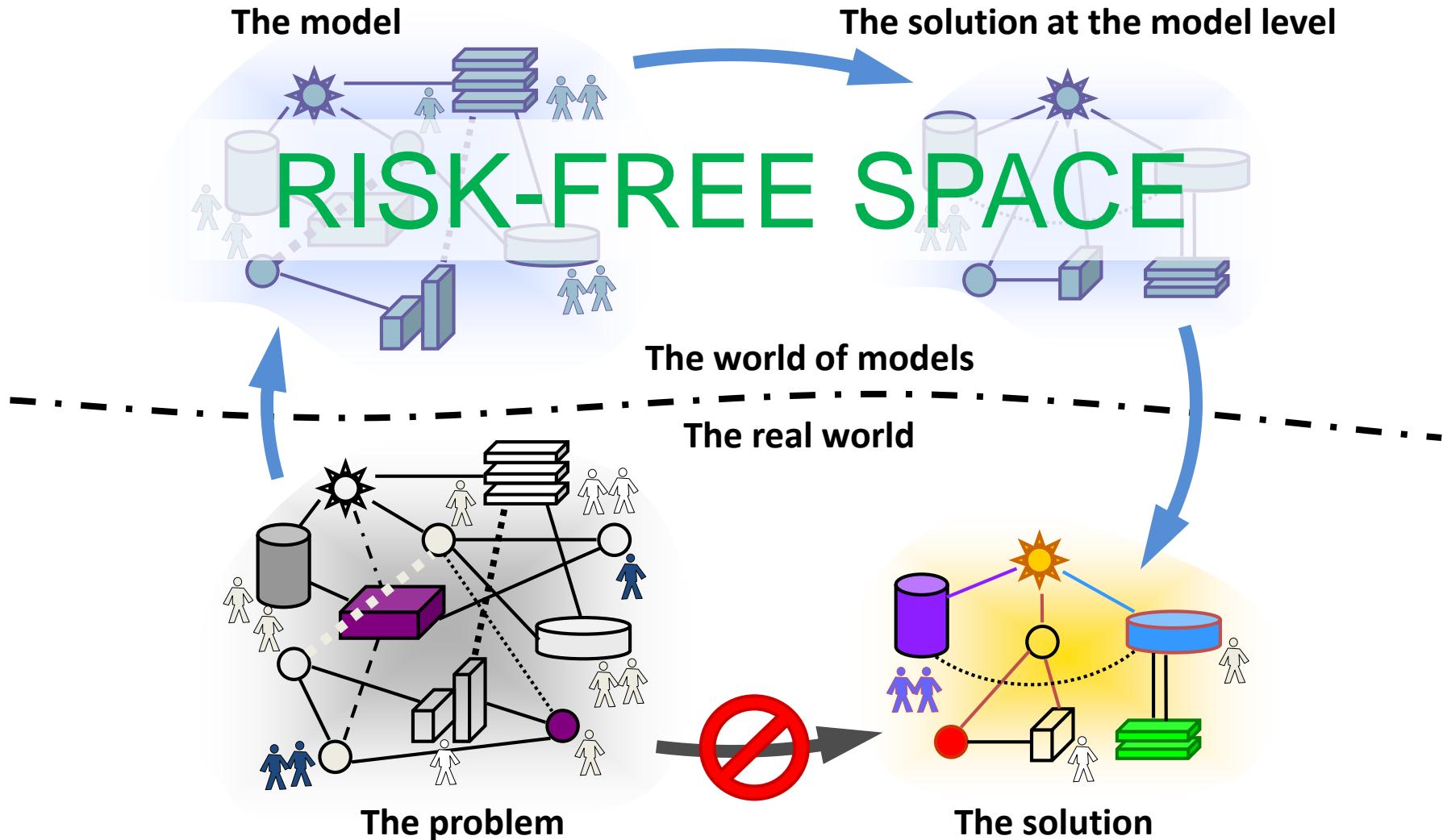


Multi-Method Simulation Modeling with AnyLogic

This presentation is a part of
AnyLogic Standard Training Program

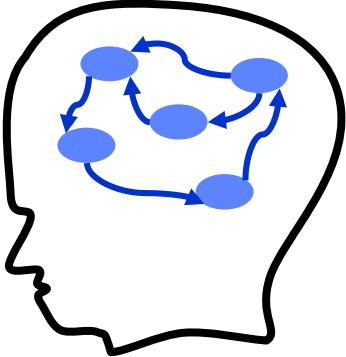


Modeling

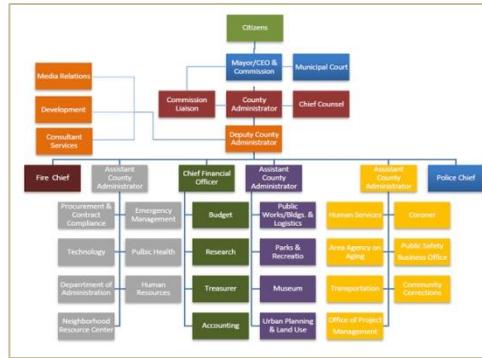


Types of models

Mental models



Boxes connected with lines



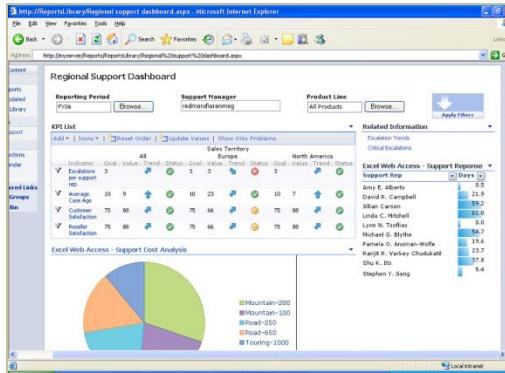
Physical models



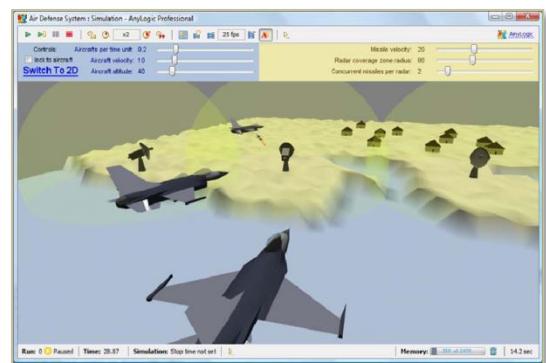
Formulas on a sheet of paper

$$\begin{aligned} \vec{F} &= m\vec{a} \quad \vec{p} = m\vec{v} \quad KE = \frac{1}{2}mv^2 = \frac{p^2}{2m} \quad W_{ext} = \Delta(KE) = KE_f - KE_i \quad A_{spikes/radical} = 4\pi r^2 \\ R &= k \frac{q_1 q_2}{r^2} = \frac{1}{4\pi\epsilon_0} \frac{q_1 q_2}{r^2} \quad \epsilon_s = 8.85 \cdot 10^{-12} \left[\frac{C^2}{N \cdot m^2} \right] \quad k = 8.99 \cdot 10^9 \left[\frac{Nm^2}{C^2} \right] \quad \epsilon_0 = 8.85 \cdot 10^{-12} \left[\frac{C^2}{N \cdot m^2} \right] \quad A_{disk} = \pi r^2 \\ F &= k \frac{q_1 q_2}{r^2} = \frac{1}{4\pi\epsilon_0} \frac{q_1 q_2}{r^2} \quad V = k \frac{q}{r} = \frac{1}{4\pi\epsilon_0} \frac{q}{r} \quad V = \frac{U}{q} \quad \sim e^{-\frac{1}{RC}} \\ E &= \frac{F}{q} \quad E = k \frac{q}{r^2} = \frac{1}{4\pi\epsilon_0} \frac{q}{r^2} \quad V = k \frac{q}{r} = \frac{1}{4\pi\epsilon_0} \frac{q}{r} \quad V = \frac{U}{q} \quad \sim e^{-\frac{1}{RC}} \\ \sum E_\perp \Delta A &= \frac{q}{\epsilon_0} \quad Q = VC \quad C = \frac{\Delta \epsilon_0}{d} \quad \sigma = \frac{Q}{A} \quad V = Ed \quad E = \frac{\sigma}{\epsilon_0} \quad U = \frac{QV}{2} = \frac{CV^2}{2} = \frac{Q^2}{2C} \\ \sum I_j &= 0 \quad \sum V_j = 0 \quad V = IR \quad P = IV = I^2 R = \frac{V^2}{R} \quad R_{tot} = R_1 + R_2 = \frac{1}{C_{tot}} = \frac{1}{C_1} + \frac{1}{C_2} \\ F &= q \times B_\perp = q \times v_\perp B = q \times B \sin(\theta) \quad B = \frac{\mu_0 I}{2\pi r} \quad \mu_0 = 4\pi(10)^{-7} Tm/A \\ F &= ILB_\perp = I_\perp LB = ILB \sin(\theta) \quad \sum B_\parallel \Delta l = \mu_0 I_\perp \end{aligned}$$

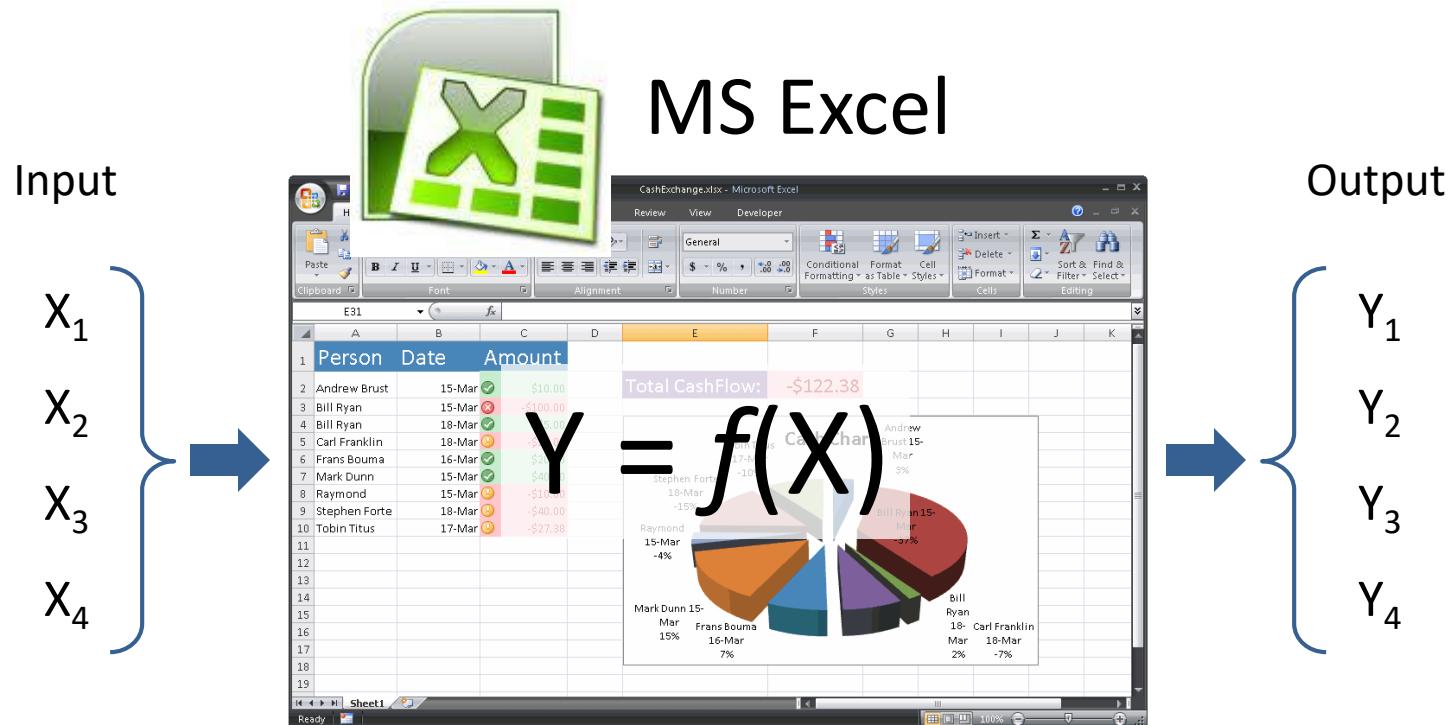
Excel spreadsheets



Simulation models



The most popular modeling tool is:



Analytical solution
(formulas and scripts)



However...

- You can find an analytical solution if:
 - The number of parameters is ‘manageable’
 - Behavior is linear
 - Dependencies are clear, easy to build a mental model
- But what if:
 - Too many parameters
 - Non-linear, non-obvious influences
 - Time and causal dependencies
 - Counter-intuitive behavior
 - Uncertainty (stochastic system)



Example: Bank

- A simplistic case:
 - On average 10 clients per hour
 - Only one teller at the counter
 - Mean service time is 5 minutes
- We want to find out:
 - Mean waiting time in the queue
 - [Other metrics can be derived from that one]



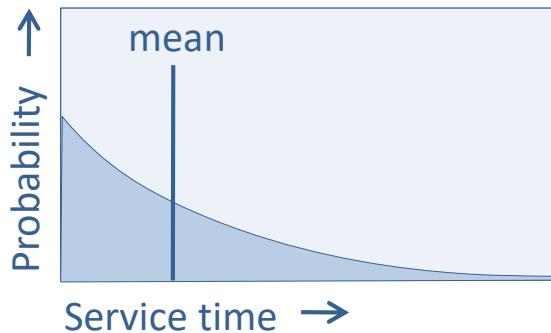
- It'll take you a few seconds to find the analytical solution:

Mean waiting time* $W = \frac{\lambda b^2}{1 - \lambda b}$, where λ - arrival rate
 b - mean service time

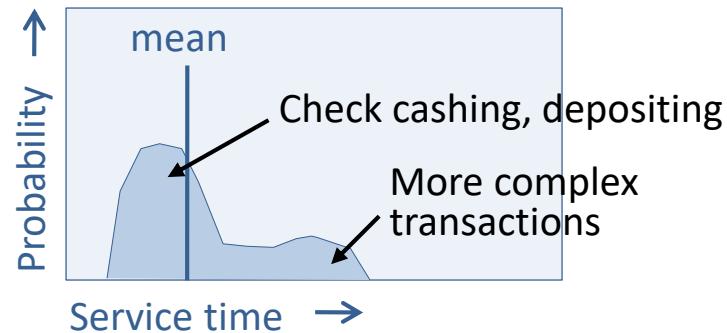
* This holds only for a Poisson stream of clients (independent arrivals with constant rate) and exponentially distributed service time.

Bank. Assumptions of the analytic approach

- What do these assumptions mean?
 - Independent arrivals of clients – this should be an OK assumption for the bank
 - Exponentially distributed service time:



This is far from reality. The distribution is more likely of this shape:



- Then the Internet search will suggest another formula:

$$w = \frac{\lambda b^2 (1 + C_b^2)}{2(1 - \lambda b)}, \text{ where } C_b \text{ - coefficient of variation of service time}$$



Bank. What if a bit less simplistic case?

- Let there be several (K) tellers
 - This is so-called “multi-server queue model”. The analytic solution*:

$$W = \frac{P_b}{K(1-\rho)}, \text{ where } \rho = \frac{\lambda b}{K} \quad \begin{matrix} \text{- system utilization,} \\ \text{- probability of all tellers being busy} \end{matrix}$$
$$P = \frac{(K\rho)^K}{K!(1-\rho)} P_0, \text{ where } P_0 = \left[\frac{(K\rho)^K}{K!(1-\rho)} + \sum_{i=0}^{K-1} \frac{(K\rho)^i}{i!} \right]^{-1} \quad \begin{matrix} \text{- probability of "no clients in the bank"} \end{matrix}$$

* This, however, is valid only for Poisson stream of clients and exponentially distributed service time.

- And what if service time has a different distribution?
 - Even for such a simple system **there is no analytic solution**



Bank example. Summary

- In the real bank the process is far more complex:
 - Some transactions can be done only by some particular employees
 - The client can be redirected to other employees
 - The tellers may share resources, such as a printer or a copier
 - Different employees may have different skills and performance
 - Etc.
- The analytic solution probably does not exist
 - Even if it exists, who will find it for you?
 - Almost any change in the process makes the previous analytic solution void
- The only analysis method for such systems that has foreseeable complexity and guarantees the result is:
simulation modeling



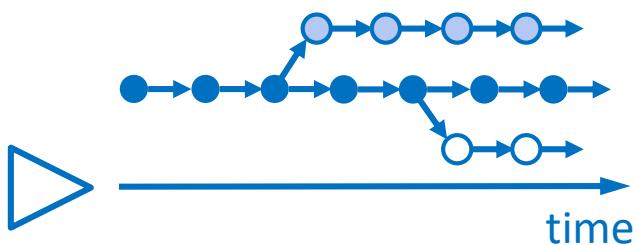
Simulation modeling

Identify input parameters (decision variables)



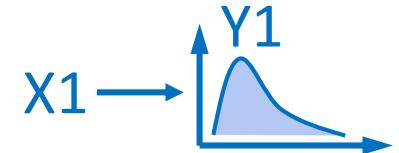
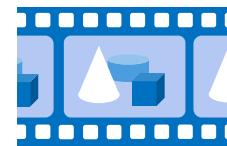
Identify outputs (key metrics, KPIs)

Build the model - describe the system dynamic behavior

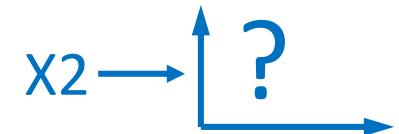


Run the model – obtain a trajectory of the system state in time

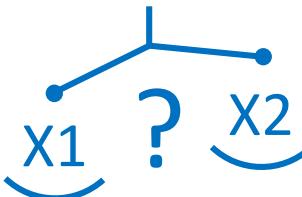
Measure outputs as the model runs



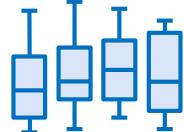
Animate the system behavior



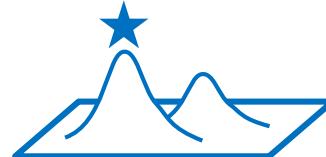
Perform what-if experiments



Compare scenarios



Measure risk



Optimize

- Dynamic simulation enables much more detailed analysis and can solve problems that spreadsheet-based or LP-based analytics can't



Application areas

High abstraction level

[minimum details
macro level
strategic level]

Medium abstraction
level

[medium details
meso level
tactical level]

Low abstraction level

[maximum details
Micro level
Operational level]

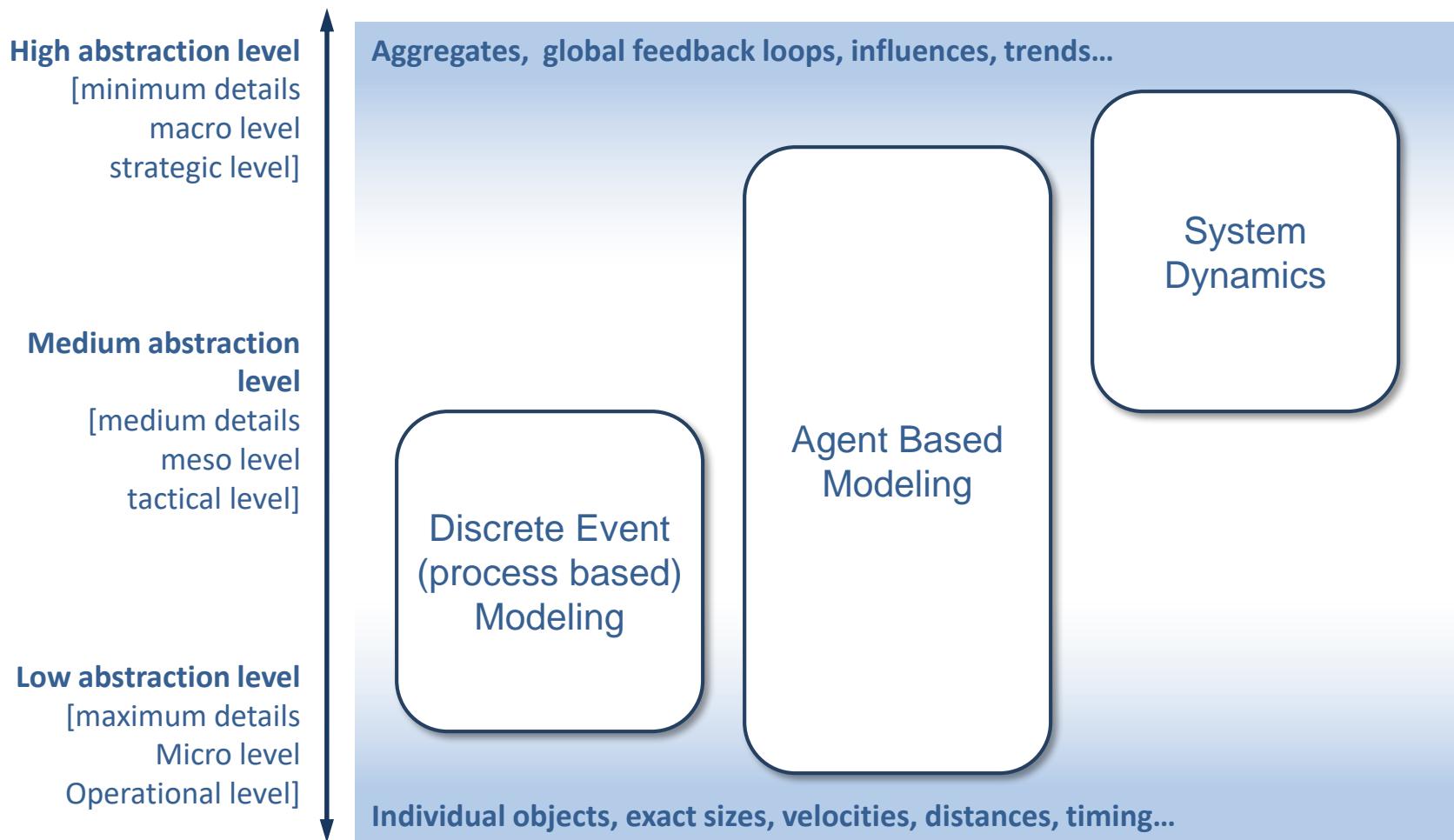
Aggregates, global feedback loops, influences, trends...

- Market and competition
- Project and product management
- HR dynamics
- Energy supply networks
- Healthcare
- Manufacturing
- Battlefield, command and control
- Computer hardware
- Social systems
- Ecosystems
- Health economics
- Asset management
- Supply chains
- Transportation
- Service systems
- Warehouse logistics
- Pedestrian dynamics
- Physical control systems

Individual objects, exact sizes, velocities, distances, timing...

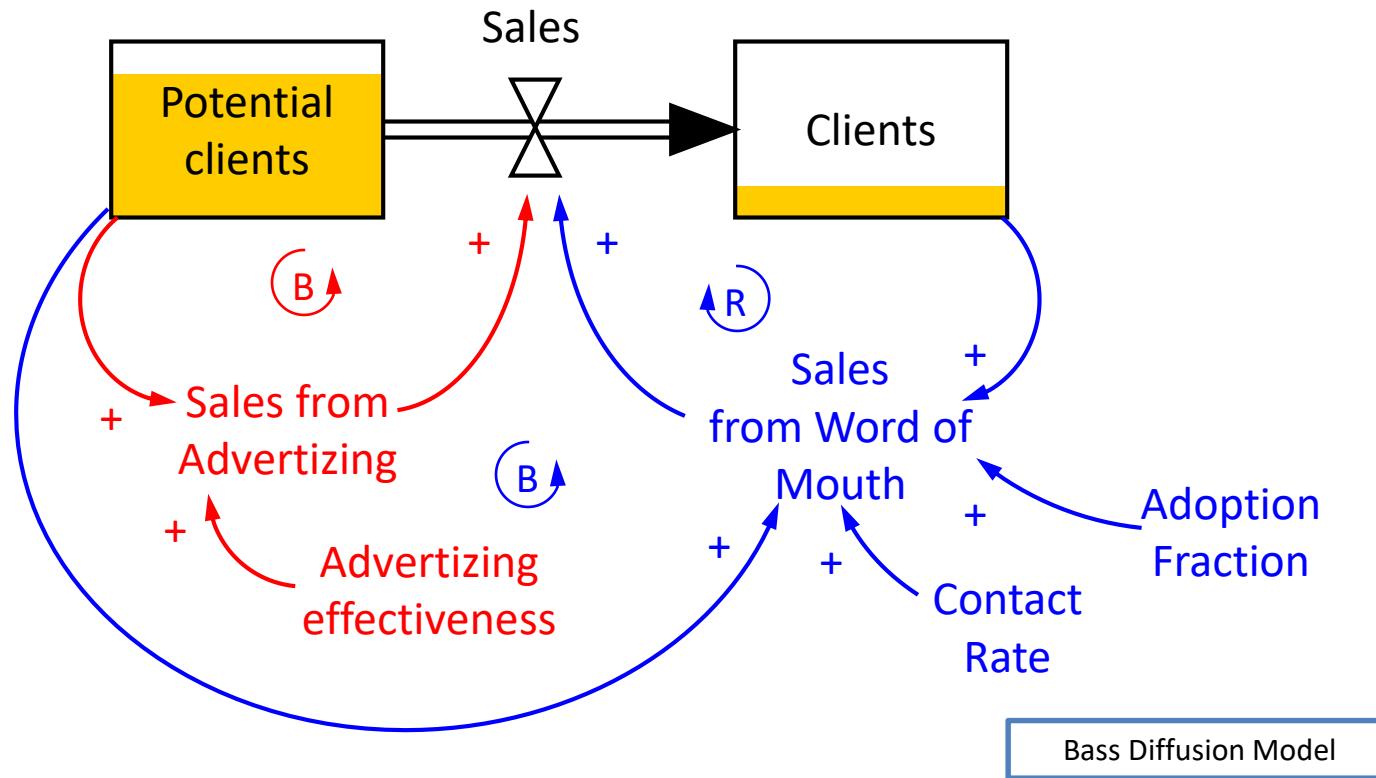


Methods in simulation modeling



System Dynamics Jay Forrester '50s

- Stocks, flows
 - Interacting feedback loops



System Dynamics Jay Forrester '50s

- Stocks, flows
 - Interacting feedback loops

The equivalent mathematical model:

$$d(\text{ Potential clients })/dt = - \text{Sales}$$

$$d(\text{ Clients })/dt = \text{Sales}$$

$$\text{Sales} = \text{Sales from Advertising} + \text{Sales from Word of Mouth}$$

$$\text{Sales from Advertising} = \text{Potential clients} * \text{Advertising effectiveness}$$

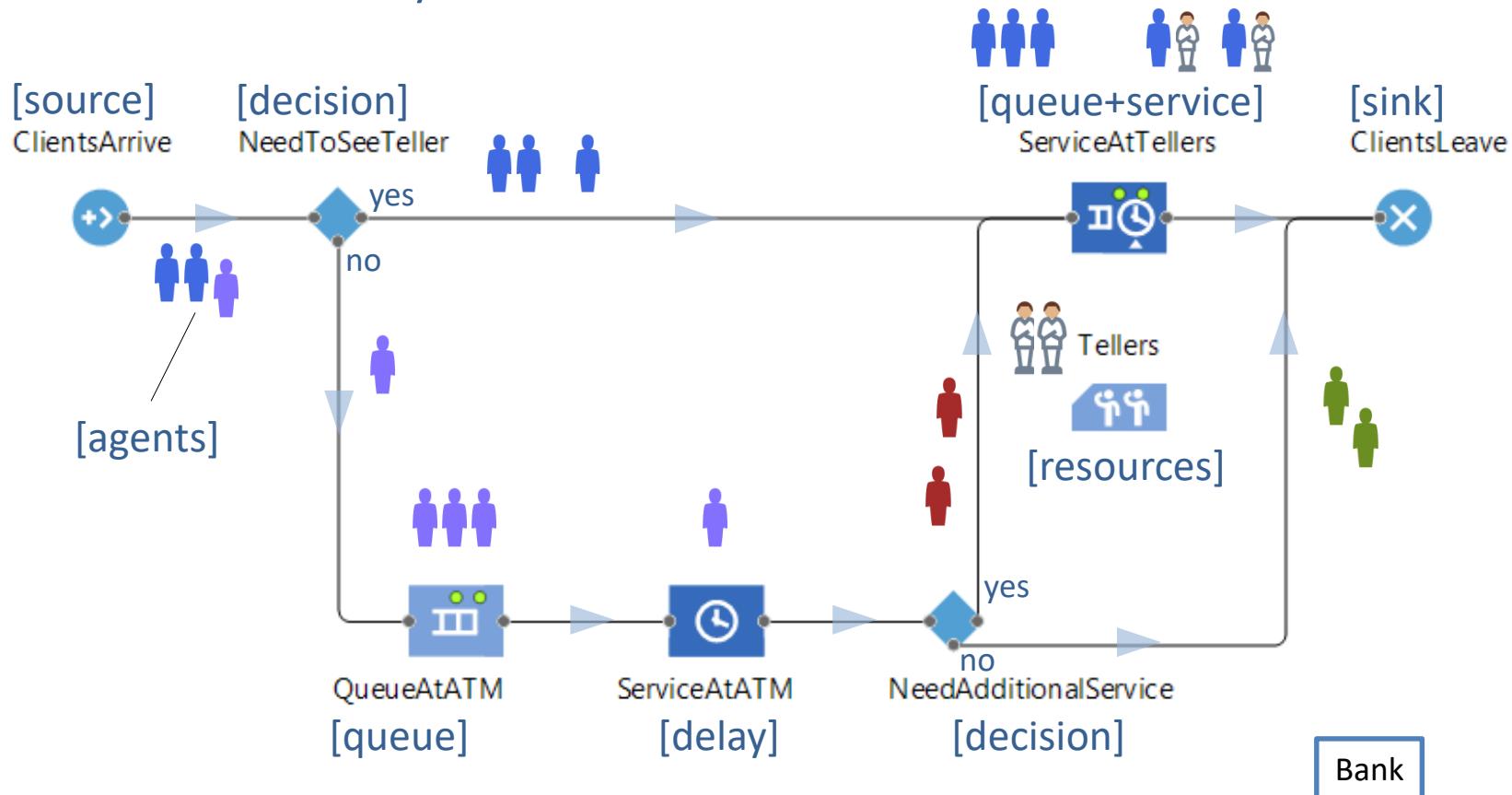
$$\begin{aligned}\text{Sales from Word of Mouth} &= \\ &\text{Clients} * \text{Contact Rate} * \\ &(\text{Potential clients} / (\text{Potential clients} + \text{Clients})) * \text{Adoption Fraction}\end{aligned}$$

Bass Diffusion Model



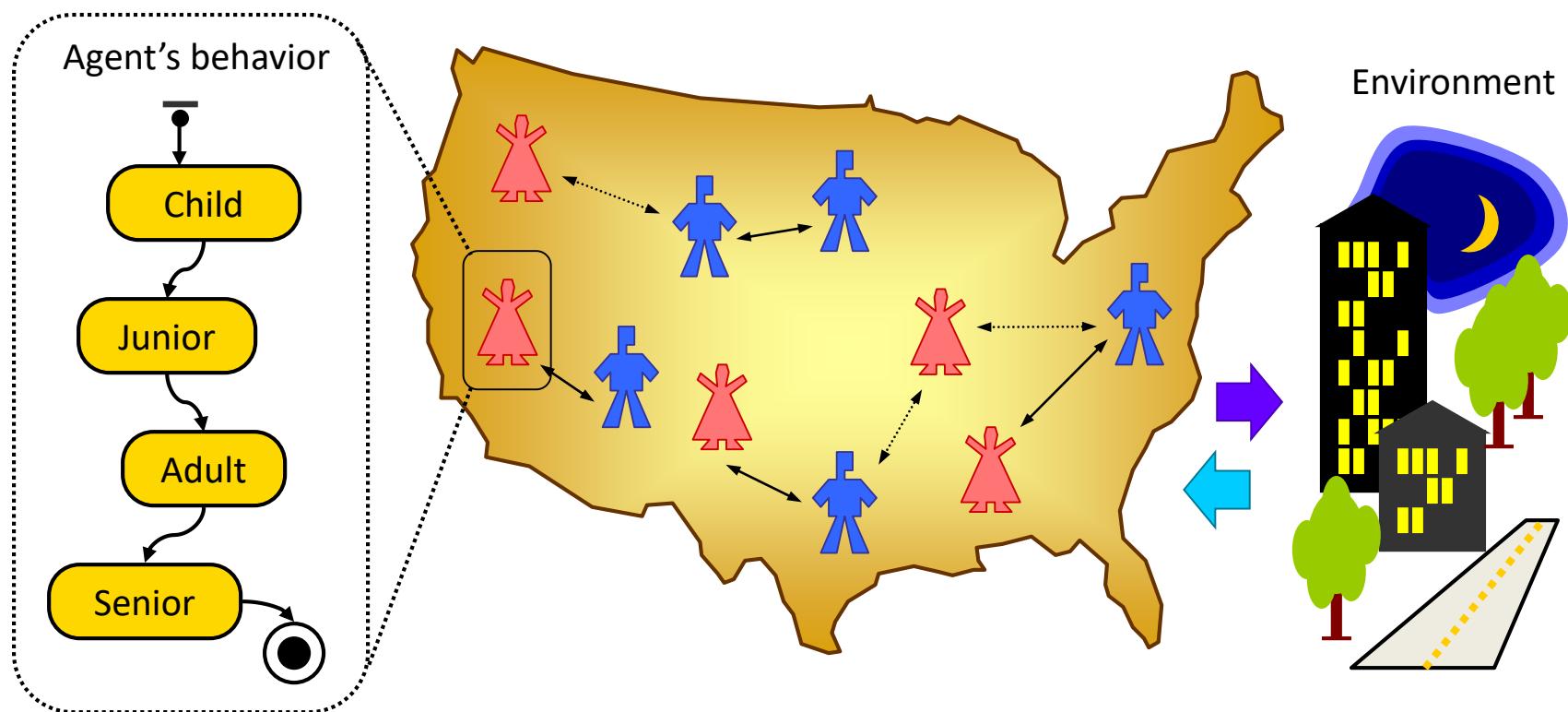
Discrete event modeling. G. Gordon '60s

- Agents and resources. Flowchart diagram
 - Queues and delays



Agent based modeling

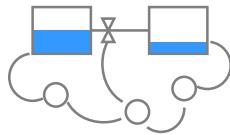
- We focus on individual objects and describe their local behavior, local rules
 - Sometimes, we also model the dynamics of the environment



Simulation modeling software

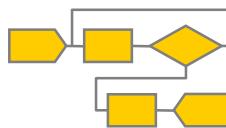
- Traditional tools are designed to support one particular modeling approach

System dynamics



VenSim
PowerSim
iThink

Discrete event modeling



Arena
ExtendSim
SimProcess
AutoMod
PROMODEL
Enterprise Dynamics
FlexSim

...

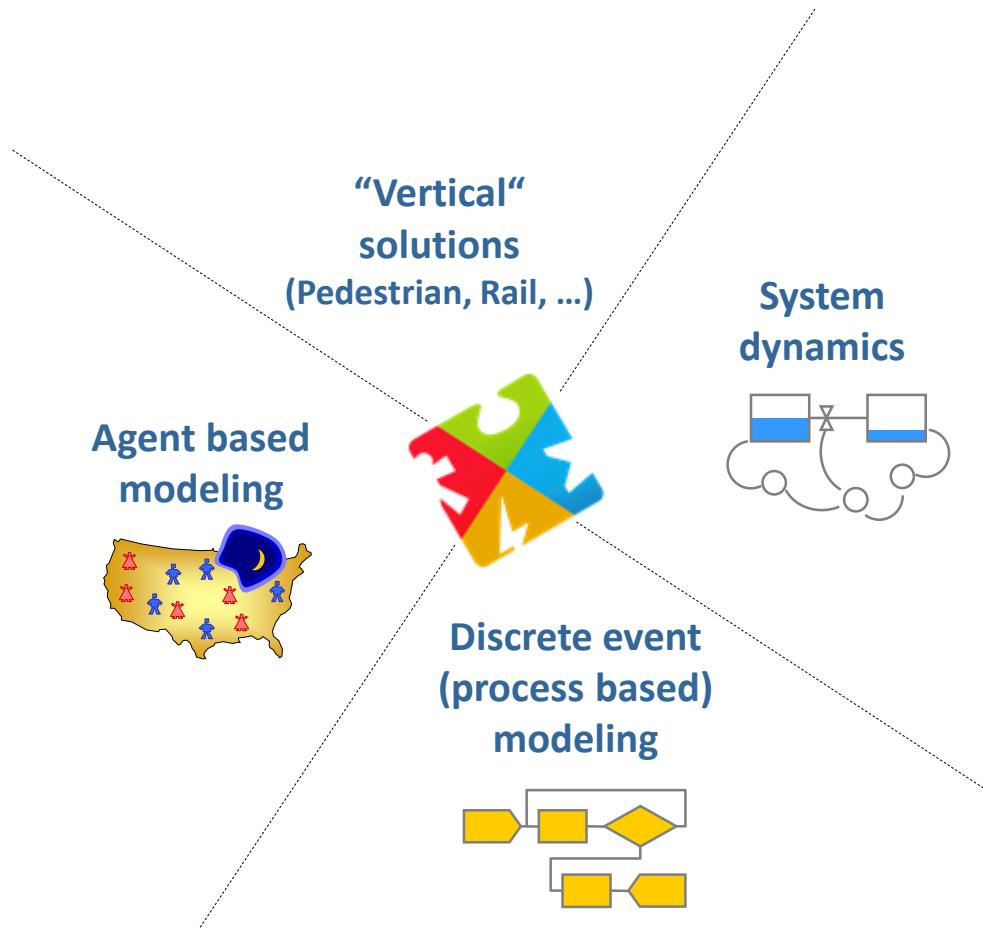
Agent based modeling



[Academic tools:]
Swarm
RePast
NetLogo
ASCAPE



AnyLogic – multi-method simulation tool



- Easy to choose and adjust the abstraction level
- Can switch between different methods
- Can mix methods in one model
- Modern and flexible OO platform



The AnyLogic Model Development Environment

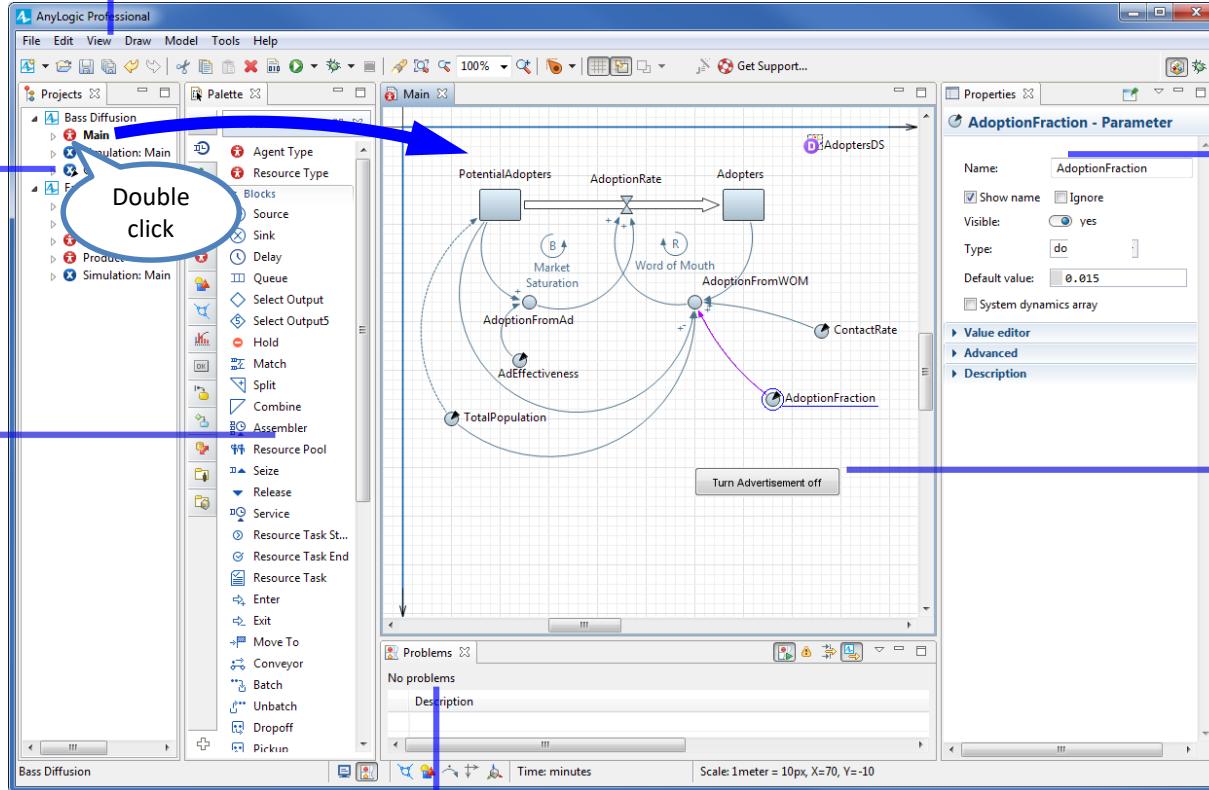
This presentation is a part of
AnyLogic Standard Training Program



AnyLogic GUI

Project view.
Easy
Navigation

Palette view.
Model items
grouped in
stencils



Menu and Toolbars. Shortcuts to Most Commands

Properties of
the currently
selected item

Graphical
Editor
Diagram

Problems view. Displays and helps to locate Errors



Menu and Toolbars

- Shortcuts to most commands
- Automatically adjust to current view

[Standard](#) (New Model, Open Model, Save Model, Save All Models)



[Edit](#) (Undo, Redo, Cut, Copy, Paste, Delete)



[Build](#) (Build Model, Run, Debug)



[Draw](#) (Zoom to 100%, Zoom In, Zoom, Zoom Out, View Areas, Show/Hide Grid, ...)



Model Structure – All Elements as a Tree

Double-Click to open editor and/or properties of an element

Models with unsaved changes are marked with asterisks

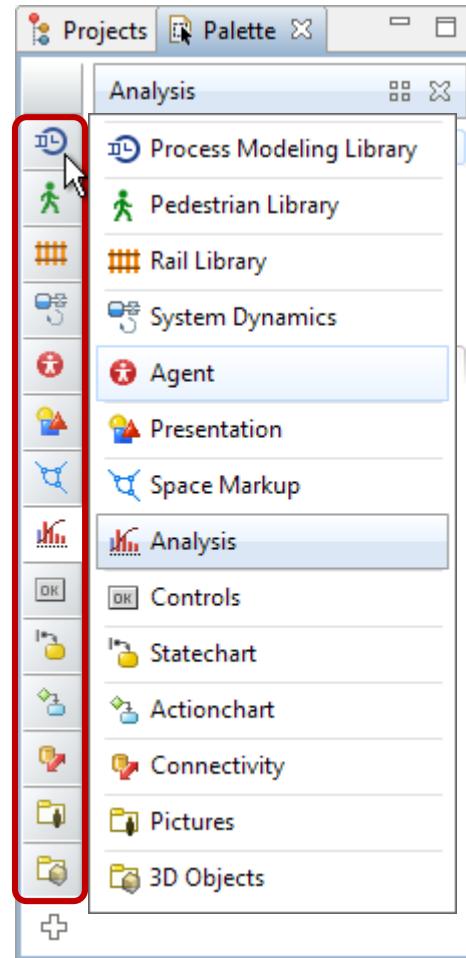
All model items are organized hierarchically

Right-Click (Mac OS: Ctrl+click) for context menu

Use Copy/Cut and Paste to copy within the tree



Navigation between palettes



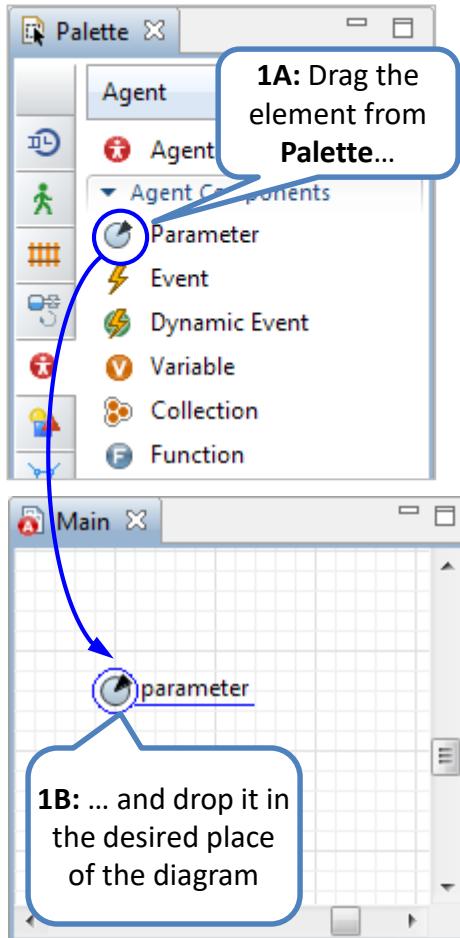
1. Hover the mouse over the vertical navigation bar

2. You will see the popup list with the palette names. Just choose the palette name from the list.



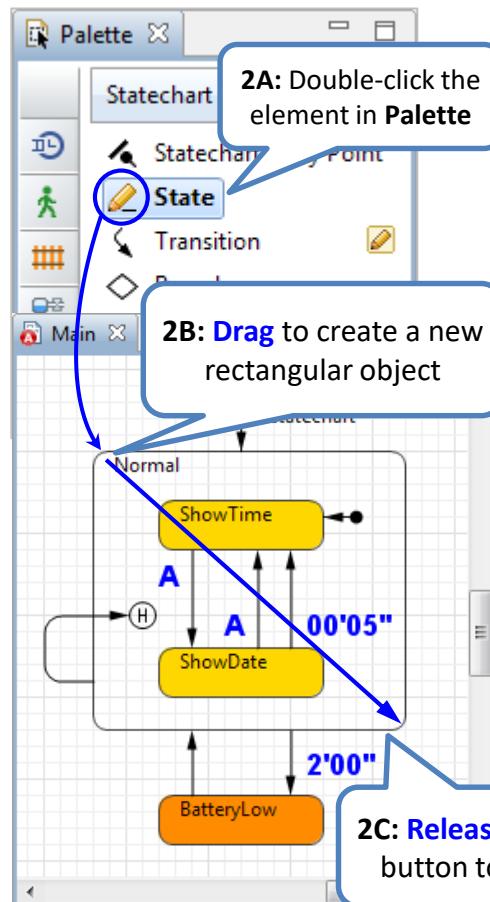
Adding Palette Items on Diagram

1. Common way



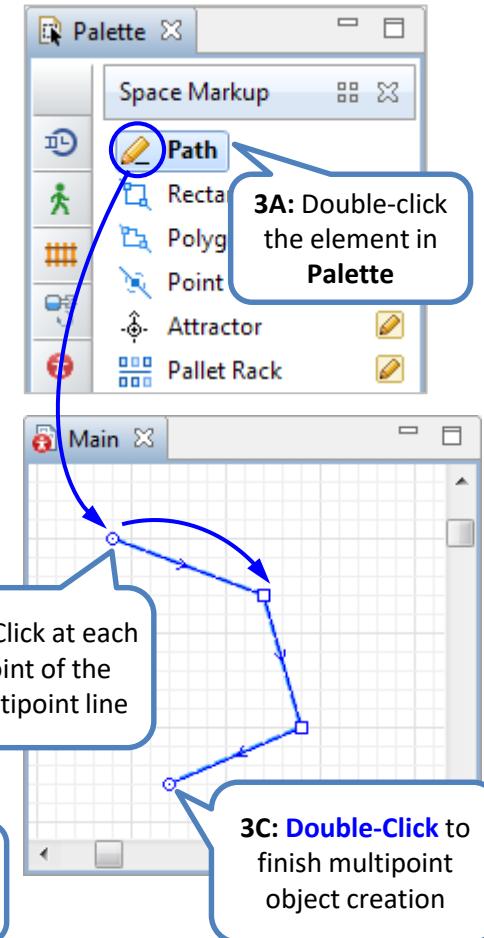
2. Rectangular objects

(State, Rectangle, Oval
Rounded Rectangle)

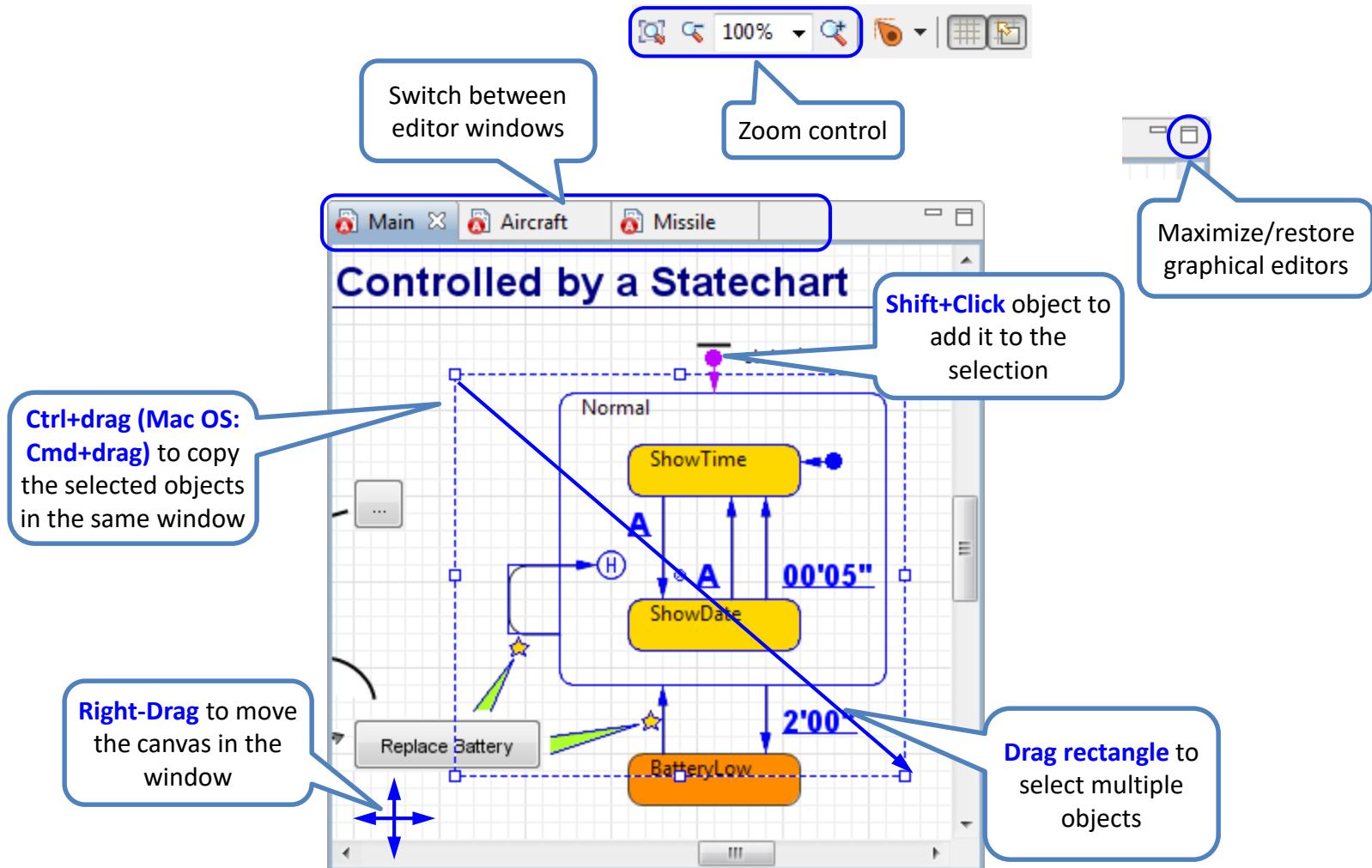


3. Multipoint objects

(Path, GIS Route, Transition, Connector)



Graphical Editor – Selecting & Copying



Properties view

- The **Properties** view allows you to view and modify the selected item's properties (you select an item by clicking it in the graphical editor, or in **Projects** view).

Legend:

= Static value

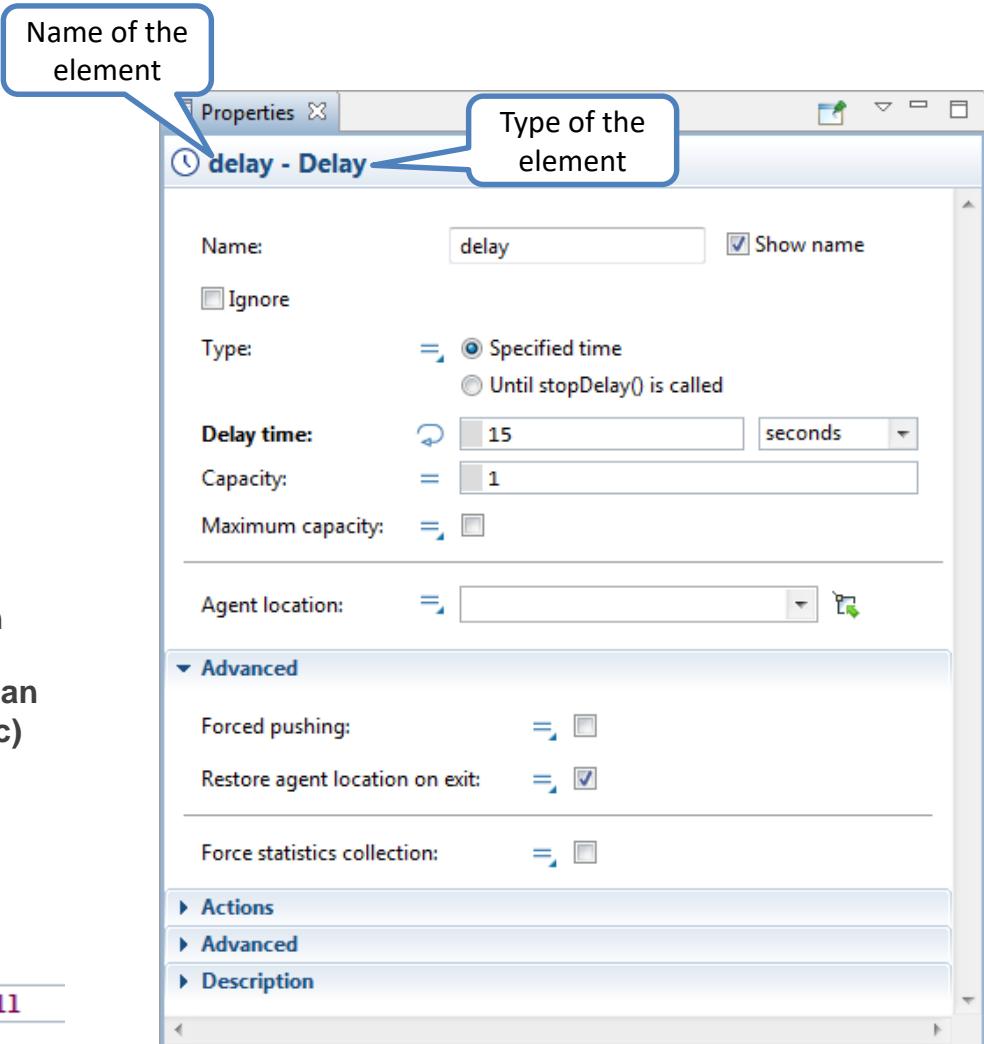
⌚ Dynamically evaluated expression

=<-- Small triangle indicates that you can switch between design-time (static) and run-time (dynamic) values

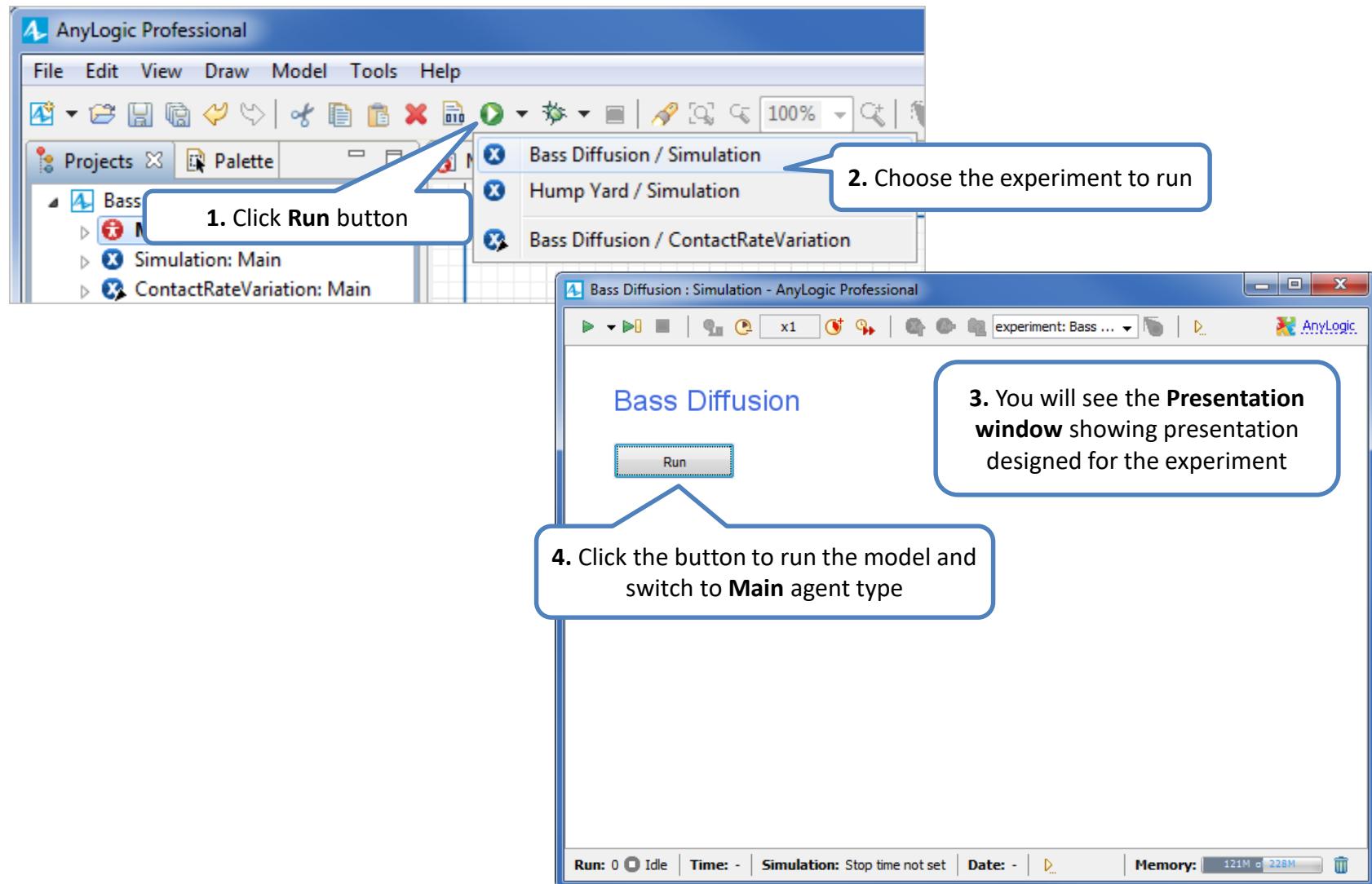
Fill color: = red



Fill color: ⌚ line.isVisible() ? red : null



Running the Model



Presentation Window

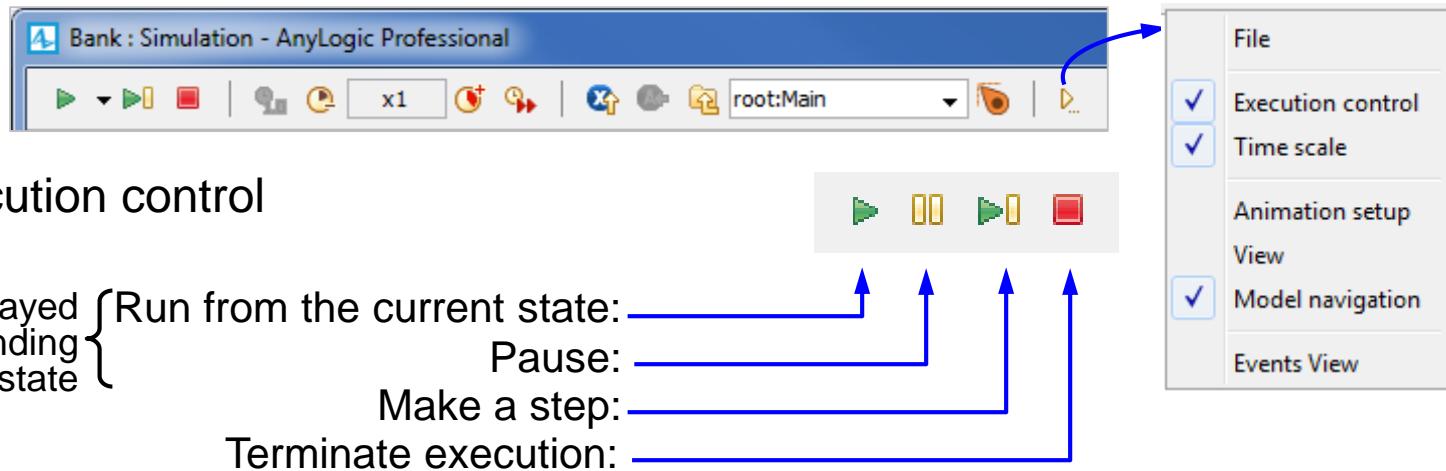
The screenshot shows the AnyLogic Professional simulation window titled "Bank : Simulation - AnyLogic Professional". The window is divided into several sections:

- Control:** Top left, shows the flowchart of the simulation model.
- 2D Animation:** Middle left, shows a 2D animation of a queue with tellers and customers.
- 3D Animation:** Bottom left, shows a 3D animation of a similar queueing system.
- Statistics and Distributions:** Middle right, displays two histograms:
 - "Waiting time distribution" (green bars) with x-axis from 0 to 10 and y-axis from 0% to 30%. Data: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
 - "Time in system distribution" (red bars) with x-axis from 0 to 15 and y-axis from 0% to 20%. Data: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.
- Wait Time Distribution Inspector:** Bottom right, shows a detailed view of the wait time distribution with 9 samples, mean of 4.228, and a table of PDF and CDF values.
- Navigation:** A blue arrow points to the scroll bar on the right side of the window.
- Annotations:** Three callout bubbles explain features:
 - "Control" points to the flowchart.
 - "2D Animation" points to the 2D queue visualization.
 - "3D Animation" points to the 3D queue visualization.
 - "Right-Drag to move the canvas in the window" points to the scroll bar.
 - "Inspect for model element" points to the Wait Time Distribution inspector.



Major Toolbar Commands

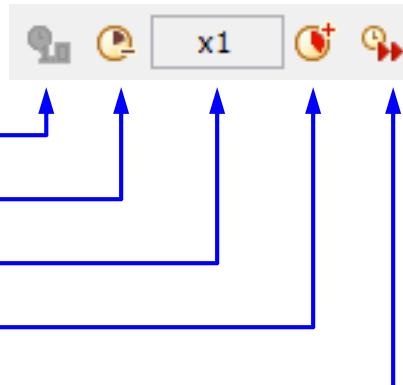
- Toolbars and status bar can be customized



- Time scale

Real Time mode only {

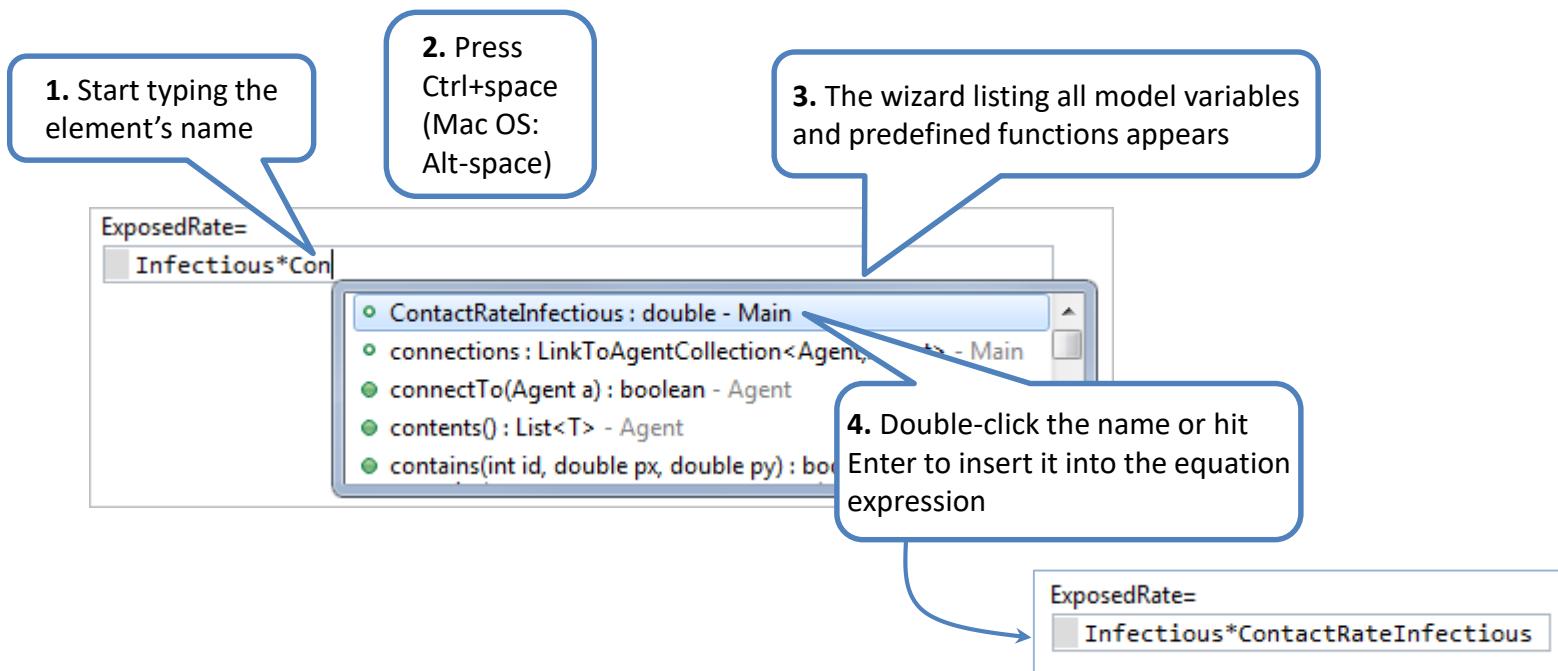
- Set real time mode at default scale:
- Decrease execution speed:
- Select execution speed factor:
- Increase execution speed:
- Toggle Real/Virtual time modes:



Code Completion Master

AnyLogic supports intelli-sense mechanism.

The wizard looks as a list, containing variables, parameters, and functions. You can simply select the name in the list, and it will be inserted in the expression automatically.



AnyLogic Help

The screenshot shows the AnyLogic Professional interface. On the left, the main workspace displays a project named "Agent" with an "Agent" component selected. On the right, a "Help - AnyLogic 7 Professional" window is open, showing the "AnyLogic Help" contents. A callout bubble from the top-left points to the search bar in the help window with the text "Help system supports search mechanism". Another callout bubble from the bottom-left points to the "Aligning elements" section in the help window with the text "Help also includes:". The help window also shows a context menu for two selected objects, "parameter" and "parameter1", with the "Align" option highlighted, and a submenu showing various alignment options like "Horizontal Align Left", "Horizontal Align Center", etc.

Help system supports search mechanism

Help also includes:

- Self-paced tutorials,
- Reference Guides on AnyLogic Libraries (Process Modeling, Pedestrian, Rail, ...)
- Java Documentation on all AnyLogic classes and functions

AnyLogic Help

Welcome

Example Models

Search: Go Scope: All topics

AnyLogic Help > AnyLogic User Interface > Graphical editor

Aligning elements

You can align elements in the graphical editor.

To align elements, first, you should select them in the graphical editor, then right-click (Mac OS: Ctrl+click) the selection and choose the required alignment command from the Align submenu.

Align

- Horizontal Align Left
- Horizontal Align Center
- Horizontal Align Right
- Vertical Align Top
- Vertical Align Center
- Vertical Align Bottom

Please note that you can align elements of different types: presentation shapes, library objects, parameter, function, database icons, etc.

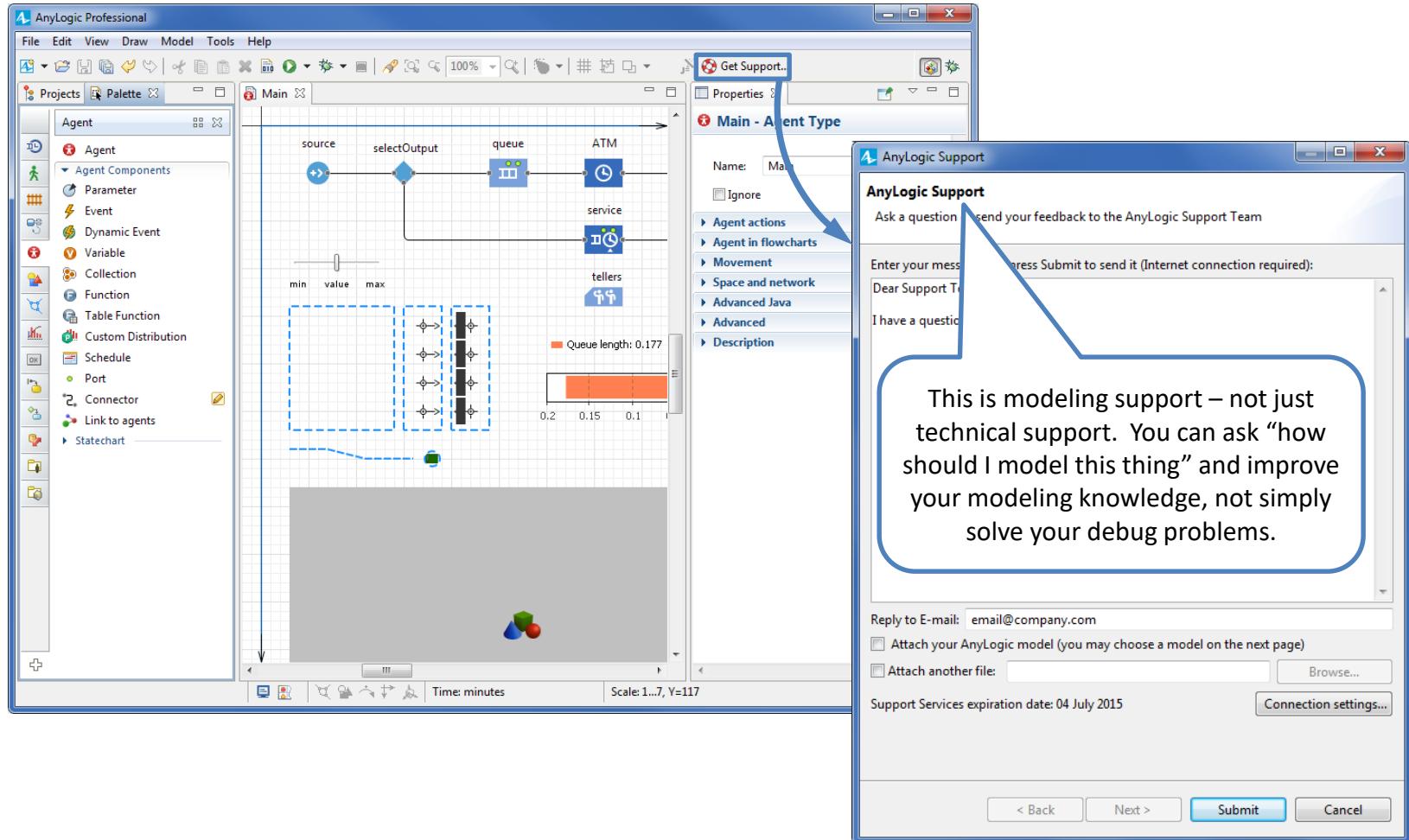
You can not align statechart and actionchart elements since this may distort their structure and logic.

Let's illustrate all alignment commands by example of three shapes:



AnyLogic Support

- Use the built-in Get Support feature



Discrete Event Modeling in AnyLogic

This presentation is a part of
AnyLogic Standard Training Program



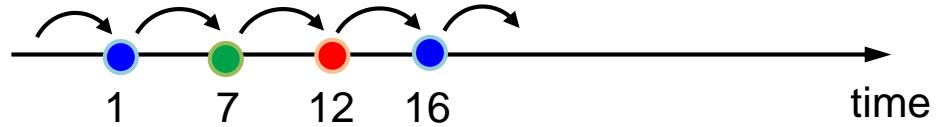
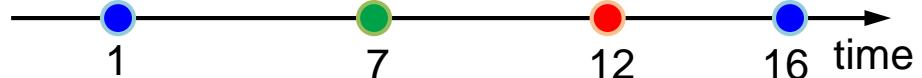
Event

- We consider only “important moments” in the system’s lifetime, which are called *events*.
 - Any change in the model may happen only as a result of an event
- Examples:
 - Customer arrives at the bank office
 - Bill finishes processing
 - Amount of raw material reaches the minimum level
- Event:
 - Takes zero model time
 - Causes changes in the model
 - May schedule or cancel other events in the future



Time as event order – Discrete Time

- We consider only a sequence of instant “discrete” events, while nothing happens in between
- No “continuous-time” processes
- Model time “jumps” from one event to another

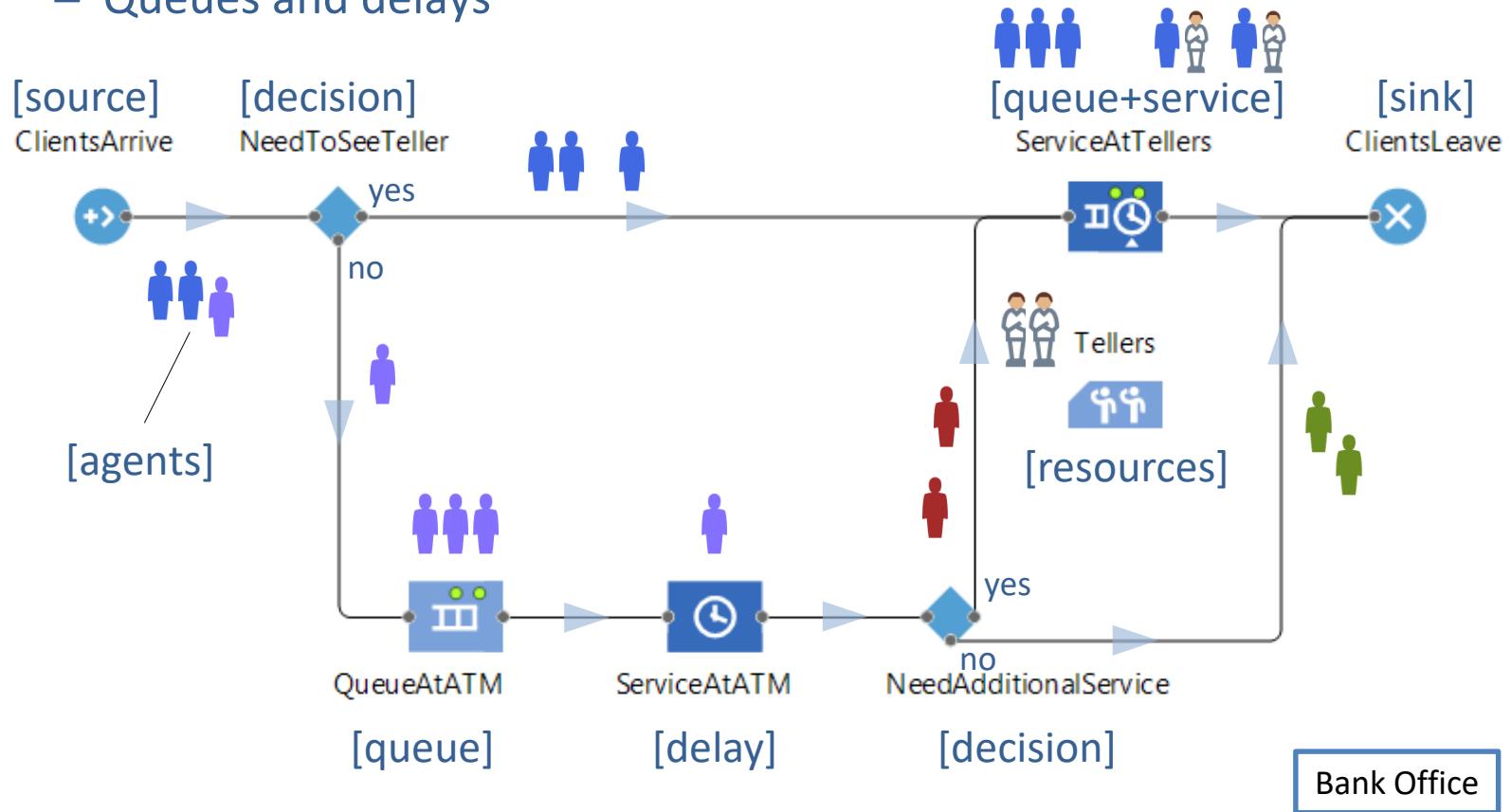


This is Discrete Event Modeling



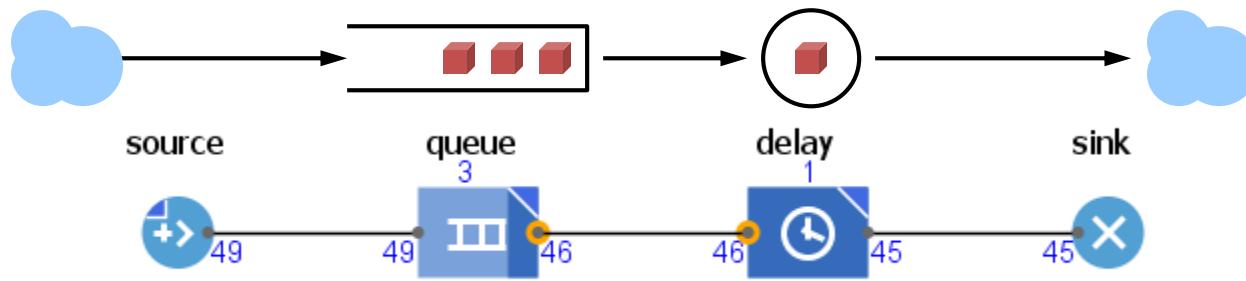
Discrete event modeling. G. Gordon '60s

- Agents and resources. Flowchart diagram
 - Queues and delays



The simple process based model

- Define the process with Process Modeling Library blocks with a simple drag and drop



Arrivals defined by: Rate
Arrival rate: 1
Limited number of arrivals:
New agent: Agent
Location of arrival:
On exit:

Capacity: 100
On enter:
On at exit:
On exit:
Enable exit on timeout:
Agent location:

Delay time: triangular(0.5, 1, 1.5)
Capacity: 1
On enter:
On exit:
Agent location:

On enter:



Process Modeling Library. Essential blocks (1/3)

Block name	Icon in graphical editor	Description
Source		Generates agents. Is usually a starting point of a process model.
Sink		Disposes incoming agents. Is usually an end point in a process model.
Queue		Stores agents in the specified order.
Delay		Delays agents by the specified delay time.
SelectOutput		Fowards the agent to one of the output ports depending on the condition.
SelectOutput5		Routes the incoming agents to one of the five output ports depending on (probabilistic or deterministic) conditions.



Process Modeling Library. Essential blocks (2/3)

Block name	Icon in graphical editor	Description
Conveyor		Simulates conveyor. Moves agents at a certain speed, preserving order and space between them.
Split		Creates a new agent (copy) of the incoming agent.
Combine		Waits for two agents, then produces a new agent from them.
Batch		Accumulates agents, then outputs them contained in a new agent.
Unbatch		Extracts all agents contained in the incoming agent and outputs them.
MoveTo		Moves an agent from its current location to new location.



Process Modeling Library. Essential blocks (3/3)

Block name	Icon in graphical editor	Description
RestrictedAreaStart		Using these blocks you can limit the number of agents in a part of flowchart between corresponding RestrictedAreaStart and RestrictedAreaEnd blocks.
RestrictedAreaEnd		
TimeMeasureStart		TimeMeasureEnd and TimeMeasureStart compose a pair of blocks measuring the time the agents spend between them.
TimeMeasureEnd		
Exit		Takes the incoming agents out of the process flow and lets the user to specify what to do with them.
Enter		Inserts the (already existing) agents into a particular point of the process model.



Parameters of Process Modeling Library blocks

- Simple static parameters:
= Evaluated **once**, but may be changed during the model execution
- Dynamically evaluated expressions (dynamic parameters):
Evaluated **each time they are needed**, e.g.
 each time the delay time, the speed or other property of an agent needs to be obtained
The corresponding agent is accessible as “agent”, etc.
- Dynamically executed code pieces (code parameters):
Evaluated **each time a certain event occurs** at the object: the agent enters/exits it, conveyor stops, etc.

Capacity =

Delay time 

Condition 

Speed 

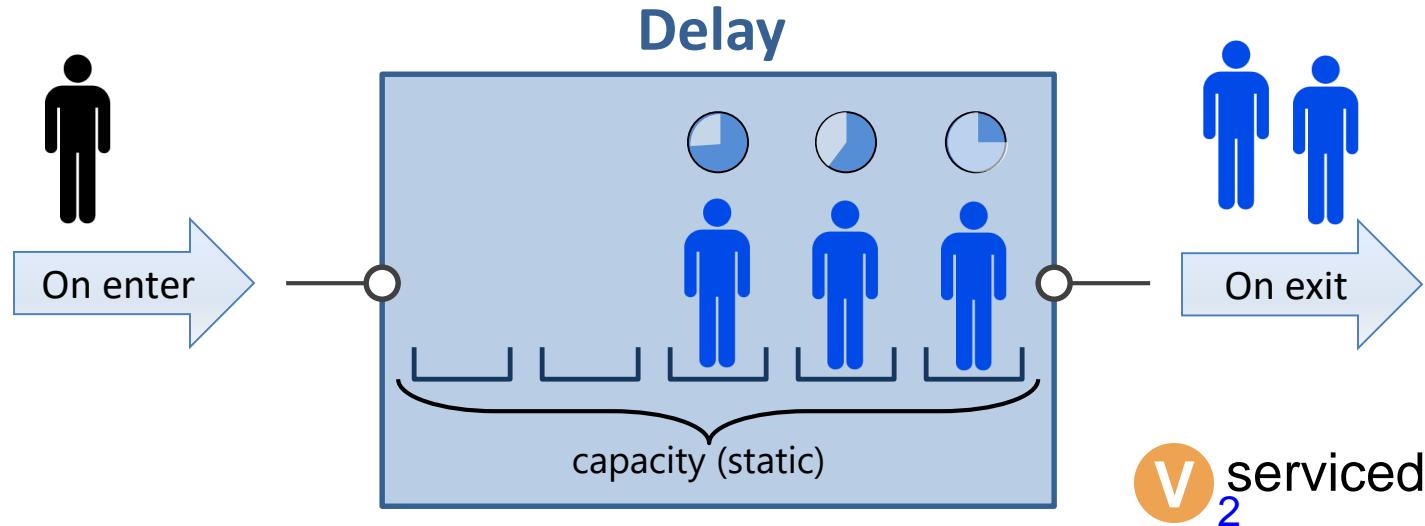
▼ Actions

On exit 

On enter  agent.destination = gate17;"/>



Parameters. Examples



Capacity: =

Delay time:

On enter:

On exit:

- To change a value of the static `capacity` parameter, call: `delay.set_capacity(20);`
- You **add a semicolon** at the end of each line of Java code in **code parameter**
- You **do not add a semicolon** to the end of **static/dynamic parameter expression**



Resources



Resource types

- Static (can't move and can't be moved): a room, a non-portable equipment, a passage, etc.
- Portable (can't move on their own, but can be moved): a wheelchair, portable xRay, etc.
- Moving (can move, carry portable resources): a doctor, a nurse, a forklift truck, etc.
- Please give the examples of resources we require to deliver this training course



PML blocks associated with resources

Block name	Icon in graphical editor	Description
ResourcePool		Defines a set of resources of the specified type: how many resources of this type exist in the system, what are they attributes.
Seize		Seizes the number of units of the specified resource required by the agent.
Release		Releases resource units previously seized by the agent.
Service		Seizes resource units for the agent, delays it, and releases the seized units. The block itself is a Seize – Delay – Release sequence of blocks.
Assembler		Assembles a certain number of agents from several sources (5 or less) into a single agent.
ResourceSendTo		Sends a set of portable and/or moving resources to a specified location.



Resource. Failures/repairs, breaks, maintenance

► ▾ Shifts, breaks, failures, maintenance...

Failures / repairs

Initial time to failure

Time to next failure

Repair type

Time to repair

Usage statistics is:

Maintenance

Initial time to maintenance

Time to next maintenance

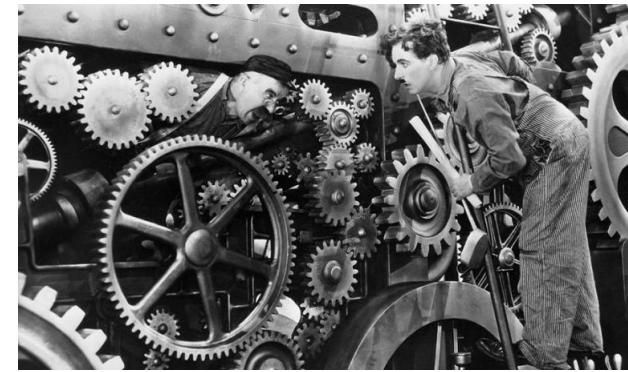
'Maintenance' priority

'Maintenance' may preempt

Maintenance type

Task start block (maintenance)

Usage statistics is:



Here you choose how the failure time is taken into account in the resource usage statistics

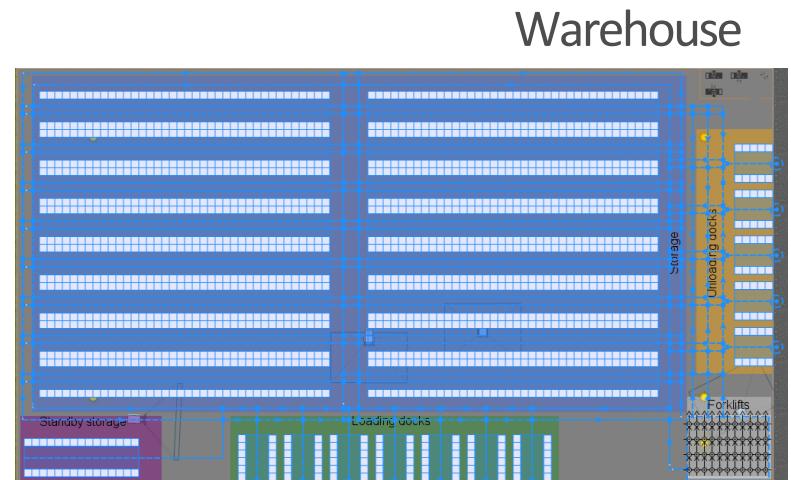
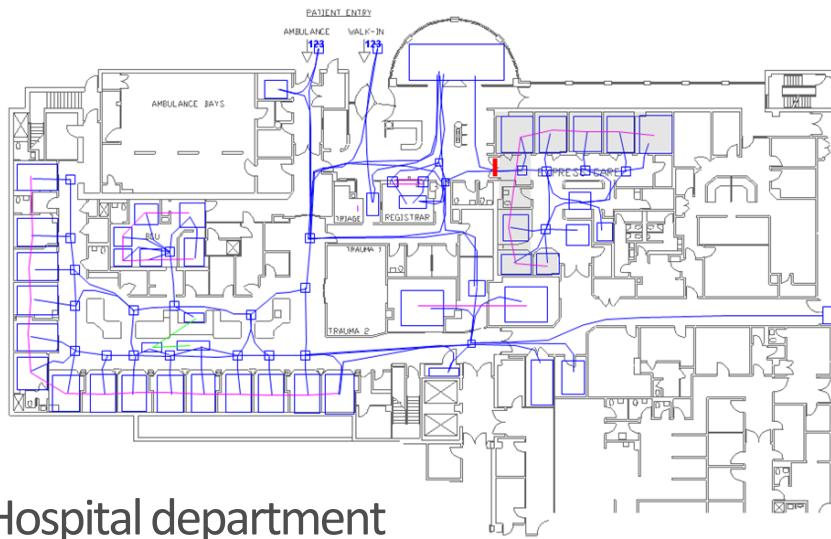
Alternative to modeling repair, or maintenance operation just with a *Delay*, you can model it with a flowchart describing the process. The flowchart start is defined with the specified **ResourceTaskStart** block.

ResourcePool



Network-based modeling

- The Process Modeling Library provides better support for certain types of problems where layout is important
 - There is a network of locations and paths between them
 - Agents and resources move along the paths, route lengths matter



Modeling warehouse storages

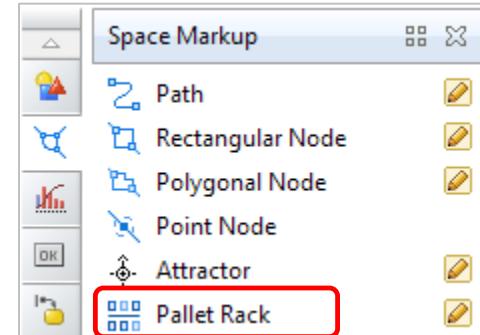
- Flowchart blocks for defining the logic:
 - RackSystem
 - RackStore
 - RackPick
- Special space markup shape:
Pallet Rack
 - Automatically connects to networks
 - Three topologies are supported:



a) One rack, one aisle

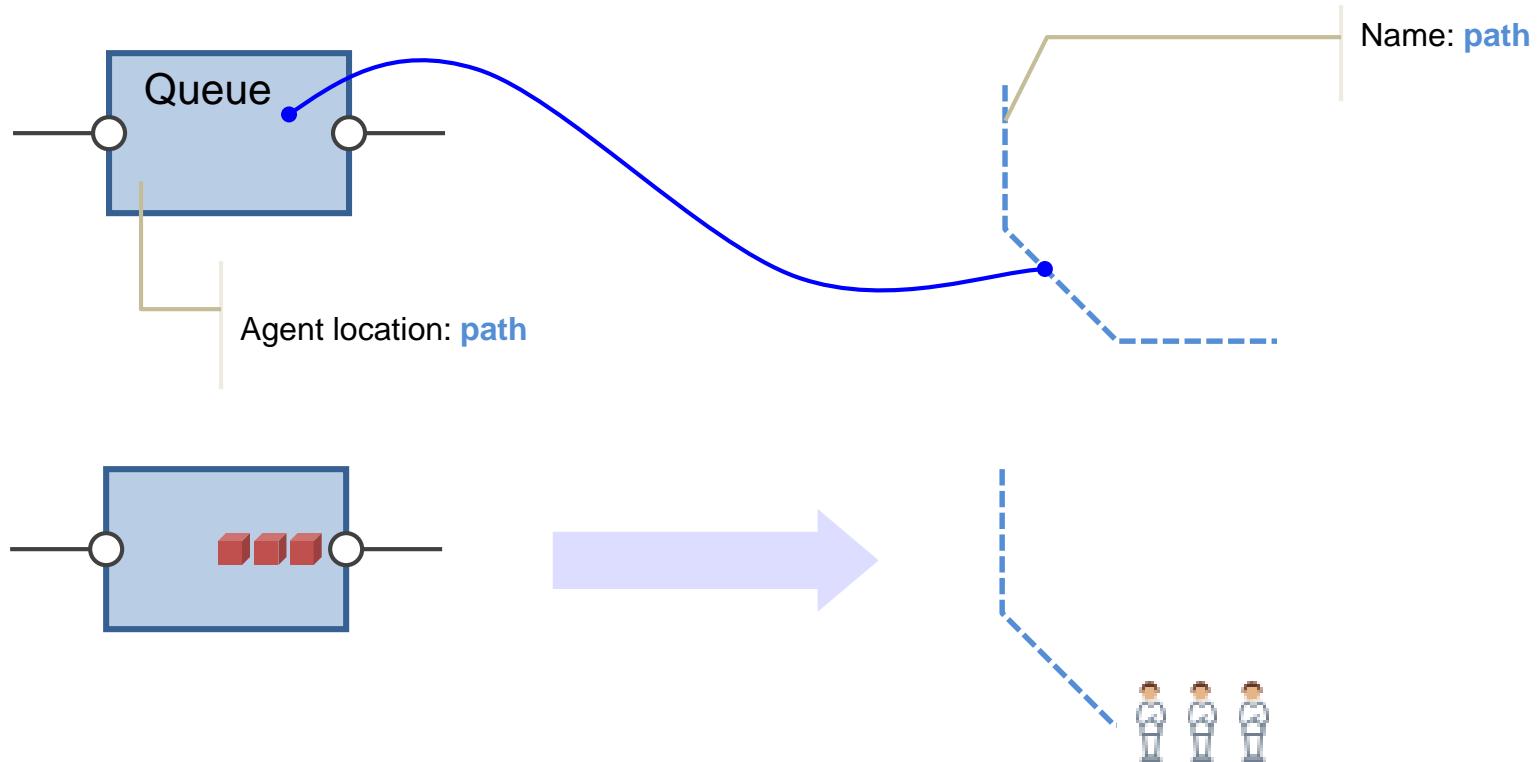
b) One rack, two aisles

c) Two racks, one aisle

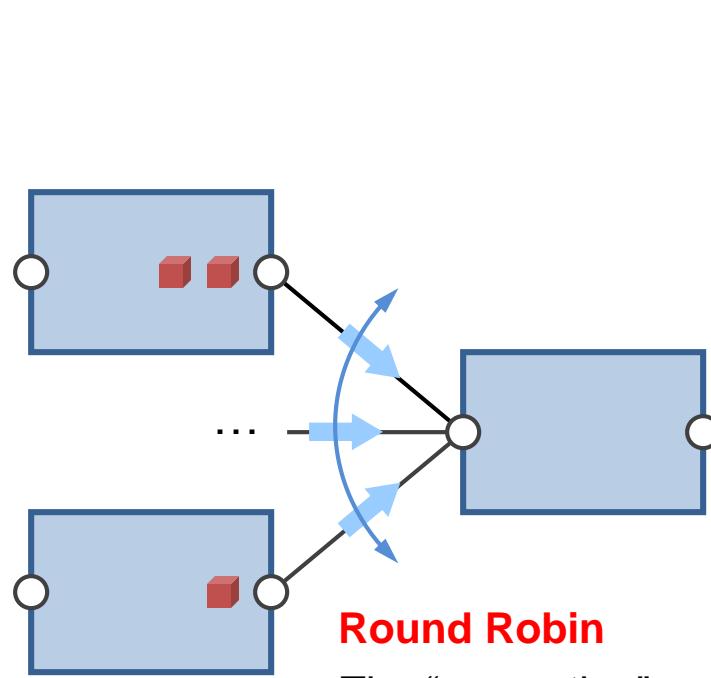


Animation of Process Modeling Library models

You can associate any object that contains agents with a **space markup** shape (**path** or **node**) defining the location of agent animations.

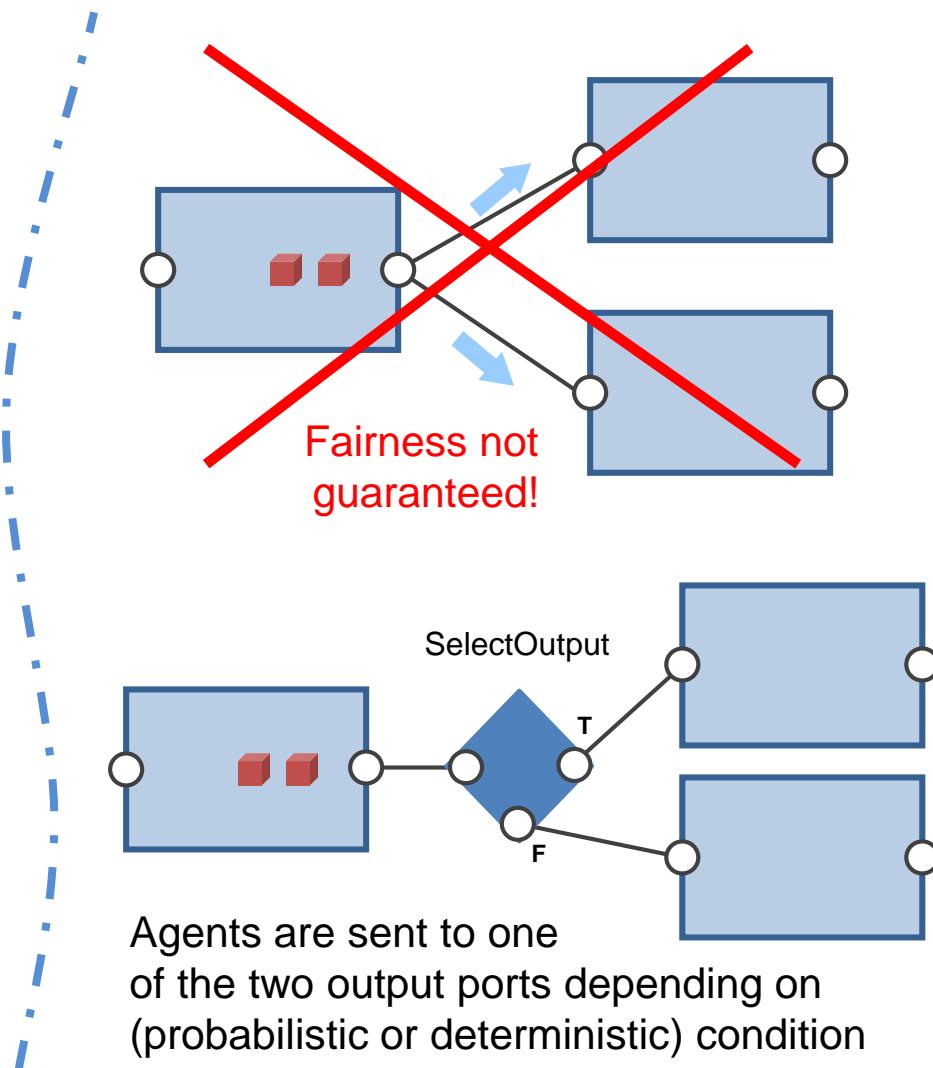


Agent flow at N:1 and 1:N connections



Round Robin

The “competing” outputs are served in round robin manner to ensure fairness



Agents are sent to one of the two output ports depending on (probabilistic or deterministic) condition



The Factory Model

This presentation is a part of
AnyLogic Standard Training Program



The Factory Model

We will create a model of a factory shop.

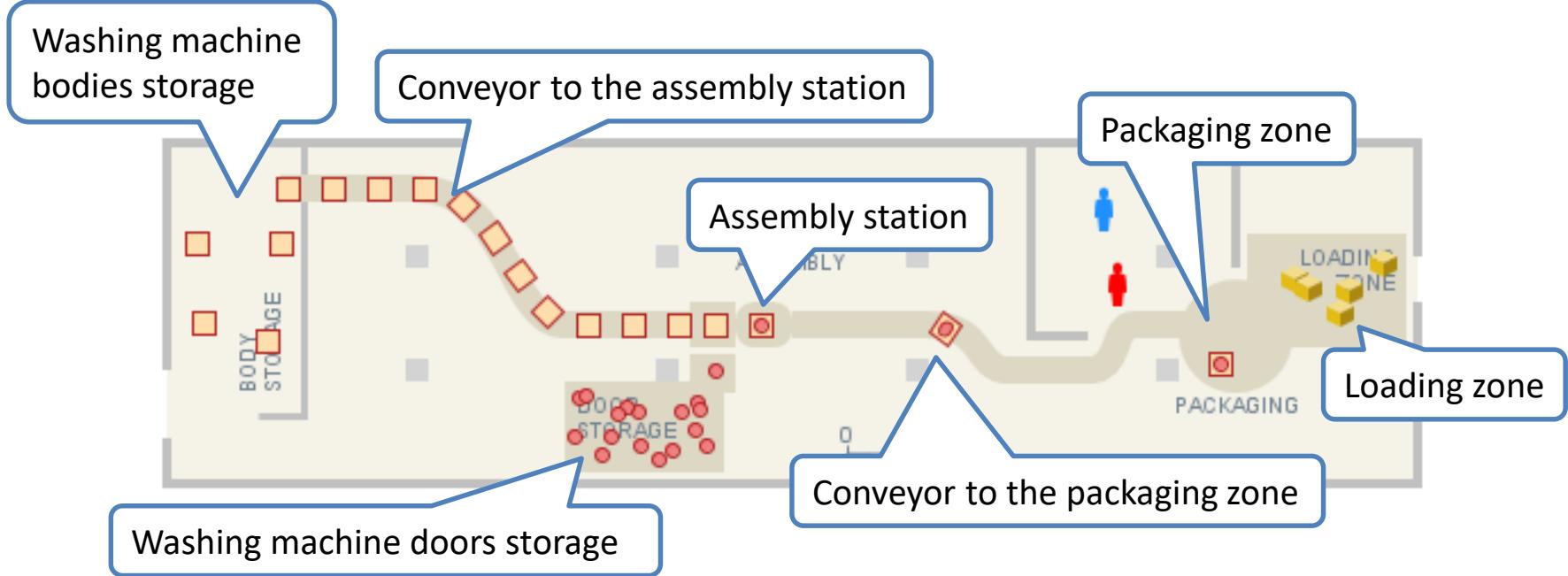
The factory produces washing machines. This particular floor performs the final stage of manufacturing – assembling a finished machine from two parts – a body and a door.

The floor works as follows:

- Parts enter the shop and are transported by conveyors to the assembly station.
- Assembly robot assembles a washing machine from a body and a door.
- Finished machine is transported by a conveyor to the packaging line, where it is packed into a box by workers.
- Each 10 finished products compose a batch that is taken away from the factory by a truck.



The Factory Model

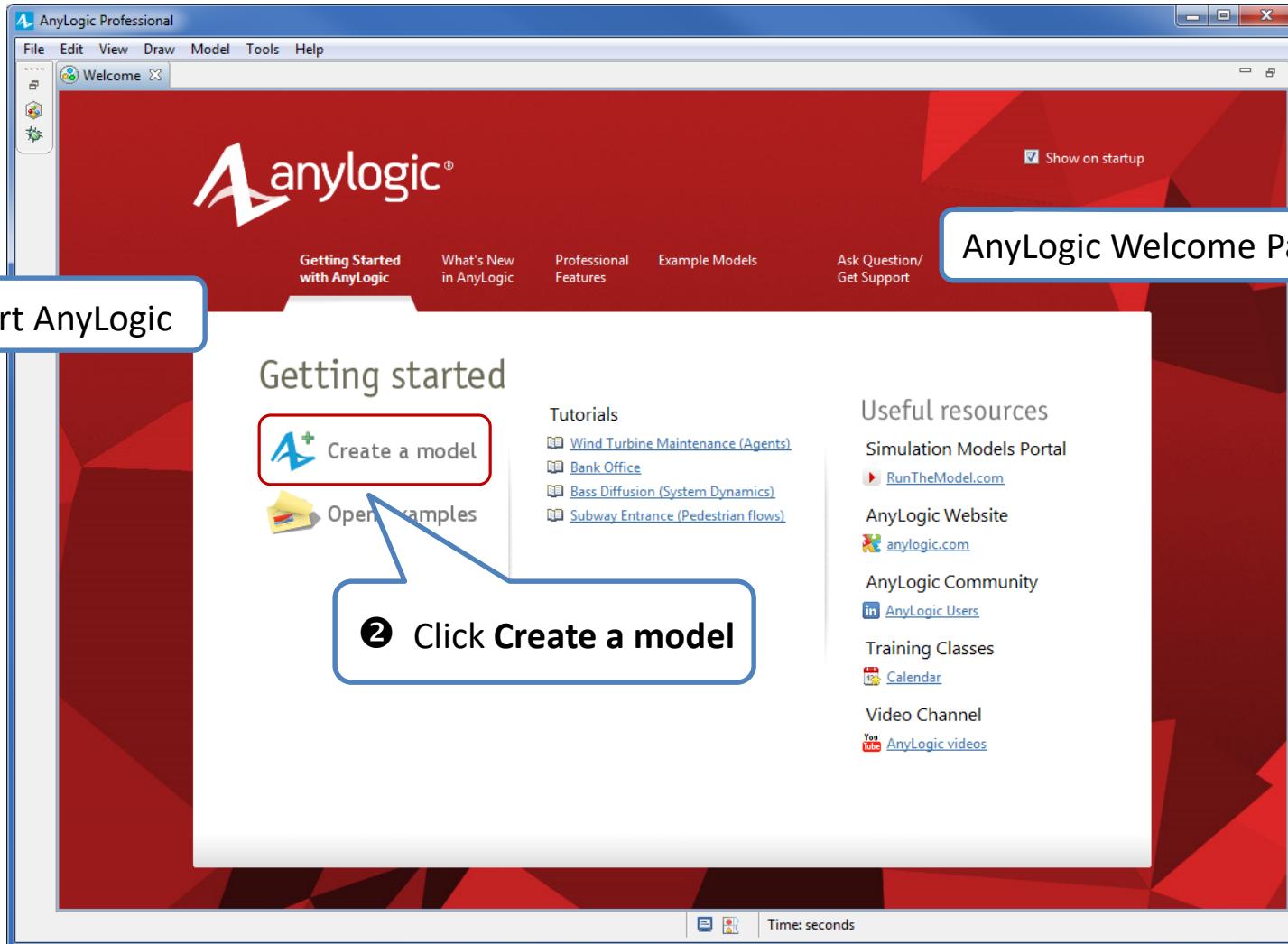


Factory. Phase 1

- We will start with a very simple model that will model how washing machine bodies enter the factory floor and are transported by a conveyor to the place where the assembly will be performed.



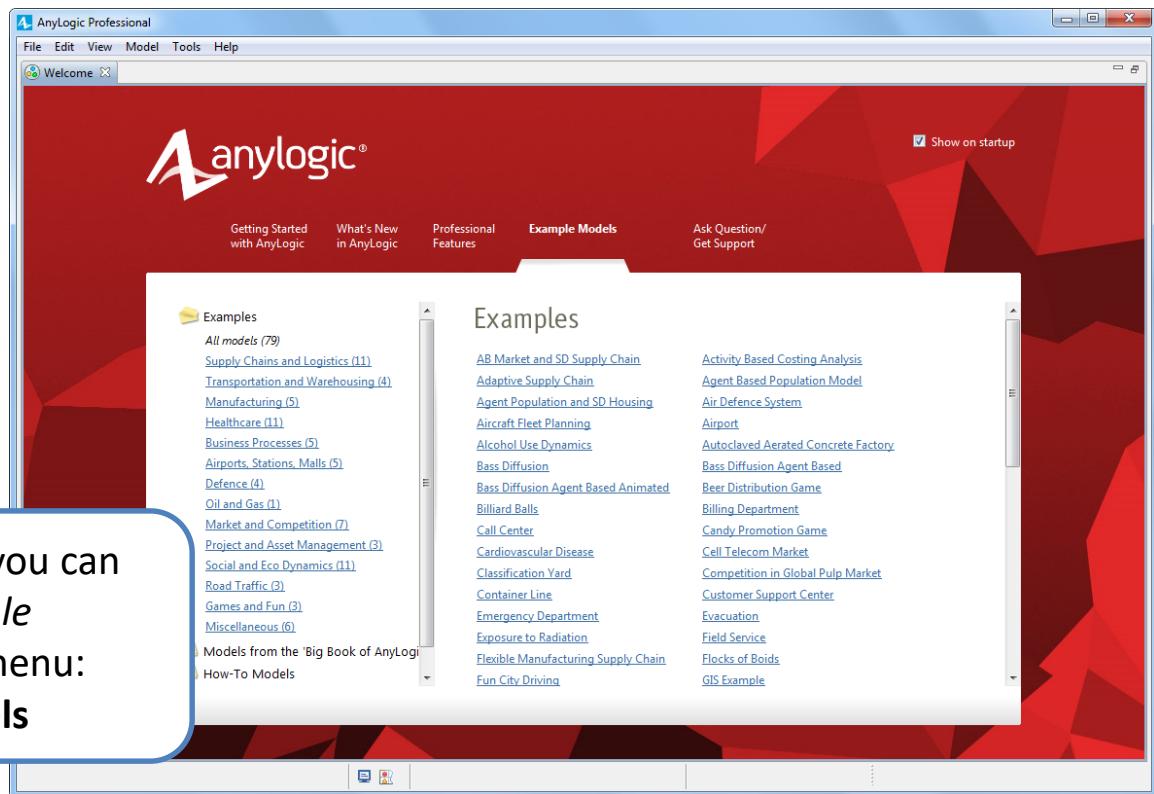
Factory. Phase 1. Step 1



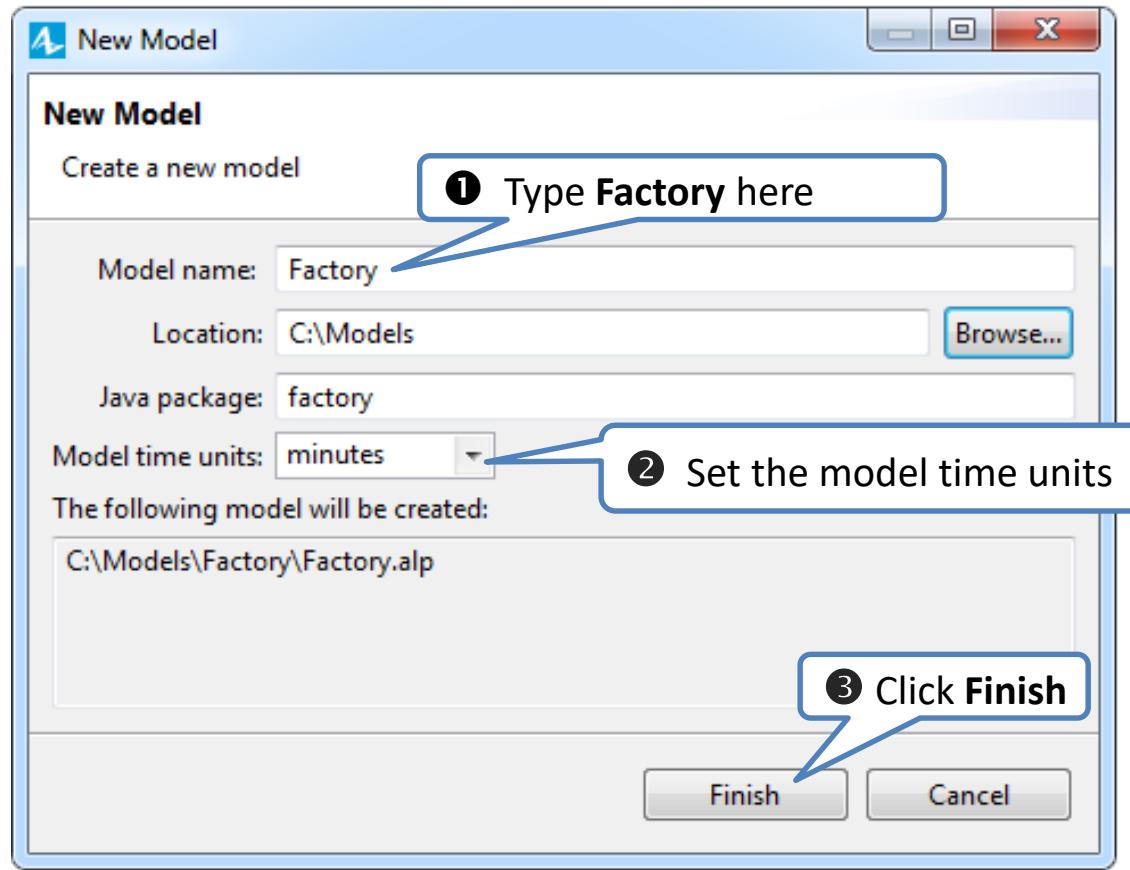
- ① After you start AnyLogic, you will see the **Welcome page**

Welcome page

- The purpose of the **Welcome page** is to introduce you to the product. The welcome page provides access to the example models, tutorials, books and other resources helpful to learn simulation modeling and AnyLogic.



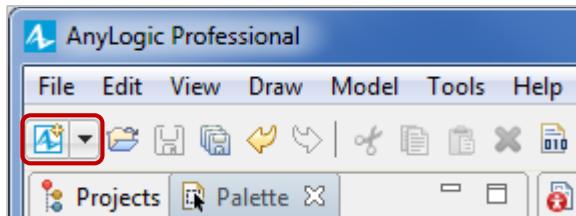
Factory. Phase 1. Step 2



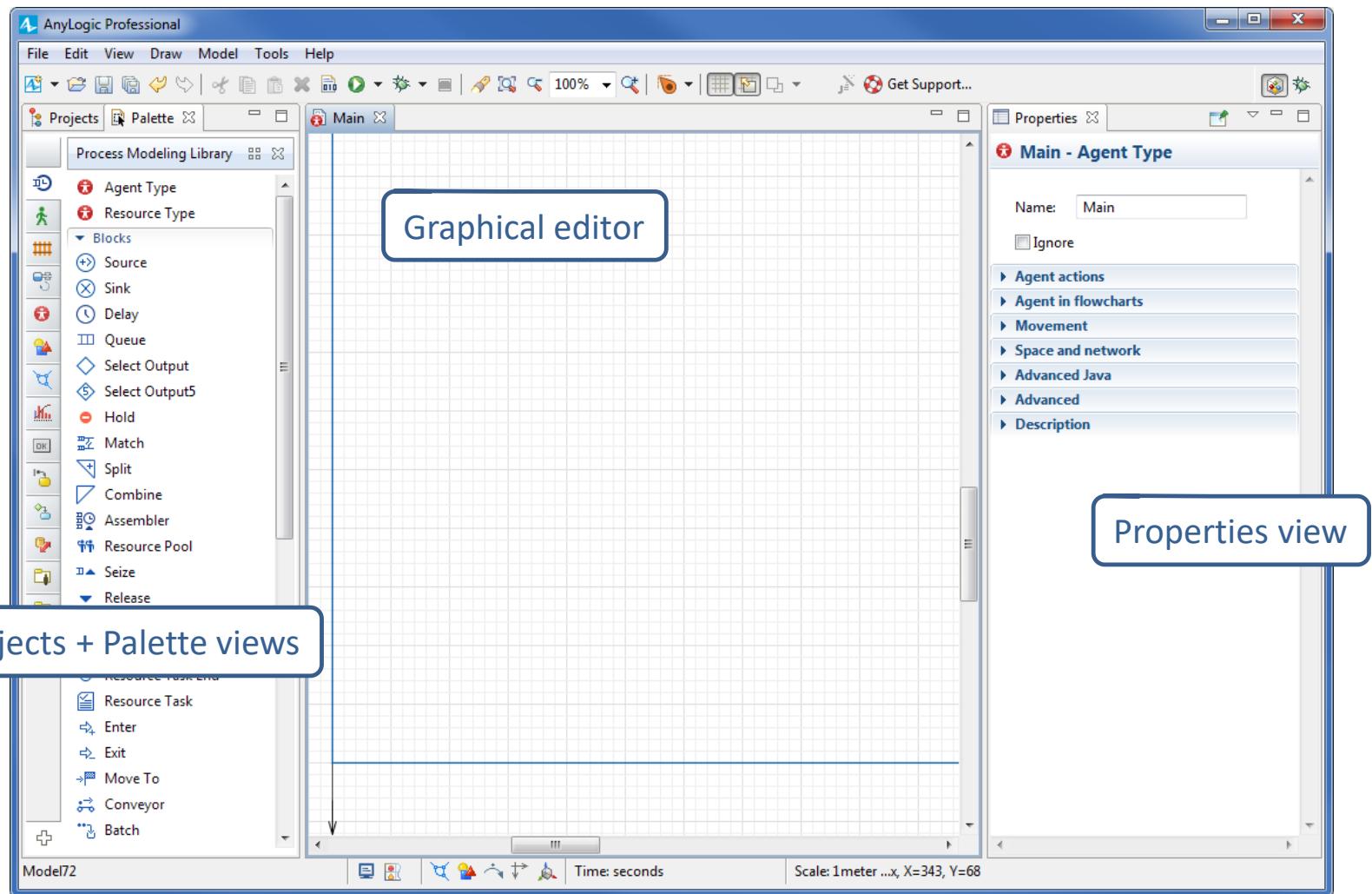
You will see the **New Model** wizard appear.

- ① Enter the name of a new model: *Factory*. The Factory model will act as a component of a global supply chain model later.
- ② Choose *minutes* as the model time units.
- ③ Click **Finish** to start creating new model from scratch.

Another way to create a new model is to click the **New** button located on the toolbar:



Factory. Phase 1. Step 3



AnyLogic Graphical User Interface

Graphical editor

- The place to graphically edit the diagram of the agent type.

Projects view

- Provides access to AnyLogic models currently opened in the workspace. The workspace tree provides easy navigation throughout the models.

Palette view

- Provides the list of model elements grouped by categories in a number of stencils (palettes).

Properties view

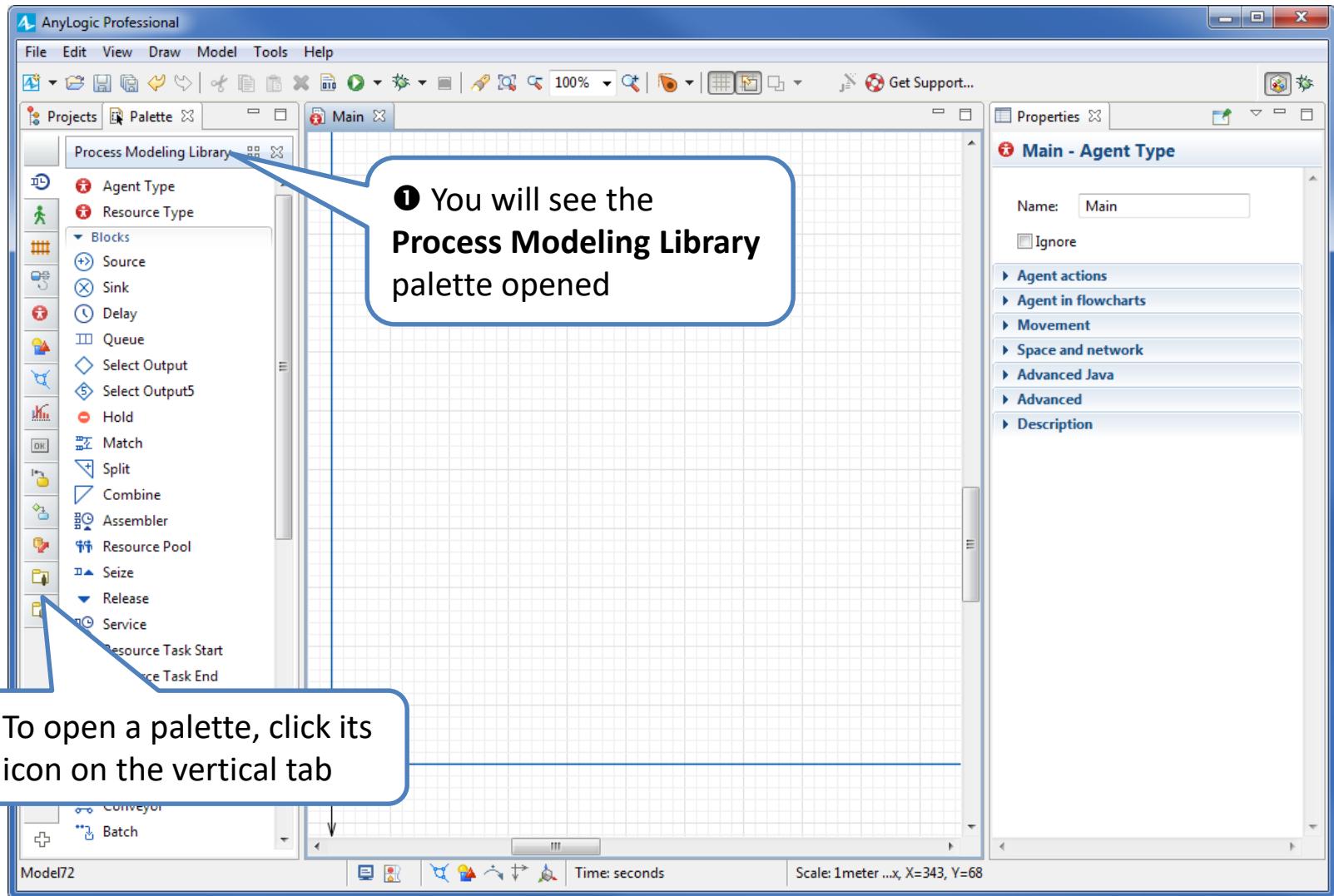
- Allows to view and modify the properties of currently selected model item(s).

Problems view

- Displays errors found during model development and compilation.



Factory. Phase 1. Step 4



We will model our factory using *AnyLogic Process Modeling Library*.

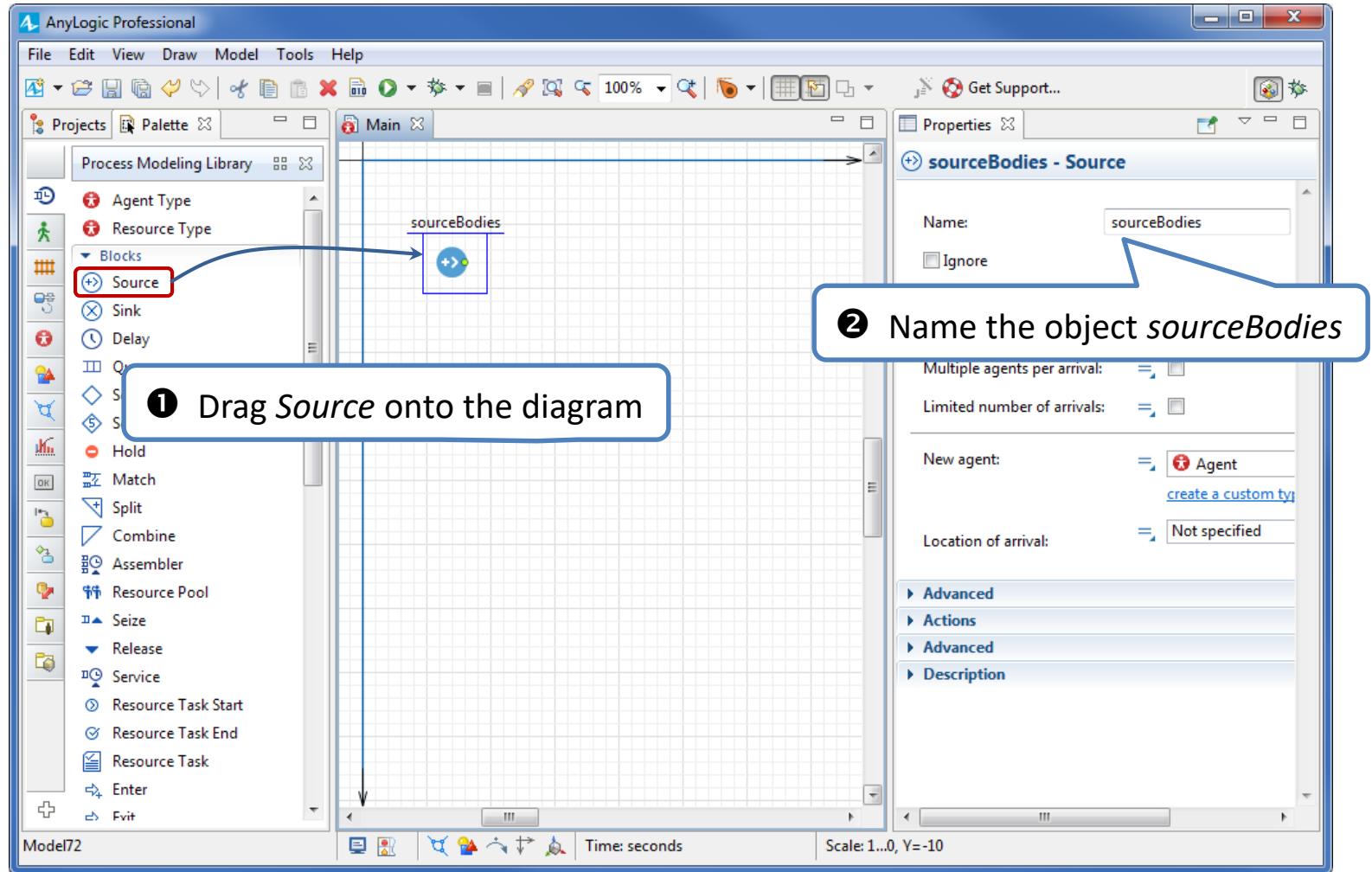
AnyLogic Process Modeling Library

- **Process Modeling Library** is the standard AnyLogic library for discrete-event, or, to be more precise, process-centric modeling. Using the Process Modeling Library objects you can model real-world systems in terms of agents (transactions, customers, products, parts, vehicles, etc.), processes (sequences of operations typically involving queues, delays, resource utilization), and resources.
- The processes are specified in the form of *flowcharts* - a widely adopted graphical representation used in many areas: manufacturing, call centers, business processes, logistics, healthcare, etc. Flowcharts are constructed from Process Modeling Library objects.

- ① The *Process Modeling Library* palette opens by default. The palette contains library objects. Now we can add library objects from this palette onto the graphical diagram of the *Main* agent type.



Factory. Phase 1. Step 5

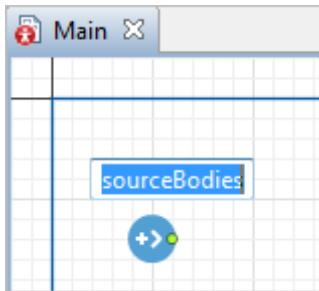


Add **Source** object. In our model it will generate washing machine bodies.

Source

- **Source** object generates agents. It usually acts as a starting point in the process model.

- ① Drag the **Source** element from the **Palette** onto the graphical diagram. Drag'n'drop is the common way of adding palette elements onto your diagram.
- ② Right after dragging an element onto the graphical editor, you can see that its name is selected in the in-place editor.

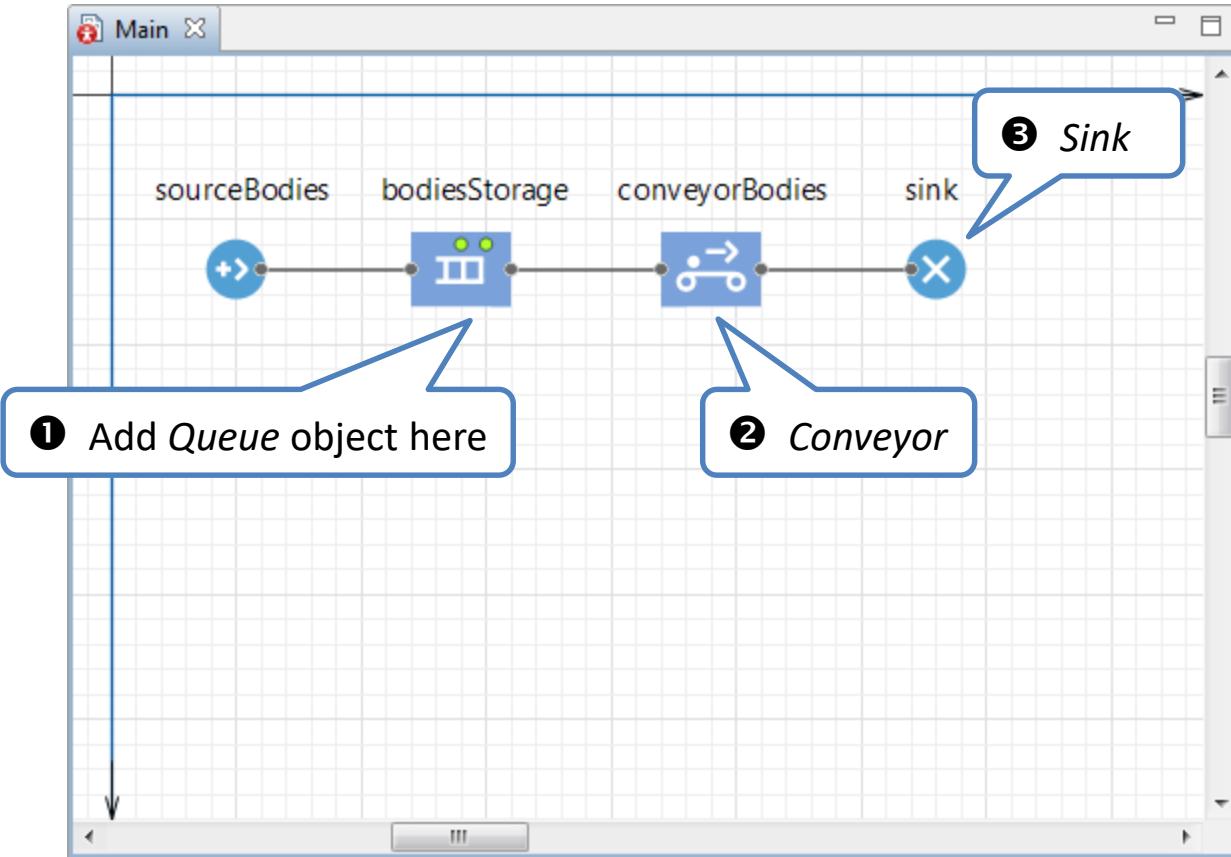


Type here a new name of this object: *sourceBodies*.

Name model elements exactly as we do since later on you will refer to these elements by their names.



Factory. Phase 1. Step 6



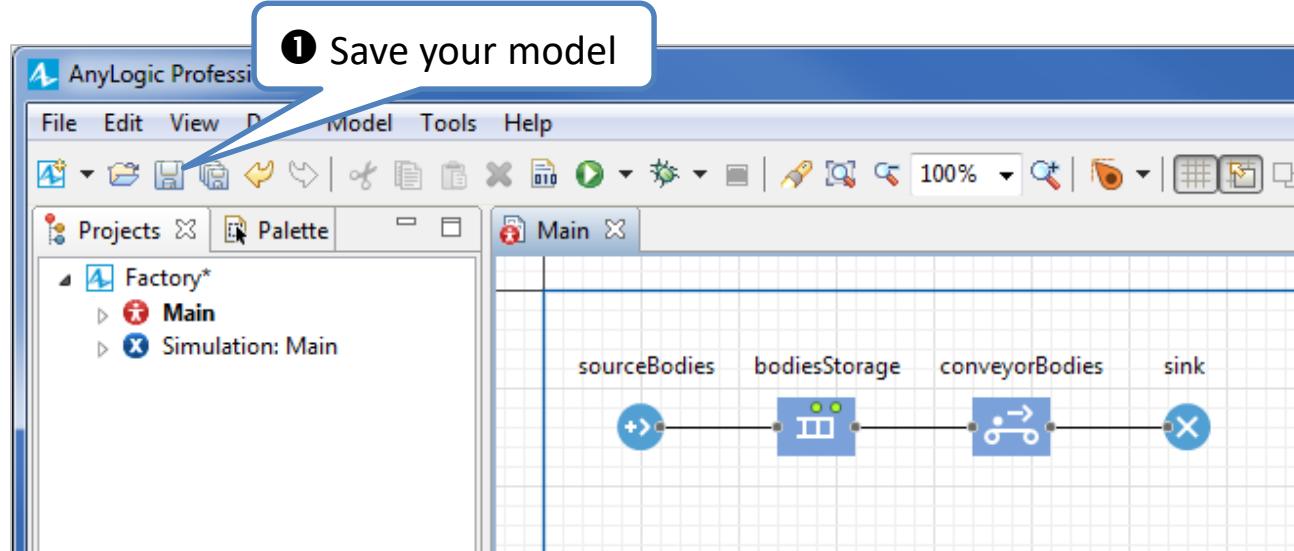
Continue constructing the flowchart by adding more Process Modeling Library objects. When adding flowchart blocks, you will see that the closest ports get automatically connected

- ① Add **Queue** object. Name it *bodiesStorage*. We add a queue to store the arrived bodies until they can be taken on the conveyor.
 - ② Add **Conveyor** object. Name it *conveyorBodies*. In our model it will represent a conveyor transporting washing machine bodies.
 - ③ Add **Sink**.
-

- **Queue** object models a queue (a buffer) of agents waiting to be accepted by the next object(s) in the process flow, or a storage of the agents.
- **Conveyor** moves the agents along a path at a given speed preserving a minimum space between them.
- **Sink** object disposes agents. It is usually an end point in a flowchart.



Factory. Phase 1. Step 7

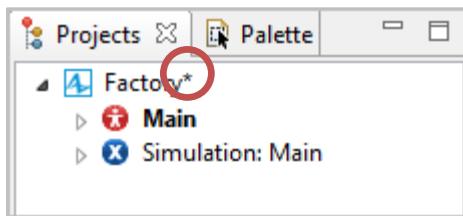


Finally your flowchart blocks should get connected as shown on the slide.

Defining a route for agent flow

- You define a route for agents by connecting the ports of flowchart blocks. You may notice that right port is connected to the left port of the successor block. The reason is that Process Modeling Library blocks have *input* and *output ports*. Input ports are located on the left side of the icon, while outputs are on the right. You may only connect output ports to input ports.
- By connecting the flowchart objects you define the path for agents passing through this flowchart. Once having entered the *Source* object, the agent is passed on to the flowchart object connected to *Source*'s output port and so on.

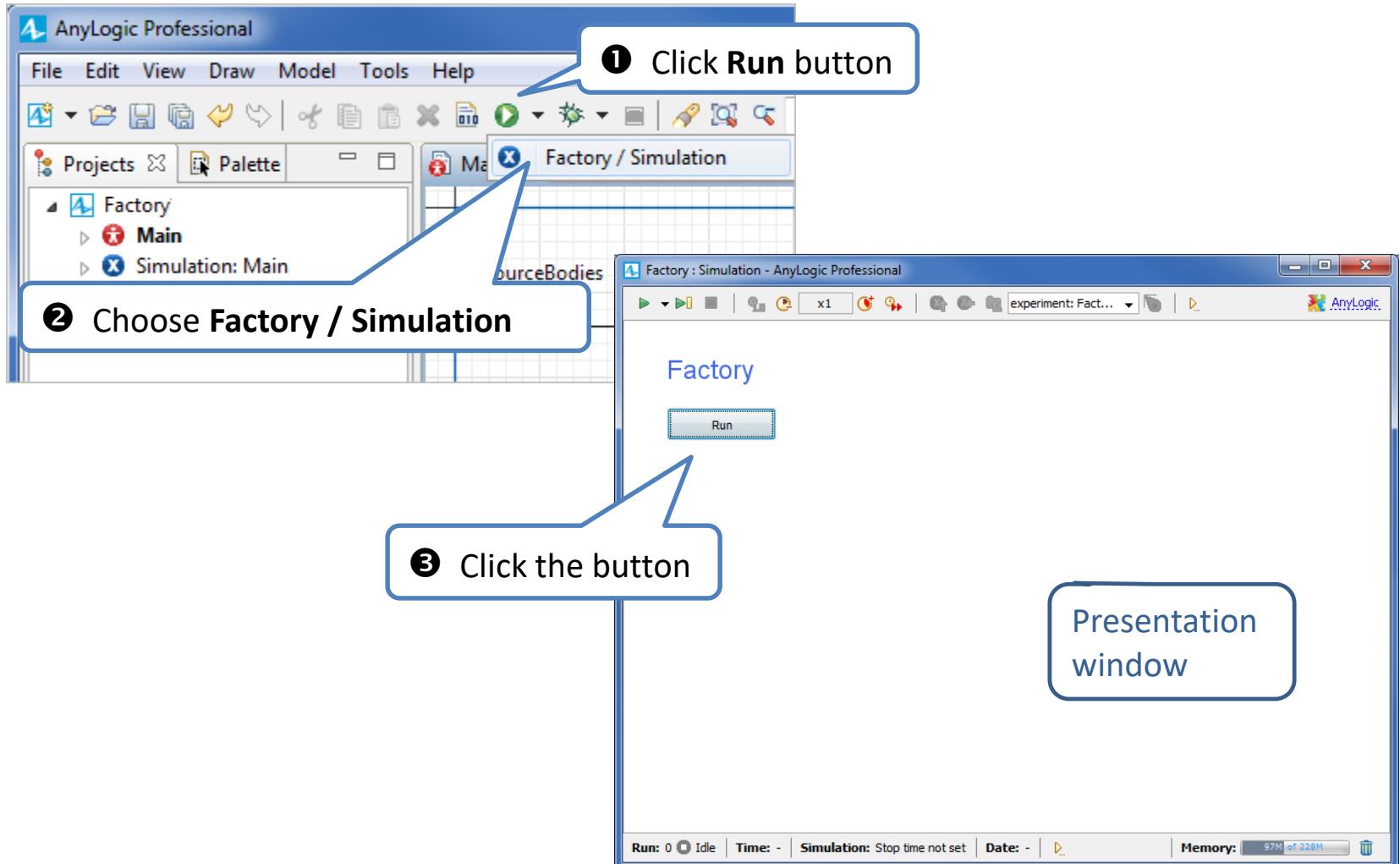
② You may notice an asterisk near the model item in the **Projects** tree:



It means that you have unsaved changes in this model. Save your model by clicking the **Save** toolbar button.



Factory. Phase 1. Step 8



Now we have finished building this very simple model. Run the model and observe its behavior.

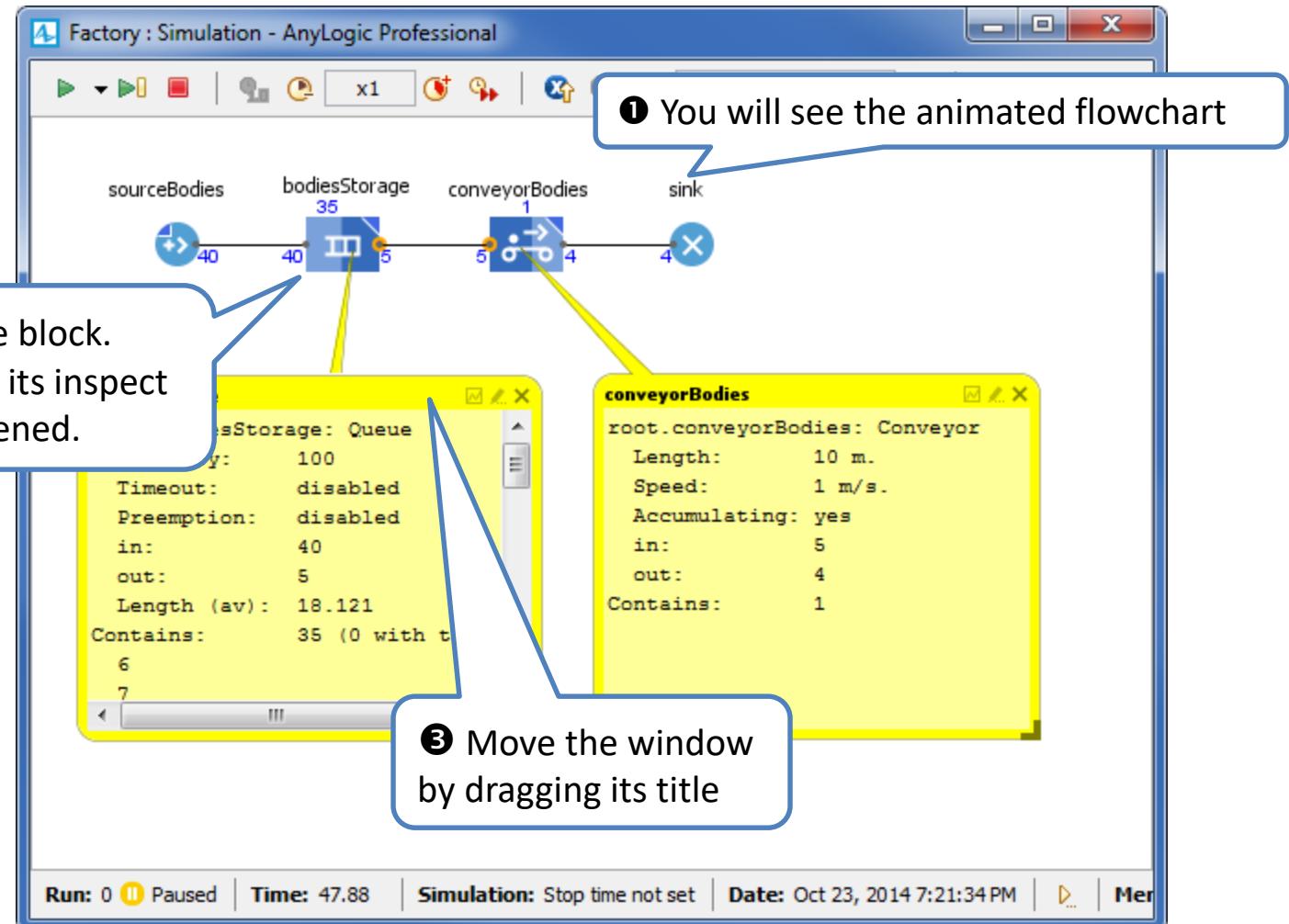
① - ② There can be several models opened in the workspace, each one having a number of experiments. So you need to tell AnyLogic, what particular experiment you want to run.

Having started the model, you will see the *presentation window*. It displays the presentation of the launched experiment (*Simulation*).

③ By default the experiment's presentation contains the button **Run**. Clicking this button starts running the model and opens the presentation of the top-level agent *Main*.

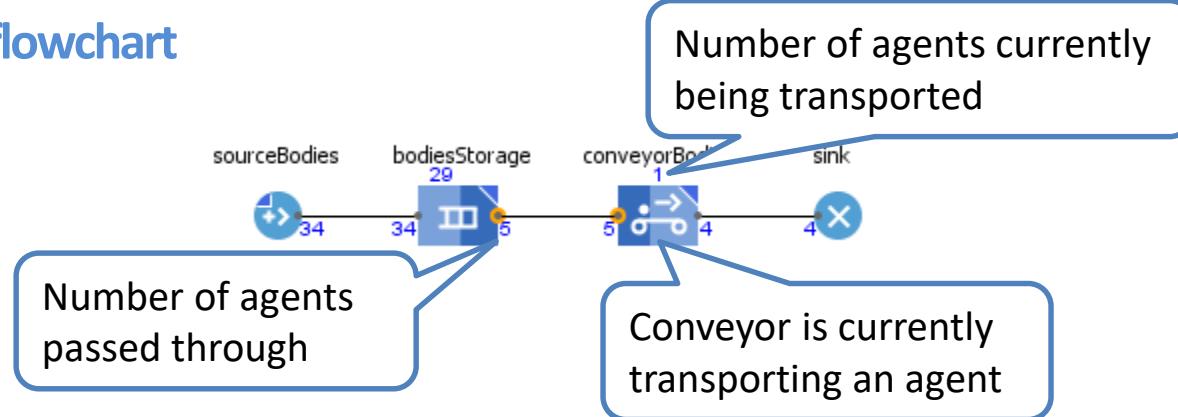


Factory. Phase 1. Step 9

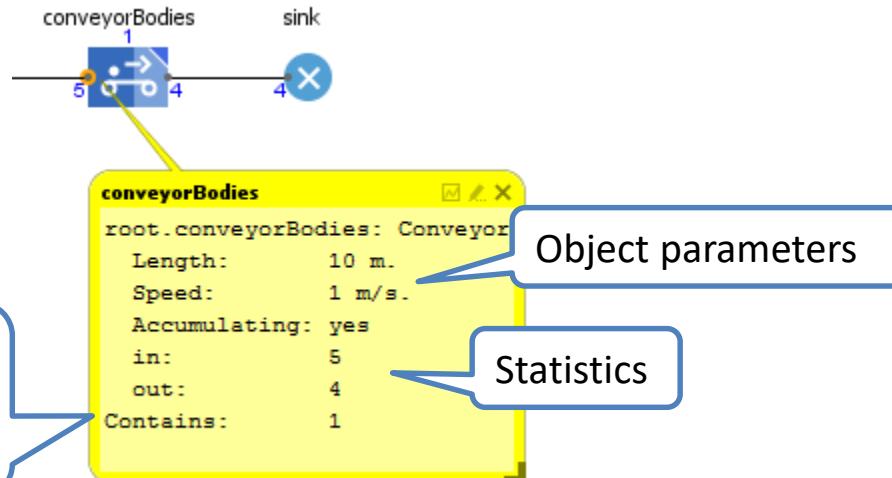


Examine the current states of flowchart objects using the animated flowchart and AnyLogic inspect windows

Animated flowchart



Inspect windows

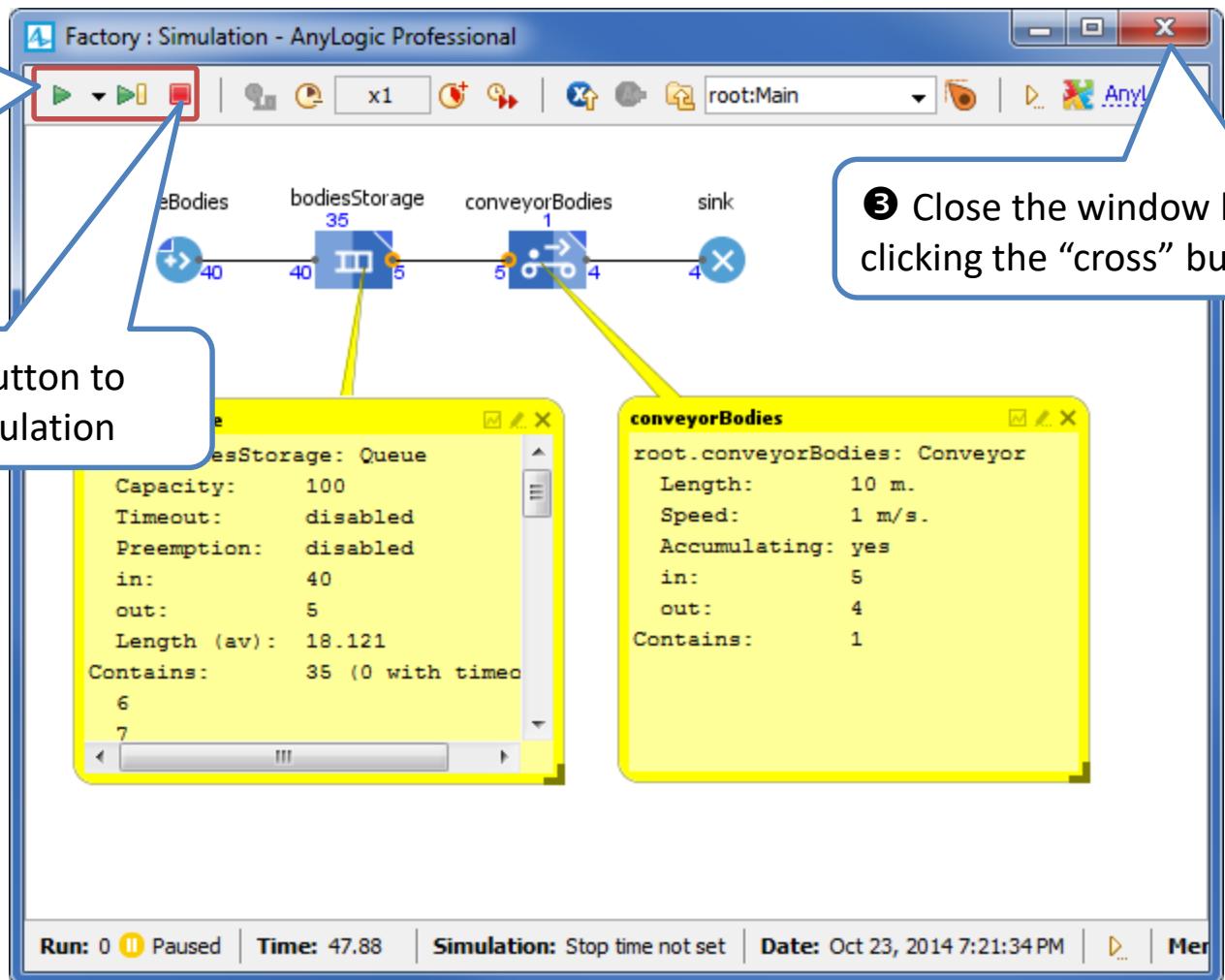


Factory. Phase 1. Step 10

① Control the model simulation using **Run**, **Step** and **Pause** buttons

② Click **Stop** button to stop model simulation

③ Close the window by clicking the “cross” button



Controlling the model execution

You can control the model execution using the toolbar, displayed at the top of AnyLogic presentation window.

▶ Run from the current state

Runs the simulation. Starts the execution, or if the simulation was paused, resumes the simulation from the current state.

▶ Step

Makes the step.

⏸ Pause

Pauses the simulation. You can resume the paused simulation anytime.

⏹ Terminate execution

Terminates the current model execution.



Factory. Phase 1. Questions

1. How can you access AnyLogic example models?
2. What is the difference between left and right ports of Process Modeling Library objects?



Factory. Phase 2

- We have created the simplest discrete event model in the first phase.
- In this phase we will define the units of time and length in the objects of our process flowchart.
- Also, we will add some simple animation of a conveyor and bodies storage zone



Factory. Phase 2. Step 1

The screenshot shows the AnyLogic software interface with the following elements:

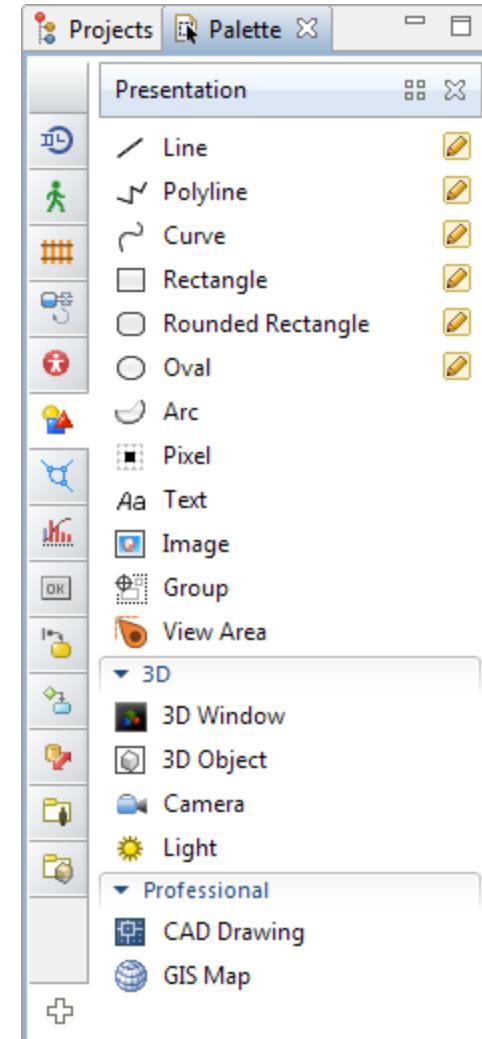
- Presentation palette (left):** A list of drawing tools including Line, Polyline, Curve, Rectangle, Rounded Rectangle, Oval, Arc, Pixel, Text, and Image. The **Image** tool is highlighted with a blue border.
- Main window (right):** A factory floor layout with a conveyor belt system at the top. The conveyor consists of four nodes: a source node (+), a storage node (with three green dots), a body storage node (with a double arrow), and a sink node (X). Below the conveyor is a detailed floor plan with labeled areas: BODY STORAGE, DOOR STORAGE, ASSEMBLY, PACKAGING, and LOADING ZONE. A scale bar indicates 0 to 5 meters. A blue callout box points to the **Image** tool in the palette with the text **① Open the Presentation palette**.
- Diagram area (center):** A large blue callout box contains the text **② Drag Image onto the diagram**, indicating the next step in the process.



Start drawing the model animation with adding a factory layout. We will use a layout from an existing image file.

The dialog window that helps you to browse to the file opens automatically when you drop the *Image* element on the diagram.

Choose the image file to be displayed by this *Image* shape: *factory_layout.png* from *Models\Factory* folder located on your USB disk.

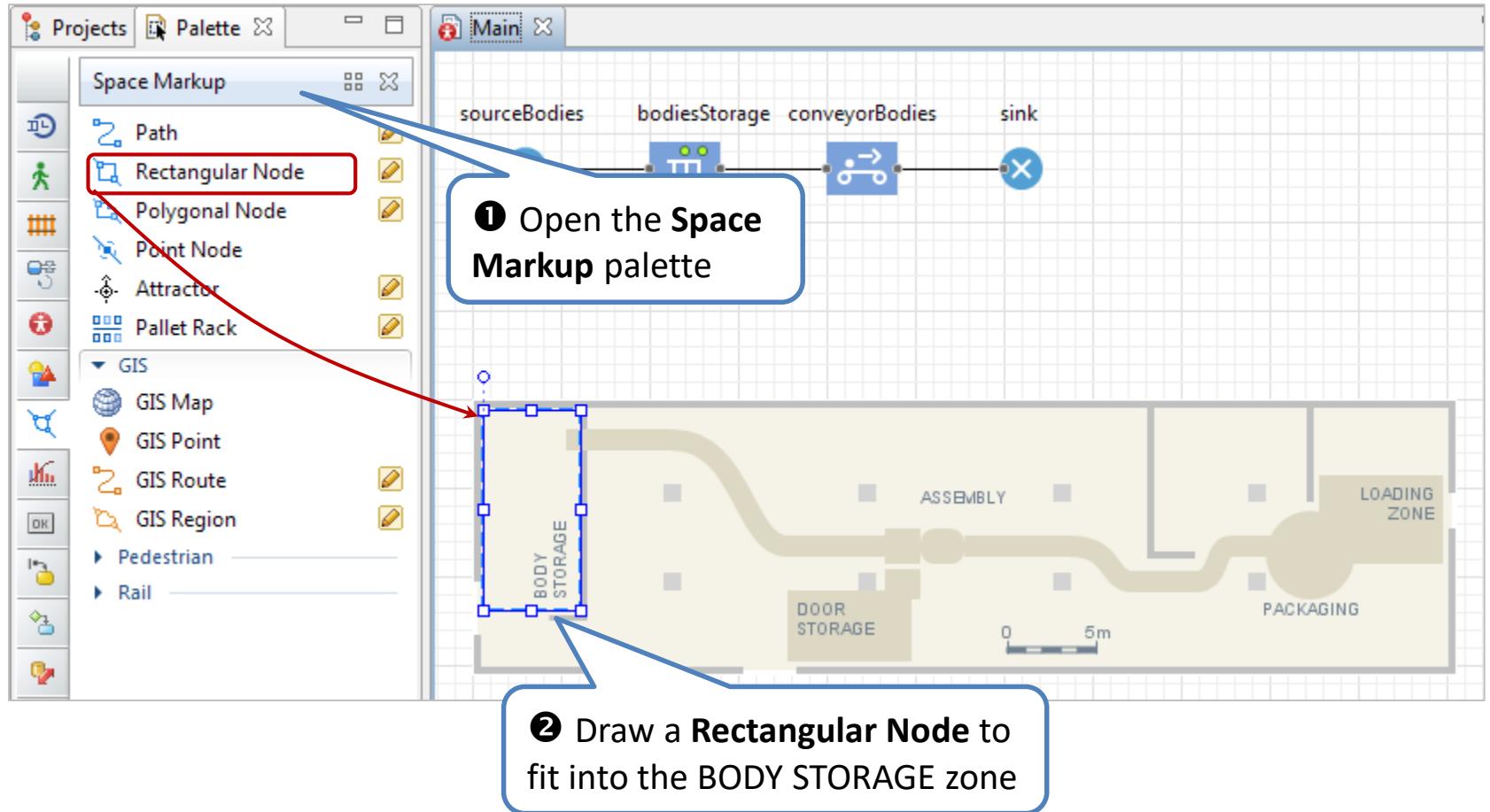


Presentation palette

- The **Presentation** palette contains a set of *geometric shapes*, used for drawing model presentations (rectangle, line, oval, polyline, curve, etc.)



Factory. Phase 2. Step 2



The screenshot shows the AnyLogic software interface with the following elements:

- Projects** tab is selected.
- Palette** tab is open, showing the **Space Markup** palette.
- Rectangular Node** icon is highlighted with a red box and a callout bubble containing the text: **① Open the Space Markup palette**.
- Main** workspace displays a factory layout with various nodes and paths. Labels include: sourceBodies, bodiesStorage, conveyorBodies, sink, BODY STORAGE, ASSEMBLY, DOOR STORAGE, PACKAGING, and LOADING ZONE.
- A blue callout bubble points to a rectangular node drawn in the workspace, containing the text: **② Draw a Rectangular Node to fit into the BODY STORAGE zone**.

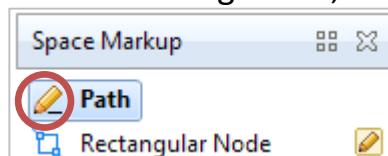


① - ② Draw a rectangular node to represent an area where the washing machine bodies are stored. You may name it *shapeBodyStorage*.

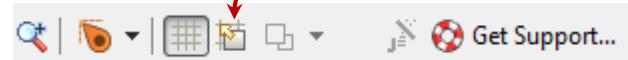
Double-clicking on the *Rectangular Node* element in the **Palette** you activate its drawing mode.

Drawing mode

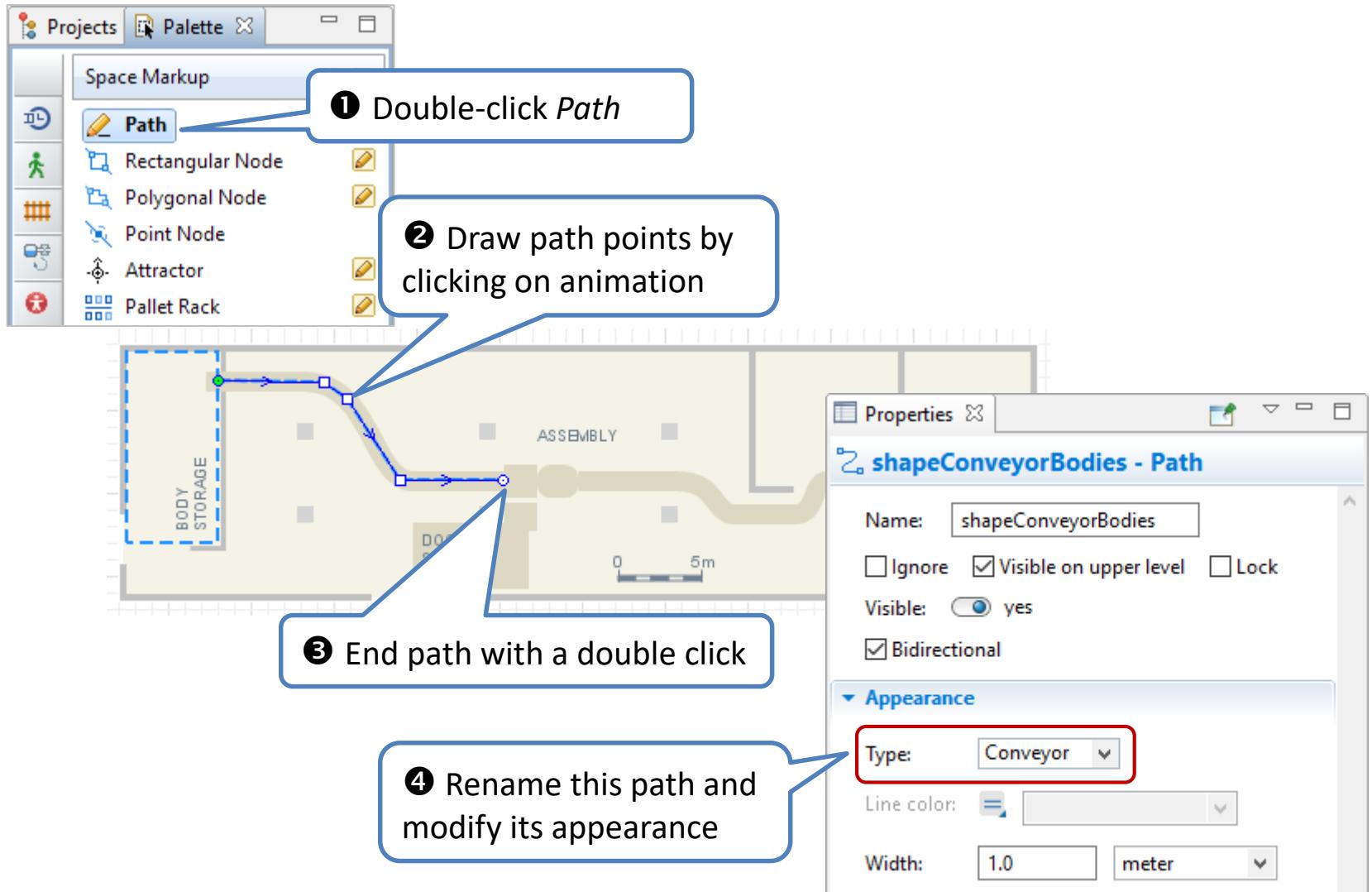
- Some elements (marked in the **Palette** with icons) support *drawing mode* - additional way of adding these elements onto the diagram in addition to common drag'n'dropping.
- In the drawing mode you can draw shapes of the required form at once, e.g. you can draw an oval or a rectangle of the required size or draw a polyline point by point, putting points where you need.
- To activate the drawing mode, double-click the element in the **Palette**. The element icon should turn to :



To make drawing easier, you can hold the Alt key while drawing or turn off the **grid button** in the toolbar:



Factory. Phase 2. Step 3



Draw a path to represent a conveyor.

- ❶ Double-clicking on the *Path* element in the **Palette** you activate its drawing mode.
- ❷ - ❸ Draw a path from left to right as shown on the slide.
- ❹ Name the path *shapeConveyorBodies*.

Animating Process Modeling Library blocks

- Flowchart block that performs some operation over agents (resources, transporters) can animate its activity by animating agents that are being handled by this block. AnyLogic suggests an easy way of doing this: you draw a space markup shape on the graphical diagram—say, a node or a path—and specify this shape as object's **Agent location**. The object then uses it as a guideline to animate the agents.
- Inside nodes, agents may appear at random positions, or arranged in 2D array that fits the given node. You can also define the exact positions using the attractors.



Factory. Phase 2. Step 4

The diagram shows a simulation model in AnyLogic. A sequence of components is connected: a sourceBodies component (represented by a blue circle with a plus sign), a bodiesStorage component (represented by a blue rectangle with three green dots), a conveyorBodies component (represented by a blue rectangle with a double-headed arrow), and finally a sink component (represented by a blue circle with a minus sign). A callout box labeled "① Select sourceBodies" points to the sourceBodies component. Another callout box labeled "② Change Arrival rate: 1 per minute" points to the "Arrival rate" field in the properties panel.

Main

sourceBodies bodiesStorage conveyorBodies sink

Properties

sourceBodies - Source

Name: sourceBodies Show name

Ignore

Arrivals defined by: Rate

Arrival rate: 1 per minute

Multiple agents per arrival:

① Select sourceBodies

② Change Arrival rate:
1 per minute



Define the parameters of these library objects we have added to the agent's diagram.

- ① Select the object *sourceBodies* to change its parameters in the **Properties** panel.
- ② Let one washing machine body arrive per one minute.



Factory. Phase 2. Step 5

The screenshot shows a simulation model in the 'Main' workspace and its properties in the 'Properties' window.

Main Workspace: The model consists of four components: 'sourceBodies' (a blue circle with a '+' sign), 'bodiesStorage' (a queue node with three green dots), 'conveyorBodies' (a conveyor node with two arrows), and 'sink' (a blue circle with an 'X'). Arrows connect 'sourceBodies' to 'bodiesStorage', 'bodiesStorage' to 'conveyorBodies', and 'conveyorBodies' to 'sink'.

Properties Window: The 'bodiesStorage - Queue' properties are displayed.

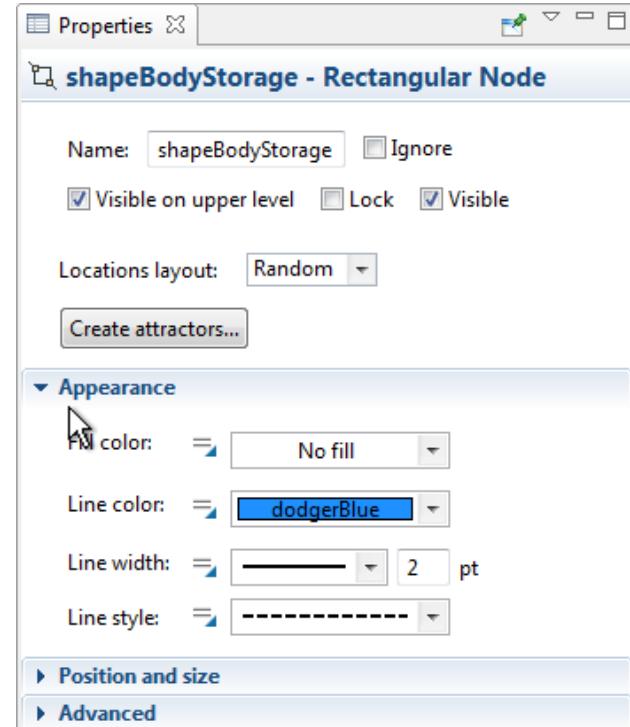
- ① Select the *bodiesStorage***: A callout points to the 'bodiesStorage' queue node in the workspace.
- ② Select the checkbox for **Maximum capacity****: A callout points to the 'Maximum capacity:' checkbox in the properties window, which is checked.
- ③ Choose the node you have drawn earlier as the **Agent location****: A callout points to the 'Agent location:' dropdown menu, which contains the node 'shapeBodyStorage'.



- ① Select the *bodiesStorage* object in the graphical editor.
- ② Remove any restrictions to the number of bodies in the queue by selecting a checkbox next to the option **Maximum capacity**.
- ③ Define the **Agent location** for this queue.

Properties view

- The **Properties** view is a context-sensitive view that displays the properties of the currently selected model element(s).
- To modify properties of some element, first select it (by clicking it in the graphical editor or in the **Projects** view) and then modify the required properties in the **Properties** view.
- The name and the type of the currently selected element are shown at the top of the view.
- The **Properties** view contains several sections. To expand/collapse a section, just click on its name.



Factory. Phase 2. Step 6

The screenshot shows a simulation model in AnyLogic. On the left, there is a process flow diagram with three components: 'bodiesStorage' (represented by a blue rectangle with two green dots), 'conveyorBodies' (represented by a blue rectangle with a white gear icon and a right-pointing arrow), and 'sink' (represented by a blue circle with a red 'X'). A purple line connects 'bodiesStorage' to 'conveyorBodies'. A callout bubble with the text '① Select conveyorBodies' points to the 'conveyorBodies' component.

On the right, the 'Properties' window is open for the 'conveyorBodies' component. The title bar says 'conveyorBodies - Conveyor'. The properties listed are:

- Name: conveyorBodies Show name Ignore
- Length is:
= Specified explicitly
 Defined by path
- Length: = 10 meter
- Speed: = 0.02 meters per second
- Accumulating: =
- Agent location: =
- Grab agent from prev.conveyor: = Smoothly, matching speed
- Change agent length: =
Agent length: = 1 meter

A callout bubble with the text '② Define conveyor's speed here' points to the 'Speed' property. Another callout bubble with the text '③ Select new Agent length' points to the 'Agent length' property, which is highlighted with a red border.



Now change the parameters of *conveyorBodies* to match the parameters of the preceding objects of the flowchart.

- ① Select the *conveyorBodies* object in the graphical editor and go to its properties.

The conveyor parameter **Speed** supports several units that you can use to define the speed. We will define the conveyor speed in the standard units here: *meters per second*.

- ② Enter new **Speed** parameter value: *0.02 meters per second*.
- ③ Let us also change the **Agent length** to *1 meter*. The **Conveyor** object will use this value when placing bodies on the conveyor animation which we will define in the next step.



Factory. Phase 2. Step 7

The screenshot shows the AnyLogic simulation environment with two main windows: 'Main' and 'Properties'.

Main Window: Displays a flowchart with nodes: 'sourceBodies' (blue circle with '+'), 'bodiesStorage' (blue rectangle with three dots), 'conveyorBodies' (blue rectangle with a gear and arrow), and 'sink' (blue circle with a cross). A blue box highlights the 'conveyorBodies' node. A callout bubble points to it with the text: ① Set the animation shape for the conveyor.

Properties Window: Shows the properties for the 'conveyorBodies' node.

- Name:** conveyorBodies
- Length is:** Defined by path
- Speed:** 0.02 meters per second
- Accumulating:**
- Agent location:** [empty field]

A blue box highlights the 'Defined by path' radio button. A callout bubble points to it with the text: ④ Set the length to be defined by path.

Graphical Editor: Shows a conveyor belt icon labeled 'shapeConveyorBodies'. The conveyor has a complex, zigzagging path. A blue box highlights the conveyor icon. A callout bubble points to it with the text: ③ You will see only the shapes of the compatible types highlighted. Select the shape by clicking it in the graphical editor.

Bottom Right Callout: Points to the 'sink' node in the Main window with the text: ② Click the button.



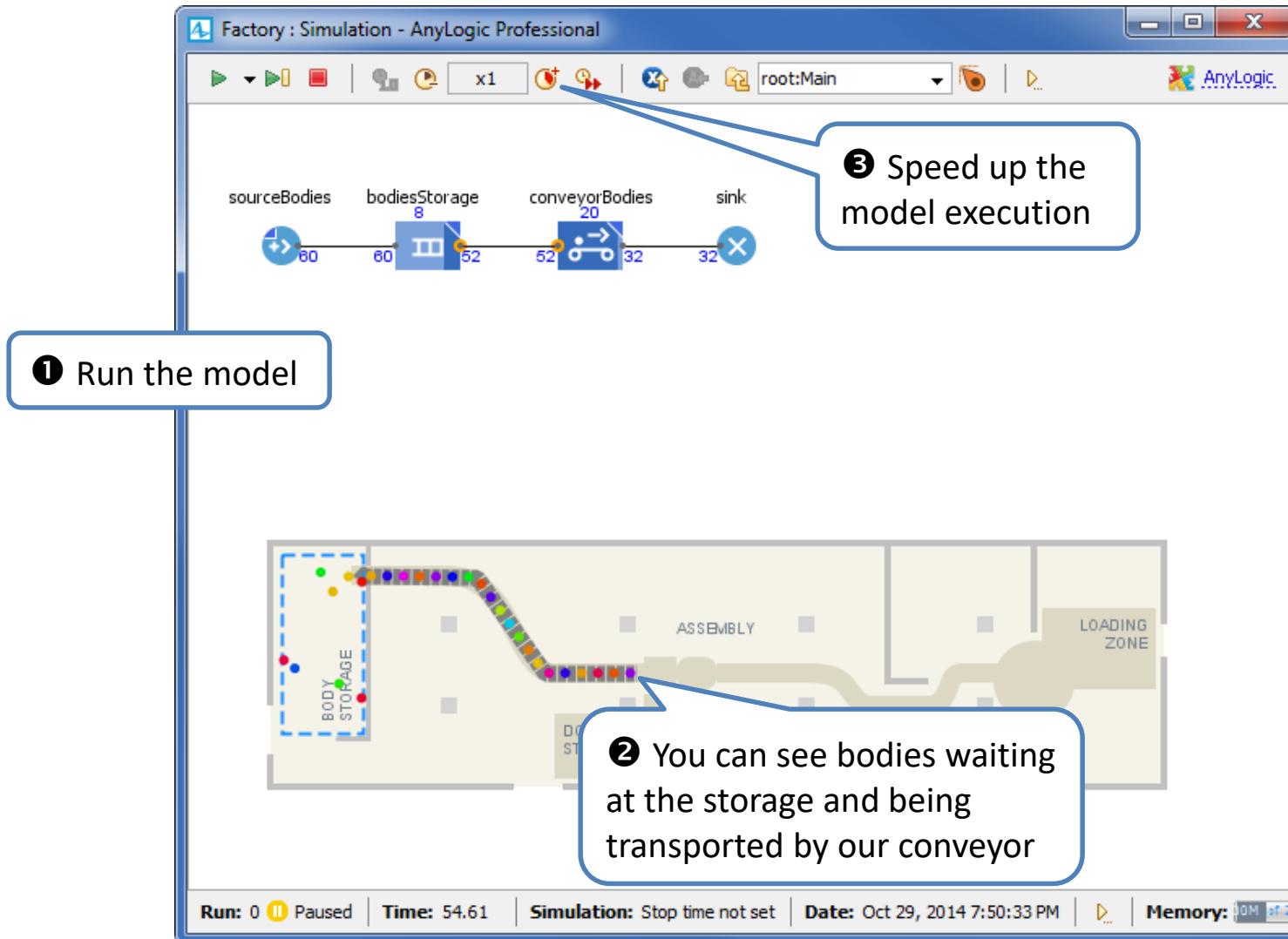
Tell the *conveyorBodies* object that the just drawn path should be used to animate agents currently being transported by this conveyor.

To make a reference to a graphical object (such as a path, a node, an area) in a logical object (such as *queue* or *delay*) you simply click the object in the editor (❸) after you click the selection button  (❷).

❹ Let us also define the conveyor length *by path* – its animation that we have drawn with a space markup element. Its actual length will be defined by the layout of the factory floor. You can find the element **Scale** above the X-axis on every agent diagram. It defines the meters to pixels scale for the presentation figures that this agent contains.



Factory. Phase 2. Step 8



Run the model and observe its animation.

- ③ Adjust the model execution speed by clicking *Slow down* and *Speed up* toolbar buttons.

Adjusting the model execution speed

- AnyLogic model can be run either in *real time* or *virtual time mode*.
- In *real time mode*, the mapping of AnyLogic model time to the real time is made, i.e. you specify how many model time units one second takes. It is frequently needed when you want your animation to appear life-like.
- In *virtual time mode*, the model runs at its maximum speed and no mapping is made between model time units and seconds of astronomical time. This time mode is useful when you need to simulate your model for a long period of time.
- In *real time mode*, you can increase or decrease model simulation speed by changing the model *simulation speed scale*. x2 means that model is run twice as fast as the specified model speed, etc.
- Control the model execution speed using the **Time scale** toolbar:



Factory. Phase 2. Questions

1. Which element you may use to draw the place for animation of agents being inside a Process Modeling Library block?
 - (a) Rectangle
 - (b) Rectangular Node
 - (c) Rectangular Area
2. Which space markup shape would you use to draw animation for:

1. Cinema hall	(a) Rectangular node, Arranged locations layout
2. ATM	(b) Rectangular node, Attractors
3. Bar with seats around the tables	(c) Point node
3. In 1:1 real time execution mode, what time will it take to simulate 100 model time units?

(a) 100	(b) 1	(c) 100 or more
---------	-------	-----------------

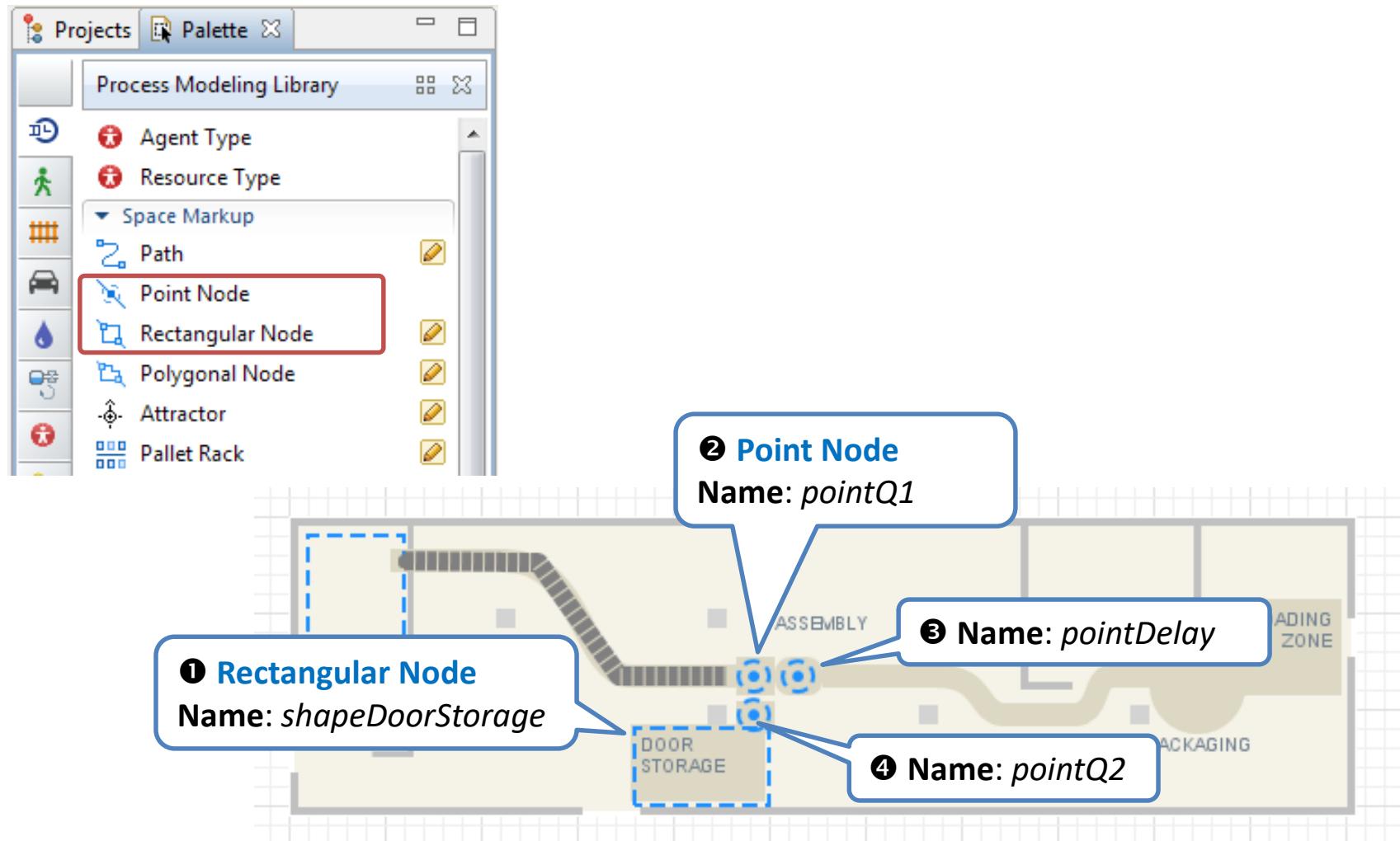


Factory. Phase 3

- In this phase we will continue developing the model. We will:
 - Add a source of washing machine doors and downstream conveyor leading to an assembly robot.
 - Add the assembly robot. The robot finishes washing machine production by attaching a door to a body.
 - Modify the model animation by drawing pictures to display washing machine parts.
 - Configure the flowchart objects with actual parameter values: define the conveyor's length, speed, space between parts, etc.



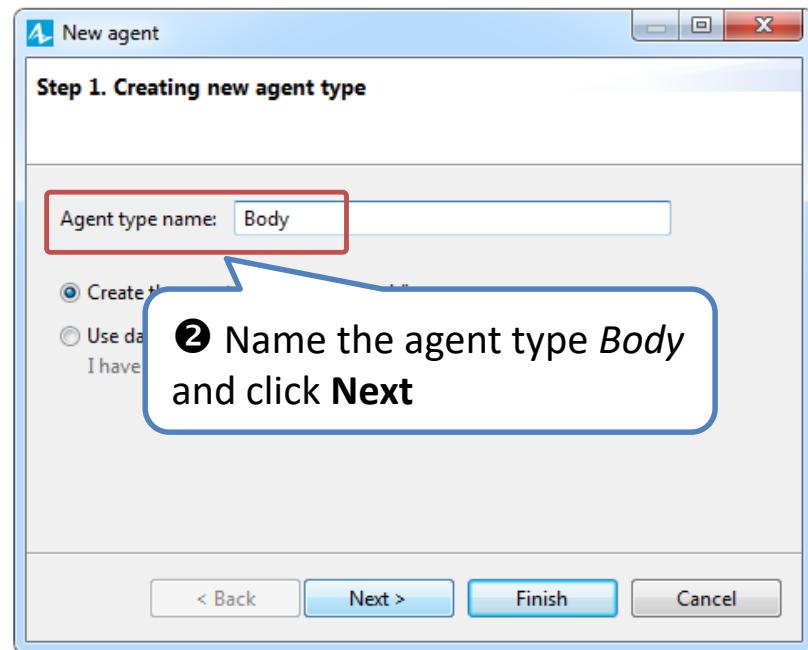
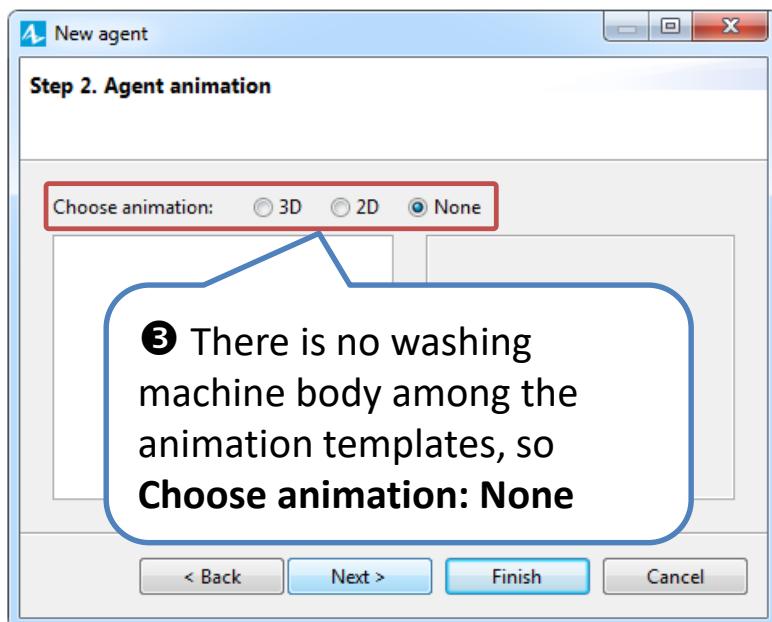
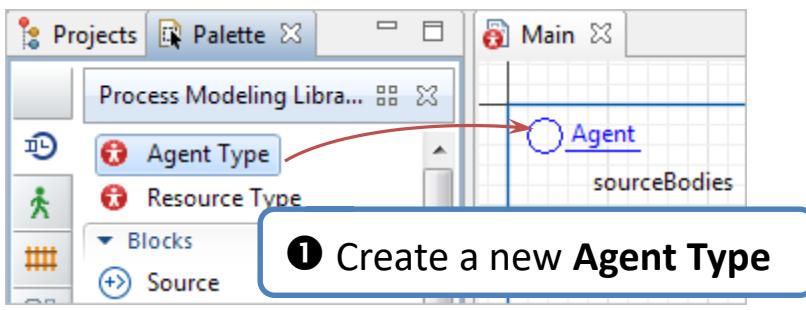
Factory. Phase 3. Step 1



- ① Draw a rectangular node to represent an area where the washing machine doors are stored when entering the shop floor. Resize it to fit into DOOR STORAGE zone. Name it *shapeDoorStorage*.
- ② Draw a point node (*pointQ1*) to represent the zone of the assembly robot where washing machine bodies are placed.
- ③ Draw one more point node (*pointDelay*) to represent robot area where doors are placed.
- ④ The third point node (*pointQ2*) will define the robot area where finished washing machines are placed.



Factory. Phase 3. Step 2



④ On the next page of the wizard you can define agent attributes. We do not need any, so just click **Finish**.



Currently, the parts of washing machines are drawn on the animation as little circles. It is the default animation used for agents in AnyLogic. We want to draw some particular image for each part so we could distinguish them. Let's draw pictures of washing machine parts: body, door and assembled washing machine.

To assign some other animation for agents in your model, you need to create custom agent types.

Custom agent types

- The default agent type allows minimum customization (for example, you can change the color of its default animation). If you need to customize your agents further (e.g. to add properties, functions, or to collect custom statistics), you need to create a custom agent.
- New agent type creation wizard allows you to set animation for the agent and add parameters of various types and set their default values.



Factory. Phase 3. Step 3

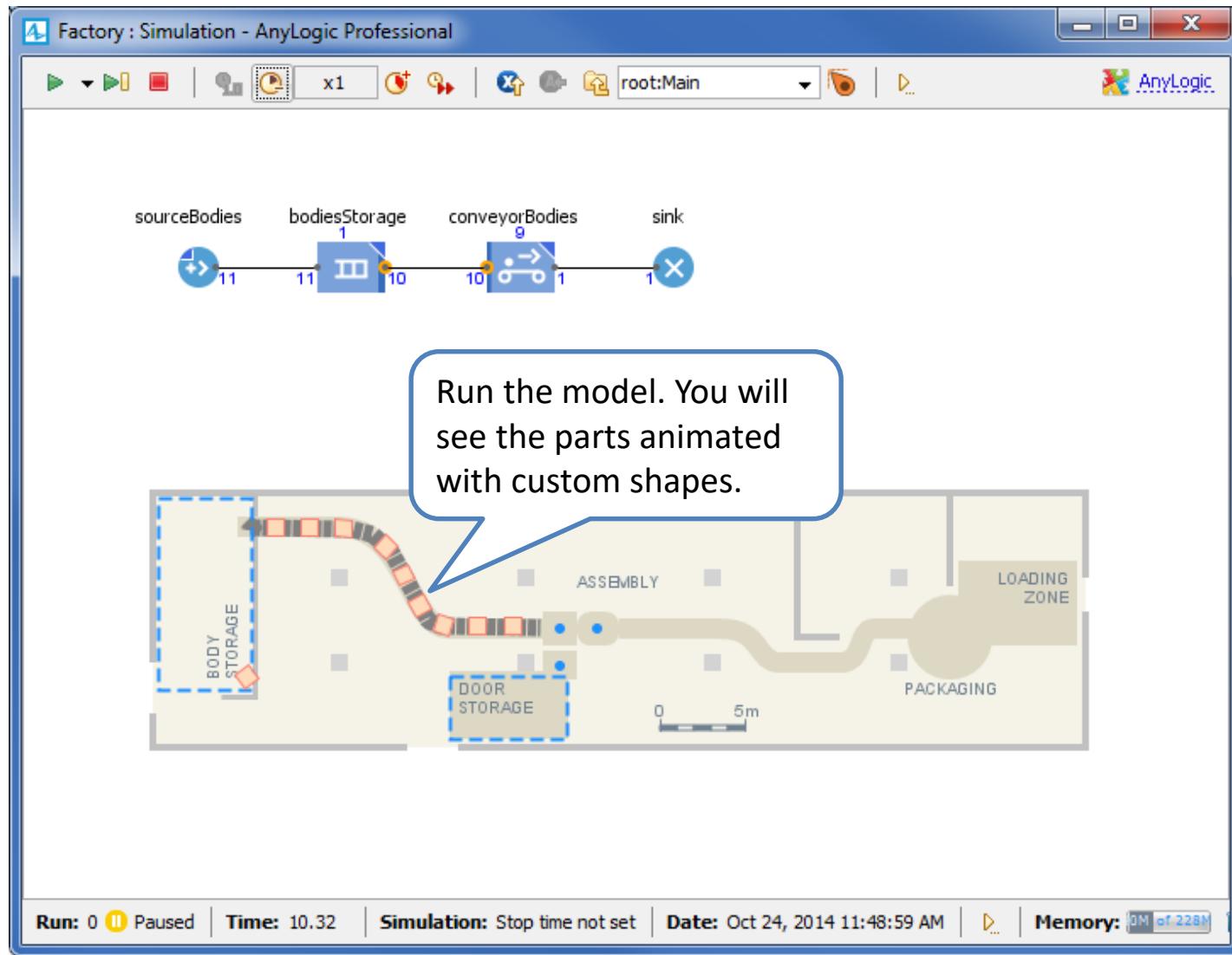
The screenshot shows the AnyLogic modeling environment divided into several panels:

- Projects**: Shows a single project named "Presentation".
- Palette**: Shows drawing tools: Line, Polyline, Curve, Rectangle (selected), and Rounded Rectangle.
- Main**: A workspace where a small orange rectangle is drawn at the origin (0,0).
- Body**: A workspace showing the created agent type "Body".
- Properties**: A panel for configuring a "sourceBodies - Source" block.
 - Name:** sourceBodies Show name
 - Ignore:**
 - Arrivals defined by:** Rate 1
 - Arrival rate:** Rate = 1
 - Multiple agents per arrival:**
 - Limited number of arrivals:**
 - New agent:** Body create a custom type
- Diagram View**: Shows a flowchart with a "sourceBodies" Source block, a "bodiesStorage" Storage block, and a "con" connector.

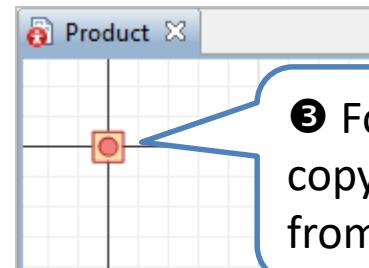
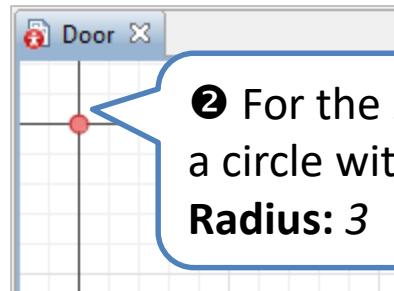
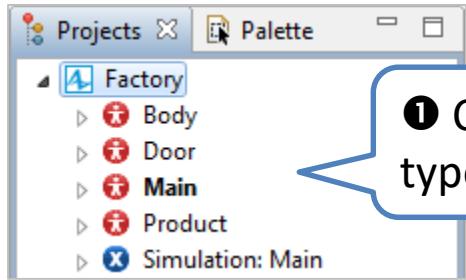
Annotations with callouts explain the steps:

- ① You will see the diagram of the created agent type *Body*
- ② Draw a rectangle 10*10 pixels and place it exactly in the axis origin (point (0,0))
- ③ Choose *Body* as the agent generated by the *Source* block





Factory. Phase 3. Step 4

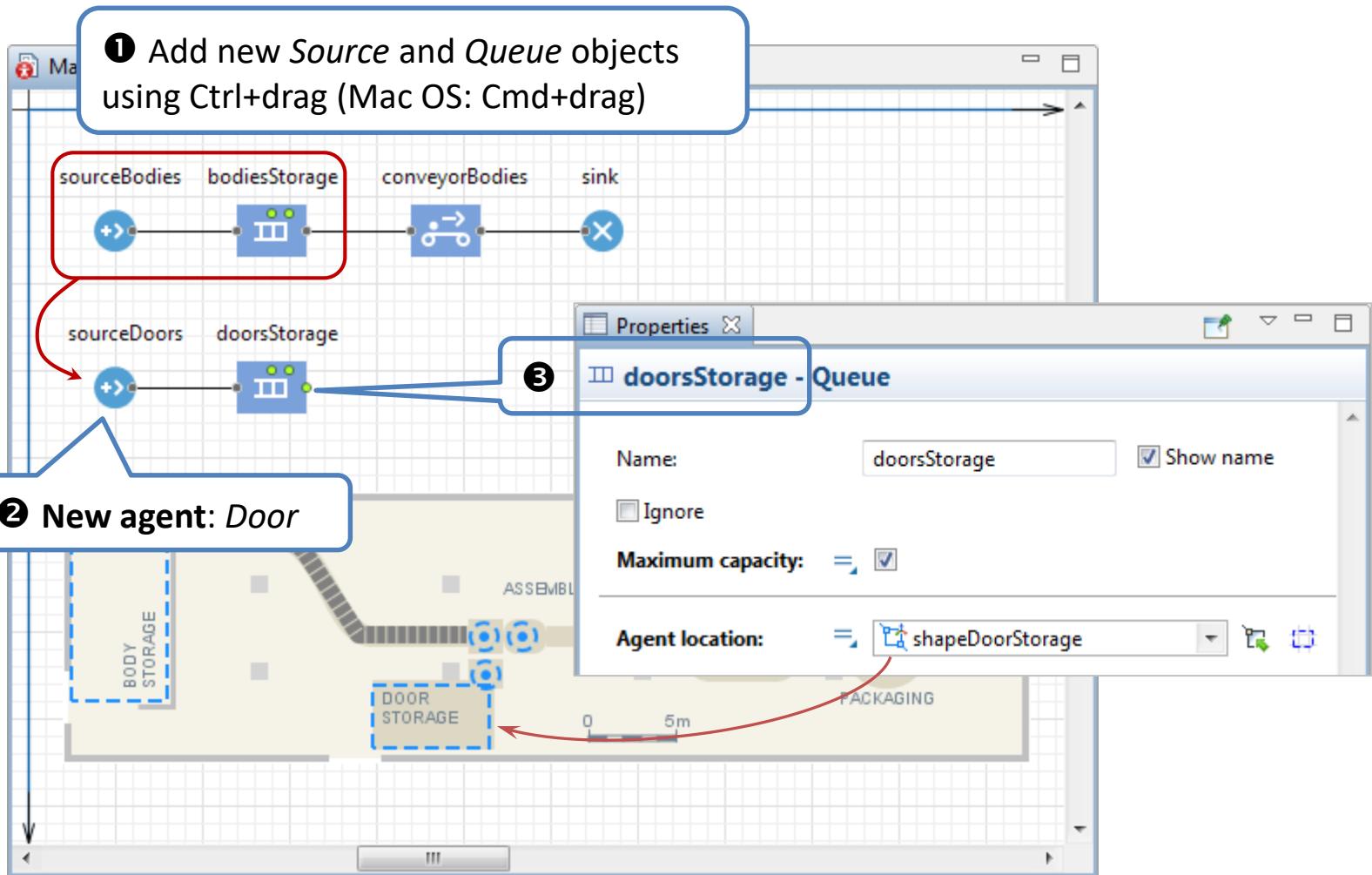


- ③ You can copy and paste model's elements across the diagrams of different agent types using conventional Ctrl+C/Ctrl+V or the context menu options **Copy/Paste**.

AnyLogic also allows copying different model elements from one model to another. This way, you can copy whole agent types together with their internal elements.



Factory. Phase 3. Step 5



Add two more blocks into our flowchart:

- *Source* block to model generation of washing machine doors
 - *Queue* block to model a storage of doors
- ② Name just created *Source* block *sourceDoors*. Choose *Door* as the type of agents generated by this block.
- ③ Name this *Queue* block *doorsStorage* and set its **Agent location**.
-

Selecting multiple elements

There are two ways of selecting several elements simultaneously:

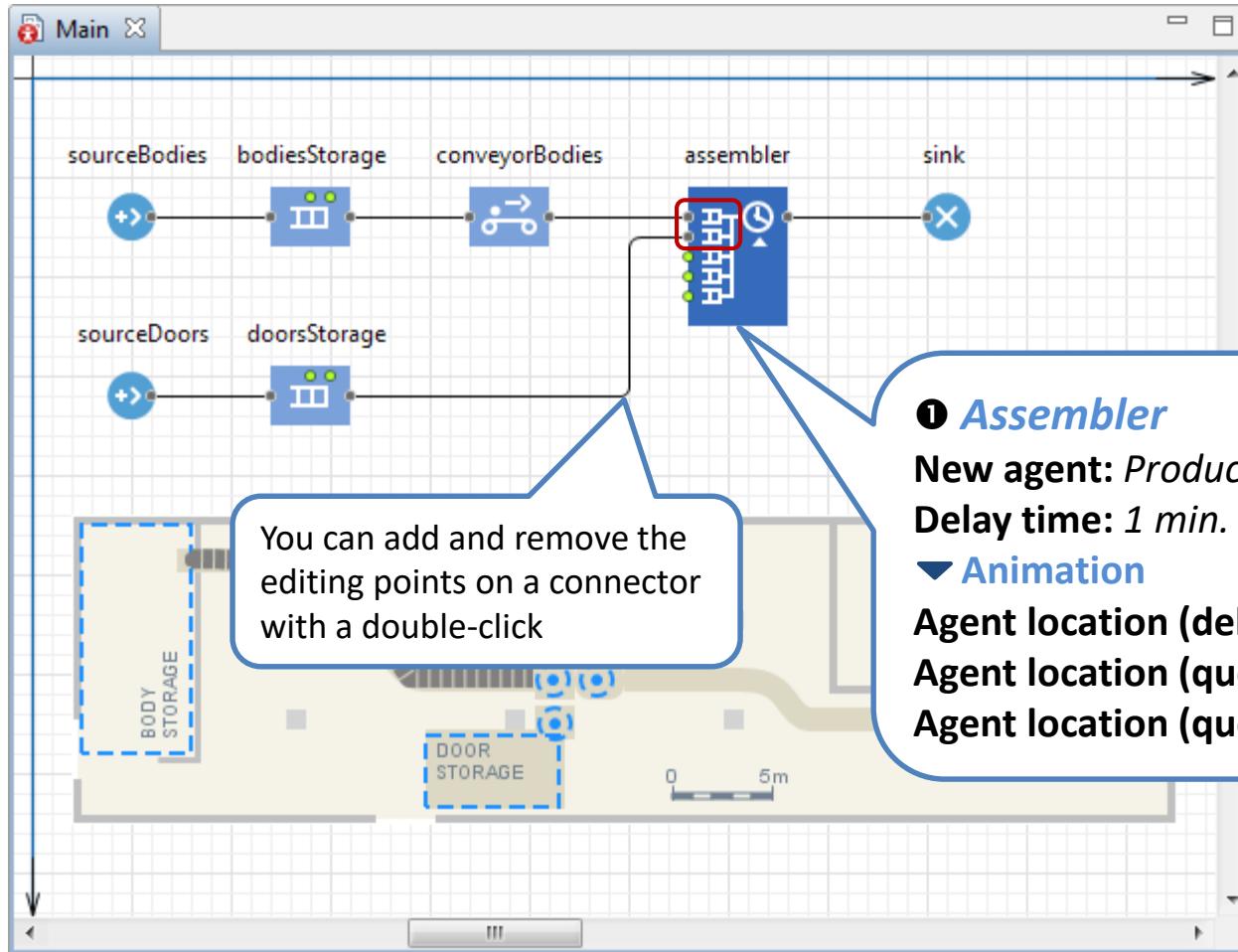
- Drag the selection rectangle around the shapes.
- Successively Ctrl-click the shapes you want to select. (Ctrl-clicking the already selected shape removes it from the selection).

Cloning elements

- Ctrl+dragging (Mac OS: Cmd+dragging) selected elements (either in the graphical editor and or in the **Projects** view) creates copies of these elements.
- Cloned elements have the same properties as the original elements except for the name that is made unique.



Factory. Phase 3. Step 6



① Add **Assembler** block and connect it to other blocks as shown on the slide.

Tell the block that it assembles *Product* agents.

Resources are not needed to perform the operation at the moment.

Set the assembly time to be equal to one minute.

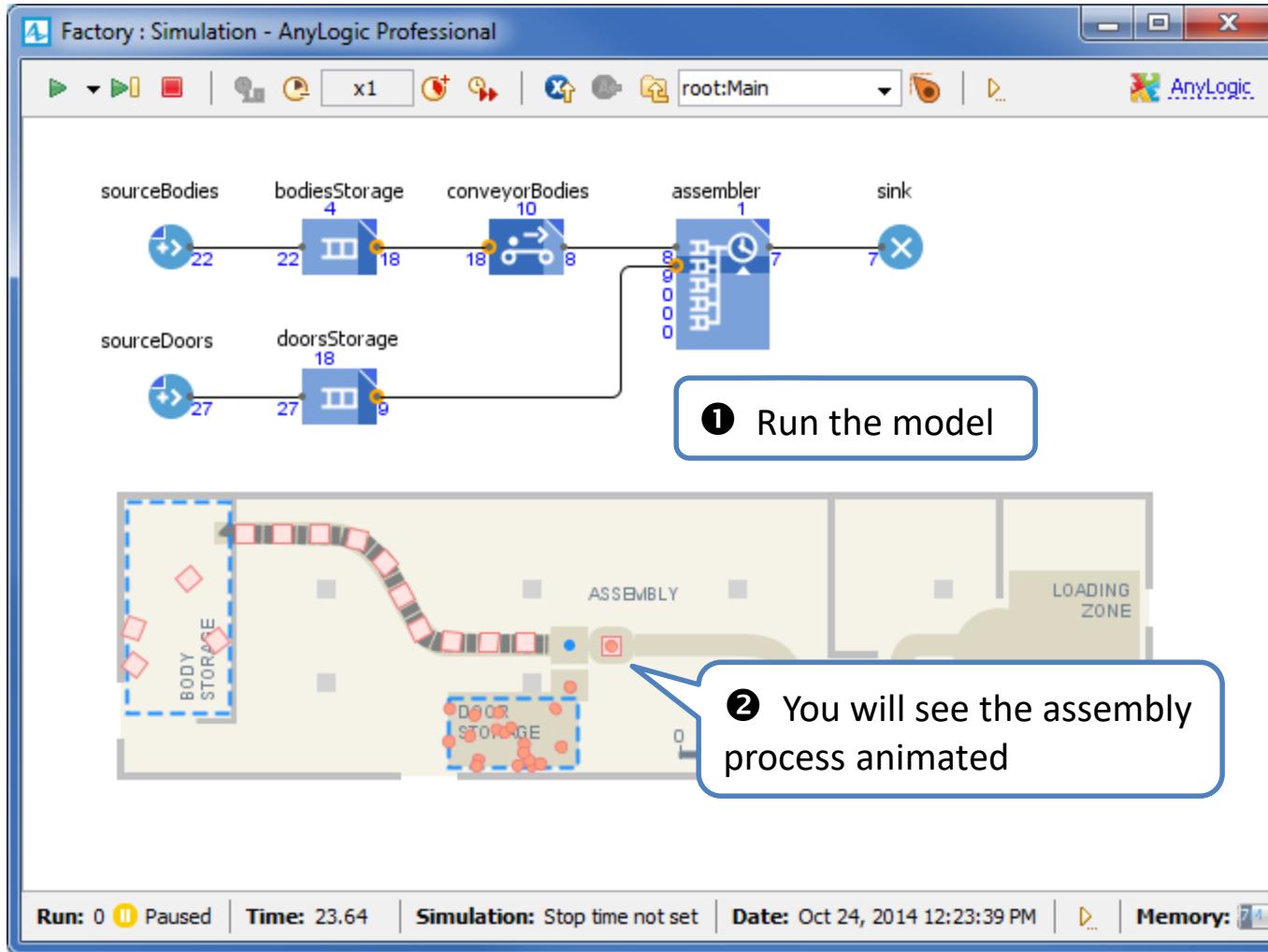
Setup animation shapes for bodies queue (queue 1), doors queue (queue 2) and assembly operation (delay).

Assembler

- **Assembler** block assembles one agent from several ones arriving into its input ports. The number of agents required to perform an assembly is specified individually for each port via object parameters *Quantity 1*, *Quantity 2*, etc. The block waits until each input port has got the required number of agents, then produces a new agent and outputs it.
- The assembly operation takes specified delay time.
- Optionally assembly may be performed using *resources* (they will be covered later.)



Factory. Phase 3. Step 7

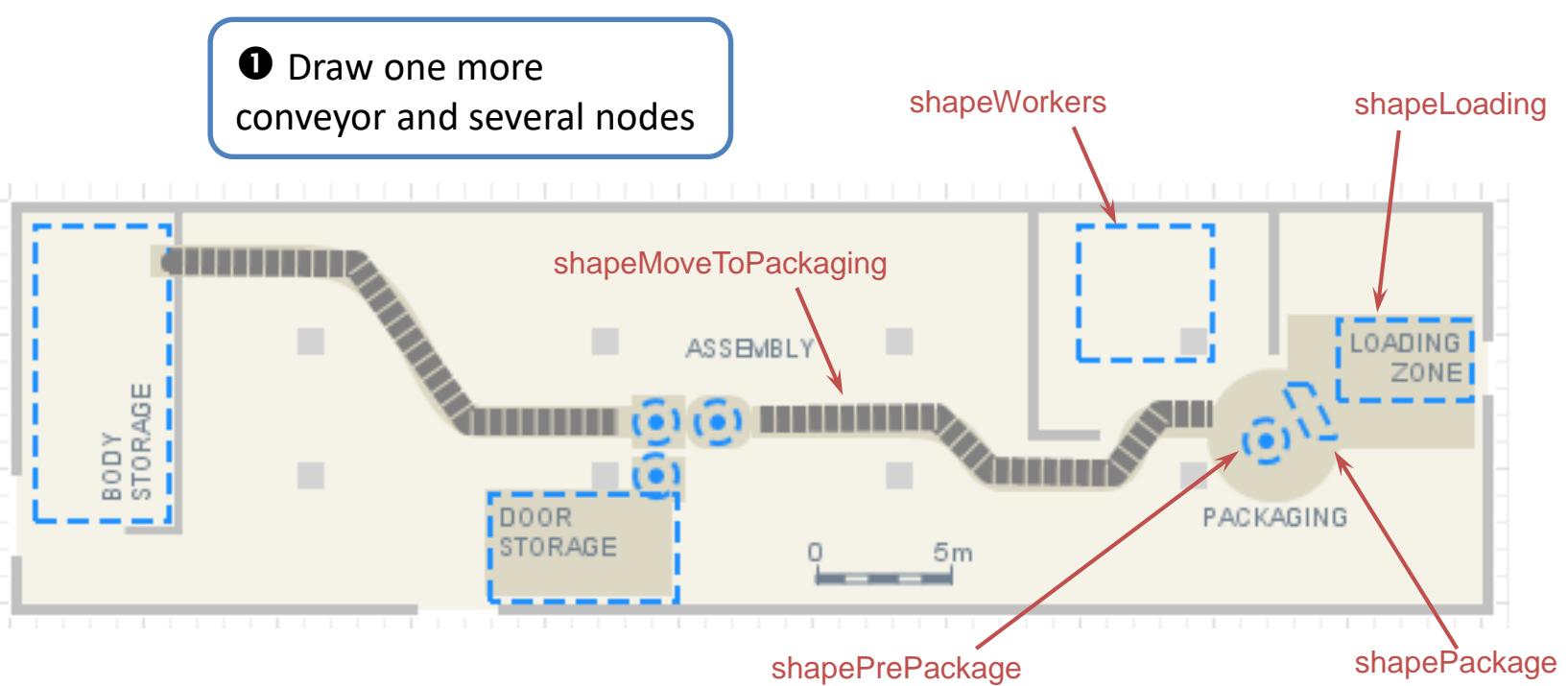


Factory. Phase 4

- Now we want to develop our model further.
- Let's add a packaging line where manufactured products will be packed into boxes. Packaging line comprises upstream conveyor and the packaging zone itself.
- Packed products will be placed into the loading zone. Each 10 boxes will be batched together and shipped from the factory.



Factory. Phase 4. Step 1

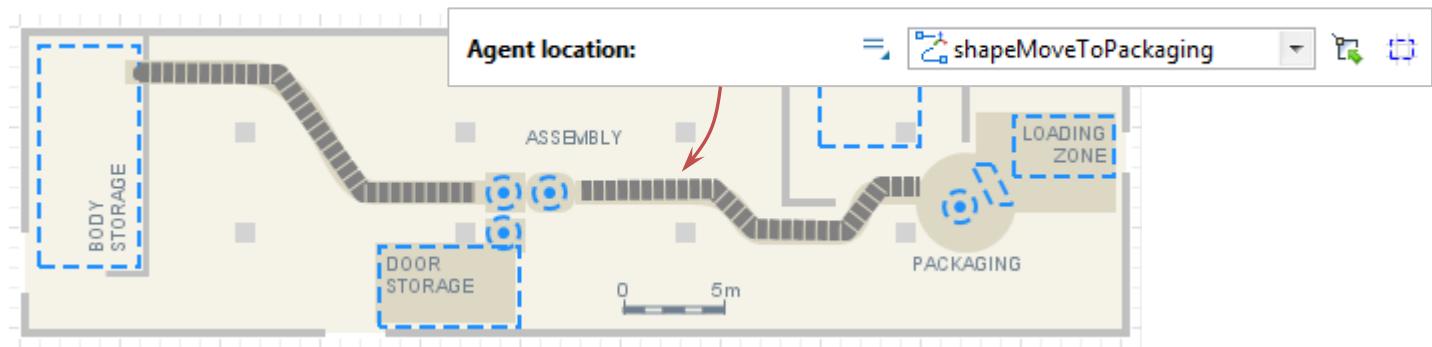
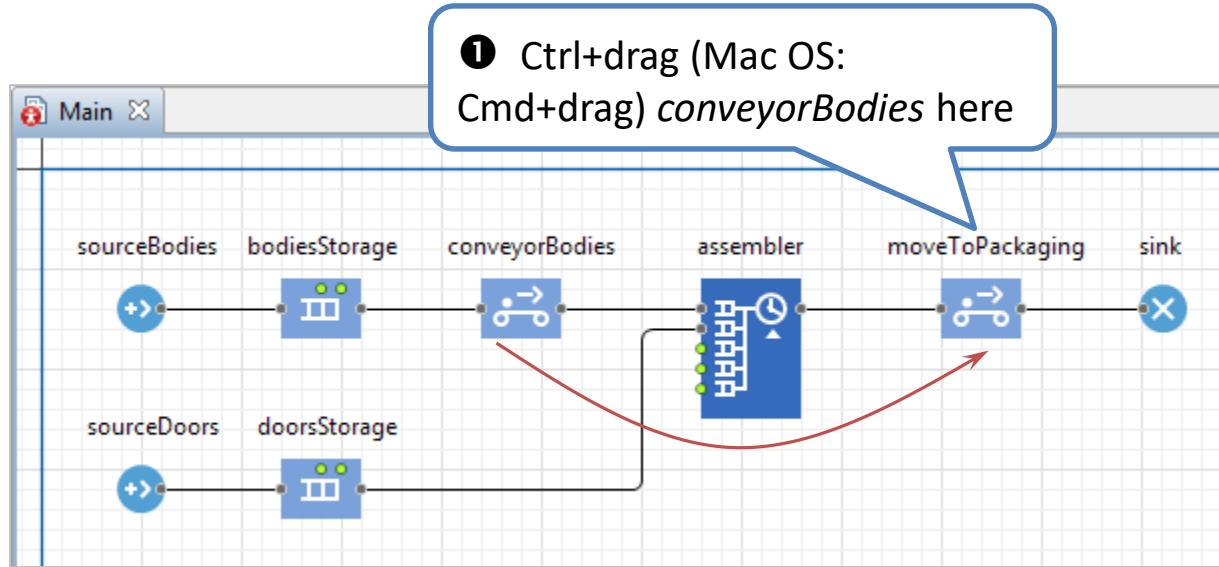


Start from updating the model animation.

- ① Draw five more space markup elements to represent a conveyor going to the packaging zone, the packaging zone itself and the loading zone. Name them as shown on the slide.

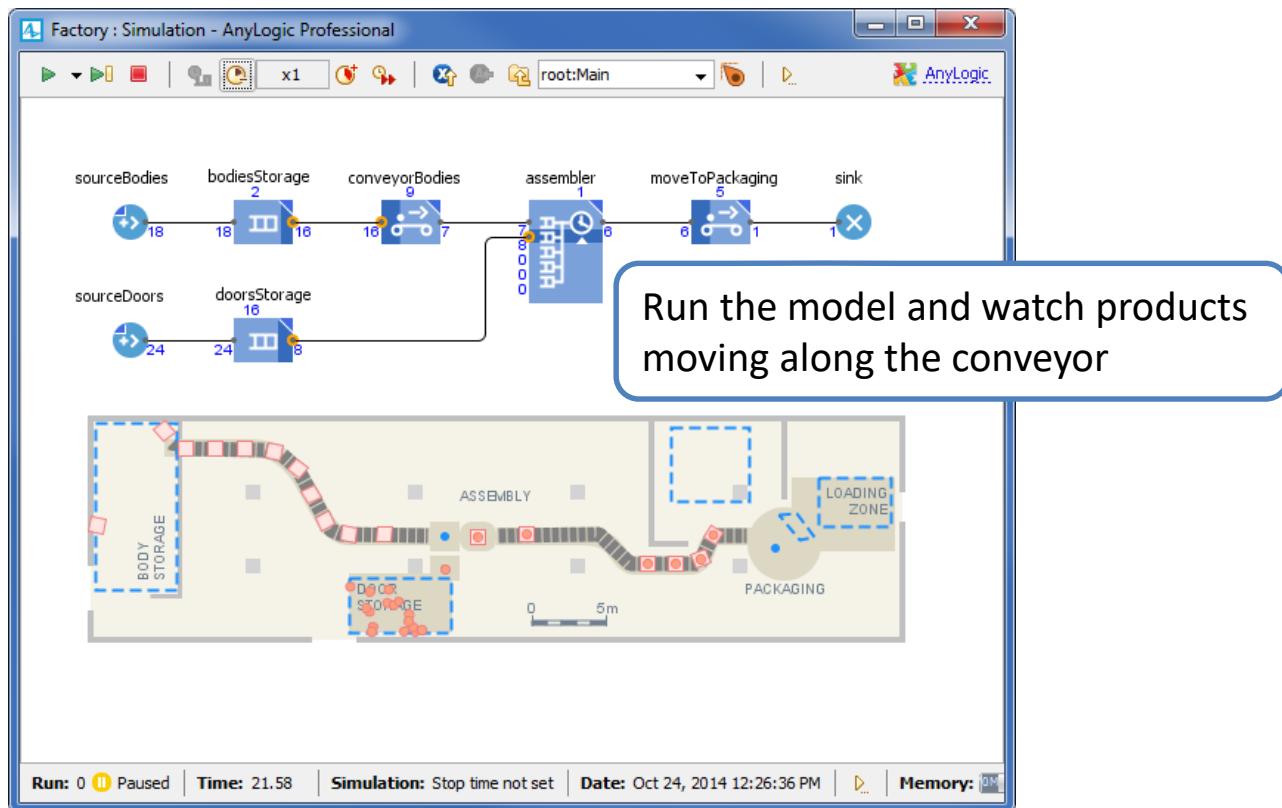


Factory. Phase 4. Step 2

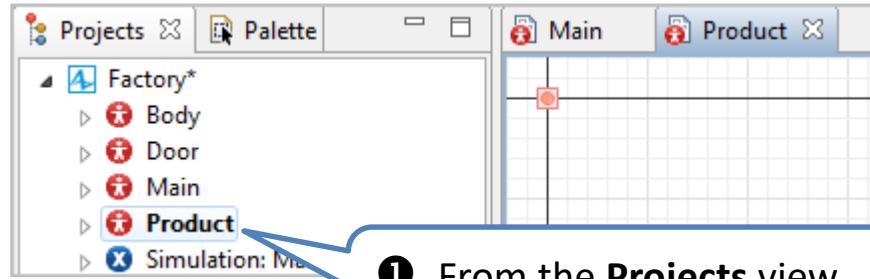


Now let's model an upstream conveyor leading to the packaging zone.

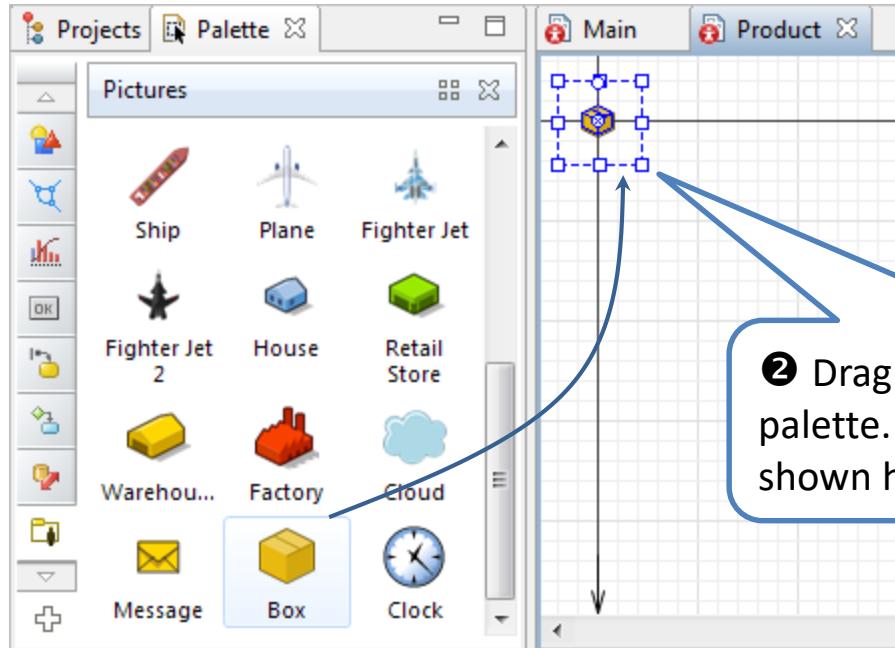
- ① Add a conveyor by cloning *conveyorBodies* block. We recommend that you clone the conveyor drawn earlier since it already has some parameters adjusted the way we need.
- ② Specify the shape that will be used as a path for agent animations



Factory. Phase 4. Step 3



- 1 From the **Projects** view, open *Product* agent diagram



- 2 Drag the *Box* from the *Pictures* palette. Place and resize it as shown here, set **Visible: no**.

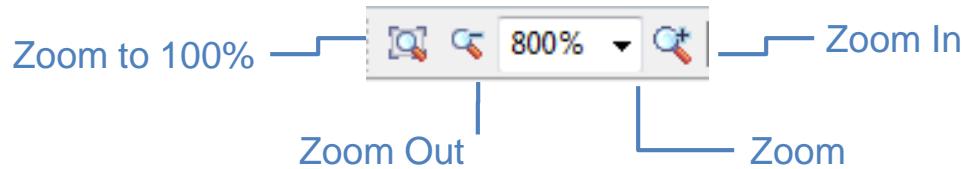


Draw a picture of a box containing our washing machine.

To simplify drawing small pictures, zoom the diagram to 400%. Resize the picture to occupy approximately one grid cell.

Zooming the graphical diagram

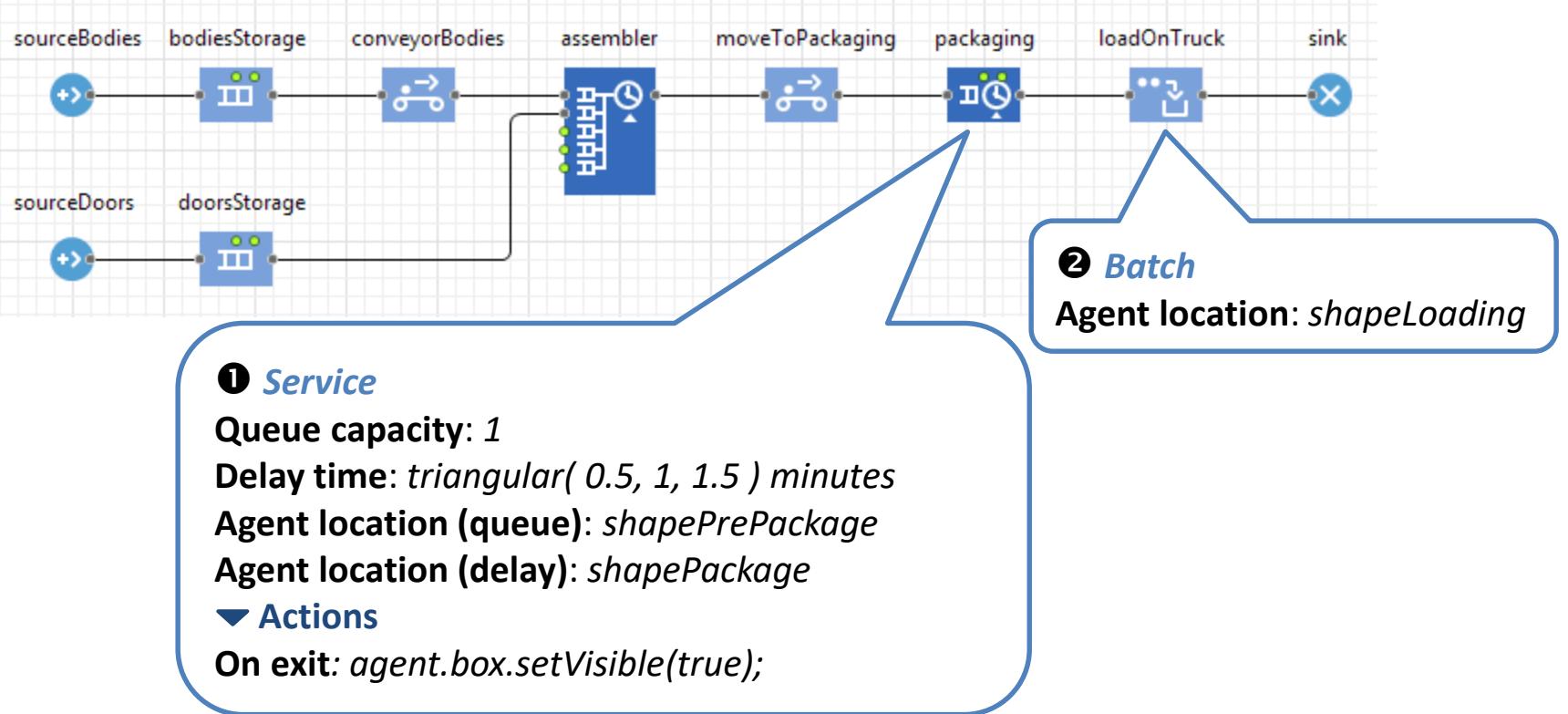
- AnyLogic allows zooming graphical diagram in and out as you like. Set up the diagram scale using the AnyLogic **Zoom** toolbar:



- Set the required zoom using the **Zoom** combo box or **Zoom Out/Zoom In** buttons.
- To return to the default zoom click the **Zoom to 100%** button.



Factory. Phase 4. Step 4

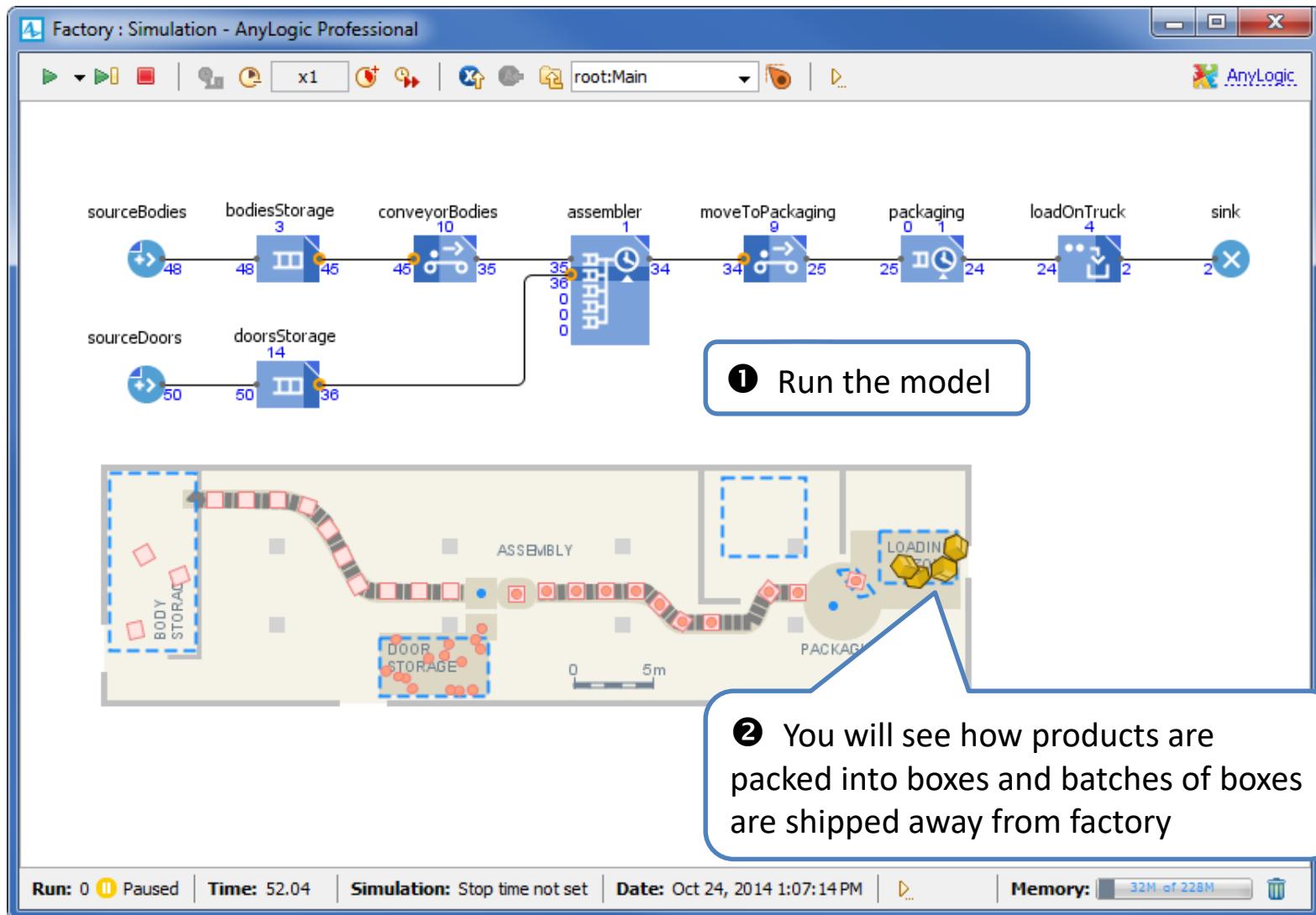


- ① This **Service** block models packaging a washing machine into a box. At this time we will not use resources – they will be added in the next phase.
- Time needed to pack a machine into a box: *triangular(0.5, 1, 1.5) minutes*
 - Set the capacity of the queue equal to 1. We assume that one washing machine may wait for packaging in a prepackage zone.
 - In the **On exit** action code we make our box shape visible for the agents that exit this *Service* block (i.e. machines packed in boxes).
- ② This **Batch** block models loading a batch of boxes onto a truck (saying in terms of the object, it creates a batch (truck) from a set of original agents (boxes)).
- Check that the **Permanent batch** checkbox is deselected since we want to create a temporary batch (later on we will unbatch it to a number of boxes again).

-
- **Service** block seizes a given number of resources to perform some operation over agent, delays the agent (this models the operation itself), and releases the seized resources.
 - **Batch** block converts a number of agents into one agent (*batch*) by either discarding the original agents and creating a new one - *permanent* batch, or by adding the original agents to the contents of the new agent - *temporary* batch that can later on be unbatched by **Unbatch** block.



Factory. Phase 4. Step 5

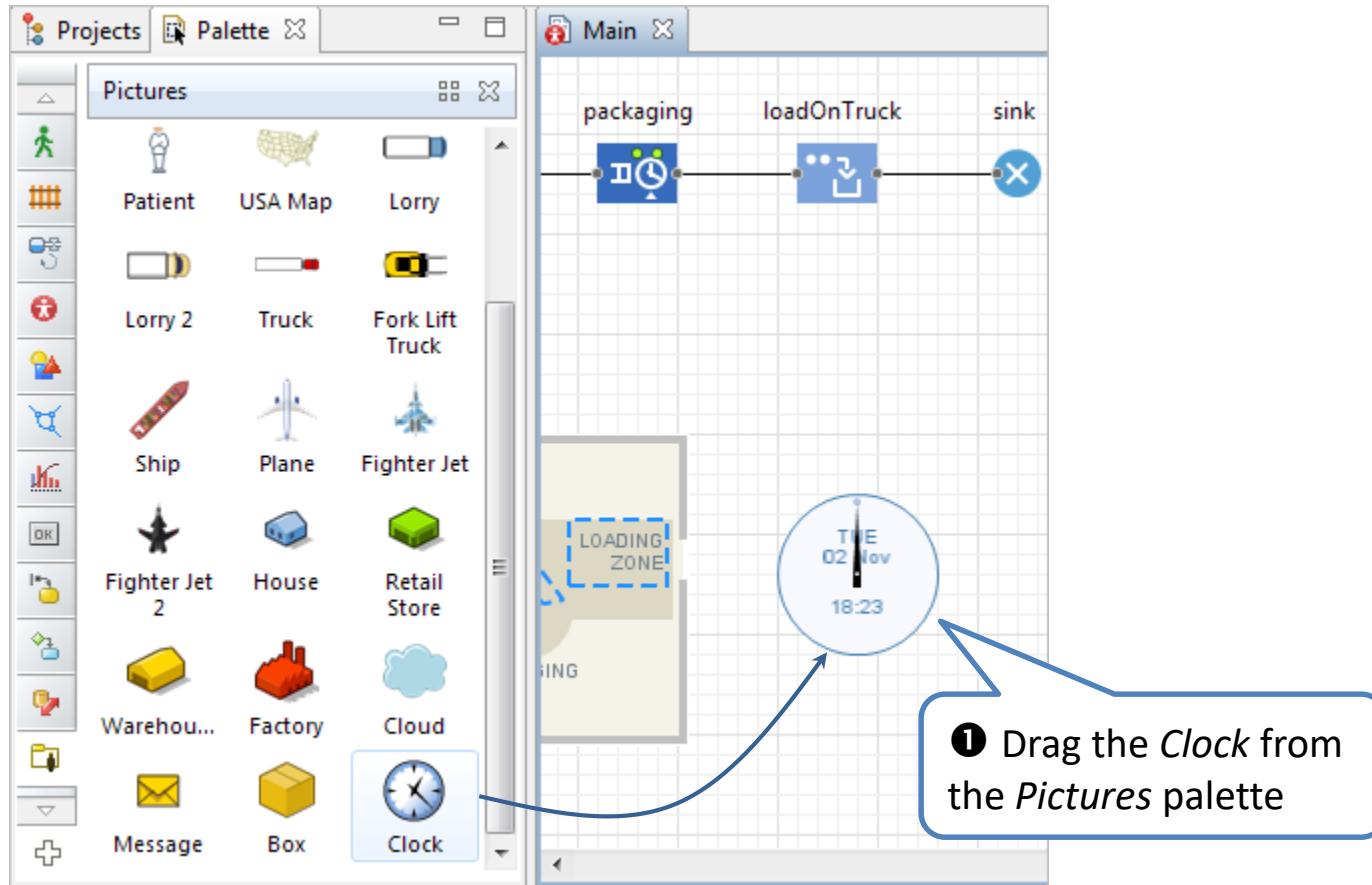


Factory. Phase 5

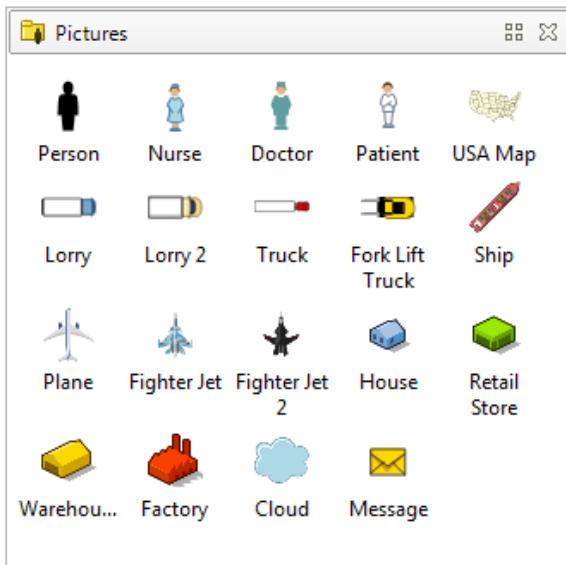
- In fact assembly and packaging operations need resources – assembly robot and packagers correspondingly.
- Let's add two types of resources into the model:
 - one robot to perform an assembly
 - two workers to perform packaging
- Collect utilization statistics for the resources and display it using bar charts.
- Introduce assembly robot downtime.



Factory. Phase 5. Step 1



A set of standard pictures



- The **Pictures** palette contains a set of pictures of frequently modeled objects: person, truck, lorry, forklift truck, warehouse, factory, etc.
- You can simply add a standard picture from the **Pictures** palette and continue developing your model, wasting no time on drawing.



Factory. Phase 5. Step 2

The screenshot shows the AnyLogic software interface with the following components:

- Projects** tab is selected in the top-left.
- Palette** tab is open, showing categories like Agent, Parameter, Event, etc.
- Main** tab is open, displaying a model diagram with nodes: sourceBodies, bodiesStorage, sourceDoors, and schedule.
- Properties** tab is open, showing the properties for the "schedule" node.
- ① Add Schedule** callout points to the "Schedule" item in the palette.
- ② Set schedule to define rates and define time pattern as shown here** callout points to the "Repeat schedule weekly:" table.

Properties - schedule - Schedule

- Name: schedule Show name Ignore
- Visible: yes
- Type: Rate
- Unit: per minute
- The schedule defines: Intervals (Start, End)
- Duration type: Week Days/Week
- Default value: 0

Repeat schedule weekly:

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Start	End	Value
✓	✓	✓	✓	✓	---	---	9:00	13:00	1.0
✓	✓	✓	✓	✓	---	---	14:00	18:00	1.0

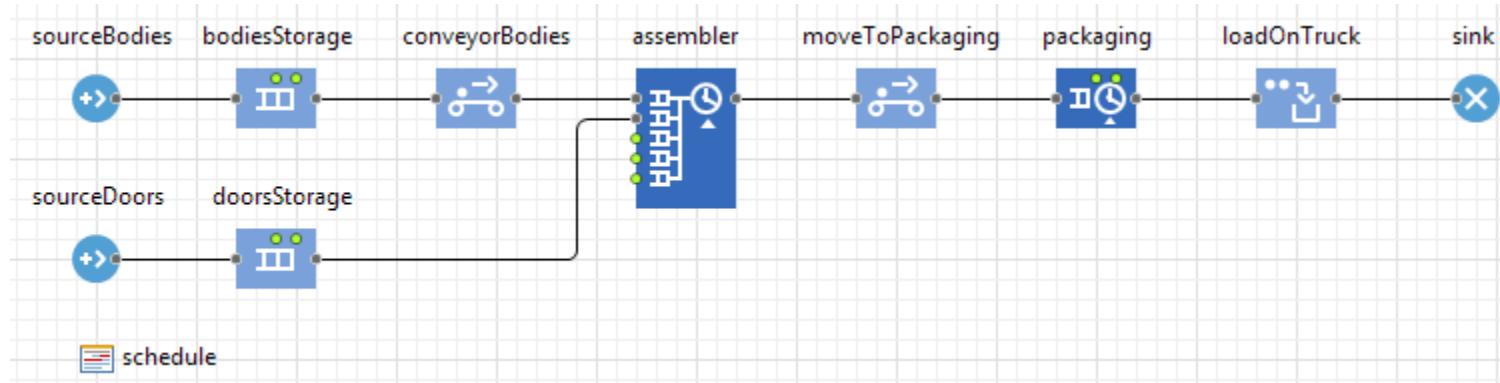


Schedule

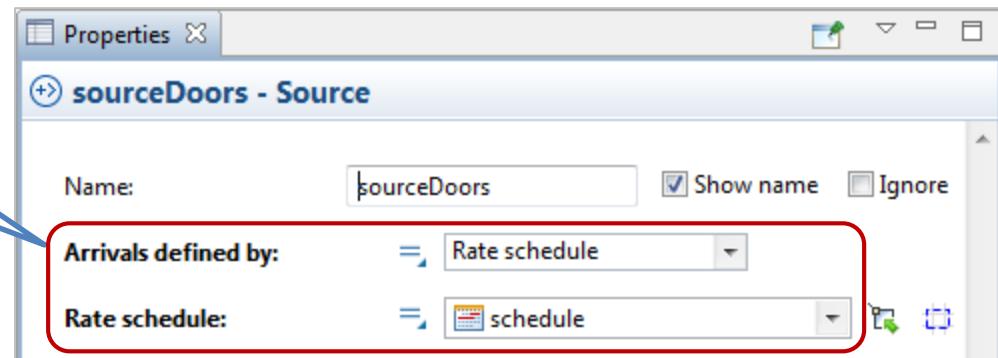
- *Schedule* is a special element allowing you to define how some value changes in time according to a defined cyclic pattern. Schedule is frequently used to define:
 - Agent generation times, or cyclic pattern of agent arrival rate in source objects.
 - Timetable for a pool of resources defined with **ResourcePool**.
- Schedule works in one of two modes: it can either define *intervals*, or *time moments*.
 - *Intervals* mode is used when you need to define how some value continuously changes in time (usually according to some cyclic pattern). Intervals are used to define work timetables for worker shifts, cyclic pattern of agent arrival rate, etc.
 - *Moments* mode is used when you need to define a sequence of key time moments and some values corresponding to this particular moments (or perform some actions. The example - train arrival schedule.
- You can define a schedule in one of three alternative views:
 - *Week* - Use this view in case your schedule has weekly recurrence, e.g. if you need to define a weekly schedule for office men.
 - *Days/Weeks* - Use it when you need to define a schedule as a sequence of calendar dates and times with non-weekly recurrence, e.g. a shift schedule 24 hours on, 48 hours off.
 - *Custom (no calendar mapping)* - There is no mapping to calendar dates. Schedule intervals/moments are defined simply as number of time units (milliseconds, seconds, minutes, hours, days, or weeks) passed from the model start. Define schedule in this view when only interval durations are important, not the exact calendar times.



Factory. Phase 5. Step 3



① Make both *Source* blocks generate parts of washing machines according to the schedule



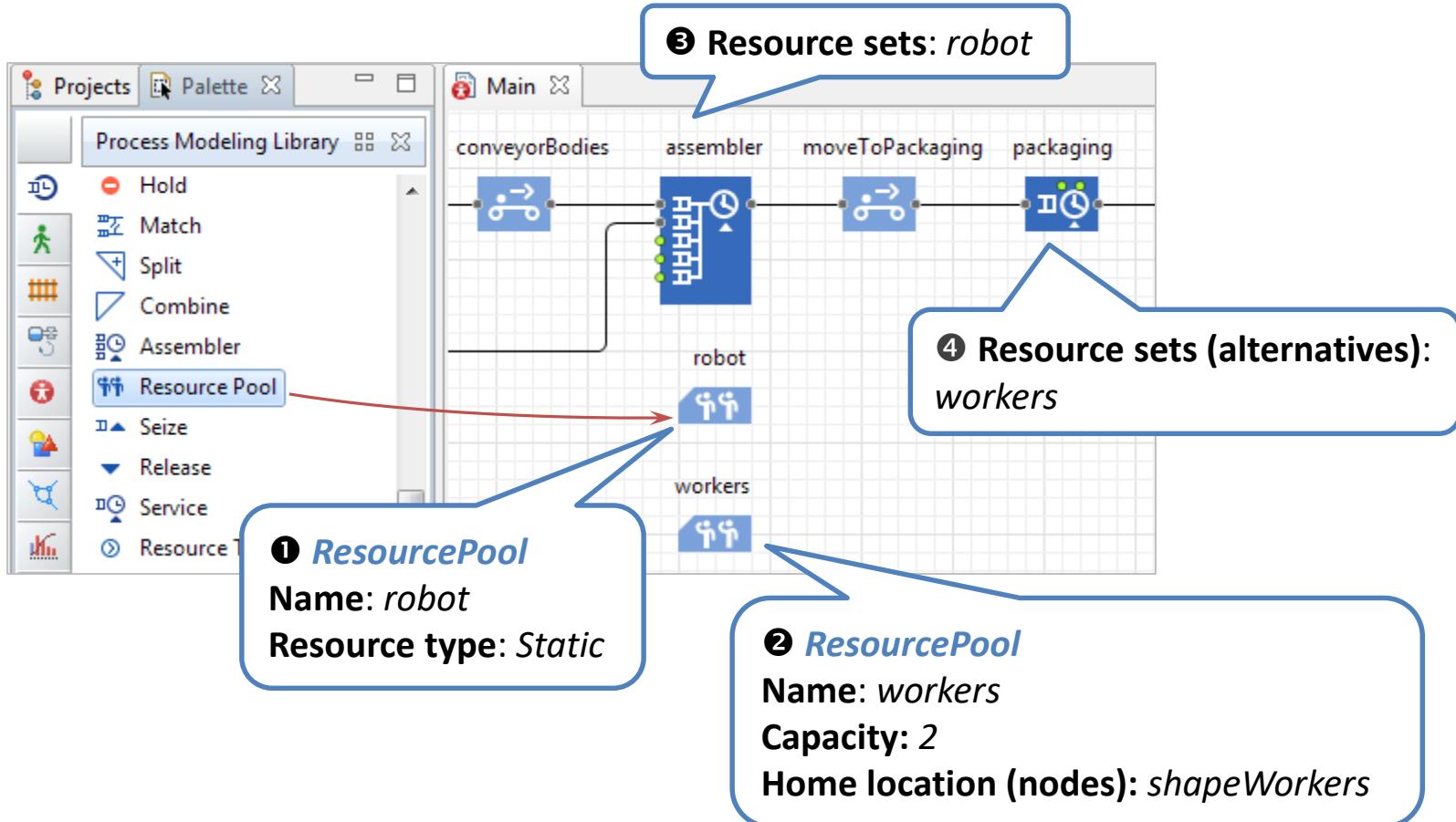
Arrival modes of source objects

- All currently present AnyLogic libraries contain objects that generate library agents. In Process Modeling Library, *Source* supports several agent arrival modes:

Arrival mode	Description
Rate	Agents are created at the specified arrival rate (which is equivalent to exponentially distributed interarrival time with mean = 1/rate).
Interarrival time	The time between two subsequent arrivals is defined by the specified value/expression.
Rate schedule	Agents are generated using rate schedule - a schedule defining how the arrival rate changes with time.
Arrival schedule	Agents are generated at exact times defined in the arrival schedule.
Calls of inject() function	Agents are not generated automatically and are only generated on calls of inject() method.



Factory. Phase 5. Step 4



- ① Add a resource to simulate the assembly robot. Name it *robot* and change the **Resource type** to *Static* in its properties.
- ② Add another **ResourcePool** object to simulate two workers that perform packaging of the product.
 - Name the object *workers*.
 - Set the **Capacity** parameter to 2.
 - Choose the *shapeWorkers* markup object as the home location of the resources.
- ③ - ④ Tell *assembler* block that now it needs one resource to perform the operation, and the object *packaging* will use *workers* resources.

ResourcePool

- **ResourcePool** object defines a set of *resources*. Resources are objects that are needed by agents to perform operations. Resources are seized and released by agents using **Seize**, **Release**, **Service** and **Assembler** objects.



Factory. Phase 5. Step 5

The image shows a simulation model in AnyLogic. On the left, a process flow diagram represents a factory line with four stages: 'conveyorBodies', 'assembler', 'moveToPackaging', and 'packaging'. A 'robot' resource (represented by a blue icon with two arms) is assigned to the 'assembler' stage. On the right, the 'ResourcePool' configuration for the 'robot' resource is displayed. A callout box highlights the 'Failures / repairs:' setting, which is checked. Another callout box points to the 'Time to repair:' field, which is set to 'uniform(0, 1000)' and 'triangularAV(1000, 0.1)' with a unit of 'minutes'. A red box surrounds the 'minutes' unit in both fields.

conveyorBodies assembler moveToPackaging packaging

robot worker

① ResourcePool
Shifts, breaks, failures, maintenance...
Failures / repairs:

② Choose minutes

'End of shift' priority:
'End of shift' preemption policy:
'End of shift' may preempt:

Breaks:

Failures / repairs:

Initial time to failure: uniform(0, 1000) minutes
Time to next failure: triangularAV(1000, 0.1) minutes
Repair type: Delay
Time to repair: triangularAV(10, 0.1) minutes
Usage statistics are: not collected

Maintenance:

Custom tasks:



Select the *robot* objects and go to its Properties. Expand the section **Shifts, breaks, failures, maintenance...**

- ① Choose the option **Failures / repairs**. You will see several new parameters that you can use to define these events for your resources.
- ② We know that an assembly robot might fail and will require some repair work, but we do not know when exactly. In this case we will use the distribution functions to define these time intervals.



Factory. Phase 5. Step 6

The screenshot shows the AnyLogic software interface with three main panels: Projects, Main, and Properties.

- Analysis palette:** The "Charts" section is open, with "Bar Chart" highlighted and selected.
- Main panel:** A bar chart titled "chart - Bar Chart" is displayed. It has one data series named "workers" with one bar. The bar's value is 4.319E-4. A callout bubble 1 points to the "Bar Chart" icon in the Analysis palette.
- Properties panel:**
 - Data section:** The title is "Workers utilization", color is set to "dodgerBlue", and the value is "workers.utilization()". A callout bubble 3 points to the "Recurrence time" field, which is set to 1 minute.
 - Appearance section:** The "Bars direction" dropdown is set to a red square icon.

Step 6 Instructions:

- 1 Add a *Bar Chart* from the **Analysis** palette
- 2 Click this button in the **Data** section, then change the data element's properties
- 3 Define the recurrence time
- 4 Resize the chart in the graphical editor and change its bars direction in the **Appearance** section



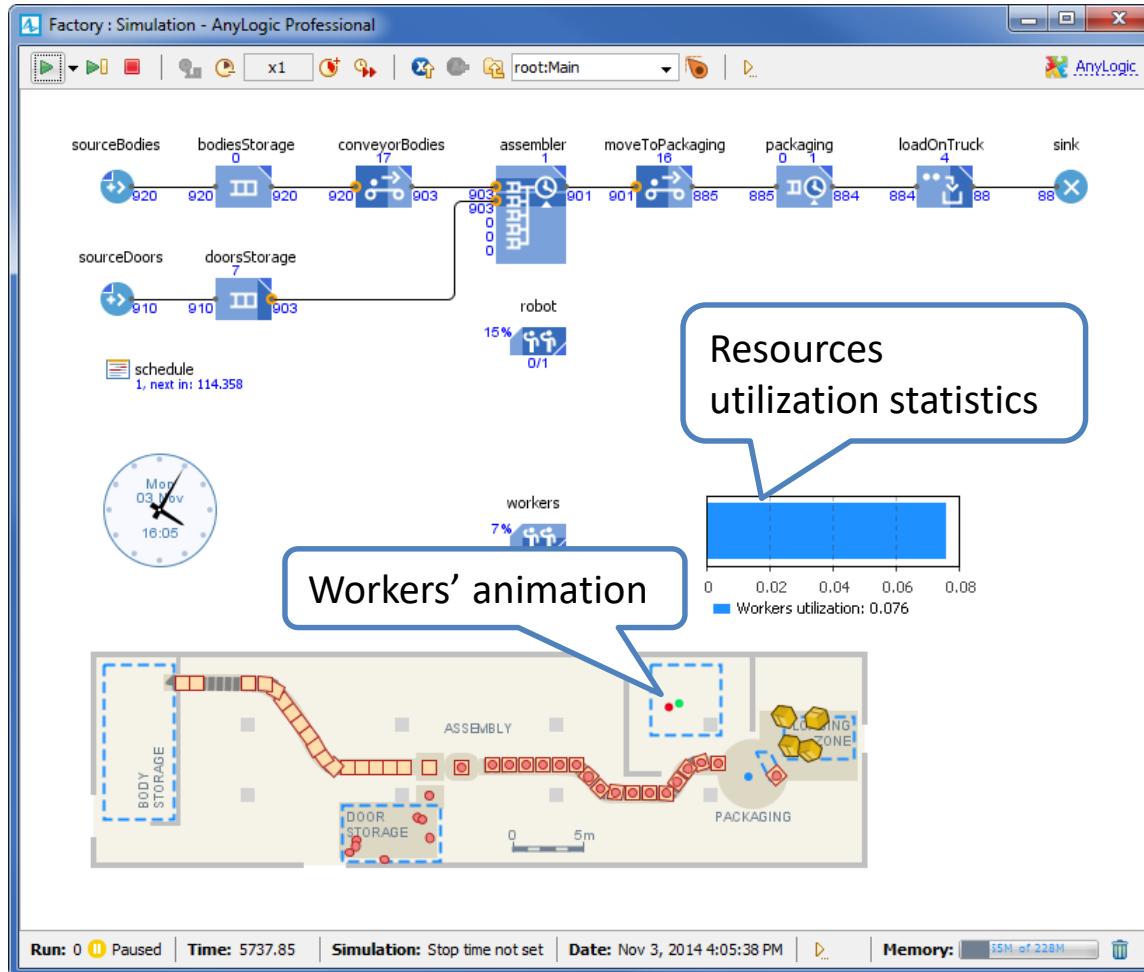
Add a bar chart to display the utilization statistics of the workers.

- ① The **Analysis** palette contains the elements that you can use to collect and analyze the simulation results (*data set, statistics ...*) and charts (*bar chart, stack chart, time plot, histogram ...*) that help you visualize that collected data.
- ② Type *Workers utilization* in the **Title** field of the data element. Define **Value** that this bar chart will display: *workers.utilization()*. *workers* here is the name of the **ResourcePool** object, and *utilization()* is the function that collects the utilization statistics of the resources.
- ③ - ④ Set the **Recurrence time** to *1 minute*, then go to the **Appearance** section and change the **Bar direction** property.



Factory. Phase 5. Step 7

Now we have finished creating our first model of a factory. You can analyze the impact of equipment downtime and maintenance period on the factory performance



Factory. Phase 5. Questions

1. If you model a truck with boxes going from one warehouse to another would you use a permanent batch or non-permanent?
2. How can you change the number of available resources of some specific type?
3. Workers are animated with circles now. How can you set 2D worker shapes for them?



Call Center Exercise

Optional Discrete Event Practice. Can be incorporated into class or done as homework.



Exercise. Call center (1/3)

- Two types of calls arrive:
 - Type 1 with arrival rate $\text{ArrivalRate1} = 1.5$ per second
 - Type 2 with arrival rate $\text{ArrivalRate2} = 1$ per second
- Callers abandon if they wait too long
 - Abandonment time is exponentially distributed
 - For call type1 its mean is $\text{AbandonmentTimeMean1} = 100$ seconds
 - For call type2 its mean is $\text{AbandonmentTimeMean2} = 100$ seconds
- Two operator groups answer calls
 - Capacities are $\text{NOperators1} = 100$ and $\text{NOperators2} = 100$
 - Service time is distributed as triangular($\text{mean}/2$, mean , $2*\text{mean}$) where mean is ServiceTimeMeanXX defined as:
 - For operators 1 answering calls type 1 $\text{ServiceTime11} = 100$ seconds
 - For operators 2 answering calls type 1 $\text{ServiceTime12} = 200$
 - For operators 2 answering calls type 2 $\text{ServiceTime22} = 100$



Exercise. Call center (2/3)

- There is a queue to each operator group
 - Maximum capacity of queue to group 1 is QCapacity1 = 50
 - Maximum capacity of queue to group 2 is QCapacity2 = 50
- The routing logic is applied when a call arrives and is the following:
 - If a call of type 1 arrives and the queue to group 1 is not full, it is placed in that queue, otherwise if the queue to group 2 is not full, it is placed there; if both queue are full, the call is balked
 - If the call of type 2 arrives and the queue to group 2 is not full, it is placed there, otherwise it is balked



Exercise. Call center (3/3)

- Build a simulation model of the call center
- Measure the percent of calls:
 - Balked
 - Abandoned
 - Serviced
- Obtain the distribution of time in system for answered calls (by call type)
- Add ability to vary the arrival rate and adjust the queue capacities on-the-fly and observe how this affects the quality of service

Tips

- Service block exposes **outPreempted** and **outTimeout** ports of the embedded Queue block



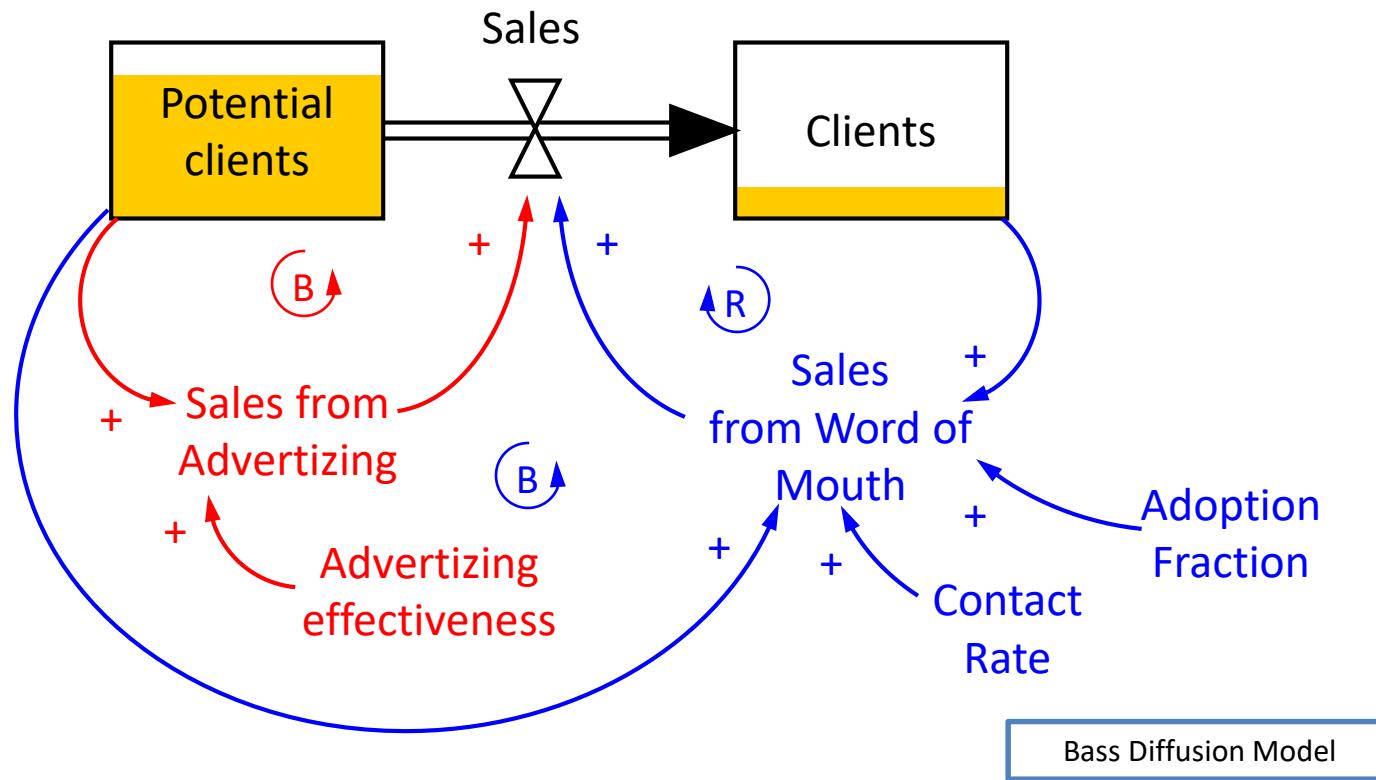
System Dynamics Modeling in AnyLogic

This presentation is a part of
AnyLogic Standard Training Program



System Dynamics Jay Forrester '50s

- Stocks, flows
 - Interacting feedback loops



System Dynamics Jay Forrester '50s

- Stocks, flows
 - Interacting feedback loops

The equivalent mathematical model:

$$d(\text{ Potential clients })/dt = - \text{Sales}$$

$$d(\text{ Clients })/dt = \text{Sales}$$

$$\text{Sales} = \text{Sales from Advertising} + \text{Sales from Word of Mouth}$$

$$\text{Sales from Advertising} = \text{Potential clients} * \text{Advertising effectiveness}$$

$$\begin{aligned}\text{Sales from Word of Mouth} &= \\ &\text{Clients} * \text{Contact Rate} * \\ &(\text{Potential clients} / (\text{Potential clients} + \text{Clients})) * \text{Adoption Fraction}\end{aligned}$$

Bass Diffusion Model



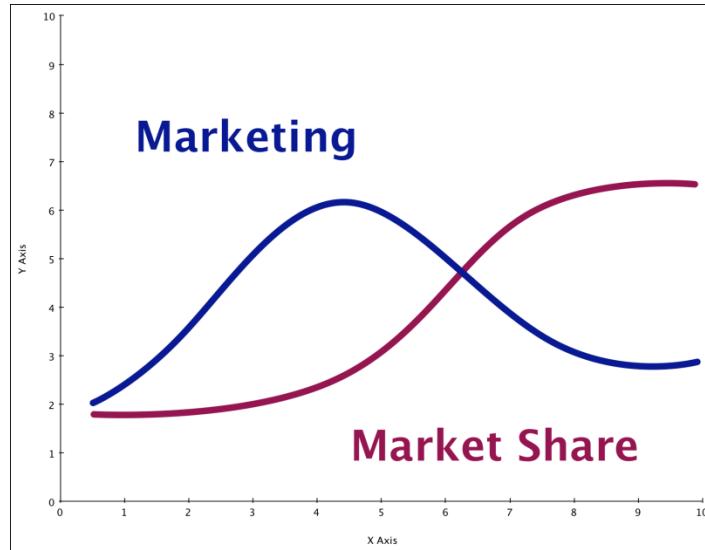
System Dynamics excels at certain things...



Cause and Effect

- Representing Causality and relationships
- System dynamics allow you to represent non-physical relationships too
 - *Morale -> Productivity, Advertising -> Perception,*

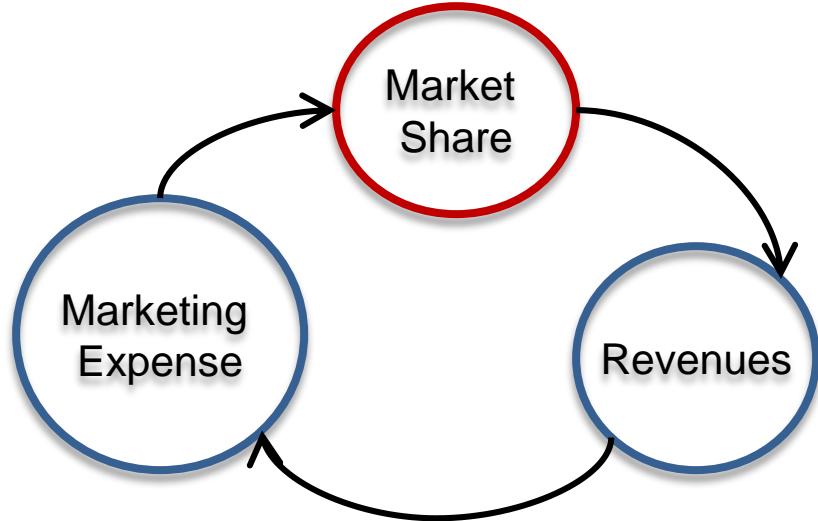
System Dynamics excels at certain things...



Delays

- It takes time for certain effects
- This can be from the nature of the elasticity between variables or from the effects of other mitigating effects
 - *Advertising -> Perception, Actions -> Reputation, Fame -> Perception*

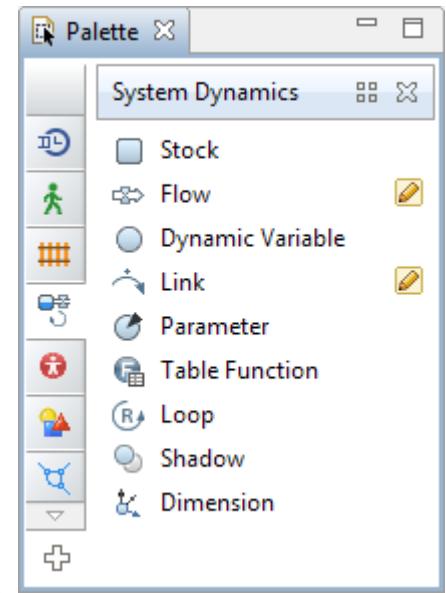
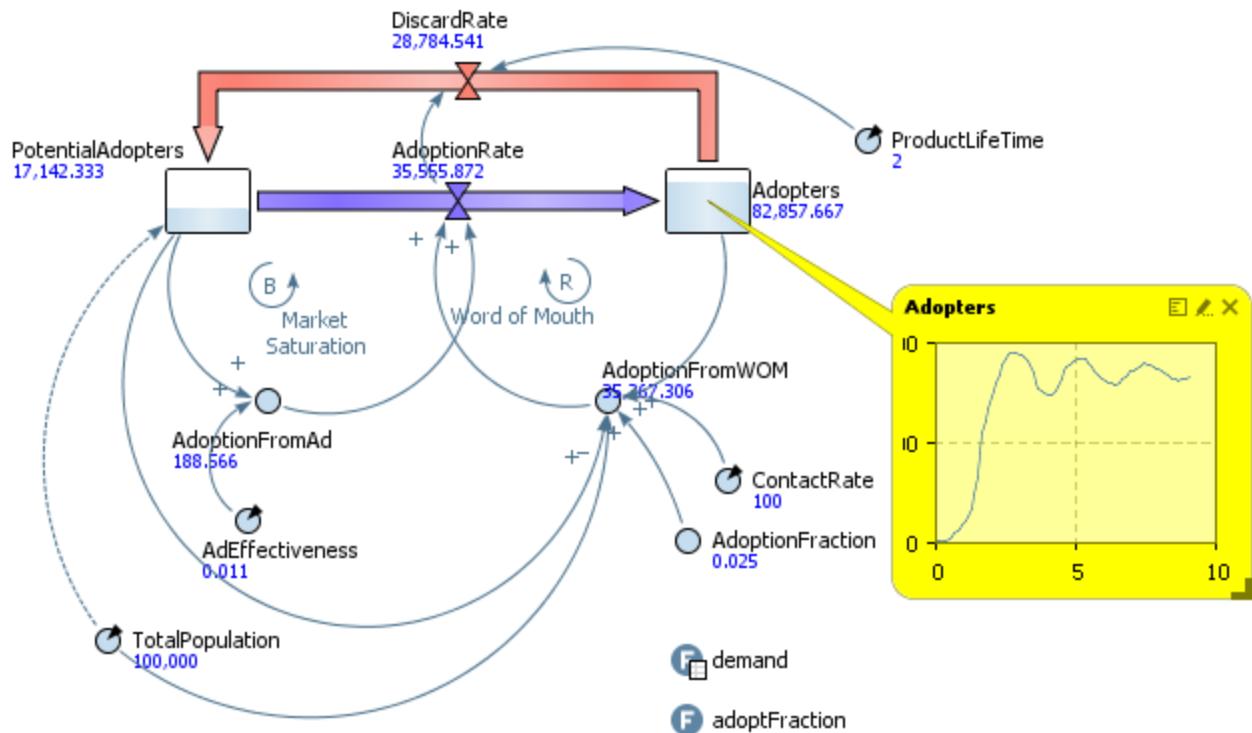
System Dynamics excels at certain things...



Feedback Effects

- “Practically” unique to System Dynamics, but common in the real world
- A logic error in excel
- Reinforcing Feedback Loops and Correcting Feedback

Stock & flow elements in AnyLogic



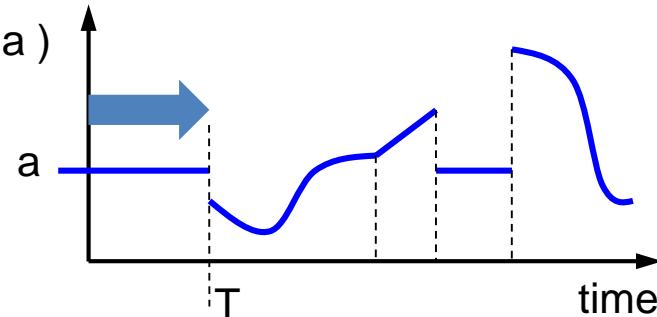
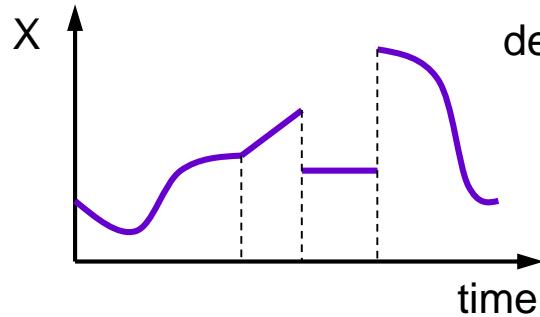
Built-in functions you can use

Mathematical functions

`abs(x) cos(x) exp(x) floor(x) limit(min,x,max) log(x)
max(a,b) min(a,b) pow(x) round(x) sin(x) sqrt(x) tan(x) ...`

Special system dynamics functions

`delay() delay1() delay3() delayInformation()
delayMaterial() forecast() npv() npve() smooth() ramp()
smooth3() trend() ...`



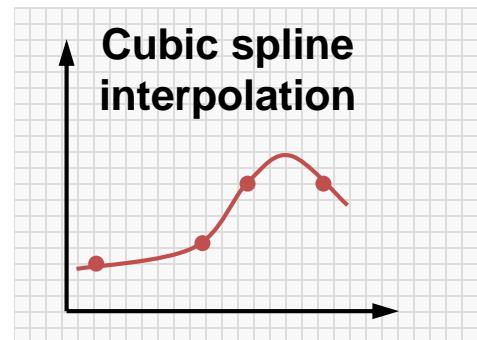
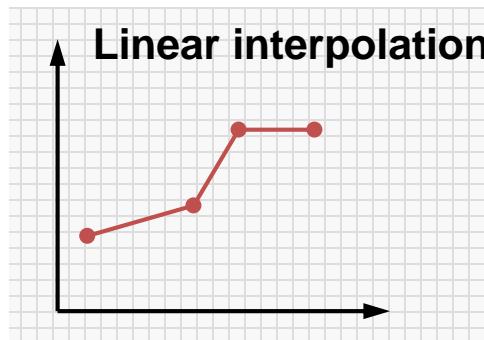
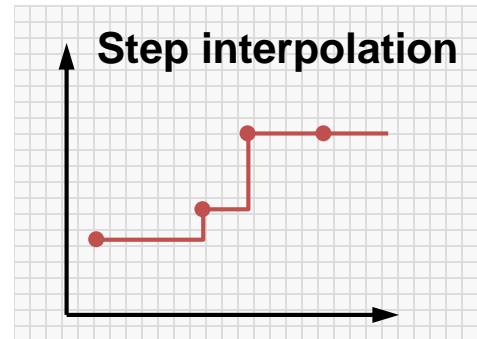
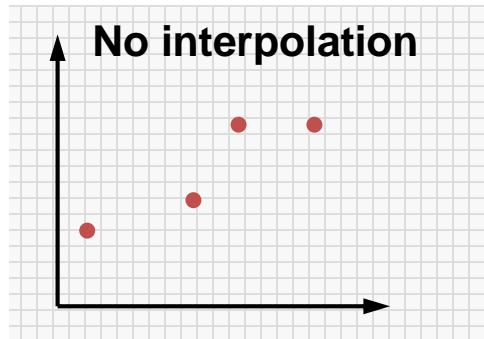
See full list of functions in AnyLogic Help -> AnyLogic Classes and Functions -> AnyLogic Functions



Table function (lookup table)

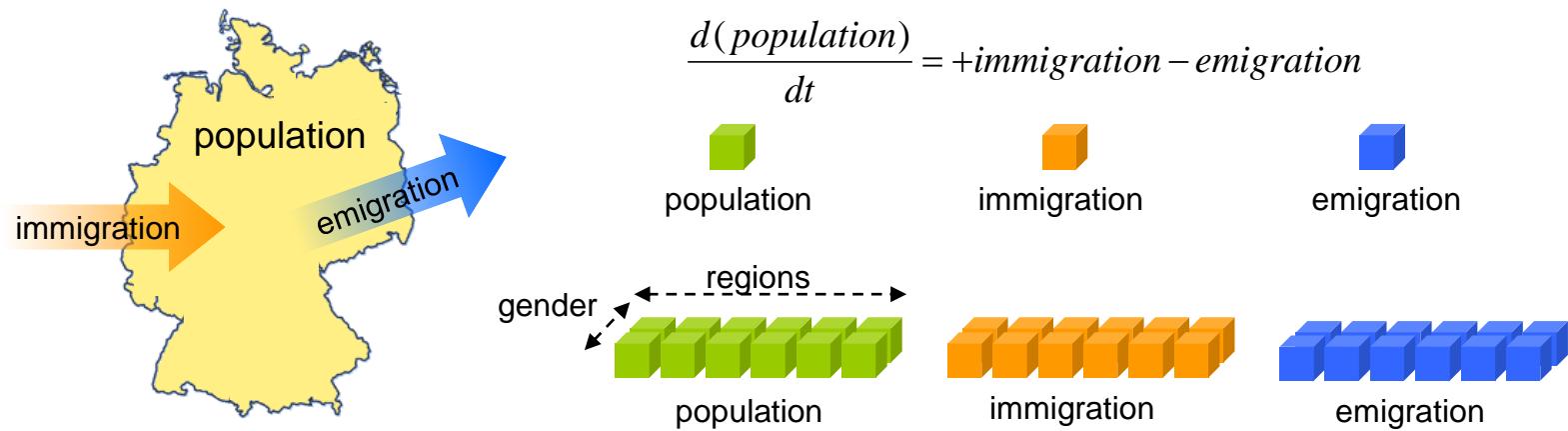
Allows to import simple two-column data table into the model and use it as a function: `myTable(x)`

Supports various interpolations and approximation



Arrays (“subscripts”)

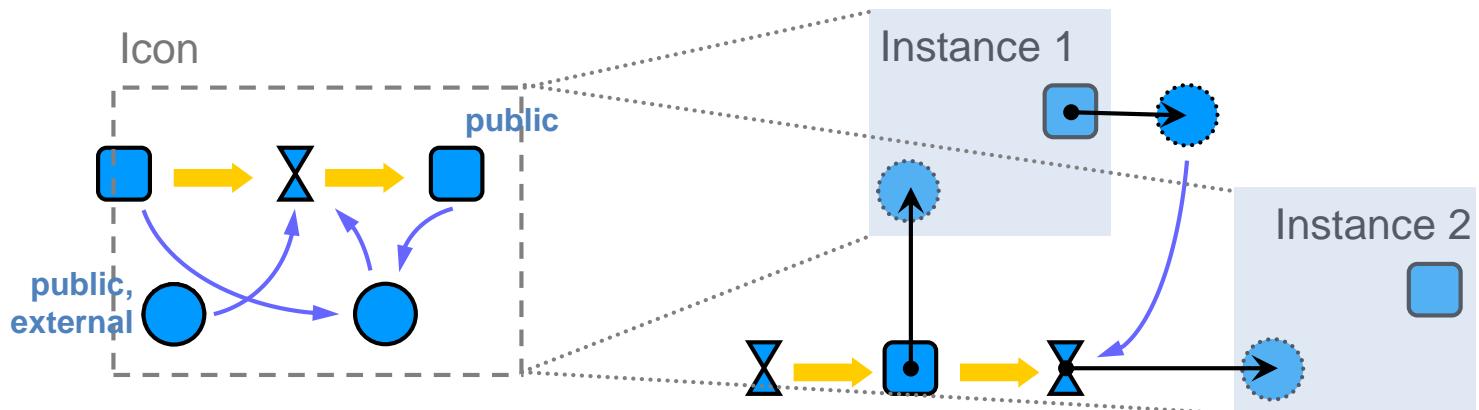
- Some problems require multidimensional data



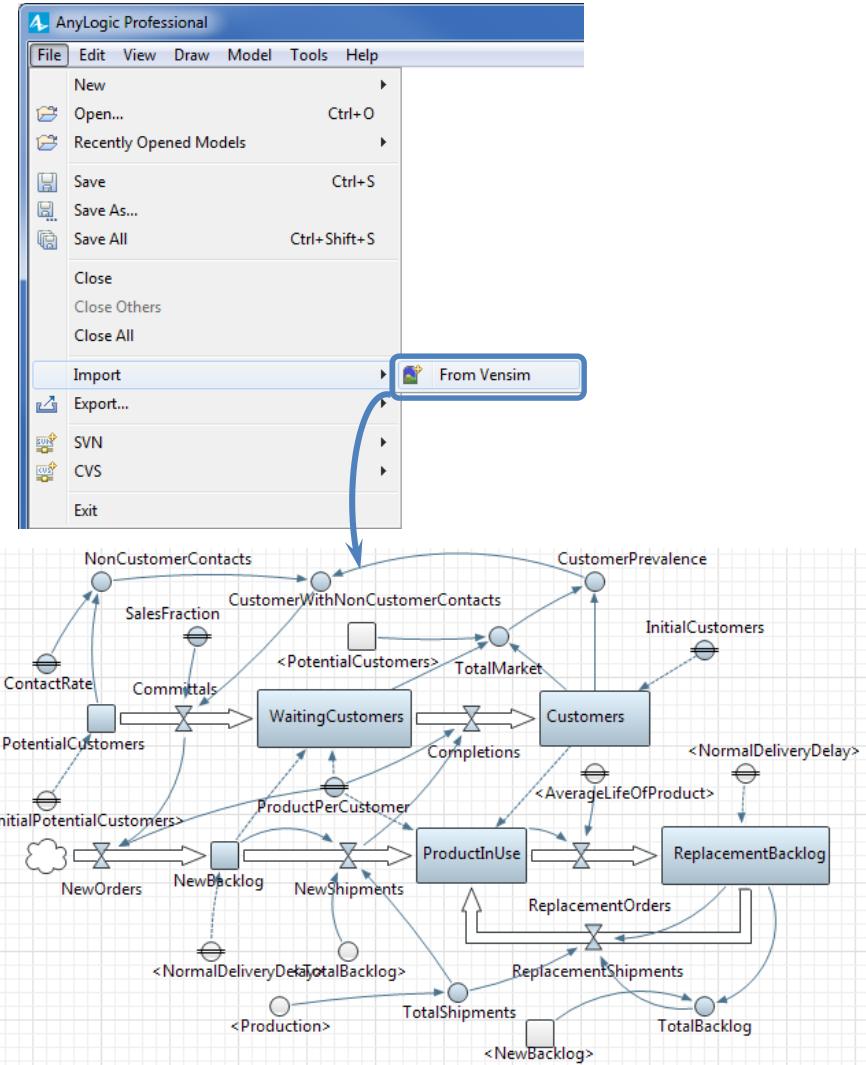
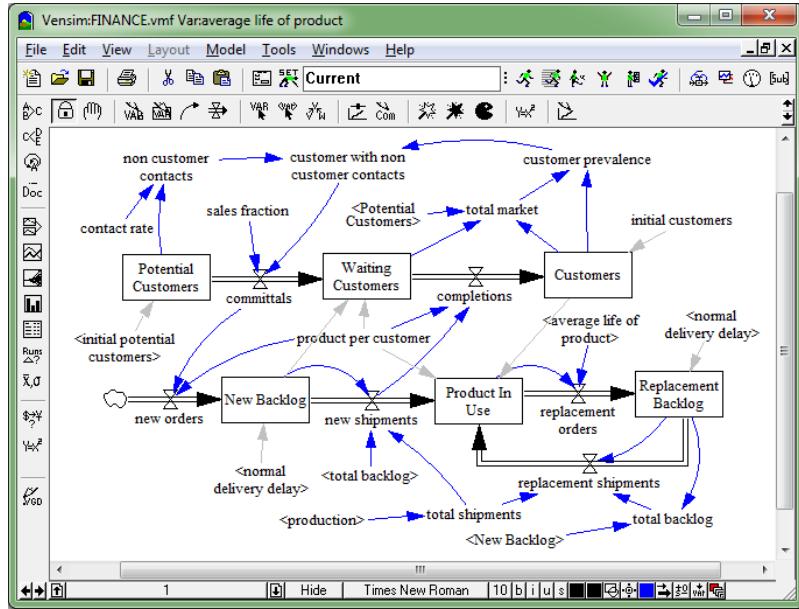
- AnyLogic supports array variables and equations
 - May have any number of dimensions
 - All arithmetic operations and most functions are supported
 - All operations are **element-by-element**, unlike in linear algebra

Object-oriented SD modeling

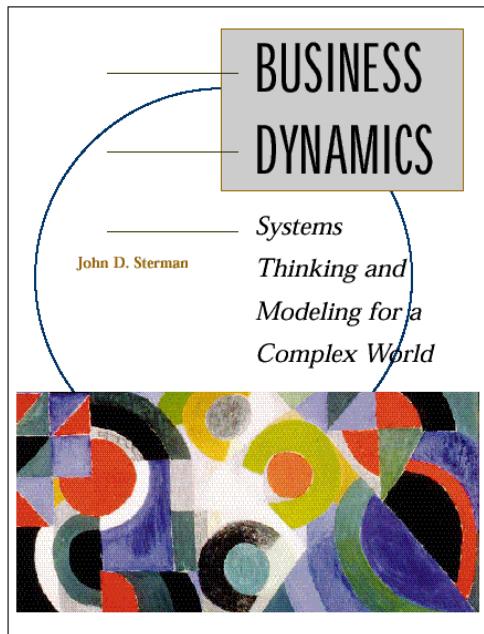
- Originally classical SD tools:
Only allowed you to build one large flat stock & flow diagram
which easily could become unmanageable
- AnyLogic:
Naturally suggests you to build well-structured models
You can build hierarchical models
You can create reusable components of SD models



Import of Vensim® models



The Sacred Book of System Dynamics Modeler



John Sterman

“Business Dynamics:
Systems Thinking and Modeling
for a Complex World”

2000. McGraw Hill, 1008 pages

<http://www.anylogic.com/business-dynamics-book-models>

The screenshot shows a section of the AnyLogic website dedicated to 'Business Dynamics' book models. At the top, there are navigation links: 'What's New in AnyLogic', 'Professional Features', 'Example Models' (which is highlighted in red), and 'Ask Question/Get Support'. Below these, a sidebar lists categories: 'Examples (101)', 'Models from 'The Big Book of Simulation Modeling' (1)', 'How-To Models (91)', and 'Models from the 'Business Dynamics' Book (50)'. A red box highlights the 'Models from the 'Business Dynamics' Book (50)' link. Underneath, a detailed list of model names is provided, grouped into four columns:

Adaptive Exp Random Walk	Adaptive Expectations	Bass Model	Bass Repeat Purch Flow
Bass with Discards	Capital Labor Coflow	Capital Vintaging Coflow	Commodity1
Faculty Aging Chain	First Order Neg FB	First Order Neg with Goal	First Order Pos FB
Floating Goals	Hillclimb	Hiring Chain1	Inv-WF Noise Switch
Inv-WF with Noise	Labor Learning Curve	Labor w Layoffs	Linear Population
Logistic Model	Market Growth 1	Multiplier Simul Eqns	Network Effect
Nonlinear Polya Process	Nonlinear Population	Nonlinear Smoothing	Pink Noise
Pink Noise Normal	Polya Process	Pop and Carrying Capacity	Population Model
Price Discovery	Price Sector	Richards Model	SIR Innovation Model
SIR Model	SIR Model	SIR Model Threshold	Stock Mgt 1st Order
Stock Mgt1	Stock Mgt2	Summary Statistics	TREND
W2Stage w DD FB	Widgets	Widgets w Backlog	Widgets w Labor
Widgets w Labor and OT	Widgets w Mat Inv		





Bass Diffusion Model

This presentation is a part of
AnyLogic Standard Training Program



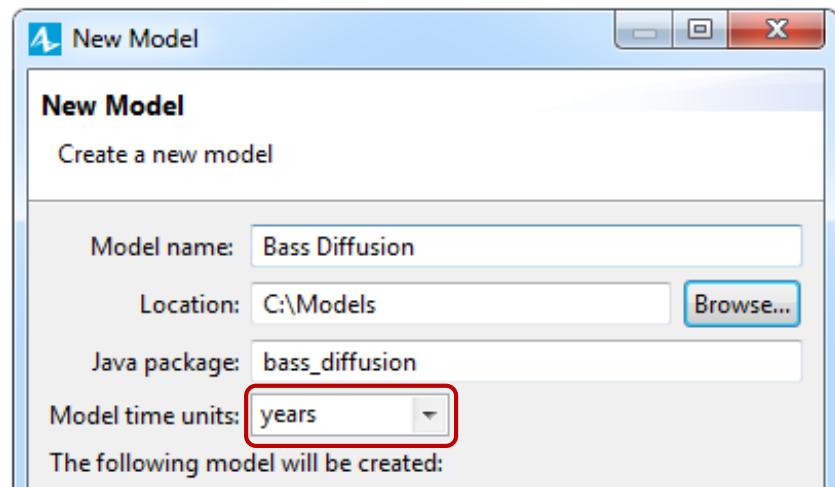
Bass Diffusion. Phase 1

- SD classical textbook model of product, or innovation diffusion
- A population group is considered to consist of **Potential Adopters** and **Adopters**; all people behave exactly same way
- **Potential Adopters** become **Adopters** at **Adoption Rate** which depends on advertising
- Advertising goes on all the time, and every time unit it converts **Advertising Effectiveness** part of **Potential Adopters** into **Adopters**
- Initially:
 - $Potential\ Adopters = 100000$
 - $Adopters = 0$
- Parameter values:
 - $Advertising\ Effectiveness = 0.011$

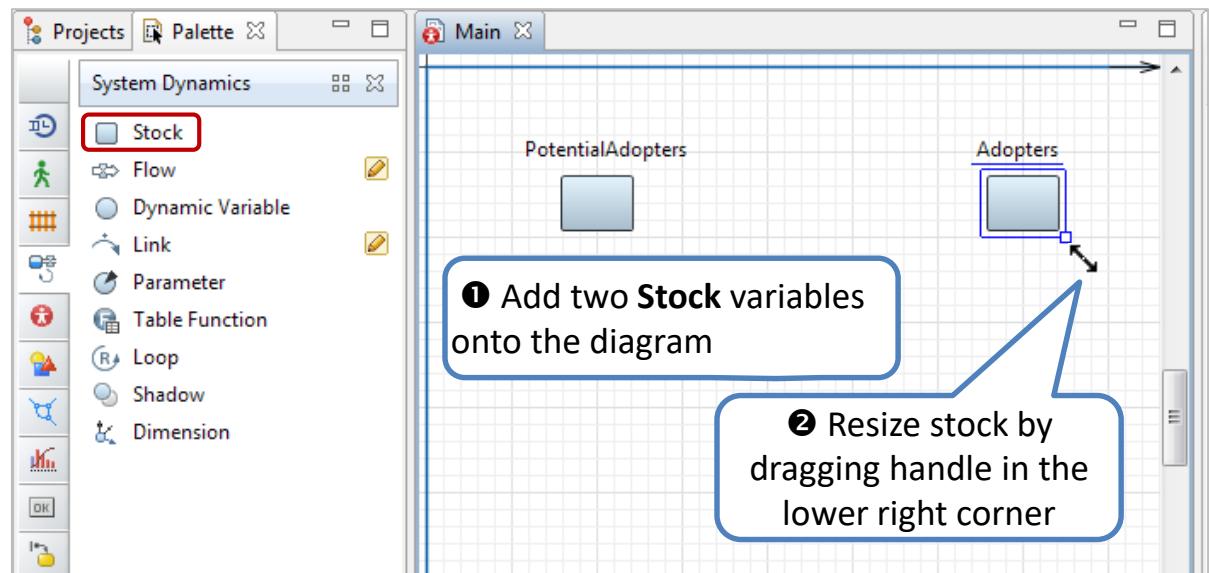


Bass Diffusion. Phase 1. Step 1

- Create new model and name it *Bass Diffusion*
- Set *years* as the model time units

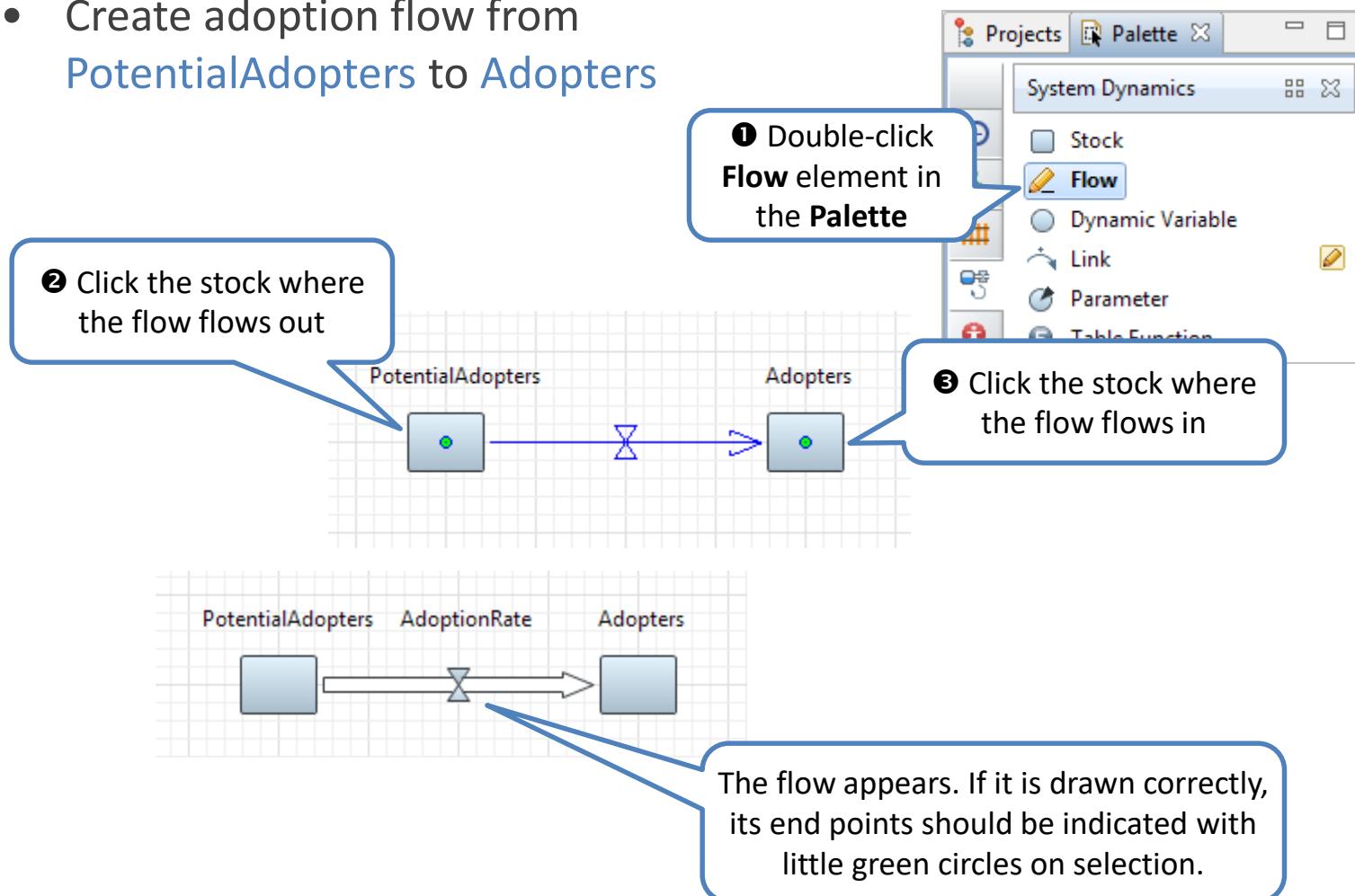


- Open the **System Dynamics** palette
- Add two Stocks onto the diagram
- Name them **PotentialAdopters** and **Adopters**



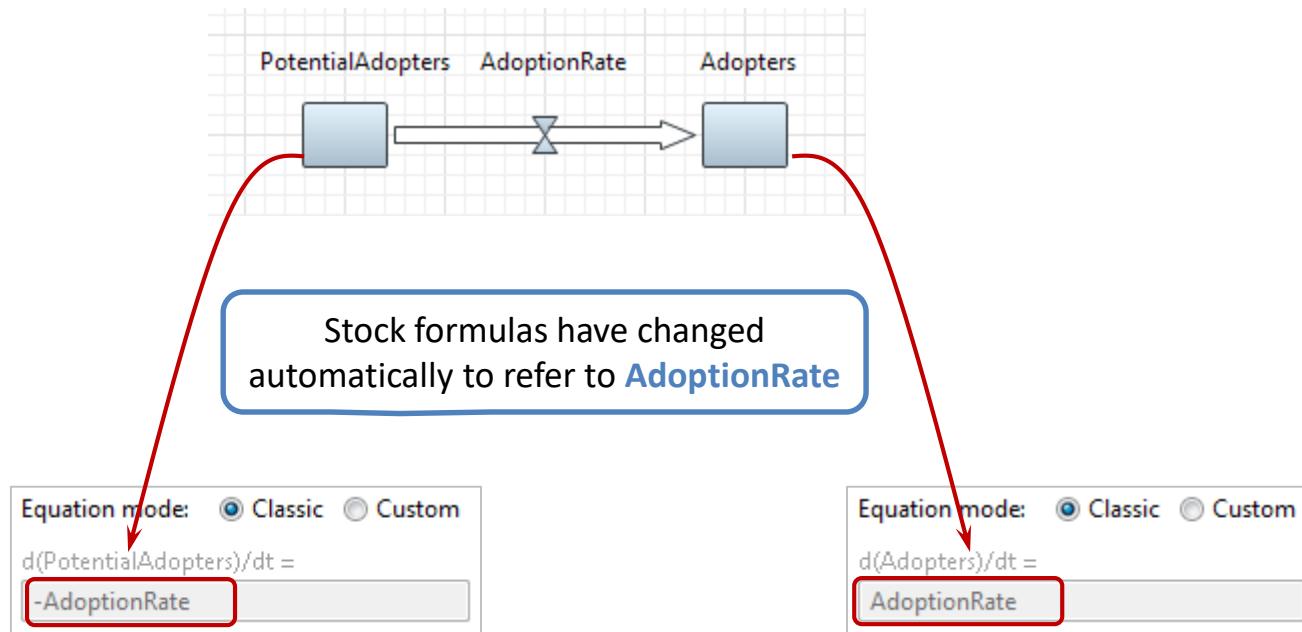
Bass Diffusion. Phase 1. Step 2

- Create adoption flow from PotentialAdopters to Adopters



Bass Diffusion. Phase 1. Step 3

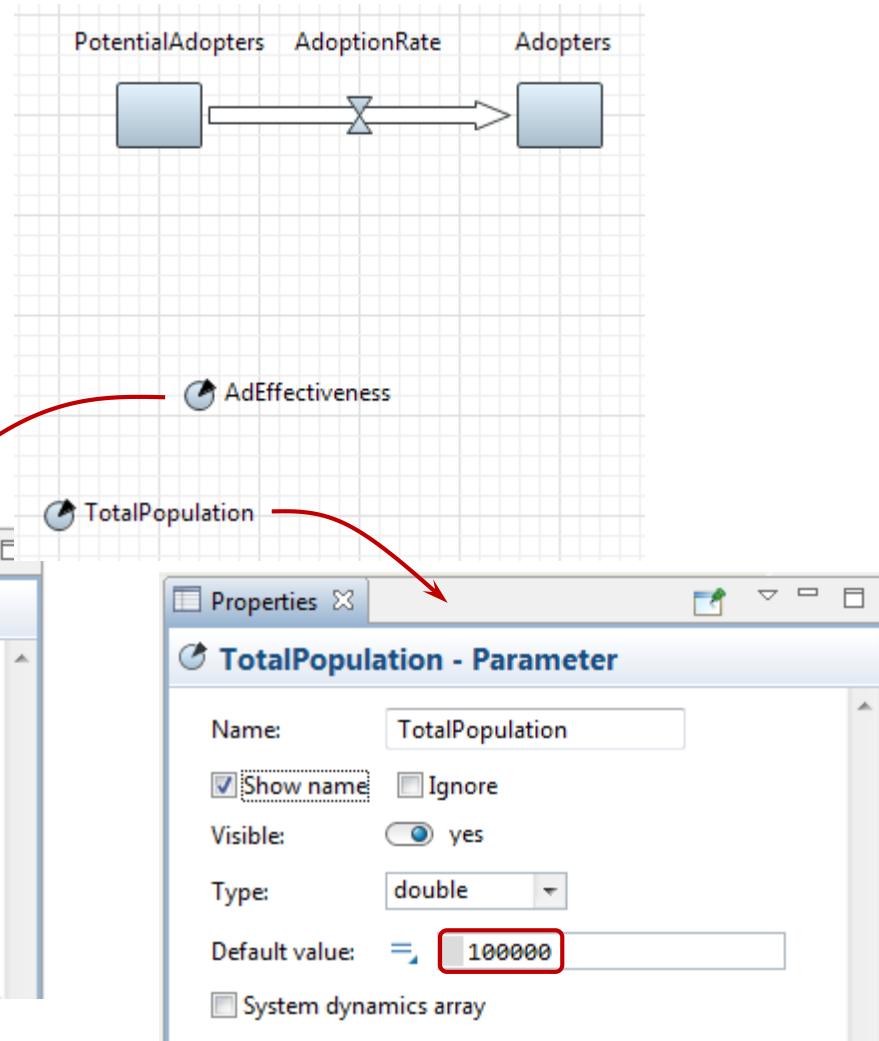
- Name the flow AdoptionRate



- Check the formulas of the stocks

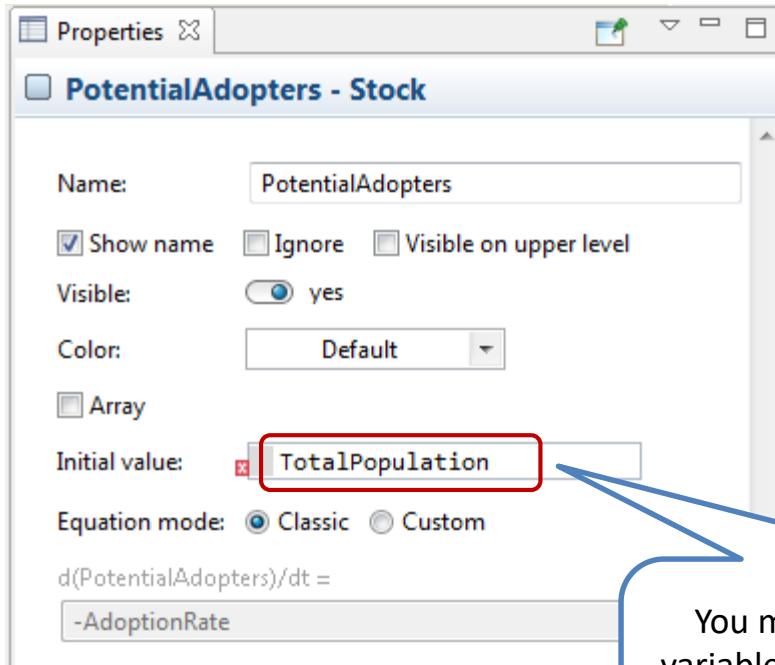
Bass Diffusion. Phase 1. Step 4

- Create TotalPopulation and AdEffectiveness parameters



Bass Diffusion. Phase 1. Step 5

- Specify initial value for PotentialAdopters



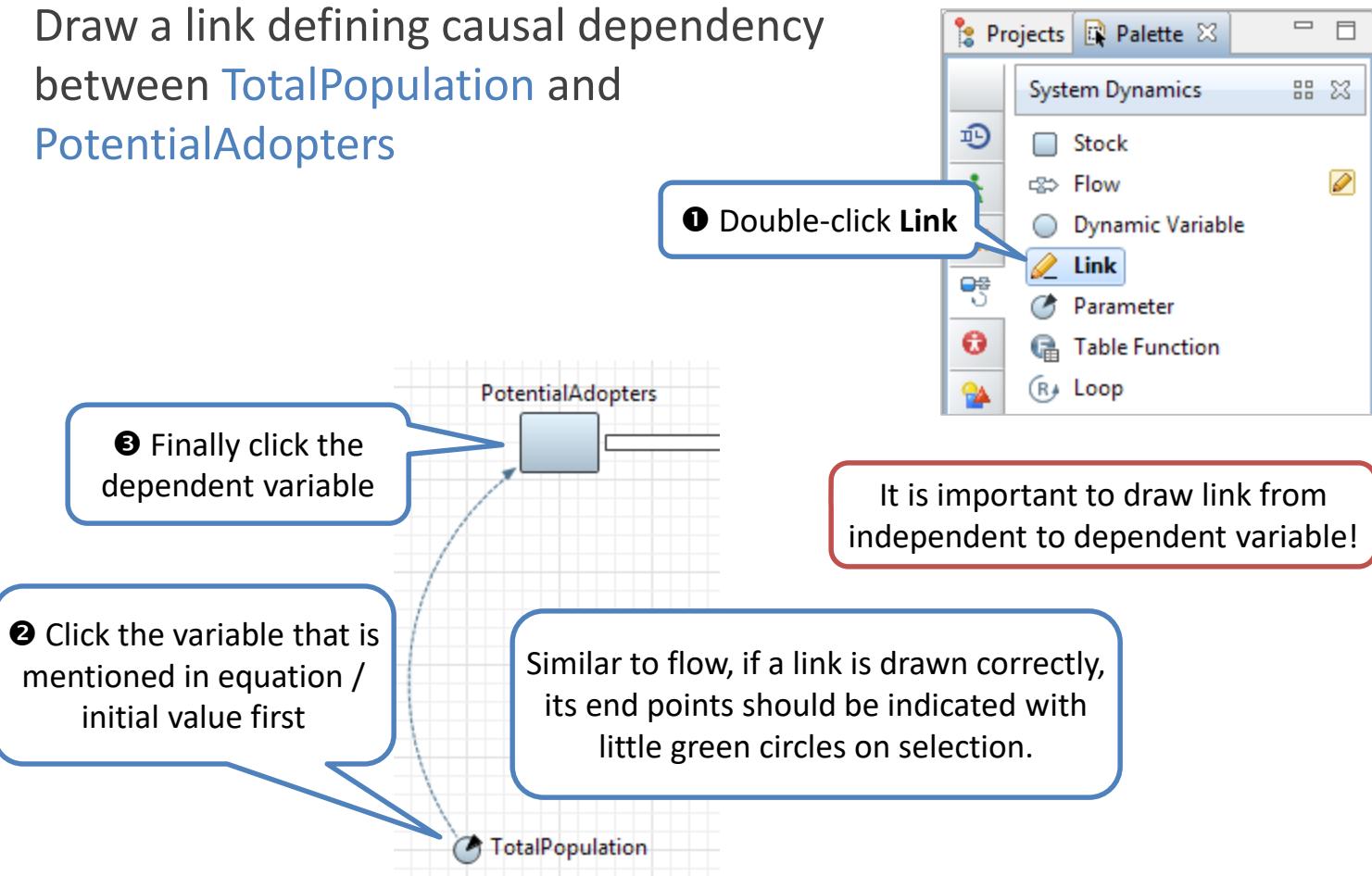
The screenshot shows the 'Properties' dialog for a stock named 'PotentialAdopters'. The 'Initial value:' field contains the variable 'TotalPopulation', which is highlighted with a red rectangular box. A blue callout bubble points from this red box to a text message: 'You may see an error occurred since a variable is mentioned in an expression but the dependency between these two variables is not defined in stock-flow-diagram.'

You may see an error occurred since a variable is mentioned in an expression but the dependency between these two variables is not defined in stock-flow-diagram.



Bass Diffusion. Phase 1. Step 6

- Draw a link defining causal dependency between TotalPopulation and PotentialAdopters



Bass Diffusion. Phase 1. Step 7

- Add Dynamic Variable AdoptionFromAd

AdoptionFromAd - Dynamic Variable

Name: AdoptionFromAd

Show name Ignore Visible on upper level

Visible: yes

Color:

Array Dependent Constant

AdoptionFromAd= ~~PotentialAdopters*AdEffectiveness~~

[Create a link from AdEffectiveness](#)

[Create a link from PotentialAdopters](#)

PotentialAdopters $\xrightarrow{\text{AdoptionRate}}$ Adopters

AdoptionFromAd

PotentialAdopters

AdoptionFromAd

AdEffectiveness

① Specify the formula and click inside the field

② Left-click the big error indicator

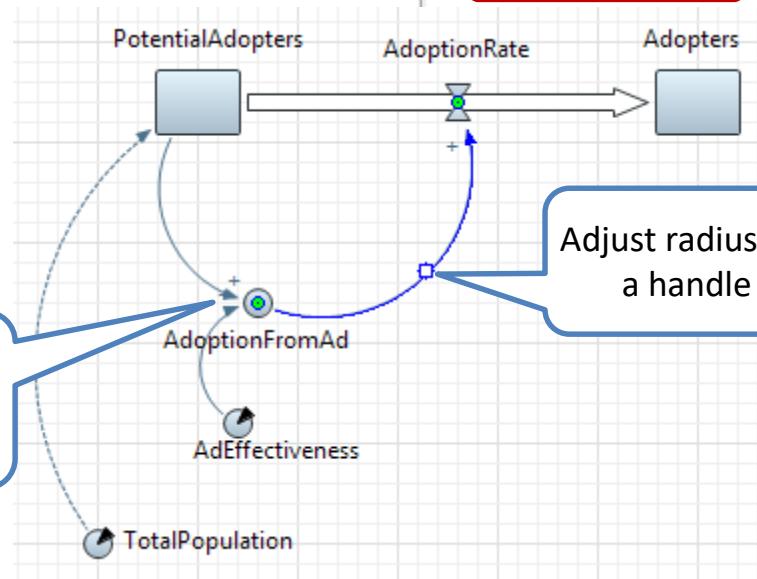
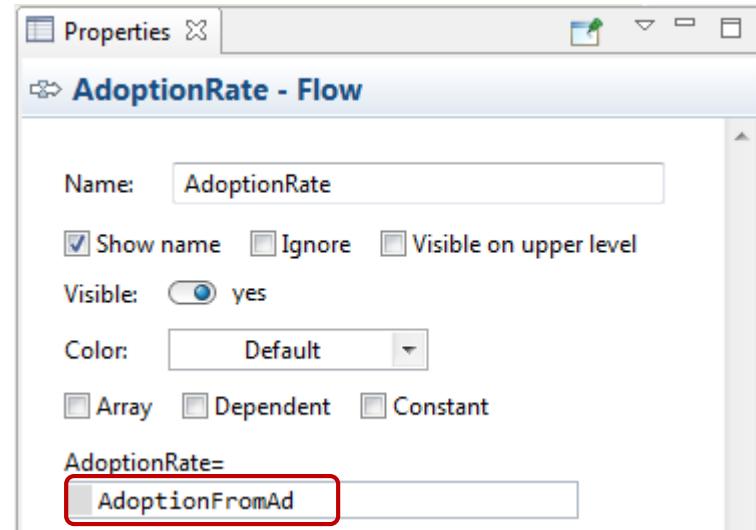
③ Create missing links by choosing corresponding menu items

You will see links automatically created



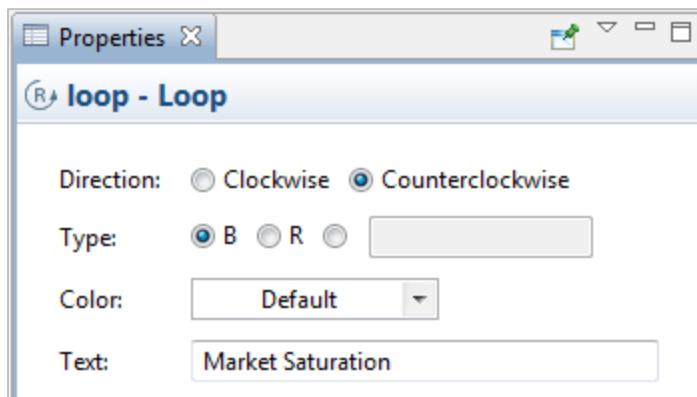
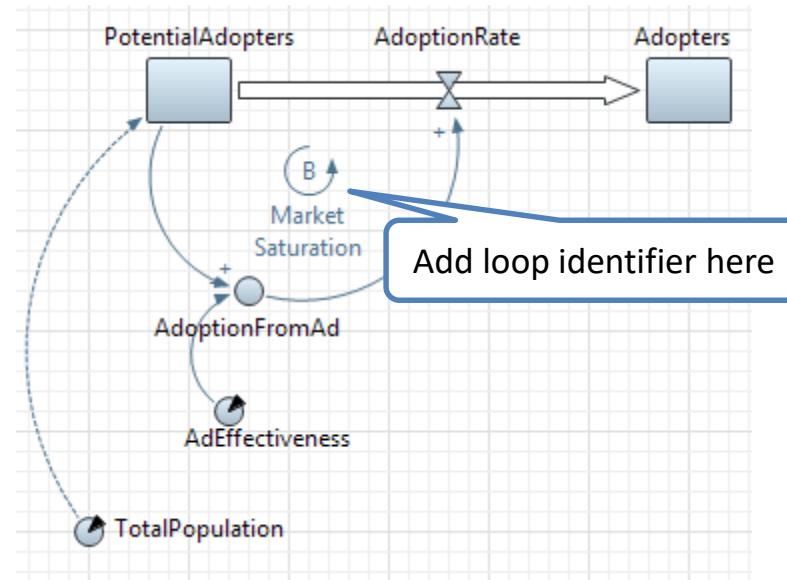
Bass Diffusion. Phase 1. Step 8

- Draw a link going from AdoptionFromAd to AdoptionRate
- Specify a formula for AdoptionRate



Bass Diffusion. Phase 1. Step 9

- Add **Loop** to indicate balancing causal loop caused by market saturation

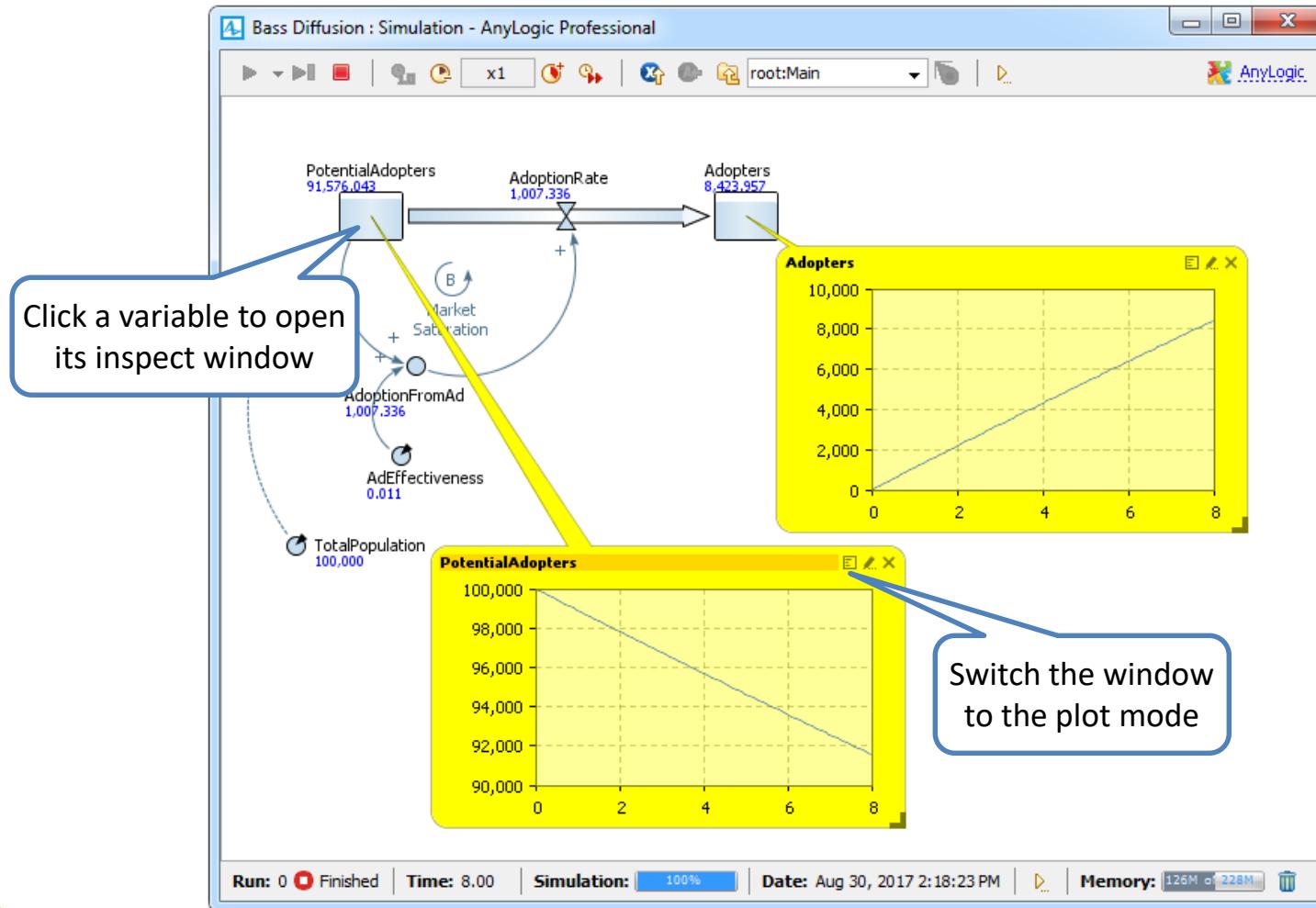


- Set the loop direction, type and specify a brief description



Bass Diffusion. Phase 1. Step 10

- Run the model and observe the dynamics using inspect windows



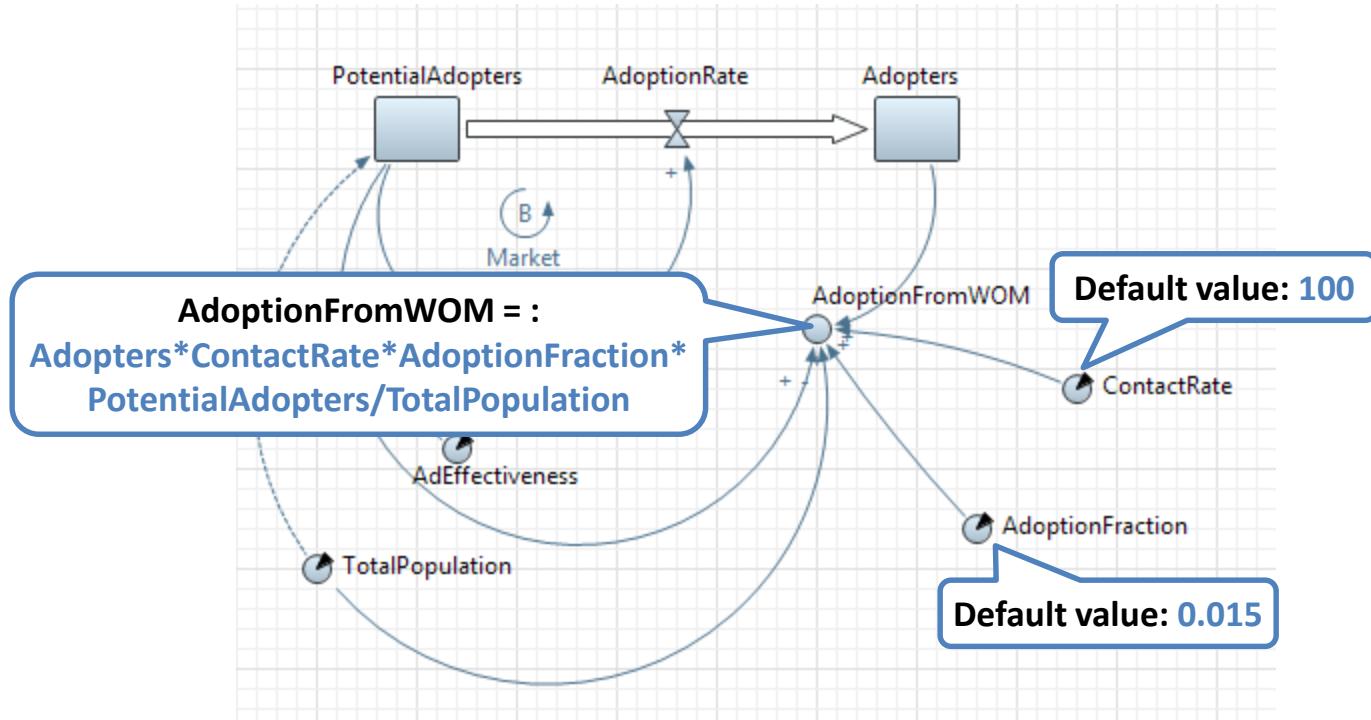
Bass Diffusion. Phase 2

- Now add the effect of word of mouth on product diffusion
- Assume everybody contacts everybody else
- The number of person's contacts per time unit is **Contact Rate**
- If an adopter contacts a potential adopter, the last one will adopt with probability **Adoption Fraction**
- The corresponding part of the **Adoption Rate** will be:
$$\text{Adoption From Word of Mouth} = \text{Adopters} * \text{Contact Rate} * \text{Adoption Fraction} * \frac{\text{Potential Adopters}}{\text{TotalPopulation}} - \text{Why?}$$
- Parameter values:
 - $\text{Contact Rate} = 100$
 - $\text{Adoption Fraction} = 0.015$



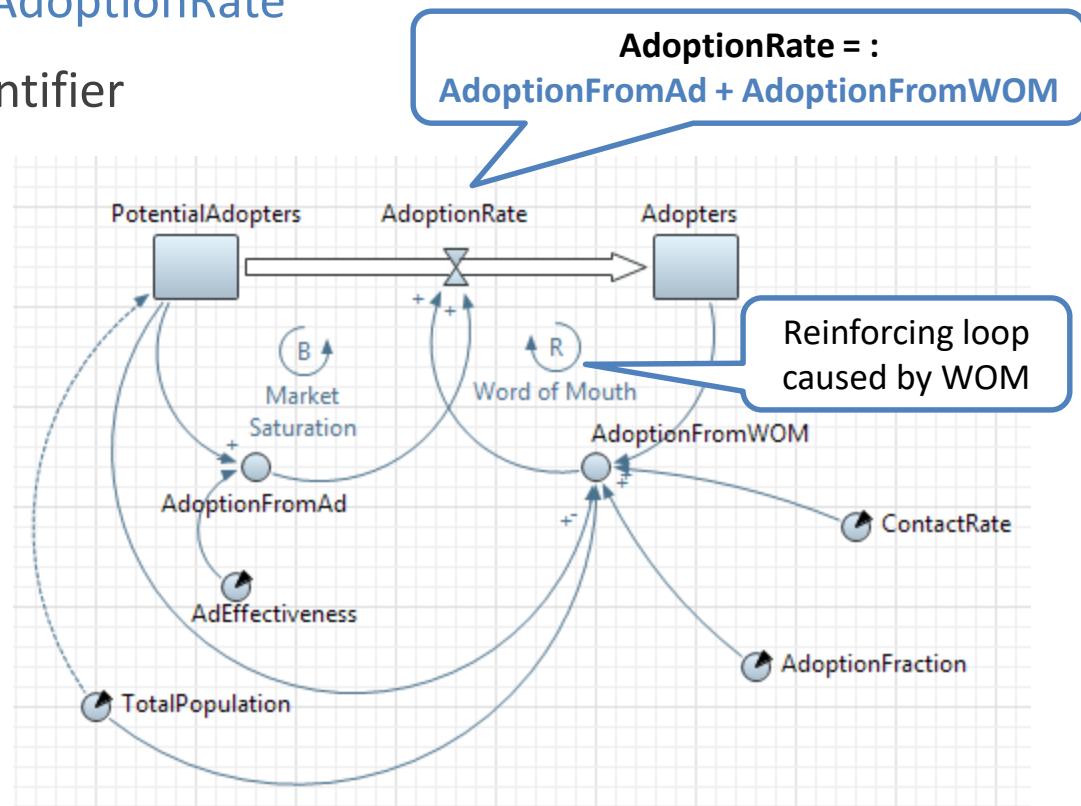
Bass Diffusion. Phase 2. Step 1

- Add `ContactRate` and `AdoptionFraction` parameters
- Add `AdoptionFromWOM` dynamic variable
- Draw dependency links

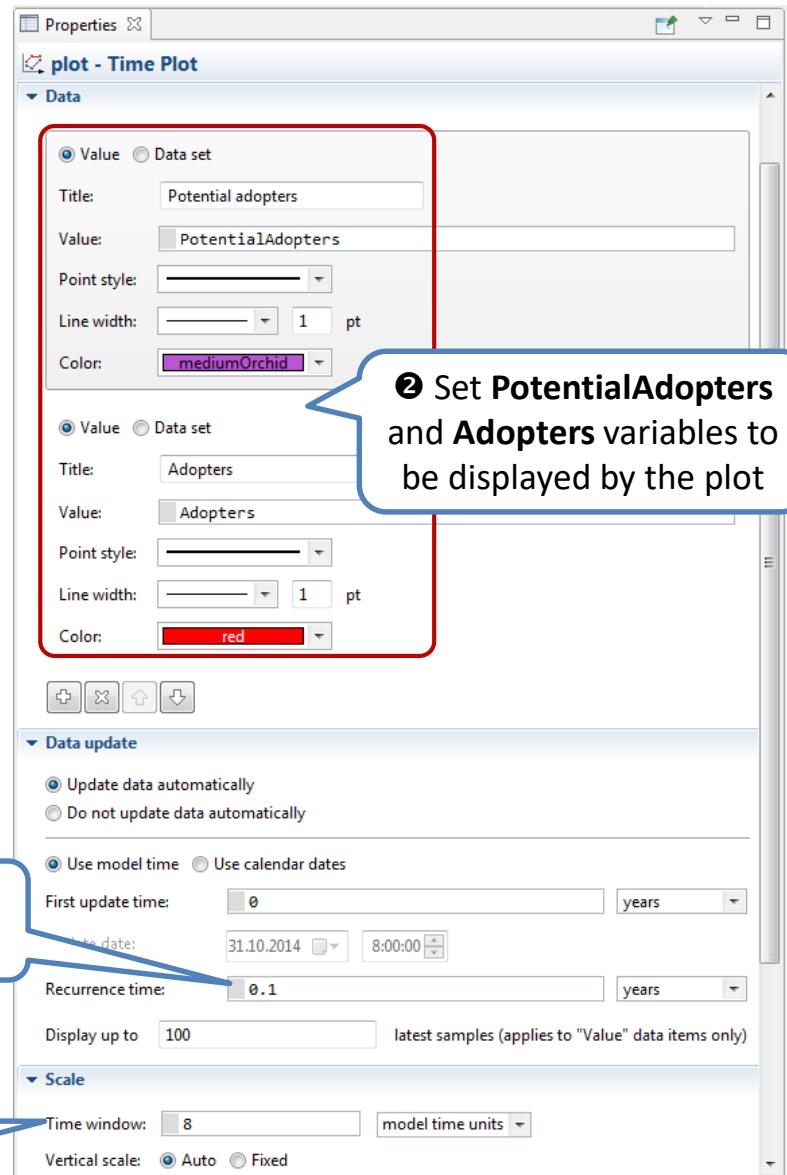
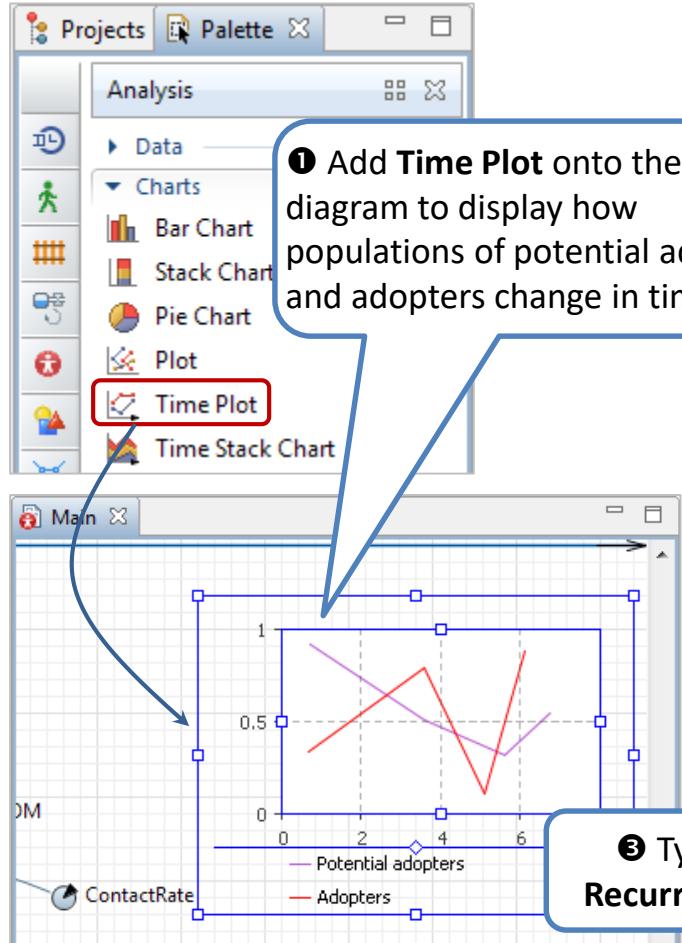


Bass Diffusion. Phase 2. Step 2

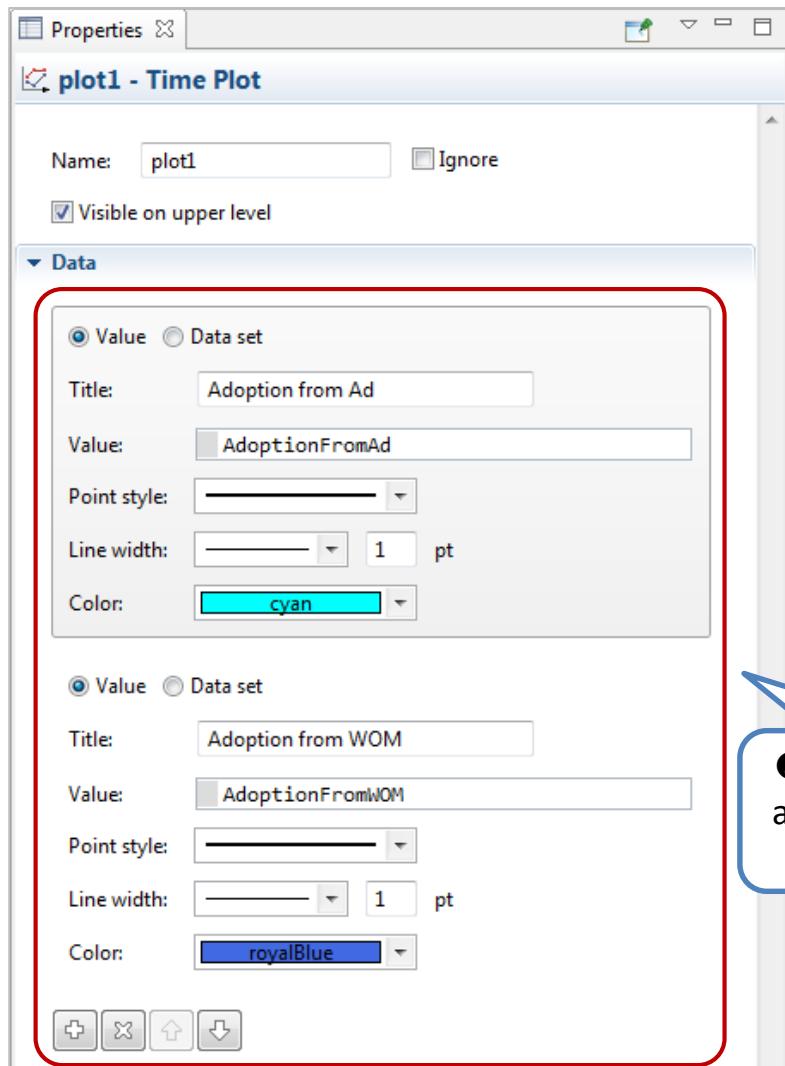
- Draw a link going from AdoptionFromWOM to AdoptionRate
- Modify the formula of AdoptionRate
- Add one more loop identifier



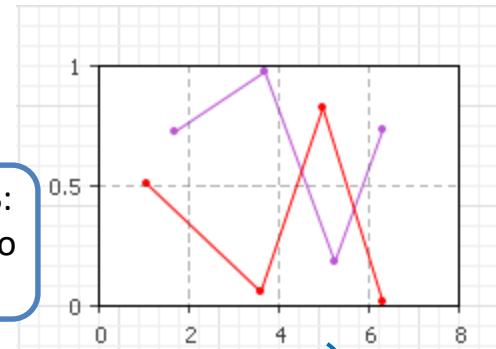
Bass Diffusion. Phase 2. Step 3



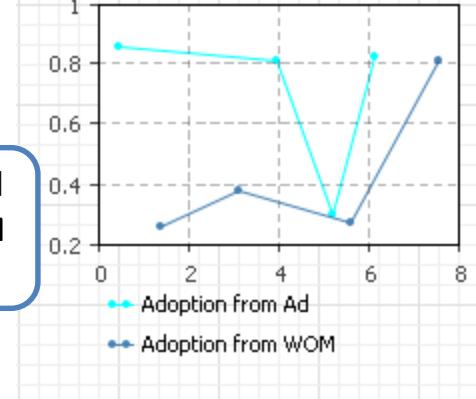
Bass Diffusion. Phase 2. Step 4



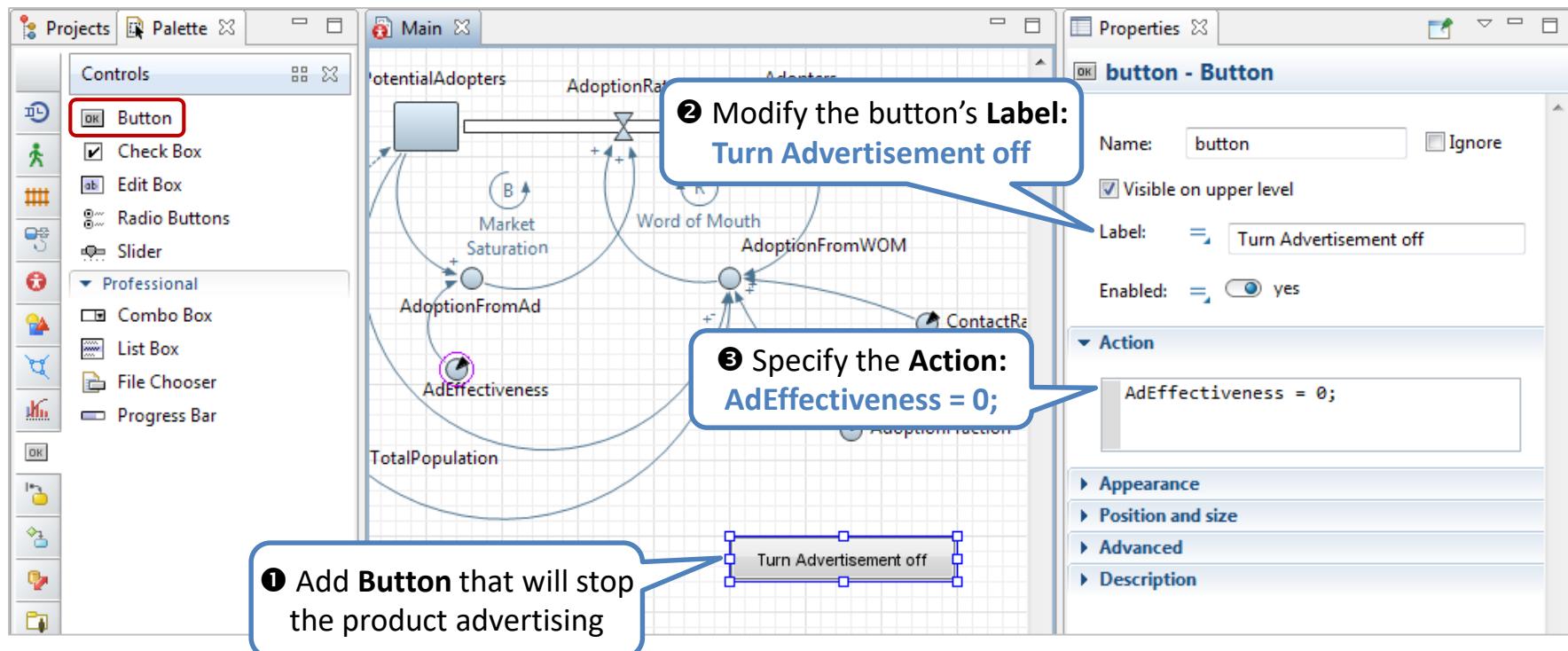
① Ctrl+drag (Mac OS: Cmd+drag) the plot to make a copy



② Add AdoptionFromAd and AdoptionFromWOM variables onto the plot

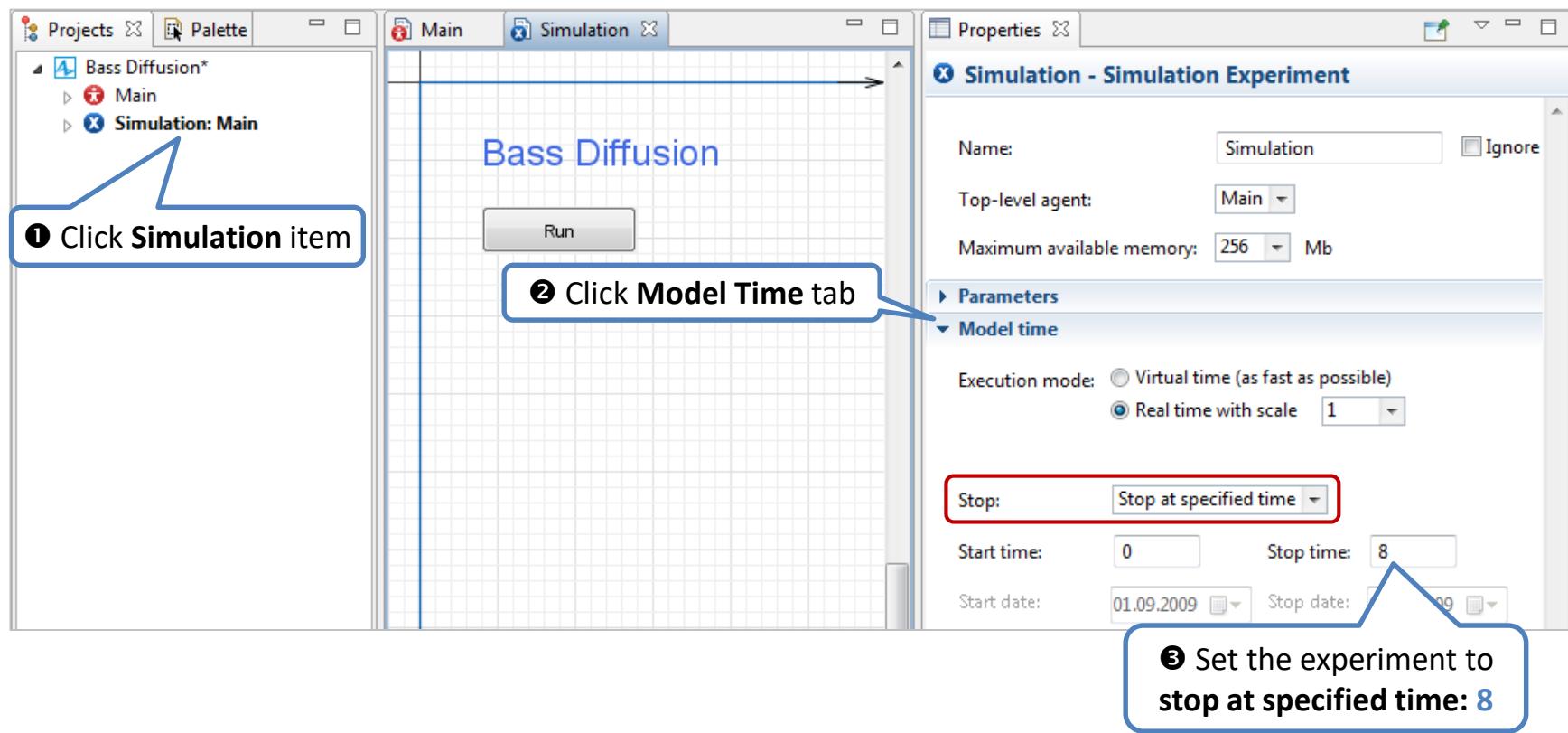


Bass Diffusion. Phase 2. Step 5



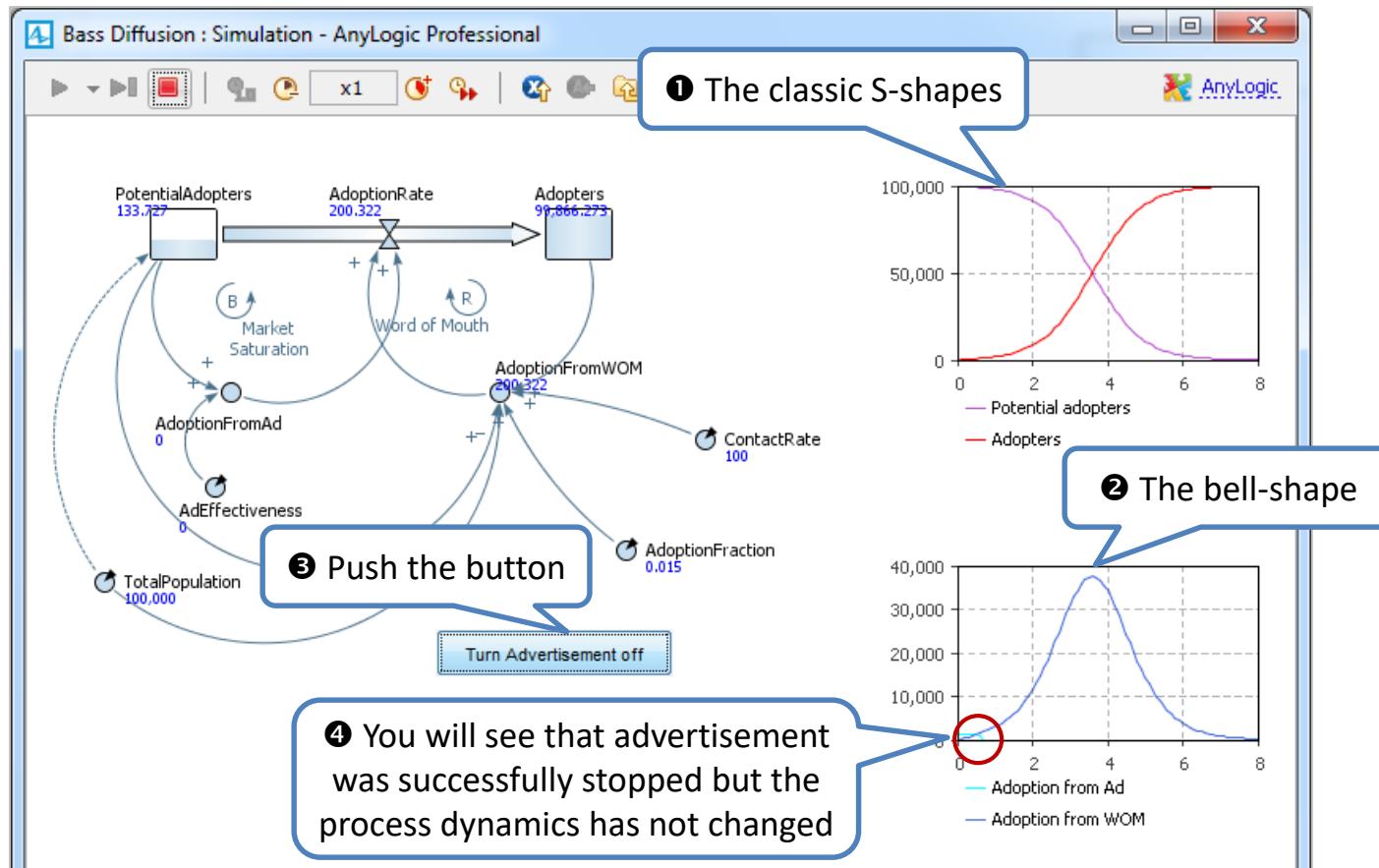
Bass Diffusion. Phase 2. Step 6

- Set the model to stop after 8 time units



Bass Diffusion. Phase 2. Step 7

- Run the model and observe the dynamics using plots
- Stop advertisement and watch how it affected the adoption process



Bass Diffusion. Phase 2. Questions

1. In the 80s, he was the System Dynamics enthusiast:
 - (a) Boris Johnson, The Mayor of London
 - (b) Milos Zeman, Czech Republic President
 - (c) Anthony Lake, Executive Director of the UNICEF
2. Please explain why one loop is balancing and another – reinforcing.



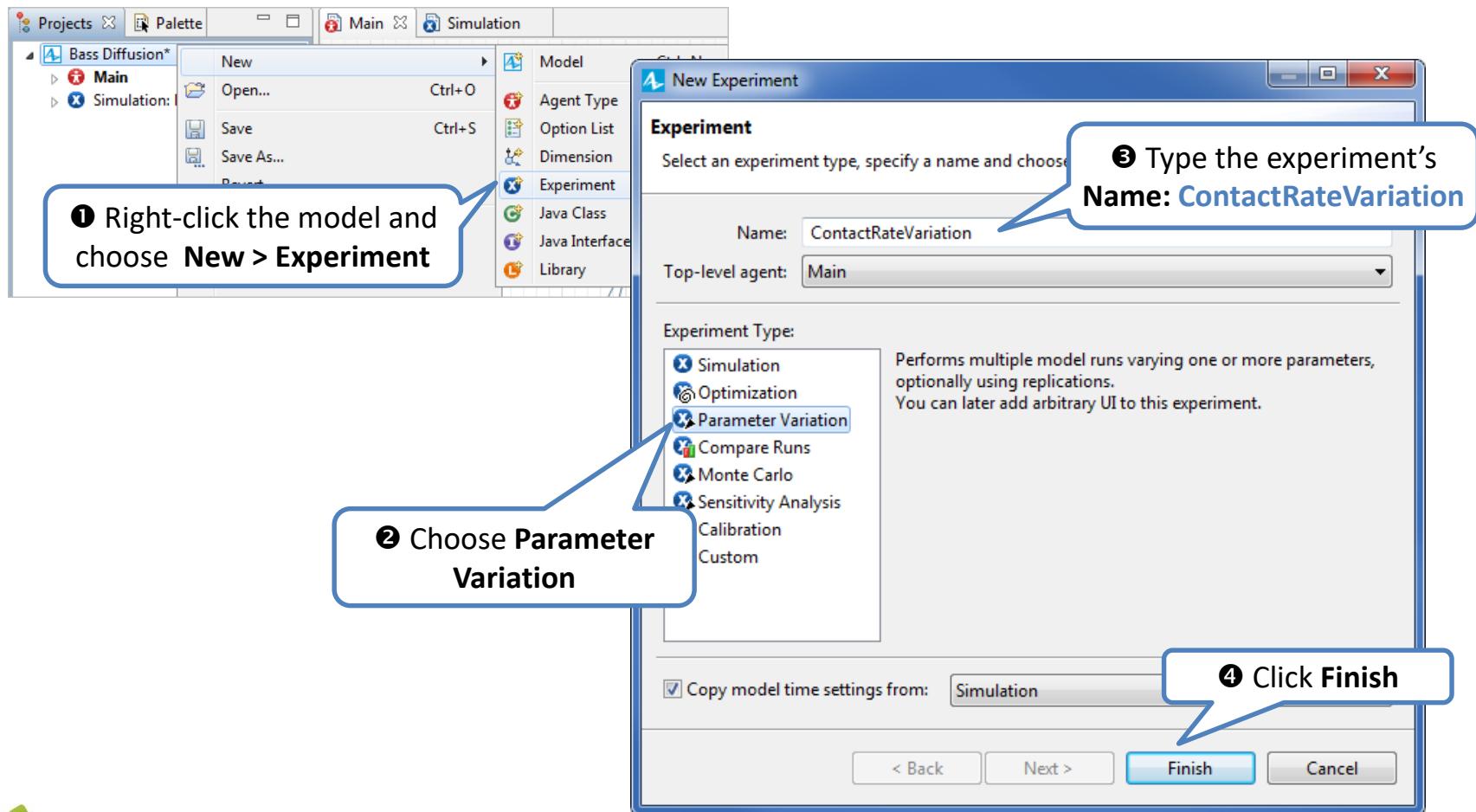
Bass Diffusion. Phase 3

- Now we want to study how adoption process depends on the adopters' contact rate
- We will use **parameter variation experiment**
- This experiment performs multiple model runs varying one or more parameters of the model



Bass Diffusion. Phase 3. Step 1

- Create new parameter variation experiment



Bass Diffusion. Phase 3. Step 2

- Set up the experiment to vary ContactRate in the range from 0 to 100 with step 10

The screenshot shows the AnyLogic software interface with two main windows: 'ContactRateVariation' and 'Properties'.

ContactRateVariation Window:

- Run** button.
- Iteration:** Input field with a question mark.
- Parameters:**

TotalPopulation	?
ContactRate	?
AdEffectiveness	?
AdoptionFraction	?

A blue callout bubble with the text "③ You will see controls appeared here" points to the iteration input field.

Properties Window:

- Name:** ContactRateVariation
- Ignore** checkbox.
- Top-level agent:** Main dropdown.
- Maximum available memory:** Input field with a question mark.
- Create default UI** button.

A blue callout bubble with the text "② Click Create Default UI button" points to the 'Create default UI' button.

Parameter Variation Experiment Settings:

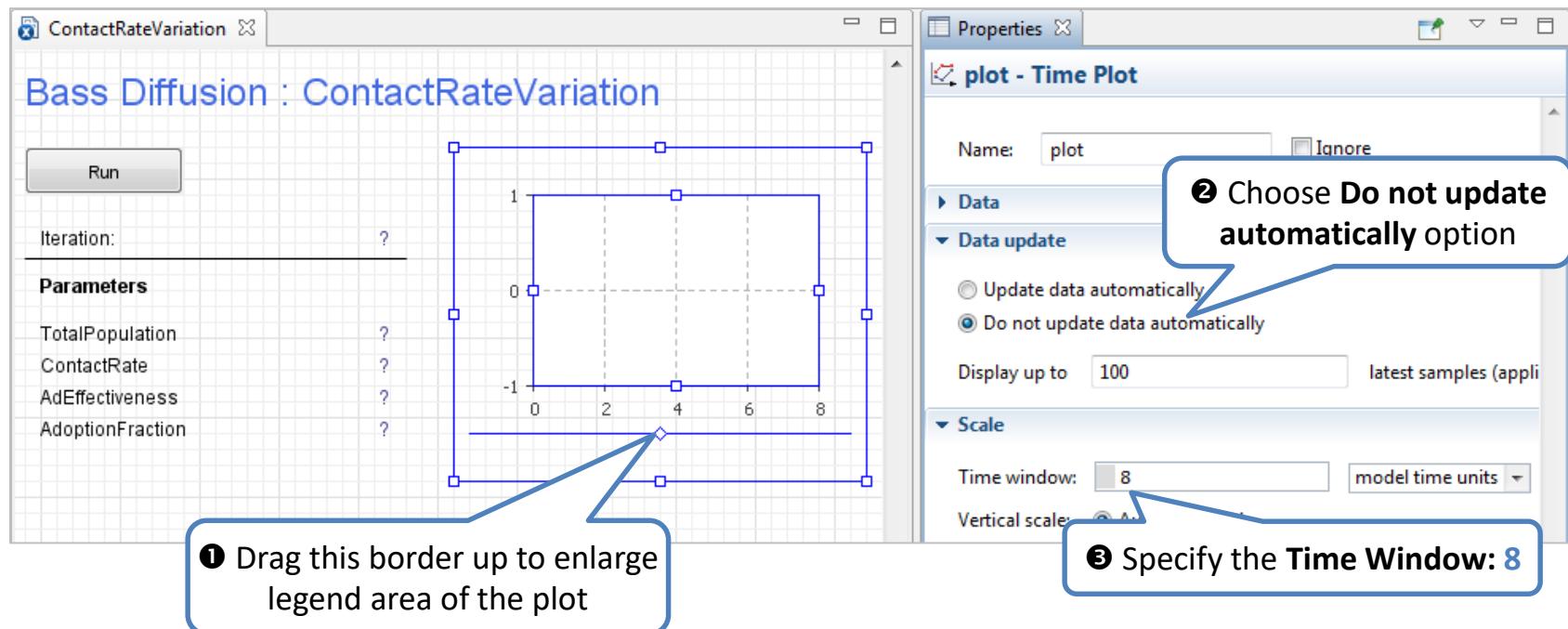
- Parameters:** Varied in range (radio button selected).
- Number of runs:** 10.
- Table:** A grid showing parameter settings:

Parameter	Type	Value		
		Min	Max	Step
TotalPopulation	Fixed	100000		
ContactRate	Range	0	100	10
AdEffectiveness	Fixed	0.011		
AdoptionFraction	Fixed	0.015		



Bass Diffusion. Phase 3. Step 3

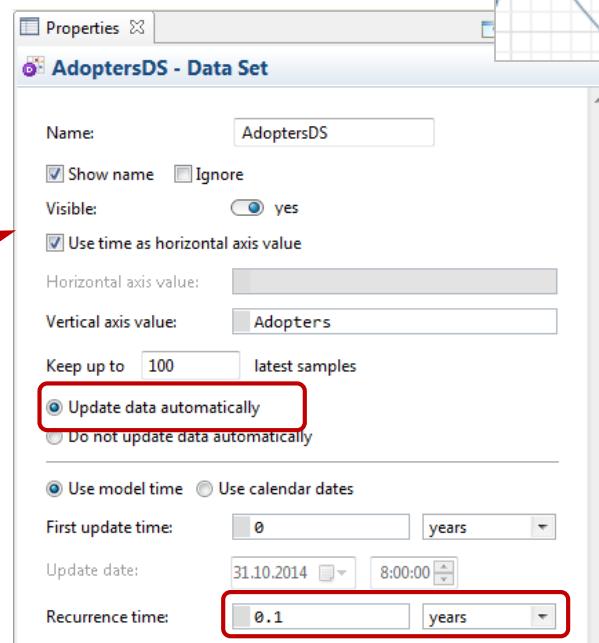
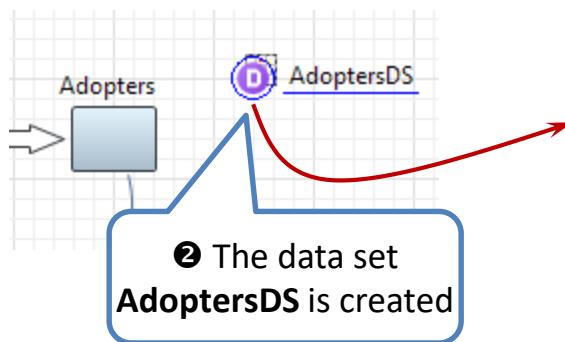
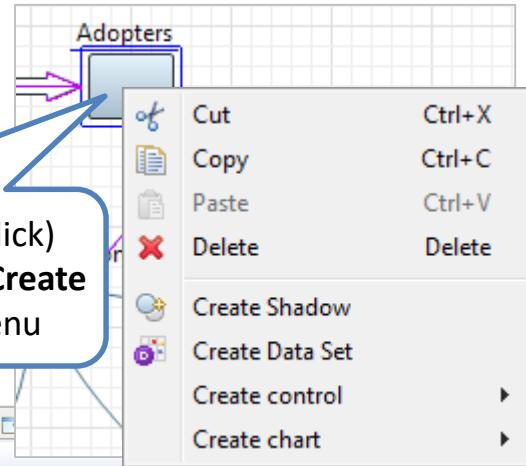
- Place time plot on the diagram of ContactRateVariation experiment



Bass Diffusion. Phase 3. Step 4

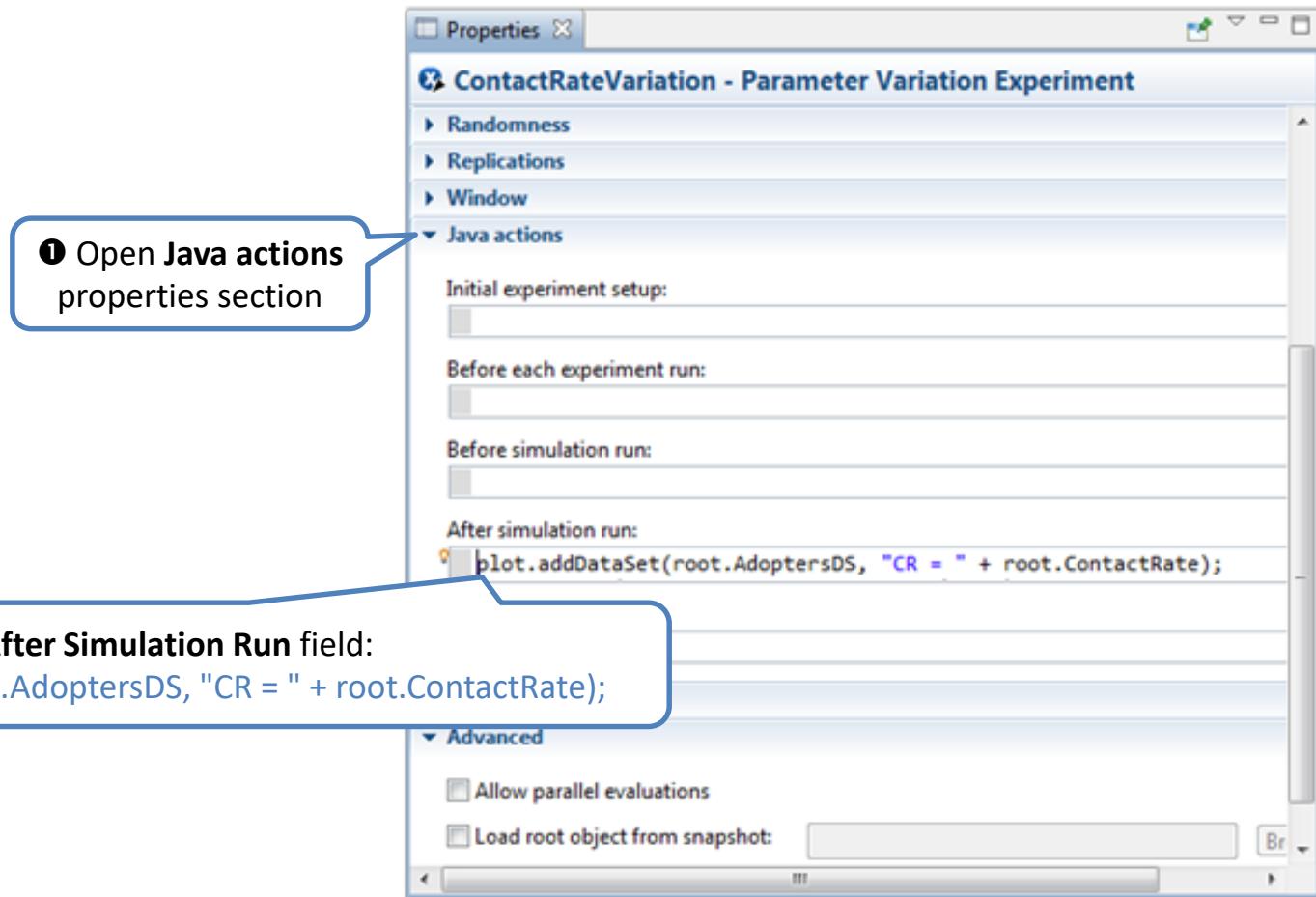
- Add a data set to store history of **Adopters** variable

① Right-click (Mac OS: Ctrl+click) **Adopters** variable and choose **Create Data Set** from the popup menu



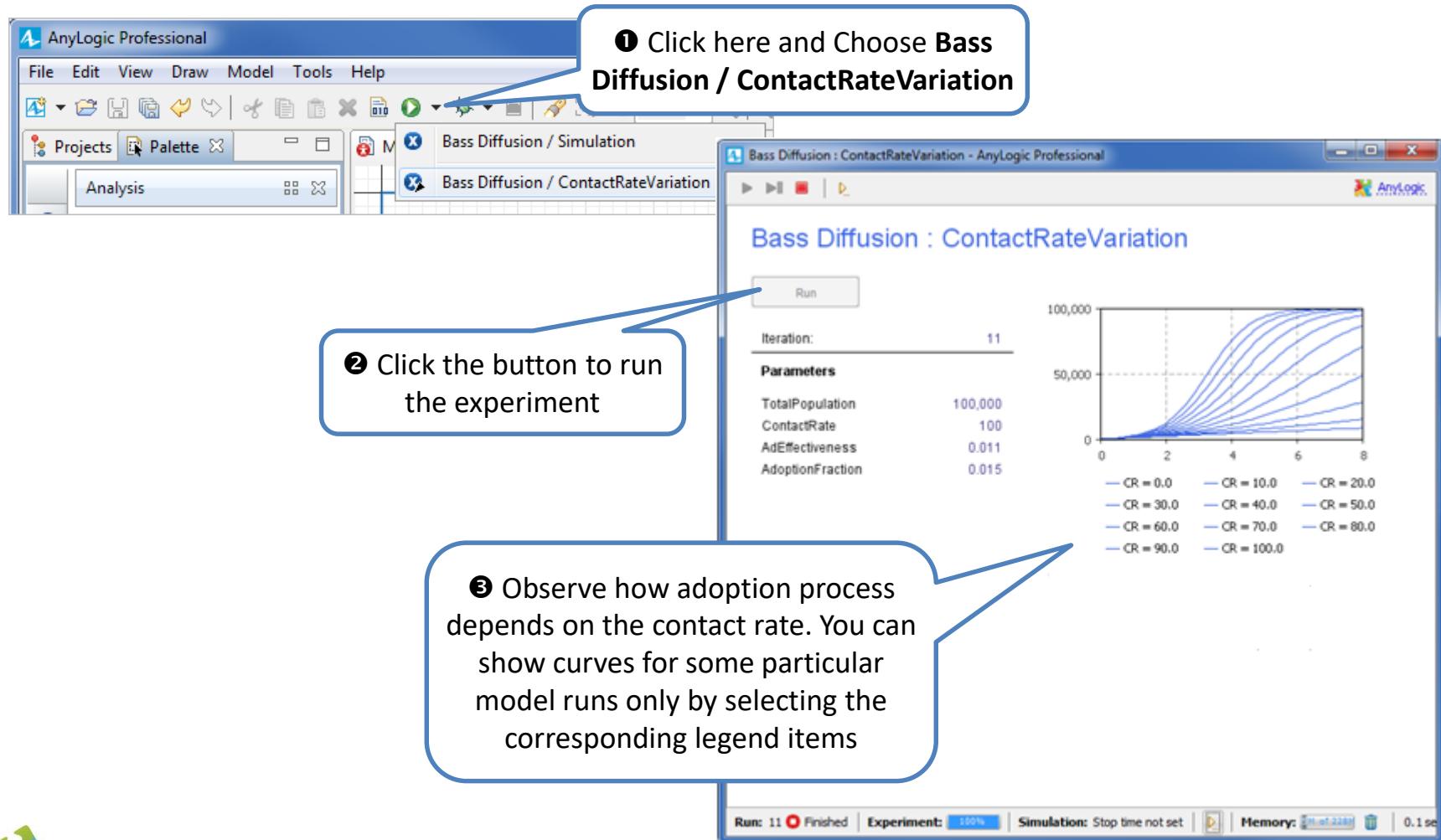
Bass Diffusion. Phase 3. Step 6

- Modify the properties of *ContactRateVariation* experiment to update plot with new values after each simulation run



Bass Diffusion. Phase 3. Step 7

- Run the model



Bass Diffusion. Phase 3. Questions

1. If Contact Rate grows then what happens with the final number of Adopters?
2. If Contact Rate equals to zero then at the end there are:
 - (a) 0 adopters
 - (b) more than 1
 - (c) more than half of all people



Exercise. SEIR Model

Model dynamics of contagion:

- A **Susceptible** person, being infected, becomes **Exposed**, then after incubation time, **Infectious**, and then recovers
- Both Exposed and Infectious people can pass the disease to Susceptible, but they have different infectivity (probability of passing disease during contact) and different contact rate
- **Recovered** become immune to the disease
- Parameters:
 - TotalPopulation: 10000
 - ContactRateExposed: 4, ContactRateInfectious: 1.25
 - InfectivityExposed: 0.05, InfectivityInfectious: 0.06
 - AverageIncubationTime: 10 days
 - AverageIllnessDuration: 15 days
- Initially: one person is Exposed, all others – Susceptible



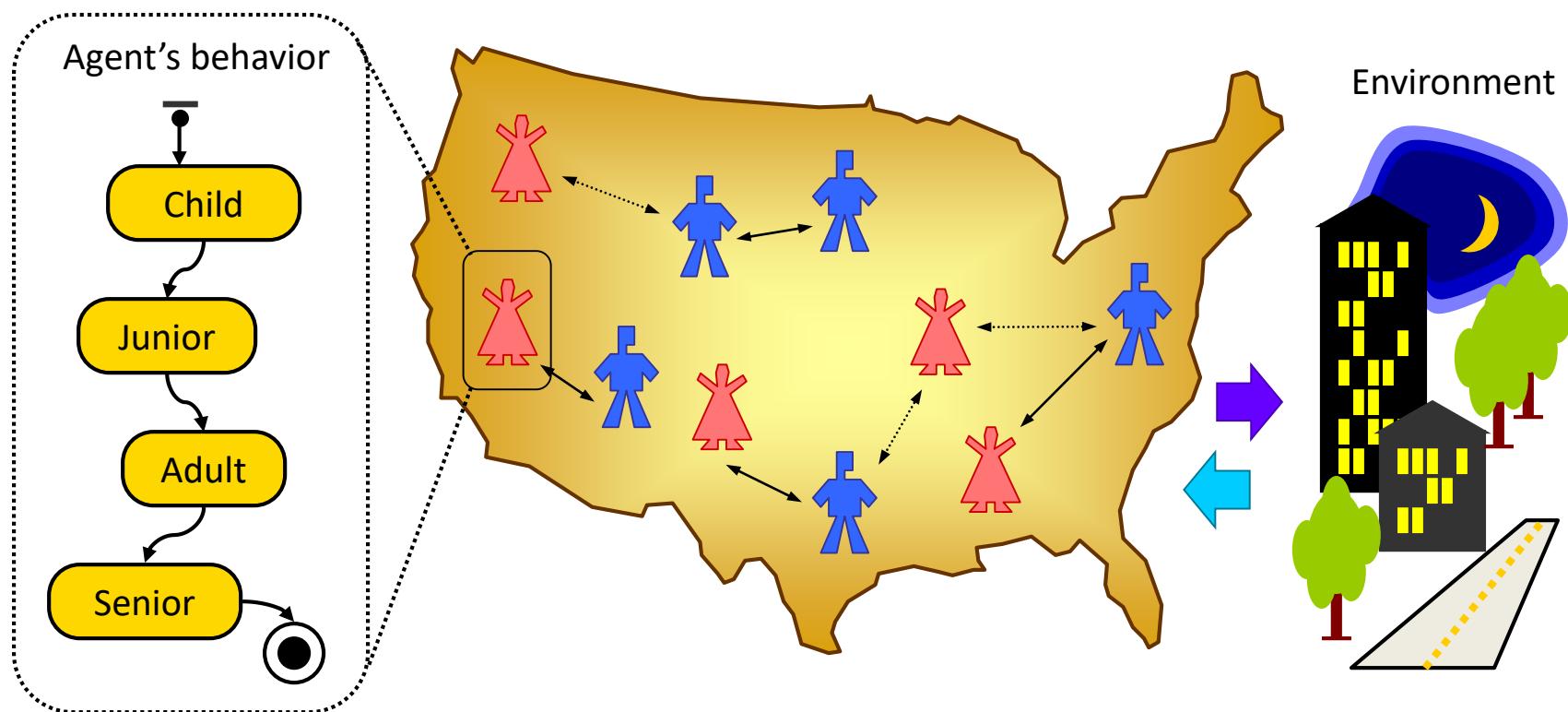
Agent Based Modeling in AnyLogic

This presentation is a part of
AnyLogic Standard Training Program



Agent based modeling

- We focus on individual objects and describe their local behavior, local rules
 - Sometimes, we also model the dynamics of the environment



Agents can be:

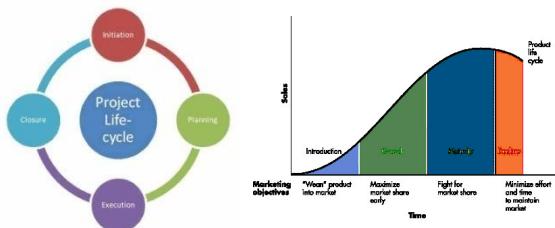
People:

consumers, habitants, employees, patients, doctors, clients, soldiers, ...



Non-material things:

projects, products, innovations, ideas, investments ...



Vehicles, equipment:

trucks, cars, cranes, aircrafts, rail cars, machines, ...

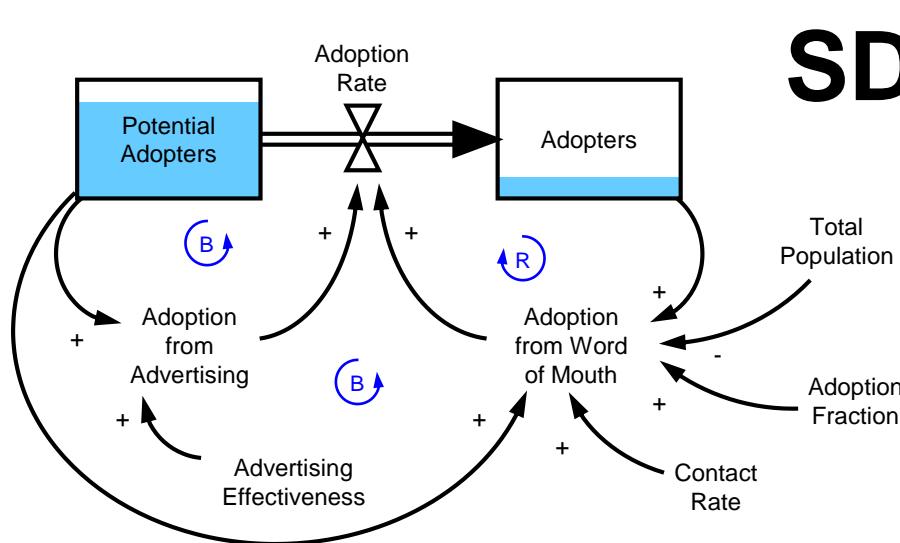


Organizations:

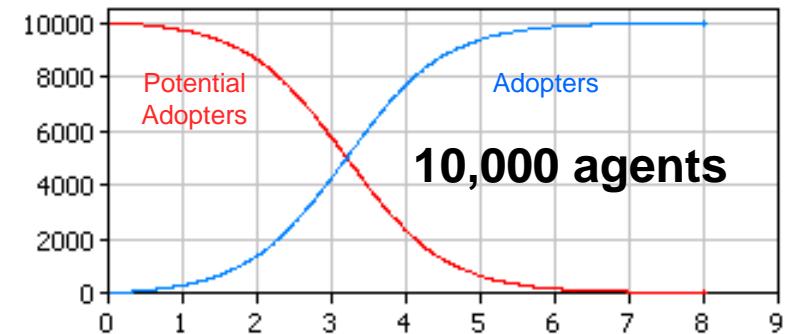
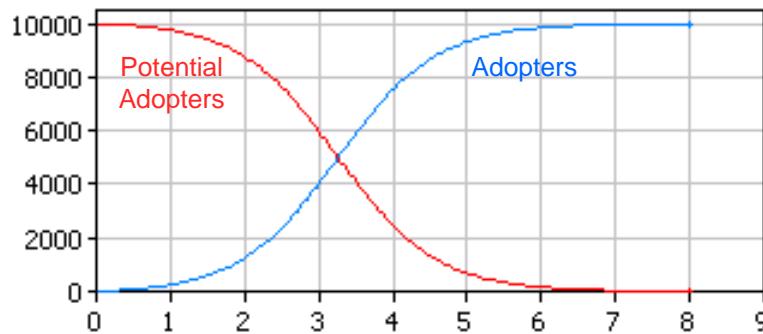
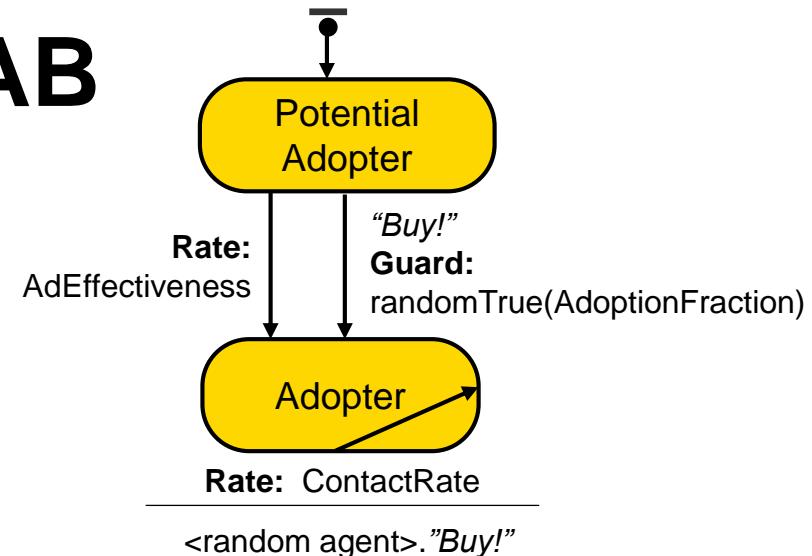
companies, political parties, countries, ...



Bass Diffusion – Agent Based version

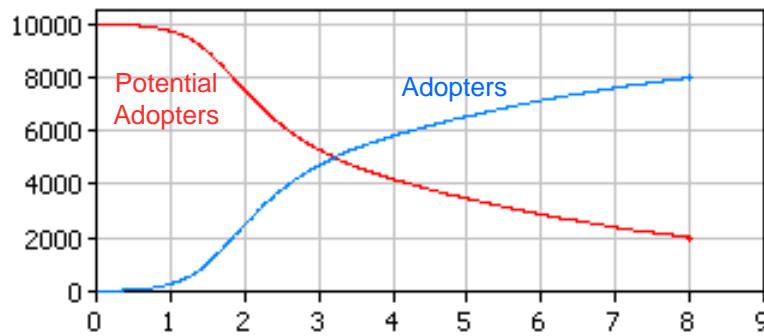
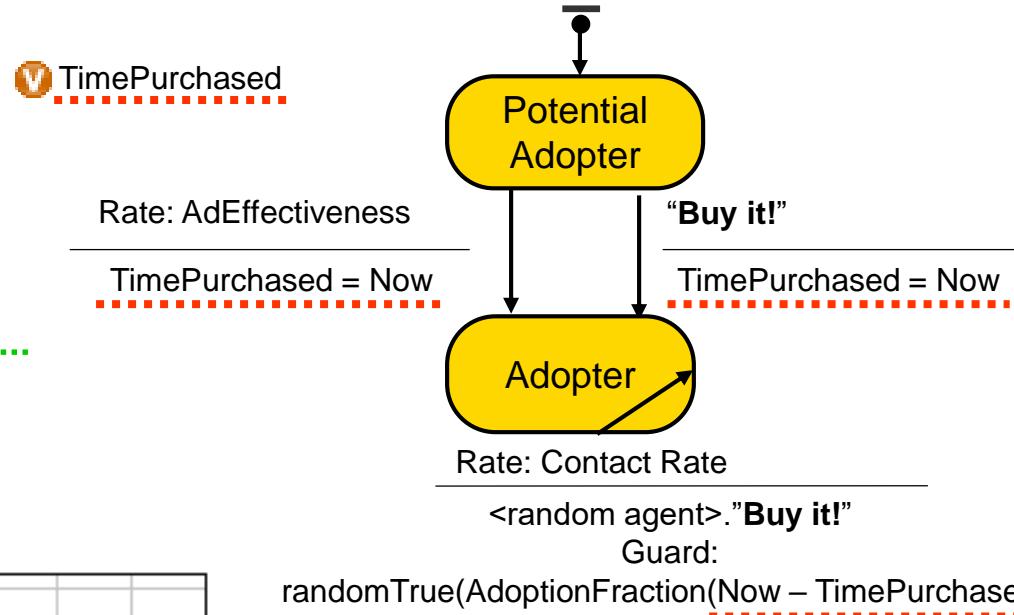
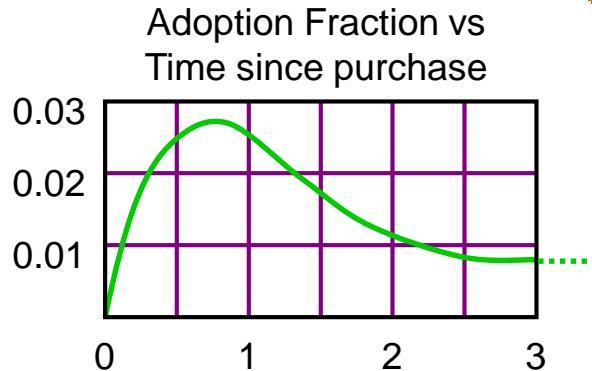


SD | AB



Capturing more with AB Model

- Let the word-of-mouth influence of an adopter depend on how recently he has purchased



- Can you build an SD model that captures such dynamics?

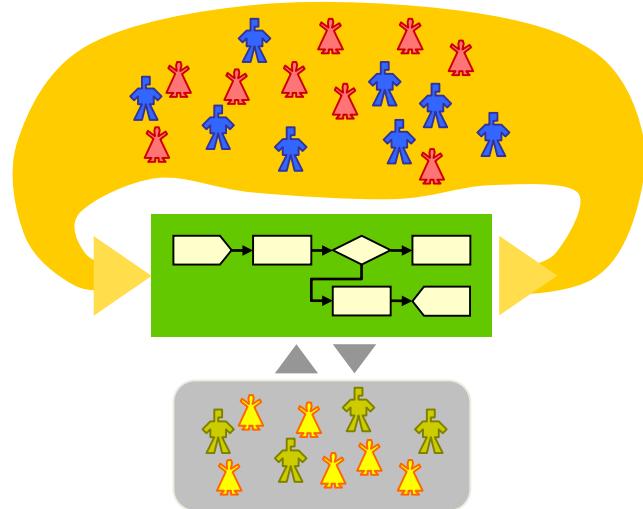
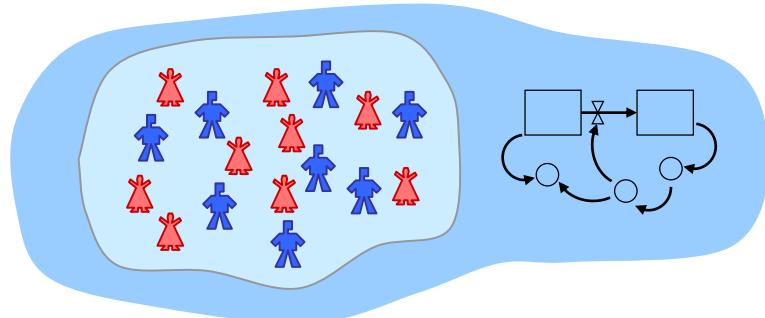
Which approach to use?

- If the problem requirements fit well into the DE or SD modeling paradigms – you can safely use these *traditional approaches*
- In cases where your system contains active objects (people, business units, animals, vehicles, or projects, stocks, products, etc.) with timing, event ordering or other kind of individual, autonomous behavior – *You will benefit from applying the AB approach*
- Sometimes these requirements are at the sub-model level. Then you can consider mixing different approaches in one model and applying most appropriate technique where needed



Multi-paradigm model architectures

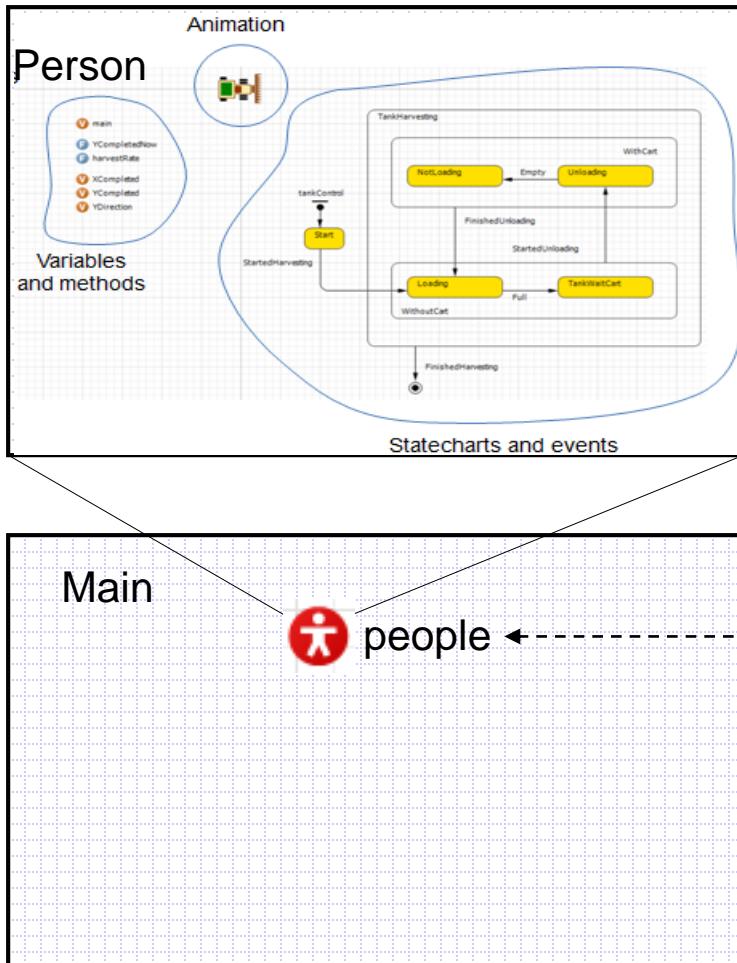
Agents (e.g. customers) interact with other agents (staff) in a Discrete Event flowchart



Agents live in an Environment modeled in System Dynamics way



Typical architecture of AnyLogic AB model



Name: *people*
Type: *Person*
Replication: *100000*

Adding/removing people:

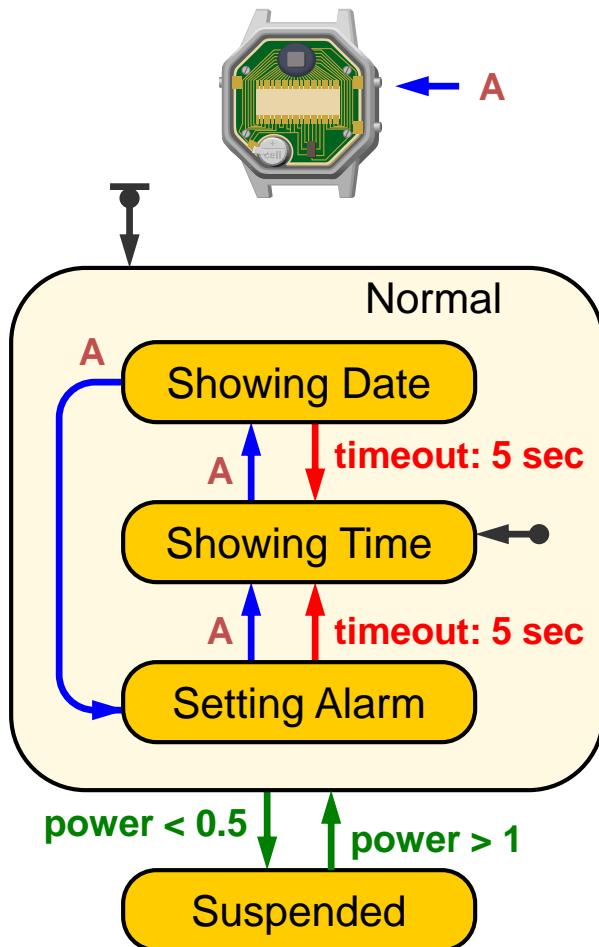
```
add_people();  
remove_people( p );
```

Iterating through all people:

```
for( Person p : people ) {  
    ...  
}
```



Statecharts



- The most powerful and naturally visual construct
- Statecharts can be used to define:
 - object states / modes of operation
 - response to the external or internal signals and conditions
 - event and time ordering



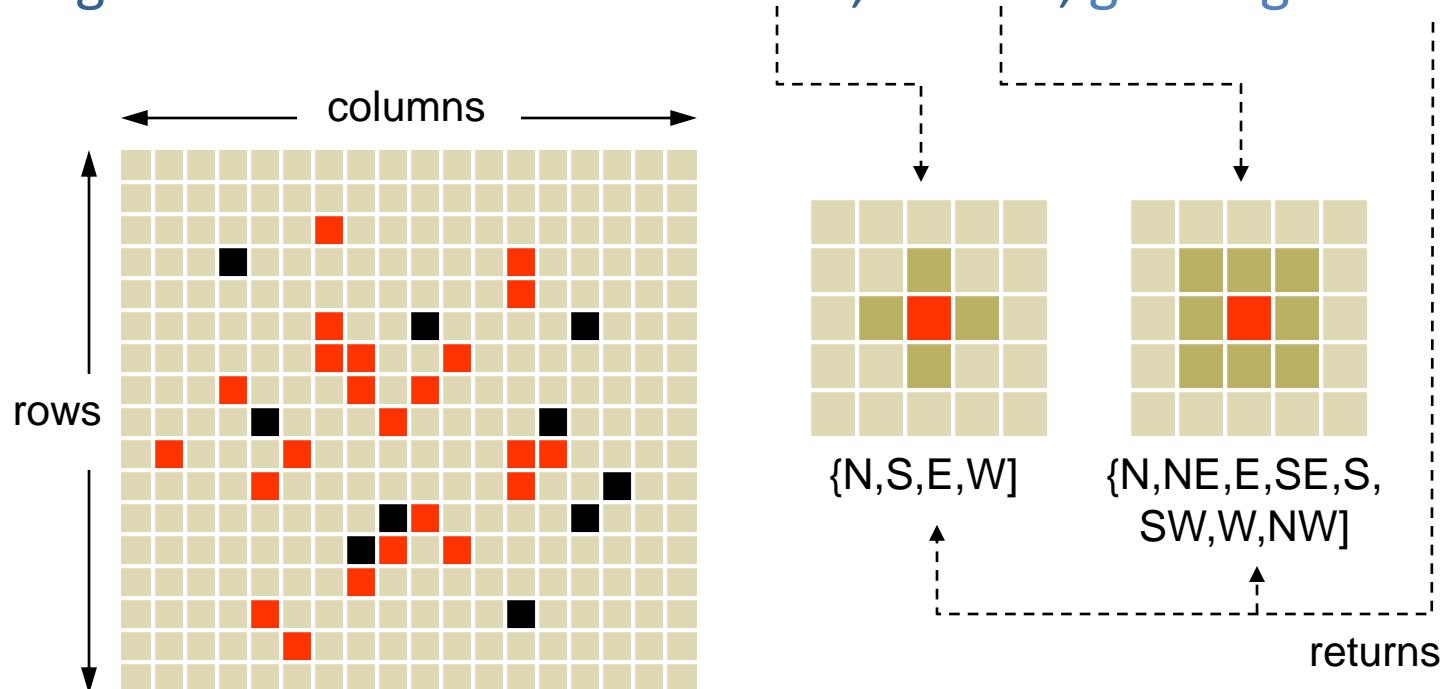
Space: Discrete

- 2D array of cells *Rows by Columns*

At most one agent per cell; to retrieve location: `getR()`, `getC()`

Movement: `jumpToCell()`, `jumpToRandomCell()`, etc.

Neighborhood models: Euclidean, Moore; `getNeighbors()`



Space: Continuous

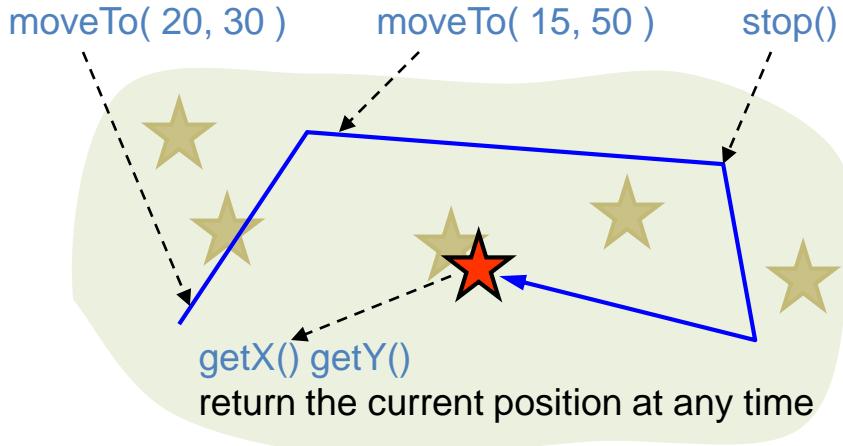
- Agent has real (x, y, z) coordinates in 3D space

- Use agent API:

*getX(); getY();
distanceTo(agent);
moveTo(x, y, z);
jumpTo(x, y);
stop();
isMoving();
timeToArrival();
setVelocity(v);*

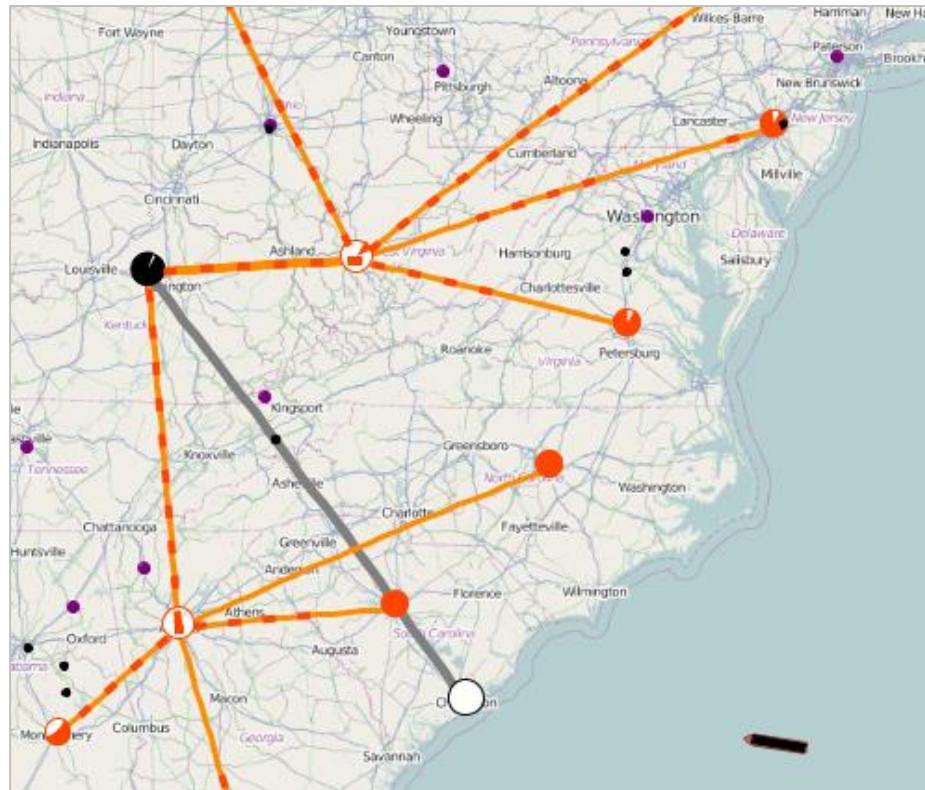
- Define action:

On arrival



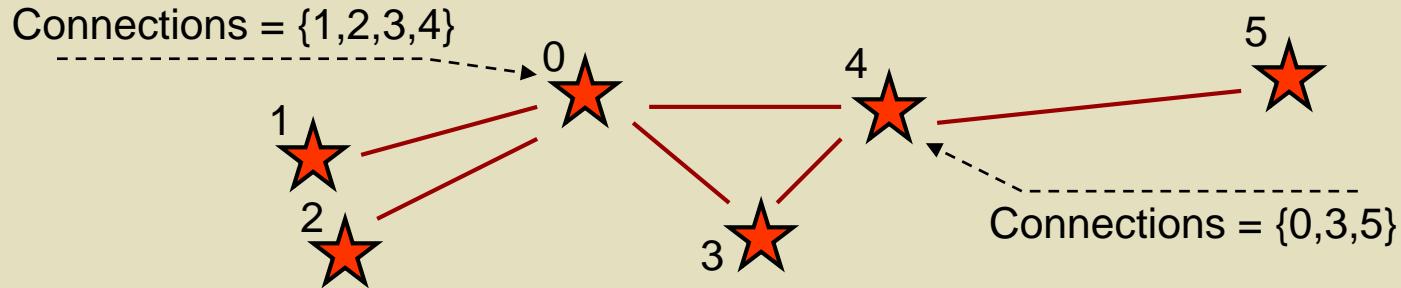
Space: GIS

- Agents in a geospatial environment defined by GIS map
- Agent has real (latitude, longitude) coordinates in space
- Use agent functions:
*getLatitude();
getLongitude();
distanceTo(agent);
jumpTo(x, y);
moveTo(x, y);
stop();
isMoving();
timeToArrival();
setVelocity(v);*
- Define action:
On arrival



Network: connections and communication

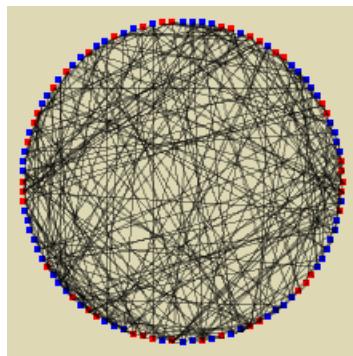
- Every agent has a list of connections – other agents
 - Use standard network types or define your own using API:
`connectTo(agent); disconnectFrom(agent);`
 - Access the collection of connected agents:
`getConnections(); getConnectedAgent(i);`
 - Communication in network:
Send messages:
`sendToAll(msg); sendToRandom(msg); sendToAllConnected(msg);
sendToRandomConnected(msg); send(msg, agent)`
- Define reaction in *connections* element: On message received



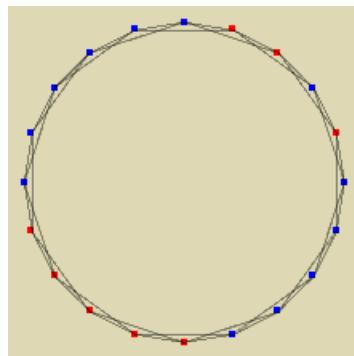
Network: Standard types

- Standard network types:

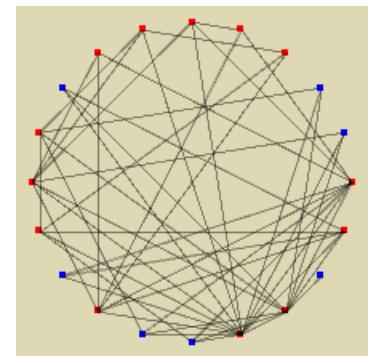
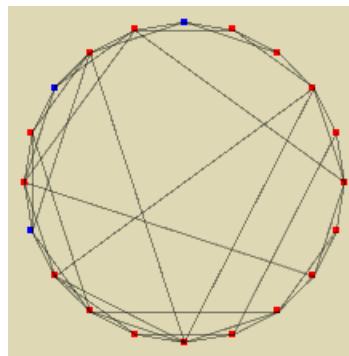
Random, Ring lattice,



Small world,

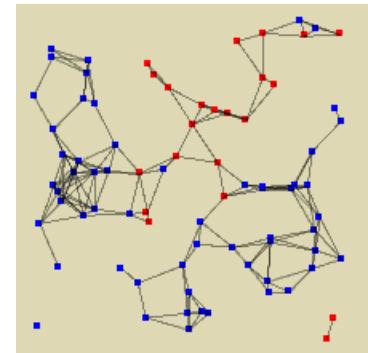


Scale free,



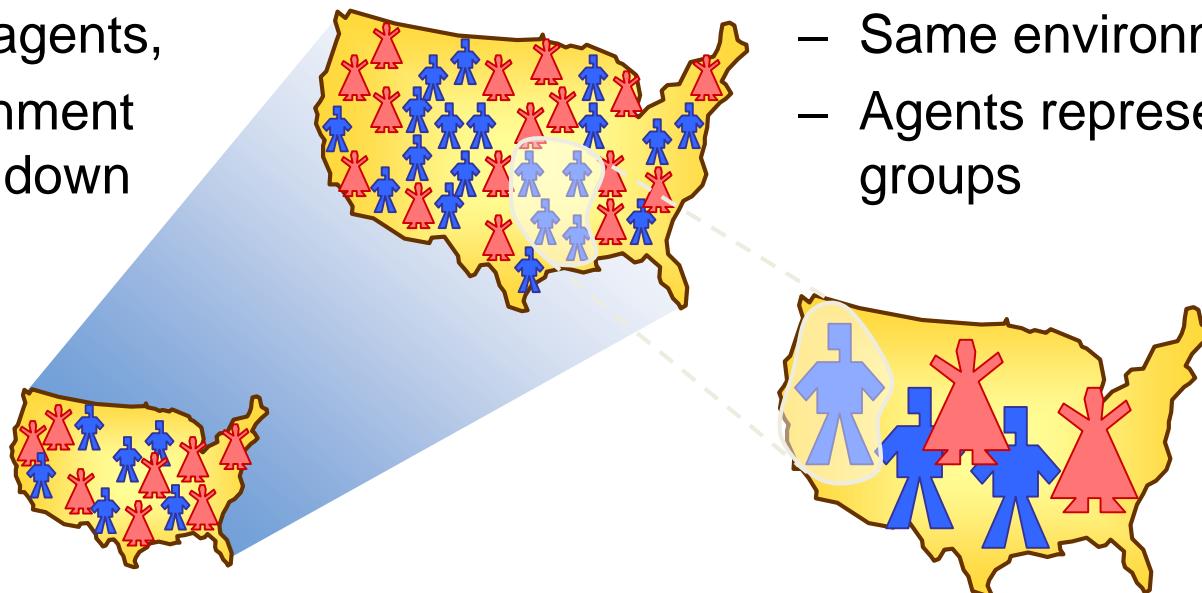
Distance based (layout dependent)

- You can
 - combine standard and custom networks
 - re-apply standard network during run, etc.



How many agents to simulate?

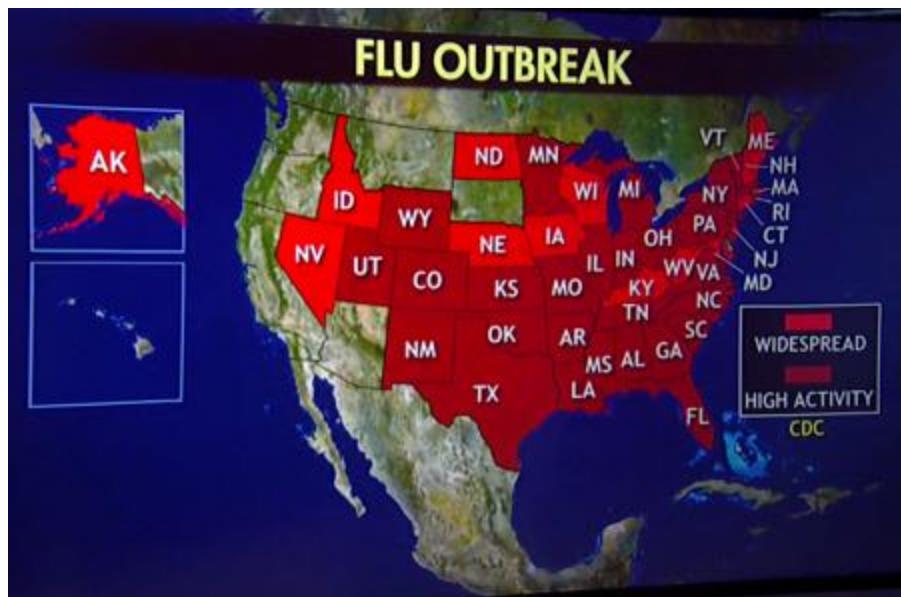
- If I need to model the US population do I need to simulate 300,000,000 agents? **Fortunately not!**
- Two main “model scaling” techniques are used:
 - Same agents,
 - Environment scaled down
 - Same environment
 - Agents represent groups





SIR Model. Calibration

This presentation is a part of
AnyLogic Standard Training Program





SIR Model. Calibration

- We will calibrate an agent based model of contagious disease diffusion.
- In the model each person has 3 possible states: **Susceptible**, **Infectious** or **Recovered** (SIR). Initially all but a few people are susceptible, and a few are infectious. Upon contact with an infectious person a susceptible person will get the disease based on a certain probability.
- The agents are placed in a continuous space. The contacts occur between random agents.



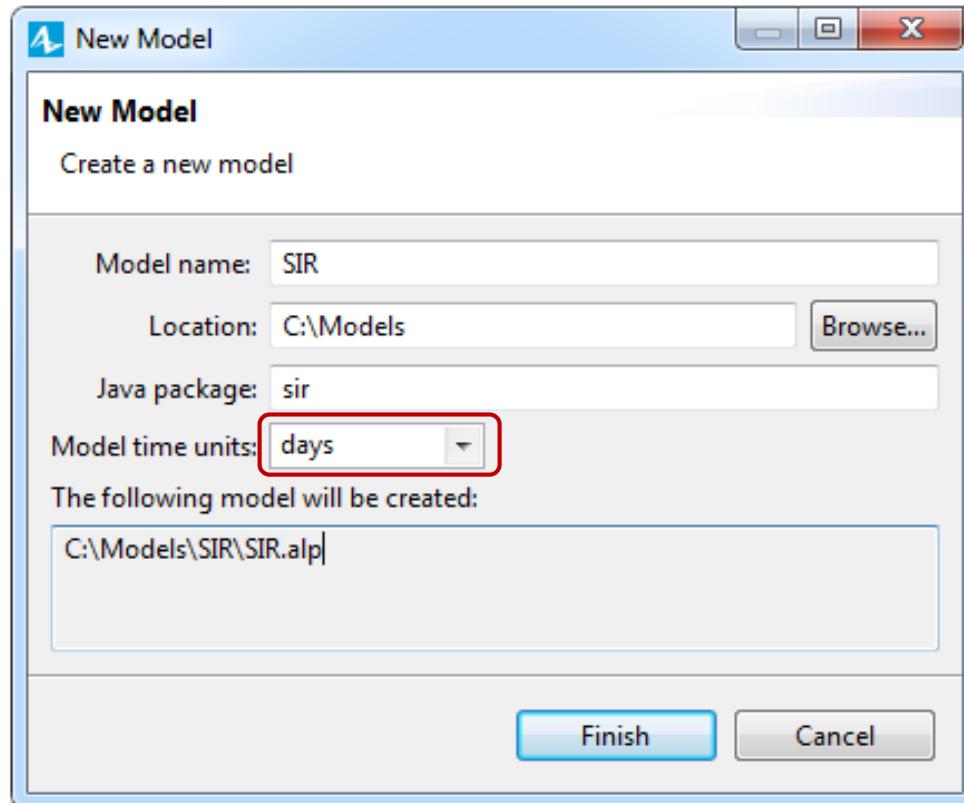
SIR Model. Calibration

- The objective is to find the right parameters for the agents (contact frequencies and infection probabilities) so that the simulation output matches historical data reflecting the dynamics of infectious disease spreading through a population. Since the model is stochastic, calibration is done under uncertainty, and simulation replications are used.



SIR Model. Phase 1

- In this phase we will build a simple SIR model.
- Create new model: *SIR*. Choose *days* as the model time units.



SIR. Phase 1. Step 1

The screenshot shows the AnyLogic modeling environment. On the left, the 'Palette' tab is selected, displaying various components: Agent (highlighted with a red arrow), Parameter, Event, Dynamic Event, Variable, and Collection. In the center, the 'Main' workspace contains a single agent named 'MyAgent'. A red arrow points from the 'Agent' icon in the palette to the 'MyAgent' icon in the workspace.

① Create a new agent population: 10000 agents of type Person

New agent
Step 1. Choose what you want to create

Population of agents
Create a number of agents of the same type living in the same environment in the current agent.
Typical cases:

- People
- Consumers
- Patients
- Trucks
- Projects or products

A single agent
Create a single agent that will always exist within the current agent.
Typical cases:

- Supplier, distributor, producer
- Building
- Factory
- Store
- Gas station
- Hospital
- Equipment unit
- City or village

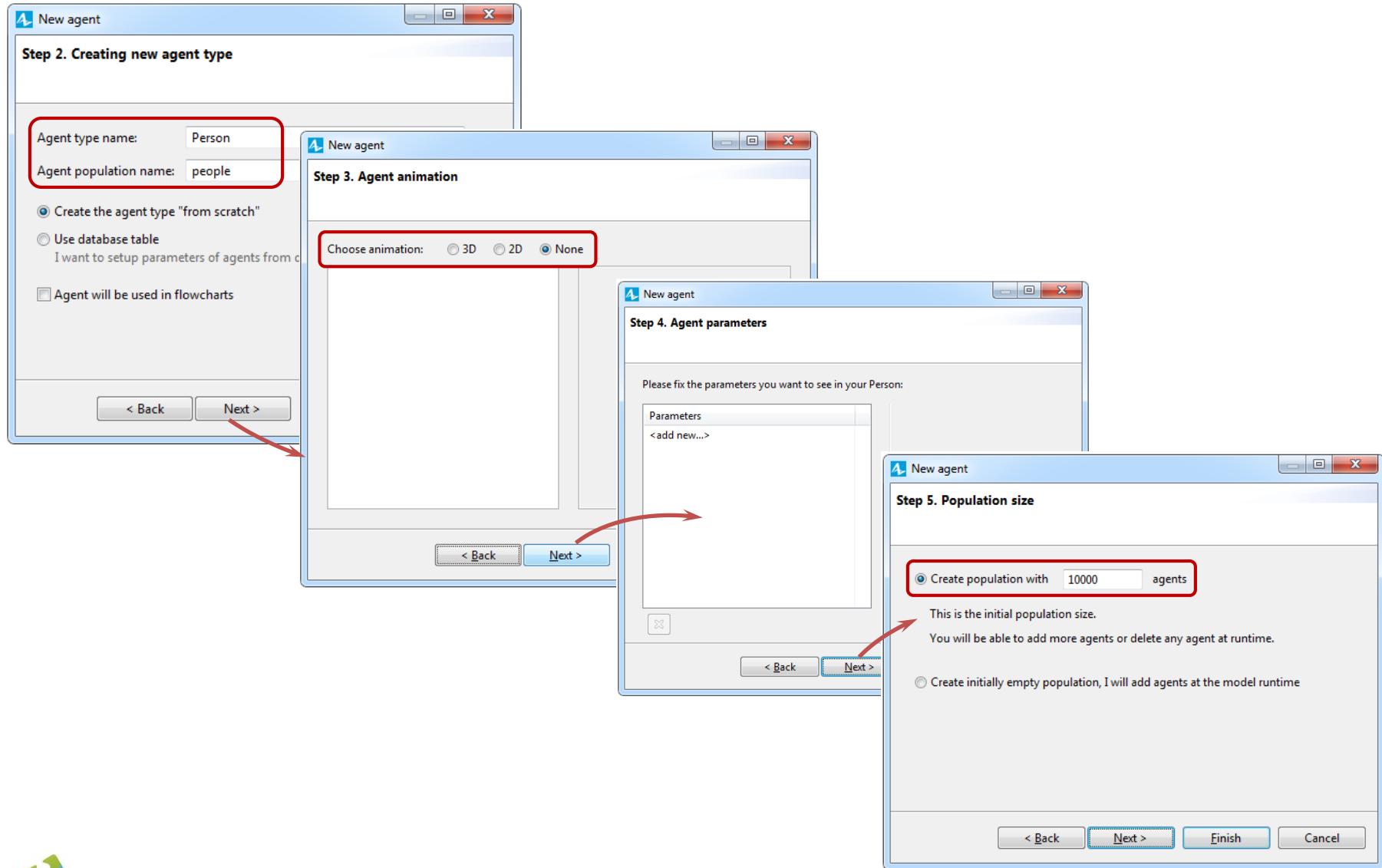
Agent type only
Create an agent type, do not create any agents at this point.
Typical cases:

- Agent type: Patient, Customer, Document, Part, Transaction
- Resource type: Doctor, Worker, ForkliftTruck
- Train or rail car type
- Pedestrian type
- Structural part of the model, such as a subprocess

< Back Next > Finish Cancel



SIR. Phase 1. Step 2



Using the New Agent Wizard, create a new *population of agents*.

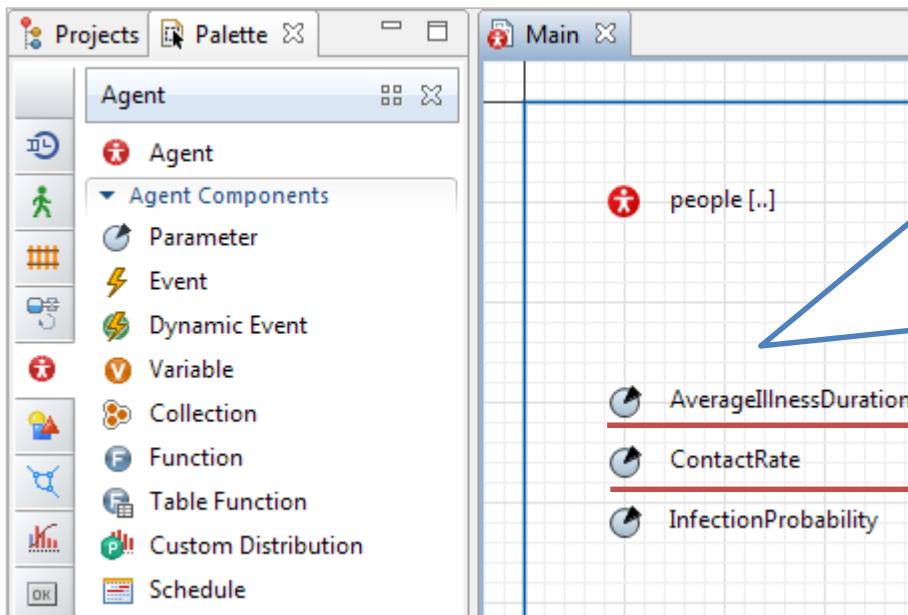
Set *Person* as the type name, *people* as the name of the population.

Do not use any animation shapes for the agents, we do not need them since we will concentrate solely on calibration.

Let 10000 agents live in the model. On the last page of the wizard do not change anything, let agents live in the *Continuous* space.



SIR. Phase 1. Step 3

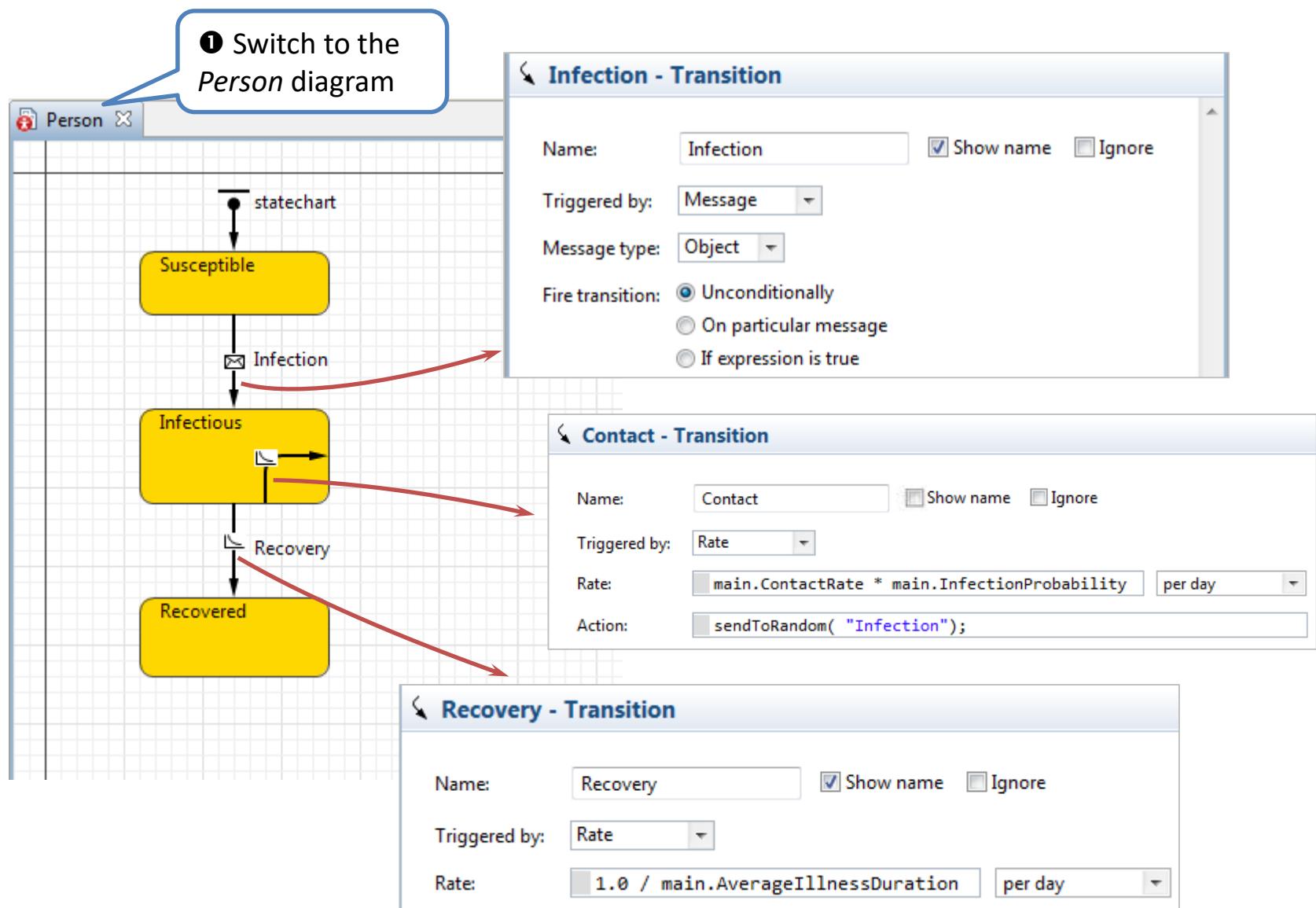


- ❶ On the *Main* diagram, create three parameters of the type *double* with the following default values:

- ① Open the diagram of *Main* and define here agent characteristics with three parameters of the *double* type.



SIR. Phase 1. Step 4



- ① In the *Person* agent type define agent behavior with a statechart. A person has 3 possible states: Susceptible, Infectious, or Recovered.

Draw the statechart with **Statechart** palette's elements.

Start drawing the statechart with **Statechart Entry Point** element. Then add three **States**, connect them with two **Transitions**, and finally add one internal transition inside the *Infectious* state.

Statechart

- The best way to define a behavior is to use a *statechart*. The states in the statechart are alternative: the object can only be in one state at a time.
- In general, single agent may have multiple statecharts, but now we need only one.



SIR. Phase 1. Step 5

Main Person

people [...]

AverageIllnessDuration
ContactRate
InfectionProbability

Properties

people - Person

Initial number of agents: 10000

Initial location

Statistics

Name: nInfectious

Type: Count Sum Average Min Max

Condition: item.inState(Person.Infectious)

① Define statistics function counting number of infectious people



We want to view how the numbers of infectious people change over time. To achieve this we will define a function to calculate number of infectious people.

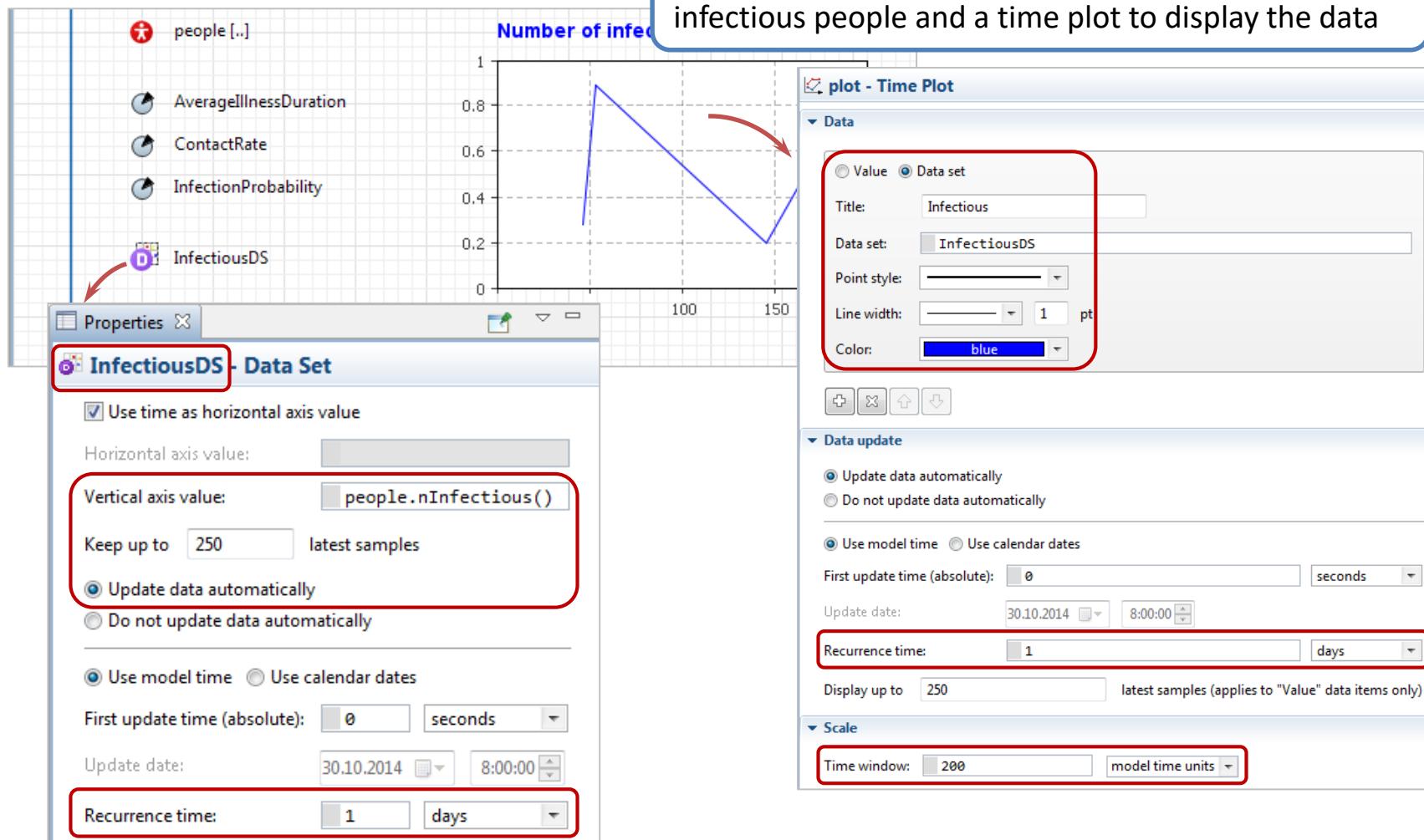
Collecting statistics on agents

- To add a new function collecting statistics over all the agents, go to the **Statistics** properties section of the agent population and click the “plus” button. We need to find out how many agents are in the *Infectious* state. The statistics of type *count* does exactly that: it iterates through the population and counts how many agents satisfy the given condition.
- Specify *item.inState(Person.Infectious)* as the function condition. Here *item* represents the current agent, *inState()* is a standard method of statechart, and *Infectious* is the name of the state defined within the agent, that is why it needs the name prefix *Person*.



SIR. Phase 1. Step 6

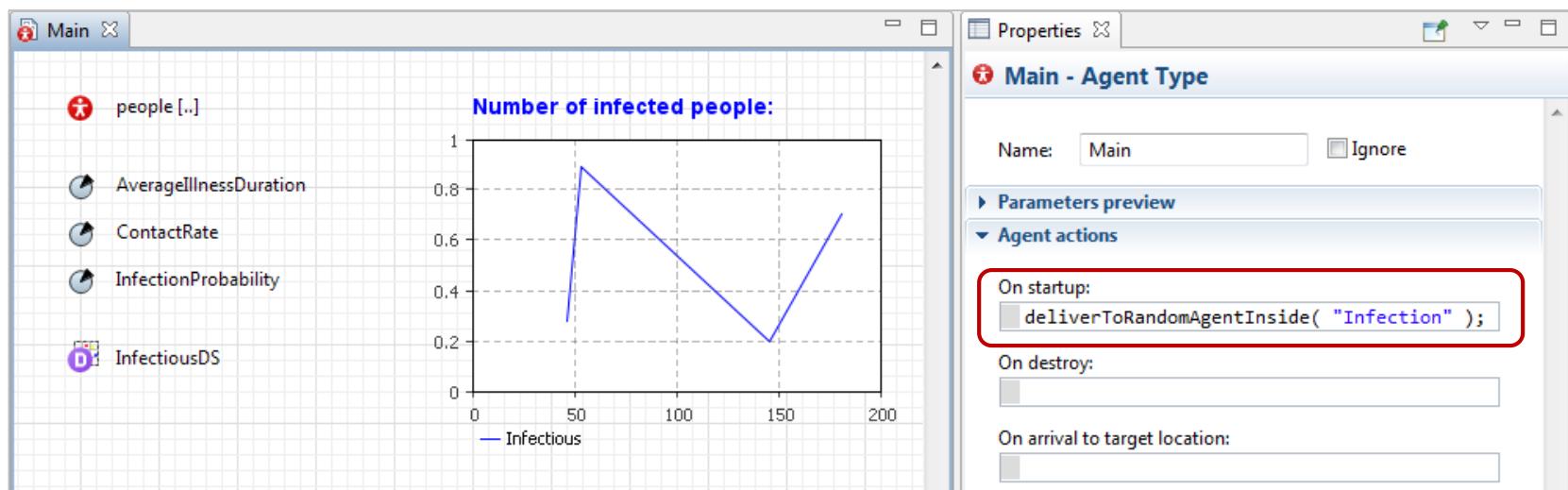
① Add a data set to store history of number of infectious people and a time plot to display the data



We configure the data set to record the number of infectious people. The number is obtained by calling the statistics function defined in the previous step. The function is defined in agent population *people*, so we call it as a function of that object: *people.nInfectious()*



SIR. Phase 1. Step 7



Simulate the index patient by making one randomly chosen agent infected on model startup.



SIR. Phase 1. Step 8

The screenshot shows the AnyLogic Professional interface for a SIR simulation. On the left, the main workspace displays a population of 10,000 people, with a chart titled "InfectiousDS" showing the number of infected people over time. The chart has a sharp peak at approximately 25 units of time, reaching about 7,000 infected individuals. A legend indicates the blue line represents the "Infectious" state. On the right, the "Properties" window is open under the "Simulation - Simulation Experiment" tab. The "Model time" section is selected, showing the execution mode set to "Real time with scale 1". The "Stop" section is set to "Stop at specified time" with a start time of 0 and a stop time of 200. Two callout boxes provide instructions: one pointing to the stop time setting, and another pointing to the chart.

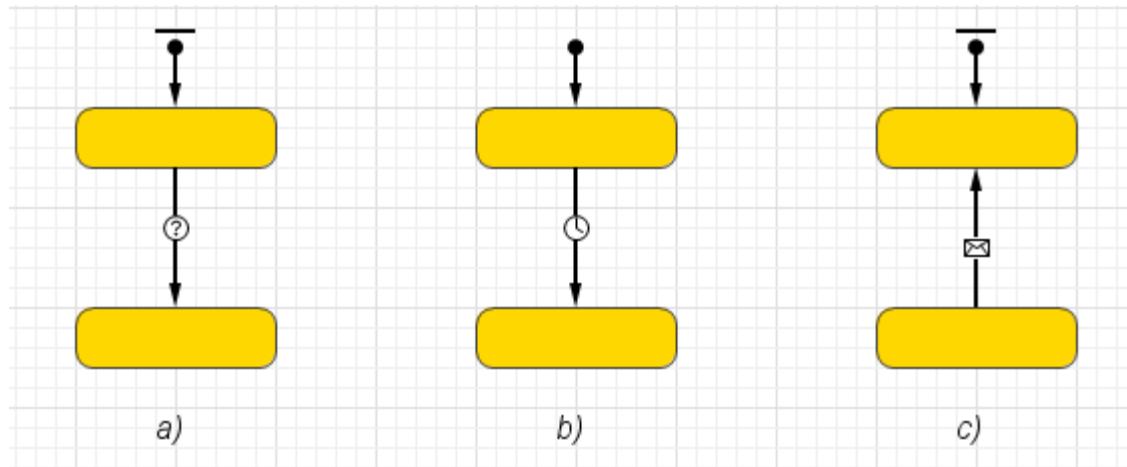
① Set the experiment to stop at time moment 200

② Run the model in virtual time mode. You will see how the number of infectious people changes.



SIR. Phase 1. Questions

1. Which statechart is correct ?



2. Please explain how the parameters defined on the *Main* diagram are accessed by agents.
3. How do we infect the first agent in the model?

SIR. Phase 2. Calibration

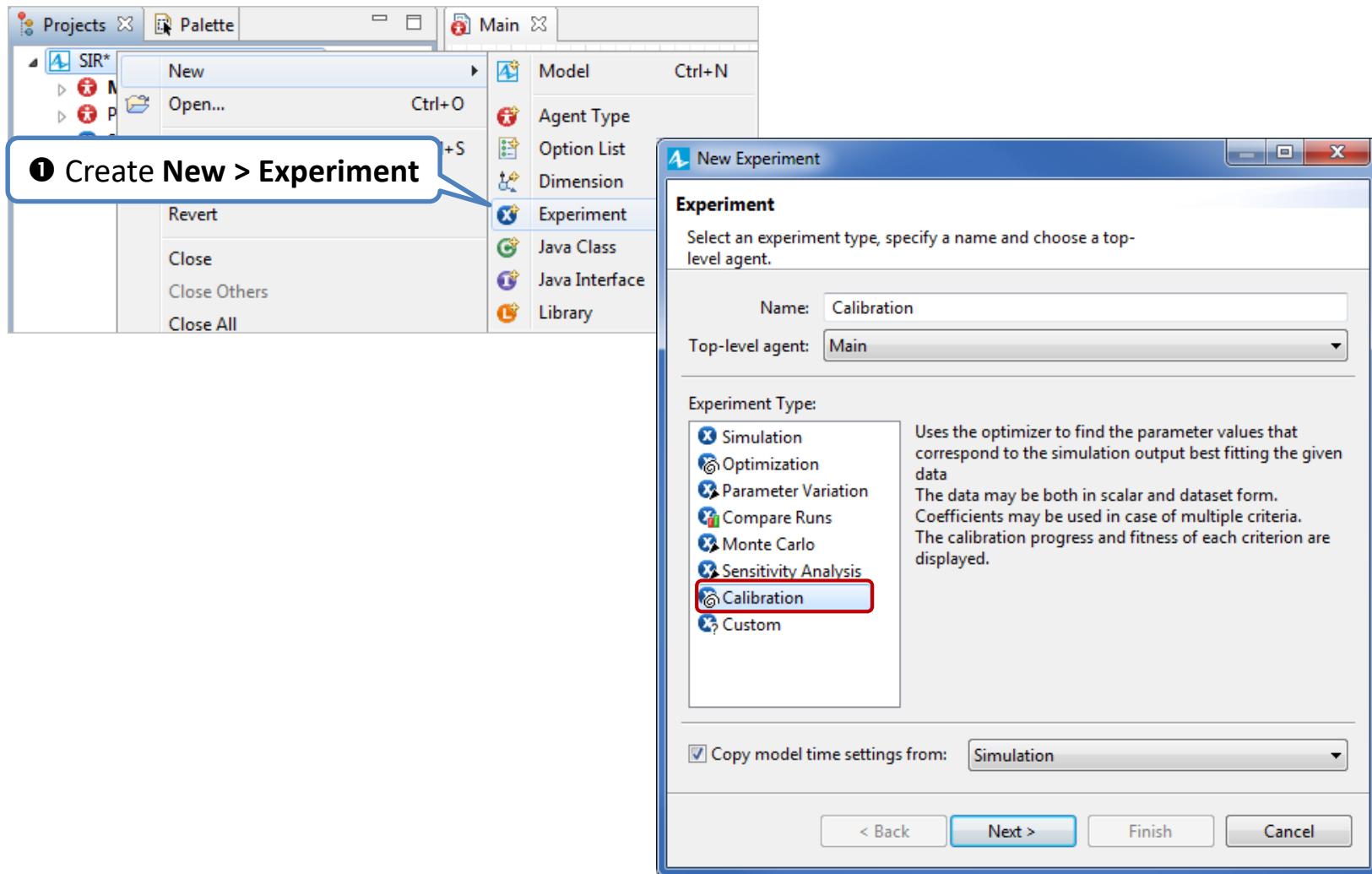
- Now we will tune the parameters of the model so that its behavior matches a known (observed) pattern under specified conditions
- To do this, we will use the AnyLogic calibration experiment

AnyLogic calibration experiment

- When you have your model structure in place, you may wish to tune some parameters of the model so that its behavior under particular conditions matches a known (historical) pattern. In cases where there are several parameters to tune it makes sense to use the built-in optimizer to search for the best combination. The objective in this case is to minimize the difference between the observed simulation output and historic data.
- The calibration experiment uses the optimizer to find the model parameter values that correspond to the simulation output which best fits the given data. The data may be both in scalar and dataset form. Coefficients may be used in the case of multiple criteria. The calibration progress and fitting of each criterion are displayed.



SIR. Phase 2. Step 1



SIR. Phase 2. Step 2

New Experiment - Calibration

Parameters and Criteria
Choose parameters the optimizer will be allowed to vary and the calibration criteria.

Parameters:

Parameter	Type	Value	Min	Max	Step
AverageIllnessDuration	fixed				
ContactRate	continuous		0.5	5	
InfectionProbability	continuous		0.1	0.8	

Criteria:

Title	Match	Simulation output	Observed data	Coefficient
Historic data, best fitting & simulation output	data series	root.InfectiousDS	InfectiousHistory	1.0

Two types of criteria are available: 'scalars' for fitting single values and 'data series' to fit datasets. "root" to refer to the top level active object, e.g. "root.myDataset". In the 'Observed data' field enter an expression of scalar type, existing dataset ("root" word may be also used) or the name of a (currently not existing) dataset or table function which should be created after the wizard finishes. Use coefficients to combine and balance multiple criteria.

① Configure parameters being calibrated

② Define the calibration criteria

< Back Next > **Finish** Cancel



Configure the calibration experiment.

- ❶ Set up the parameters whose values will be varied when performing the calibration. To include a parameter in the calibration process, choose some value other than *fixed* in the parameter's **Type** cell in the **Parameters** table of the wizard.
- ❷ We will search for minimal differences between the simulation output (our *InfectiousDS* data set defined in the top-level agent of the experiment and thus accessible here as *root.InfectiousDS*) and the historic data (just specify the name of the object containing the historic data series, we will actually create it later in the next step).



SIR. Phase 2. Step 3

The screenshot shows the AnyLogic software interface with the following components:

- Projects** tab is selected.
- Palette** tab is open, showing various modeling elements like Agent, Parameter, Event, etc.
- Main** and **Calibration** tabs are open in the center workspace.
- SIR : Calibration** window contains:
 - A **Run** button.
 - A table with columns **Current** and **Best**.
 - Iteration: **infeasible** (red)
 - Objective: **?** (blue)
 - Parameters** section with three variables:
 - AveragellnessDuration: **?**
 - ContactRate: **?**
 - InfectionProbability: **?**
 - A yellow rounded rectangle containing a **copy** button.
 - A red arrow points from the **InfectiousHistory** icon in the palette to this yellow box.
 - A red arrow points from the **InfectiousHistory** icon in the calibration window to the **InfectiousHistory** entry in the table function editor.
 - A red arrow points from the **InfectiousHistory** icon in the calibration window to the **InfectiousHistory** icon in the palette.
- InfectiousHistory - Table Function** editor on the right:
 - Name:** InfectiousHistory
 - Visible:** Yes
 - Interpolation:** Linear
 - Out of range:** Nearest (highlighted with a red box)
 - Table data:** A table with Argument and Value columns.

Argument	Value
2	11
4	73
6	171
8	341
10	511
12	681
14	851
16	1021
18	1191
20	1361
 - Preview:** A plot showing a red line graph with a sharp peak around x=40 and a long tail decaying towards zero.

① Add *Table Function*, name it *InfectiousHistory*, right as you specified it in the experiment creation wizard

② Paste historic data from *HistoricData.txt* file

③ You will see the plot of the function built upon the loaded table data



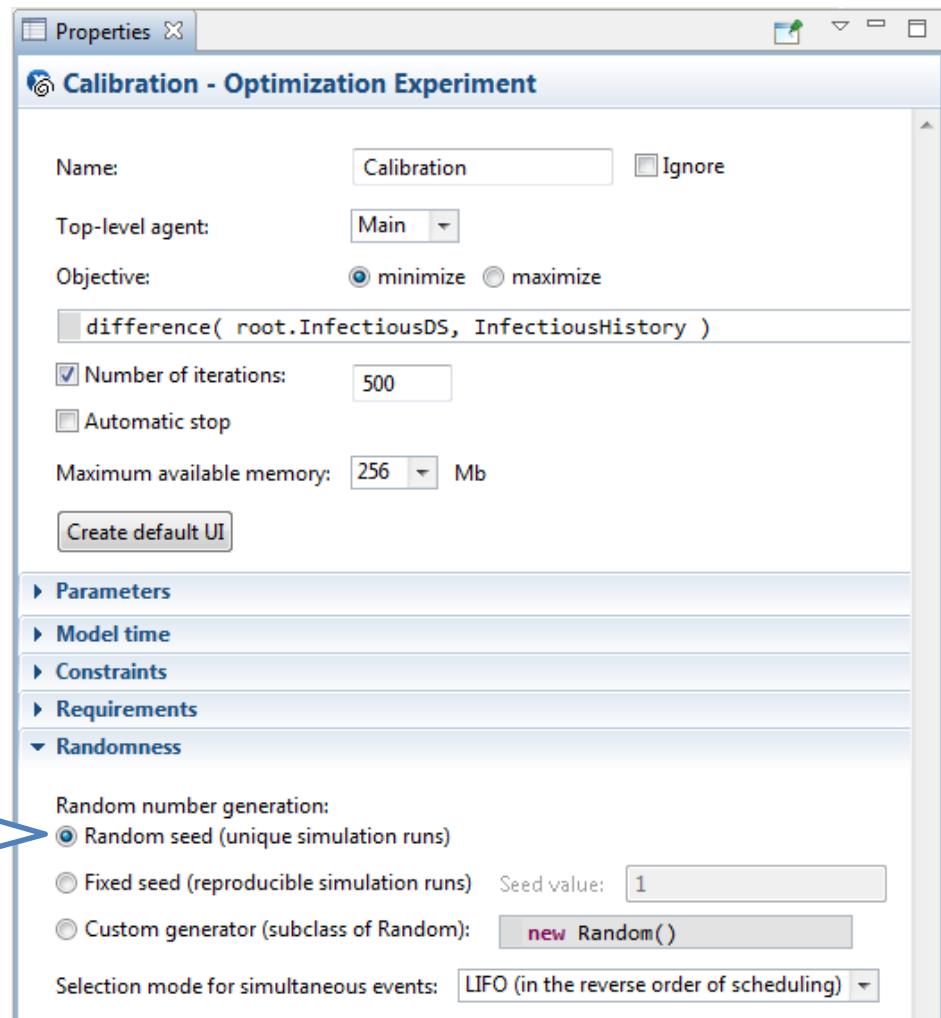
Load the historic data into the model by adding a table function on the diagram of the calibration experiment.

Table functions

- *Table function* (AKA *lookup table*) is a function defined in the table form. You can make it continuous by interpolating and/or extrapolating. Table functions are commonly used to load experimental data in the model as a table function and convert the function to a continuous form.



SIR. Phase 2. Step 4



① Set **Random seed** for the
Calibration experiment's
random number generator



SIR. Phase 2. Step 5

SIR : Calibration - AnyLogic Professional

Run

1 Run the *Calibration* experiment

Current	Best
Iteration: 261	57
Objective: ↓ 1,741.387	266.218

Parameters

AverageIllnessDuration	15	15
ContactRate	2.618	2.011
InfectionProbability	0.545	0.177

Copy the best solution to the clipboard ➤ COPY

2 You can see that multiple processors perform experiment iterations in parallel

Objectives

Criteria

Historic data, best fitting & simulation output

Comparison graphs

Run: 262 | Experiment: 54% | Simulations: | Memory: 132M / 247M | 57.0 sec



SIR. Phase 2. Step 6

SIR : Calibration - AnyLogic Professional

SIR : Calibration

Run

Current Best

Iteration:	494	111
Objective:	1,614.87	323.915

Parameters

AverageIllnessDuration	15	15
ContactRate	3.173	1.738
InfectionProbability	0.144	0.201

Copy the best solution to the clipboard

copy

Statistics on currently used and best found objective and parameter values

Criteria

Historic data, best fitting & simulation output

Run: 1,500 Finished Experiment: 100% Simulations: Stop time not set Memory: 127M of 247M 329.7 sec

- ① Copy the found optimal parameter values to the simulation experiment

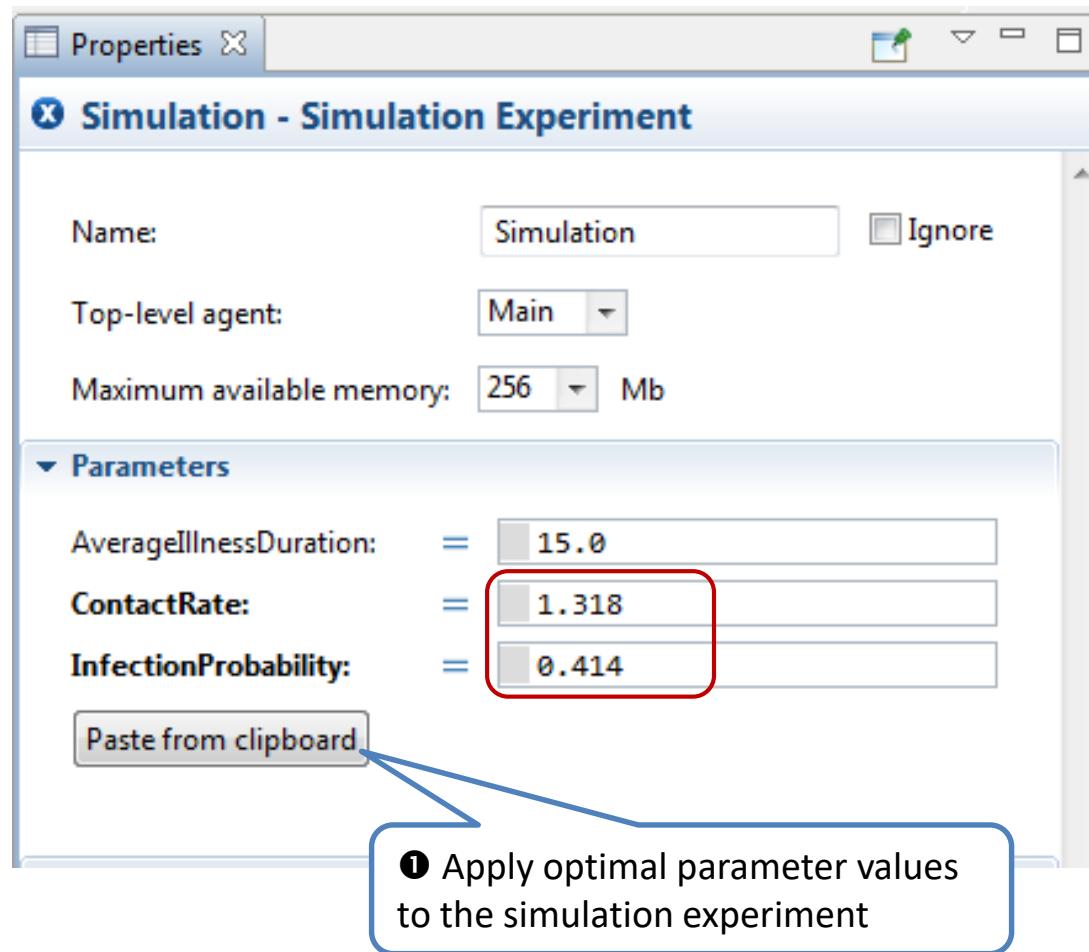


Taking advantage of processors with multiple cores

- If your computer has more than one processor core, AnyLogic will place different simulation runs on different cores in parallel. Performance is increased and experiments are performed significantly quicker than when using a processor with a single core. This applies to all experiments where multiple simulation runs can be done independently or partially independently: Parameter variation, Monte Carlo, Sensitivity analysis, Optimization, and Calibration.
- To find out how many processor cores there are on your computer and to set the number of cores to be used by AnyLogic, open **AnyLogic Preferences** (menu item **Tools > Preferences**). The simulation progress indicator in the status bar of the running model shows as many progress bars as there are simulation runs going in parallel.
- You should be aware that in the case of parallel execution simulations get unordered and the code you may have written in the fields **Before simulation run**, **After simulation run**, and **After Iteration** may also be executed in an arbitrary order. The other thing to notice is that optimization experiments become un-reproducible. You can prevent parallel execution for a particular experiment by un-checking the checkbox **Allow parallel evaluations** at the very bottom of the **Advanced** properties of the experiment.



SIR. Phase 2. Step 7



Applying the result of the calibration in the model

- Your simulation experiment will typically be run with the parameter values that resulted in the best fit to the known historical data.



Java Basics for AnyLogic

This presentation is a part of
AnyLogic Standard Training Program



General remarks

- You do not have to learn full OO programming
- You need to understand Java data types, expression and statement syntax
- Please note:
 - Java is case-sensitive: MyVar is different from myVar!
 - Spaces are not allowed in names: “My Var” is illegal name!
 - Each statement has to be finished with “;”: MyVar = 150;
 - Each function has to have parentheses: time(), add(a)
 - Dot “.” brings you “inside” the object: agent.event.restart()
 - Array elements have indexes from 0 to N-1



Data types

- Primitive Types

double – represent real numbers: *1.43, 3.6E18, -14.0*

int – represents integer numbers: *12, 16384, -5000*

boolean – represents Boolean (*true/false*) values. Boolean values are only true and false, you cannot use 1 and 0

- Compound Types – Classes

String – represents textual strings, e.g. “MSFT”, “Hi there！”, etc

ArrayList, LinkedList – collections of objects

Shape3DPolyLine – represents AnyLogic polyline shape
... many others. See AnyLogic and Java Class References



Expressions

Arithmetic operations:

- + *addition and String concatenation*
- *subtraction*
- * *multiplication*
- / *division*
- % *remainder from integer division*

Mind integer division!!! $3/2 = 1$, not 1.5

In integer divisions, the fraction part is lost,
e.g. **3 / 2** equals 1, and **2 / 3** equals 0

Multiplication operators have priority over
addition operators

The '+' operator allows operands of type String

Comparison operations:

- < *less than*
- <= *less than or equal*
- > *greater than*
- >= *greater than or equal*
- == *equal*
- != *not equal*

Left-to-right precedence for operators; parentheses can be used to alter precedence

Boolean operations:

- && *AND*
- || *OR*
- ! *NOT*

Conditional operator:

condition ? value-if-true : value-if-false

Assignments and shortcuts:

- = *assignment*
- += *a+=b shortcut for a=a+b*
- = *a-=b shortcut for a=a-b*
- *= *a*=b shortcut for a=a*b*
- /= *a/=b shortcut for a=a/b*
- ++ *a++ shortcut for a=a+1*
- *a-- shortcut for a=a-1*



Some examples

`5 % 2 ≡ ?`

`5 / 2 ≡ ?`

`5.0 / 2 ≡ 5 / 2.0 ≡ ?`

`(double)5 / 2 ≡ ?`

`a += b; ≡ ?`

`a++; ≡ ?`

`“Any” + “Logic” ≡ ?`

Let $x = 14.3$, then:

`“x =” + x ≡ ?`

`“” ≡ ?`

`“” + x ≡ ?`

`y = x > 0 ? x : 0`

`≡ ?`

`x == 5 ≡`

`x = 5 ≡ ?`



Calling functions and accessing fields

- Function call

To call a function, type its name followed by parentheses. If necessary, put parameters separated by commas within the parentheses. Examples:

```
x = time();  
moveTo( getX(), getY() + 100 );  
traceIn( "Population is increasing" );
```

- Accessing object fields and functions

To access a field or function of a model element (statechart, event, animation shape), use the element name followed by dot '.' followed by the field/function name. Examples:

```
rectangle.setFillColor(red);  
company.event.restart(10);
```



Agent populations



people [..]

- Agent population is a collection.
Items are indexed from 0 to N-1
Getting the current size of the population:
people.size();
Obtaining i-th agent of the population:
people.get(i);
Obtaining a random agent:
people.random();
Adding a new agent to the population:
add_people();
Removing an agent from the population:
remove_people(person);



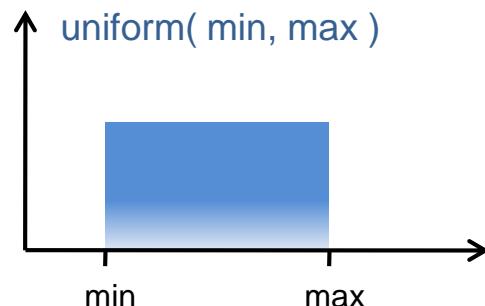
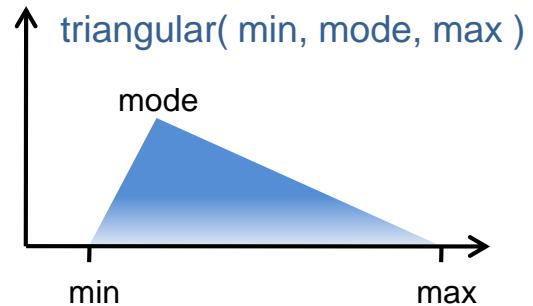
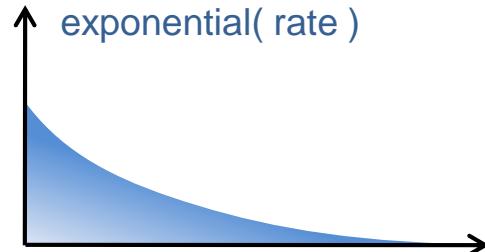
Built-in functions

- System functions
 - `time()`, `date()`, `pauseSimulation()`, `inState(...)`, etc
- Mathematical functions
 - `sqrt()`, `sin()`, `cos()`, `exp()`, `log()`, `round()`, etc
- Random distributions
 - `uniform()`, `exponential()`, `bernoulli()`, `beta()`, etc
- Print operators
 - `traceIn()` - prints text to Console
 - `traceToDB()` - prints text to the model's database
- And more...
 - See [Utilities](#), [Engine](#), [Agent](#) classes in AnyLogic Class Reference



Probability distributions

- Exponential
 - It has been proven that if events occur at a constant rate but are independent, the time between two subsequent events is distributed exponentially
 - Example: phone calls to a call center, client arrivals, and so on.
- Triangular
 - Is used in the conditions of limited knowledge about the stochastic variable, namely we know the minimum, the maximum, and the modal value
 - Example: service time
- Uniform
 - We only know the minimum and the maximum values, and do not know if any value in between is more frequent than another



Decision operator (if ... else)

- Checks the condition and executes one section of code (after **if**) if a condition evaluates to **true**, and optionally another code (after **else**) if it evaluates to **false**
- Full form (**if** ... **else**):
if (*condition*)
 statement if true
else
 statement if false
- Short form (**if**):

```
if  ( inventory < s )
{
    source.inject(10);
    inventory += 10;
}
```

In case "if" or "else" code sections contain more than one statement, they should be enclosed in braces { ... } to become a block, which is treated as a single statement

How would the code look like if we need to check TWO conditions?



for loop

- The loop mostly used to iterate through agent populations and collections
- Population/collection iterator:

```
for ( agent_type name : population )  
{  
    statements executed for each agent/element  
}
```

```
for ( Person p : people )  
    total += p.income;
```

- Index-based loop:

```
for ( initialization; continue condition; increment )  
{  
    statements executed for each agent/element  
}
```

```
for ( int i=0; i<distributor.numberOfTrucks(); i++ )  
{  
    Truck t = add_trucks();  
    t.setPosition(distributor.getPosition());  
}
```



while loops

- "While" loops are used to repeatedly execute some code while a certain condition evaluates to **true**

- While loop

```
while (continue condition)  
{  
    statements  
}
```

- Do While loop

```
do  
{  
    statements  
}  
while (continue condition);
```

```
int i= 0;  
while (excelFile.cellExists(i, 1))  
{  
    Supplier s = add_suppliers();  
    s.set_name(excelFile.  
        getCellStringValue(i,1));  
    i++;  
}
```

```
double x;  
double y;  
do  
{  
    x = uniform(0, 1000);  
    y = uniform(0, 1000);  
}  
while (!citybounds.contains(x,y));
```



return

- Exits from the current function.
- The return statement has two forms: one that returns a value (e.g. result of calculations), and one that doesn't. To return a value, simply put the value after the **return** keyword.



FUNCTION

Return type: Truck

```
for (Truck t : fleet)
{
    if (t.inState(Truck.AtDistributor))
        return t;
}
return null;
```



Writing comments in Java code

There are two kinds of comments:

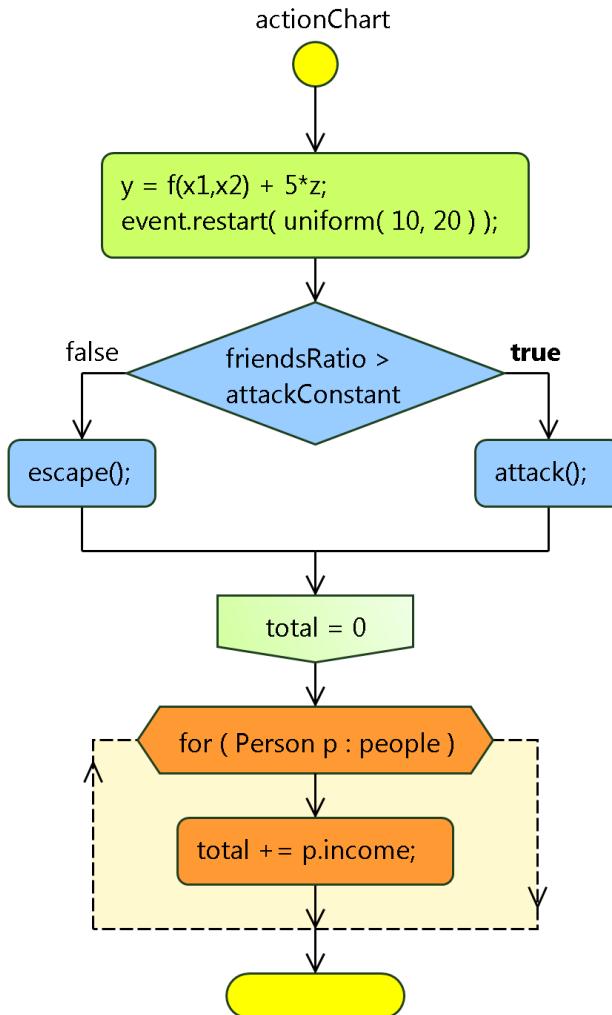
```
/* text */          A traditional comment: all the text from the ASCII  
                    characters /* to the ASCII characters */ is ignored  
                    (as in C and C++)  
  
/*  
   The class represents AnyLogic 3D animation. It contains the canvas object.  
   author Daniil Chunosov  
   version 5.0  
*/  
  
public class Animation3DPanel extends javax.swing.JPanel;
```

// text An end-of-line comment: all the text from the ASCII
 characters // to the end of the line is ignored (as in C++).

```
// Prepare Engine for simulation:  
Engine.start(root);  
Engine.runFast(); // fast mode - no animation
```



Action chart



- Assignment / action statement:

```
y = f(x1,x2) + 5*z;  
event.restart( uniform( 10, 20 ) );
```

- Decision statement:

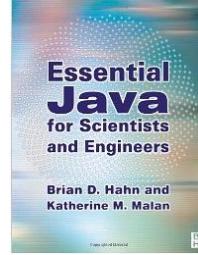
```
if ( friendsRatio > attackConstant )  
    attack();  
else {  
    escape();  
}
```

- Loop statement:

```
double total = 0;  
for ( Person p : people )  
    total += p.income;  
for( int i=0; i<100; i++ )  
    sendToRandom( msg );
```



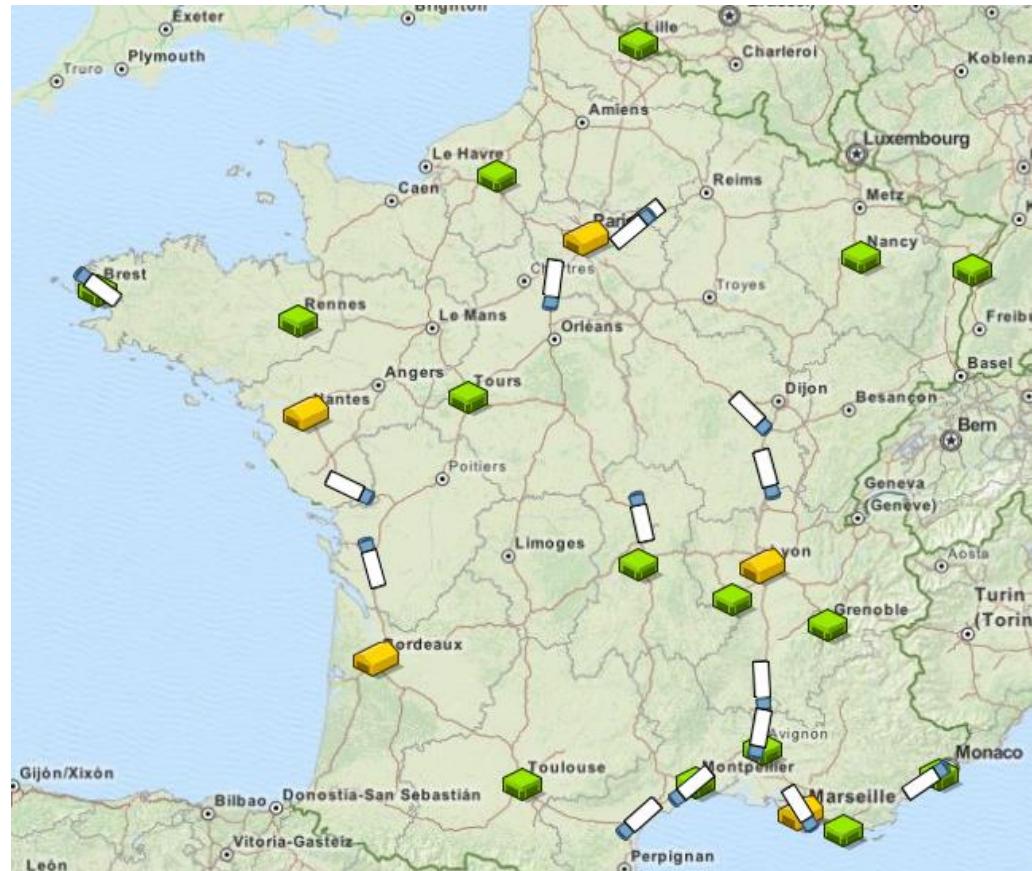
Reference information sources

- All the Java essentials for AnyLogic
The complete chapter on Java from *The Big Book of Simulation Modeling* by A. Borshchev included in AnyLogic documentation, see [AnyLogic Help > Java Basics for AnyLogic](#)
- Oracle tutorials:
<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>
- Online training:
<http://www.javapassion.com/portal/>
- Non-programmer book on Java:
Essential Java for Scientists and Engineers.
By B. Hahn & K. Malan
Paperback: 352 pages ISBN-10: 0750659912
- Java API: <http://java.sun.com/javase/8/docs/api>



Supply Chain Model

This presentation is a part of
AnyLogic Standard Training Program





Supply Chain Model

- Build a simulation model of a simple supply chain.
- Distributors and retailers will be located in various places.
- From time to time retailers order a certain amount of product from distributors. Products are delivered by trucks. After delivery, truck returns to nearest distributor.



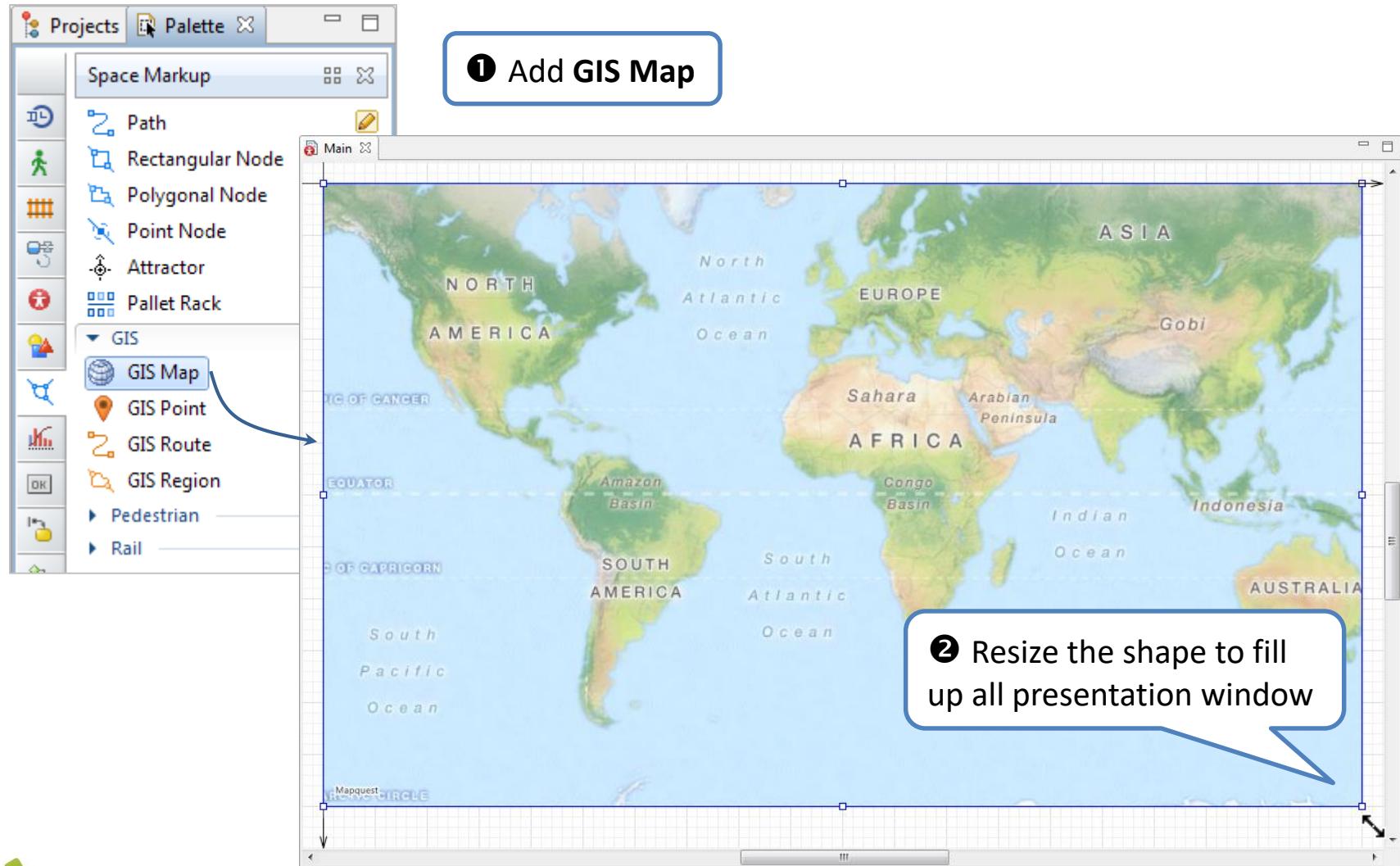
Phase 1. Creating distributors and retailers

- In the first phase we will populate our model with distributors and retailers and place them on a GIS map.

-
- We will demonstrate how to:
 - Create agent populations by reading data from an MS Excel file
 - Place agents into GIS space



Supply Chain. Phase 1. Step 1



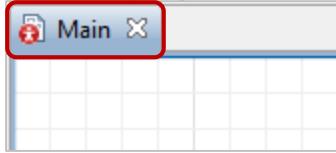
Create a new model. Name it *Supply Chain*.

Choose **hours** as **Model time units**.

Our model will be agent-based, with several types of agents: distributors, retailers, trucks. Our agents will live in a physical space defined by a GIS map. So, first of all, let us add a GIS map on the presentation.

- ① Drag the **GIS Map** object from the **GIS** section of the **Space Markup** palette.
- ② Resize this shape. We want it to fill all of the presentation window at the model run-time.

Hint: Please pay attention to the name of the agent type being edited at the moment:

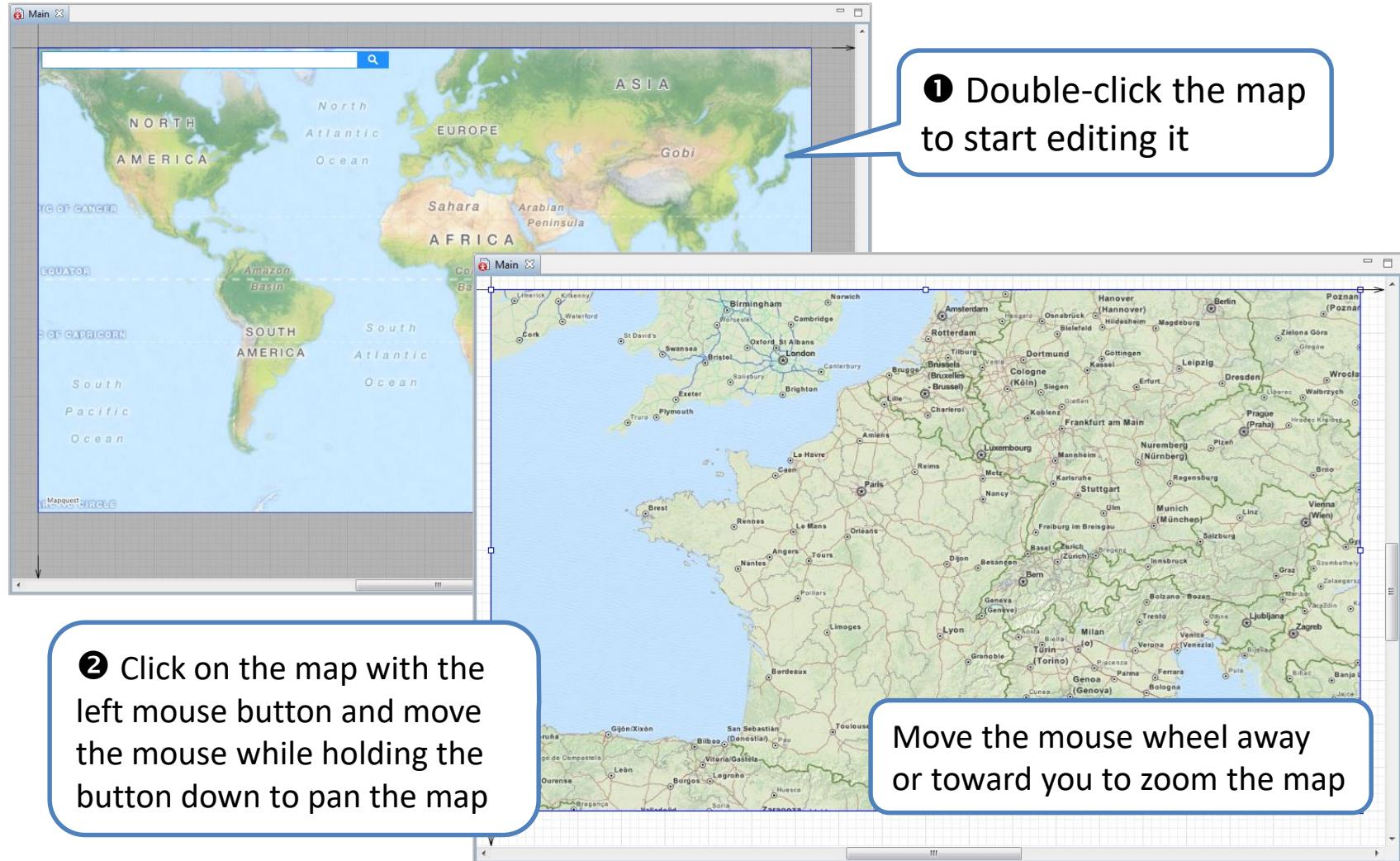


GIS

- You can add GIS (Geographic Information System) maps in your models. With GIS at your disposal you can develop geographically-aware models.
- GIS is commonly used in agent-based models. You can place agents on a GIS map, move agents with some specified speed from one location to another, execute actions upon arrival, animate agent at specific locations, establish connections based on agents layout, etc.
- AnyLogic GIS implementation is based on the OpenMap GIS visualization toolkit.



Supply Chain. Phase 1. Step 2



① Double-click the map to enter its edit mode, or right-click and select the **Edit map** option from the context menu, so that you can find the area at the appropriate scale to use in the model.

The graphical editor outside the map becomes grayed out when you are editing the map.

② We have chosen France as the place to develop this model. To pan the map, hold down the left mouse button and move it. To zoom in the area, move the mouse wheel away from you. The map will zoom around the area where your mouse cursor points.

With these actions, edit the map to display France on it.

Double-click the map again or click in the graphical editor to exit the edit mode.

GIS Map

- GIS map shape automatically loads tiles from the selected server.
- A tiled map is a map that consists of seamlessly connected square images, the number of which changes as you zoom in or out.



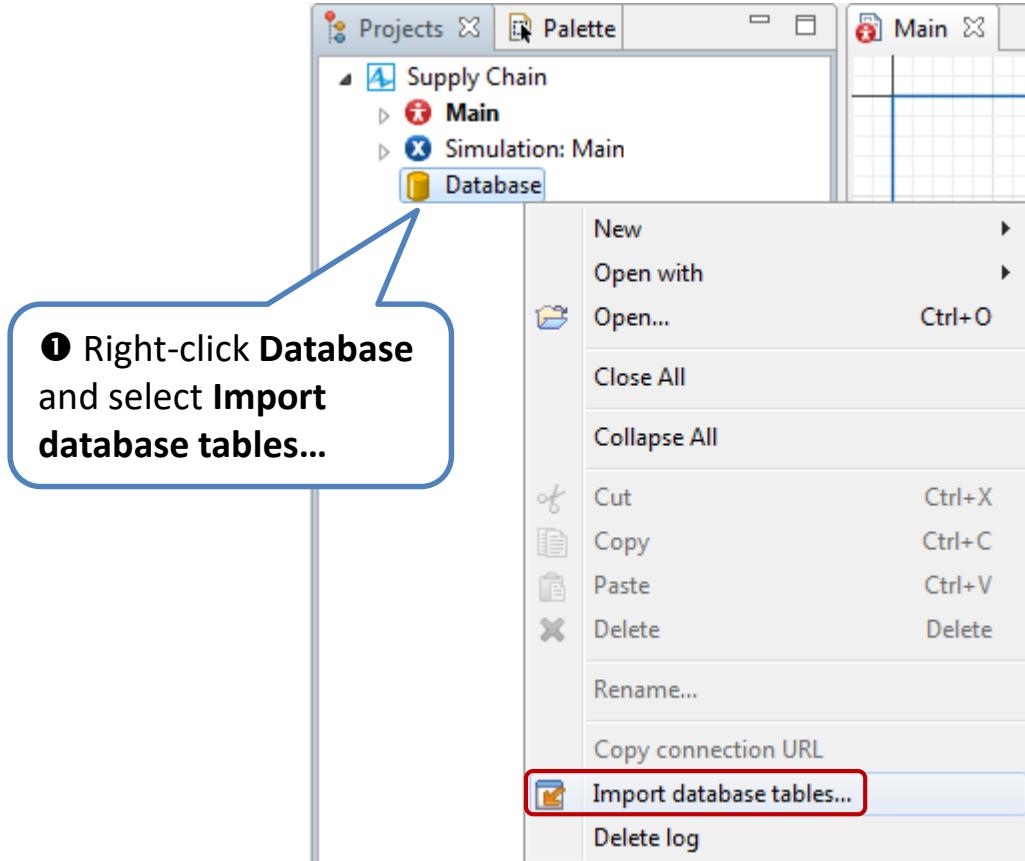
Supply Chain. Phase 1. Step 3

- We have input data stored in MS Excel spreadsheet file.
- The file contains two sheets with data on distributors and retailers.
- Let us import the data in the AnyLogic built-in database before we start using them in the model.

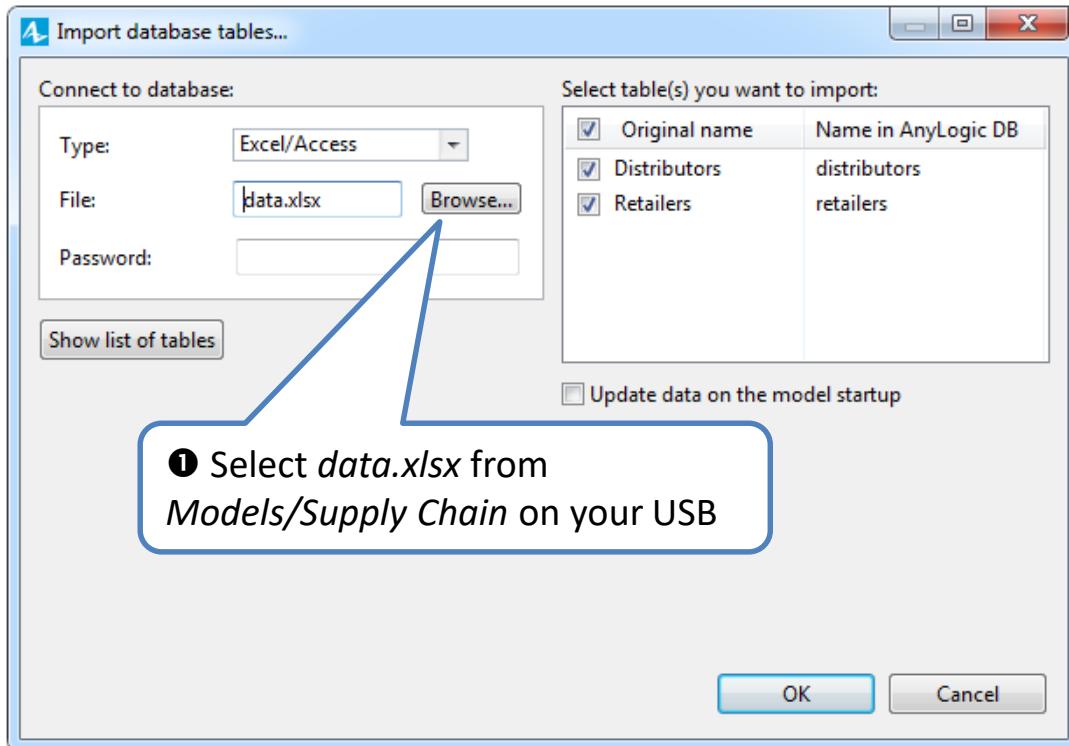
A	B	C
1 Location	N trucks	
2 Marseille	4	
3 Paris	5	
4 Bordeaux	2	
5 Lyon	3	
6 Nantes	1	
7		
8		
	Distributors	Retailers
1 Location		
2 Toulouse		
3 Lille		
4 Nice		
5 Brest		
6 Strasbourg		
7 Grenoble		
8 Rennes		
9 Rouen		
10 Toulon		
11 Montpellier		
12 Avignon		
13 Saint Etienne		
14 Tours		
15 Clermont-Ferrand		
16 Nancy		
17		
	Distributors	Retailers



Supply Chain. Phase 1. Step 4



Supply Chain. Phase 1. Step 5



② In Database, you'll see two tables created: *distributors* and *retailers*. Double-click the table to see its contents.

The screenshot shows the AnyLogic interface. On the left, the 'Projects' palette lists a project named 'Supply Chain*' containing 'Main', 'Simulation: Main', and a 'Database' folder which contains 'distributors' and 'retailers'. A red arrow points from the 'distributors' entry in the palette to the table view on the right. The 'Main' tab of the database view shows a table with columns 'location' and 'n_trucks'. The data rows are:

	location	n_trucks
1	Marseille	4
2	Paris	5
3	Bordeaux	2
4	Lyon	3
5	Nantes	1
*		



We import data from external database (MS Excel file) into the built-in AnyLogic database.

AnyLogic database

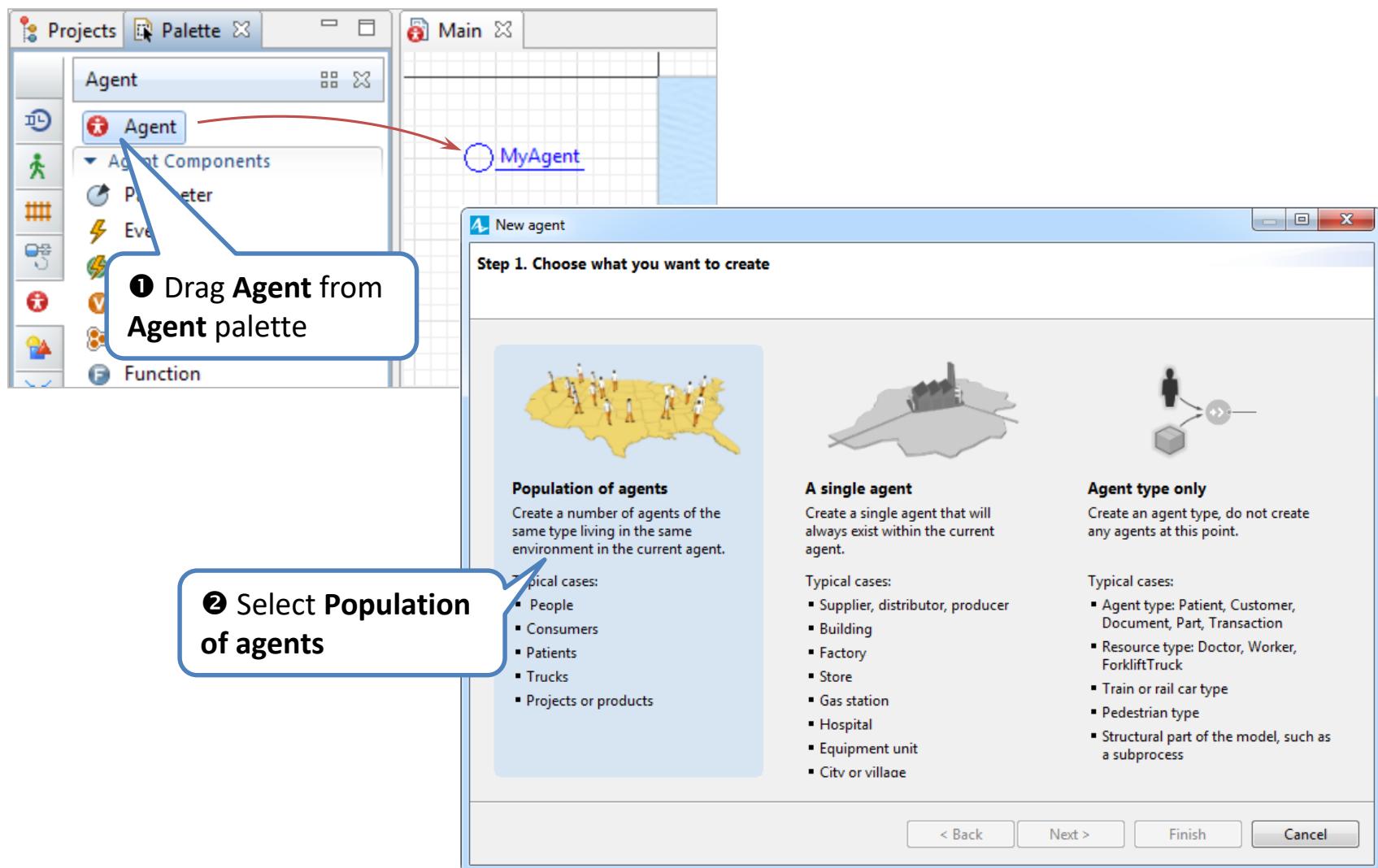
Each AnyLogic model now has a built-in fully integrated database to read input data and write simulation output. Export the database with the model, it is as portable and cross-platform as the model itself. With the new database, you can:

- Read parameter values and configure models
- Create parameterized agent populations
- Generate entity arrivals in the process models
- Import data from other databases or Excel spreadsheets and store it in the readily available form
- Log flowchart activities, events, statechart transitions, message passing, and agent biographies
- View resource utilization, waiting, processing and travel times
- Store and export statistics, datasets, and custom logs
- Export data to MS Excel spreadsheets.

You can easily import the database in your AnyLogic project from database, or spreadsheet. Alternatively, you can create empty database tables and enter data manually.



Supply Chain. Phase 1. Step 6



Supply Chain. Phase 1. Step 7

The image shows two windows from the AnyLogic software's 'New agent' wizard:

Step 2. Creating new agent type

- Agent type name:** Distributor (highlighted with a blue box)
- Agent population name:** distributors
- Options:**
 - Create the agent type "from scratch"
 - Use database table
 - I want to setup parameters of agents from database
 - Agent will be used in flowcharts

Step 3. Choose the database table that contains data for this agent type

- Use existing table:** distributors (selected, highlighted with a blue box)
- Import table from external data source**
- Connect to database:**
 - Type: Excel/Access
 - File: [Browse...]
 - Password: [Text input]
- Select table(s) you want to import:**

Original name	Name in AnyLogic DB
[Text input]	[Text input]

Update data on the model startup

Buttons: < Back, Next >, Finish, Cancel

Annotations:

- ① Set the Agent type name:** Distributor (callout pointing to the Agent type name field)
- ② Select Use database table** (callout pointing to the 'Use database table' radio button)



Supply Chain. Phase 1. Step 8

Step 4. Set up the agent parameters

Parameter	Column
location	location
nTrucks	n_trucks

Step 5. Agent animation

Choose animation: 2D 3D None

- TRUCK
 - Fork Lift Truck
 - Ship
 - Plane
 - Fighter Jet
 - Fighter Jet 2
 - House
 - Retail Store
 - Warehouse**
 - Factory



We will start with creating a population of agents modeling a set of distributors.

Creating agent populations

- AnyLogic provides a simple way of creating populations of agents by simply dragging the **Agent** element from the **Agent** palette on the diagram of the agent where you want to place the population (typically, *Main*).
- The Wizard creates:
 - Agent type (*Distributor*). This is the place where the agent's inner structure, behavior, animation, etc. are defined.
 - The agent population (*distributors*). It is embedded in the agent where you dragged the **Agent** (*Main*) and consists of several (in our case, 5) instances of agent type *Distributor*, each one modeling a particular agent-distributor. Near the element you can see the embedded agent's presentation.



Supply Chain. Phase 1. Step 9

The screenshot shows a software interface for configuring agent population. On the left, a sidebar titled "Main" contains a section for "distributors [...]" with a red circular icon. A blue callout bubble points to this icon with the text: "① Move agent population here, outside the area visible at model runtime". Below this is a "Initial location" dropdown menu. Inside the menu, a red arrow points to the "in the first result of map search" option, which is highlighted with a red border. The text "Place agent(s):" followed by three radio button options is also visible. The "Location name:" field is set to "self.location" and is also highlighted with a red border. The "Routing:" field is set to "Default". To the right of the sidebar is a map of France and surrounding regions, showing major cities like Paris, Lyon, and Brussels. A yellow cube icon is placed on the map near Dijon.

① Move agent population here, outside the area visible at model runtime

Initial location

Place agent(s):

- at the agent animation location
- in the latitude/longitude
- in the node
- in the first result of map search

Location name: = self.location

Routing: = Default

② Configure the population to place agents to the locations read from database



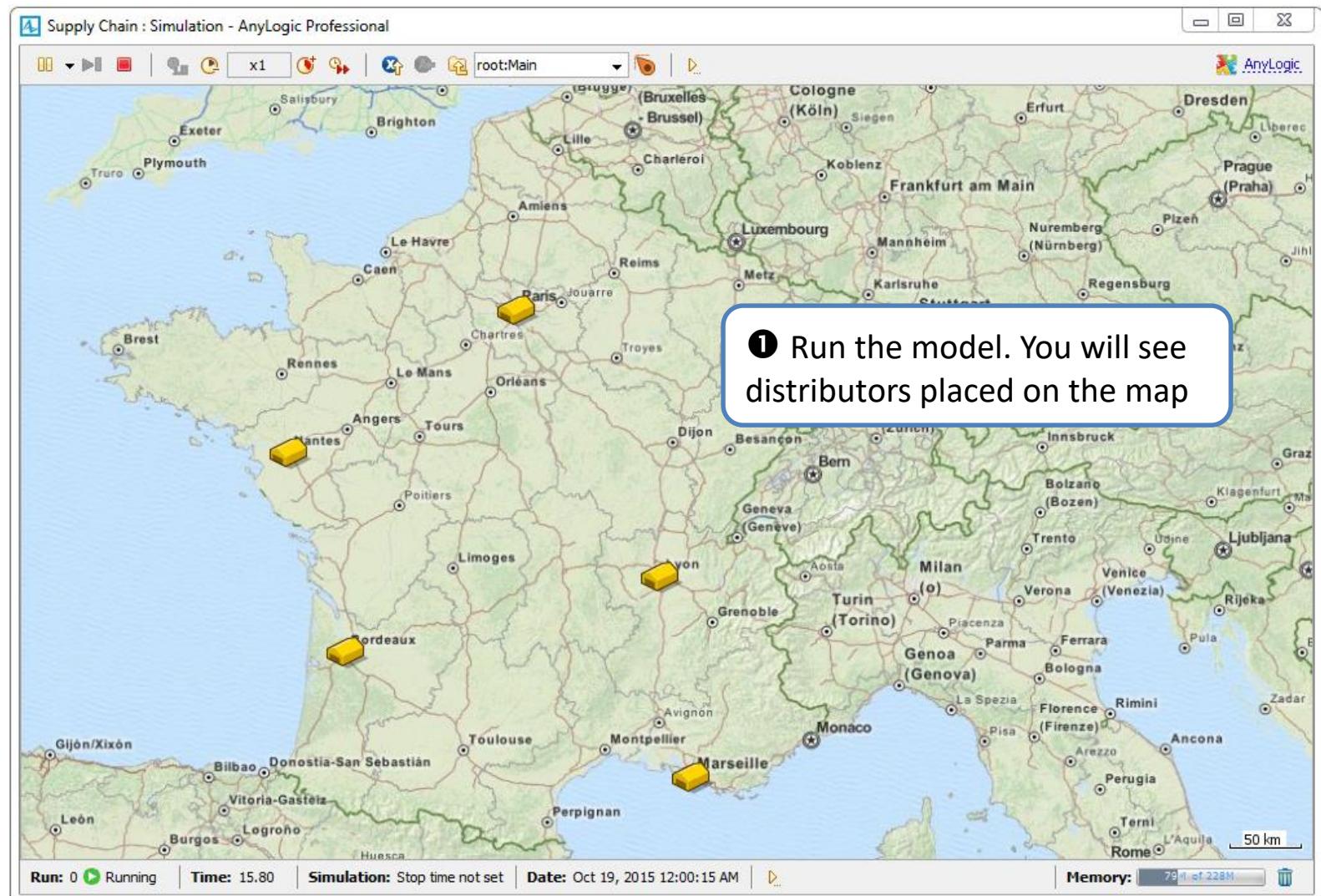
Placing agents in GIS space

In the **Initial location** properties section of the agent population, you can select where you want to place the agents from this population:

- **At the agent animation location** - Find the yellow warehouse shape on your map. It is agent's embedded presentation. With this option selected, agent will be placed exactly where you place this shape on the GIS map.
In case you draw animation for an agent afterwards creating its population on *Main*, and you can not see the agent's embedded animation, click **Show presentation** in the advanced properties of the agent population (e.g. *distributors*).
- **In the latitude / longitude** – You specify the agent's coordinates in the Latitude and Longitude fields below. If you have agent population, and agent has *lat* and *lon* parameters holding the coordinate values, specify *self.lat* and *self.lon* in the fields to arrange agents in GIS space according these coordinates.
- **In the node** – You draw locations on the map with GIS points, and specify the name of the GIS point where you want to place the agent. This option is helpful when you have a single agent, not agent population consisting of multiple agents.
- **In the first result of map search** – Use this option, when the locations are defined as text names (it is our case): "Paris", "Marseille", etc. These names are taken by GIS map, it searches for the locations with these names and uses the first search result. If your agent has parameter storing the location name (as we do, parameter *location*), type *self.location* in the field below to place agents to the required points in GIS space.

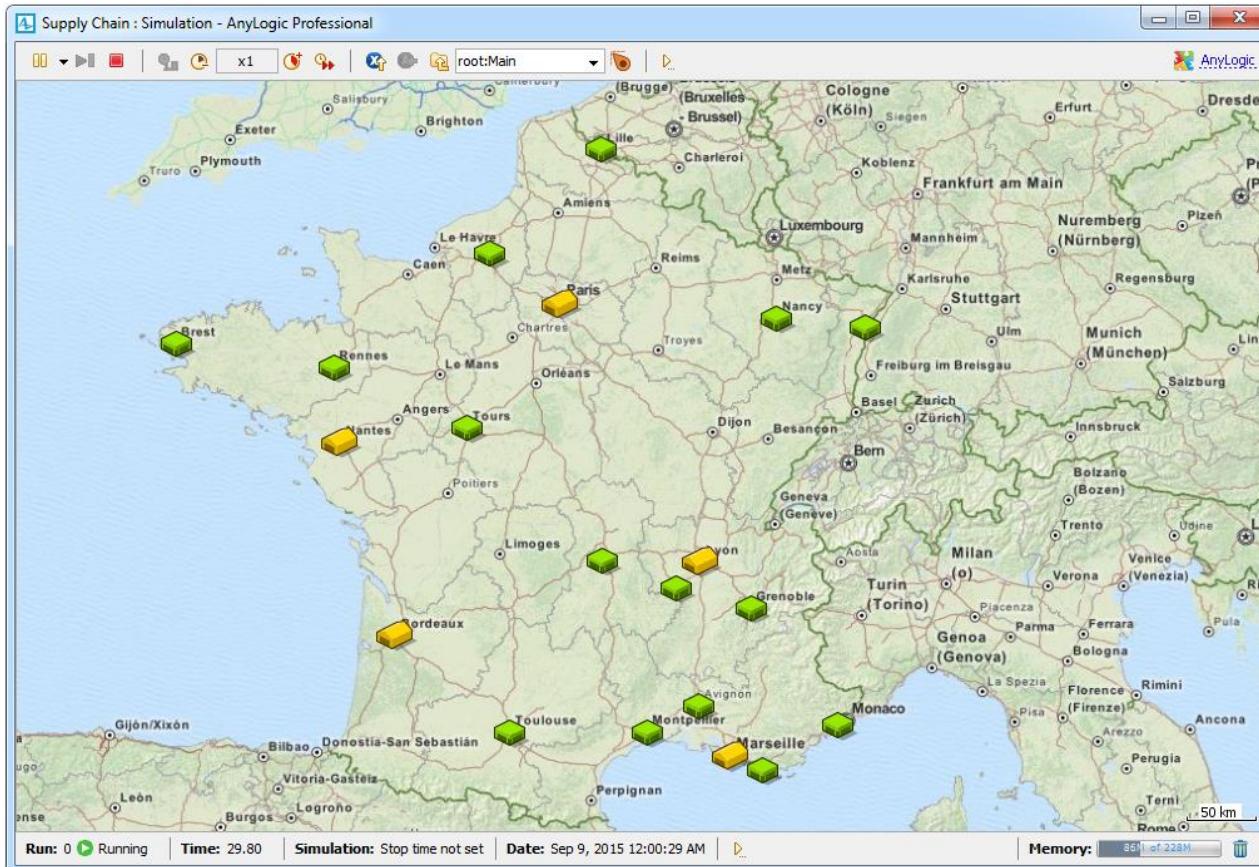


Supply Chain. Phase 1. Step 10



Supply Chain. Phase 1. Step 11

- Create retailers in the same way as distributors. Name the agent type *Retailer* and read data (locations) from the AnyLogic database.



Phase 2. Creating trucks

- Now we will create one more agent population modeling trucks.
- We will read data (number of trucks and their location) from the database.



Supply Chain. Phase 2. Step 1

The screenshot shows the AnyLogic software interface. In the top left, the Projects palette lists 'Agent' and 'MyAgent'. A red arrow points from the 'Agent' entry to the 'Main' workspace, where a blue oval labeled 'MyAgent' is visible. A callout bubble says 'Create a population of trucks'.

Step 2. Creating new agent type

Agent type name: **Truck** (highlighted with a red box)

Agent population name: trucks

Create the agent type "from scratch"

Use database table
I want to setup parameters of agents from database

Agent will be used in flowcharts

Step 3. Agent animation

Choose animation: 3D 2D None

General

- Person
- Nurse
- Doctor
- Patient

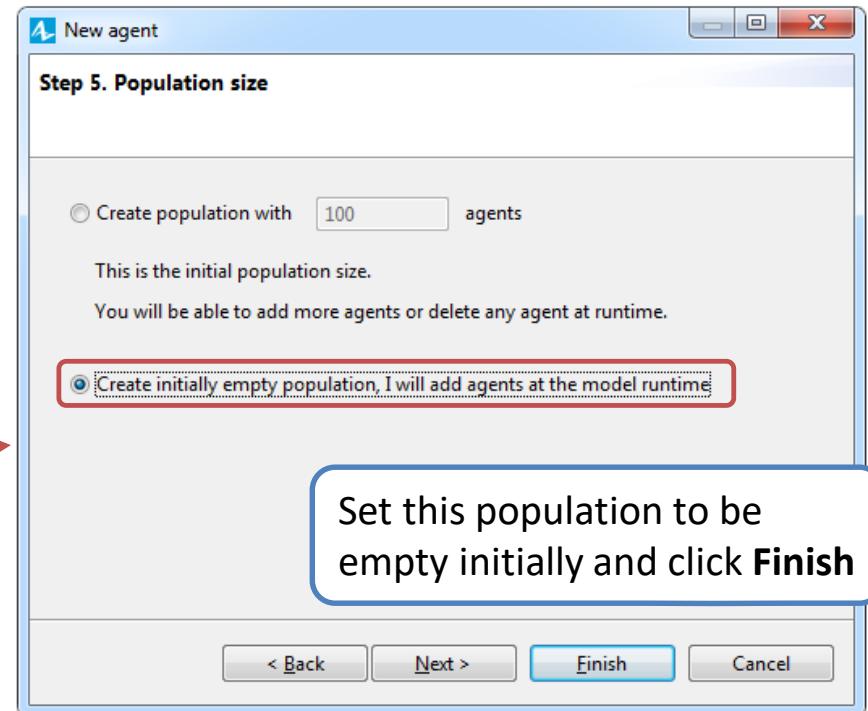
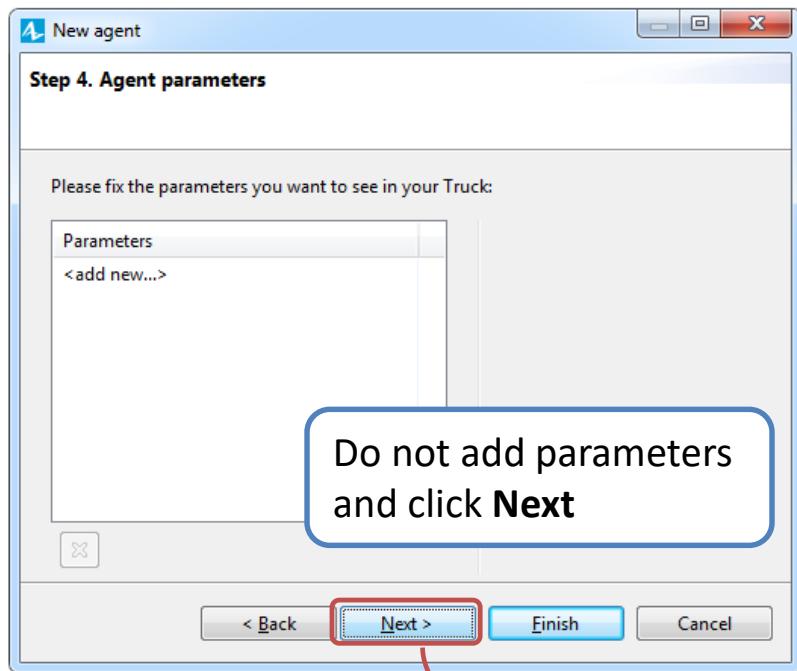
USA Map

- Lorry (highlighted with a red box)
- Lorry 2
- Truck
- Fork Lift Truck

A red arrow points from the 'Next >' button in the Step 2 dialog to the 'Next >' button in the Step 3 dialog. A callout bubble in the Step 3 dialog says 'Click Next'.



Supply Chain. Phase 2. Step 2



Supply Chain. Phase 2. Step 3

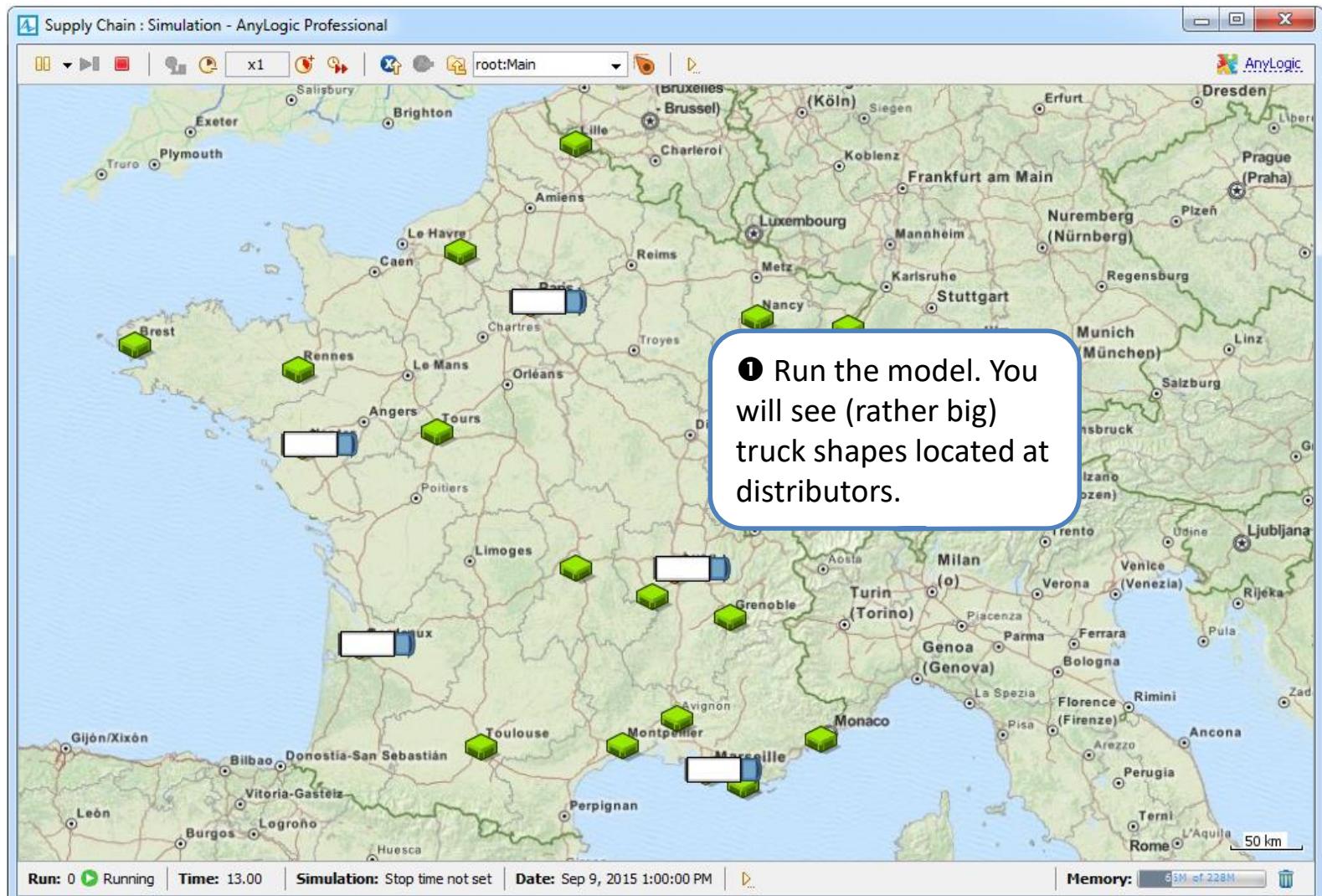
The screenshot shows the AnyLogic simulation environment. On the left is the 'Main' window, which contains a map of a coastal region with locations like Truro, Plymouth, and Brest marked. To the left of the map is a sidebar with icons for 'distributors [..]', 'retailers [..]', and 'trucks [..]'. The 'trucks [..]' icon is highlighted with a blue circle and has a blue callout bubble pointing to the text 'Modify trucks properties'. On the right is the 'Properties' window, titled 'trucks - Truck'. It displays various configuration options for the truck agent:

- Name: trucks
- Show name: Ignore:
- Single agent: Population of agents:
- Population is:
 - Initially empty:
 - Contains a given number of agents:
 - Loaded from database:
- Table: distributors
- Choice conditions: (with icons for adding, removing, and reordering)
- Mode:
 - One agent per database record:
 - Multiple agents per record:
- Quantity is contained in column: n_trucks
- Agent parameters mapping:

Parameter	Column
GIS location name	location



Supply Chain. Phase 2. Step 4



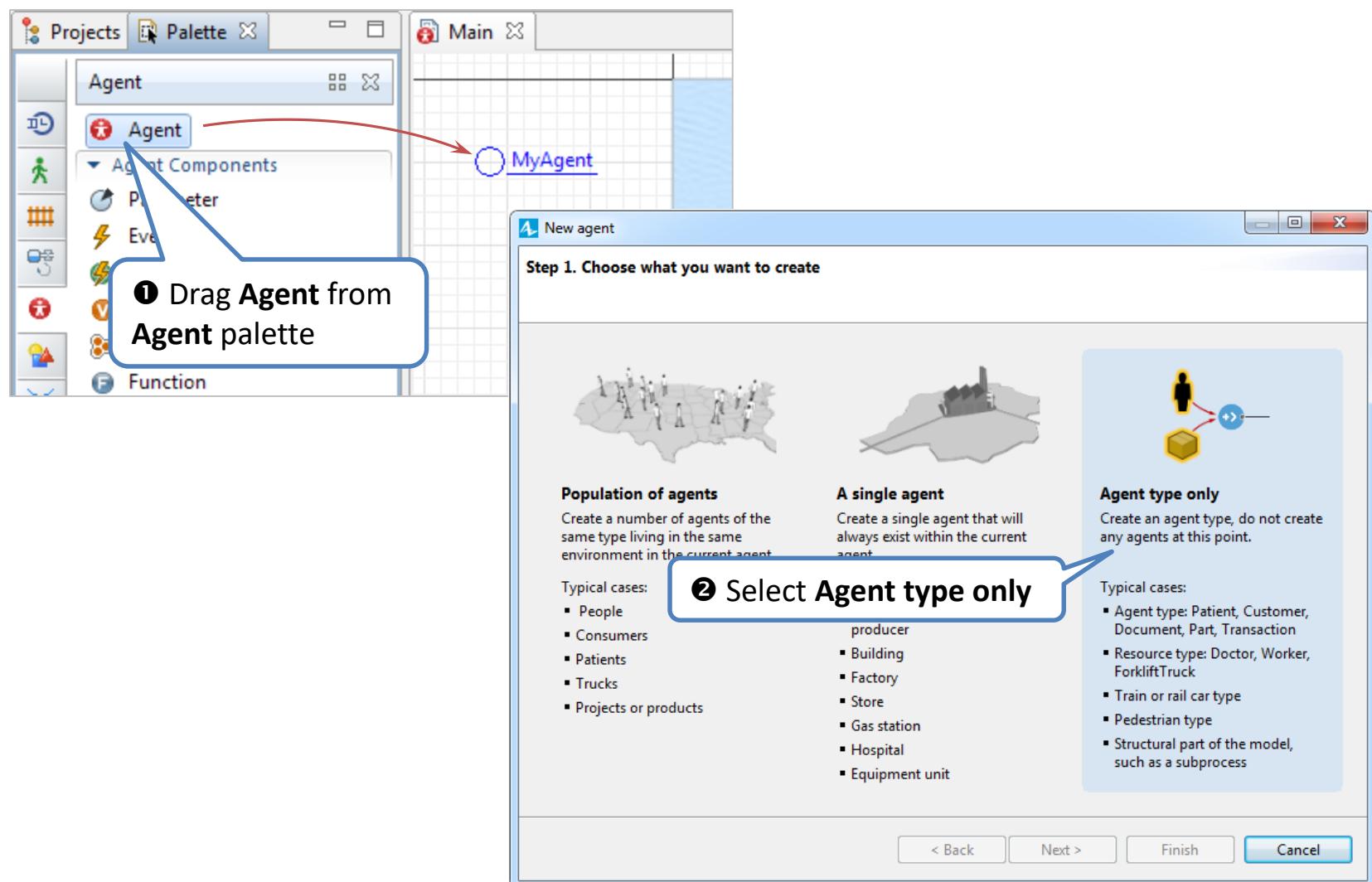


Phase 3. Defining the logic

- Now we will model how products are requested by retailers and delivered from distributors to retailers.
- This requires us to generate product requests and assign them to trucks and make trucks move to retailers and return to their home locations at the distributor.



Supply Chain. Phase 3. Step 1



Supply Chain. Phase 3. Step 2

The image shows three sequential steps in the AnyLogic 'New agent' dialog:

- Step 2. Creating new agent type**: The 'Agent type name:' field contains 'Order'. A callout bubble points to it with the text **① Agent type name: Order**.
- Step 3. Agent animation**: The 'Choose animation:' dropdown has 'None' selected. A callout bubble points to it with the text **② Choose animation: None**.
- Step 4. Agent parameters**: A callout bubble points to the 'Parameters' section with the text **③ Add a parameter to store the amount of products requested by this order: amount of type int**. The 'Parameters' list shows 'amount <add new...>'. To its right, a parameter is defined with the name 'amount' and type 'int'.

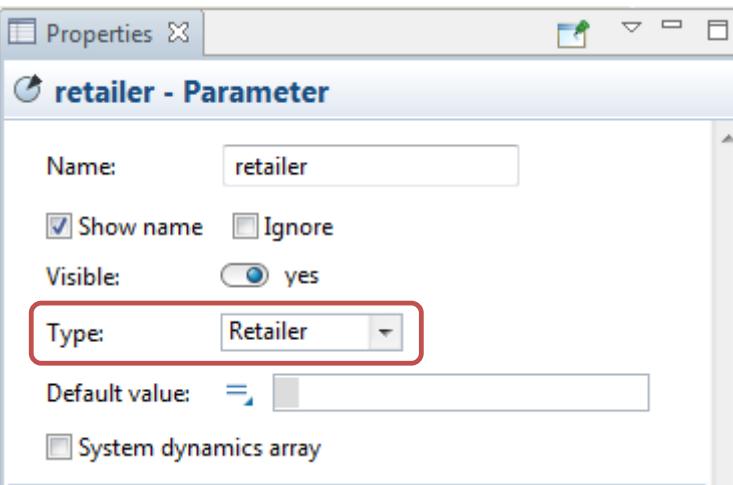


Supply Chain. Phase 3. Step 3

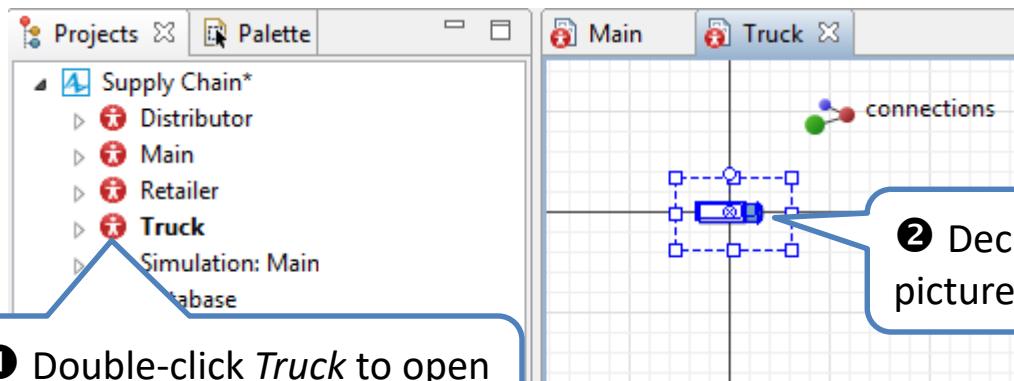
The screenshot shows the AnyLogic modeling environment. On the left, the Projects palette lists a project named 'Supply Chain*' containing agents like Distributor, Main, Order, Retailer, Truck, and a Main population. A blue callout points to the 'Order' agent with the instruction: '① Double-click Order to open its graphical diagram'. In the center, the Main workspace shows a graphical diagram with two nodes: 'amount' and 'retailer'. A blue callout points to the 'retailer' node with the instruction: '② Add one more parameter (Parameter from Agent palette) to remember the retailer - originator of this order (retailer of type Retailer)'. On the right, the Properties palette is open for the 'retailer' parameter, showing settings: Name: 'retailer', Show name: checked, Ignore: unchecked, Visible: yes, Type: 'Retailer' (highlighted with a red border), Default value: '=', and System dynamics array: unchecked.

① Double-click *Order* to open its graphical diagram

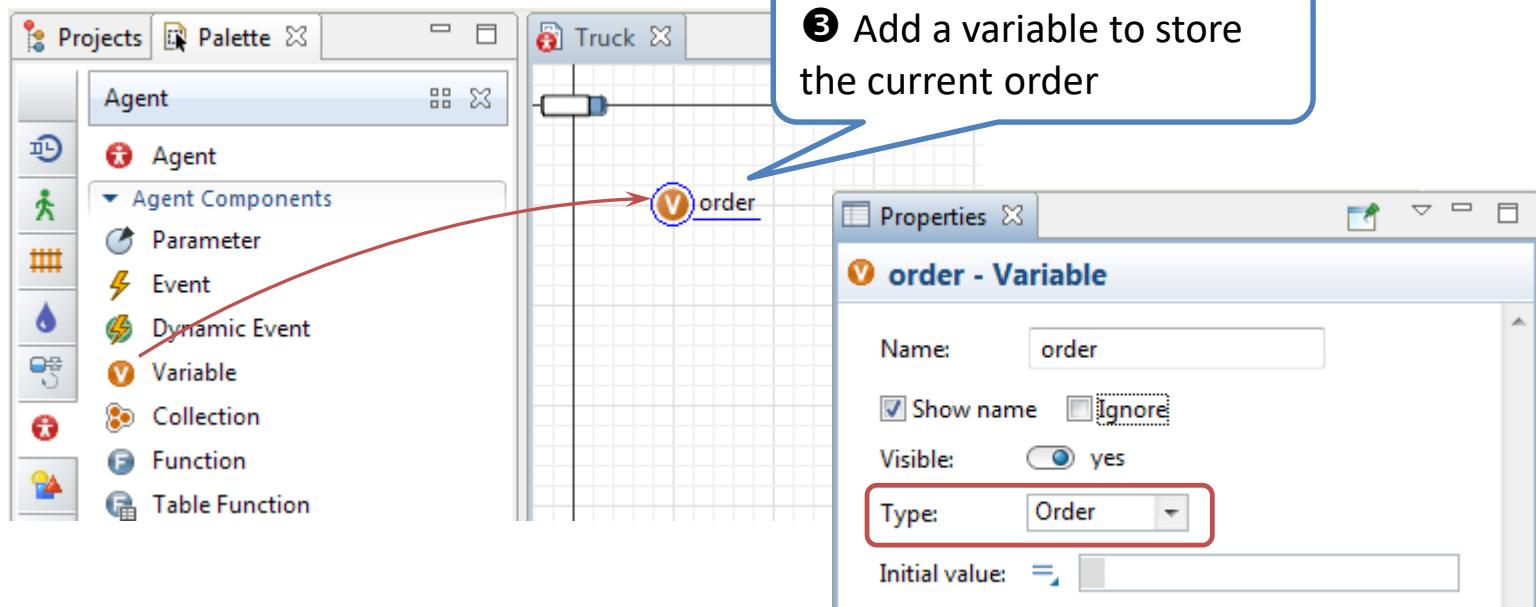
② Add one more parameter (Parameter from Agent palette) to remember the retailer - originator of this order (*retailer* of type *Retailer*)



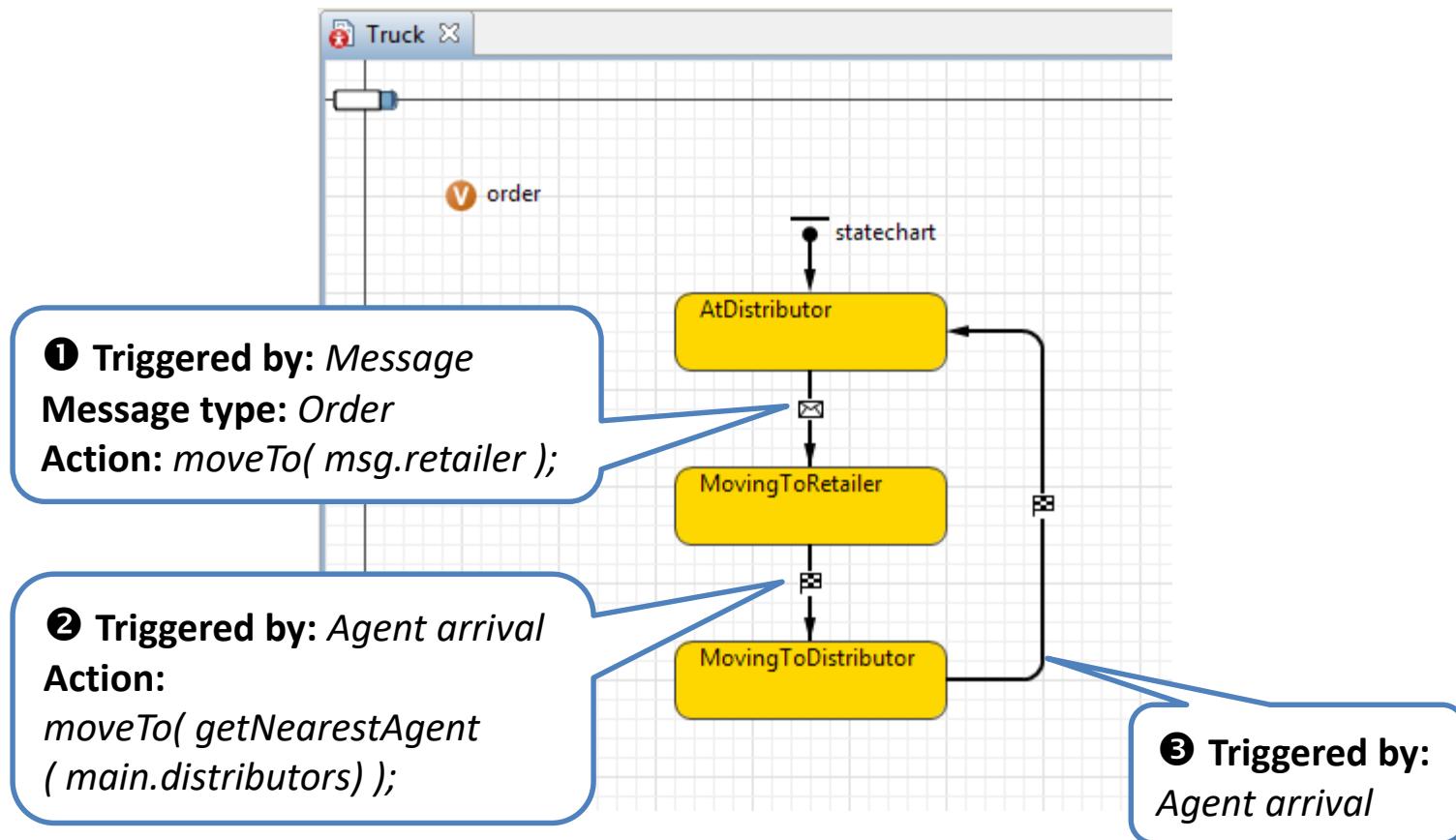
Supply Chain. Phase 3. Step 4



② Decrease the *Lorry* picture.



Supply Chain. Phase 3. Step 5



Define truck behavior with a statechart. In our case the truck is initially at the distributor location. (So this is the first state – *AtDistributor*). While there the truck can receive an order and move to the retailer – the originator of this order. For now we assume that having reached the retailer, it returns immediately. Movement is a continuous action and we can represent it with a state. So there are two more states – *MovingToRetailer* and *MovingToDistributor*. To end being at a state, we use transitions triggered upon agent arrival to the defined destination.

Local variables

- We use *msg* variable in code of state's **Action**. *msg* is a local variable available in the given context. To see a list of local variables click in the field and then hover the mouse over the bulb shown to the left of the field:



Agent movement in GIS space

- To move an agent to some new location, call *moveTo()* method of this agent, passing coordinates of the new destination as parameters.
- The following methods may be of interest also, consult *AnyLogic Help* for the full list:

Method	Description
<i>moveTo(double x, double y)</i>	Starts movement to the given target location
<i>jumpTo(double x, double y)</i>	Immediately places the agent at a given location.
<i>moveTo(String geographicPlace)</i>	Starts movement to the specified location on the map.
<i>moveTo(node)</i>	Starts movement to the specified GIS markup element: point or region.
<i>stop()</i>	Stops the agent and leaves it at the current location.
<i>double timeToArrival()</i>	Returns time to arrival to the target, in model-time units.
<i>boolean isMoving()</i>	Tests if the agent is currently moving.
<i>double getVelocity ()</i>	Returns the current velocity of the agent (agent's "cruising speed", non-zero velocity does not mean the agent is moving).
<i>setVelocity(double newVelocity)</i>	Sets the velocity ("cruising speed") of the agent. If the agent is moving, it will continue moving from the current location with the new velocity. If the agent is not moving, it will not start moving until you call <i>moveTo()</i> .
<i>setLatLon(double latitude, double longitude)</i>	Sets the coordinates of the agent location.
<i>double getLatitude()</i>	Returns the current latitude of the agent.
<i>double getLongitude()</i>	Returns the current longitude of the agent.



Supply Chain. Phase 3. Step 6

The screenshot shows the AnyLogic software interface. On the left is the Projects palette, which contains the 'Agent' component. The main workspace shows a 'Retailer' diagram with a green hexagonal agent icon and a blue circular location icon. A blue arrow points from the 'location' icon to a callout box labeled '1 Open Retailer diagram'. A red arrow points from the 'Parameter' icon in the palette to the 'orderProducts' event in the diagram. To the right is the Properties panel, which is currently displaying the settings for the 'orderProducts - Event'. The event has a name of 'orderProducts', is set to trigger at a rate of 5 per day, and is visible. The Action section contains Java code for creating an order and sending it to a truck if one is available.

1 Open Retailer diagram

Properties

orderProducts - Event

Name: orderProducts Show name

Ignore

Visible: yes

Trigger type: Rate

Rate: 5 per day

Log to database

Action

```
Order order = new Order( uniform_discr(5, 20), this );  
Truck truck = getNearestAgent(  
    filter(main.trucks, t -> t.inState(Truck.AtDistributor)));  
  
if (truck != null)  
{  
    send(order, truck);  
}
```



Create *orderProducts* event that periodically sends product requests. Make the event occur on average 5 times per day, and define the event's action.

Here we create a new order by calling a constructor of *Order* type:

new Order(amount, this). The constructor takes two arguments initializing *Order* agent type's parameters: *amount* and *retailer*. Take a look at the code of *Order* Java class to understand the order for the arguments to follow. We initialize the *retailer* field with a reference to this *Retailer* agent accessible via reserved Java word *this*.

Then we search for a nearest free truck.

- The function *getNearestAgent(population)* returns the nearest agent from the specified *population*. We do not consider all the population, but we filter the population using the *filter()* function, and search for the nearest truck only from the subpopulation, containing only the trucks that are currently idle.
- The first argument of *filter()* function takes the agent population, in this case *main.trucks*. Then you specify the name for the current agent (*t ->*), and then you may specify the filter condition(s), *t.<condition>*. The function will return the subpopulation containing the agents satisfying this condition(s).
- The function *inState(state)* checks whether the specified state of the agent's is currently active. This way we filter the results and consider only idle trucks (that are currently located at distributor).

If we finally find an idle truck, we send the message-order to its statechart to initiate movement. If all the trucks are busy, *null* is returned.



AnyLogic event object

- Event schedules some action at the specified moment of time in the future
- Events may be triggered by
 - Rate
 - Condition
 - Timeout. Timeout triggered events have three alternative occurrence modes:
 - *Occurs once*
 - *Cyclic*
 - *User control*



Trigger type:

Timeout

Mode:

Cyclic

First occurrence time:

0

Recurrence time:

1 day

Action:

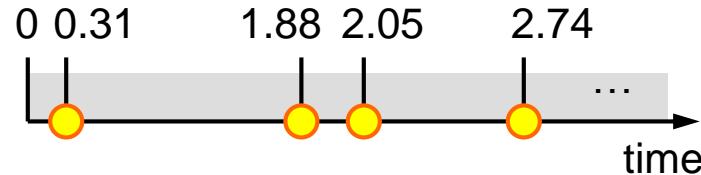
onEndOfDay();



Rate triggered event

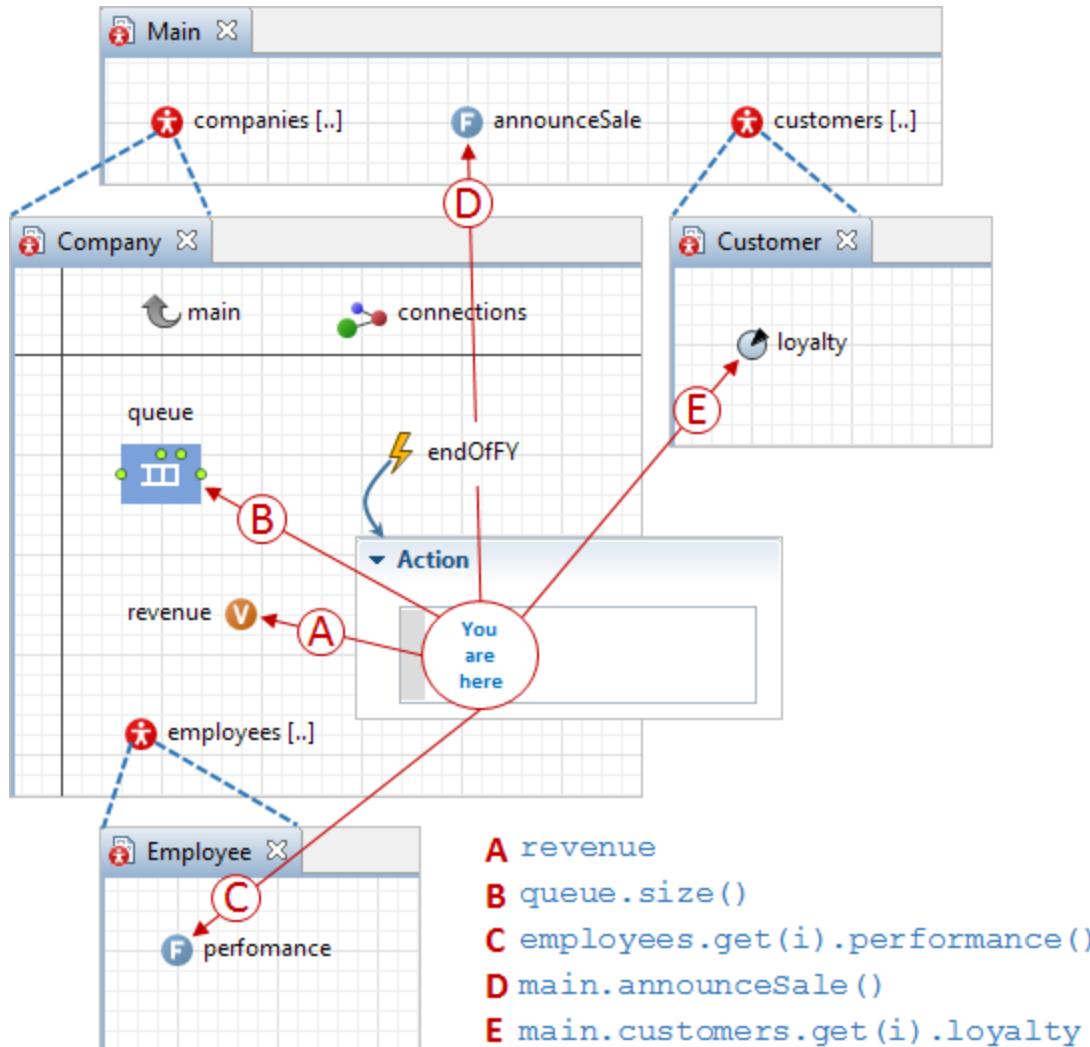
- Similar to timeout cyclic events with stochastic recurrence time

Trigger type: Rate
Rate: 1
Action: arrival();



- Rate – mean number of events in time unit
- $1/\text{Rate}$ – mean recurrence time
- Recurrence time distribution is an exponential distribution with Rate as the shape parameter

Writing Java code. How do I get to... ?

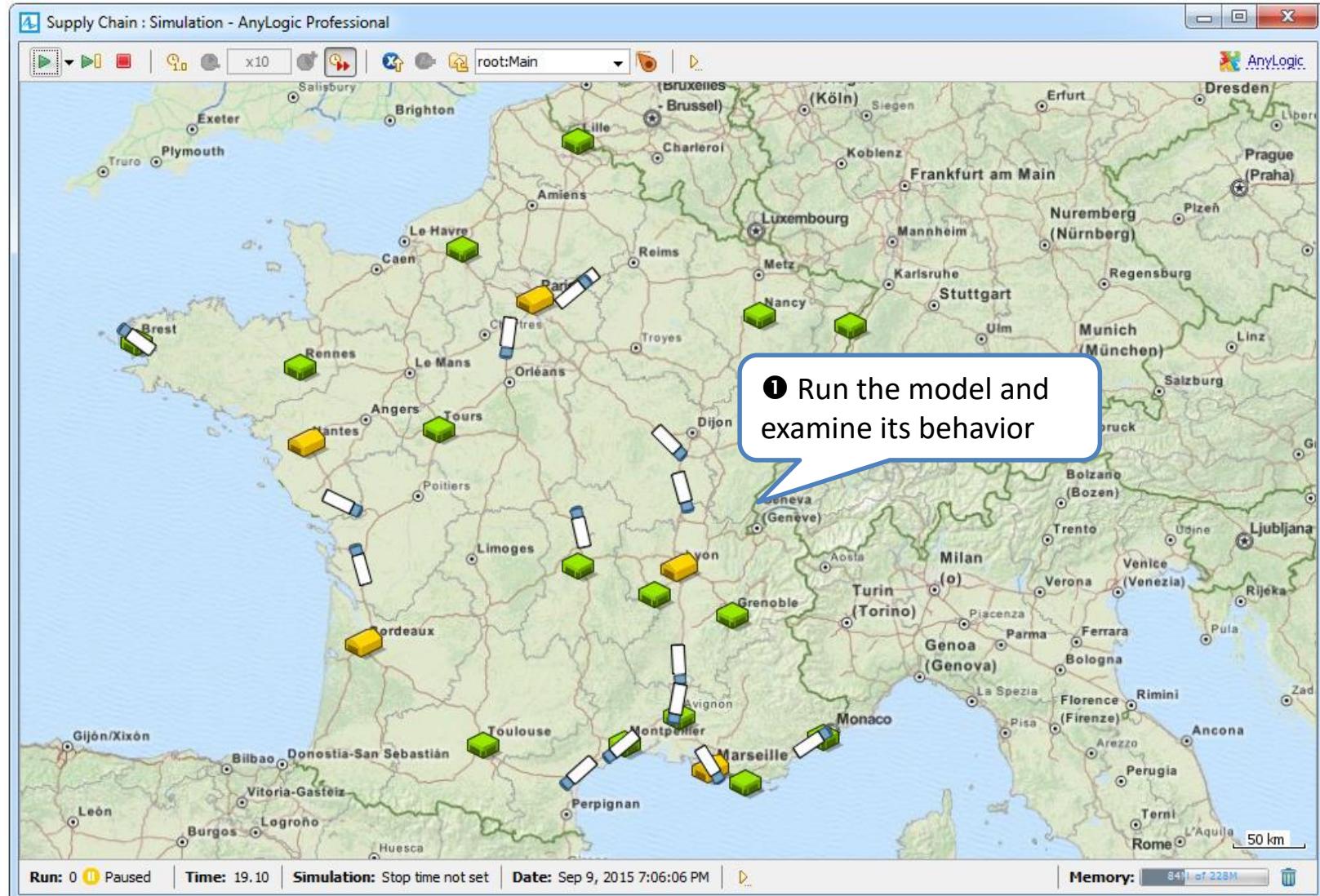


In AnyLogic you enter pieces of code in numerous properties of various model elements. It is important to understand exactly where you are writing the code (which class and method it belongs to) and how you can access other model elements from there.

- The elements of the same agent type are accessed simply by their names. For example, you are in the **Action** field of the event *endOfFY* of agent type *Company*. To access the block *queue* you simply write *queue*. To increase the variable *revenue* you write *revenue += amount*.
- To access an element of an object you should put a dot "." after the object's name and then write the element name. For example, to obtain the size of the *queue* you write *queue.size()*. For the agent population, you should say which agent exactly you want to access. To call the function *performance()* of the employee number 247 among employees of the company, write: *employees.get(246).performance()*.
- Access the upper level agent by the name of the Link to upper level. If the agent lives in *Main*, write *main* to get access to *Main* from the agent. To call the function *announceSale()* of *Main*, write *main.announceSale()*.
- To access the agent from other population living in the same environment, first get one level up to *Main*, and then get down to another agent. To access a customer's *loyalty* from a company, we need first to get to *Main* and then to *Customer* (with some index): *main.customers.get(i).loyalty*.



Supply Chain. Phase 3. Step 7





Pedestrian Modeling

This presentation is a part of
AnyLogic Standard Training Program



What are pedestrian models built for?

- At a preliminary project assessment stage
 - Assess the ability of a facility to cope with a planned loading and comply with safety requirements
- At the stage of the design of a new facility
 - Assess alternatives, promptly assess revisions, seek the best solutions
- During construction /maintenance works at an operating facility
 - Seek the least inconvenient temporary routes
- As well as for presenting your project in a contest
 - Pedestrian models enable to obtain high quality and convincing animation and vividly demonstrate your offer
- At operating facility
 - Increase a throughput capacity, arrange queues
 - Optimize the operation of services (number of personnel, working hours)
 - Allocate signage
 - Assess the throughput capacity of a facility at a planned increase of loading
 - Optimize time schedules (for example, train schedules)
 - Allocate advertisement, goods, retail outlets
- Safety
 - Plan escape routes
 - Vulnerability assessment for terroristic attacks and catastrophes



Which facilities are modeled?

- Railway stations
- Metro stations
- Airports
- Pedestrian passageways

In general all the facilities where the arrangement of physical space for pedestrians affects throughput capacity, quality of service, and safety

transport

"attractions"

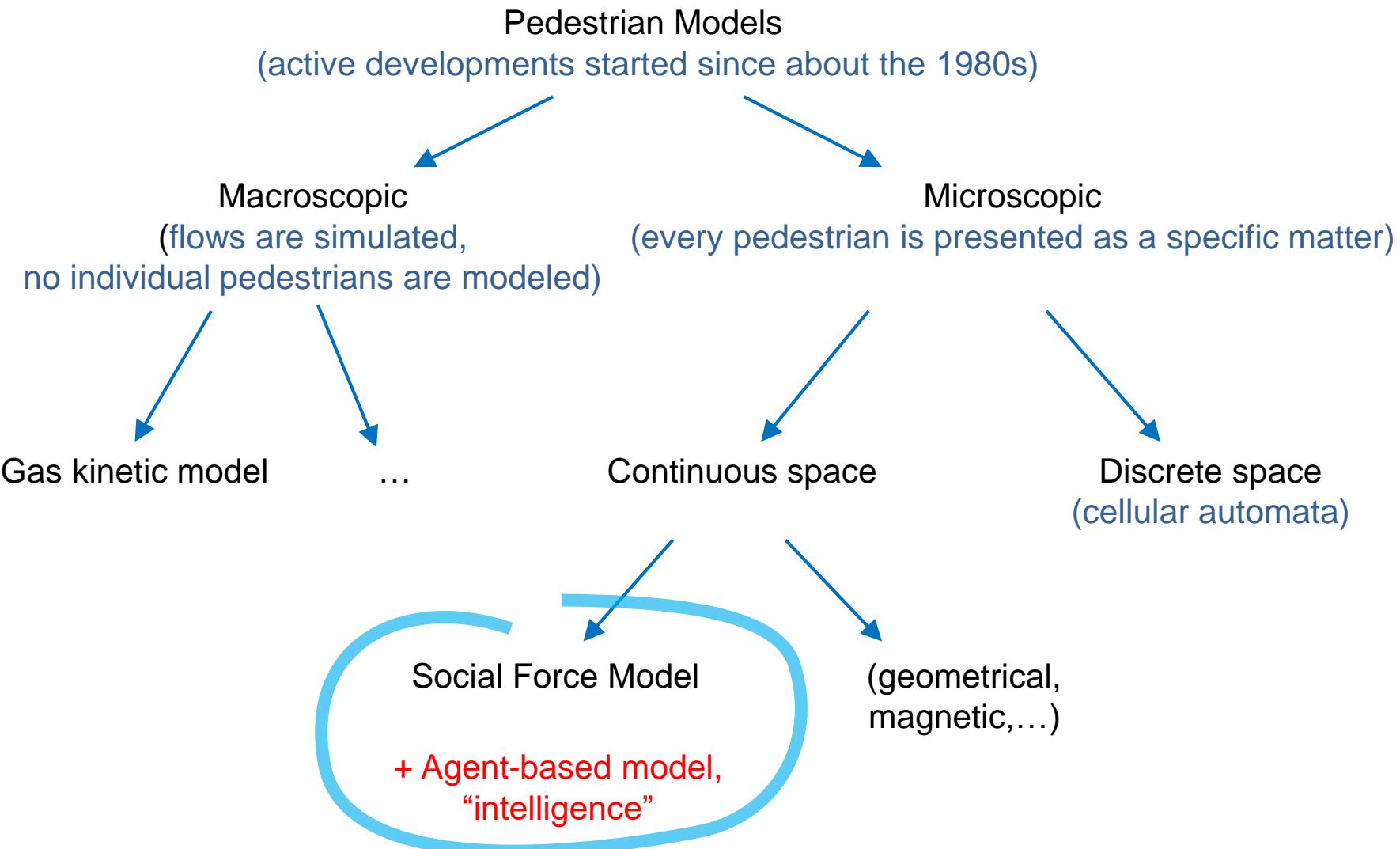
events

- Shopping malls
- Museums
- Amusement parks

- Stadiums
- Concert halls
- Street events (festivals, rallies, demonstrations)

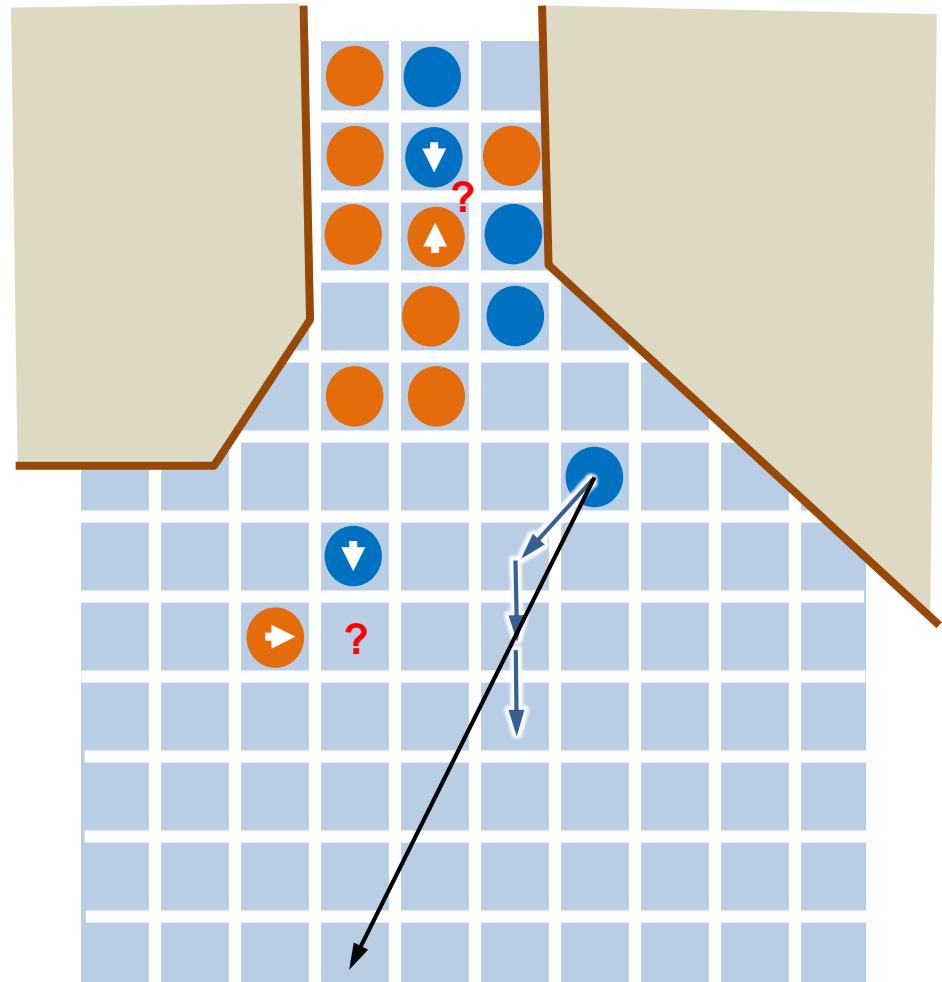


Theory. Pedestrian Model Types



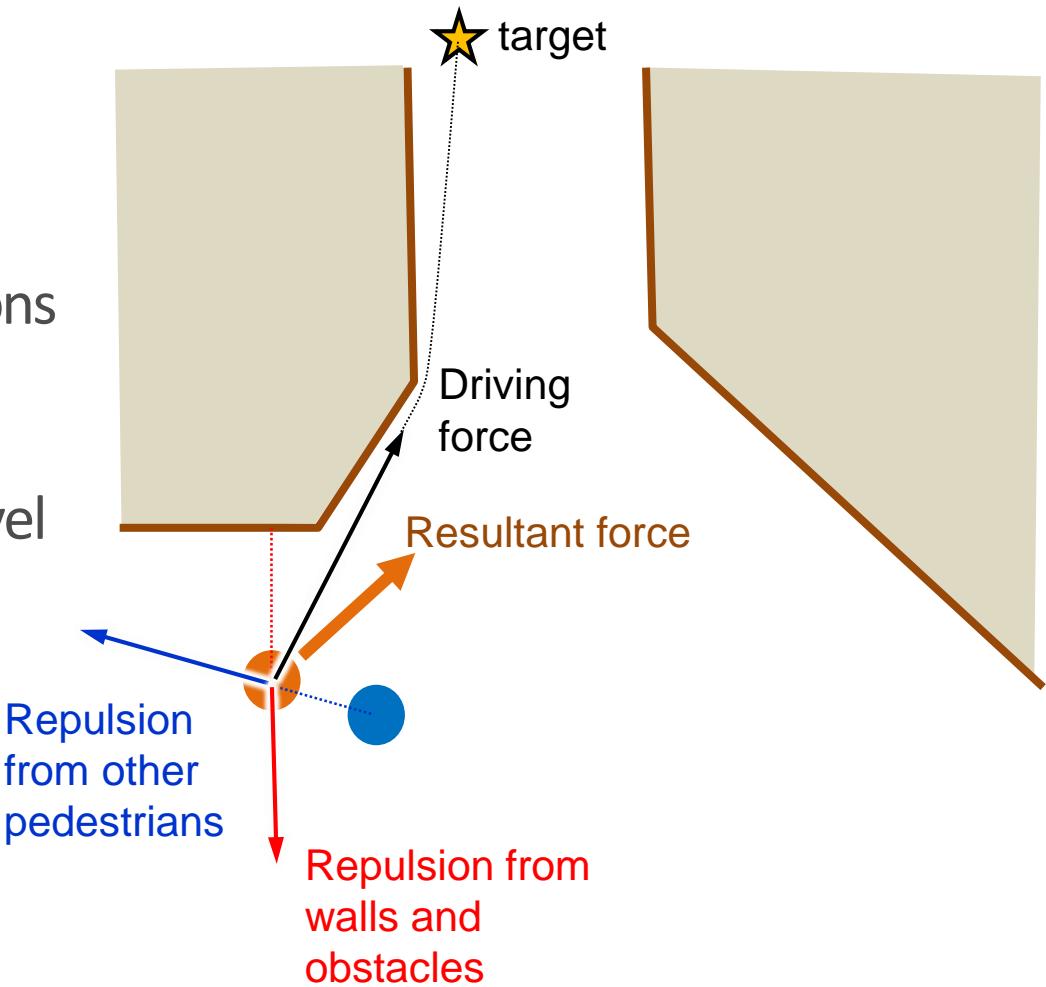
Cellular automata

- Easy local rules
- Fast-to-calculate
- Can be well-calibrated
- Poor animation
- See Blue & Adler

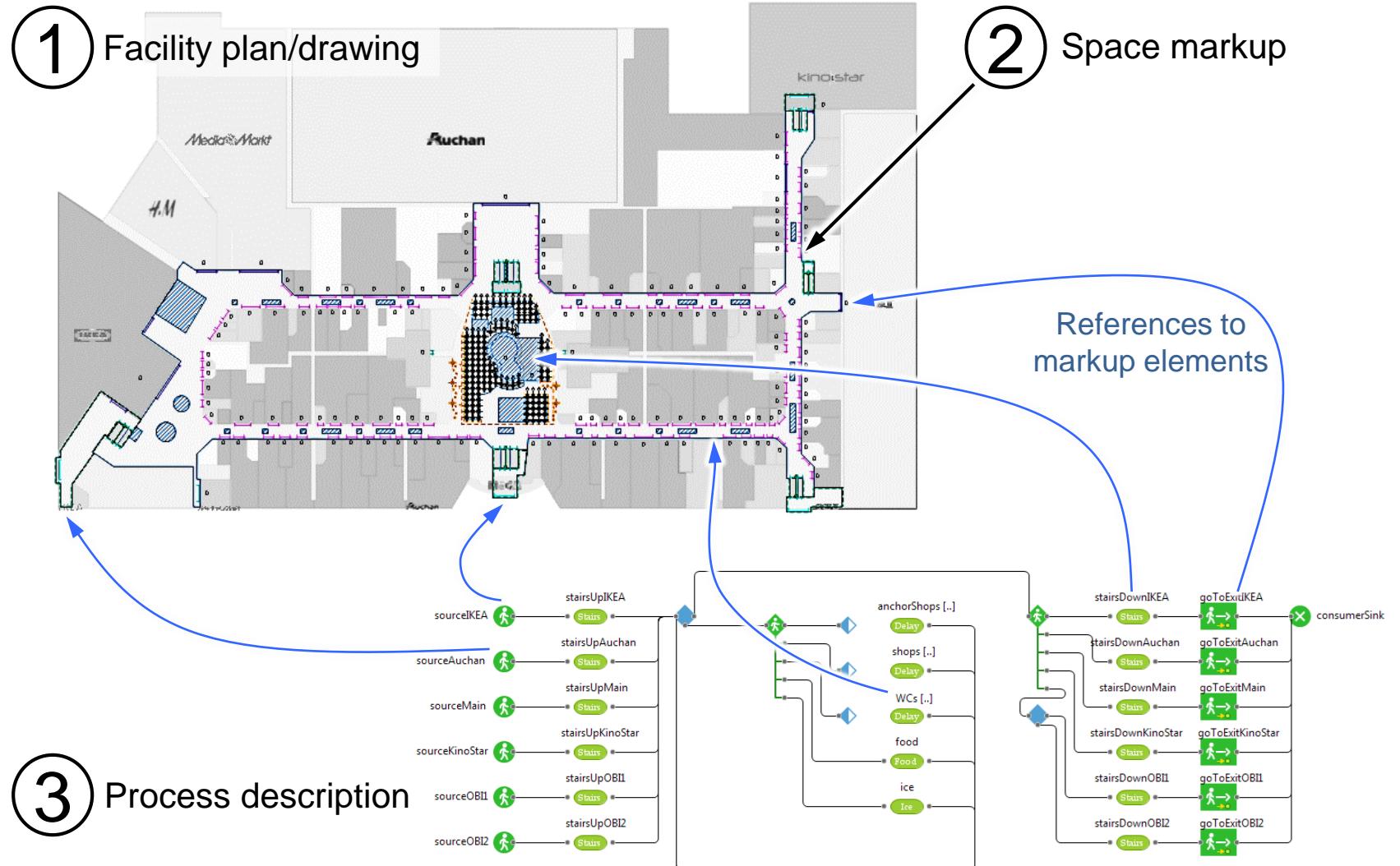


Social Force Model

- Newton mechanics
- Realism
- Relatively slow calculations
- Very realistic animation
- Extended with higher level decision making logic
- See Helbing & Molnar



How are pedestrian models built?



Space Markup elements

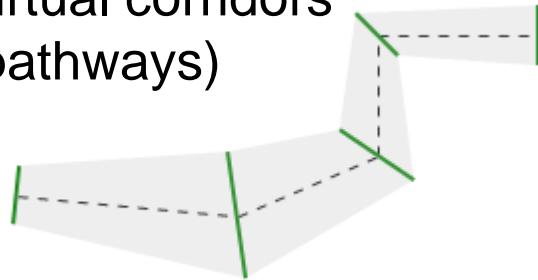
Walls



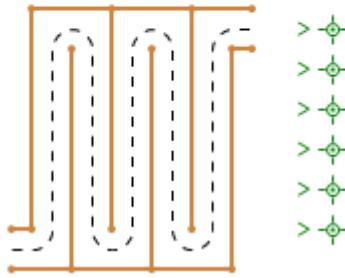
Target lines / pedestrian appearance lines



Virtual corridors (pathways)



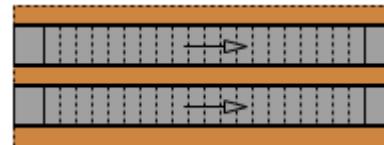
Services (service points) and queues



Waiting areas / target areas



Escalator



Process Description Basic Blocks

PedSource



Creates pedestrians on a line, at a point or in an area with a given rate, according to a time schedule, etc.

PedGoTo



Sets up an objective or a route

PedService



Sets servicing parameters (where is a delay, the selection of a queue, etc.)

PedSelectOutput



Divides a passenger flow

PedWait



Sets waiting parameters (where to wait, in relation to time, until an event)

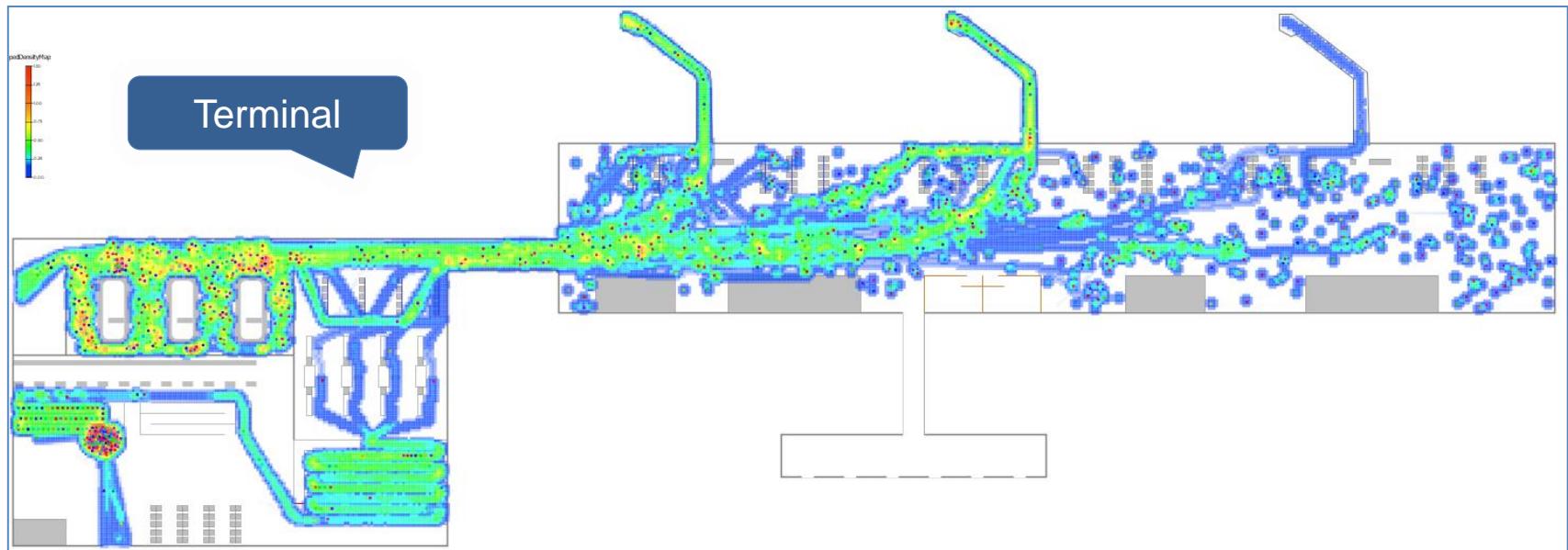
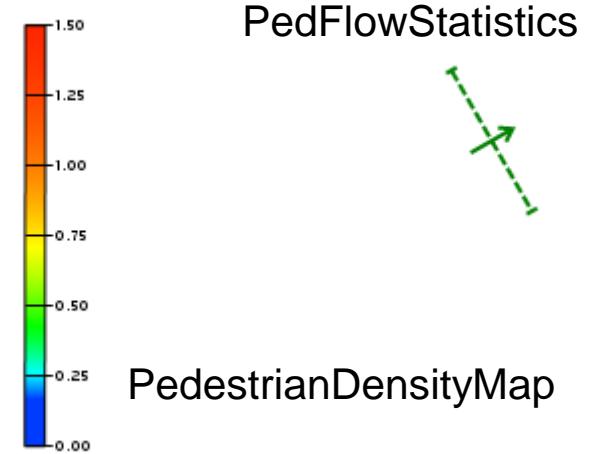
PedSink



Deletes passengers from the model

Measurements and Statistics in Pedestrian Models

- Metrics specific for pedestrian models
 - Flow characteristics: the total number of passenger having passed through a section per a unit of time, the same quantity per a unit of length
 - Density in a certain area: the number of passengers per square meter (average per a unit of time); density charts



Individual Features of a Pedestrian

- Since each pedestrian is modeled as an agent, individual features can be adhered to them
- Features built in a basic model:
 - Comfortable speed
 - Dimension (“diameter”)
- These can be added to them:
 - Individual targets (flight, platform, shop)
 - Servicing class (first / business / economy)
 - Citizenship (US/EU/other, ...)
 - Servicing speed
 - ...
- These features can be checked during the pathway of a pedestrian throughout a process diagram and affect their behavior



Groups

- Pedestrian behavior in groups considerably differs from that of independent pedestrians
 - Hold together; how to go (“in a rank”, “in a convoy”, “in a flock”)?
 - The presence or absence of a leader (for example, a guide)
 - Service: one pedestrian is serviced for all? (buys tickets for all, whereas a security check should be passed by everybody)
 - Does everybody stand in a queue? Where are those who are not standing in a queue waiting?

PedSource Is able to set groups, sets initial construction as well as default behavior at servicing points



PedGroupChangeFormation
Changes a group formation type



PedGroupAssemble



The same but out of the existing independent pedestrians

PedGroupDisassemble



Divides a group into individual independent pedestrians



Groups. Demo



Stairs, Escalators, Roads, Floors

- To set sloped surfaces as well as to modify speed on a surface PedArea with a Sloped option and its "descriptor" is used

PedAreaDescriptor



Limits / multiplies speed of pedestrians or sets a moving surface

- Each markup element belongs to a certain level
 - On default it is a ground level, but you may set new ones
- Special object is used to move between the levels:

PedChangeGround



Moves pedestrians from one level to another

Demo

Slope Demo

Demo

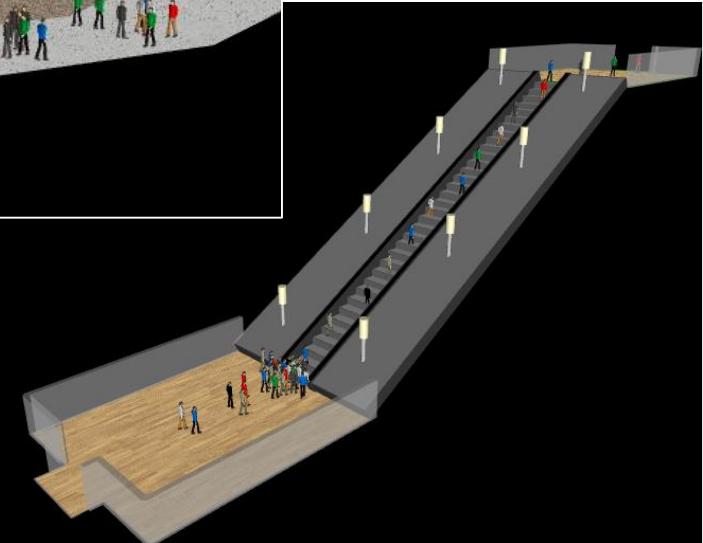
Escalator Demo

Demo

Change ground

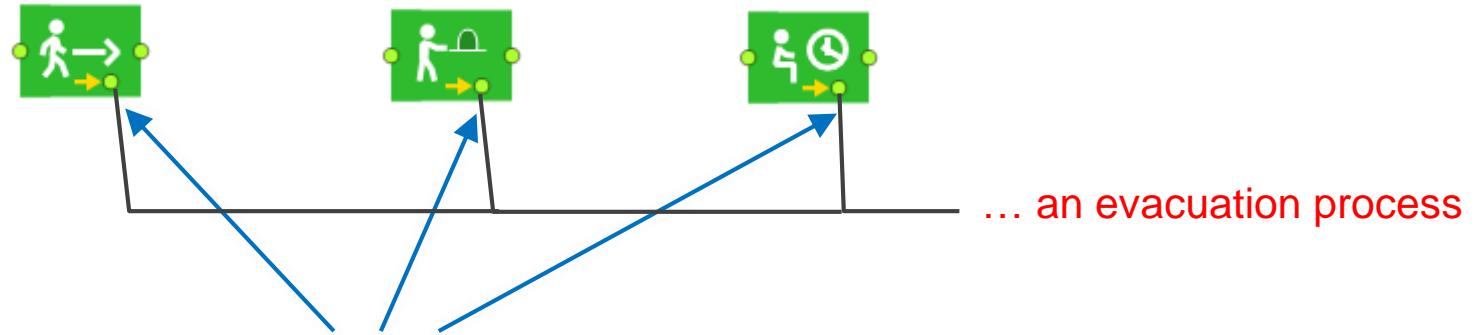


Stairs, etc. Demo



Normal Process Interruption. Evacuation

- All the pedestrian process objects where pedestrian spend nonzero amount of time , have a special exit where pedestrians can be redirected at any time in the event of a special event



When a pedestrian block's **cancel()** function is called pedestrians are immediately directed to this exit and thereafter they will follow a process defined by you by using pedestrian blocks

Demo 

Terrorist Attack



Combination with Other Model Types

- There are other objects in the model requiring other technologies
 - For example, transport
- If a large facility is modeled, then pedestrian modeling is important for some of its parts only
 - For example, in an airport terminal pedestrian problems are suspected in security check and passport control areas, however everything should be modeled from check-in to gate
 - Then for the remaining process part **pedestrians should not be used**. Use ordinary discrete event modeling which is much faster

Demo

Railway station

PedExit



Transforms a pedestrian into an ordinary “request”

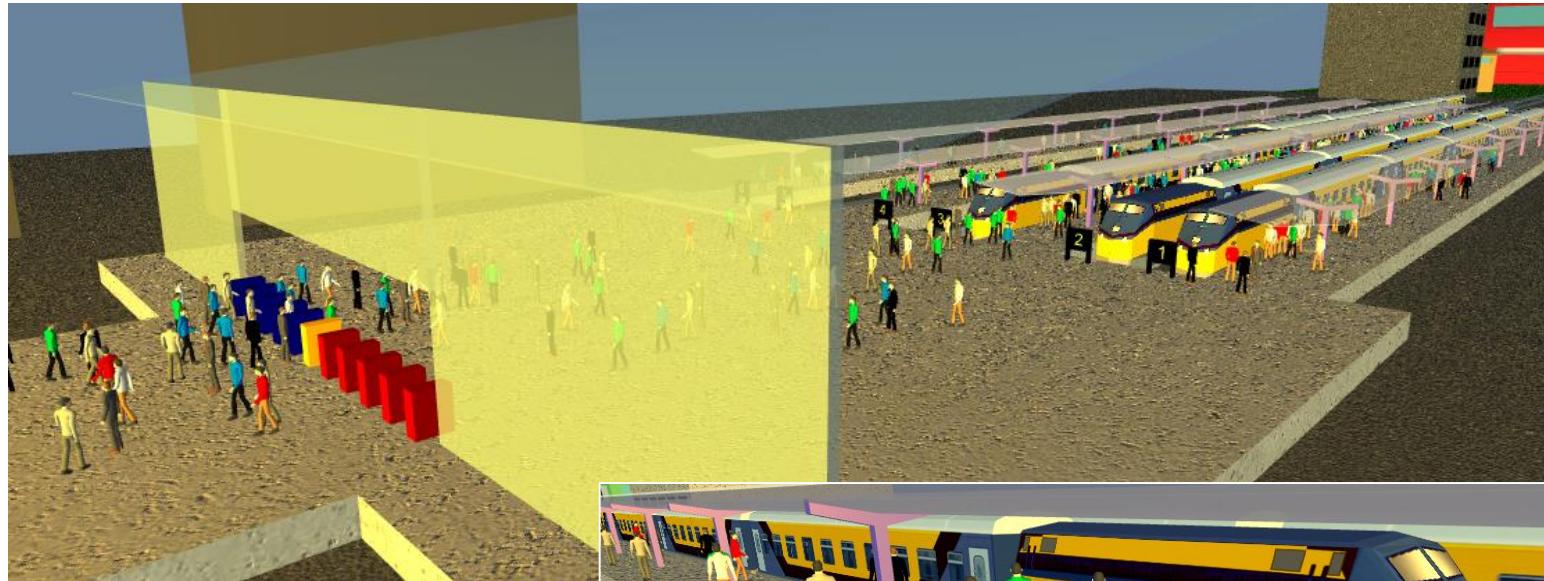
PedEnter



Transforms a request into a pedestrian



Pedestrians + ... Demo



Eiffel Tower. Demo

Eiffel Pedestrian : Simulation - AnyLogic Professional

1 : 2 root:Tower

AnyLogic

The screenshot shows a simulation environment for the Eiffel Tower. On the left, two floor plans of the tower's base show the distribution of black dots representing pedestrians. A compass rose is located between the floor plans. In the center, a 3D model of the Eiffel Tower is shown with visitor counts: Montants: 123, Descendants: 37, and a note that Rotation DUO 1: 6 and DUO 2: 6. To the right, four graphs provide performance metrics: 1) A histogram of visitor travel times between levels 3 and 4, peaking at approximately 18 seconds. 2) A histogram of elevator wait times at level 3, peaking at approximately 2 seconds. 3) A line graph of the number of visitors at levels 3 and 4 over time, showing a sharp increase starting around second 30. 4) A memory usage monitor at the bottom right indicating 39M of 247M and 28.4 sec. The status bar at the bottom shows Run: 0 Paused, Time: 36.76, Simulation: 31%, and Memory: 39M of 247M.

Simulation dynamique des projets de modification des espaces au sommet de la tour Eiffel. Situation projet

Montants/heure: 473

Rotation
DUO 1: 6
DUO 2: 6

Montants: 0
Descendants: 0

Montants: 21
Descendants: 4

Montants: 123
Descendants: 37

Temps de parcours des visiteurs niveaux 3 et 4

Temps d'attente des ascenseurs au niveau 3

FA DUO 1 & 2

Nombre de visiteurs au niveau 3 et 4

Run: 0 Paused | Time: 36.76 | Simulation: 31% | Memory: 39M of 247M | 28.4 sec



Project Example: Printemps Shopping Mall



Airport Model

This presentation is a part of
AnyLogic Standard Training Program



Airport Model

- Let's build a simulation model of passenger flows inside a small airport.
- Passengers arrive at the airport, check in at counters, pass the security check point and go to the waiting area. When boarding starts, passengers get on board.
- In the last phase we will simulate how an infection is spreading in the airport. From time to time infected people will enter the airport, and in case of long queues they will infect the passengers that will be standing near them for a reasonable time.



Airport Model. Phase 1

- In the first phase we will create a simple model where passengers arrive at the airport and move to the gate inside the building.

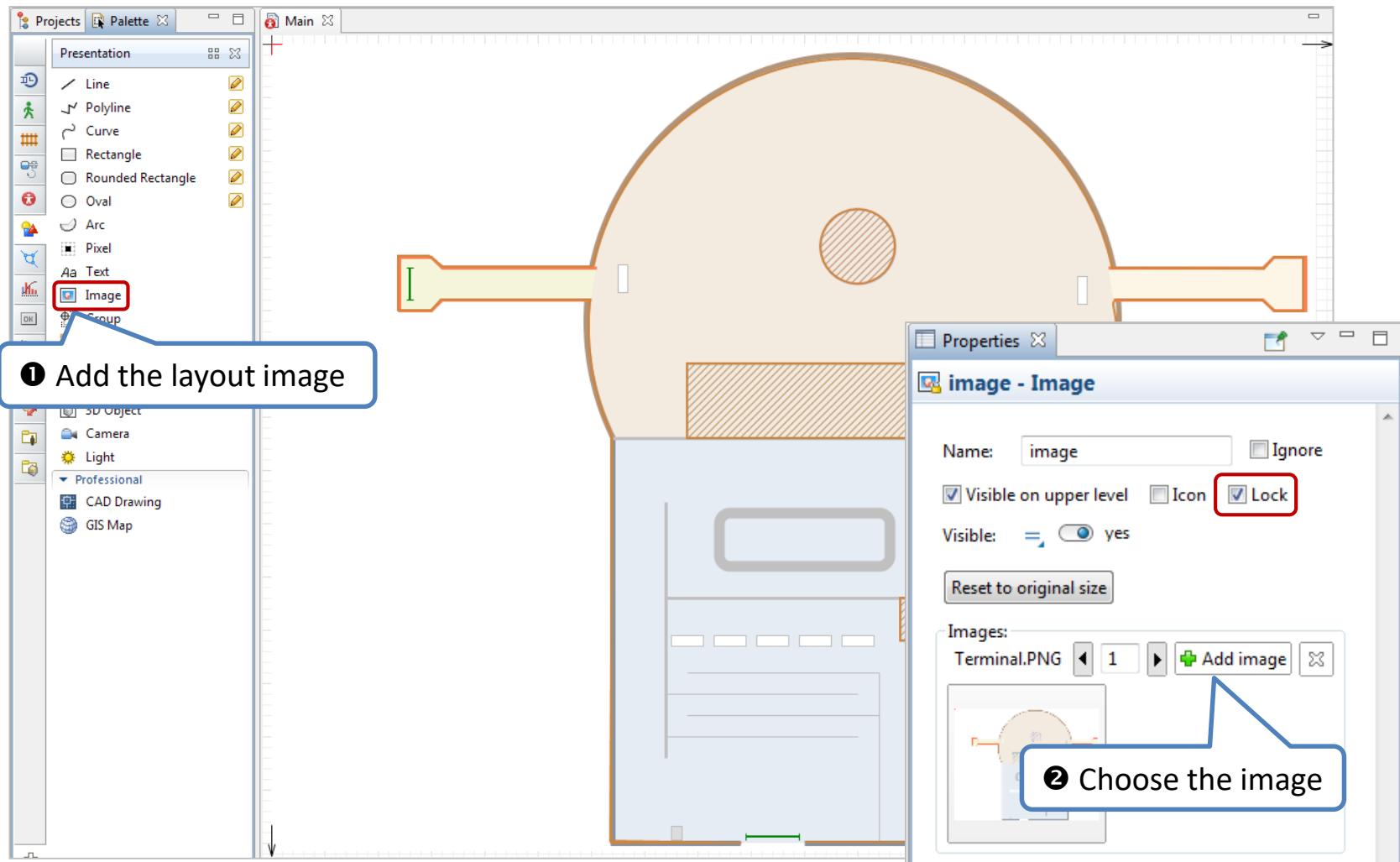


Pedestrian Library

- Traditional (discrete-event, queuing) modeling may give incorrect results in areas with dense pedestrian movement.
- AnyLogic **Pedestrian Library** enables simulating pedestrian flows in a “physical” environment. It allows you to create models of buildings (like subway stations, security checks etc.) or streets (large number of pedestrians).
- In models created with the Pedestrian Library, pedestrians move in continuous space, reacting to different kinds of obstacles (walls) and other pedestrians. You can accurately model the interaction of pedestrians and visualize pedestrian movement.
- Models enable you to collect statistics on pedestrian density in different areas, to ensure acceptable performance levels for service points with a hypothetical load, estimate lengths of stay in specific areas, or detect potential problems with interior geometry changes such as the effect of adding or removing obstacles.



Airport. Phase 1. Step 1



Create a new model. Name it *Airport*.

Typically you start creating pedestrian dynamics model by adding a layout of the simulated space and drawing walls over it. Superimposing Anylogic **Walls** over the walls in the image.

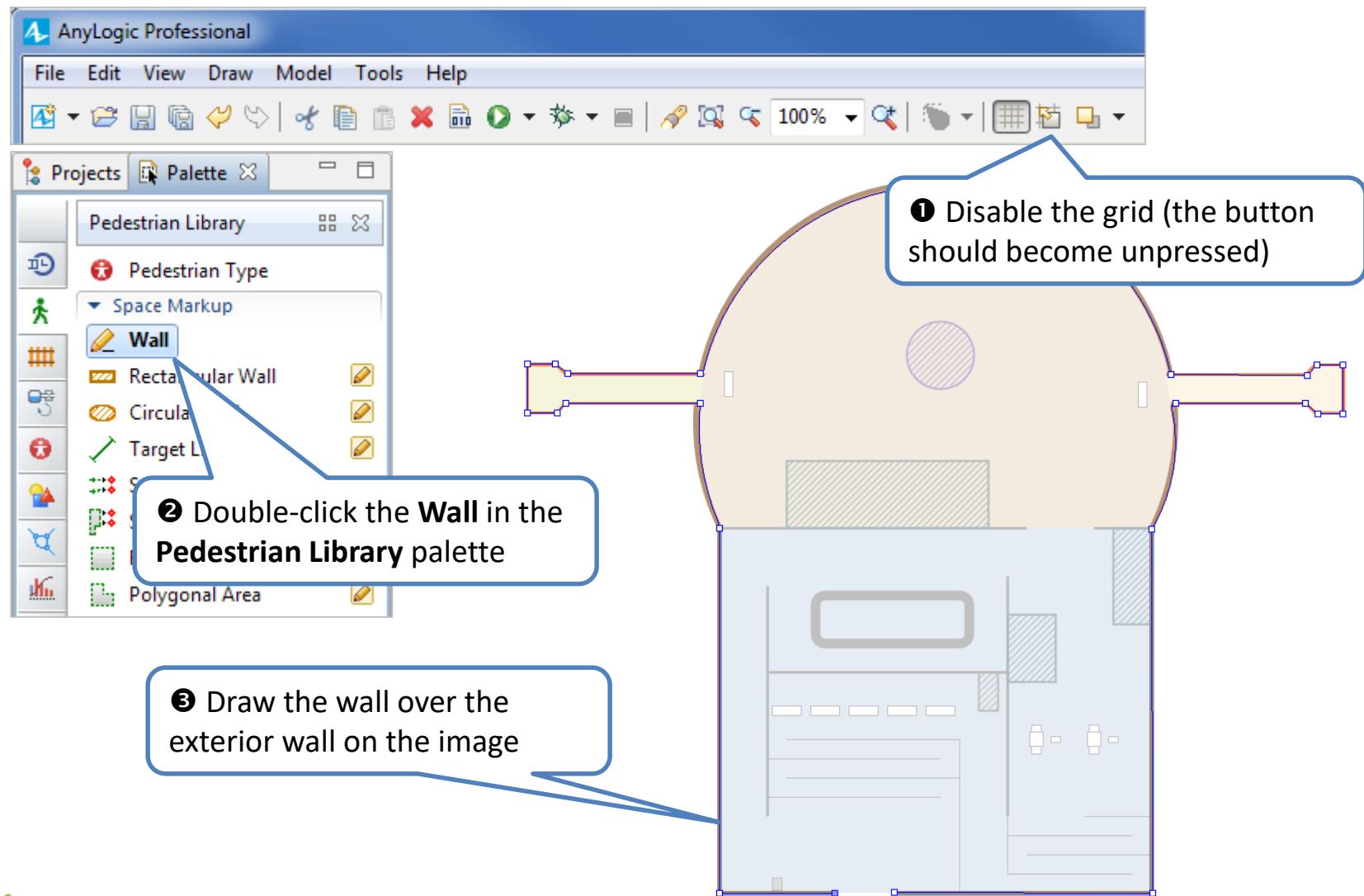
- ① Add *Image* from the **Presentation** palette.
 - ② Choose the image file to be displayed by this object: [terminal.png](#) from **Models\Airport** folder located on your USB disk.
Select the **Lock** checkbox to lock the image shape.
-

Locking presentation shapes

- You can lock a shape so it will not react on mouse click and it will be impossible to select it in the graphical editor until you unlock it. This is frequently needed when you have some background image on the presentation (e.g. a layout of a factory or a hospital department), used as a base for your animation. In this case when editing some shape laying over your layout you may accidentally edit the layout itself. Locking your background image will significantly simplify animation editing as you will not be able to select the layout with an inaccurate mouse click.



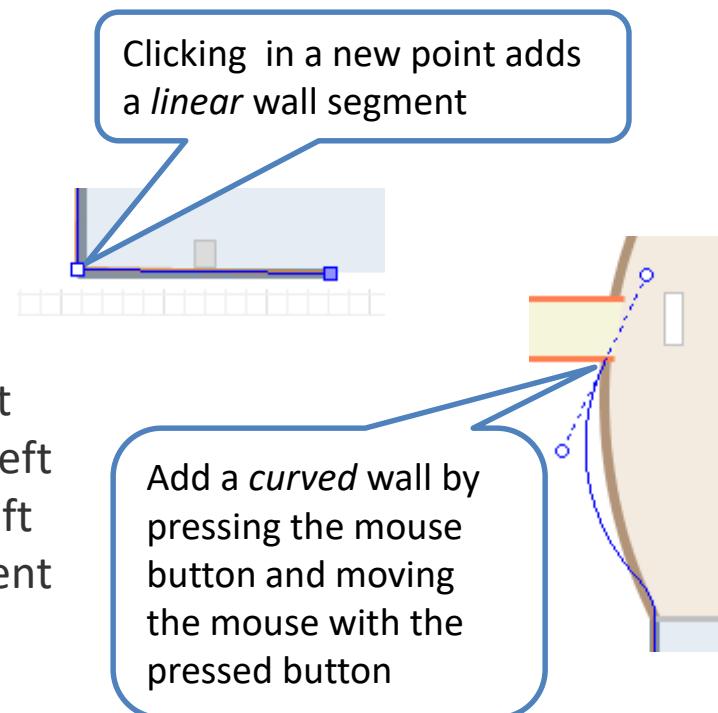
Airport. Phase 1. Step 2



Now let's draw the exterior wall using new space markup shape **Wall**.

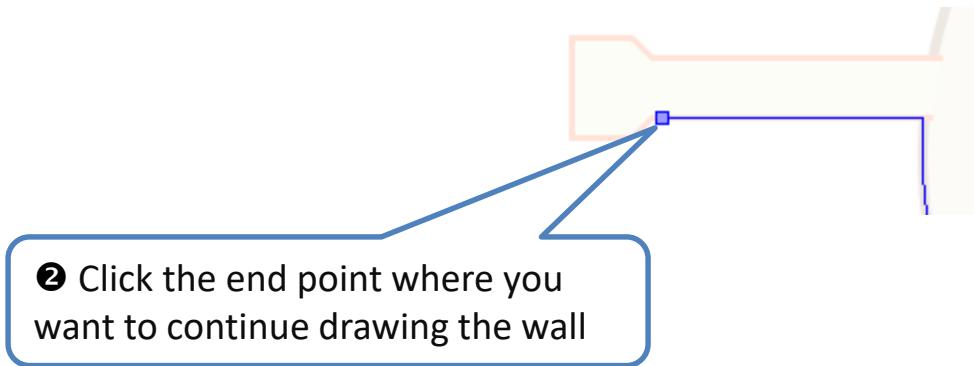
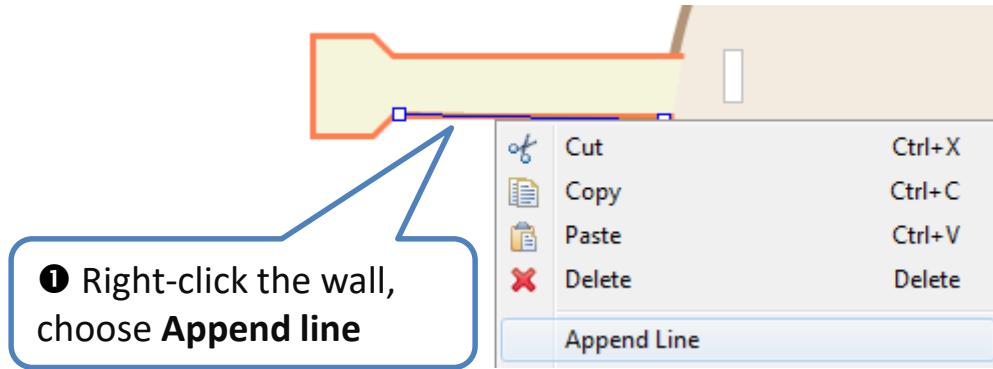
Drawing walls

- When drawing walls, use the drawing mode: first double-click the **Wall** in the palette and then draw the wall in the diagram, subsequently adding points with mouse clicks and putting the final point with a double click.
- To add a *linear* wall, just do the mouse click where this wall ends.
- To add a *curved* segment, press the mouse button in a point where the curved segment ends, and then move your mouse with the left mouse button being pressed. Release the left mouse button only when you got the segment of the required form.

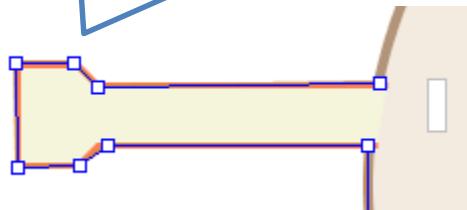


How to continue drawing the wall

- You may end drawing the wall for a while (with a double-click) and continue drawing it later following the scenario described below:

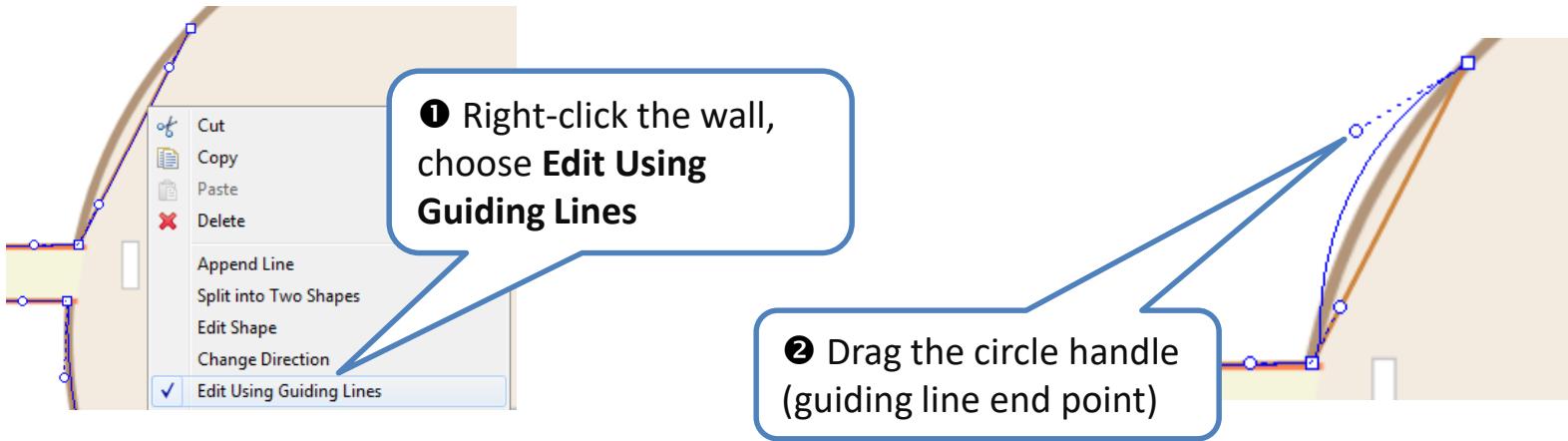


③ Continue drawing the wall from this point as usual



How to transform a linear wall to a curved (and vice versa)

Linear -> Curved



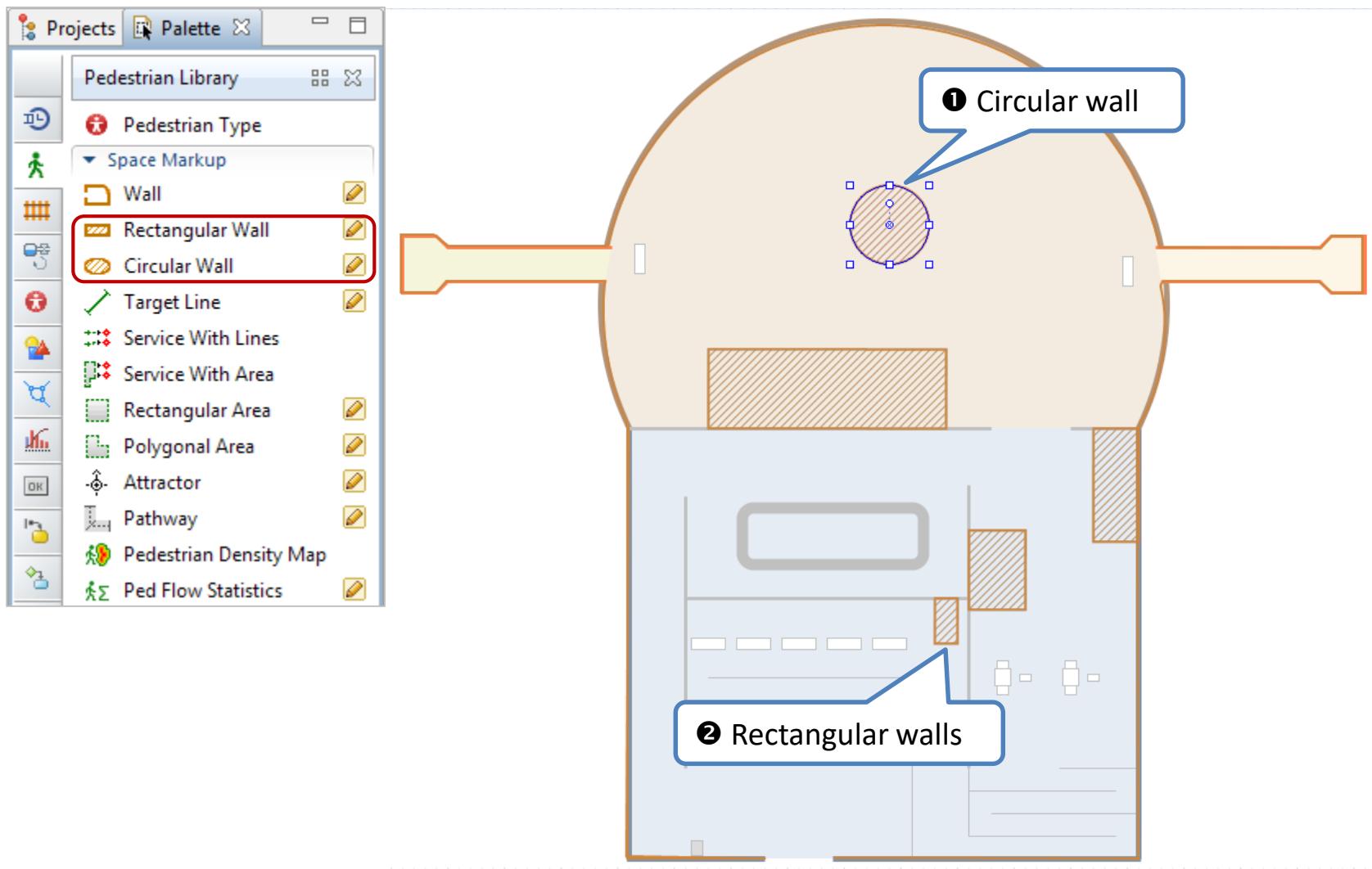
Curved -> Linear

① Select the wall &
Ctrl-drag the point

② The neighboring walls
will become linear



Airport. Phase 1. Step 3



Let's continue drawing the walls and obstacles in the airport building.

- ① Draw the column using the **Circular Wall**.
 - ② Draw the working spaces in the airport using the **Rectangular Wall**.
-

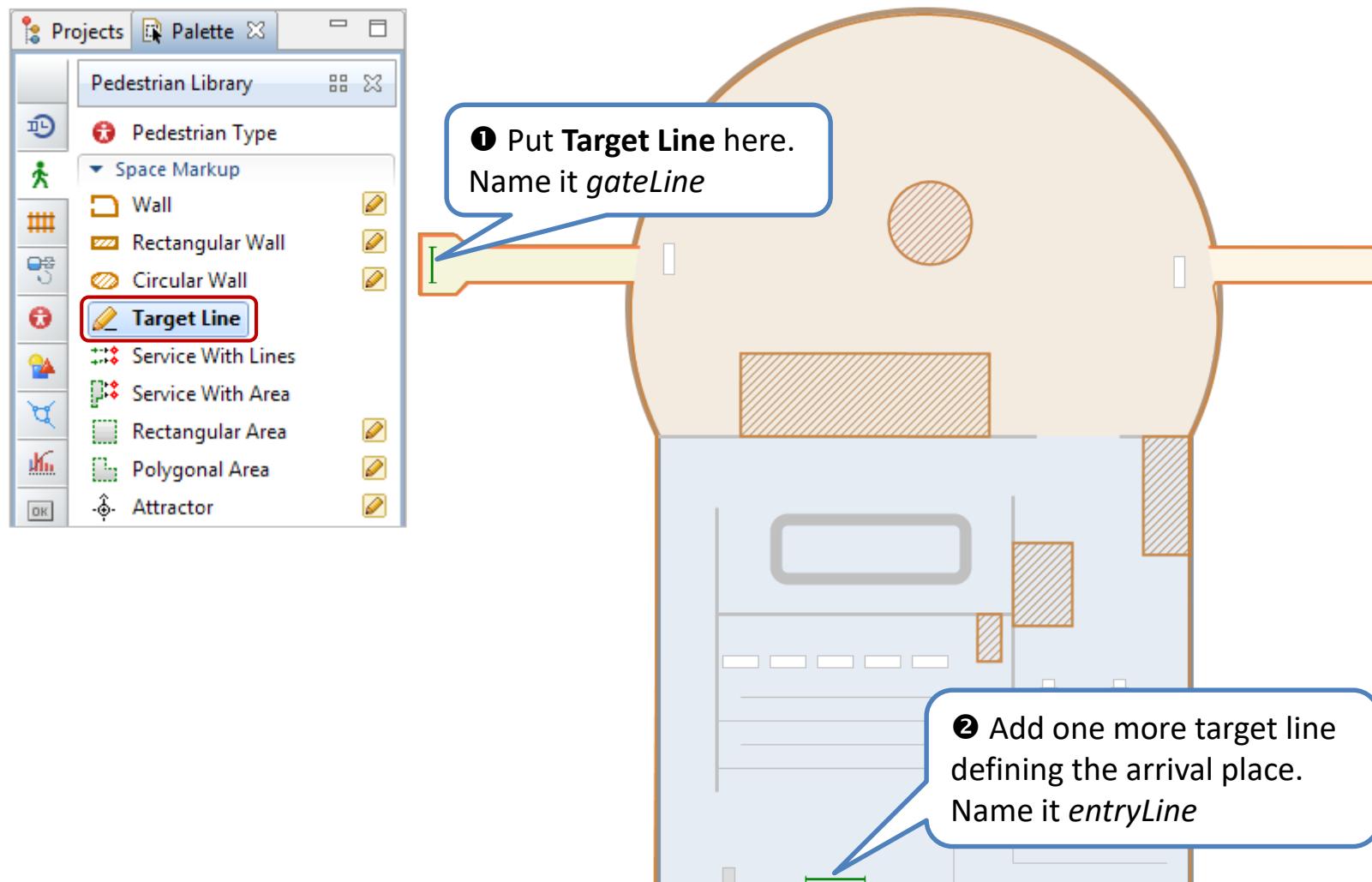
Walls

There are three space markup shapes to draw walls in pedestrian models:

- **Wall** - Use it to draw walls of complex forms (exterior wall, interior walls, guard rails, etc.)
- **Rectangular Wall** - Use it to draw rectangular areas non-accessible by pedestrians (e.g. working spaces).
- ▨ **Circular Wall** - Use circular walls to draw circular obstacles inside the simulated area (e.g. columns, fountains, skating rinks, etc.)



Airport. Phase 1. Step 4



- ① Draw the line to which passengers will move on entering the airport. Place it e.g. in the gate area. Name this line *gateLine*.
- ② Draw the airport entry with a line. Name it *entryLine*.

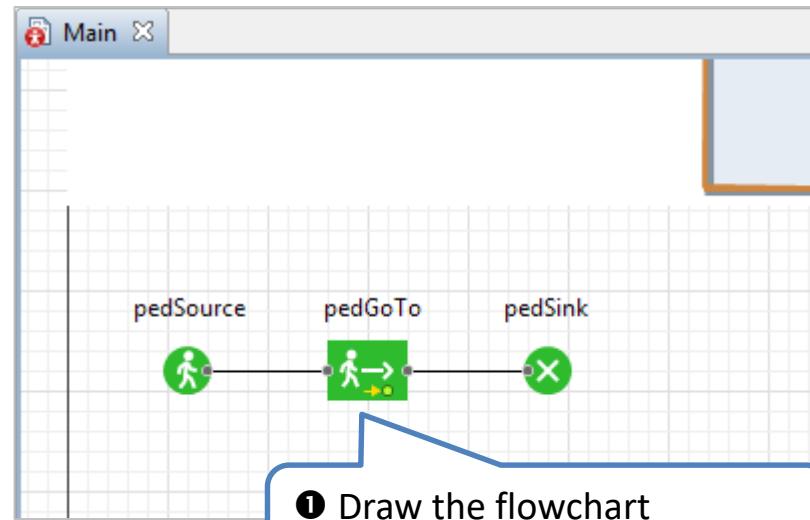
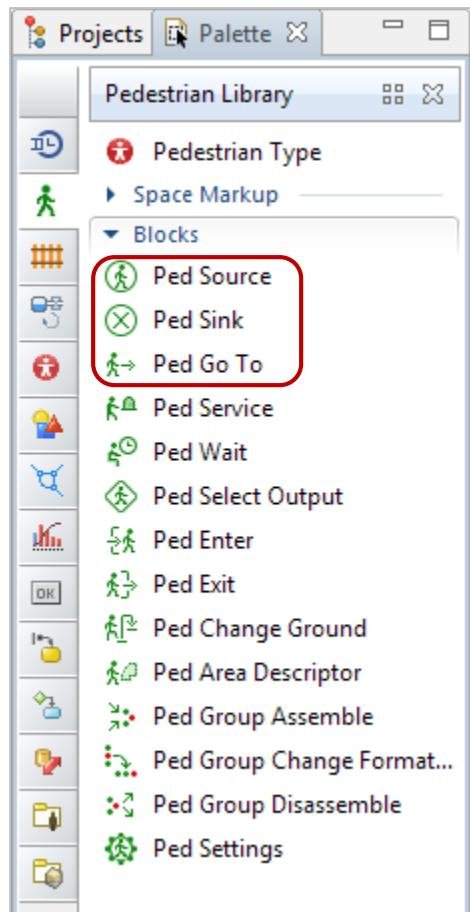
Please mind that either target lines, and all space markup shapes defining services, queues, etc. must be placed **inside** the walls so that they are reachable by pedestrians.

Target line

- **Target line** is used to graphically define the following things in pedestrian models:
 - Place where pedestrians appear in the simulated space (referred by **PedSource** or **PedEnter** flowchart block)
 - The destination of the pedestrian movement (referred by **PedGoTo** block)
 - The pedestrians waiting location (referred by **PedWait**)
 - When you model how pedestrians go from one floor to another you draw two target lines: one for the place where pedestrians leave the current floor and another one for the place where pedestrians enter new floor. In this case you specify both target lines in **PedChangeGround** flowchart block.



Airport. Phase 1. Step 5



- ① Create a flowchart for a simple process. Start with the pedestrian generator **PedSource**, then place the **PedGoTo** block that simulates how pedestrians move to the specified location and finally place **PedSink** – which discards incoming pedestrians. Each pedestrian flowchart typically starts with **PedSource** and ends with the **PedSink** block.

Defining processes in pedestrian models

- Processes in pedestrian dynamics models are defined using flowcharts, in the same way as in regular discrete-event models built with Process Modeling Library. Alike agents in discrete-event models, pedestrians pass through a flowchart, subsequently performing commands and operations defined by the blocks.



Airport. Phase 1. Step 6

The diagram illustrates a pedestrian flow model in AnyLogic. A green stick figure icon labeled "pedSource" is connected by a line to a green arrow icon labeled "pedGoTo". From "pedGoTo", a line continues to a red cross icon labeled "pedSink". A blue callout box labeled ① *PedSource* contains the text: Target line: *entryLine* and Arrival rate: 100 per hour. Another blue callout box labeled ② *PedGoTo* contains the text: Target line: *gateLine*.

Main

pedSource pedGoTo pedSink

Properties

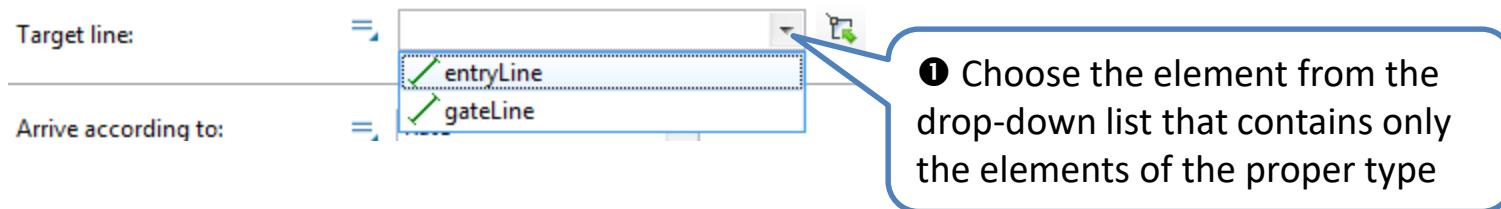
pedSource - PedSource

Name:	pedSource	<input checked="" type="checkbox"/> Show name	<input type="checkbox"/> Ignore
Appears at:	<input checked="" type="radio"/> line	<input type="radio"/> point (x,y)	<input type="radio"/> area
Target line:	<input checked="" type="radio"/> entryLine <input type="button" value="..."/> <input type="button" value="x"/> <input type="button" value="copy"/>		
Arrive according to:	<input checked="" type="radio"/> entryLine <input type="button" value="..."/> <input type="button" value="x"/> <input type="button" value="copy"/>		
Arrival rate:	100	per hour	<input type="checkbox"/>
Limited number of arrivals:	<input type="checkbox"/>		

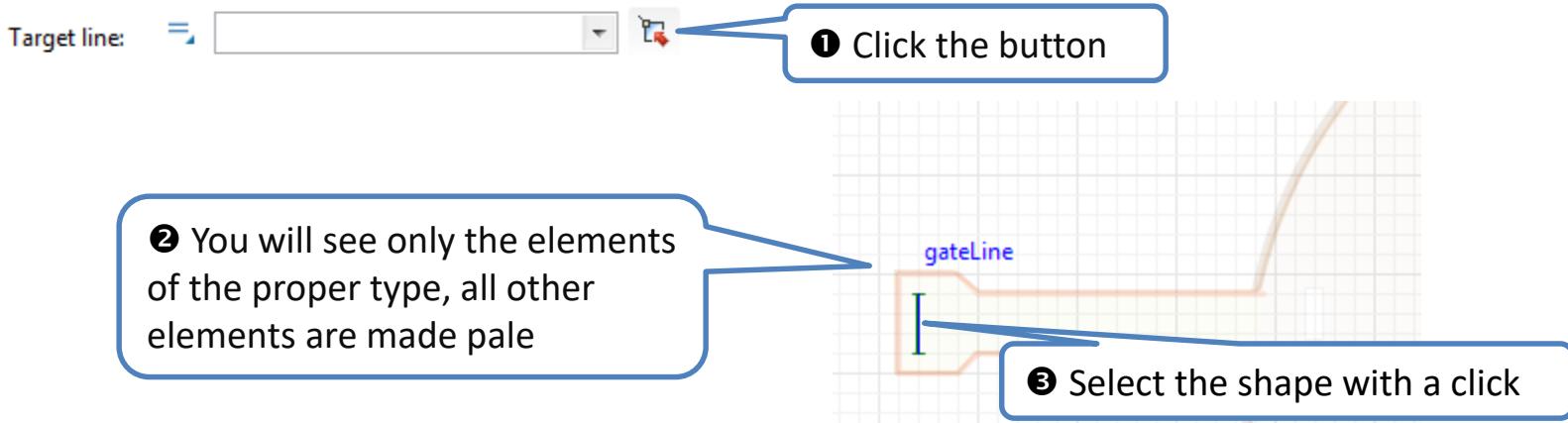


- ① We tell the *pedSource* block to generate in average 100 passengers per hour and put them on the target line *entryLine*.
- ② *pedGoTo* block simulates how the passengers move to the gate (the destination is defined with *gateLine*).

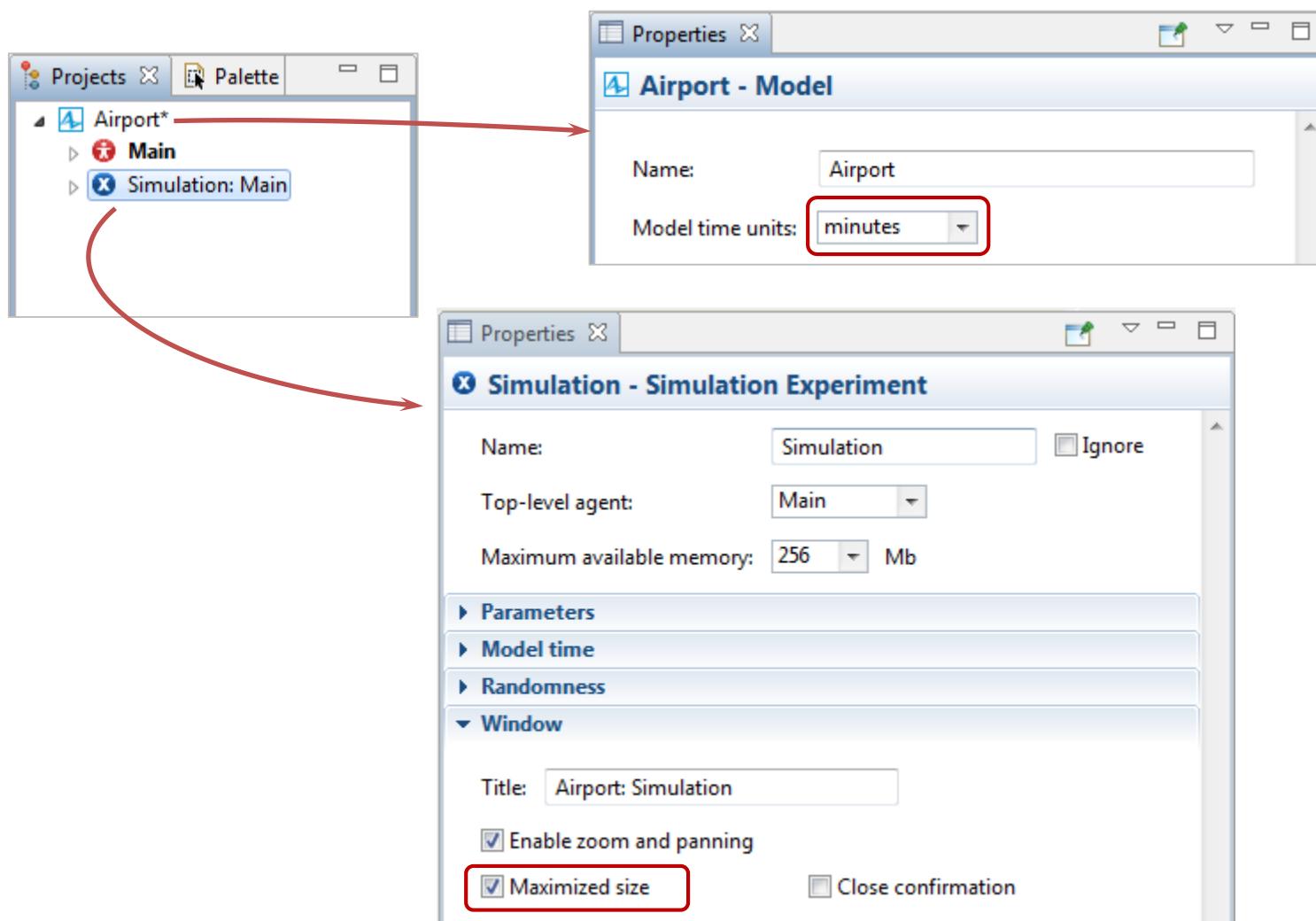
Choosing the element from the drop-down list



Selecting the element in the graphical editor



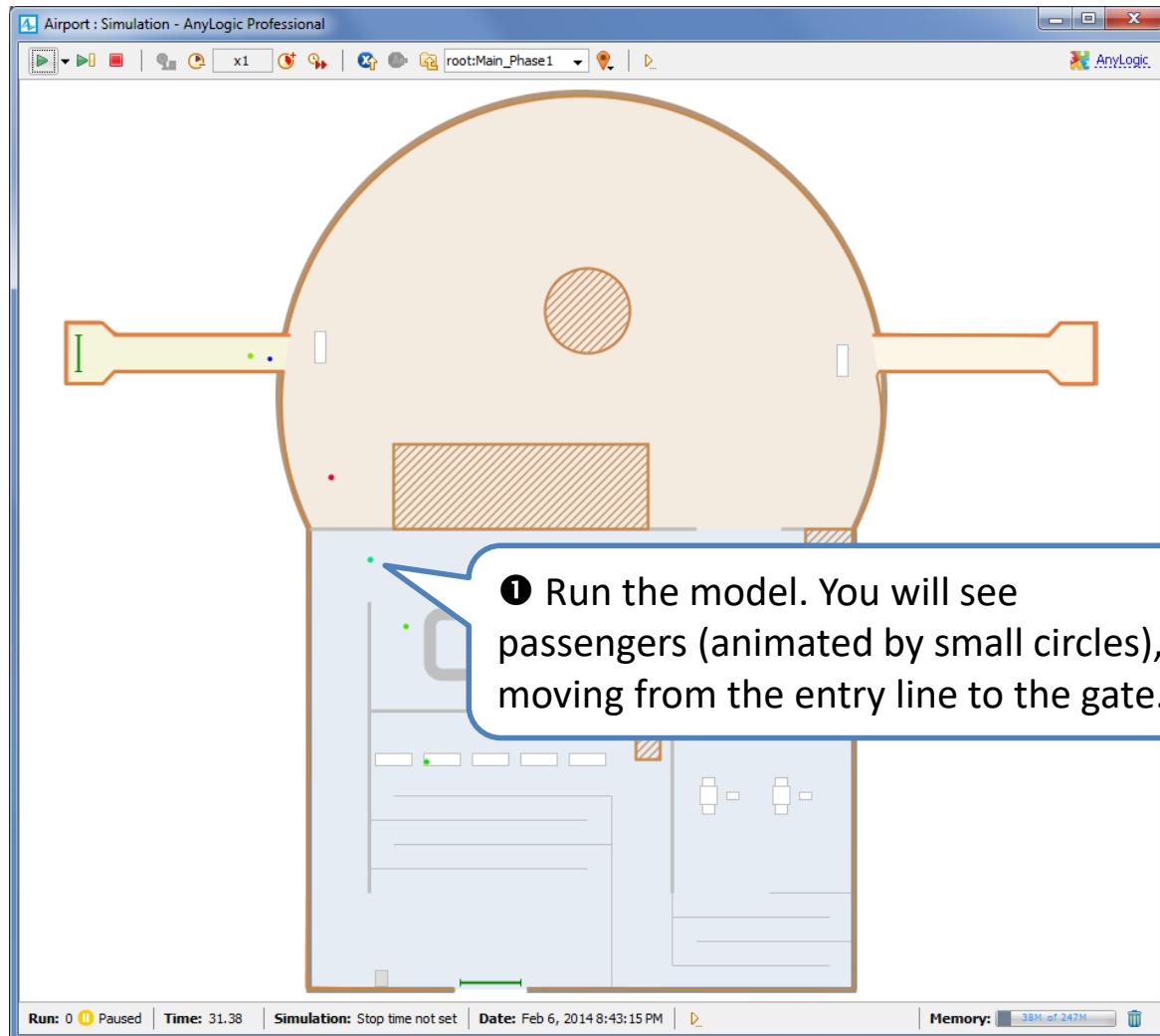
Airport. Phase 1. Step 7



- ① Choose *minutes* as the model time units (in the properties of the model).
- ② Set the model window to be of maximum size, choose the **Maximized size** option.



Airport. Phase 1. Step 8



Airport Model. Phase 2

- In this phase we will add check-in counters.
- We will also add 3D animation.



Airport. Phase 2. Step 1

The screenshot shows the AnyLogic modeling environment. On the left, the Projects palette lists a 'Pedestrian Library' containing a 'Pedestrian Type' and various 'Space Markup' elements like 'Wall', 'Rectangular Wall', 'Circular Wall', and 'Target Line'. The Main diagram window shows a green circle labeled 'pedSource' connected by a line to a green rectangle labeled 'pedGoTo'. A blue arrow points from the 'Pedestrian Type' icon in the palette to the 'Pedestrian' node in the diagram. To the right, a 'New agent' dialog box is open, titled 'Step 1. Creating new agent type'. It contains a text field 'Agent type name:' with 'Passenger' typed in, which is highlighted with a red box. Below it are two radio button options: '(radio) Create the agent type "from scratch"' and '(radio) Use database table'. The 'Use database table' option has a note: 'I have agent data stored in a database'. A red arrow points from the 'Main' tab of the diagram window to the 'Main' tab of the 'New agent' dialog. Another red arrow points from the 'Step 1' dialog to the 'Step 2' dialog below it. The 'Step 2. Agent animation' dialog shows a list of people types: 'Person' (highlighted with a red box), 'Office Worker', 'Worker', 'Doctor', and 'Nurse'. To the right is a preview area showing a 2D walking person icon.

① Drag Pedestrian Type

You will see the diagram of the created pedestrian type with 3D animation shape in the axis origin. Open *Main* diagram to continue developing the model.



① Create a new pedestrian type. We need it to define a custom animation for our pedestrian (in our case – 3D shape instead of the default circles that will appear in 3D just as many-colored cylinders).

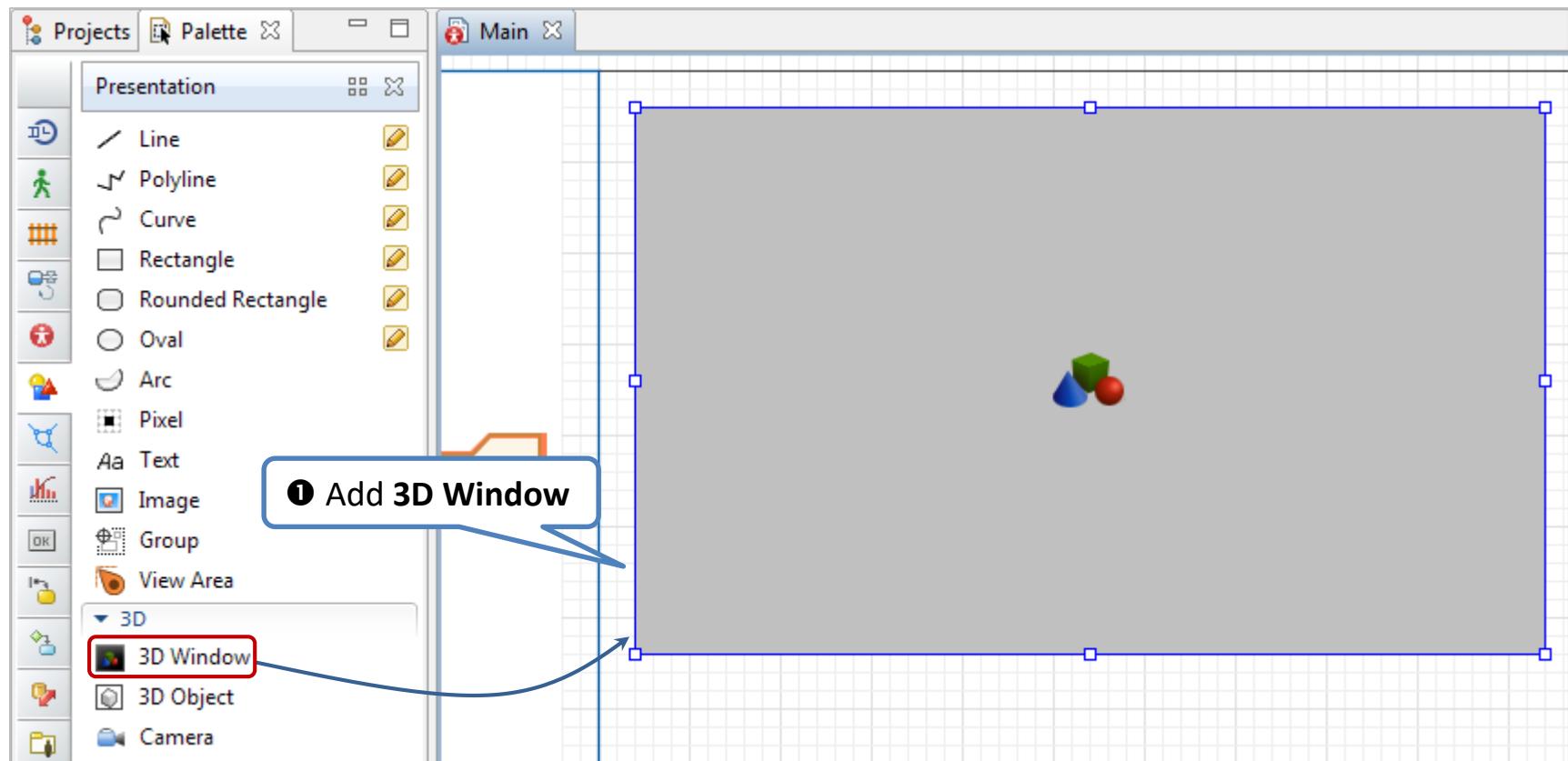
Name the pedestrian type *Passenger* and leave the default animation choice: 3D figure *Person*.

Creating custom pedestrian types

- If you need to define some custom animation, add some attributes or behavior to pedestrians, you need to create a custom pedestrian type.
- Since pedestrians are agents and support all the agent functionality, you can add any model elements on the graphical diagram of the pedestrian alike you do with agents: define behavior with a statechart, add variables and parameters, define some continuous behavior with system dynamics stock-and-flow diagram, schedule events, etc.



Airport. Phase 2. Step 2



Airport. Phase 2. Step 3

The image shows the AnyLogic simulation environment. On the left is the Main workspace, displaying a flow diagram with three nodes: 'pedSource' (green person icon), 'pedGoTo' (green person with arrow icon), and 'pedSink' (green circle with X). Arrows connect them in sequence. On the right is the Properties window for the 'pedSource' component.

Properties - pedSource - PedSource

- Name: pedSource Show name
- Ignore:
- Appears at: line point (x,y) area
- Target line: entryLine
- Arrive according to: Rate
- Arrival rate: 100 per hour
- Limited number of arrivals:

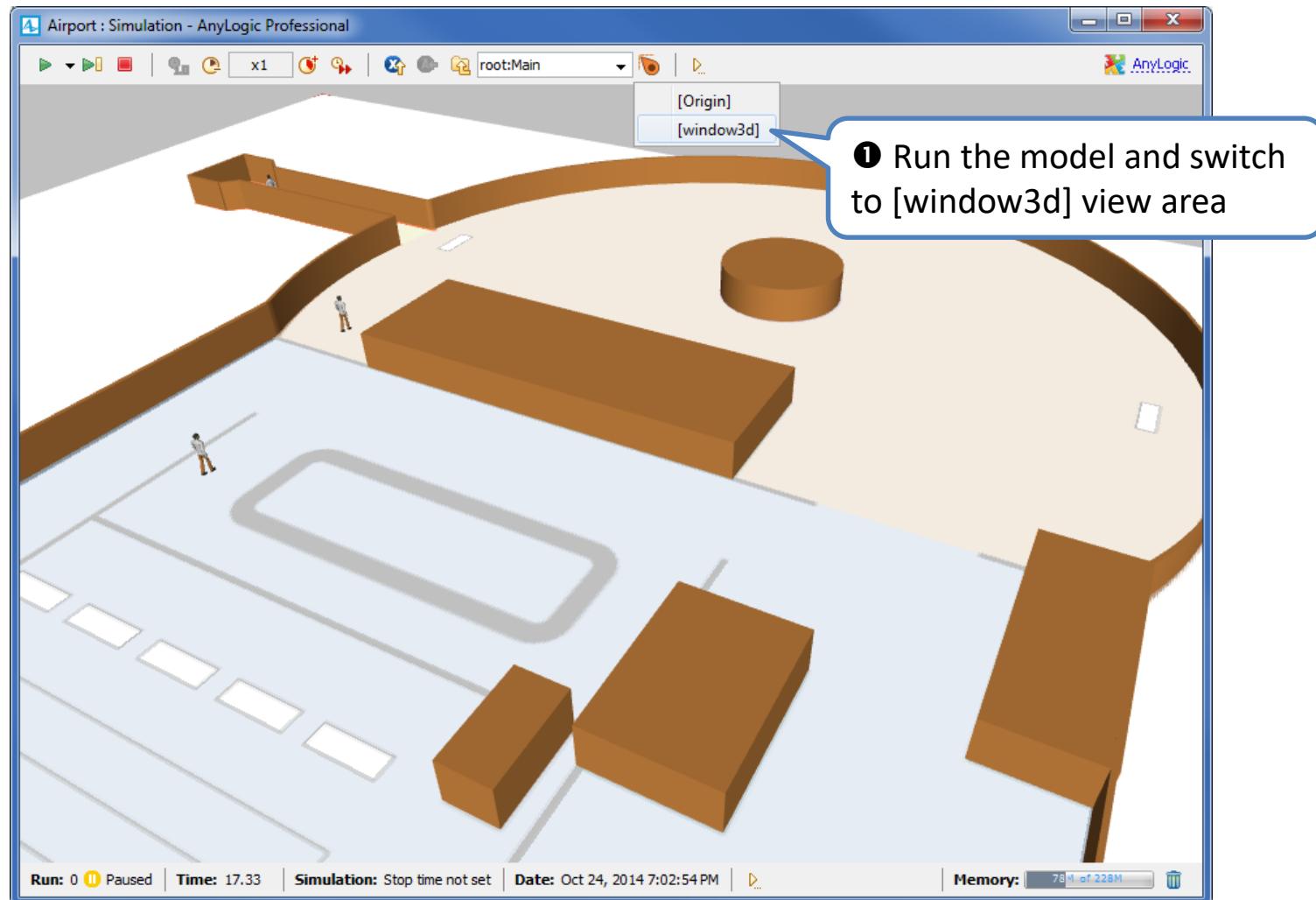
Pedestrian

- New pedestrian: Passenger
- Comfortable speed: uniform(0.5, 1) meters/second
- Initial speed:
- Diameter:

① Choose *Passenger* as the type of pedestrians generated by *pedSource*.



Airport. Phase 2. Step 4



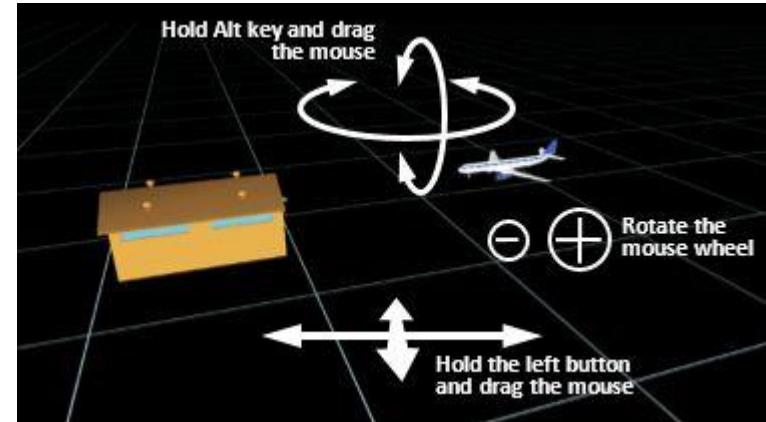
To be able to see 3D animation of the simulated process, we added **3D Window** from the **Presentation** palette.

- ① Now we can run the model and see 3D animation. Navigate to 3D animation scene using the automatically created view area [window3d].

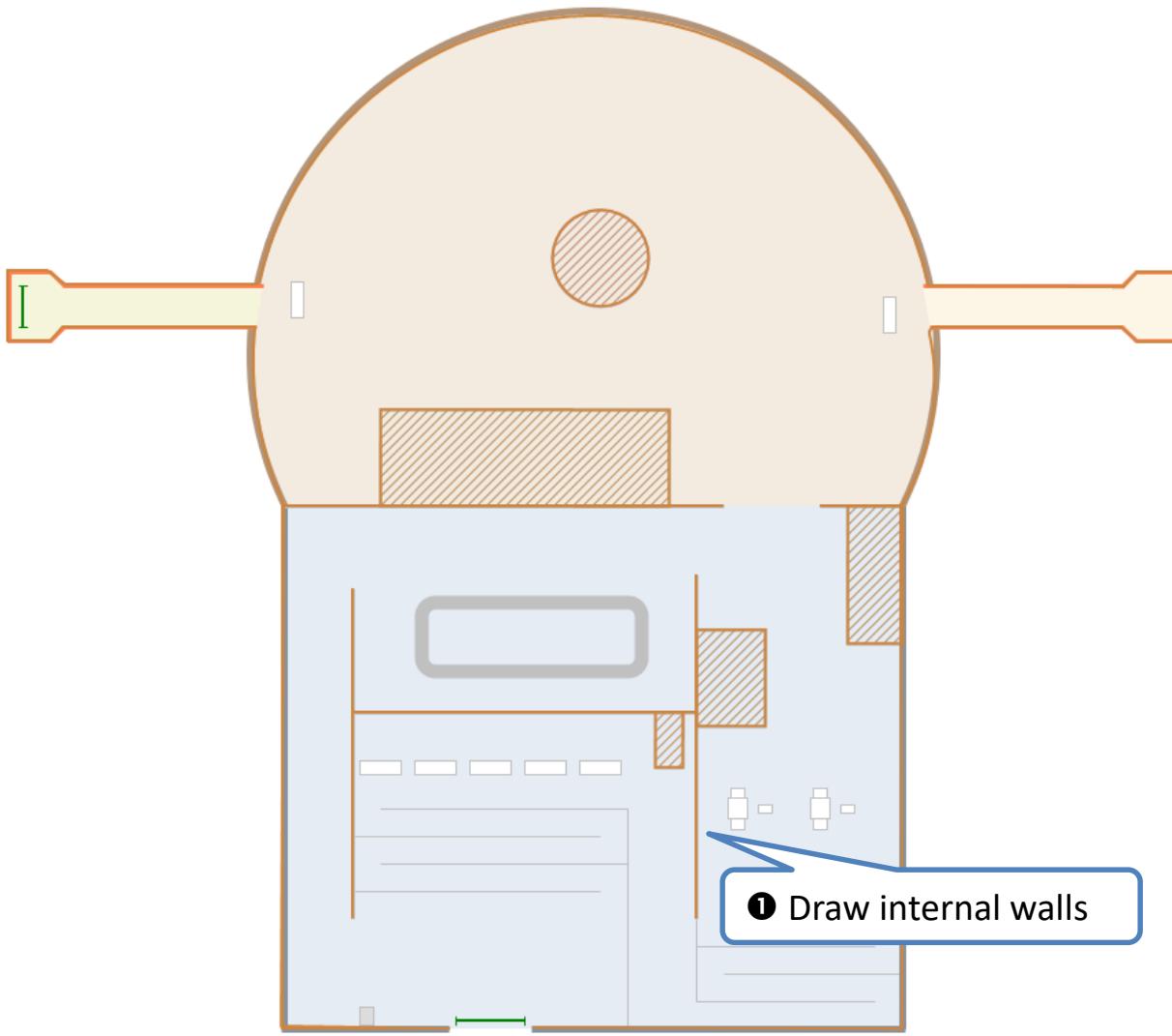
You will see pedestrians moving from the entry to the gate inside the airport building.

Navigation in 3D scene

- **Drag the mouse** to move the camera right, left, forward or backward on the same height.
- **Rotate the mouse wheel** to move the camera closer or further from the current center of the scene.
- **Hold Alt (Mac OS: Option) + drag the mouse** to rotate the scene relative to the camera.



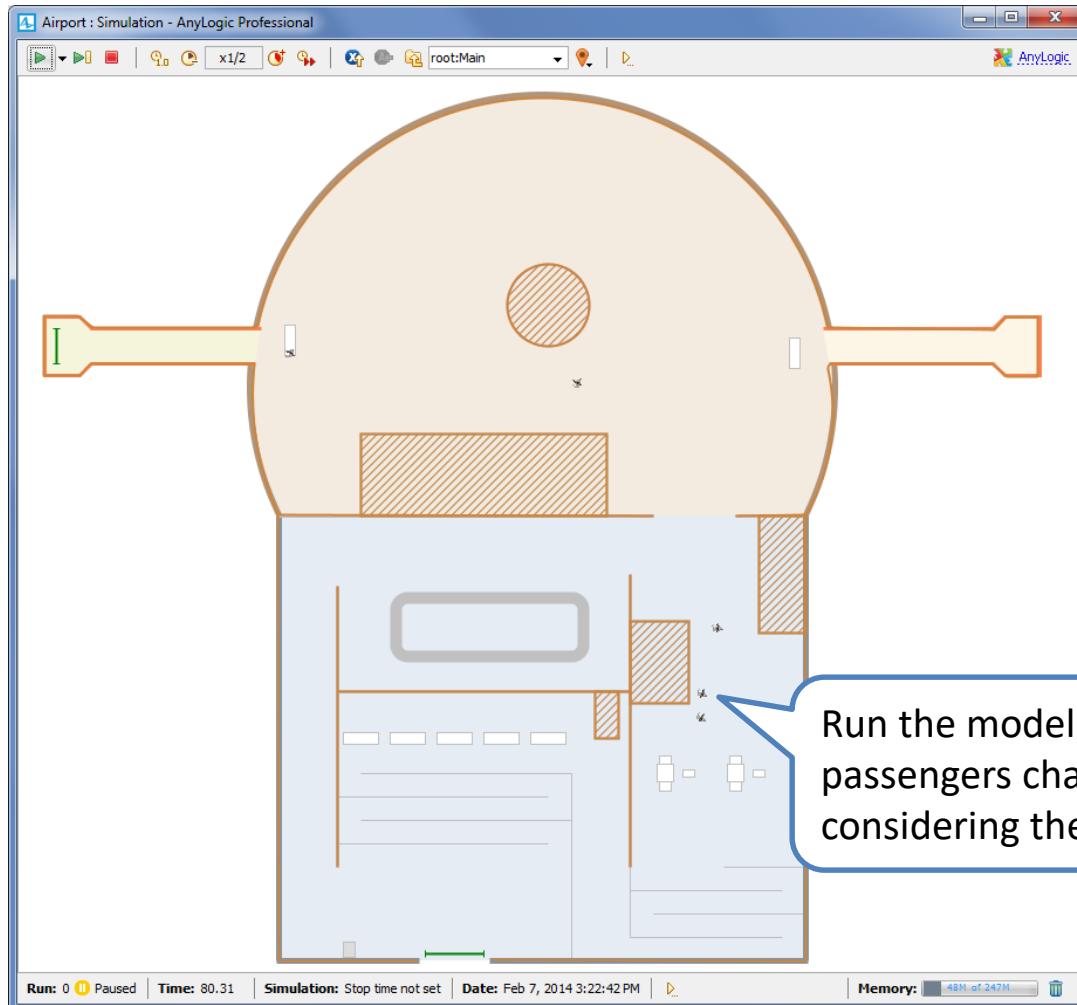
Airport. Phase 2. Step 5



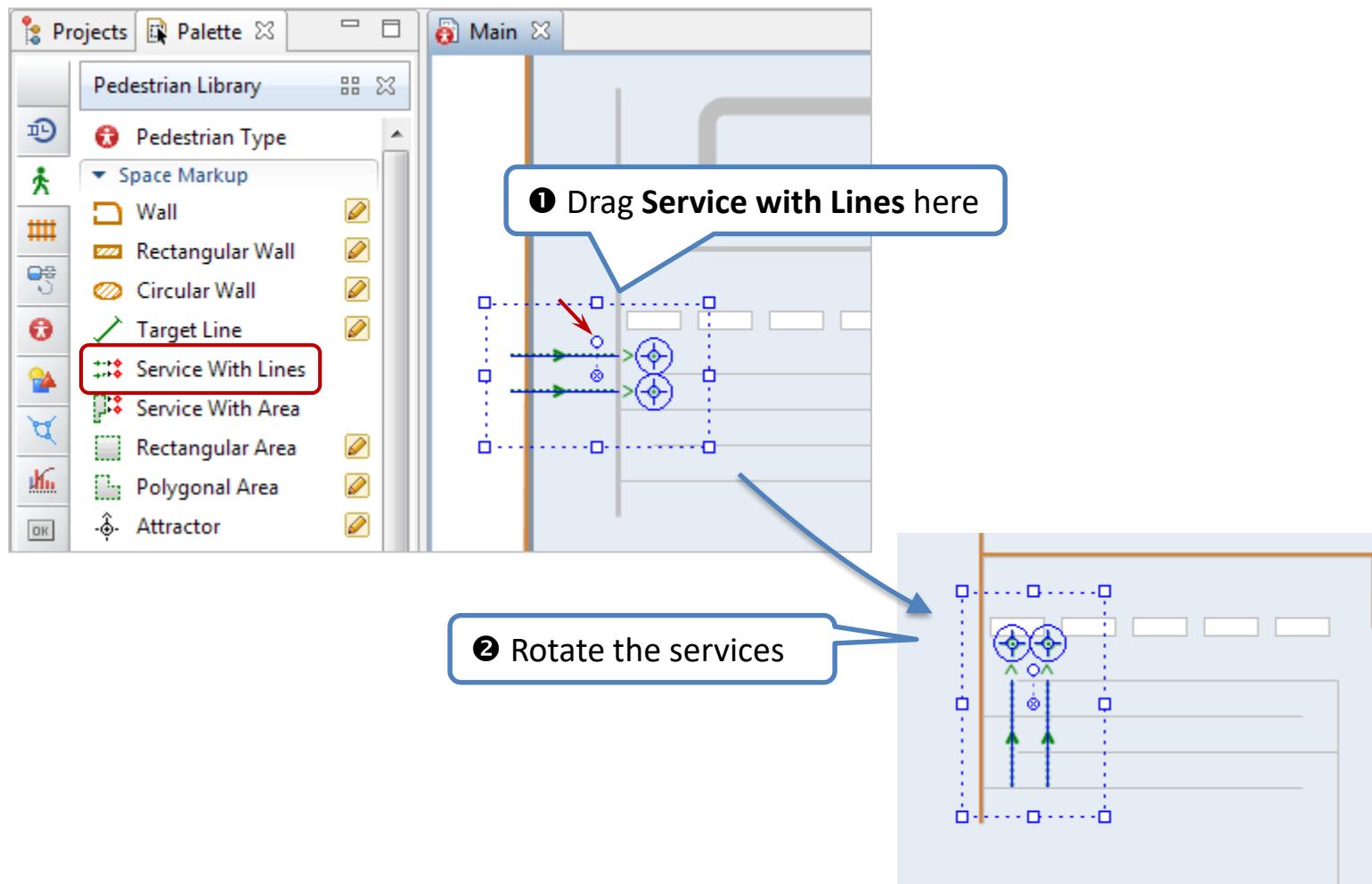
① Draw internal walls



Let's draw internal walls. Use the same space markup shape **Wall** as before.
Please check that there is no gap between walls.



Airport. Phase 2. Step 6



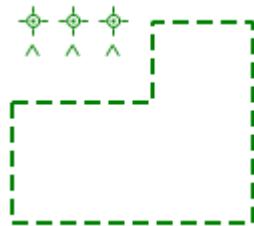
Now we can start modeling processes going on in the airport. Let's simulate the check-in counter first. It is actually a service – passengers should get serviced there and in case the service is in use by some other passenger they wait in a queue until it becomes free.

Services in pedestrian models

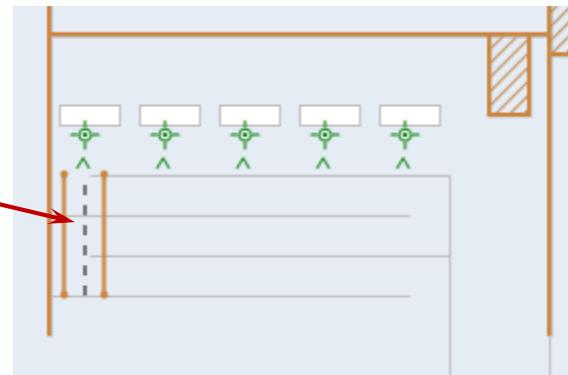
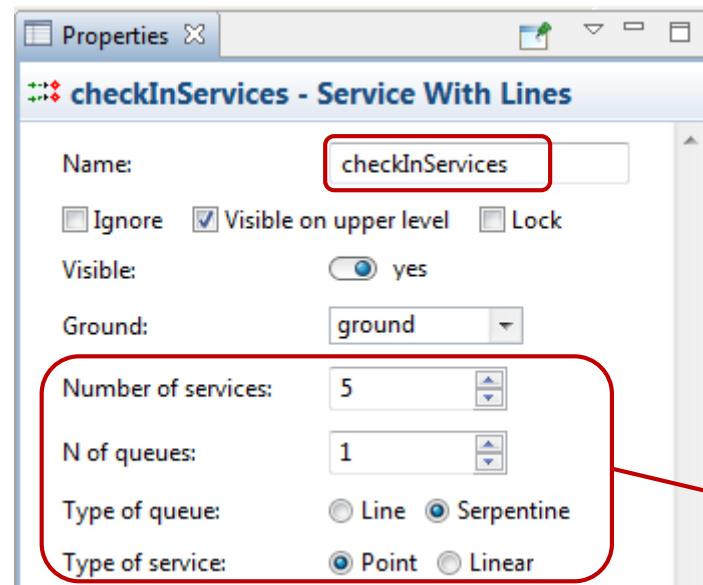
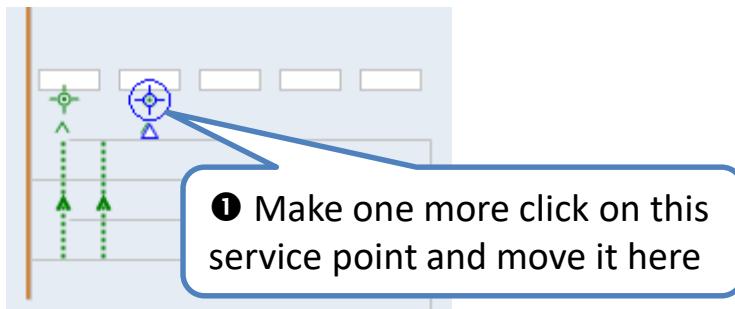
- Service with Lines - Used to define service(s) with queue lines (e.g. turnstiles, cash desk, passport control checkpoint.)



- Service with Area - Used to define service(s) with electronic queue, when pedestrians wait in the hall /office until the service becomes available (e.g. ticket office, bank office, info point.)

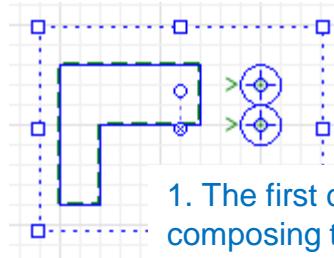


Airport. Phase 2. Step 7

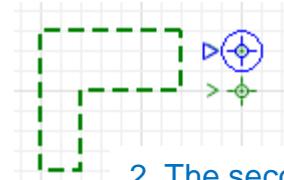


Working with complex space markup elements

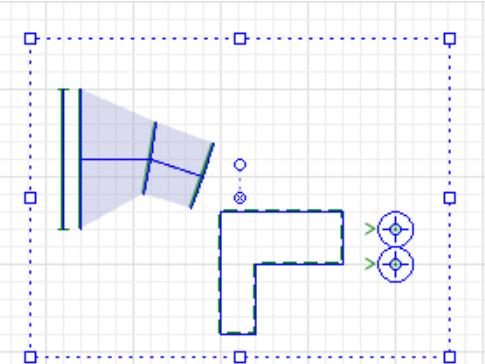
- Since some space markup shapes may consist of several primitive markup elements, and also all pedestrian markup shapes make up ground(s), it is important to understand how you select markup shapes to be able to edit them graphically in an easy way.



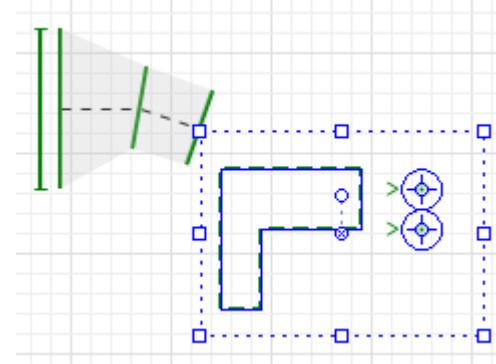
1. The first click selects all the shapes composing the complex element



2. The second click selects the shape you have clicked



3. One more click on the currently selected shape selects all the shapes of this ground

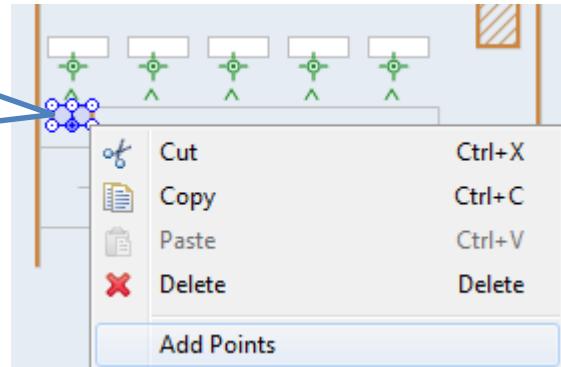


4. Clicking another shape selects it and starts the selection process from the beginning

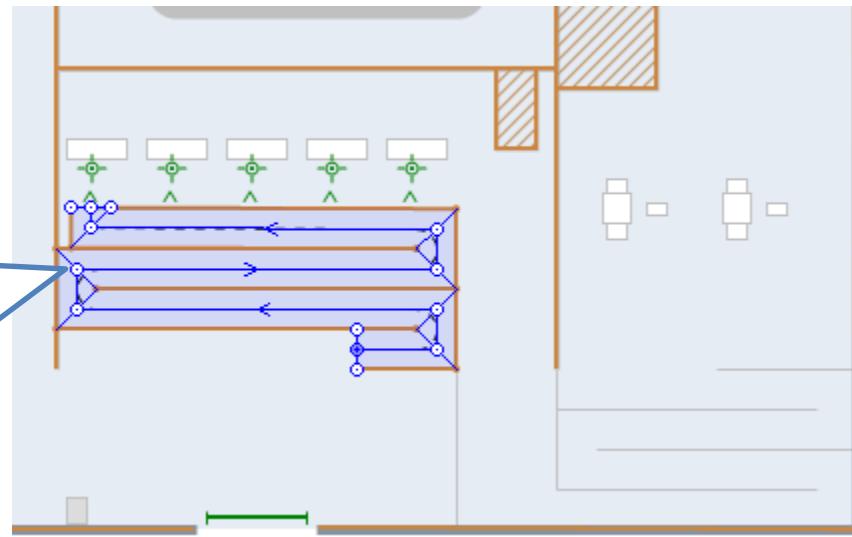


Airport. Phase 2. Step 8

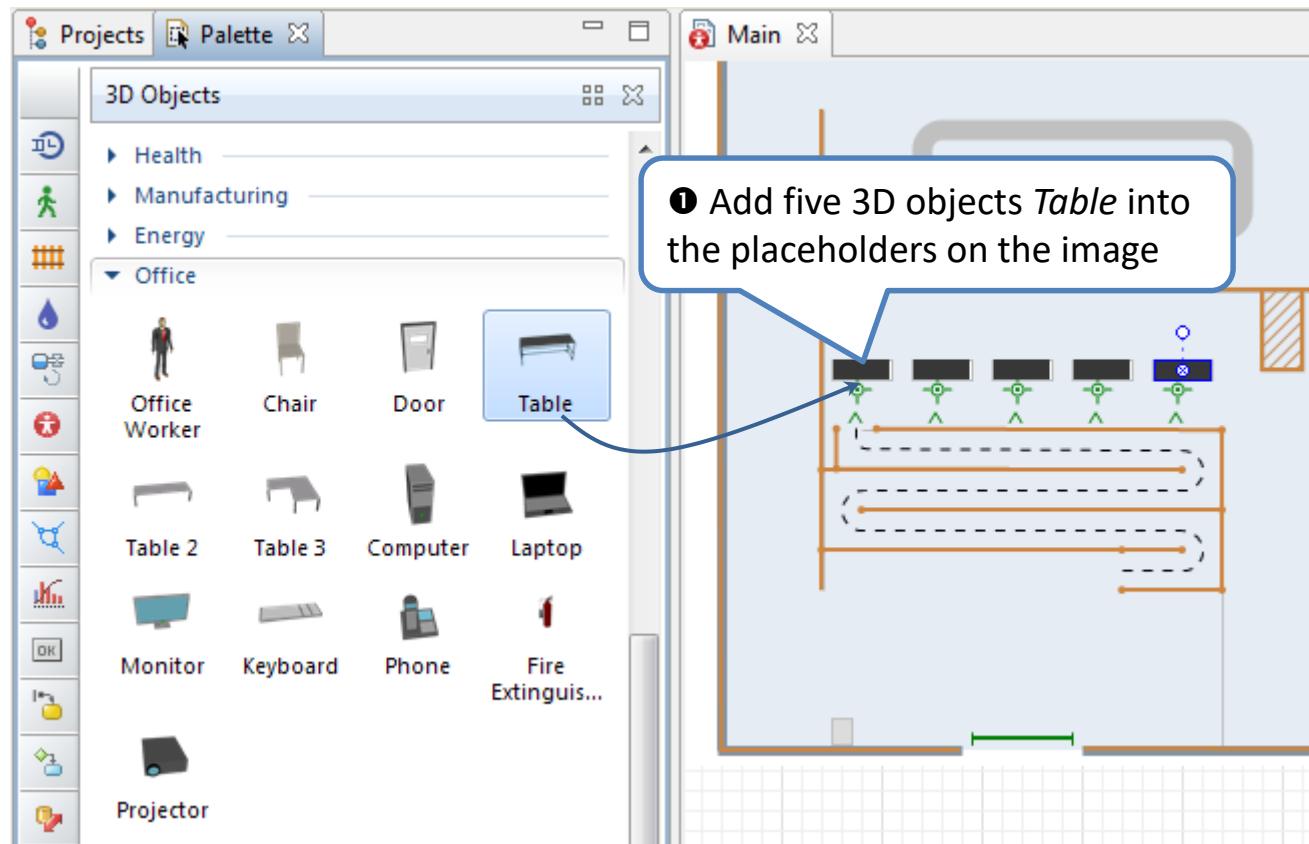
- ① Reduce the queue shape, right-click it and choose **Add points**



- ② Draw the queue by putting its turning points with mouse clicks. Place the points in the middle of the passage. Put the end point with a double click.



Airport. Phase 2. Step 9



Airport. Phase 2. Step 10

The screenshot shows the AnyLogic software interface with three main windows:

- Projects** window: Shows the "Pedestrian Library" with various blocks listed, including "Ped Source", "Ped Sink", "Ped Go To", and "Ped Service". The "Ped Service" block is highlighted with a red border.
- Main** window: Displays a pedestrian flow diagram. A green circle labeled "pedSource" is connected to a green square labeled "checkIn". This is followed by a green square labeled "pedGoTo" and a green circle labeled "pedSink". A callout box labeled "PedService" points to the "checkIn" block.
- Properties** window: Opened for the "checkIn" block. It shows the following settings:
 - Name: checkIn
 - Show name: checked
 - Ignore: unchecked
 - Services: checkInServices (highlighted with a red border)
 - Queue choice policy: Shortest queue
 - Delay time: uniform(1, 4) minutes

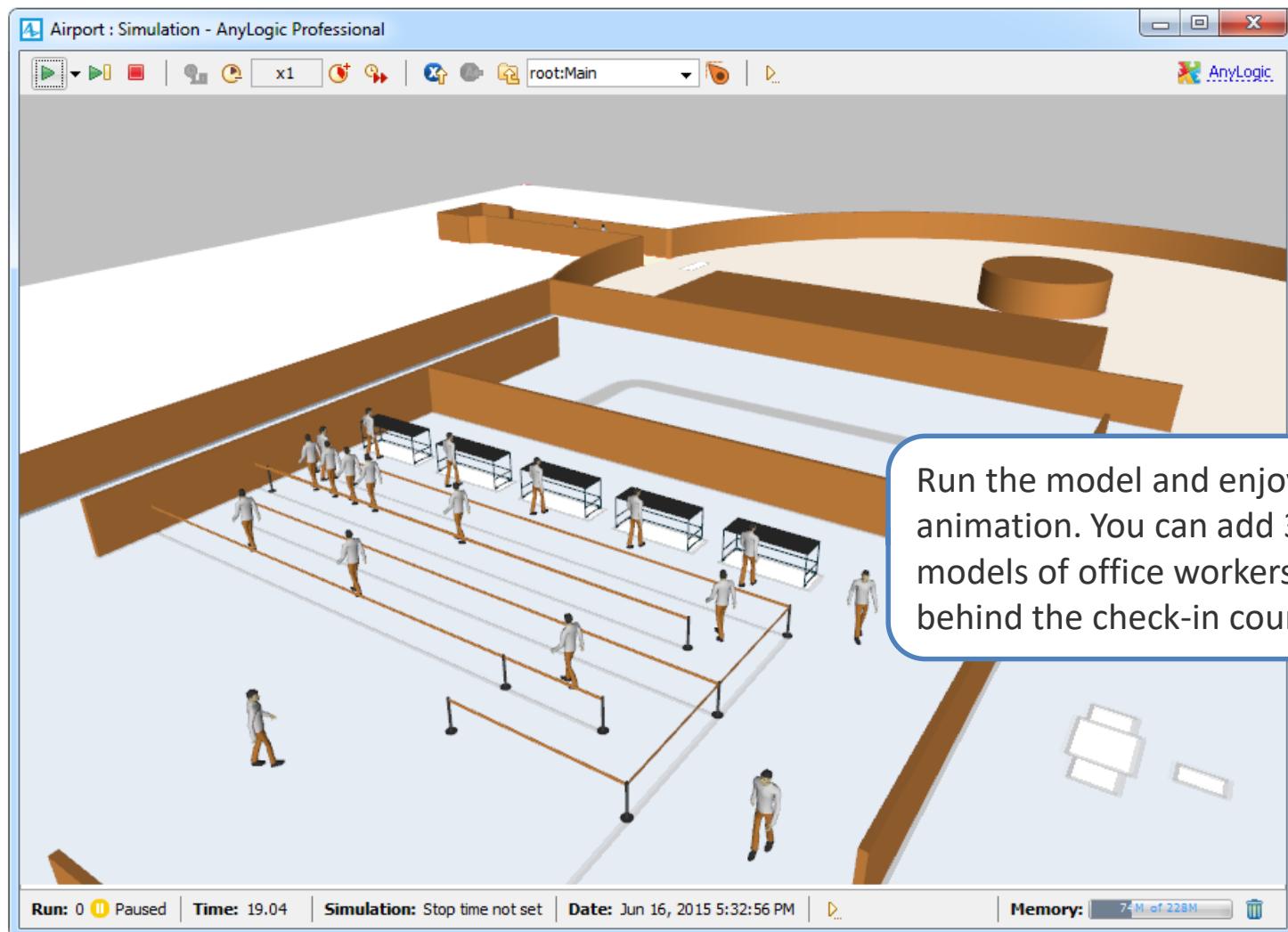


Now let's add flowchart block simulating security check point services. Insert **PedService** in the flowchart to make pedestrians pass through the service defined via the referenced services shape.

Connect the block as shown on the slide above.



Airport. Phase 2. Step 11

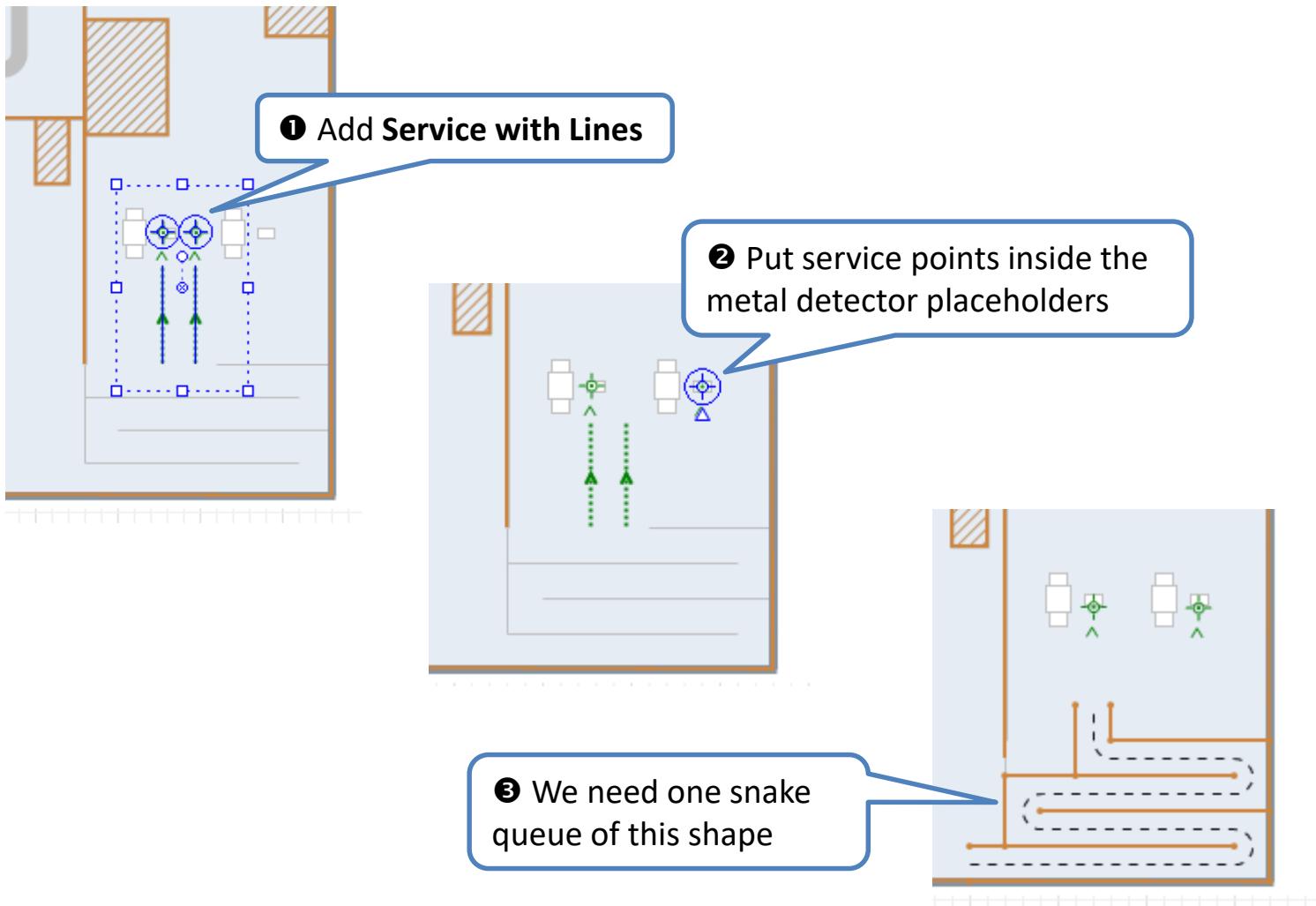


Airport Model. Phase 3

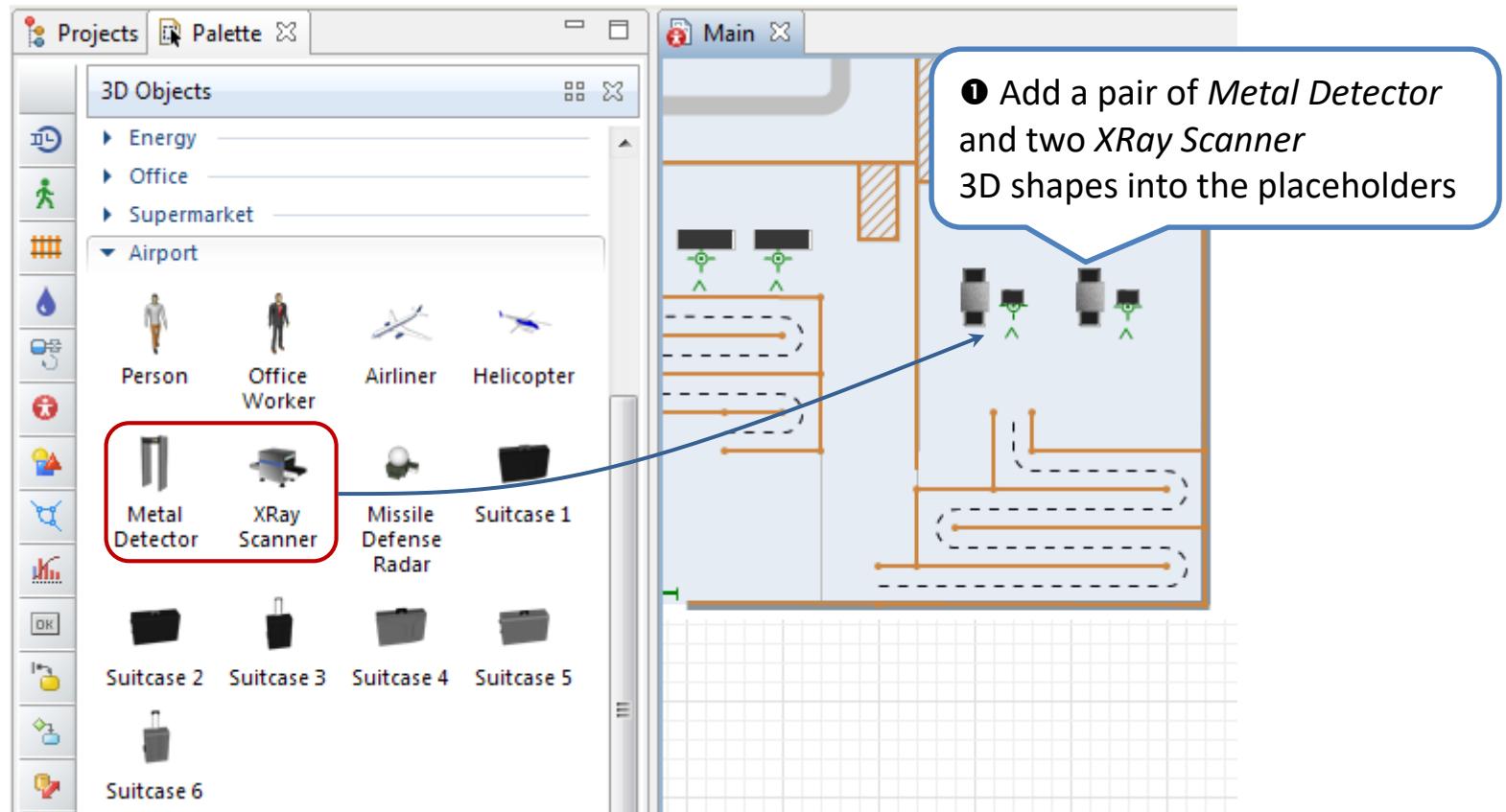
- In this phase we will add security check points.
- Having passed security check points, passengers wait in a waiting area and after some time go on board.



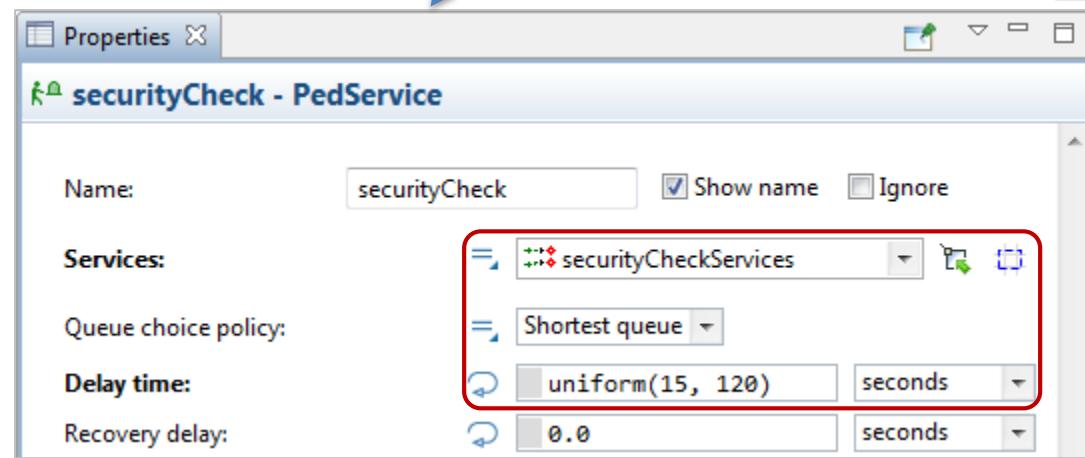
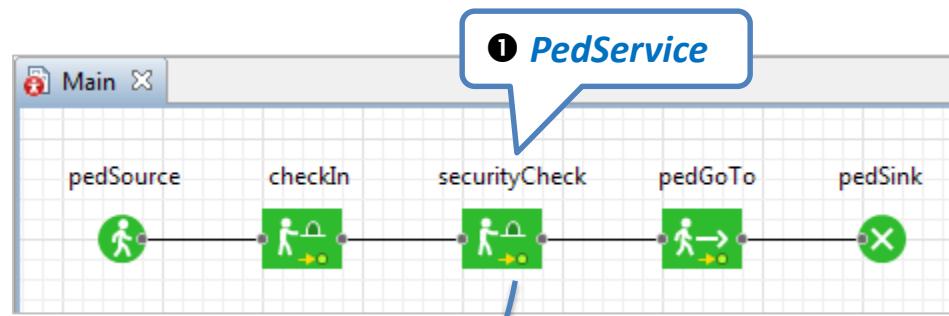
Airport. Phase 3. Step 1



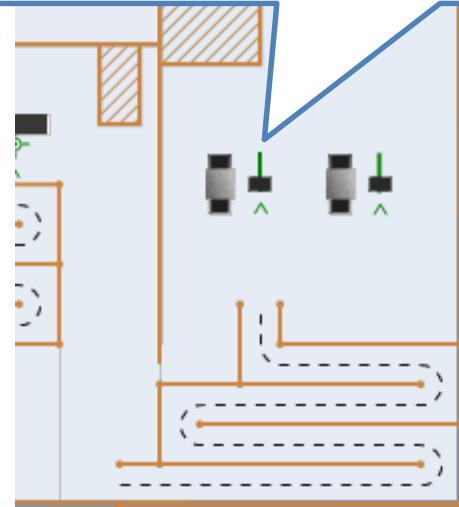
Airport. Phase 3. Step 2



Airport. Phase 3. Step 3



2 Set Type of service: Linear
and place the service lines so that they go through the metal detector



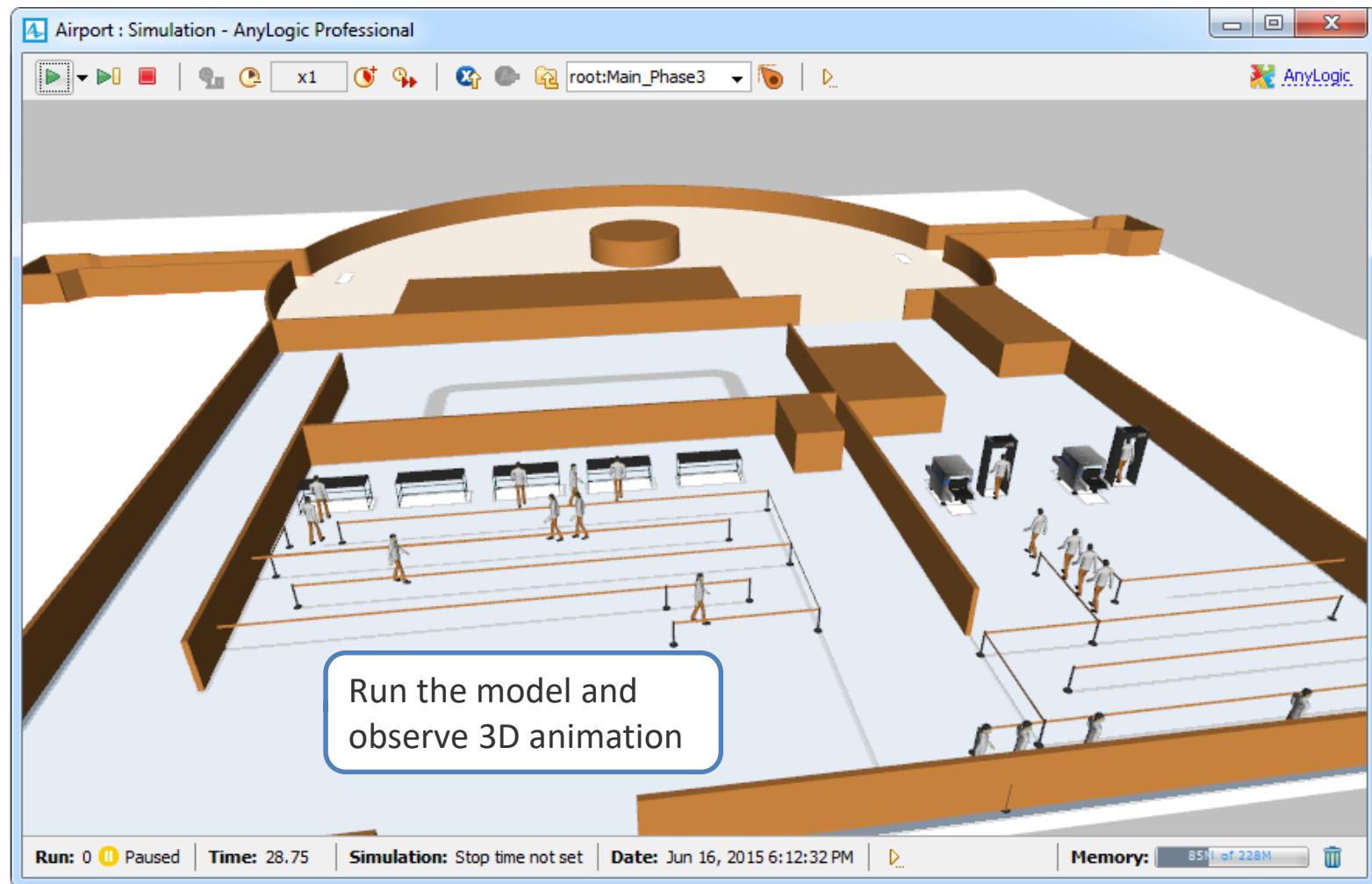
Point and linear services

There are two types of services: *Linear* and *Point*. We used point services to define check-in services, now we need linear services to define how people go through metal detectors at security checkpoints.

- *Linear* service defines a line, along which pedestrians should move. Pedestrian goes to the start point of the line and starts waiting there for the specified delay. Then pedestrian moves to end point of line. Recovery phase begins when pedestrian either finishes service (wait for ped to exit is off) or passes end point of line (wait for ped to exit is on). After recovery, the service becomes idle and ready to accept new pedestrians.
- *Point* service defines a point, where pedestrians should stay on for service delay time.



Airport. Phase 3. Step 4



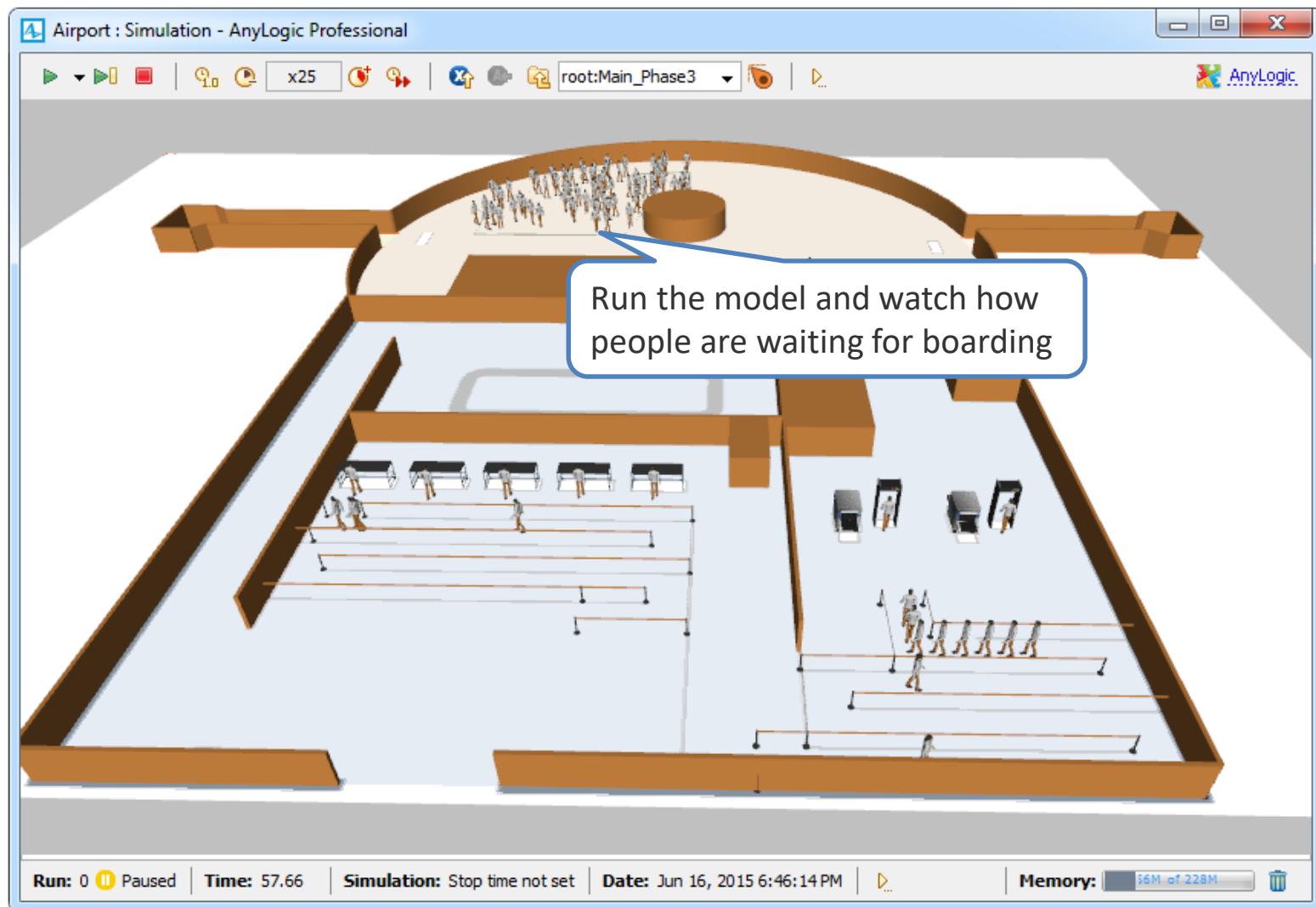
Airport. Phase 3. Step 5

The screenshot shows the AnyLogic software interface with the following components:

- Projects** tab: Shows the current project structure.
- Palette** tab: Shows the "Pedestrian Library".
 - Pedestrian Type**: A red icon of a person walking.
 - Space Markup**: A dropdown menu.
 - Wall**: A yellow square icon.
 - Rectangular Wall**: An orange square icon.
 - Circular Wall**: An orange circle icon.
 - Target Line**: A green line icon.
 - Service With Lines**: A blue line icon.
 - Service With Area**: A blue shaded area icon.
 - Rectangular Area**: A grey shaded area icon.
 - Polygonal Area**: A blue polygon icon (highlighted with a red border).
 - Attractor**: A blue star icon.
 - Pathway**: A blue path icon.
- Diagram Area**: Shows a flowchart with four nodes:
 - securityCheck**: A green square node with a person icon.
 - pedWait**: A green square node with a person icon and a clock icon (highlighted with a blue box and labeled "② PedWait").
 - pedGoTo**: A green square node with a person icon and an arrow icon.
 - pedSink**: A green circle node with a cross icon.
- Properties Panel**: Shows the properties for the "pedWait - PedWait" behavior.
 - Name:** pedWait Show name
 - Ignore:**
 - Waiting location:** area line point (x, y)
 - Area:** area line point (x, y) (highlighted with a red border)
 - Attractor choice:** None
 - Delay ends:** On delay time expiry On free() function call
 - Delay time:** 2 hours (highlighted with a red border)



Airport. Phase 3. Step 6





Airport. Phase 4

- In this phase we will show how to benefit from our true integration of all modeling methods.
- We will model how an infection is spreading in the airport.
- From time to time infected people will enter the airport, and in case of long queues they will infect the passengers that will be standing near them for a reasonable time.



Airport. Phase 4. Step 1

The screenshot shows the AnyLogic software interface. At the top, there is a green bar with the text "Airport. Phase 4. Step 1". Below it, the main window has two tabs: "Passenger" and "Properties".

Passenger Diagram: This tab shows a simple model with a single variable named "infection" represented by a small orange circle.

Properties Tab: This tab displays the properties of the "infection" variable. The variable is defined as follows:

- Name:** infection (highlighted with a red box)
- Visible:** yes (radio button selected)
- Type:** double
- Initial value:** randomTrue(0.05) ? 1 : 0 (highlighted with a red box)

Two numbered callouts point to specific parts of the interface:

- ① Open Passenger diagram by double-clicking it in the Projects tree** (points to the "Passenger" tab)
- ② Add a variable that will define the health status** (points to the "Initial value" field in the Properties tab)



Airport. Phase 4. Step 2

① Make the existing shape be shown only in 3D

② Draw a circle of radius 3 and make it shown only in 2D. Let the shape color depend on the health status.

Properties for person - 3D Object:

- Name: person
- File: x3d/person.x3d
- Scale: 100%
- Orientation: (radio buttons for various orientations)
- Show in: 3D only (highlighted with a red box)

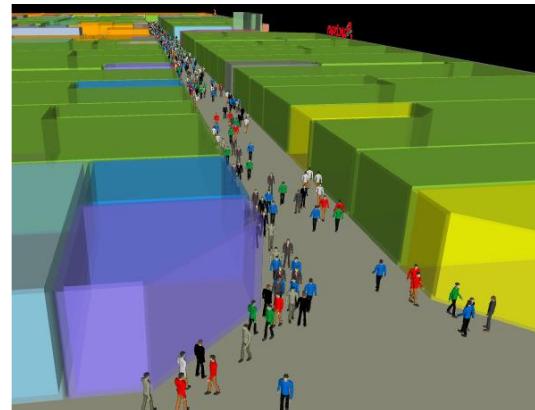
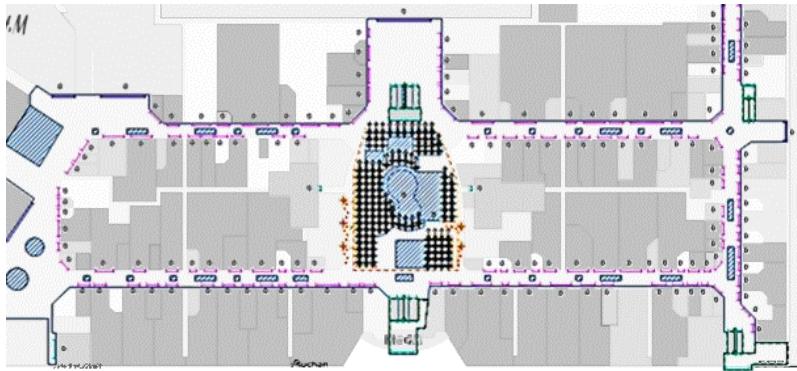
Properties for oval - Oval:

- Name: oval
- Fill color: lerpColor(infection, blue, red) (highlighted with a red box)
- Show in: 2D only (highlighted with a red box)

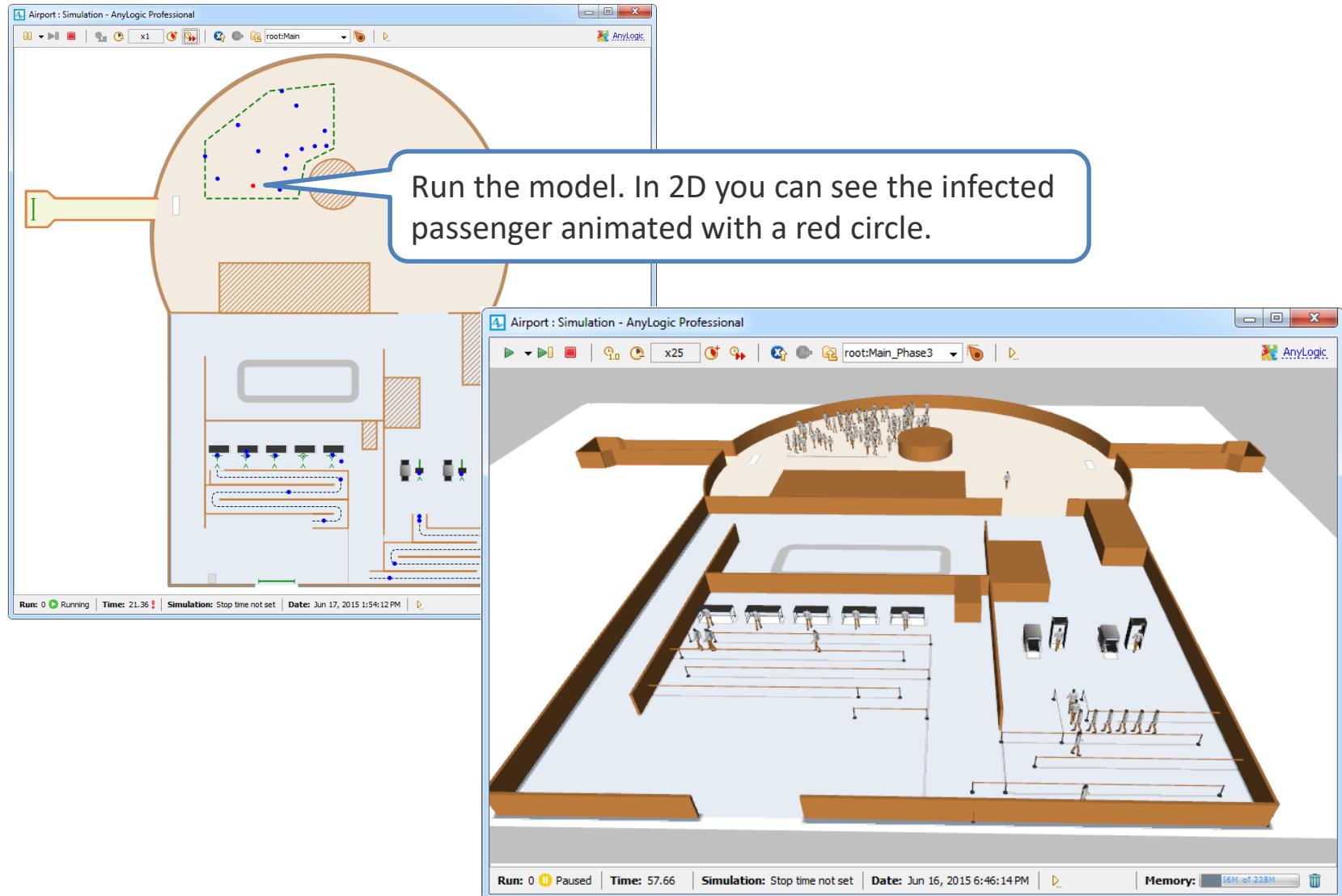


Different animation shapes for 2D & 3D

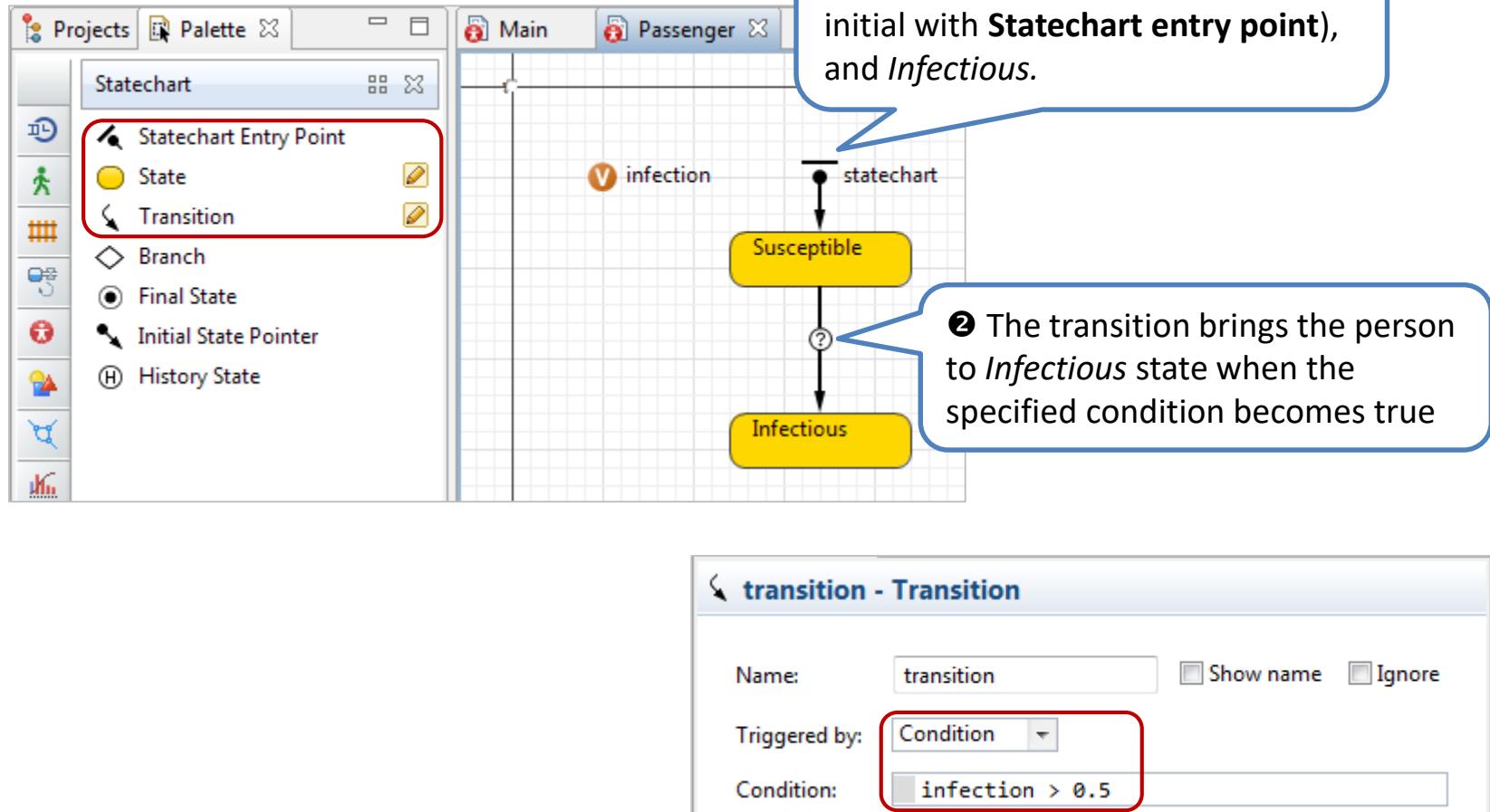
- By default all shapes are shown both in 2D and 3D.
- Sometimes you may need to animate the same object with different 2D and 3D shapes.
- You can draw two animations, and set one shape to be shown only in 2D, while another only in 3D (in the shape's **Advanced** property section, **Show in** property).



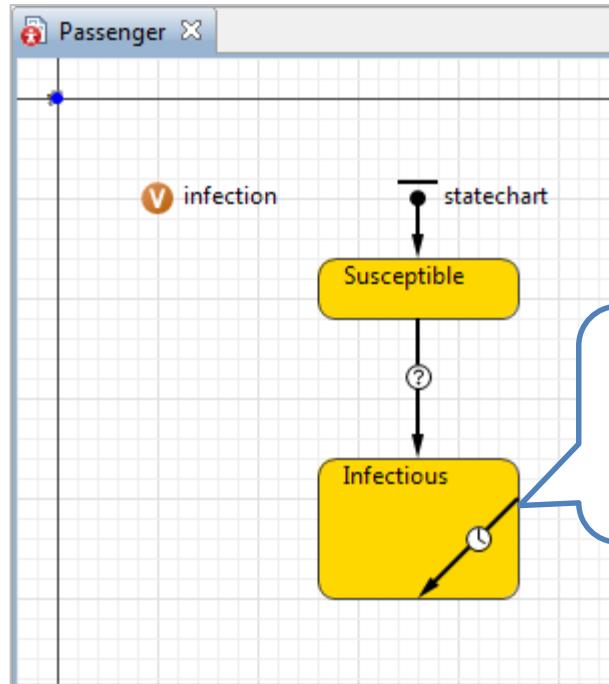
Airport. Phase 4. Step 3



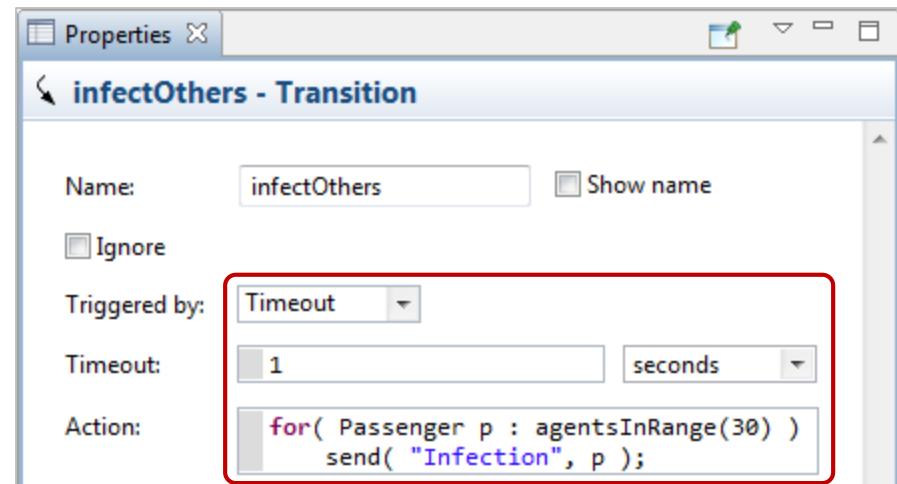
Airport. Phase 4. Step 4



Airport. Phase 4. Step 5



① Add an internal transition inside the state *Infectious* that will model how the infected person gradually infects his neighbors.

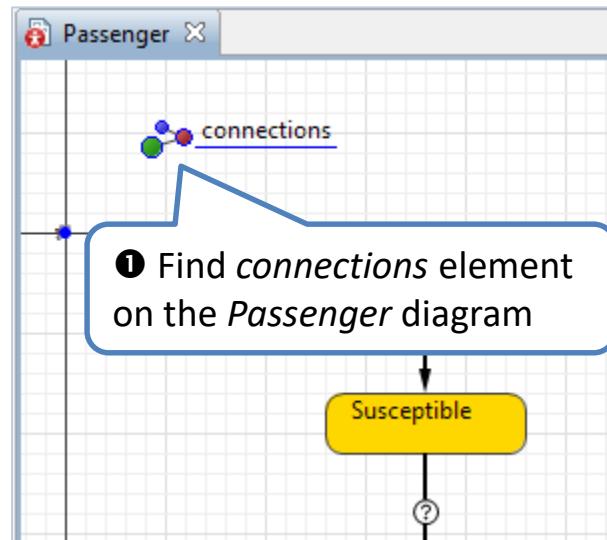


“for” loop. “Collection iterator” form

- Here we use the “collection iterator” form of the for loop. This form is used to iterate through populations and collections. Whenever you need to do something with each agent of some population we recommend to use this loop because it is more compact and easy to read, and also is supported by all collection types (unlike index-based iteration).



Airport. Phase 4. Step 6



The properties dialog for the 'connections' element is shown. A callout box contains the text: '② Here you define how the connected agents react on the message reception'. The 'On message received:' field contains the code: `infection += 0.001;`, which is highlighted with a red border.

Properties for 'connections - Link to agents':

- Name: connections
- Show name: checked
- Ignore: unchecked
- Visible: yes
- Agent type: Agent
- Collection of links (radio button selected)
- Single link (radio button unselected)

This standard link is always bidirectional

Communication

These actions are executed for messages from all applicable connections

Message type: Object

On message received:

```
infection += 0.001;
```

Forward message to:

Statecharts	
<input checked="" type="checkbox"/> statechart	



Each agent has a non-removable link *connections* – this is for standard networks with bidirectional connections.

In the **Communication** properties section, you define the reaction on the message reception. Here we increase the value of *infection* variable by 0.001.

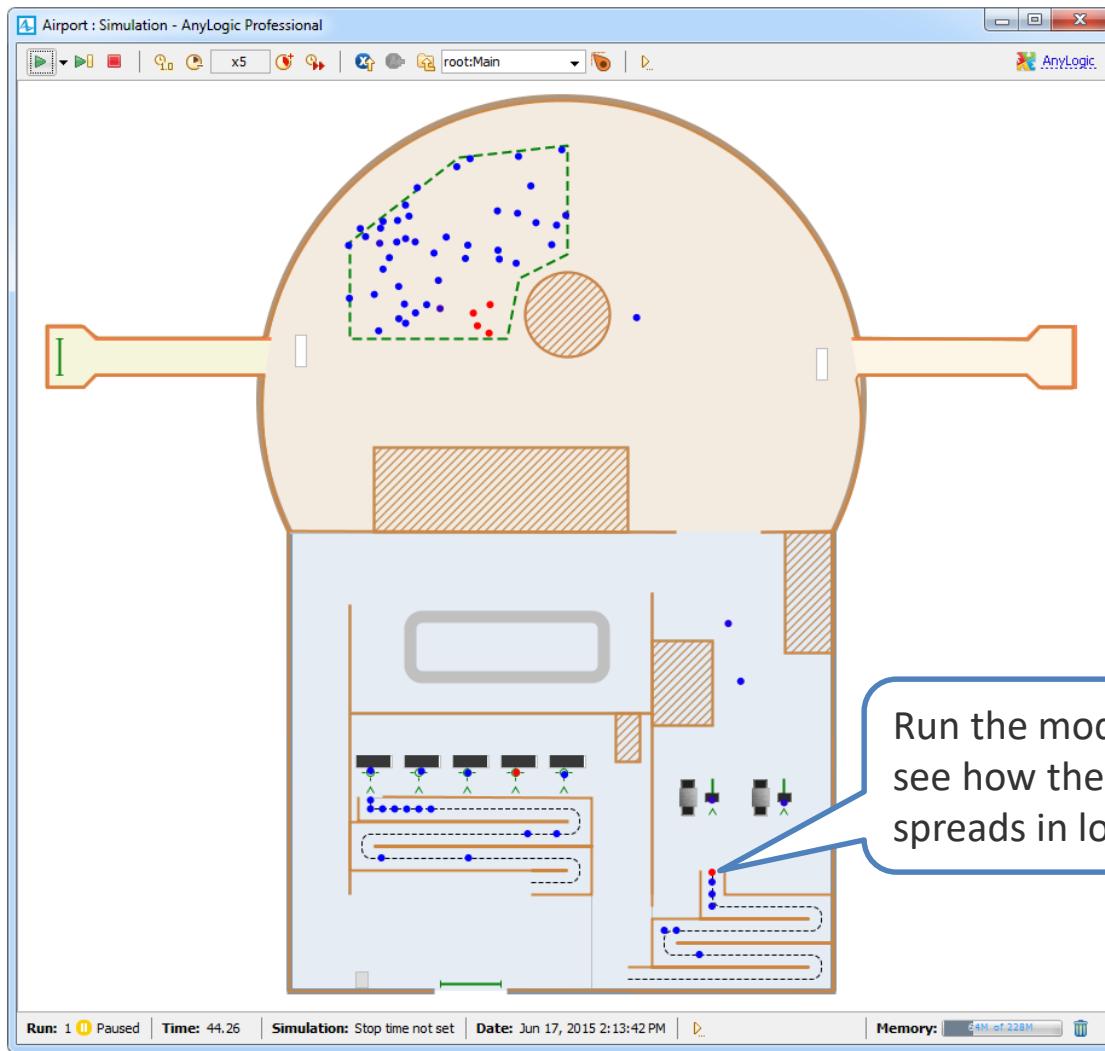
As we defined earlier (Phase 4, Step 4), the person is considered infected when *infection* variable surmounts the given threshold (0.5).

According to this logic, what time does it take for the person to become infected when standing near some infectious person?

Can you propose some alternative implementations of the infection logic?



Airport. Phase 4. Step 7



Airport. Exercise

- Let one security checkpoint operate permanently, while another service point will be operating only when the number of people in the queue is critical (e.g. exceeds 20 people).



Piazza del Colosseo

This presentation is a part of
AnyLogic Standard Training Program



Piazza del Colosseo

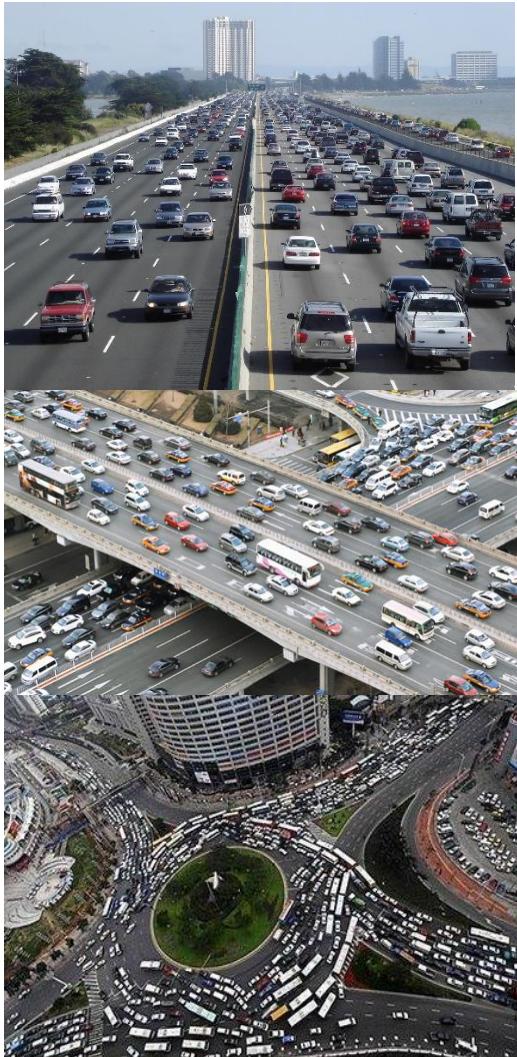
- Let's build a simulation model of road traffic around Piazza del Colosseo in Rome, Italy, with the help of AnyLogic Road Traffic Library. We will model an intersection with cars arriving from four different directions.
- In the last phase we will add traffic lights to the model and examine the distribution of time spent by vehicles on the intersection. We will then set up an optimization experiment that will analyze the phases' length of the traffic lights to optimize the road network throughput.



[Click here to view the video of the Piazza del Colosseo intersection](#)



Piazza del Colosseo



Road Traffic Library

- Roads, lanes, intersections, parking lots, traffic lights
- Logic:
 - Routing (shortest distance)
 - Keeping distance
 - Overtaking
 - Giving way
- Animation: 2D, 3D, Heat (density) map
- Interoperability with the Process Modeling and other standard Libraries

Piazza del Colosseo. Phase 1

- In this phase, we will add the layout and simulate the vehicle traffic along the first road.



Piazza del Colosseo. Phase 1. Step 1

The screenshot shows the AnyLogic simulation environment. On the left, the 'Main' workspace displays an aerial view of a plaza and surrounding urban infrastructure. A blue callout box labeled '① Add the Image element' points to the image element in the workspace. On the right, the 'Properties' panel is open, showing settings for an 'image' element named 'image'. A second blue callout box labeled '② Lock the Image element' points to the 'Lock' checkbox, which is checked and highlighted with a red border. A third blue callout box labeled '③ Verify the Position and size properties' points to the 'Position and size' section, where the X and Y coordinates are both set to 0, and the Width is 1100 and Height is 700, all of which are also highlighted with a red border.

① Add the Image element

② Lock the Image element

③ Verify the Position and size properties

Name: image

Visible on upper level Icon Lock

Visible: yes

Reset to original size

Images:

piazza_layout.png

X: 0 Width: 1100

Y: 0 Height: 700



Create a new model and name it *Piazza del Colosseo*.

- ❶ Add the layout image onto the Main agent diagram to draw the actual road network upon it.

Choose the image file to be displayed by this **Image** shape:
[*piazza_layout.png*](#) from the *Models\Piazza del Colosseo* folder located on your USB disk.

We will now set the **Image** element properties.

- ❷ Select the **Lock** check box. Since we will not need to further move or resize the image, we can lock it in the graphical editor. This will prevent us from accidentally selecting it.
- ❸ Adjust the **Image** element **Position and size** properties. We will set **X** and **Y** to *0* so that the image top-left corner matches the axis origin. We will also ensure that the **Image** element **Width** and **Height** correspond to the picture dimensions (*1100 x 700*).



Piazza del Colosseo. Phase 1. Step 2

① Disable grid

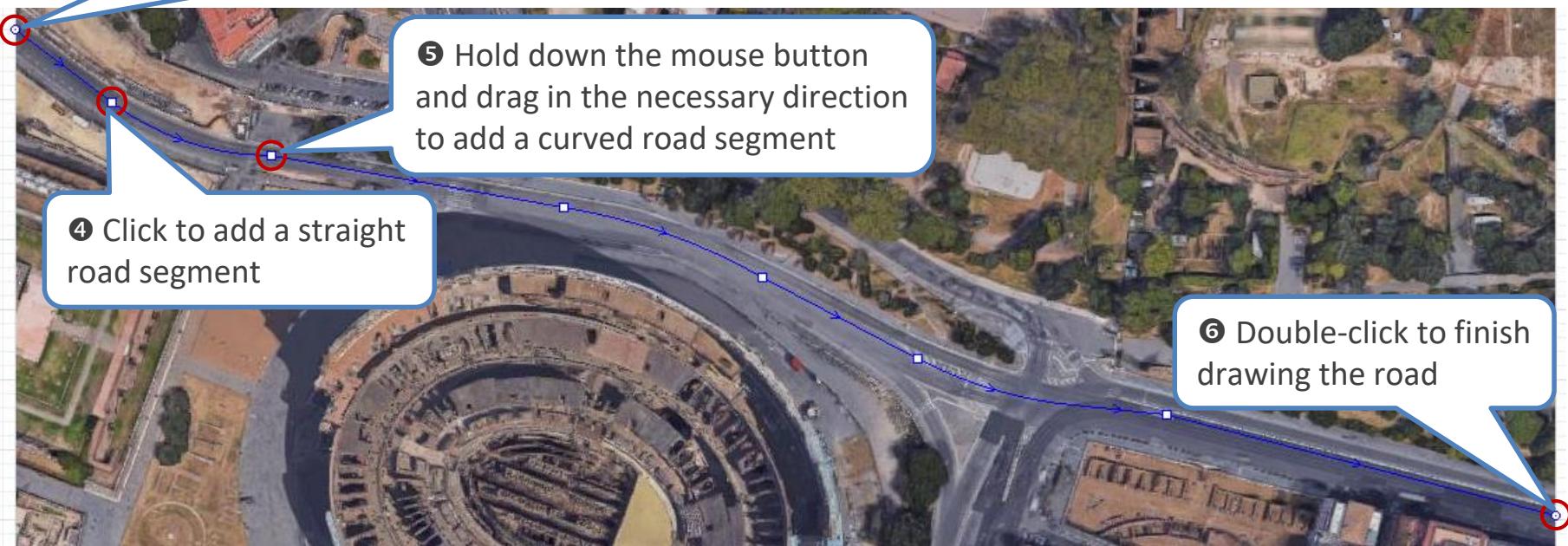
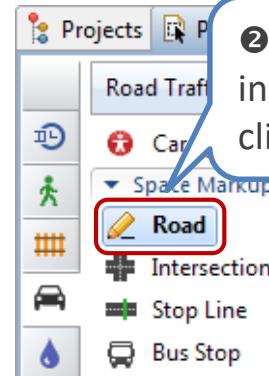


③ Click to start drawing the road

⑤ Hold down the mouse button and drag in the necessary direction to add a curved road segment

④ Click to add a straight road segment

⑥ Double-click to finish drawing the road



We will draw the first road in the West-to-East direction (Via dei Fori Imperiali – Via Labicana).

- ① Disable grid by clicking the  **Enable/Disable grid** button on the main toolbar. This will allow us to position the road points freely, without snapping them to grid.
- ② Open the **Road Traffic Library** palette and double-click the **Road** element to enter the road drawing mode.

We will now draw the road point by point.

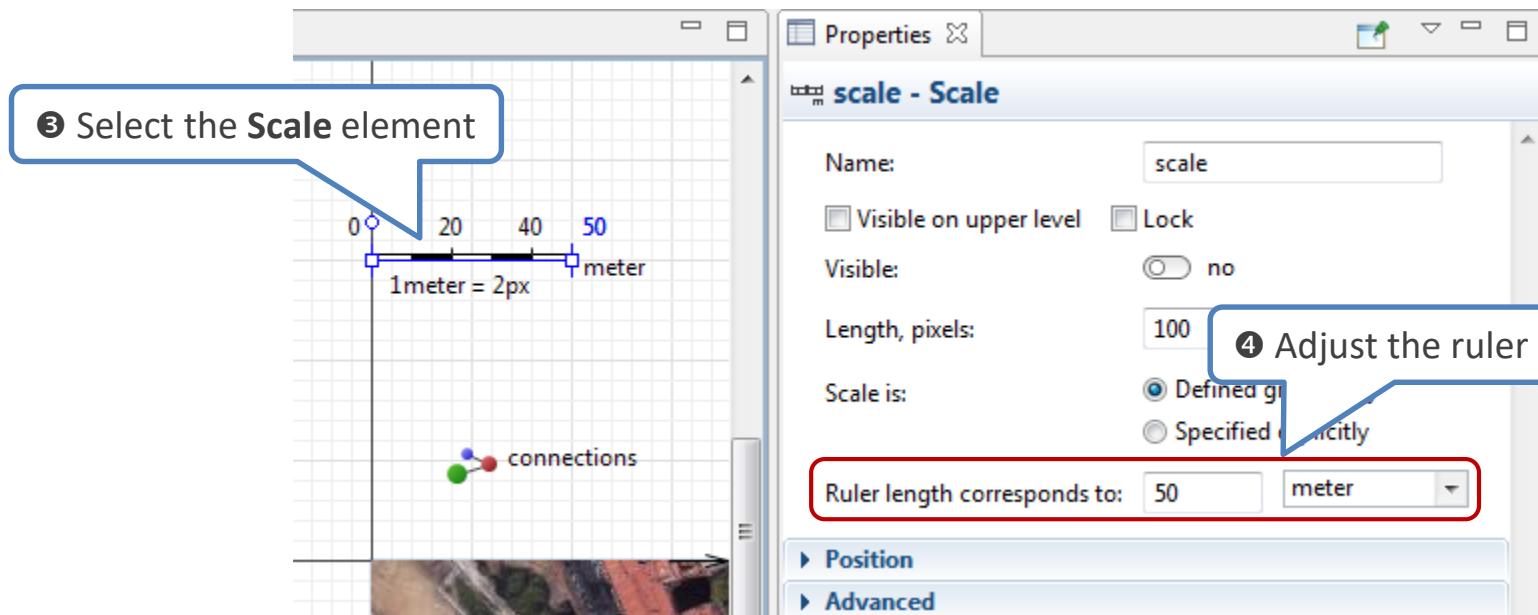
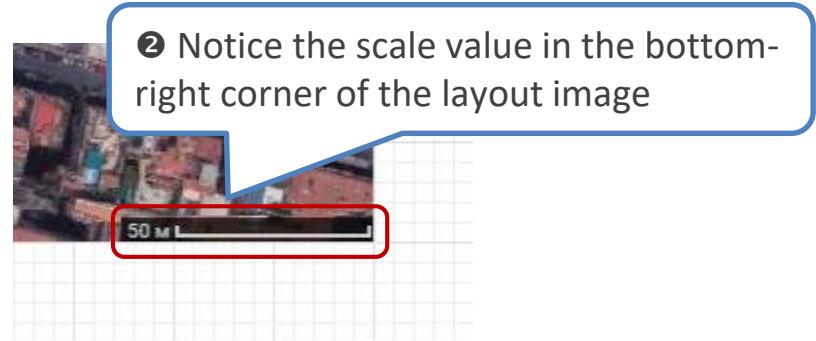
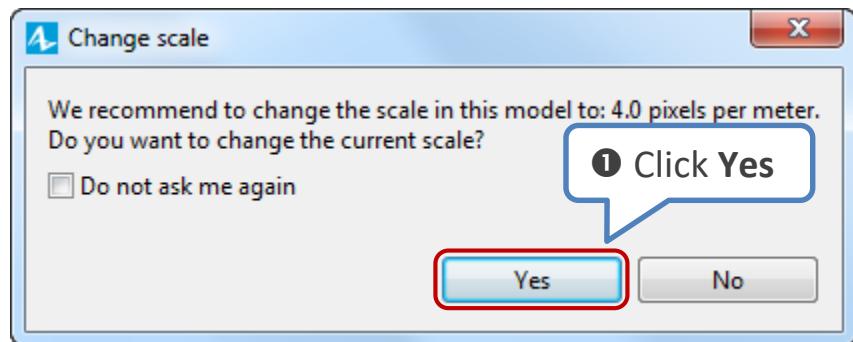
- ③ Start drawing by clicking in the graphical editor.
- ④ To add a straight road segment, click in the graphical editor.
- ⑤ To add a curved road segment, click in the graphical editor and hold down the mouse button. Drag in the necessary direction to adjust the curve shape and release the mouse button when done.

Repeat the steps ④-⑤ as necessary so that the resulting road shape resembles the one on the layout image.

- ⑥ To finish drawing the road, double-click in the graphical editor.



Piazza del Colosseo. Phase 1. Step 3



The interface shows a workspace with a scale element selected. The scale element is a horizontal ruler with markings at 0, 20, 40, and 50, labeled "1 meter = 2px" below it. A callout bubble points to this element with the text: "③ Select the Scale element". To the right is the "Properties" window for the selected "scale - Scale" element. The properties are:

- Name: scale
- Visible on upper level:
- Lock:
- Visible: no
- Length, pixels: 100
- Scale is:
 - Defined by
 - Specified explicitly
- Ruler length corresponds to: 50 meter

A callout bubble points to the "Ruler length corresponds to" field with the text: "④ Adjust the ruler scale".



We will adjust the model scale so that it fits the selected layout image.

- ❶ When you finish drawing the road, AnyLogic will prompt you to automatically adjust the model scale. Click **Yes**. The model scale will be adjusted to *4px per meter*.
- ❷ If you look at the bottom-right corner of the layout image, you will notice that the image is scaled so that two grid squares correspond to 50 meters. We will now adjust the model scale accordingly.
- ❸ Select the **Scale** element located above the layout image.
- ❹ Set the **Ruler length corresponds to** property value to *50 meters*. Our model is now scaled in full correspondence with the layout image.



Piazza del Colosseo. Phase 1. Step 4

The screenshot shows an aerial view of the Colosseum area in Rome. A blue dashed line highlights a section of the road network. Three numbered callouts provide instructions:

- ① Click the road to select it.
- ② Click the road again to select the road network.
- ③ Adjust the road network **Lane width** parameters.

To the right, the "Properties" panel is open for the selected "roadNetwork - Road Network" object. The properties shown are:

- Name: roadNetwork
- Ignore:
- Visible on upper level:
- Lock:
- Visible: yes
- Traffic: Right-hand
 Left-hand
- Lane width: 3.0 meter

A red box highlights the "Lane width" field, indicating it is the parameter being adjusted.



We will adjust the parameters of the road network.

- ① Select the road by clicking it.
- ② With the road selected, click it again to select the road network.
- ③ Set the road network **Lane width** to *3.0 meters*.

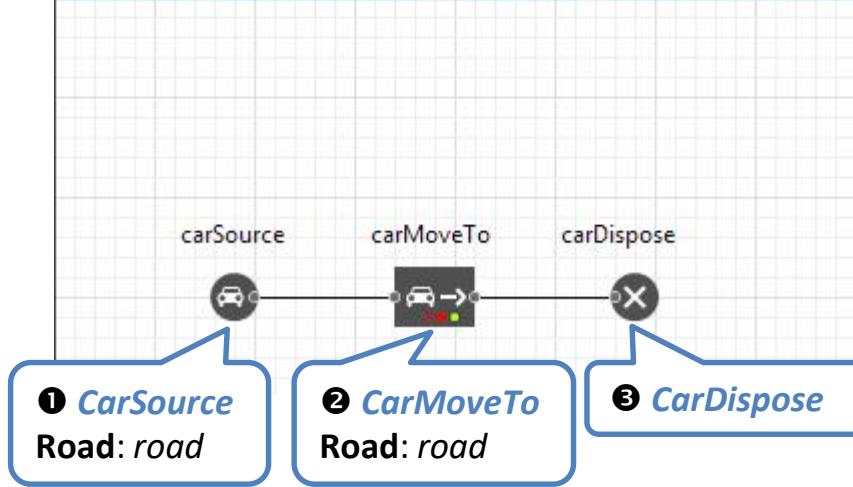
Road network

A road network joins several roads together. Its parameters affect all roads belonging to the network.

Several important parameters, such as lane width, traffic direction ([left-hand](#) or [right-hand](#)) and road background color can be defined only for the road network but not for individual roads.



Piazza del Colosseo. Phase 1. Step 5

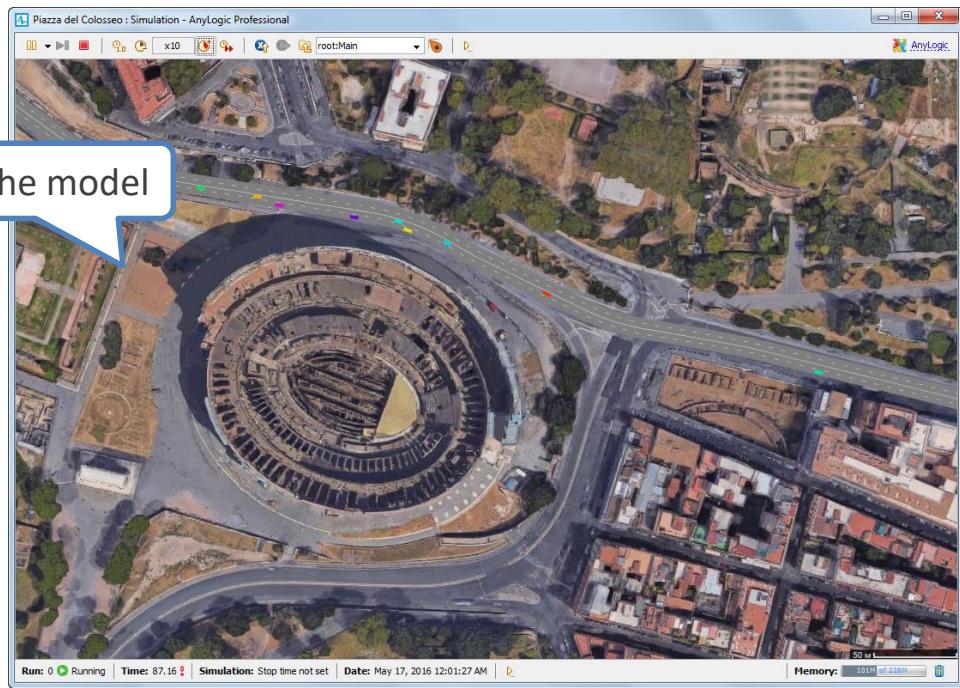
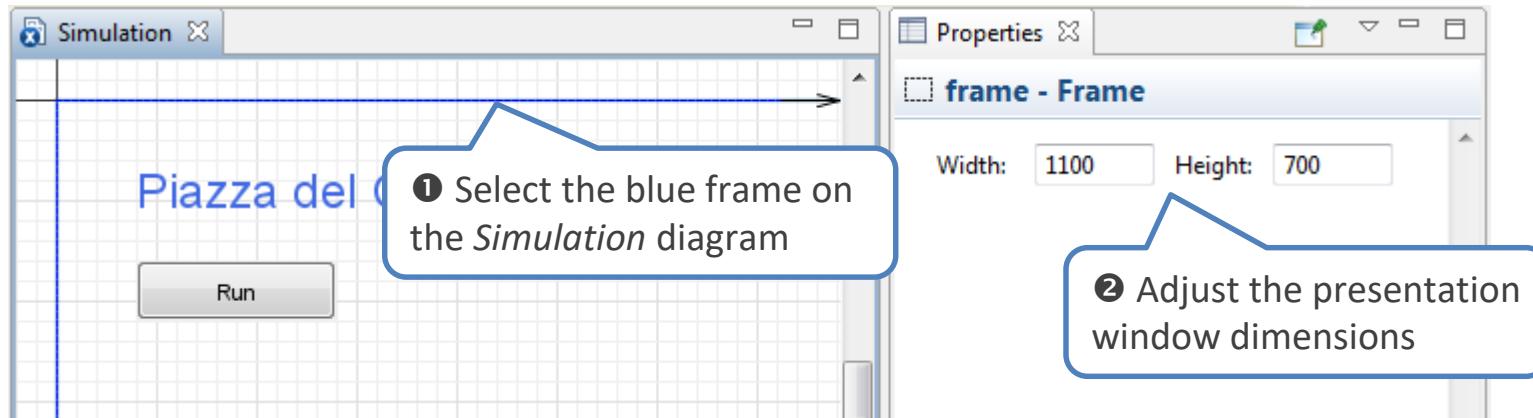


We will now create the flowchart to define the vehicles movement along the road.

- ① - ③** Add flowchart elements that define the West-to-East vehicles movement along the forward lane.



Piazza del Colosseo. Phase 1. Step 6

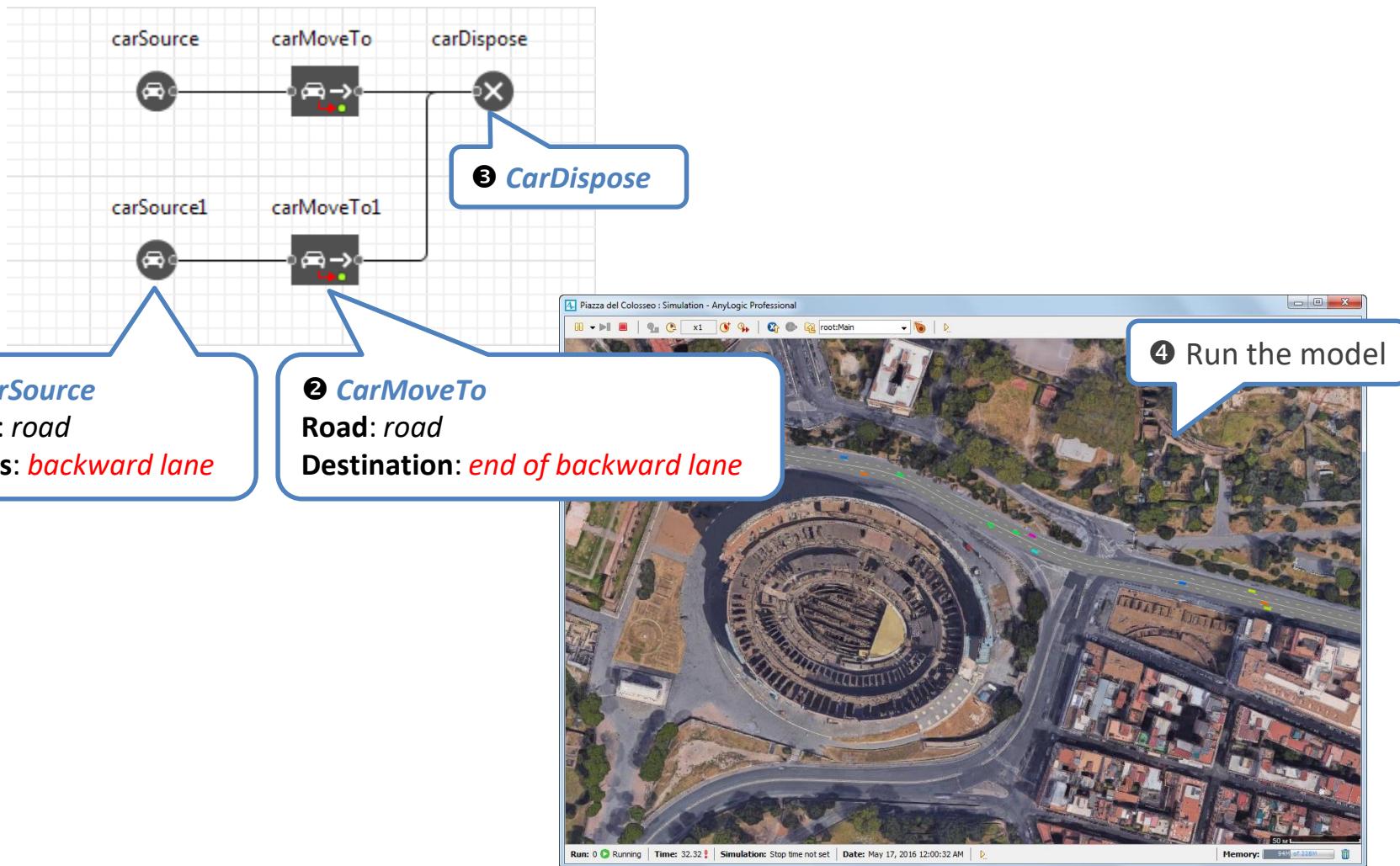


We are ready to run the simulation experiment.

- ① - ② Adjust the presentation window dimensions so that the layout image fits the window. Set **Width** and **Height** to *1100* and *700*, respectively.
- ③ You can now run the model and watch the animation in the presentation window. You will see the vehicles moving along the road in the West-to-East direction.



Piazza del Colosseo. Phase 1. Step 7



We will extend the flowchart to define the vehicles movement along the road in the East-to-West direction.

- ① - ② Add the elements to the flowchart and customize them to define the East-to-West vehicles movement along the backward lane.
- ③ Connect the **Car Move To** element to the **Car Dispose** element defined on the previous step.
- ④ Run the model. You will see the vehicles moving along the road in both directions.



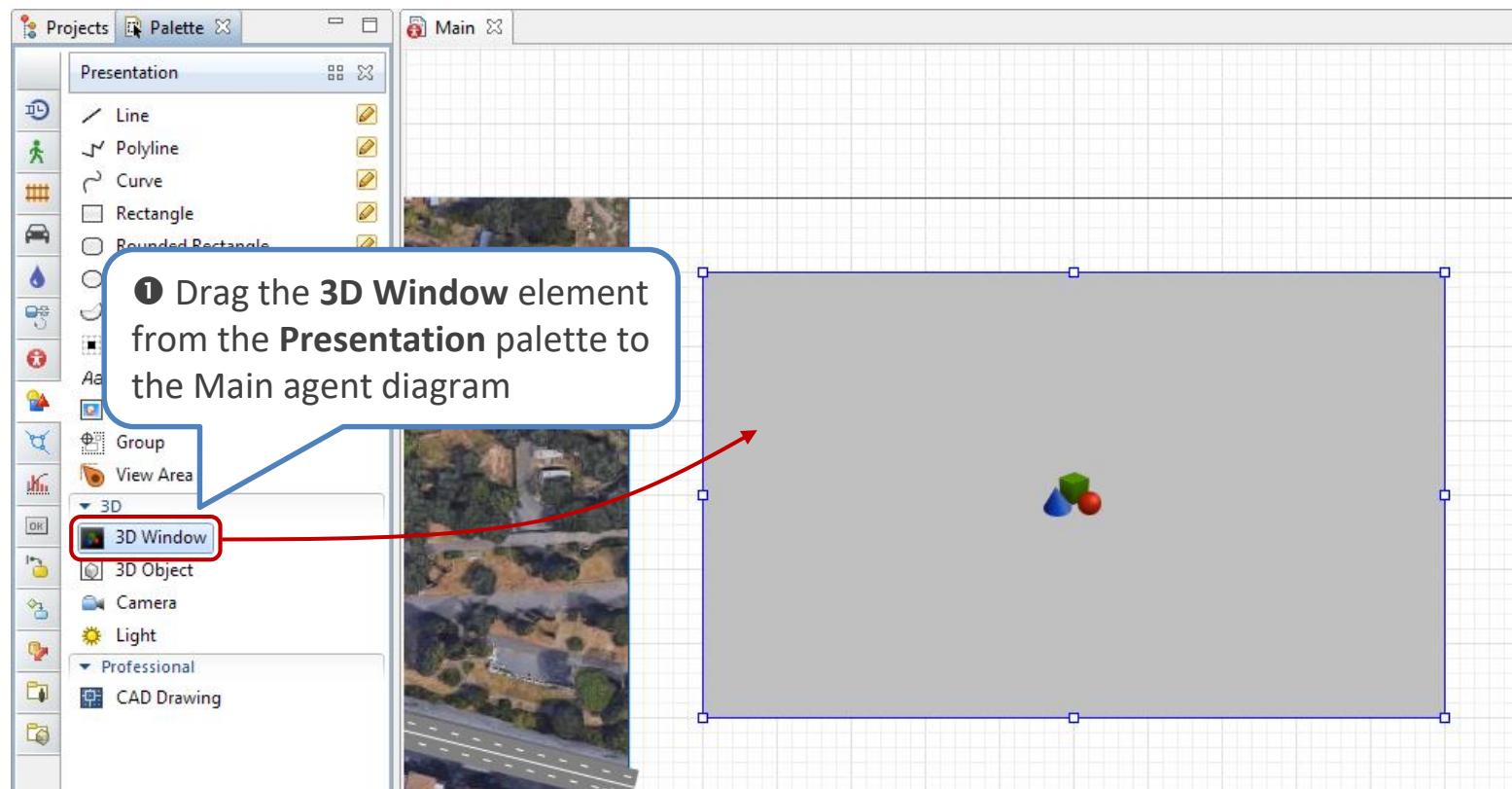
Piazza del Colosseo. Phase 2

- In this phase, we will set up 3D animation and add 3D models for cars and the Colosseo.





Piazza del Colosseo. Phase 2. Step 1



We will start creating 3D animation.

- ① Drag the **3D Window** element from the **Presentation** palette onto the Main diagram and place it on the right of the layout image.

AnyLogic will automatically create a view area for the 3D window, which will allow us to easily navigate to it during the model run.



Piazza del Colosseo. Phase 2. Step 2

The screenshot shows the AnyLogic software interface with three main windows:

- Projects** window (left): Shows the project structure with "Piazza del Colosseo" selected. A blue callout box labeled ① Select image points to the "image" node under "Main". A red arrow points from this node to the "image" node in the "Simulation - Simulation Experiment" window.
- Simulation - Simulation Experiment** window (center): Displays simulation parameters. A blue callout box labeled ③ Select Simulation: Main points to the "Simulation: Main" node under "Main". A red arrow points from this node to the "Simulation" node in the Properties window.
- Properties** window (right): Shows properties for the "image" node. A blue callout box labeled ② Adjust the image Z-coordinate points to the "Z" field, which is set to "-0.1". A red arrow points from this field to the "Enable zoom and panning" checkbox in the "Simulation" section of the central window. Another blue callout box labeled ④ Disable panning and zoom points to the "Close confirmation" checkbox in the same section.



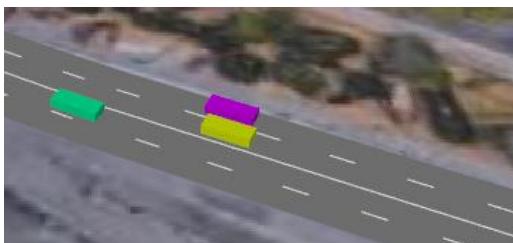
We will perform several preparatory steps for running the model in 3D.

①-② In the **Projects** view, select *image* and set its Z-coordinate to *-0.1*.

Initially, the Z-coordinate is set to *0* and the road shares the same plane with the layout image. This causes visual interference when viewing 3D animation:

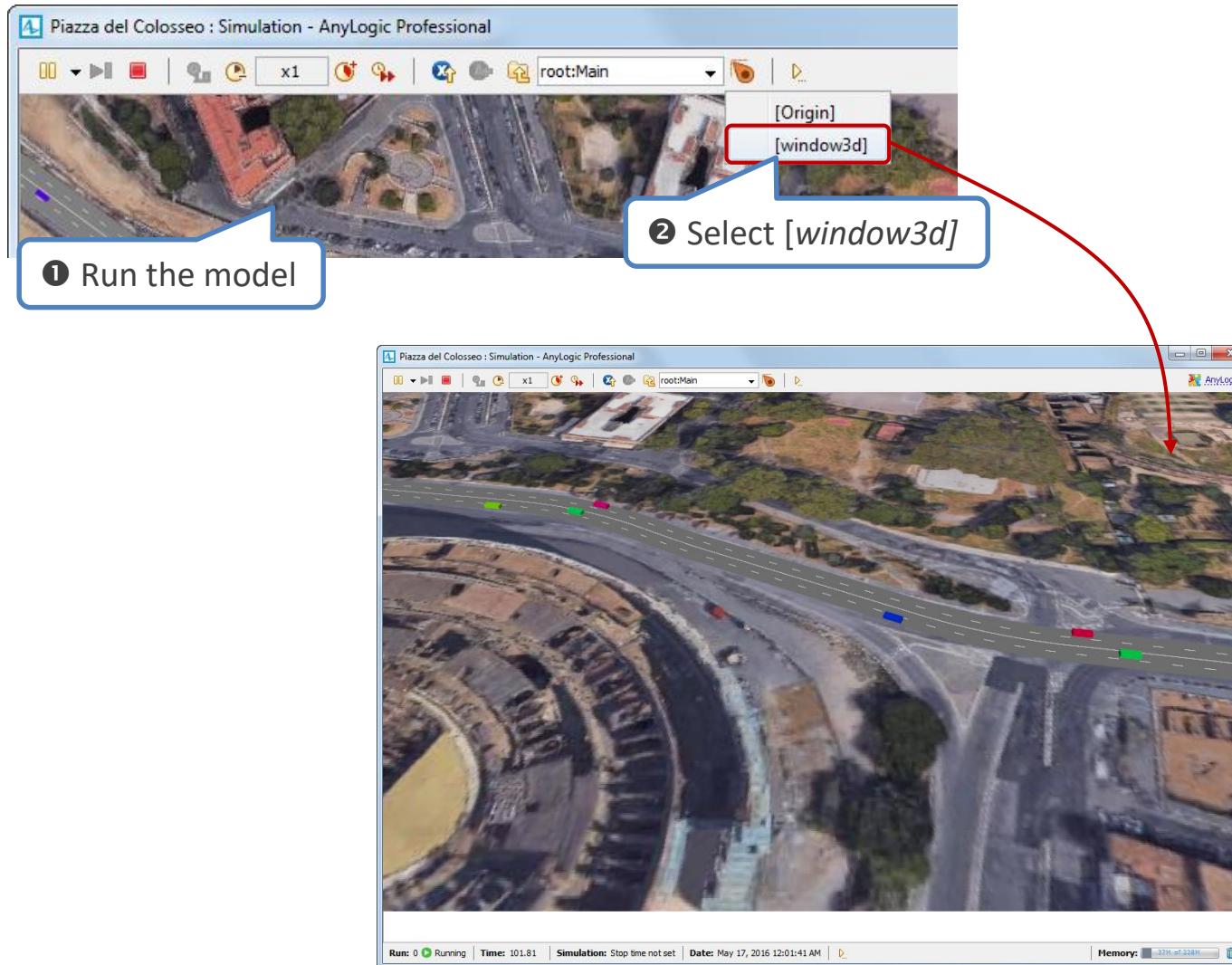


By setting the Z-coordinate to *-0.1*, we move the image slightly below the road thus eliminating interference:



③-④ In the **Projects** view, select *Simulation: Main* and disable panning and zooming at model runtime, since we will rely on view areas for navigating the model instead of doing it manually.

Piazza del Colosseo. Phase 2. Step 3



We are ready to run our model in 3D.

- ① Run the model. You will see the 2D animation of the model.
- ② Click the  **Navigate to view area** button on the toolbar and select *[window3d]*. The model animation will switch to 3D.

Note that cars are currently displayed as colored parallelepipeds. In the next step, we will adjust the model so that custom 3D shapes are used for displaying cars.



Piazza del Colosseo. Phase 2. Step 4

The screenshot shows the AnyLogic software interface. At the top, there's a palette titled "Road Traffic Library" containing various icons. A red box highlights the "Car Type" icon. A blue callout bubble with the number "1" points from this icon to a "Car" agent object on the "Main" agent diagram to its right. Below this, two windows are open: "Step 1. Creating new agent type" and "Step 2. Agent animation".

Step 1. Creating new agent type

- Agent type name: Car
- Create the agent type "from scratch"
- Use database table
I have agent data stored in a database

Step 2. Agent animation

Choose animation: 3D 2D None

Road Transport

- Car (selected)
- Lorry
- Truck
- Bus 1
- Bus 2
- Bus 3
- Ambulance Car

Two blue callout bubbles with numbers provide instructions: "1 Drag the Car Type element from the Road Traffic Library palette to the Main agent diagram", "2 Click Next", and "3 Click Finish".



We will now add a new **Car Type** agent type that will allow us to use 3D models for cars.

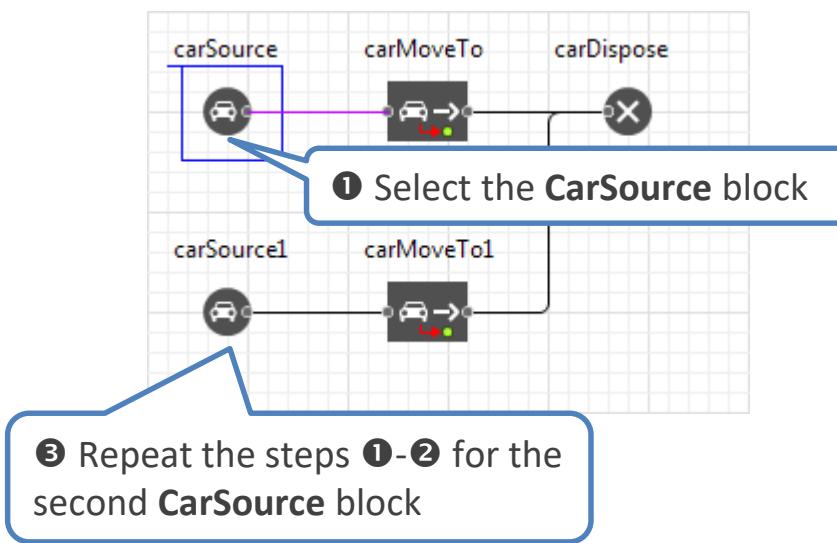
- ❶ Drag the **Car Type** element from the **Road Traffic Library** palette onto the Main agent diagram. The **New Agent** wizard will open.
- ❷-❸ We will start with creating a new car type with the default properties. Click **Next** and **Finish** to create a car type and exit the wizard.

Why should we create a car type?

- We need this to set realistic 3D shapes to our cars. To diversify cars in your model, you should create several car types, each one with a specific 3D model. Afterwards, set **Car Source** blocks to generate cars of the different car types.



Piazza del Colosseo. Phase 2. Step 5



Properties window for the **carSource - CarSource** block:

- Name: carSource Show name
- Ignore:
- Arrivals defined by: Rate per hour
- Set agent parameters from DB:
- Limited number of arrivals:
- Appears: on road in parking lot
- Road: road
- Enters: forward lane backward lane
- Random lane:
- Car** section:
 - New car:
 - Length: 5 meter
 - Initial speed: 60 kilometers per hour
 - Preferred speed: 60 kilometers per hour
 - Max acceleration: 1.8 meters per second²
 - Max deceleration: 4.2 meters per second²

A callout box labeled ② Set Car as New car points to the 'New car:' dropdown menu.

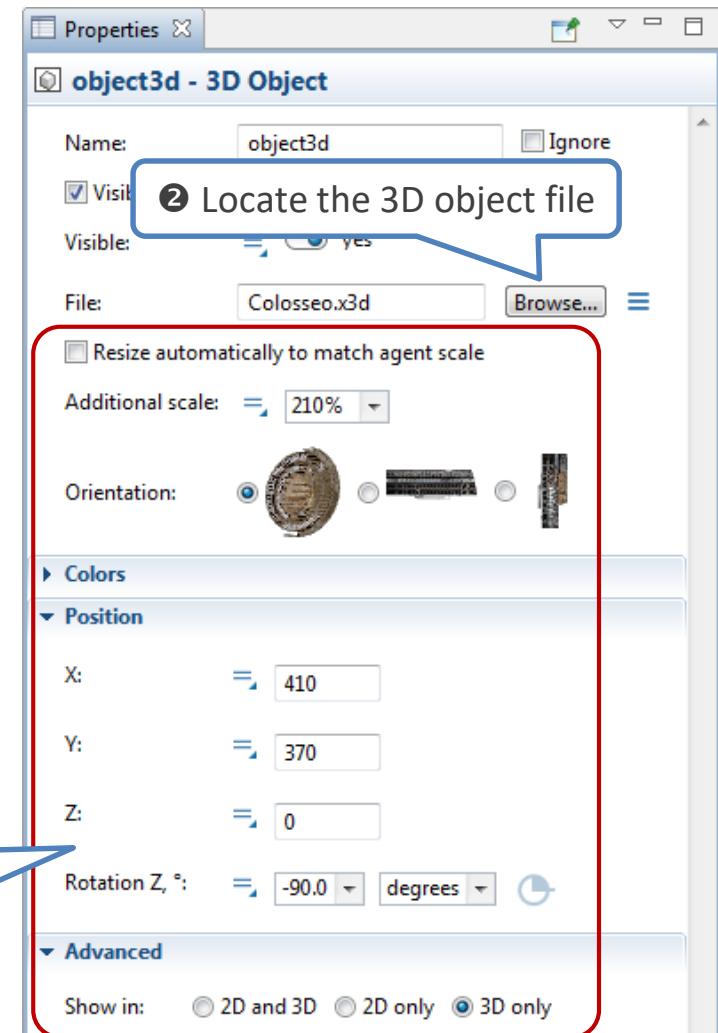
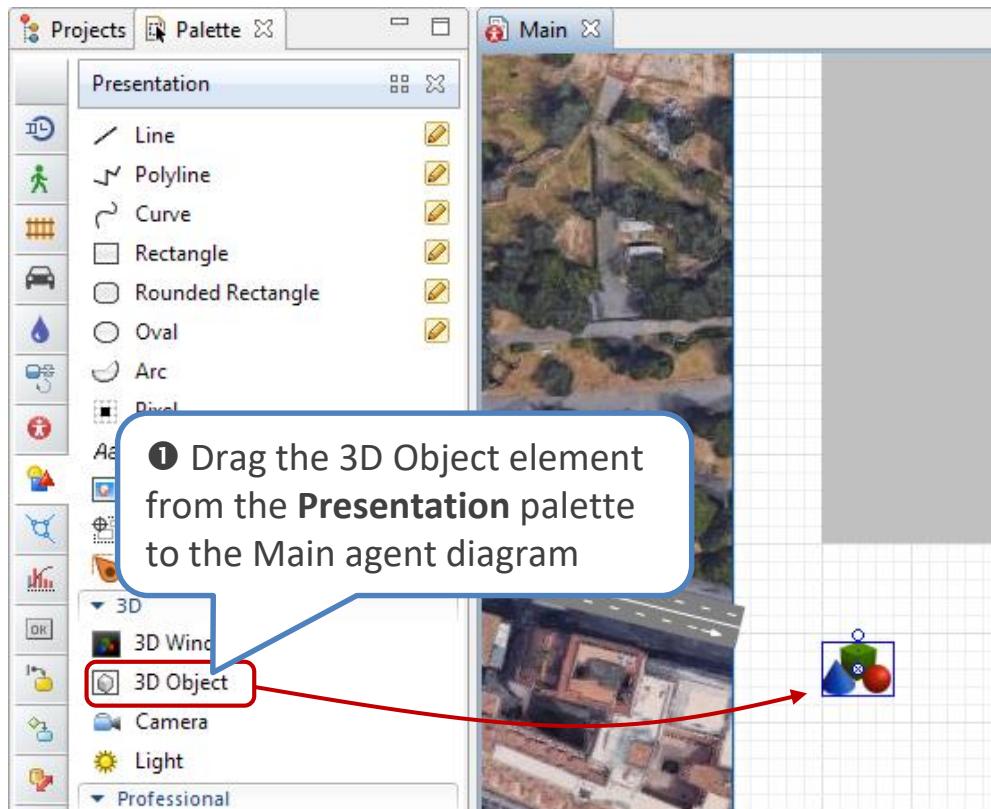


We will now configure the flowchart so that the newly created car is used in the road network.

- ① - ② Select the first **Car Source** element in the flowchart and set *Car* as **New car** in the **Properties** section.
- ③ Repeat the steps ① - ② for the second **Car Source** element. Both elements will now generate cars of the *Car* type.



Piazza del Colosseo. Phase 2. Step 6



We will add a 3D object for the 3D figure of the Colosseo.

- ❶ Drag the **3D Object** element from the **Presentation** palette onto the Main diagram.
- ❷ Click the **Browse** button and browse for the file used by **3D Object**: *Colosseo.x3d* from the *Models\Piazza del Colosseo* folder located on your USB disk.
- ❸ Set the element properties as displayed on the screenshot.

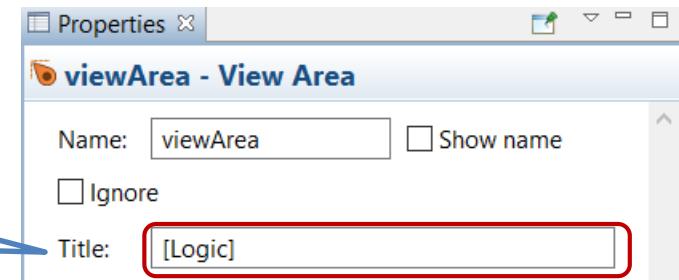
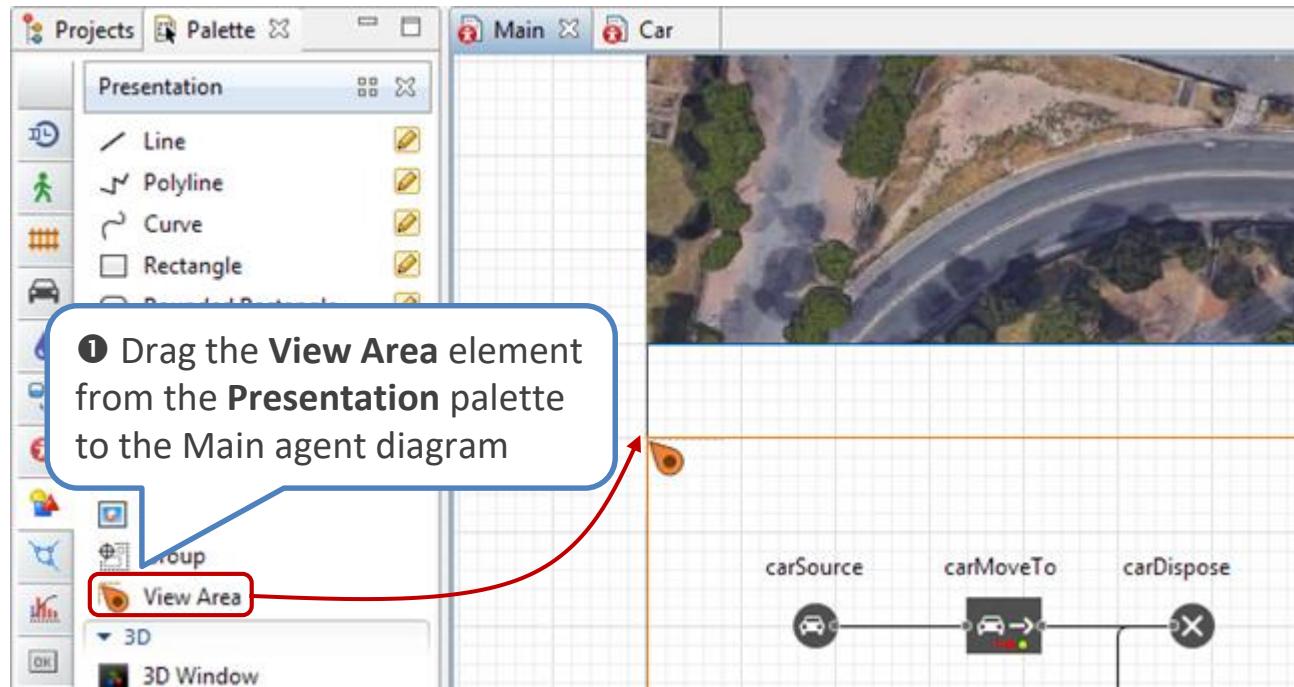
Importing 3D objects into the model

3D object supports importing of 3D objects stored in X3D and VRML files. You can follow this scenario to import a custom 3D object into a model:

- Search for the 3D object file and download it in the *Google Earth KMZ* format. You can get it from an online gallery (for example, [3DWarehouse](#)).
- Change the file extension to *zip* and extract its contents into a folder. Verify that the Collada model file (with the *.dae* extension) is present inside the folder.
- Import the *.dae* model file into 3D modeling software (for example, [Blender](#)) and export it as an *.x3d* file. The exported file can now be added in AnyLogic model as **3D object**.



Piazza del Colosseo. Phase 2. Step 7



We will add a view area to the model, which will allow us to navigate between animation and flowchart during the model run.

- ① Drag the **View Area** element from the **Presentation** palette onto the Main agent diagram and position it between the layout image and the flowchart (as displayed on the screenshot). Change the view area's title (it will appear in the navigation drop-down list, see the next slide).

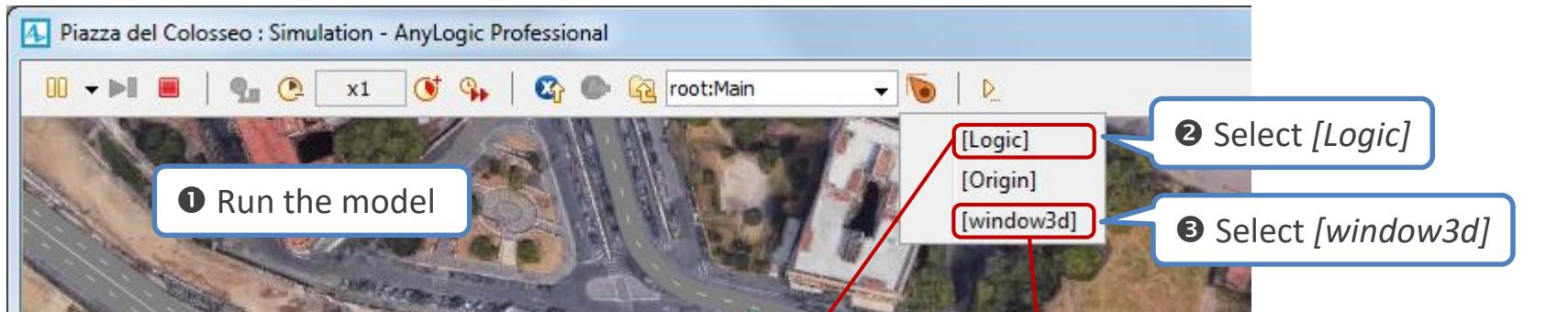
This will mark out the flowchart area and allow us to easily navigate to it during the model run.

View areas

- *View area* enables the user to mark out some particular areas on a diagram, containing some logically detached parts or groups of elements (e.g. presentation, flowchart, charts, etc.). Once having defined the view areas, you can easily navigate between them using special navigation tools at design time and/or runtime.



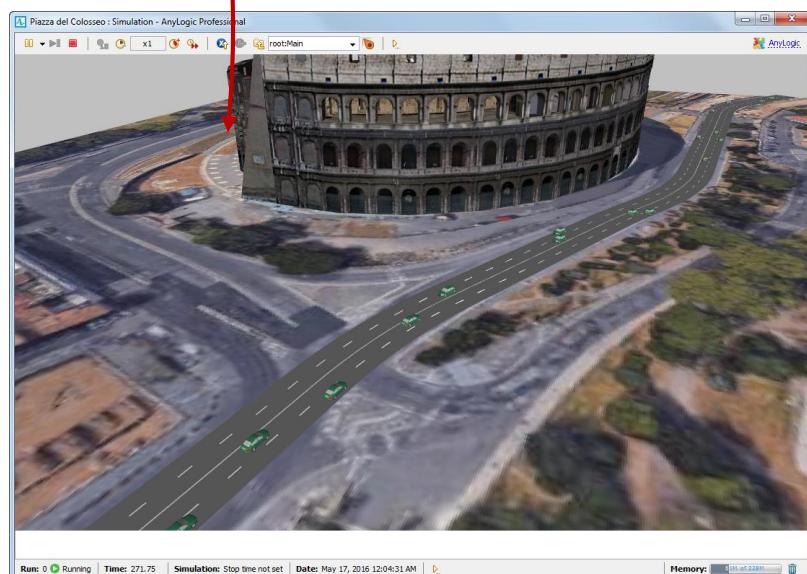
Piazza del Colosseo. Phase 2. Step 8



The screenshot shows the AnyLogic Professional interface with three overlapping windows:

- ① Run the model**: A callout pointing to the Logic view.
- ② Select [Logic]**: A callout pointing to the Logic button in the toolbar.
- ③ Select [window3d]**: A callout pointing to the window3d button in the toolbar.

A red arrow points from the Logic view down to the AnyLogic Professional simulation window below, which displays a 3D rendering of the Colosseum and surrounding roads.



The simulation window shows a 3D perspective of the Colosseum in Rome. The scene includes a road network with several cars in motion. The interface at the bottom provides simulation status information: Run: 0 (Running), Time: 271.75, Simulation: Stop time not set, Date: May 17, 2016 12:04:31 AM, and Memory: 311 of 2388 MB.

Here we switch between the view areas that display the model logic, 2D and 3D animation

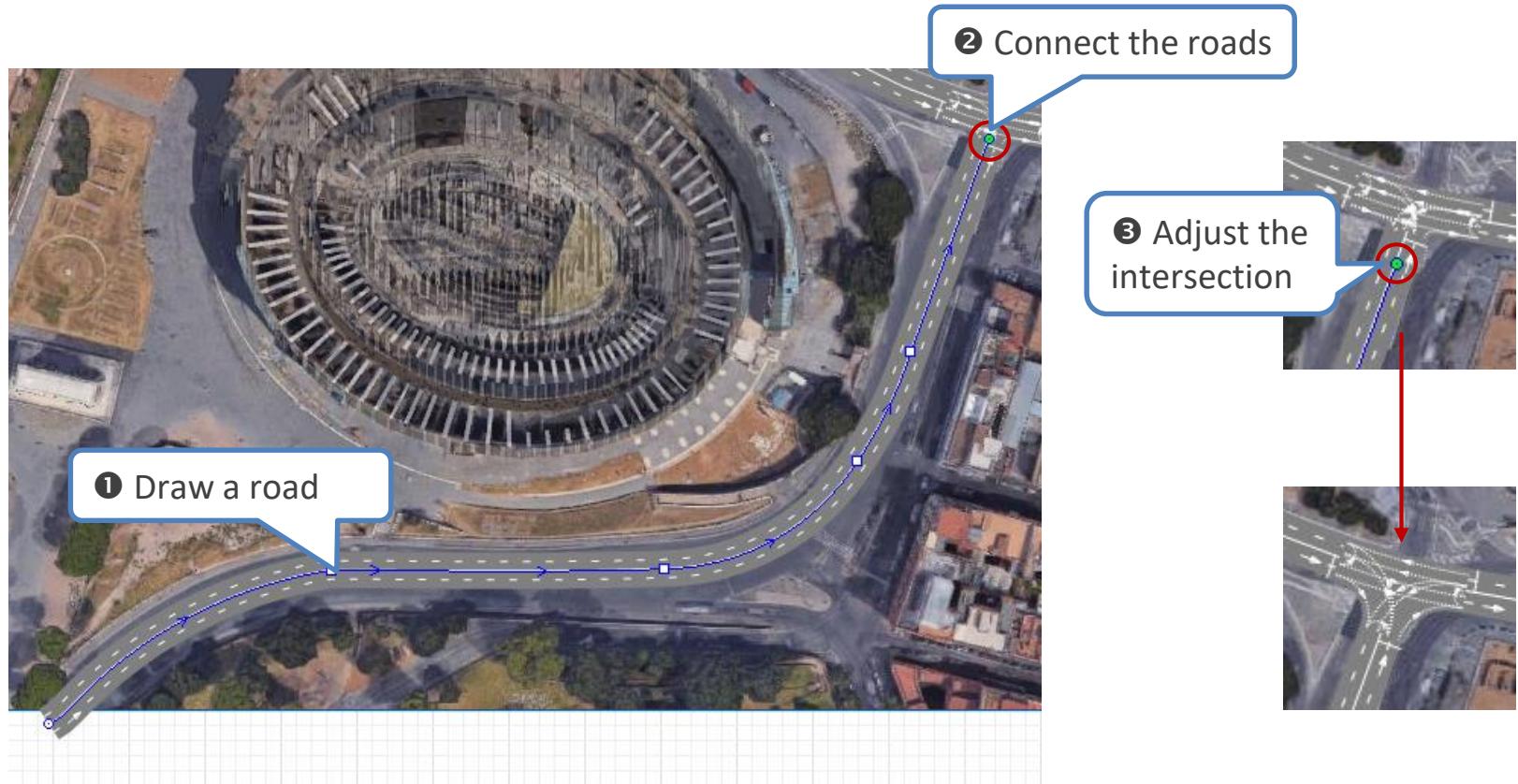


Piazza del Colosseo. Phase 3

- In this phase, we will add two more roads, set up the intersection between them and define cars movement on the intersection



Piazza del Colosseo. Phase 3. Step 1

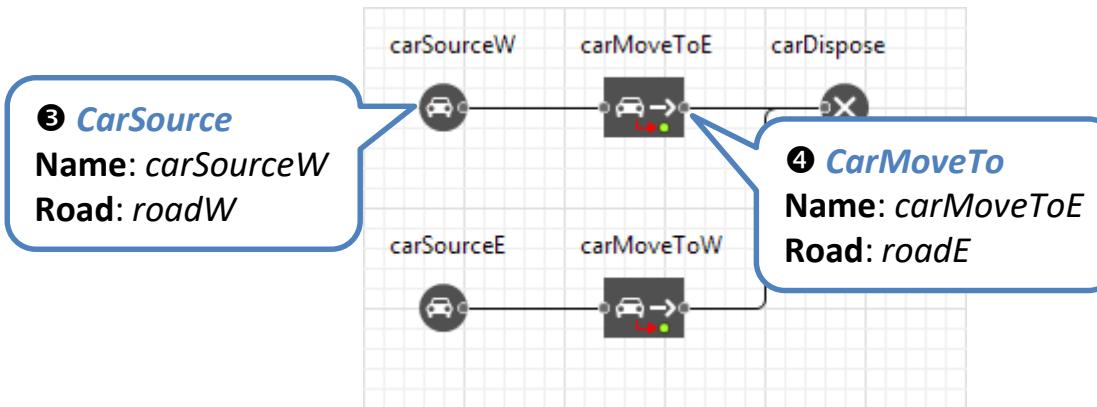
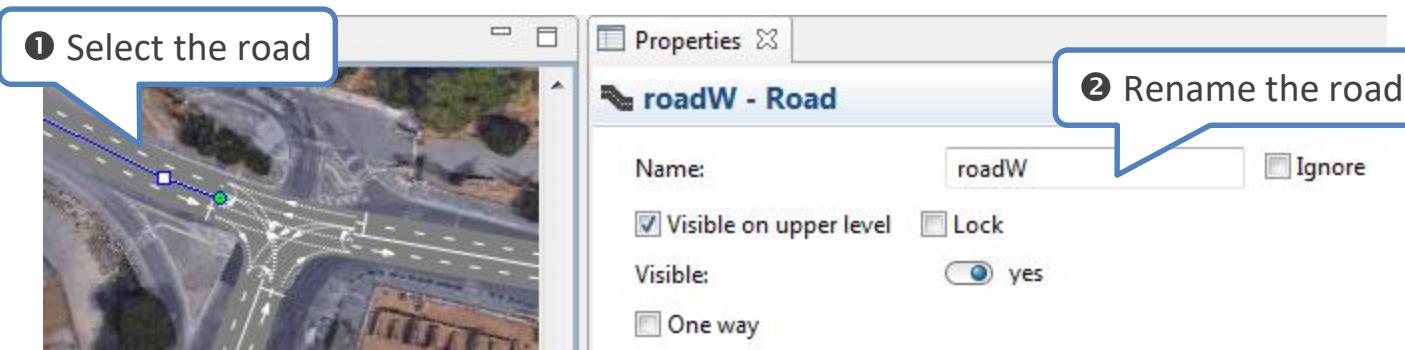


We will now add a second road in the South-to-North direction (Via Celio Vibenna) and create an intersection connecting both roads.

- ① Draw a new road (as displayed on the screenshot).
- ② Create an intersection by connecting the roads. To connect roads, create the end point by double clicking when the point is cyan-colored.
- ③ Adjust the shape of the intersection so that it better fits the layout. Drag the connection point farther from the intersection. Note that you should release the mouse button while the connection point cyan-colored. If the connection point becomes white-colored, releasing the mouse will disconnect the road from the intersection.
If you accidentally disconnect the road, press *Ctrl+Z* to undo the action.



Piazza del Colosseo. Phase 3. Step 2



When we added the intersection, our existing West-to-East road was split into two separate roads. We will now rename the roads and make the appropriate edits in the flowchart.

- ①-② Select the road to the West of the intersection and rename it to *roadW*. Similarly, rename the second road to *roadE*.

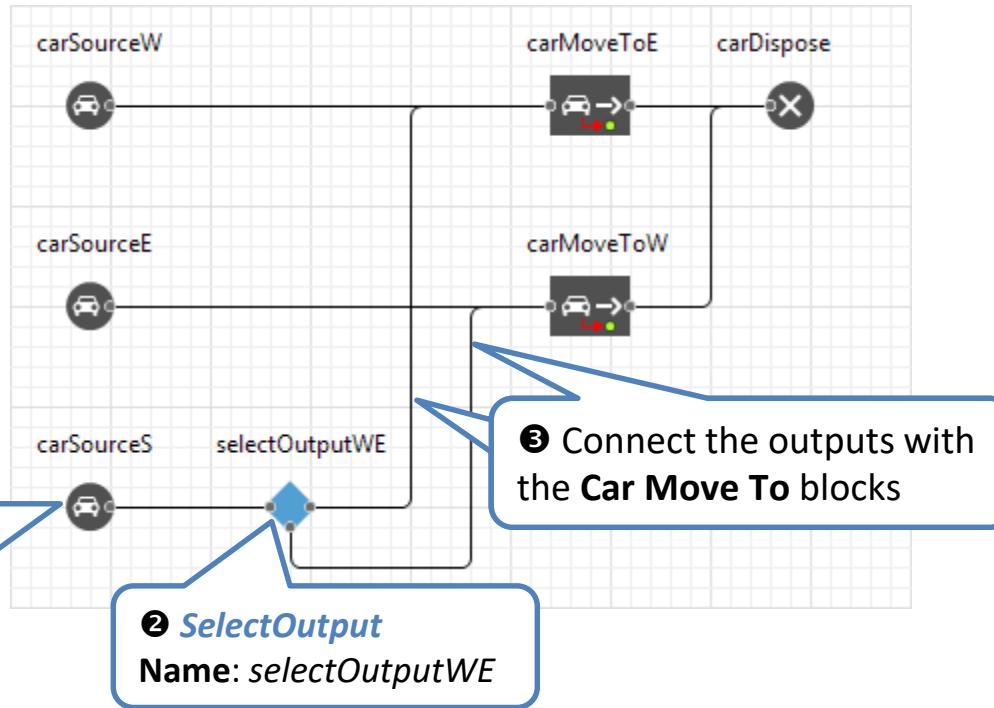
Note that since the road is now split into two, cars are generated at the beginning of one road and move to the end of the other road.

- ③ Rename the **Car Source** element to *carSourceW*. In the block properties, set **Road** to *roadW*.
- ④ Rename the **Car Move To** element to *carMovesToE* and set **Road** to *roadE*.

Perform the steps ③-④ for the second **Car Source - Car Move To** pair. In this case, cars will be generated at the beginning of *roadE* and move to the end of *roadW*.



Piazza del Colosseo. Phase 3. Step 3

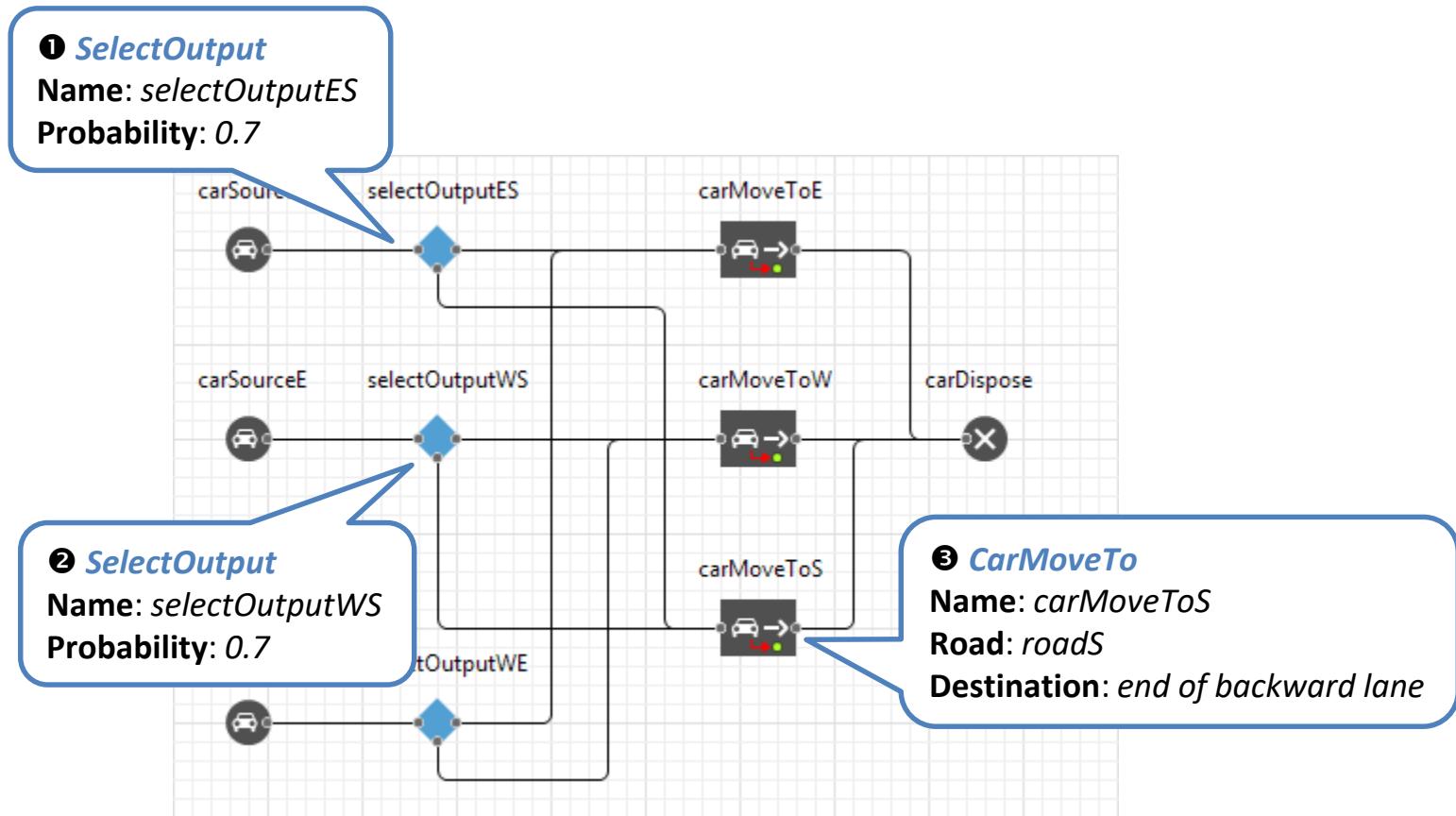


Our South-to-North road does not currently have any cars moving along it. We will now add the corresponding elements to the flowchart.

- ① Rename the South-to-North road to *roadsS*. Add a new **Car Source** element to the flowchart, which will generate cars at the beginning of the *roadsS* forward lane.
- ② Add a **Select Output** block and connect it to the **Car Source** element. We will use this block to split the single car flow (South-to-North) into two flows, each moving in a separate direction (to the East and to the West). We will leave the default **Probability** (0.5) so that the flows are approximately equal.
- ③ Connect the output ports of the **Select Output** block with the *carMoveToE* and *carMoveToW* input ports. Our cars moving from the South can now move in the eastern and western directions.



Piazza del Colosseo. Phase 3. Step 4



We will now add **Select Output** blocks to our flowchart so that some of the cars moving along the *roadE* and *roadW* will turn to *roadS*.

① Add a new **Car Move To** block and set it up so that cars move to the end of its backward lane.

②-③ Add two **Select Output** blocks, and connect them as follows:

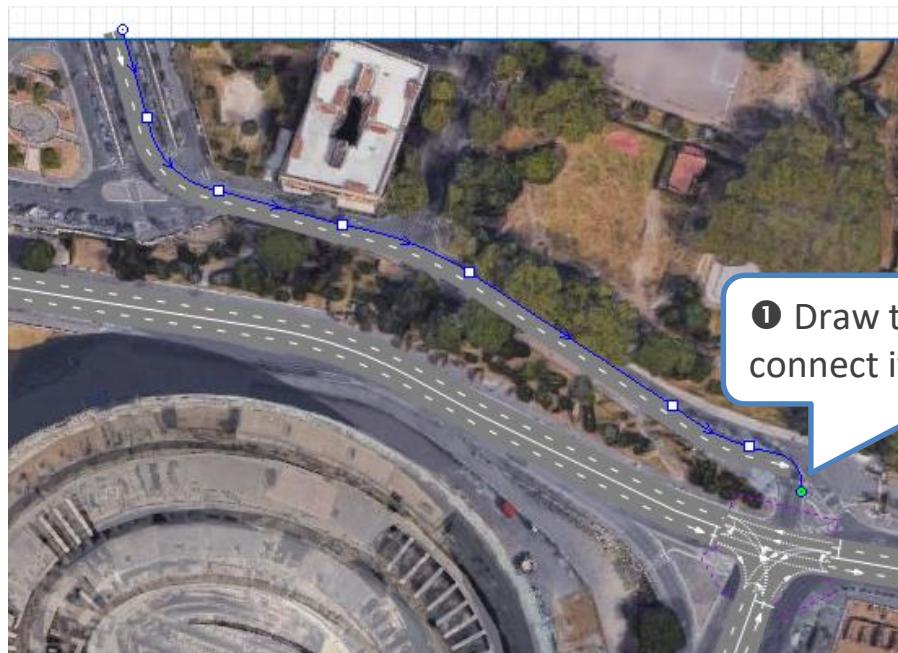
- *carSourceW* is connected with *carMoveToE* and *carMoveToS*
- *carSourceE* is connected with *carMoveToW* and *carMoveToS*

We assume that at intersections cars mostly do not turn but continue moving directly. For this reason, we will set **Probability** to 0.7 for both blocks.

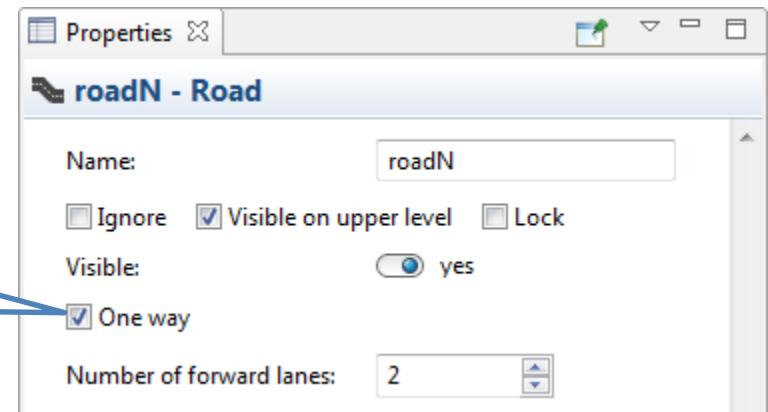
If you run the model now, you will see the cars moving according to the directions we have specified:



Piazza del Colosseo. Phase 3. Step 5



① Draw the road and connect it to the intersection



② Select One way



We will now add the last road (Via Nicola Salvi) and connect it to our road network.

① - ② Draw the road in the North-to-South direction (as displayed on the screenshot) and name it *roadN*. Set the road to be one-way.

When connecting the road to the intersection, release the mouse button as soon as the connection point is cyan-colored. By doing so, you will achieve the required shape of the intersection:

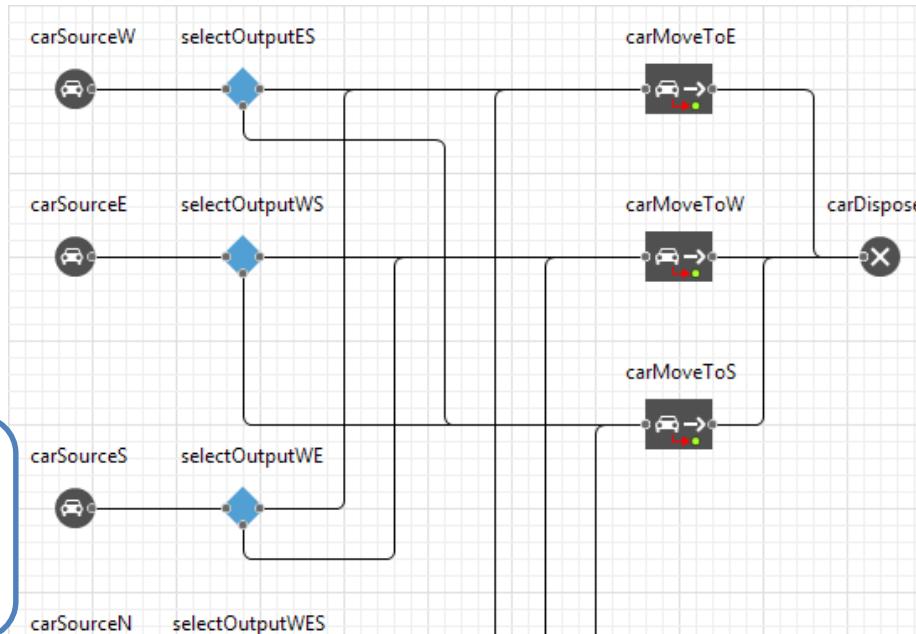


If you drag the connection point too far into the middle of the intersection, the intersection shape will be distorted:



Correcting this intersection will probably be time consuming.
We recommend that you press **Ctrl+Z** (Mac OS: **Cmd+Z**) to undo the action or select the road, delete it and draw a new one.

Piazza del Colosseo. Phase 3. Step 6



① Car Source

Name: `carSourceN`
Road: `roadN`
New car: `Car`

② Select Output

Name: `selectOutputWES`
Probability 1: 0.2
Probability 2: 0.2
Probability 3: 0.6
Probability 4: 0
Probability 5: 0



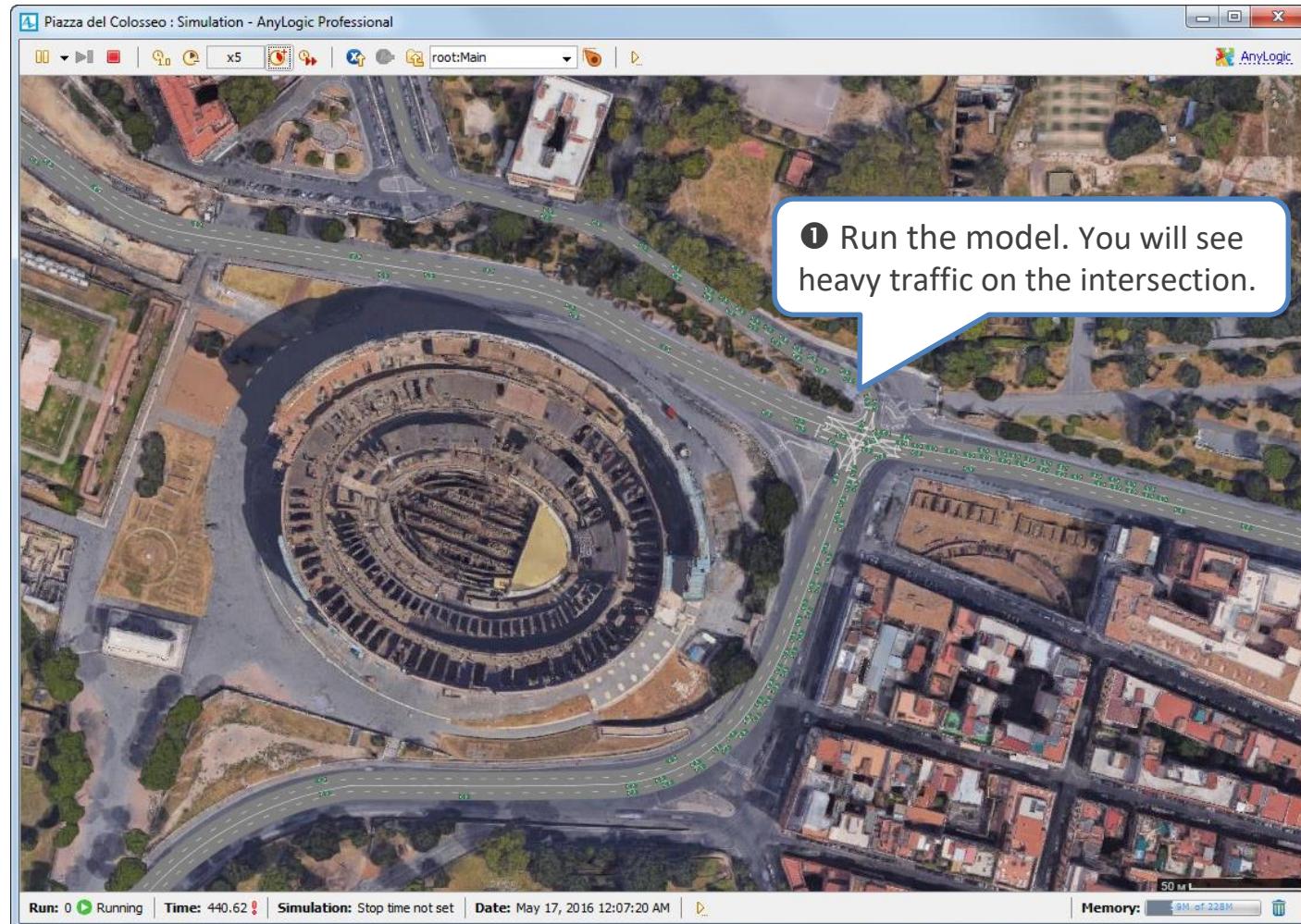
We will now adjust our flowchart so that it includes *roadN*.

- ① Add a new **Car Source** element, which will generate cars at the beginning of the *roadN* forward lane.
- ② Cars entering from the North can move in three different directions: to the West, to the East and to the South. For this reason, we add a **Select Output5** element instead of **Select Output**. We assume that the majority of cars do not change the initial direction (North-to-South) and set the appropriate probabilities.

Name the **Select Output5** element *selectWES* and connect its outputs to *carMoveToE*, *carMoveToW* and *carMoveToS*.



Piazza del Colosseo. Phase 3. Step 7

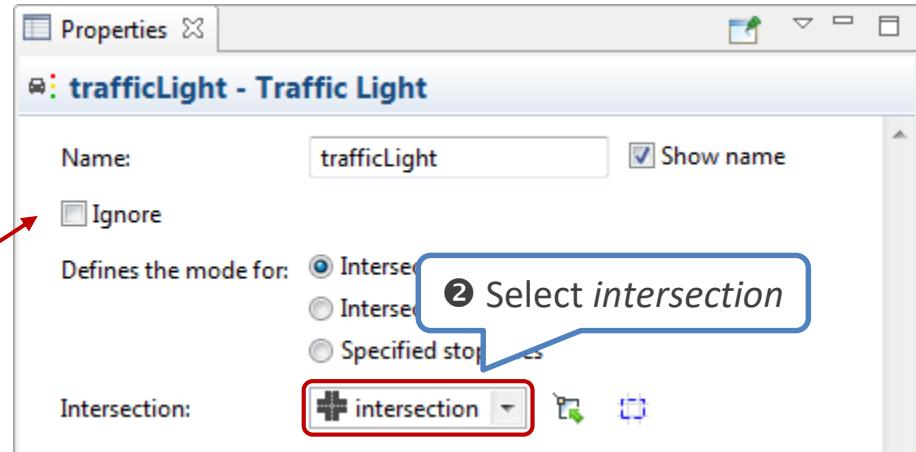
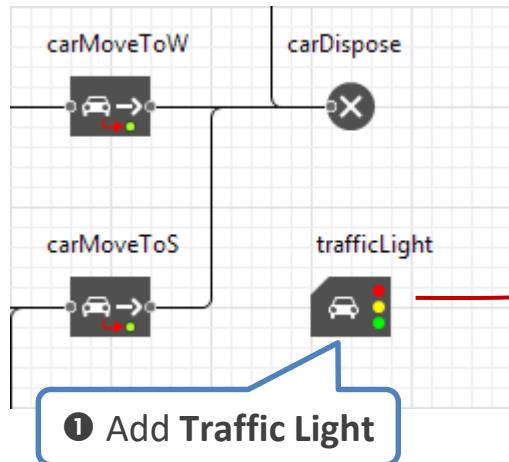


Piazza del Colosseo. Phase 4

- In this phase, we will make the intersection controlled by traffic lights.
- We will also measure time-in-system for our cars.



Piazza del Colosseo. Phase 4. Step 1



We will now make the intersection controlled by adding traffic lights.

- ① Add the **Traffic Light** element from the **Road Traffic Library** palette to the flowchart. You can position it freely since it is not connected to any other elements.
- ② Select *intersection* for the **Intersection** property to link this element to our intersection.



Piazza del Colosseo. Phase 4. Step 2

① Double-click and edit the phase duration

② Click the header

③ Click the stop line to change its state for the phase

Phases:	Durations (sec):	Stop lines:
	10	stopLine
		stopLine1
		stopLine2
		stopLine3

Phases:	Durations (sec):	Stop lines:
	15	stopLine
		stopLine1
		stopLine2
		stopLine3

Phases:	Durations (sec):	Stop lines:
	15	stopLine
		stopLine1
		stopLine2
		stopLine3

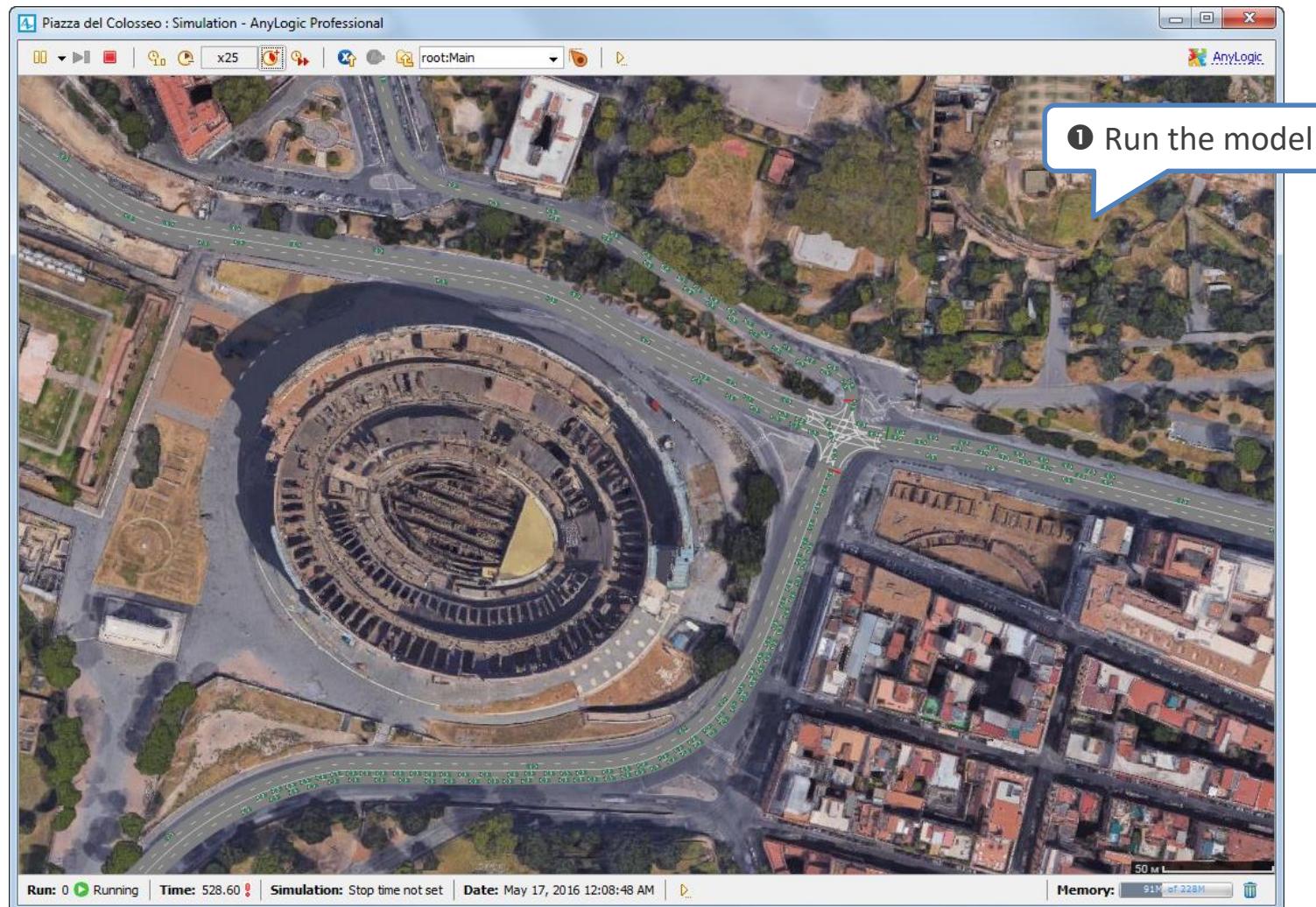
Phases:	Durations (sec):	Stop lines:
	15	stopLine
		stopLine1
		stopLine2
		stopLine3

We will now set up traffic lights phases.

- ❶ Double-click in the table header to adjust the phase duration. Set the first phase duration to 15 seconds and the second phase duration to 10 seconds.
- ❷ Click the table header to select the entire phase column. The graphical editor will switch to editing mode with all elements except the stop lines grayed out.
- ❸ Click the stop line to switch the traffic light phase. The stop line and the corresponding cell in the table will change its color. Set up phases in accordance with the following logic:
 - During the first phase, cars move in the East-to-West and West-to-East directions.
 - During the second phase, cars move in the North-to-South and South-to-North directions.



Piazza del Colosseo. Phase 4. Step 3



We have successfully created a controlled intersection.

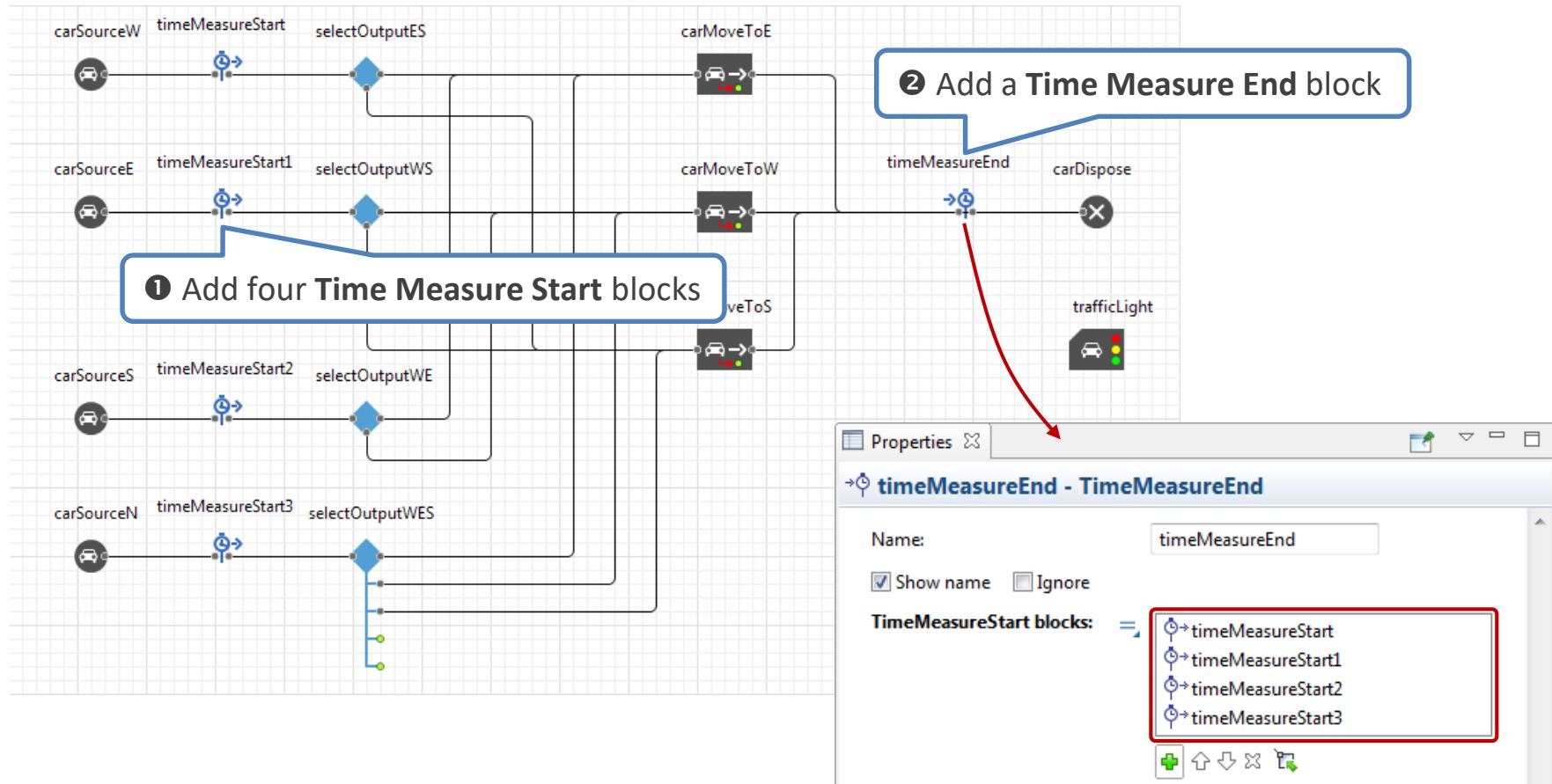
- ① Run the model. You will see traffic lights switching. If you increase execution speed, you will notice traffic jams. You may also notice that when such a jam occurs, cars stop entering a road after a certain time



This happens due to the fact that a car enters the road network only when there is enough distance for it to stop in accordance with its initial speed and maximum deceleration. The more initial speed of the car, the longer distance is needed for the car to safely enter the network.

Later we will adjust vehicles' initial speed for the *CarSourceE* block. We will also modify arrival rates based on historical data collected for the intersection.

Piazza del Colosseo. Phase 4. Step 4



Now we will measure the time that cars spend in our road network (*time in system*). We will do it by extending our flowchart with **Time Measure Start** and **Time Measure End** blocks.

- ① Add four **Time Measure Start** blocks. Place each block between the corresponding **Car Source** and **Select Output** block (as displayed on the screenshot). Ensure that all blocks are connected properly.
- ② Add a single **Time Measure End** block and place it between the **Car Move To** and **Car Dispose** blocks. Ensure that all three **Car Move To** blocks are properly connected to it. Select all four **Time Measure Start** blocks as the **TimeMeasureStart blocks** value.



Piazza del Colosseo. Phase 4. Step 5

The screenshot shows the AnyLogic modeling environment. On the left, the Analysis palette is open, displaying various data and chart types. A red arrow points from the 'Histogram' icon in this palette to the main agent diagram. The main diagram features a 'carMoveToS' agent and a 'trafficLight' resource. A histogram chart is placed on the workspace, showing the distribution of 'Time in system, seconds'. The x-axis ranges from 0 to 1 second with major ticks at 0, 0.2, 0.4, 0.6, 0.8, and 1. The y-axis shows percentages from 0% to 15%. The histogram bars are blue. A red arrow points from the chart to the Properties panel on the right. The Properties panel is titled 'chart - Histogram' and contains settings for the chart. A red box highlights the 'Title' field set to 'Time in system, seconds' and the 'Histogram' field set to 'timeMeasureEnd.distribution'. Another red box highlights the 'Add histogram data' button at the bottom of the panel. Three numbered callouts provide instructions:

- ① Drag the **Histogram** element from the **Analysis** palette to the Main agent diagram
- ② Click **Add histogram data**
- ③ Set the histogram title and data source



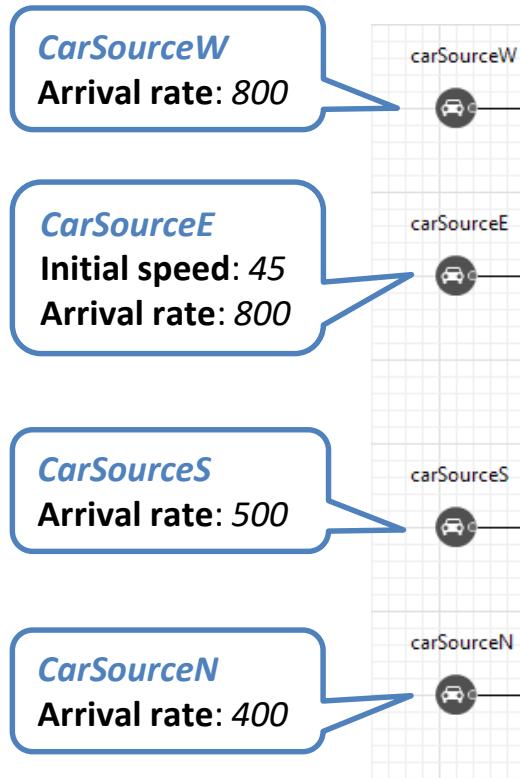
The **Time Measure End** block collects the distribution of the time spent by the agents in the marked part of the flowchart (starting from the **Time Measure Start** block).

We will now add a **Histogram** chart to view the cars' time-in-system distribution.

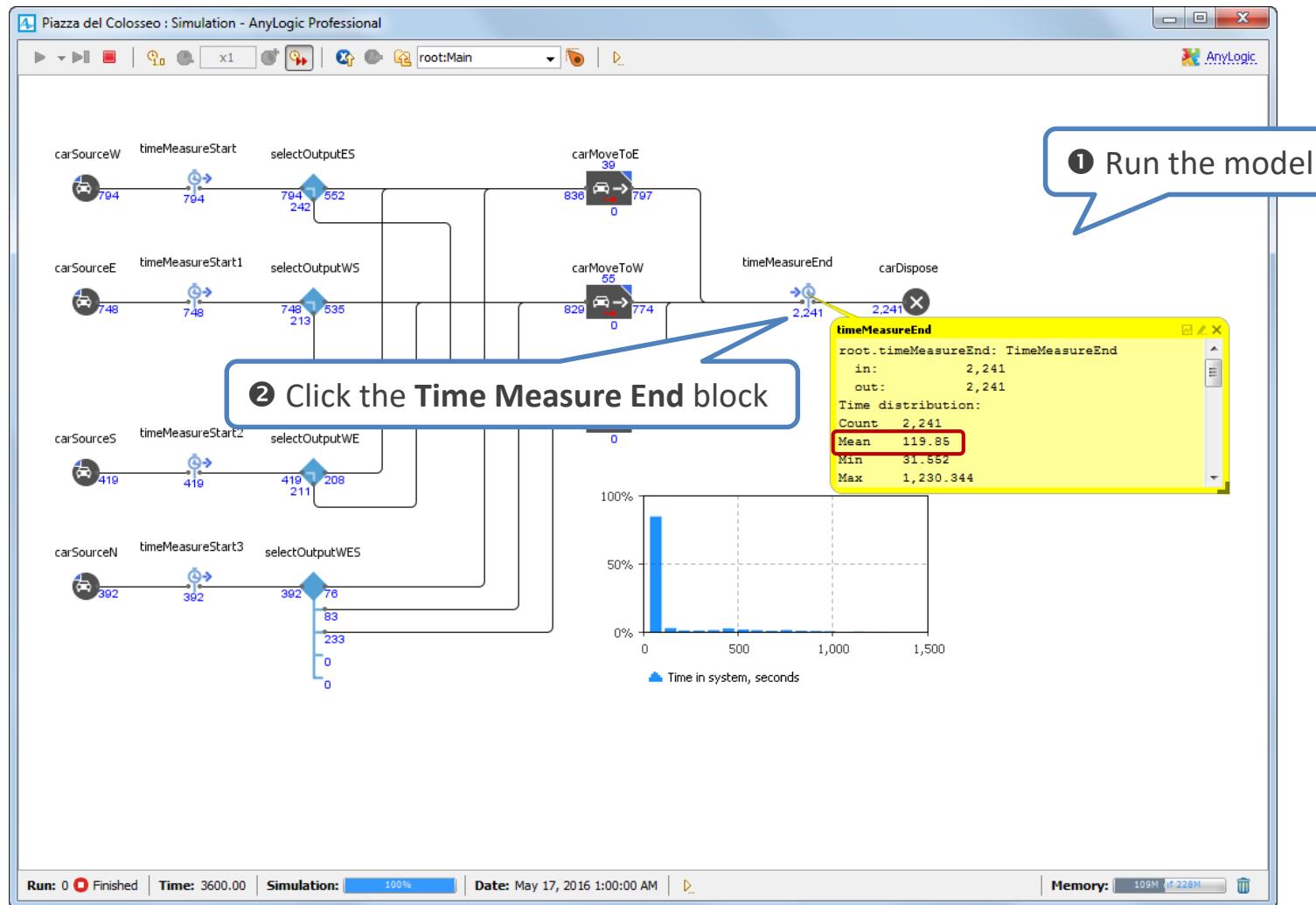
- ① Drag the **Histogram** element from the **Analysis** palette onto the Main agent diagram.
- ② The added histogram contains no data. Click the **Add histogram data** button to add a new data element.
- ③ Name the data element *Time in system, seconds* and set the **Histogram** field to *timeMeasureEnd.distribution*.



Piazza del Colosseo. Phase 4. Step 6



Piazza del Colosseo. Phase 4. Step 7



- ① Run the model and switch to the *[Logic]* view area. You will see the chart displaying the distribution of time in system for the cars at the intersection.
- ② Click the **Time Measure End** block to open the inspect window. Notice the *Mean* value (the mean of the cars' time in system) that is automatically calculated based on the data collected by the **Time Measure End** block.

In the next phase, we will create an optimization experiment, which will calculate the optimal durations for the traffic lights' phases so that time in system for the cars at the intersection is minimal.



Piazza del Colosseo. Phase 5

- In this phase, we will run the optimization experiment, which will comprise multiple simulation runs. During each run, the system will apply a different combination of the traffic lights phases' durations and measure the mean of the cars' time in system. The experiment goal is to achieve the minimum mean.



Piazza del Colosseo. Phase 5. Step 1

The screenshot shows the AnyLogic interface with three main windows:

- Agent palette (left):** Shows categories like Agent, Agent Component, Parameter, Event, and Dynamic Event. A red box highlights the "Parameter" icon.
- Main agent diagram (center):** Displays two car source blocks: "carSourceW" and "carSourceE". Each has a "timeMeasureStart" connector. A red arrow points from the "Parameter" icon in the palette to the "greenEW" parameter in the diagram.
- Properties panel (right):** For the "trafficLight - Traffic Light" object. It includes fields for Name (set to "trafficLight"), Show name (checked), Ignore (unchecked), and Defines the mode for (radio buttons for "Intersection's stop lines" and "Intersection's lane connectors").
 - Intersection section:** Shows "Duration (sec)" for "Stop lines".
 - Phases section:** Shows "Durations (sec)" for "Stop lines":
 - stopLine: greenEW (highlighted with a red box)
 - stopLine1: redEW
 - stopLine2: infinity
 - stopLine3: Choose Probability Distribution...A blue box labeled "④ Use the parameters for the phases' durations" points to the "greenEW" entry.

① Drag two Parameter elements from the Agent palette to the Main agent diagram

**② Name: greenEW
Type: int
Default value: 15**

**③ Name: redEW
Type: int
Default value: 10**

④ Use the parameters for the phases' durations



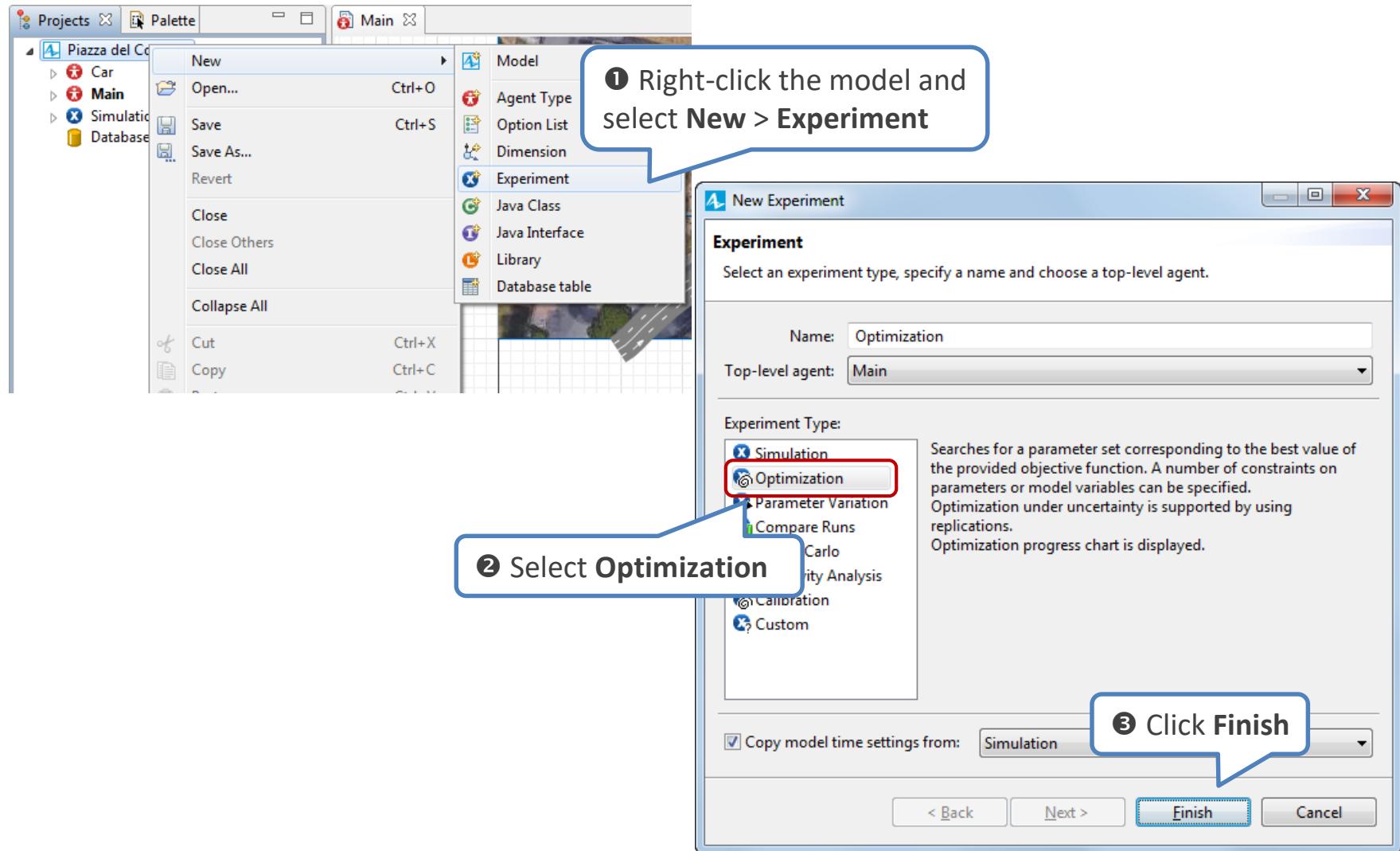
We will add two parameters to the model and assign them to the traffic lights' phases durations.

- ① Drag two **Parameter** elements from the **Agent** palette onto the Main agent diagram. Place these elements outside of the view area since we do not want to observe them during the simulation run.
- ②-③ Set the parameters' **Name** and **Default value**.
- ④ In the **Properties** view of the **Traffic Light** element, double click each column heading and choose the *greenEW* for the first phase and *redEW* for the second phase.

The phases' durations are now defined by parameters instead of constant values, which allows us to vary them during the experiment.



Piazza del Colosseo. Phase 5. Step 2



We will now create an optimization experiment.

- ❶ In the **Projects** view, right-click the model and select **New > Experiment**. The **New Experiment** wizard will open.
- ❷ Choose **Optimization** in the **Experiment Type** list and click **Finish**. The experiment will be created.

Optimization

- If you need to run a simulation and observe system behavior under certain conditions, as well as improve system performance (for example, by making decisions about system parameters and/or structure) you can use the optimization capability of AnyLogic. Optimization is the process of finding the combination of conditions resulting in the best possible solution.
- AnyLogic optimization is built on top of the OptQuest Optimization Engine, one of the most flexible and user-friendly optimization tools on the market. The OptQuest Engine automatically finds the best parameters of a model, with respect to certain constraints. AnyLogic provides a convenient graphical user interface to set up and control the optimization.
- The optimization process consists of repetitive simulations of a model with different sets of parameters. Using sophisticated algorithms, the OptQuest Engine varies controllable parameters from simulation to simulation to find the optimal parameters for solving a problem.



Piazza del Colosseo. Phase 5. Step 3

The screenshot shows the AnyLogic software interface with the Projects palette on the left and the Properties window on the right.

Projects Palette:

- Piazza del Colosseo
 - Car
 - Main
 - Simulation: Main
 - Optimization: Main** (selected)
 - Database

Properties Window - Optimization - Optimization Experiment:

- ① Select Optimization: Main** (points to the selected item in the Projects palette)
- ② Define the objective function** (points to the Objective field: `root.timeMeasureEnd.distribution.mean()`)
- ③ Adjust the number of iterations** (points to the Number of iterations field: 200)
- ④ Set up the experiment parameters** (points to the Parameters table)

Parameter	Type	Value	Min	Max	Step	Suggested
greenEW	int	10	40	1	15	
redEW	int	10	40	1	10	
- ⑤ Adjust Stop time** (points to the Stop time field: 3600)

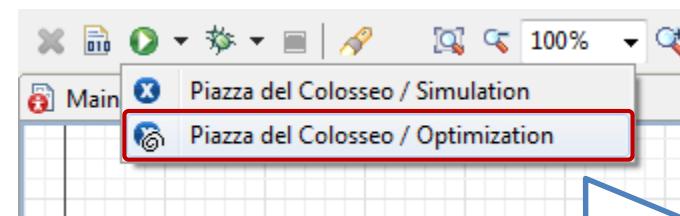
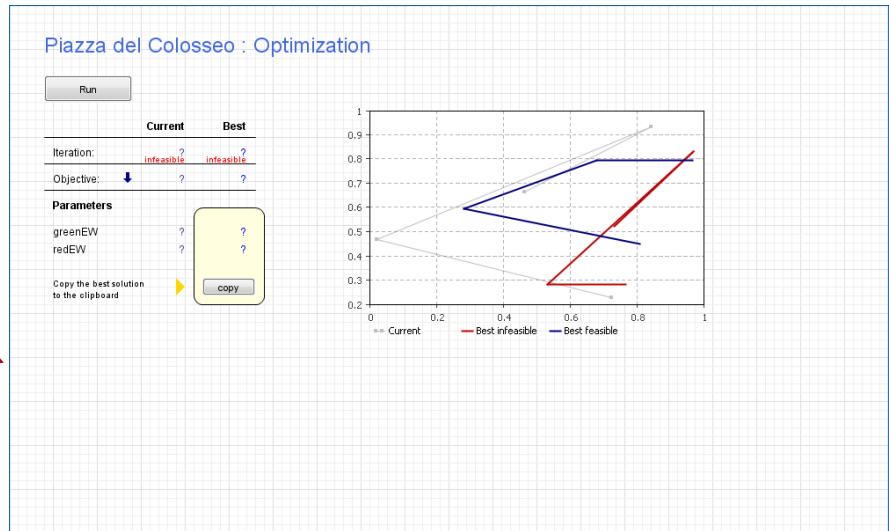
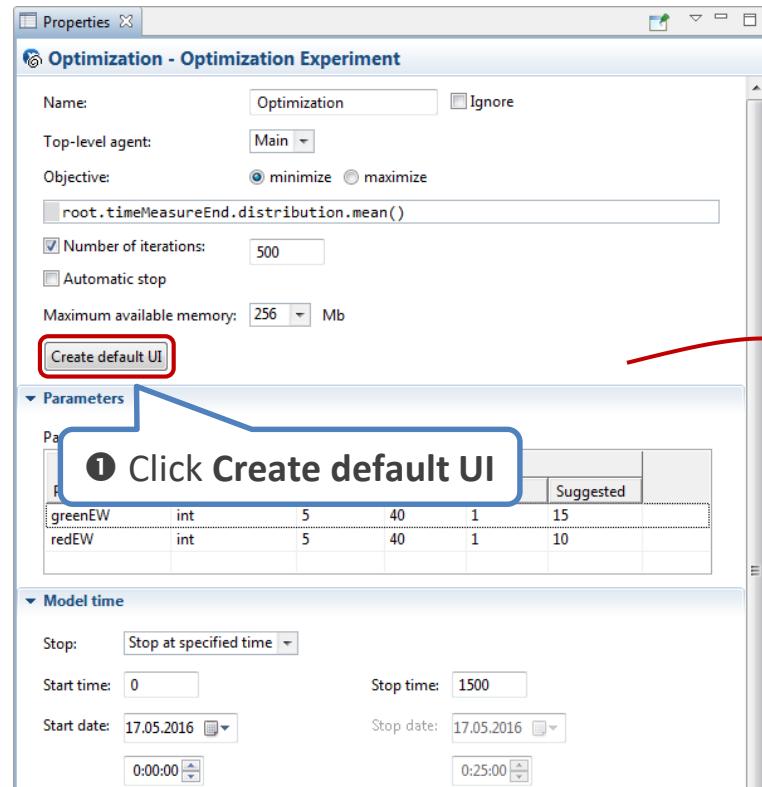


We will now set up the created experiment.

- ① Select *Optimization: Main* in the **Projects** view.
- ② Define the objective function, which is the value that we want to minimize as a result of the experiment. In our case, it is the mean of the cars' time in system. This value is provided by the **Time Measure End** block defined in the top-level agent of the experiment. Since top-level agent is accessed as *root* in experiment's code, we type here:
root.timeMeasureEnd.distribution.mean().
- ③ Set the **Number of iterations** to 200. We decrease the value so that the optimization process does not take too long. In business cases, the number of iterations is normally much higher (e.g. 1000).
- ④ The table in the **Parameters** section lists all parameters of the top level agent. Adjust the *greenEW* and *redEW* parameters as displayed on the screenshot. During the experiment run, the parameters will vary according to the provided values.
- ⑤ Set the experiment to simulate one hour and thus set the experiment **Stop time** to 3600 (since our model time units are seconds, this value defines the model lifetime in seconds).



Piazza del Colosseo. Phase 5. Step 4



We are ready to run the experiment.

- ❶ In the **Properties** view, click the **Create default UI** button. The default user interface for the experiment window will be created. It contains a number of controls displaying all the necessary information regarding the current optimization status.
- ❷ Click the arrow to the right of the toolbar **Run** button and choose the optimization experiment from the list.



Piazza del Colosseo. Phase 5. Step 5

The system will run multiple simulations using different combinations of the *greenEW* and *redEW* parameters. Here you will see statistics on current and best found objective and parameter values.

① After the experiment is finished, copy the found optimal parameter values to the clipboard

	Current	Best
Iteration:	200	196
Objective:	141.728	57.786
Parameters		
greenEW	25	28
redEW	10	23

Run: 200 Finished | Experiment: 100% | Simulations: Stop time not set | Memory: 117M of 255M | 565.7 sec



Piazza del Colosseo. Phase 5. Step 6

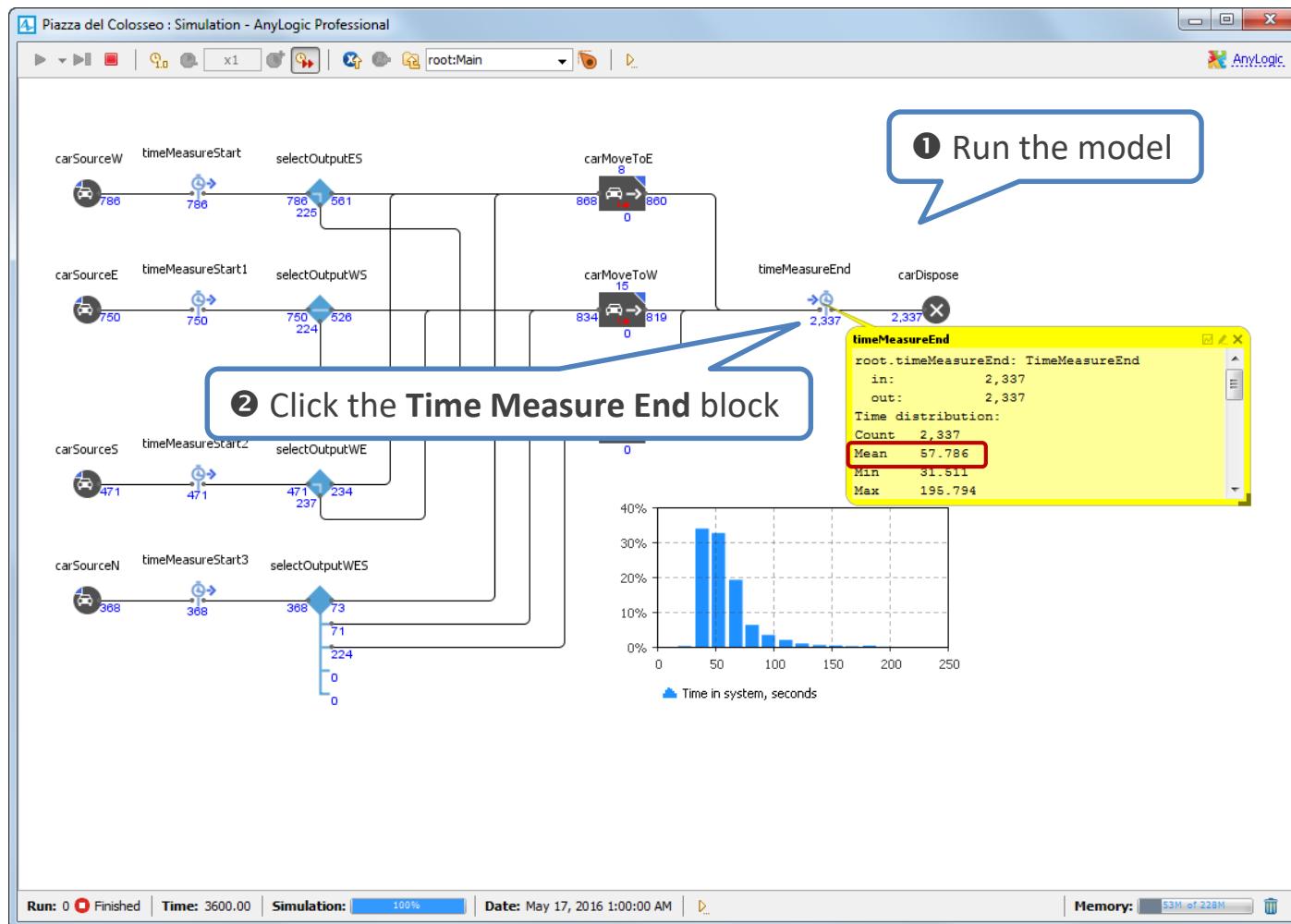
The screenshot shows the AnyLogic software interface. On the left, the **Projects** window displays a project named "Piazza del Colosseo" with several sub-items: "Car", "Main", "Simulation: Main" (which is highlighted with a red box), and "Optimization: Main". A blue callout box labeled "① Select Simulation: Main" points to the "Simulation: Main" item. A red arrow points from this selection to the **Properties** window on the right. The **Properties** window is titled "Simulation - Simulation Experiment". It contains the following settings:

- Name: Simulation
- Top-level agent: Main
- Maximum available memory: 256 Mb
- Parameters:
 - greenEW: = 28
 - redEW: = 23
- Paste from clipboard button

A blue callout box labeled "② Apply optimal parameter values to the simulation experiment" points to the parameter settings in the Properties window.



Piazza del Colosseo. Phase 5. Step 7



①-② Run the simulation experiment again and switch to the [Logic] view area. Click the **Time Measure End** block to open the inspect window.

Notice the difference in the shape of the histogram and the calculated *Mean* value before and after the optimization experiment:

