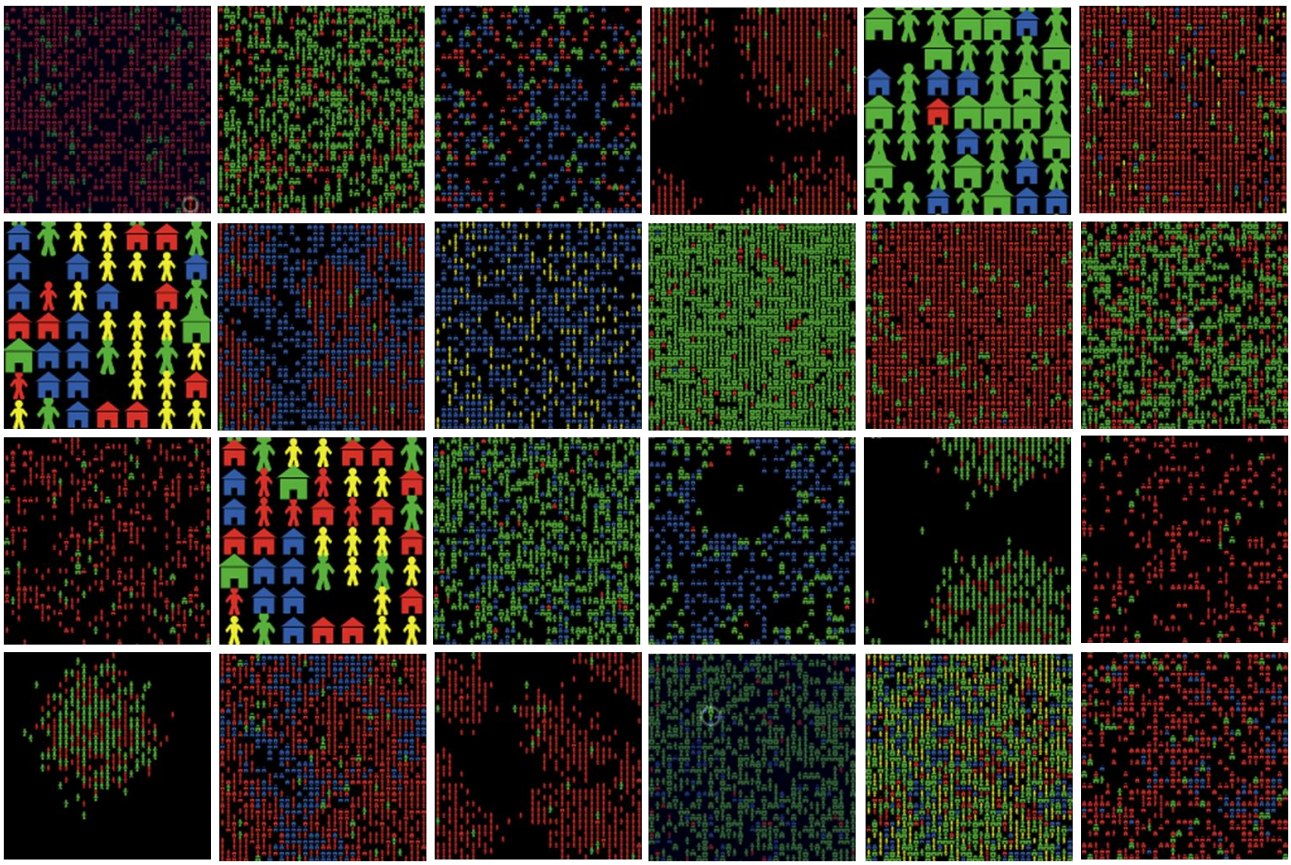
UNIVERSITÀ DEGLI STUDI DI TORINO

Facoltà di Economia

SIMULATION MODELS FOR ECONOMICS

Final report

"RUNNING CORRUPTION"



Professor Pietro Terna

Edited by Silvia Casale, Matteo Kersbamer, Luca Piero Aldo Rosazza Gat

Introduction

According to the paper *"Conditional corruption",* written by B.Dong - U.Dullek - B.Torgler, corruption in a social economic environment could be explained as a combined effect of two different forces:

(i) a personal perception, which allows us to consider agents' behavior partially determined by the behavior of agents surrounding them;

(ii) a more general one, given by past experiences, like the past level of corruption.

Under the methodological approach given by the authors, several variables will determine the individual behavior and its propensity to be corrupted.

We start with these assumptions, adapted in our case with some simplifications, in order to build our simulation in an easier way to understand. For example, one of these simplifications, is in the use of different names for the most important variables, since in the paper are presented in a misunderstanding way with a scale where the maximum level of corruption is represented by zero. We call "honesty level" what in the paper is "level of corruption".

We created the project ("Running corruption"), which models the economic interactions between two types of agents: buyers and sellers (e.g. they can be suppliers and purchasing office managers of firms). Our aim is to simulate transactions taking part among buyers and seller in a world in which corruption, at social and individual level, exists and affects the economic results.

To this purpose, our model generate the propensity to be corrupt, named *Propensity* in the code, for every agent of both categories. This individual characteristic, determines the inclination towards a bribery behavior of the agents in the interactions, higher value means low bias in favor of acting corrupt, low value means high degree of corruption in the behavior.

Propensity comes from the combination of different macro and micro variables, in order to diversify whether they are referred to the single agent or to the whole society.

Such macro variables we considered are the world's general level of honesty, *HonestyLevel*, i.e. the honesty level at t-1, *ruleOfLaw*, i.e. the effectiveness of legislation, *#Agents*, i.e. the population size and *democraticAccountability*, i.e. how responsive government and institutions are towards their people.

We have taken the results of a linear regression developed by the authors and applied it in our simulation. The regressors are *HonestyLevel, ruleOfLaw, #Agents, democraticAccountability* and in our case the dependent variable is GHL (general honesty level).

This value is obviously equal for all the agents, however, combined with another random value, *microVariable* will give us our main variable: the specific *Propensity* of every agent. This random value summarize differences in individual characteristics such as education, religion, age, etc. set randomly in the simulation.

In every cycle, each seller looks for a buyer to trade with. The nature of the transaction, corrupt or not, is defined comparing the *Propensity* of the two agents with a given threshold worked out earlier. At this point the ratio between the total number of corrupt transactions with the total number of transactions in general will modify the honesty level getting ready for the next cycle. *Propensity* can be modified moving the slider *howDistant?*. This can be intended as the quantity of available information. Increasing it allows agent to "see" gradually the nature of agents' surrounding him last transactions and consequently modify his behavior.

We worked in this way trying to simulate what written in the paper, that corruption today can be affected both by the environment surrounding agents and by the past level of corruption.

The most interesting parts of the code:

```
to findNewSpot

 while [any? other turtles-here]
  [lt random 360 repeat 5 [fd 0.5] ]

 move-to patch-here

end
```

The *findNewSpot* procedure is needed to get a clearer view of what's happening in the world at each loop. After being created in the center of the world, buyers one at time move on a randomly chosen patch. If no other buyers already stand on this patch, the *findNewSpot* procedure is completed by moving the agent to the center of the patch, on the contrary the buyer will turn left and move forward. Then he will run again the procedure. If this time no one stands on the chosen patch, the program will go on, asking to the next agent, otherwise the buyer will move and repeat again. Once each buyer has completed this commands, the same steps will be executed by the sellers. The result of this procedure will be to provide each agent his own patch, by this way the observer will be able to clearly see each agent in the windows.

```
to setVariable

 set #Agent howManyBuyers + howManySellers
 set radius howDistant?

 set GHL honestyLevel * 0.461 + ruleOfLaw * 0.204 + democraticAccountability * 0.180 +((ln #Agents ) * -0.2)

 if GHL < 0 [set GHL 0]

 ask turtles
  [
   setPropensity
  ]

end
```

setVariable is a procedure called both in the *setup* and in the *go* procedure, which assigns a certain number to the *macrovariable* variables of the model and calls the subsequent procedure *setPropensity*.

The sum of the numbers set in the two sliders *howmanybuyers* and *howmanysellers* gives the number of agents acting in the world: *#Agent*.

The value defined through the slider *howDistant?* assigns a range of vision to the agents: *radius*.

The variable *GHL* is the general level of honesty characterising our world. This variable takes a number that is the result of a formula that combines the values set in the sliders *ruleOfLaw*, *democraticAccountability* and the one of the variable *#Agents*. *GHL* increases with the values of the first two variables and it is negatively correlated to size of the population. In order to exclude negative numbers for GHL (since it does not make any sense) if the value happens to be lower than zero, the program sets it automatically equal to zero.

After this, we ask to the agents to execute the procedure *setPropensity*.

SetVariable is called in the setup to create the initial state of the world and then it is also called in the *go* procedure after the execution of the procedure fixNewLevelHonesty in order to update the variables.

```
to go

 set-current-plot "General Honesty Level"
 plot GHL

 bornDie
 getThreshold
 checkAround
 act
 fixNewLevelHonesty
```

```
 setVariable

 tick

end
```

This procedure is executed pressing the GO button in the interface and it runs the simulation. The button has the option "forever" ticked for default, therefore once you press the button it is executed continuously. The simulation stops when you press it again. The first lines are just to plot a graph in the interface tab.

To go is composed by six procedures:

- *to getThreshold;*
- *to fixNewLeveHonesty;*
- *to setVariable*, which are all executed by the observer;
- *to bornDie* that is executed by a random turtle;
- *to checkAround*, which is carried out by all agents. This procedure contains also *to lookAround;*
- *to act* executed only by sellers;

```
to getThreshold

 set numberCorrupted 0
 set numberNon 0
 set T 6  + (0.4 * standard-deviation [Propensity] of turtles)

end
```

Sets the variables *numberCorrupted* and *numberNon* equal to 0, so that at the beginning of each loop they will restart from 0. This makes possible to get at each cycle the exact number of transactions that have had a corrupt or honest outcome.

What it is important in this function is the *getTreshold* assigns the variable *T* a value, based also on the agent's propensity' standard deviation. Given that the propensity changes at each cycle, *T* will vary too.

```
to fixNewLevelHonesty

  set-current-plot "% transactions"
  set-current-plot-pen "Total transactions
  plot 100

  set-current-plot Total /2"
  plot 50
  set-current-plot-pen "Non corrupt"
  plot (numberNon / (numberCorrupt + numberNon)) * 100
  set-current-plot "# Agents"
  set-current-plot-pen "# Sellers             "
  plot count sellers
  set-current-plot-pen "# Buyers"
  plot count buyers
  set-current-plot  “Interaction"
  set-current-plot-pen “How distant?"
  plot howDistant?

  let r numberCorrupt / (numberCorrupt + numberNon)
  ifelse r > 0.5 [set honestyLevel GHL + (10 * r *(1.66 - exp r) - random-float (r * 3))] [set honestyLevel GHL - (5 *
(0.65 +  ln (r + 0.01)) + random-float 2)]

  if honestyLevel < 0 [set honestyLevel 0]

end
```

It is called at each *go*, after the negotiation has been done. It calculates the ratio of corrupt transactions over the number of total ones, to verify whether it exceeds the number of non corrupt agents. If it does, the global variable *honestyLevel* will be decreased by setting it equal to *GHL* minus a certain randomly defined quantity to make it available for the next cycle. On the contrary, it will be increased by setting it equal to *GHL* plus that random quantity. In the case the resulting *honestyLevel* is negative, the program will set it equal to 0.

```
to bornDie

  ask one-of turtles
  [
   set b howManyBuyers - count buyers
   while [b != 0]
     [ifelse b < 0 [kill][generate]]

   set s howManySellers - count sellers
   while [s != 0]
     [ifelse s < 0 [kill][generate]]
  ]

end
```

This procedure was built to solve the problem coming from the fact that moving the two sliders *howManyBuyers* and *howManySellers* the number of agents effectively shown in our world was not changing. We could not recall again *to create*, because that would have affect all the previuos setting, like colors, dimensions and variables, like *microVariable* etc. In this way, at the beginning of every cycle, we control if the number of agents has been changed or not. If so we "generate" or "kill" a number of agents equal to the difference we have found through the one of the two procedures *generate* and *kill*, if not the simulation will go on with the next procedure *getThreshold*.

```
to checkAround

 ask turtles
 [
  while [any? other turtles with [breed = [breed] of myself] in-radius radius with [color = red or color = green ]]
    [lookAround stop]
 ]

End


to lookAround

 let r count other turtles in-radius radius with [breed = [breed] of myself and color = red ] / (count other turtles
in-radius radius with [breed = [breed] of myself and color = green ] + count other turtles in-radius radius with
[breed = [breed] of myself and color = red ] )
 ifelse r <= 0.5 [set Propensity Propensity - (1 + random-float 0.5 + r)] [set Propensity Propensity + (2 + random-
float 0.5 - r)]

 if Propensity < 0 [set Propensity 0]

end
```

At this point we had to create something to make all the turtles to interact in some way. We thought to introduce the variable *radius* to make turtles able to "see" around them how other agents have behaved previously. With *checkAround* we first check if, in the radius we set, stood any agent of the same breed with some specific characteristic, it has to be red or green. This to avoid that agents would change their *Propensity* without any reason. This would have happened in the first cycle of the simulation or in the case our agent would have not been surrounded by any other agent. If it happens to be true, *to lookAround* will be called. In the first step every agent will calculate the ratio between agents, with the characteristics explained earlier, colored in red and the total number of agents surrounding it. This ratio will obviously be included between 0 and 1, if it turns out to be included between 0 and 0.5, a number, equal to the sum in the round brackets, will be subtracted to

the *Propensity* of the given agent, in the opposite case a similar number will be add. This gives the possibility that agents can modify their behavior as a consequence of what happened in the past in the patches around them.

```
to act

 ask sellers
 [
  move
  negotiate
  countTransaction
  getColor
  findNewSpot
 ]

end
```

Since sellers look for buyers to make transaction, they are asked to execute the procedure *act* that includes other five procedures:

- *to move*
- *to negotiate*
- *to countTransaction*
- *to getColour*
- *to findNewSpot*.

```
to move

   while [not any? buyers-here]
      [lt random 360 fd 1 move]

end
```

```
to negotiate

  let jb item 0 [Propensity] of buyers-here
  let js item 0 [Propensity] of sellers-here

  if T < jb and T < js [ifelse random-float 1 < 0.05 [set NT 1] [set NT 0]]          (i)
  if T > jb and T > js [ifelse random-float 1 < 0.95 [set NT 1] [set NT 0]]
  if T < jb and T > js [ifelse random-float 1 < 0.6 [set NT 1] [set NT 0]]
  if T > jb and T < js [ifelse random-float 1 < 0.4 [set NT 1] [set NT 0]]

end
```

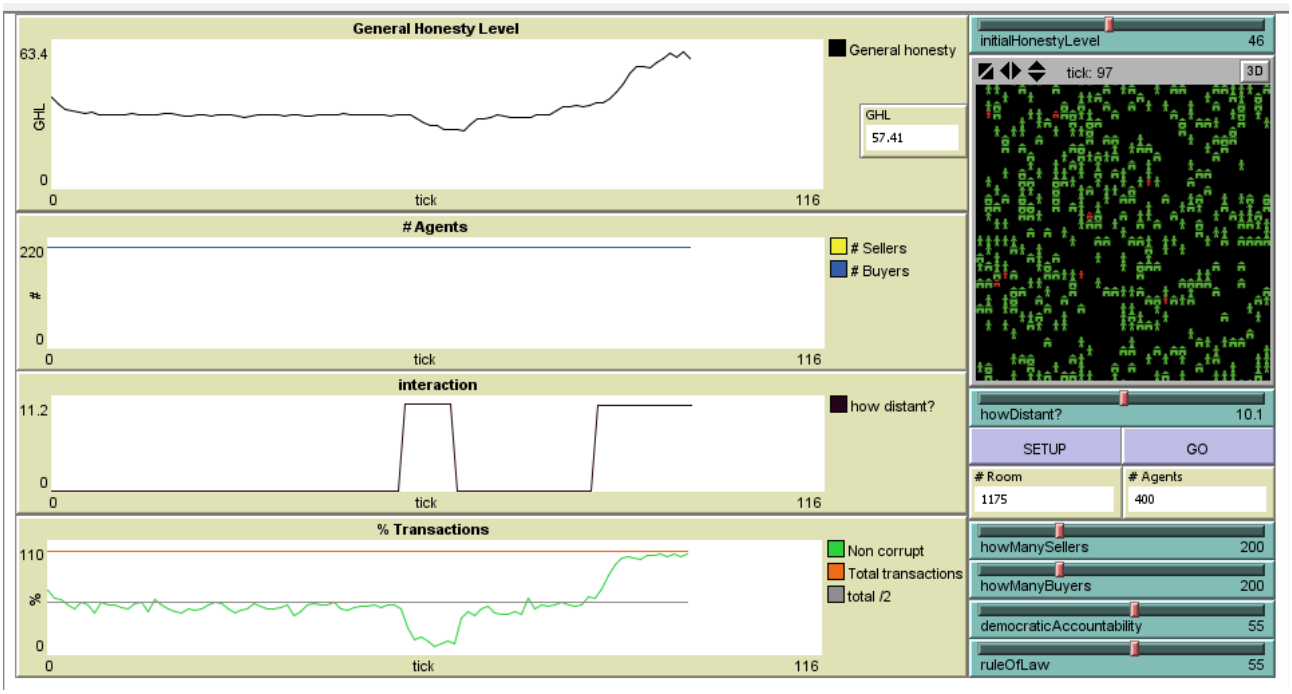
This procedure is the “core” of the simulation. Every seller, moving around, finds a random buyer with whom starts a “negotiation”. This trade will be labeled as corrupt or non corrupt comparing the two propensity values with the value of the threshold fixed earlier. These two values can be below or above the line and any of the combinations that comes out will be treated in a particular way with a different probability that the exchange will be affected by bribery. If, for example, both of the values will be below the threshold as in the in line (i), the probability that the exchange is corrupt is equal to the 95%. Jb and Js are two variables used to extract the propensity of the agent from the list buyers\sellers-here and allow NetLogo to recognize them as numbers.

```
to getColor

 ask turtles-here
 [
  ifelse NT = 0 [set color green set size 1.3] [set color red set size 1]
 ]

end
```

Once the negotiation has taken part, the two agents involved will change their aspect, they will change color and modify their size as a consequence of the particular nature of the trade. If it happened to be honest, they will become bigger and green, on the contrary they will turn red and decrease their size.

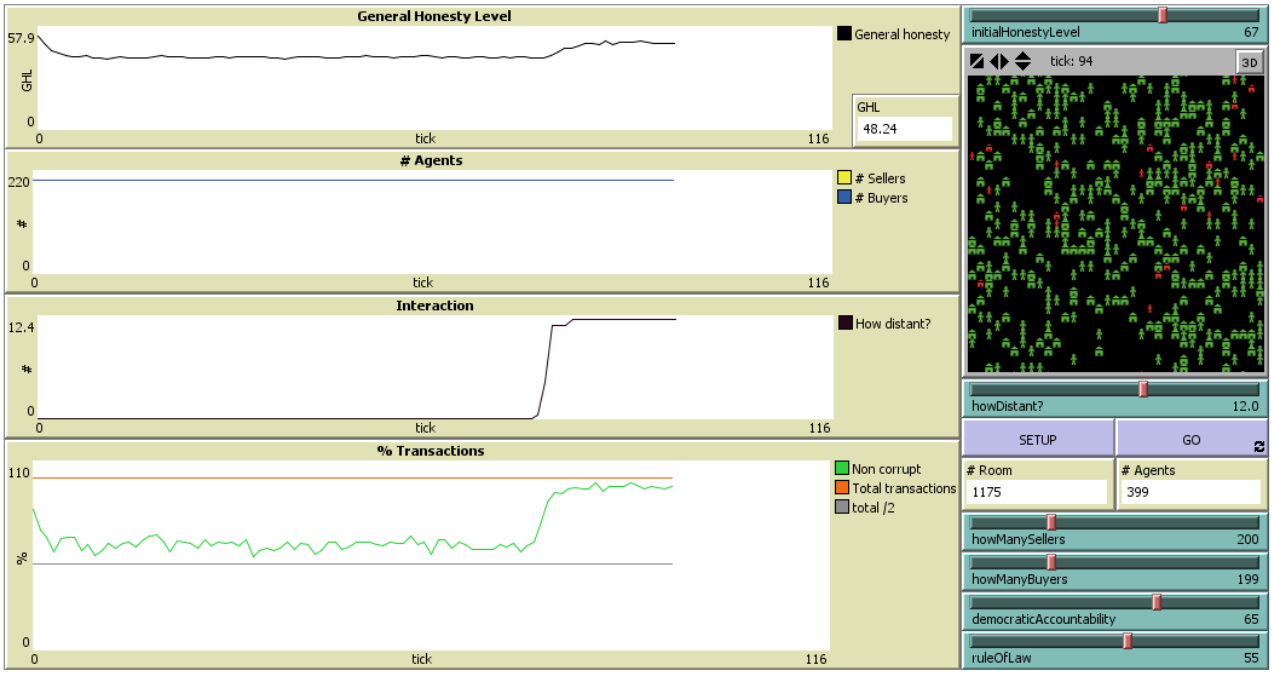
Experiments

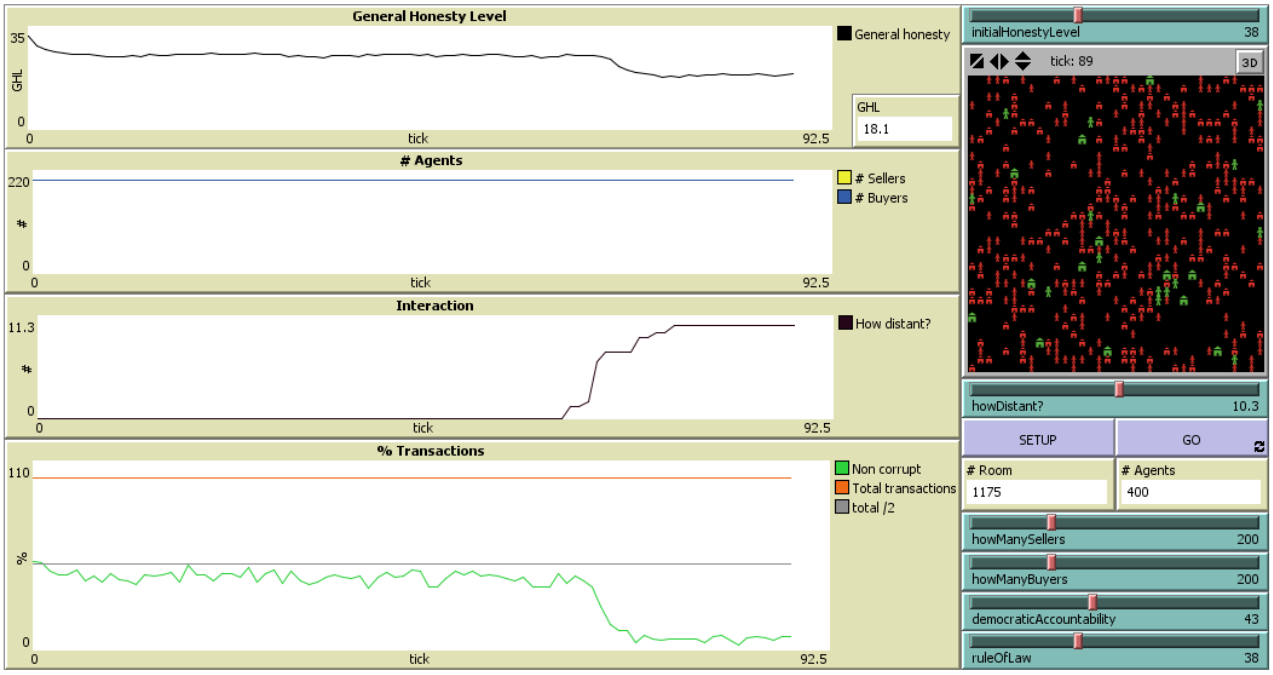
We can see from the next pictures how the simulation works. In the first one we see that setting the variable in an average range, the general honesty level has a quite stable run. The same can be said about the number of non corrupt transactions.



The situation changes as soon as we modify some variables. What happened from a certain point, is that we increased the *howDistant?* value, allowing agents to "see" around them. We can observe that both the curves have a notable increase. As we expected working on this variable remarkably modify our agents' behavior, in both the cases, there is an increase or decrease of our main variable. This could be read as follows: if we are in a situation where agents cannot check what happens around them, they keep on behave following only their own instinct and the general situation doesn't change much. On the contrary, if we are in the opposite situation where agents have the possibility to modify their behavior as a consequence of the environment in which they live, the result is the one we explained, their interactions would change the environment too. This is what we expected building our model, changing the available information agents would modify their behavior.

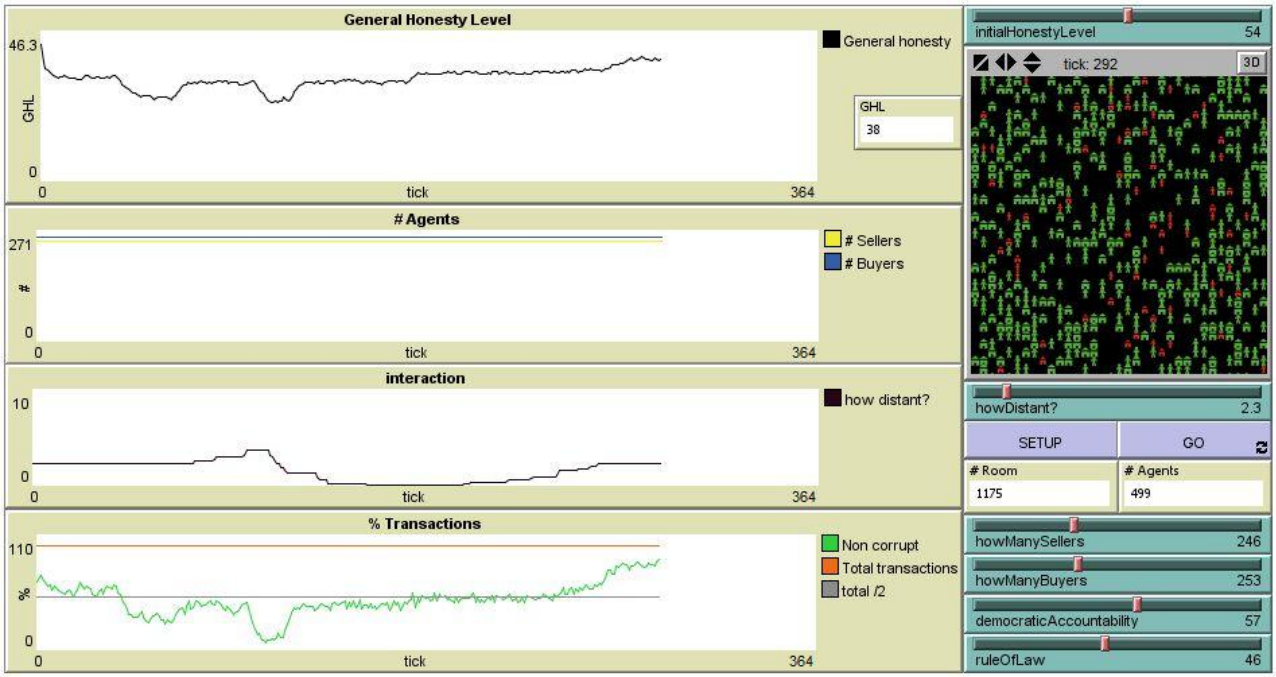
We can see we just explained from the next two pictures. There are two similar cases, with opposite settings, the first where variables are set randomly high and the second one where are set low. The output follows what just observed.





Until we let the variable *howDistant?* equal to zero, the general situation is pretty much constant. It changes radically after we change that value. We can observe that in the second case the general honesty level doesn't go below a certain value. The reason comes from the fact that *honestyLevel*

can’t be lower than zero for construction. One thing that should be noticed is that the situation where agents cannot see at all the behavior of other agents is definitely unusual in any society. An example that could be more interesting from this point of view is the one that follows.



Here we set our variables in order to simulate a sort of real society with values in an average range. We can notice that all the curves in the different graphs are correlated and that their trends are consistent with the assumptions we made building our simulation.