Model verification

L. Andrew Bollinger

6-10-2013

Lecture goals

- What is model verification?
- What are the steps for verifying an ABM?
- Verifying *your* model

What is model verification?

- **At this point:**
  - Your model is coded and (sort of) works
  - But can't be sure it's working *correctly*
- **Key question:**
  - Has the model been implemented correctly?
  - Have all the relevant entities and relationships from the conceptual model been translated into the computational model correctly?
  - "Have we built the thing right?" (not "Have we built the *right thing*?")
- Building up an *evidence file*
- Making sure the model generates *insights*, not *artifacts*

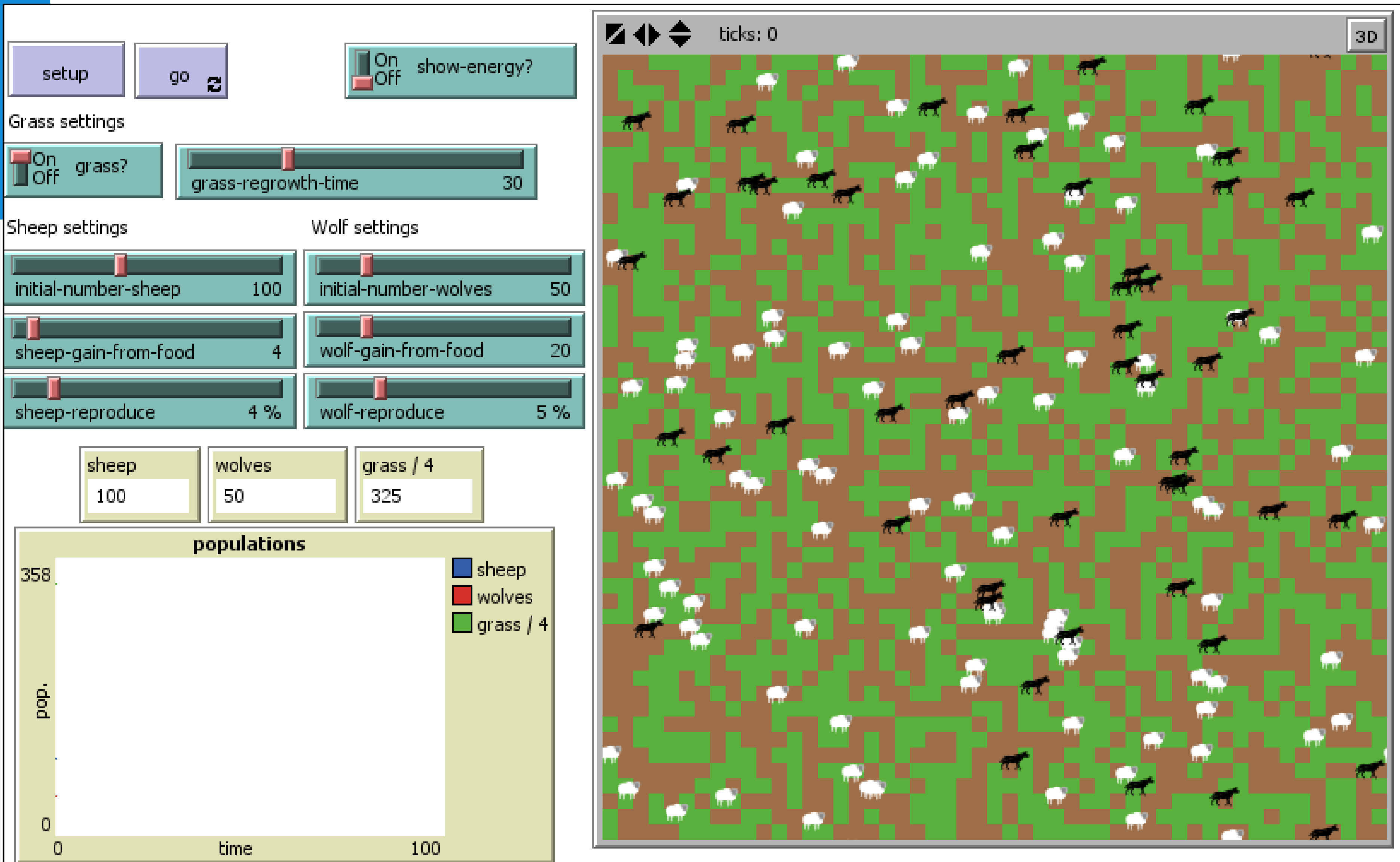
The 4 steps for verifying an ABM

1. Recording and tracking agent behavior
2. Single-agent testing
3. Interaction testing in a minimal model
4. Multi-agent testing

Modified version of the wolf-sheep predation model as an example

- see LectureModelVerification page on the Wiki to download

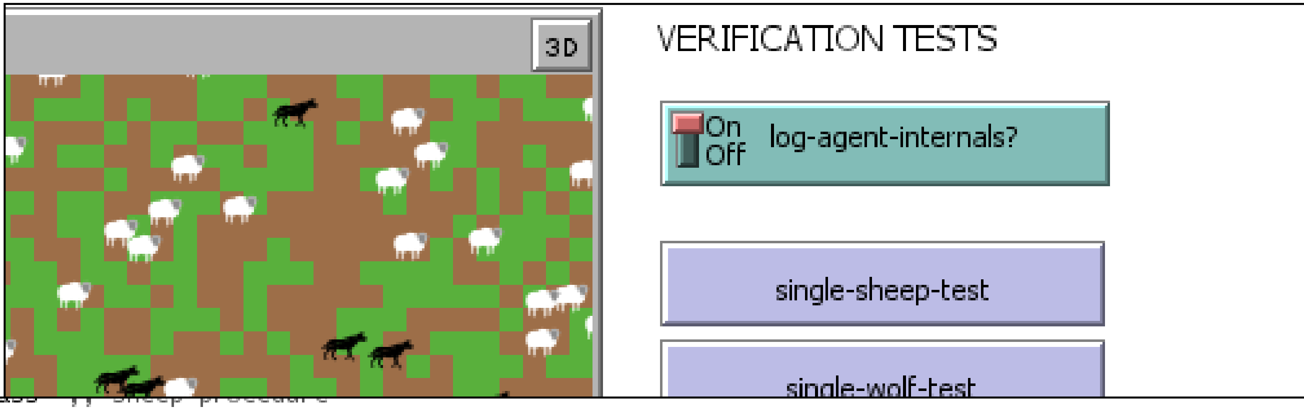
Wolf-sheep predation model



Step 1: Recording and tracking agent behavior

- **What:** Select relevant output variables and set up a way to monitor their values
- **How:**
  - OPTION 1: Record the inputs, states and outputs of each *agent*
  - OPTION 2: Record the inputs, states or outputs of each *internal process*
  - OPTION 3: Walk through the source code using a *debugger*

Recording and tracking agent behavior (example)




```
to eat-grass  ;; sheep procedure
  ;; sheep eat grass, turn the patch brown
  if pcolor = green [
    set pcolor brown
    let previous-energy energy
    set energy energy + sheep-gain-from-food  ;; sheep gain energy by eating
    if (log-agent-internals?) [print (word self " just ate some grass. energy was " previous-energy ". energy is now " energy)]
  ]
end

to reproduce-sheep  ;; sheep procedure
  if random-float 100 < sheep-reproduce [  ;; throw "dice" to see if you will reproduce
    let previous-energy energy
    set energy (energy / 2)                ;; divide energy between parent and offspring
    hatch 1 [ rt random-float 360 fd 1 ]   ;; hatch an offspring and move it forward 1 step
    if (log-agent-internals?) [print (word self " had a baby. energy was " previous-energy ". energy is now " energy)]
  ]
end

to reproduce-wolves  ;; wolf procedure
  if random-float 100 < wolf-reproduce [  ;; throw "dice" to see if you will reproduce
    let previous-energy energy
    set energy (energy / 2)               ;; divide energy between parent and offspring
    hatch 1 [ rt random-float 360 fd 1 ]  ;; hatch an offspring and move it forward 1 step
    if (log-agent-internals?) [print (word self " had a baby. energy was " previous-energy ". energy is now " energy)]
  ]
end

to catch-sheep  ;; wolf procedure
  let prey one-of sheep-here                    ;; grab a random sheep
  if prey != nobody                             ;; did we get one?  if so,
    [ let sheep-eaten (word prey)
      ask prey [ die ]                          ;; kill it
```

Recording and tracking agent behavior (example)

Step 2: Single-agent testing

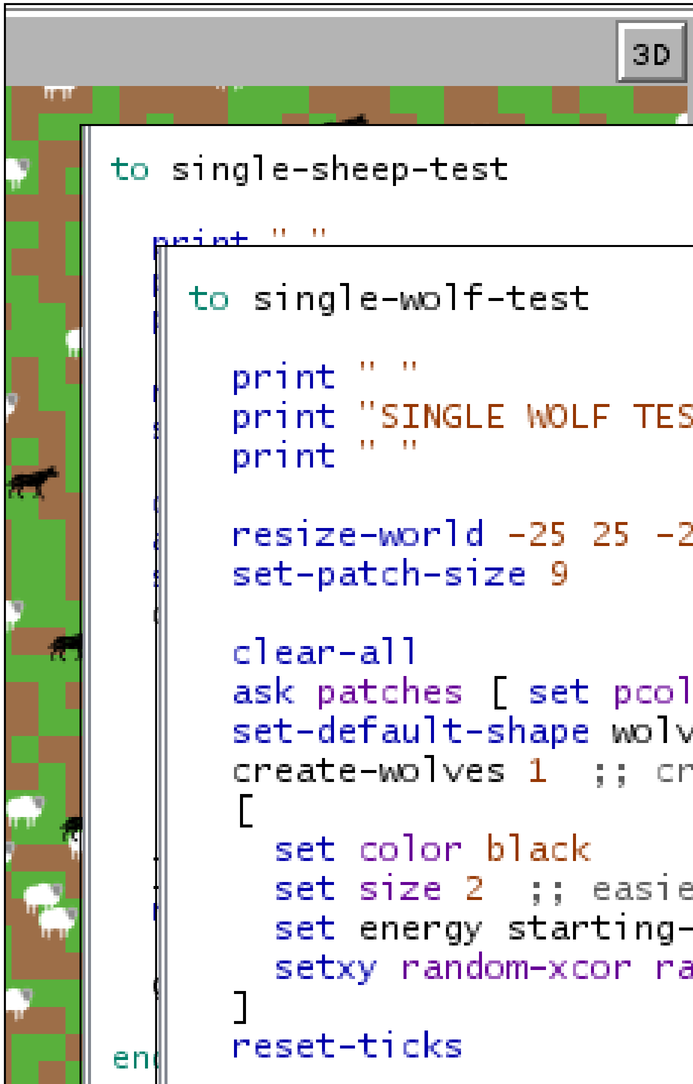
- **What:** Explore the behavior of a *single agent*
- **2 sets of tests:**
  1. Theoretical prediction tests and sanity checks
     - Tests using *normal* inputs
     - Does the agent behave as expected under normal conditions?
  2. "Break-the-agent" tests
     - Tests using *extreme* inputs
     - Where are the edges of normal behavior?

Step 2: Single-agent testing

- **Theoretical prediction tests and sanity checks:**
  1. Define some “normal” inputs to the agent
  2. Explicitly predict how the agent will behave given these inputs
  3. Test how the agent behaves given these inputs
  4. If not as predicted, check if it’s a logical error or an implementation error.

- **Break-the-agent tests**
  1. Define some “extreme” inputs to the agent (e.g. zero, negative numbers, extremely high numbers, decimals)
  2. Predict and test how the agent behaves given these inputs
  3. Define boundaries for agent input variables (and make sure the agent will never receive values outside these boundaries)

Single-agent testing (example)




```
to single-sheep-test

  print " "
```

```
to single-wolf-test

  print " "
  print "SINGLE WOLF TEST"
  print " "

  resize-world -25 25 -25 25
  set-patch-size 9

  clear-all
  ask patches [ set pcolor green ]
  set-default-shape wolves "wolf"
  create-wolves 1  ;; create the wolves, then initialize their variables
  [
    set color black
    set size 2  ;; easier to see
    set energy starting-wolf-energy
    setxy random-xcor random-ycor
  ]
  reset-ticks

  go

end
```

Model verification example – single agent and minimal model interaction testing

Single agent: Update-Satisfaction

- If all neighbours have a Profpersurf of 100000 (assuming normal profits are way below this figure), the Statisfaction of the single agent should be -1 for all the technologies currently owned. **Confirmed.**
- If all neighbours have a Profpersurf of 100000 (assuming normal profits are way above this figure), the Statisfaction of the single agent should be 1 for all the technologies currently owned. **Confirmed.**
- If all neighbours have a Profpersurf of 0, the Statisfaction of the single agent should be 1 or -1 for all the technolocies currently owned, depending whether its Profpersurf is positive or negative. **Confirmed.**

Single agent: Update-opinionlibraries

- If one neighbour has an opinion of 1 for a given technology (and the Opinionchangerate is 1 and the Stubbornness is 0) the Opinionlibrary of the single agent should change from 0 on all technologies to 1 on that given technology (rest remains 0) after one tick. **Confirmed.**
- If one neighbour has an opinion of -1 for a given technology (and the Opinionchangerate is 1 and the Stubbornness is 0) the Opinionlibrary of the single agent should change from 0 on all technologies to -1 on the given technology (rest remains 0) after one tick. **Confirmed.**
- If one neighbour has an opinion of 1 for all the technolocies (and the Opionionchangerate is 1 and the stubborness is 0) the Opinionlibrary of the single agent should change from 0 on all technologies to 1 on all technologies. **Confirmed.**

Minimal model: Technology and Satisfaction update

- If one neighbour has a given technology but another does not, then after the technology update code, the new technology should appear in the second agent's technology library. **Error found**. Technologies were added to the second agent's technology library that did not correspond to technologies owned by neighbours.

Example from the book

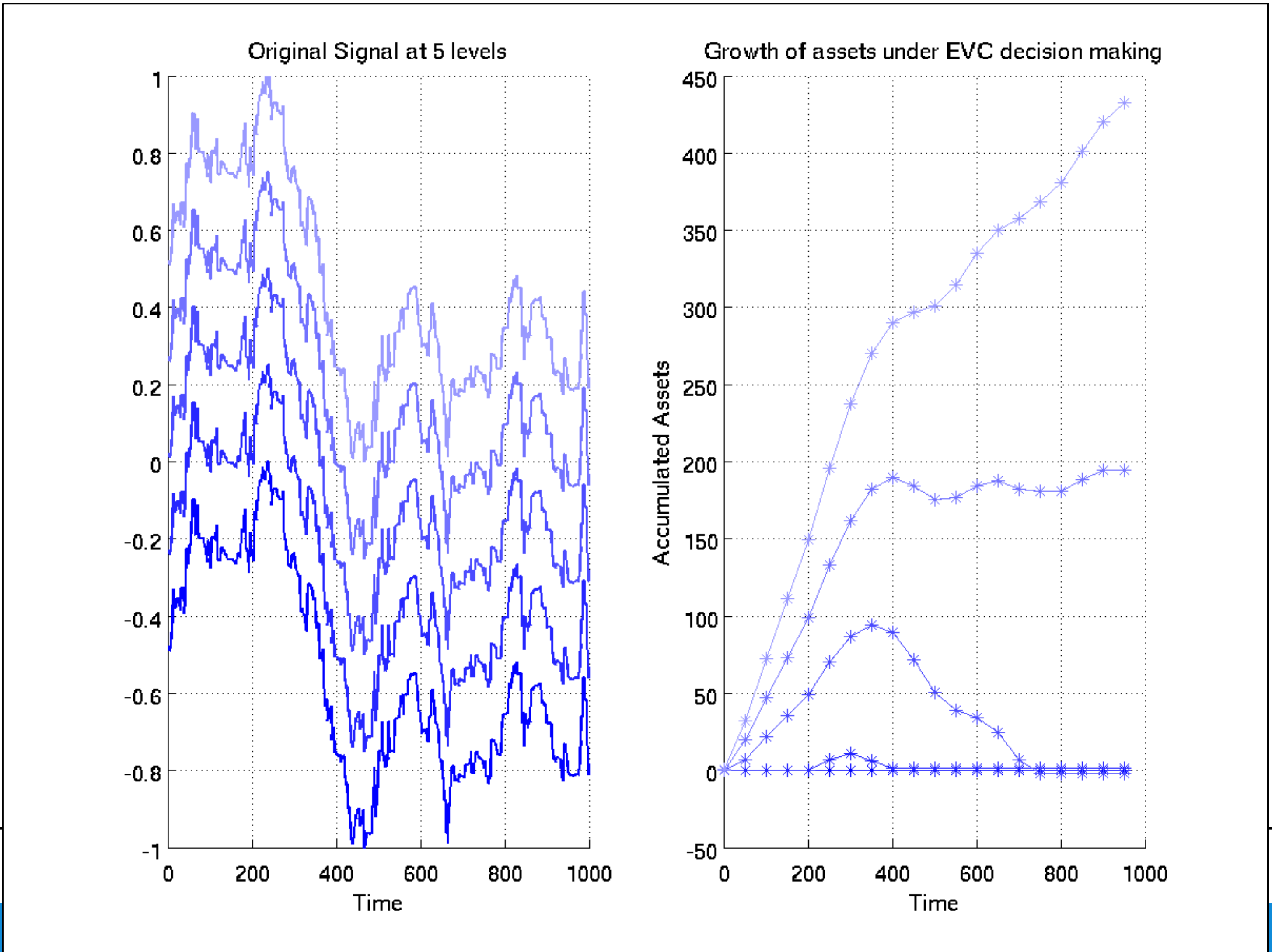
Step 2: Single-agent testing

What if the agent has a memory?

- The agent needs some sort of “history” to make decisions
- We need to create some artificial histories and see how the agent performs.
- **Dynamic signal testing:**
  - Test the agent with different time-varying signals
  - E.g. random signals, signals with continuous in/decreasing values, signals with step functions and power law distribution of values

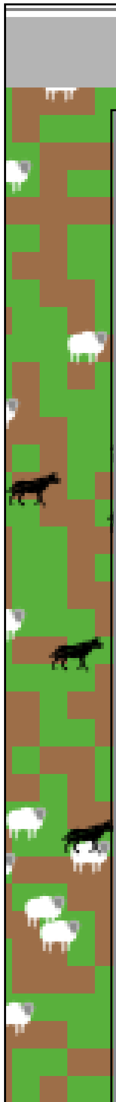
Dynamic signal testing (example)

Accumulation of assets as a function of a series of input signals
MSc thesis Theo van Ruiven



Step 3: Interaction testing in a minimal model

- **What:** Explore the behavior of a minimal set of agents
  - If only one type of agent, include 2 of them
  - If more than one type of agent, include one of each
- **Same tests as in the previous step:**
  1. Theoretical prediction tests and sanity checks
  2. Break-the-agent tests
- **Answer the following questions**:
  - Do the agents find each other?
  - Do the agent interactions happen as defined in the narrative?
  - Are the results of these interactions as expected?
  - Are there any unintended interactions?



VERIFICATION TESTS

```
to minimal-model-test

  print " "
  print "MINIMAL MODEL TEST"
  print " "

  resize-world 0 1 0 0
  set-patch-size 100

  clear-all
  ask patches [ set pcolor green ]
  set-default-shape sheep "sheep"
   set-default-shape sheep "sheep"
  create-sheep 1  ;; create the sheep, then initialize their variables
  [
    set color white
    set size 1.5  ;; easier to see
    set label-color blue - 2
    set energy random (2 * sheep-gain-from-food)
    setxy random-xcor random-ycor
  ]
  set-default-shape wolves "wolf"
  create-wolves 1  ;; create the wolves, then initialize their variables
  [
    set color black
    set size 2  ;; easier to see
    set energy random (2 * wolf-gain-from-food)
    setxy random-xcor random-ycor
  ]
  reset-ticks

  go

end
```

Step 4: Multi-agent testing

- **What:** Explore the behavior of the entire model with all agents present.
- **4 sets of tests:**
  1. Theoretical prediction tests and sanity checks
  2. Break-the-agent tests
  3. Variability tests
  4. Timeline sanity tests

Step 4: Multi-agent testing

- **Variability tests**
  - **What:** Explore the variability of the output in different regions of the parameter space
  - **How:**
    1. Many repetitions (100-1000) across the parameter space
    2. Collect values for multiple outputs variables
    3. Statistical examination of the results (e.g. variance, std. dev.)
    4. Do strange outcomes make sense, or are they artifacts?

Variability testing (example 1)

Source: SPM 9555 report of Manuel Harmsen and Job Veltman

2. For each input variable value , determine by theory which effect it should have on each output varia

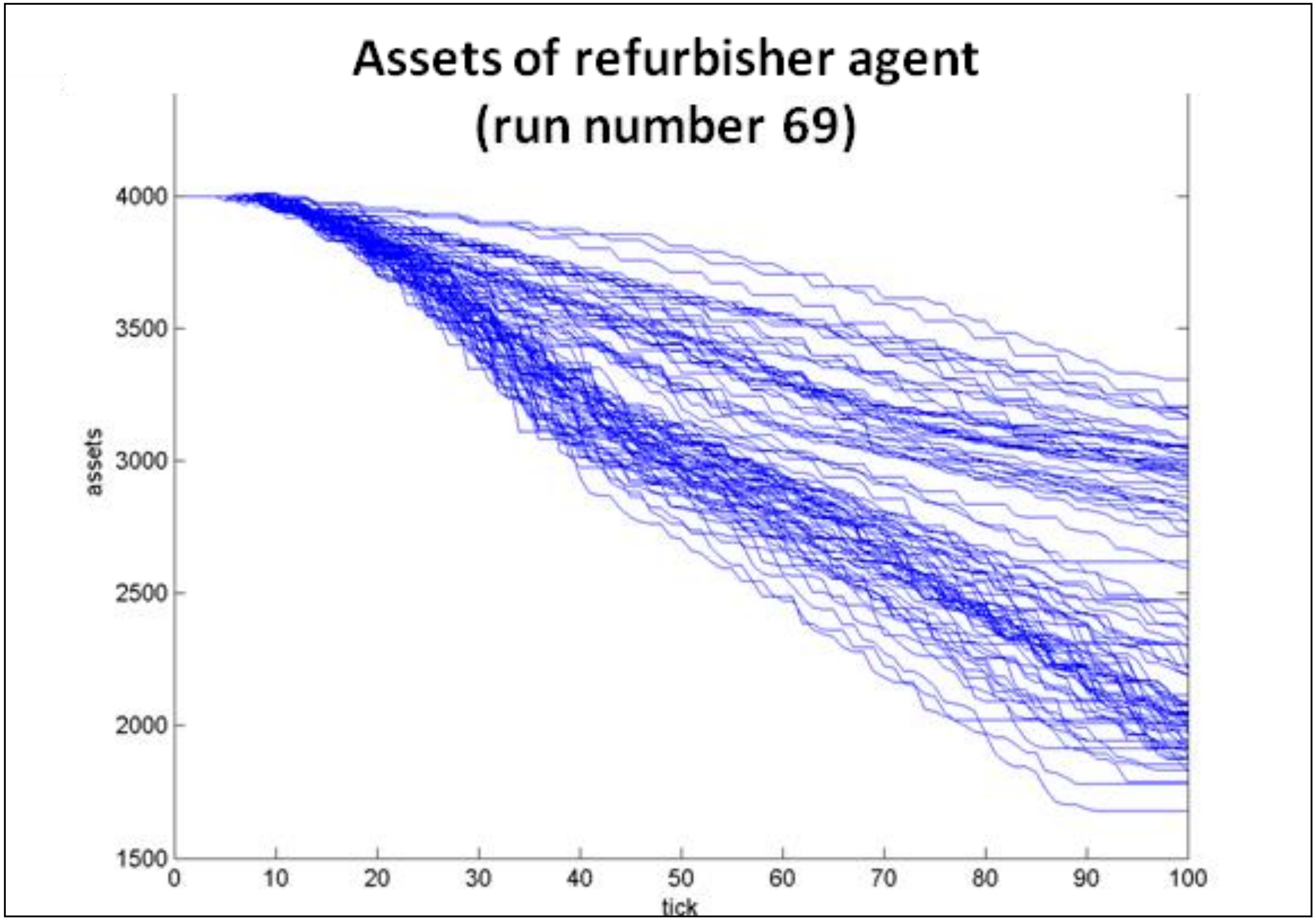
Output / Input	Share of biggest chain	POR balance	Product types	Dismantling costs
↑ Tender levy	↘	↑	↓	↗
↑ Tender rent	↓	↗	↓	↗
↑ Chains	↗	↗	→	↗
↑ Products/chain	→	→	↑	→
↑ Long trend	↓	→	?	↑
↑ Short trend	↘	→	?	↗
↑ Clustering fctr	↗	?	↑	?

3. Execute a whole range of verification tests by choosing standard settings and then change one var

Output / Input	Share of biggest chain	POR balance	Product types	Dismantling costs
↑ Tender levy	↘	↑	↓	↗
↑ Tender rent	↘[4]	↗	↓	↗
↑ Chains	↗	↗	→	↗
↑ Products/chain	→	→	↗	→
↑ Long trend	↓	→	?[1]	↑
↑ Short trend	↘	→	?[1]	↗
↑ Clustering fctr	↘[3]	?[1]	↗	?[1]

Variability testing (example 2)

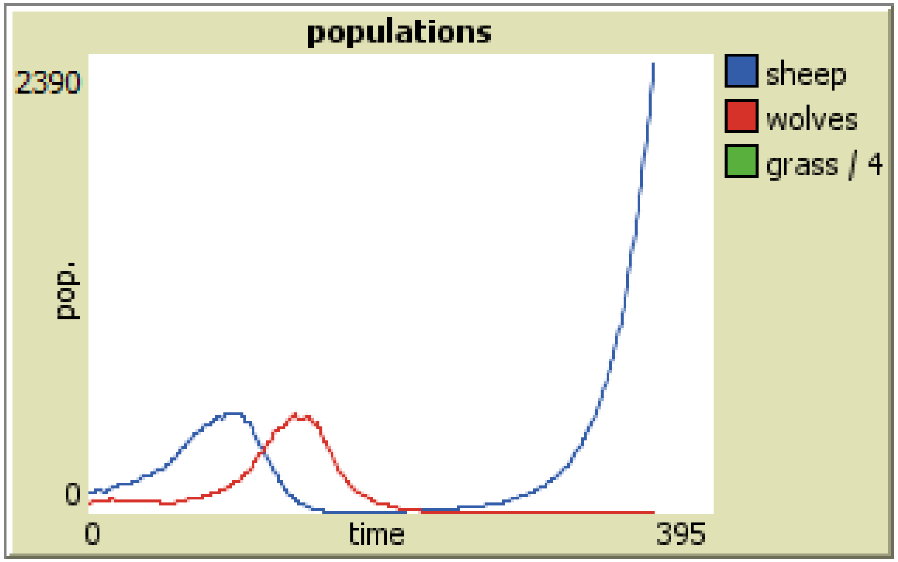
Source: MSc thesis Andrew Bollinger

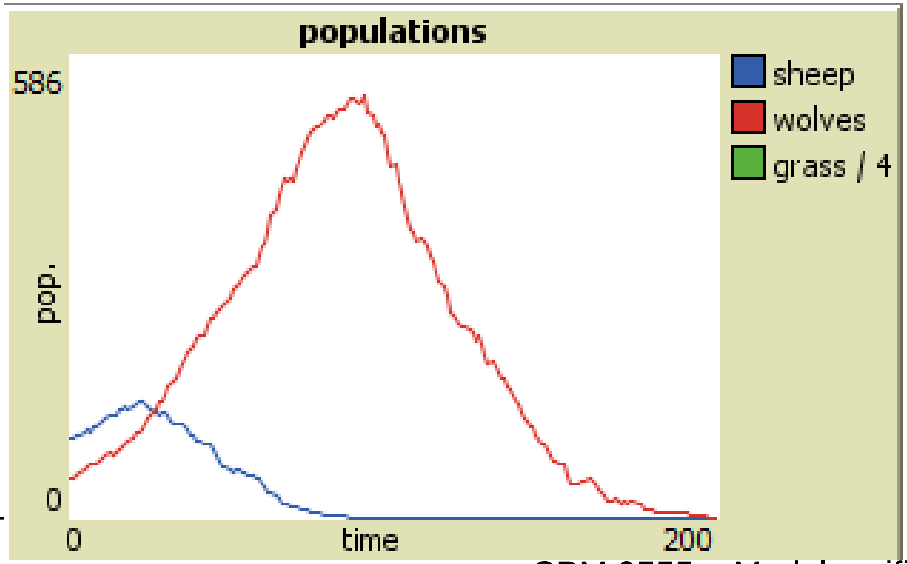


Step 4: Multi-agent testing

- **Timeline sanity tests**
  - **What:** Can the outputs be explained by reasoning through the model logic?
  - **How:**
    1. Perform several runs at the default parameter settings
    2. Examine the output plots carefully
    3. Are there any patterns you can't explain?

Timeline sanity testing (example)





Verifying *your* model

- Remember, you're building an *evidence file.*
- You're demonstrating to yourself and others:
  1. This model is bug-free (as much as possible)
  2. I know where this model starts to give bogus results
  3. I/you can be confident that the results are not artifacts
- What should be in your report:
  - See the example from the book (Chapter 3.6.5)
  - See the examples from the previous SPM 9555 class (e.g. Maasvlakte model)