**Requirements**

* Unity (Latest version)
* SUMO 0.23 (<http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/>)

**The Project**

The main task was to take simulations generated from SUMO and recreate them in Unity to create 3D simulations with more visual appeal and allow for user interaction. To give the scene camera, or user, a sense of looking down at the simulation all y-coordinates provided from SUMO are used on the z-axis of Unity. This is so that the cars stay on 1 level and are not flying upwards out of the camera’s view. Provided in this Unity project are the following:

* Two scenes created to test our ideas and develop 3D SUMO simulations
* The scripts developed and used to read data and generate cars
* A prefab for displaying a well-placed camera over a Google Map display
* Resource objects used in scripts to dynamically generate cars in the scene

**Developed Scenes**

Scene 1 – sumoConcept

The original program used to create scripts, cars and test the data coming out of SUMO. The road was simply added later using 3D cubes to give a more visual look while testing. An intersection was chosen as it demonstrates the core concepts of car simulations, speed, stopping, and turning. Not only this, but we are able to see all directions of car movement at once. The XML data used for this was obtained through the cross1ltl example provided in the SUMO examples archive located at <http://www.dlr.de/ts/en/Portaldata/16/Resources/projekte/sumo/sumo-doc-0.24.0.zip>.

The information used for car positions is contained in this project package under the name “crossVehicles”. To run this scene, simply replace the current path in XMLReader1 with the directoryURL to crossVehicles.

Scene 2 – newSUMO

The second version of the program now incorporates Google Map imaging to generate the roads and visuals. The Google Maps portion of this scene was accomplished using an asset located on the Unity’s asset store found here, <https://www.assetstore.unity3d.com/en/#!/content/3573>. Rather than use crossVehicles for the data, this scene uses a combination of real world road networks provided from the OpenStreetMaps website and randomly generated vehicle routes created using SUMO’s randomTrips.py code. The OSM Tutorial document provides an explanation on how to generate and implement data from these two sources.

The intersection currently being displayed in this scene is Komoka Road and Glendon Drive. In the format used for this asset, that would be the longitude and latitude, -81.43135 and 42.945. At the current state of this scene, a zoom level of 17 is preferred as it displays enough of the image without going too far out to make the intersection, and roads, too small.

This simulation currently uses the car positions found in the file “carData”. To run this scene, replace the current path in XMLReader2 with the location of carData.

Scene 3 – sumoMiddlesex

The third scene included in this package demonstrates everything incorporated into one simulation on Western’s own intersection of Perth Drive and University Drive (-81.26993, 43.00801). As shown, this scene demonstrates the issue involved in using real world imaging (as noted in Outstanding Tasks). Notice how the cars appear to turn at the intersection accordingly, however at the Delaware Hall drive thru we can see cars going off the road due to OSM’s data of the roads. There are some cars acting as though the side-walks are roads as well, this may be due to how OSM code’s a sidewalk.

This simulation uses the data found in the file carDataMiddlesex. To run this scene, the path in XMLReader3 must reflect the location of this data file.

**The Scripts Used**

XMLReader

Attached to each scene’s camera, is a script responsible for reading the passed in generated XML files, crossVehicles or carData. Again, from the OSM Tutorial document these files can be generated from the SUMO command:

* sumo –c map.sumo.cfg --fcd-output *FILE\_NAME\_HERE*

Once loaded, the reader iterates through every node named “timeStep”, each of these nodes represents a second of the simulation and contains the data of every car element that is alive that second. When the reader discovers a car in a timestep, it first checks if this car’s ID is already active in the scene. Depending on this outcome, the program does one of the following:

|  |  |
| --- | --- |
| **ID Found** | **ID Not Found** |
| * Locates the car in the scene * Update the car’s properties through its script CarProperties (*see below*) | * Instantiate a new car GameObject with this newly found ID * Store the car’s data within * Place it in the scene based on the XML data * Add it to the list of cars in the scene |

If any active cars are not visited by the end of a timeStep, they are destroyed and removed from the program’s scene. This is done to lower the amount of GameObjects active in the scene at any time and to emulate the SUMO simulation as it also does not display inactive cars.

The reader continues this process until it no longer has any timestep nodes to read, AKA it has reached the end of the XML file. The end of the XML file also represents the end of the SUMO simulation.

CarProperties

This script is attached to every car GameObject that is currently in the scene. Its purpose is to contain all data, relevant to this program, which comes from a car element inside a timestep:

* Id
* X
* Y
* Angle
* Speed

When the ID of an existing car is found in the reader, that car is then located in the scene and its data is then updated with its new coordinates (x, y) and new statuses (angle, speed). Prior to this update however, the orientation of the car is altered to face (turn) towards its next destination. This movement is to give the sense of the car actually driving so that once it’s completed a turn at an intersection, it is now completely facing 90 degrees to the left or right. As of now the speed of a car is not used as there were issues trying to smoothly move cars between points (See Outstanding Tasks for more detail).

**Outstanding Tasks/Ideas**

Smooth Moving Cars

Rather than having the cars simply transport to their next positions, we attempted to get the cars to smoothly transition to their next positions. Included in CarProperties is some commented code for Update() attempting to use the moveTowards function to smoothly move the cars to their next destination. Although it functioned properly to move cars smoothly, they failed to move in the correct fashion or direction.

Coordinate Locations of cars

When examining the car coordinates of both Scene 1 and Scene 2, you will see how vastly different the x/y/z coordinates of the cars relate to each other. This is due to SUMO’s way of placing cars using rather random and obscene values. In Scene 2’s case, the values are somewhat real world placements (e.g. if you were to remake the entire scene using an intersection up more North in the city, all the x and z coordinates would increase). Although this is a nice feature, it makes it difficult to track as everything doesn’t focus or begin around the origin (0,0,0).

Real World Imaging

One major issue is the OSM data not directly matching the images developed through GoogleMaps. This causes unintended behavior such as cars driving over sidewalks/through buildings or other lanes shown in the Google image. As can be shown in Scene 2, the cars begin to go off road the further they are from the intersection. The reason for this is due to GoogleMaps satellite imaging not being directly overhead as opposed to OSM’s source of data. By combining these two sources, a skew is formed due to the SUMO car positions moving in a grid like format whereas GoogleMap is actually real world.