# Traffic Visualization Project Report

## Abstract

In this report we look at our attempt the visualize a system, more specifically a traffic system around a roundabout, using Autonomous agents. We will be creating Traffic Lights for it to listen to and paths for it to leave out of without interruption from incoming cars. All of this will be shown only when the user clicks the 'start' toggle on the screen and the user can leave the simulation by clicking the 'escape' toggle screen.

## Introduction

Craig Reynolds "Steering Behaviour" [1] are a fascinating and diverse field which can be used to visualize a wide range of topics. They work by giving the agents certain rules to follow (these are usually forces) and the agent will dynamically do the rest. These forces include "seek", "follow", "separate" etc. They work by Craig Reynolds simple formula:

**Steer Force = Desired location – Current Velocity**

This means we simply must calculate how much a vehicle must steer and apply the forces to the vehicle for it to move to a desired location.

I decided to create autonomous agents which followed the typical rules humans use when driving i.e. The Traffic System. To recreate an entire traffic network would be a huge task however so therefore I decided to simply focus on one component of traffic – a 4-way roundabout.

A roundabout will require multiple paths in and out of the roundabout, traffic lights and a steady stream of cars being generated and removed once they leave the screen.

## Requirement Analysis

In order to create the traffic system there is multiple component we need to be developed. These are:

1) Vehicle – This should stay on a path and when reaches the roundabout, go around it, and go to the next path. This should follow the traffic lights, have a distance from other vehicles. These should be done using PVectors and the idea that acceleration affects velocity which affects position. We should also know which path the vehicle is on and which path it was generated on.

2) Roundabout – This must make any vehicle on the roundabout go in a circle until it reaches the new path. Vehicle come from an incoming path and goes out to an outgoing path.

3) Roads – This should have a start point and end point. We will have to use the normal point between the path and the predicted position of the vehicle to see if the vehicle is an acceptable distance away from the centre of the path. This should be saved as the "radius" and can be used to draw the path. There should be 4 outgoing paths and 4 incoming paths. Each path will have an id to recognize it.

4) Traffic lights – This will be switching from on to off every few seconds and will go in order. There will be one traffic light for each outgoing path.

5) The Simulation which controls the logic – In this class we should initialize the 4 traffic lights, the 8 roads, generate the vehicles into an ArrayList, control the order which the traffic lights switch, have the logic deciding what the vehicle does when it reaches the end of the road, delete the vehicle from the ArrayList when it hits the border, control the cars blinkers etc.

6) The Main Interface – Over here we will use the ControlP5 library to create a start screen with a start button, a slider to control the number of vehicles generated on a road and an escape button to get out of the simulation. The purpose of the start screen is so the user has more control over the visualization and to create an interface which with the user can use to adjust the simulation.

Knowing this we will have to create 5 classes in processing on top of the main tab.

Fortunately, a template for a few of these classes are already available in "The Nature of Code" Github Page [3]. I will need to modify those classes to suit my need.

## Design

Based on the Requirement Analysis we will create the following classes:

1) Vehicle Class
2) FlowField Class
3) Path Class
4) Light Class
5) Simulation Class
6) Main Class

### Vehicle Class

We used PVectors to store the position, velocity and acceleration of the vehicle. Additionally, we also stored the path the vehicle is on, the path it was generated on, the maximum speed, the maximum force, the radius of the vehicle and its colour.

This is a rather important class as it decides the rules the vehicle follows so there are a bunch of important methods to consider:

- The “update” method updates the position the vehicle is on based on the velocity and acceleration.
- The “display” method draws the vehicle.
- The “borders” method checks if the vehicle is off the screen.
- The “run” methods does all of the above
- The “applyForces” method uses the formula f = m*a. We add the forces into acceleration by rearranging it to a = f/m where m is 1 for all vehicles and we therefore don’t consider m, making f = a.
- The “seek” method uses Craig Reynolds “steer = desired – velocity” formula to return the force needed to go to a certain point.
- The “follow” method check if the vehicle is on the path using “getNormalPoint” method. If it is we “seek” the end of the path, if not we “seek” the path.
- The “separate” method checks if there are any other vehicles around itself and gives a force to avoid them. We make the desired separation very high in order to have each car give each other a good separation distance.
- The “reachedEnd” method checks if we have come towards the end of the path.

### The Other Classes

The FlowField class is what will make the Vehicle go around the roundabout as the field force should be a spiral around a certain radius. Everywhere else there will be no force in the FlowField. This will be done by making every vector in the FlowField point to the center using trigonometry (atan(diffy/diffx)) and rotated by -1*HALF_PI.

The Path Class contains its starting point and ending point, a radius and its id.

The Light Class should hold the Traffic Lights data such as whether It is on or off, its coordinates and its rotation.

The Simulation Class creates contains the constructor for all the previous mentioned classes, draws them and uses various methods to run the entire traffic simulation.

Its methods include:

1) generateCars – which adds cars to an Arraylist at the starting point of any of the 4 paths incoming paths id the traffic light is green
2) newPath – which selects a path randomly from any of the 4 outgoing paths. In addition to this it also uses nested switch statements in order to turn on the blinkers of a car based on the path it came from and the path it is going to go to.
3) trafficLogic – should switch the traffic light from one to another. There should be a small pause between when one light is switched off and the next is switched on where all the lights should be off. We use the modulus operator (%) in order to get a number which repeats.
4) go – This is the method that makes the cars move. The rules which the cars follow should be in here. We constantly apply the force of the FlowField, call the applyBehaviors method and the run method. If the cars reached the end, we calculate a new path and switch on the blinkers. After a small time where the blinkers is on we must switch it off. And finally, when the car hits the border and goes off the screen we remove the car from the ArrayList.
5) Run – This is the main method of the Simulation class. It constantly displays the path, lights, cars etc. This method also calls generateCars constantly, takes the value from trafficLogic and decides which light to switch on/off and checks if the car is on a path with a green light or not. If it is we call go, if not we stop the car.

And finally, the main class will use the controlP5 Library to create a Start screen, Simulation screen, a reset method and a slider which determines how many cars will be generated from each road.

## Implementation

In creating the vehicle class, we were able to get a template [3] and adjust it slightly to create a good vehicle class. In addition to this creating the FlowField, Path and Light class were rather straightforward in its implementation.

In order to create the simulation class, we had to make several adjustments and design decision.

I realized that in order to make the blinkers turn off after a small timeframe I must add some additional values into the vehicle class which checks when the blinkers turned on.

One Major design decision I made was that if I generated a car and allowed it to reach the end of the road when the traffic light was red, the vehicles would overlap with each other at the end. Due to the timeframe I had to the deadline I decided that I would only generate vehicle when the light was green. This has the disadvantage of making visualization less realistic, but once you have many cars going around it becomes less noticeable.

I decided to us 'frameCount' rather than 'millis()' because the framerate would drastically decrease when I was saving each frame to create a video to output, making the traffic light switch very fast compared to the vehicles which moved very slowly. Therefore, to make the time difference the same no matter the frame rate I switched to frameCount.

Implementing the main interface had mixed results. It was a success in terms of functionality where I was able to give a user adjustable slide which influences the number of car generate, however a bug arose which made the car appear to be blinking in certain situations. Due to the deadline I was forced to ignore this, however I would like to point out that this bug didn't arise when the controlP5 interface was absent.

## Evaluation

By the end of the of the project I was pleased with how the project came out. The autonomous agents quite accurately mimicked traffic and there were no 'game breaking' bugs. However, there is certainly several areas that could be improved upon.

Ideally, I would make vehicles reach the end of the road if the traffic light was red and then stop.

On top of that I could have made the blinkers stop a few frames after it reaches a new path rather than after a set time frame.

I would also have enjoyed to fix the bug where cars appear to be blinking.

Finally, I would have liked to make an outside loop and inside loop for the roundabout so that vehicles will be less likely to collide with each other. In order to stop vehicle collision, I made the 'desiredDistance' in my separation function 10 time the radius of my car but there are still some rare occurrences when cars appear to overlap.

## Conclusion

In conclusion we were able to use the concept of Reynolds [1] to create a visualization of traffic around a simple roundabout using autonomous agents. We had to use a FlowField, and multiple forces to make the agents follow the traffic rules. The interface I made with the ControlP5 objects adds a good layer of functionality and offers the user some control over the visualization. However there were many possibilities left to explore such as making the vehicles reach the end of a road.

There is also potential to expand this visualization further by creating a larger road with multiple cars on it, next to each other. We could link this to a larger chain of roads and roundabouts to make a complete traffic system throughout a city. We could add different types of vehicles, with different masses, different speeds, forces etc.

## References

1. Reynolds, C. (1999). *Steering Behaviors For Autonomous Characters*. [online] Red3d.com. Available at: https://www.red3d.com/cwr/steer/gdc99/ [Accessed 27 Mar. 2019].

2. Shiffman D. The Nature of Code [Internet]. Natureofcode.com. 2012 [cited 27 March 2019]. Available from: https://natureofcode.com/book/

3. Shiffman D. The Nature of Code [Internet]. GitHub. 2012 [cited 27 March 2019]. Available from: https://github.com/nature-of-code