To Alleviate Traffic Congestion

Siyuan "Daniel" Chen - Captain, Programmer
Yifan "William" Wu - Programmer, Art Design
Andrew "Drew" Kirk - Programmer, Researcher
Sam Monaghan - Data Analyst, Researcher

“

By making each driver adjust his/her speed logically, one can alleviate traffic congestion without using traffic lights.

Part 1.
Introduction

1.1 The Problem

Traffic congestion: the accumulation of motor vehicles at an intersection due to the suboptimal spacing between them.

Irrational human behaviours: adjustments to vehicle speeds that do not necessarily benefit individual vehicles nor the whole system.

1.2 The Goal

Reduce the extent of traffic congestion at intersections by **changing** how **individual** drivers **accelerate their vehicles** based on the **limited information** they obtain.

1.3 The Assumptions

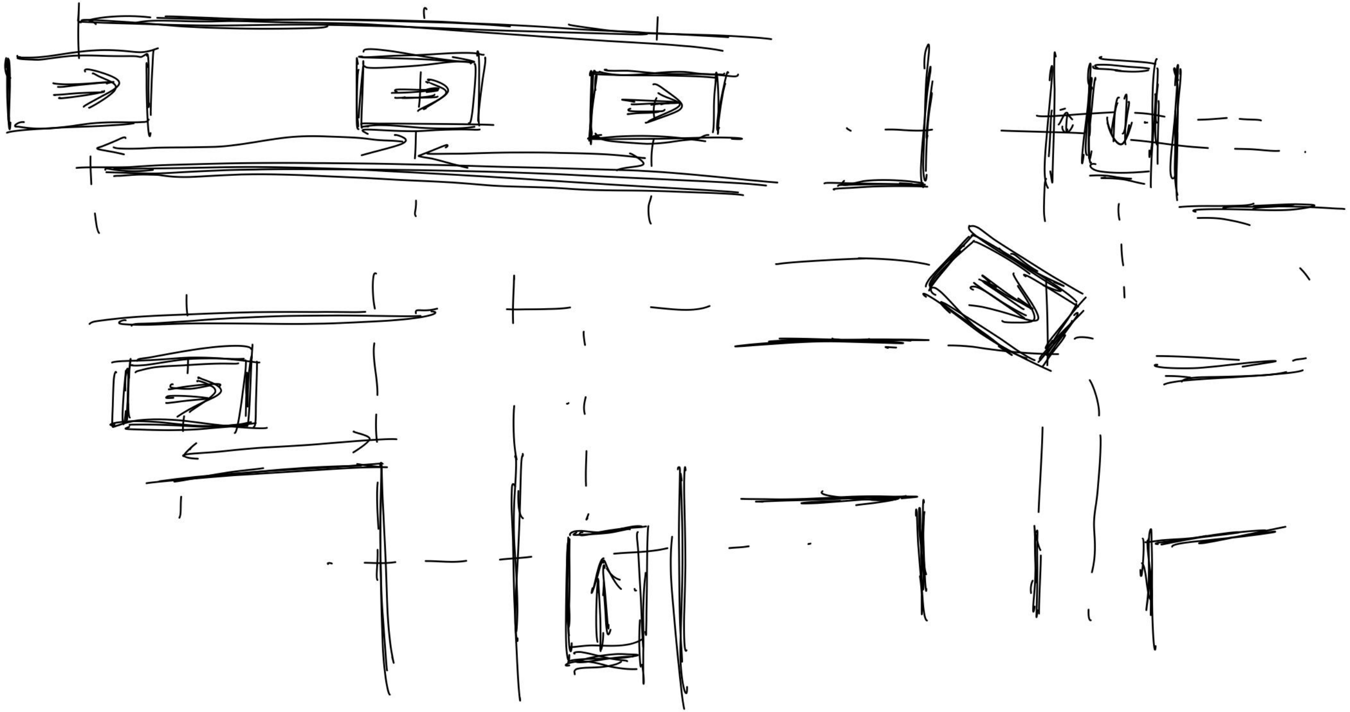
See **Handouts**.

- **Relevance** of the control group data.
- **Validity** of the outcomes of the simulation.

Part 2.
Method

2.1 The Algorithm

1. Find the closest vehicle **in front**. Record the distance.
2. Repeat Step 1 with the closest vehicle **behind**.
3. Adjust acceleration, if possible, based on the **difference** of the two distances.



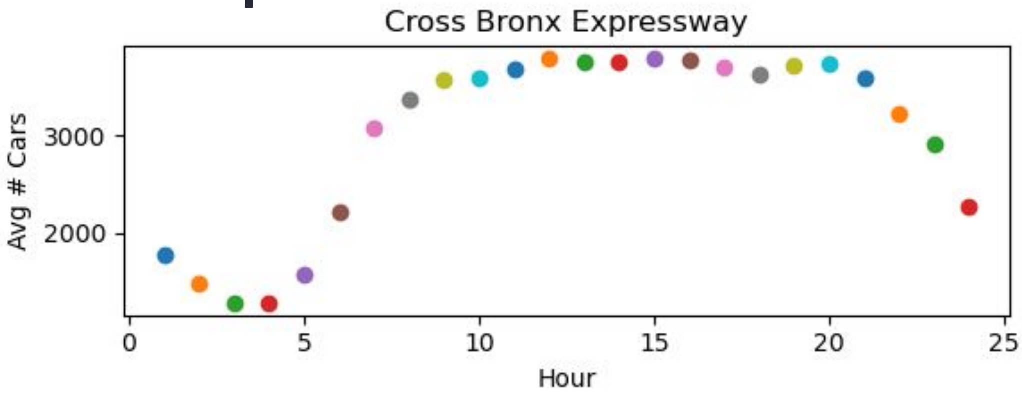
2.1 The Algorithm – Diagram

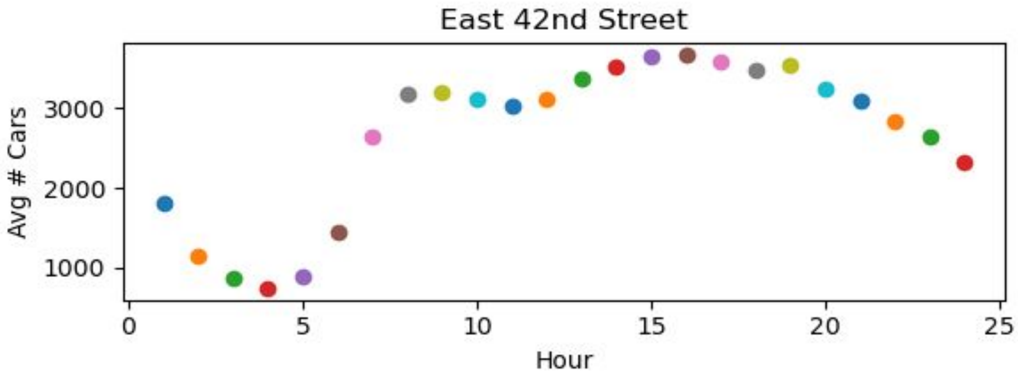
2.2 The Control

ID	Segment ID	Roadway Name	From	To	Direction	Date	12:00-1:00 AM	1:00-2:00AM	2:00-3:(
2	70376	3 Avenue	East 154 Street	East 155 Street	NB	09/13/2014	204	177	
2	70376	3 Avenue	East 155 Street	East 154 Street	SB	09/13/2014	140	51	
56	176365	Bedford Park Boulevard	Grand Concourse	Valentine Avenue	EB	09/13/2014	94	73	
56	176365	Bedford Park Boulevard	Grand Concourse	Valentine Avenue	WB	09/13/2014	88	82	
62	147673	Broadway	West 242 Street	240 Street	SB	09/13/2014	255	209	
62	158447	Broadway	West 242 Street	240 Street	NB	09/13/2014	255	209	

NYC DOT: Number of cars that pass through a street/an intersection each hour from 2013 to 2018

2.2 The Control - Graphs





2.2 The Control - Steps

1. Take **heavily congested** intersections from the dataset
2. Retrieve 5 hours where traffic congestions are the most **apparent** for that street
   a. Top 5 to represent the typical allotted rush hours
3. Inject the number into our model and **compare** the time it takes to consume that congestion.

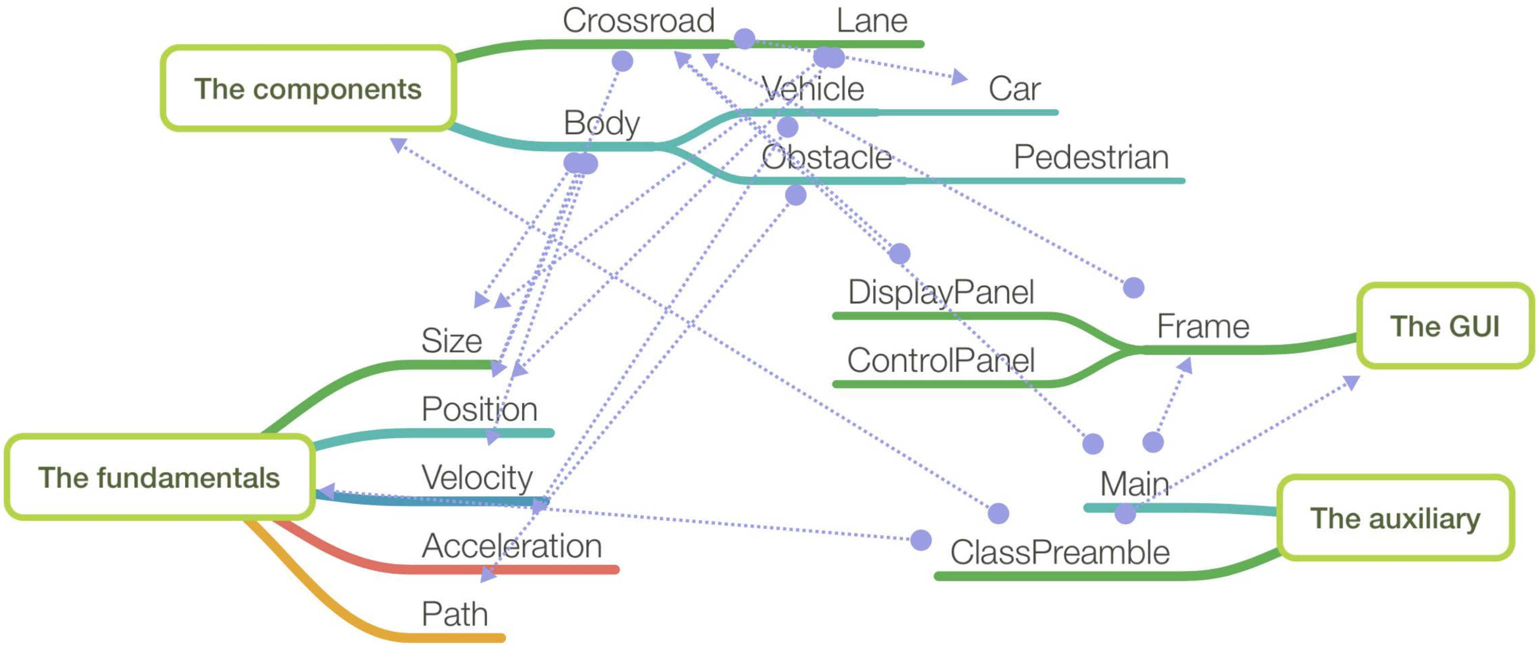
2.3 The Simulation

Four layers: frame, (panel), crossroad, vehicle

- Frame -> (panel): initUI()
- displayPanel -> crossroad: passTime(), timeElapsed
- crossroad -> vehicle: passTime(), getAccelerationFor()

2.3 The Simulation - Details

- Calculating individual acceleration
  - Determining limited information
  - Comparing distance
- Spawning and cleaning vehicles
- Updating vehicle positions



The components
Crossroad
Lane
Body
Vehicle
Car
Obstacle
Pedestrian
The fundamentals
Size
Position
Velocity
Acceleration
Path
The GUI
Frame
DisplayPanel
ControlPanel
The auxiliary
Main
ClassPreamble

2.4 The Code

```
public Acceleration getAccelerationFor(int index) {

    if(isTurning(index)) {
        states.set(index, 0);
        return this.getAccelerationTurningFor(index);
    }

    if(states.get(index) == 0) states.set(index, 1);

    return getAccelerationStraightFor(index);
}
```

2.4 The Code - Screenshots

```java
public Acceleration getAccelerationTurningFor(int index) {

    Vehicle vehicle = vehicles.get(index);

    if((vehicle.getOrigin() + vehicle.getDestination()) % 2 == 0) return new Acceleration( magnitude: 0,  orientation: 0);

    if((vehicle.getDestination() - vehicle.getOrigin() - 1) % 4 == 0) {
        return new Acceleration( magnitude: Math.pow(vehicle.getVelocity().getMagnitude(), 2) / (laneWidth * 1.5),
                orientation: vehicle.getVelocity().getOrientation() - Math.PI / 2);
    }

    if((vehicle.getDestination() - vehicle.getOrigin() + 1) % 4 == 0) {
        return new Acceleration( magnitude: Math.pow(vehicle.getVelocity().getMagnitude(), 2) / (laneWidth * 0.5),
                orientation: vehicle.getVelocity().getOrientation() + Math.PI / 2);
    }

    return null;
}
```

```java
public Acceleration getAccelerationStraightFor(int index) {

    int laneNum = getLaneNum(index);
    boolean isAlong = laneNum % 2 == 0; // true: horizontal, false: vertical
    Vehicle vehicle = vehicles.get(index);

    ArrayList<Double> allPositions = new ArrayList<>();

    boolean centerOccupied = false;

    for(int i = 0; i < states.size(); i++) {

        if(getLaneNum(i) == laneNum || lanes[laneNum].inRange(vehicles.get(i), needAll: false))
            allPositions.add(vehicles.get(i).getPosition().getPosition(isAlong));

        if(states.get(i) == 0) centerOccupied = true;

    }

    if(centerOccupied) allPositions.add(virtualBlock(vehicle.getOrigin()));

    double absolutePosition = vehicle.getPosition().getPosition(isAlong);

    Collections.sort(allPositions);

    int relativeIndex = allPositions.indexOf(absolutePosition);

    double frontGap;
    double backGap;

    if(laneNum > 1) {
        frontGap = absolutePosition - (relativeIndex > 0 ? allPositions.get(relativeIndex - 1) : -Double.MAX_VALUE);
        backGap = (relativeIndex < allPositions.size() -1 ? allPositions.get(relativeIndex + 1) : Double.MAX_VALUE / 2) - absolutePosition;
    } else {
        frontGap = (relativeIndex < allPositions.size() -1 ? allPositions.get(relativeIndex + 1) : Double.MAX_VALUE / 2) - absolutePosition;
        backGap = absolutePosition - (relativeIndex > 0 ? allPositions.get(relativeIndex - 1) : -Double.MAX_VALUE / 2);
    }
```

```java
    if("map.Car".equals(vehicle.getClass().getName())) {

        // If there is no vehicle in front then accelerate to max speed if possible

        if(frontGap > RANGE_OF_INTEREST) {
            if (vehicle.getVelocity().getMagnitude() < Car.MAX_VELOCITY_MAGNITUDE) {
                return new Acceleration(Car.MAX_ACCELERATION_MAGNITUDE, vehicle.getVelocity().getOrientation());
            } else {
                return new Acceleration( magnitude: 0,  orientation: 0);
            }
        }

        // Given that there is a vehicle in front, decelerate if there is no vehicle at back and if possible

        if(backGap > RANGE_OF_INTEREST) {
            if(vehicle.getVelocity().getMagnitude() > Car.MAX_ACCELERATION_MAGNITUDE * Main.INTERVAL) {
                return new Acceleration(Car.MAX_ACCELERATION_MAGNITUDE,  orientation: Math.PI + vehicle.getVelocity().getOrientation());
            } else {
                return new Acceleration( magnitude: 0,  orientation: 0);
            }
        }

        // Given that there are vehicles both in front and at back, balance

        if(backGap > frontGap) {
            if(vehicle.getVelocity().getMagnitude() > Car.MAX_ACCELERATION_MAGNITUDE * Main.INTERVAL) {
                return new Acceleration(Car.MAX_ACCELERATION_MAGNITUDE,  orientation: Math.PI + vehicle.getVelocity().getOrientation());
            } else {
                return new Acceleration( magnitude: 0,  orientation: 0);
            }
        } else {
            if (vehicle.getVelocity().getMagnitude() < Car.MAX_VELOCITY_MAGNITUDE) {
                return new Acceleration(Car.MAX_ACCELERATION_MAGNITUDE, vehicle.getVelocity().getOrientation());
            } else {
                return new Acceleration( magnitude: 0,  orientation: 0);
            }
        }

    }

    return null;
}
```

2.5 The Videos

Slider **mechanism**: click here.

Simulation **highlights**: click here.

Simulation **bugs**: click here.

Part 3.
Results and Analysis

3.1 The Output

Output below is of two of the highest traffic congested roadways in New York

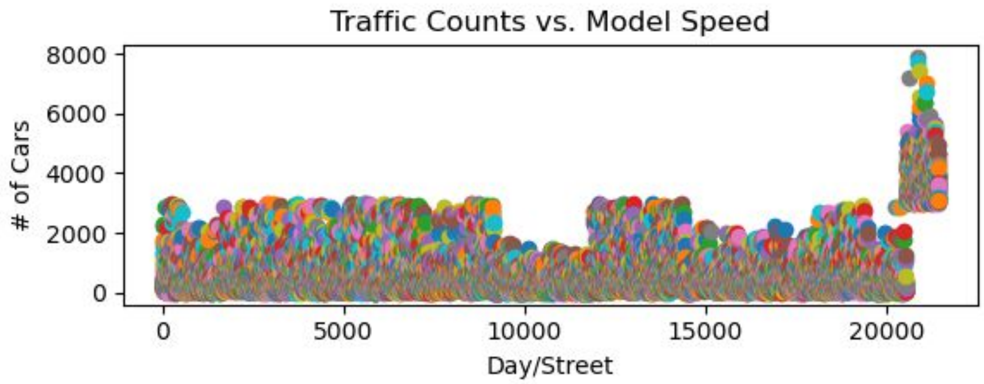
```
Average of top 5 hours for traffic counts
Cross Brox Expressway:
3020.5311111111105

Average of the top 5 traffic counts
East 42nd Street
2889.766666666667
```

3.2 The Comparison

- Control group output: **2890** cars per hour
  - Most congested intersections at rush hours
- Simulation group output: **3000** cars per hour
  - 300 cars per 360 seconds on average, ten trials

3.3 The Comparison - Graph

From 2013-2018, 95.79% of intersections during rush hour from 8am to 9am had less than 3000 cars pass through the intersection (our model benchmark)



Part 4
Conclusion

4.1 The Limitations

- Intermittent collisions occur.
- Intermittent back-driving occurs.
- Does not account for complex real world factors such as sudden surges of vehicles.
- Is limited to two lane intersections.

4.2 The Conclusion

- Simulation
  - Very flexible, few hard-coded parameters
  - Could be used for **future research**
- Control
  - Valuable data
  - Could be used for **future research**