Traffic Modeling with Unity 3D

Isaiah Martinez
CSUN
Mathematics Department
isaiah.martinez.891@my.csun.edu

02/3/2021

Contents

1 Change History

Version: 0.19
Modifier: Isaiah Martinez
Date: 3/29/24
Description of Change: Added API for TomTom to obtain Images of Traffic Flow. Demo car scene implemented. Pathfinding added.

2 Progress

Where are we with design/implementation?

We've implemented moving the goal post for the player to follow, this being implemented on the prior pathfinding demo with the obstacle wall and plain field. The goal post is moved by the player with a drag and drop mechanic, but there's still the question of whether we'll stick with a drag and drop interface for deciding the goal and deciding the start or if we will allow the player to just click on a spot, that will be dependent on how it feels.

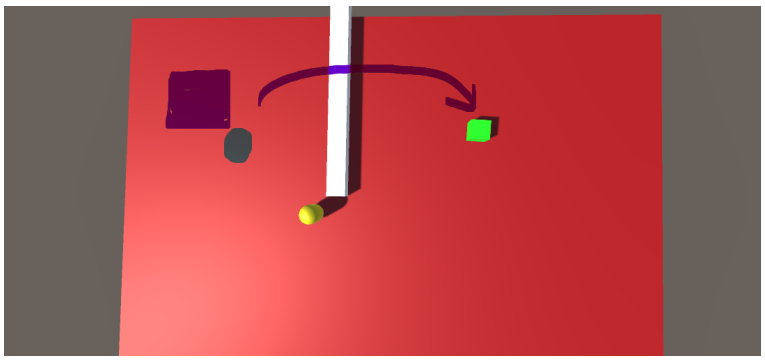

Figure 1: A screenshot of the current pathfinding scene. The blue dot represents where the capsule last stopped to catch the goal at the other side of the wall, then the goal was dragged by the player to the other side where the purple box and arrow point to its new location from where it was dragged from.

We pulled this demo into our car scene now, where the car will now follow a goal to the end of a long straight road. We want to test what path the car will take if given three roads, each with varying difficulty (one road with many turns and curves, one road with one way, and a long curved road.

We have connected to the TomTom API which provides us with images for the traffic flow.

Additionally, we have added a function that will take a given latitude and longitude coordinate, along with zoom level, to convert into units usable by the TomTom API. TomTom provided the formula to calculate the conversion between the systems, and this formula was then converted to Python Code.

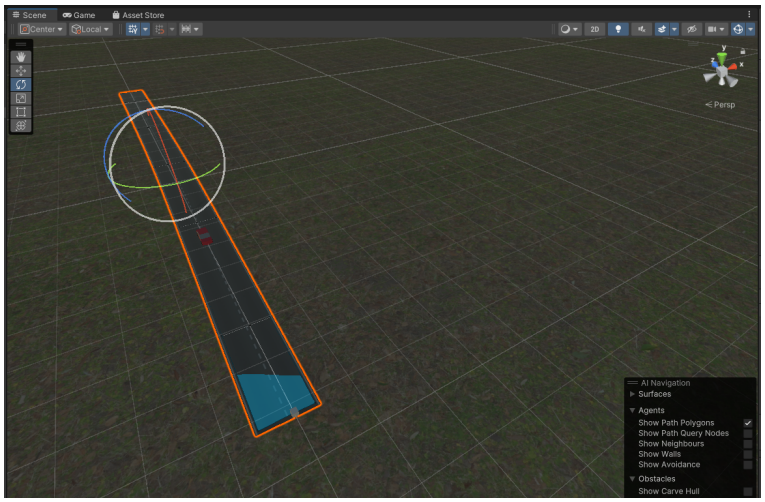

Figure 2: The Unity Car Scene with the goal at the end of the road.

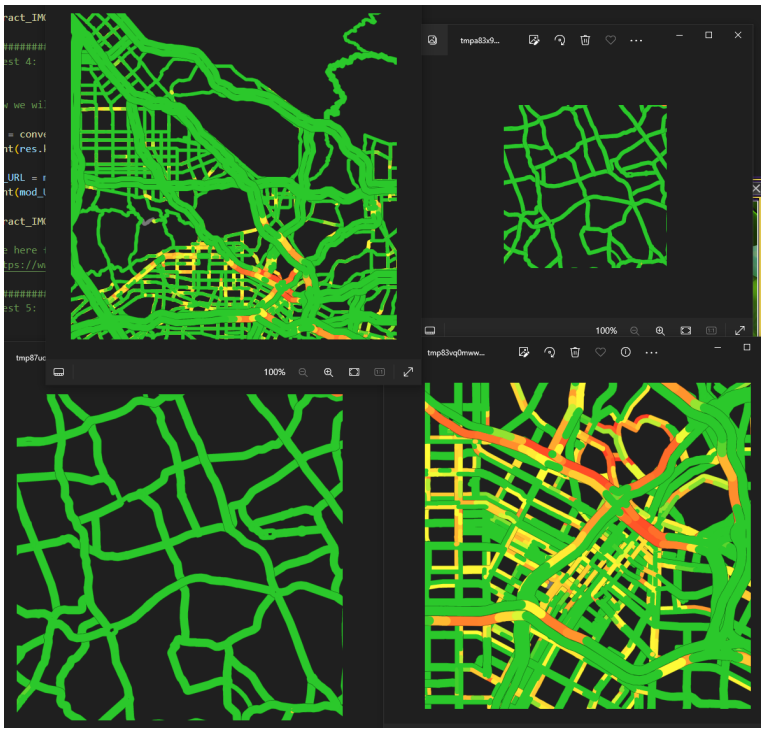

Figure 3: Sample Traffic Flow Images.

3 Challenges

What challenges have occurred for this week?

A challenge we had with the implementing the pathfinding scene into the main Unity scene was configuring the correct places that the agent is allowed

```python
#function to convert latitude/longitude & zoom to x, y coords
#eg.    dict('x' : VALUE, 'y' : VALUE, 'zoom' : VALUE)
#provided by TomTom and converted to Python
def convertLatLonZ (latitude, longitude, zoom):
    #basic type checking
    res = typeChecker(latitude, longitude, zoom)
    if (res == 0):
        #typing OK
        latitude = float(latitude)
        longitude = float(longitude)
        zoom = int(zoom)

        #if bounds are correct then utilize formula
        if boundChecker(latitude, longitude, zoom):
            x = math.floor((longitude + 180) / 360 * pow(2, zoom))
            y = math.floor((1 - math.log(math.tan(latitude * math.pi / 180) + 1 / math.cos(latitude * math.pi / 180)) / math.pi) / 2 * pow(2, zoom))

            #dictionary with keys for each calculated value (x, y) and zoom
            return {
                "x" : x,
                "y" : y,
                "zoom" : zoom
            }
```

Figure 4: TomTom Conversion Function.

to traverse.

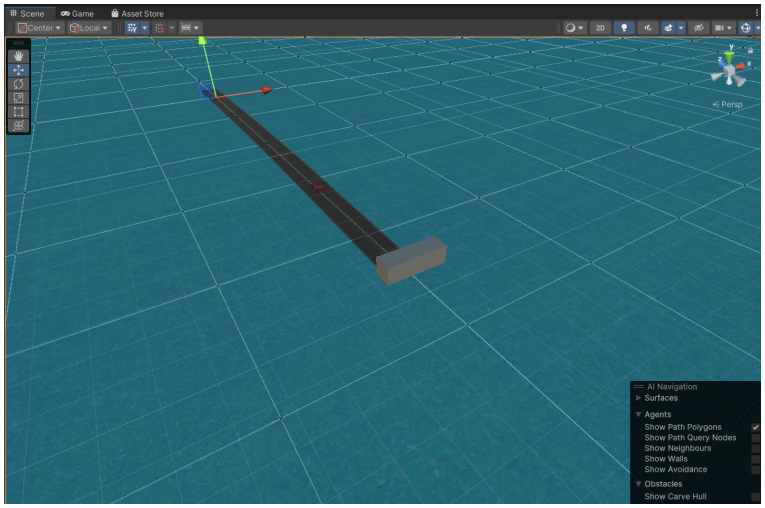

Figure 5: The baked Nav Mesh recognizing everything but the road we want the car to go on.

The blue represents the path that the program will let the car traverse. This is a problem because the mesh covers every part of the plain but the actual road that we want the car to stick to. Fortunately, this isn't a very hard fix but just took a good amount of research. We just need to make sure the "Navigation Static" option below was ticked off.

Additionally, there's currently a problem that the model for the Sedan is by default upright, so we'd need to change the model somehow.

Obtaining the information to use for the TomTom API was rather interesting. The documentation was clear, but very segmented; The information was spread across several pages that were not all accessible from each other. A difficult challenge was deciding which API's to utilize (since this meant we might have to change the access key). We are considering utilizing multiple API's and combining the images together to get a more complete sense of direction/understanding of the area.

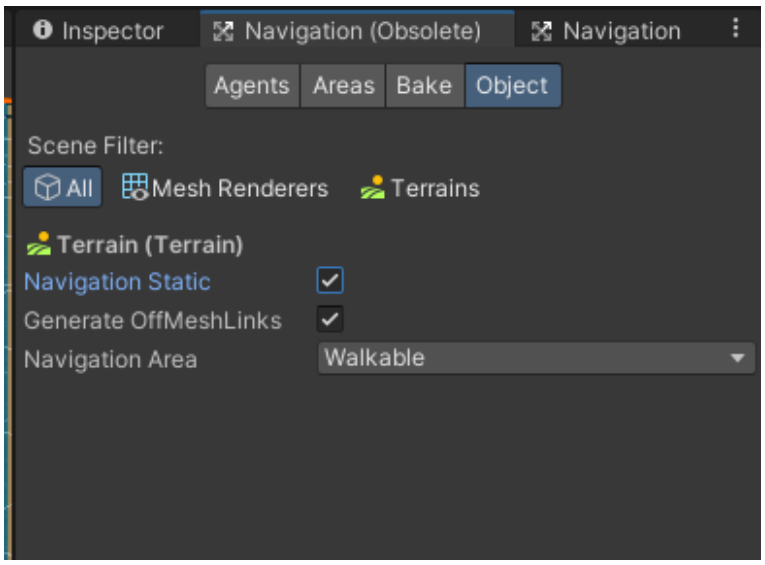

Figure 6: The settings we need to keep an eye out for.

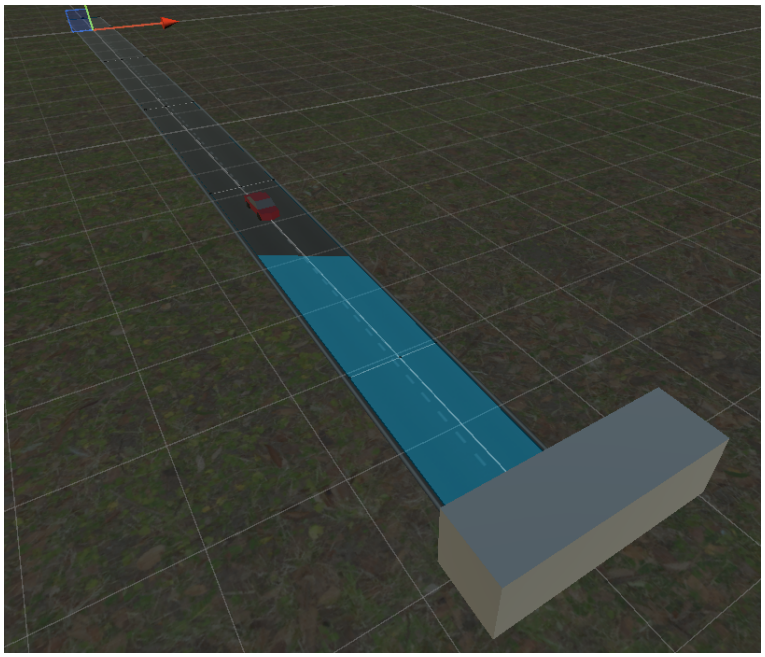

Figure 7: Though it's hard to see in the screenshot, as you go through the scene the whole road is blue and it's correctly baked.