endPt).X, myPts(mySegs(i).endPt).Y)
  Next i
End If
If first = 0 Then
  For i = 1 To ptN
    If (Abs(X - myPts(i).X) < 6 And Abs(Y - myPts(i).Y) < 6) Then
      first = i
      pctPix.FillStyle = 1
      pctPix.Circle (myPts(i).X, myPts(i).Y), 6, Rd
      Exit For
    End If
  Next i
End If
Else
  For i = 1 To ptN
    If (Abs(X - myPts(i).X) < 6 And Abs(Y - myPts(i).Y) < 6) Then
      second = i
      segN = segN + 1
      mySegs(segN).startPt = first
      mySegs(segN).endPt = second
      Call Redraw
      first = 0
      second = 0
      Exit For
    End If
  Next i
End If
ElseIf optMove = True Then
  If first = 0 Then
    For i = 1 To ptN
      If (Abs(X - myPts(i).X) < 6 And Abs(Y - myPts(i).Y) < 6) Then
        first = i
        pctPix.FillStyle = 1
        pctPix.Circle (myPts(i).X, myPts(i).Y), 8, Gn
        Exit For
      End If
    Next i
  End If
  Else
    If myPts(first).X = X
       myPts(first).Y = Y
       Call Redraw
       first = 0
    End If
ElseIf optRemoveSeg = True Then
  Dim dum(5) As Integer, j As Integer, k As Integer
  For i = 1 To ptN
    If (Abs(X - myPts(i).X) < 6 And Abs(Y - myPts(i).Y) < 6) Then
      first = i
      pctPix.FillStyle = 1
      pctPix.Circle (myPts(i).X, myPts(i).Y), 8, Be
      Exit For
    End If
  Next i
  For i = 1 To segN
    If mySegs(i).endPt = first Then
      j = j + 1
      dum(j) = i
    End If
  Next i
  For i = j To 1 Step -1

For k = dum(i) To segN - 1
    mySegs(k) = mySegs(k + 1)
Next k
segN = segN - 1
Next i
first = 0
Call Redraw
End If
End If
End Sub
Private Sub yield(i As Integer, j As Integer, k As Integer, r As Integer)

Dim s As Integer

For s = 1 To segN
    If mySegs(r).startPt = mySegs(s).endPt And r <> s And (mySegs(s).nextSegL = r Or mySegs(s).nextSegR = r)
        mySegs(drive(i).path(j).segment(k).leftSegs(1) = r)
        mySegs(drive(i).path(j).segment(k).leftSegs(2) = s)
        mySegs(drive(i).path(j).segment(k - 1).leftSegs(1) = s)
        mySegs(drive(i).path(j).segment(k - 1).leftSegs(2) = s)
        Next s
    End If
End Sub

Private Sub yield2(i As Integer, j As Integer, k As Integer, r As Integer)

Dim s As Integer, z As Integer

For s = 1 To segN
    If mySegs(drive(i).path(j).segment(k)).startPt = mySegs(s).endPt And drive(i).path(j).segment(k) <> s And (mySegs(s).nextSegL = drive(i).path(j).segment(k) Or mySegs(s).nextSegR = drive(i).path(j).segment(k))
        mySegs(r).leftSegs(1) = drive(i).path(j).segment(k)
        mySegs(r).leftSegs(2) = s
        For z = 1 To segN
            If mySegs(z).nextSegL = r Or mySegs(z).nextSegR = r
                mySegs(z).leftSegs(1) = drive(i).path(j).segment(k)
                mySegs(z).leftSegs(2) = s
                Next z
            End If
        Next s
End Sub

Private Sub RemovePoint(i As Integer)

Dim j As Integer
For j = i To ptN - 1
    myPts(j).X = myPts(j + 1).X
Next j
ptN = ptN - 1
End Sub

Private Sub FindLefts()

Dim a As Integer, b As Integer, c As Integer, d As Integer, e As Integer, f As Integer, g As Integer, h As Integer
Dim sum As Integer, back(2) As Integer

For a = 1 To segN
    sum = 0
    If mySegs(a).leftSegs(1) <> 0 Then
        b = mySegs(a).leftSegs(1)
        c = mySegs(a).leftSegs(2)
        For d = 1 To segN
            If mySegs(c).startPt = mySegs(d).endPt
                sum = sum + 1
                back(sum) = d
            End If
        Next d
        If sum > 1 Then 'code to see which segment is in the circle
            For h = 1 To 42
                If back(h) = Wrong(h) Then
                    d = back(h)
                    Exit For
                ElseIf back(h) = Wrong(h) Then
                    d = back(1)
                    Exit For
                End If
            Next h
        End If
        For e = 1 To segN
            If mySegs(d).startPt = mySegs(e).endPt
                sum = sum + 1
                back(sum) = e
            End If
        Next e
        If sum > 1 Then 'code to see which segment is in the circle
            For h = 1 To 42
                If back(h) = Wrong(h) Then
                    e = back(h)
                    Exit For
                ElseIf back(h) = Wrong(h) Then
                    e = back(1)
                    Exit For
                End If
            Next h
        End If
    Else
        e = back(1)
        End If
        sum = 0
        For f = 1 To segN
            If mySegs(e).startPt = mySegs(f).endPt
                sum = sum + 1
            End If
        Next f
    End If
End Sub
back(sum) = f
End If
Next f
If sum > 1 Then 'code to see which segment is in the circle
    For h = 1 To 42
        If back(1) = Wrong(h) Then
            f = back(2)
            Exit For
        ElseIf back(2) = Wrong(h) Then
            f = back(1)
            Exit For
        End If
    Next h
Else
    f = back(1)
End If
sum = 0
For g = 1 To segN
    If mySegs(f).startPt = mySegs(g).endPt
        Then
            sum = sum + 1
            back(sum) = g
        End If
    Next g
If sum > 1 Then 'code to see which segment is in the circle
    For h = 1 To 42
        If back(1) = Wrong(h) Then
            g = back(2)
            Exit For
        ElseIf back(2) = Wrong(h) Then
            g = back(1)
            Exit For
        End If
    Next h
Else
    g = back(1)
End If
sum = 0
With mySegs(a)
    .leftSegs(3) = d
    .leftSegs(4) = e
    .leftSegs(5) = f
    .leftSegs(6) = g
End With
End If
Next a
End Sub

Private Sub Form_Unload(Cancel As Integer)
frmPath.Hide
frmNewPts.Enabled = True
frmNewPts.Show
End Sub

Attribute VB_Name = "frmMainMenu"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub btnQuit_Click()
End
End Sub

Private Sub btnRoundSetUp_Click()
frmNewPts.Show
frmMainMenu.Hide
End Sub

Private Sub Command2_Click()
frmSim.Show
frmMainMenu.Hide
End Sub

Attribute VB_Name = "Form5"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub cmdHide_Click()
Form5.Hide
End Sub
Private Sub Form_Load()
Dim startnum As Integer, endnum As Integer
Dim startname(6) As String, endname(6) As String
Dim i As Integer, j As Integer
End Sub

Private Sub Form_Initialize()
frmPath.Hide
frmNewPts.Enabled = True
frmNewPts.Show
End Sub

Private Sub Form_Terminate()
frmPath.Hide
frmNewPts.Enabled = True
End Sub

A Roundabout Animation - 49
Private Sub cmdOK_Click()
   Form4.Hide
End Sub
Private Sub Form_Load()
   Dim i As Integer, X As Integer
   grdInput.ColWidth(0) = grdInput.ColWidth(0) * 1.1
End Sub
Private Sub grdInput_Click()
   Dim usrinput As String
   grdInput.Col = grdInput.MouseCol
   grdInput.Row = grdInput.MouseRow
   usrinput = InputBox("What is the new number?", "Input Number")
   grdInput.Text = usrinput
End Sub
Private Sub grdInput_KeyPress(KeyAscii As Integer)
   Dim usrinput As String
   usrinput = InputBox("What is the new number?", "Input Number")
   grdInput.Text = usrinput
End Sub
End Sub
grdInput.Text = usrinput
End Sub
Attribute VB_Name = "Form3"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub cmdhide_Click()
   Form3.Hide
End Sub
Private Sub cmdVPH_Click()
   Label10.Caption = "Vehicles per Hour"
   For i = 1 To frmSim.startnum
      grdVPH.Row = i
      For j = 1 To frmSim.endnum + 1
         grdVPH.Col = j
         If i = 5 And j = 5 Then Exit For
         grdVPH.Text = Format((CSng(grdVPH.Text) * 60) / (CSng(Form4.txtRTime.Text)), "###.0")
      Next j
   Next i
   cmdVPH.Enabled = False
End Sub