Explanation of the code used will be in this appendix. It will be structured by going over the code piece by piece and explaining its’ functionality and logic. It is done this way to facilitate anyone trying to understand how and why something works or is made in a particular way. Code will simply be copy/pasted due to fears of OneNote not handling pictures properly. Code snippets will be separated by dotted lines from comments. Firstly, the code will be shown and explanation will be below it.

The solutions starts off by defining all variables:

--------------------------------------------------------------------------------------------------------------------------------

globals [

selected-car ; the currently selected car

lanes ; a list of the y coordinates of different lanes

]

turtles-own [

speed ; the current speed of the car

top-speed ; the maximum speed of the car (different for all cars)

desired-speed ; speed driver wants to drive

target-lane ; the desired lane of the car

patience ; the driver's current level of patience

speed-difference

experience ; how experienced the driver behind the wheel is

mistake-chance

perfect-speed ; the perfect is the speed the driver should be driving at (inexperienced drivers may need to drive slower than they want to)

crashes

heavy-crashes

mistakes

no-mistake

]

--------------------------------------------------------------------------------------------------------------------------------

Globals are the variables that are shared throughout all objects within the solution, or as the name suggests – they are global. A turtle is a NetLogo agent type which is used in this project for the car entity. Each have their own, private variables, which are – speed (current speed the car is travelling at), top-speed (the maximum allowed speed), desired-speed (the speed the driver wants to drive at), target-lane (the lane in which the driver is aiming to be in), patience (used in earlier versions of the simulation, used to be as the name implies for driver patience), speed-difference (used to store the difference in speed with the car ahead), experience (a value of how experienced the driver is), mistake-chance (a variable to hold the value of the chance for a mistake at this particular moment), perfect-speed (the speed the driver should be aiming for, which may differ from the desired speed), crashes & heavy-crashes (counting crashes), mistakes (variable which holds the number of mistakes made), no-mistake (holds the number of times not made a mistake).

--------------------------------------------------------------------------------------------------------------------------------

to setup

clear-all

set-default-shape turtles "car"

draw-road

create-or-remove-cars

set selected-car one-of turtles

ask selected-car [ set color red ]

reset-ticks

end

--------------------------------------------------------------------------------------------------------------------------------

This is a standard setup method within NetLogo. It clears everything, sets the default shape of the turtles and calls the methods draw-road, create-or-remove-cars. Sets the selected-car to one of the turtles randomly and turns it red. It resets the ticks as well since it’s starting a new session. This method is executed when the “Setup” button in the interface is clicked.

--------------------------------------------------------------------------------------------------------------------------------

to create-or-remove-cars

; make sure we don't have too many cars for the room we have on the road

let road-patches patches with [ member? pycor lanes ]

if number-of-cars > count road-patches [

set number-of-cars count road-patches

]

create-turtles (number-of-cars - count turtles) [

set color car-color

move-to one-of free road-patches

set target-lane last lanes ; 1 is left lane -1 is right lane , pycor is default value

set heading 90

set top-speed 0.6

set speed 0.2

set patience random max-patience

set desired-speed 0.4 + random-float 0.2 ; set to a random value, 0.7 is for testing purposes!

set speed-difference 0.0

set experience 0.85 + random-float 0.15 ; experience of each driver is a random value of 0.9 to 1.0

set mistake-chance 0 ; debugging purposes

set perfect-speed desired-speed \* experience

set mistakes 0

set no-mistake 0

]

if count turtles > number-of-cars [

let n count turtles - number-of-cars

ask n-of n [ other turtles ] of selected-car [ die ]

]

End

--------------------------------------------------------------------------------------------------------------------------------

The first part of the method makes sure we don’t have too many cars on the road. If not, it creates the required amount of turtles and sets their default values. These start off with setting the car color by calling the “car-color” method. It then moves the newly spawned car to a free patch of road which won’t necessarily be on the right side – spawning happens everywhere so we can have many cars. It sets its target lane, which is the right lane, so every car that isn’t in the correct lane tries to go back to it. Sets heading to 90, which is the direction in which the car will be facing. It is very important to be facing the correct way, since cars move forward and look forward. It then sets the top speed, which works like a motorway speed limit. It then sets the speed of the spawned car, which is at a low 0.2 just so they start off slow and not crash into each other in the first couple of frames/ticks. It then sets the desired-speed to a value of 0.4 to 0.6. The desired speed is the speed the driver is aiming to drive at. It then goes on to set the speed-difference variable to 0.0 for debugging purposes. Then follows the experience, which is negatively related to mistakes and bad habits. It then sets the mistakes-chance variable to a value of 0.0 for debugging purposes. It then goes on to set the perfect speed to the desired one multiplied by the experience. This is done on order to have inexperienced drivers wanting to drive at a speed higher than they can handle. It finishes by setting the mistakes and no-mistakes variables to 0. In the end, the script checks if there are more cars on the road than requested and deletes random ones if there are.

--------------------------------------------------------------------------------------------------------------------------------

to-report free [ road-patches ] ; turtle procedure

let this-car self

report road-patches with [

not any? turtles-here with [ self != this-car ]

]

End

--------------------------------------------------------------------------------------------------------------------------------

To-report is a function in NetLogo which is a report procedure. It reports if there are any free road patches.

--------------------------------------------------------------------------------------------------------------------------------

to draw-road

ask patches [

; the road is surrounded by green grass of varying shades

set pcolor green - random-float 0.2

]

set lanes n-values number-of-lanes [ n -> number-of-lanes - (n \* 2) - 1 ]

ask patches with [ abs pycor <= number-of-lanes ] [

; the road itself is varying shades of grey

set pcolor grey - 2.5 + random-float 0.25

]

draw-road-lines

end

--------------------------------------------------------------------------------------------------------------------------------

Draws the world. It paints everything green to simulate grass and then paints the amount of lanes requested with grey.

--------------------------------------------------------------------------------------------------------------------------------

to draw-road-lines

let y (last lanes) - 1 ; start below the "lowest" lane

while [ y <= first lanes + 1 ] [

if not member? y lanes [

; draw lines on road patches that are not part of a lane

ifelse abs y = number-of-lanes

[ draw-line y yellow 0 ] ; yellow for the sides of the road

[ draw-line y white 0.5 ] ; dashed white between lanes

]

set y y + 1 ; move up one patch

]

End

--------------------------------------------------------------------------------------------------------------------------------

This method calculates the position to draw the white and yellow lines on the (grey) road. It finds the last lane and starts from its end. “Y” difference of 1 is half a lane, so it decreases the “y” value by 1 to draw it at the correct spot. As an example, if you switch “y” by 1 from 1 to 0 for a turtle, it will be exactly between the two lanes. It then calls the draw-line script to draw the lines at the correct spots.

--------------------------------------------------------------------------------------------------------------------------------

to draw-line [ y line-color gap ]

; We use a temporary turtle to draw the line:

; - with a gap of zero, we get a continuous line;

; - with a gap greater than zero, we get a dasshed line.

create-turtles 1 [

setxy (min-pxcor - 0.5) y

hide-turtle

set color line-color

set heading 90

repeat world-width [

pen-up

forward gap

pen-down

forward (1 - gap)

]

die

]

End

--------------------------------------------------------------------------------------------------------------------------------

Creates an object which will go through the road in a straight line and draw a line along the path.

--------------------------------------------------------------------------------------------------------------------------------

to go

create-or-remove-cars

ask turtles [ calculate-mistake-chance ]

ask turtles [ move-forward ]

;ask turtles with [ patience <= 0 ] [ choose-new-lane ]

ask turtles with [ ycor != target-lane ] [ move-to-target-lane ]

ask turtles [ collision-check ]

tick

end

--------------------------------------------------------------------------------------------------------------------------------

This is a common NetLogo method. It is executed when the “Go” button in the interface is clicked and gets executed every single tick. It does a couple of things. Firstly, it checks if there are any cars to be added or removed. Then it goes on into “ask” methods. These work by “asking” the turtles or cars to execute certain methods. The “;” before one of the lines makes it commented out, so it doesn’t execute. So the first two lines execute the script to calculate the chance for a mistake and to move the turtle forward (which is where most of the behavior is). It then asks the turtles that have a target lane of a different lane than their current one to execute a method that will change their lane. It then checks for collisions and finishes with the tick/frame.

--------------------------------------------------------------------------------------------------------------------------------

to calculate-mistake-chance

set mistake-chance 0 ; debugging purposes, just in case

let speed-to-perf 0

;if above perfect speed - increase mistake chance!

if speed > perfect-speed

[

set speed-to-perf abs (speed - perfect-speed) ; absolute value for debugging purposes, shouldn't make a difference ever

]

let speed-to-desired abs (speed - desired-speed) / 25 ; speed != desired speed isn't as bad

set mistake-chance ((speed-to-perf + speed-to-desired) / experience) / 30

end

--------------------------------------------------------------------------------------------------------------------------------

This method calculates the chance for a mistake. Just to be sure, it starts off with setting the chance to 0, so that’s what it’s working with. It creates a local variable with the “let” command called “speed-to-perf” with a value of 0. It then checks if the speed is higher than the perfect speed. The idea of the perfect speed is that it’s the speed at which the driver can control the vehicle. So if the current speed is above that, the speed-to-perf variable is used as a multiplier in a formula to increase the chance for a mistake. Even though the number will always be positive, for debugging purposes/just to make sure, the absolute value is derived. Then the speed-to-desired local variable is the value of the difference between the speed and the desired speed. This is an absolute value because it may be positive or negative, because the driver may be driving slower than they wish or faster, but the effect should be the same for both – increased chance for a mistake. Finally, the mistake-chance variable is created by adding the two local variables together and dividing them by the experience. This will mean that if the experience is lower, the mistake chance will be higher (for example: 1 / 0.9 = 1.11). The final result is divided by 30, because there are 30 ticks per second, so if it isn’t divided by 30 the issue will be that for 1 second behind another vehicle the AI driver will have 30 chances of making a mistake that is 1% chance for example. Which wouldn’t account for a 1% mistake chance. By diving by 30, that is somewhat countered.

--------------------------------------------------------------------------------------------------------------------------------

to move-forward ; turtle procedure

set heading 90

speed-up-car ; we tentatively speed up, but might have to slow down

let cars-ahead other turtles in-cone 10 180 with [ y-distance <= 1 ] ; 1 + speed is distance before we start braking, 180 is cone angle

let car-ahead min-one-of cars-ahead [ distance myself ]

ifelse car-ahead != nobody [set speed-difference abs (speed - [speed] of car-ahead)] [set speed-difference speed]

let blocking-cars other turtles in-cone (2 + speed-difference \* (6 \* experience)) 180 with [ y-distance <= 1 ] ; 1 + speed is distance before we start braking, 180 is cone angle

let blocking-car min-one-of blocking-cars [ distance myself ]

if blocking-car != nobody

[

if ycor = 1

[

let me self

let dist [distance blocking-car] of self

if dist < (1 \* experience)

[

slow-down-car

]

ifelse speed > [speed] of blocking-car

[

;set speed [ speed ] of blocking-car

slow-down-car

]

[

speed-up-car

]

]

if ycor = -1

[

set heading -60

let mistake random-float 1 ; generate random number to decide if AI makes mistake or not

ifelse mistake > mistake-chance

[

set no-mistake no-mistake + 1

let car-other-lane other turtles in-cone (8 - speed \* 4) 180 with [ ycor = 1] ; removing the speed because if you drive fast you wouldn't care to look too far behind

let cars-other-lane min-one-of car-other-lane [distance myself]

if cars-other-lane = nobody

[

if speed > [speed] of blocking-car

[

; if speed difference is too big, slow down a bit to not crash into car ahead

ifelse speed \* 1.8 > [speed] of blocking-car

[

;set speed [speed] of blocking-car \* 1.8

slow-down-car

set target-lane first lanes

set color pink;

]

[

set target-lane first lanes

set color pink;

]

]

]

if cars-other-lane != nobody

[

let me self

let dist [distance blocking-car] of self

if dist < (1 \* experience) ; drivers with less experience keep more dangerous distances (lower) to car ahead

[

slow-down-car

]

ifelse speed > [speed] of blocking-car

[

slow-down-car

]

[

speed-up-car

]

]

]

[

set target-lane first lanes

set color white;

set mistakes mistakes + 1 ; count mistakes

]

set heading 90

]

]

if blocking-car = nobody

[

if ycor = 1

[

set heading 110

let car-right-lane other turtles in-cone (5 + ((speed \* 4) \* experience)) 180 with [ ycor = -1 ]

let cars-right-lane min-one-of car-right-lane [distance myself]

if cars-right-lane = nobody

[

set target-lane last lanes

set color green;

]

set heading 90

]

]

forward speed

end

--------------------------------------------------------------------------------------------------------------------------------

This method is previously discussed in the implementation section of the report. It uses cones to check for other vehicles. Firstly, it checks if there is a car in front. If there is, it checks in which lane the driver is currently. When the driver is in the overtake lane and there is a car ahead, it checks the speed difference to it and slows down if necessary or speeds up if possible. If the driver is in the cruising lane and there is a car ahead, it checks if there is car behind in the other lane. It does so by turning the driver backwards, using a cone to check and decide if overtake is possible or not and turns the driver back in the normal position. It is also important to note that each cone has a different distance, which is manipulated by the experience of the driver, so in turn – drivers with less experience will look closer and allow for less reaction time than more experienced drivers.

--------------------------------------------------------------------------------------------------------------------------------

to collision-check

; only check for collisions while in-lane driving since lane-change is a bit messy

; on improper lane-change, cars would create the problem once they have completely

; swapped lanes, so it won't alter results either way

if ycor = 1 or ycor = -1

[

let crash-cars other turtles in-cone 0.65 45 with [ ycor = -1 or ycor = 1 ] ; ycors are because only in-lane collisions should be counted

let crash-car min-one-of crash-cars [ distance myself ]

if crash-car != nobody

[

let me self

let spdd abs (([speed] of self) - ([speed] of crash-car))

ifelse spdd > 0.2

[

set heavy-crashes heavy-crashes + 1

]

[

set crashes crashes + 1

]

set color black

; slow down massively - debugging purposes to not count a crash many times

set speed ([speed] of crash-car) - 0.1

]

]

End

--------------------------------------------------------------------------------------------------------------------------------

The collision check firstly works only when cars are inside the lane. This is because sometimes the lane change can be messy, so this removes any false crash counts. It works by using a cone which is very short. If another vehicle is inside this cone, it would in turn mean that the two vehicles are too close to each other and have collided. If the cone detects someone, it checks the speed difference between the two cars to decide if it was a big crash or not. If the speed difference is big, it would count it as a big crash. It ends by setting the current speed of the car behind significantly lower than the one of the car ahead to prevent double collision counts.

--------------------------------------------------------------------------------------------------------------------------------

to slow-down-car ; turtle procedure

set speed (speed - (deceleration + speed-difference / 10))

if speed < 0 [ set speed deceleration ]

; every time you hit the brakes, you loose a little patience

;set patience patience - 1

End

--------------------------------------------------------------------------------------------------------------------------------

Simply decelerates the car by deceleration value plus the speed difference to the car ahead.

--------------------------------------------------------------------------------------------------------------------------------

to speed-up-car ; turtle procedure

if speed < desired-speed [ set speed (speed + acceleration) ]

if speed > desired-speed [ set speed (speed - deceleration) ] ; shouldn't matter, just debugging purposes

; set speed (speed + acceleration)

if speed > top-speed [ set speed top-speed ]

end

--------------------------------------------------------------------------------------------------------------------------------

If the speed is lower than the desired one, accelerates the vehicle. Also makes sure the top speed isn’t exceeded by returning back to it if acceleration above it is done.

--------------------------------------------------------------------------------------------------------------------------------

to choose-new-lane ; turtle procedure

; Choose a new lane among those with the minimum

; distance to your current lane (i.e., your ycor).

let other-lanes remove ycor lanes

if not empty? other-lanes [

let min-dist min map [ y -> abs (y - ycor) ] other-lanes

let closest-lanes filter [ y -> abs (y - ycor) = min-dist ] other-lanes

set target-lane one-of closest-lanes

;set patience max-patience

]

End

--------------------------------------------------------------------------------------------------------------------------------

This method chooses a new lane of the available ones. It was used back when “patience” was a thing, however currently the lane choosing is hard-coded as we need proper highway driving, not all cars driving in random positions. It works by checking which are the closest lanes and choosing a random one of them.

--------------------------------------------------------------------------------------------------------------------------------

to move-to-target-lane ; turtle procedure

set heading ifelse-value (target-lane < ycor) [ 180 ] [ 0 ]

let blocking-cars other turtles in-cone (1 + abs (ycor - target-lane)) 180 with [ x-distance <= 1 ]

let blocking-car min-one-of blocking-cars [ distance myself ]

ifelse blocking-car = nobody [

forward 0.2

set ycor precision ycor 1 ; to avoid floating point errors

] [

; slow down if the car blocking us is behind, otherwise speed up

ifelse towards blocking-car <= 180 [ slow-down-car ] [ speed-up-car ]

]

End

--------------------------------------------------------------------------------------------------------------------------------

This method handles the lane switching procedure. Firstly it checks which direction the lane change will be made, then turns the car that way. It then goes on to check if the car in the other lane (if any) is ahead or behind and slows down or accelerates accordingly to get inside the lane whilst moving sideways towards it.

--------------------------------------------------------------------------------------------------------------------------------

to-report x-distance

report distancexy [ xcor ] of myself ycor

end

to-report y-distance

report distancexy xcor [ ycor ] of myself

end

to-report car-speed

report speed

end

--------------------------------------------------------------------------------------------------------------------------------

These three methods simply report x, y and speed values.

--------------------------------------------------------------------------------------------------------------------------------

to select-car

; allow the user to select a different car by clicking on it with the mouse

if mouse-down? [

let mx mouse-xcor

let my mouse-ycor

if any? turtles-on patch mx my [

ask selected-car [ set color car-color ]

set selected-car one-of turtles-on patch mx my

ask selected-car [ set color red ]

display

]

]

End

--------------------------------------------------------------------------------------------------------------------------------

This allows for selecting particular cars to follow via the “watch selected car” function in the interface.

--------------------------------------------------------------------------------------------------------------------------------

to-report car-color

; give all cars a blueish color, but still make them distinguishable

report one-of [ blue cyan sky ] + 1.5 + random-float 1.0

end

--------------------------------------------------------------------------------------------------------------------------------

Paints all cars blue with a slight variance in the color for personal distinction. Cars get painted while doing certain tasks, for easier understanding of what each car is doing, so this won’t affect much.

--------------------------------------------------------------------------------------------------------------------------------

to-report number-of-lanes

; To make the number of lanes easily adjustable, remove this

; reporter and create a slider on the interface with the same

; name. 8 lanes is the maximum that currently fit in the view.

report 2

end

--------------------------------------------------------------------------------------------------------------------------------

Reports that there are 2 lanes on the road. Adjust this number for more lanes, however since lane switching is hard-coded, new lanes won’t be used by the AI.