Project 4 Urban: Traffic
CS 4803 PG, Spring 2018

The goal of this assignment is to create procedurally generated traffic. Unlike other projects in this course, this one depends on an earlier project selection: you must use the procedurally generated streets from Project 1. Your virtual automobiles will obey standard traffic laws: they will drive on the right side of the roads, they will come to a stop at intersections, and they will avoid colliding with other vehicles.

Note that this project must be done individually. There is no group option for this assignment.

Due Date and Grading

This project is due on Wednesday, April 26 at 11:55pm. Each day late after the due will cause your grade on the project to drop by 5%, and we will not accept the project more than three days beyond the due date. This project will have a 15% "effort" component. If your project meets all of the listed criteria, you will get at least 85% credit on this project. The final 15% of your grade will be based on our judgment of whether you showed creativity and care in your project.

Authorship Rules

Each student must work on this assignment on their own (no group projects). Seek out the TA's and the instructor for help with Unity. You may **not** use code from other sources, including code from the web, from videos, from Github, from books, or from other students.

Project Description

You will write a procedural traffic system in Unity. When your program starts, you will have a number of vehicles on your streets that are moving and obeying traffic laws.

You must have completed the Streets version of Project 1 in order to do the traffic project. You will build your traffic simulator as an extension of your streets project.

It is not required that you use vehicles from Project 3. That is, you do not have to use detailed vehicle geometry for this traffic project. Something simple that is made out of a few primitives is fine as the body of your vehicles.

Characteristics of your procedural traffic:

- Demonstrate virtual **traffic** along your synthetic streets.
- Have your cars always stay on the **roads**.
- Make sure your cars travel on the **right side** of the virtual streets. Each street should be capable of carrying traffic in both directions.
- Include these three types of **tiles** in your streets: straight, turn, and 4-way intersections.
- Make your cars move in smooth **arcs** when they turn.
- Cause all cars to come to a complete stop before proceeding at each 4-way intersection.
- Allow only one car to proceed at a time at a 4-way intersection.
- Include user-editable **number of cars**, and create or remove cars when this variable is changed. You may wish to clamp this to never exceed some high value (e.g. 200).
- When the user presses the space bar, cause the cars to jump to random street locations.

Your streets should be entirely made of two-lane roads (one lane in each direction). That way, your traffic simulator does not need to handle issues such as lane changes.

Note that you do not need to include dead end streets and 3-way intersections in your road layouts. Including these would add extra complications in implementing traffic, and this is beyond the scope of the current project.

Effort

Part of the grade for this project is our judgment of "effort". The majority of the effort score will be based on how well your vehicles move. Do the vehicles stop and start abruptly, or do they smoothly accelerate and decelerate? Do the vehicles all move at exactly the same speed (which looks artificial), or do they all have slightly different desired velocities? Do any cars pass through each other?

Possible Additions (Not Required)

Below are some ideas about possible extensions to your traffic. These are not required elements, but are ideas that you may consider if you feel inspired to go beyond the project basics.

- Include dead ends in your streets. Have a car at a dead end turn around using a 3-point turn.
- Include 3-way intersections.
- Have some 4-way intersections be controlled by a traffic light.
- Allow some 4-way intersections have only two stop signs (on opposite sides), so that one direction (e.g. north/south) does not need to stop.
- Include garages with driveways, and have cars occasionally enter or exit a garage.

Suggested Approach

You can follow my suggestions or not, depending on your level of confidence about programming.

You should begin by working with just one car on a very simple road. This simple road should consist of one closed loop (four turns, four straight stretches of road). Later, you will add more cars and 4-way intersections.

You may wonder how you can create a road that has only 4-way intersections, and that entirely avoids 3-way intersections. Look at the Olympic Rings for inspiration. Create several closed loops of roads, and make sure that pairs of loops cross each other at 90 degrees. The term "loop" should be interpreted broadly as any stretch of street that eventually comes back and re-joins itself.

Each car should be a GameObject. You should also have a GameObject that manages the traffic itself, perhaps called Traffic. This procedural animation GameObject for traffic should maintain a list of cars, and may create or remove cars if the user changes the desired number of cars on the road. This GameObject probably should be in charge of updating all of the cars' positions at a given Update step, especially since it may have to decide which car has the right-of-way at a given intersection.

As you will recall from Project 1 (Streets), your street representation should consist of various tiles. The straight and turn tiles should each have two queues of cars. For example, a north-south running straight street should have a queue for northbound traffic, and another queue for southbound traffic. A queue in one direction might be empty, may contain just one car, or may contain two or more cars.

When you begin with just a single car on the simple street, your main task is to write the algorithm for moving a car forward along its current stretch of road. When the car reaches the end of a tile, you should consult a look-up table that tells you which queue the car should be placed in when it enters the new tile. I recommend giving each kind of tile a number (0-3 for turns, 4 for intersection, 5 & 6 for straight). I also recommend labeling each queue of a tile with either 0 or 1 in a consistent manner for all tiles of a given type. For example, each north/south straight tile should label the north side as being 0 and the south side as being 1. That way, you can create a table that says, for a given origin tile, destination tile, and origin

queue, which destination queue the car should be place into. Using such a table will allow you to avoid the headache of dozens of "if" statements to figure out which queue to move the car into.

Once you have one car moving around the simple loop of streets, you are ready to add multiple cars. When considering the motion of a given car, your algorithm should find out whether there is another car in front of it. If this is the case, the rear car may need to decelerate in order to avoid running into the lead car. If there is no car in front, then the car should slowly accelerate to bring it closer to its desired velocity.

Once you have multiple cars going around a simple loop without colliding, you should turn your attention to 4-way intersections. All of the intersections in this project should be 4-way stops. That is, every car should come to a complete stop at each intersection. The first car to the intersection has the right-of-way, and only that car can proceed. Once that car has left the tile, then other cars can go. Each car will select at random whether to go straight, turn right, or turn left at a given intersection. Note that the motion of a car as it moves through an intersection will be exactly the same as if the intersection is either a straight tile, a right turn tile, or a left turn tile. This means you should be able to re-use your code that you wrote for moving a car when it is moving along a straight or turn tile.

Additional Rules

As with each of the projects in this course, all of the objects in your scene should be created by you from within Unity. During this project, you may be tempted to use another program to create a polygon mesh for your vehicle. Do not do this! You should **not** include game assets that have been made using other programs (Maya, Blender, etc.), nor should you include assets from the Assets Store. The one exception to this rule is that you can load texture image assets (including a sky box). You may also create such textures using a paint program or by taking photos.

Turning in the Project

You will turn in your project on T-square. To do this, first create an empty folder that is called your name. Next, determine where Unity stores your projects, find the directory that contains your project, and copy its two sub-directories called Assets and ProjectSettings into the empty folder. Zip up the folder that contains Assets and ProjectSettings, and submit this zipped file to T-square. Do not create a .rar or .tar file – please create a .zip file. If your zipped file is more than a few megabytes in size, you probably copied too many sub-directories, and in particular the Library sub-directory should **not** be included because of its large size.