

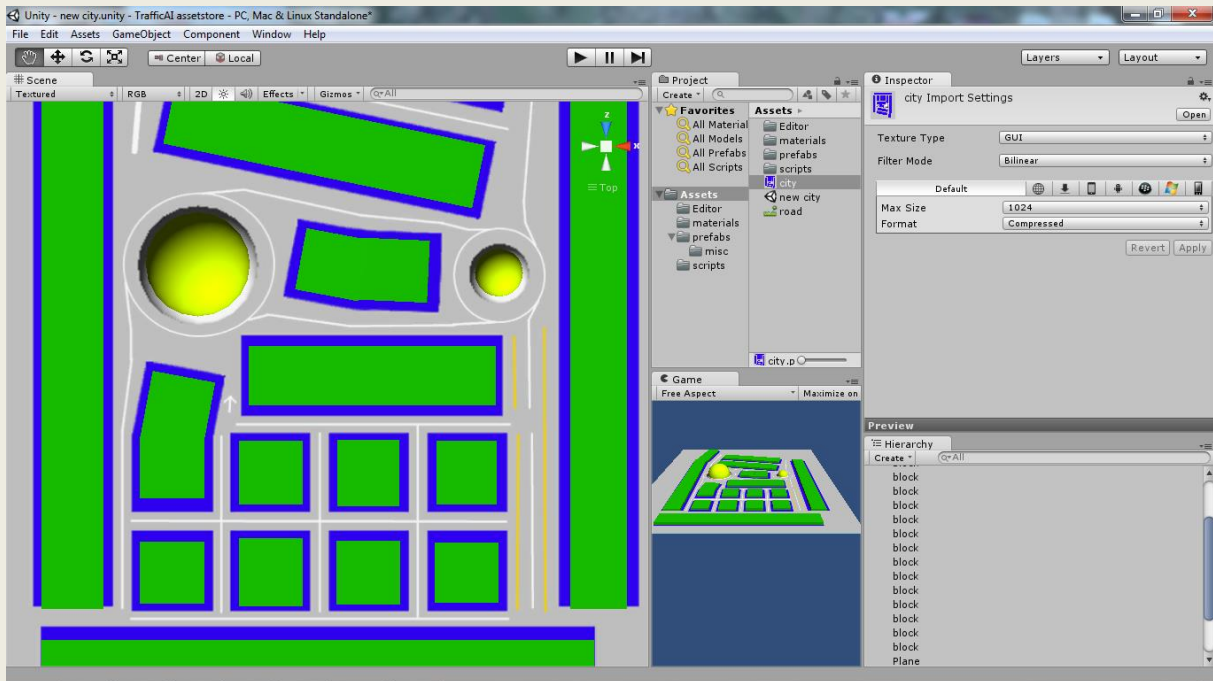
Ti Traffic AI User Guide

Part I – Tutorial 2

Part II – How it works25

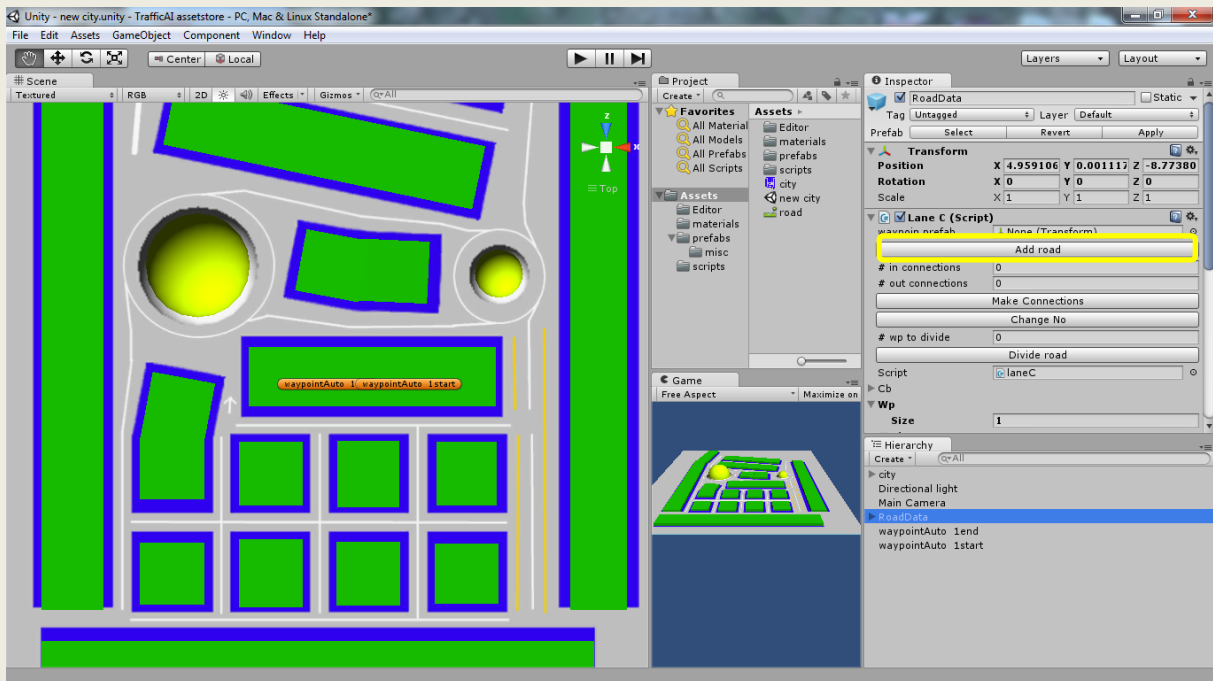
Part I - Tutorial

In this tutorial we will describe how to set-up the traffic AI for a city model. We have a simple city model as seen below:

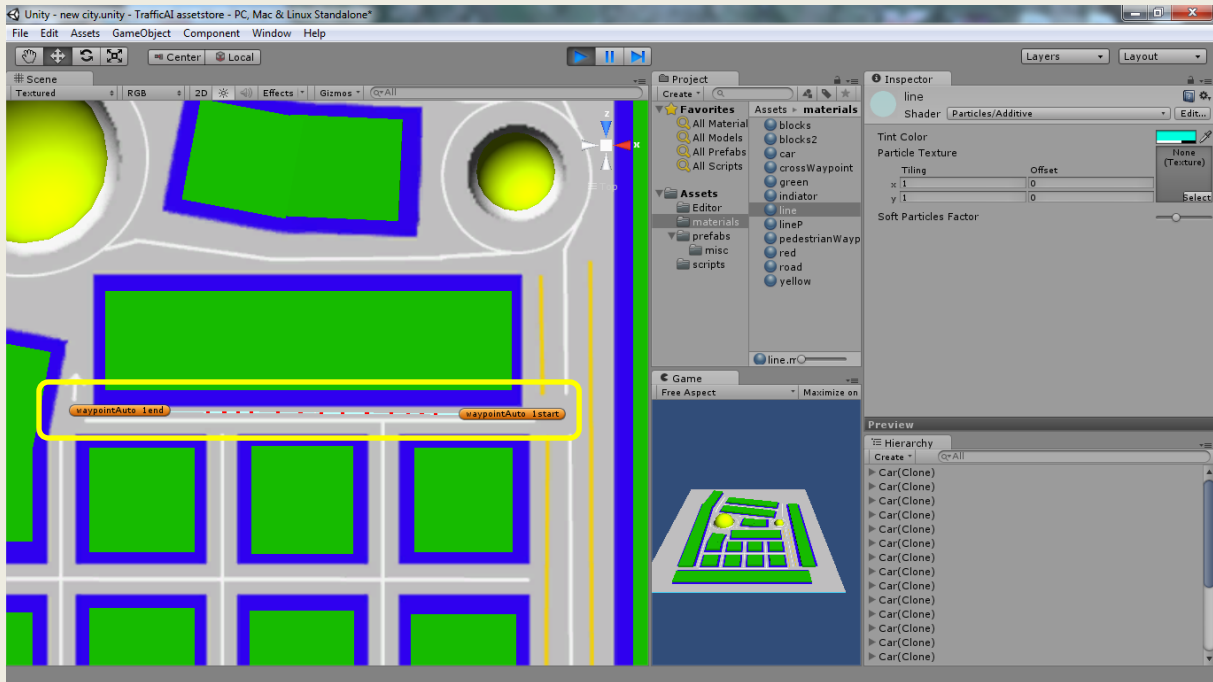


All we need to add to the scene to start is the **RoadData** prefab. Once it is added we are ready to go. For almost all the operations below, we need to select the RoadData gameobject that's places into the scene. Also make sure you are looking straight down to the scene with orthogonal view.

RoadData being selected, we see a number of buttons and input fields. Let's start by clicking "Add road". Two control points appear on the scene.

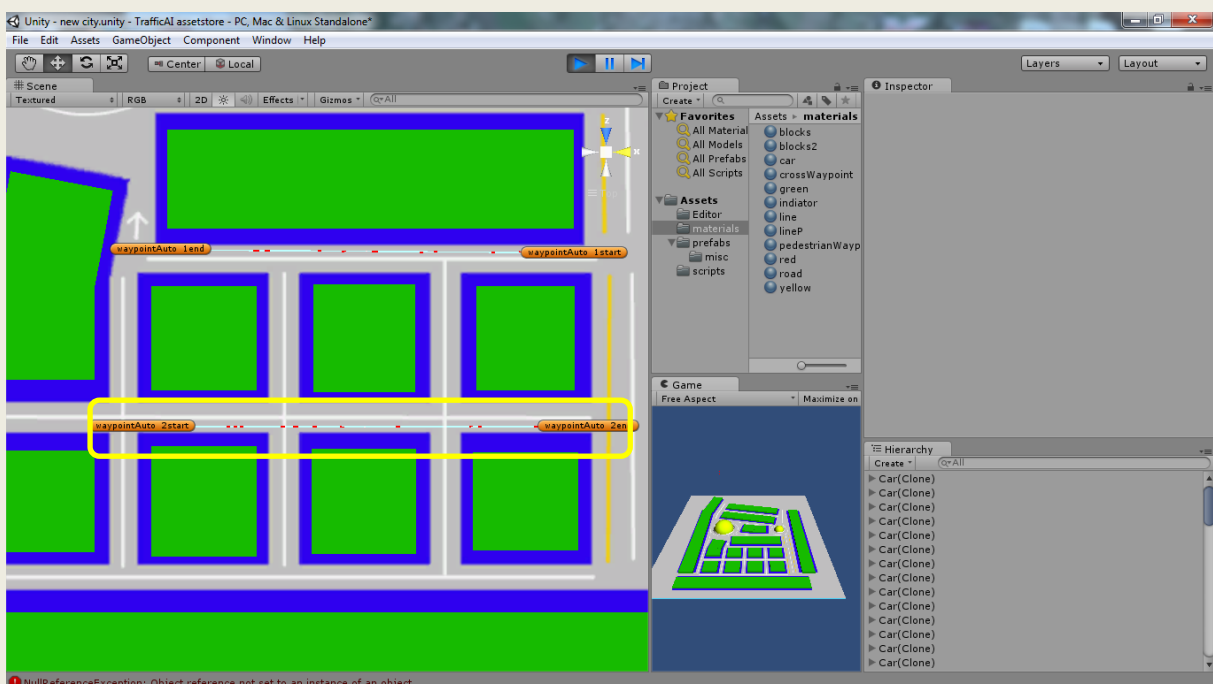


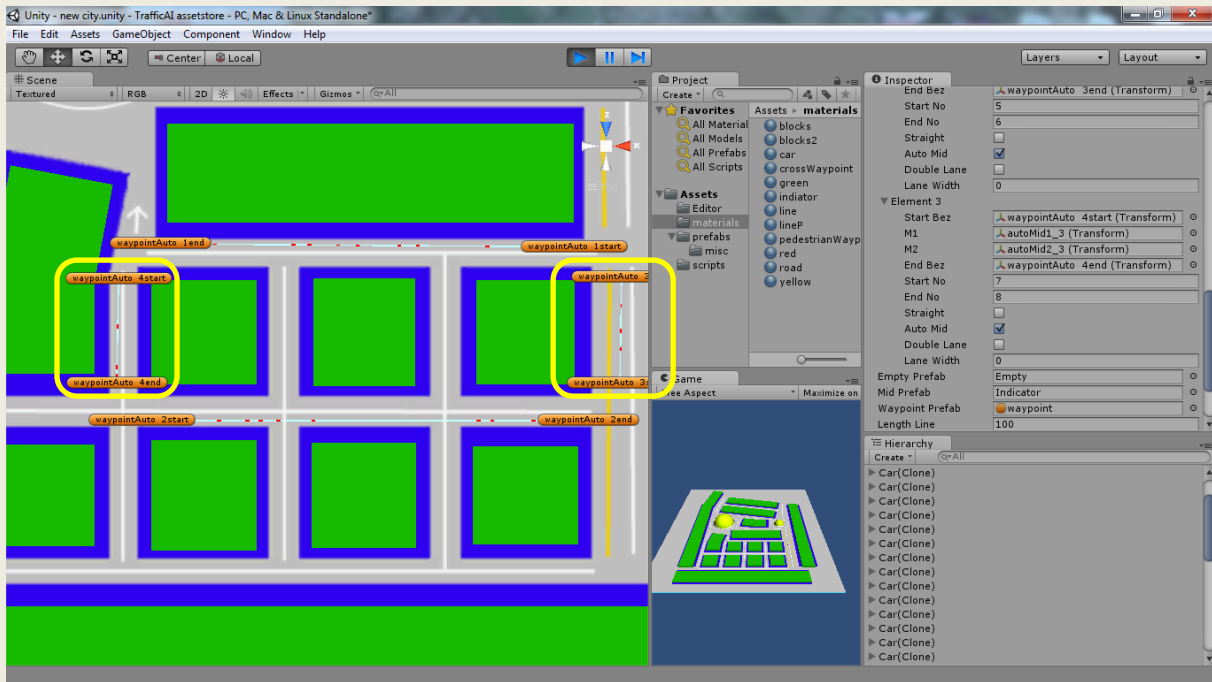
Drag them to proper positions on the road. Once you play the scene, a lane will appear between them. The lane consists of a Bezier curve and some additional data. The red rectangles on the screenshot below are cars. Cars are physics based and uses Unity 3D's wheel colliders. They drive around by applying torque and brake and steering the wheels. So they are as realistic as possible from a game physics standpoint. They would act naturally in the event of a collision or force acting upon them. Lanes are one directional.



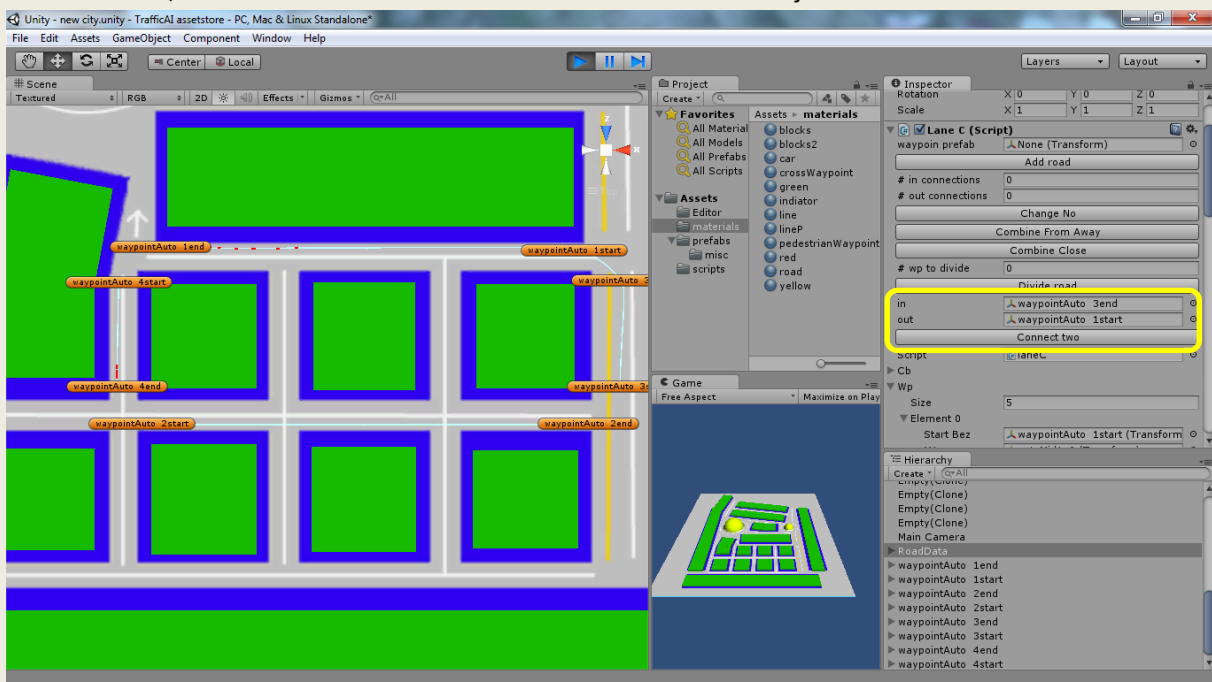
Let's add one more lane and position it properly:

And two more:

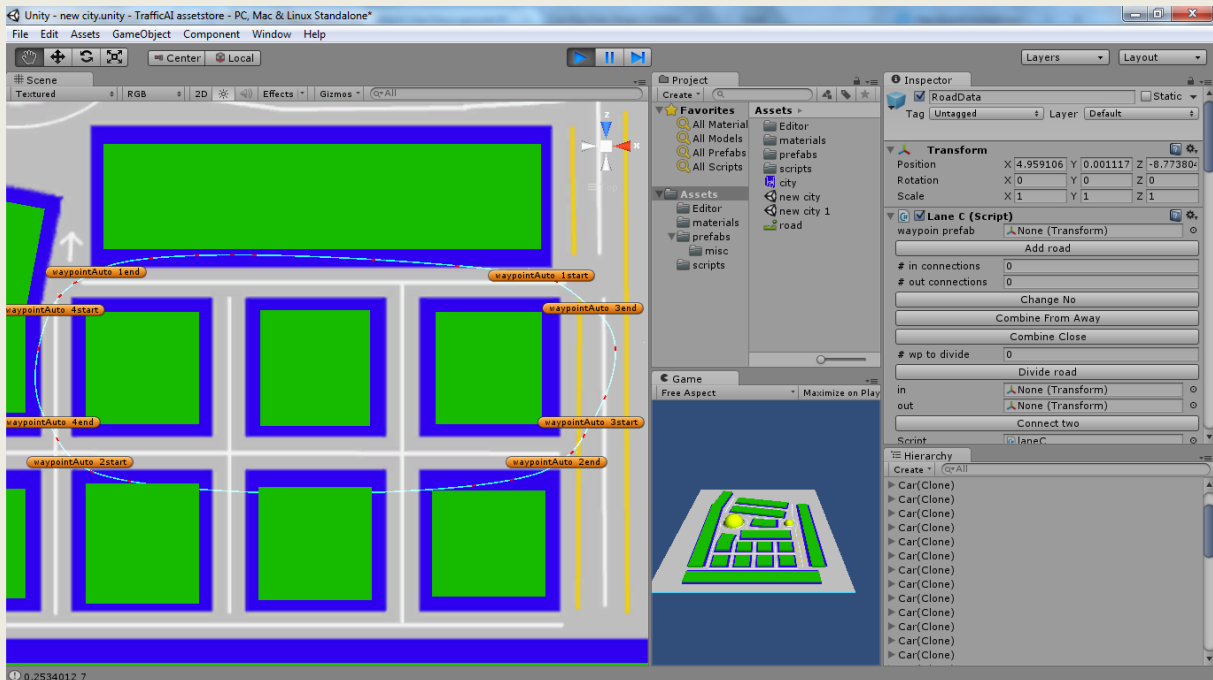




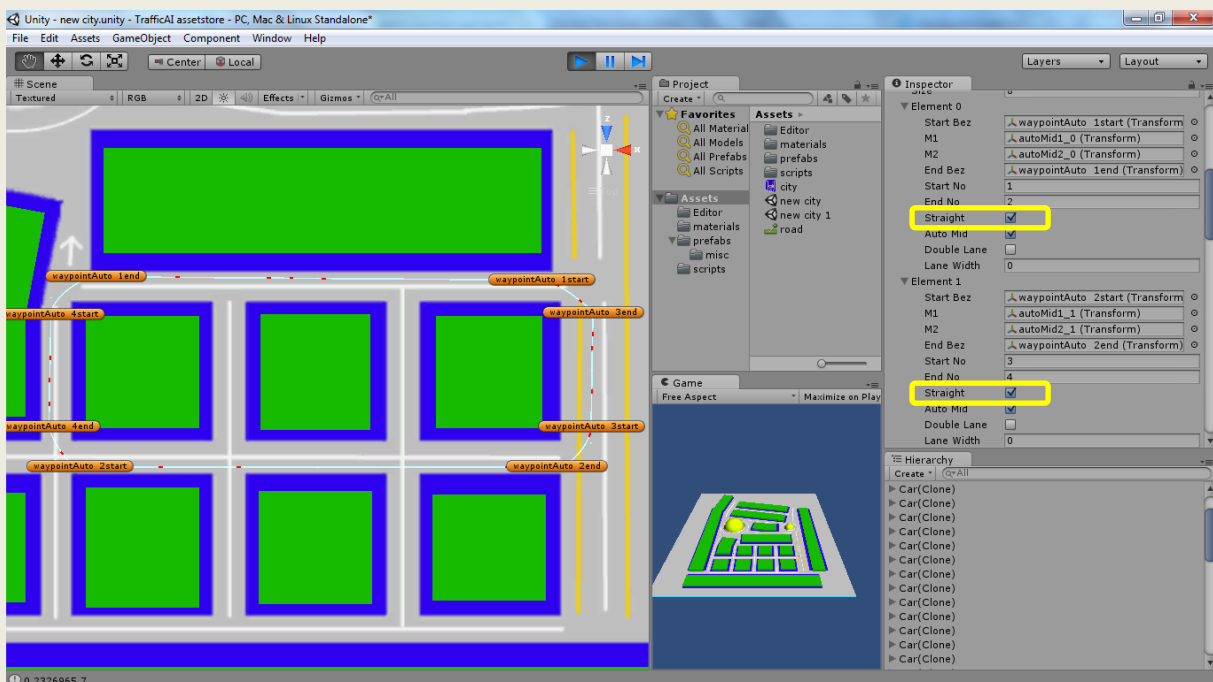
Now that we have 4 lanes, we can proceed with connecting them. For this we will use the **Connect two** button. First we drag the two waypoints to the slots above the button. The order of the waypoints is important. We need to think about the new way and the way traffic flows in and out. We place the control point that traffic flows in to the new road to the “in” slot, and the other one to the “out” slot, and click **Connect two**. It connects the lanes and adjust the curvature.



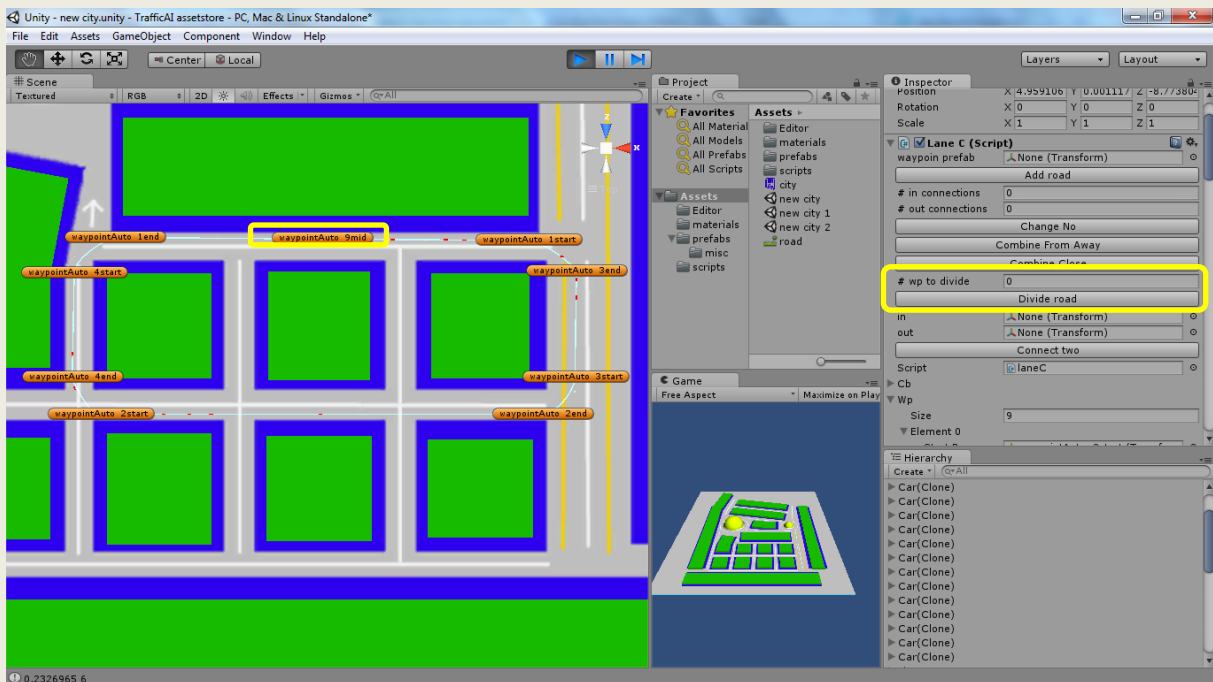
Now we connect other edges as well, by repeating the operation above three times. The result we get does not look good as the system tries to auto adjust the curvature for all lanes, it does not know which ones should be straight:



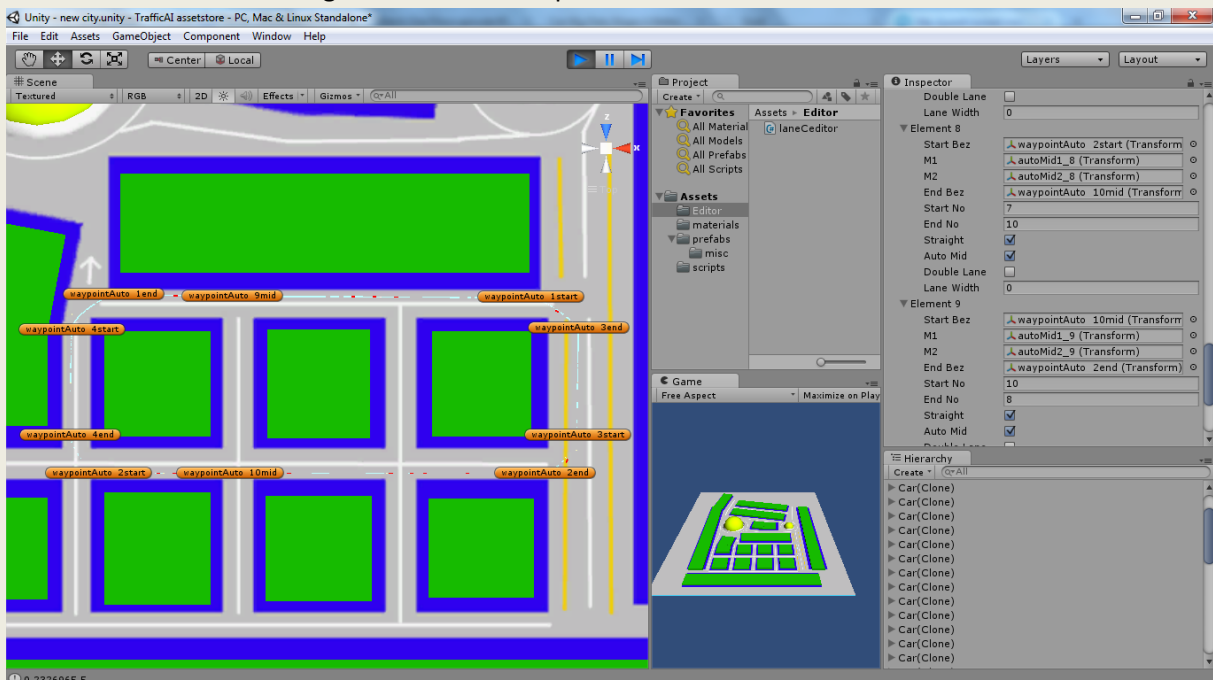
To make them straight we need to find the array “Wp” in RoadData’s inspector. In the array find the numbers of the lanes you want to straighten and choose the “Straight” toggle. (Edit: To make finding lane numbers easier we have modified the system to show the lane numbers in green, not shown at this stage of the tutorial). This will fix the weird curvatures. From now on whenever you need to make a road straight apply this method, it will not be mentioned in following steps, if you see a road that needs to be straight, just straighten it. As you can see we have nice lanes now:



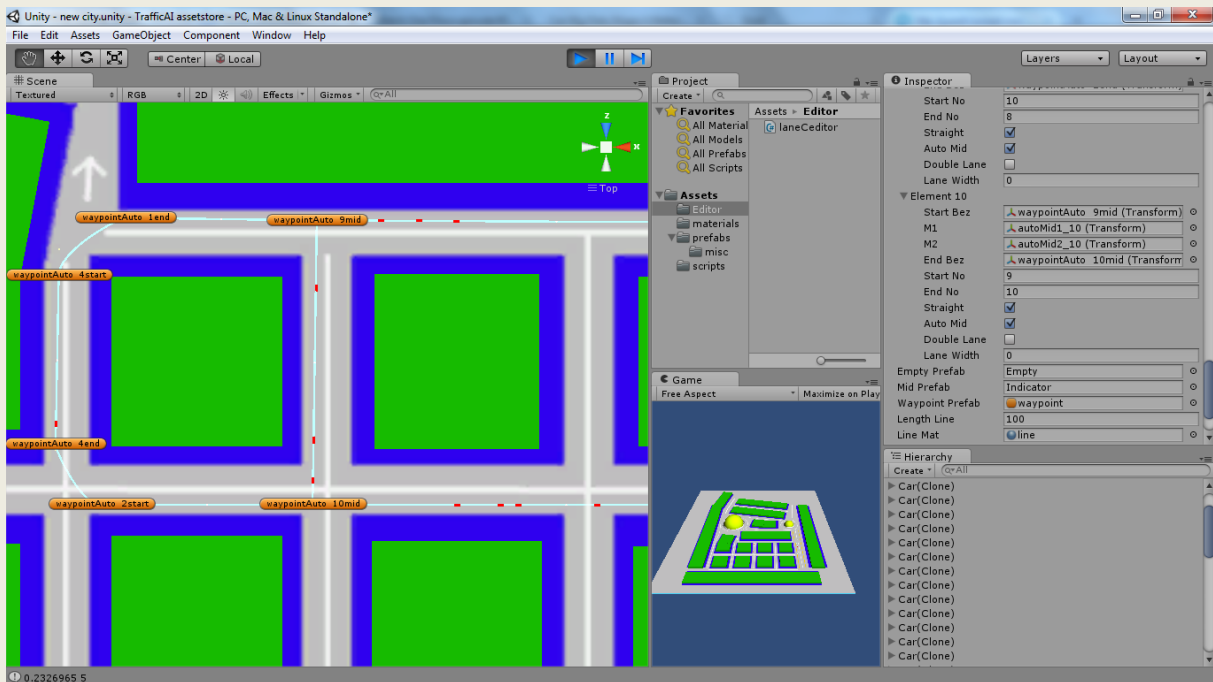
Now we will connect the upper and lower roads. To do that, we need to use **Divide Road** button. In the input slot write the lane that needs to be divided, for this time "0" and click the button; it will divide the road to add a new control point:



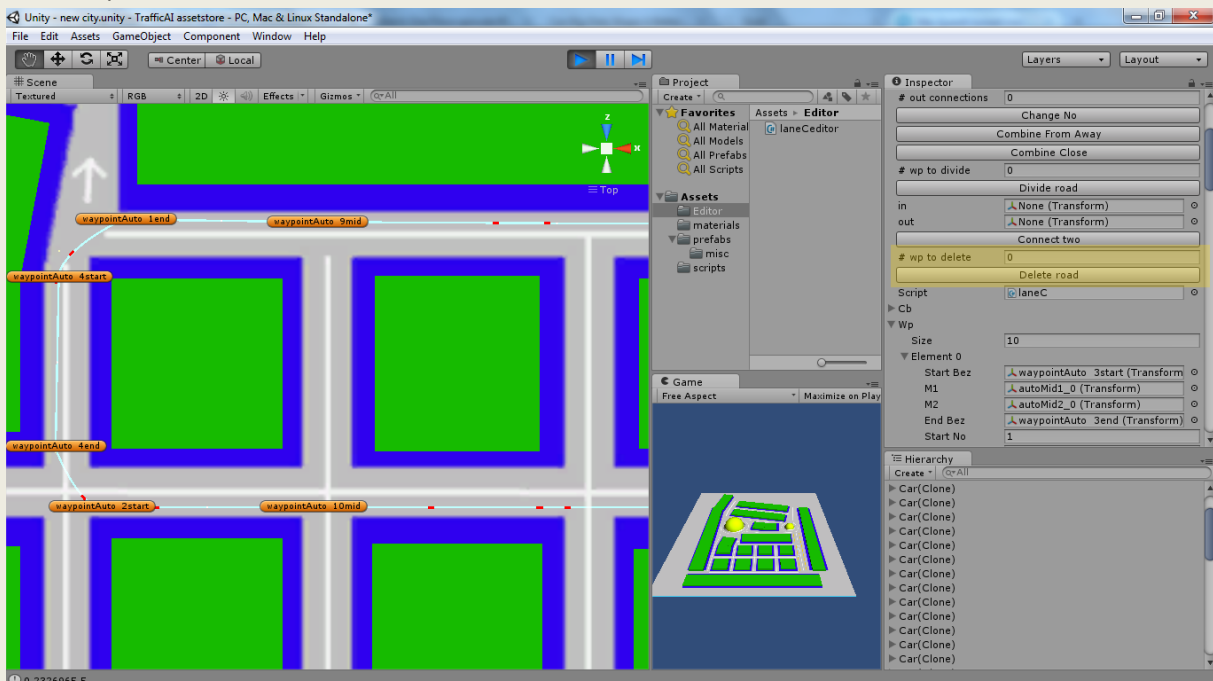
Now drag the control point to a proper position at the start of a vertical road, divide the lower horizontal road too and align the new control point:



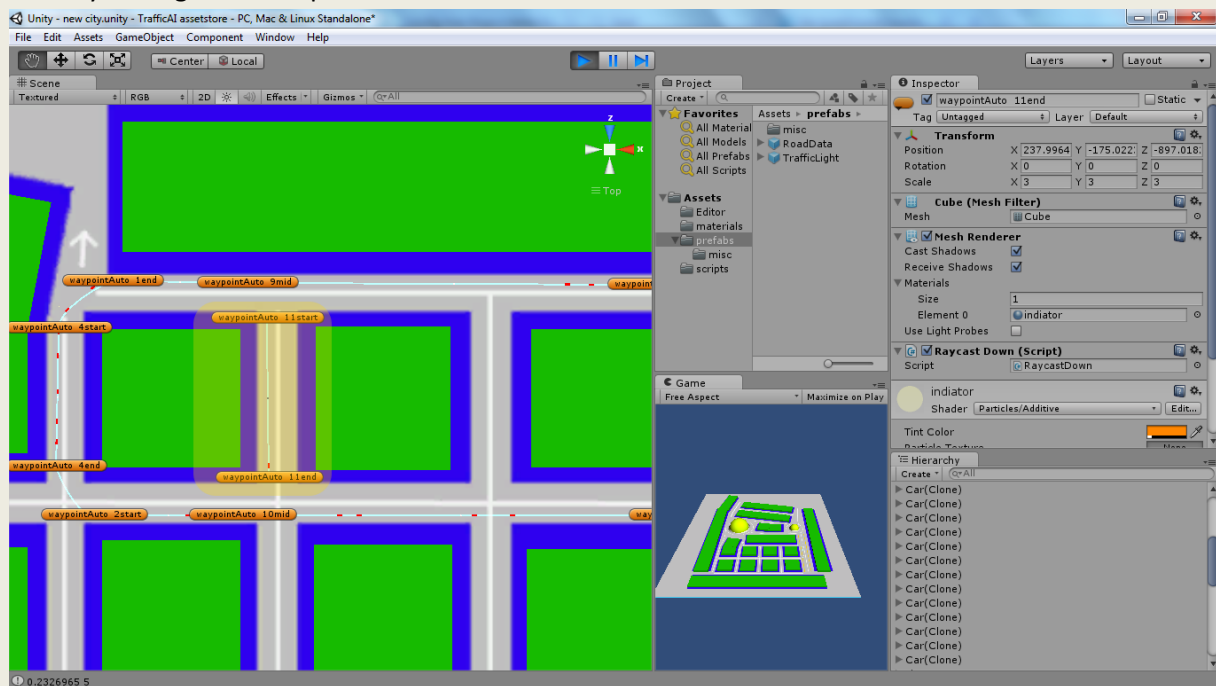
Now connect the two new control points with a lane by using **Connect two** button as mentioned earlier:



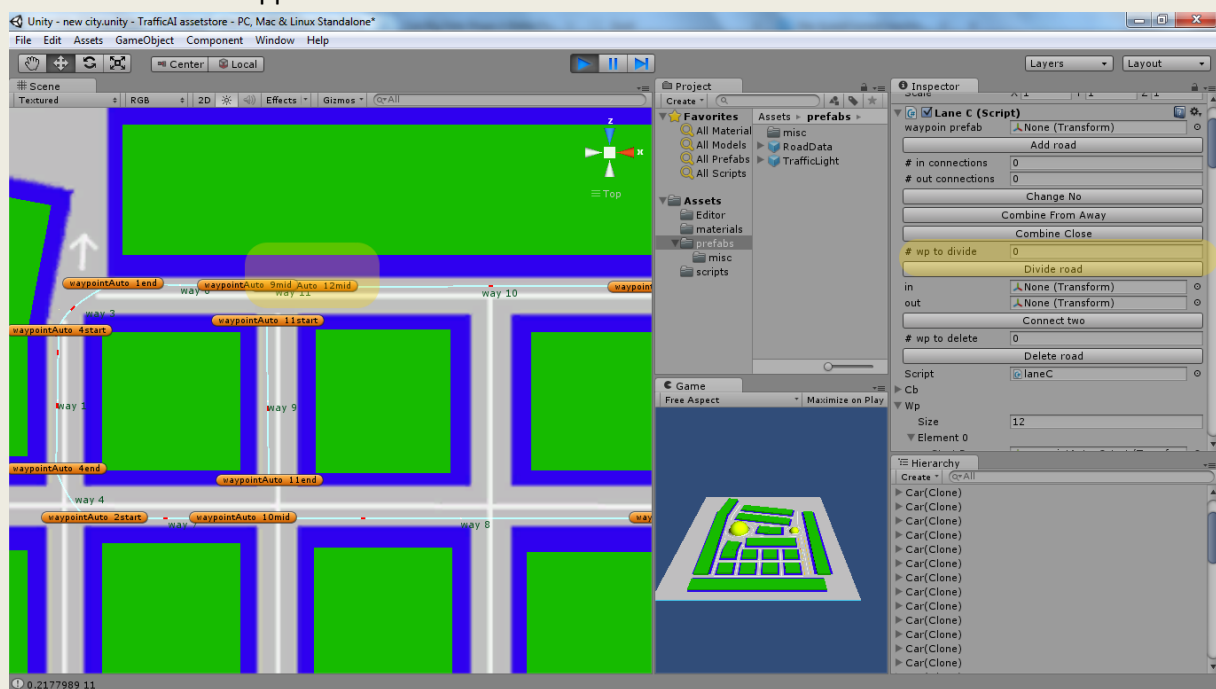
Now this would be a nice way to connect the lanes if we were doing a simple grid, but for our purposes we need a better one that can deal with multiple directions. Now delete the new lane by using the **Delete road** button. Simply input the lane number to be deleted and click the button. At this sample the lane number is "8".



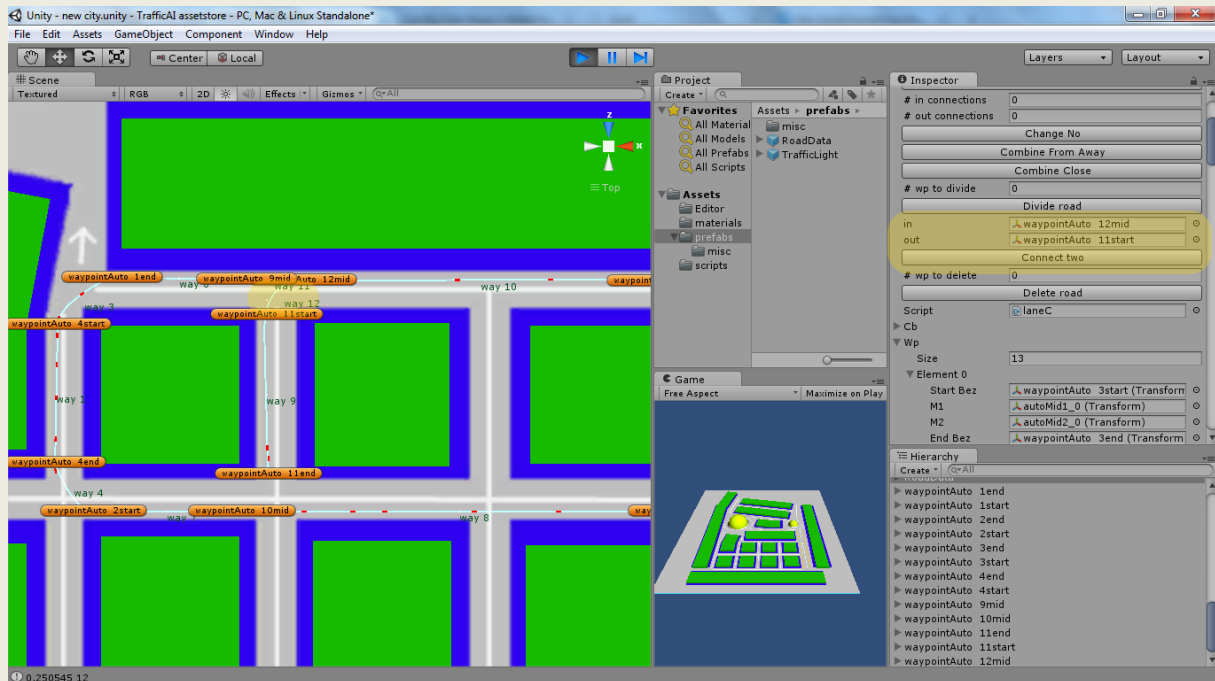
For a complex junction we need separate lanes for roads and separate ones for the junctions. We start by adding a vertical piece of road:



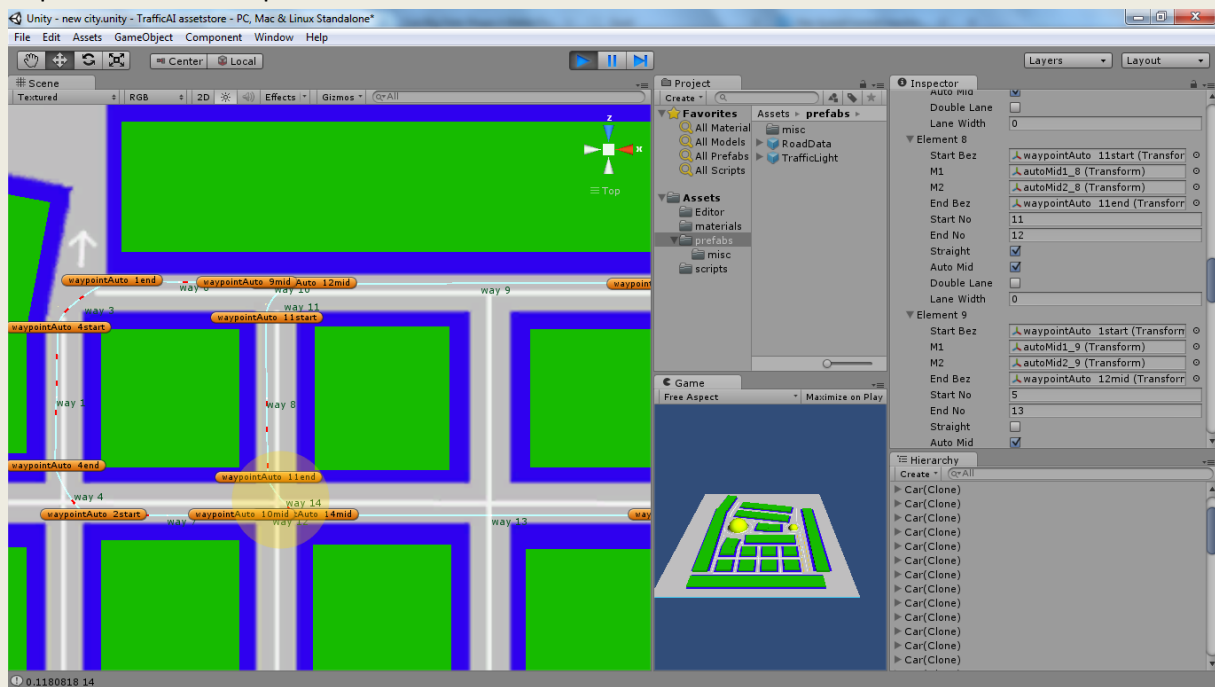
Then we divide the upper lane once more:



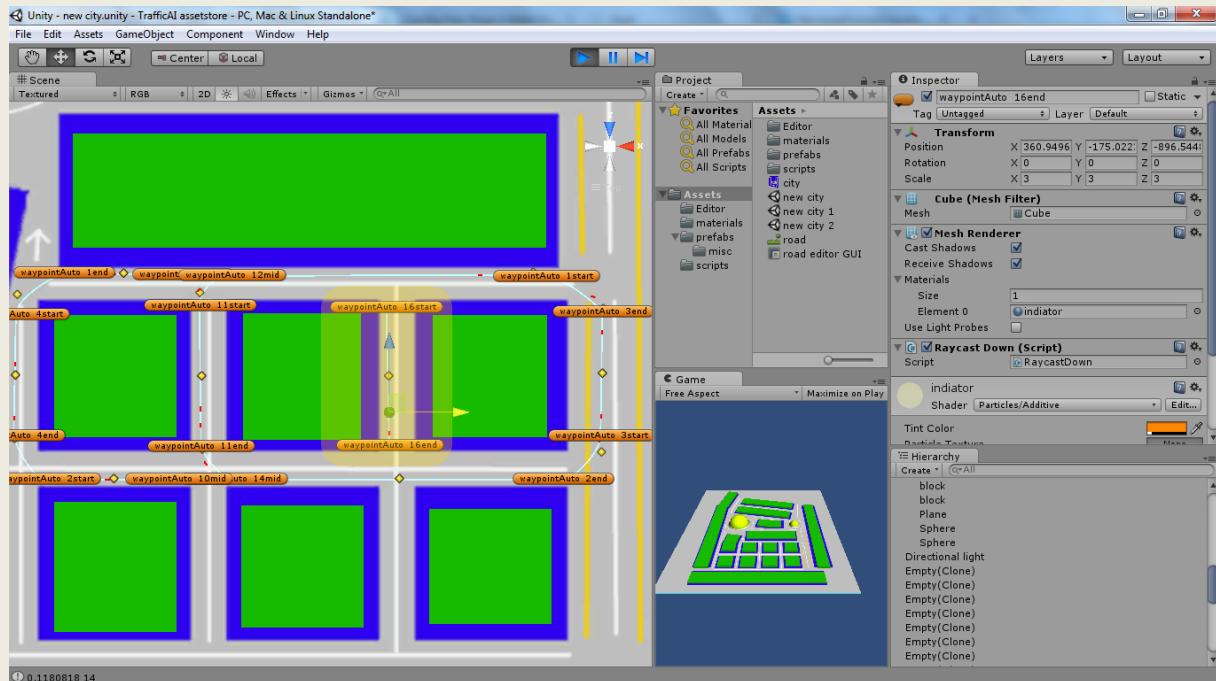
And we connect it to the vertical lane, keeping the direction of the traffic in mind:



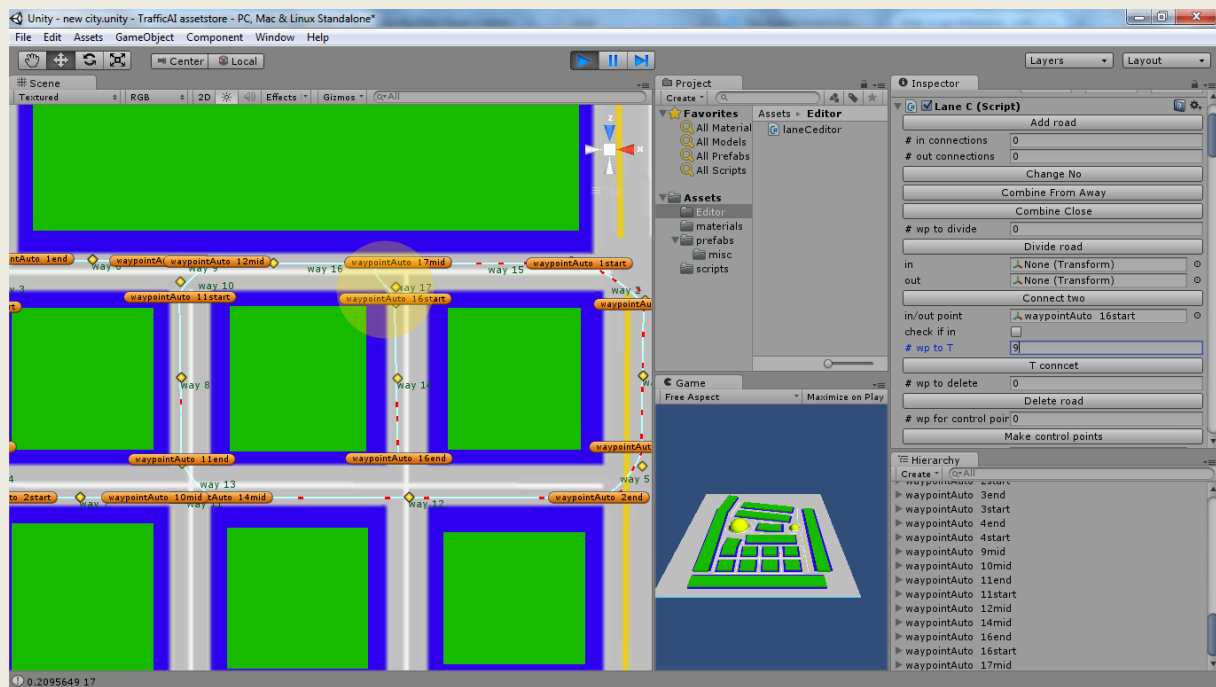
Repeat for the lower part:



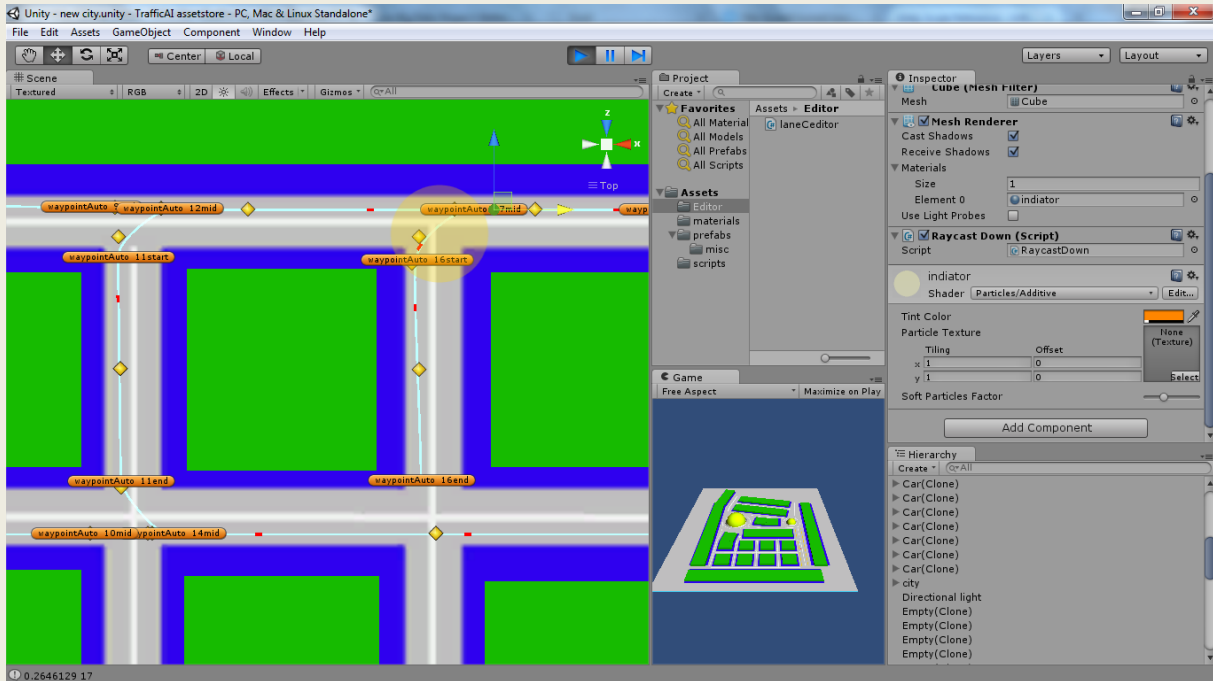
Now we add another vertical lane:



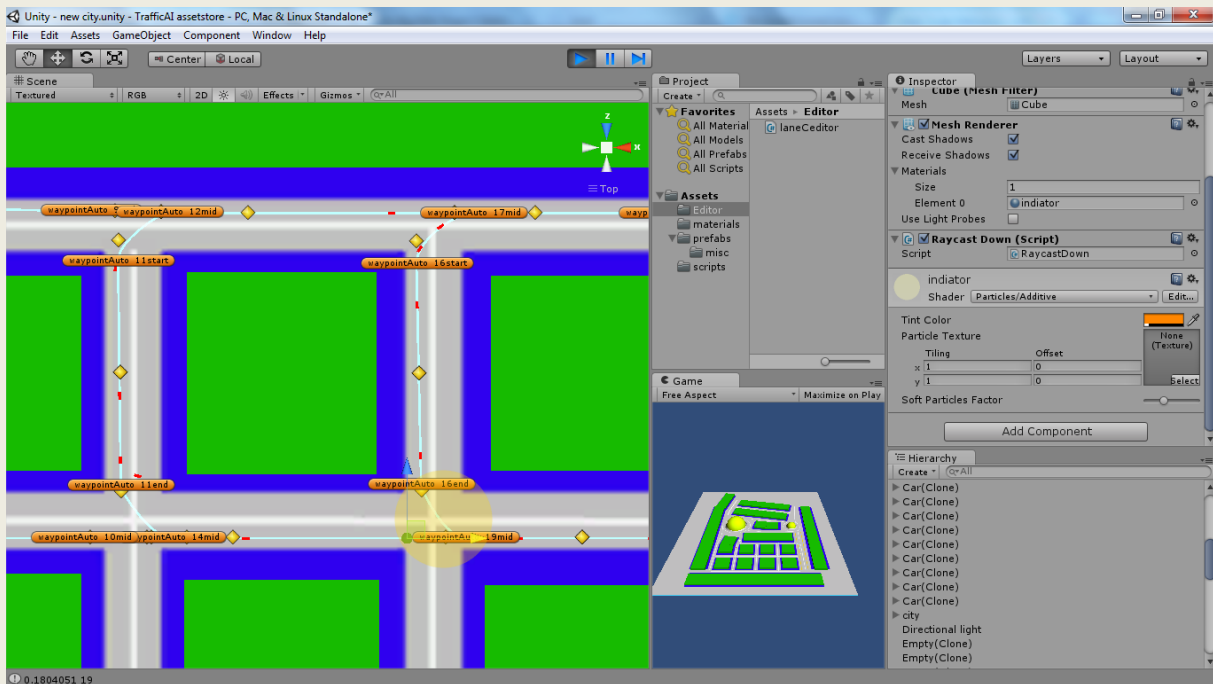
Divide and connect as before:



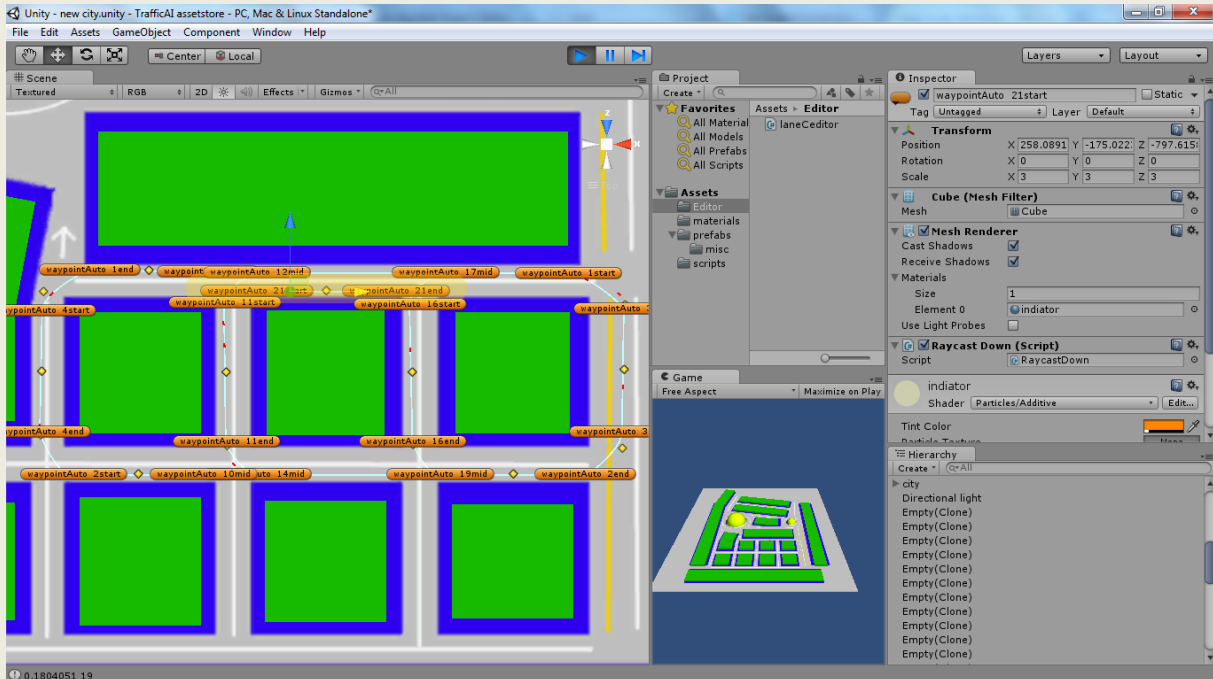
It seems we did not account for the flow of traffic, we fix the position of the control point:



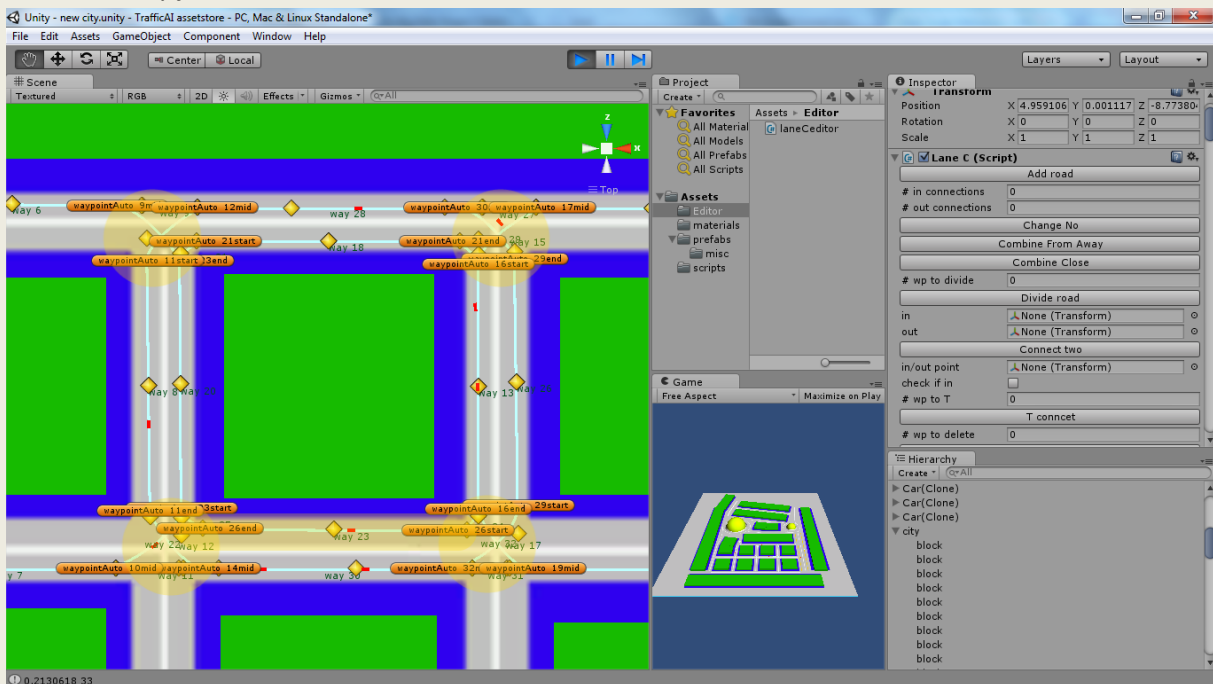
Same for the lower part:



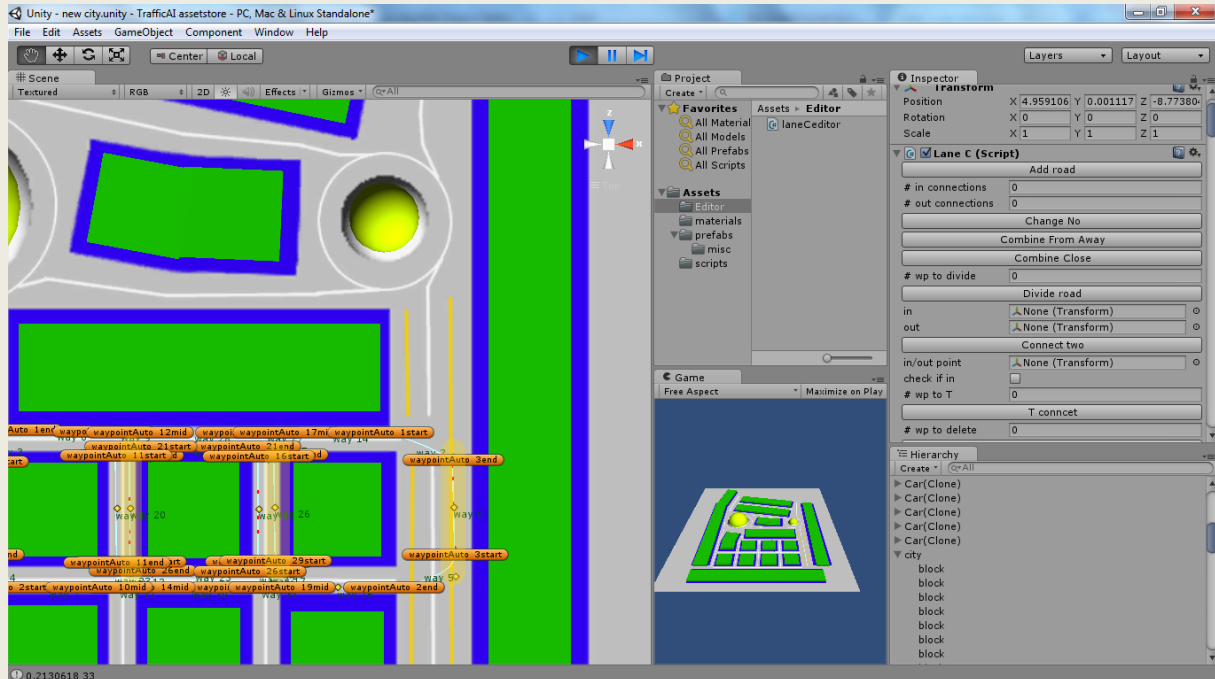
We add a horizontal piece of lane in the opposite direction as we would have in a real road:



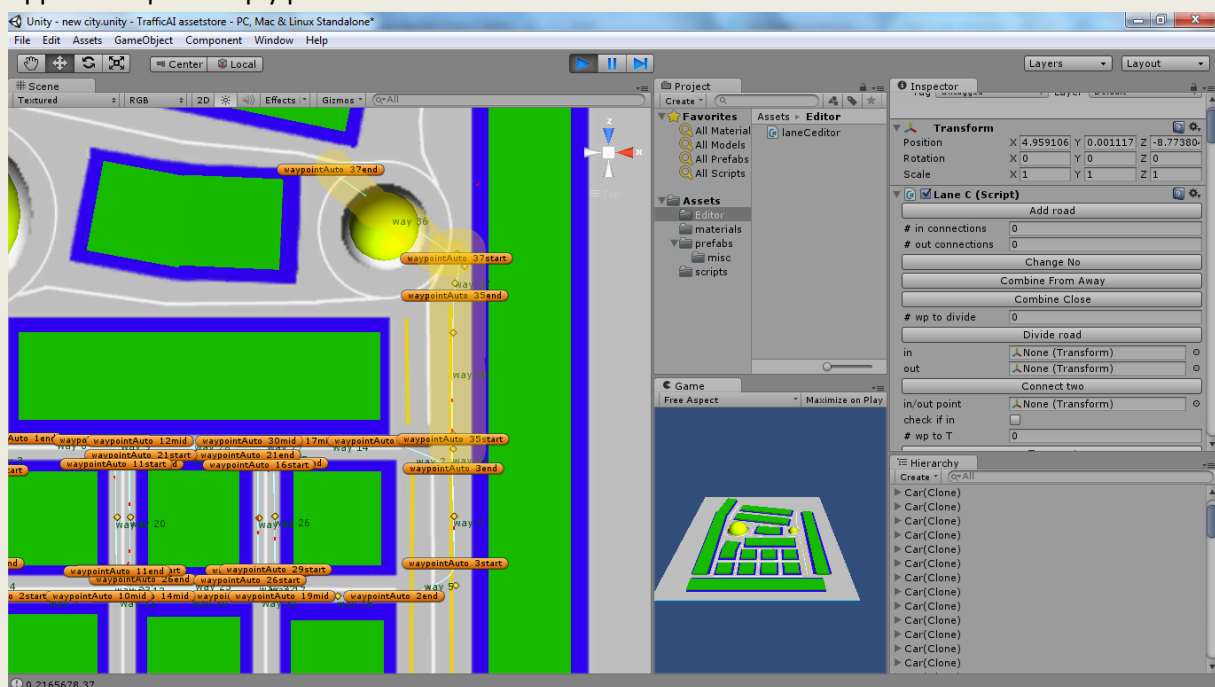
We add another horizontal piece to the bottom. Add Opposite lanes for vertical lanes. Connect all the necessary junctions with **Connect two** button:



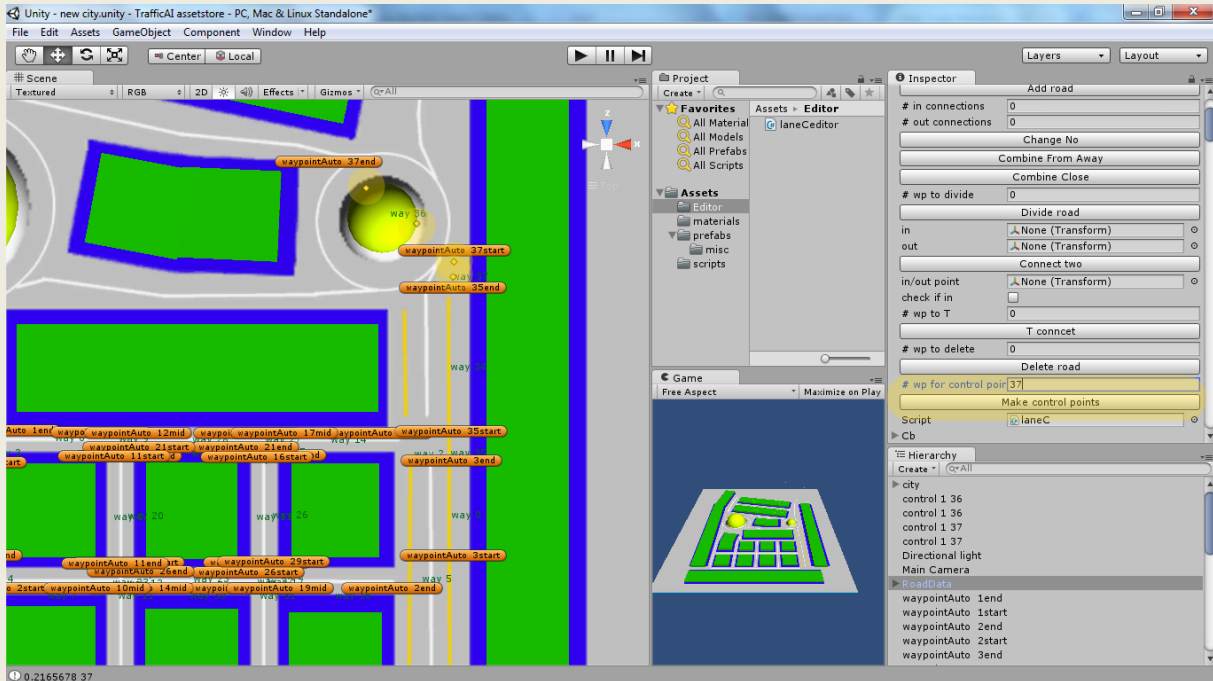
We note than we placed the rightmost lane in the wrong place, moving it to its proper place:



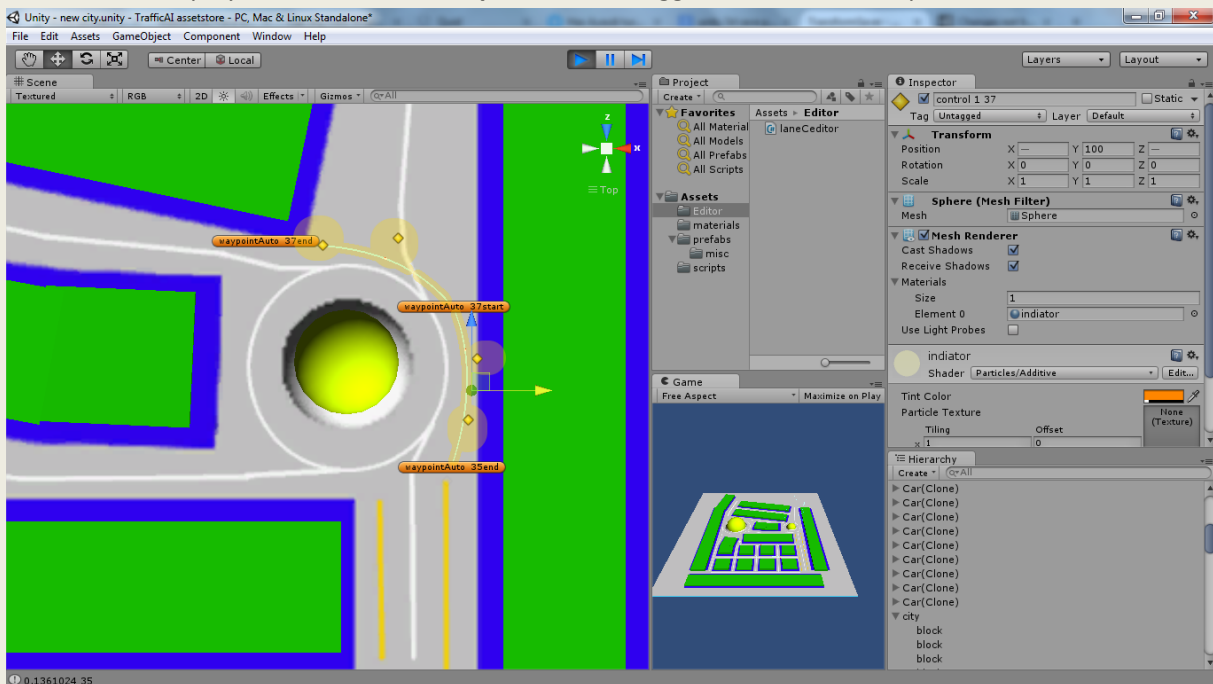
We add some pieces of lanes all the way up to the circular dome. Not that at this point the uppermost part simply penetrates the dome:



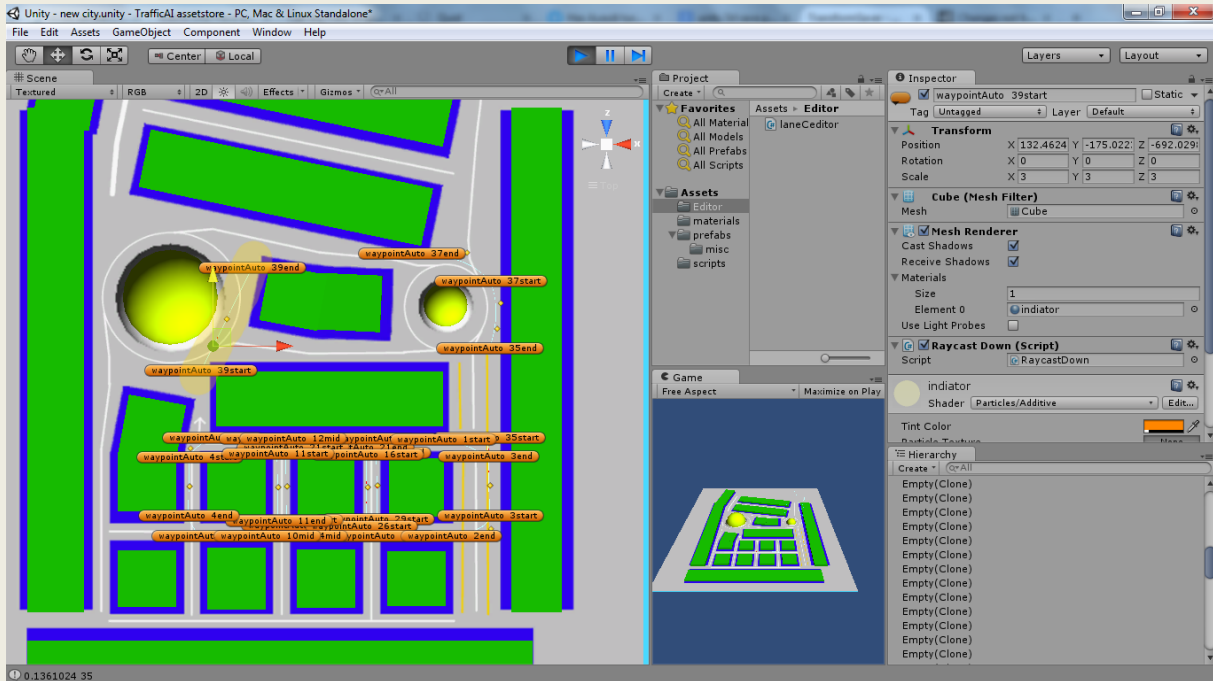
In order to make it flow around the dome, we need to control the Bezier curve manually. Usually the system does not expose the 2 of the 4 control points of the Bezier curves to us; it exposes just the start and end points. To reveal the remaining 2, we use the **Make control points** button. We input the number of the lane to be exposed and click the button. In this sample we use it for the two uppermost lanes. You can notice the yellow diamond shapes:



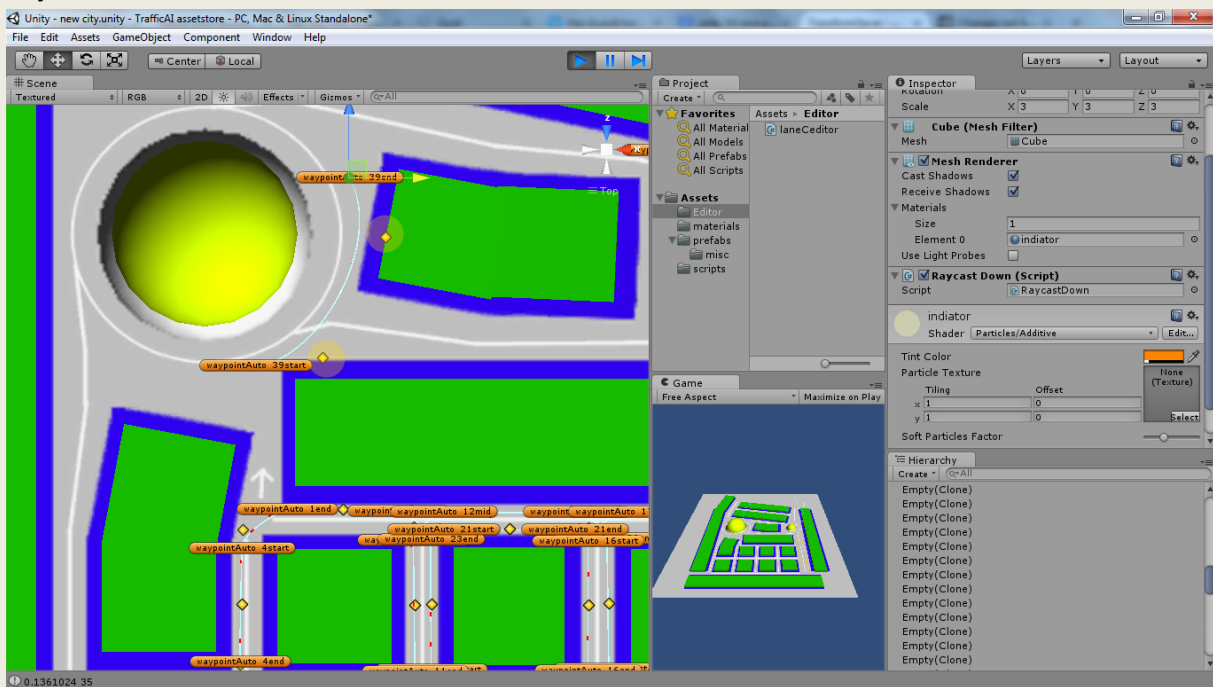
We drag the control points to suitable positions to obtain a flowing lane. Note that outside play mode, lanes are not shown. During play mode lane changes are not shown. To enable lane change visualization in play mode choose the **Update Lanes** toggle at **Road data** inspector.



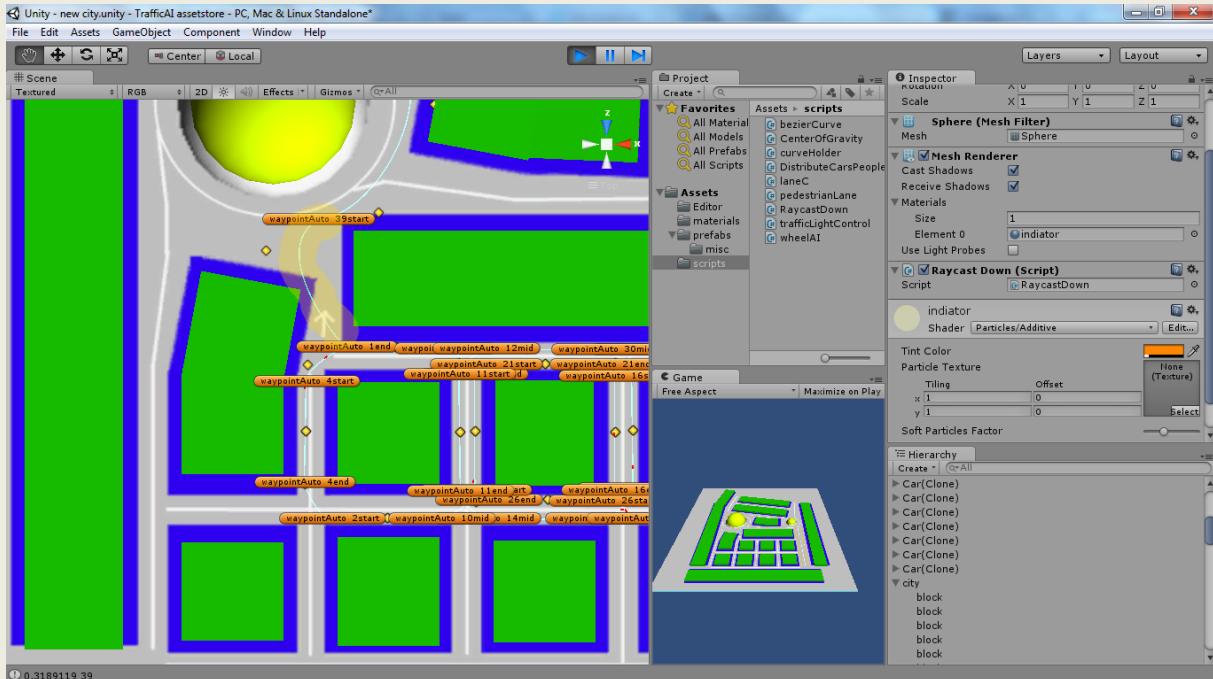
Now let's do the same for the other dome side, first place lanes, than adjust the Bezier curves:



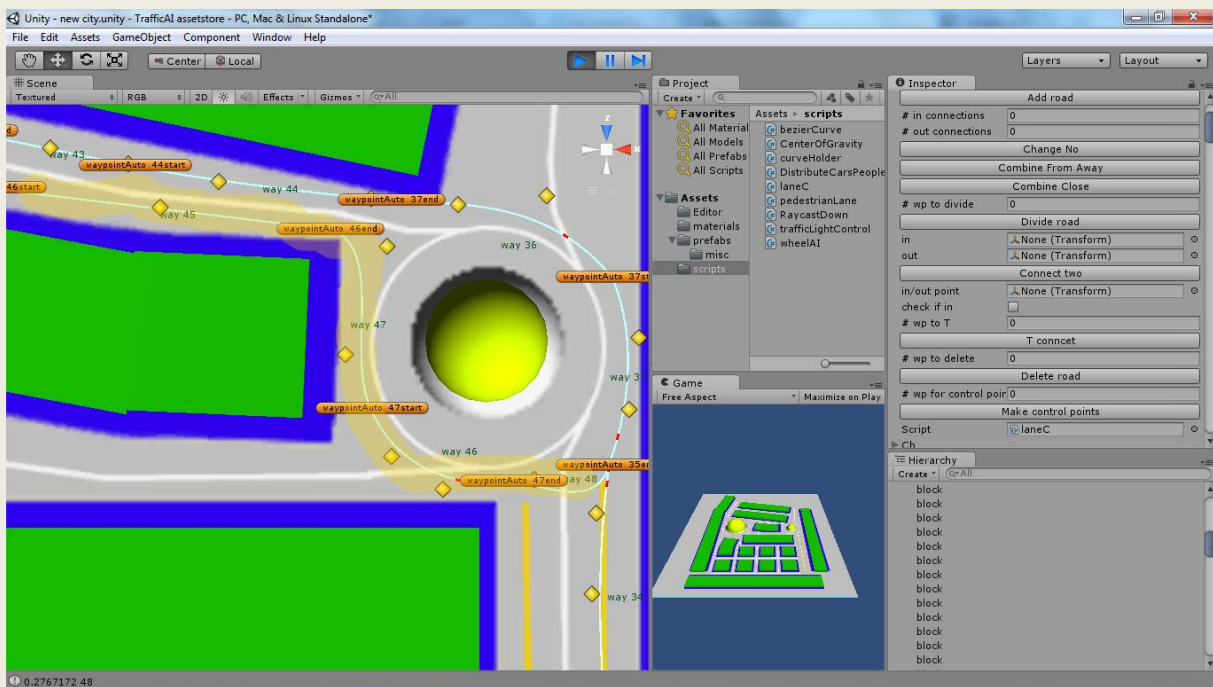
Adjustment:

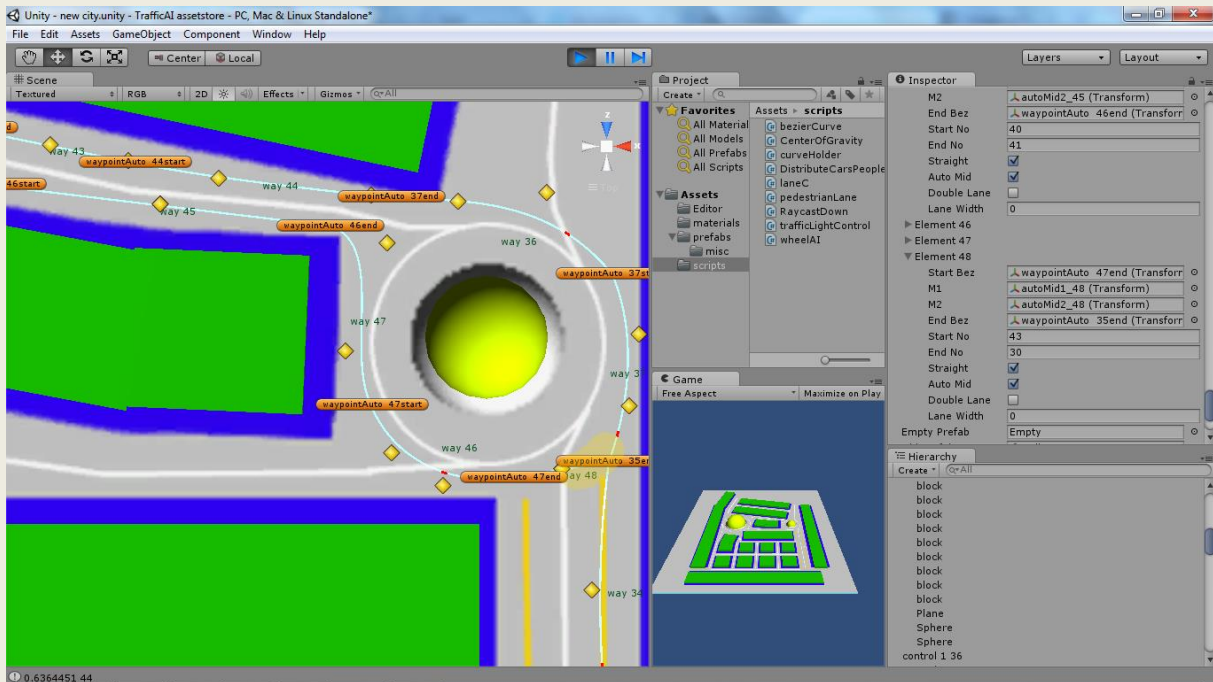


Connect and adjust:

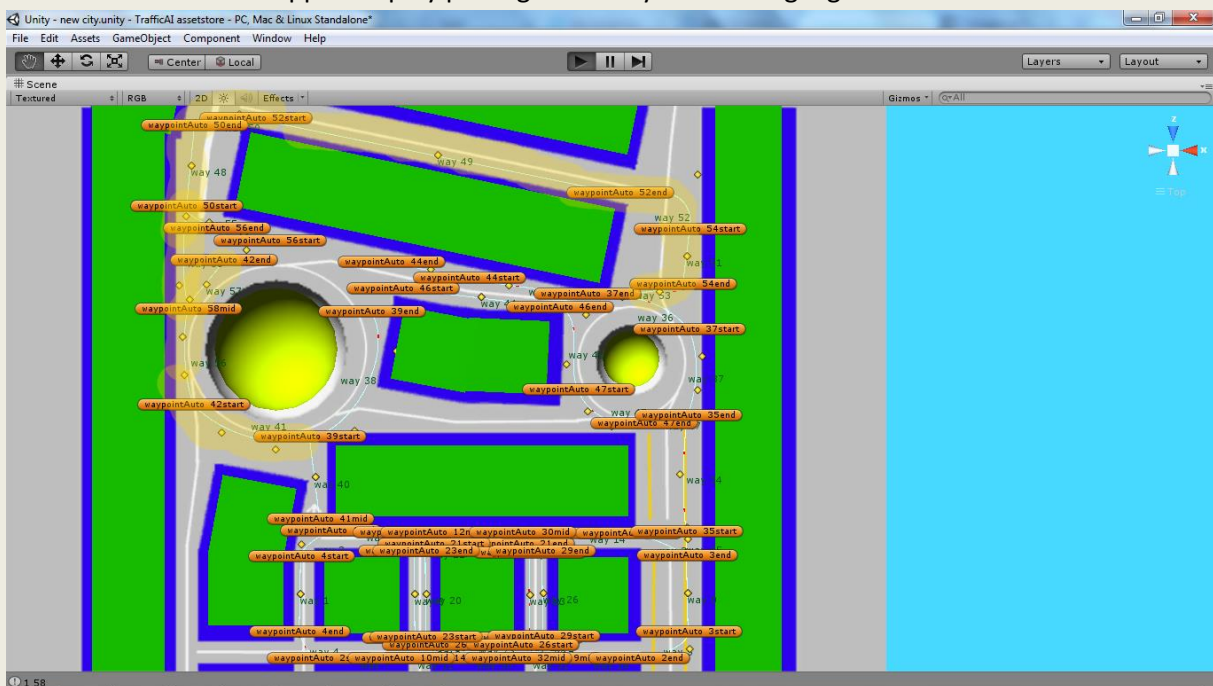


Let's add some other lanes:

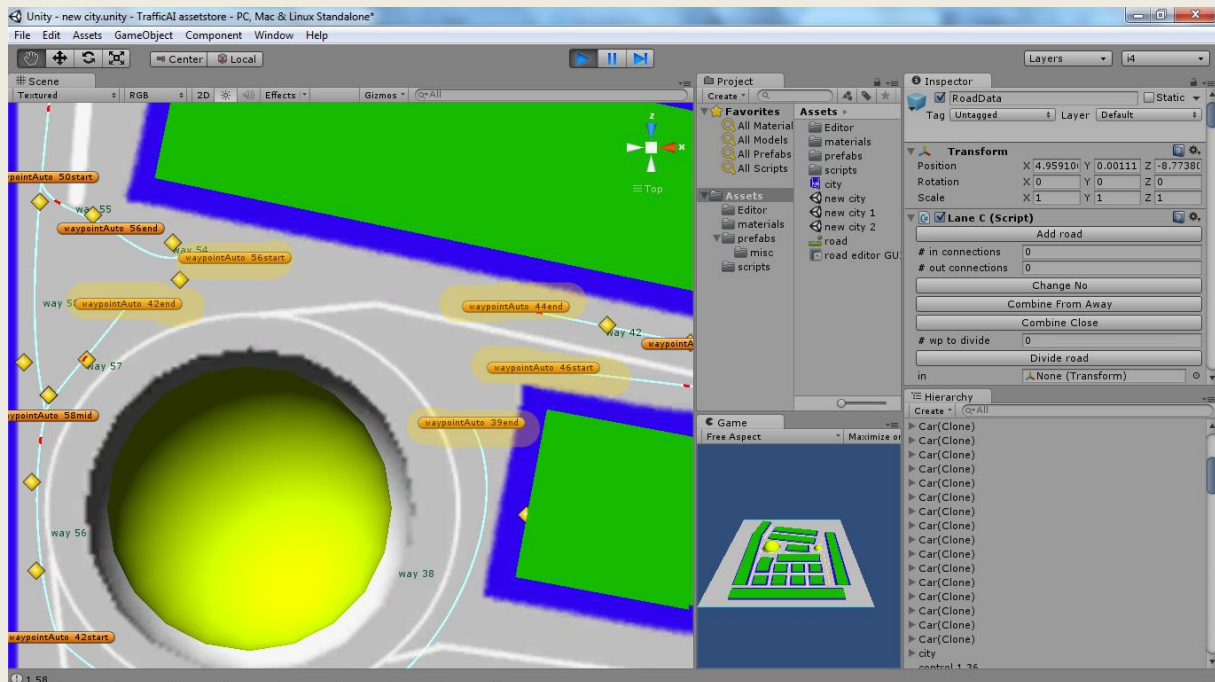




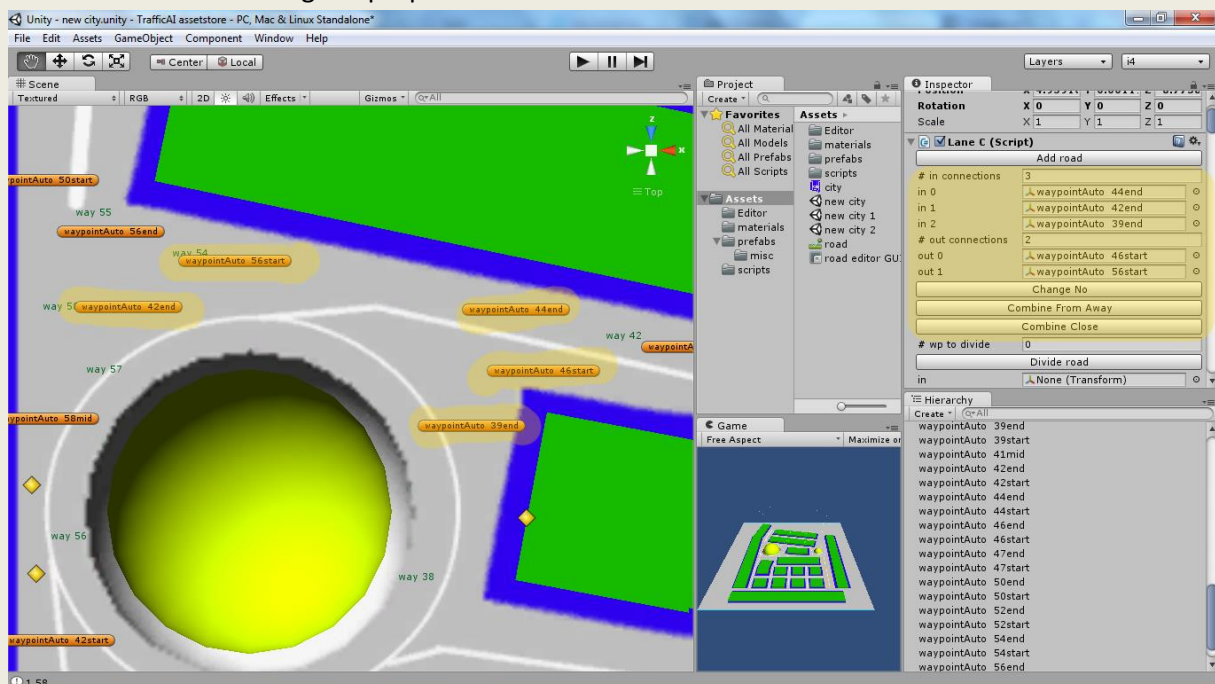
Now we will close the upper loop by placing necessary lanes as highlighted below:



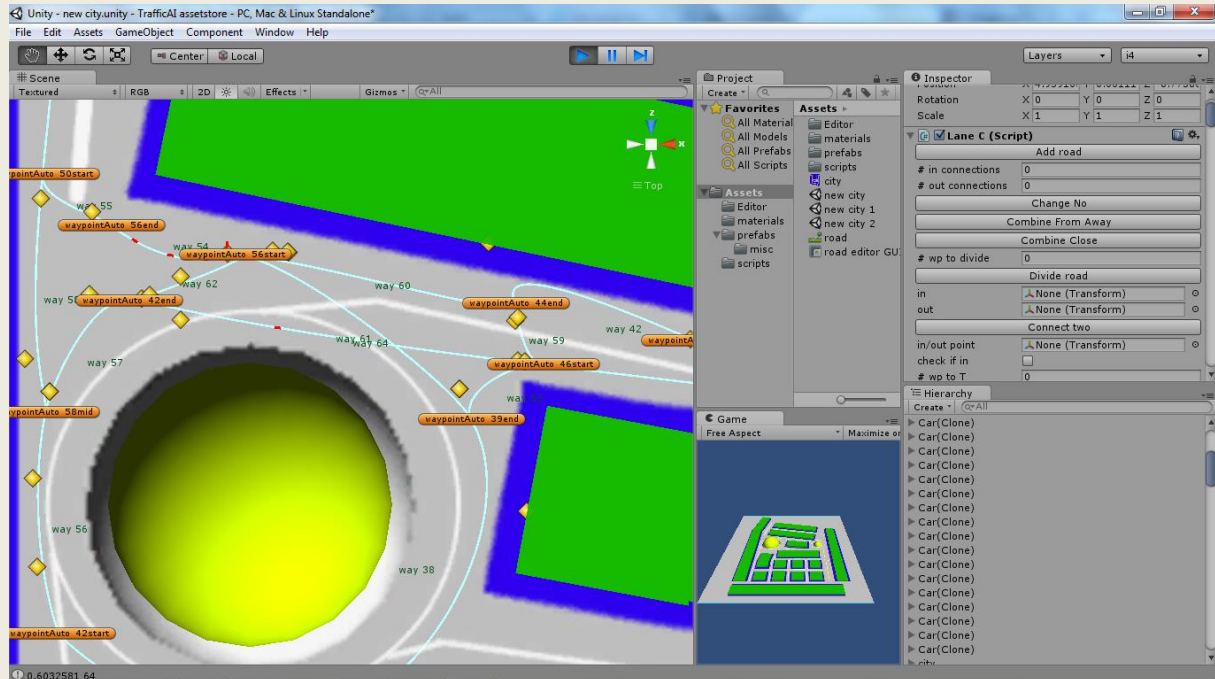
As you may have noted, we left 5 lanes unconnected. Connecting them by Connect two button can be a bit time consuming. We will use another technique for that.



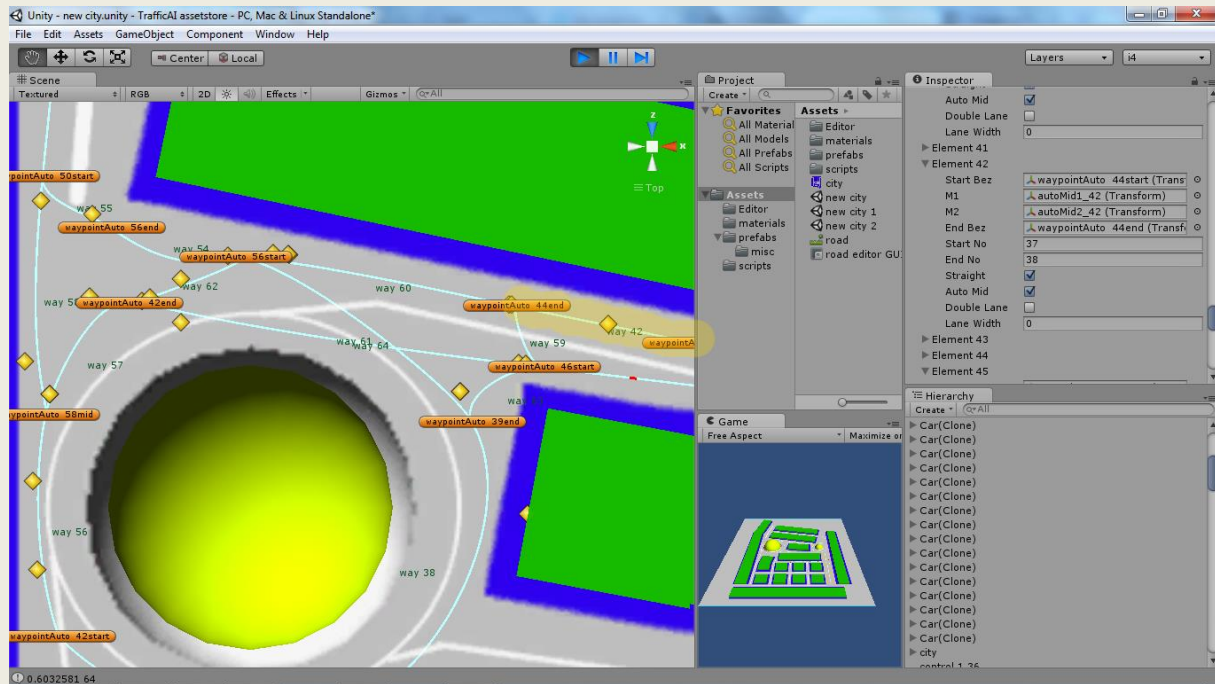
We will use the **Combine Close** button. For this button to work, we need to input all the incoming and outgoing connections properly. We have 3 in 2 out connections. We write “3” to “# in connections” and “2” to “# out connections” and click **Change No**. This creates the proper number of slots for us. Then we drag the proper connections to slots:



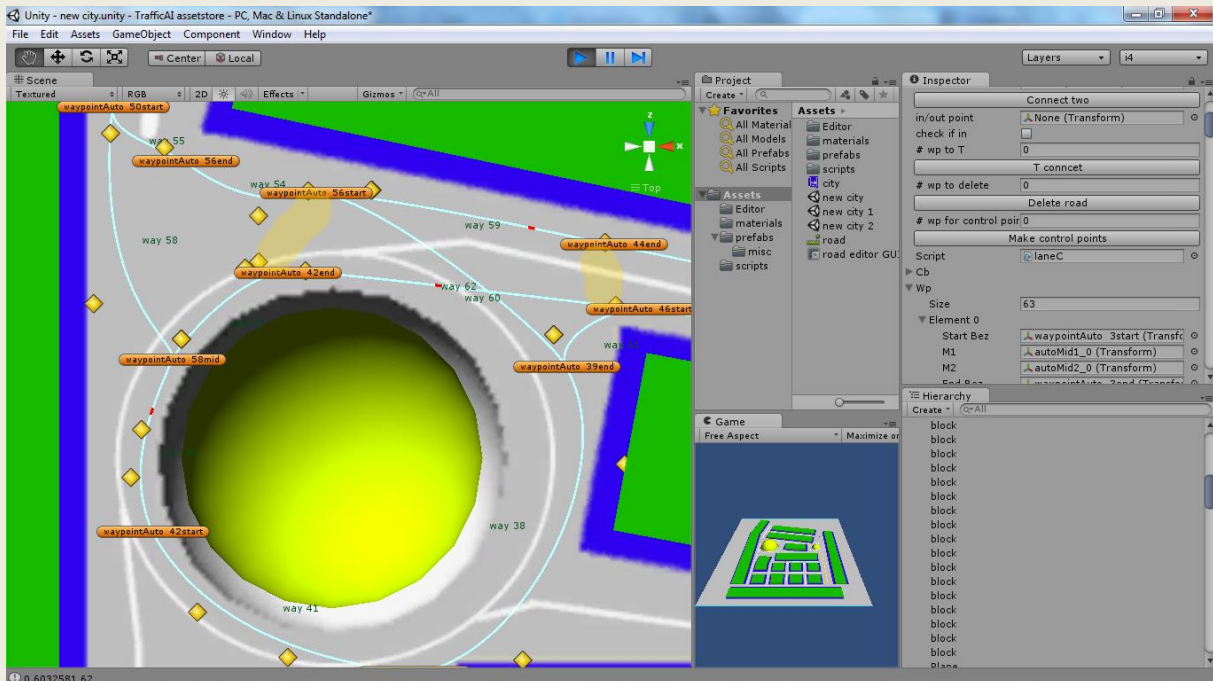
When we click the **Combine Close** button, system forms all the necessary lanes:



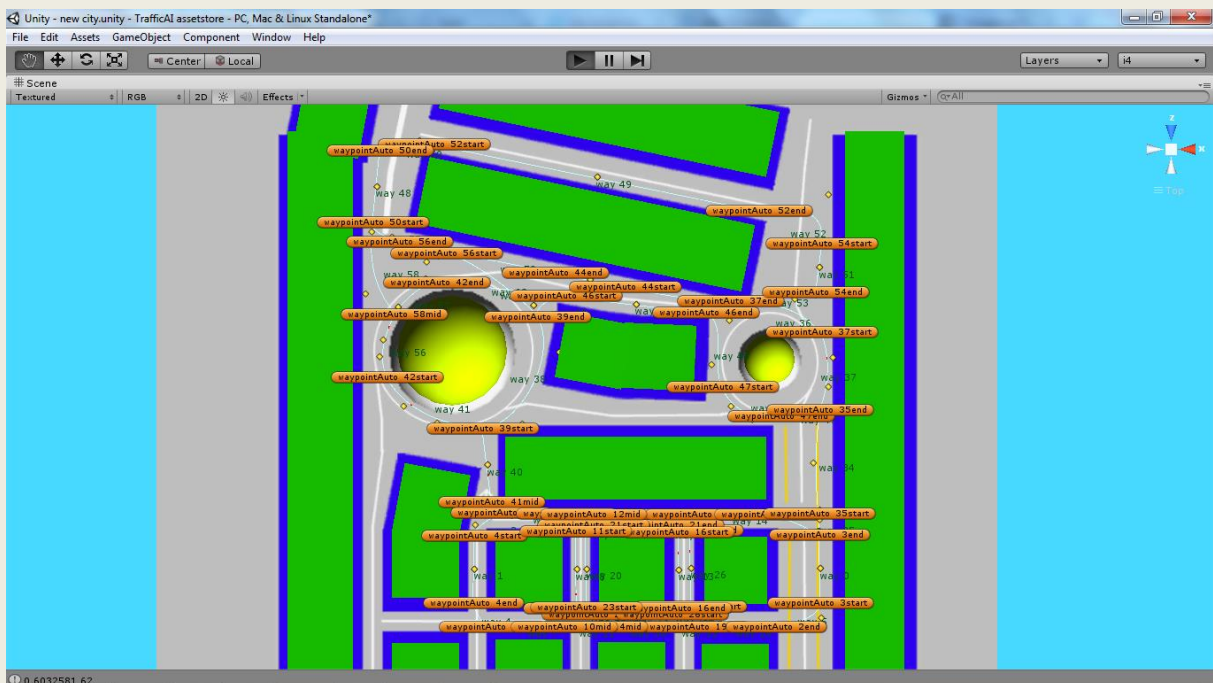
We note that one lane needs straightening:



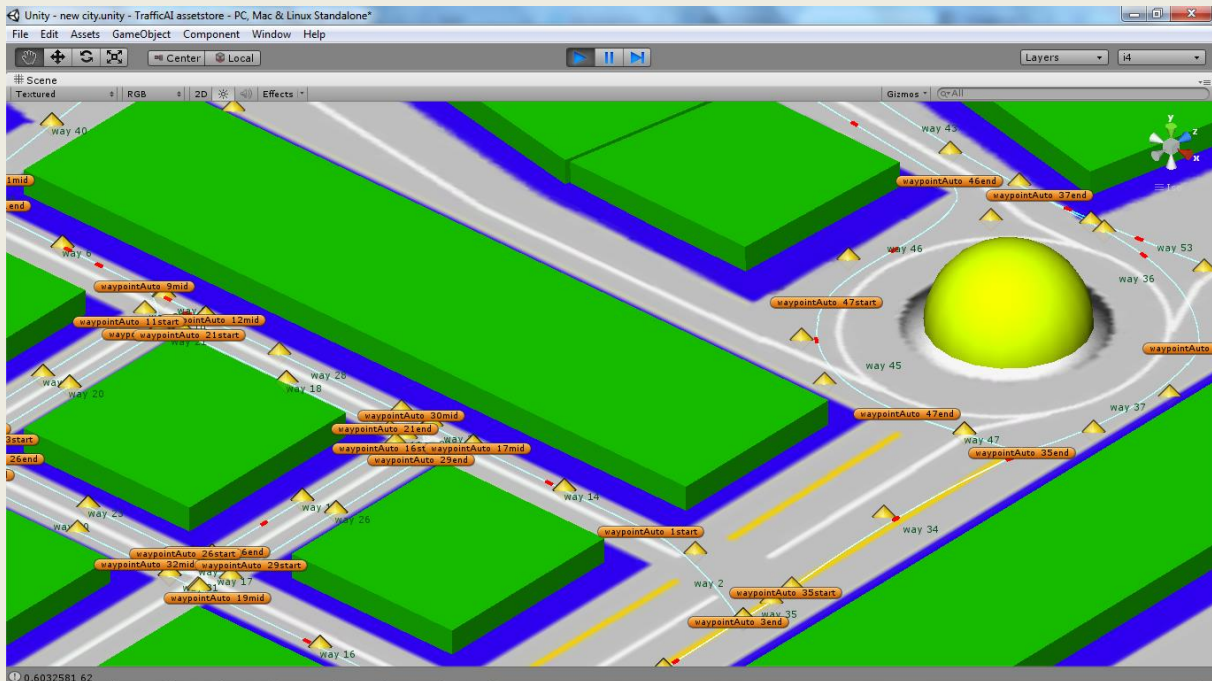
Also two lanes in directions we do not want traffic to flow, thus we delete them. **Combine Close** forms lanes between all incoming and outgoing connections. It forms $\#in \times \#out$ many lanes. For this sample it forms $2 \times 3 = 6$ lanes. As it cannot know where we don't want traffic, which is usually where we don't allow U turns, we may need to delete some.



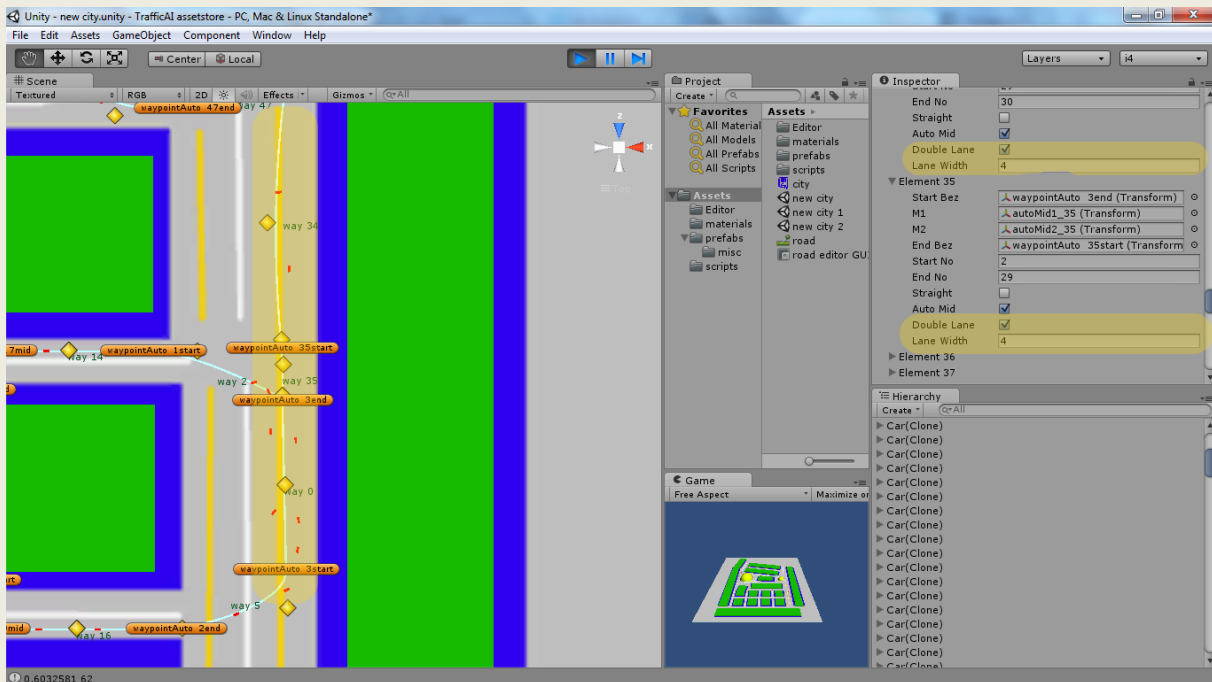
This is the final traffic network we have built:



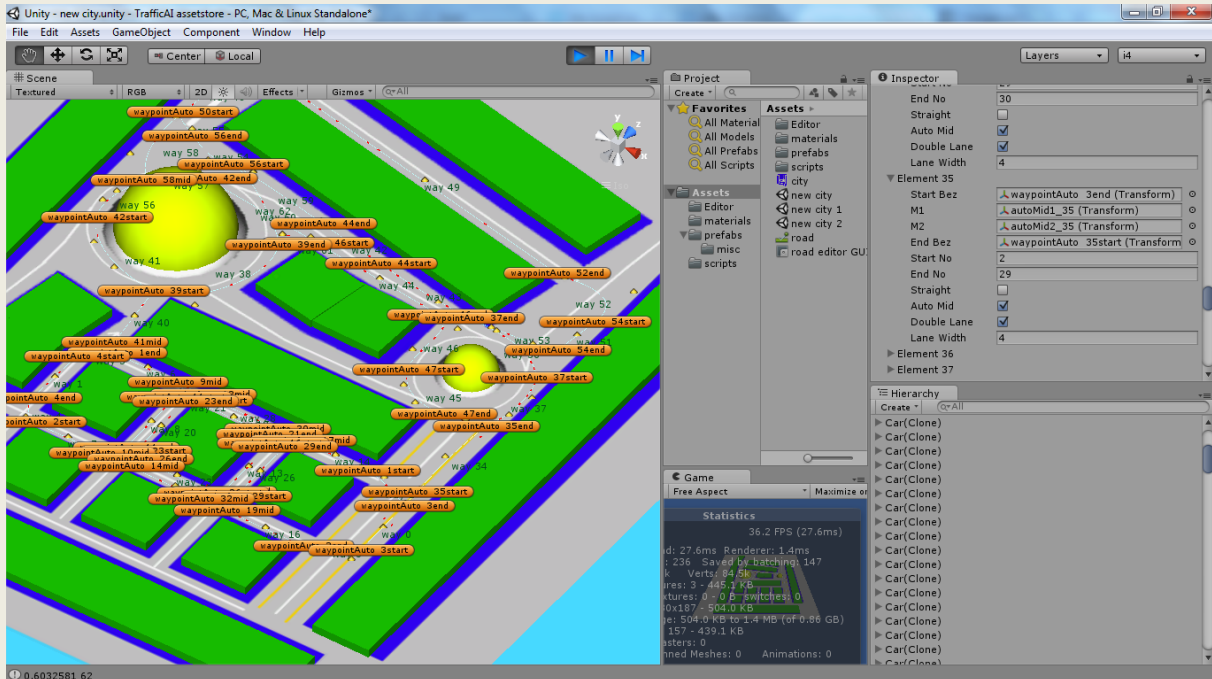
An orthogonal view:



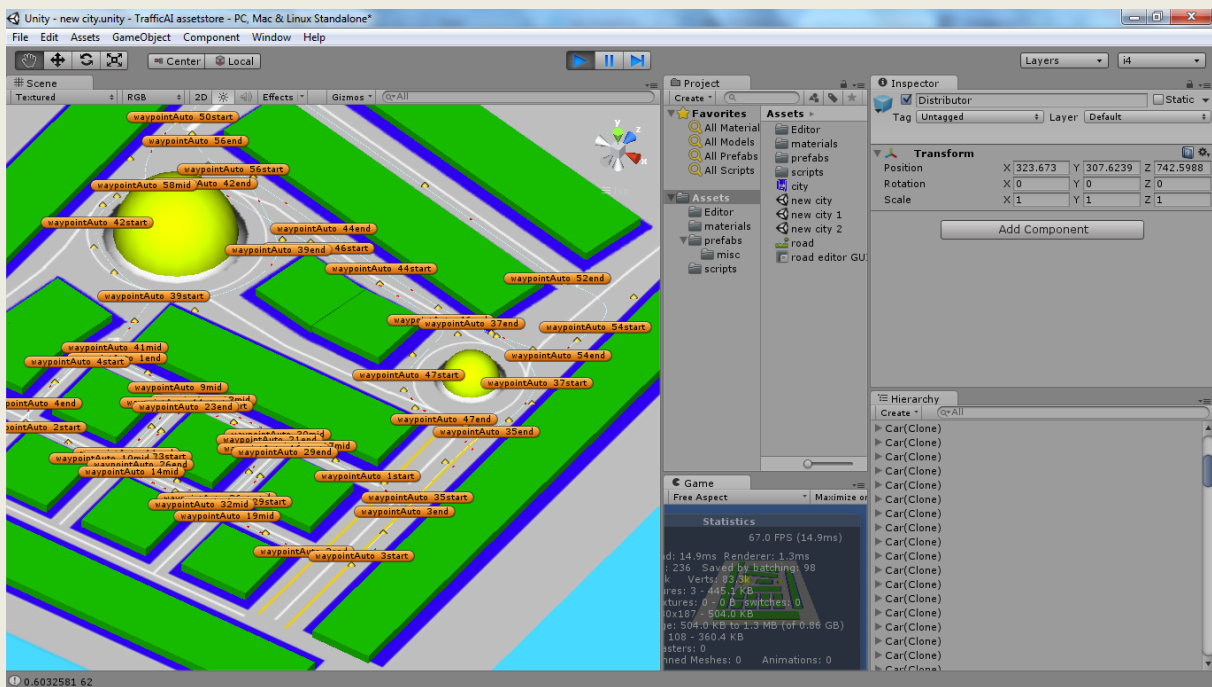
You may note that the rightmost road is very wide; it is intended to be a multi-lane one. Currently Ti Traffic AI supports double lanes. To enable double lanes find the lane in “Wp” array and choose the **Double lane** toggle. Also input the width of the lane. In double lanes, cars can change lanes freely.



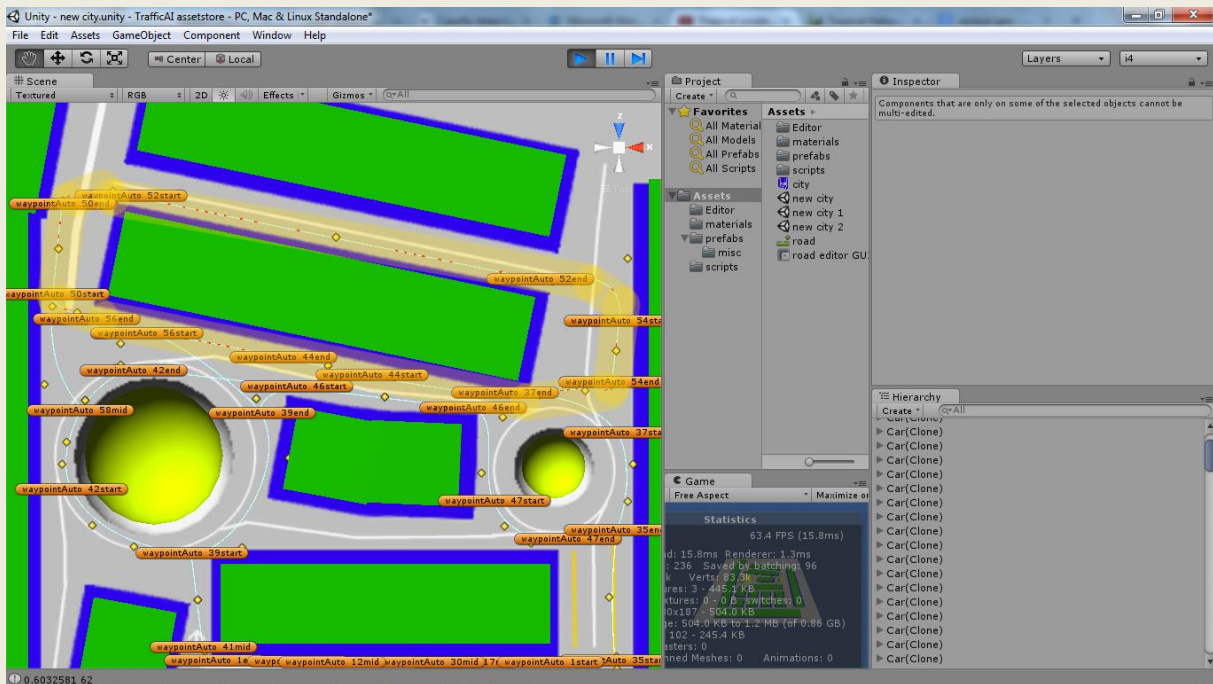
Another orthogonal view. We have 200 cars in the system, to demonstrate the capabilities of it:



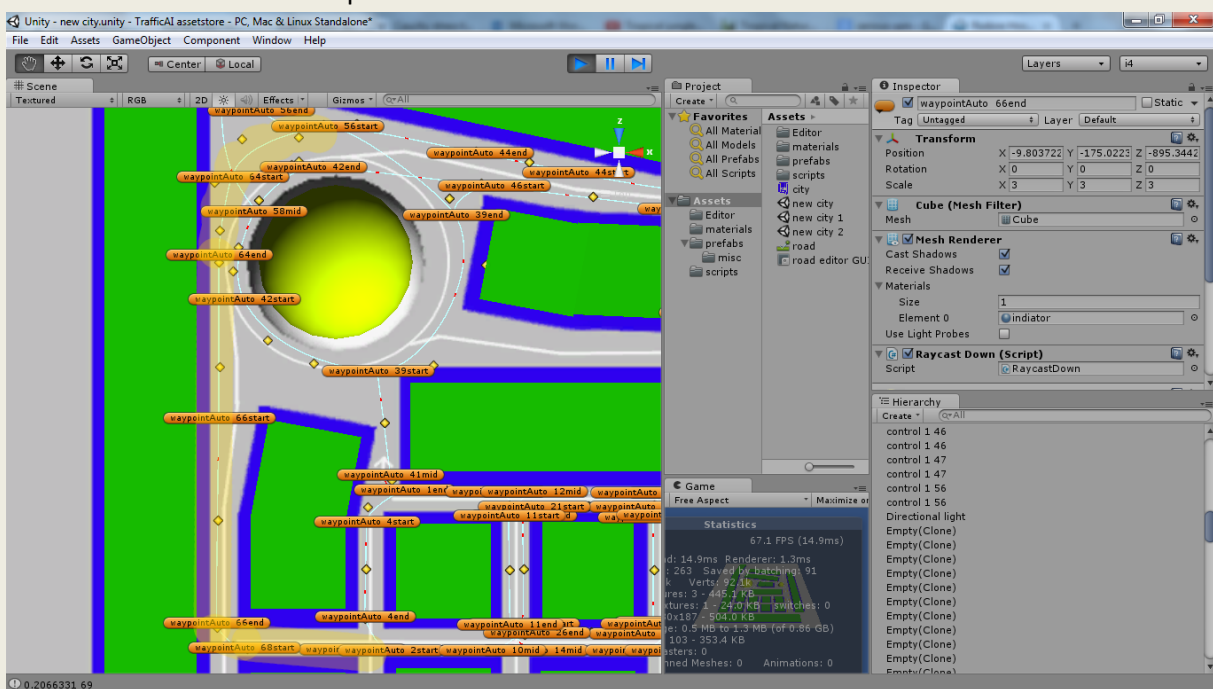
100 cars:



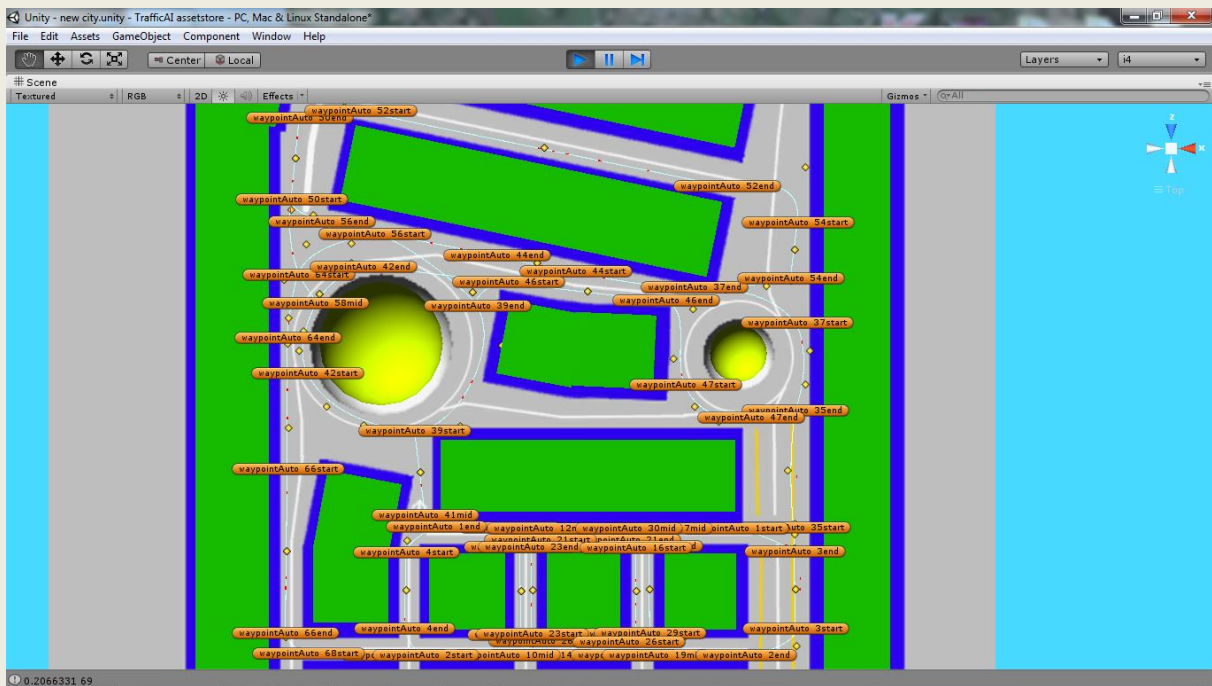
After running the traffic for a few minutes we note that cars accumulate on the uppermost loop. Closer inspection reveals that there is no way out of the loop.



We add a lane out of the loop:



We have a balanced traffic flow now:



Part II – How it works

Ti Traffic AI has three parts.

1. A lane part that uses Bezier curves, connects them and adjusts them.
2. An editor part that makes laying down lanes easy.
3. Car AI that follows the lanes.
4. A traffic light system
5. Pedestrian system

The Bezier curves are the usual curves used everywhere in the industry.

The editor part is rather complex and its usage is detailed in the tutorial section.

Car AI is a physics control system that makes the cars drive around while obeying laws of physics. It works by acting on Unity 3D wheelcolliders. It changes the torque, brake and steering angle to control the car. Thus it is just like a regular user driven car. It obeys all physics in the game engine. If a boulder hits it will drag it, if an explosion force acts on it, the car will fly, if it hits a box, car will lose speed and box will fly away. Beauty of making physics based things is these things happen naturally, without extra coding.

The car tries to follow the lane by changing its steering angle based on its distance to the lane, its relative angle, the rate of change of distance and angle. Each of these has coefficients that can be adjusted based on your cars weight, wheel distribution, power etc. The adjustment process is more of a trial and error.

The **Distributor** gameobject located inside Road Data distributes cars in the start. You can write your own distributor if you have different needs. In the **Distributor** there is a car prefab slot. Whatever car you drag there will be distributed. By changing the properties of that car prefab you can control the way your car behaves.

There are a number of parameters to be tweaked in the “Distribute Cars And People” scripts:

maxTorque is trivial, it is the maximum torque the car has.

maxSpeed is the maximum speed.

speedTweak reduces torque with speed.

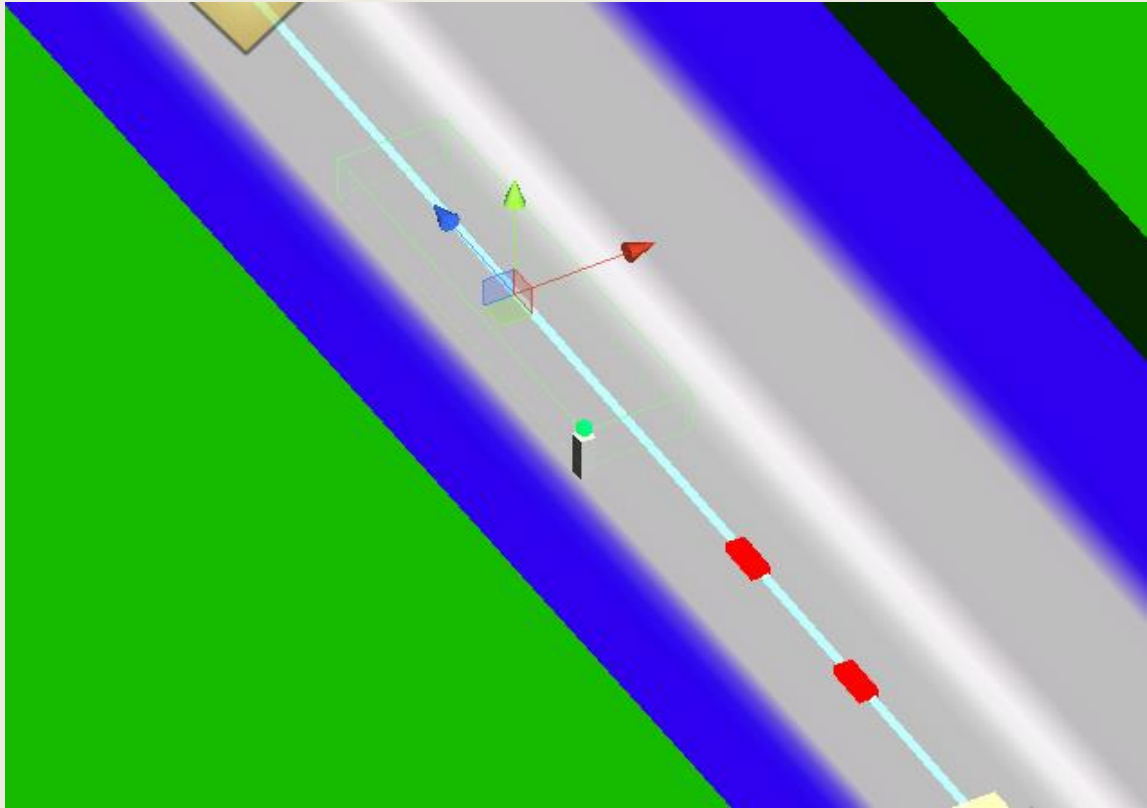
kDeviation (seen as Deviation in inspector) is the amount of steering with distance from lane.

kAngle (seen as Angle in inspector) is the amount of steering with relative angle from lane direction.

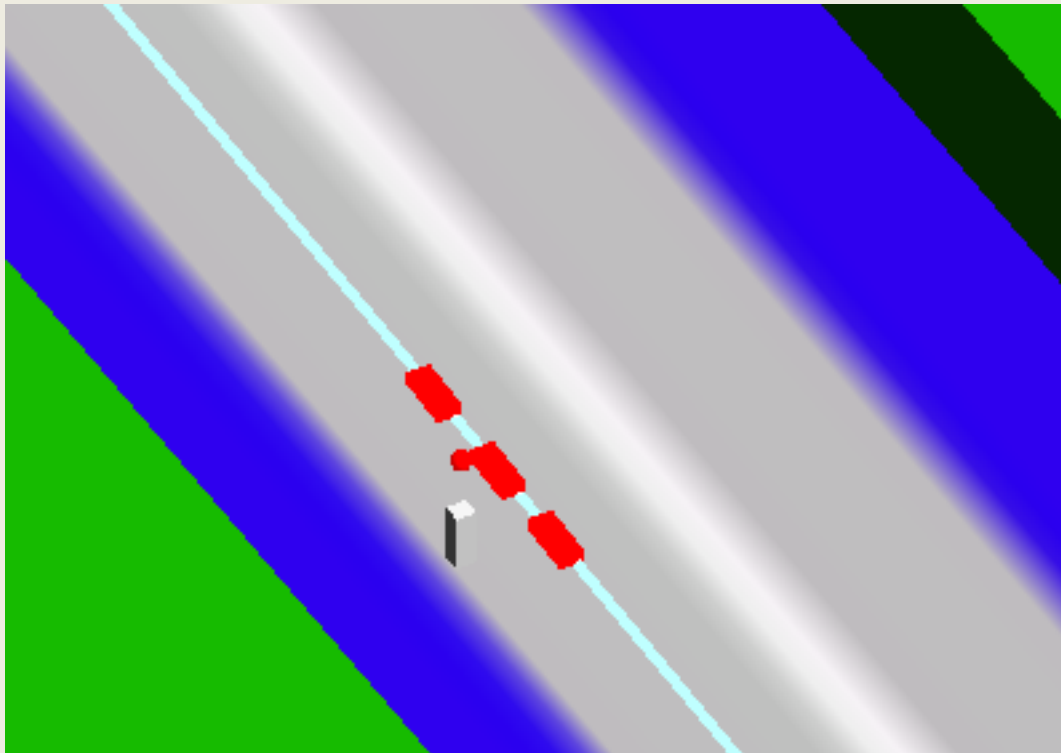
kDeltaAng (seen as DeltaAng in inspector) is the amount of steering with rate of distance change. Should be used to cure oscillation around lane line, especially at high speed.

kDeltaDev (seen as DeltaDev in inspector) is the amount of steering with rate of angle change. Should be used to cure oscillation around lane line, especially at high speed.

There is also a traffic light system in the package. It should be placed as such that the trigger covers a part of the flowing traffic on one side. Cars that enter this trigger start getting the state of the light, and they act accordingly. As provided the light just has a box bottom and 3 lights that appear in order. You can apply your own appearance to the light.



Cars stopping at red light:



The pedestrian system can be used by simply dragging the pedestrian zone to the scene and adjusting the waypoints. In the pedestrian zone gameobject you will see the gameobject `waypointCross`. It shows the direction to cross the street. The pedestrian units raycast when they are crossing the street, thus they tend not to collide with incoming traffic.