



How to Import a Vensim Model into AnyLogic and Combine it with Agent Based Model

Tutorial

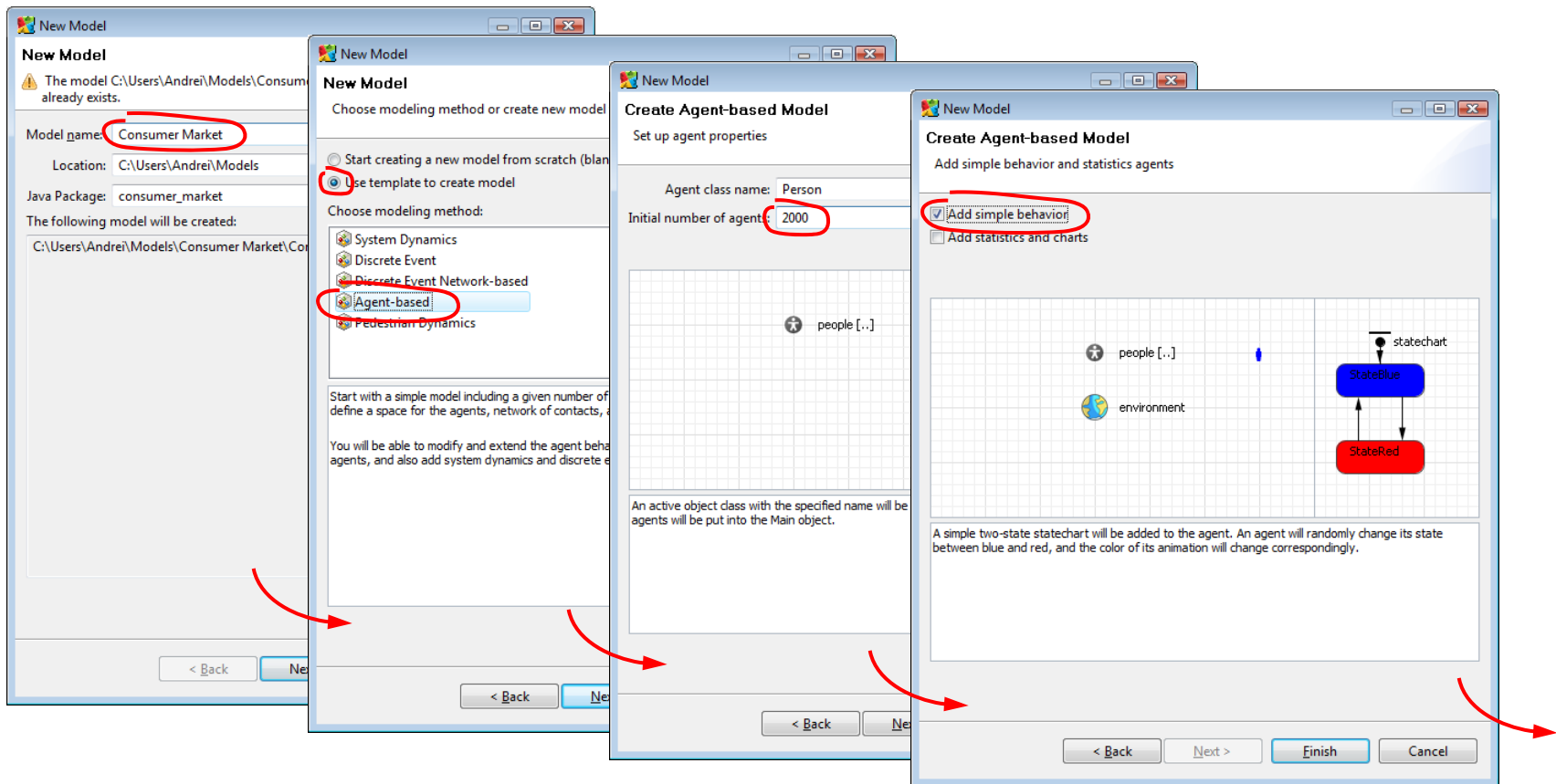
International System Dynamics Conference
Albuquerque, USA
July 2009

The plan

- Create an AB model in AnyLogic using wizard and template
- Modify the behavior of agent (consumer)
- Import a Vensim model of a supply chain
- Integrate the system dynamics model of supply chain with agent based model of consumer market
- Add interaction between agents
- Add statistics collection and visualization
- Export the model as Java applet

Step 1.1 Create an agent based model

- Use **New Model** wizard and Agent Based model template



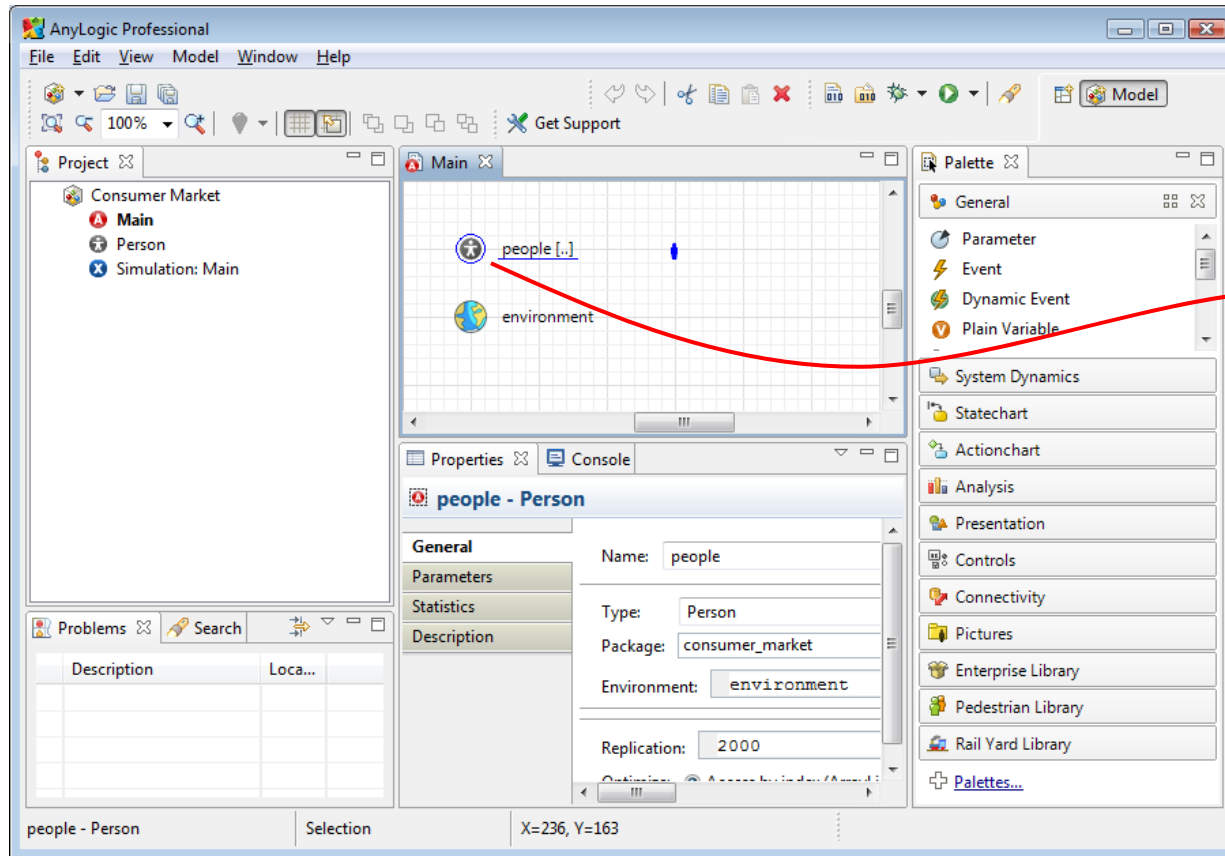
Step 1.1 Comments

When you are creating a new model, AnyLogic offers you a set of templates you can use to minimize your work. Each template contains the basic constructs you would typically create when developing a model using a particular method: system dynamics, process-based, agent based modeling, pedestrian dynamics, etc.

The agent based model template includes two classes: one for agent (in our case Person), and one for environment (Main). You can choose the initial number of agents (2000). Although the physical space will not be important in our consumer market model, we will use it for visualization. We will assume that any person can contact any other person, so we skip the network step of the wizard.

Frequently the event-driven behavior of agents is modeled using statecharts, and a simple statechart can be added to class Person. We will modify the statechart later to make it meaningful.

Step 1.2 After New Model wizard



x 2000

Step 1.2 Comments

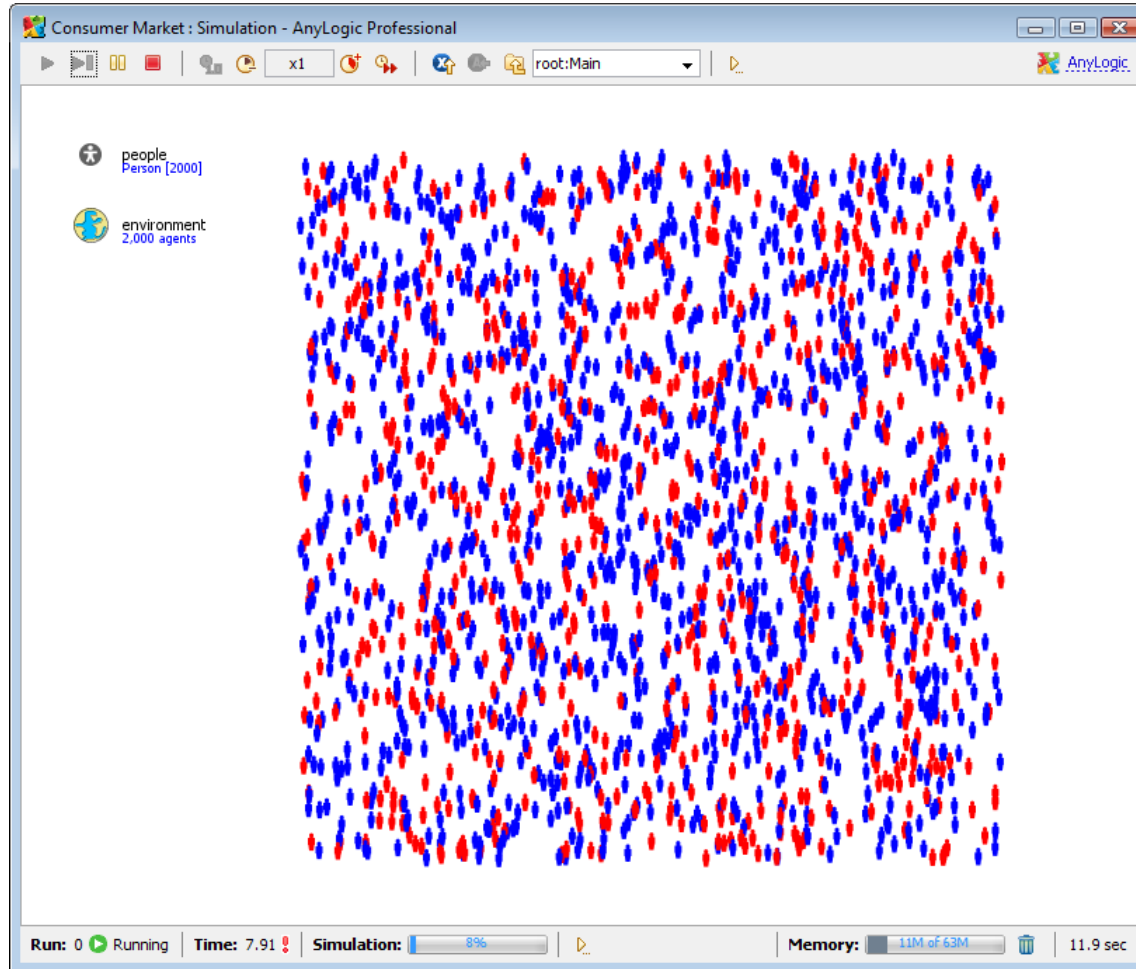
When the New Model Wizard finishes, a newly created model opens in AnyLogic model development environment. All objects of the model can be accessed from the Project tree on the left. The top-level items are active object classes Person and Main and also a class Experiment – this is the where you can set up the parameters of your simulation and run it.

The Main object (its diagram opens right away) contains a collection of agents (people) and environment object (this is where the space, network, and layout are defined).

If you double-click the item Person in the tree, its internal structure opens in another graphical editor. There you will be able to see the statechart with two states (blue and red) and transitions from one to another, and animation of agent – a small picture.

The model is ready to run, you can choose run from the context menu of Experiment, or press Run button on the toolbar.

Step 1.3 Run the model




Step 1.3 Comments

The model starts in a separate window and initially shows the object Main. The agents' animations (small pictures of a man) are randomly distributed in the space 500 x 500 pixels.

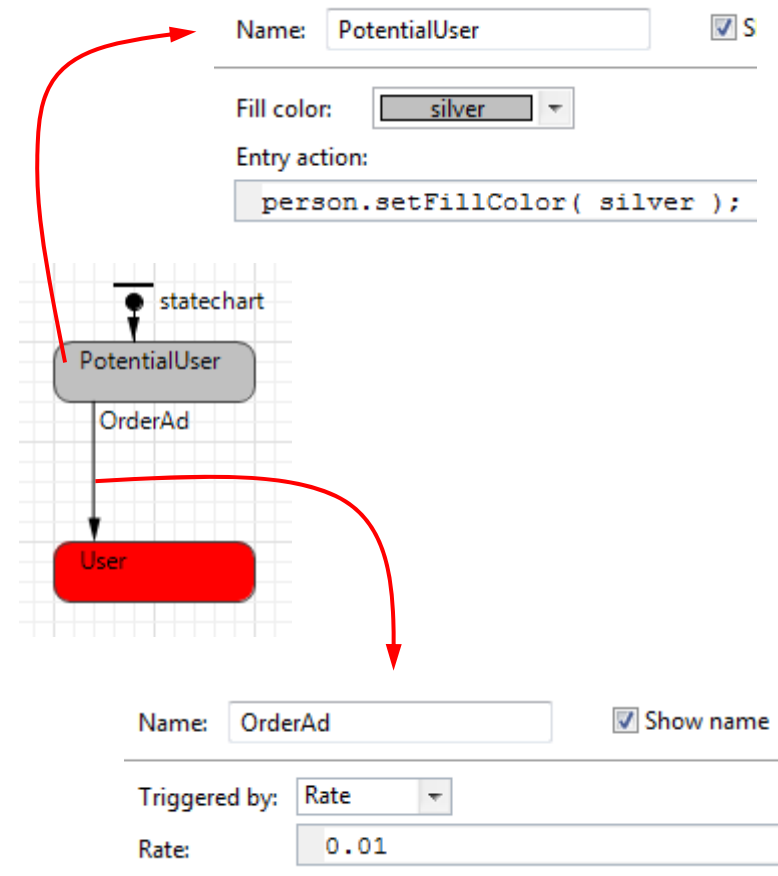
As each agent is set up to randomly change its state between blue and red, and when entering a state it sets the color of its animation, the pictures will turn from blue to red and vice versa.

You can look into the internals of any of 2000 agent at runtime – use Navigation section of the toolbar.

The default settings of the experiment are to run simulation for 100 model time units (e.g. days) and stop. The model will start running in scale to real time (one model time unit = 1 real time second), but you can change it using the toolbar. The button  switches the model to virtual time (fastest) execution mode.

Step 2.1 Modify the statechart of the agent

- Rename the upper (Blue) state to **PotentialUser** and change its color to **silver**
- Change its **Entry action** to `person.setFillColor(silver);`
- Rename the **Red** state to **User**
- Delete transition from **User** to **PotentialUser**
- Rename the remaining transition to **OrderAd** (order because of ad influence)
- Change the rate of transition to **0.01** (advertising results in one order in 100 days)

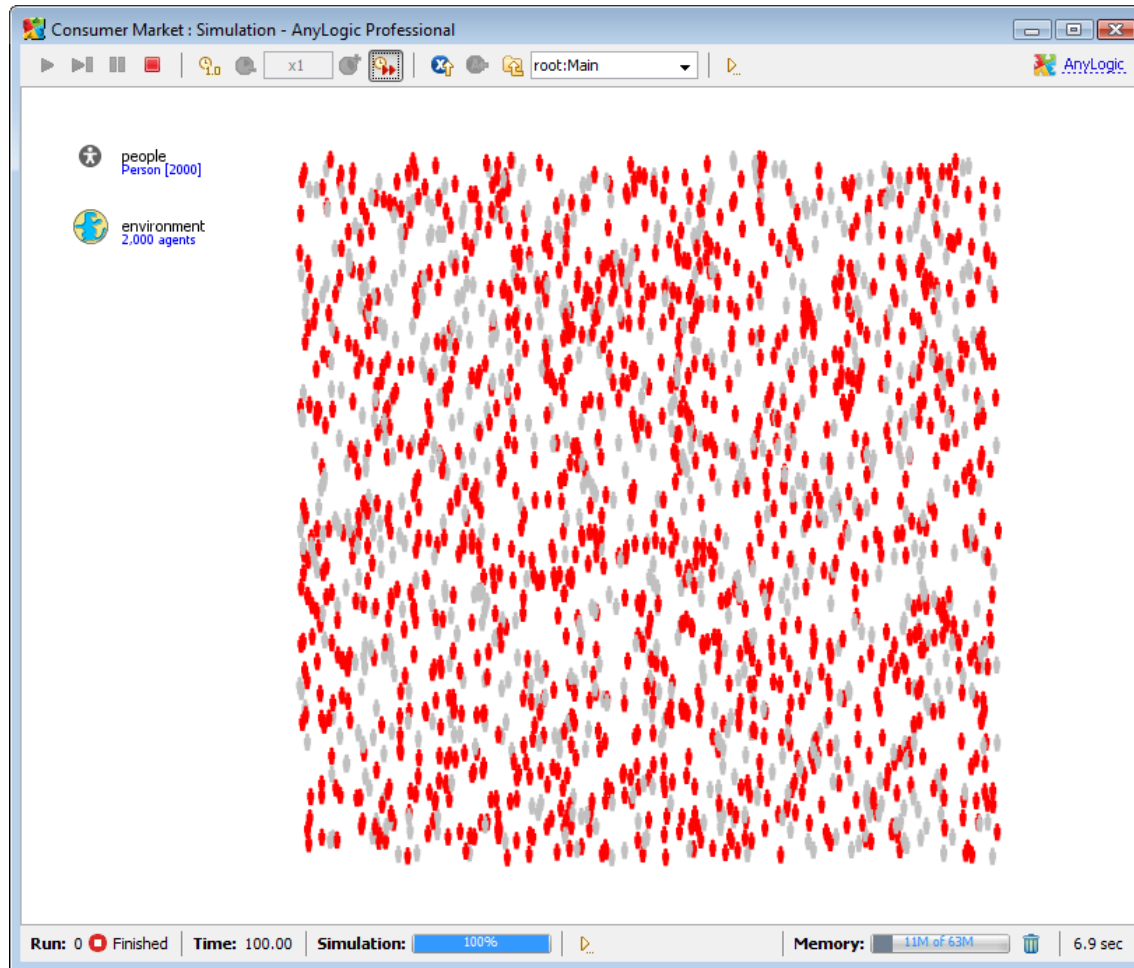


Step 2.1 Comments

We are now modifying the statechart to make it meaningful. On this step we will assume that a person (consumer) can will have two states: PotentialUser (before purchase) and User – after purchase. The event of purchasing the product will be modeled by a transition from the first state to the second. At this step we will only model purchase triggered by advertising, so the transition is set up to execute after a random (exponentially distributed) timeout with mean time 100 – that is why we use option “Rate” and value 1/100.

Note that to access the properties of any item you should select it in the graphical editor or in the tree. The properties are displayed in the bottom window. When changing “blue” to “silver” you do not need to fully type the word “silver”, you can start typing it and press Ctrl+Enter to invoke “completion” feature of AnyLogic.

Step 2.2 Run the Consumer Market model

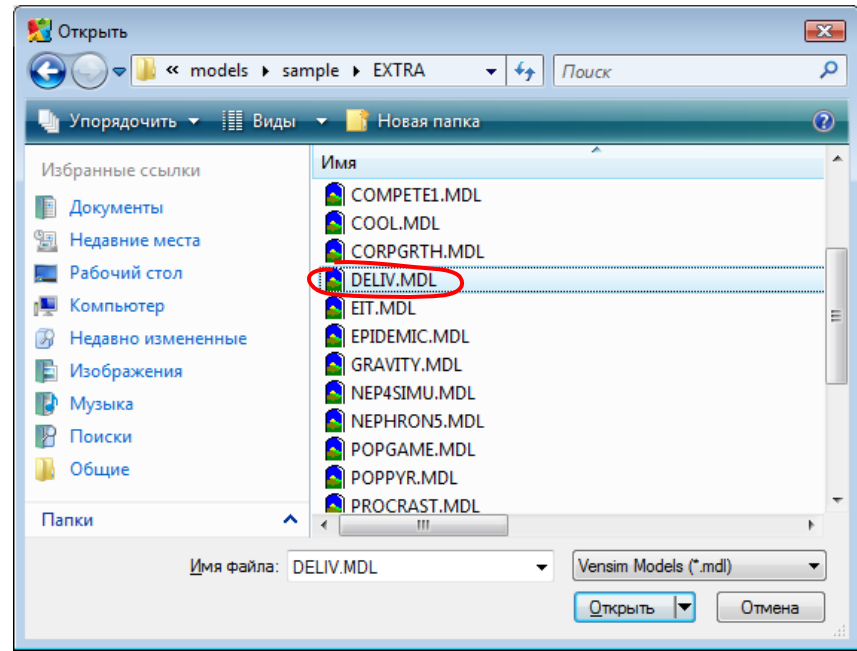
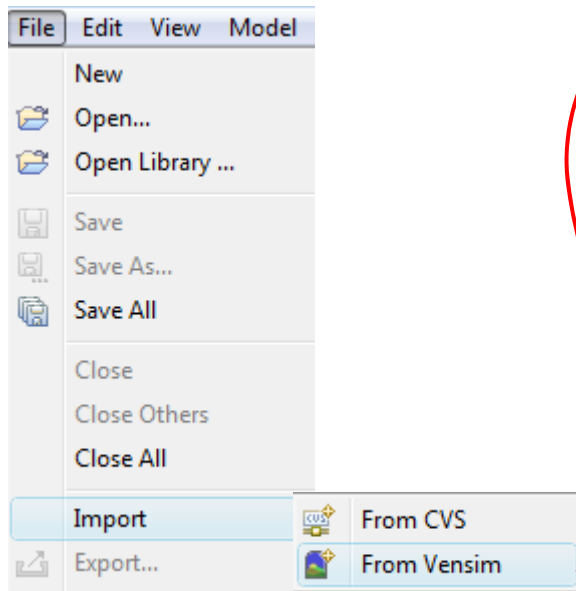


Step 2.2 Comments

The agents are now obviously turning from silver color to red and then stay red forever. By the end of the simulation (100 days) most people are red (i.e. most people become users of the product).

Step 3.1 Import a Vensim model

- Assuming Vensim is installed on your computer, locate the Vensim EXTRA models directory. It can be something like [C:\Program Files\Vensim\models\sample\EXTRA](#)
- In AnyLogic choose File | Import | From Vensim
- Open [DELIV.MDL](#) model



Step 3.1 Comments

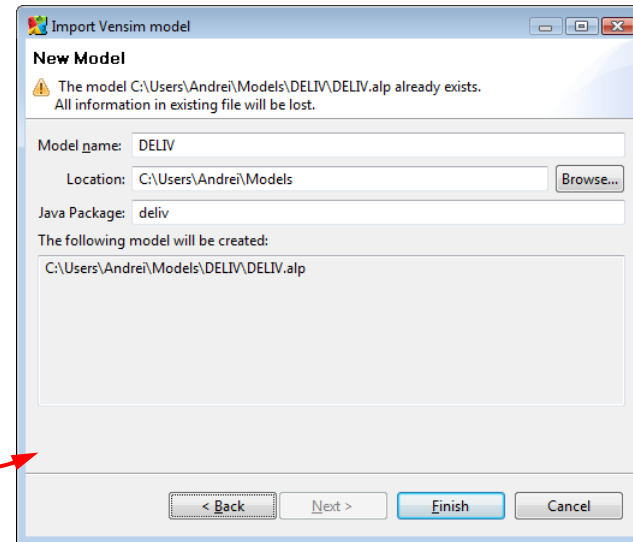
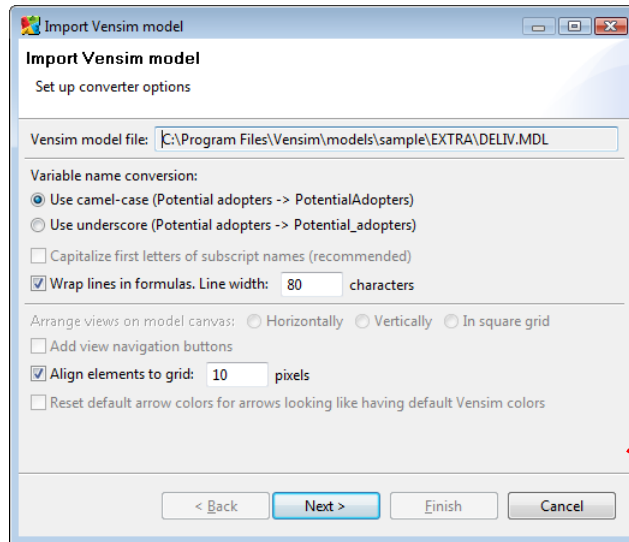
AnyLogic can import Vensim models stored in textual format (.MDL files). Binary format models (.VMF files) need to be saved in text format before you can import them.

The DELIV model you are about to import represents a simple supply chain of a product and contains a simple feedback loop reflecting the fact that the more product are sold, the more there are new orders – in other words, this is a simplest possible model of a product diffusion in the market.

Our plan is to replace that system dynamics feedback loop with a more realistic agent based model of a consumer market.

Step 3.2 Import Vensim model wizard

- Leave all options at their default values



Step 3.2 Comments

The Import from Vensim wizard offers some import options. For example, variable names in AnyLogic cannot contain spaces, so names with spaces can either be converted to CamelCase names or to names_with_underscores.

If the Vensim model contains multiple views, the wizard will create multiple AnyLogic view areas. In DELIV model there is just one view, so that option is grayed.

Step 3.3 After Import Vensim model wizard

The screenshot displays the AnyLogic Professional interface. The main workspace shows a system dynamics model diagram with various components and flows. The 'Project' pane on the left lists the model structure, with 'Main' under the 'DELIV' folder circled in red. The 'Properties' pane at the bottom shows the 'Main - Active Object Class' settings.

Project Structure:

- Consumer Market
 - Main
 - Person
 - Simulation: Main
 - DELIV** (circled in red)
 - Main** (circled in red)
 - Simulation: Main

Model Diagram Components:

- NormalEffectiveReferralFraction
- NewPeopleReadyToOrder
- ProductsPerCustomer
- MinimumCycleTime
- NewCustomers
- Customers
- Orders
- Backlog
- Shipments
- Capacity
- DeliveryDelay
- EffectOfDeliveryDelayOnOrders
- NormalDeliveryDelay
- EffectOfDeliveryDelayOnOrdersLookup

Properties Pane (Main - Active Object Class):

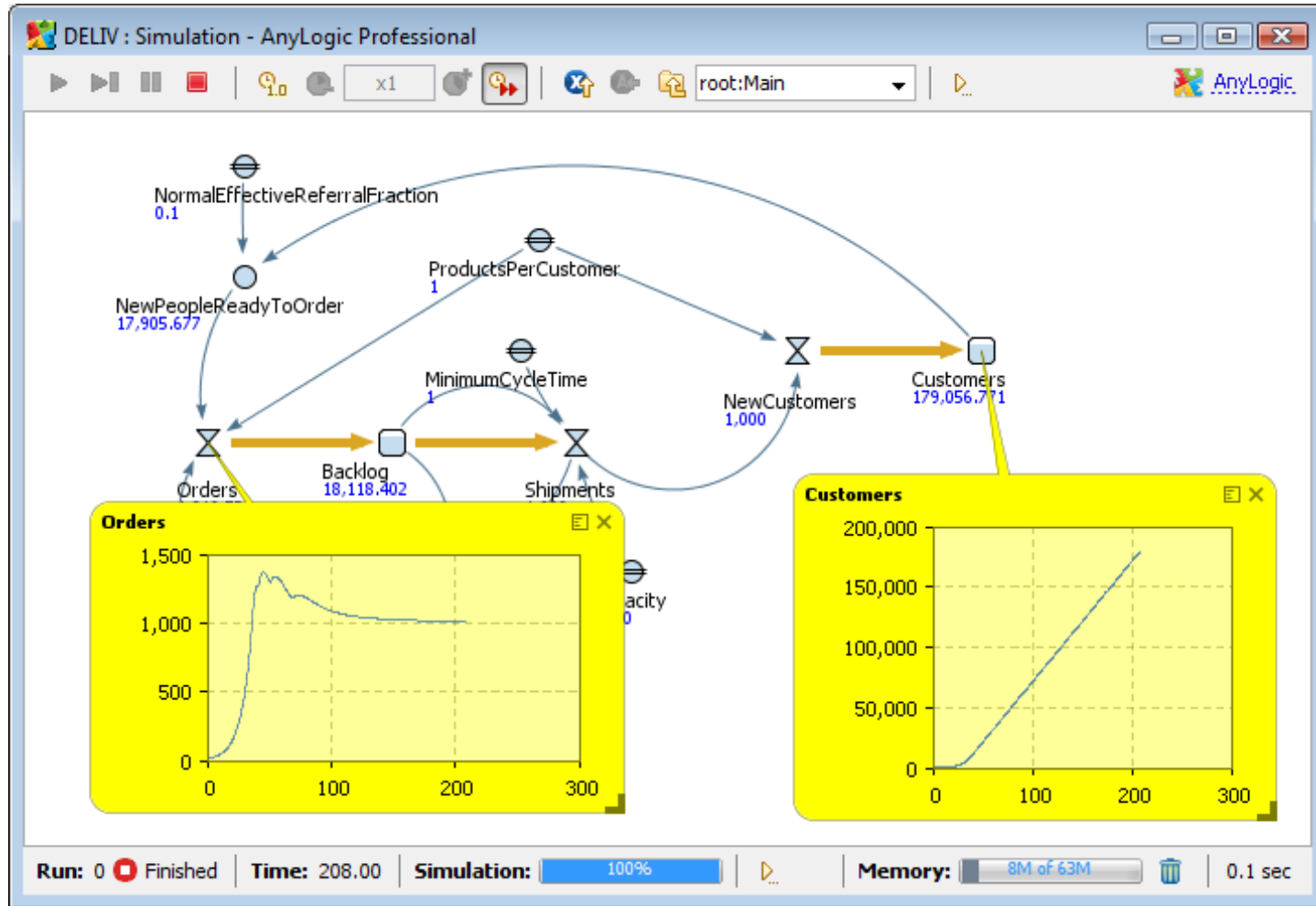
- General: Name: Main, Ignore:
- Advanced:
- Agent: Agent, Generic

Step 3.3 Comments

The imported model opens as a separate project (separate file) – see the DELIV top-level item in the Projects tree. Its Main object contains the stock-and flow diagram of the system dynamics model. The Vensim model layout is preserved. The model is ready to run.

To run the imported model choose Run from the context menu of the model's Simulation experiment.

Step 3.4 Run the imported Vensim model

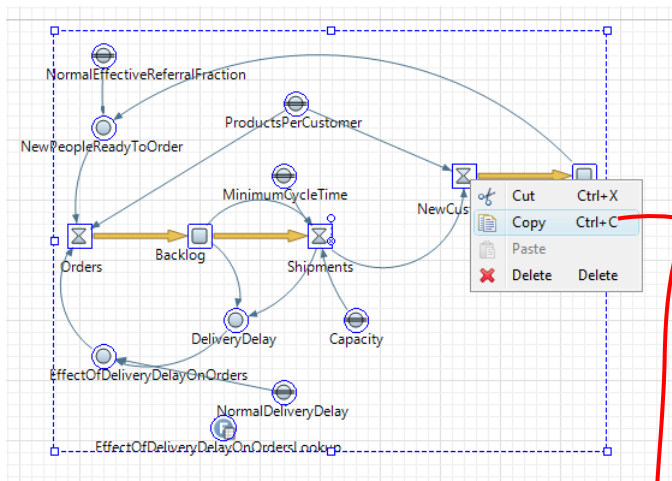


Step 3.4 Comments

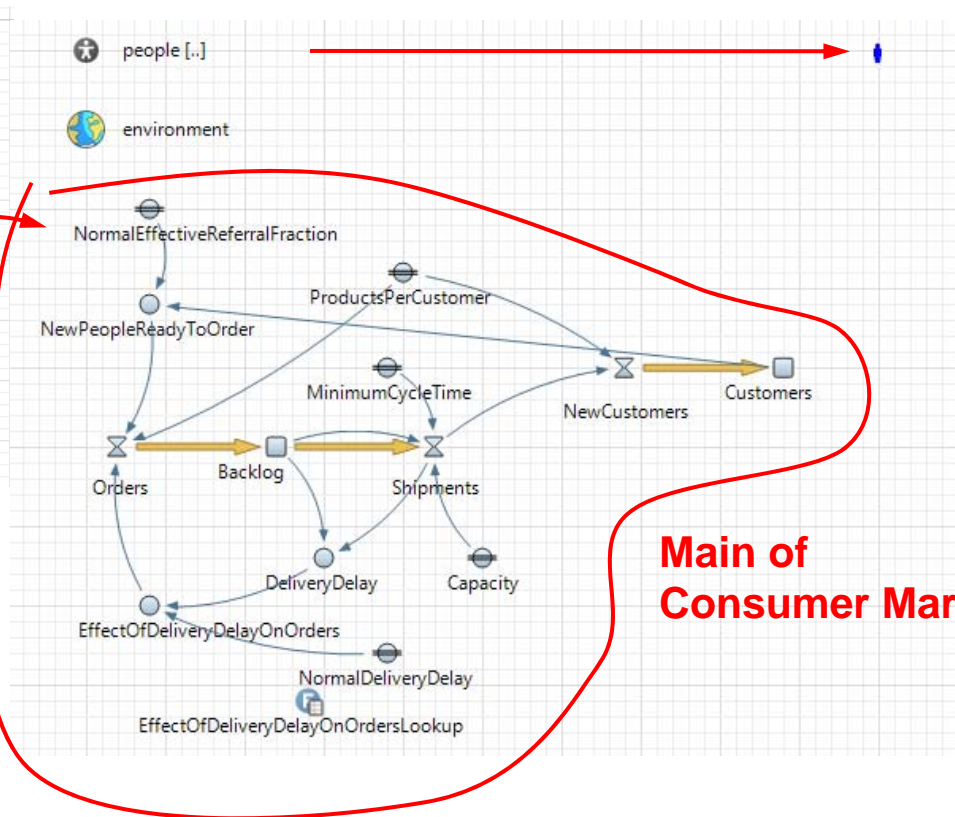
The model is set up to run in virtual time (as fast as possible), which is the default execution mode in Vensim. Therefore the model finishes almost instantly. If you click on a variable, the yellow callout pops up. In that callout window you can switch between the textual information on the selected variable, or its time chart.

Step 4.1 Copy SD diagram into AB model

- Select the whole SD diagram in Main of DELIV project, copy it
- Paste the diagram into Main of Consumer Market, arrange it



Main of DELIV



Main of Consumer Market

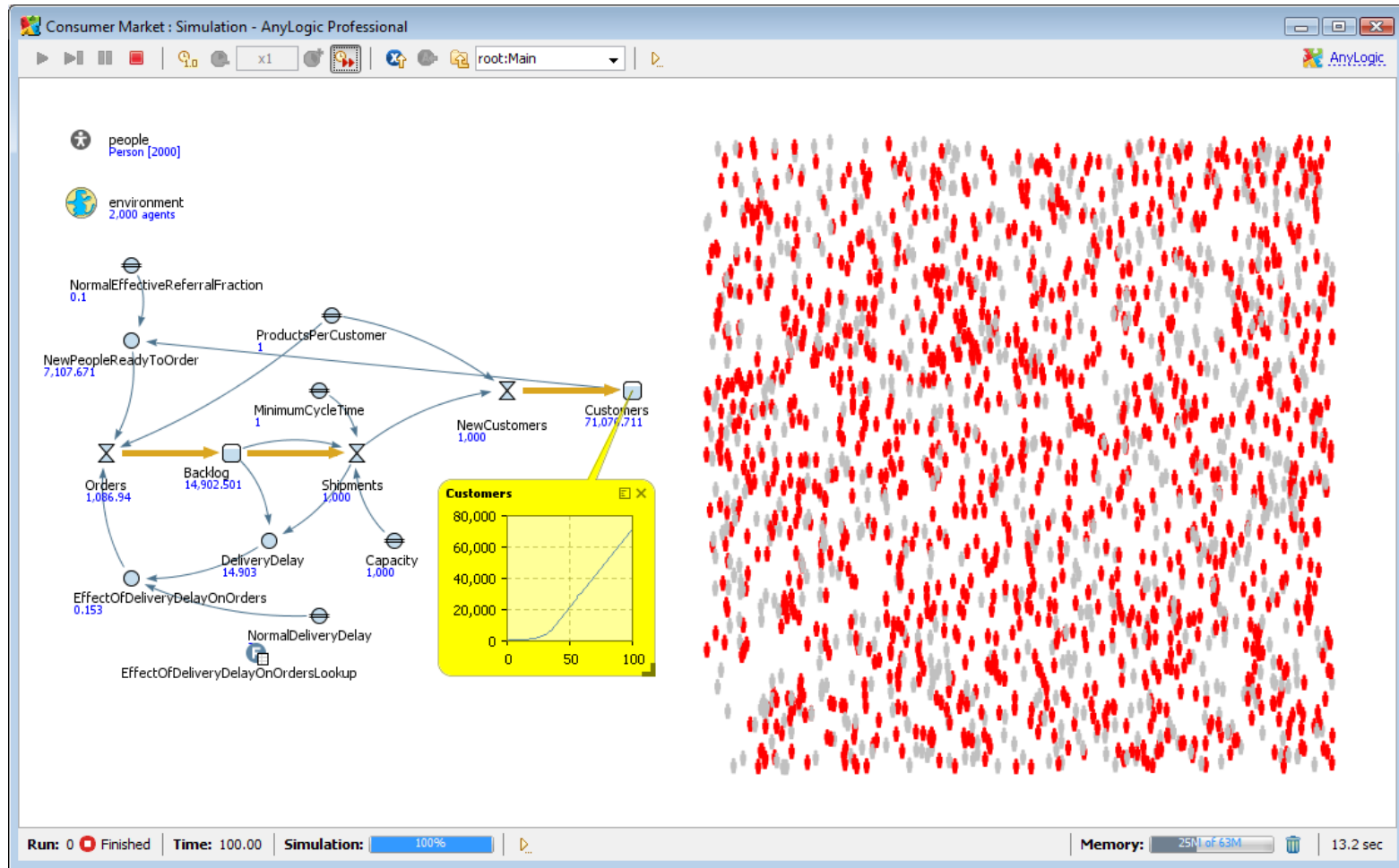
Step 4.1 Comments

We are about to start combining the imported system dynamics model with the agent based model we created in the beginning. There is a number of ways to make the models interact, but for simplicity we will just copy the whole stock-and-flow diagram into the Main object of Consumer Market project.

To copy select the whole diagram in Main of DELIV project, and choose Copy from the toolbar or from the context menu. Then switch to the Main of Consumer market and choose Paste. Place the pasted variables below the environment object and move the animation of consumer to the right (this will shift the space where the agents are animated so they are not drawn on to of the SD diagram).

The DELIV project is no longer needed and you can close it.

Step 4.2 Run the Consumer Market model

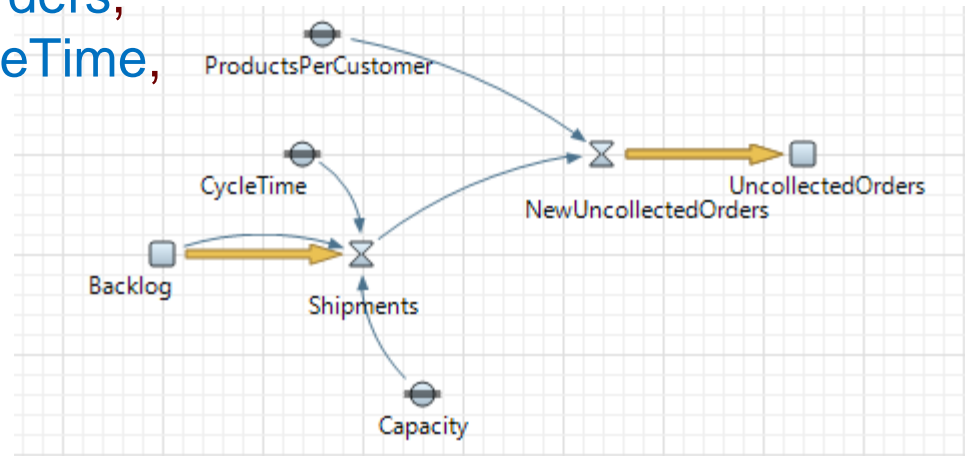


Step 4.2 Comments

Run the Consumer Market model. You will see that the agents (consumers) keep turning red, while the supply chain dynamics lives its own life: the two models are executed in the same space and time, but they do not interact yet.

Step 5.1 Modify the SD diagram

- In the SD model of product delivery we will delete all elements that relate to the market – we will link the AB model of market
- Leave the following variables: Backlog, Shipments, NewCustomers, Customers, MinimumCycleTime, ProductsPerCustomer, Capacity
- Rename (! Press Ctrl+Enter after editing each name !)
NewCustomers -> NewUncollectedOrders,
Customers-> UncollectedOrders,
MinimumCycleTime -> CycleTime,
- Set initial values of Backlog and UncollectedOrders to 0



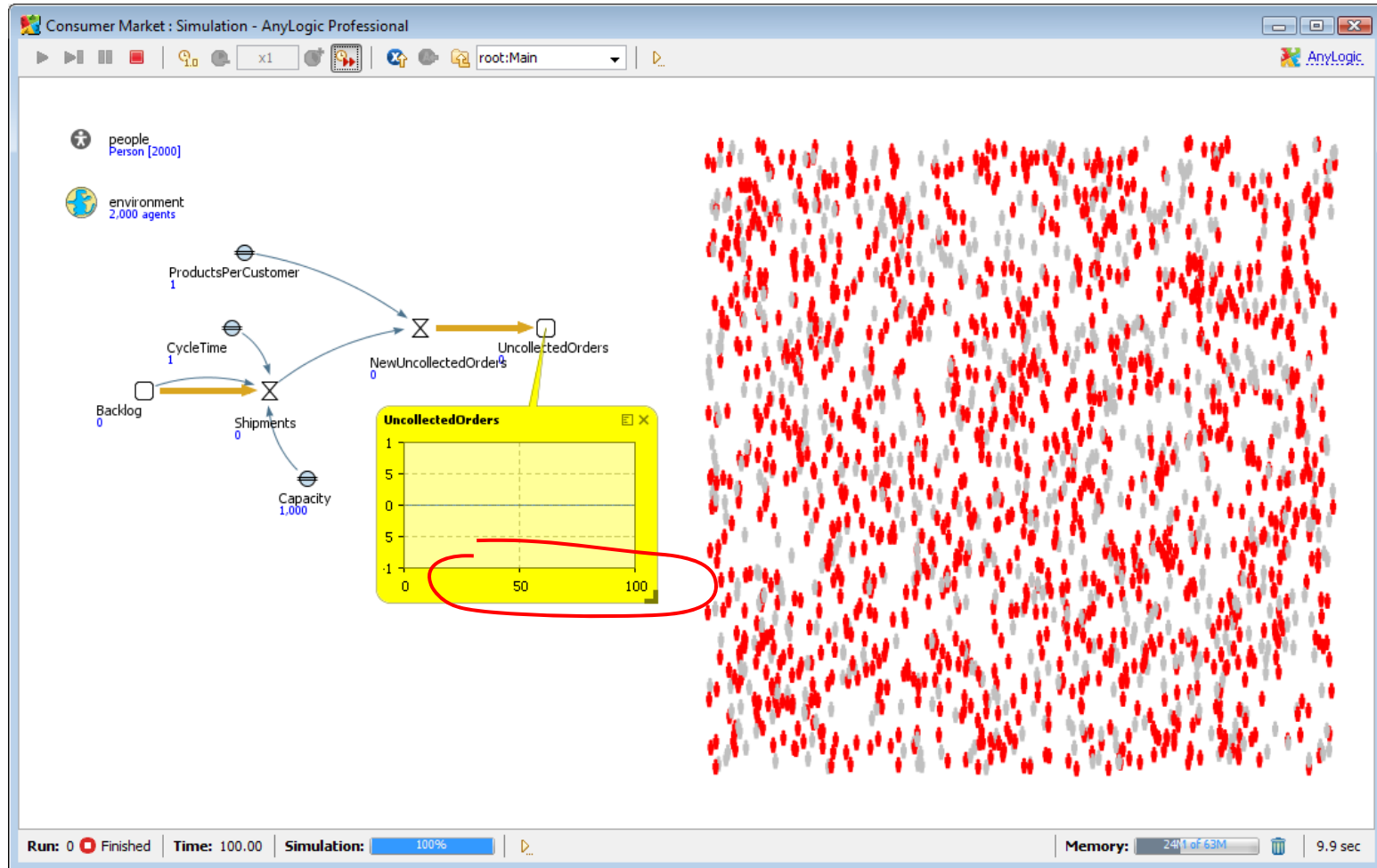
Step 5.1 Comments

Now we will modify the SD model of the supply chain to delete everything related to the market and leave everything related to the supply chain itself.

The stock customers will get a new meaning now: this will be a kind of “retailer stock” where the agents-consumers collect their orders, therefore a couple of renamings is needed. In AnyLogic when you rename an object and wish to keep all references to it, you need to press Ctrl+Enter to “refactor” the model, i.e. to update the references.

There will be no initial items in the order Backlog and in the “retailer stock”, so we will set their initial values to 0.

Step 5.2 Run the model

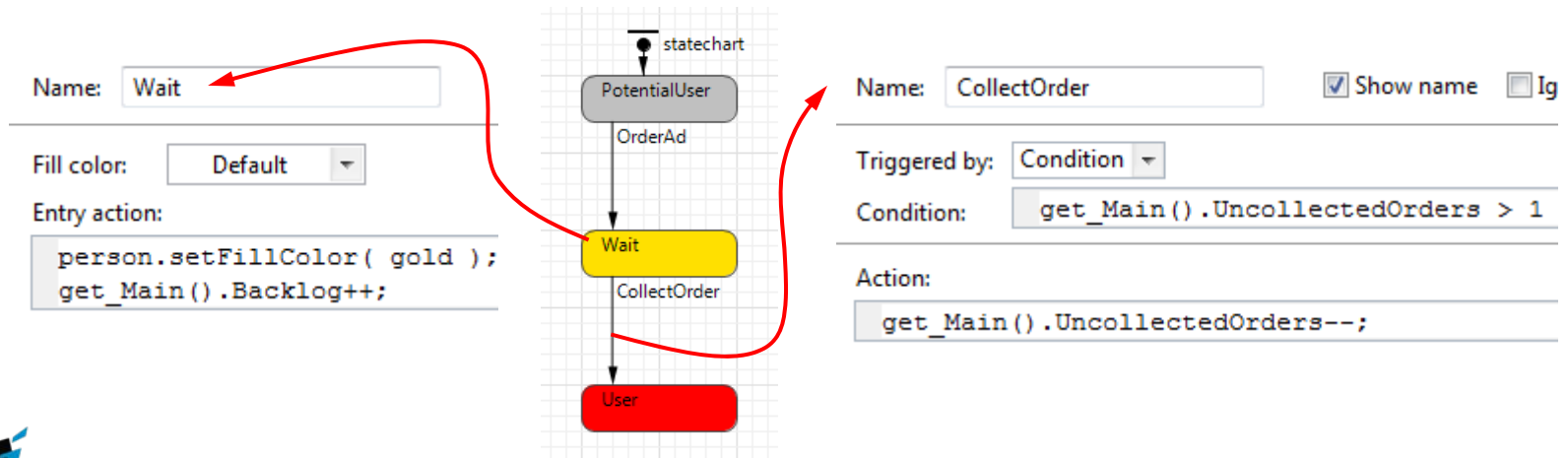


Step 5.2 Comments

We are now running the model just to make sure the SD diagram is still consistent after the changes. As you can expect, the supply chain delivers no products because there are no orders in the backlog – we have not yet linked the model to the AB consumer market.

Step 6.1 Link AB model to SD model

- In statechart add a state **Wait** between **PotentialUser** and **User**
- The **OrderAd** transition now should go to **Wait**, not to **User**
- Set the color of **Wait** to **gold** and specify this **Entry** action:
`person.setFillColor(gold);`
`get_Main().Backlog++;`
- Add a transition from **Wait** to **User**, name: **CollectOrder**,
condition: `get_Main().UncollectedOrders > 1`,
action: `get_Main().UncollectedOrders--;`

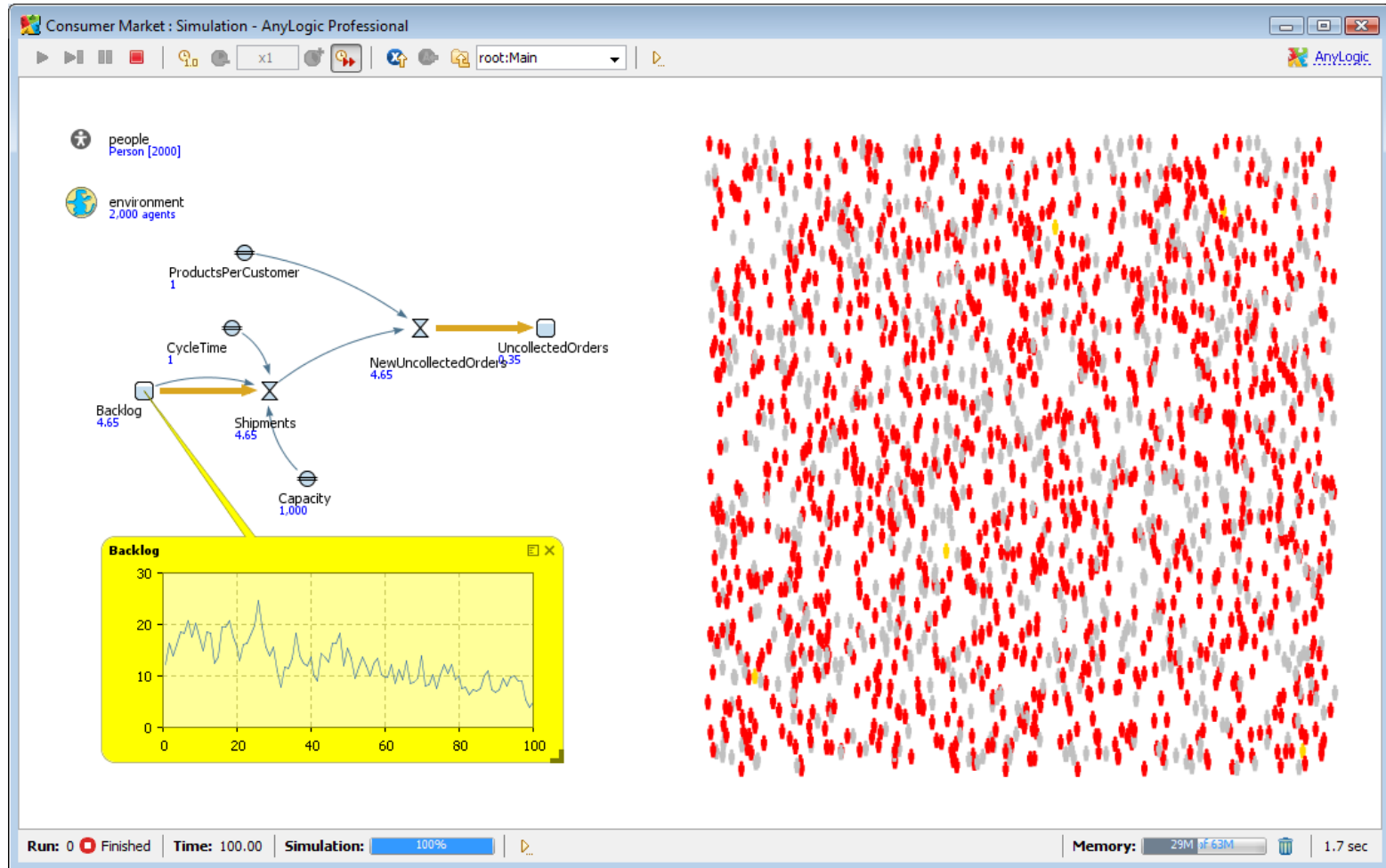


Step 6.1 Comments

In this step we are linking the two models. The idea is the following: we add a third state Wait to the statechart of Consumer reflecting the phase between the placing the order and collecting the delivered order. The transition OrderAd will now increment the value of the stock Backlog by 1 – the consumer places the order. This is the first place where the AB and SD model interact.

To proceed from the state Wait to the state User the consumer will need to collect the delivered order. This is modeled by the transition CollectOrder triggered by the condition “there is at least one item on the retailer’s stock”. The action of that transition is to remove that item from the stock – and this is the second point of SD-AB interaction. The semantics of the condition-triggered transition is that it waits on the condition, and when it is taken, it is taken “atomically”, i.e. no other action can be executed between the test of the condition and the action.

Step 6.2 Run the model



Step 6.2 Comments

You will see now how the supply chain modeled in SD delivers products to consumers modeled as agents. The supply chain works OK, there are very few waiting people (shown in gold color).

The order backlog dynamics (which you can see by clicking on the Backlog stock) is essentially stochastic as the value of the stock is controlled by a stochastic agent based model. The backlog size goes down as the model executes forward: the market gets closer to saturation, but does not reach it within the 100 days.

Step 7.1 Add interaction between agents

- In the statechart add an internal transition to state User: name: **Contact**, rate: **0.1**, action: **send("Buy it!", RANDOM);**
- Add a transition from **PotentialUser** to **Wait**: name: **OrderWOM**, triggered by message **"Buy it!"**
- In the Agent page of the **Person** properties in the field **On message received** write: **statechart.receiveMessage(msg);**

The image shows a statechart editor interface. On the left, a statechart diagram has three states: PotentialUser (grey), Wait (yellow), and User (red). Transitions are: PotentialUser to Wait (labeled OrderAd), PotentialUser to Wait (labeled OrderWOM), and Wait to User (labeled CollectOrder). Red arrows point from the OrderWOM transition to the configuration panel for 'OrderWOM', and from the 'Contact' transition in the User state to its configuration panel.

OrderWOM Configuration:
Name: OrderWOM Show name
Triggered by: Message
Message type: boolean int double String
Class name:
Fire transition: Unconditionally If message equals "Buy it!" If expression is true

Contact Configuration:
Name: Contact
Triggered by: Rate
Rate: 0.1
Action: send("Buy it!", RANDOM);

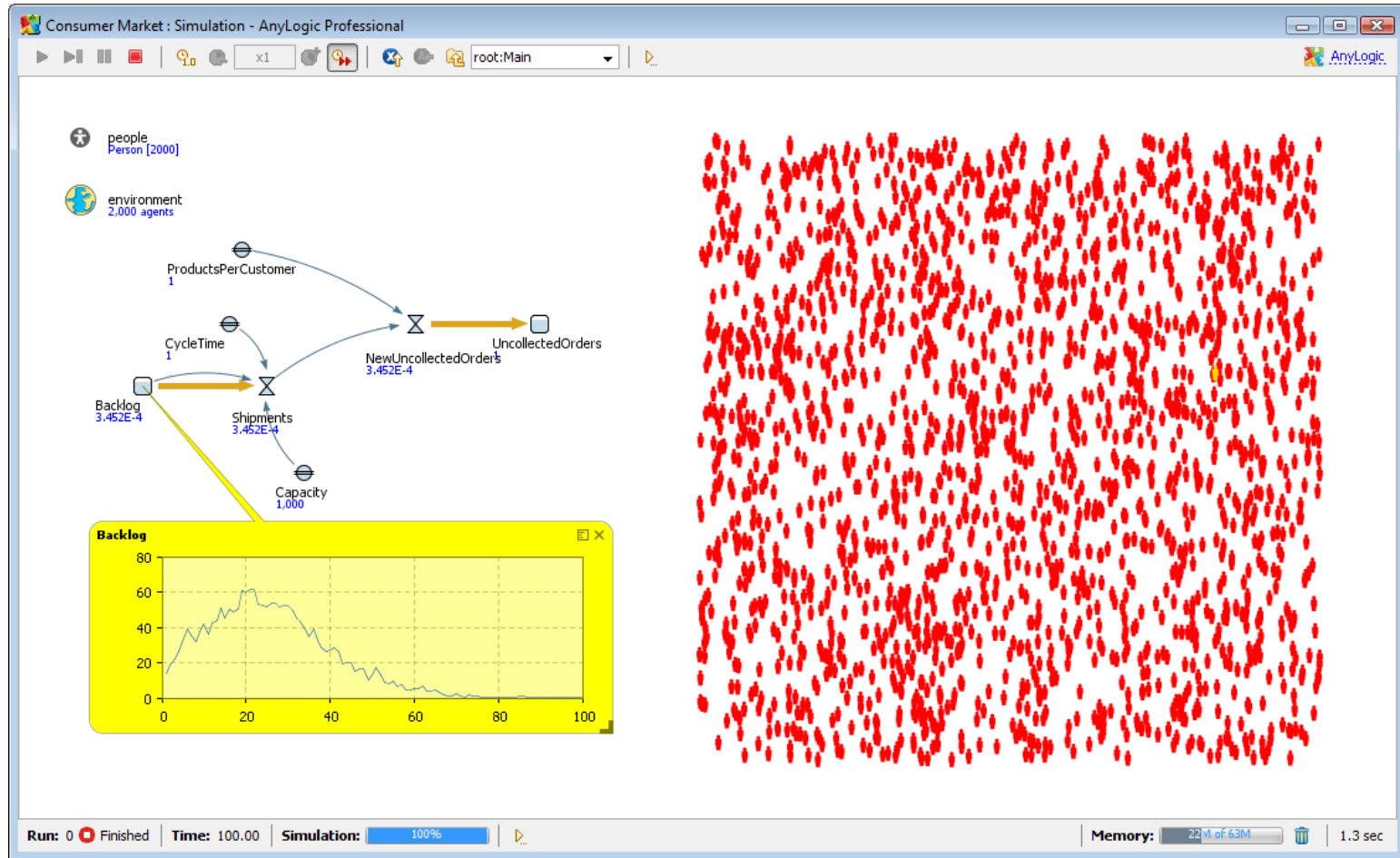
Person - Active Object Class Properties:
General: Space type: Continuous Discrete GIS
Advanced: Environment defines initial location
Agent: Initial coordinates: X: Y:
Movement parameters: Velocity: Rotation:
On arrival:
On message received: **statechart.receiveMessage(msg);**

Step 7.1 Comments

So far we were assuming the only thing that could cause a consumer to buy the product is advertizing. Therefore there was just one transition from PotentialUser to User (later on to Wait) triggered by a stochastic time delay, and the consumers were acting completely independently. In this step we are adding interaction between consumers that results in product diffusion.

The transition Contact is an internal transition of state User, which means it will perpetually be executed while the agent is in that state. The time between two subsequent executions of that transition is distributed exponentially with mean 10 days (rate = 0.1). When that transition is taken, a user selects a random other agent and sends him a message “Buy it!”. When the message arrives to the other agent, it is forwarded to its statechart. And if the other agent is in the state PotentialUser, it will react to the message by taking the transition OrderWOM (WOM – for Word Of Mouth)

Step 7.2 Run the model



Step 7.2 Comments

The diffusion of the product now goes much faster because of the word of mouth “positive feedback loop” in agent based part. The market gets completely saturated in about 60 days. The history of order backlog has a peak at approx. day 20 and then goes down to zero.

Step 8.1 Add statistics to AB part

- In the **Main** object go to the properties of collection **people** and on the **Statistics** page create three statistics items:

The screenshot shows the 'people - Person' configuration window in the IDE. The 'Statistics' tab is selected in the left sidebar. Three statistics items are configured:

- Name:** NUser
Type: Count Sum Average Min Max
Expression: (empty)
Condition: `item.statechart.isStateActive (Person.User);`
- Name:** NWait
Type: Count Sum Average Min Max
Expression: (empty)
Condition: `item.statechart.isStateActive (Person.Wait);`
- Name:** NPotentialUser
Type: Count Sum Average Min Max
Expression: (empty)
Condition: `item.statechart.isStateActive (Person.PotentialUser);`

At the bottom, there is a button labeled '+ Add statistics'.

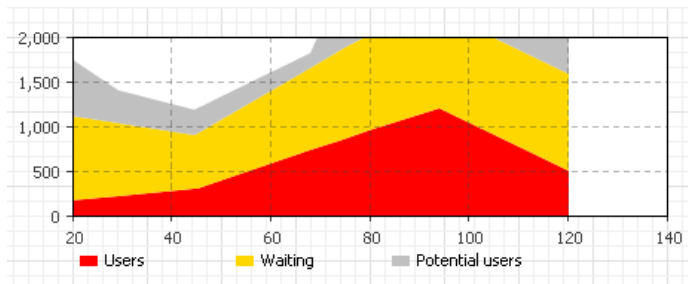
Step 8.1 Comments

Let us visualize the dynamics of sales / product diffusion. While for system dynamics variables the histories and time charts are built by default, for the disaggregated agent based part of the model we need to specify what kind of statistics we would like to collect. The collection of agents (people object in Main) is capable of collecting several kinds of statistics: count the number of agents satisfying a certain condition, calculate the sum of individual variables, etc.

We are interested in how many consumers are potential users, waiting for the product delivery, and use the product at each point in time. Therefore we need three statistics items of type “count”. The condition for each of them is the state of the agent. In the field for condition you should use “item” to identify the agent that is being tested. As states are defined within the Person active object, you need to use prefix “Person.” before the state name.

Step 8.2 Add time stack chart

- In the **Main** object add a **Time stack chart** below the SD diagram
- Add three data items to the chart as shown in this slide
- Specify vertical scale **Fixed to 2000**



Name: Show

Value Data set Title:
Value:
Color:

Value Data set Title:
Value:
Color:

Value Data set Title:
Value:
Color:

Time Window:

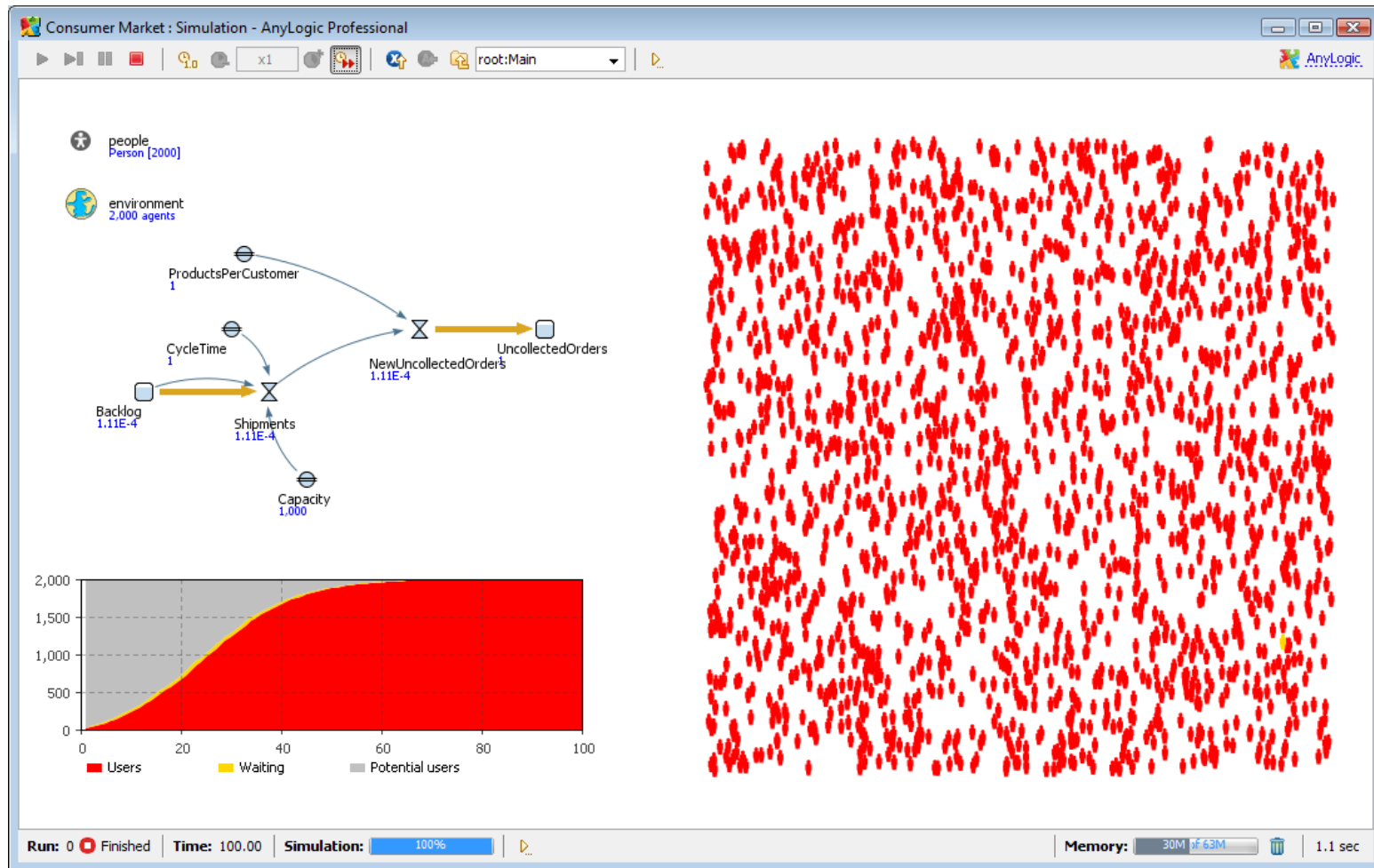
Vertical scale: to:

Step 8.2 Comments

Now that we know how to count the agents in particular states, we can create a time stack chart to show the dynamics of our market. The time stack chart is one of many AnyLogic data visualization objects that can be found in the Analysis palette. We choose time stack chart as the total number of agents will stay the same (2000) but fractions will change over time.

We need to add a data item for each fraction. For example, for the consumers waiting for the product the data item will have the value `people.NWait()`, where `people` is the collection of agents, and `NWait` is the name of the statistics. The stack chart will evaluate the values of all data items with a given frequency and display their histories.

Step 8.3 Run the model



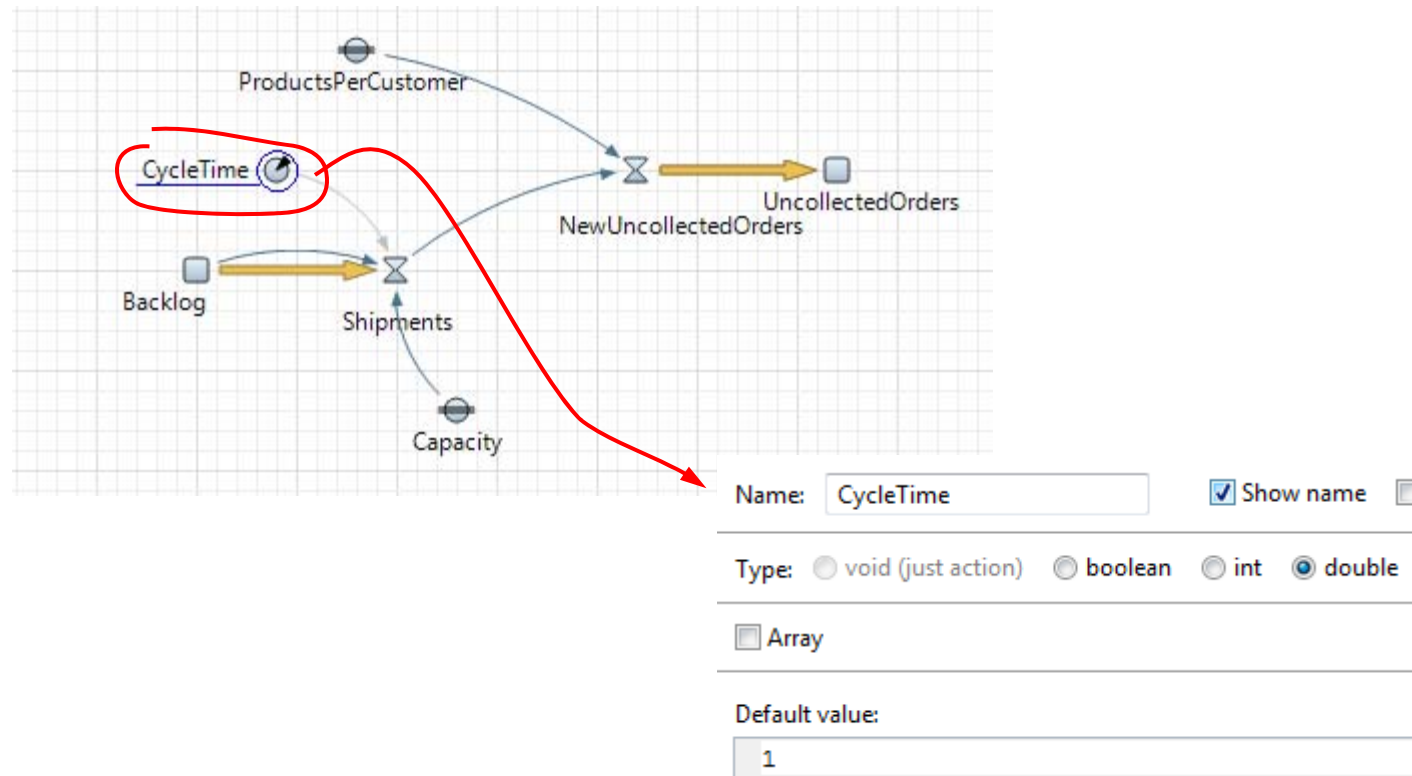
Step 8.3 Comments

The chart shows the classical S-shaped diffusion dynamics. You can see the thin yellow layer between the users and potential users: these are consumers waiting for the product delivery.

In any AnyLogic chart you can select a data item by clicking at the legend, and you can copy the numeric data from the chart to the clipboard by choosing Copy selected or Copy all in the context menu of the chart legend. The data gets copied in tab-separated format and can be pasted directly into a spreadsheet or a text file.

Step 9.1 Create a parameter

- In the SD diagram delete the variable **CycleTime** and add a parameter **CycleTime** (from the General palette), default value: 1



Step 9.1 Comments

Now let us create a simple experiment framework around our model. We will explore how the cycle time in the supply chain affects the sales dynamics, in particular, the number of consumers waiting for the product delivery.

The CycleTime is currently a constant, and we will change it to be a parameter. The difference is that parameters are exposed to the interface of the active object and can be varied from outside.

Note that the dependency arrow from CycleTime to Shipments is restored automatically as you finish typing the name “CycleTime”. Here AnyLogic differs from Vensim: the dependency from A to B is drawn automatically if the name “A” is found in the formula for B. You can arrange the shape of the dependency arc by dragging it.

Step 9.2 Create a slider in experiment

- Open the editor of the **Simulation** experiment (dbl+click in the tree)
- Add a slider from Controls palette: min: **1**, max: **25**, default: **1**

The screenshot displays the XJ Technologies simulation editor interface. On the left, a project tree shows the hierarchy: Consumer Market > Main > Simulation: Main. A red arrow points from the 'Simulation: Main' item in the tree to the main editor window. The main editor window is titled 'Simulation' and contains a grid with the text 'Consumer Market' and 'Experiment setup page'. Below this text is a button labeled 'Run the model and switch to Main view'. A slider control is added to the grid, highlighted with a red oval. A red arrow points from the slider to its configuration panel on the right. The configuration panel includes the following settings:

- Name: slider
- Orientation: Horizontal Vertical
- Link to: []
- Minimum value: 1
- Maximum value: 25
- Default value: 1
- Enabled:

Step 9.2 Comments

You probably have noticed that when the model starts, it first shows the white screen with the model name and “Run...” button. This is the front end of “experiment” – a top-level item in the model project that controls the mode of model execution. There can be multiple experiments in one project, for example a simulation experiment, an optimization experiment, etc. In our case there is just one experiment “Simulation” and all it does is launching one simulation run.

You can edit the experiment front end; to do it you must double-click the item “Simulation” in the project tree. We will add a slider just below the “Run...” button. The slider will then be linked to the parameter CycleTime of the Main object of the model. In this step we will set up the minimum, maximum and the default (initial) value of the slider.

Step 9.3 Link the slider to the parameter

- On the General page of the experiment properties set the value of **CycleTime** to: `slider.getValue()`
- Now you will be able to set the value of that parameter before each simulation run

Simulation - Simulation Experiment

General

Name: Main active object class (root):

Random number generation:

Random seed (unique simulation runs)

Fixed seed (reproducible simulation runs) Seed value:

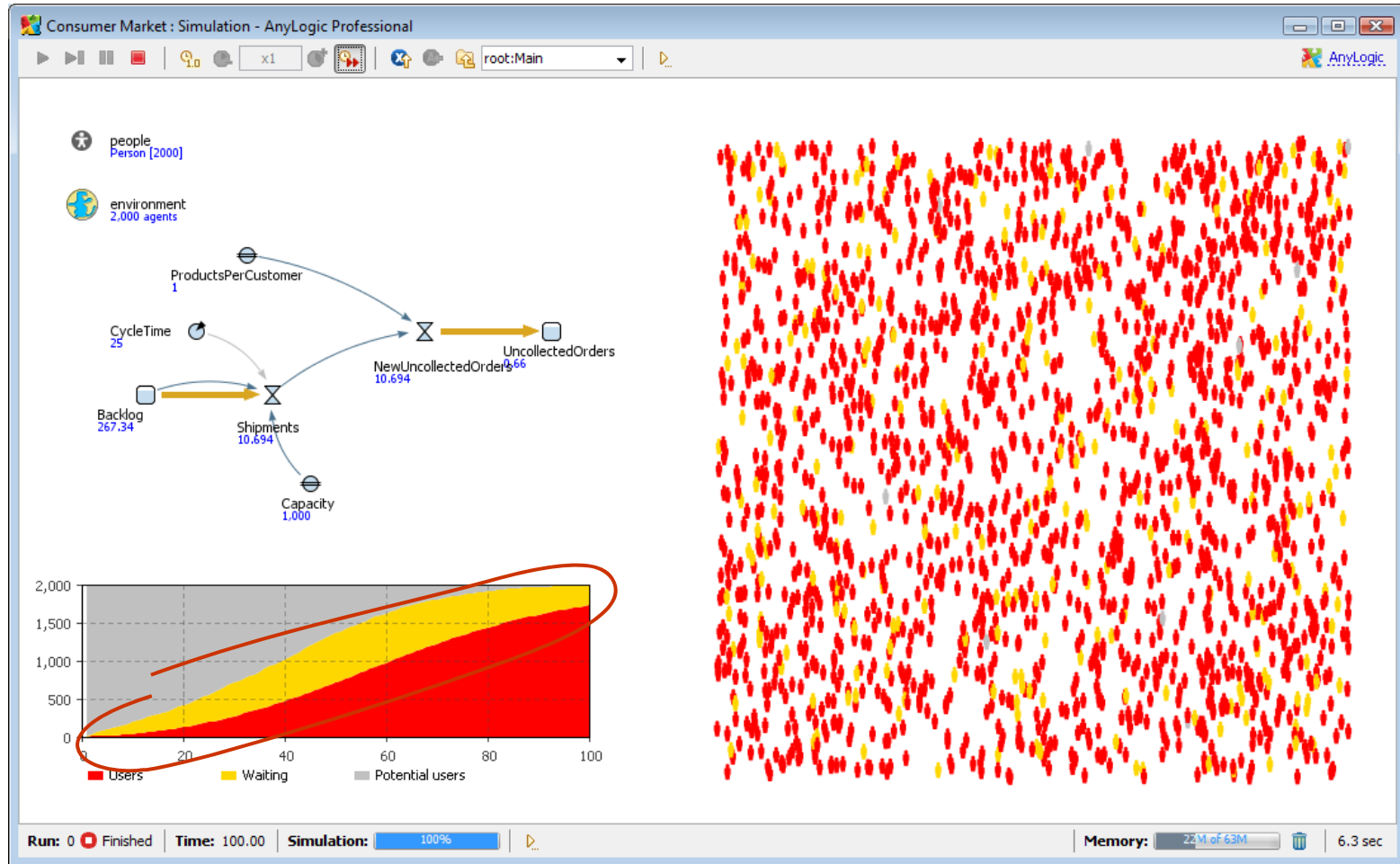
CycleTime

Step 9.3 Comments

In the General page of the experiment properties (to view them you can click on the experiment item in the tree or click an empty spot of the experiment editor) you will see the name of the top-level model object (Main in our case) and the list of its parameters. So far there is only one – CycleTime.

The expression “slider.getValue()” obviously refers to the slider object we have created and calls its method getValue() to retrieve the current value of the slider (use Ctrl+Space that invokes completion feature to avoid typing the full name). When the simulation starts, the model parameters are set to the expressions specified in those fields.

Step 9.4 Run the model



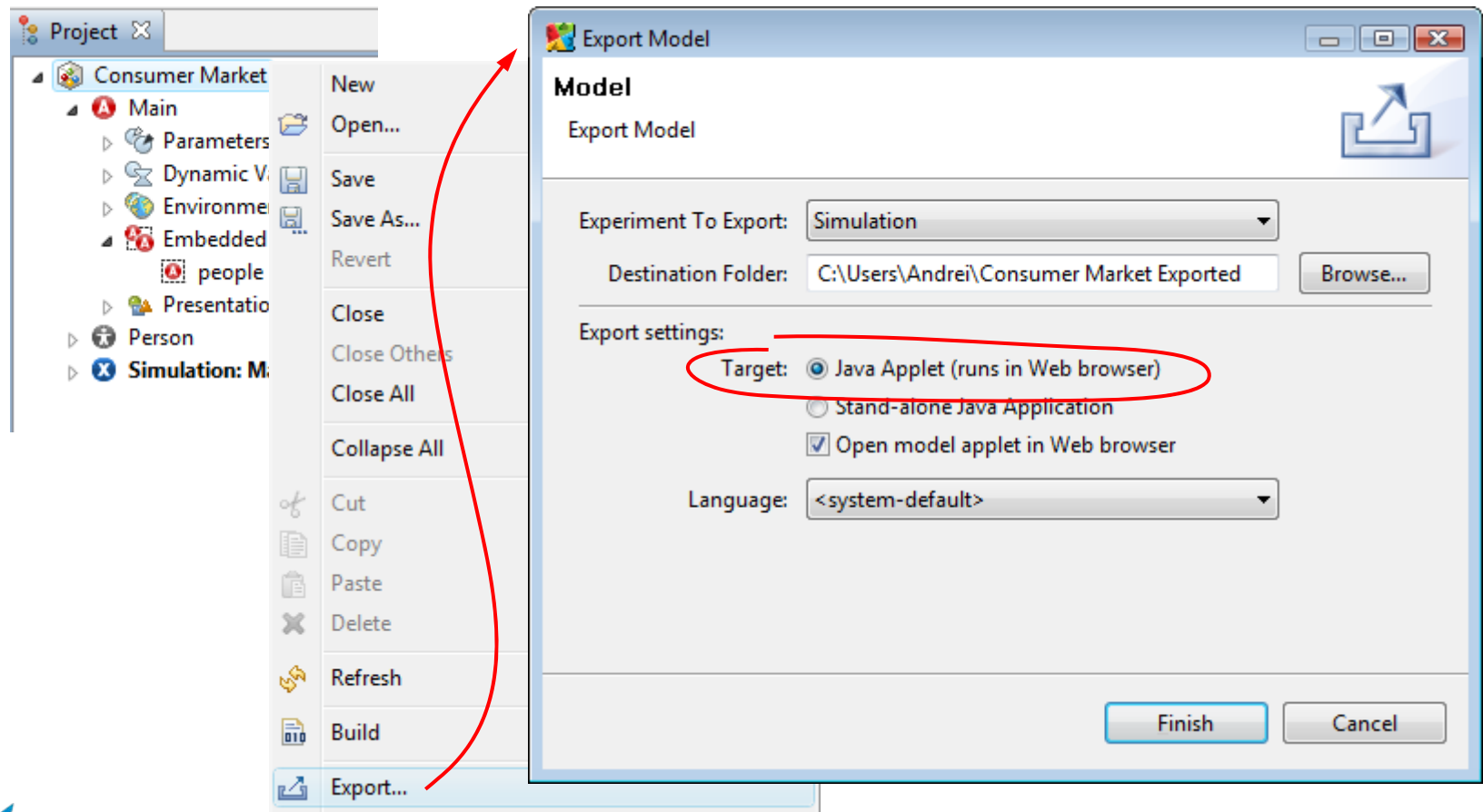
Step 9.4 Comments

You will see the slider on the first screen after the model starts. Try different parameter values and see how the Cycle Time affects the efficiency of the Supply Chain. Obviously, you need to move the slider before pressing the “Run...” button

The screenshot on the previous slide corresponds to the completed simulation with CycleTime = 25. As you can see, a significant number of orders are still not fulfilled.

Step 10.1 Export the model as applet

- From the context menu of the model **Consumer Market** choose **Export**. Then choose **Target: Java applet** and click **Finish**



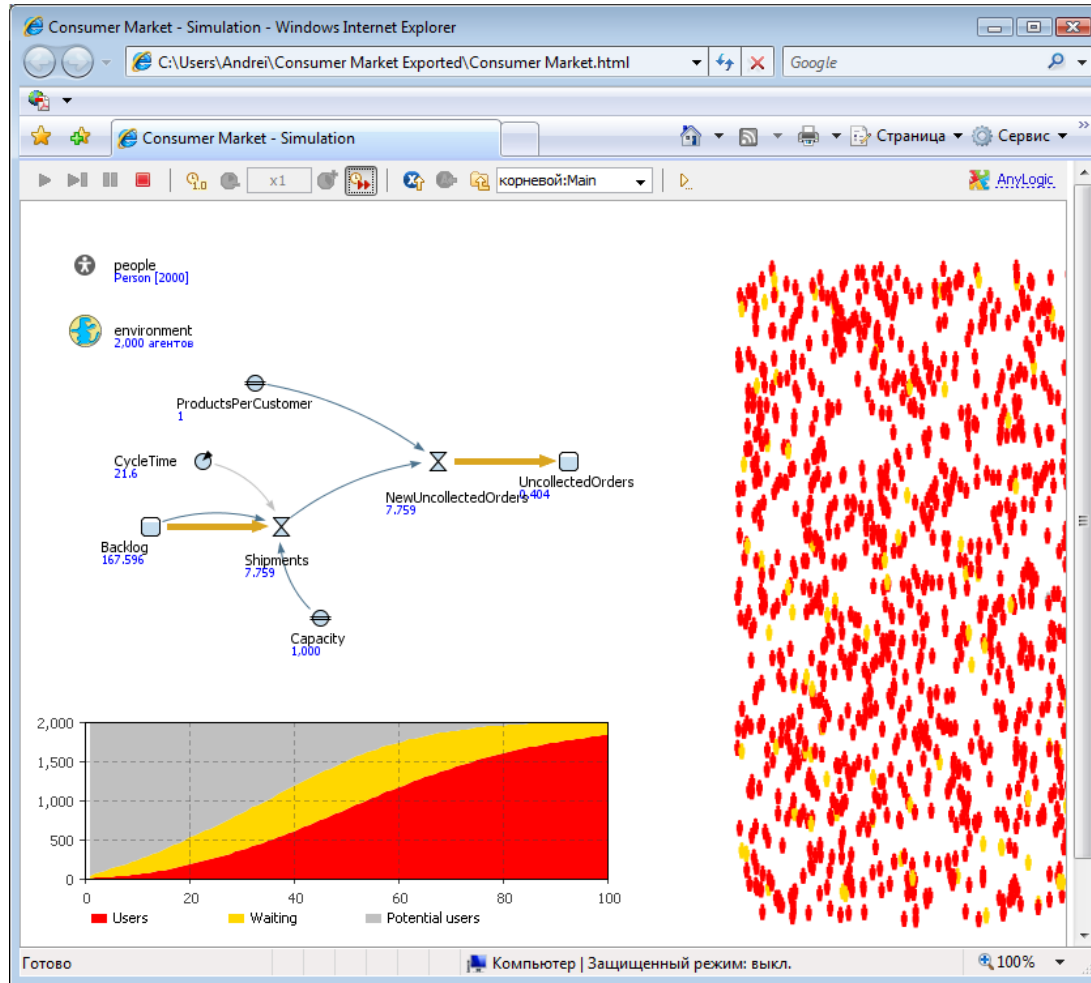
Step 10.1 Comments

So far any time we were running the model, it was running under control of AnyLogic model development environment. However, this is not the only mode a model can run. As long as Anylogic simulation engine is pure Java, the model itself is completely standalone and independent application that you can launch in a number of modes.

One of them is Java applet mode. The model runs within a web browser, and the end user does not need to install any software on his machine.

When you choose Export | Target: Java applet AnyLogic generates about five files that you can easily publish on your website (to make the model accessible to the remote users), or send someone by email. To test how the model runs as an applet, AnyLogic would open the applet in your default browser.

Step 10.2 Run the model in a browser



Step 10.2 Comments

The model now runs within the browser, and you can even close AnyLogic model development environment to make sure it is not needed to run the model. The model is interactive: all control elements are accessible from the applet UI.

Applets are the easiest way to deliver the model to your client or colleague who does not have AnyLogic installed. However, applets are run in so called sandbox: they are not allowed to access the data on the computer, such as spreadsheets, databases, or files. If your model needs external input or output data to run, you may consider exporting it as Java application instead of applet.