Matteo Abrate

719738

Simulation Models for Economics

Torino, 18/02/2016

New Way of Insurance

Introduction

The NetLogo project is based on the traffic in an intersection in a city, where the cars run in both directions. In the intersection and all along the roads there are many cars that driving may have one or more incidents.
The main features of this simulation are the cars (or drivers) which are of two categories: city car and regular car and they drive this intersection for one year.

I want focus on the number of claim that every car can have running the ambient I create. To do this I recreate four sets of possible driver dividing them respect to the use of the car. Farther I add to the driver of the car the possibility of being drunk or distracted. Despite this world is no real, I try to make it more realistic by creating agents that are able to give precedence to the right and look ahead before moving.

Of course, if the driver of the car is drunk or distracted, the movements of the car will be different with respect to a sober and attentive. One of the strong hypothesis of the model is that carefully driver have no incidents between them, but they can have incidents with a drunk or distracted driver.

I suppose in this world exists two insurance companies that try to insure all the cars: one of them does not consider the using of the car made by the owner by charging the same premium for all drivers; the other company considers it and so has different premium for different using of car.

As city and regular car have different values, they pay a different premium. In the same way for the companies, the cost of claim, that is equal to a percentage of the value of the car, will be different.

To see the differences between the two ways of insurance, I suppose that the two companies has the same dataset and they work with all agents.

The second company, usually, will cash less than the first company will, but offering the same service at an inferior price, can beat the concurrence and insure the entire cars.

The aim is to create a model that can show how much a company can reduce the amount of premium paid by the insurer using in a better way the characteristics of drivers about the real use of the car.

How does it works?

Pressing setup button we create the world and the agents.

The agents are settled uniformly at the start of the four carriageways and then we check that there are not two car in the same patch. As input, we can decide the total number of agents and the number of the drivers, which are drunk and distracted.

Then pressing *go* button the simulation starts and the cars move along the streets. I suppose the agents not distracted and not drunk do not have incidents, so every *ticks* before moving the drivers check that there is no machine nor in the patch in front, nor in that of facing right.

Drunk drivers have a probability of going straight on without checking if there are cars, or move like normal drivers. At the same time, the distracted can move them self normally or with a probability low with respect to drunk drivers go straight on.

At every *tick,* the cars move and, due to the "bad" drivers, we have the possibility that two cars stay in the same patch. This means that we can have a claim. The presences of two car in one patch is a condition necessary but not sufficiently: one of two drivers can be able to stop before crashing or avoid the other car dodging last.

To put these ideas in the simulation, I introduce a Poisson probability distribution, which regulates incidents. The users of the model can choose the parameter of the distribution.

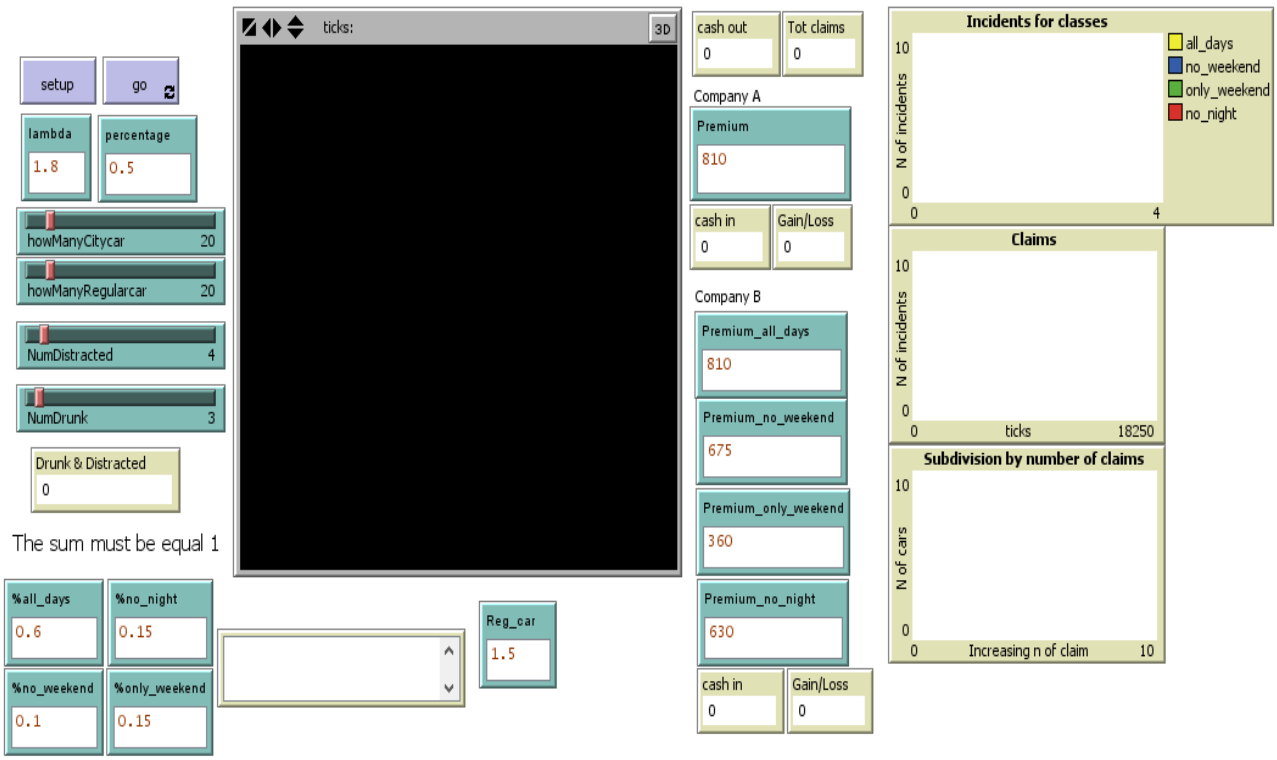
The four classes of drivers try to reflect the reality in which some driver use car only for a few time, so they like to pay insurance only for that time, because their risk exposure is lower than other drivers are.

The classes are:

- All days
- No night
- Only weekend
- No weekend

People who wants to drive and insure the car for all days in the year compose the first part (and usually the more numerous).
People who use car during the days, but during the night put it in the garage compose the second class.
People, who, during the week do not use car, but they want to use it in the weekend, compose the third class, for example: people that work in other cities, people using bike or bus or company car. They can be seen as a class of people considering car only for enjoy during free time. Alternatively, in the opposite, people who use car to go to work, working only during weekend.
The fourth class is the contrary of the third.

The Interface



In order to explain to the user how to manage the interface, I describe it starting from the top of the left part of it.

Two buttons:

- Setup: create the world.
- Go: start the simulation.

Two input monitors defined by the user:

- Lambda: the parameter of Poisson distribution that regulate incidents.
- Percentage: the percentage of the value of the car paid by the companies to the insurer.

Four sliders defined by the user:

- howManyCitycar: the number of agent with medium-low value of car.
- howManyRegularcar: the number of agent with medium-high value of car.
- NumDistracted: the number of agent with the characteristics of being distracted while driving.
- NumDrunk: the number of agent with the characteristic of being drunk while driving.

The sum of first two input is the total number of agent in the simulation.

There is the possibility that one or more than one agent can be distracted and drunk and the display down the sliders prints the number of this kind of agents. The rest of agents are heedful and they do not have car crash, unless they find on their street another driver, which is drunk or distracted.

Four input monitors defined by the user:

- %all_days: the upper limit of percentage of driver using car all day.
- %no_night: the upper limit of percentage of driver not using car during the night.
- %no_weekend: the upper limit of percentage of driver not using car in the weekend.
- %only_weekend: the upper limit of percentage of driver using car inly in the weekend.

To avoid errors, up the input there is written that the sum of the percentage must be equal 1, but the model before going on controls this sum: if the sum is different from one, the program prints an error message in the output screen near the percentage.

In the centre of the interface, there is the graphic representation of the world.

In the right part of the interface, starting from the top we can find two monitor:

- Cash out: the amount of money paid by the company to the insurer after incidents.
- Tot claims: the total number of claim.

The next three figures are about Company A:

- Premium: the premium paid by drivers to the company.
- Cash in: total amount received by the company.
- Gain/Loss: difference between *Cash in* and *Cash out*.

The next six figures are about Company B:

- Premium_all_days: premium paid by the drivers using car always to the company.
- Premium_no_weekend: premium paid by the drivers using car only during week to the company.
- Premium_only_weekend: premium paid by the drivers using car only during weekend to the company.
- Premium_no_night: premium paid by the drivers using car only during the day to the company.
- Cash in: total amount received by the company.
- Gain/Loss: difference between *Cash in* and *Cash out*.

All of the monitors about premiums are linked to the drivers that own a city car, while owner of a regular car paid the above premium multiplier for a factor:

- Reg_car: the factor to multiply premiums to find regular car premiums.

On the right, there are three graphs:

- Incident for classes;
- Claims;
- Subdivision by number of claims.

The code

The code developed in three part: declaration of all variables, *setup* procedures and *go* procedures.

1. Declaration

In this first part of the code, introduce the entire variables to run the simulation. The variables can be turtles-only or global variable, which are not related to any agent.

The variables owned by the agents are:

- *no_weekend?*
- *only_weekend?*
- *all_days?*
- *no_night?*

- *distracted?*
- *drunk?*

- *value*
- *n_of_accidents*
- *value_of_claim*

The first six are Boolean variables, they have only two values: true or false.

From the first to the fourth, we have the variables that characterized the simulation, because they represent for how many time the driver use the car and this hold the size of the premium due to the company.

Instead, the fifth and the sixth variables made division on a different level: no more about the use of the car, but about the characteristics of the driver. In this way, we have most of all driver that are "good" driver: in my model, good driver means not drunk and not distracted.
Then we will have an agent set of distracted driver, one of drunk driver and finally the "worst" composed by driver drunk and distracted.

The variable *value* represents the value of the car from which depends the premium paid by the insurer and the cost of the claim for the companies.

The variable *n_of_accidents* counts the number of accidents of every car, while *value_of_claim* is the sum of the reimbursement received from the company.

I create also a variable not owned by the agents that is a counter of the incidents happen during the simulation.

Afterwards, I generate two breeds, to characterise the type of car that can be "citycar" or "regularcar": the difference is the value of the car, hence, the value of the claim and premium paid.

2. Setup

This procedure invokes other eight procedures:

- *ca*
- *if (%all_days + %no_night + %no_weekend + %only_weekend) != 1 [output-write "Check the percentage" stop]*
- *create_cars*
- *setup_road*
- *set_cars_position*
- *check_cars*
- *assign_characteristics*
- *reset-ticks*

Some of them are built-in commands, while other procedures creating the world and give to the agents the characteristics.

The first procedure is *clear-all (ca)* and it cancels the effects of all procedures called before: so every time we press setup, NetLogo removes all the results about the previous simulation and it is ready to start another one.

The next line checks if the percentages asked to the user are corrected (sum of them equal 1) or not: if it is not, the command *output-write* prints on an output monitor in the interface "Check the percentage" and *stop* the simulation.

Afterwards *create_cars* creates the regular and city car for a number given by the user, it gives them a shape similar to car and declare all of the turtle-own variables, setting *n_of_accidents* and *value_of_claim* equal zero and the Boolean variable false.
In addition, the global counter of incidents *tot_claim* is set to zero.

The procedure *setup_road* is a patches procedure, because it colours some of them in order to have a graphic distinction between the street and the other patches.
The next two procedures are:

```
to set_cars_position

 ask turtles

  [ let choice random-float 1

    if choice < 0.25
    [ set xcor -1 set ycor random-pycor set heading 180 ]
    if choice >= 0.25 and choice  < 0.5
    [ set xcor random-pxcor set ycor -1 set heading 90]
    if choice >= 0.5 and choice  < 0.75
    [ set xcor random-pxcor set ycor 1 set heading 270]
    if choice >= 0.75
    [ set xcor 1 set ycor random-pycor set heading 0]]
  ] end
```

```
to check_cars

  ask turtles

  [     if any? other turtles-here [ fd 1 ]   ]

end
```

Set_cars_position uses the help of *random-float 1*, which generates a random number between 0 and 1. Depending on the number, the car is settled all along the four-direction travel, setting also the heading in order to make them go in the right direction.

The procedure *check_cars* controls if there are no patches with more than one car and, if there are another car or cars in the patches, makes the car one-step forward until it finds a free patch.

This is the code of the procedure *assign_characteristics.*

```
to assign_characteristics

  ask n-of NumDistracted turtles [set distracted? true ]
  ask n-of NumDrunk turtles [set drunk? true ]

  ask turtles
    [
      let t random-float 1
      ifelse t < 0.5
      [ ifelse breed = citycars [ set value 10000 - random 1000 ] [ set value 15000 - random 2000 ]]
      [ ifelse breed = citycars [ set value 10000 + random 1000 ] [ set value 15000 + random 2000 ]]
    ]
     let ad int(%all_days * (howManyCitycar + howManyRegularcar))
     let nn int(%no_night * (howManyCitycar + howManyRegularcar))
     let ow int(%only_weekend * (howManyCitycar + howManyRegularcar))
     let nw int(%no_weekend * (howManyCitycar + howManyRegularcar))
     if (ad + nn + ow + nw) != (howManyCitycar + howManyRegularcar)
     [ set ad ad + (howManyCitycar + howManyRegularcar - (ad + nn + ow + nw))]

     ask n-of ad turtles [set all_days? true]
     ask n-of nn turtles with [ all_days? = false ][ set no_night? true ]
     ask n-of ow turtles with [ all_days? = false and no_night? = false ][ set only_weekend? true ]
     ask n-of nw turtles with [ all_days? = false and no_night? = false and only_weekend? = false ][ set
no_weekend? true ]
end
```

Line 1-2 estabilish to an agentset of the dimension chosen by the user the characteristic distracted and to another the characteristic of being dunk. It is possible that some agents belong to each group, making them the "worst" possible driver.

Then, I give to the city car a value equal to 10000 and to the regular car 15000, but to differentiate the value between car with of the same type, I use *random* 1000 (or 2000) to increase or decrease the value of the car.

In the interface, the user declares the percentage of every class of driver. This can led the model to have a floating number of car that is impossible; hence, I apply *int* to the multiplication of the percentage and total number of agent.
If the sum of the integer part of every class is different to total (due to the cut of decimal part of number), the difference of them is set to driver using car every day.

The last four lines of the procedure correspond to an extraction without replacement: first takes a number of car and set them all days' drivers; second takes a number of car (excluding the previous cars) and set them no night's drivers and so on.

The last command is *reset-ticks,* which resets the tick counter to zero.

3. Go

The button *go* is composed by four procedures, which comprehend all the other procedures that permit cars to move and have incidents.
The third is for change the characteristics like drunk and distracted every week.
The fourth is for decrease the value of the car (-5%) every two months.
The final two lines stop the simulation after one year and increase the value of *ticks* after all procedures.

I suppose that one day is equal to 50 ticks, so 1 week is 350 ticks and 1 year is 18250 ticks, hence, when ticks equals to 18250, the simulation stop.

The first two procedures are *check_day* and *check_crashing*.

Before there is another one that is not invoked in *go*, but in *check_day,* and its name is *re_check_cars.* It attends to put on the street the cars after they are parking out because their driver does not use the car.

The code of procedure *Check_day*.

```
to check_day
  ask turtles
 [
   if no_weekend?
   [ let time remainder ticks 350
     if time = 0 [ re_check_cars ]
     ifelse time <= 249 [ moving ][ setxy 9 9]]

  if only_weekend?
   [  let time remainder ticks 350]
     if time = 250 [ re_check_cars ]
     ifelse time <= 249  [ setxy 12 12]  [ moving ]]

  if no_night?
   [ let time remainder ticks 50
     if time = 0 [ re_check_cars ]
     ifelse time <= 29   [moving] [ setxy 15 15]]

   if all_days? [moving]
 ]
end
```

The procedure makes car drive or not depending on the day or the hour in which we are. To divide the year by interval I use the built-in function *remainder* that give us the rest of a division. For every class of driver we have two intervals in the time: in one, they drive, in the other they stay outside the road.
When ticks is equal to the value that divide the two interval, the car comes back to the road by the procedure *re_check_cars.*

Moving regulates the car's movement: it checks the type of driver and later, for every of them, there is another procedure to make move:

- *move_drunk_distracted;*
- *move_drunk;*
- *move_distracted:*
- *move.*

To explain all the procedures, start from *move,* which regulates the "good" cars: before moving himself by one patch, the driver checks if there is nobody in the subsequent patch and in the patch at right of the subsequent.
As we can see, they never have an incident because they always control if the street is free before moving.
The central point is that not all driver move always in this way because drunks and distracted drive in this way only sometimes.

To make them drive well only sometimes, use again *random-float 1* to regulate the probability of driving normally o driving badly: under a determined number, the driver does not check street before going and can go faster; otherwise, the driver checks it.

Hence, distracted driver moves by 1 patch, drunk driver moves by 2 patches while drunk and distracted by 3.

After that all car the cars move, the procedure *check_crashing* regulate the incidents. In this simulation, the presence of two car in one patch is not a sufficient condition to have an incident, but it is a necessary condition.

```
to check_crashing

 ask turtles
 [
  if any? other turtles-here and pcolor = grey
 [
  if drunk? and distracted?[ if random-poisson lambda > lambda + 1 [ claim ] ]
  if drunk? [if random-poisson lambda > lambda +  2 [ claim ] ]
  if distracted? [if random-poisson lambda > lambda + 3 [ claim ]]
  if not drunk? and not distracted?[if random-poisson lambda > lambda + 4 [ claim ]]
 ]]

end
```

This idea try to reflect all the case in which two car crush without have damage, or one of the two driver can change direction at last second avoiding crash.

Hence, the procedure checks if there is another car in the patch and if yes, depending on the "psychological" status of the driver we have an increase probability (highest for drunk and distracted, low for conscious driver) to having an incident.
The probability distribution is a Poisson distribution with parameter *lambda* given by the user.

Now proceed with the procedure *claim,* which describes all the events related to the probability of doing or not an incident.

```
to claim

 let mcars one-of other turtles-here
 let perc random-float percentage

 ask mcars

 [
  if drunk? and distracted? [ if random-poisson lambda > lambda  + 1
   [ set tot_claim tot_claim + 1 set n_of_accidents n_of_accidents + 1 ifelse value_of_claim = 0
    [set value_of_claim  value * perc][set value_of_claim value_of_claim + value * perc]  ] ]
```

```
  if drunk? [ if random-poisson lambda > lambda + 2
    [ set tot_claim tot_claim + 1 set n_of_accidents n_of_accidents + 1 ifelse value_of_claim = 0
      [set value_of_claim  value * perc][set value_of_claim value_of_claim + value * perc] ] ]

  if distracted? [ if random-poisson lambda > lambda + 3
    [ set tot_claim tot_claim + 1 set n_of_accidents n_of_accidents + 1 ifelse value_of_claim = 0
      [set value_of_claim  value * perc][set value_of_claim value_of_claim + value * perc]  ] ]

  if not drunk? and not distracted? [ if random-poisson lambda > lambda + 4
    [ set tot_claim tot_claim + 1 set n_of_accidents n_of_accidents + 1 ifelse value_of_claim = 0
      [set value_of_claim  value * perc][set value_of_claim value_of_claim + value * perc] ] ]
 ]

end
```

The procedure controls which kind of driver is in the same patch and then here interfere the Poisson probability distribution with a different parameter for every kind of driver. At the same time, the counter of crash, *tot_claim*, increases by one and the turtle-own variable, *n_of_accidents,* increases by one.

In *value_of_claim*, we use an *ifelse* structure: if it is the first incidents, I set *value_of_claim* as a percentage of the value of the car; if it is not the first incidents, I add the new value of the claim that is always a percentage of the value of the car, to the old one. On the interface the user must set the percentage of reimburse, but to make more realistic, the company paid a random percentage between zero and the one choose by user.

The last two procedures are executed only for a certain value of *ticks*: the first one when *ticks* equal 350 (hence one week), the second one when *ticks* equal 3041 (hence about two months).

Variate_characteristics changes the parameters drunk and distracted. Hence, first reset the entire agent putting *false* both the characteristics, and then always following user's indication, set to some of them *true* to be drunk or distracted.

Update_value decreases the value of the cars by 5% every two months.

Simulations

The model is useful to analyse, from the point of view of the companies, the difference between premiums earned and claims paid. Both companies use the same dataset modifiable from the sliders on the interface, as number of regular or city cars, or the number of people drunk, or the size of the premium and many more parameters.

The ruin or the win of the companies depends, most of all, on three parameters: the percentage of reimburse of the claim, the drivers' composition respect attention and temporal characteristics and the parameter of the Poisson's distribution. In the interface, the user have to declare the upper limit of the percentage of reimburse.

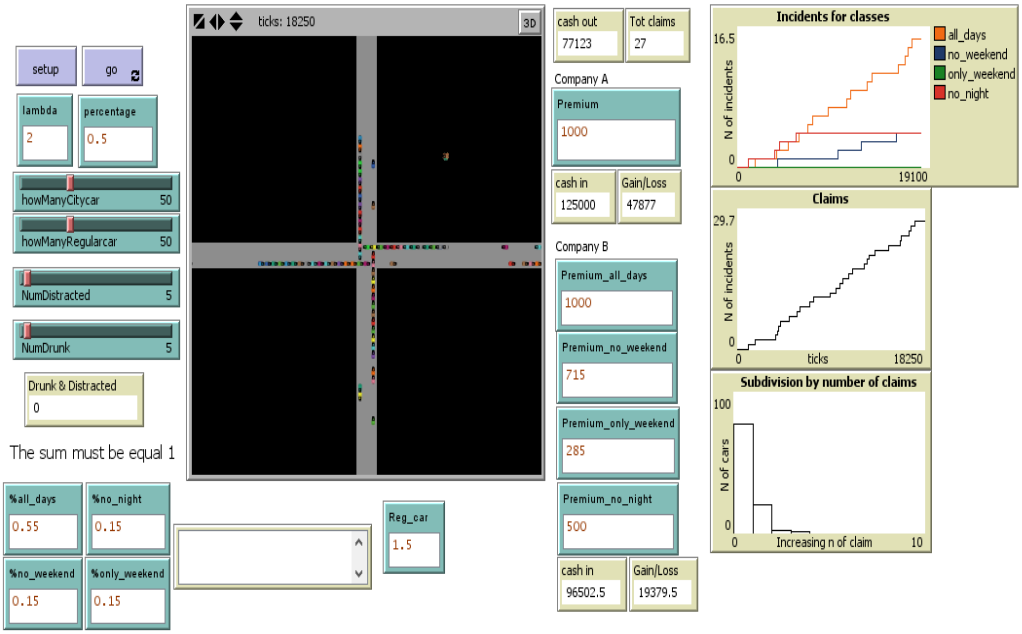
Company A applies the same premium to every driver: if it is equal to all day's premium of *Company B*, it means that *Company A* will always cash more than *Company B*. In this situation, the totality of drivers with temporal characteristics will choose *Company B*, while all day's driver will choose indifferently.

Company A, to increase the number of insurers, can decrease the size of the premium, setting it equal to the weighted average of premiums offered by *Company B*: in this case, the all day's driver will choose this company while drivers with temporal characteristics will choose the other company.

As I said explaining the interface, the companies spend the same amount of money to reimburse the driver, so it is very important for them calibrates the premium in order to avoid the ruin.

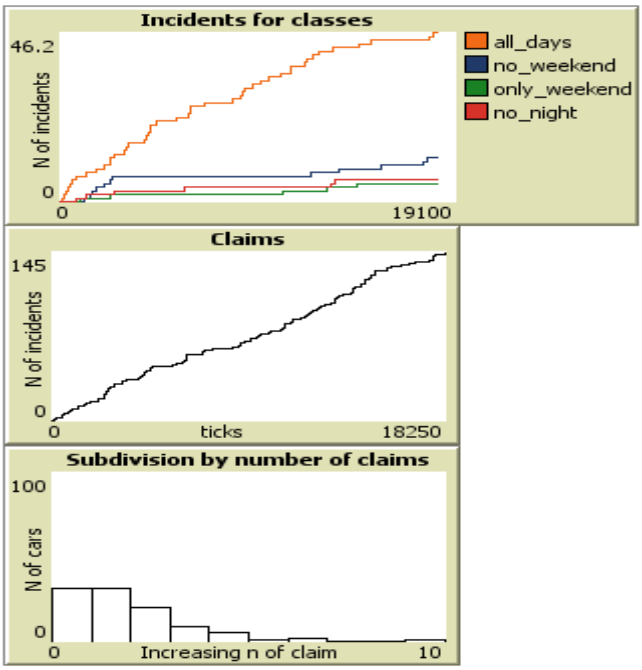
In the first simulation, *Company B* sets its premium proportionally: all day is equal to 1000, no weekend equal to 715, only weekend equal to 285 and no night equal to 500. On the other hand, *Company A* sets its premium equal to 1000. The number of car is 100; drunk drivers are 5% and distracted people too.

In this case, *Company A* always cash more than *Company B* but in most of all simulations with these parameters, both companies end the year with positive results, that means for *Company B* the possibility to beat the competition and takes all the market.



The high number of cars in the world causes the presence of tail during the simulation; nevertheless, the number of incidents is not so high. The graph of incidents increase vertically during tail, while without tail the increment is irregular and slow.

To see the graph change is sufficient increase the parameter of Poisson distribution (example from 2 to 3): we can have simulation with more than 140 incidents and both companies fall into ruin.



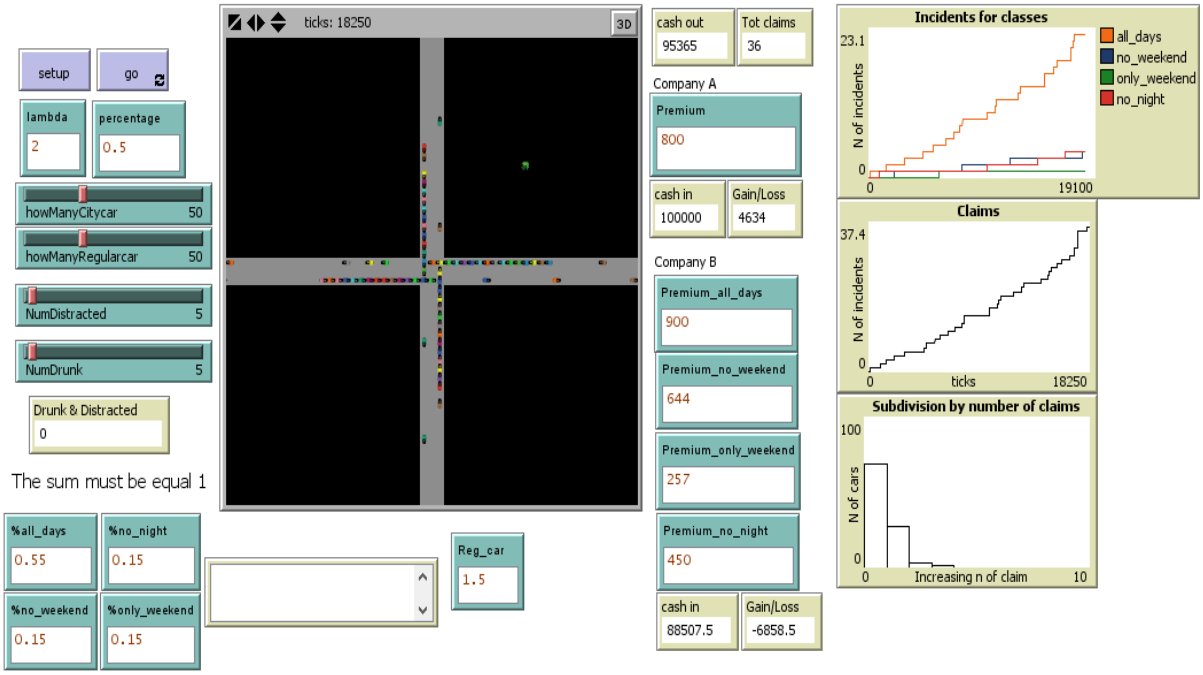
The tail is one of the principal causes of the incidents in the model: the car driven by “good” driver occupies all the patches around the corner and so, “bad driver”, going straight on, induces incidents.

Both companies can decrease the size of the premium, *Company A* (-20%) while *Company B* (-10%). The difference between premiums earned by the companies is smaller than before and the results now are more close to the zero, but *Company A* is offering a very interesting price to all day drivers.

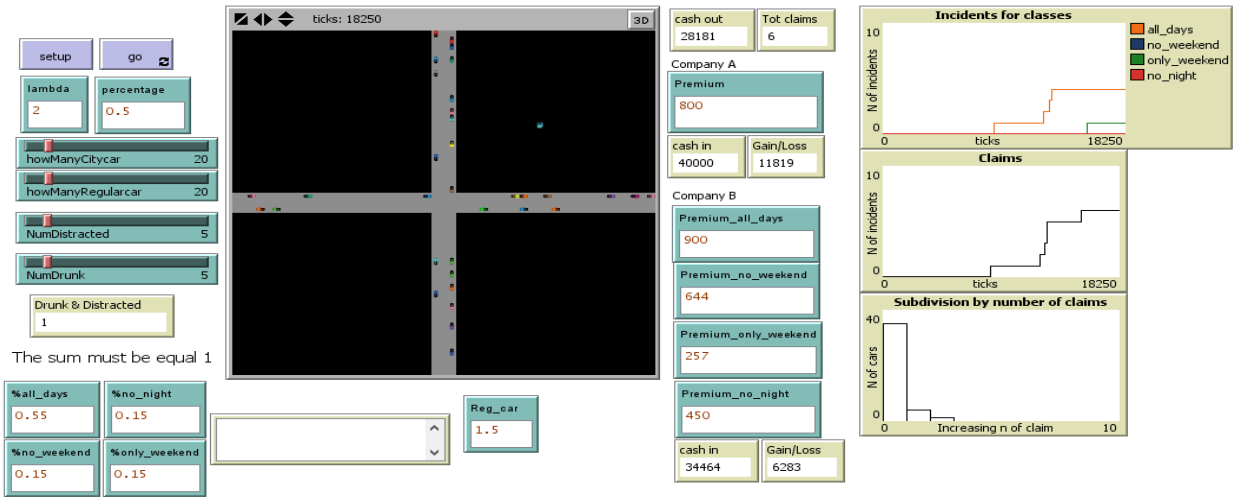
In the model, I keep constant for both companies the upper level of percentage of reimburse because the drivers must obtain the same services from the companies to make a comparative analysis.

Hence, decreasing size of premium earned the percentage of negative scenarios increases respect before.

The following interface is the result of a simulation after premium reduction: *Company A*’s balance is positive while *Company B* not. This result is an average of the simulations obtained in which the gain decreases to zero and sometimes becomes loss for both companies, especially for *Company B*.



Decreasing the number of agents (from 100 to 40) but keeping constant the other variables, the result is few incidents and not in proportionally way. The principal cause is the sample size, which does not create tail in the simulation except for two periods: during the first one only one incident happens, but during the second, we can see in the graphs an “exponential” growth of the number of incidents.



Both companies can decrease their premium (until -50% respect the second simulation) before going bankruptcy.

Changing the database’s composition respect temporal characteristics, let us see the difference with a sample composed only by all day drivers, only by no weekend drivers and so on. The other variables are fixed for all the simulations (N=100, drunks=10%, distracted=10%) and the premiums paid by the insurer are:

	All_days	No_night	Only_weekend	No_weekend
Company A	800	800	800	800
Company B	900	450	257	644

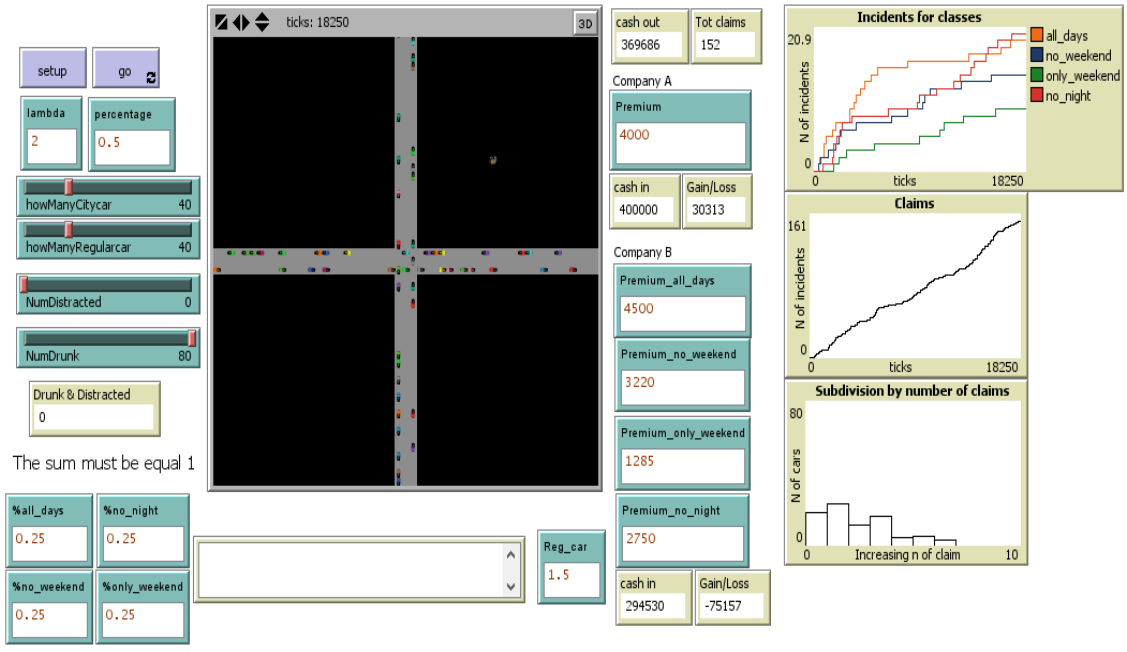
The number of incidents in the simulations decreases respect the time spent by drivers in the cars, starting from the top (all days) to the bottom (only weekend): the difference is due to the different risk's exposure.

The simulations with only one typology of drivers lead the companies to solve the same problem because having only one temporal characteristic is equal to do not have it. Hence, to avoid bankruptcy in simulations with *no_night, only_weekend* and *no_weekend*, *Company B* must increase drastically the premiums received, while Company A could reduce them to make the offer more desirable. The premiums find to cover the expenses in the simulation are the same for both the companies.

Interesting instance is when drivers are drunk, no matter their typology (the database is equally distributed between driver's typologies).

First simulation	10 cars, 10 drunks	0- 5 incidents ca.
Second simulation	20 cars, 20 drunks	5-10 incidents ca.
Third simulation	30 cars, 30 drunks	10-20 incidents ca.
Fourth simulation	40 cars, 40 drunks	20-30 incidents ca.
Fifth simulation	60 cars, 60 drunks	70-80 incidents ca.
Sixth simulation	80 cars, 80 drunks	140-160 incidents ca.
Seventh simulation	100 cars, 100 drunks	260-300 incidents ca.

Sixth simulation:

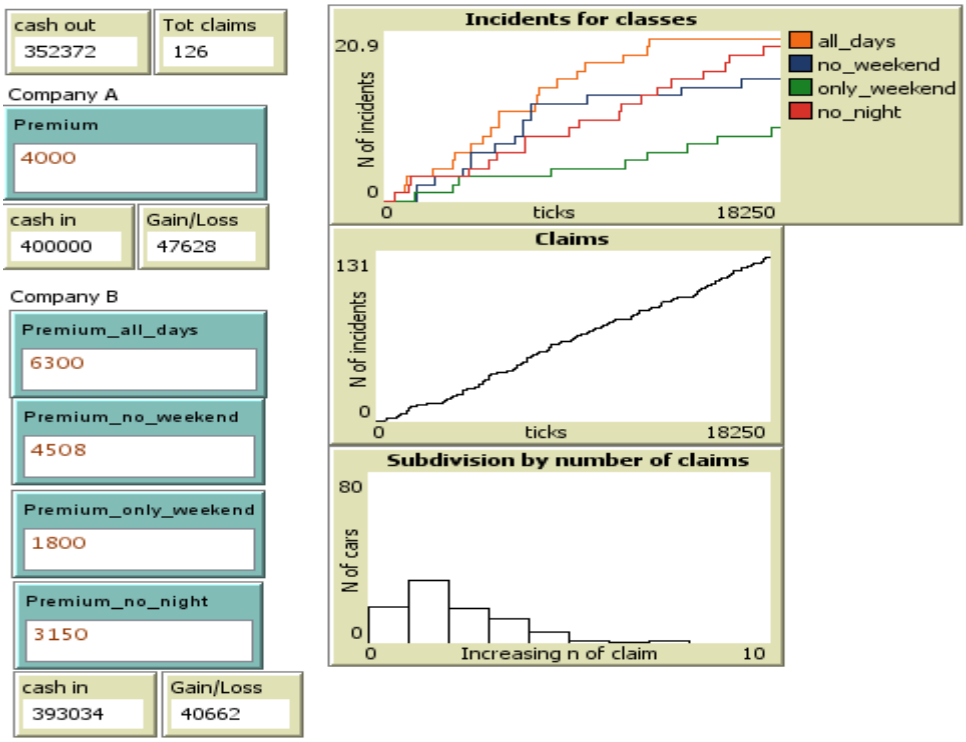


The growth of number of incidents is not proportionally to the increasing number of agents.

To cover the expenses and to get some positive scenarios, the companies have to increase the size of the premium:

- +500% for *Company A*;
- +700% for *Company B.*

Analysing the premium, we get under Company B drivers' *all_days* and *no_weekend* pay a larger premium respect *Company A* and also driver's *no_night* can decide to pay a larger premium to get the use of the car all days.



In case of all drunks and distracted, the situation for the companies is catastrophic.

Drunks and distracted	N = 100	Number of clams = 8700 ca

The case with all drivers distracted, expected to be very similar to the one with all drunk, just a bit better for the companies, but the simulations show an "equilibrium" in the model. Indeed, on one hand we do not have tail blocking traffic and creating incidents and on other hand, the number of claims (always with 100 cars) is about 20.

Hence, considering a dataset with only distracted driver seems to be a "realistic" representation of the world respect the model created.

Try to implement new strategies about pricing in no-life insurance is one of the topic more interesting due to the power of technological progress, which permits to know, analyse and use (see Black Box Car Insurance) all the characteristic about the insurer. This model reflects the attempt of pricing using the subjective features of the insurers.