Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098.

A Practical Journal Submitted in fulfillment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2022-23

Part 2 Semester - 4

Subject Code -

Simulation & Modelling

BY

Ashwin Solaki

Application ID - 117833

(Seat No.-)

Table of Contents

Practical 1

AIM : Design and develop agent based model by

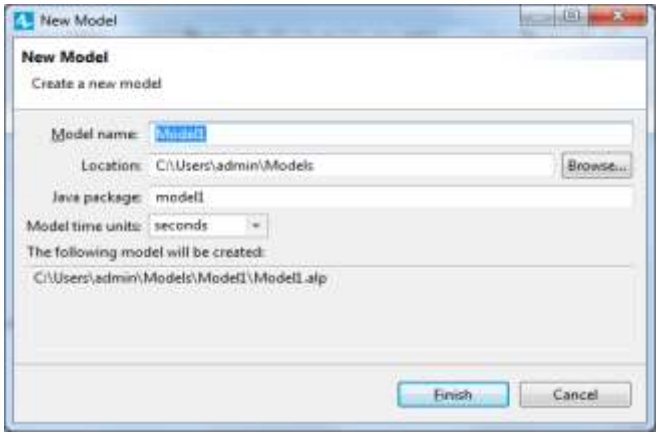
- Creating the agent population
- Defining the agent behavior
- Add a chart to visualize the model output.

[Use a case scenario like grocery store, telephone call center etc. for the purpose].

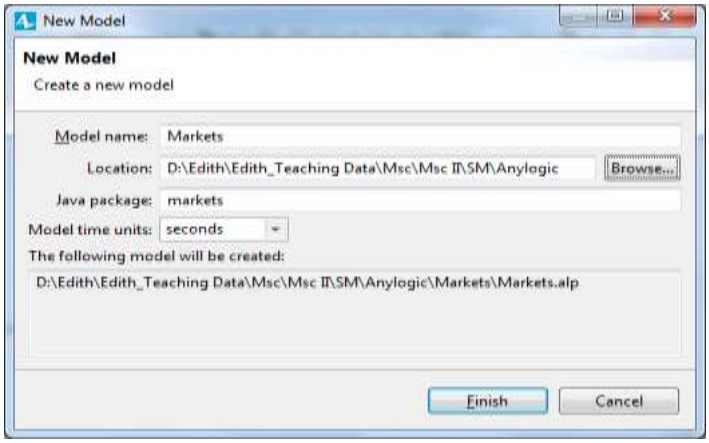
Solution:

Step 1 : Close the Welcome page, and create a new model by selecting **File** > **New** > **Model** from AnyLogic's main menu.

Step 2 : The **New Model** wizard will open.
In the **Model name** box, enter the new model's name: Market.



Step 3 : In the **Location** box, select the folder where you want to create the model. You can browse for a folder by clicking **Browse** or type the name of the folder you want to create in the **Location** box.

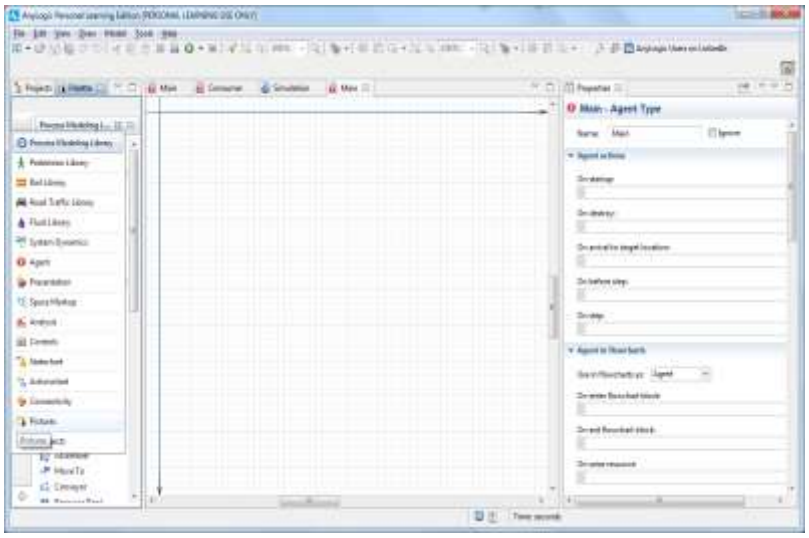


Step 4: Click Finish

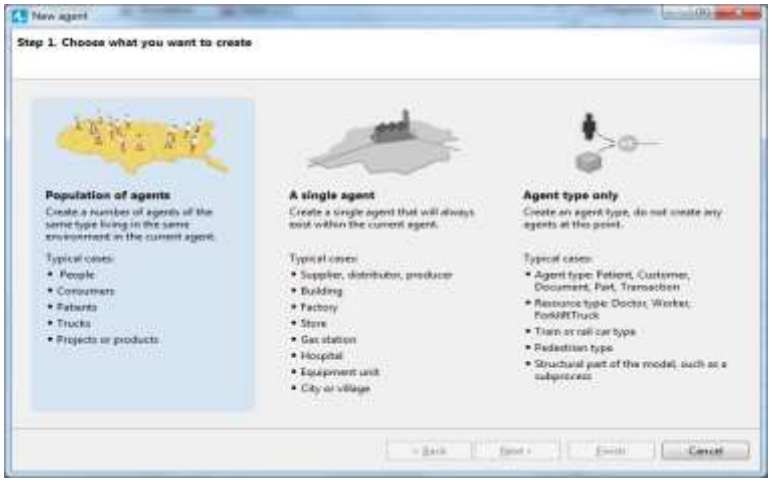
Creating the agent population :

Our model has one agent type, Main. To add consumers, we'll need to create an agent type to represent consumers, and then create an agent population made up of instances of this consumer agent type. With the help of **New agent** wizard agents can be created.

Step 1 : Click the **Palette** tab

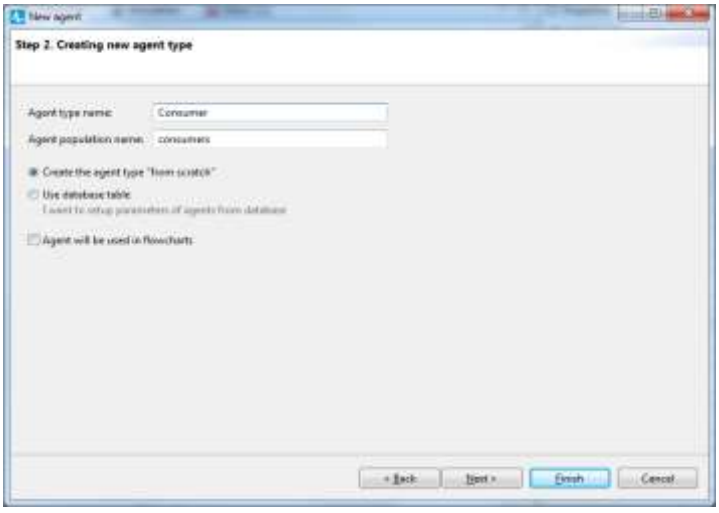


Step 2 : Drag the **Agent** from the **Agent** palette on to the Main diagram, and the **New agent** wizard will open.



Step 3 : Choose what you want to create page - select **Population of agents**, Since we want to create multiple agents of the same type and click **Next**.

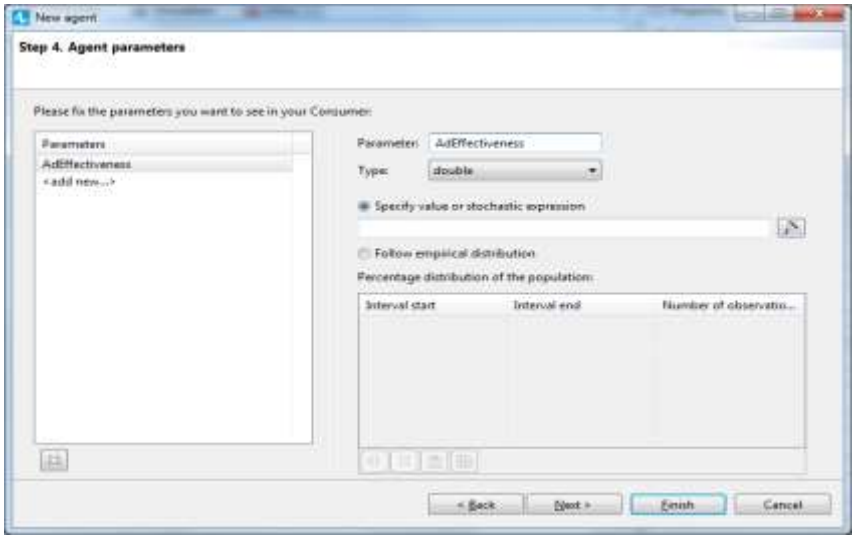
Step 4 : Creating new agent type page - in **Agent type name** box, type Consumer. The information in the **Agent population name** box will automatically change to consumers and click Next.



Step 5 : On the **Agent animation** page,

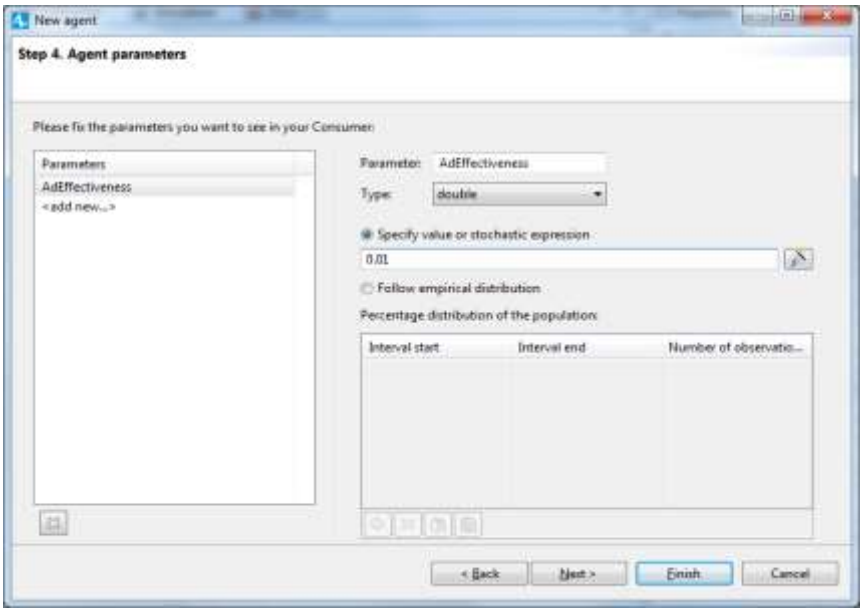
- choose the agent's animation shape - choose **2D**,
- select the **General** list's first item: **Person**, and click **Next**.

Step 6 : On the **Agent Parameters** page, define the agent's parameters or characteristics.
to create a parameter "AdEffectiveness" : On the left section, in the **Parameters** table, click **<add new…>**
AdEffectiveness - to define the percentage of potential users who become ready to buy the product during a given day.



Step 7 : In the **Parameter** box,

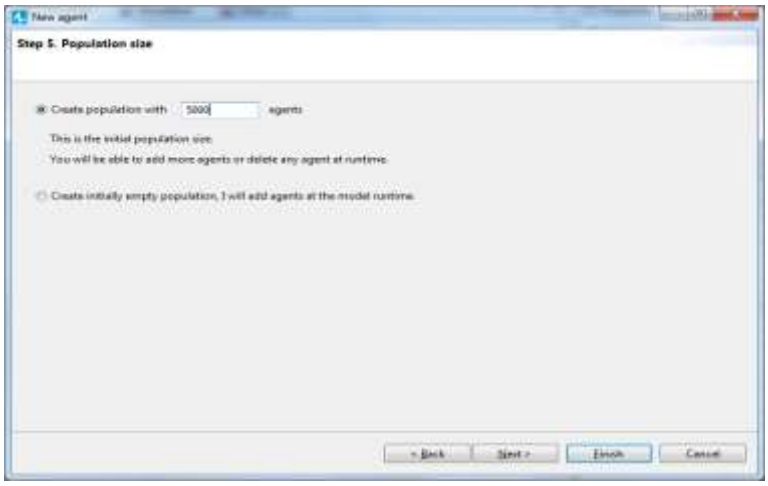
- change the default parameter's name to AdEffectiveness,
- choose **double** as the parameter **Type**.
- specify 0.01 as the parameter's value (We'll assume an average of 1% of our model's potential users will want to buy the product during a given day.)



Click Next.

Step 8 : On the **Population size** page

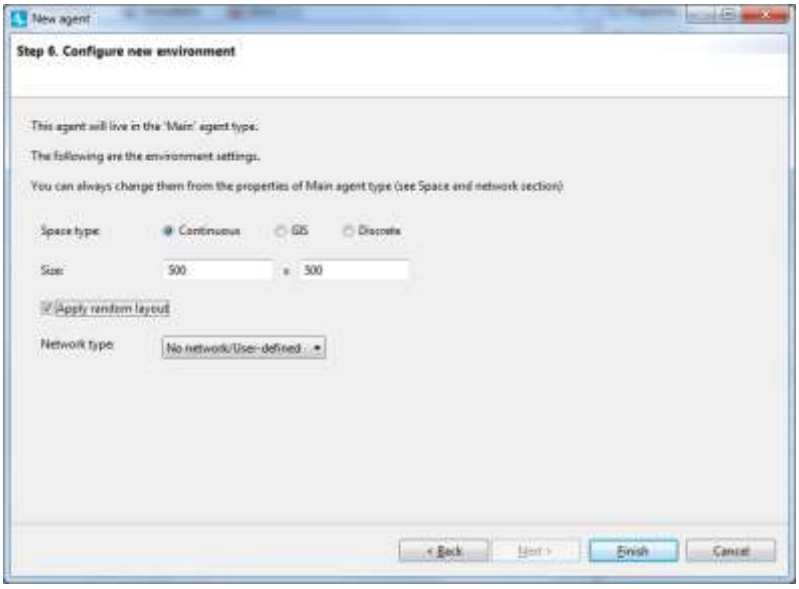
- type 5000 in the **Create population with agents** box - to create 5000 instances of the Consumer type. Each instance in the population will model a specific agent-consumer.



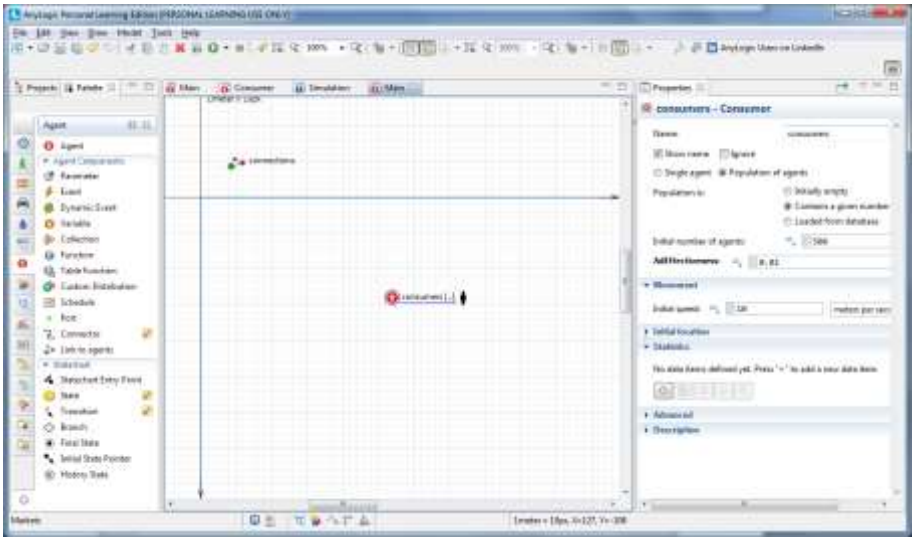
Click Next.

Step 9 : On the **Configure new environment** page,

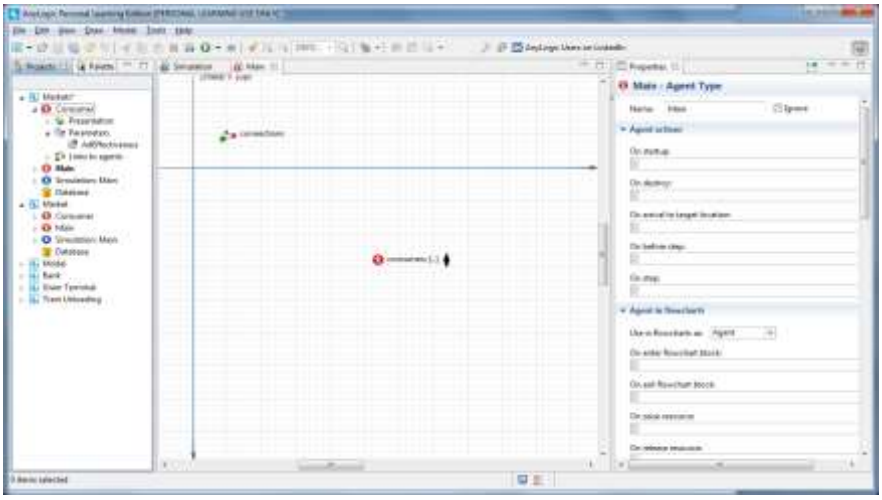
- accept the default values for the environment's space type(**Continuous**) and both its **Width** and **Height** values (500).
- Select the **Apply random layout** box to randomly distribute the agents
- accept the default **No network/User-defined** network type. (Since we don't want to create an agent network)



Click Finish.

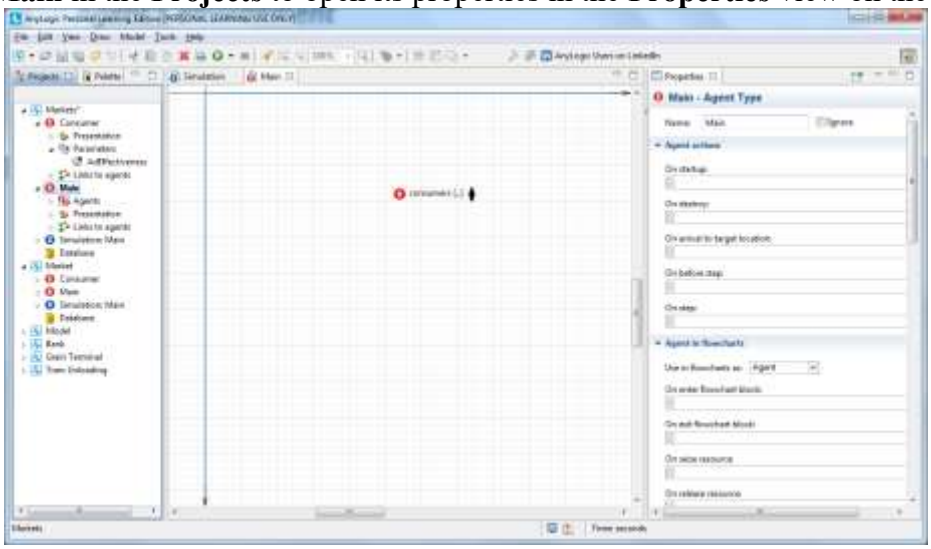


Step 10 : Click on **Project** tab – View and Expand the New Element created.

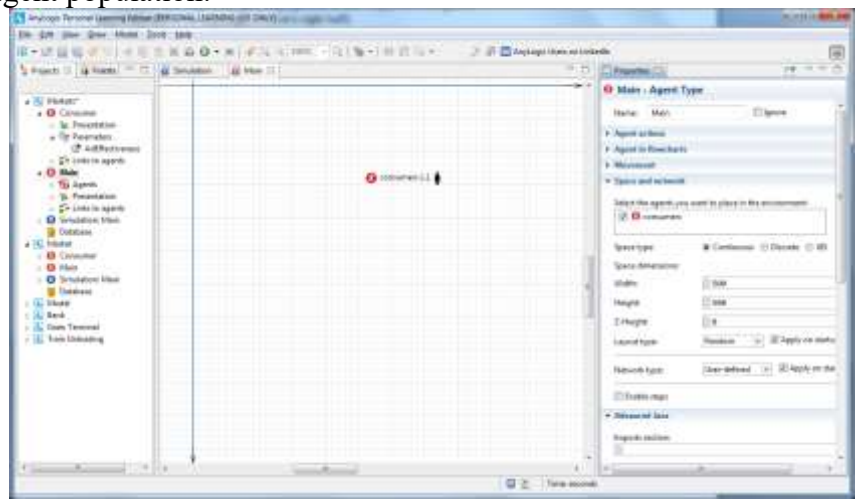


Our model now has two agent types: Main and Consumer.
• The Consumer agent type has the agent's animation shape (person, in the **Presentation** branch) and the parameter AdEffectiveness.
• The Main agent type contains the agent population consumers (set of 5000 agents of type Consumer).

Step 11 : Click **Main** in the **Projects** to open its properties in the **Properties** view on the right side.

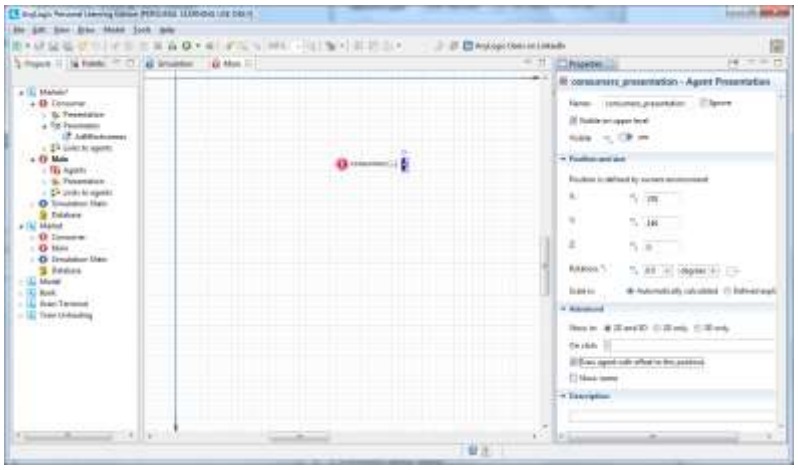


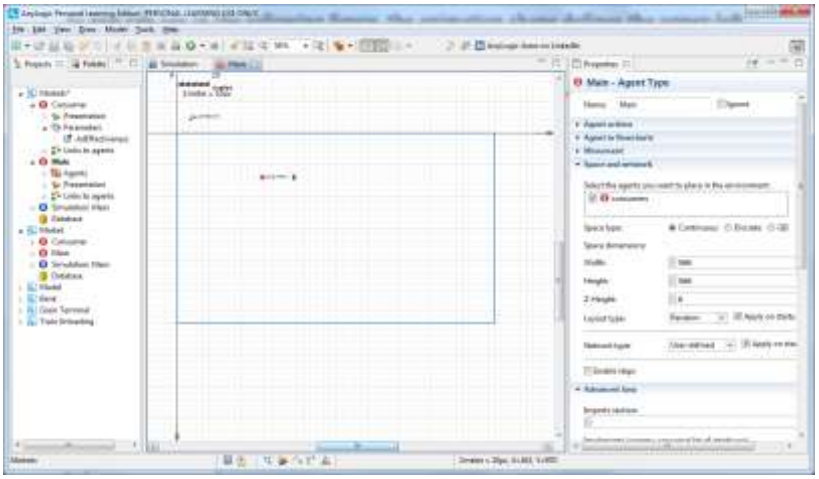
Step 12 : In the **Space and network** section of Main properties, you can adjust the environment settings for the consumers agent population.



Step 13 : On Main diagram,

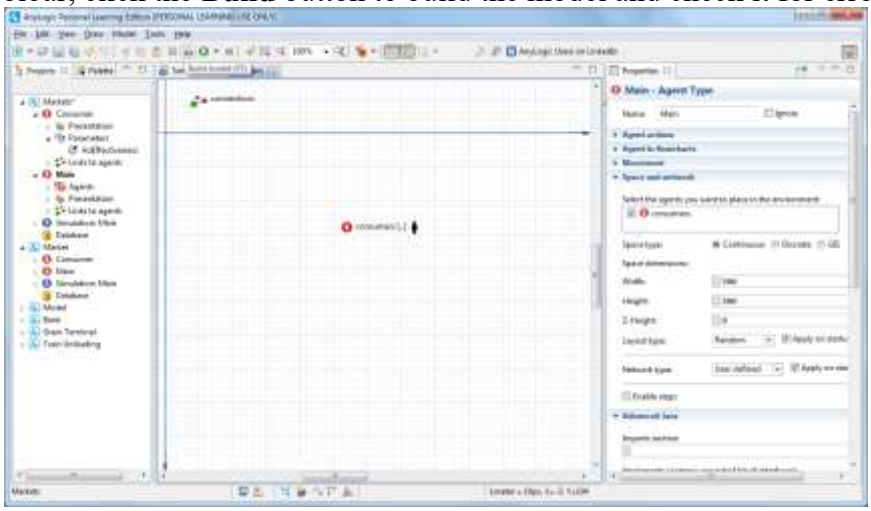
- select the agent population's non-editable embedded animation shape ,
- open the **Advanced** properties section, and
- select the **Draw agent with offset to this position** option.



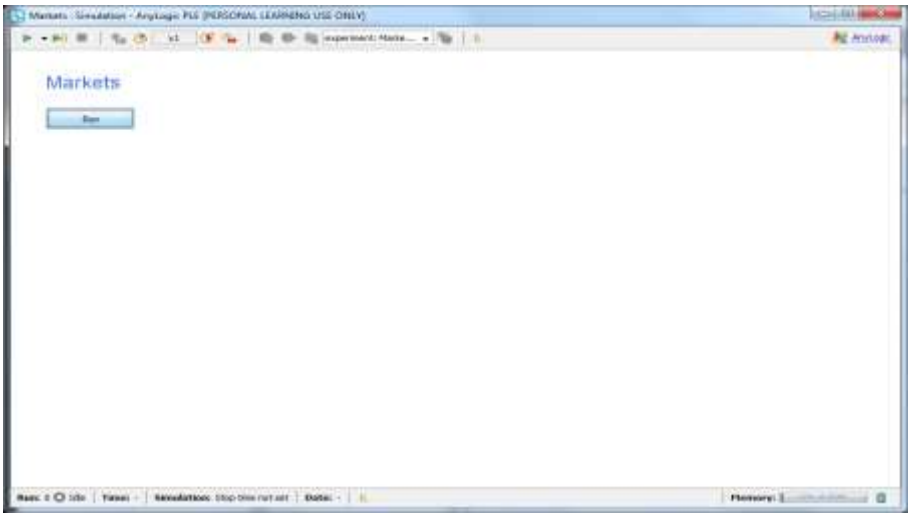


We've finished building this very simple model, and you can now run it and observe its behavior.

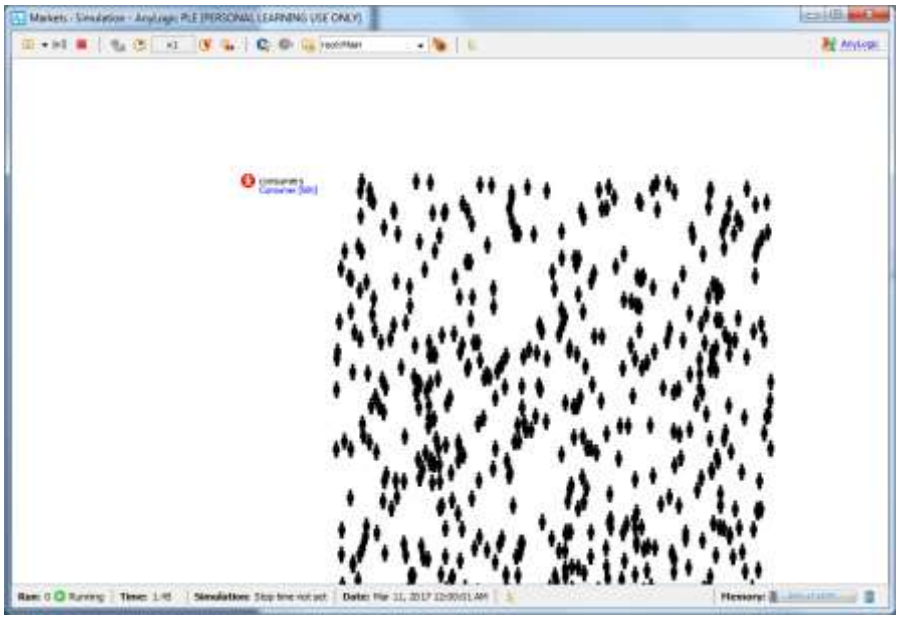
Step 14 : On the toolbar, click the **Build** button to build the model and check it for errors.



Step 15 : Locate the **Run** button, and click the small triangle to the right. Select the experiment you want to run. Choose **Markets / Simulation** from the list.



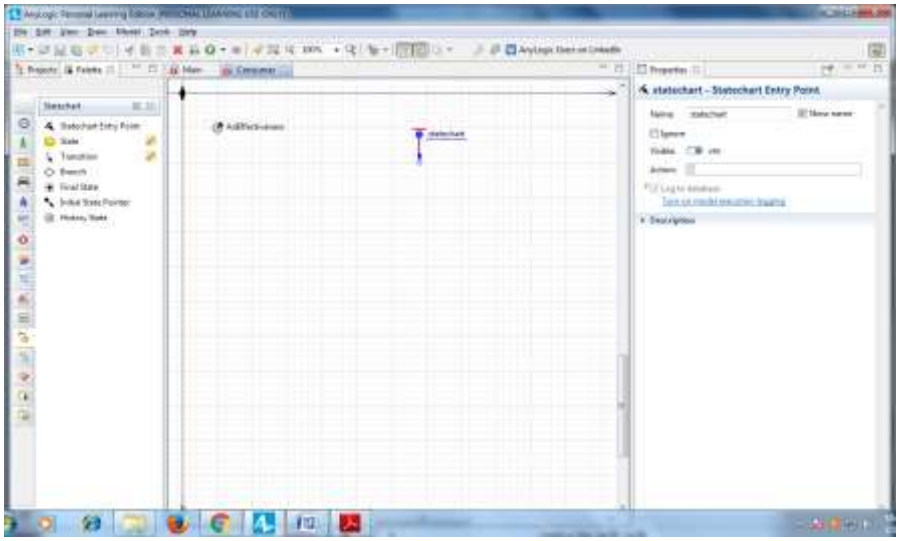
Step 16 : Click the **Run** button to run the model.



Defining a consumer behavior

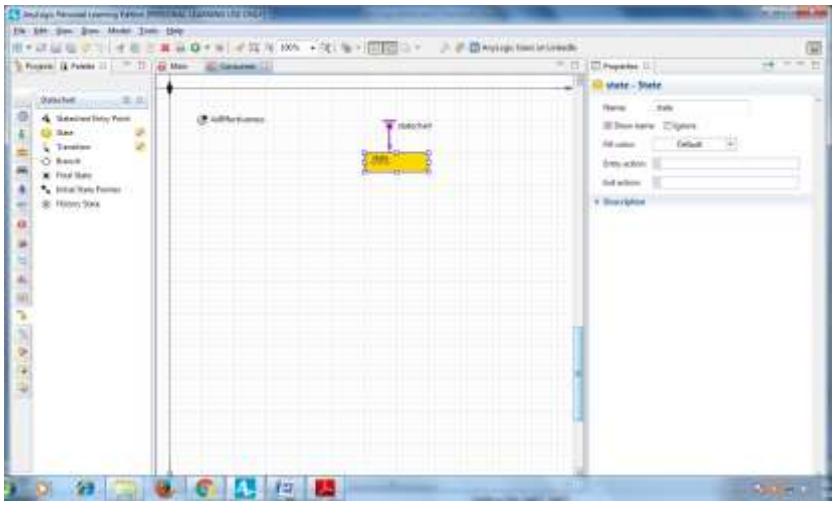
Step 1 : from the **Statechart** palette

- Drag the **State** on to the graphical diagram.
- connect it to the statechart entry point.

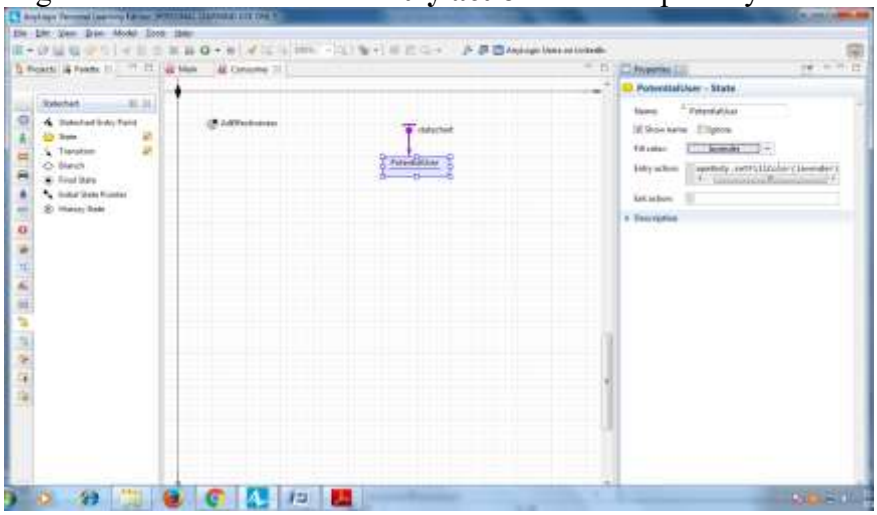


Step 2 : Select the state in the graphical editor, and modify its properties.

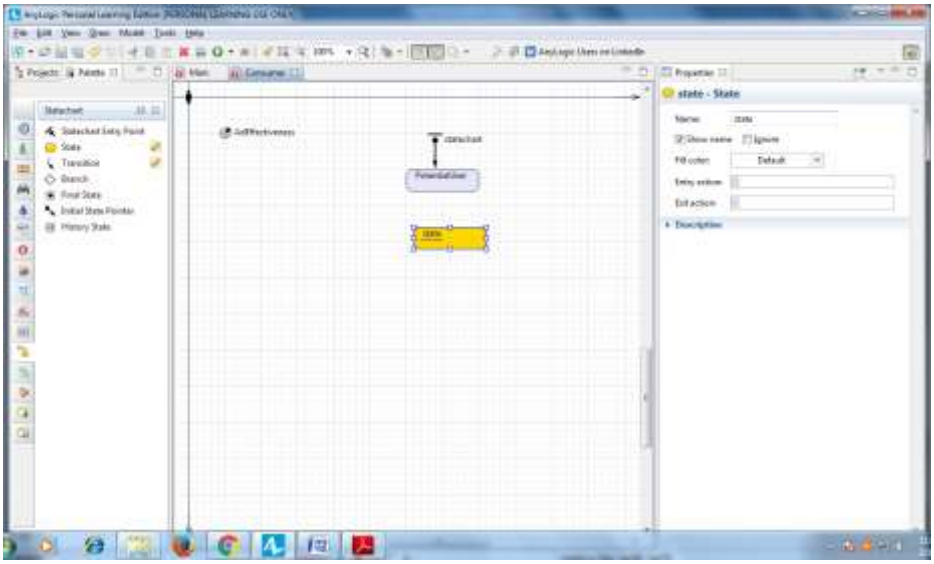
- Name the state PotentialUser.



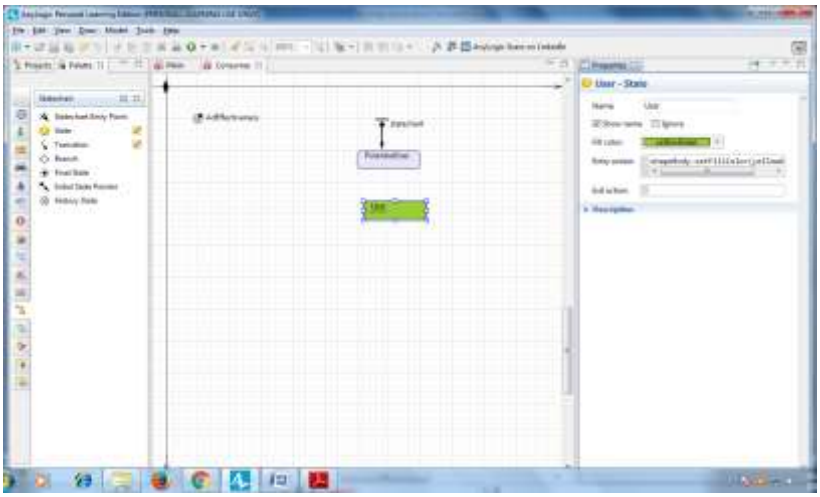
- Use the **Fill color** control to change the state's color to lavender
- Type the following Java code in the state's **Entry action** field: shapeBody.setFillColor(lavender)



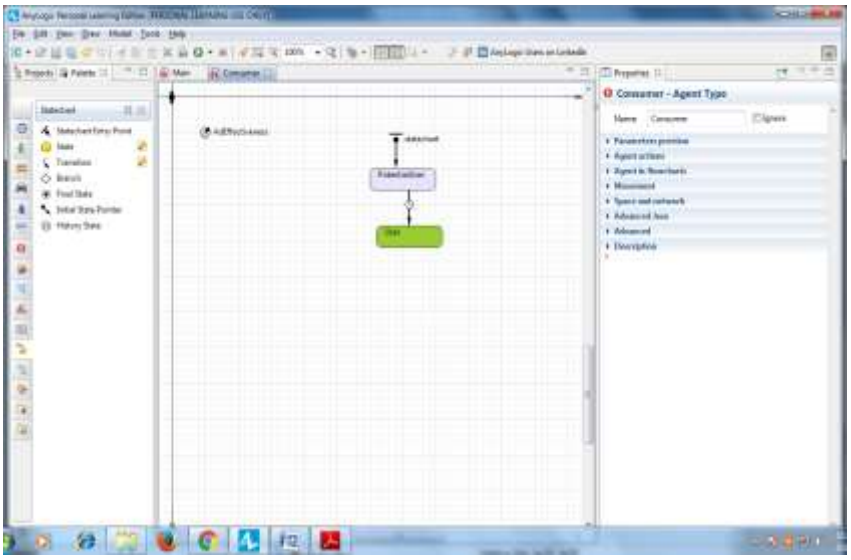
Step 3 : Add another state in the consumer's statechart:



Step 4 : Modify the state's properties like you did earlier:
Name: User
Fill color: yellowGreen
Entry action: shapeBody.setFillColor(yellowGreen)

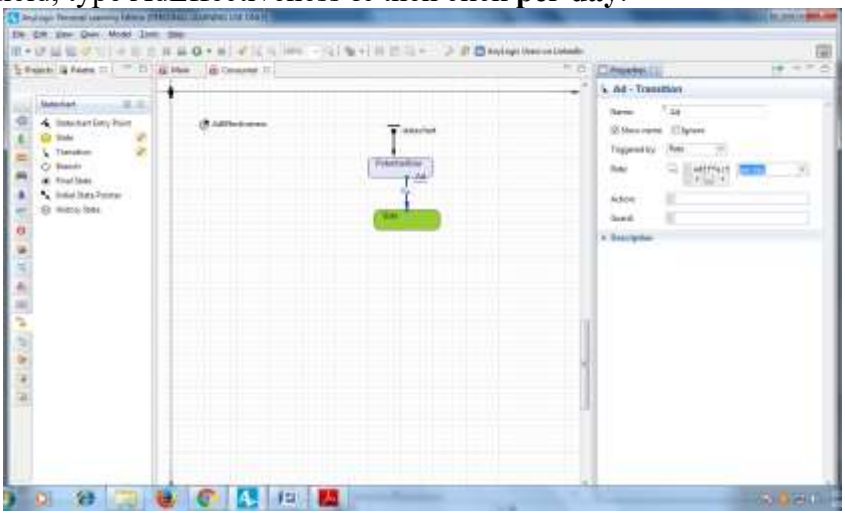


Step 5 : Draw a transition from PotentialUser to User
double-click the **Statechart** palette's **Transition** element - click PotentialUser state, and click User**.**



Step 6 : Transition Properties –

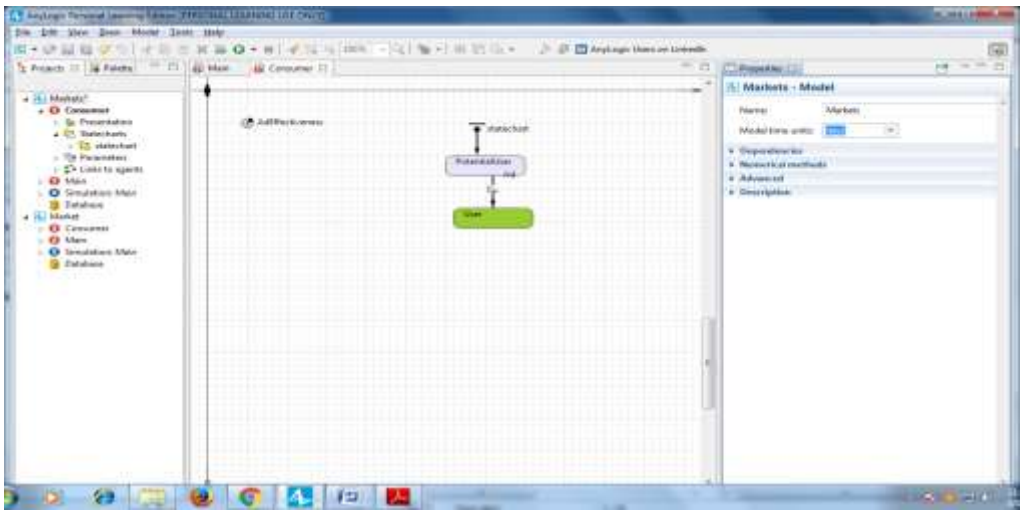
- Name the transition Ad to represent "advertising".
- Select the **Show name** checkbox to display the transition's name on the graphical diagram.
- In the **Triggered by** list, click **Rate**.
- In the **Rate** field, type AdEffectiveness & then click **per day**.



Step 7 : set up the model's time units.

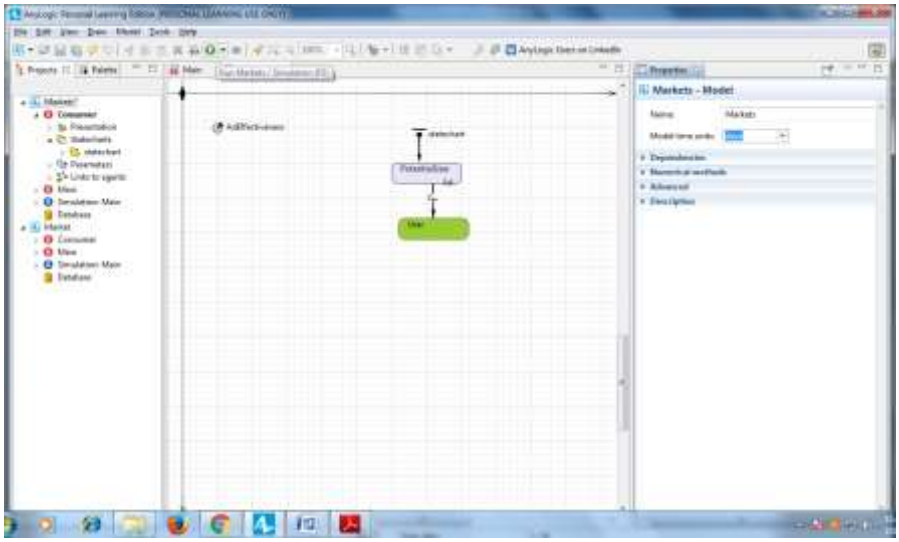
- switch from **Palette** to **Projects**
- then click the model item in the tree .

- In the **Properties** view, choose **days** as the **Model time units**.

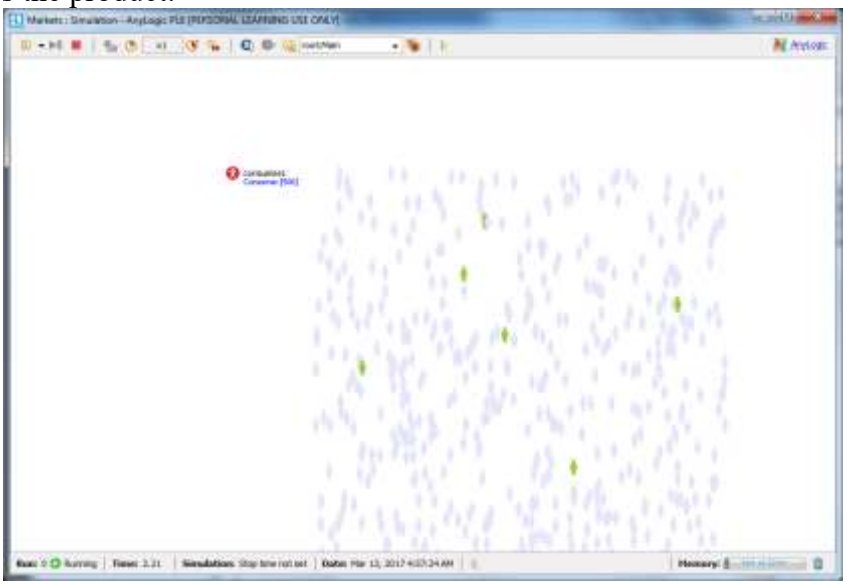


Step 8 : Run the Model .

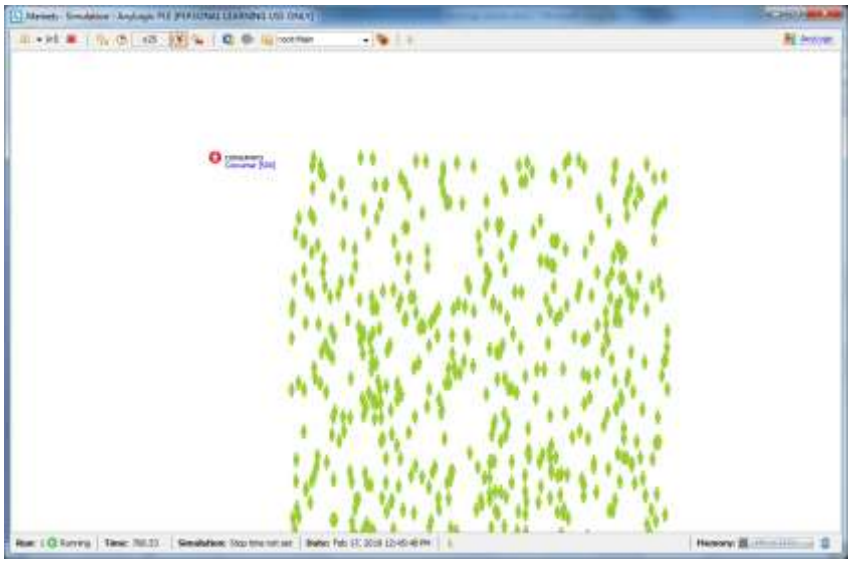
- Build
- Run



The population should gradually turn green – a change that represents the effect of advertising - until every consumer buys the product.

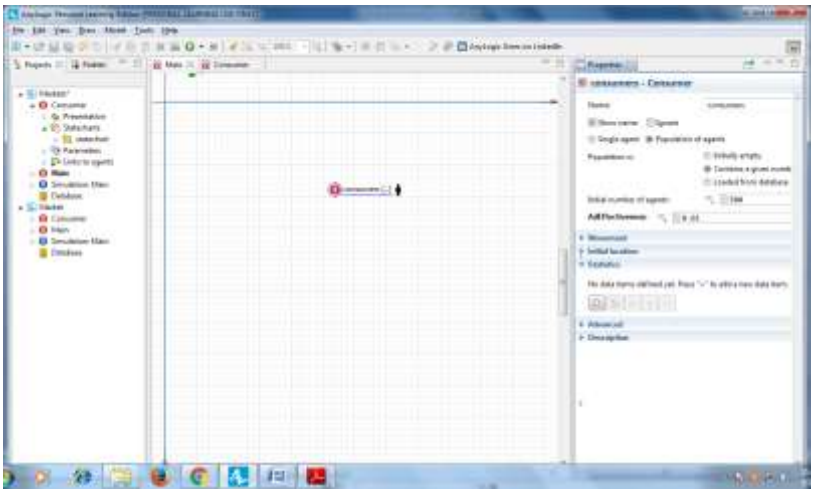


To adjust the model's execution speed, click the toolbar's **Slow down** or **Speed up** buttons. If you increase the speed to 10x – you'll see the speed at which the population turns green also increase.

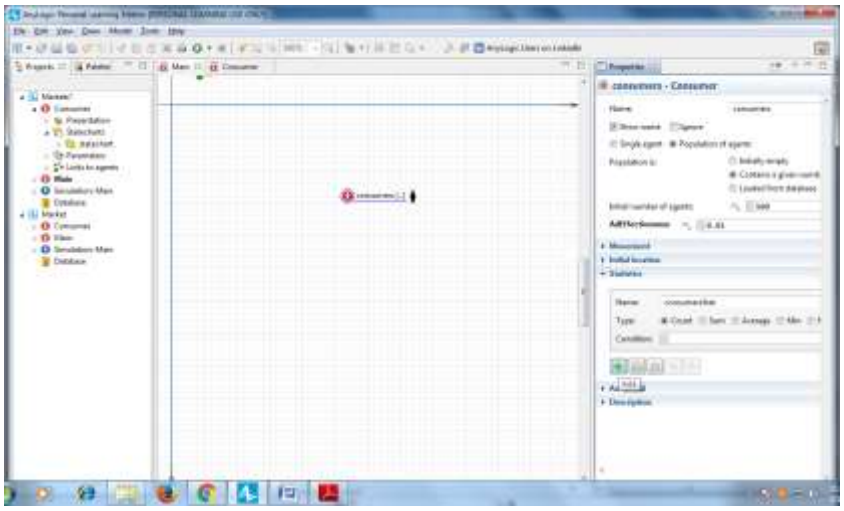


Add a chart to visualize the model output.

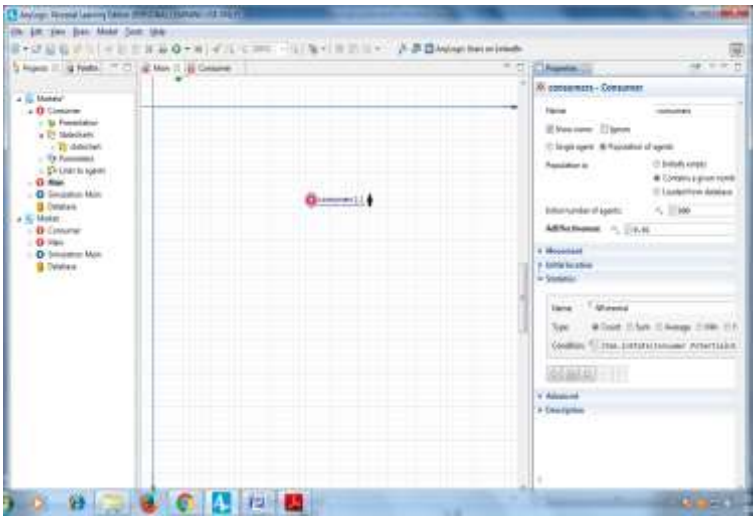
Step 1 : open the diagram of the agent type Main, select the agent population consumers, and go to the **Statistics** properties section.



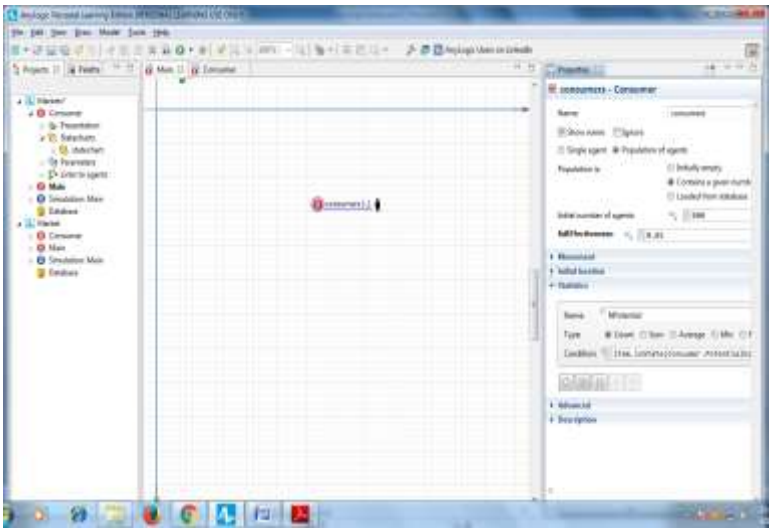
Step 2 : Define a statistic function in Consumer Properties –

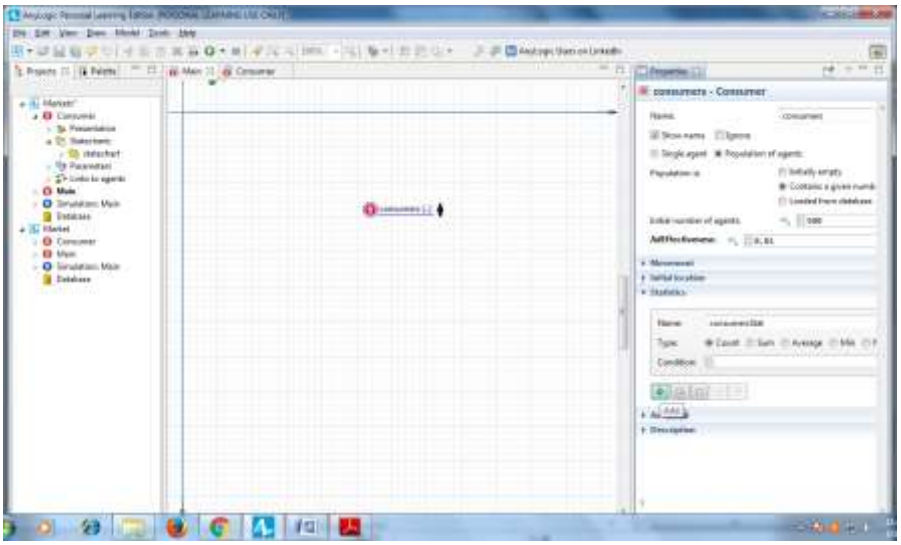


- **Name** NPotential.
- type **Count**



- Enter item.inState(Consumer.PotentialUser) as the function **Condition**.

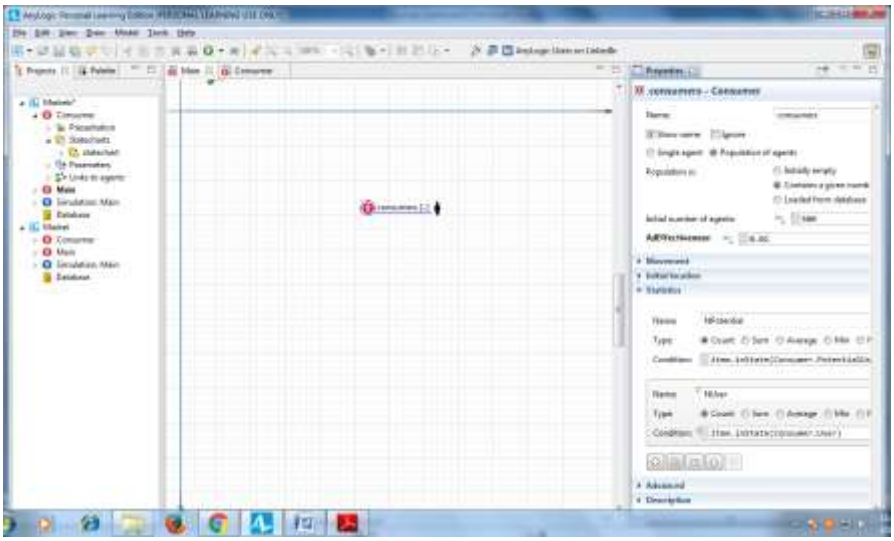




Step 3 : Define a second statistic function in Consumer Properties –

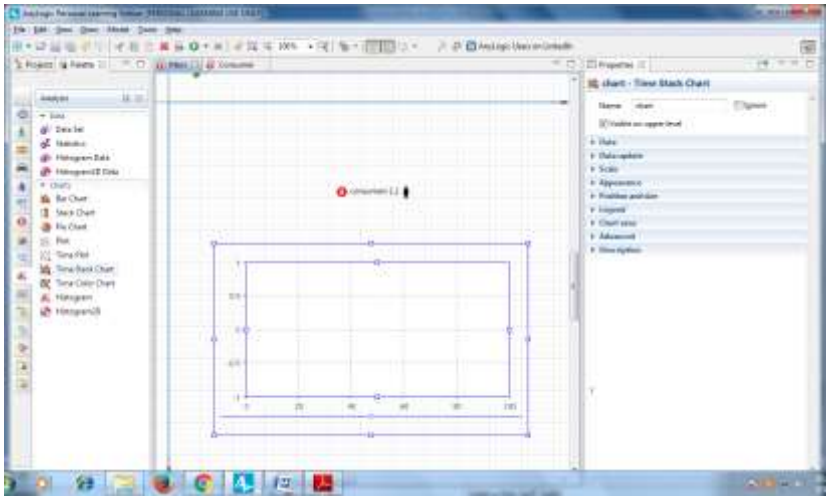
- **Name :** NUser
- **Type :** count the number of agents
- **Condition :** item.inState(Consumer.User).

You can duplicate the other statistics function by clicking the **Duplicate** button and changing its **Name** and the C**ondition**.



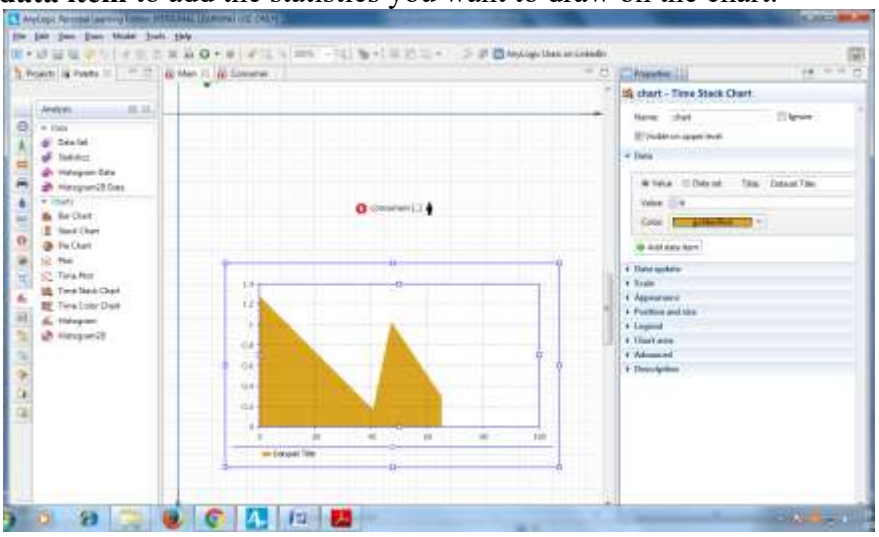
Step 4 : To add a Chart –

- Open the **Analysis** palette
- drag the **Time Stack Chart**



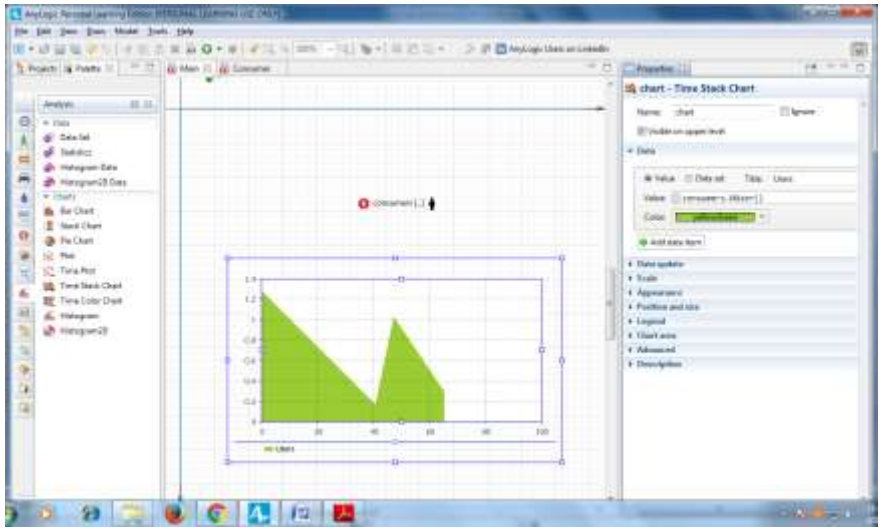
Step 5 : Add two data items for the chart to display.

- Click **Add data item** to add the statistics you want to draw on the chart.

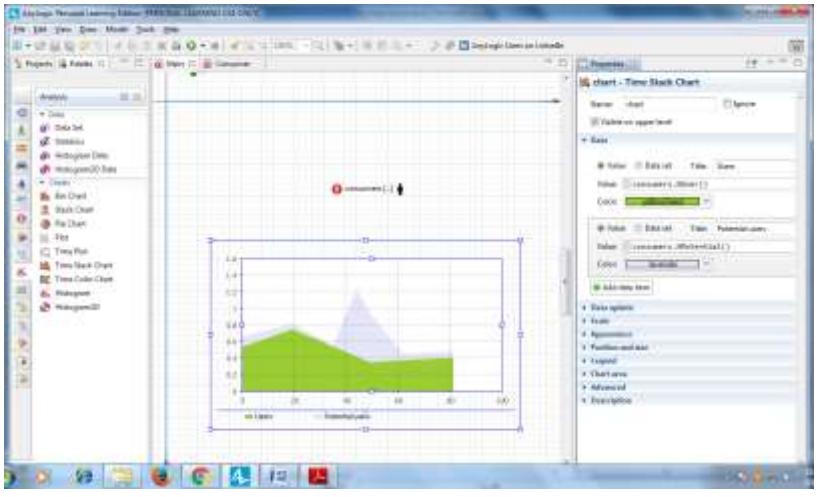


- Modify the data item's properties:
  - **Title**: Users – the data item's title.
  - **Color**: yellowGreen
  - **Value**: consumers.NUser()

Our agent population name is consumers, and NUser() is the statistics function that we defined for this population.

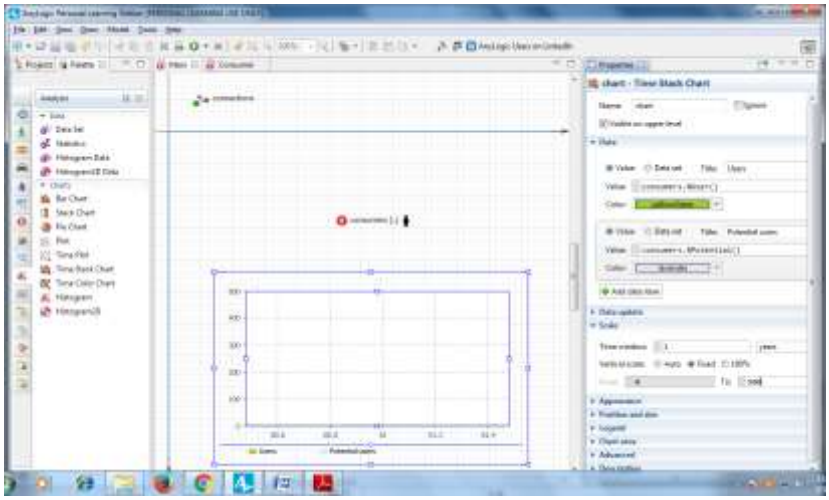


- Add one more data item:
  - **Title**: Potential users **7** AnyLogic 7 in Three Days **61**
  - **Color**: lavender
  - **Value**: consumers.NPotential()



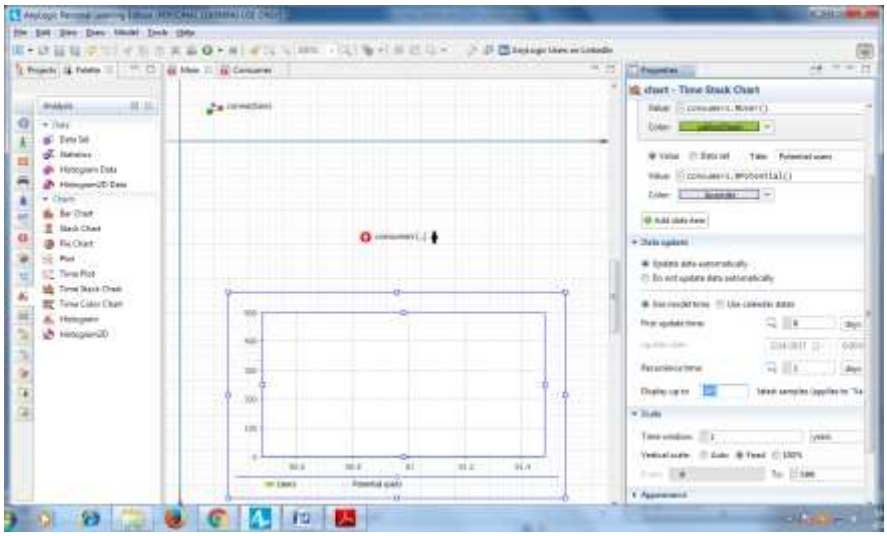
Step 6 : Go to the **Scale** section

- set **Time window** equal to 1 year.
- **Vertical scale** to **Fixed**
- enter 5000 in the **To:** box.



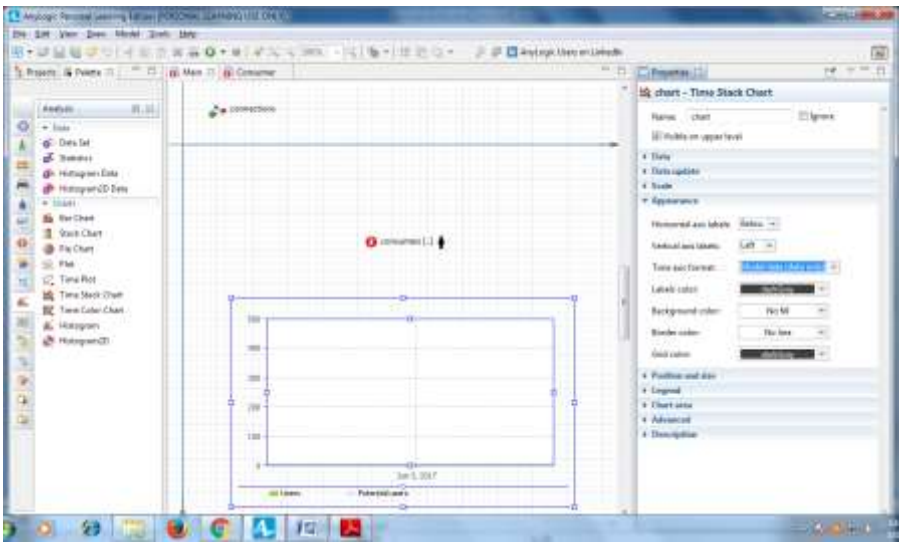
Step 7 : change the maximum number of data samples

- navigating to the section **Data update**
- set **Display up to** 365 **latest samples**.

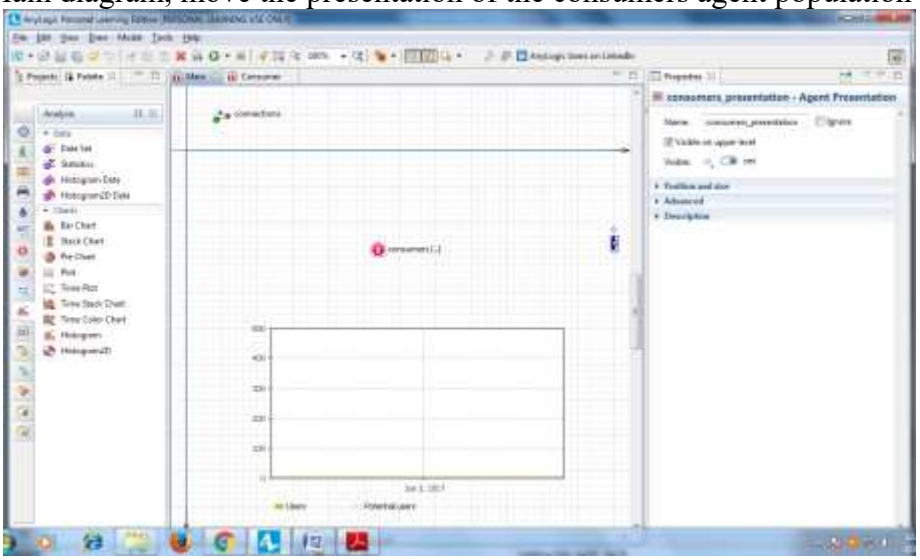


Step 8 : Go to the time stack chart's **Appearance** properties

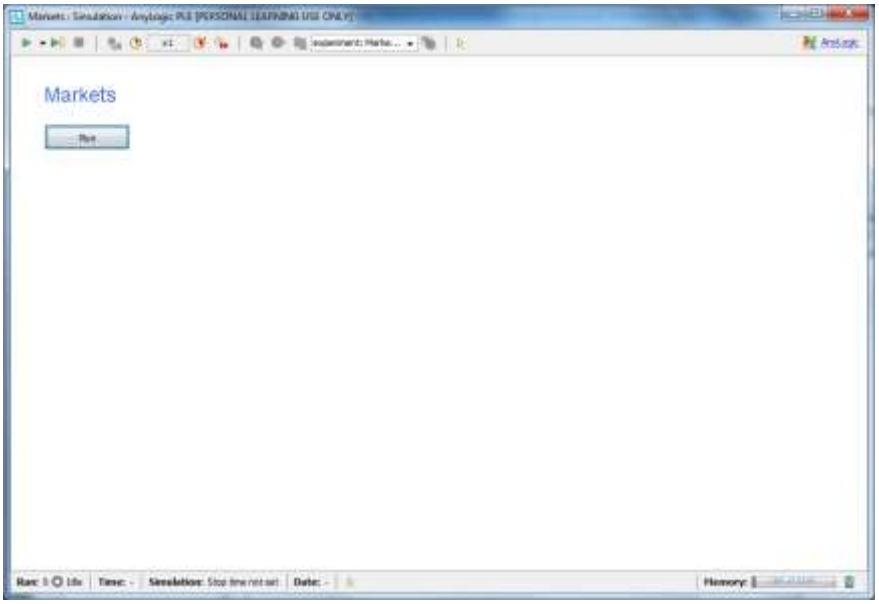
- set **Time axis format : Model date (date only)**

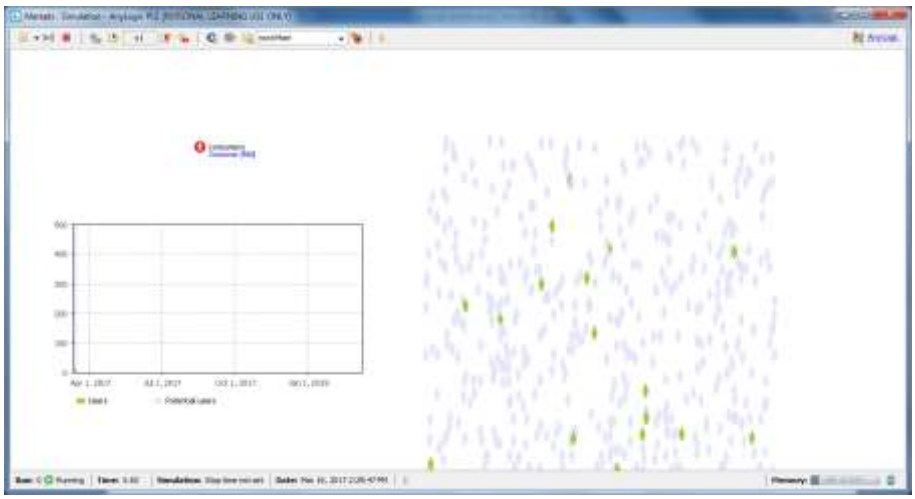


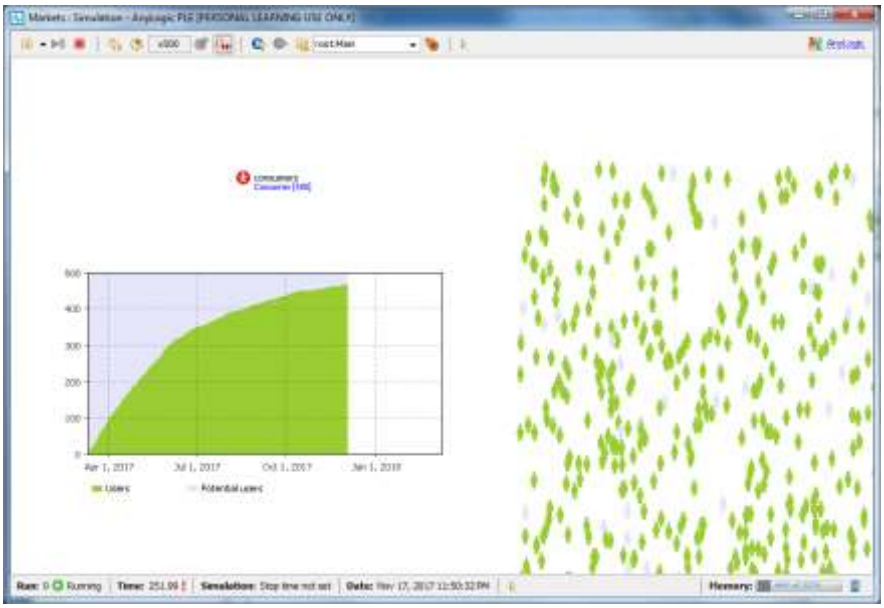
Step 9 : On the Main diagram, move the presentation of the consumers agent population to the right.

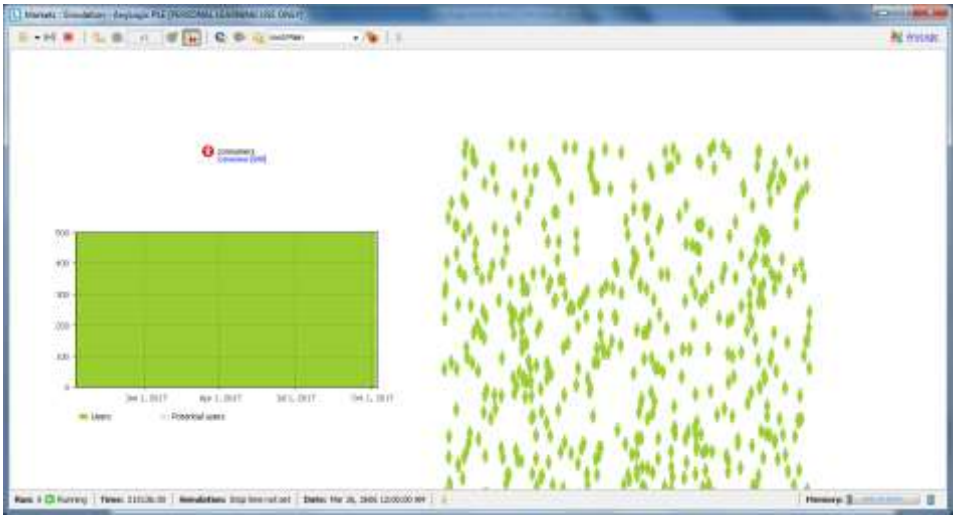


Step 10 : Run the model and use the time stack chart to review the process.









Practical 2

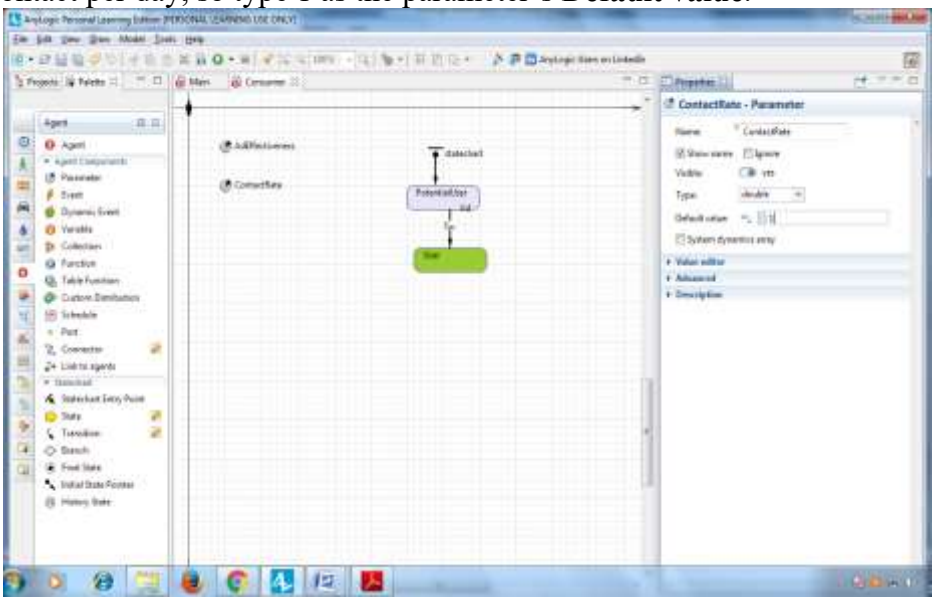
AIM : Design and develop agent based model by

- Creating the agent population
- Defining the agent behavior
- Adding a chart to visualize the model outpu
- Adding word of mouth effect
- Considering product discards
- Considering delivery time

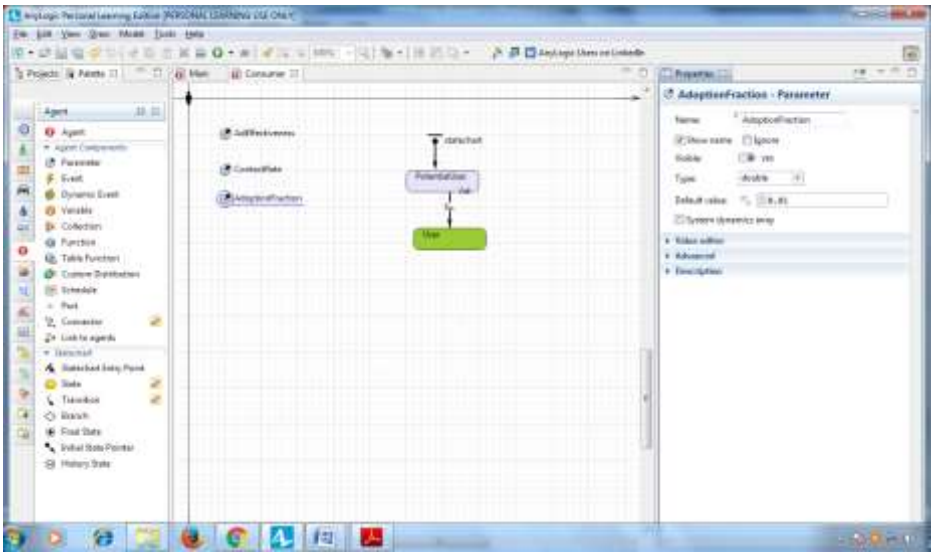
Solution:

Adding word of mouth effect

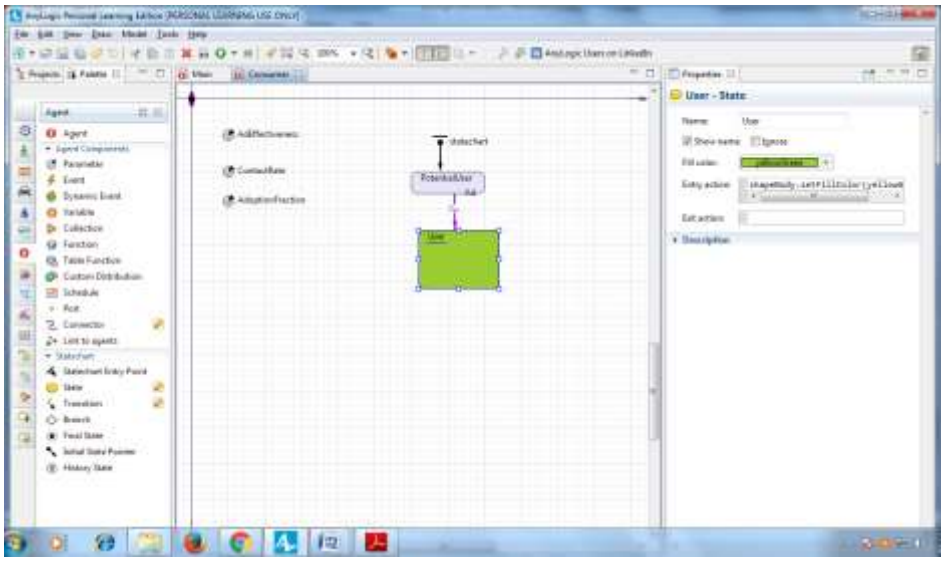
1. In the **Projects** tree, open Consumer diagram by double-clicking Consumer.
2. Add a parameter to define a consumer's average daily contacts. Drag the **Parameter** from the **Agent** palette on to the diagram.
3. Name the parameter ContactRate.
4. The rate is 1 contact per day, so type 1 as the parameter's **Default value**.



Add another parameter - AdoptionFraction - to define a person's influence on others, a number that we'll express as the percentage of people who will use the product after they come into contact with the consumer. Leave the default parameter's **Type**: double, and set the **Default value:** 0.01. The Consumer diagram should look like this:

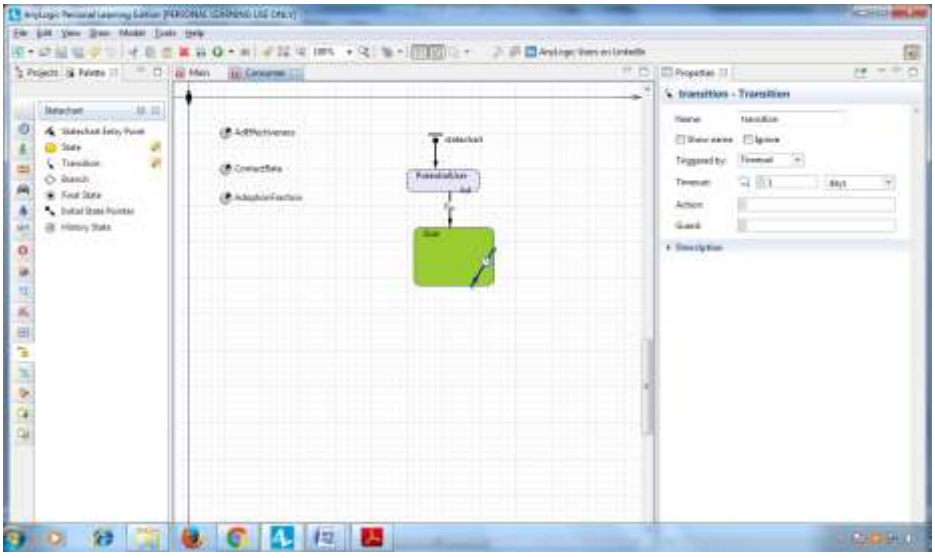


Open the Consumer diagram, and increase the User state to fit the internal transition we'll draw inside the state on the next step.



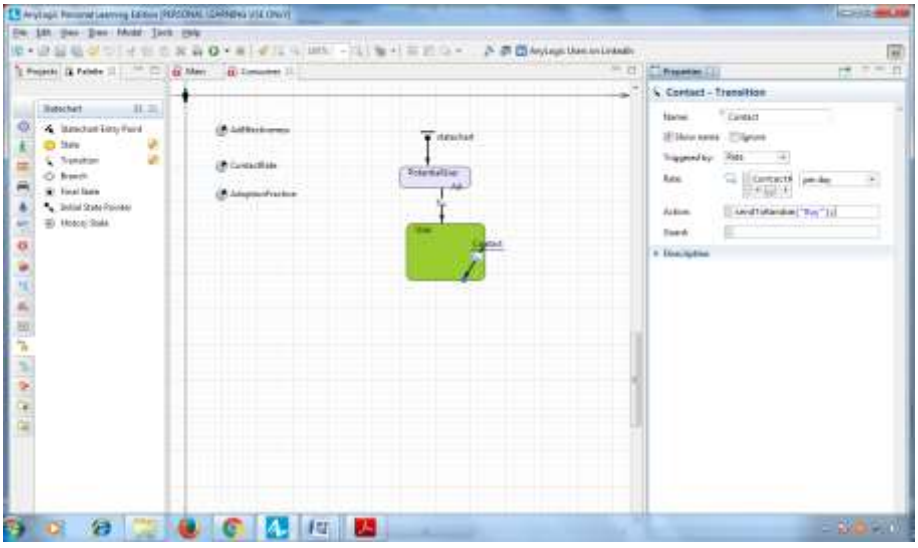
Draw an internal transition inside the User state. To draw a transition like the one shown below, drag the **Transition** from the **Statechart** palette inside the state so the transition's start point lies on the state border.

Afterward, you can move the transition end point to another point on the state border. To add a salient point, double-click the transition.

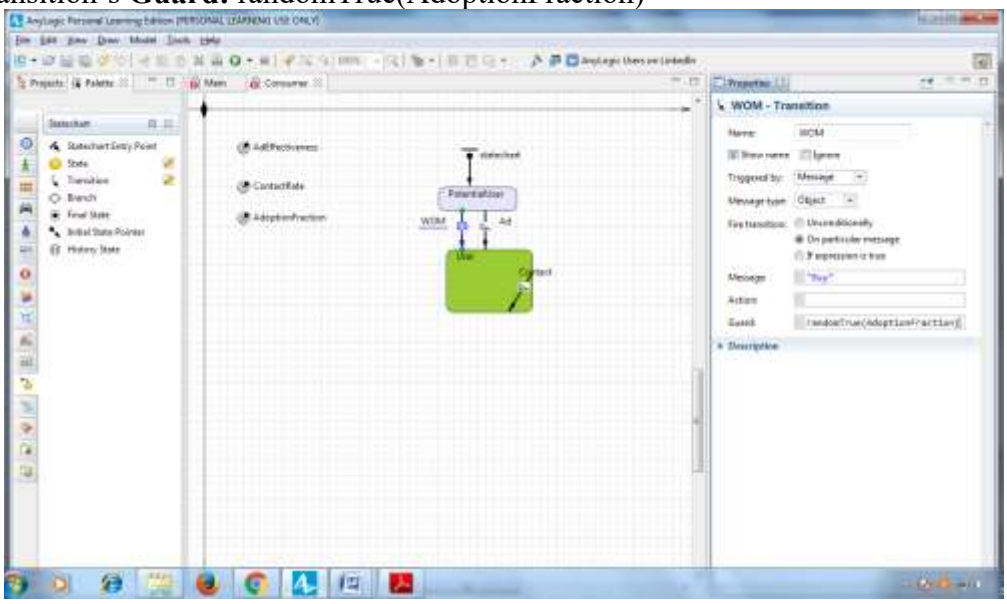


Modify the transition properties. This transition will occur with the specified **Rate** ContactRate (use code completion rather than typing the parameter's full name). Name the transition Contact and set it to show its name.

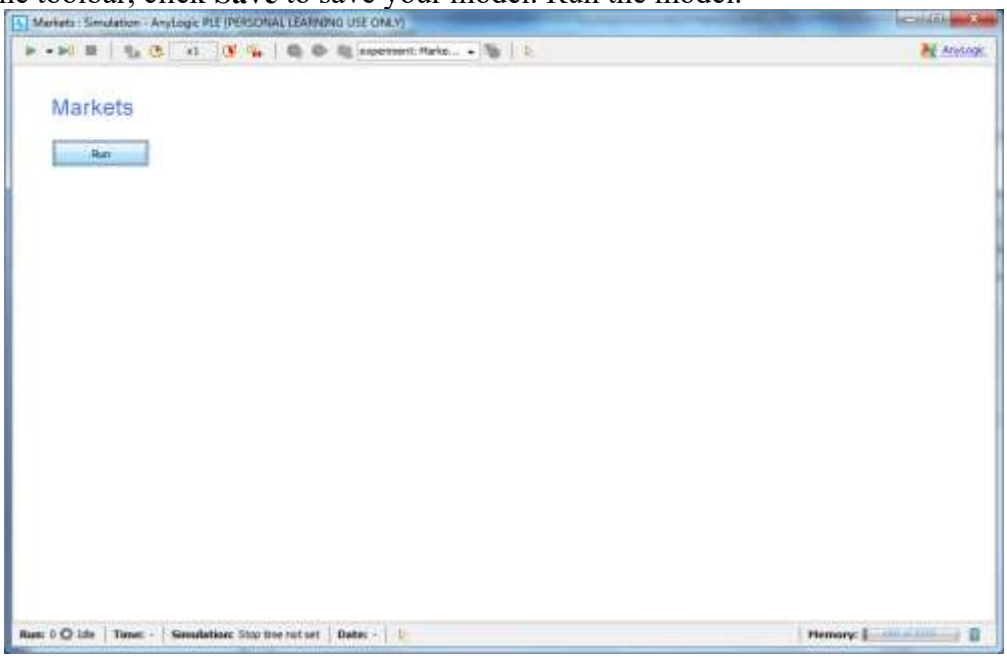
Specify the **Action** that will be executed on triggering this transition (use the code completion to write the code): sendToRandom("Buy");

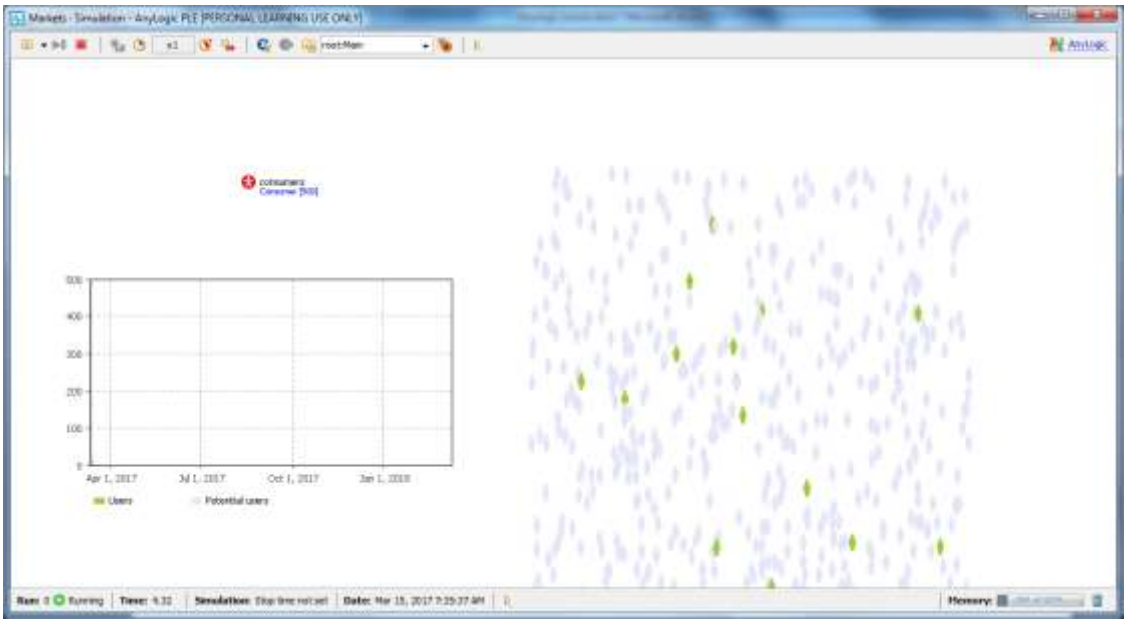


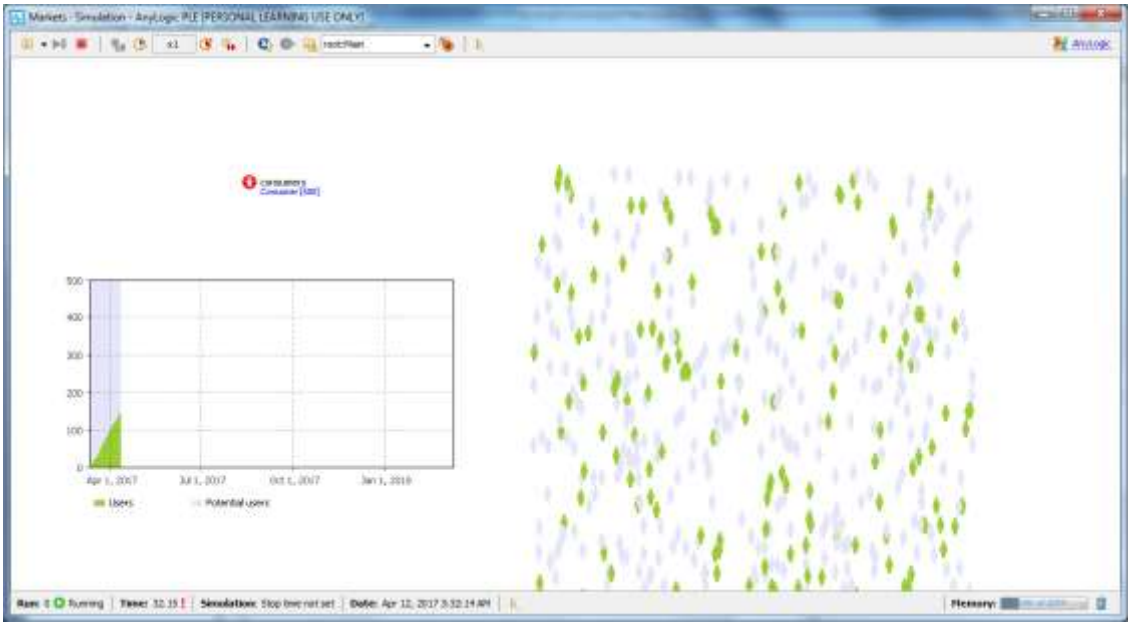
Draw another transition from PotentialUser to User state, and name it WOM (Word of Mouth). This transition will model purchases caused by word of mouth.
Modify the transition properties:
• In the **Triggered by** list, click **Message**.
• In the **Fire transition** area, select **On particular message**.
• In the **Message** field, type "Buy"
• Since we know not every contact is successful – in other words, a contact may not convince the potential user to buy our product – we'll use AdoptionFraction to make successful contacts less common. Specify the transition's **Guard:** randomTrue(AdoptionFraction)

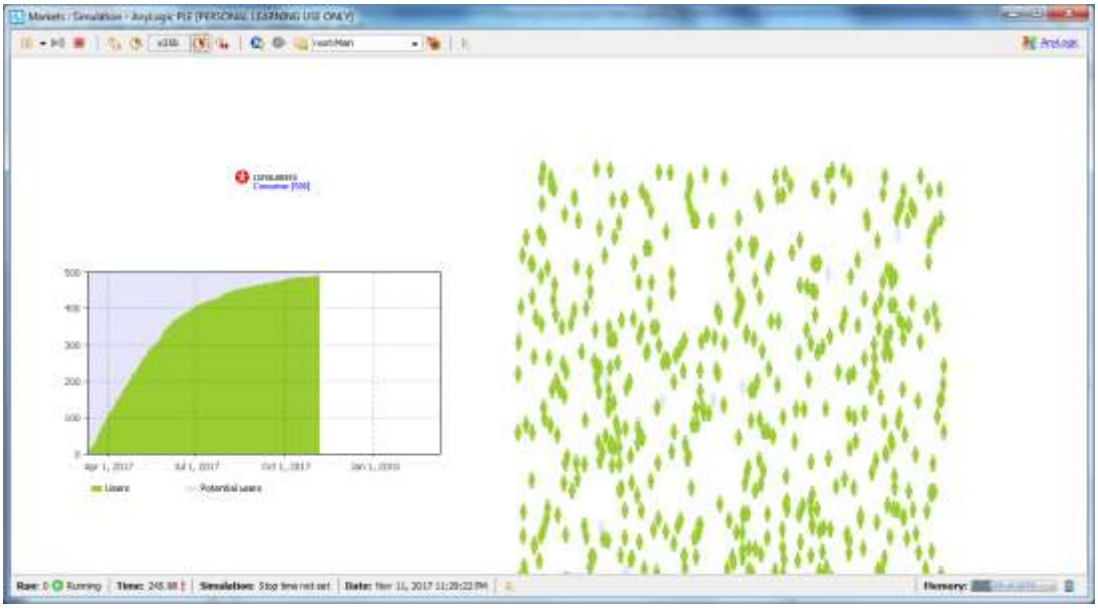


In the **Projects** view, you may see an asterisk near the model item that shows your model has unsaved changes. On the toolbar, click **Save** to save your model. Run the model.



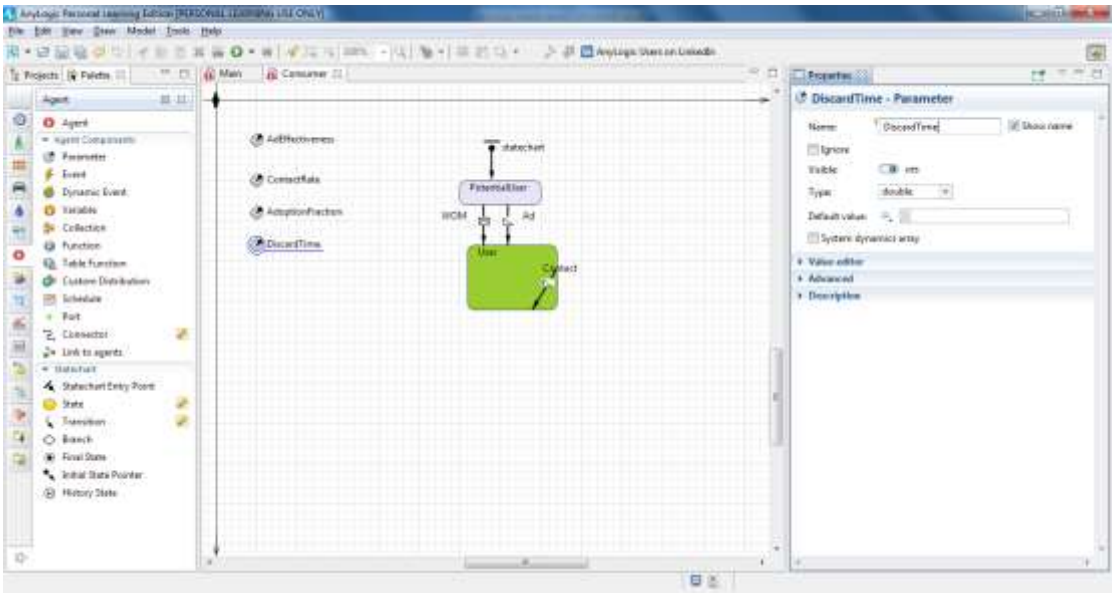




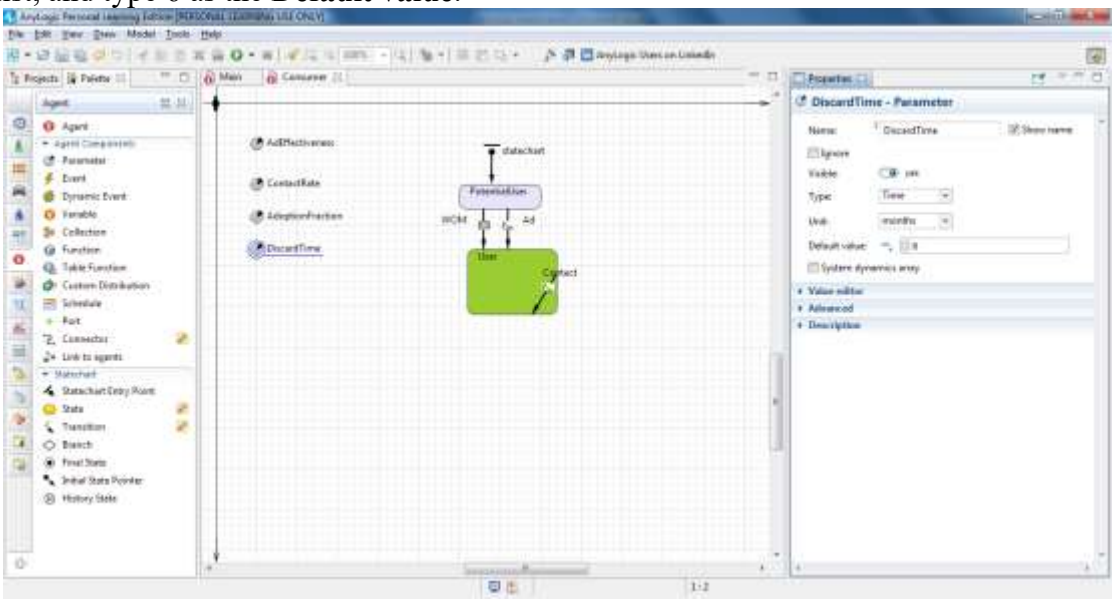


Considering product discards

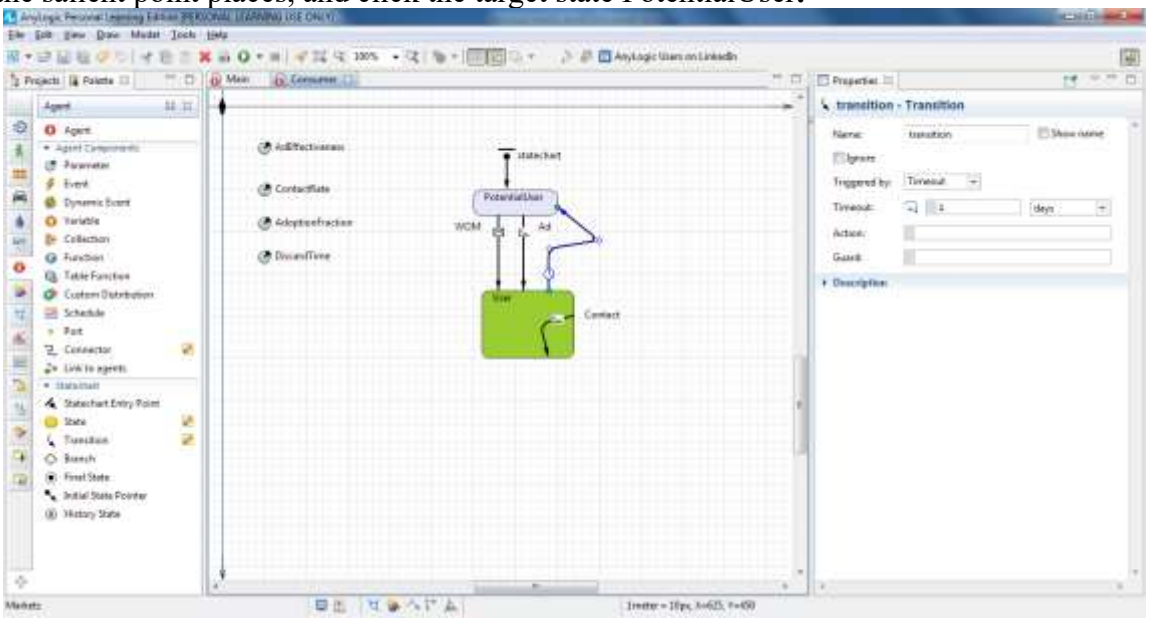
Open the Consumer diagram and add a DiscardTime parameter



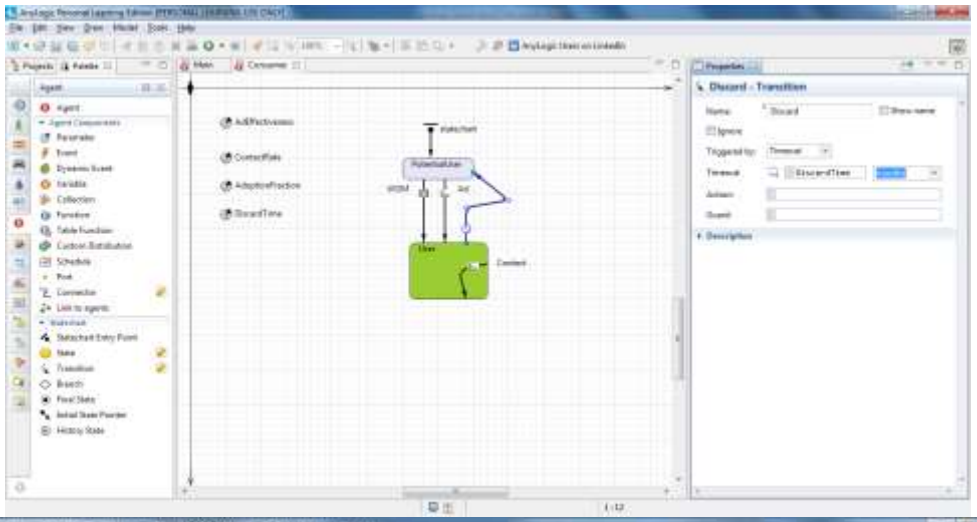
This parameter will define our product's lifespan. Choose **Time** as the parameter's **Type**, click **months** in the **Unit** list, and type 6 as the **Default value**.

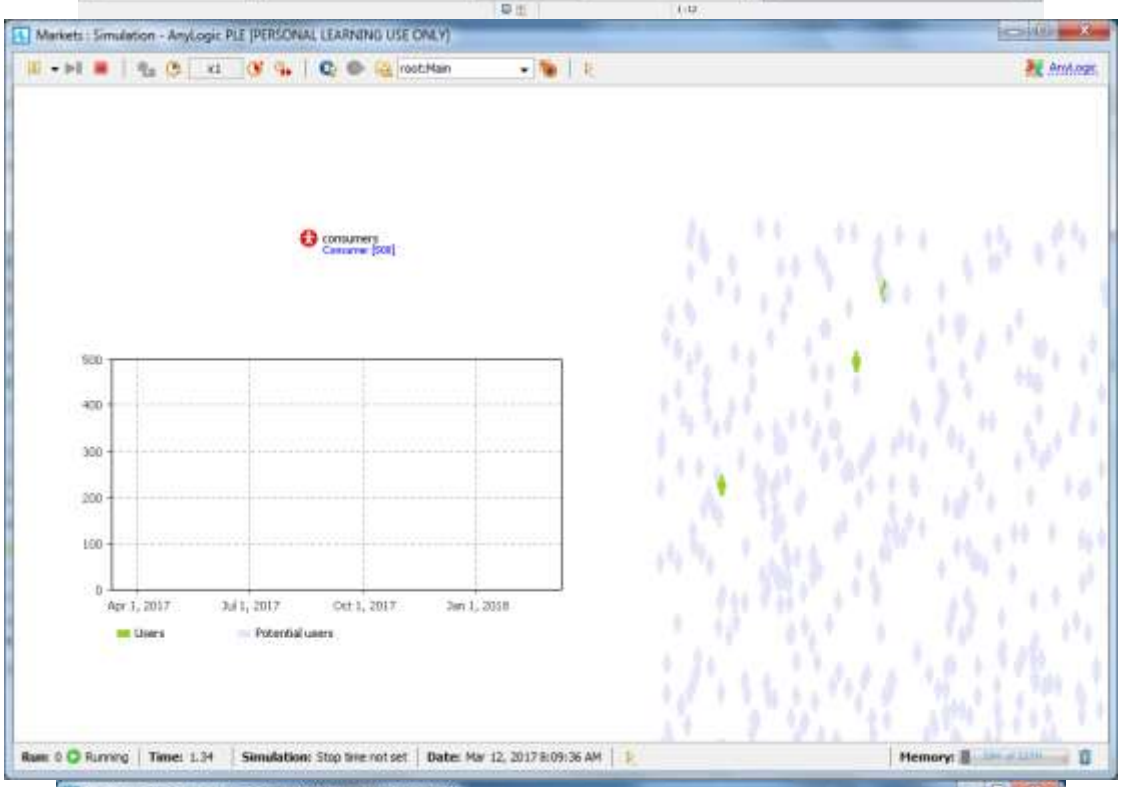


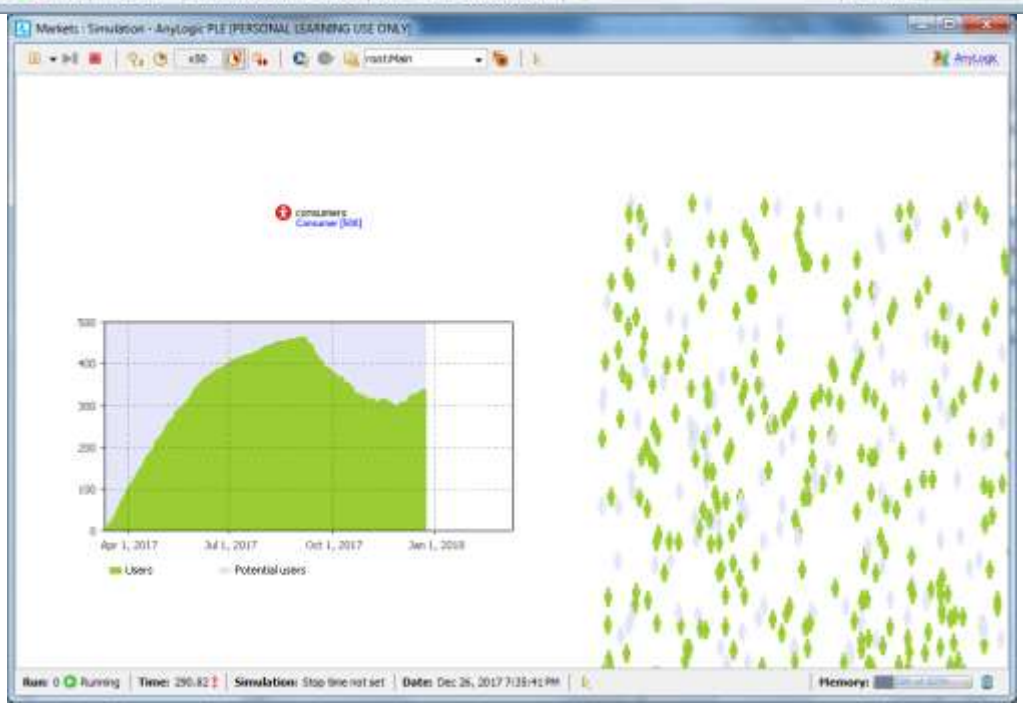
Draw a transition from User to PotentialUser state to model product discards. To draw a transition with salient points like those shown in the figure, double-click the **Transition** element in the **Statechart** palette (this should change the element's icon in the palette to), click the transition's source state User, click at the salient point places, and click the target state PotentialUser**.**



Name the transition Discard and set it to be triggered by a constant timeout DiscardTime. In the list to the right, click **months**.

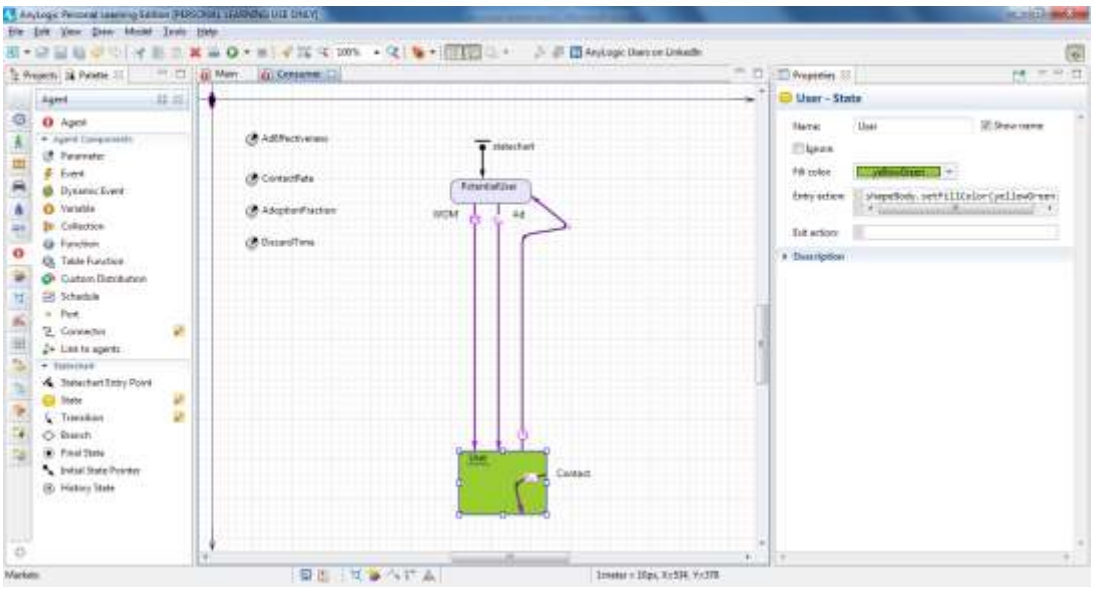




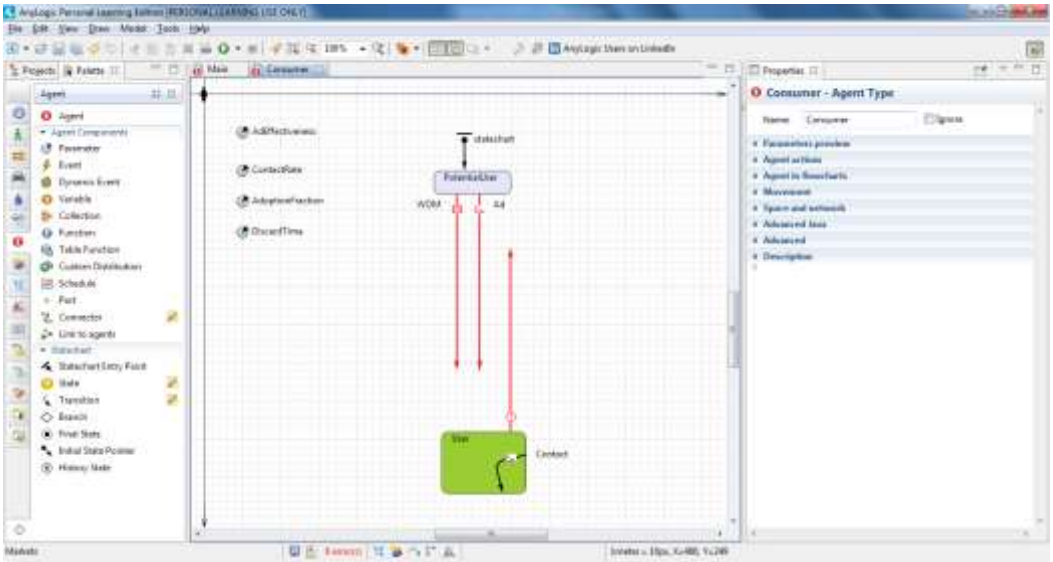


Considering delivery time

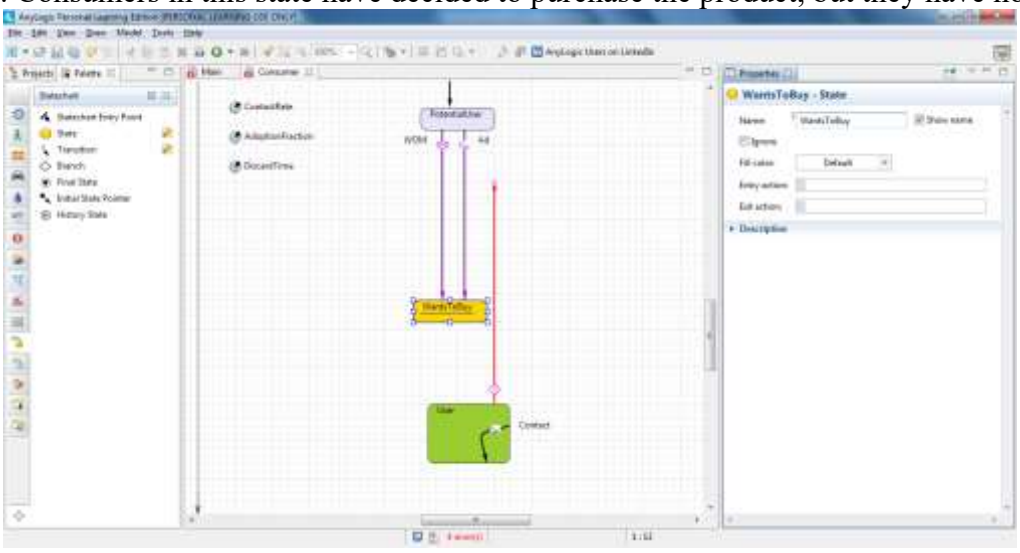
Prepare a place for another state between PotentialUser and User by moving the User state toward the bottom of the screen.



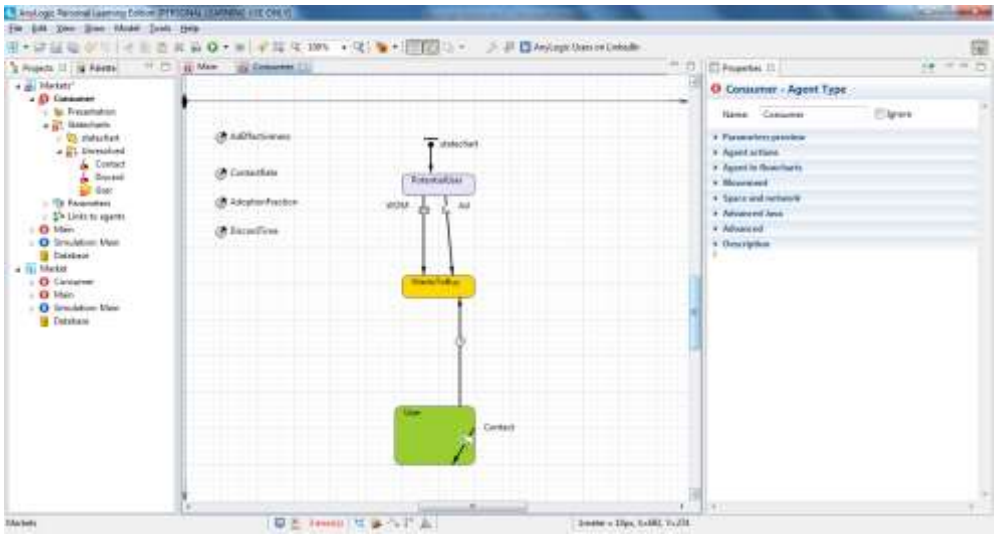
Disconnect the User state from the transitions. Select the WOM and Ad transitions, move their end points toward the top of the screen, and disconnect the Discard transition from PotentialUser. Afterward, you'll notice the disconnected transitions are drawn in red.



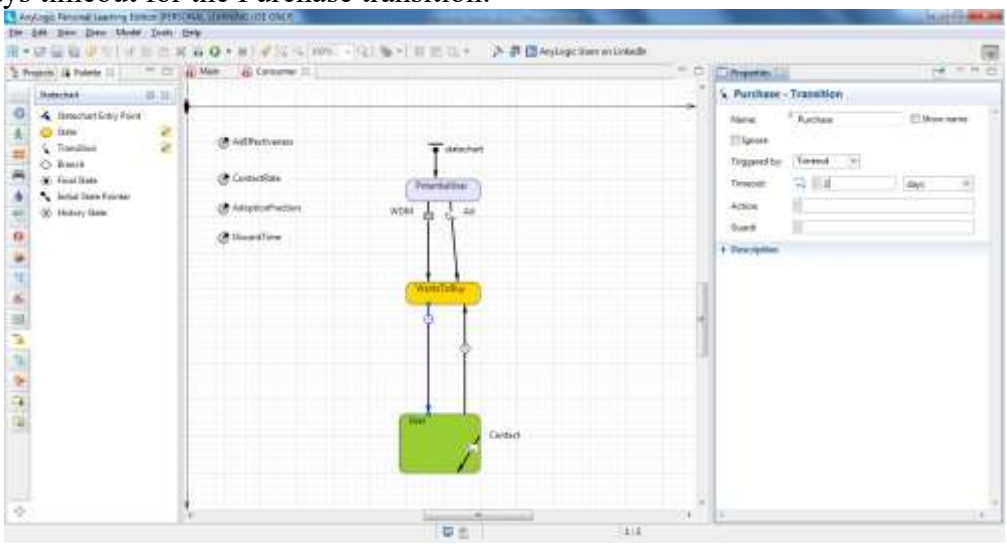
Add another **State** from the **Statechart** palette to the middle of the consumer's statechart and name it WantsToBuy. Consumers in this state have decided to purchase the product, but they have not done so.



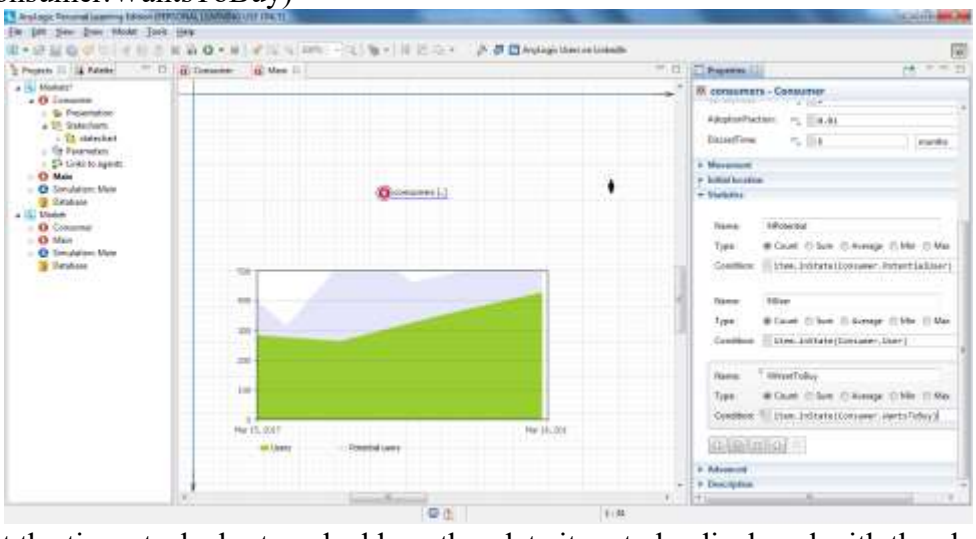
Reconnect transitions to the middle state: the WOM, Ad, and Discard transitions should now end in the WantsToBuy state.



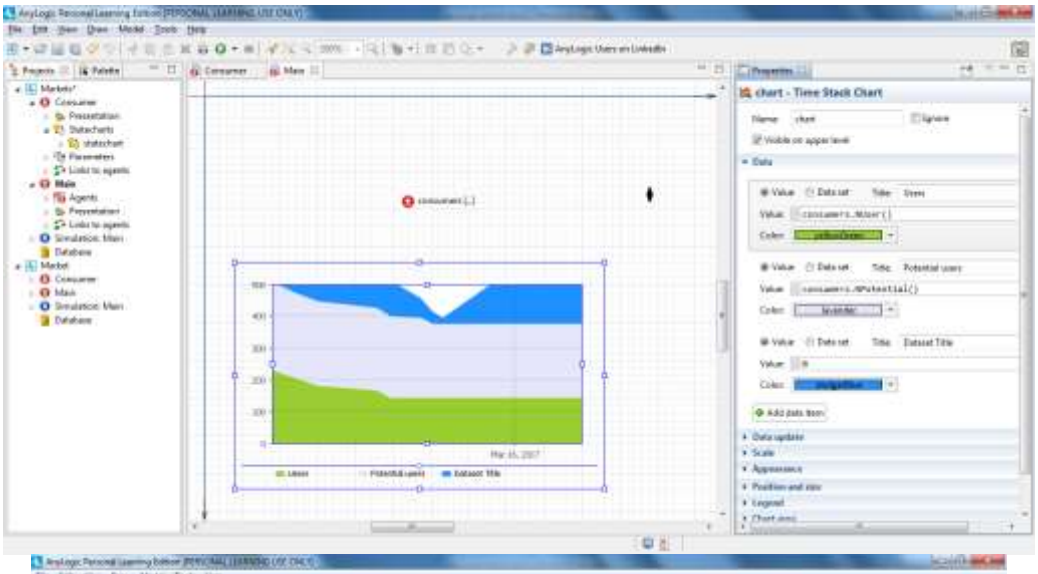
Modify WantsToBuy similar to other states:
Fill color: gold
Entry action: shapeBody.setFillColor(gold)
6. Add a transition from WantsToBuy to User state to model the product shipment and name it Purchase. Let's assume it typically takes a user two days to get the product. This means once the consumer's statechart enters the state WantsToBuy, it will proceed to the state User with a two-day delay. With this in mind, set 2 days timeout for the Purchase transition:

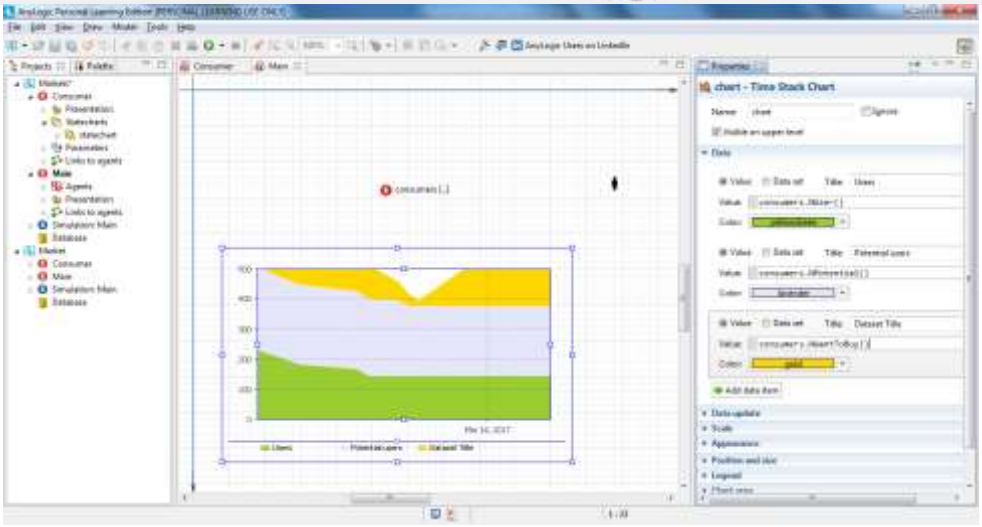


Define one more statistics function to count the product's market-driven demand. In the editor of Main, click the consumers, go to the **Statistics** properties section, and add a statistics item: NWantToBuy with condition
item.inState(Consumer.WantsToBuy)

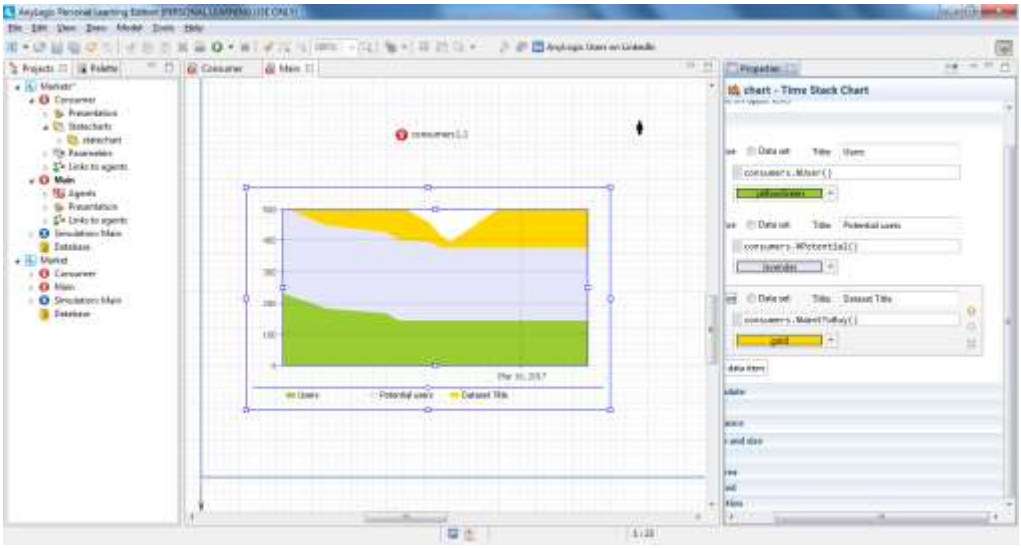


On Main, select the time stack chart, and add another data item to be displayed with the chart: consumers.NWantToBuy() with the title Want to buy and color gold.

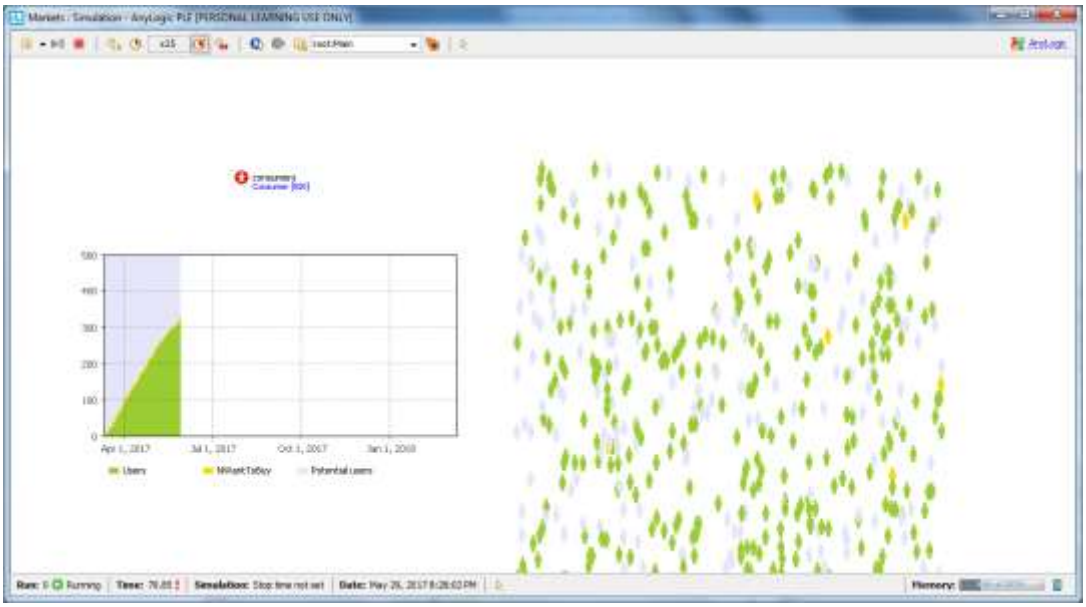


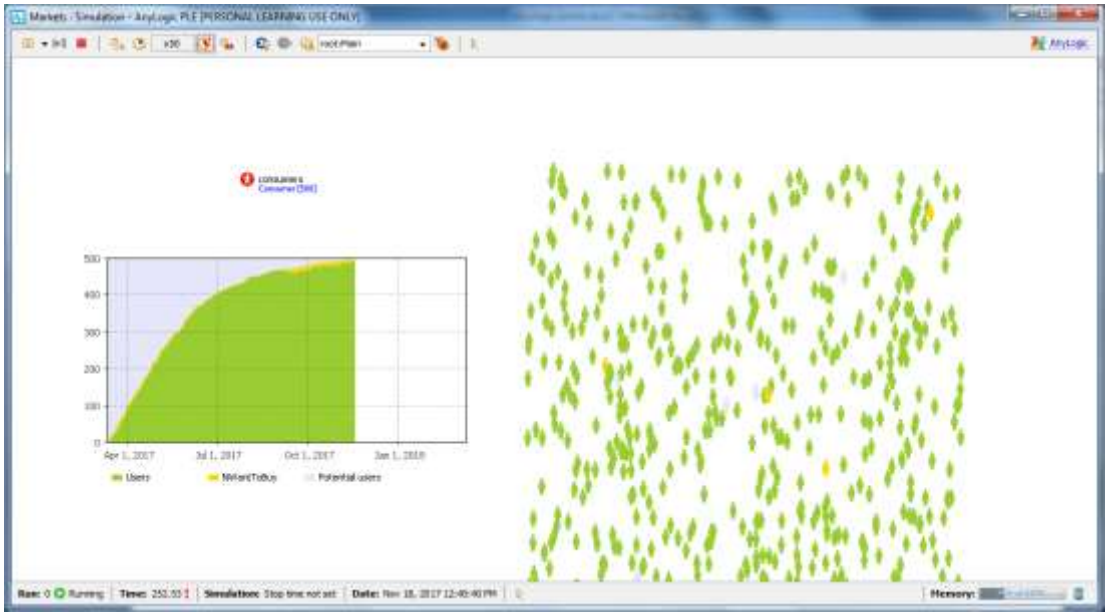


Make the newly-defined data item second in the list by selecting the item's section and clicking the "up" button.



Run the model, and you'll notice AnyLogic displays the number of consumers who are waiting for the product in yellow.





Practical 3

AIM : Design and develop agent based model by

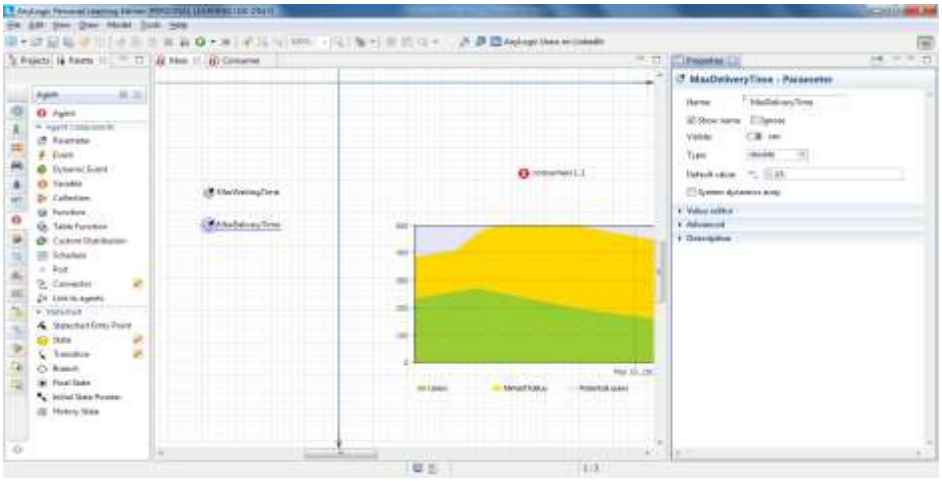
- **Creating the agent population**
- **Defining the agent behavior**
- **Adding a chart to visualize the model output**
- **Adding word of mouth effect**
- **Considering product discards**
- **Consider delivery time**
- **Simulating agent/Consumer impatience**
- **Comparing model runs with different parameter values**

Solution:

Simulating agent/Consumer impatience

Step 1 : Configure the parameters :

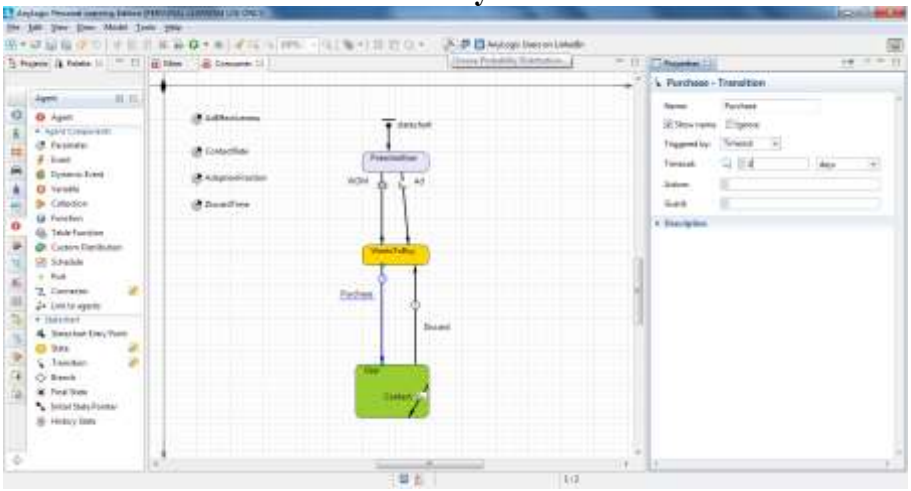
- MaxWaitingTime defines the maximum time a consumer will wait for the product for 7 days.
- MaxDeliveryTime to 25 days to reflect our assumption it may take up to 25 days to deliver a product.



Step 2: Open the Consumer diagram and select the Purchase transition set the following :

- Name : Purchase
- Triggered by : Timeout
- Timeout : 1 days

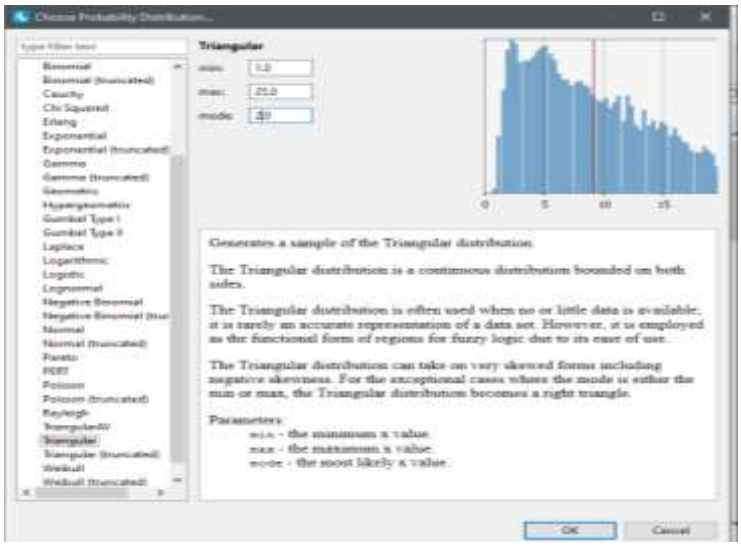
Step 3 : Click the Toolbar Button - **Choose Probability Distribution**



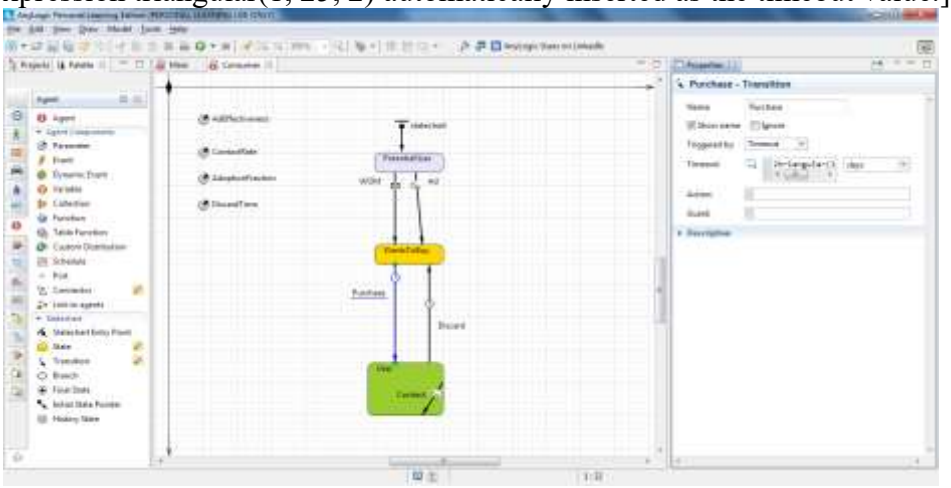
Step 4 : The **Choose Probability Description** screen allows you to view the list of supported distributions. (you can click any name in the list to view the distribution's description)

- Choose triangular in the list.
    - Set **min** = 1
    - Set **max = 25** and
    - Set **mode = 2**.

(In the upper right, you'll see PDF instantly built for the distribution with the specified parameters.)

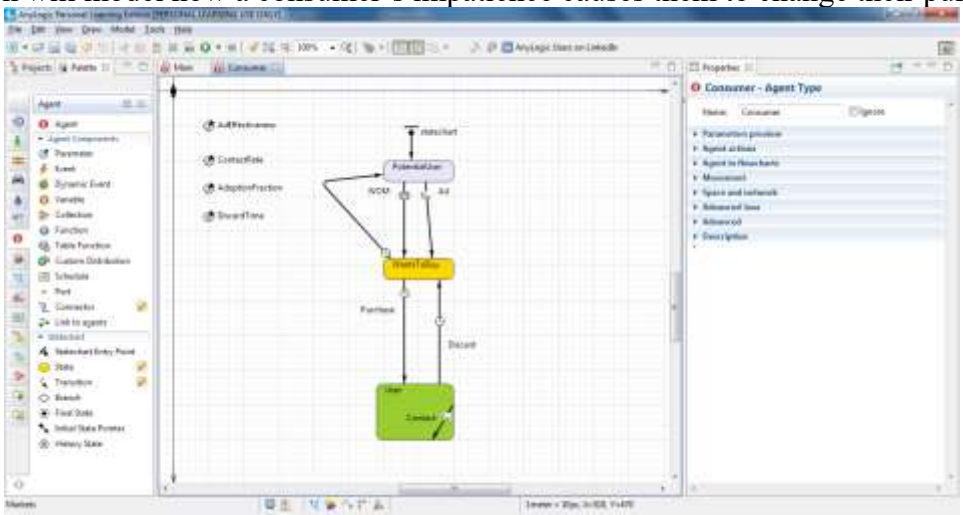


Click **OK** when finished.
[You'll see the expression triangular(1, 25, 2) automatically inserted as the timeout value.]

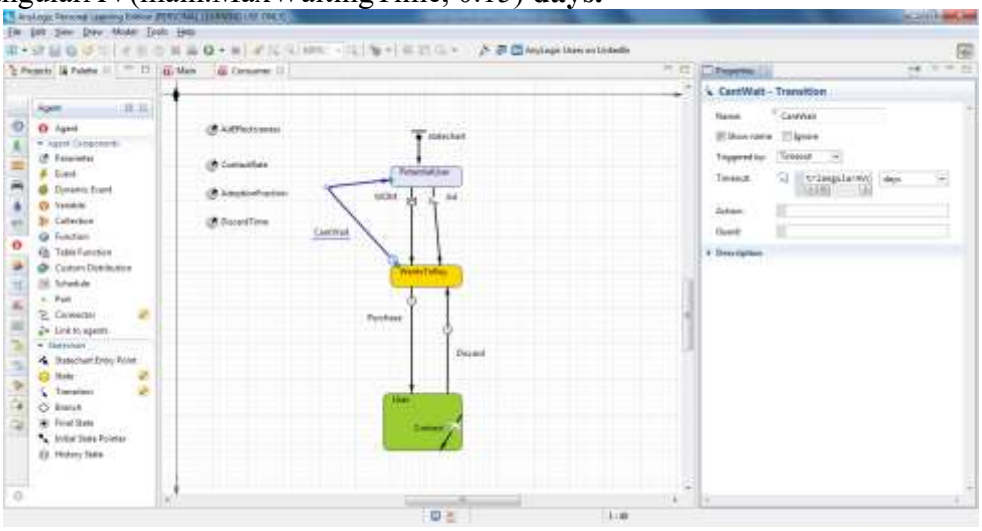


Step 5 : Modify the line to triangular(1, main.MaxDeliveryTime, 2)

- Draw the last transition CantWait that goes from WantsToBuy to PotentialUser state. (This transition will model how a consumer's impatience causes them to change their purchase decision)



Step 6 : Modify the transition properties:
Timeout : triangularAV(main.MaxWaitingTime, 0.15) **days.**



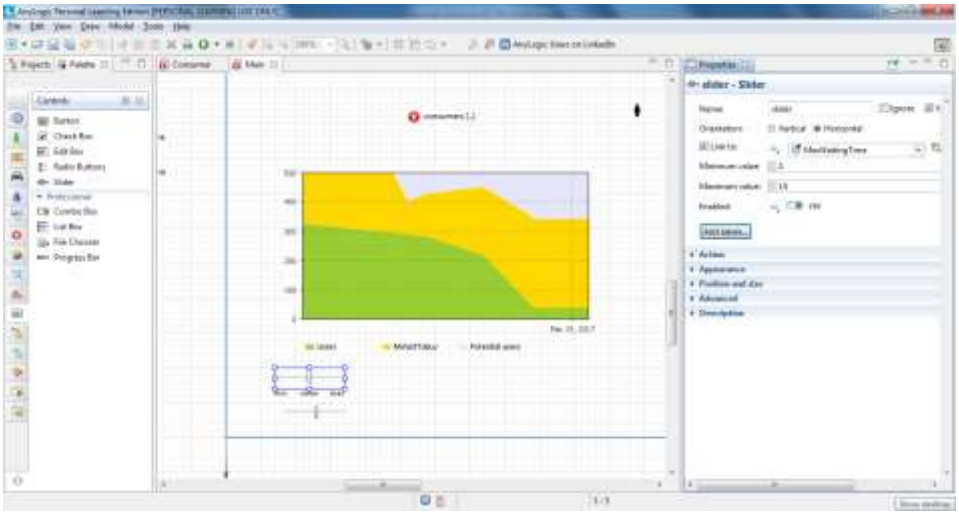
(Rather than setting the maximum waiting time equal to constant MaxWaitingTime, we assume it follows a triangular distribution with an average of one week and a possible variation to up to 15 percent.)

Step 7 : Go back to Main diagram and Add 2 Sliders

- Open the **Controls** palette
- drag two **Sliders** on to the diagram below the chart. (We'll eventually link the sliders to our two parameters.)

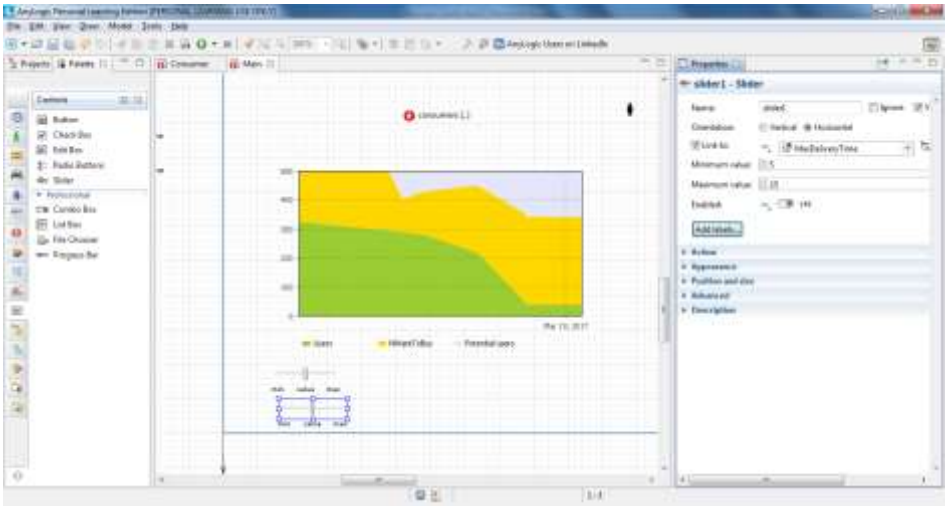
For Slider1:

- Select the checkbox : **Link to**
- Select the parameter MaxWaitingTime to the right.
- Set the slider's **Minimum= 2** and **Maximum = 15 values**.
- Finally, click the **Add labels** button - to display the slider's minimum, maximum, and current values at runtime



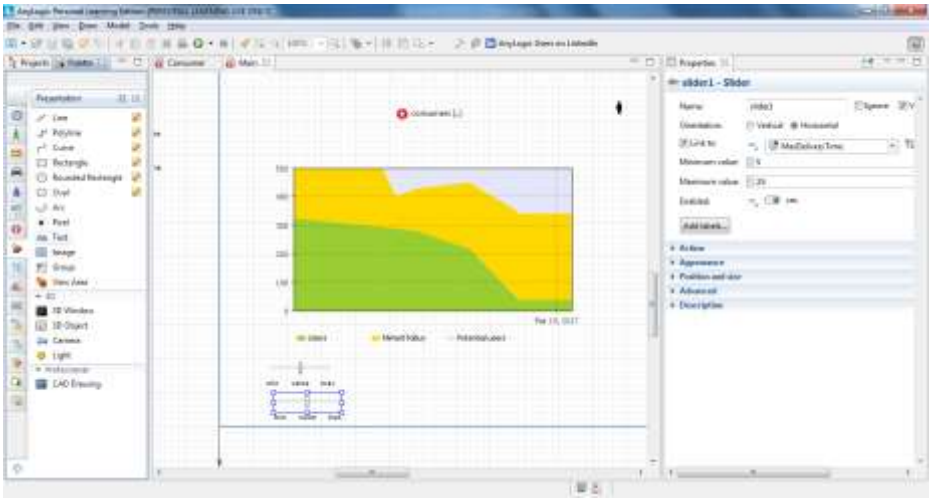
For Slider2:

- Select the checkbox : **Link to**
- Select the parameter MaxDeliveryTime to the right.
- Set the slider's **Minimum= 5** and **Maximum = 25 values**.
- Finally, click the **Add labels** button - to display the slider's minimum, maximum, and current values at runtime

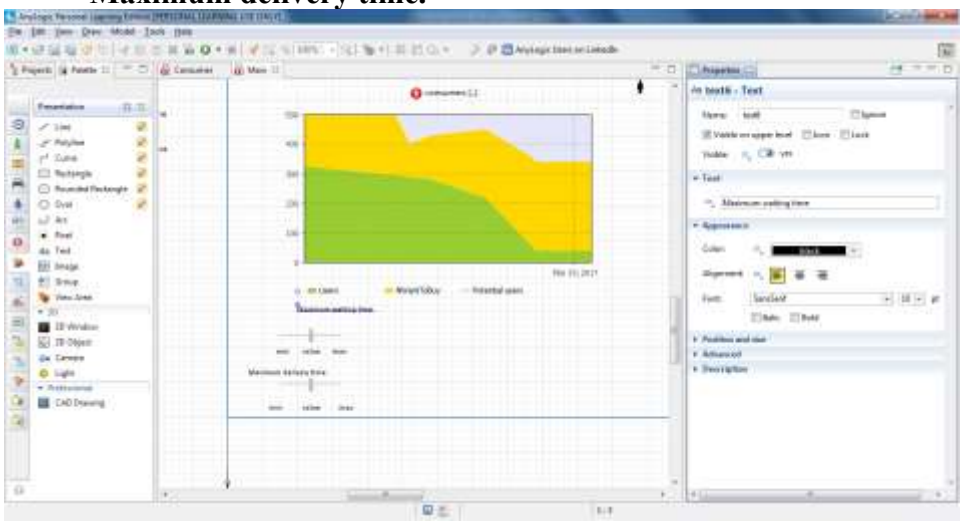


Step 8 : Open the **Presentation** palette and add 2 Text

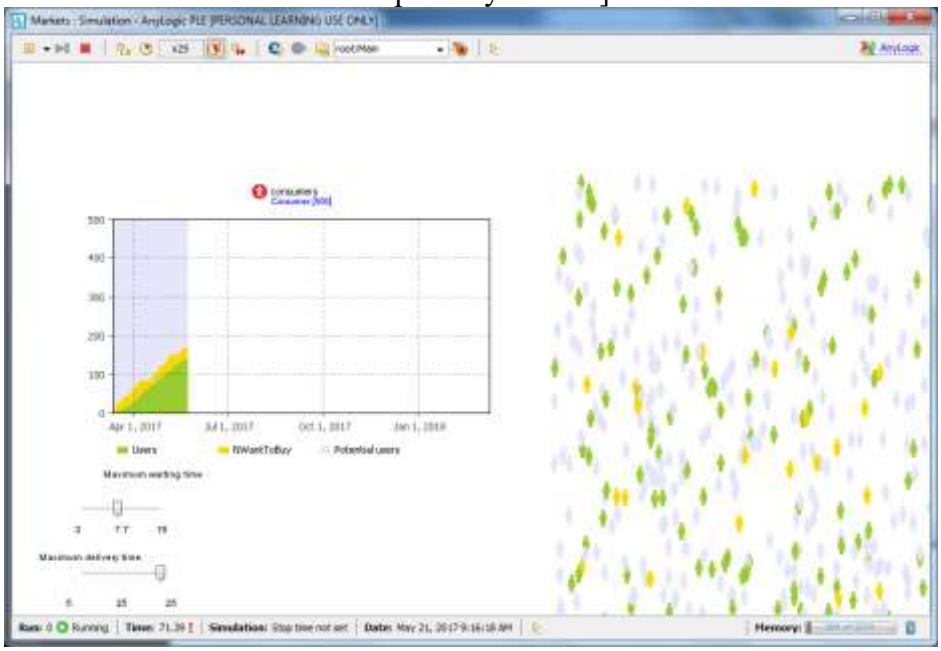
- Drag two **Text** shapes on to the diagram
- Place them above the sliders.

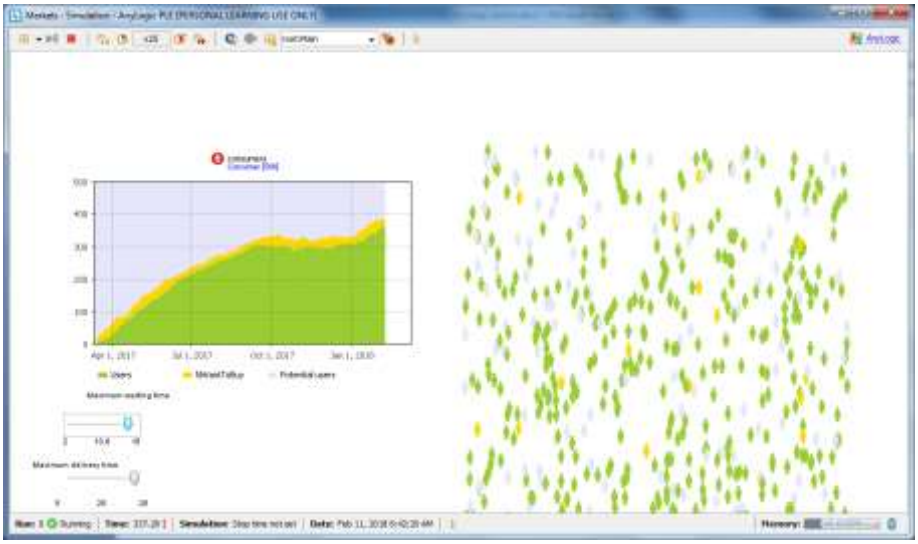


- Let's configure the titles of these controls : In the properties view,
  - in the **Text** section
    - enter the text that the model will display :
    - Using text shapes, name one slider **Maximum waiting time** and the other slider **Maximum delivery time.**



Step 9 : Run the model and observe the behavior.
[As you use the sliders to change the maximum waiting time or delivery time, you'll see your changes reflected in consumer behaviors and whole adoption dynamics.]

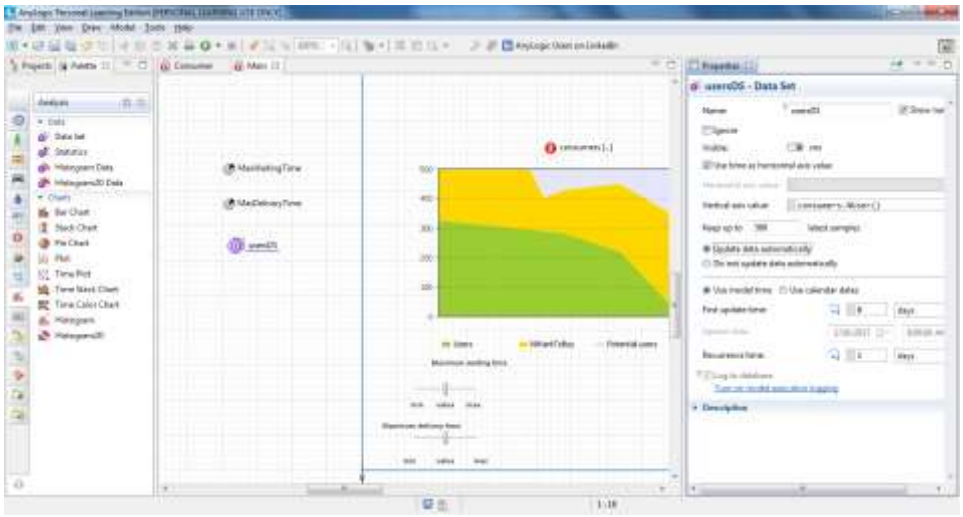




Comparing model runs with different parameter values

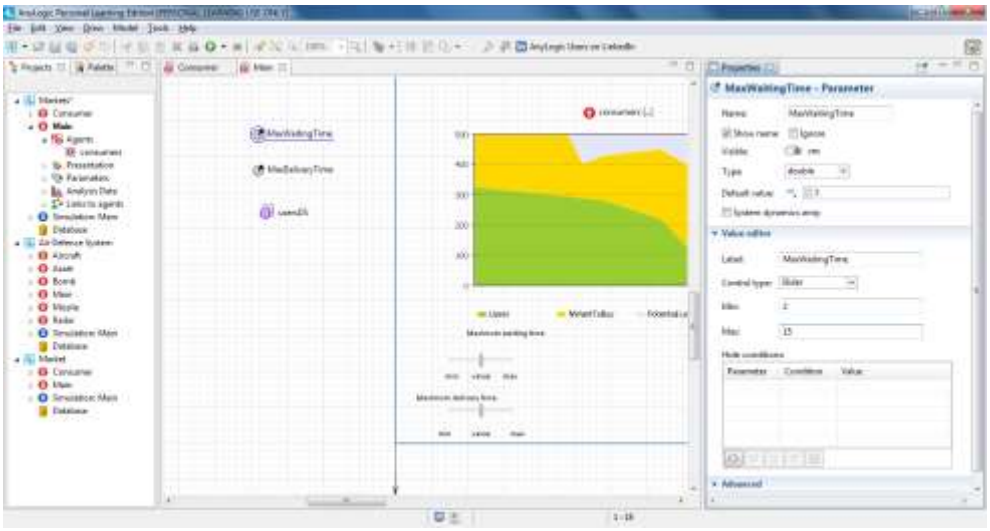
Step 1 : Open the Main diagram and add a **Data Set** from the **Analysis** palette.
[dataset to store the history of product sales dynamics]

- Name : usersDS
- In the **Vertical axis value** property, type: consumers.NUser().
- **Keep upto** : 500 **latest samples**
- Set it to **Update data automatically** with the default **Recurrence time**: 1 days

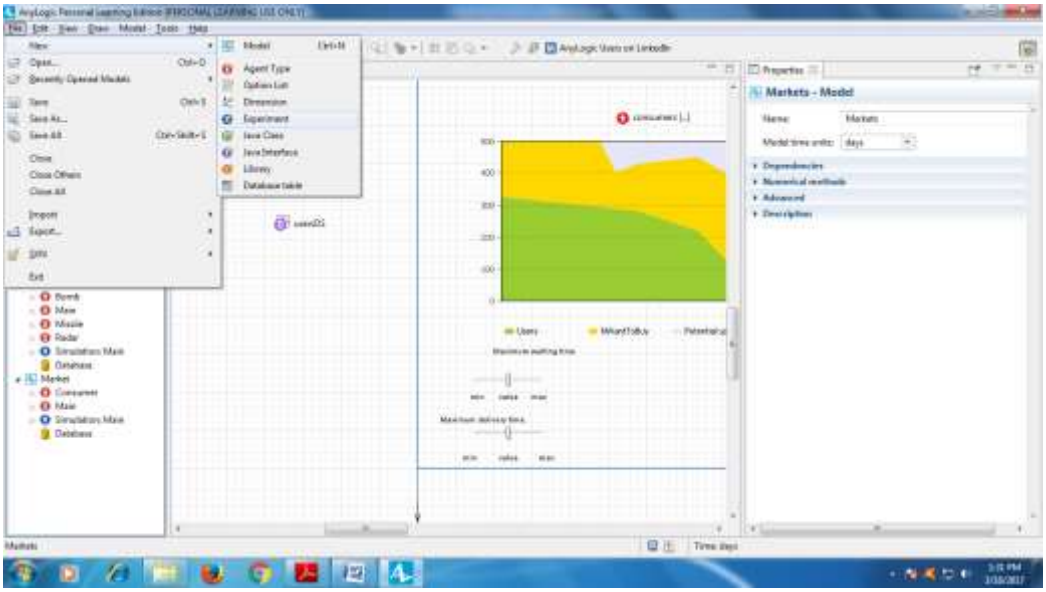


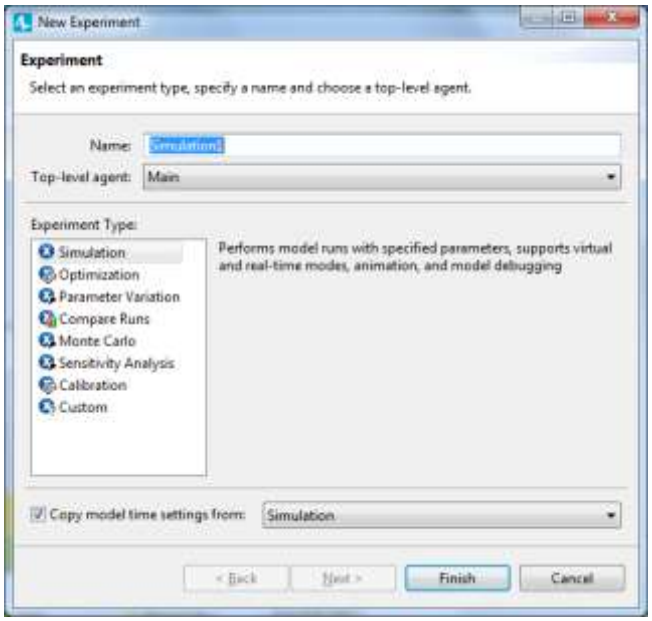
Step 2 : For MaxWaitingTime and MaxDeliveryTime make changes in the **Value editor** section

- Choose Slider as **Control type**,
- Set **Min** and **Max** values the same as we have in the sliders on Main, and if you want, change the default label

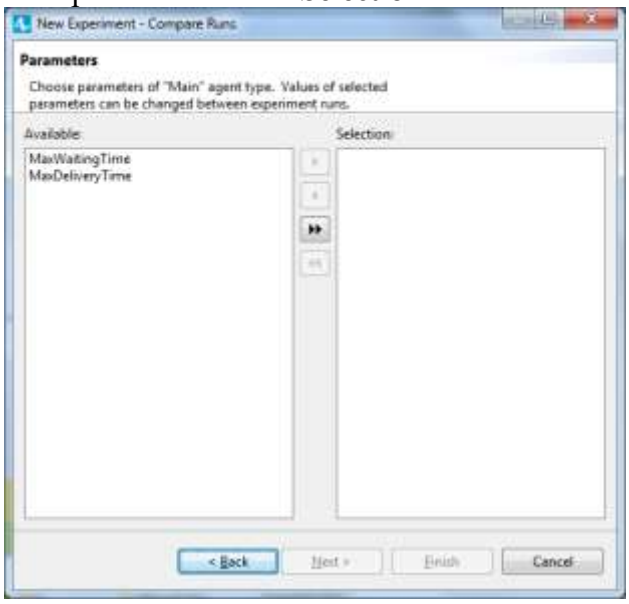


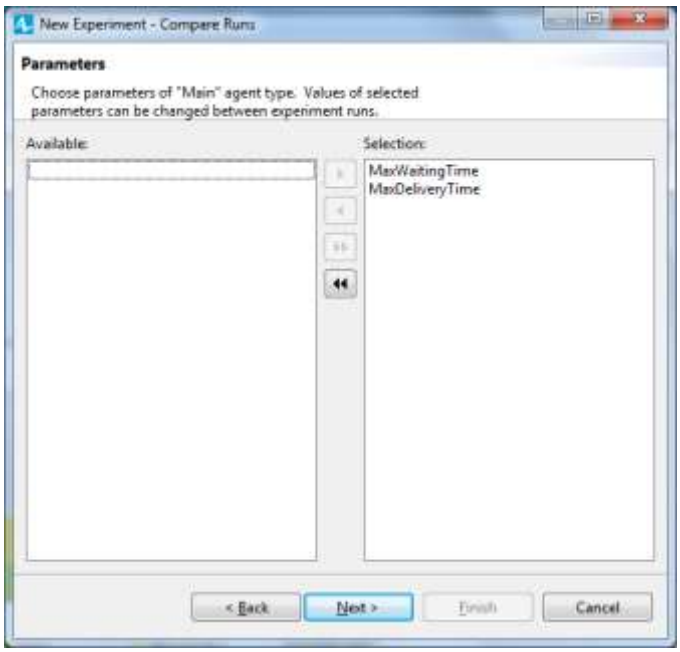
Step 3 : Open the **Projects** view, right-click the model item, and select **New > Experiment** from the context menu. The **New experiment** wizard will pop up.





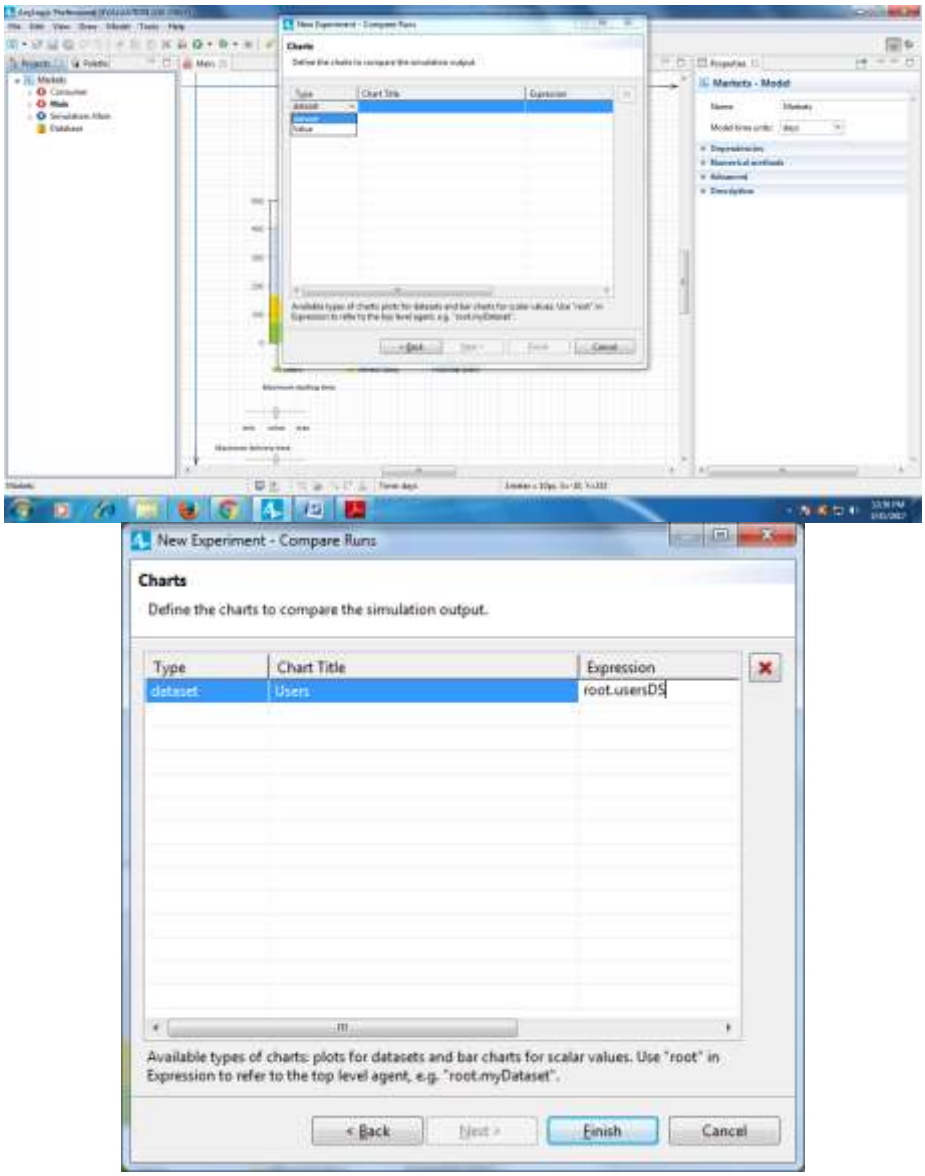
Step 4 : Select **Compare Runs** experiment from the list of experiment types and click **Next**.
Step 5 : On the **Parameters** page, add both parameters to the **Selection** column. To add a parameter, select it in the **Available** list on the left and click the arrow. You can also click the button to add all the parameters. Click **Next** after both parameters are in **Selection**



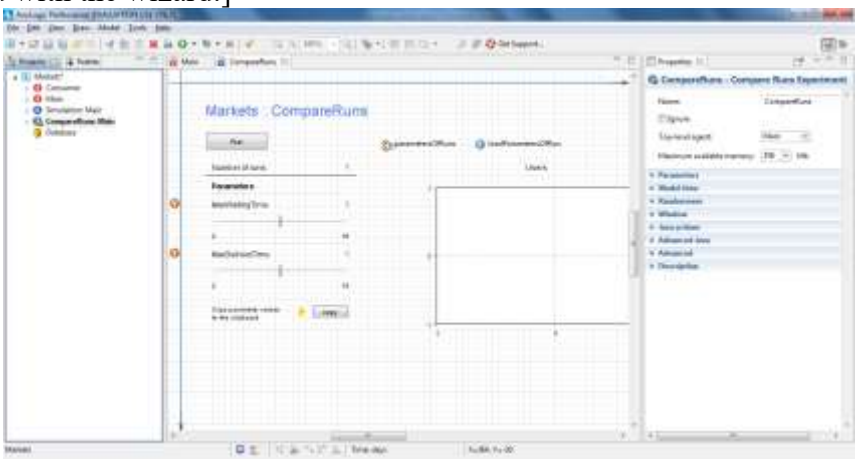


Step 6 :

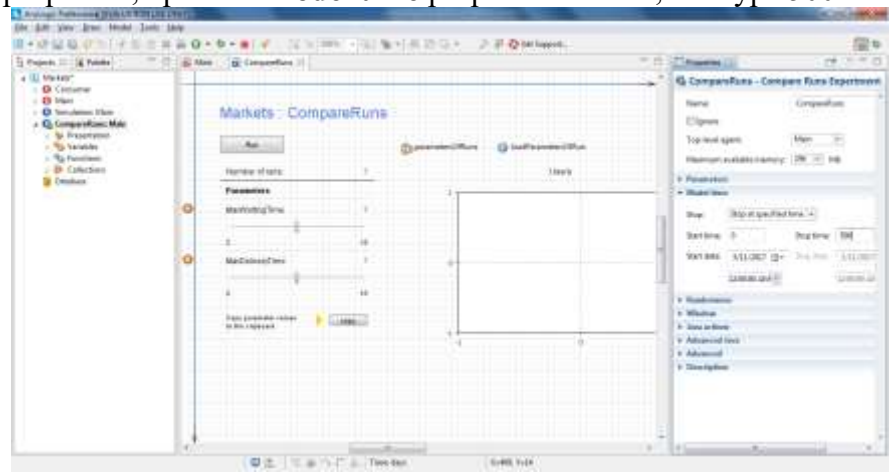
- In the **Type** column, select **dataset**.
- In the **Chart Title** column, type Users.
- In the **Expression** column, refer to the dataset you defined in Main as root.usersDS where root is the model's top-level agent (Main)



The chart will display the data collected by the dataset usersDS. Click **Finish**.
[The CompareRuns experiment diagram should open automatically, and you'll see the default user interface we created with the wizard.]

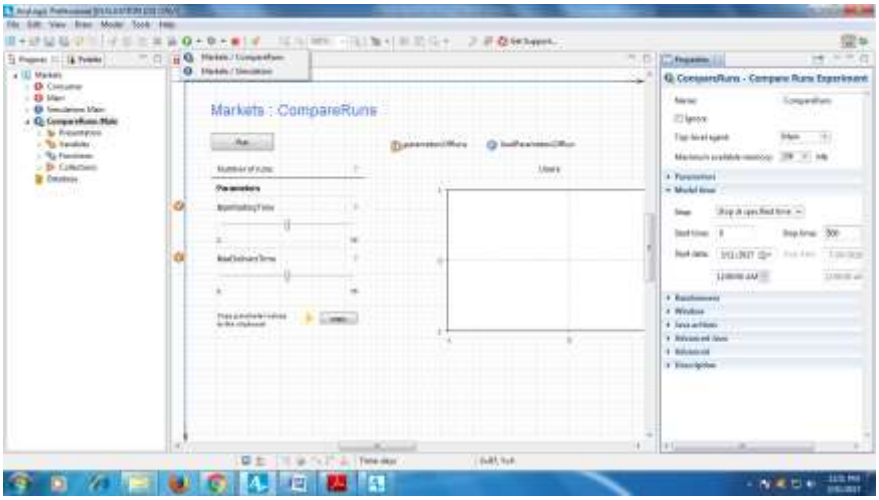


Step 7 : to simulate the model for just 500 days
Select CompareRuns experiment in the **Projects** tree.
In the experiment properties, open the **Model time** properties section, and type 500 in the **Stop time** field.

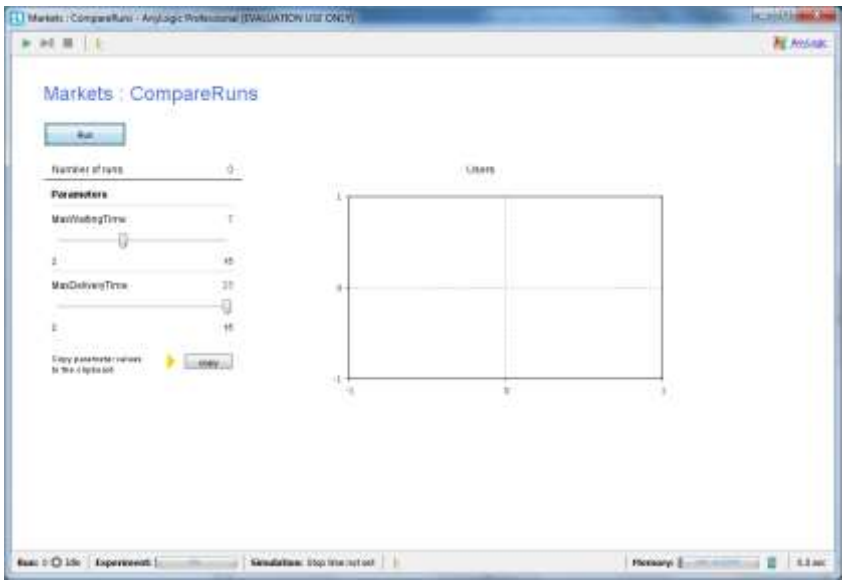


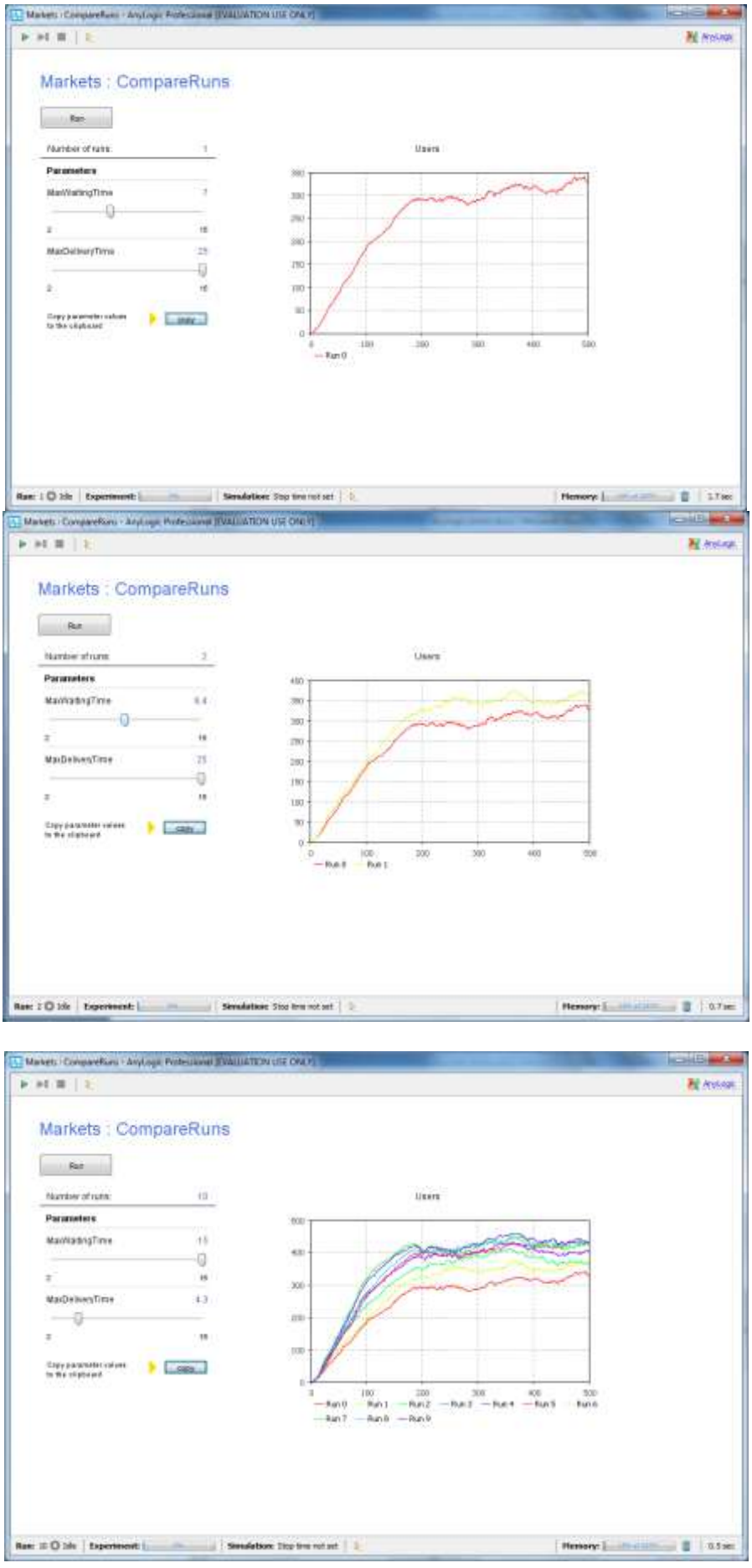
Step 8 : Run the experiment.

- Select the newly-created experiment from the **Run** list: **Market / CompareRuns**, OR right-click the CompareRuns experiment in the **Projects** tree
- select **Run** from the context menu.

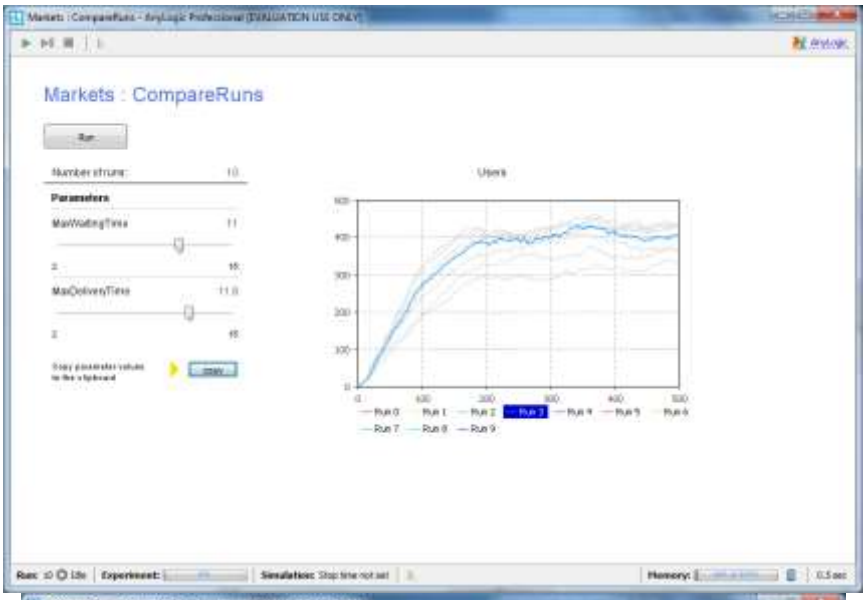


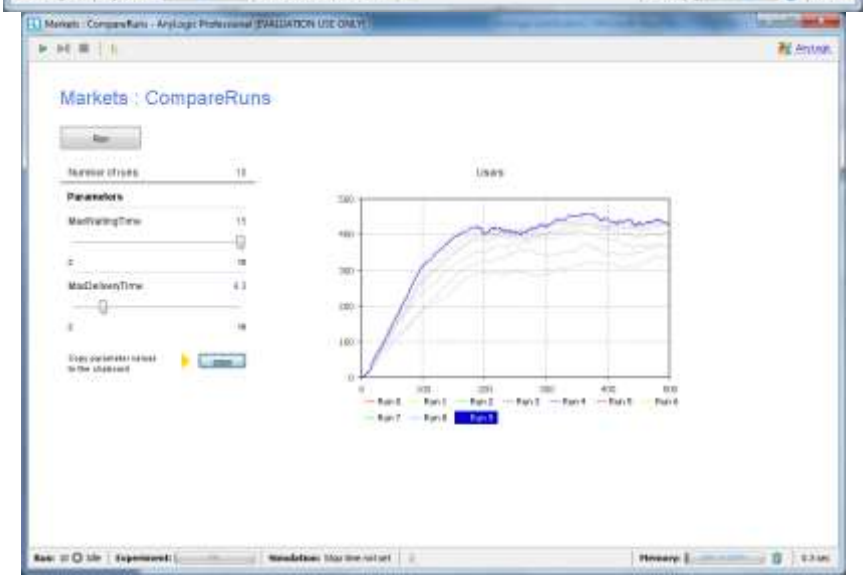
- click the **Run** button to see the result





Each curve in a chart corresponds to a specific simulation run, and you can click any item in the chart's legend to highlight the curves that correspond to the run. The controls on the left will show the values that led to this result. To deselect a curve, click on its legend a second time.





You can copy the datasets by clicking in the legend and selecting **Copy all** or **Copy selected** from the context menu.

Practical 4

AIM : Design and develop System Dynamic model by

- Creating a stock and flow diagram
- Adding a plot to visualize dynamics
- Parameter Variation
- Calibration

[Use a case scenario like spread of contagious disease for the purpose]

Solution:

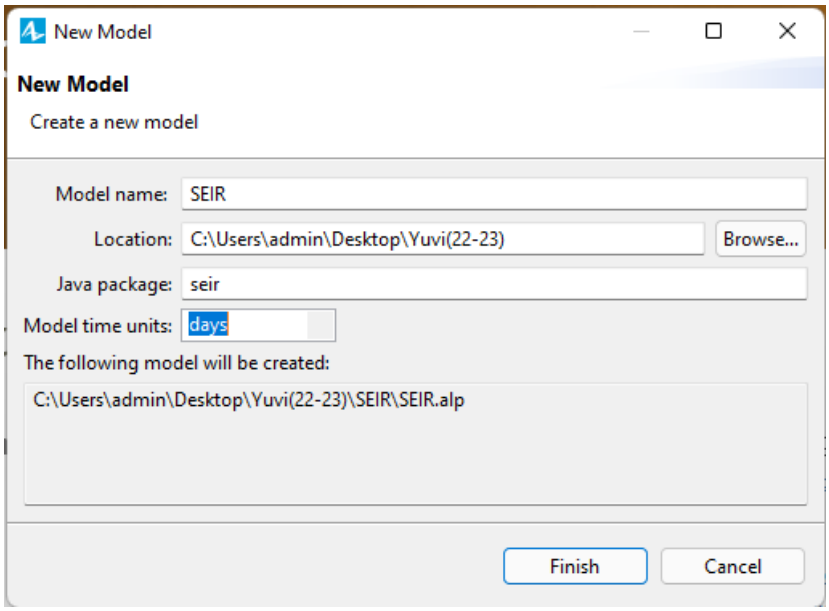
Scenario :

- Build a model that displays the spread of a contagious disease among a large population of 10,000 people i.e. TotalPopulation – of which one person is infectious.
- During the infectious phase, a person comes into contact with an average of ContactRateInfectious = 1.25 people each day.
- If an infectious person comes into contact with a susceptible person, the susceptible person's probability of infection is Infectivity = 0.6.
- After a susceptible person is infected, the infection latent phase lasts for AverageIncubationTime = 10 days. We'll use the word exposed to describe people who are in the latent phase.
- After the latent phase, infectious phase starts. This phase lasts for AverageIllnessDuration = 15 days.
- Persons who have recovered from the disease are immune to a second infection.

In this example, we'll consider four important characteristics:
• Susceptible - people who are not infected by the virus
• Exposed - people who are infected but who can't infect others
• Infectious - people who are infected and who can infect others
• Recovered – people who have recovered from the virus
SEIR is an acronym that represents the four stages: Susceptible-Exposed- Infectious-Recovered.

The flow's arrow shows its direction.
In our model, Susceptible people are exposed to the virus, become Infectious, and then recover. It's a progression that requires our model to use three flows to drive people from one stock to the next.

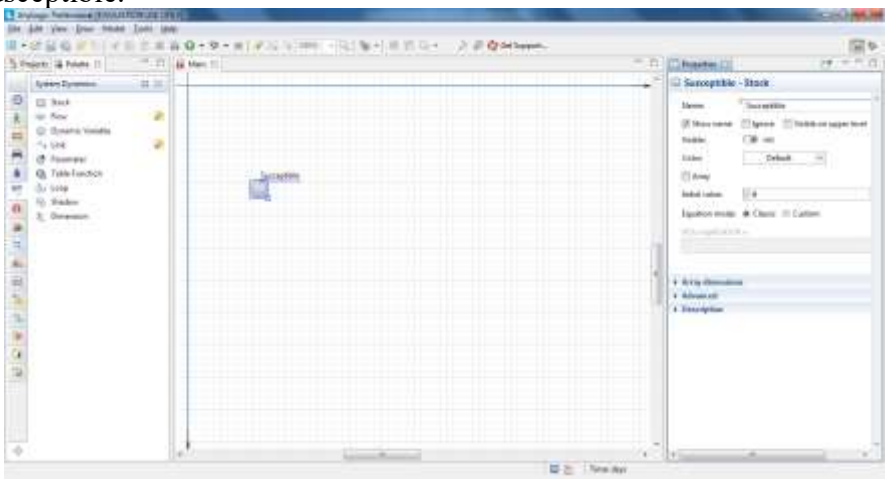
Phase 1 - Creating a stock and flow diagram:
Create a new model by selecting **File > New > Model** from the menu, and then name it SEIR. Select **days** as the **Model time units**.



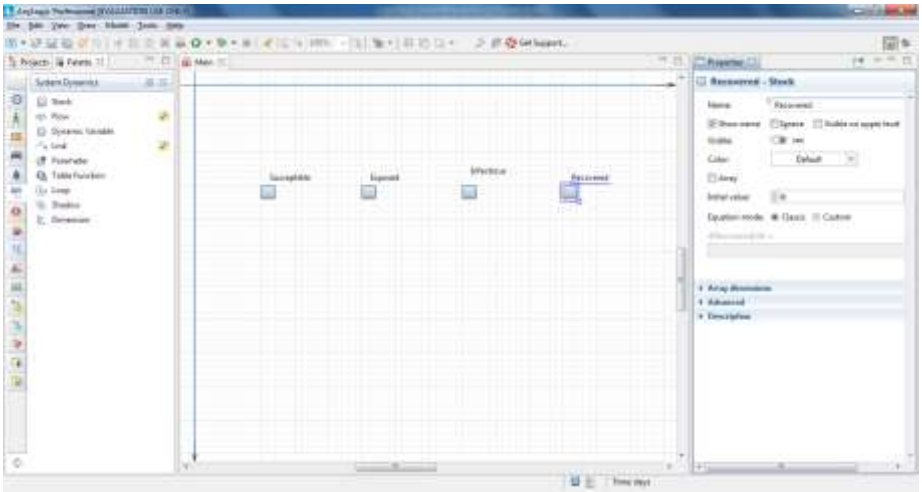
There are four stocks in our model - one for each stage.
1. Susceptible :

- Open the **System Dynamics** palette.

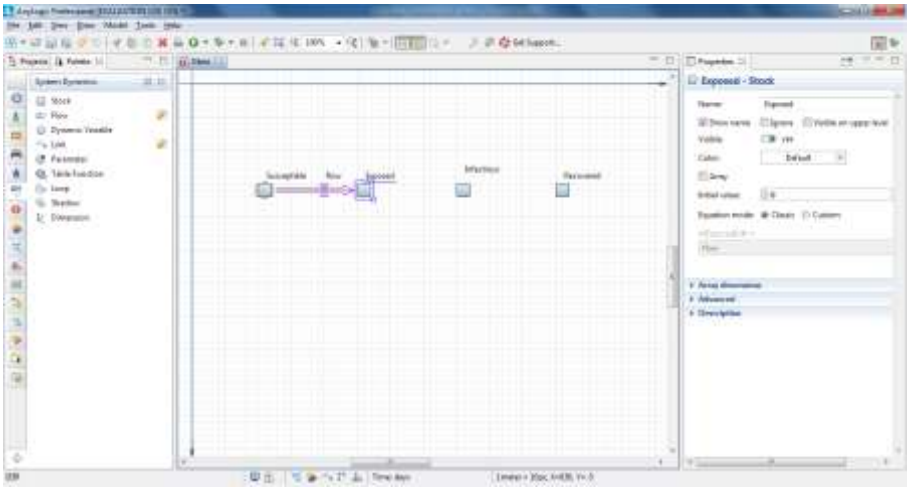
- Drag the **Stock** from the **System Dynamics** palette on to the diagram.
- Name it Susceptible.



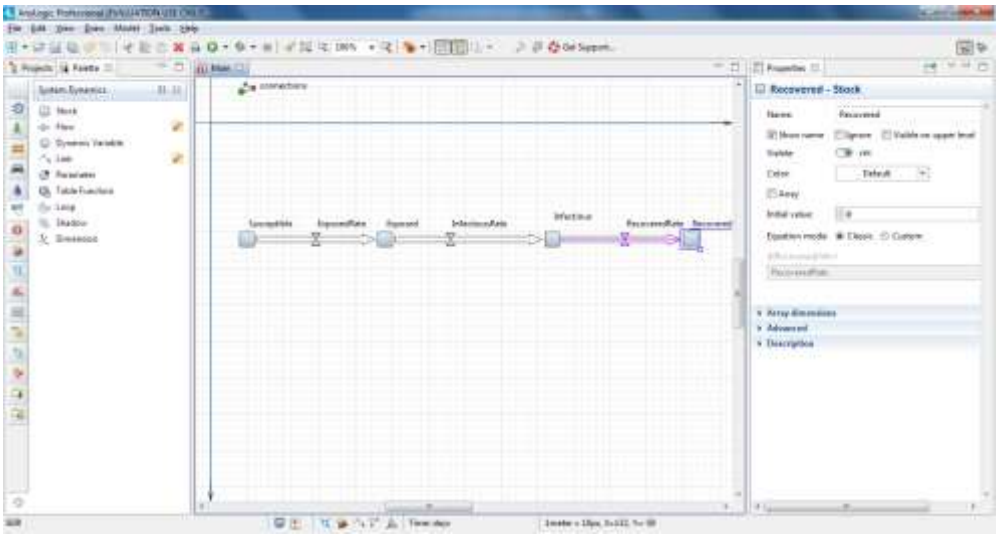
- Add three more stocks. Place them as shown in the figure and name them Exposed, Infectious, and Recovered.



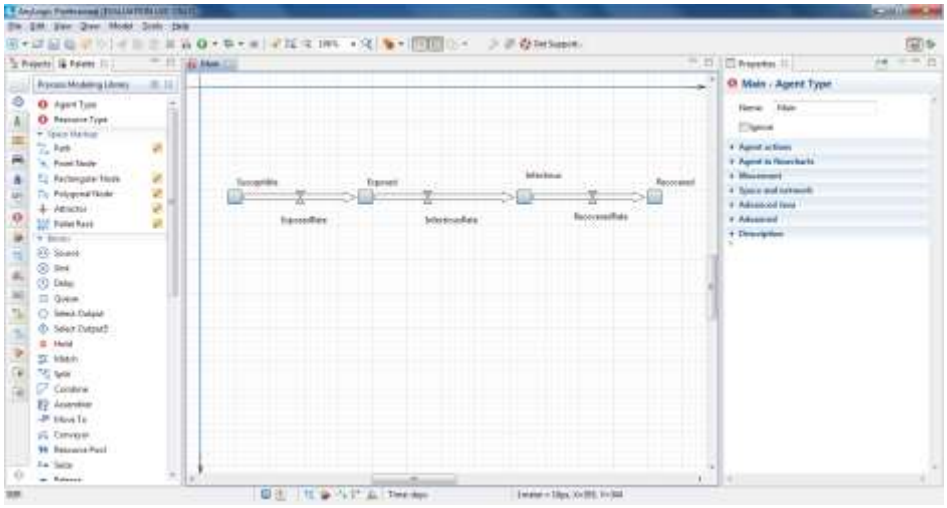
2. Add the first flow that flows from the stock Susceptible to Exposed. Double click the stock where the flow flows out (Susceptible), and then click the stock where it flows in (Exposed).



- Name the flow - ExposedRate.
- Add a flow from Exposed to Infectious, and then name it -InfectiousRate
- Add a flow from Infectious to Recovered, and then name it - RecoveredRate

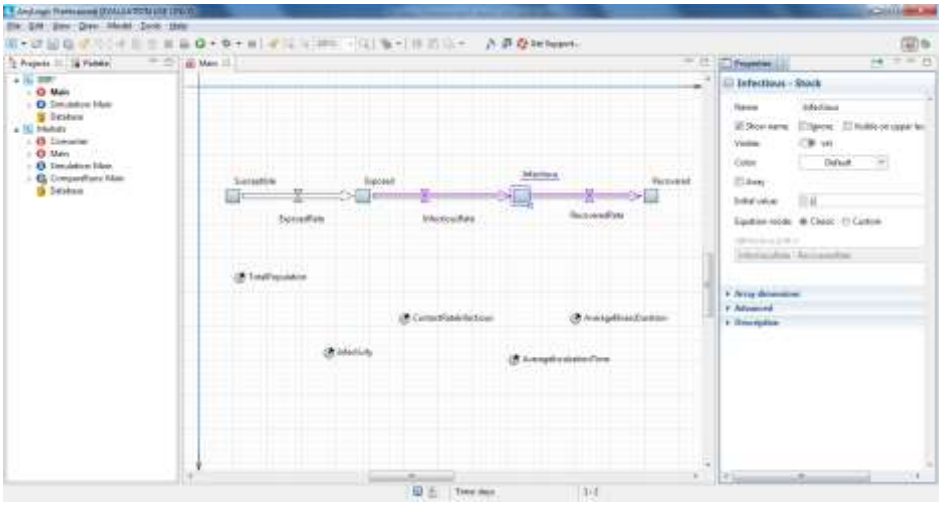


- Rearrange the flow names as shown in the figure below. To do this, select a flow and then drag its name.



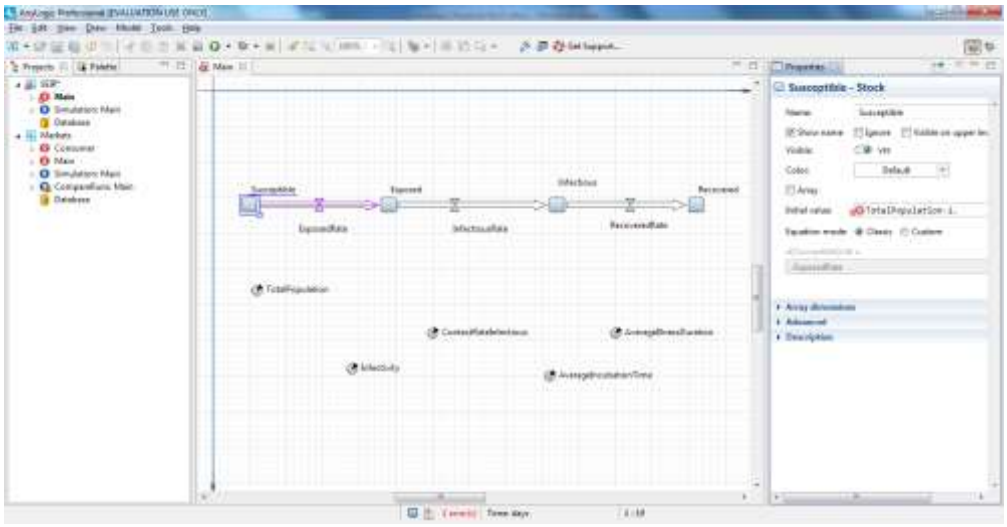
3. Add five **Parameters**, rename them, and define their default values according to the information below:

• TotalPopulation = 10000
• Infectivity = 0.6
• ContactRateInfectious = 1.25
• AverageIncubationTime = 10
• AverageIllnessDuration = 15

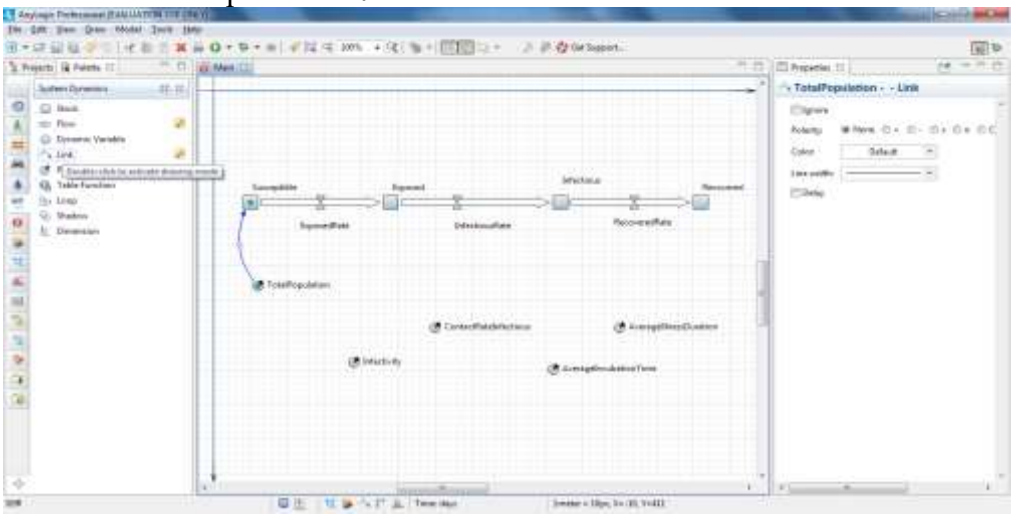


4. Define the number of infected people by specifying 1 as the **Initial Value** of the stock Infectious.

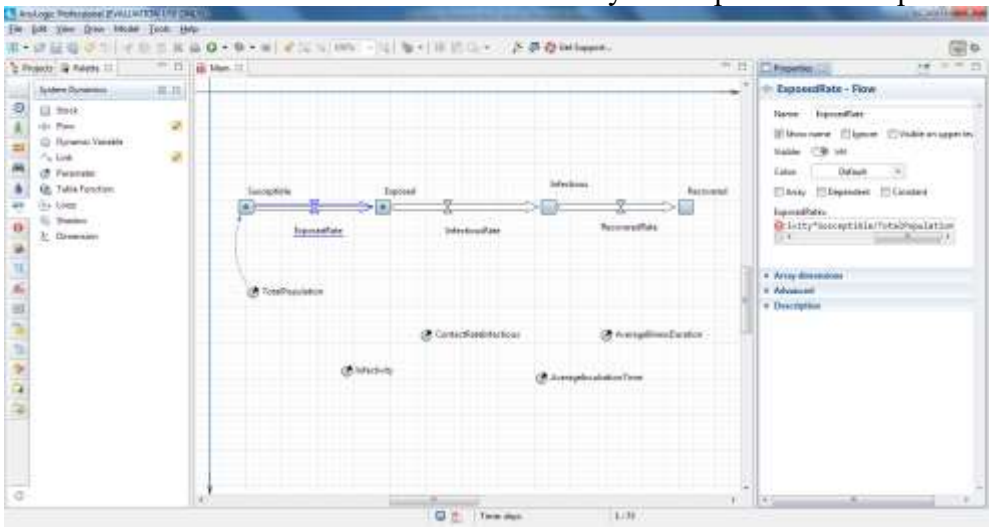
5. Define the **Initial Value** for the stock Susceptible: TotalPopulation-1.



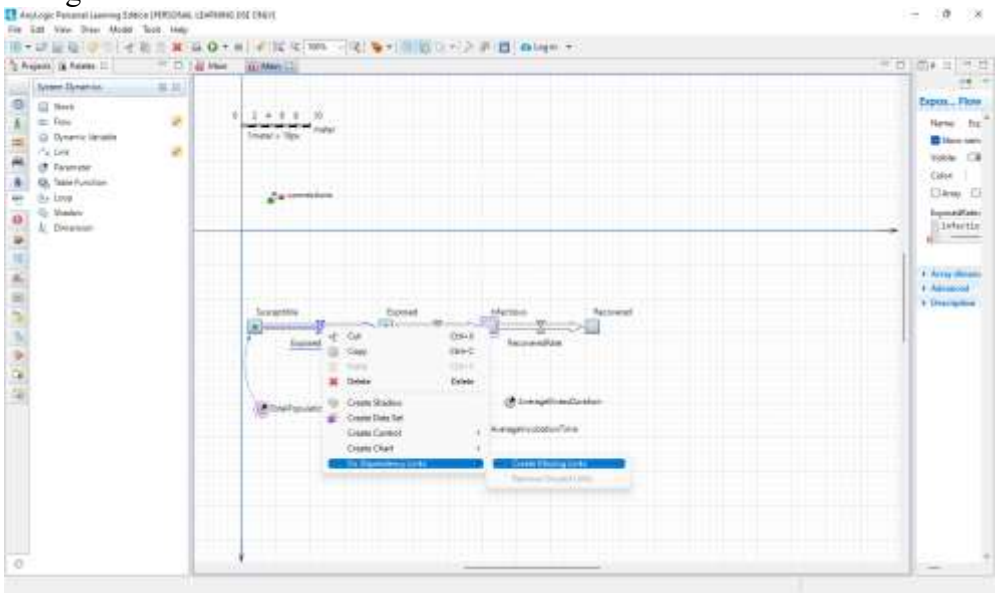
6. Draw a dependency link from TotalPopulation to Susceptible:
   - In the **System Dynamics** palette, double-click the **Link** element, click TotalPopulation, and then click the stock Susceptible. You should see the link with small circles drawn on its end points formula for the flow ExposedRate**.**

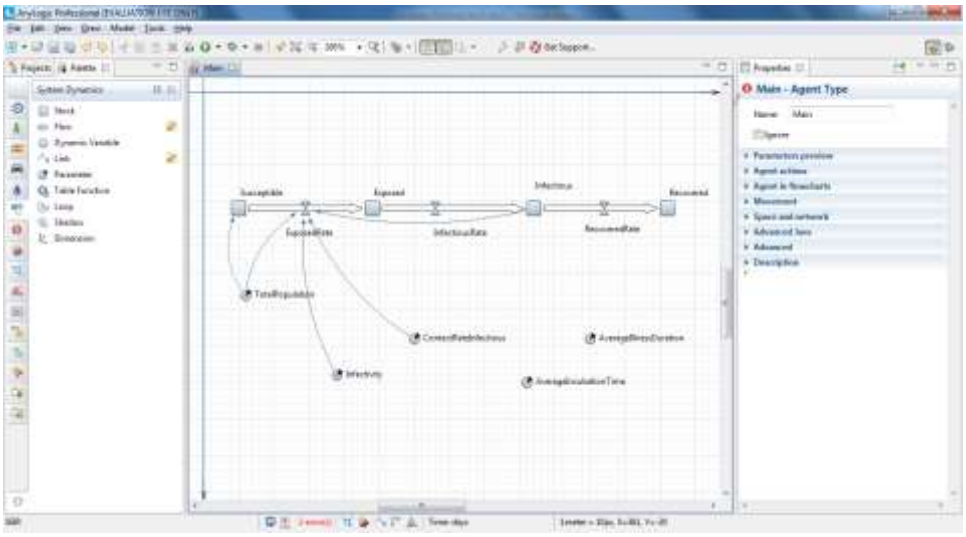


   - Click the flow ExposedRate and define the following formula using the Code Completion assistant : Infectious*ContactRateInfectious*Infectivity*Susceptible/TotalPopulation

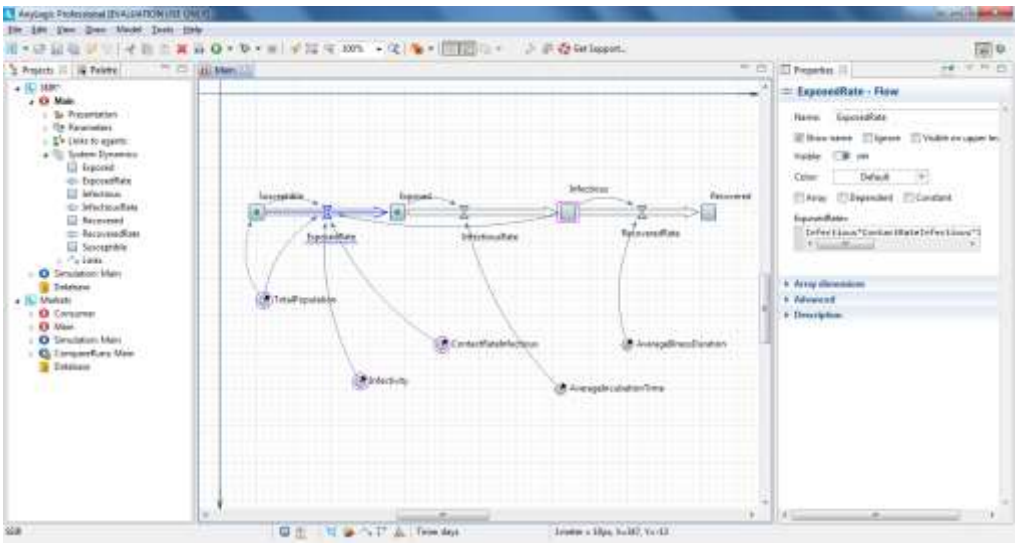


- Right-click ExposedRate flow in the graphical diagram, and choose **Fix Dependancy Links > Create Missing Links** from the context menu. Afterward, you should see the links in the stock and flow diagram:

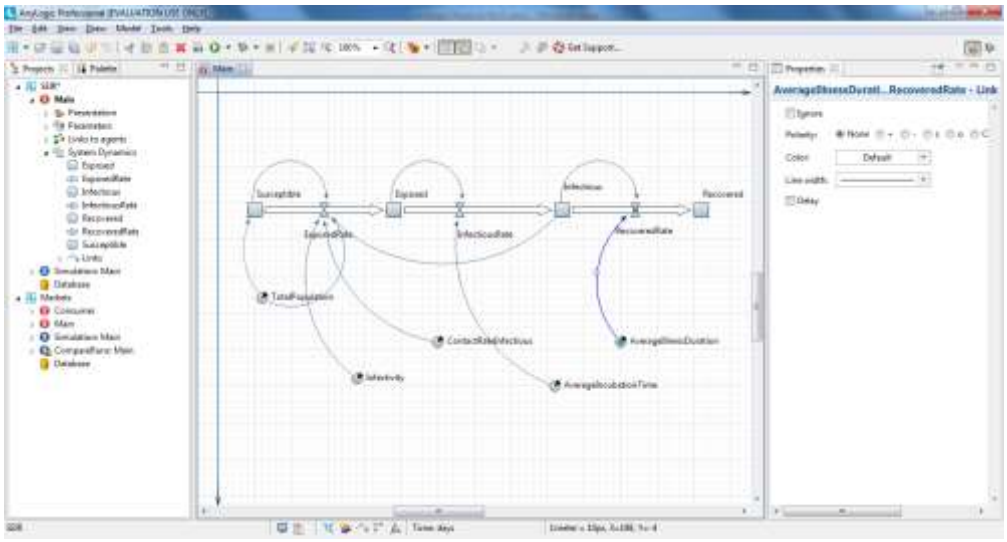




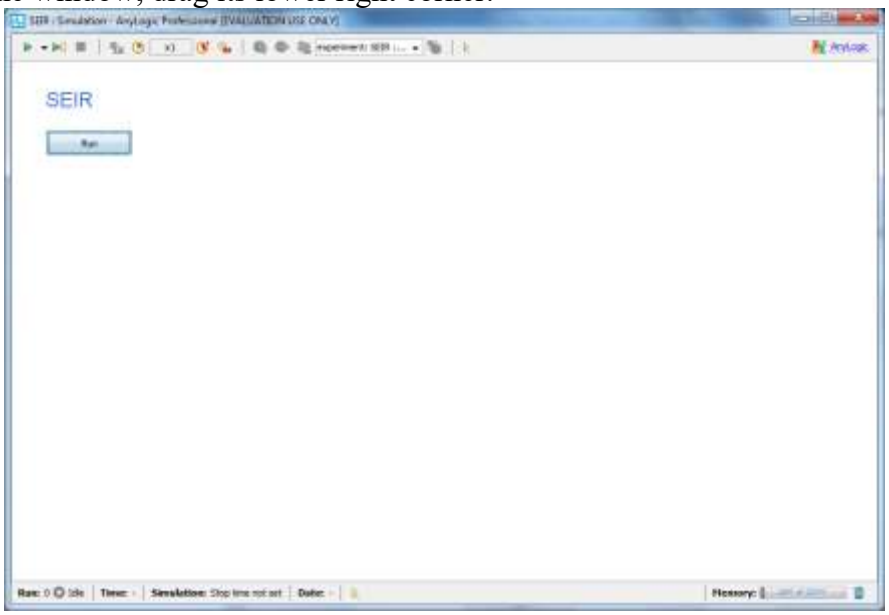
7. Define the following formula for InfectiousRate: Exposed/AverageIncubationTime
8. Define the following formula for RecoveredRate: Infectious/AverageIllnessDuration
9. Draw the missing dependency links, and your stock and flow diagram should resemble the following image:

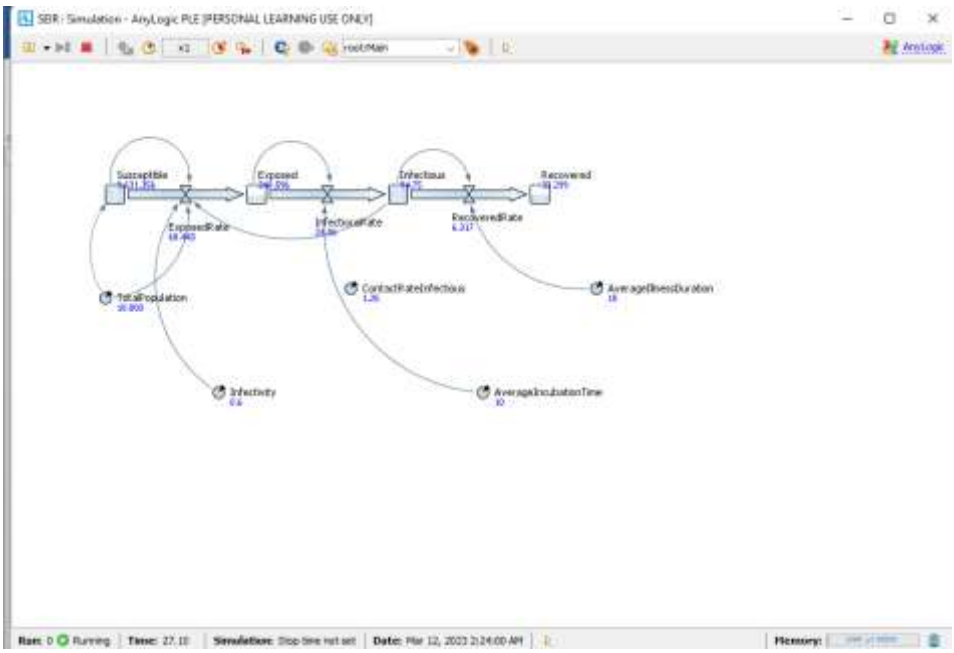


- Adjust the appearance of dependency links. Modify the links' bend angles to make the diagram match the figure below. To adjust the link's bend angle, select it, and then drag the handle in the middle of the link.



10. Run the model and inspect the dynamics using the variables' inspect windows.
    - To open a variable's inspect window, click the variable to select it.
    - To resize the window, drag its lower right corner.





Practical 5

AIM : Design and develop a discrete-event model that will simulate process by:

- Creating a simple model
- Adding resources
- Creating 3D animation
- Modeling delivery

Scenario :

Our goal is to create a discrete-event model that will simulate a small job shop's manufacturing and shipping processes. The raw materials that are delivered to the receiving dock are placed into storage until processing takes place at the CNC machine.

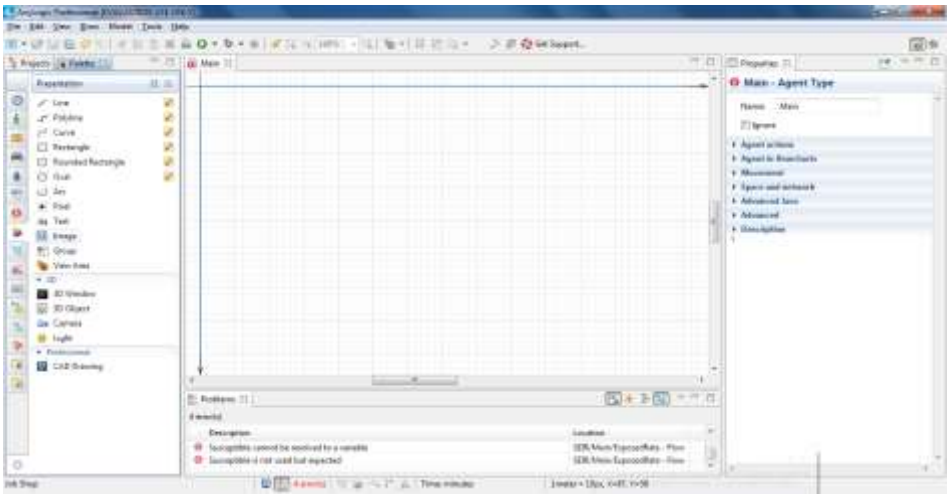
- *Phase 1*. Creating a simple model to simulate the pallets' arrival at the job shop and their storage at the shipping dock.
- *Phase 2*. Expanding the model by adding forklift trucks to store the pallets in the pallet rack and then move them to the production area.
- *Phase 3*. Adding 3D animation.
- *Phase 4*. Adding trucks that deliver pallets to the job shop.
- *Phase 5*. Modeling CNC machines where the raw materials are processed.

Solution:

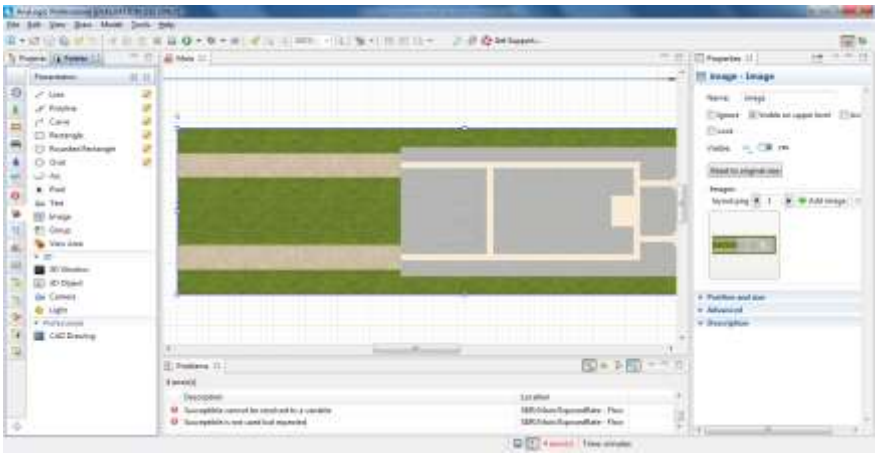
➢ **Create a new model.**

1. In the **New Model** wizard,
   - set the **Model name**: Job Shop, and
   - **Model time units**: **minutes**.

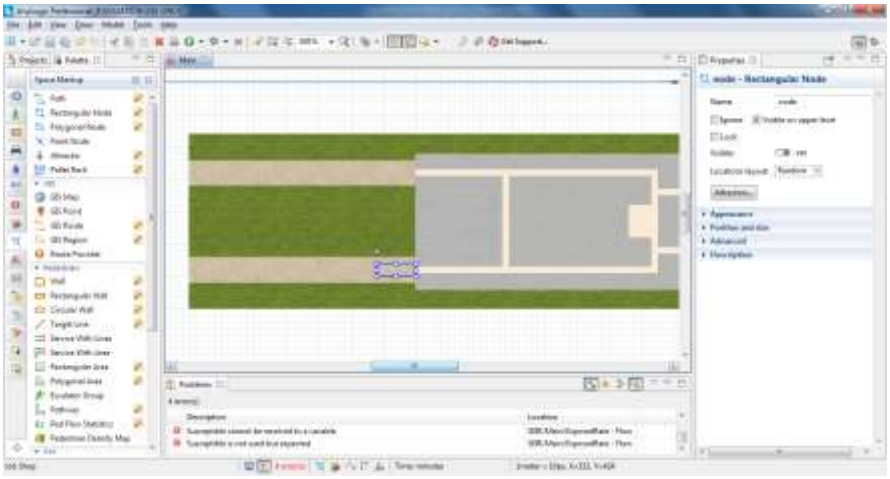
2. Open the **Presentation** palette. The palette has several shapes that you can use to draw model animation, including a rectangle, a line, an oval, a polyline and a curve.



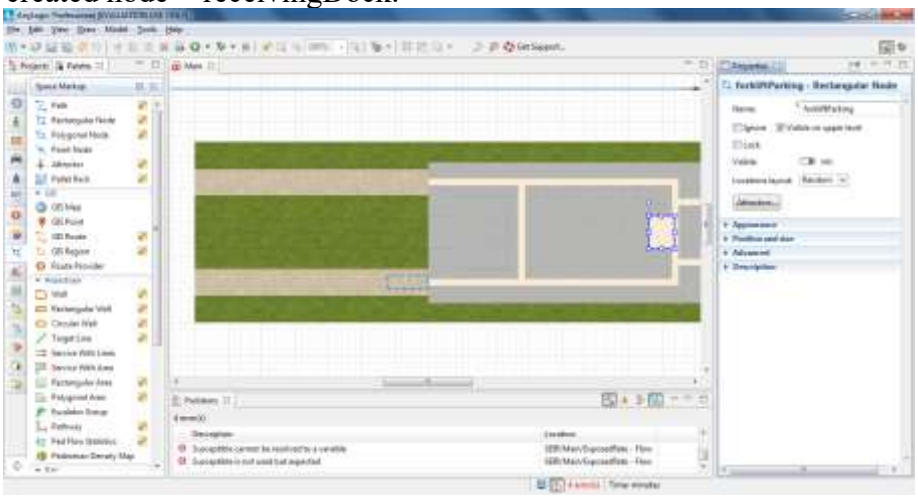
- On the **Presentation** palette, select the **Image** shape and then drag it on to the Main diagram. [You can use the **Image** shape to add images in several graphic formats -- including PNG, JPEG, GIF, and BMP – to your presentation.]
- The dialog box will appear that prompts you to choose the image file the shape will display.
- Browse to the following location and then select the layout.png image: AnyLogic folder/resources/AnyLogic in 3 days/Job Shop



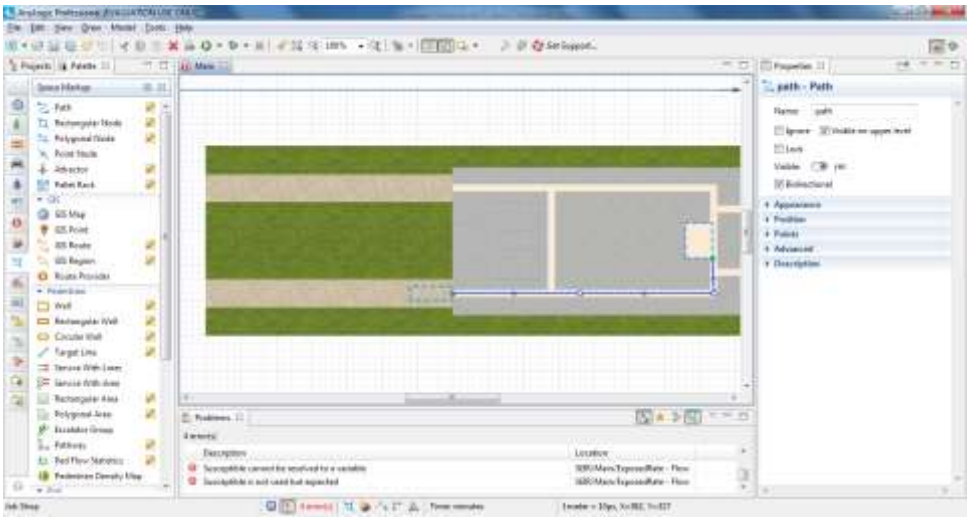
3. Select the image in the graphical editor. In the **Properties** view, select the **Lock** checkbox to lock the image.
4. Open the **Space Markup** palette, and drag the **Rectangular Node** element on to the Main diagram. Resize the node. The node should look as in the figure below.



- Name the created node - receivingDock.

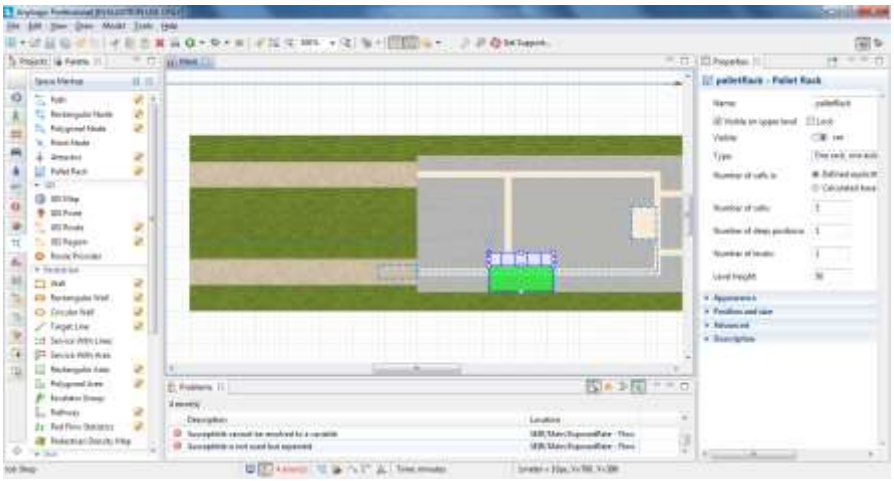


5. Use another **Rectangular Node** to draw the parking area as shown in the figure below and then name this node - forkliftParking.
6. Do the following to draw a movement path that will guide our model's forklift trucks:
   a. In the **Space Markup** palette, double-click the **Path** element to activate its drawing mode.
   b. Draw the path as shown in the figure below by
      - clicking the receivingDock border,
      - clicking in the diagram to add the path's turning point, and
      - then clicking the forkliftParking node's border.

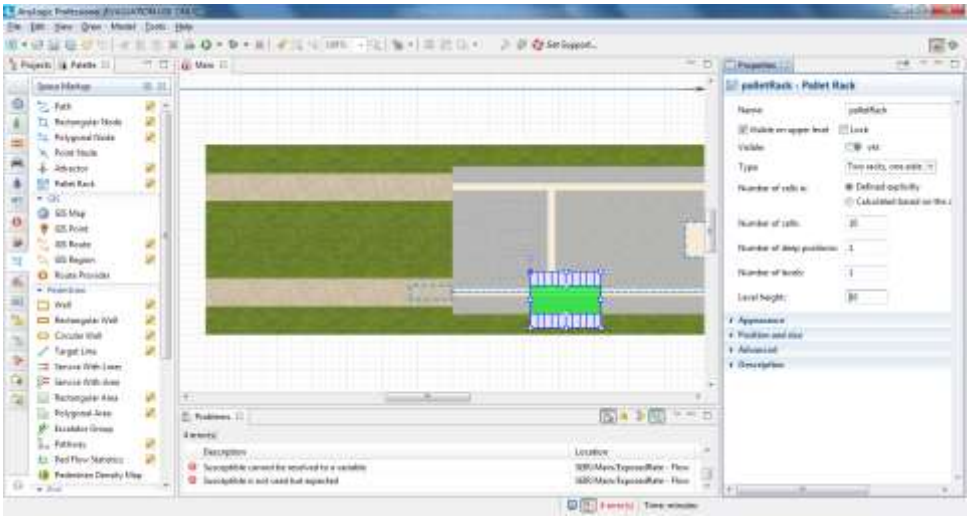


If you've successfully connected the nodes, the path's connection points will display cyan highlights each time you select the path.

7. Define your model's warehouse storage by dragging the **Pallet Rack** element from the **Space Markup** palette on to the layout and placing its aisle on the path. A correctly-placed pallet rack will display a green highlight that shows it is connected to the network.

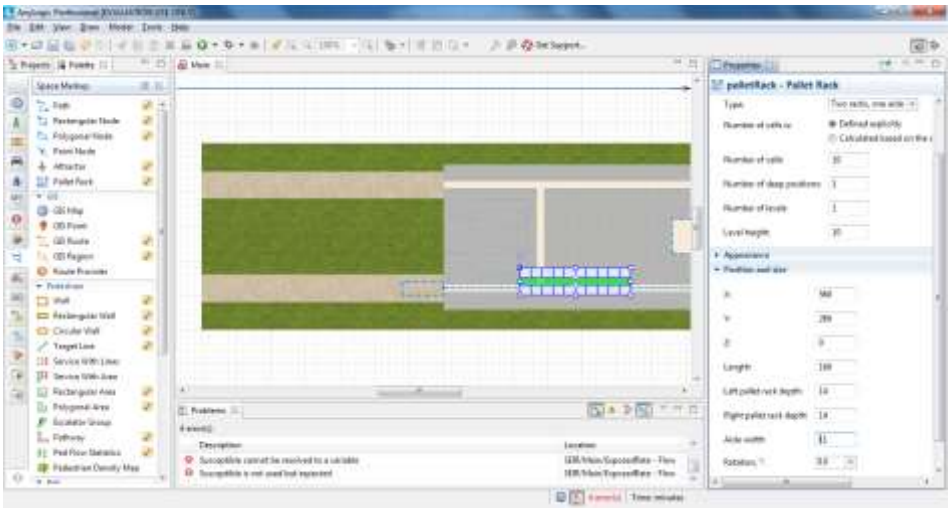


8. In the pallet rack's **Properties** area, do the following:
    a. Set **Type** to: two racks, one aisle
    b. **Number of cells**: 10
    c. **Level height**: 10

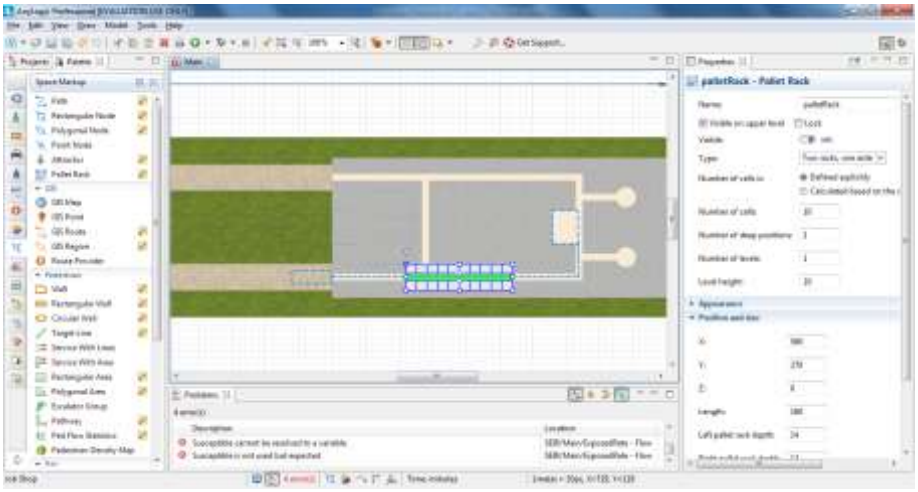


9. In the **Position and size** section:
    a. **Length**: 160
    b. **Left pallet rack depth**: 14

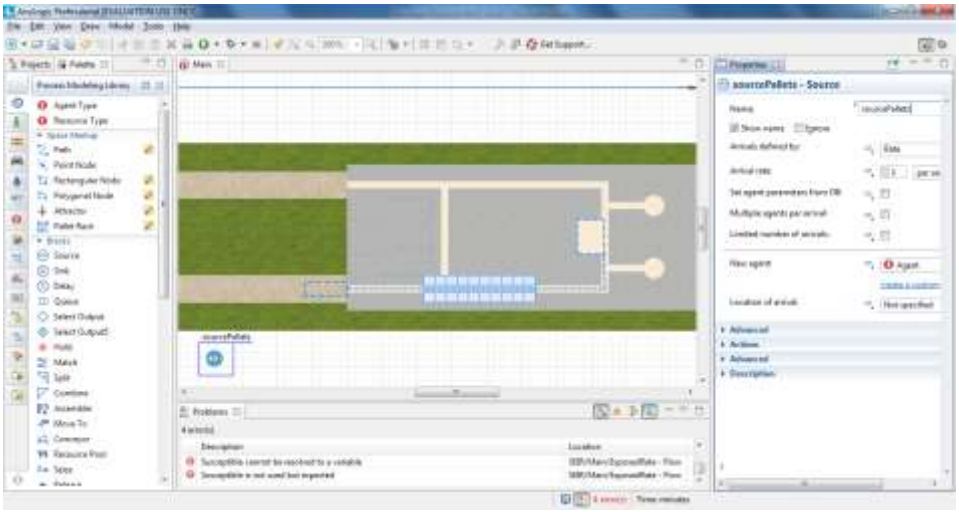
c. **Right pallet rack depth**: 14
d. **Aisle width**: 11



10. After you've completed these changes, the pallet rack should resemble the pallet rack shown in the figure below. If necessary, move the pallet rack so that its center aisle lies on the path. Make sure the pallet rack is connected to the network by clicking it twice to select it. Your first click will select the entire network, and the second will select the pallet rack. The pallet rack should display a green highlight that shows it is connected to the network.



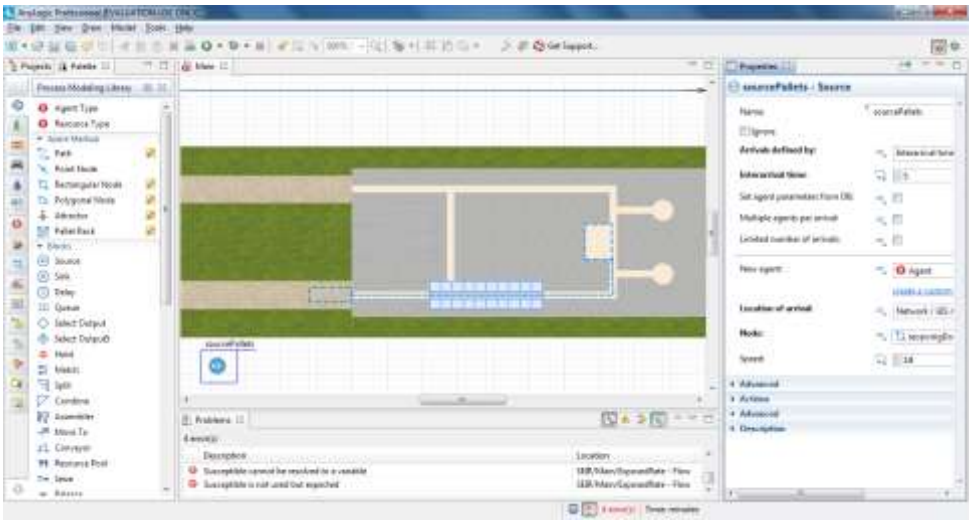
11. Drag the **Source** element from the **Process Modeling Library** palette on to the graphical diagram and name the block - sourcePallets.



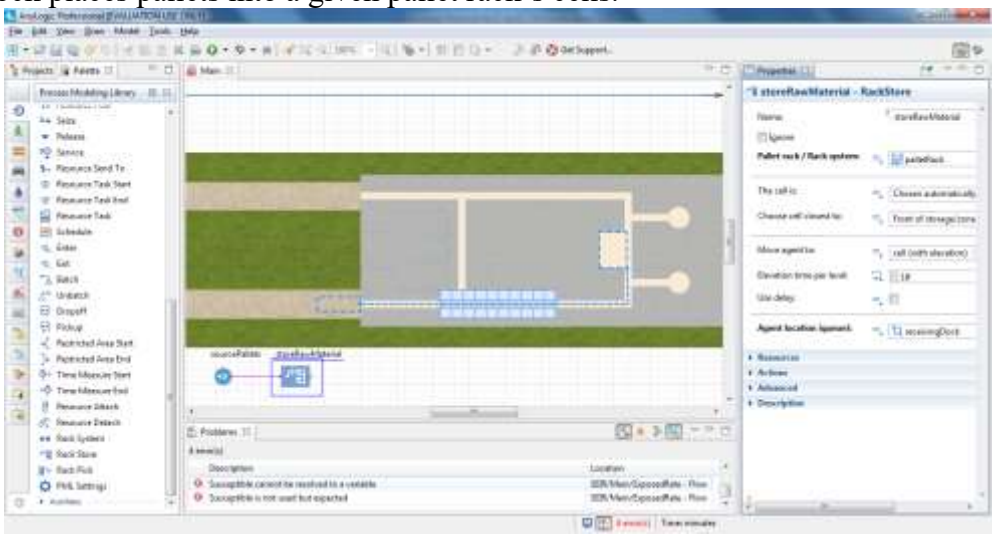
While the **Source** block usually acts as a process starting point, our model will use it to generate pallets.

12. In the sourcePallets block's **Properties** area, do the following to ensure the model's pallets arrive every five minutes and appear in the receivingDock node.

a. In the **Arrivals defined by** area, click **Interarrival time**.
b. In the **Interarrival time** box, type 5, and select **minutes** from the list on the right to have pallets arrive every five minutes.
c. In the **Location of arrival** area, click **Network / GIS node** in the list.
d. In the **Node** area, click receivingDock in the list.



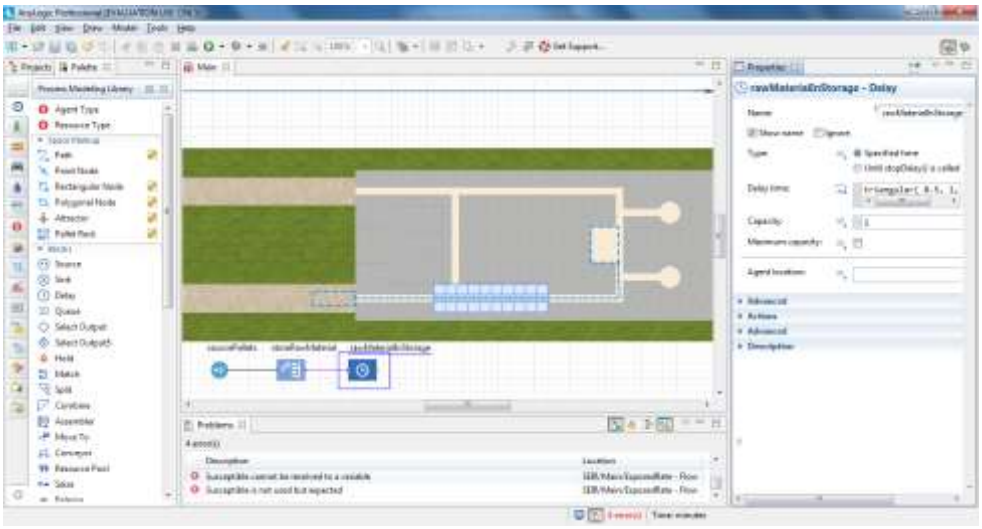
13. Drag the **RackStore** block from the **Process Modeling Library** palette on to the diagram and place it near the sourcePallets block so they are automatically connected as shown in the diagram below. The **RackStore** block places pallets into a given pallet rack's cells.



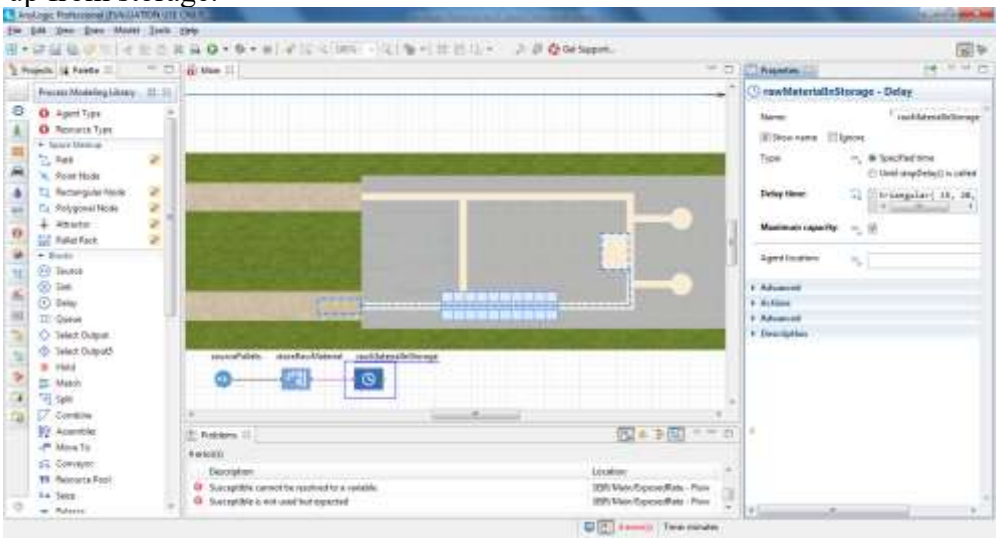
14. In the rackStore block's **Properties** area, do the following:

a. In the **Name** box, type storeRawMaterial.
b. In the **Pallet rack / Rack system** list, click palletRack.
c. In the **Agent location (queue)** list, click receivingDock to specify the location where agents wait to be stored.

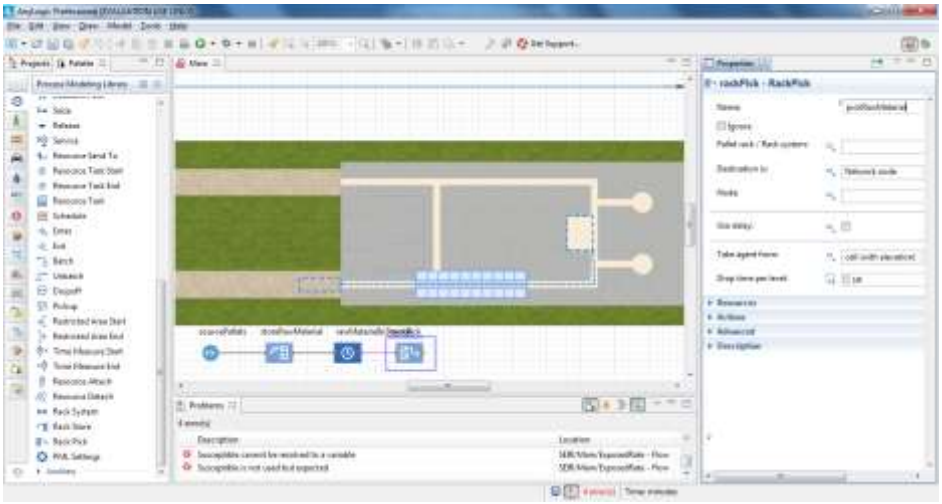
15. Add a **Delay** block to simulate how pallets wait in the rack and then name the block - rawMaterialInStorage.



16. In the rawMaterialInStorage block's **Properties** area, do the following:
    a. In the **Delay time** box, type triangular(15, 20, 30) and select **minutes** from the list.
    b. Select the **Maximum capacity** checkbox to ensure agents will not get stuck as they wait to be picked up from storage.

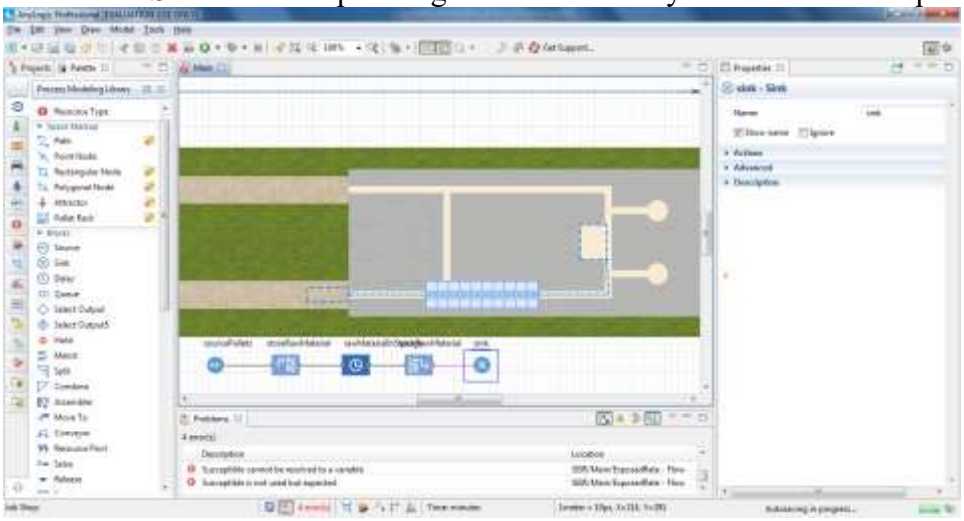


17. Add a **RackPick** block, connect it to the flowchart, and then name it pickRawMaterial.

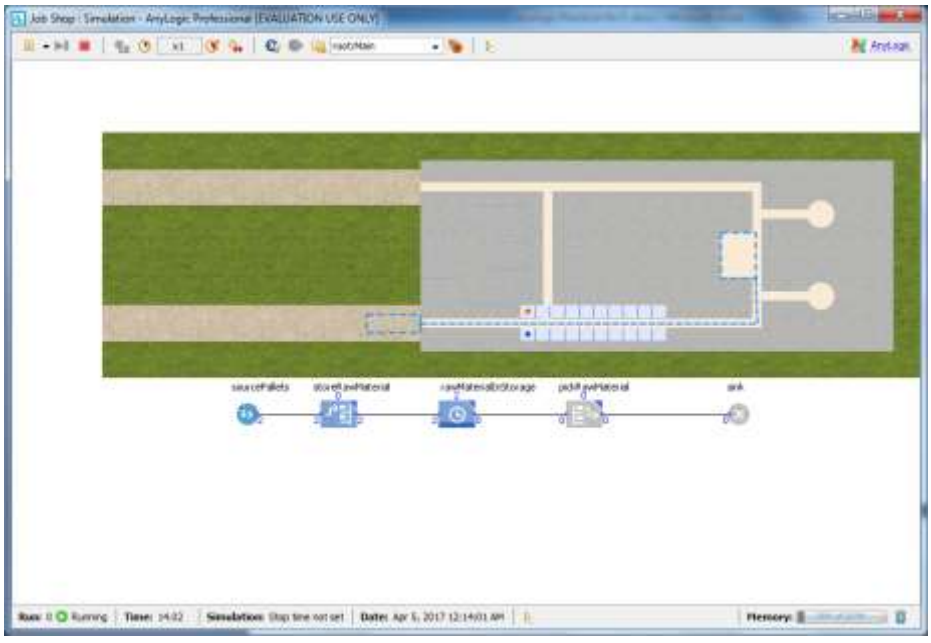


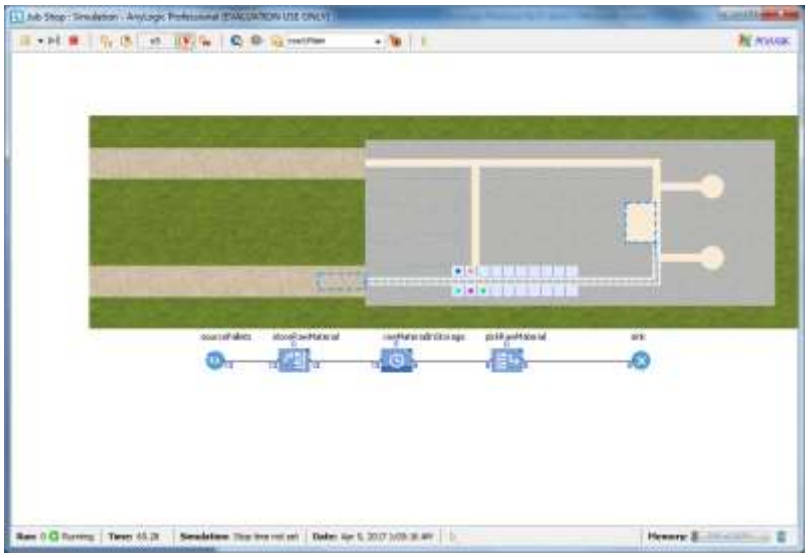
18. In the pickRawMaterial block's **Properties** area, do the following:
    a. In the **Pallet rack / Rack system** list, click palletRack to select the pallet rack that will provide pallets to agents.
    b. In the **Node** list, click forkliftParking to specify where the agents should park forklift trucks.

19. Add a **Sink** block. The **Sink** block disposes agents and is usually a flowchart's end point.



20. Run the model (Job Shop / Simulation experiment).

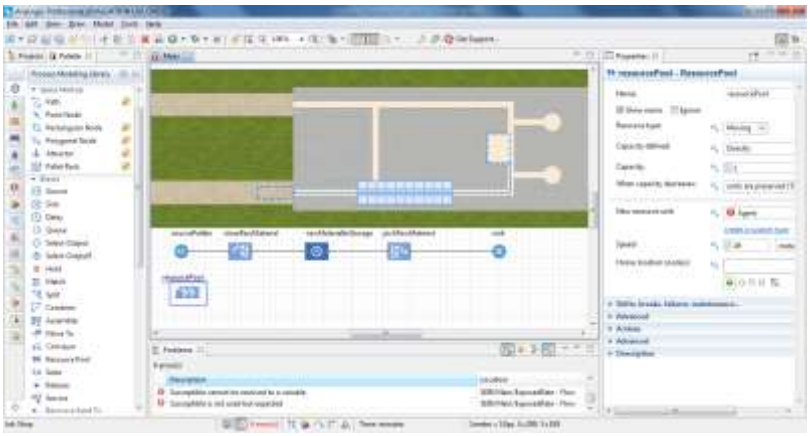




- Adding resources

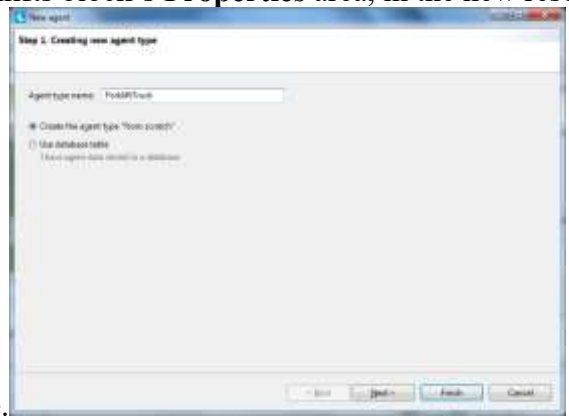
[Some examples of resources include:
• A hospital model's doctors, nurses, equipment, and wheelchairs
• A supply chain model's vehicles and containers
• A warehouse model's forklift trucks and workers
There are three types of resources: *static*, *moving*, and *portable*.
• Static resources are bound to a specific location, and they cannot move or be moved.
• Moving resources can move independently.
• Portable resources can be moved by agents or by moving resources.]
Our model's resources are the forklift trucks that move pallets from the unloading zone to a pallet rack and then deliver pallets from the rack to the production zone.
1. On the **Process Modeling Library** palette,

- drag the **ResourcePool** block on to the Main diagram. You do not have to connect the block to the flowchart.

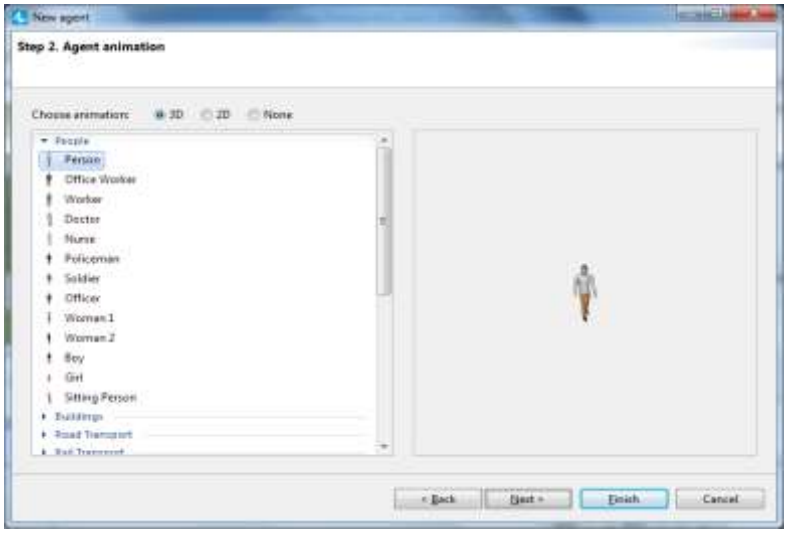


- Name the **ResourcePool** block – forklifts

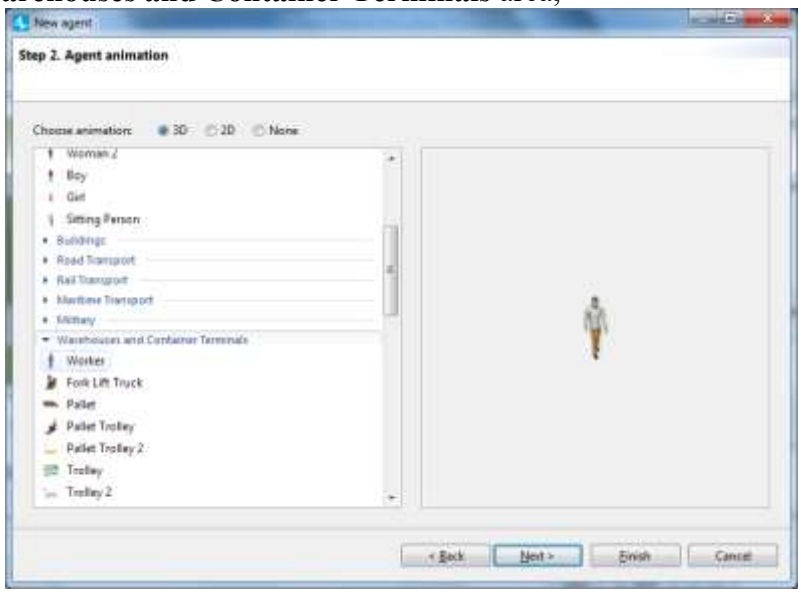
2. In the forklifts block's **Properties** area, in the new resource unit->click the button **create a custom type**.



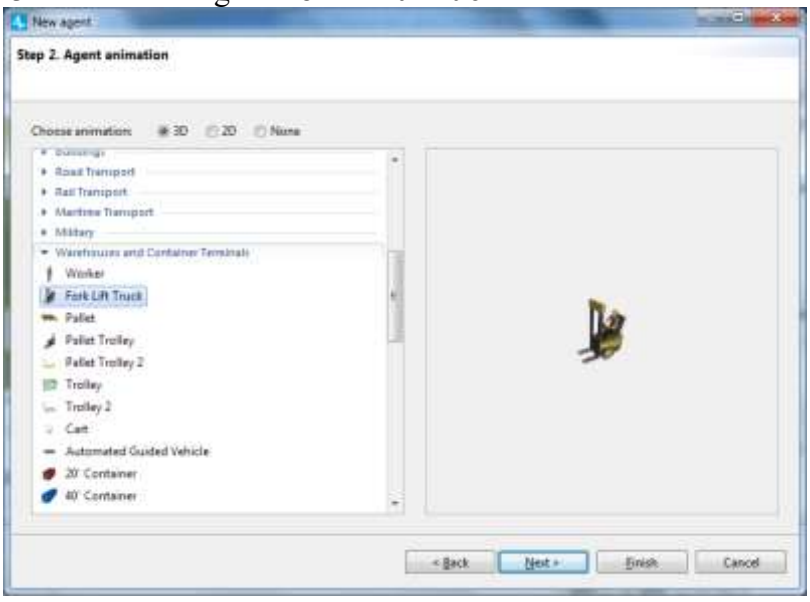
- In the **New agent** wizard:
  a. In **Agent type name** box, type ForkliftTruck.
  b. Click **Next**.



- Expand the **Warehouses and Container Terminals** area,

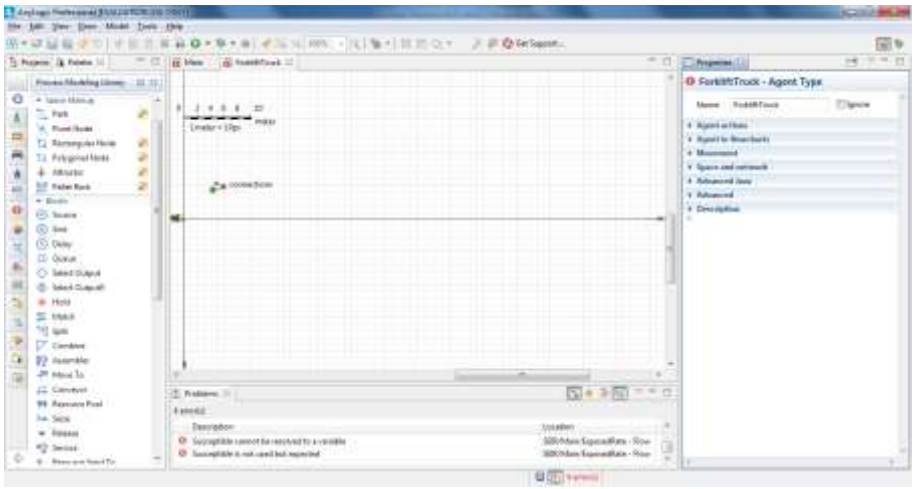


- Then click the 3D animation figure **Fork Lift Truck**.

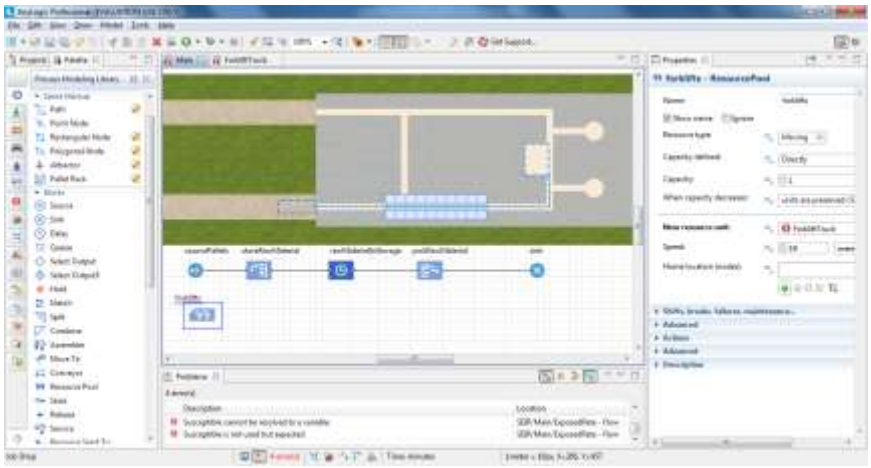


- Click **Finish**.

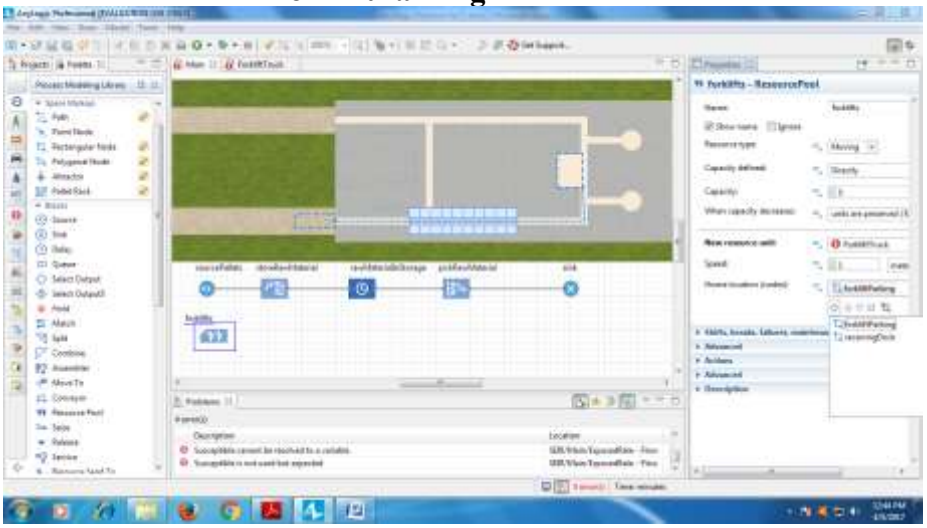
3. The ForkliftTruck agent type diagram will open and display the animation shape you selected in the wizard.



You'll see the ForkliftTruck resource type has been selected in the **ResourcePool** block's **New resource unit** parameter.

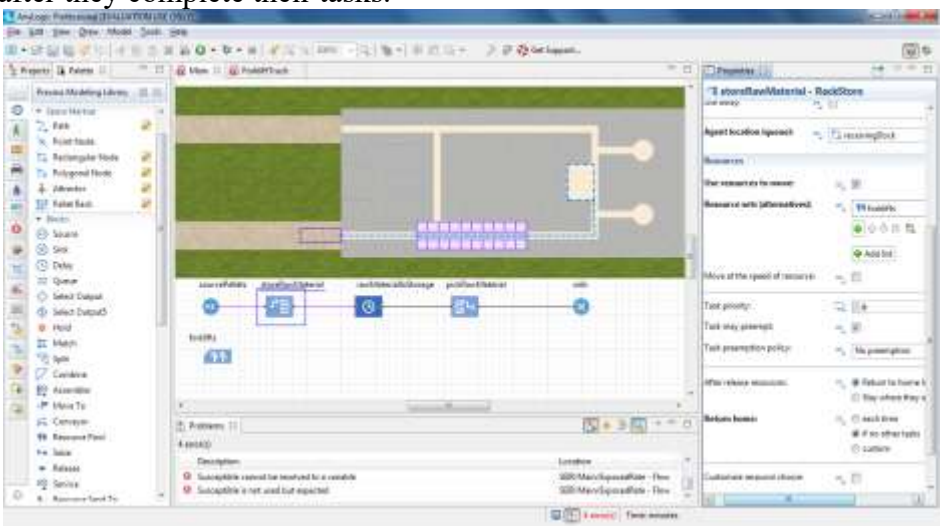


4. Modify the forklifts resource type's other parameters:
   a. In the **Capacity** box, type **5** to set the number of forklift trucks in our model.
   b. In the **Speed** box, type **1** and choose **meters per second** from the list on the right.
   c. In the **Home location (nodes)** area, select the forkliftParking node.
5. Click the plus button and then click **forkliftParking** in the list of the model's nodes.

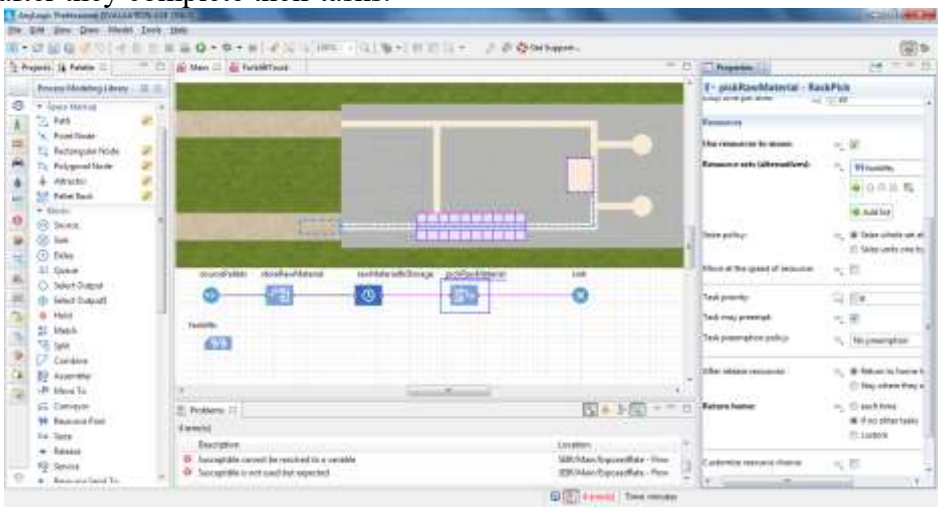


6. In the storeRawMaterial block's **Properties** area, do the following:
   a. Click the arrow to expand the **Resources** area.
   b. Select the **Use resources to move** check box.
   c. In the **Resource sets (alternatives)** list, select **forklifts** to ensure the flowchart block uses the selected resources -- in our case, the forklift trucks -- to move the agents. Click the plus button and then click **forklifts** in the list of the model's resources.

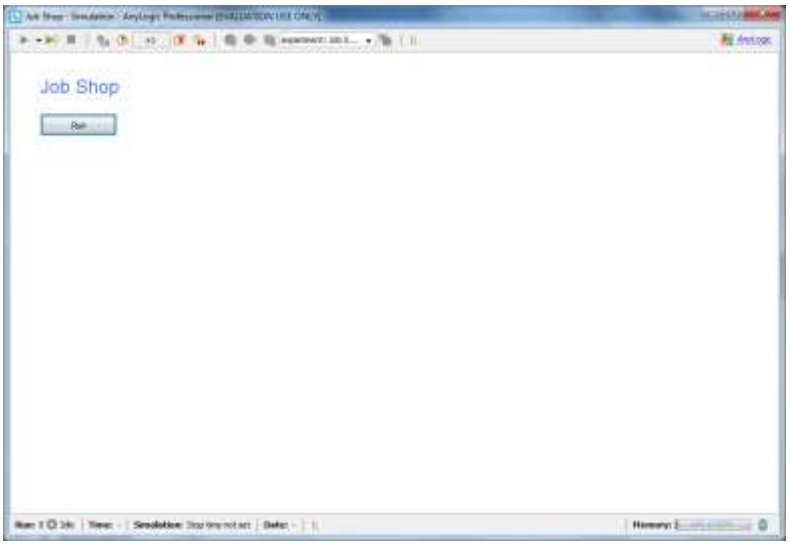
d. In the **Return home** area, select **if no other tasks** to ensure the forklift trucks return to their home location after they complete their tasks.

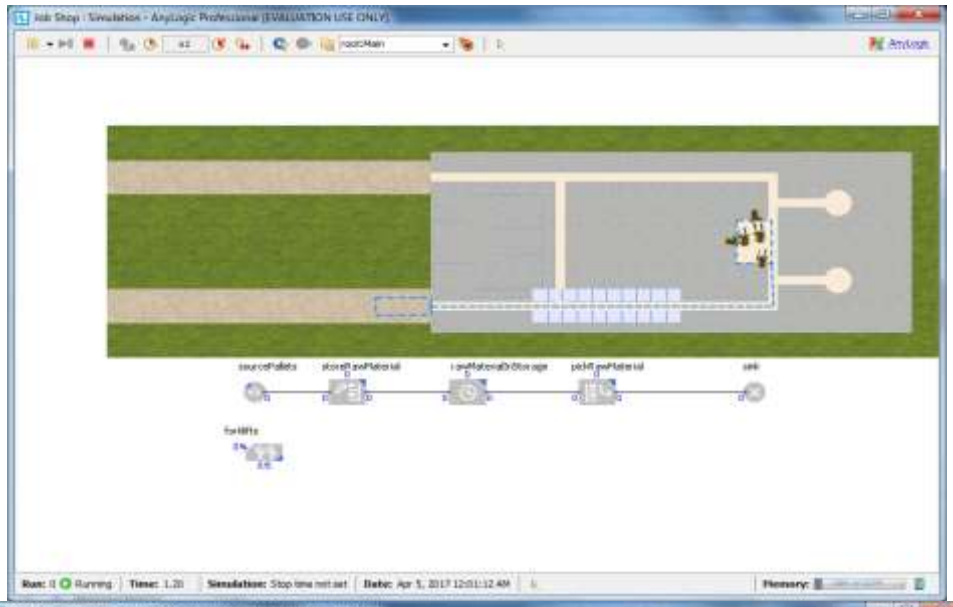


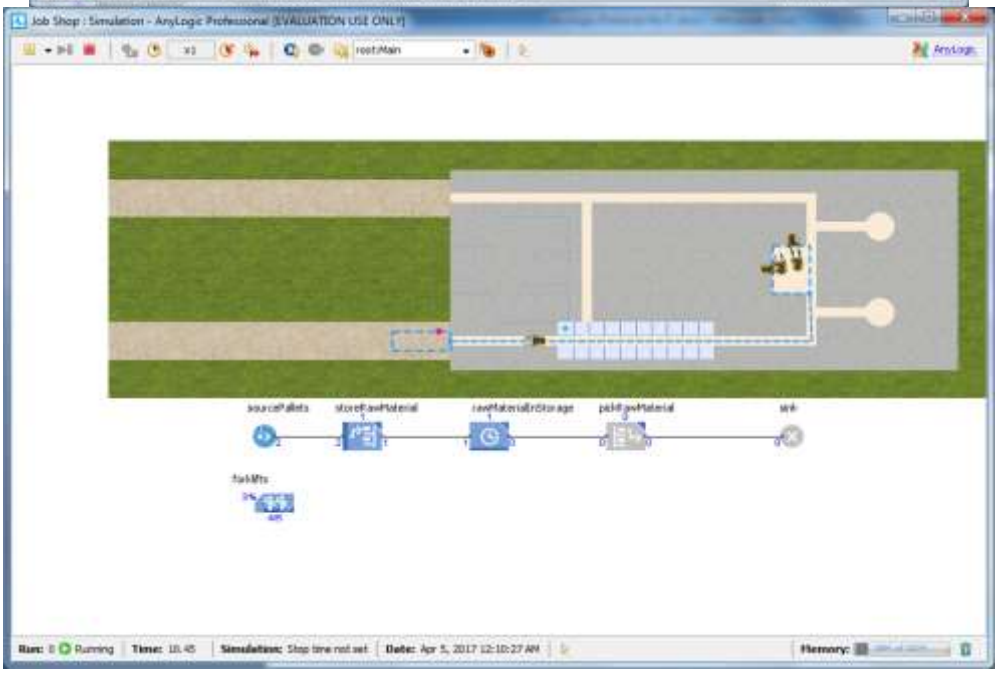
7. In the pickRawMaterial block's **Properties** area, do the following:
   a. Click the arrow to expand the **Resources** area.
   b. Select the **Use resources to move** check box.
   c. In the **Resource sets (alternatives)** list, click **forklifts** to ensure the flowchart block uses the forklift trucks to move the agents.
   d. In the **Return home** area, click **if no other tasks** to ensure the forklift trucks return to their home location after they complete their tasks.

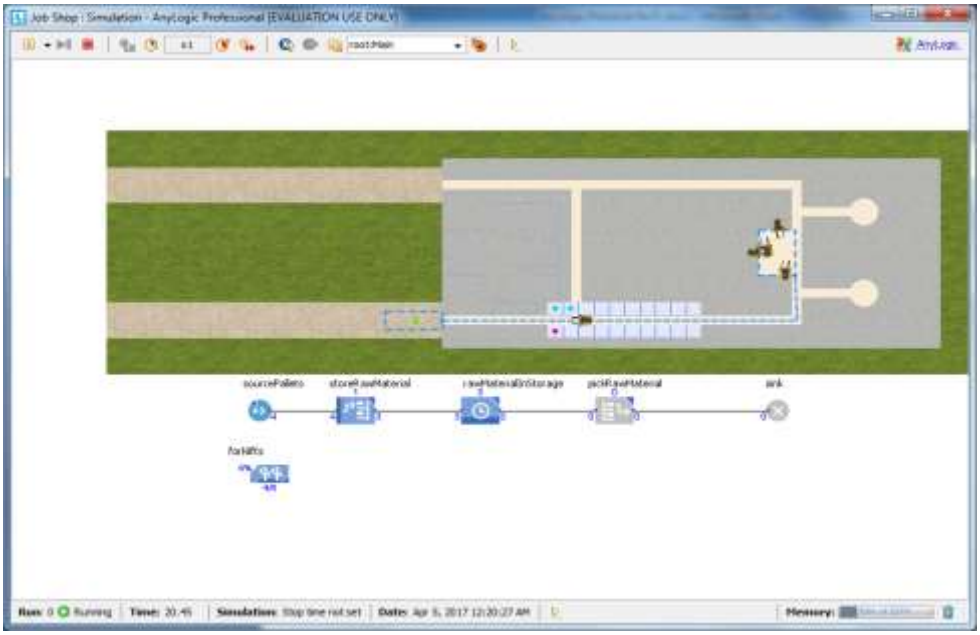


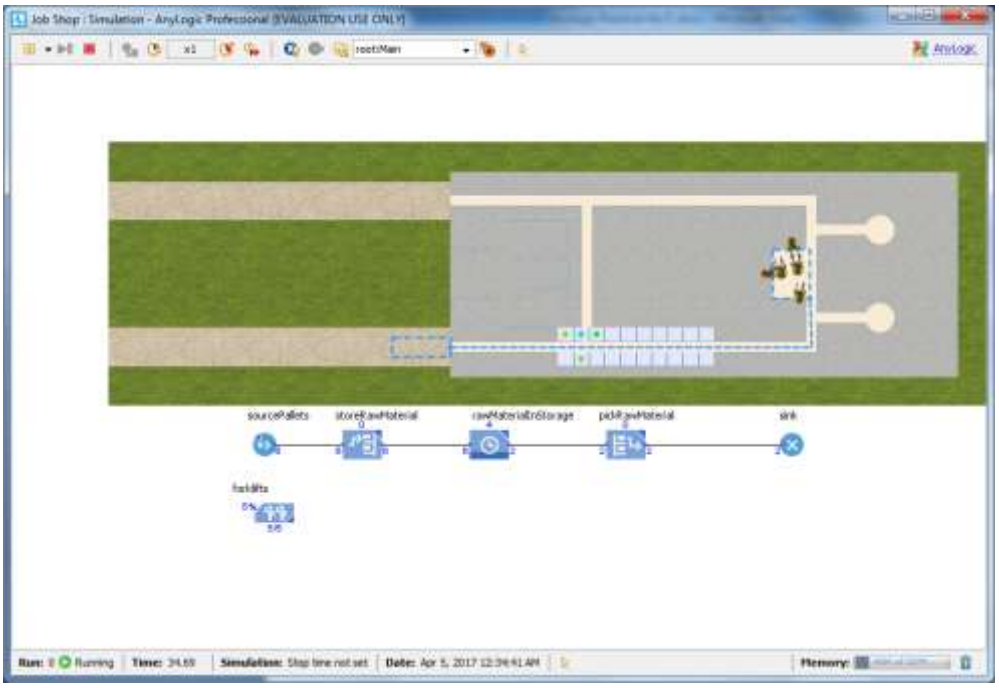
8. Run the model.

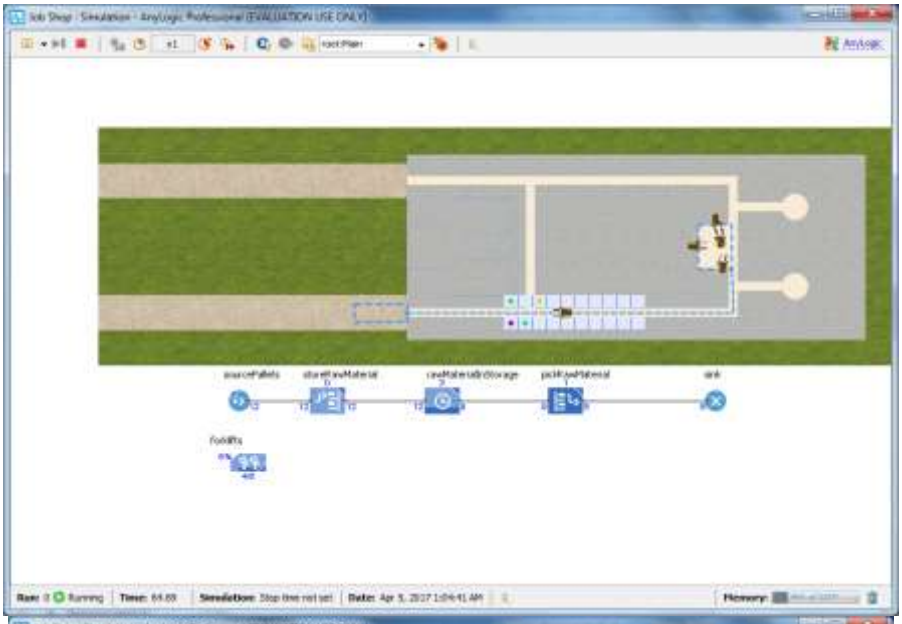


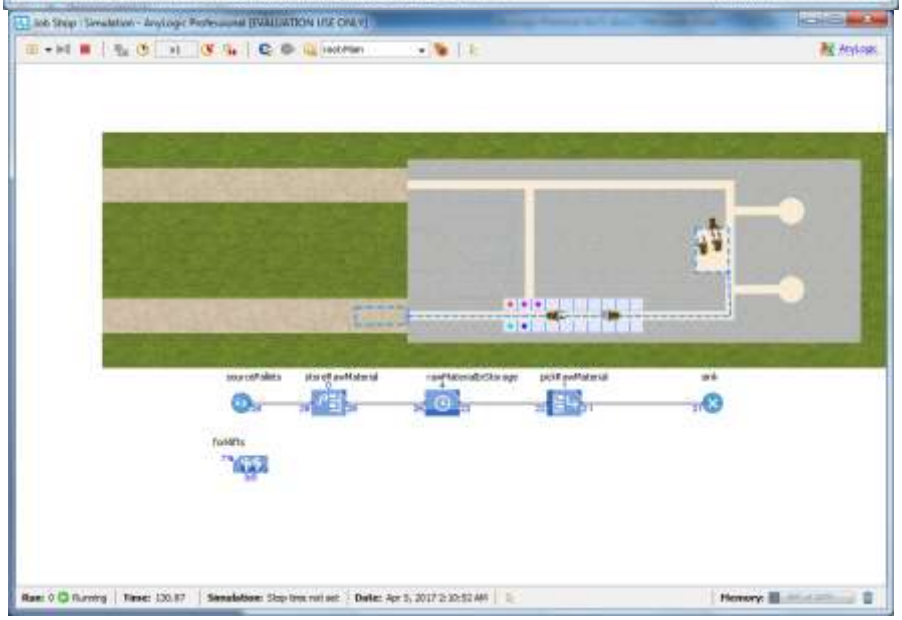

Job Shop - Simulation - AnyLogic Professional [EVALUATION USE ONLY]
root:Main
AnyLogic
sourcePallets
storeRawMaterial
rawMaterialStorage
pickRawMaterial
sink
forklifts
Run: 0 Running
Time: 1.20
Simulation: Stop time not set
Date: Apr 5, 2017 12:01:12 AM
Memory:


Job Shop - Simulation - AnyLogic Professional [EVALUATION USE ONLY]
root:Main
AnyLogic
sourcePallets
storeRawMaterial
rawMaterialStorage
pickRawMaterial
sink
forklifts
Run: 0 Running
Time: 10.45
Simulation: Stop time not set
Date: Apr 5, 2017 12:10:27 AM
Memory:


Job Shop - Simulation - AnyLogic Professional [EVALUATION USE ONLY]
root:Main
AnyLogic
sourcePallets
storeRawMaterial
rawMaterialStorage
pickRawMaterial
sink
forklifts
Run: 0 Running
Time: 20.45
Simulation: Stop time not set
Date: Apr 5, 2017 12:20:27 AM
Memory:

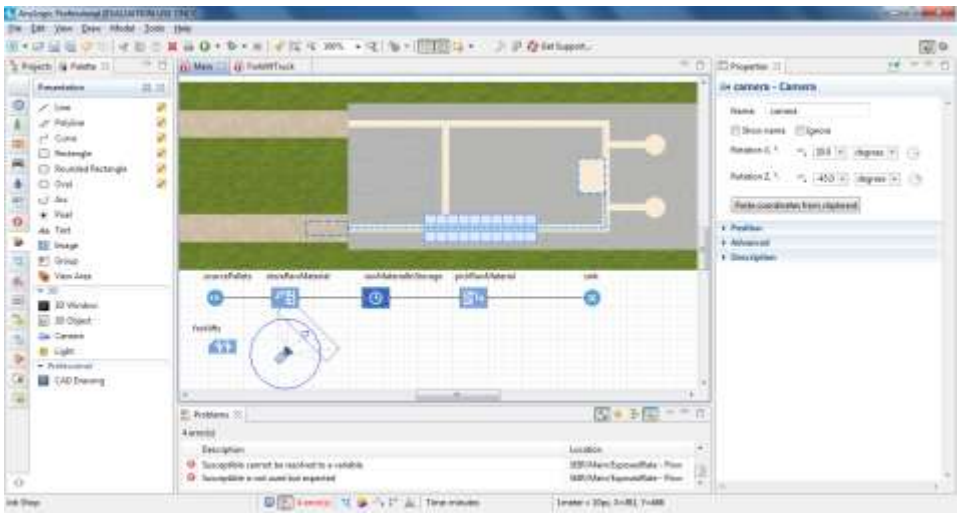




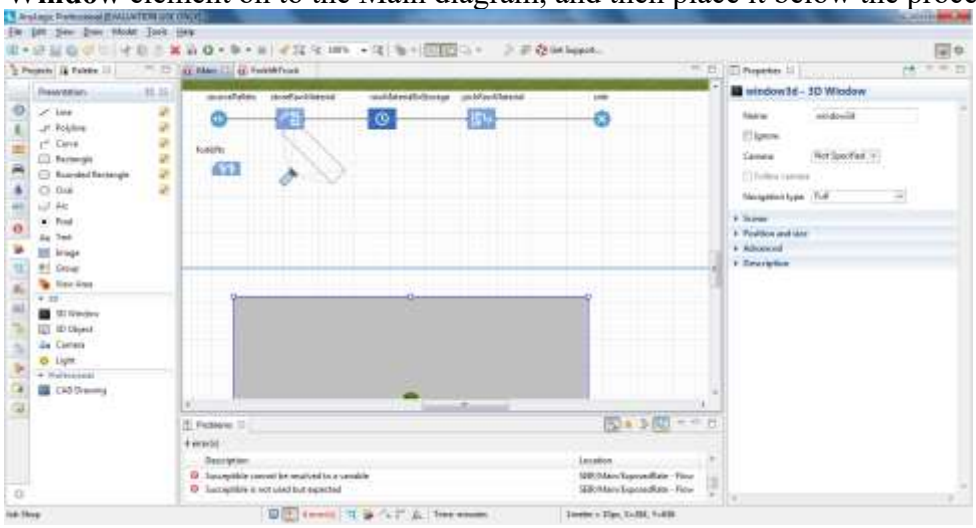


- **Creating 3D animation**

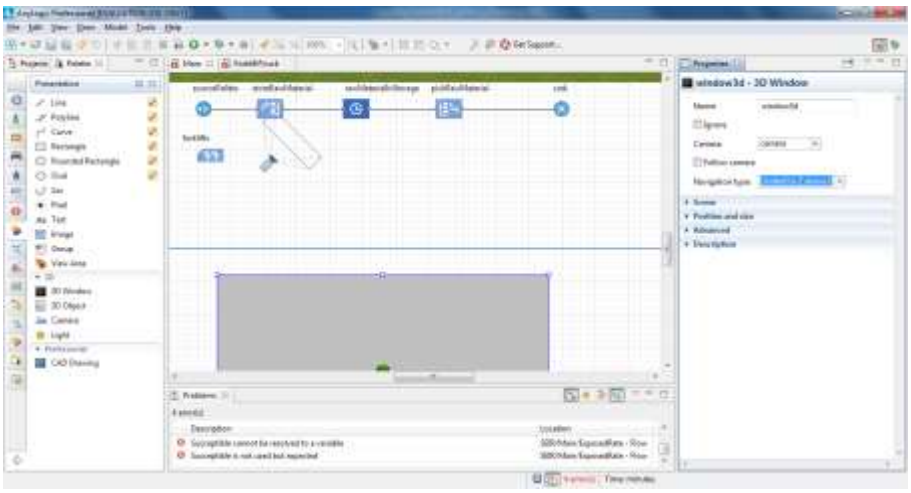
1. On the **Presentation** palette, drag the **Camera** object on to the Main diagram so it faces the job shop layout.



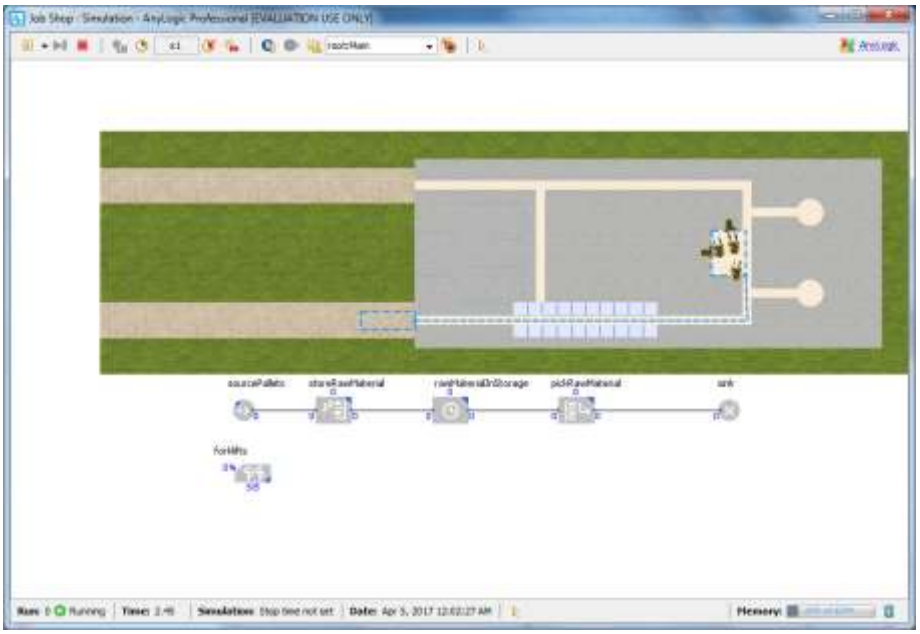
2. Drag the **3D Window** element on to the Main diagram, and then place it below the process flowchart.



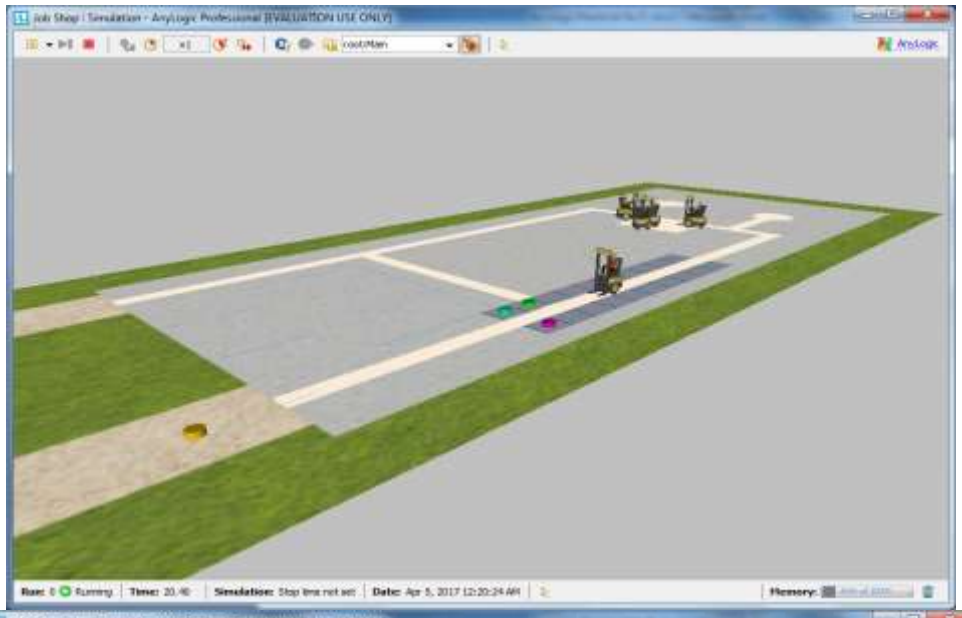
3. Let the camera "shoot" the picture for the 3D window. In the 3D window's **Properties** area,
   - click **camera** in the **Camera** list.
   - Prevent the camera from shooting the picture from under the floor by selecting the option **Limited to Z above 0** from the **Navigation type** list.

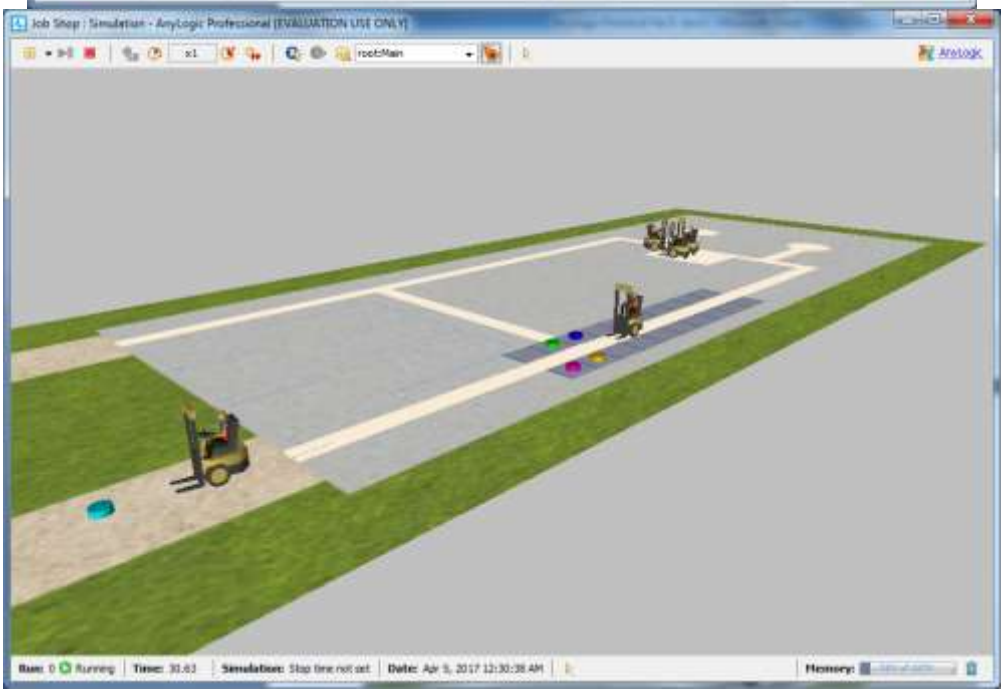


4. Run the model.



5. To switch to this 3D view, click the toolbar's **Navigate to view area…** button and then click **[window3d]**.

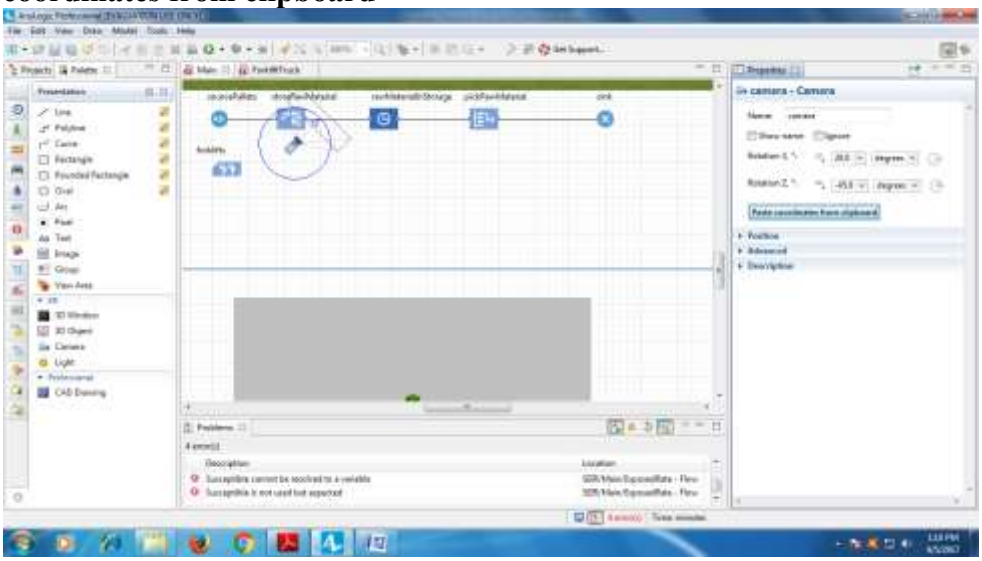




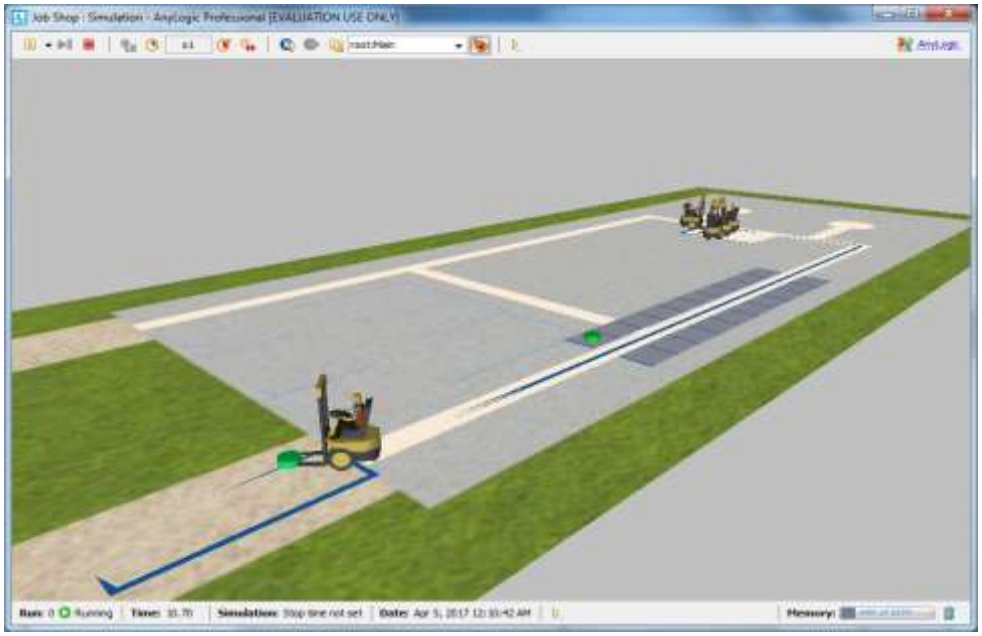
6. Do one or more of the following to navigate in 3D at runtime: AnyLogic 7 in Three Days **161**

- To move the camera left, right, forward or backward, drag the mouse in the selected direction.
- To move the camera closer to or further from the scene's center, rotate the mouse's wheel.
- To rotate the scene relative to the camera, drag the mouse while you press and hold ALT and the left mouse button.
- Choose the view you want to display at runtime, right-click inside the 3D scene, and then click **Copy the camera's location**.
- Close the model's window.

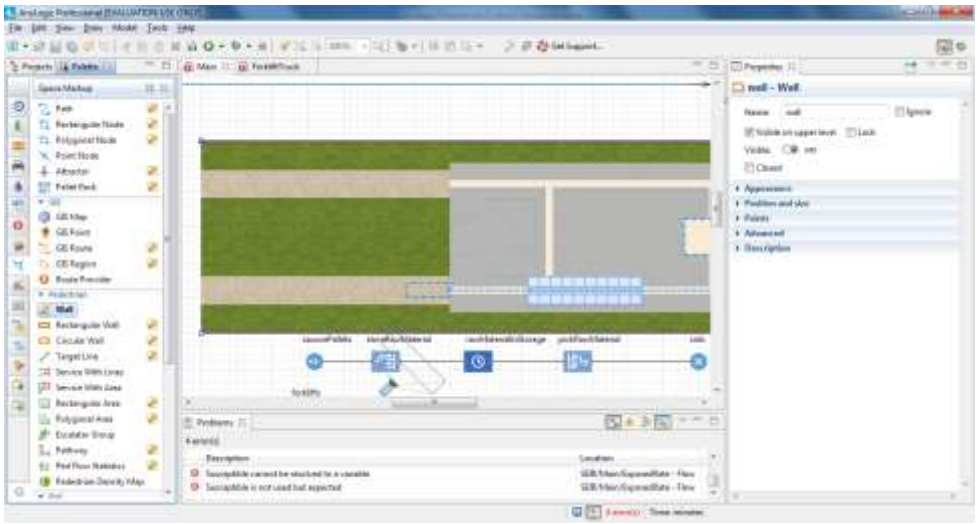
7. On the camera's **Properties** area, apply the camera location you selected during the previous step by clicking **Paste coordinates from clipboard**



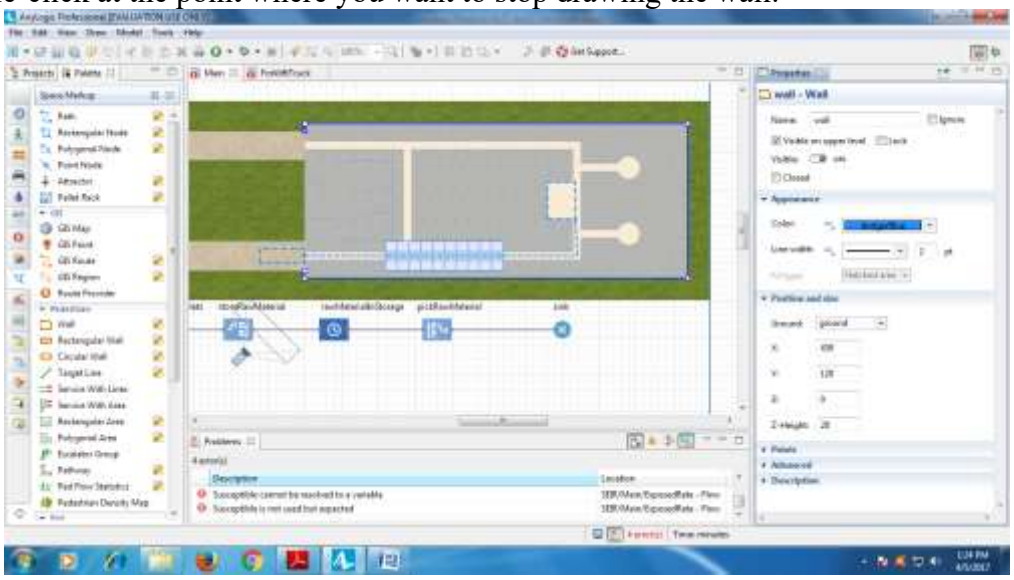
8. Run the model to view the 3D view from the new camera position, and then close the model window.



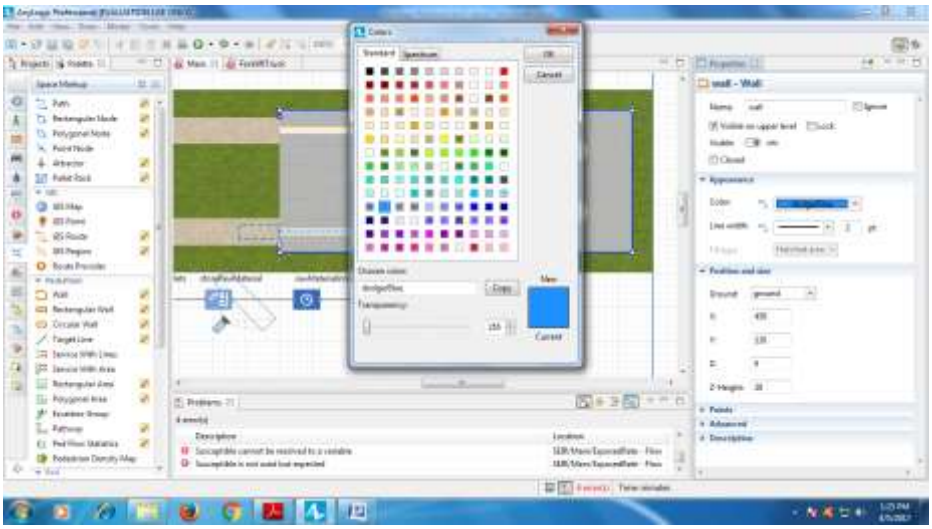
9. Expand the **Space Markup** palette's **Pedestrian** area and then double-click the **Wall** element's icon to enable wall drawing mode.

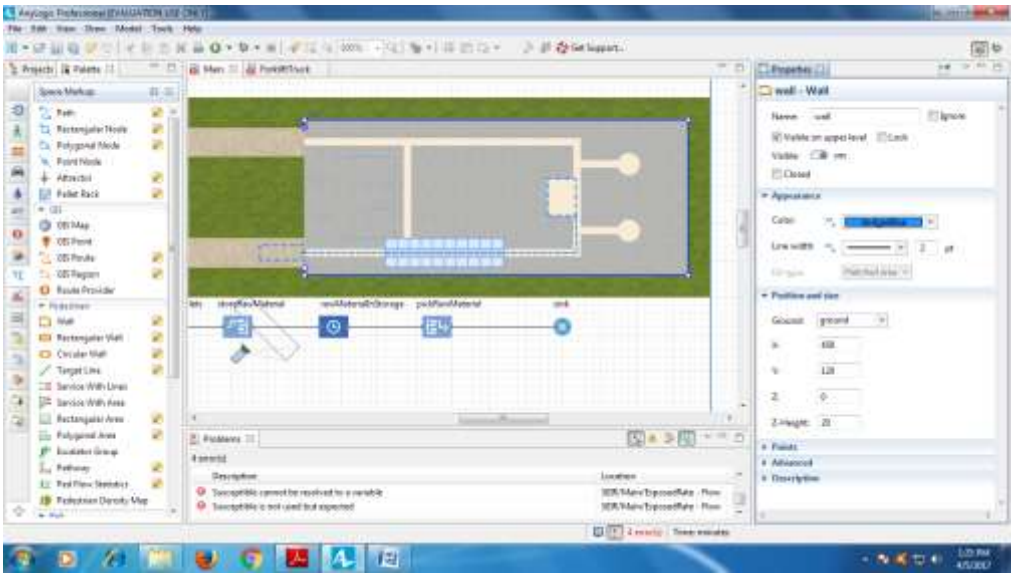


10. Do the following to draw walls around the job shop layout's working area:
    a. Click the position in the graphical editor where you want to start drawing the wall.
    b. Move the pointer in any direction to draw a straight line, and then click at any point where you want to change direction.
    c. Double-click at the point where you want to stop drawing the wall.

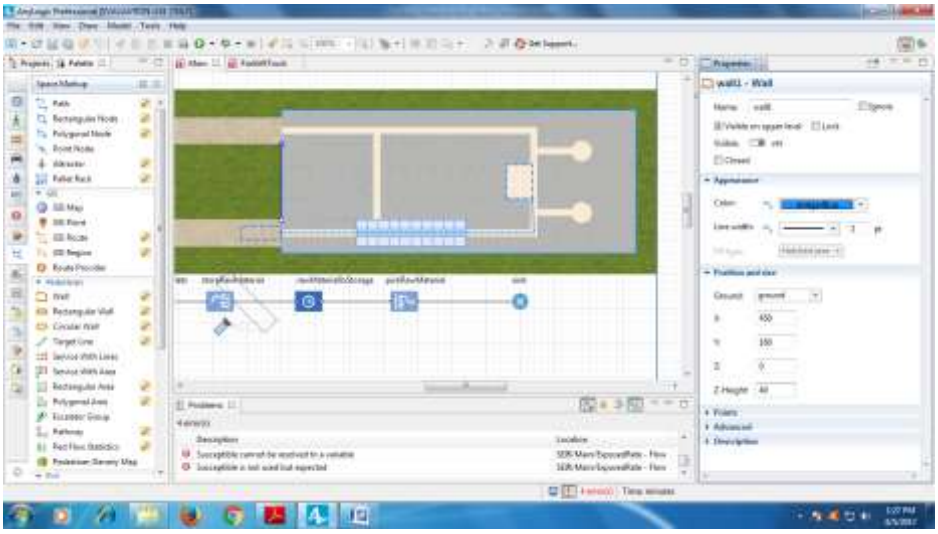


11. Do the following to change the wall's fill color and texture:
    a. On the wall's **Properties** area, expand the **Appearance** section.
    b. In the color menu, click **Other colors**.
    c. In the **Colors** dialog, select the color that you want to apply to the wall from the palette or the spectrum.

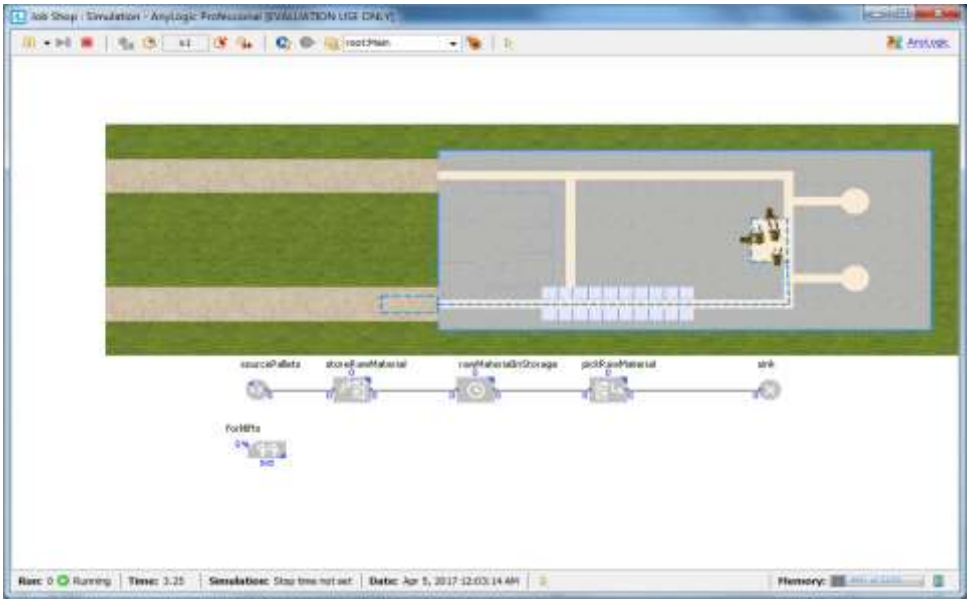


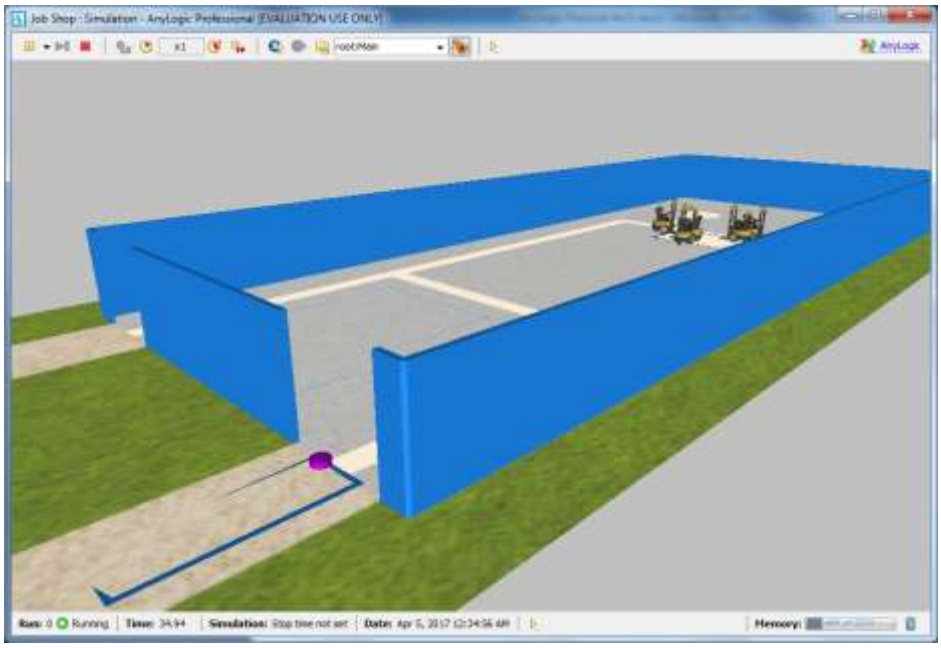


12. Go to the wall's **Position and size** section and change the **Z-Height** to 40. Draw another wall between the exits and then change the settings in the second wall's **Properties** area to match the first wall.

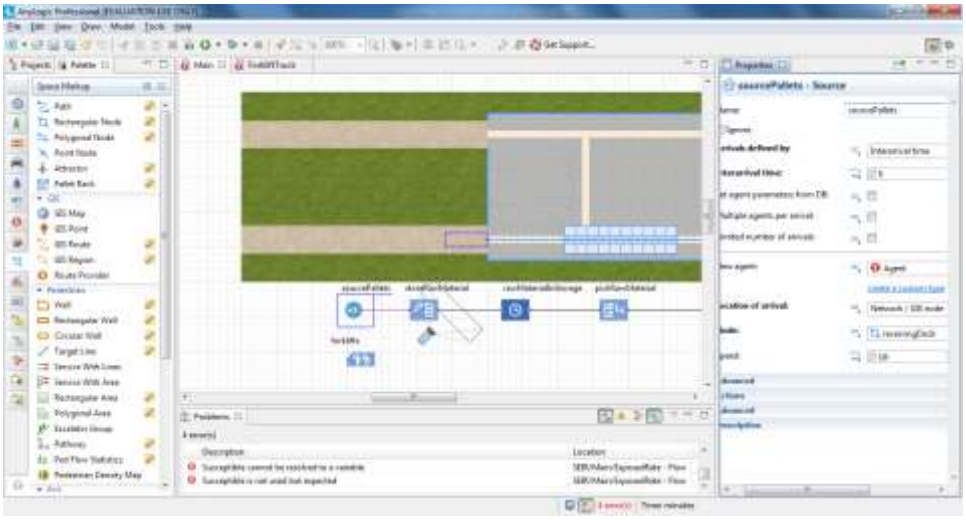


13. Run the model and view the 3D animation.

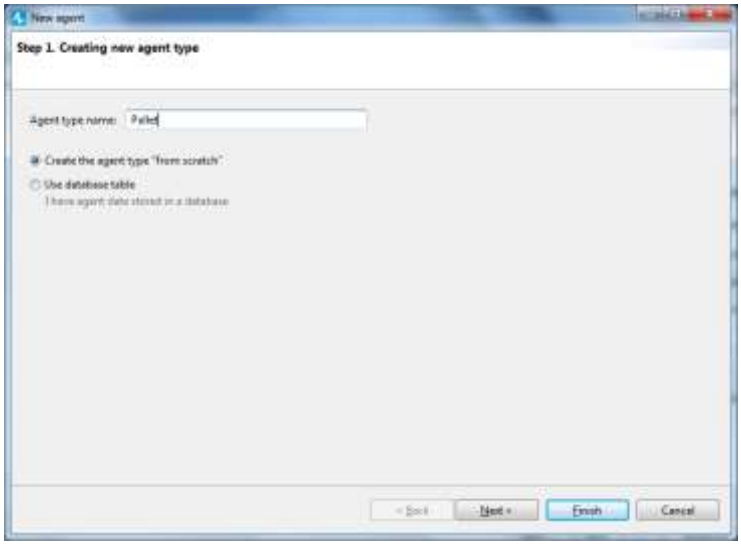


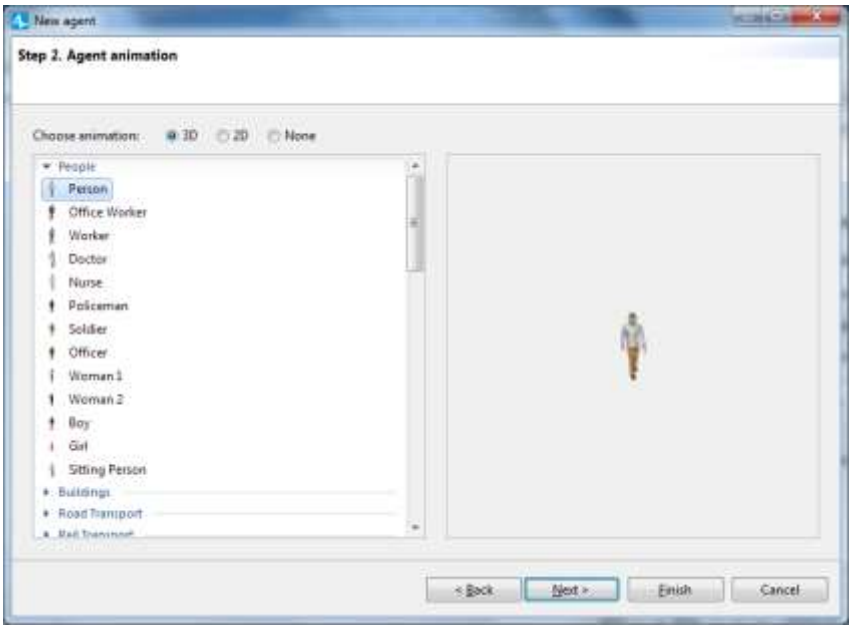


14. In the sourcePallets block's **Properties** area, under the **New agent** list, click **create a custom type** link.



15. In the **New agent** wizard, do the following:
    a. In the **Agent type name** field,
        i. type Pallet.
        ii. Click **Next**.



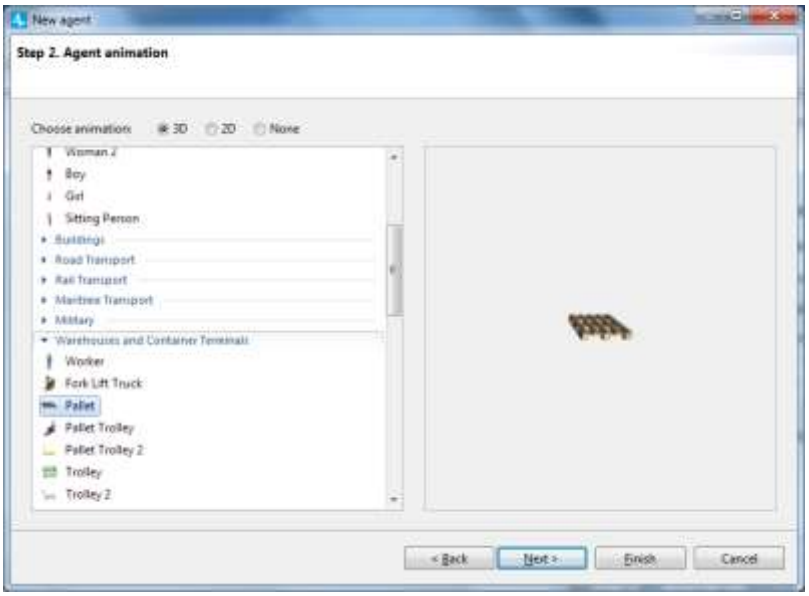


b. On the next page of the wizard,

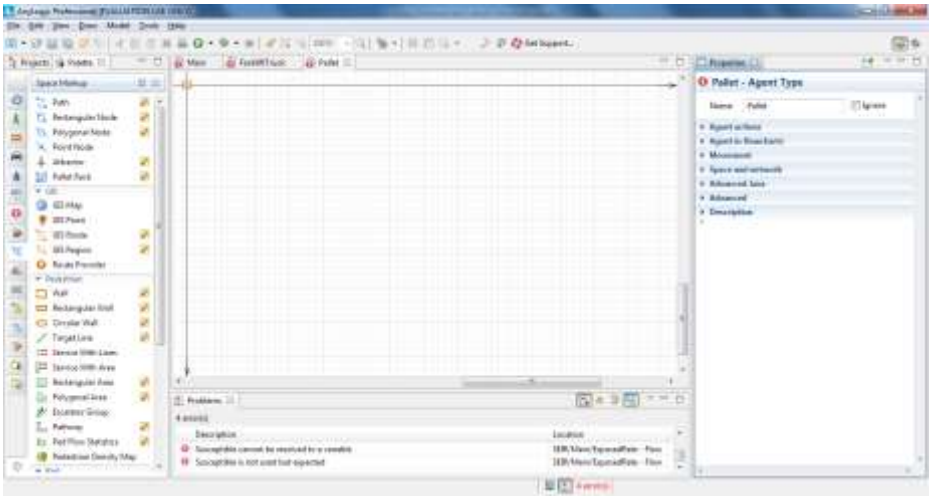
i. expand the **Warehouses and Container Terminals** section in the list on the left, and

ii. then click the 3D animation figure **Pallet.**

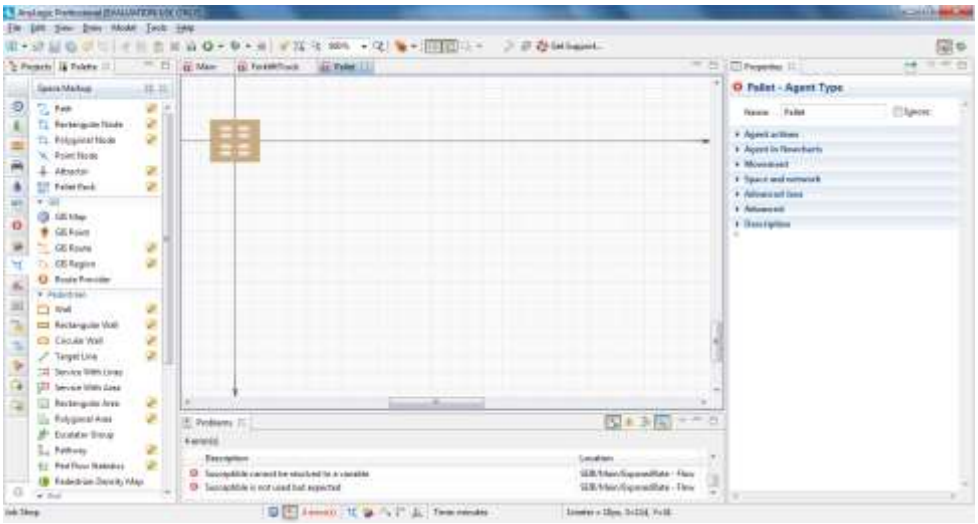
iii. Click **Finish.**



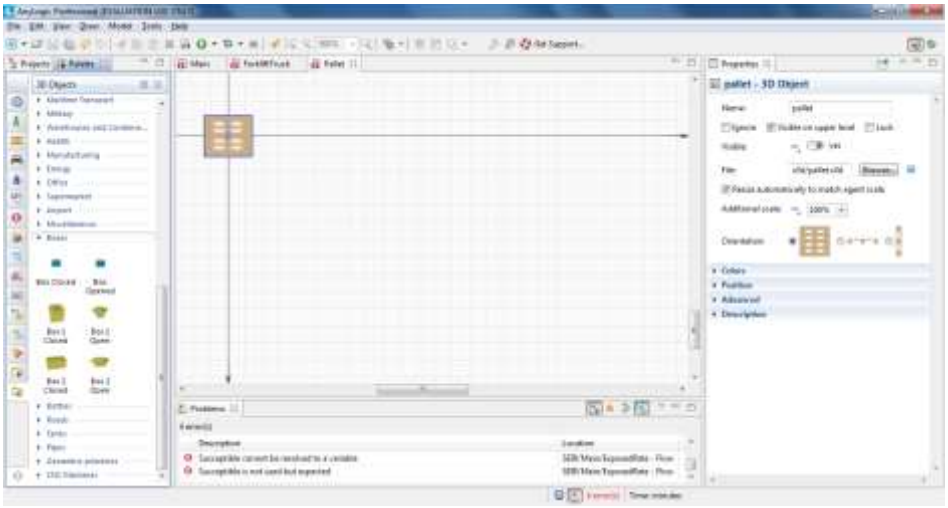
.



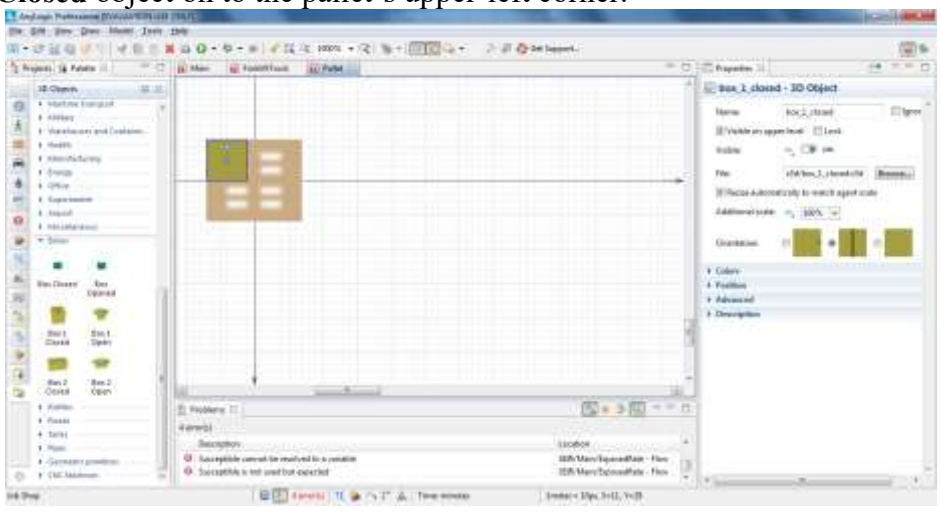
16. Using the **Zoom** toolbar, enlarge the Pallet diagram to 500%, and then move the canvas to the right and down to view the axis' origin point and pallet animation shape.



17. On the **3D Objects** palette, expand the **Boxes** area.



18. Drag the **Box 1 Closed** object on to the pallet's upper-left corner.

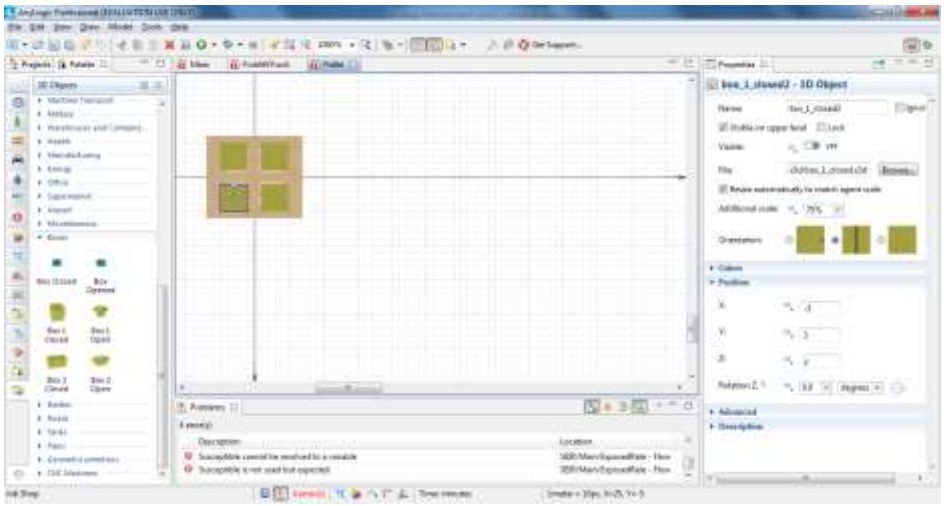


19. Since this box appears to be too large when compared to the pallet, let's change the box's **Scale** to 75%.In the box's **Properties** area,

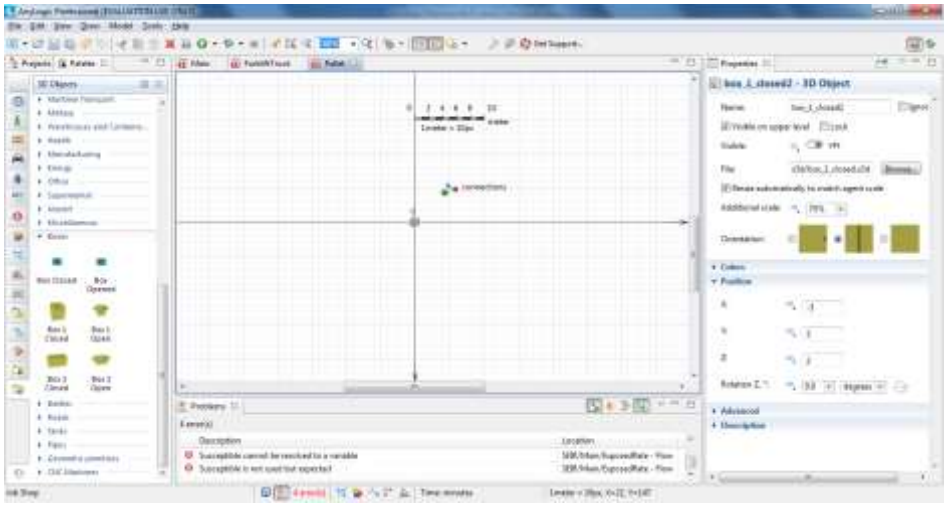
- expand the **Position** section, and
- then change thebox's **Z** coordinate to 2.

Our change reflects the fact that we want to place boxes on the pallets and each pallet's height is about 2 pixels.

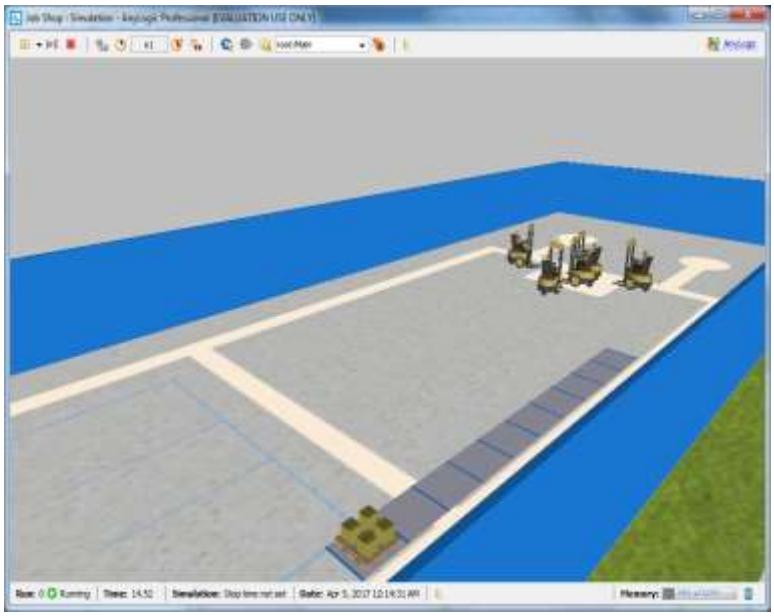
20. Add three boxes by copying the first box three times. To copy the box, select it and then press and hold CTRL as you drag the box.



21. Our pallet now has four closed boxes, and you can now change the zoom level back to 100% by clicking the toolbar's **Zoom to 100%** button.

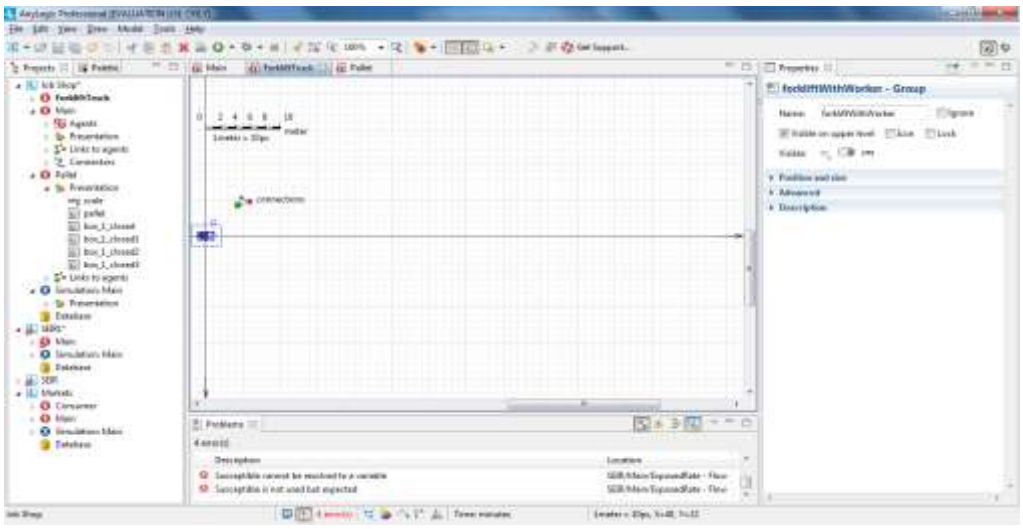


22. Return to the Main diagram.
If you open the sourcePallets block's **Properties** area, you'll see Pallet is selected as **New agent**. This block will generate agents of the Pallet type.
23. Run the model.

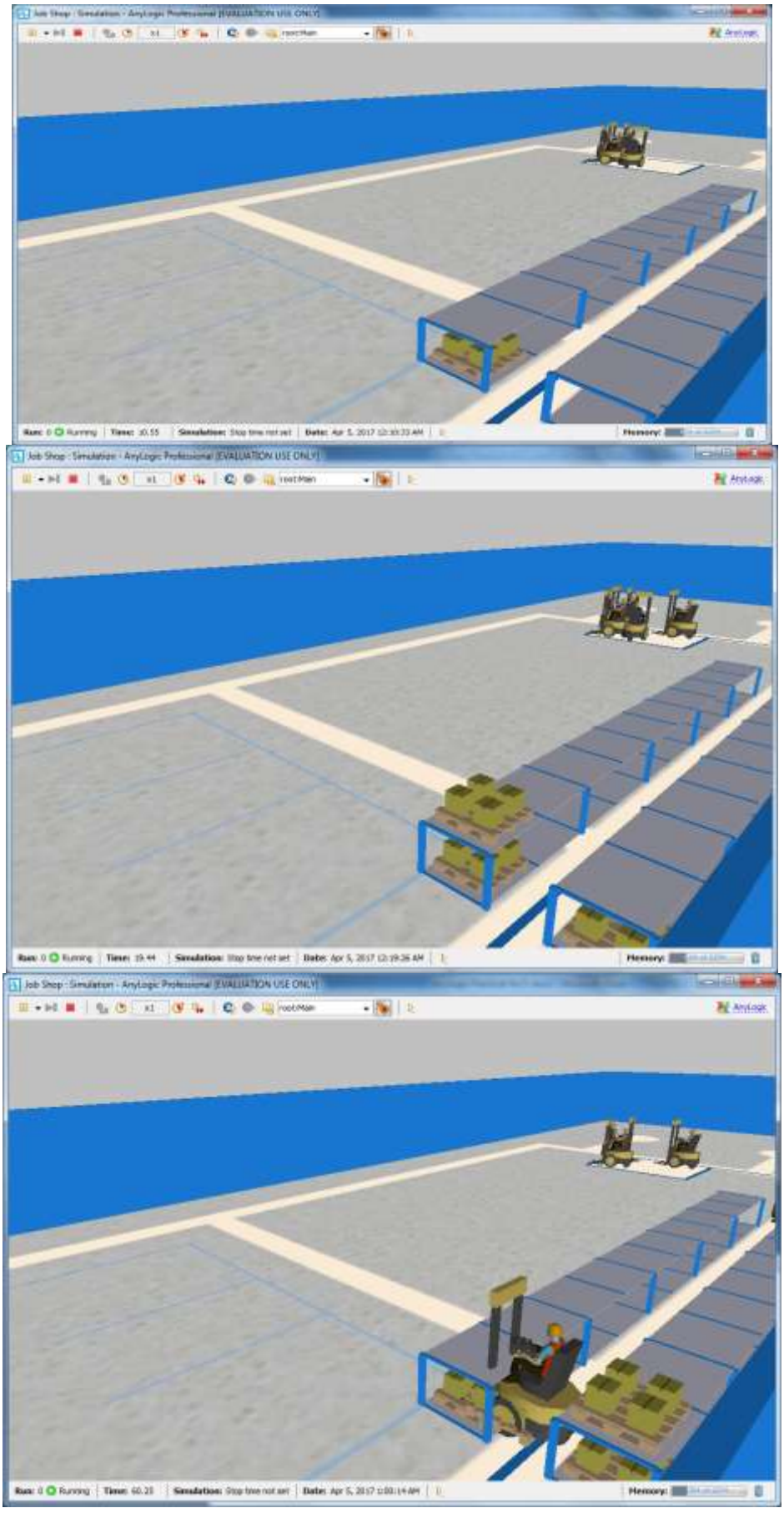


You'll see pallet shapes have replaced the multicolored cylinders. However, if you zoom in the 3D scene, you'll notice that the forklift trucks aren't transporting pallets. We'll correct this problem by moving our model's pallet animation in a way that allows the forklift trucks to pick up the pallets.

24. In the **Projects** view,
    - double-click the ForkliftTruck agent type to open its diagram and
    - then move the forkliftWithWorker figure one cell to the right.



25. Open Main diagram,
    - **a.** in the pallet rack's **Properties** area, in the **Number of levels** box, type 2.
      TIP: Remember that your first click will select the network and your second click will select the network element.
    - **b.** In the storeRawMaterial flowchart block's **Properties** area, set the **Elevation time per level** parameter to 30 **seconds**.
    - **c.** In the pickRawMaterial block's **Properties** area, set the **Drop time per level** parameter to 30 **seconds**.
26. Run the model and you'll see a pallet rack with two levels.



- **Modeling pallet delivery by trucks**

1. On the **Process Modeling Library** palette, drag the **Agent Type** element on to the Main diagram.

2. In the **New agent** wizard, do the following:

a. In the **Agent type name** box,

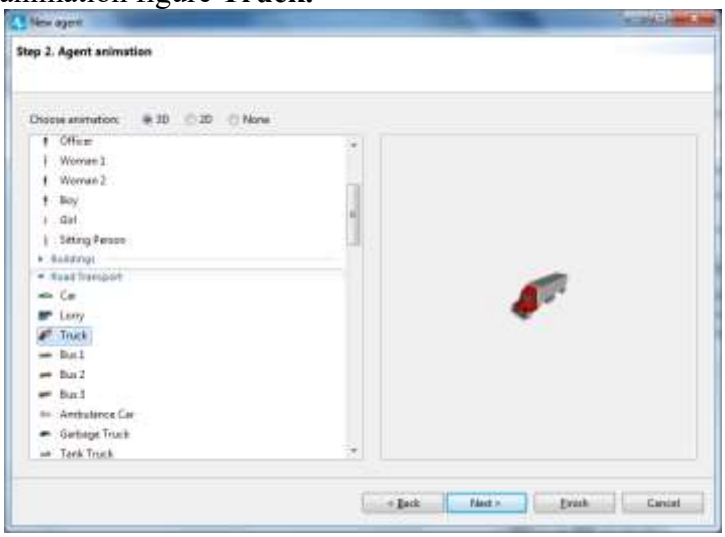
i. type Truck.

ii. Click **Next**.b.

b. On the next page of the wizard,

i. expand the **Road Transport** section in the list on the left, and

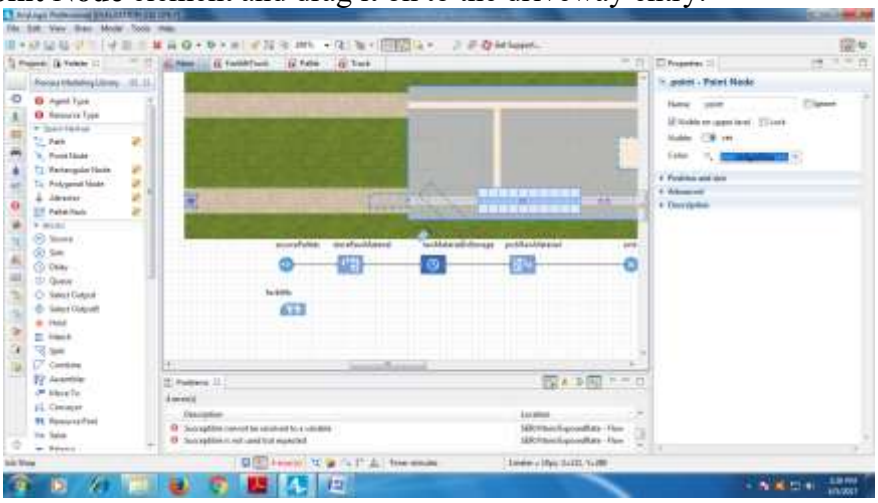
ii. then click the 3D animation figure **Truck.**



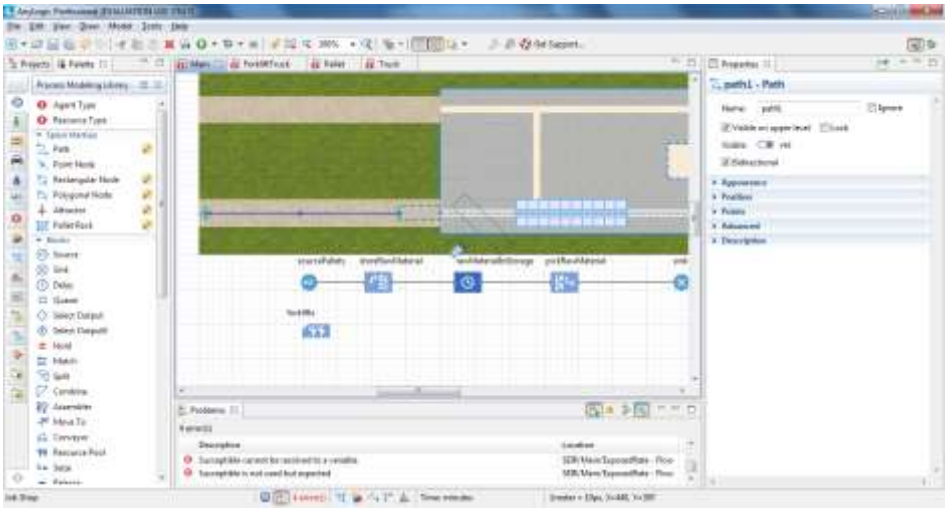
c. Click **Finish**.

3. Open the Main diagram,Under the **Space Markup** palette,

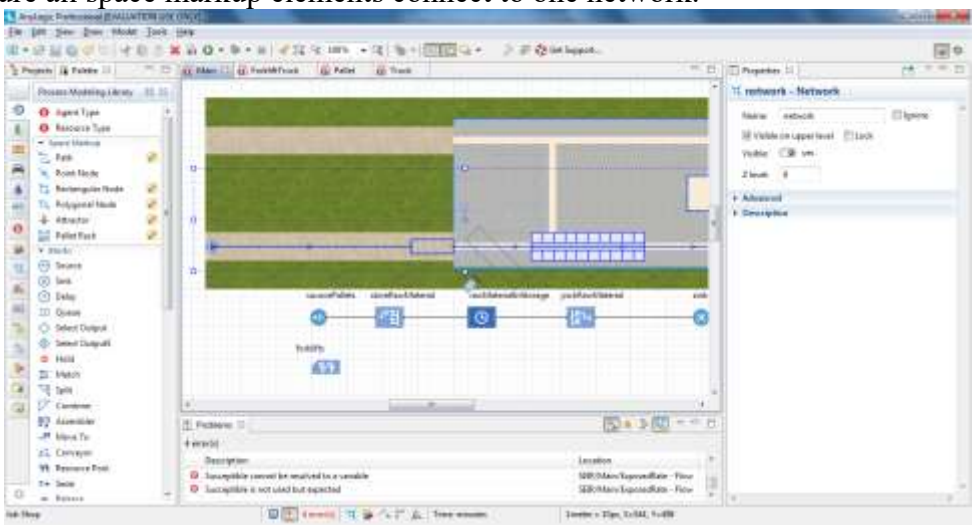
- click the **Point Node** element and drag it on to the driveway entry.



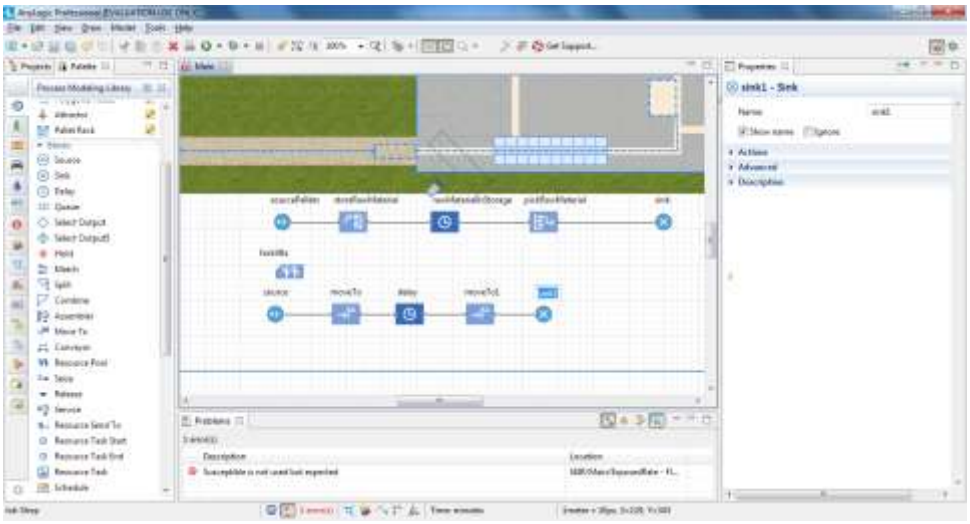
- Name the node exitNode. Draw a **Path** to connect the exitNode to the receivingDock.



- Make sure all space markup elements connect to one network.

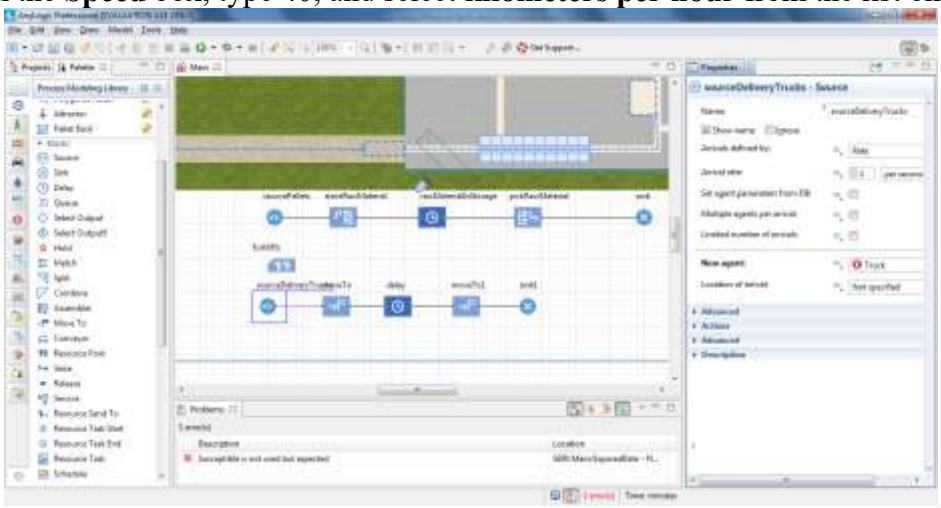


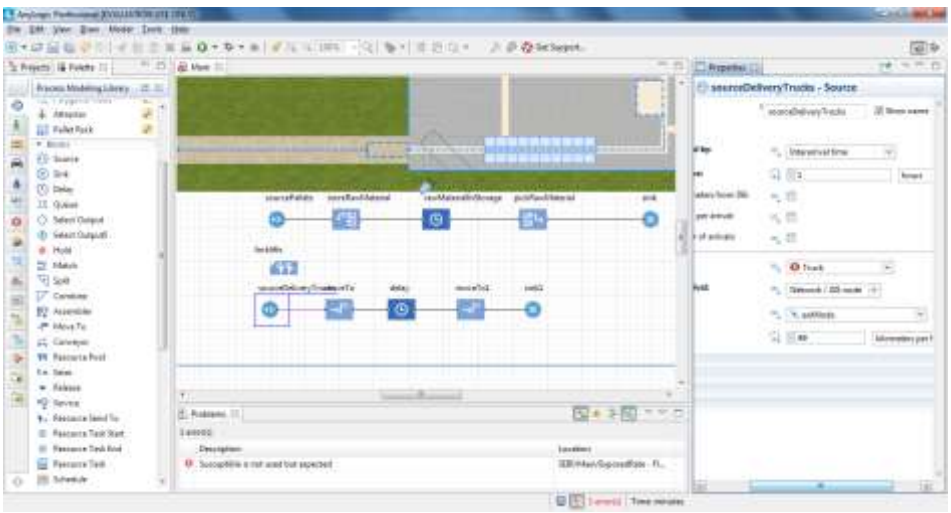
3. Create another process flowchart to define the truck movement logic by connecting the **Process Modeling Library** blocks in the following order: **Source** – **MoveTo** – **Delay** – **MoveTo** – **Sink**.



4. In the sourceDeliveryTrucks block's **Properties** area, do the following to have a new agent of the custom Truck type arrive to the driveway entry once per hour at a specific speed:

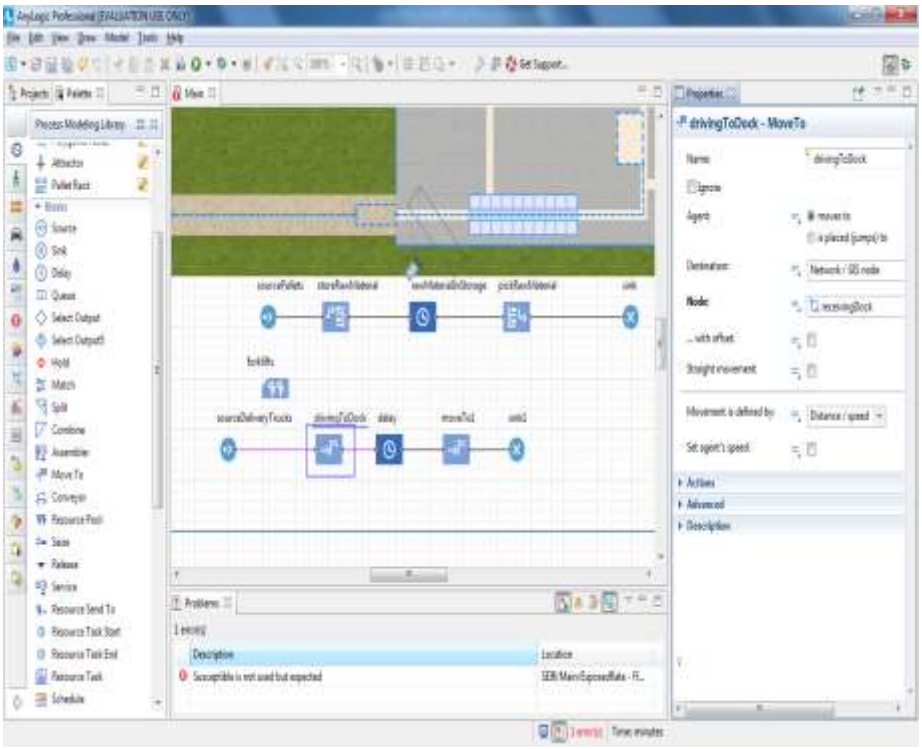
a. In the **Arrivals defined by** list, click **Interarrival time**.
b. In the **Interarrival time** box, type 1, and select **hours** from the list on the right.
c. In the **New agent** list, click **Truck**.
d. In the **Location of arrival** list, click **Network/GIS node**.
e. In the **Node** list, click **exitNode**.
f. In the **Speed** box, type 40, and select **kilometers per hour** from the list on the right.





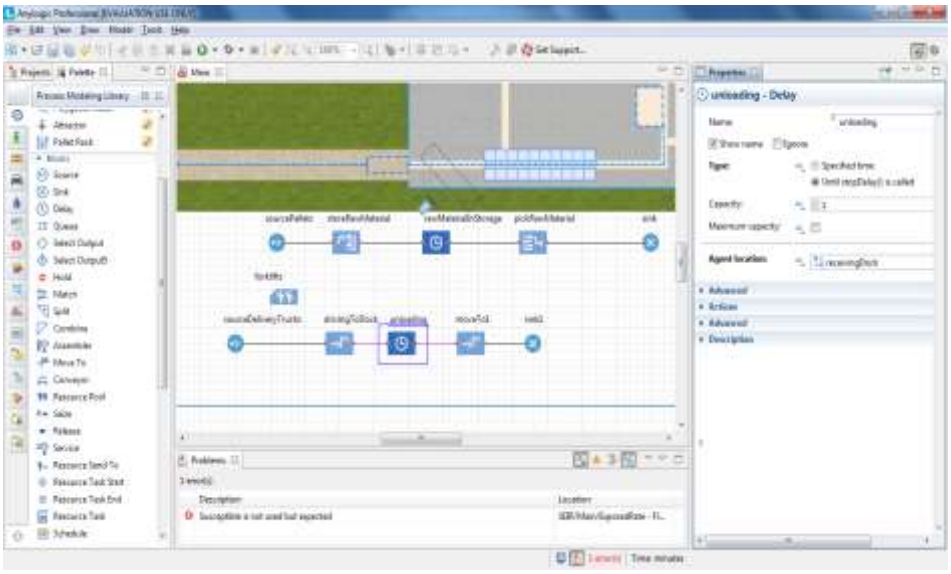
5. In the drivingToDock block's **Properties** area

- Name the first **MoveTo** block drivingToDock.
- in the **Node** list, click **receivingDock** to set the agent's destination.



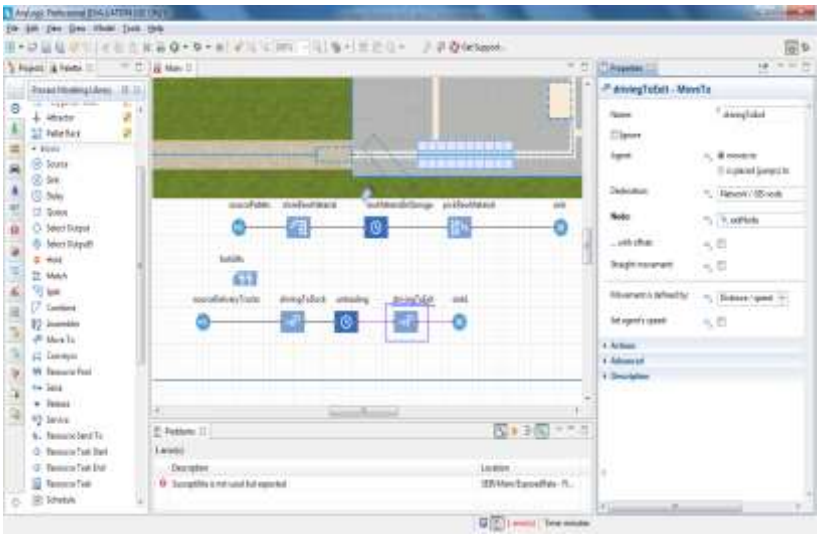
13. In the unloading block's **Properties** area, do the following:

- Rename the **Delay** block to unloading.
- In the **Type** area, click **Until stopDelay() is called**.
- In the **Agent location** list, click **receivingDock**.



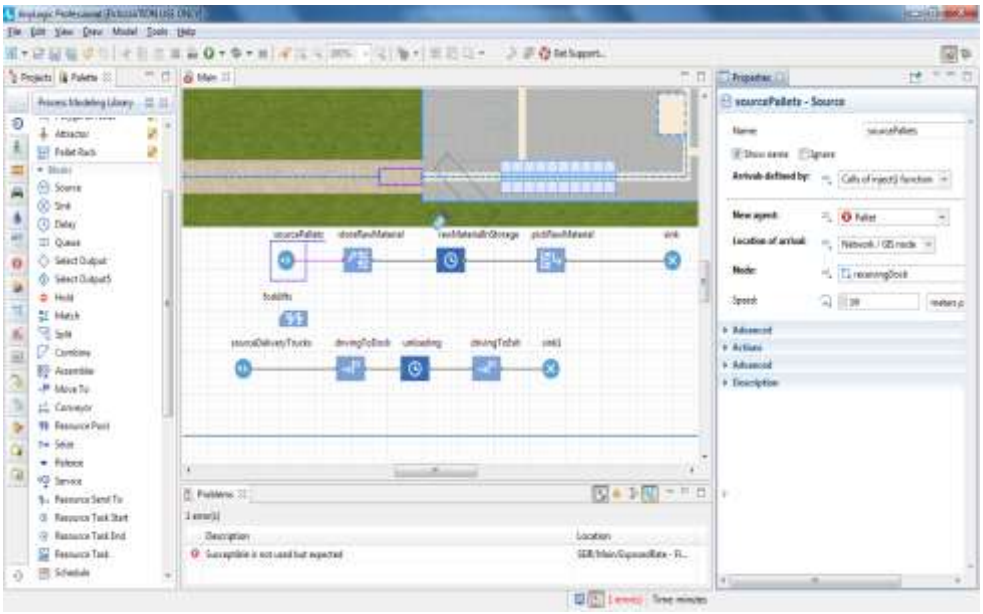
15. In the drivingToExit. block's **Properties** area,

- Name the second **MoveTo** block drivingToExit
- in the **Node** list, click **exitNode** to set the destination node.



Our model's two **Source** blocks generate two agent types: the trucks that appear each hour and the pallet that is generated every five minutes. Since we want pallets to appear when the truck unloads, we'll change the arrival mode for the **Source** block that generates them.

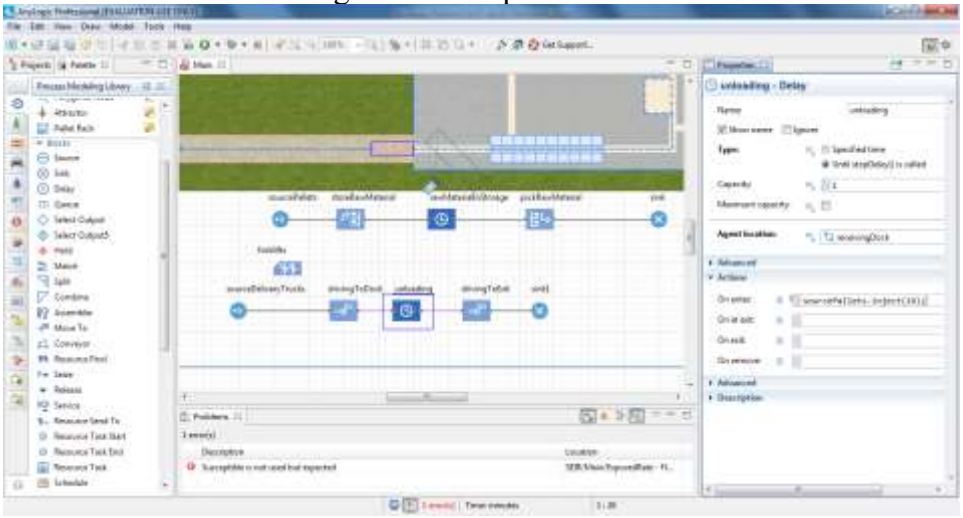
16. In the sourcePallets block's **Properties** area, in the **Arrivals defined by** list, click **Calls of inject() function**.



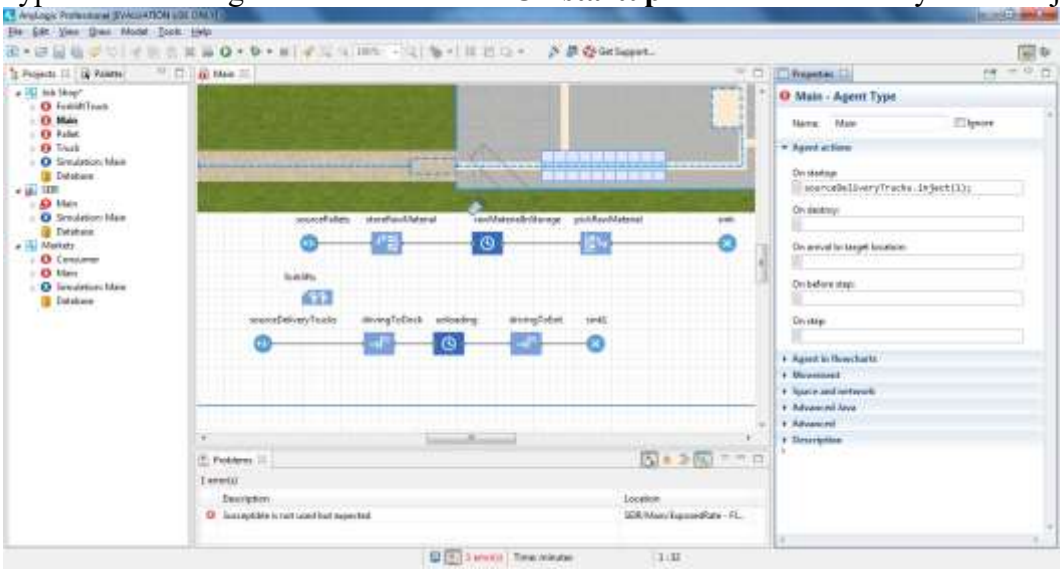
17. Do the following to have the sourcePallets block generate pallets when a truck enters the unloading block:

a. In the unloading block's **Properties** area, expand the **Actions** section.
b. In the **On enter** box, type the following: sourcePallets.inject(16);

This Java function will ensure our model generates 16 pallets each time a truck starts to unload.

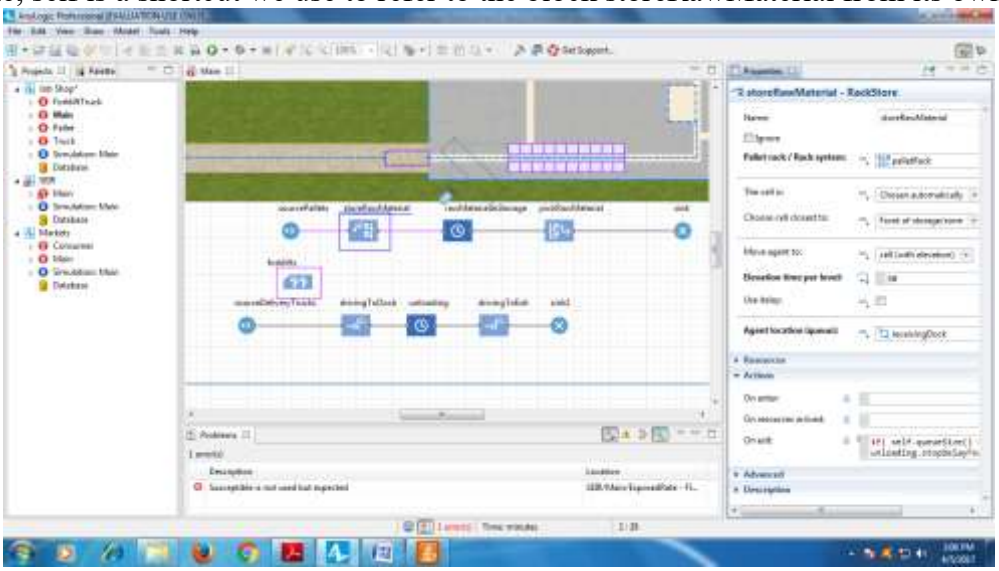


18. In the Main agent type's **Properties** area,
    - expand the **Agent actions** section and
    - then type the following Java function in the **On startup** box: sourceDeliveryTrucks.inject(1);

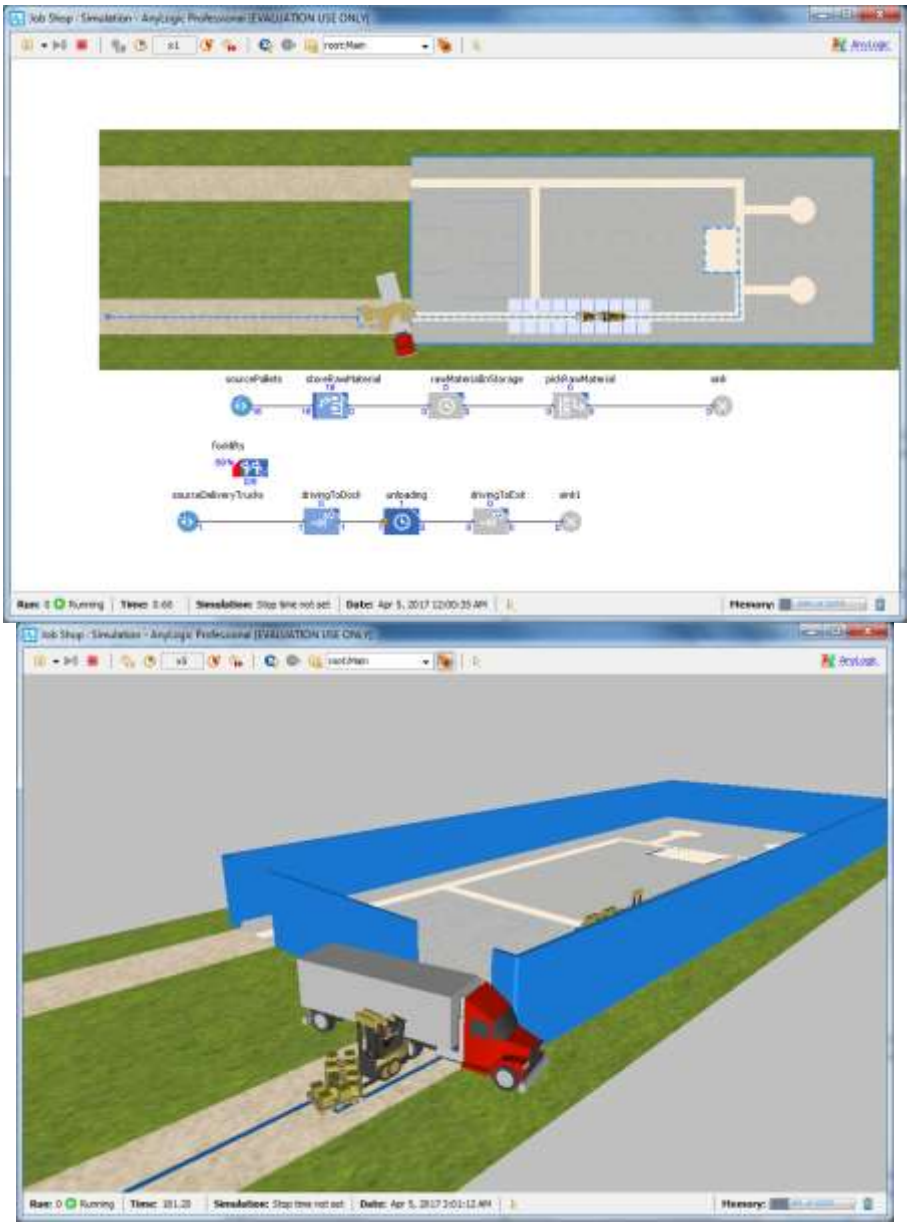


19. In the storeRawMaterial block's **Properties** area,
    - expand the **Actions** section, and
    - in the **On exit** box, type the following:
    - if(self.queueSize() == 0)
    - unloading.stopDelayForAll();

In this example, self is a shortcut we use to refer to the block storeRawMaterial from its own action.



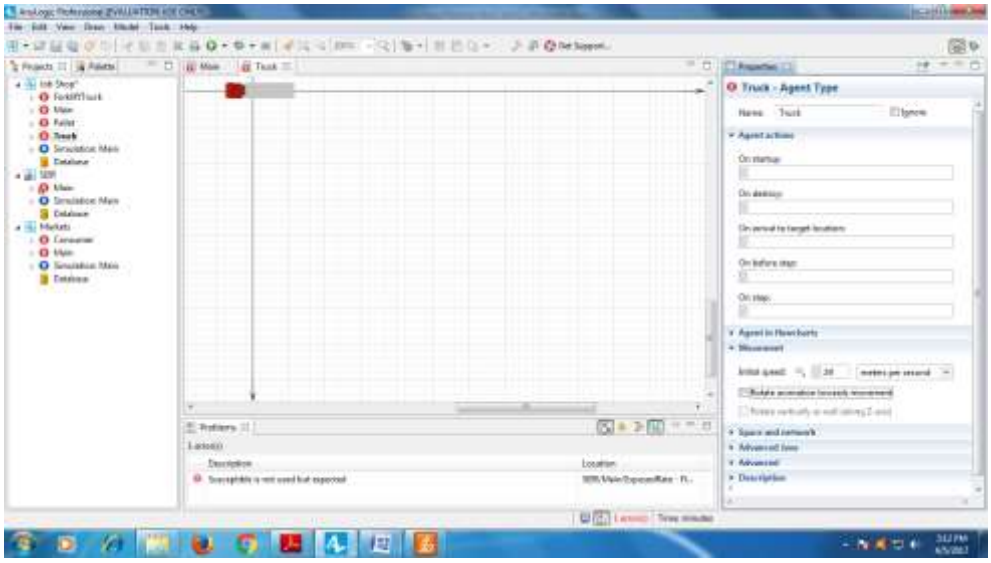
20. Run the model.



21. If the truck is aligned incorrectly as in the figure above, do the following to fix it.
    a. In the **Projects** tree, double-click the **Truck** agent type to open its diagram and view the truck animation figure.
    b. In the graphical editor, select the truck shape and then use the round handle or the **Rotation Z,°** property in the shape's **Position** properties area to rotate the truck to -180 degrees.

We've changed the truck figure's position, but we'll also need to change AnyLogic's default setting to make sure the program doesn't rotate it a second time.

22. Do the following to change AnyLogic's default setting:
    a. In the **Projects** area, click **Truck**.
    b. On the Truck agent type's **Properties** area, click the arrow to expand the **Movement** area.
    c. Clear the **Rotate animation towards movement** check box.

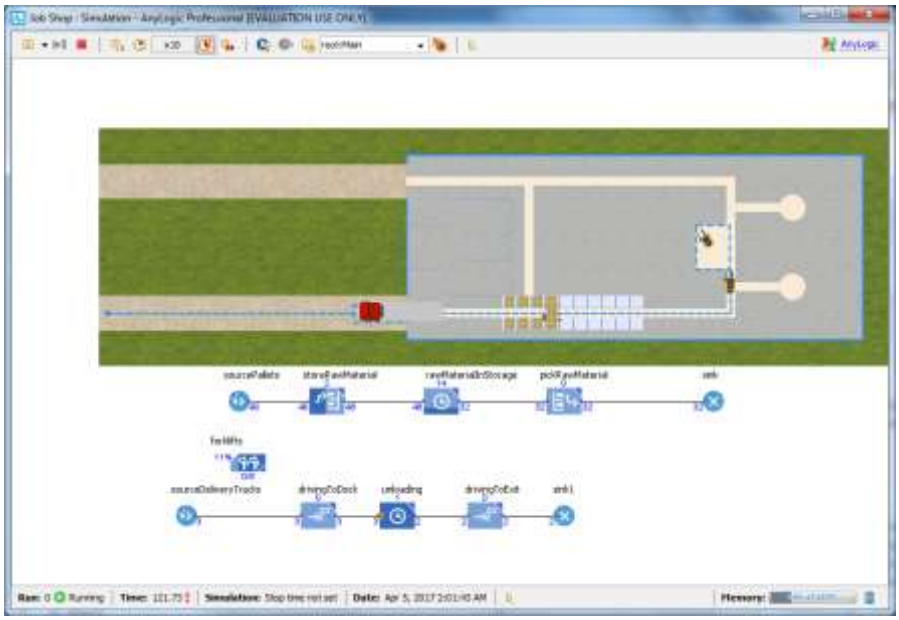


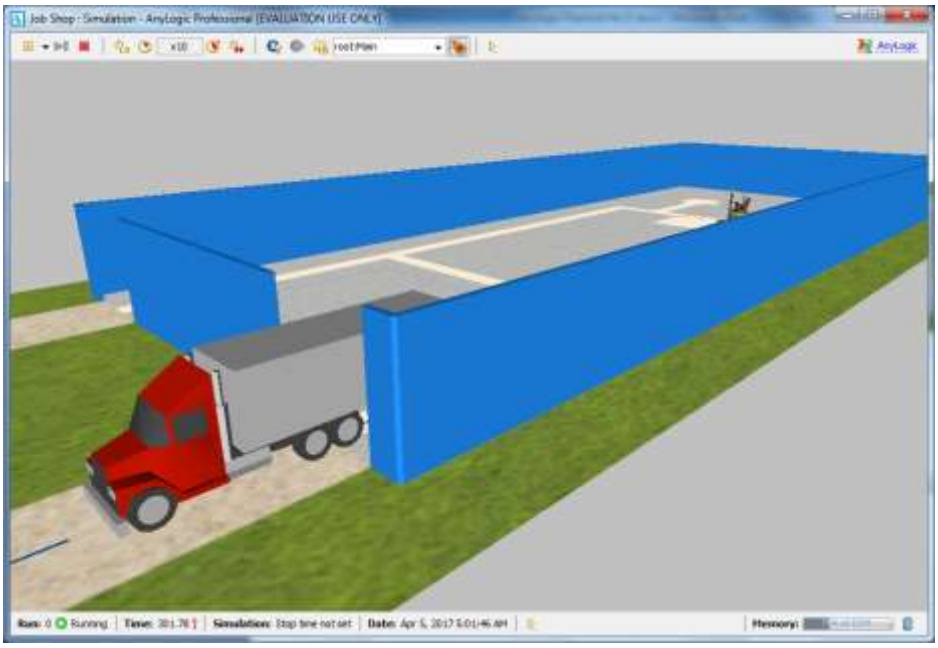
23. Open the Main diagram.

To ensure the pallets are correctly positioned in the receivingDock network node,

- open the **Space Markup** palette, and
- drag an **Attractor** into receivingDock. Let it face the entrance.

24. Run the model to check the truck behavior.





25. Removing the wall & checking the output

Practical 6

AIM : Design and develop time-slice simulation for a scenario like airport model to design how passengers move within a small airport that hosts two airlines, each with their own gate. Passengers arrive at the airport, check in, pass the security checkpoint and then go to the waiting area. After boarding starts, each airline's representatives check their passengers' tickets before they allow them to board.

Scenario :

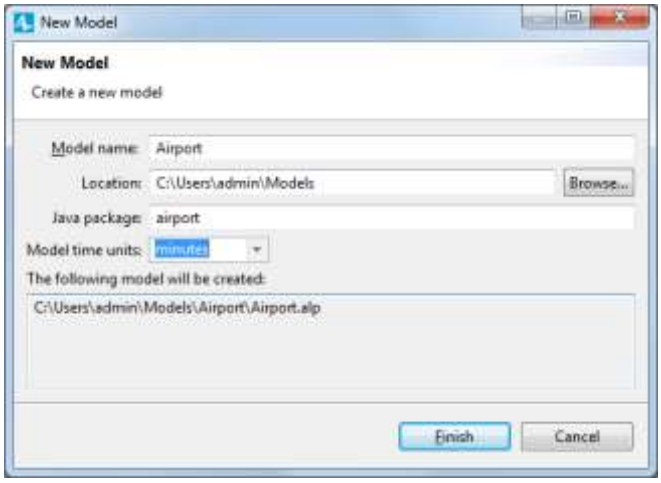
Time Slice :

In time-slicing or time-stepped simulations, the time is divided into equal slots. The simulation evolves in steps of time and the state of the system is computed for each step.

- ***Phase 1*** *:* Creating a simple model to simulate the passenger's arrival at the airport and their boarding process.
- ***Phase 2*** *:* Drawing 3D animation.
- ***Phase 3*** *:* Adding security checkpoints.
- ***Phase 4*** *:* Adding check-in facilities.
- ***Phase 5*** *:* Defining the boarding logic.

Solution:

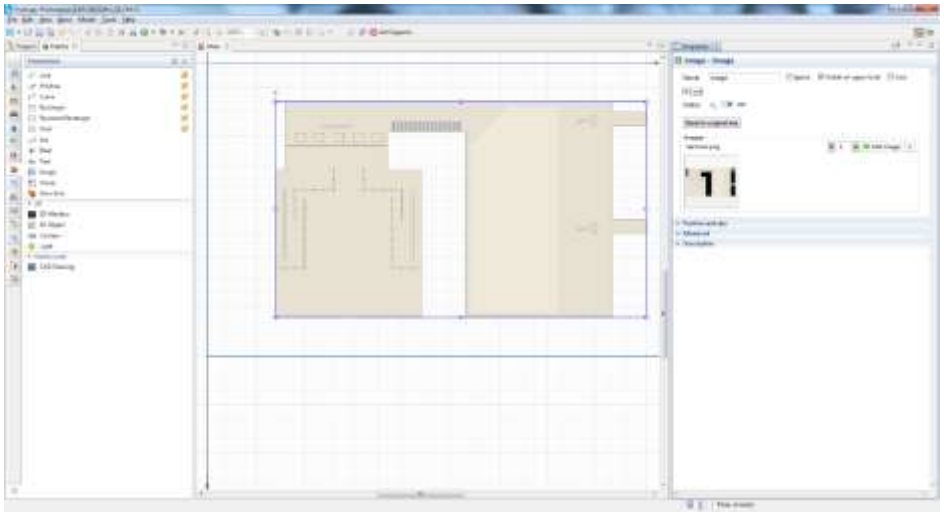
1. Create a new model and name it Airport. Select minutes as the model time units.



2. Drag an **Image** from the **Presentation** palette on to the Main diagram. Choose the image file you want to display.

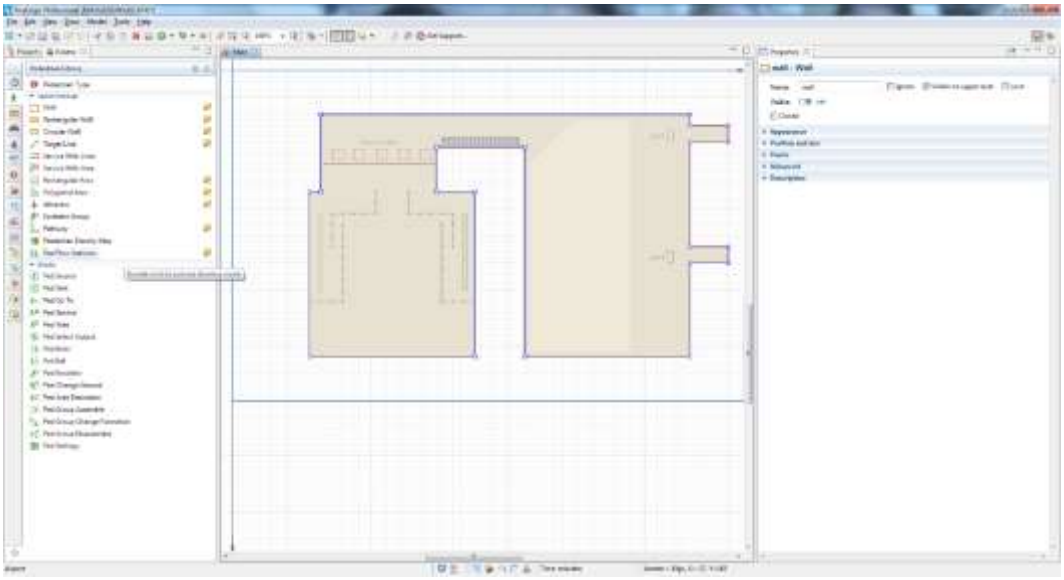
In this example, you'll select the *terminal.png* image file from *AnyLogic folder/resources/AnyLogic in 3 days/Airport*

3. On the Main diagram, place the image in the blue frame's lower left corner. If the image is distorted, click the **Reset to original size** button and then select the **Lock** checkbox to lock the image shape.

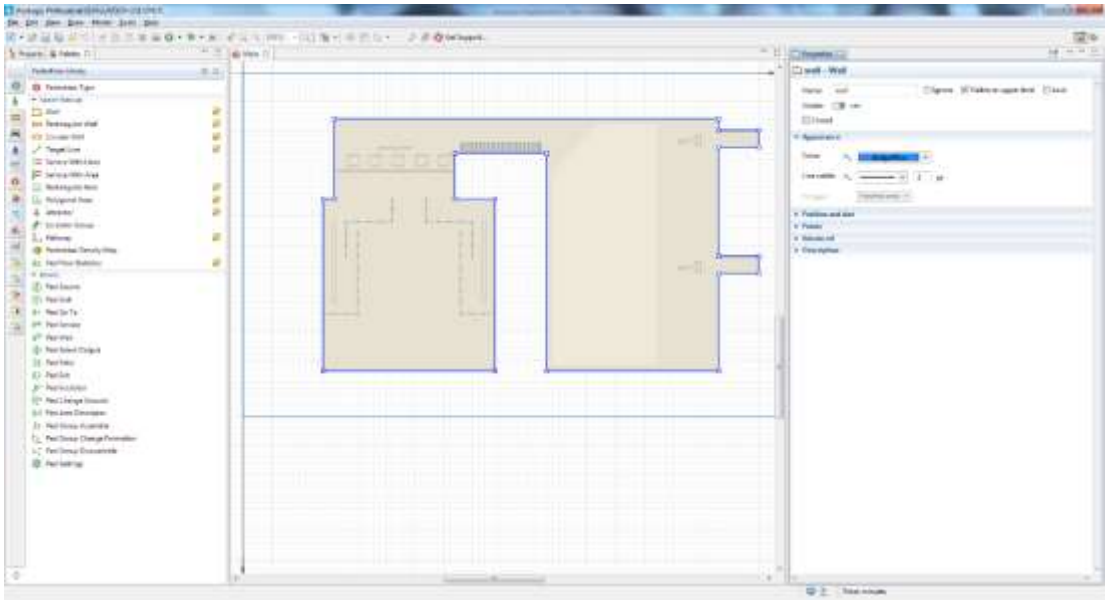


4. Use the **Pedestrian Library** palette to draw the airport's walls.

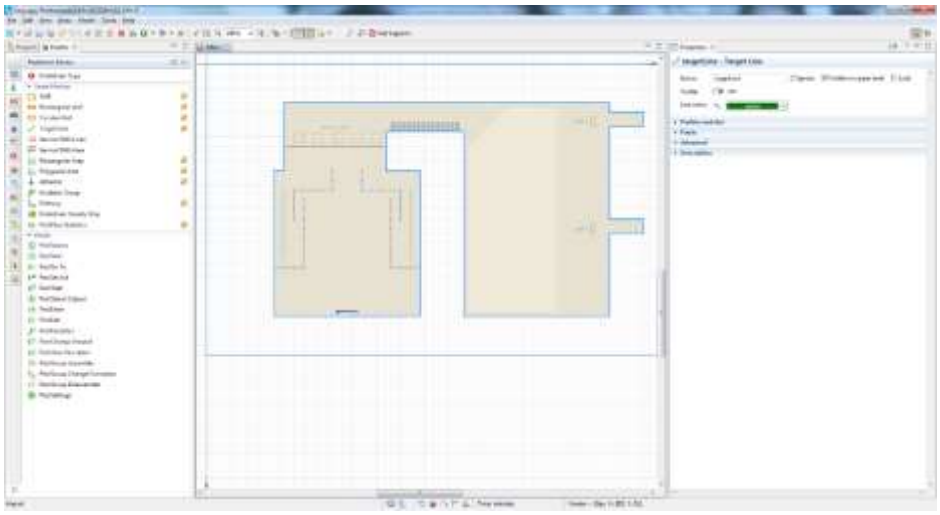
- Double-click the **Wall** element you'll find in the **Pedestrian Library** palette's **Space markup** section and
- then draw the wall around the airport building's border by clicking your mouse each time you want to add a point.
- When you're ready to set the wall's final point, double-click your mouse.



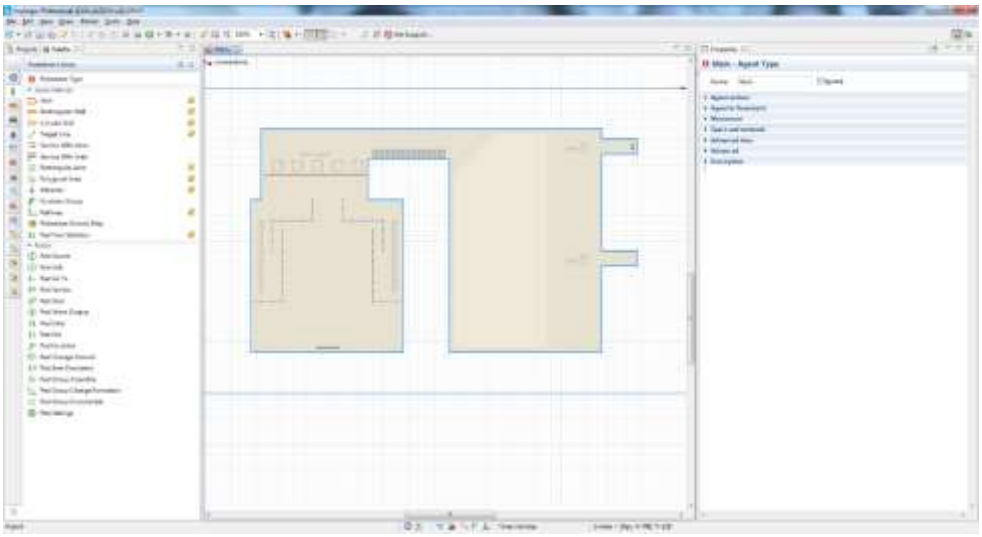
5. Navigate to the wall's properties and then select the **Color**: dodgerBlue in the **Appearance** section.



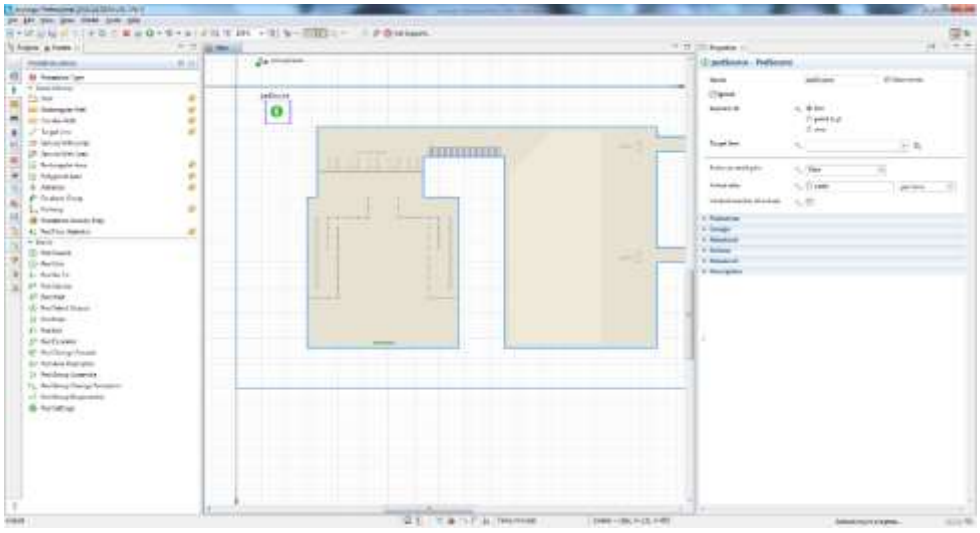
6. Define the location where your model's passengers appear by dragging the **Target Line** element from the **Pedestrian Library** palette on to the graphical diagram, as shown in the figure below.



7. Define a second target line that passengers will move toward after they enter the airport, place it in the gate area as shown in the figure below, and then name it gateLine1.

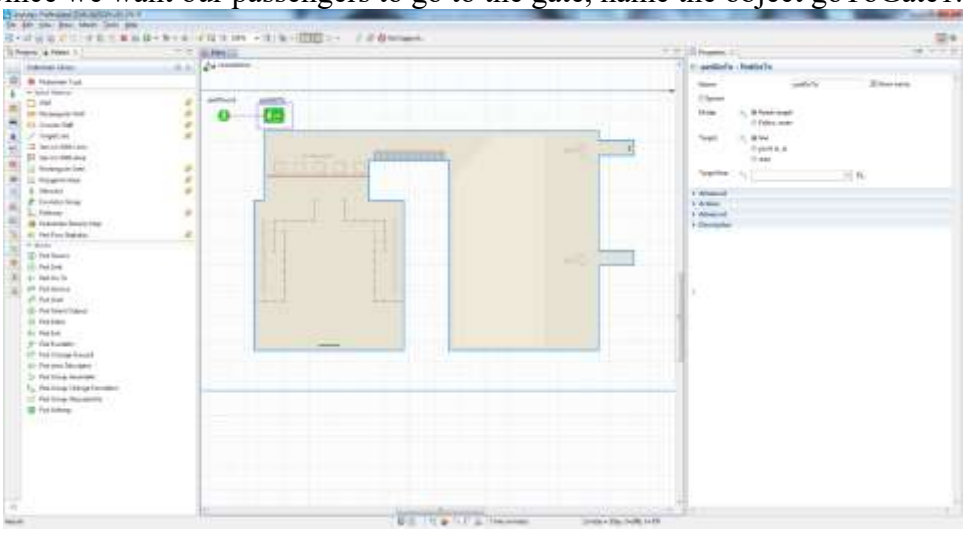


8. Start by dragging the **PedSource** block from the **Pedestrian Library** palette on to our Main diagram.

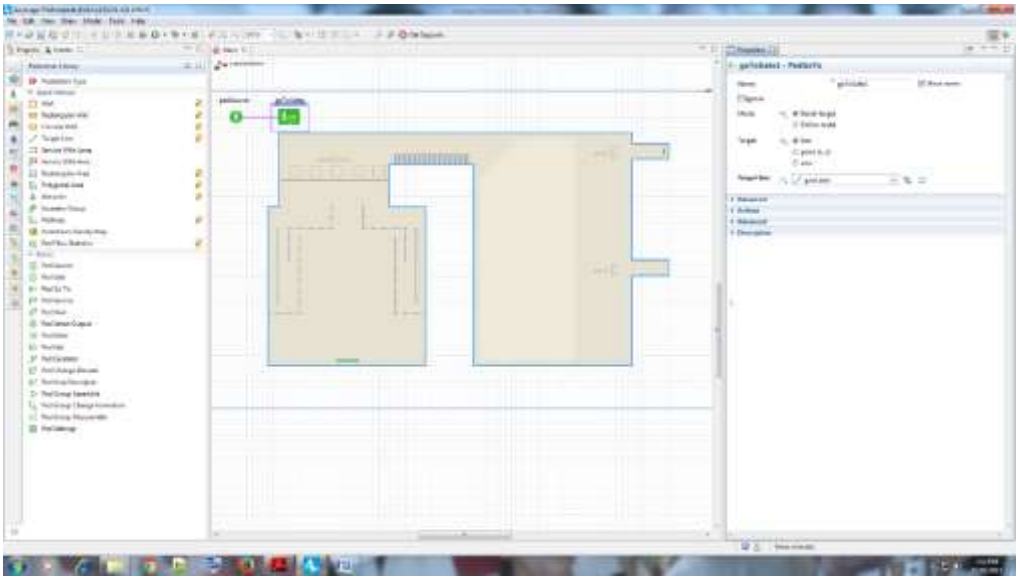


9. Since we want passengers to arrive randomly at an average rate of 100 passengers per hour, go to the pedSource block properties and then type 100 in the **Arrival rate** box.
10. Specify the location where the passengers appear in the simulated system by clicking arrivalLine in the **Target line** list.

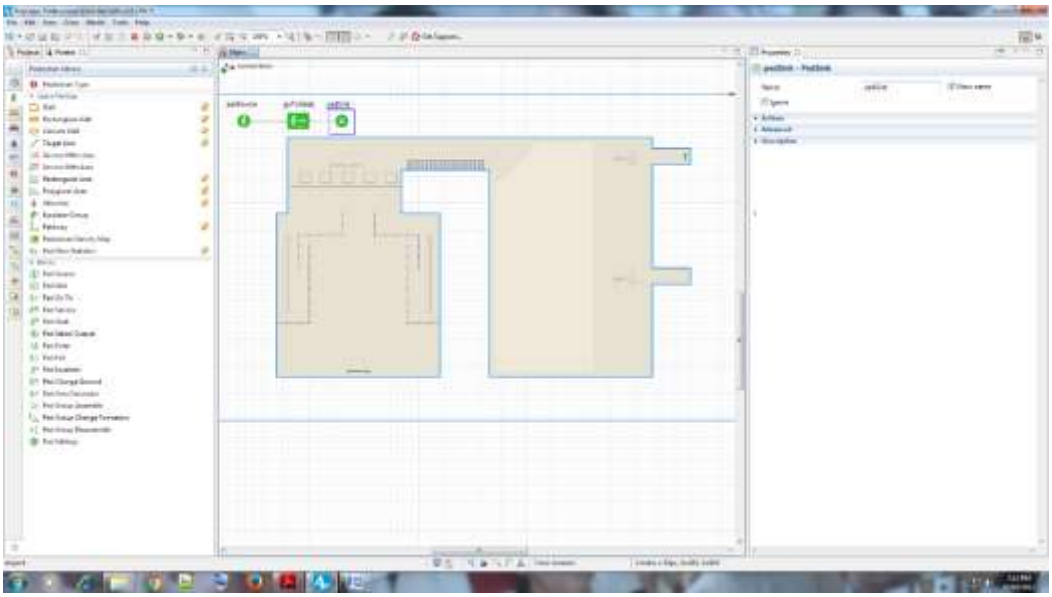
11. Add a **PedGoTo** block to simulate pedestrian movement to the specified location and then connect it to pedSource. Since we want our passengers to go to the gate, name the object goToGate1.



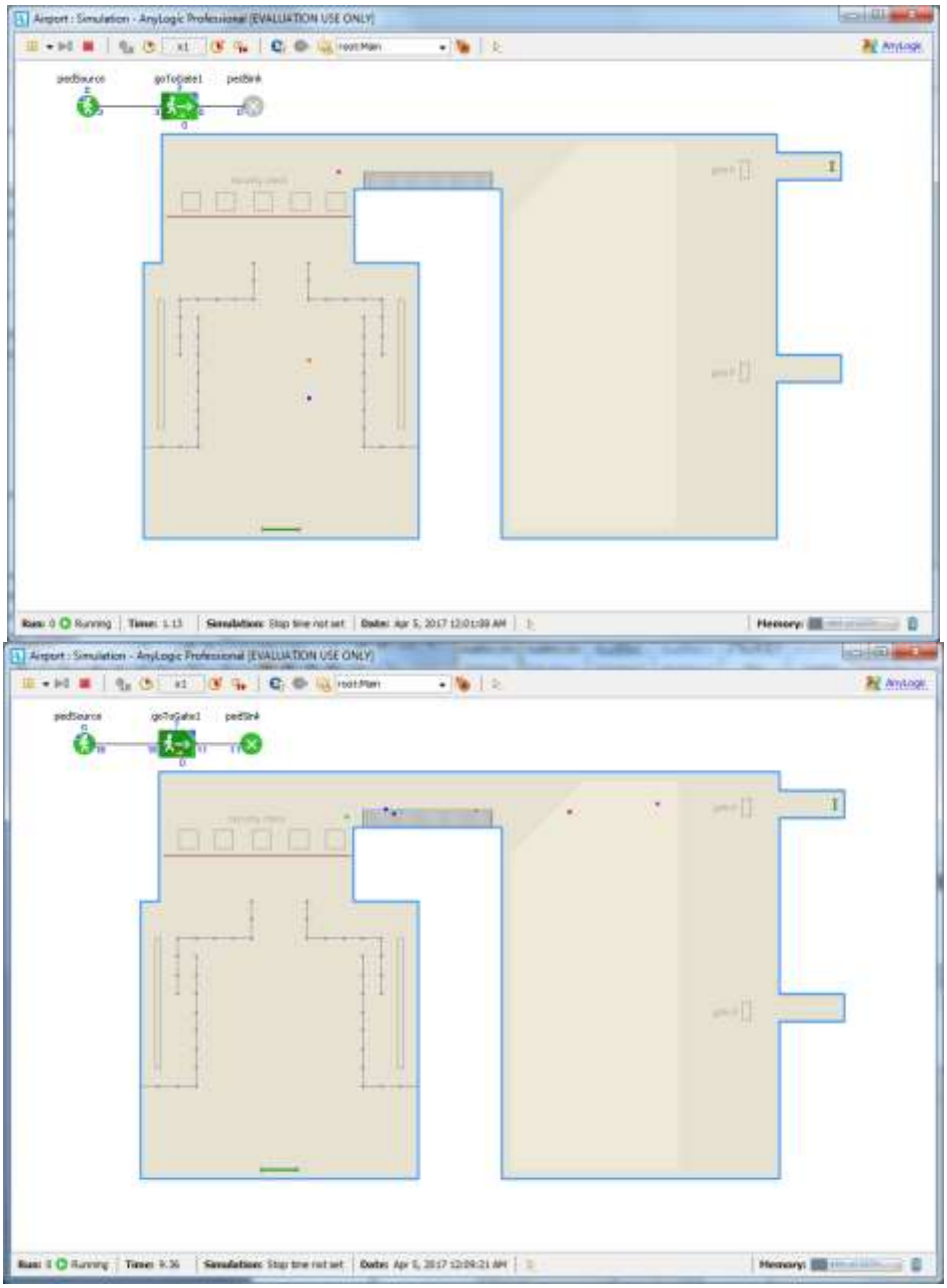
12. Specify the movement destination by selecting gateLine1 from the **Target line** combo box.



13. Add a **PedSink** block to discard incoming pedestrians. Pedestrian flowcharts typically start with a **PedSource** block and end with a **PedSink** block.

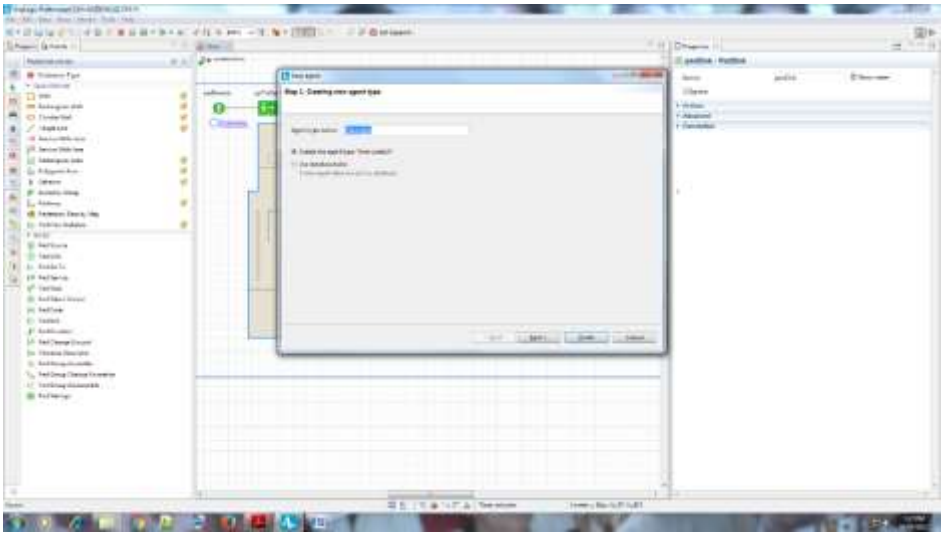


14. Run the model. In the 2D animation, you'll see the pedestrians move from the airport entrance to the gate.

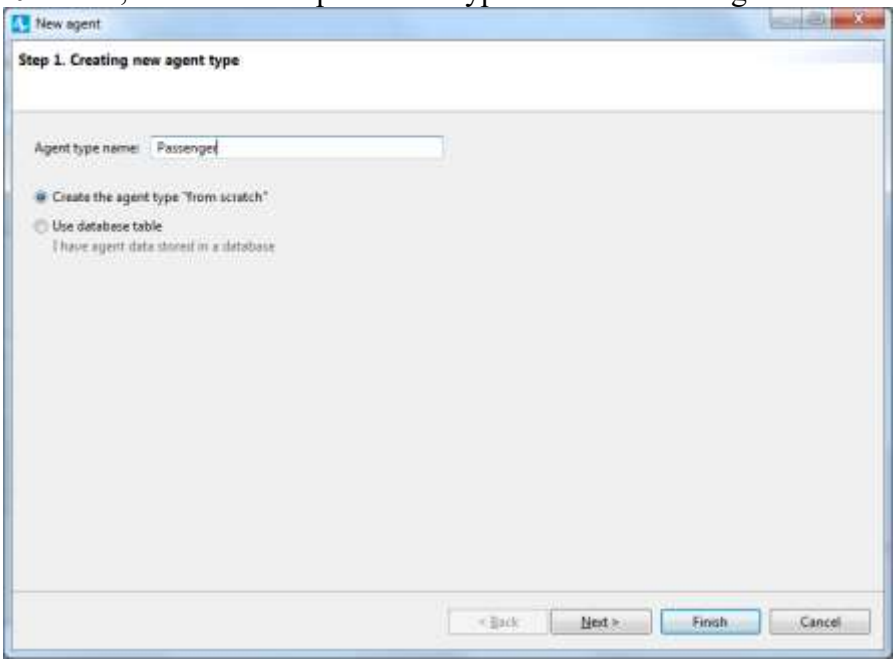


- **Drawing 3D animation**

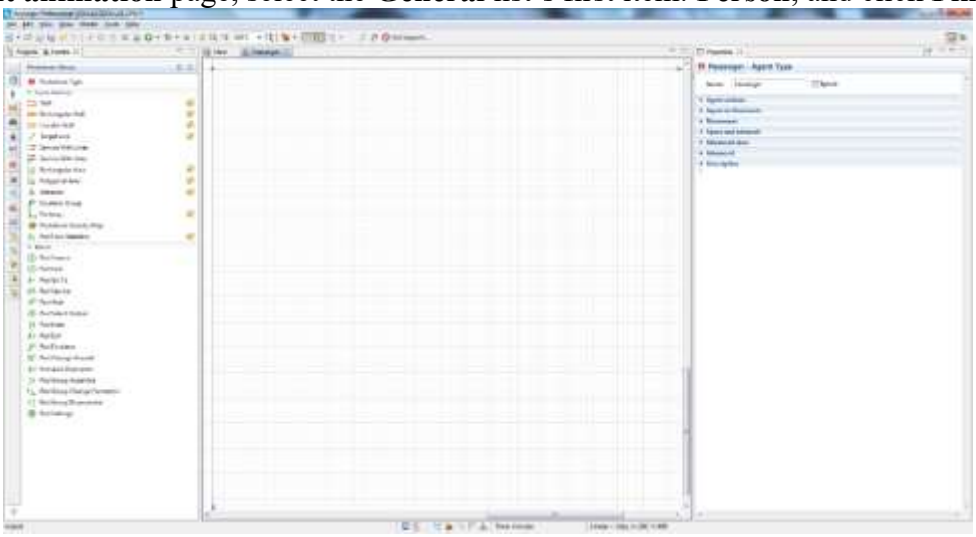
1. From the **Pedestrian Library** palette, drag the **Pedestrian Type** element on to the Main diagram.



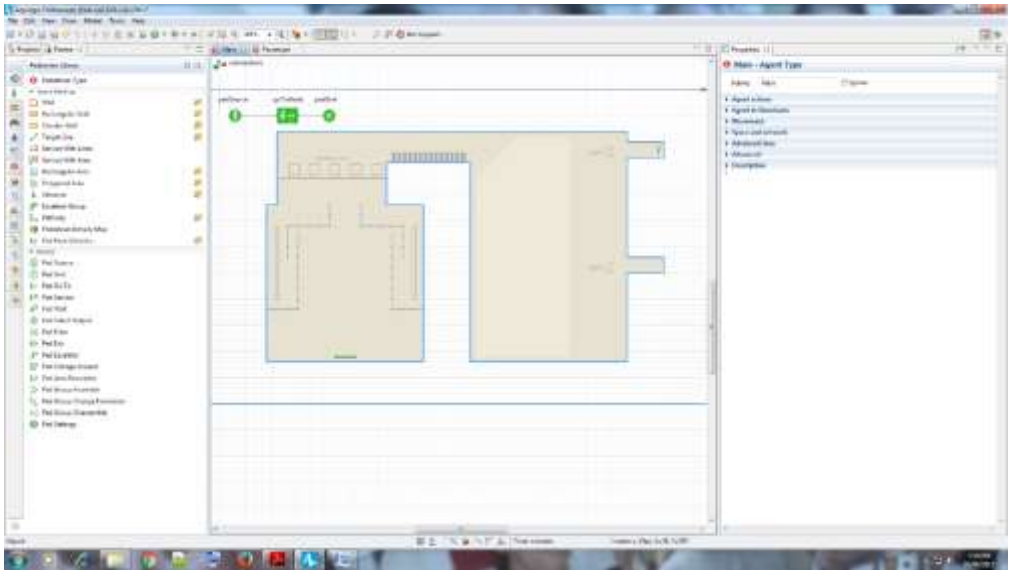
2. In the **New agent** wizard, enter the new pedestrian type's name – Passenger – and then click **Next**.



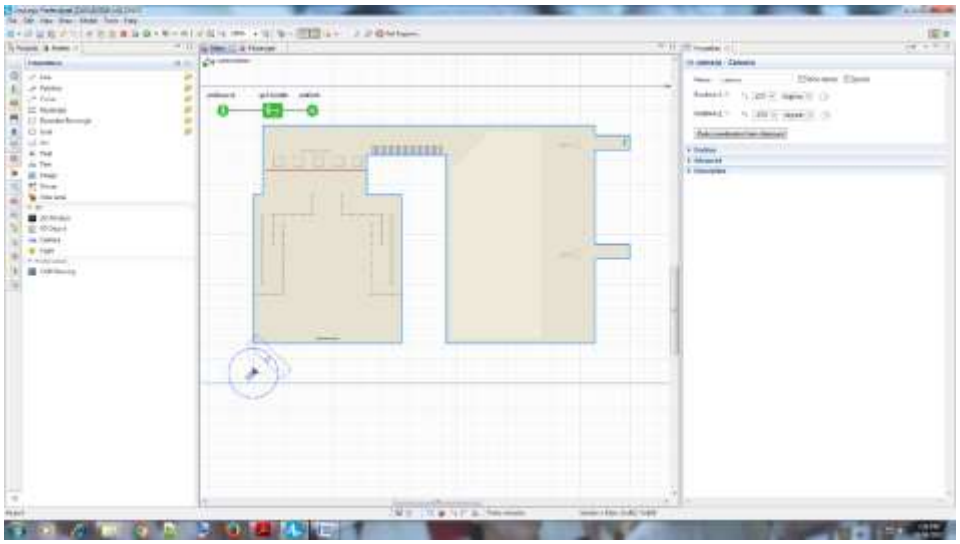
3. On the **Agent animation** page, select the **General** list's first item: **Person**, and click **Finish**.



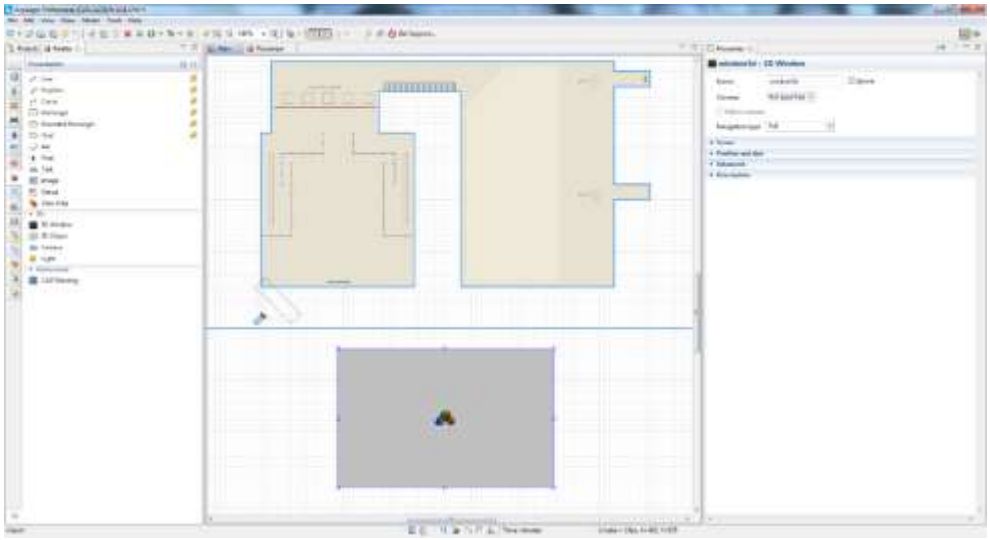
4. After the Passenger diagram opens, return to the Main diagram.



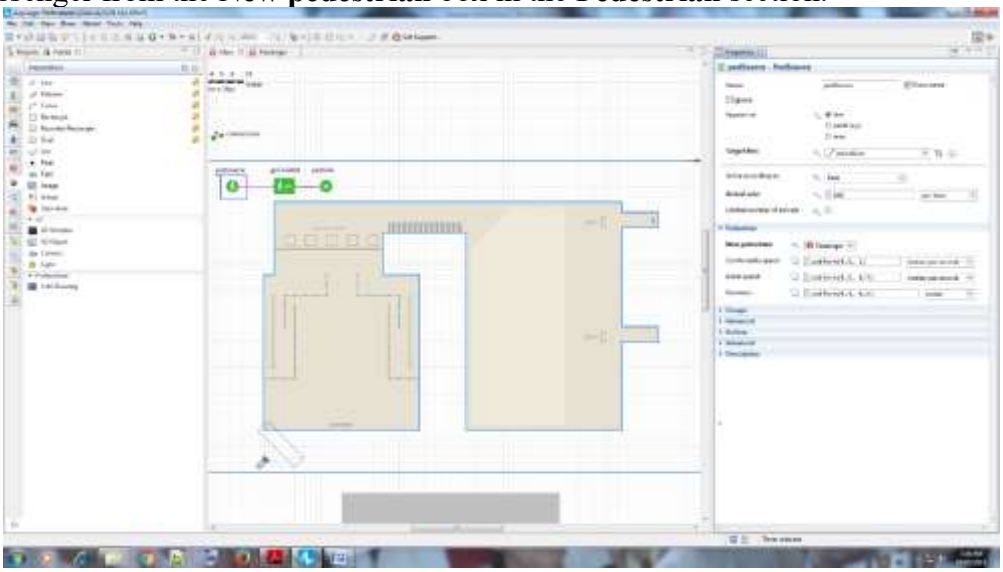
5. From the **Presentation** palette, drag the **Camera** on to the Main diagram and place it so it faces the terminal.



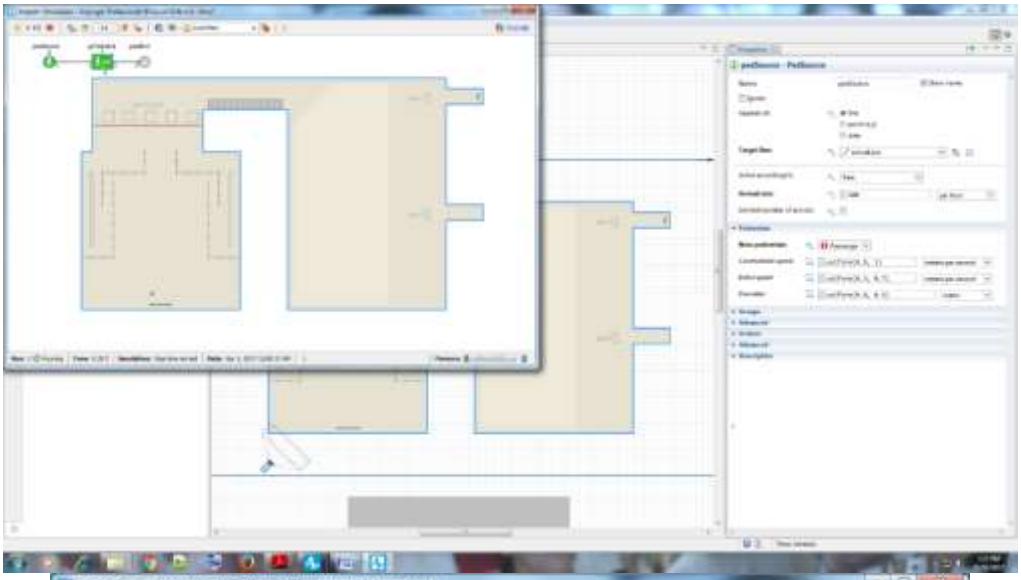
6. Drag the **3D Window** on to the Main diagram and place it below the terminal layout image.

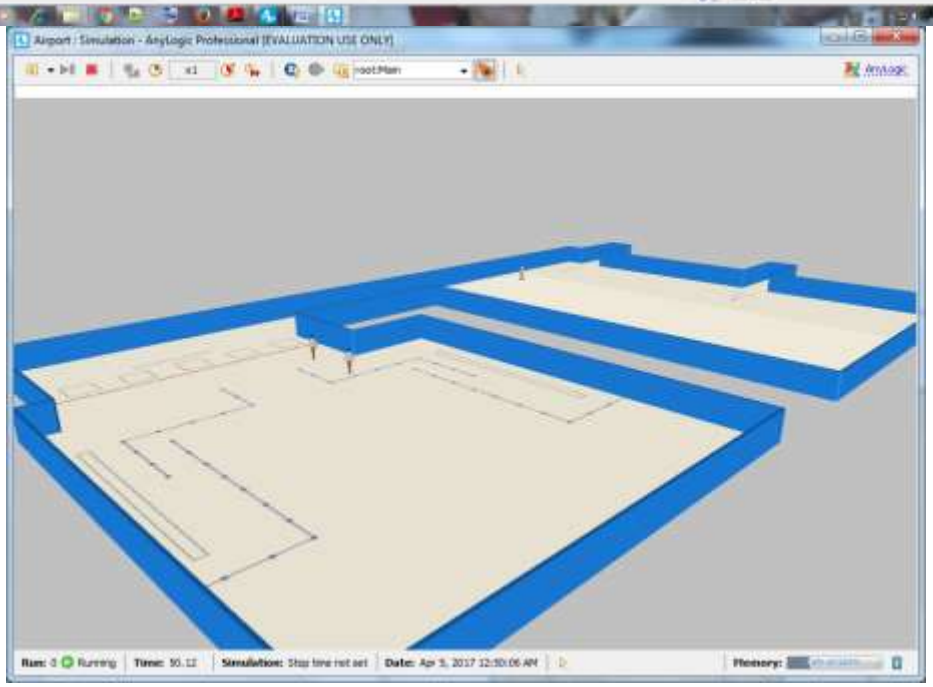


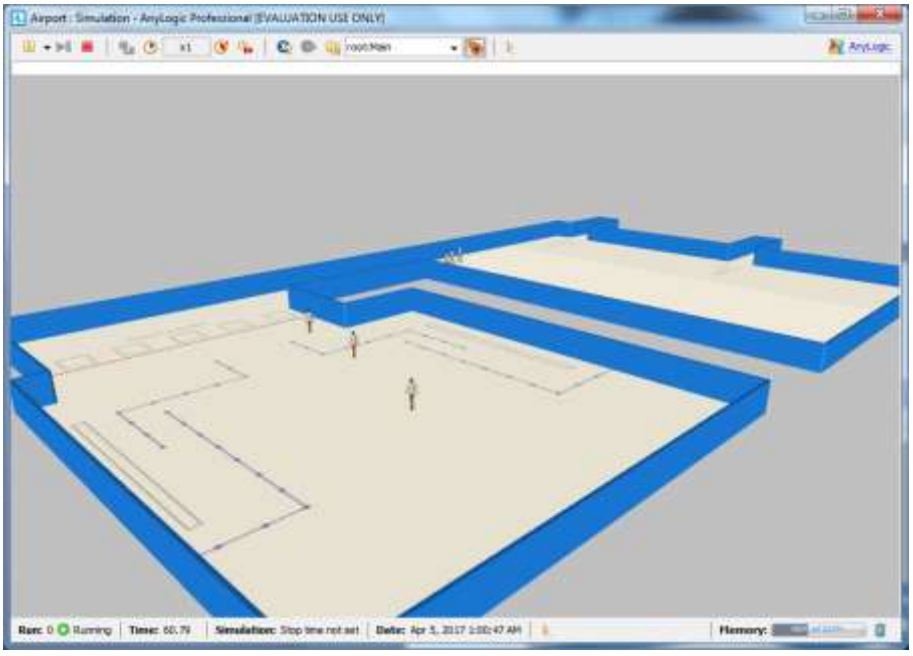
7. Open the **3D Window** properties, and then select camera from the **Camera** list. We want our flowchart block pedSource to create pedestrians of our custom Passenger type. Open the pedSource properties, and then select Passenger from the **New pedestrian** box in the **Pedestrian** section.



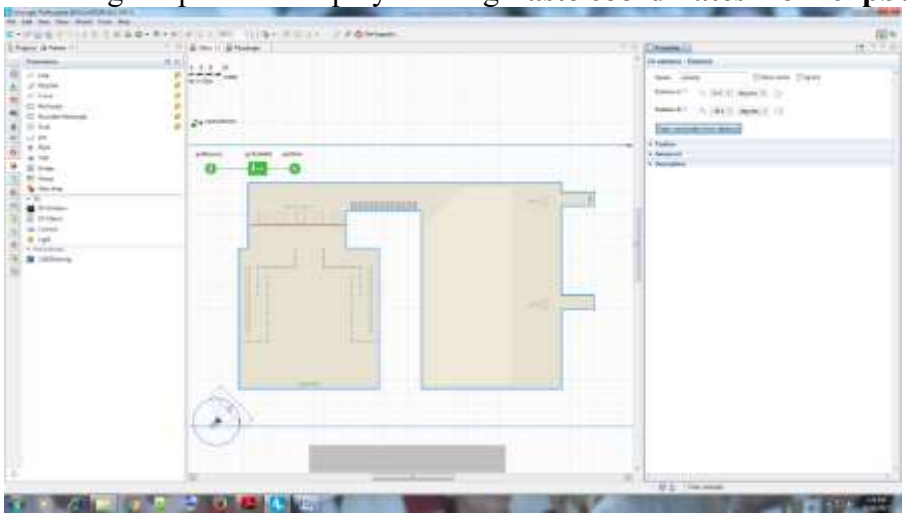
8. Run the model, and you'll see pedestrians move from the entry to the gate inside the building. You can switch to a 3D view by clicking the toolbar's **Navigate to view area…** button and then selecting **[window3d]** from the list.



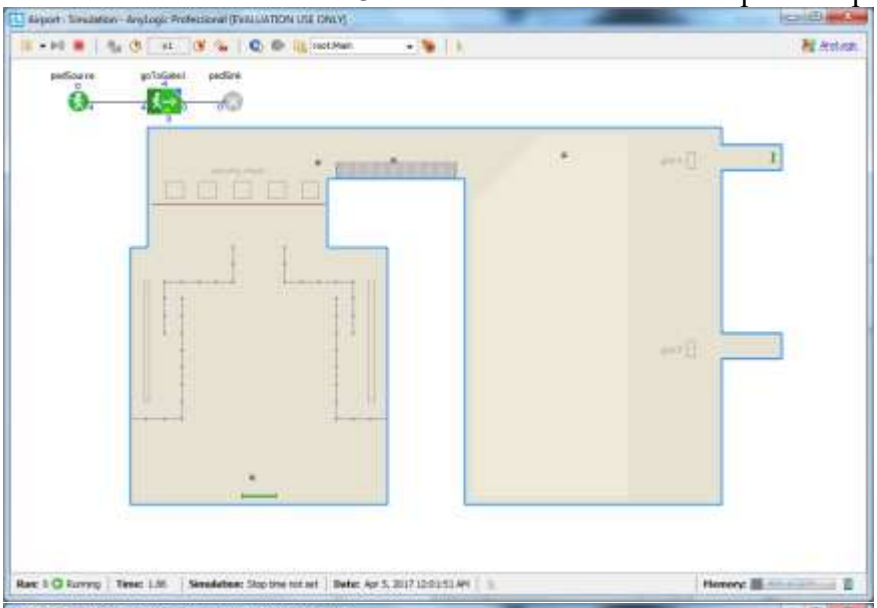


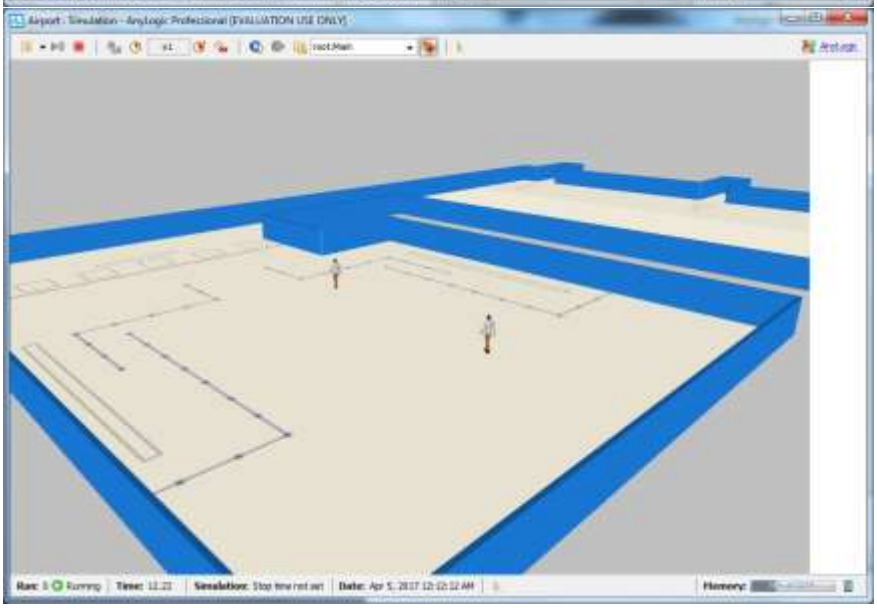


9. Navigate the scene to get the best view, right-click inside the 3D scene, and then click **Copy the camera's location**. Close the model's window, open the camera's properties, and then apply the optimal camera you selected during the previous step by clicking **Paste coordinates from clipboard**.



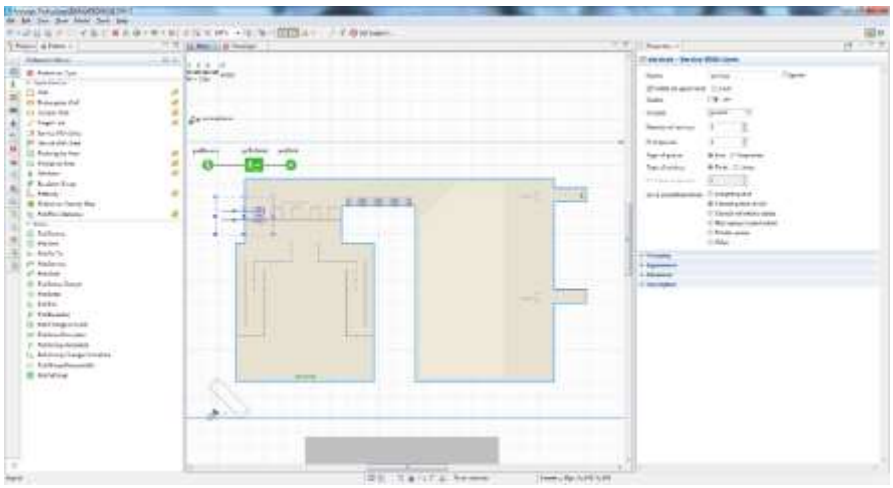
10. Run the model a second time and view the 3D view that the new camera position provides.



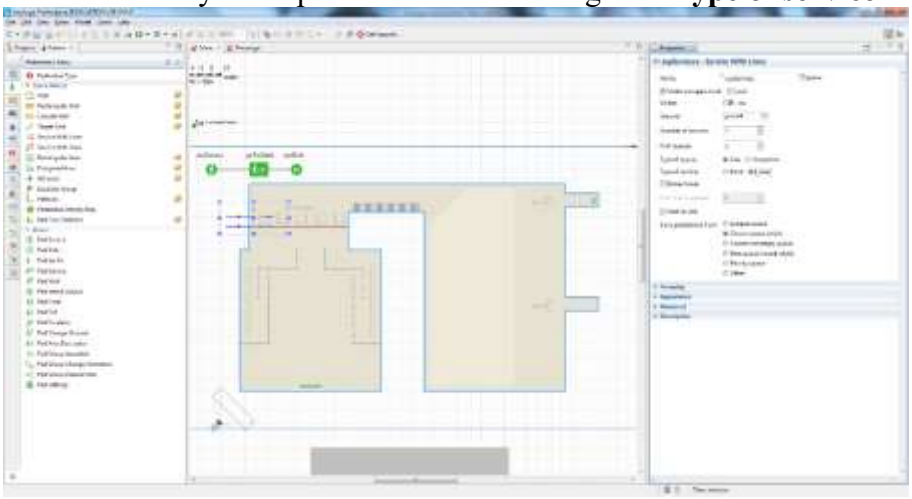


- **Adding security checkpoints**

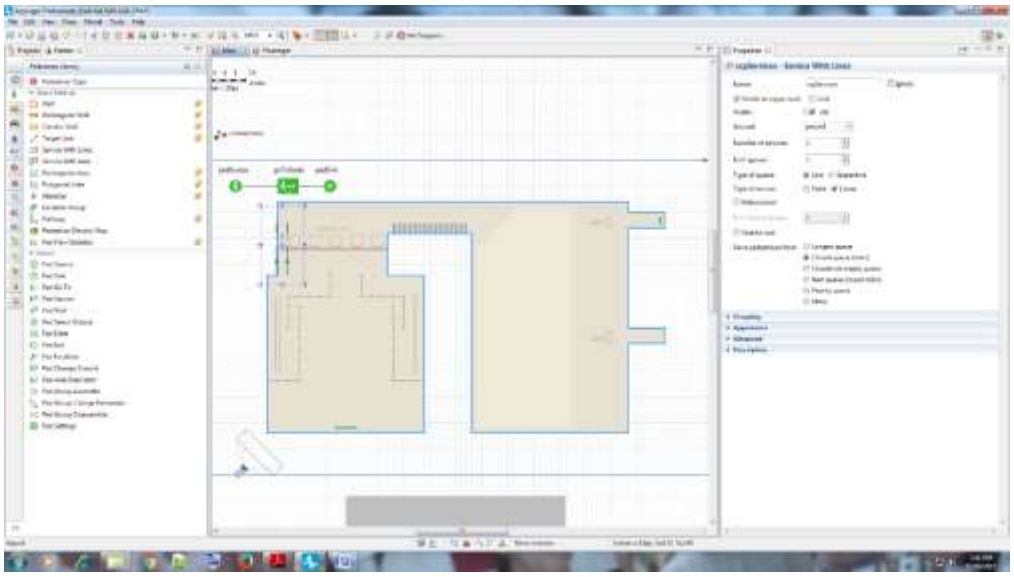
1. Drag the **Service with Lines** element from the **Pedestrian Library** palette on to the terminal layout. By default, a service will have two service points and two queue lines that lead to the service points.



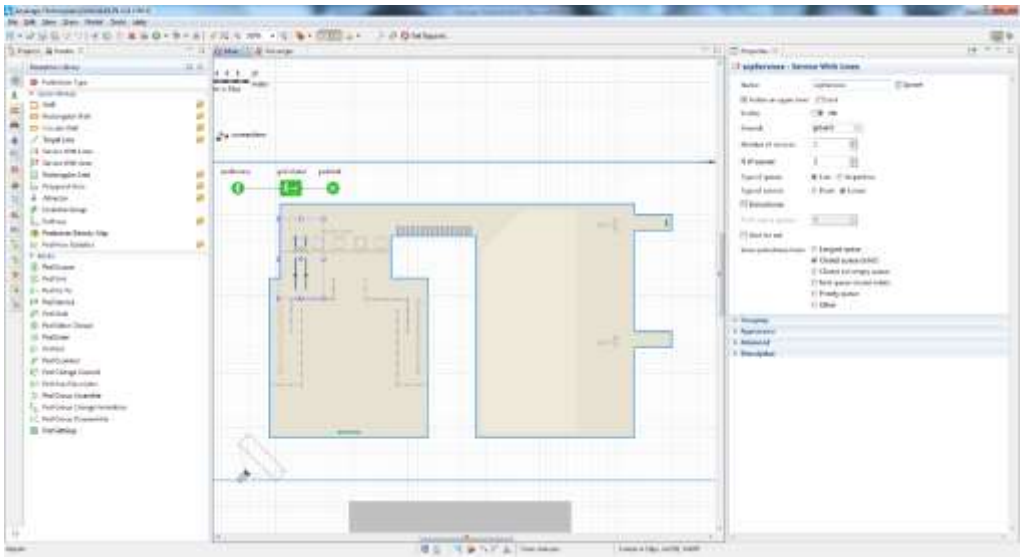
2. Open the **Service With Lines** properties area, use the **Name** box to name the shape scpServices - in this case, “scp” stands for security checkpoints – and then change the **Type of service** to Linear.



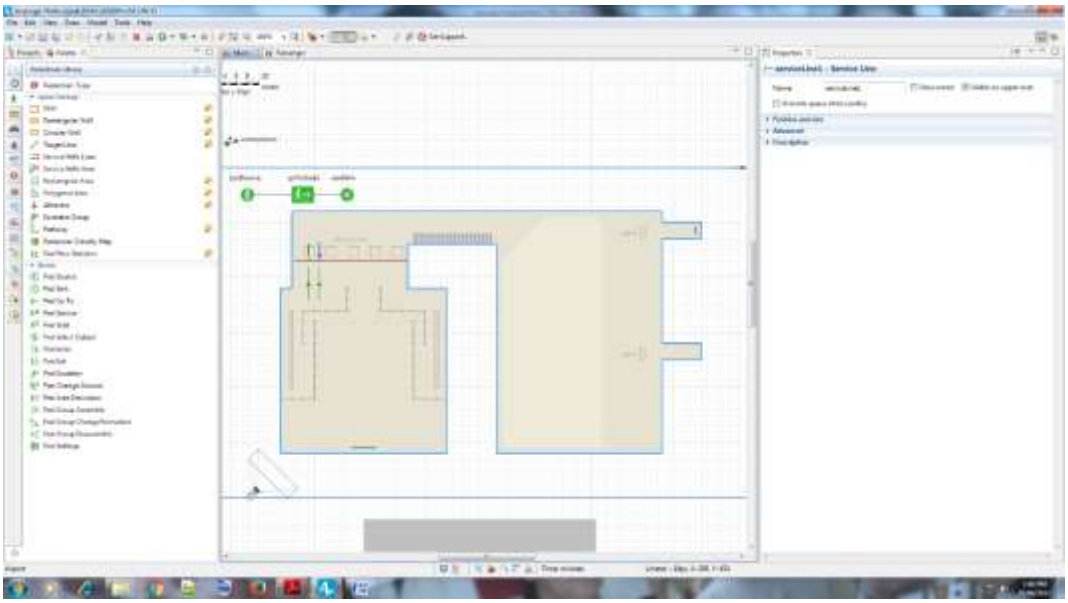
3. Use the round handle above the shape’s center to rotate the service. So that the arrow points upward.



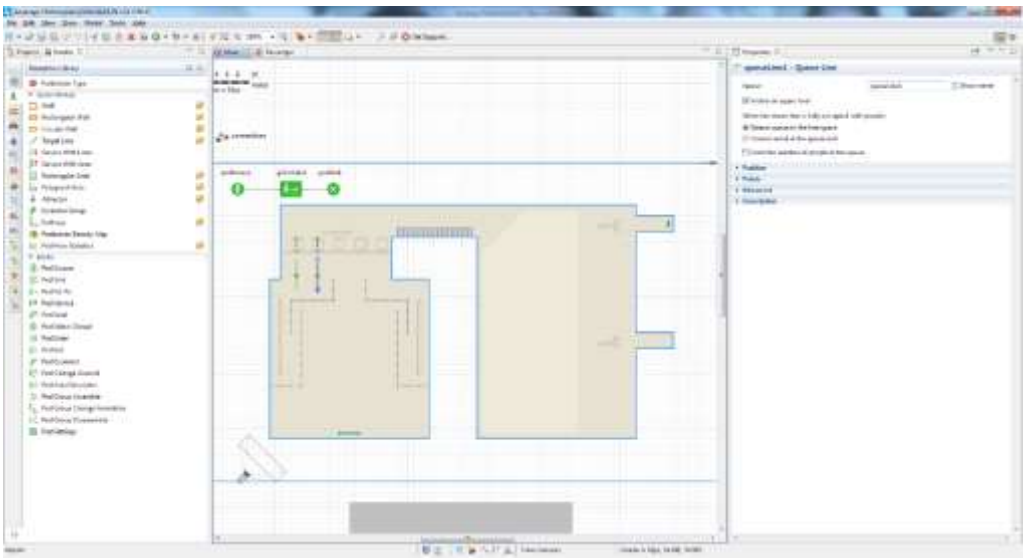
4. Move the service in a way that ensures the first linear service crosses the rectangle that represents the metal detector frame.



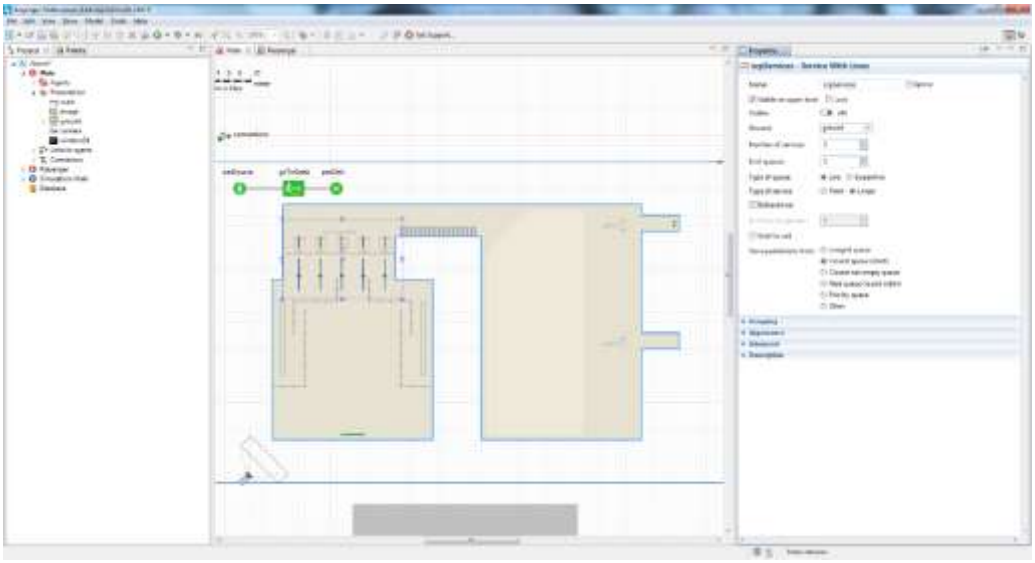
5. Select the next service line.



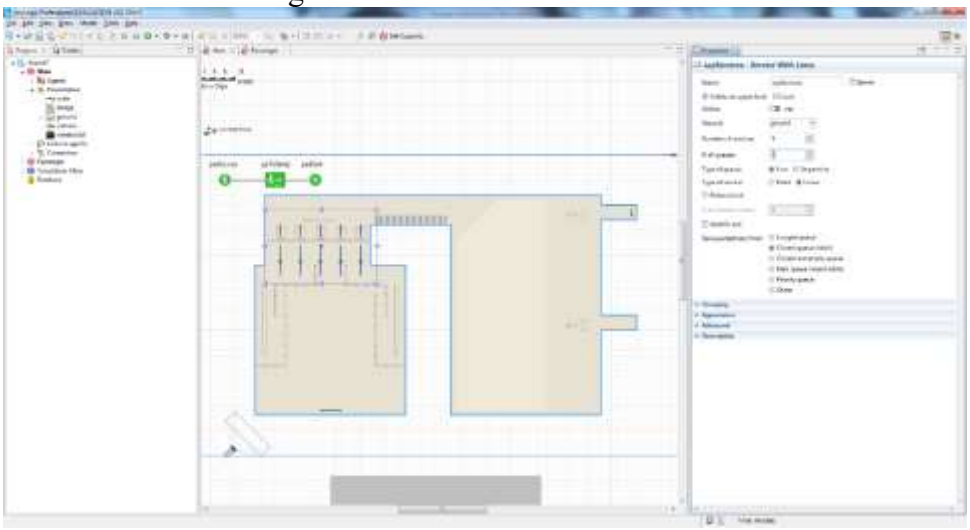
6. Accurately place the service line on top of the second security checkpoint placeholder and then adjust the queue location.



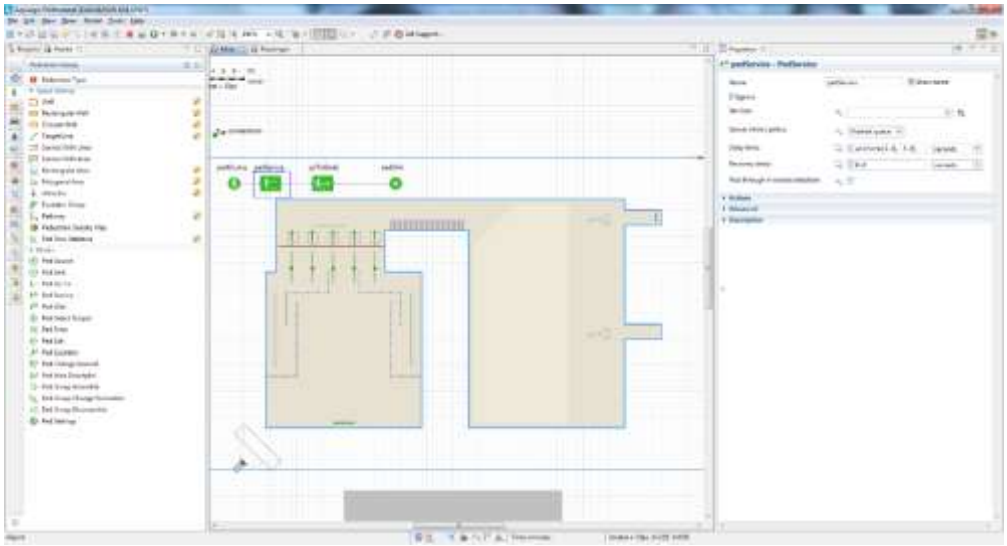
7. Navigate to the **Service with Lines** shape's properties and then change both the **Number of services** and **Number of queue** to 5.



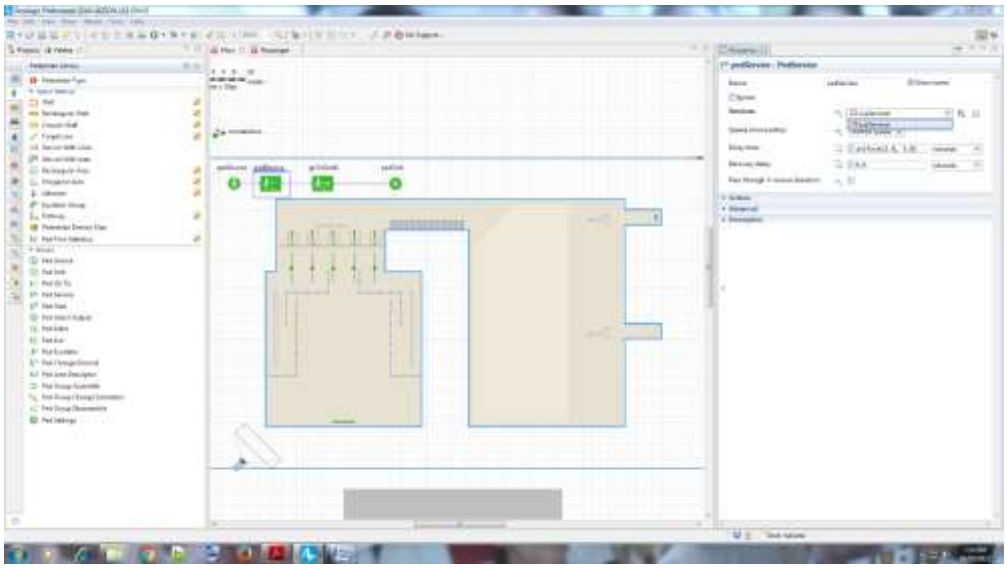
8. If necessary, adjust the new service and queue lines. After you've completed this step, the service shapes should look like those in the figure below.



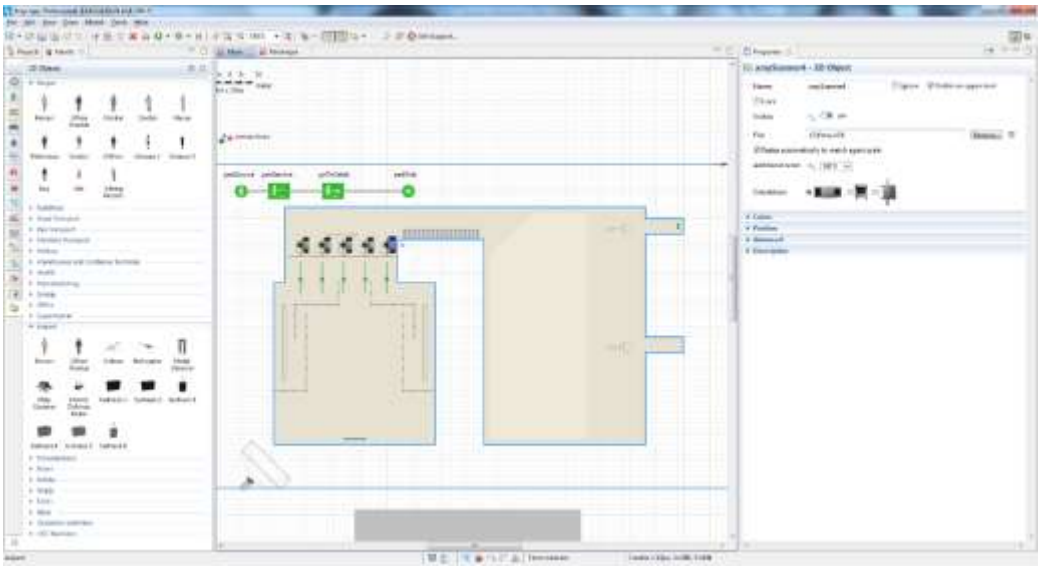
9. Add the **PedService** block on to the flowchart between the **PedSource** and **PedGoTo** blocks to make pedestrians pass through the service we defined using the referenced Service with Lines shape, and then name it securityCheck.



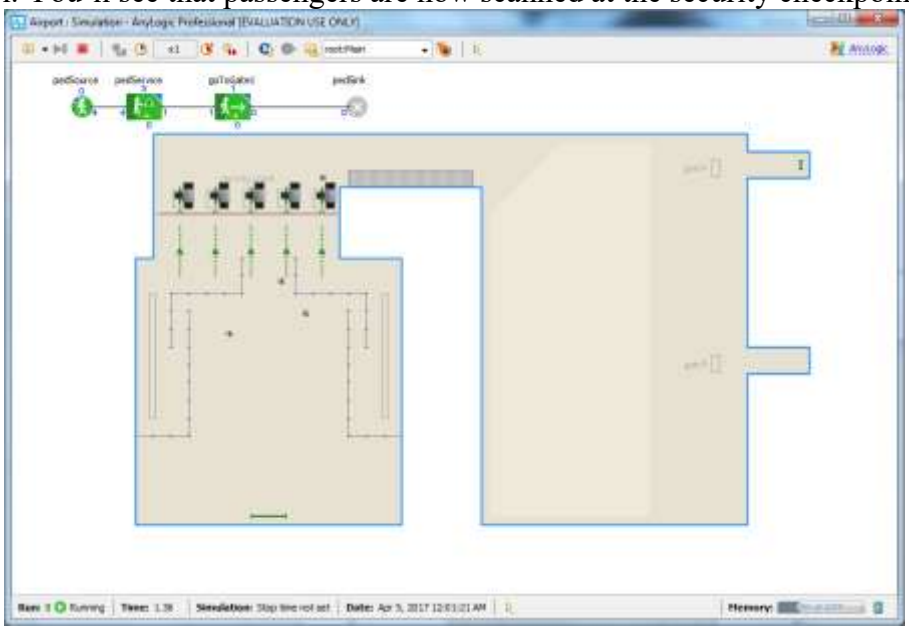
10. Go to the securityCheck block's properties. Select the services scpServices as **Services**.

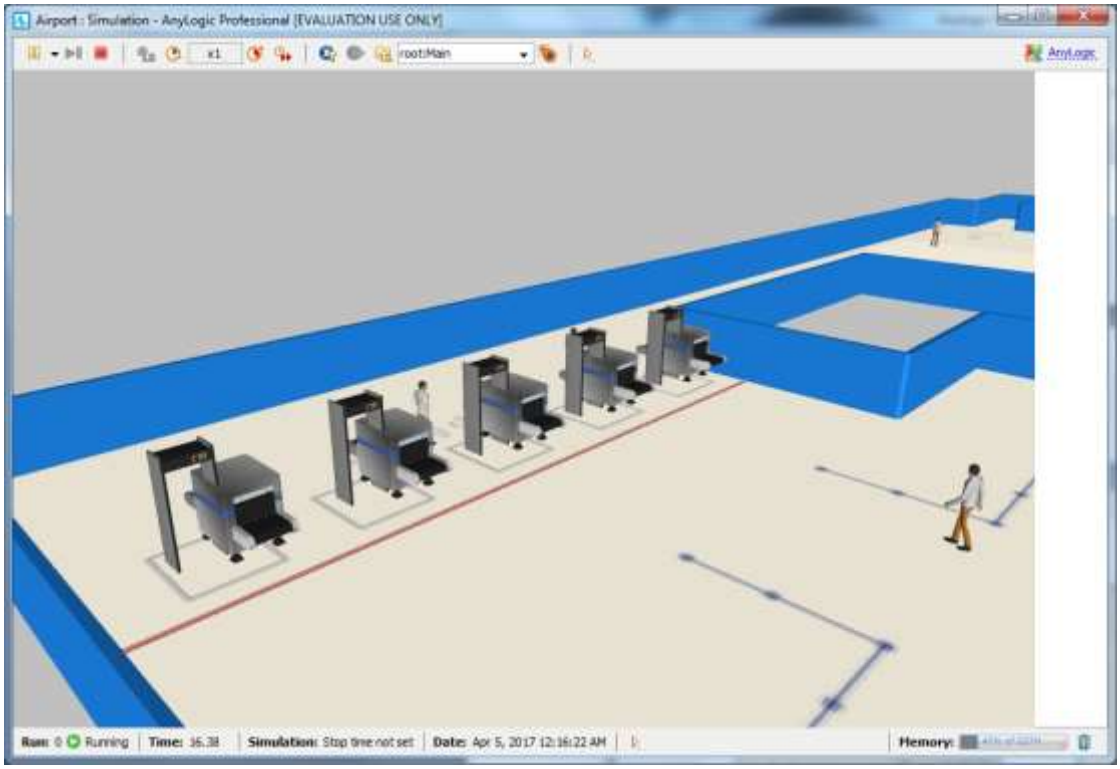


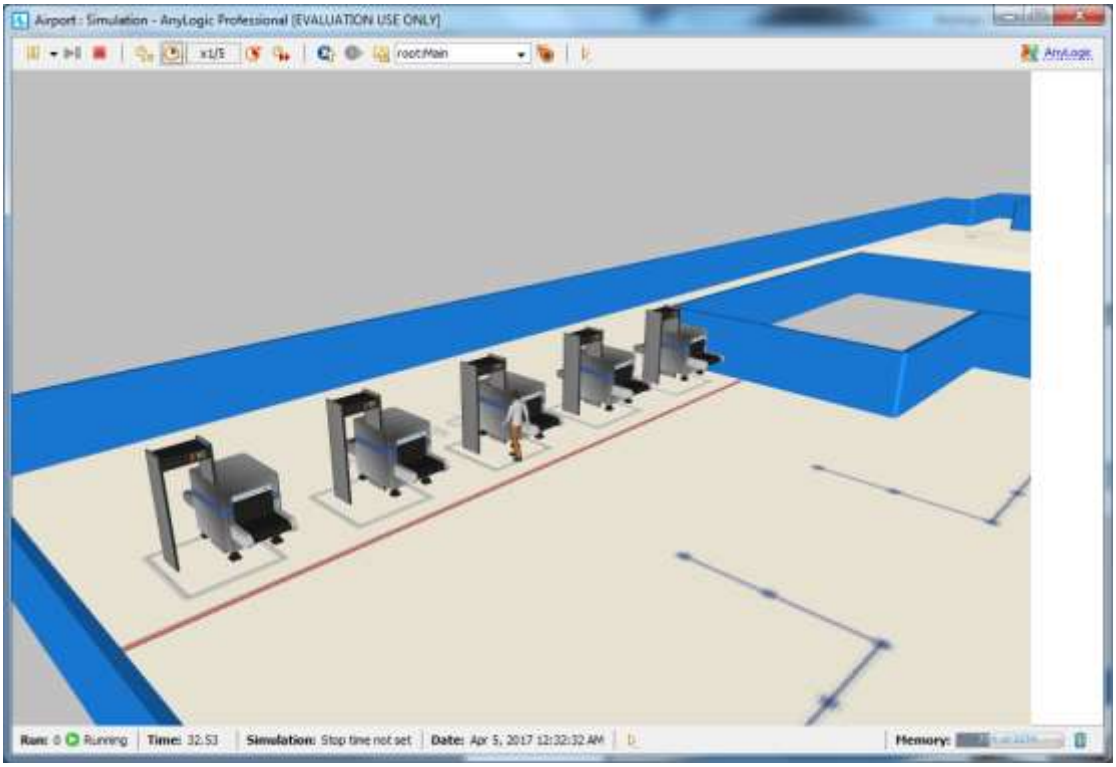
11. Since we assume it takes between 1 to 2 minutes to pass through the security checkpoint, type uniform(1, 2) **minutes** as the **Delay time**. Now let's add 3D models of the security checkpoints. Using the **3D Objects** palette, **Airport** section's Metal Detector and XRay Scanner elements, draw five security checkpoints. You will see the message box, prompting you to change the scale of 3D object. Select the option **Do not ask me again** and click **OK**.



12. Run the model. You'll see that passengers are now scanned at the security checkpoints.

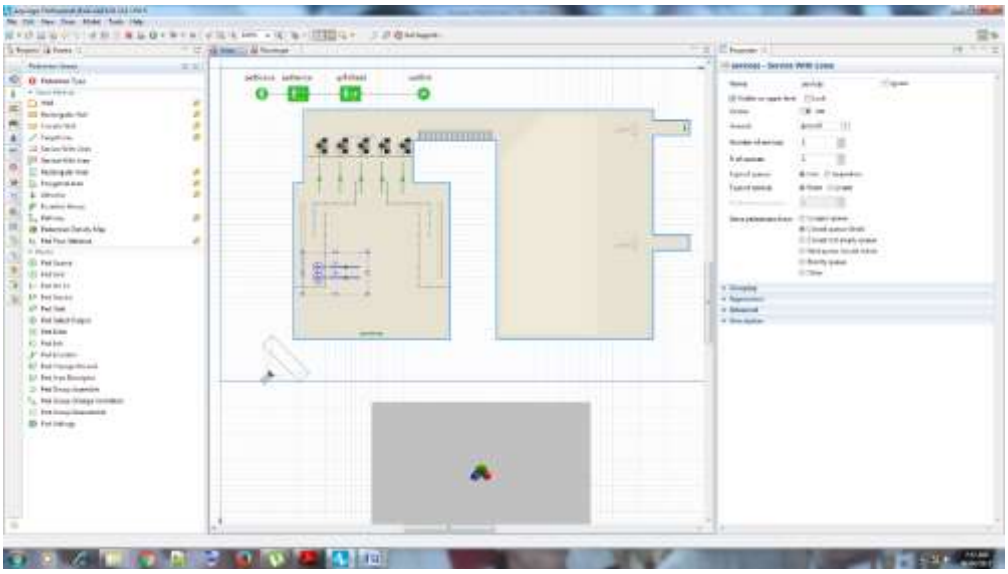




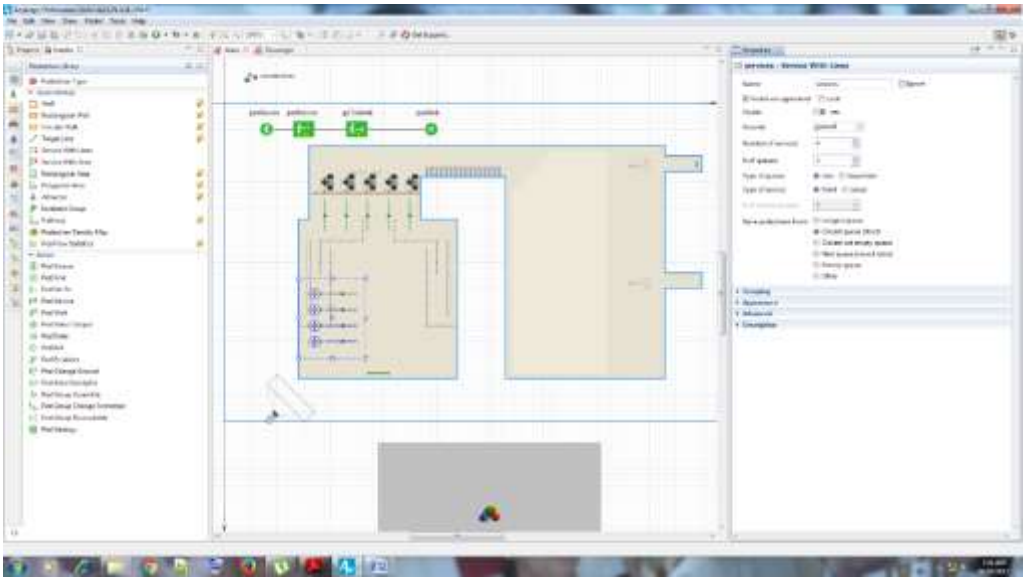


- **Adding check-in facilities**

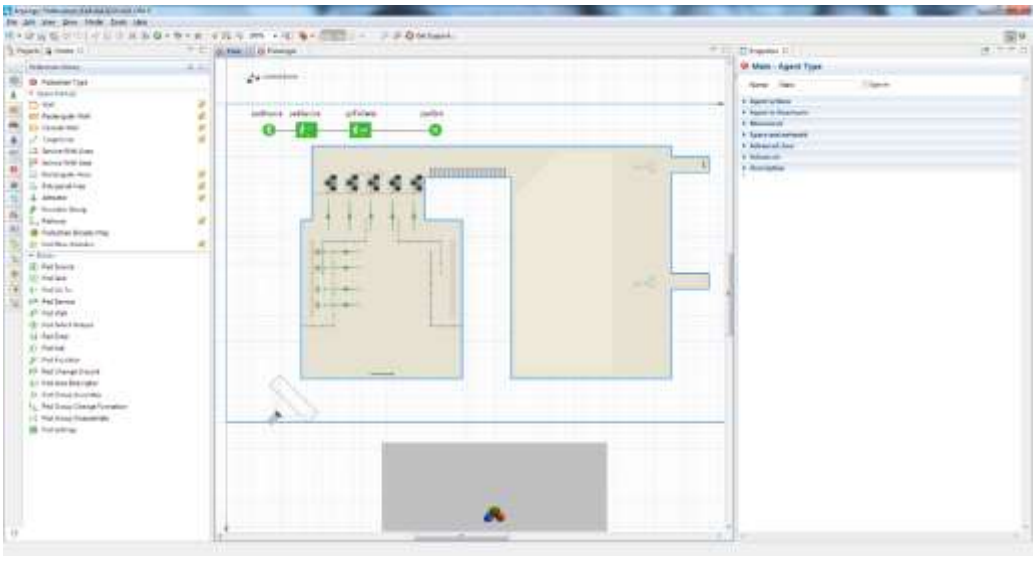
1. Draw check-in locations with another **Service with Lines** shape. Rotate it so that the arrow points towards left.



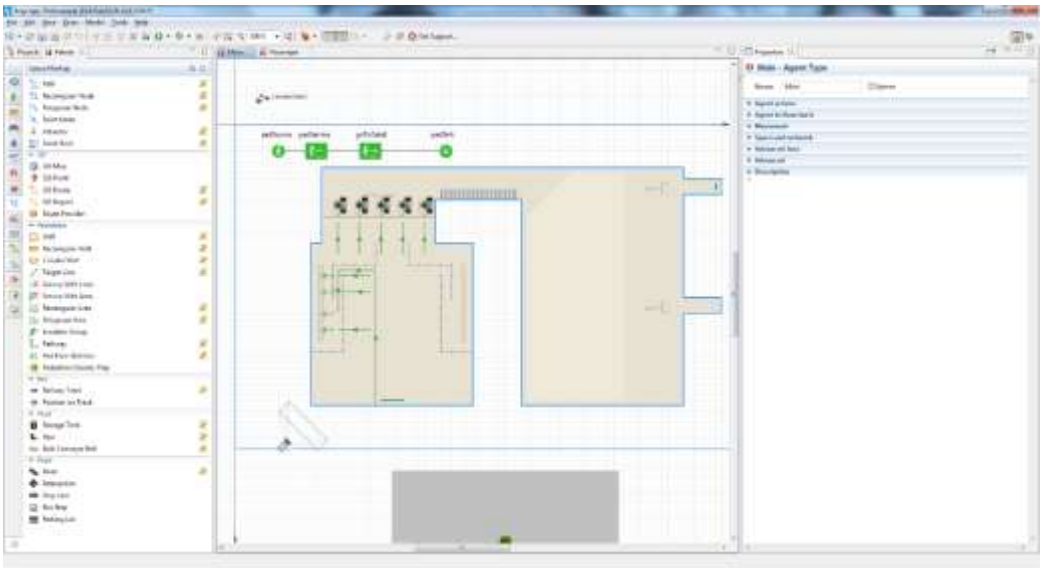
2. Navigate to the **Service with Lines** shape's properties & then change both the **Number of services** & **Number of lines** to 4.



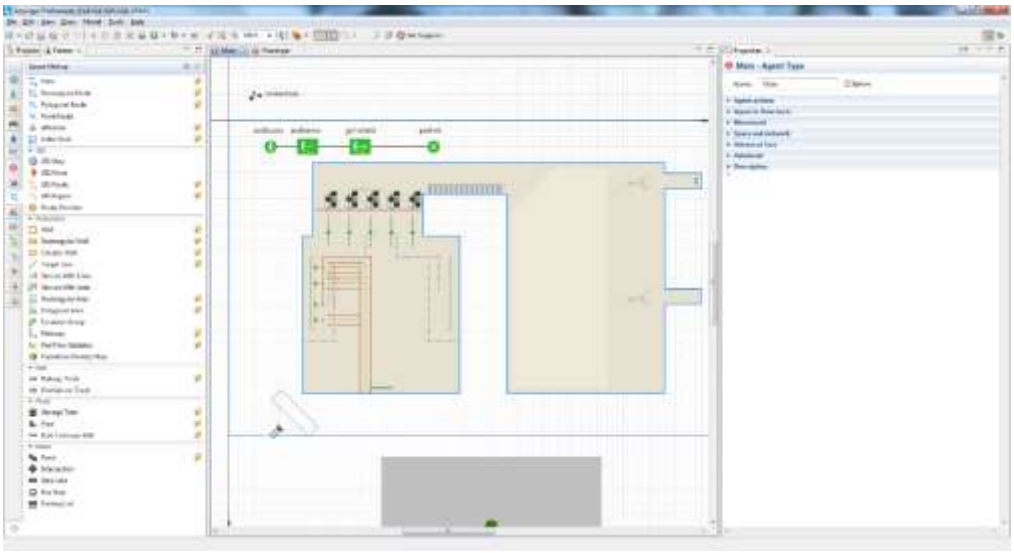
3. Name the services checkInServices. Place the shape in the location shown in the figure below



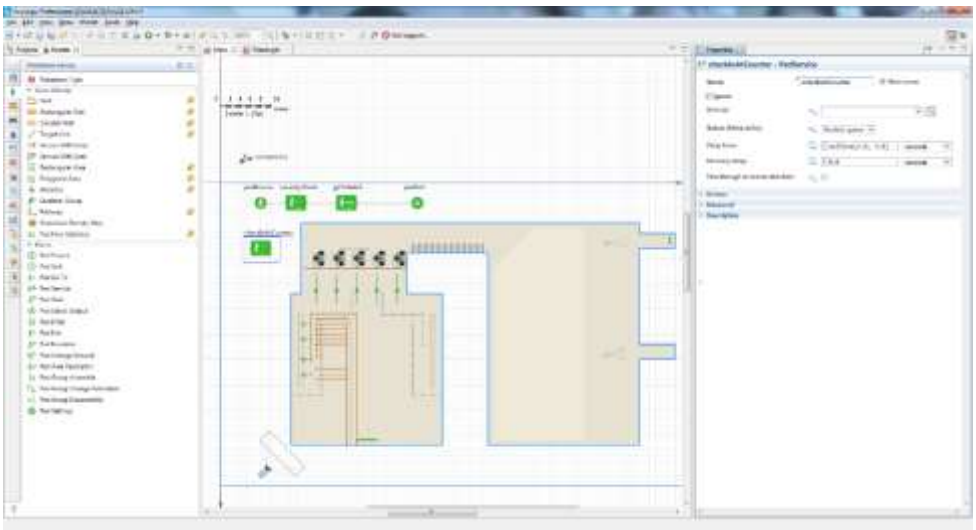
4. Add more salient points to the line. Right-click the queue line and choose **Add points** from the pop-up menu. Add more points by clicking where you want to place the line's salient points. Finish drawing the line by double-clicking. Finally you should get the queue line of the following form



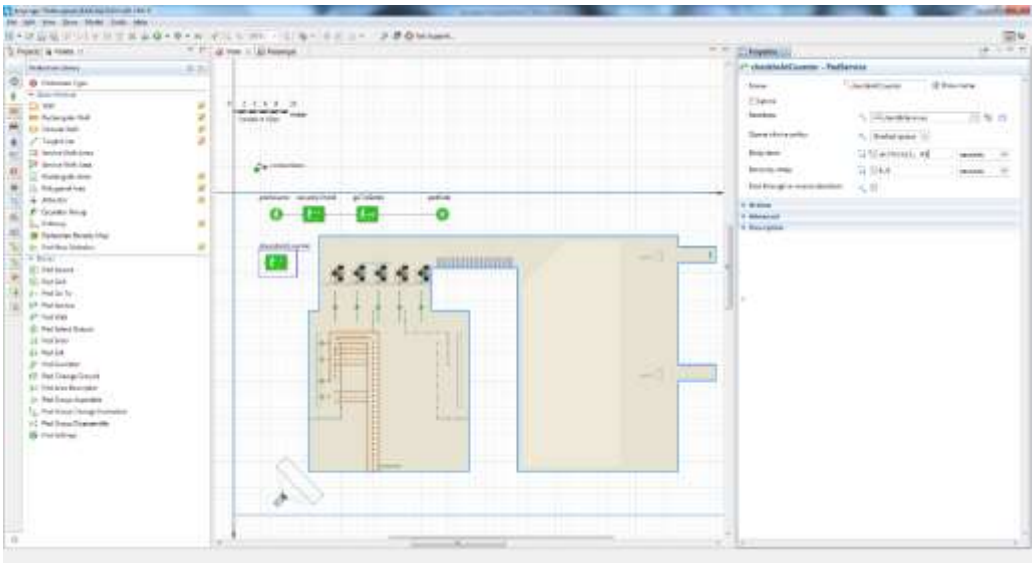
5. In the properties of this **Service with Lines** shape, change the **Type of queue** to **Serpentine**. Use this option to simulate serpentine (also named "zigzag") queues. You will see that the queue has borders now. In 3D animation they will appear as belt barriers.



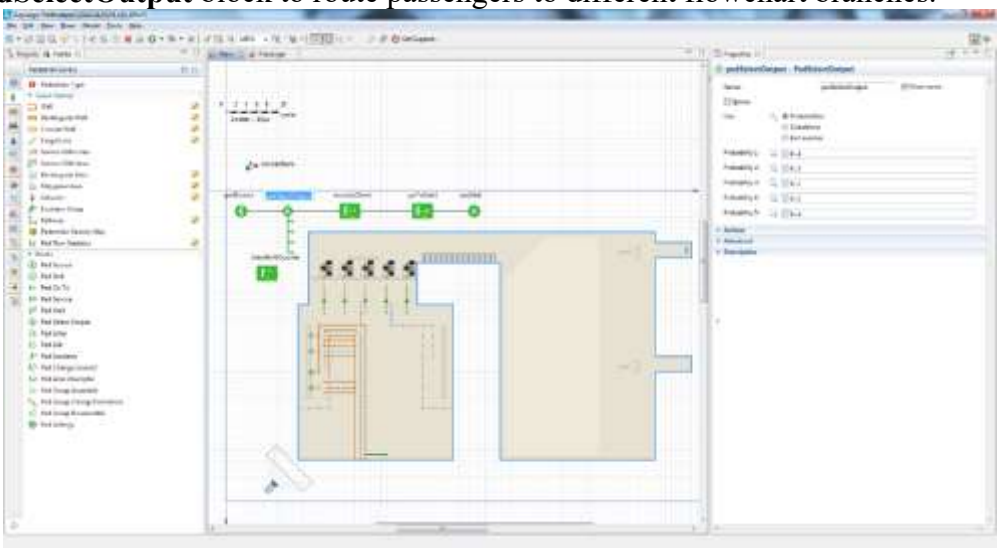
6. Add another **PedService** block and name it checkInAtCounter



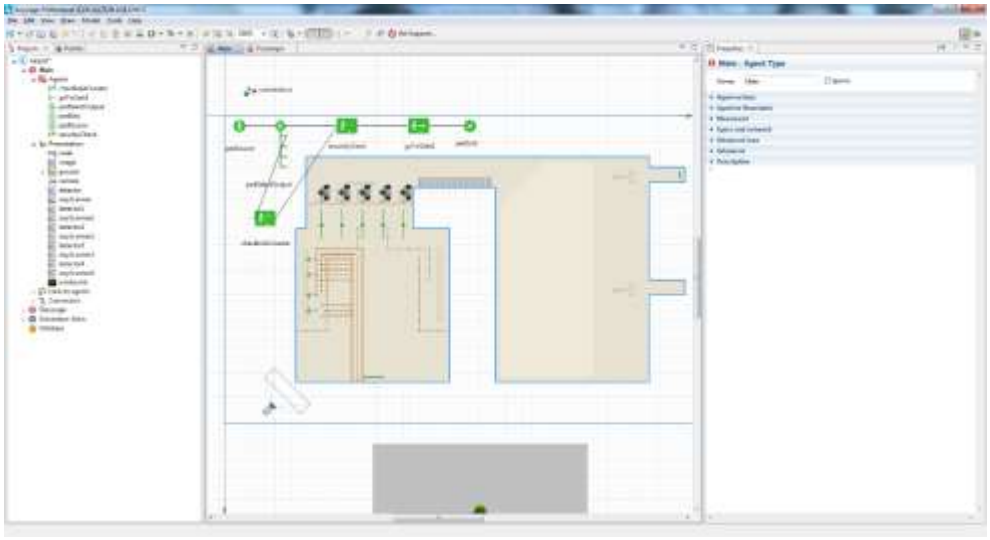
7. In the block's properties, select the space markup shape checkInServices as **Services**. Since we assume it takes between 2 to 4 minutes to check in, type uniform(2, 4) **minutes** as the **Delay time**.



8. Add the **PedSelectOutput** block to route passengers to different flowchart branches.

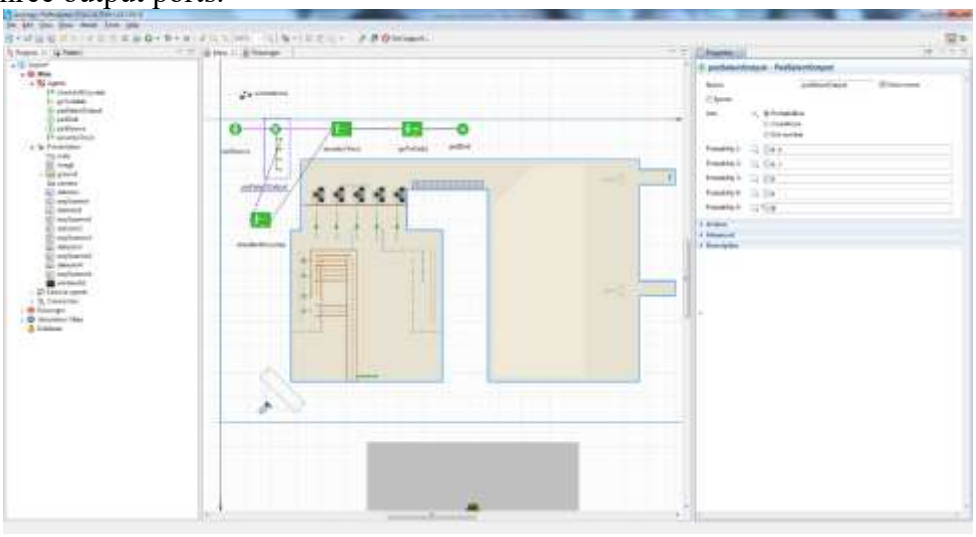


9. Connect the checkInAtCounter block to the existing flowchart blocks as shown in the figure below.

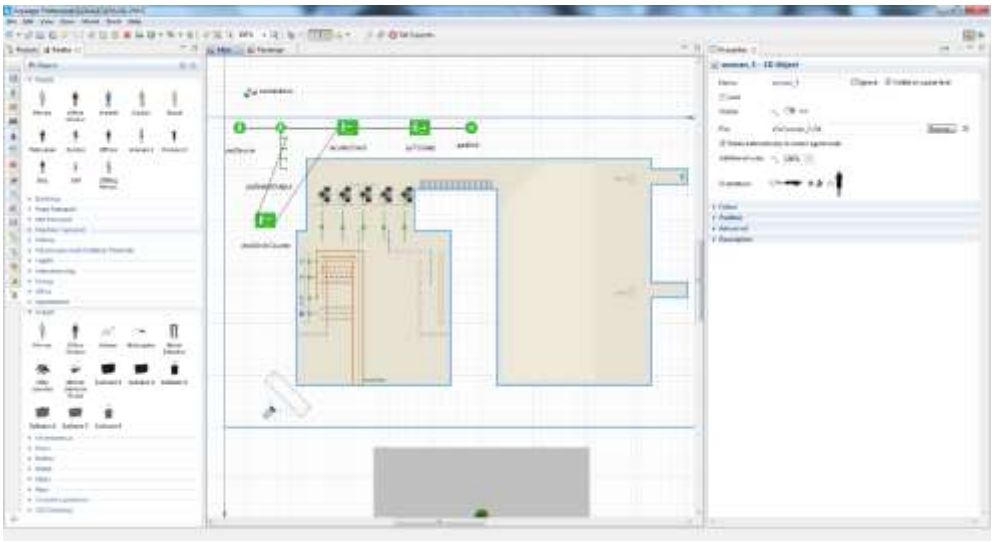


10. Since we're assuming 30 percent of our passengers will check in online and 70 percent will check in at the counter, we'll model this behavior by setting pedSelectOutput's **Probability 1**: to 0.3 and **Probability 2**: to 0.7. This action will route 30 percent of the passengers to the upper flowchart branch and 70 AnyLogic 7 in Three Days **219** percent of the passengers to the lower branch. You must set

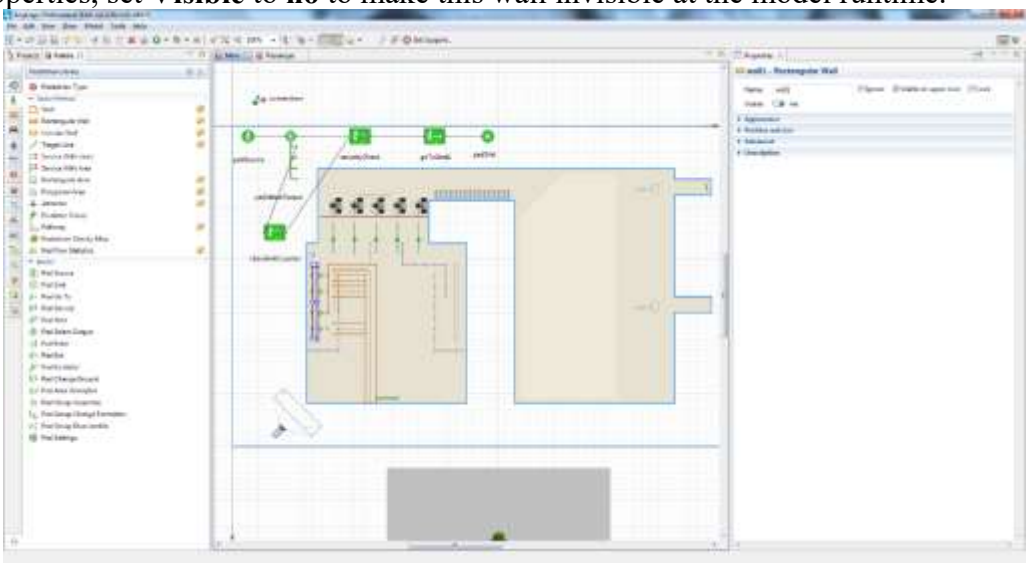
Probability 3, **Probability 4**, and **Probability 5** to 0 to prevent AnyLogic from routing passengers to the block's lower three output ports.



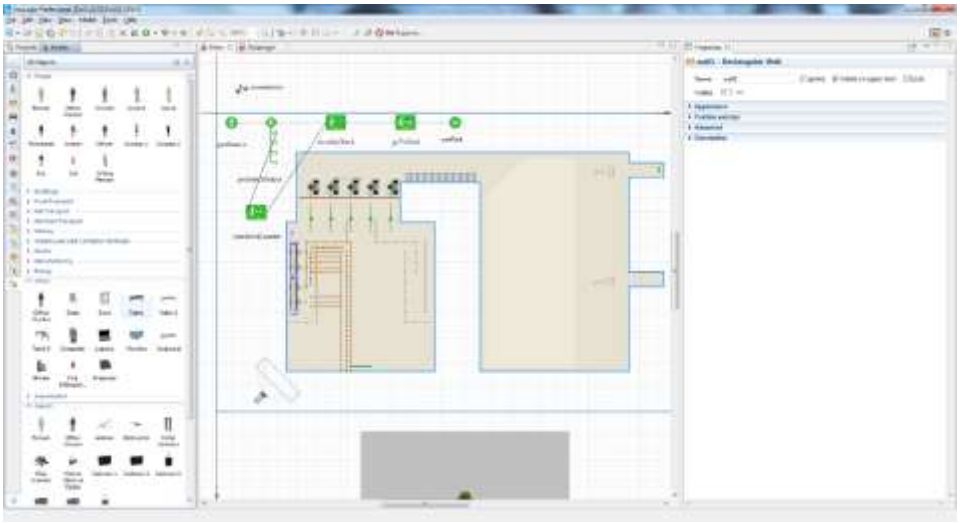
11. Let's add ready-to-use 3D models to the airport's check-in area. On the Palette's **3D Objects** tab, expand the **People** section, and then add two copies of both Office Worker and Woman 2. to the diagram. As shown below

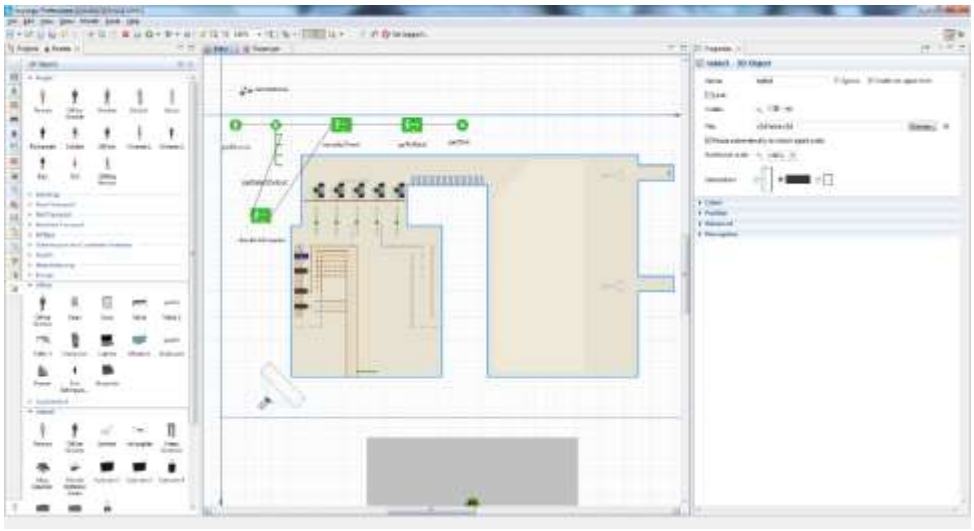


12. Define the area that is not accessible by passengers. Add the **Rectangular Wall** element from the **Space markup** section of the **Pedestrian Library** palette and place it as shown in the figure below. In the wall's properties, set **Visible** to **no** to make this wall invisible at the model runtime.

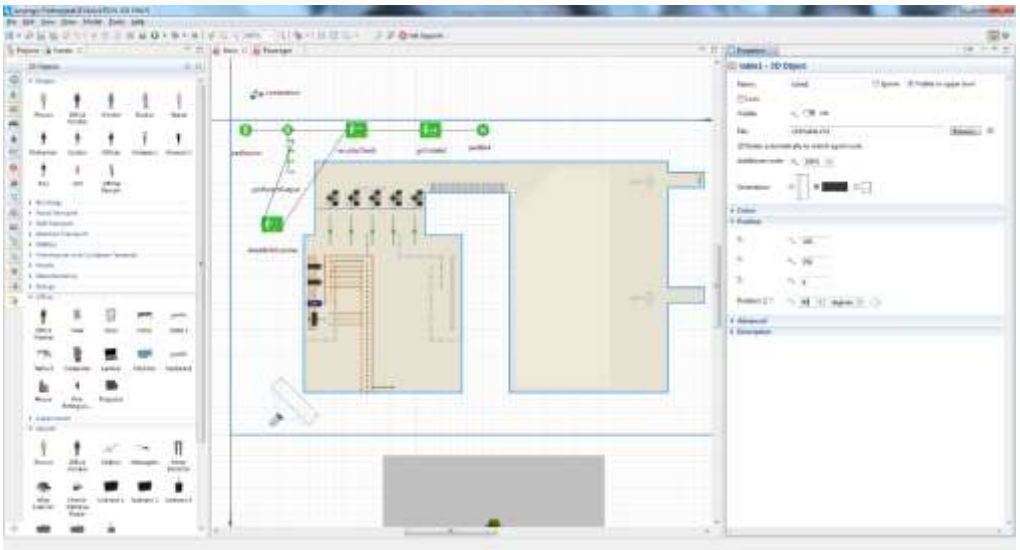


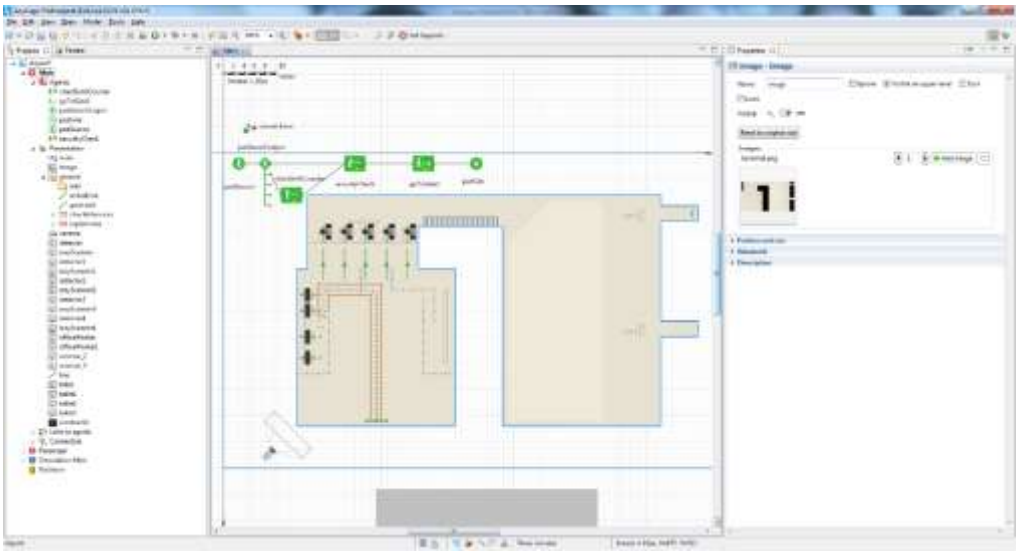
13. On the Palette's **3D Objects** tab, select the **Office** section, and then drag four copies of the **Table** object on to the diagram



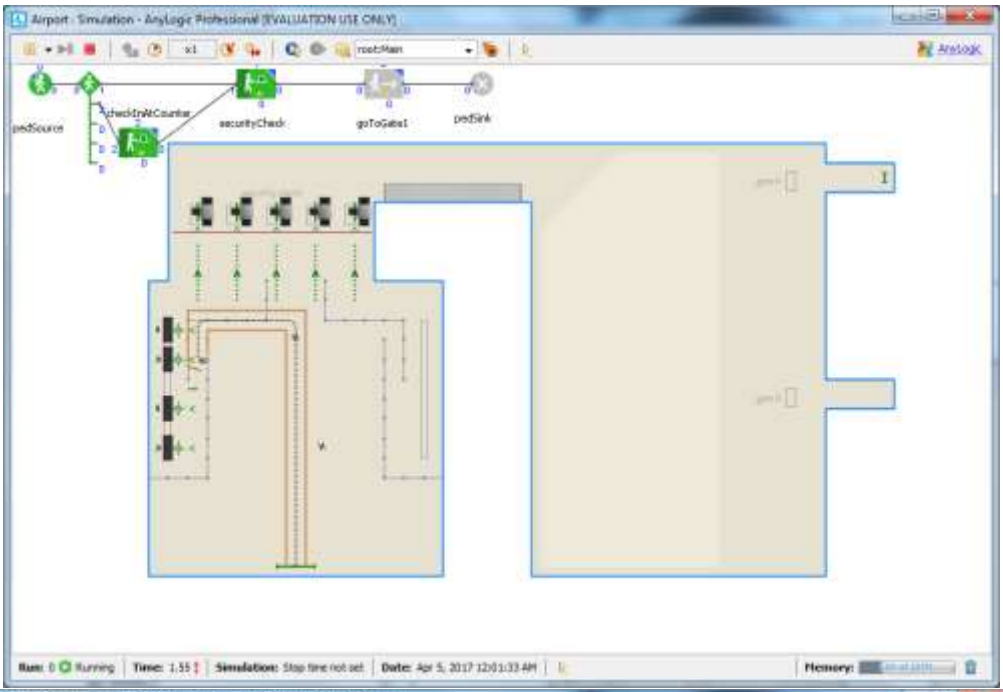


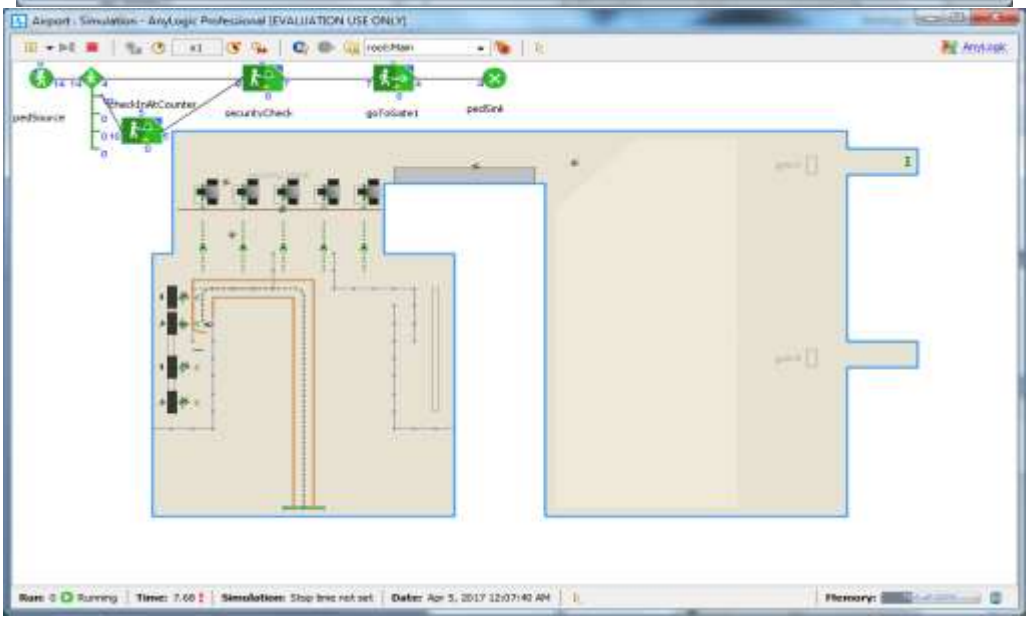
14. Since the tables aren't facing the correct direction, use their **Properties** section's **Position** area to set **Rotation, Z**: 90 degrees.

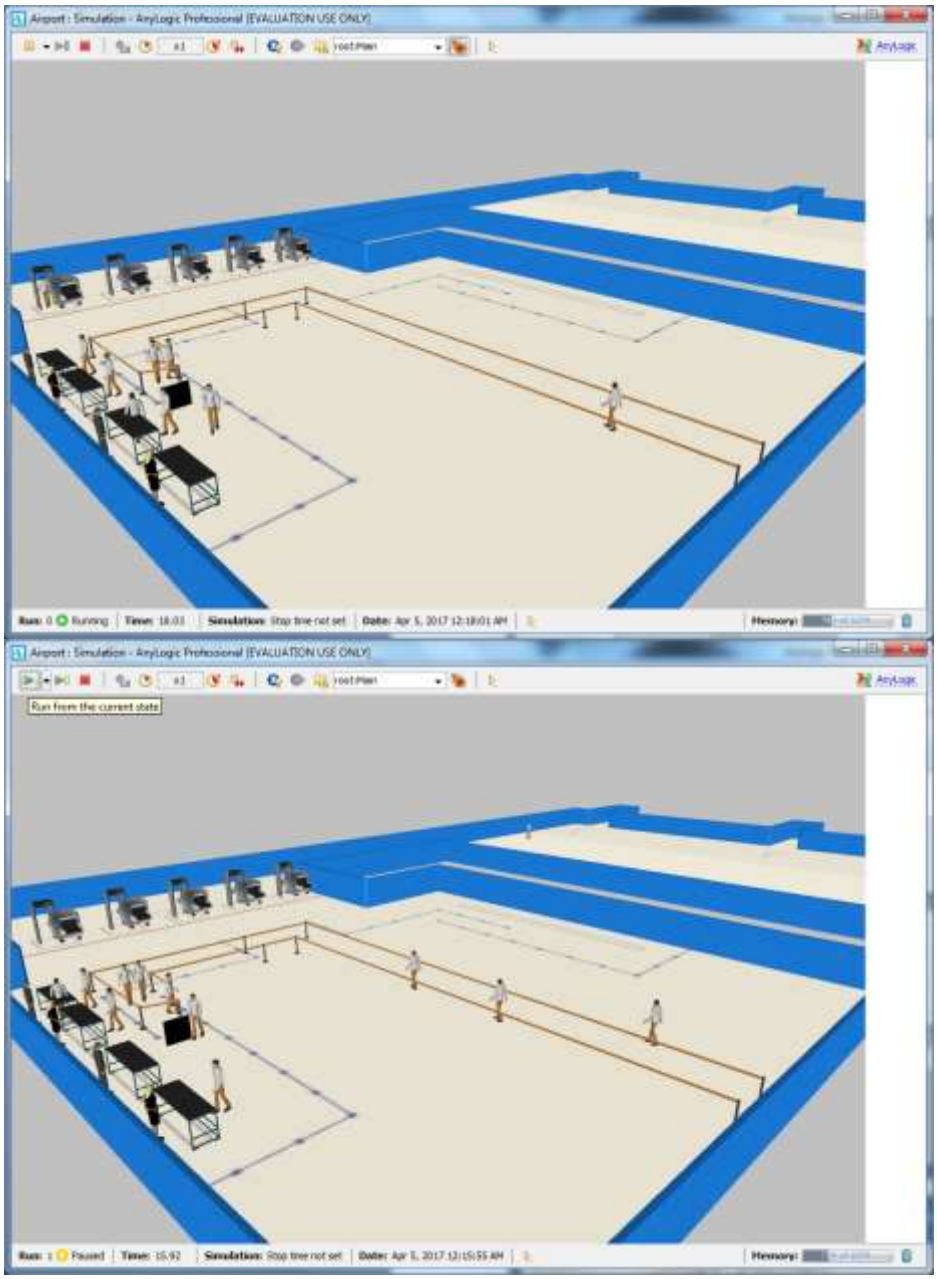




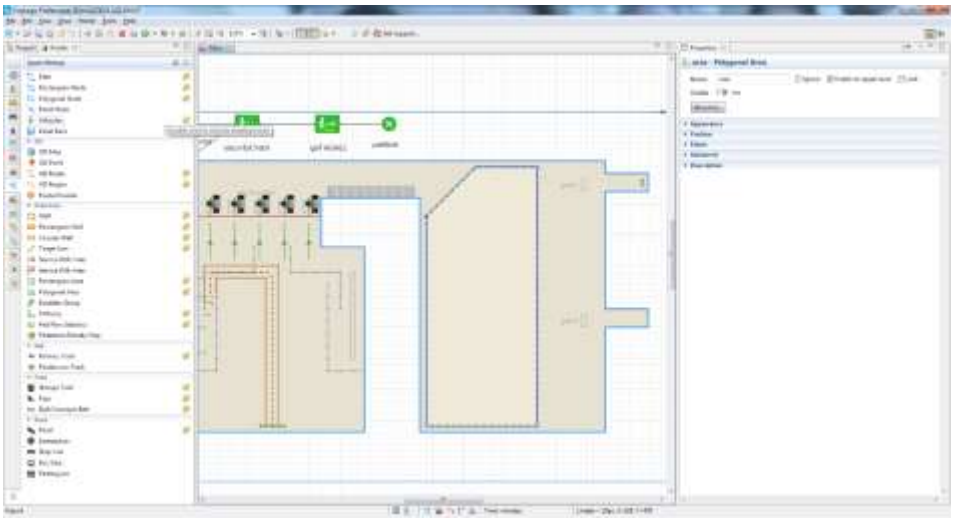
15. Run the model. You'll see some passengers check in and then go through the metal detector.



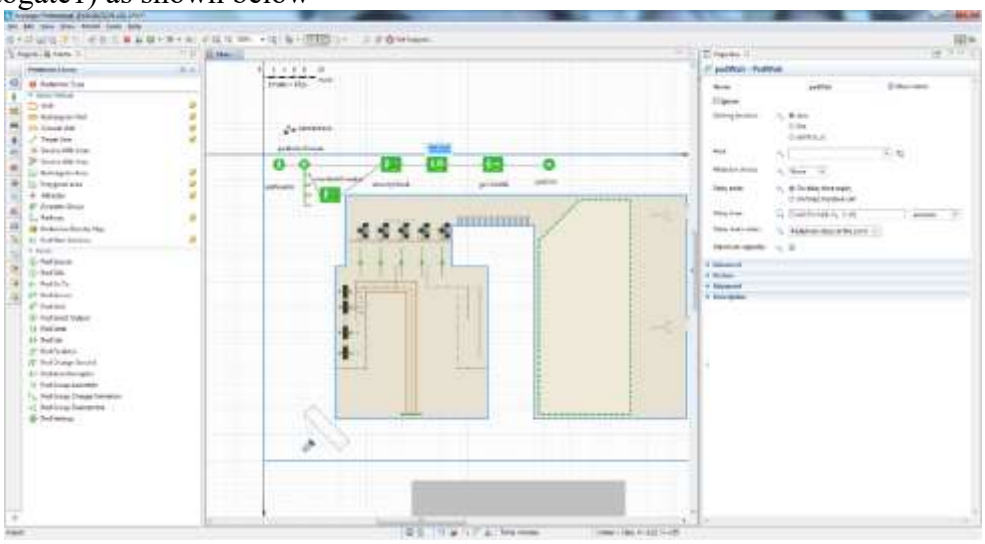




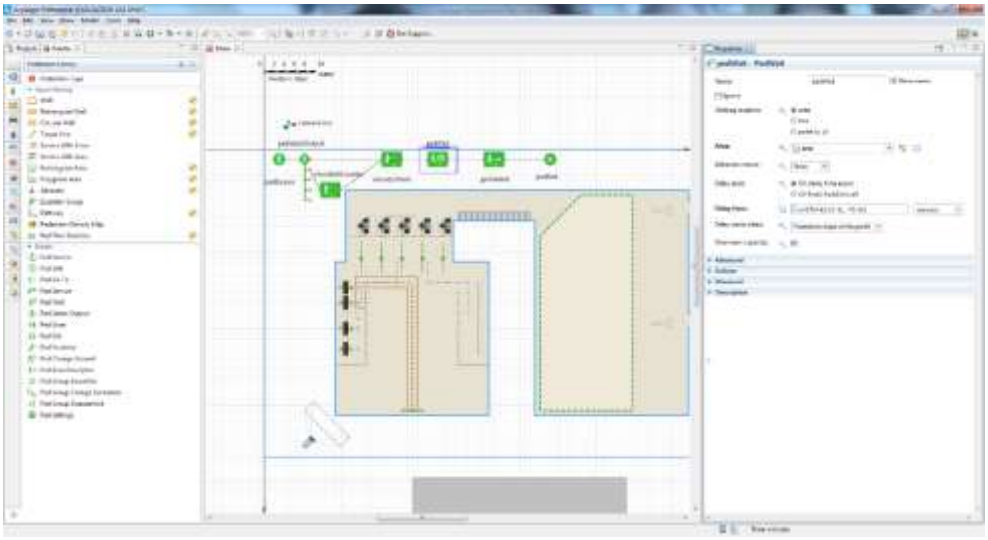
16. We want the passengers to wait before they go to the gate. To do this, we need to draw the waiting area where the passengers will wait and then add a flowchart block (**PedWait**) to simulate the waiting. Draw the waiting area before the gates using the **Polygonal Area** element from the **Space Markup** section of the **Pedestrian Library** palette. Switch to the drawing mode and draw the area as shown in the figure below by clicking your mouse each time you want to add a point. When you're finished, doubleclick your mouse.



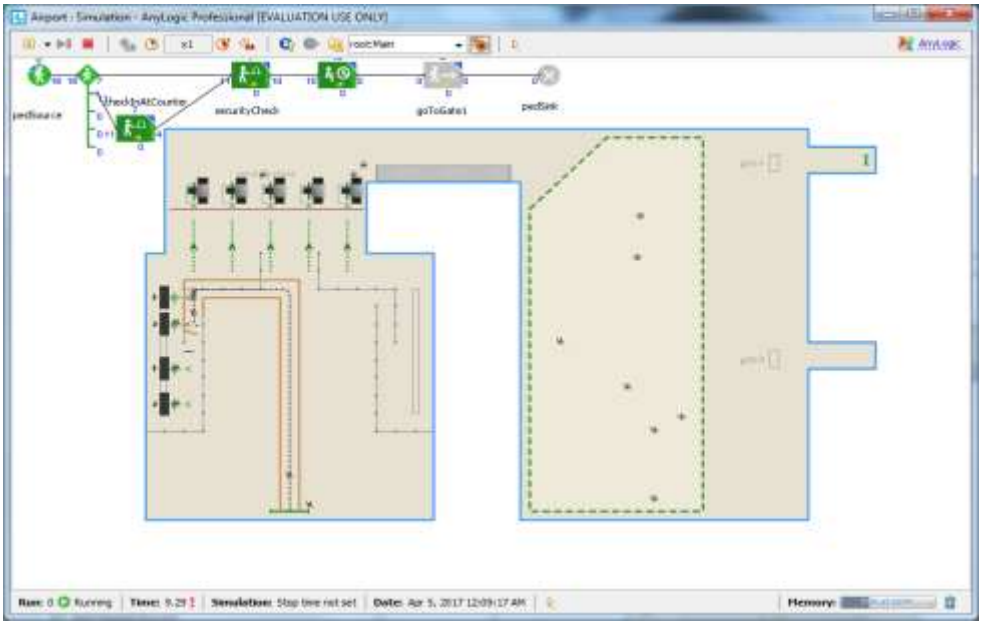
17. Add the **PedWait** block on to the flowchart between **PedService (securitycheck)** and **PedGoTo**.(gotogate1) as shown below

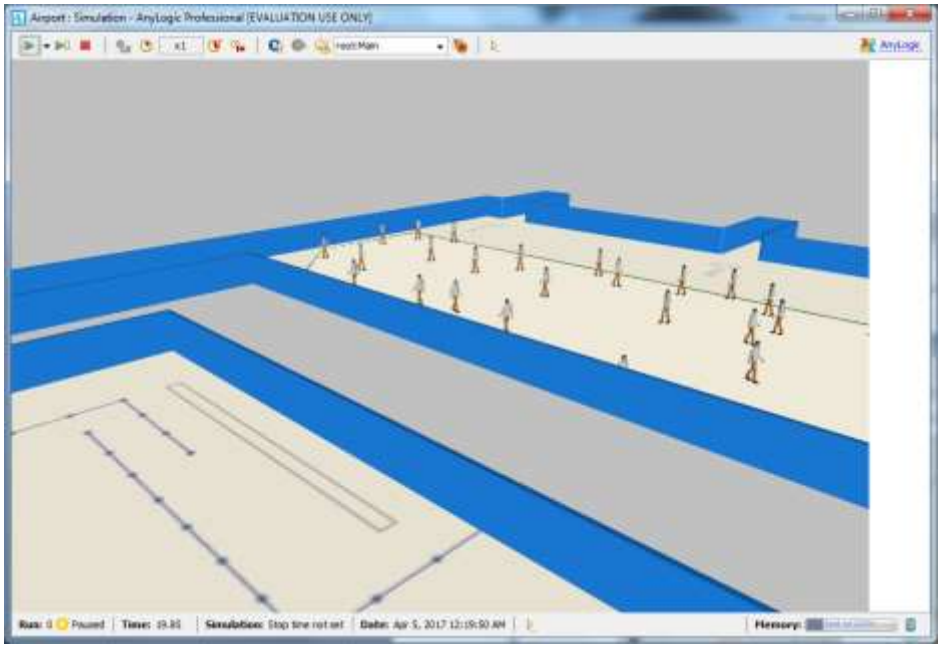


18. Modify the block's properties by selecting the area from the **Area** list, and then setting the **Delay time** to uniform(15, 45) **minutes**.



19. Run the model again, and you'll see the passengers now wait in the waiting area before they proceed to the gate.

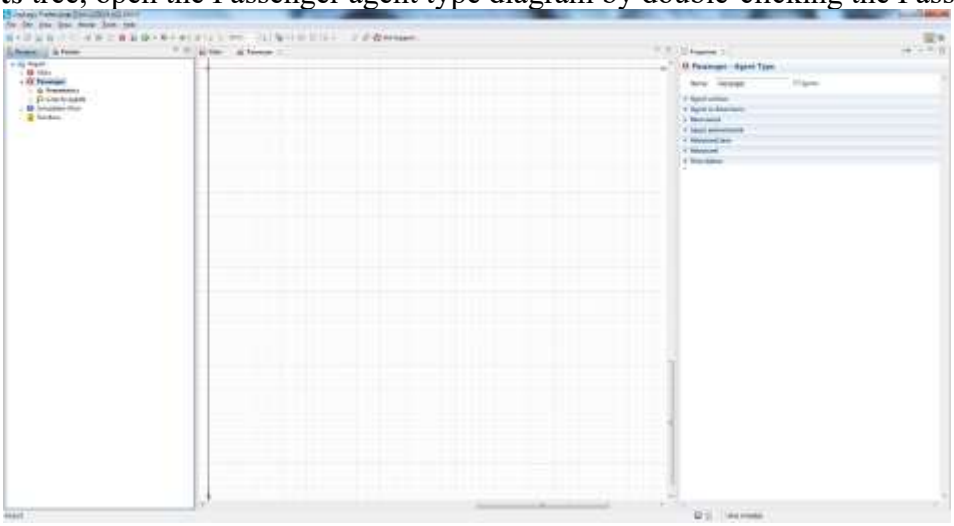




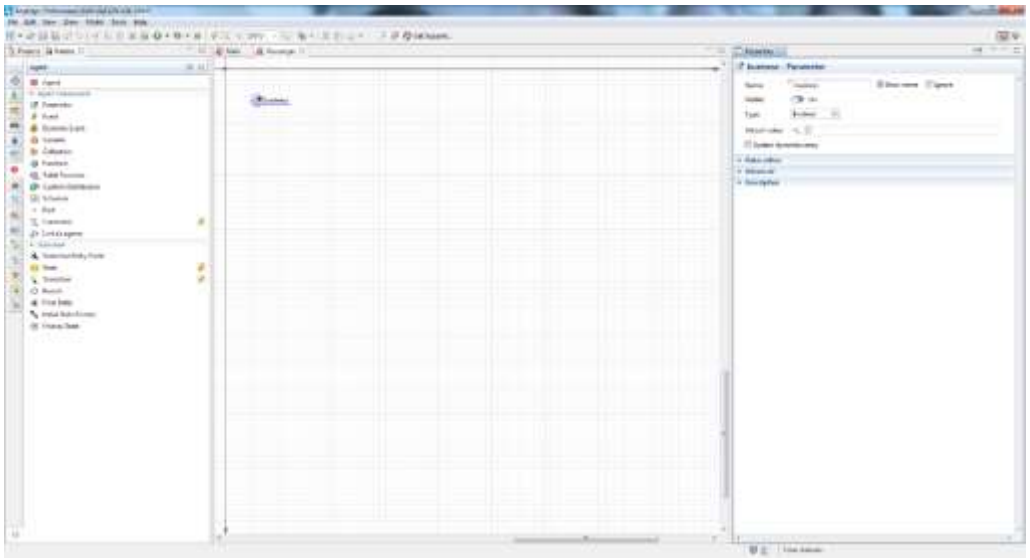
20. You can add more check-in facilities on the right and configure the **PedSelectOutput** to separate the pedestrian flow to more branches.

- **Defining the boarding logic**

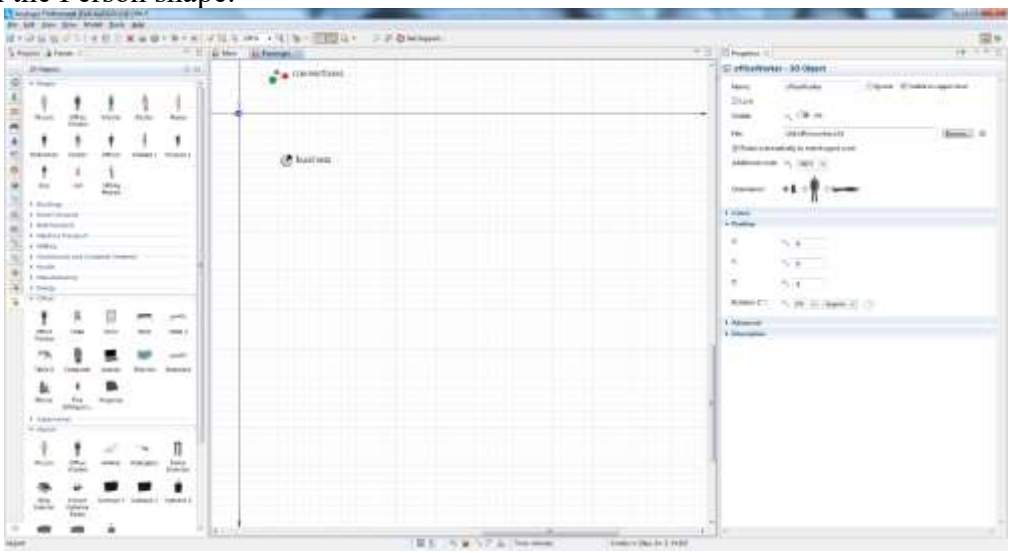
1. In the **Projects** tree, open the Passenger agent type diagram by double-clicking the Passenger item.



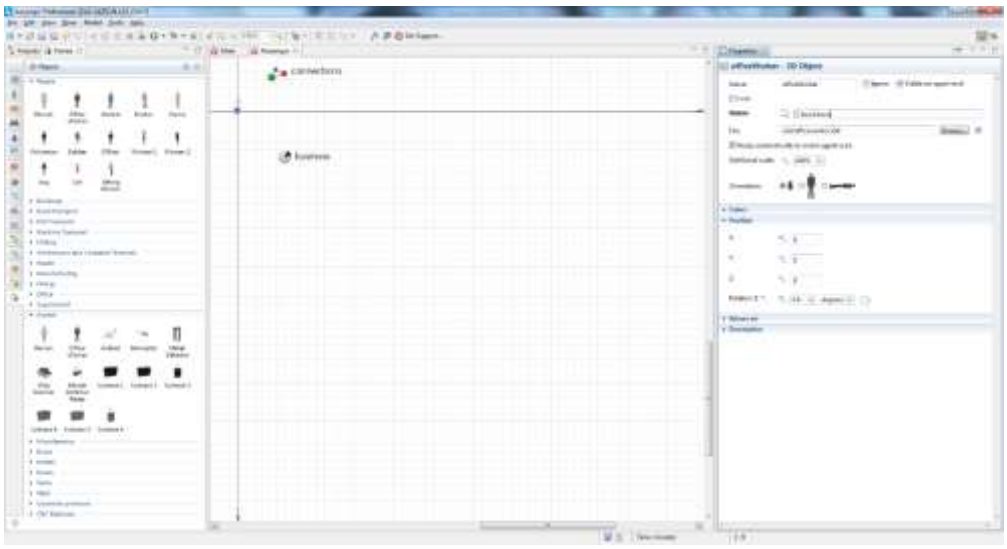
2. Add a **Parameter** from the **Agent** palette to define the passenger's class. Name it business, and set **Type**: Boolean. If the parameter is equal to true, this is a business class passenger, otherwise (if the parameter is false), this is an economy class passenger. We want to distinguish passengers in 3D animation, namely animate business class and economy passengers with different 3D models. To do this, we'll use the existing Person 3D object to represent economy passengers and add another 3D shape to represent business class passengers. Add the 3D object **Office worker** to animate a business class passenger and
then place the figure on the axis origin point (0,0), right on the Person shape.



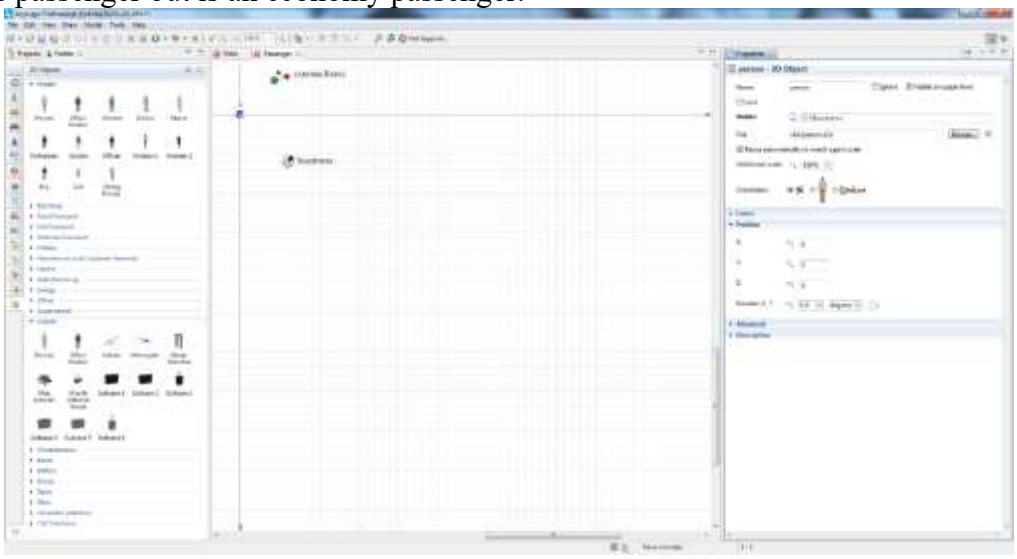
3. We can distinguish passengers in 3D animation, namely animate business class and economy passengers with different 3D models. To do this, we'll use the existing Person 3D object to represent economy passengers and add another 3D shape to represent business class passengers. Add the 3D object **Office worker** to animate a business class passenger and then place the figure on the axis origin point (0,0), right on the Person shape.



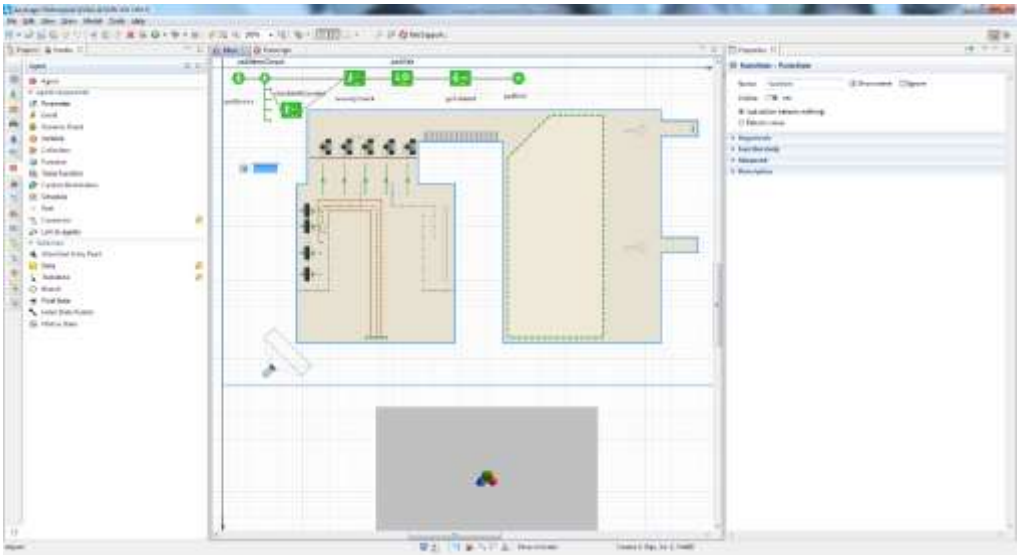
4. Change the visibility of these objects. First, click the Office worker shape. We want this shape to be visible only when this is a business class passenger, that is, when its business parameter is true. Switch to the **Visible** property's dynamic value editor by clicking the icon to the right of the **Visible** label, and then type business in the box. By doing this, we're making this 3D shape visible only when pedestrian's business parameter is true.



5. Now select the person 3D object (you can do this from **Projects** tree), and set **Visible**: ! business . This shape will be visible only if the passenger is an economy passenger. The symbol ! is the boolean operand NOT. The expression !business returns true when the business is NOT true – when the passenger is not a business class passenger but is an economy passenger.



6. We want to set up the passengers' class when they arrive to the airport terminal. Return to the Main diagram and add a **Function** from the **Agent** palette. Name it setupPassenger.

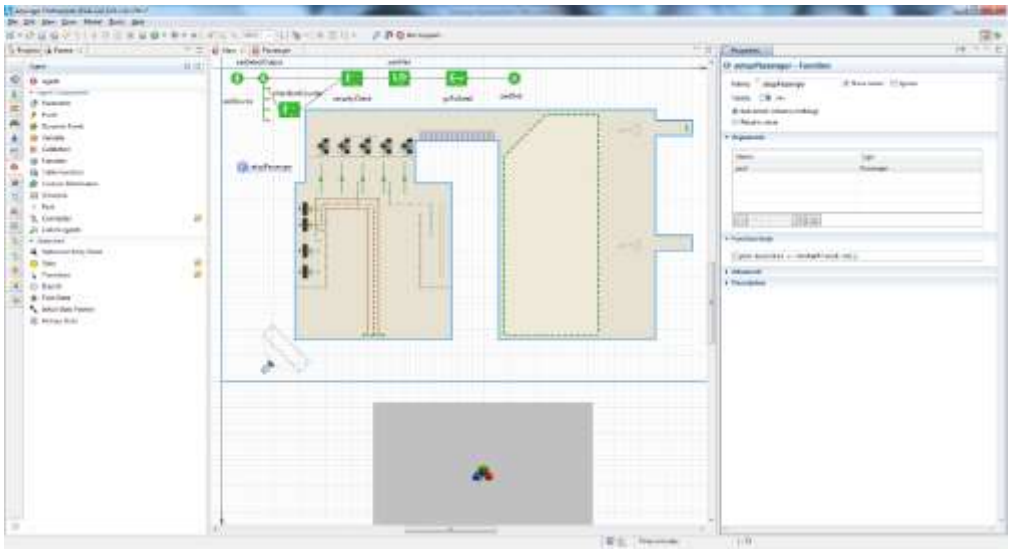


7. Configure the function as follows:

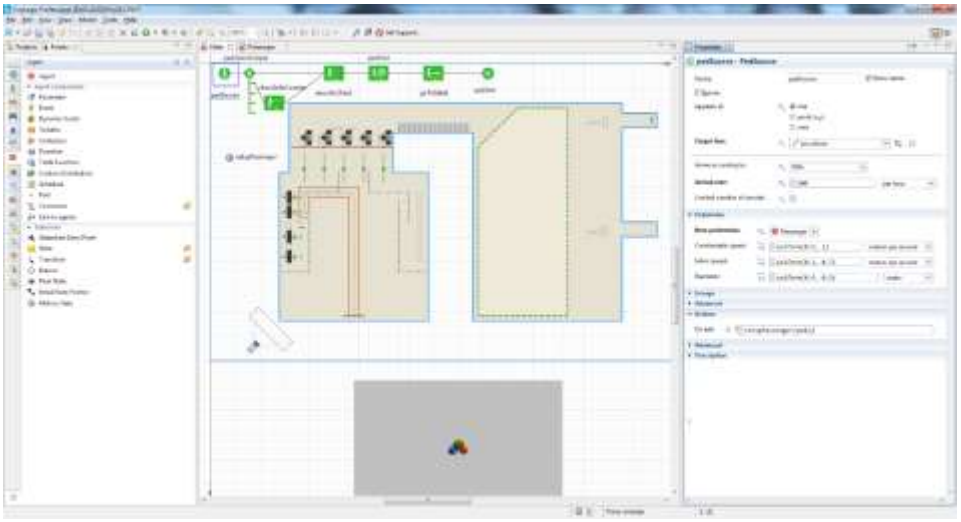
• Create one argument in the agument drop down list to pass the newly-created passenger to the function:

- **Name**: ped
- **Type**: Passenger

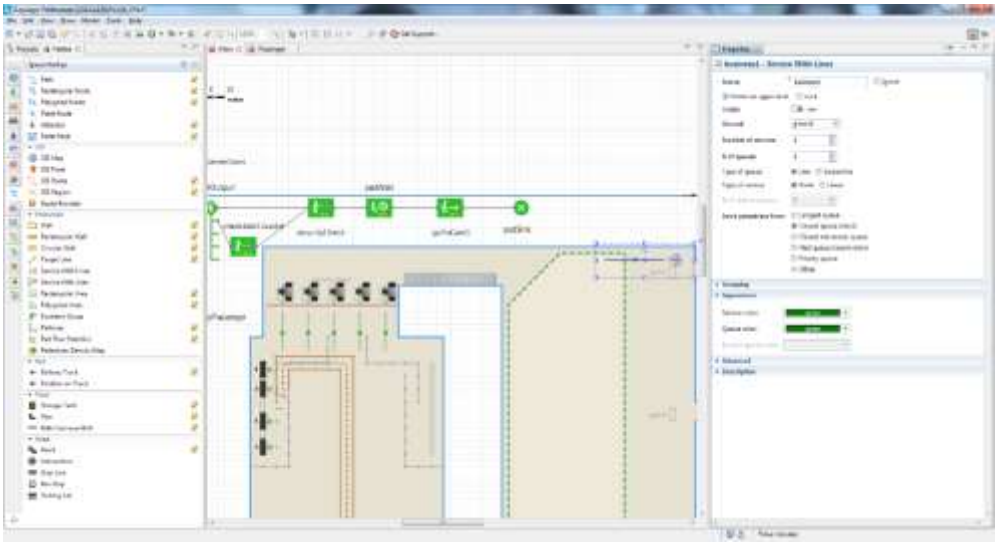
• The function's code defines the frequency with which the business class passengers appear in the model: in the function body list write this ped.business = randomTrue(0.15);



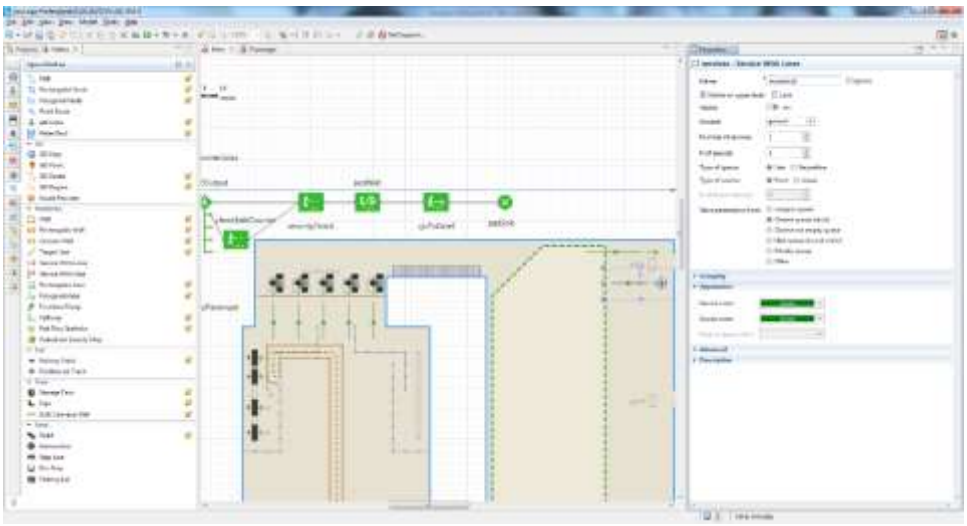
8. In this case, ped is the function's argument, of type Passenger. Having set up the Passenger as the argument type, we can directly access the custom pedestrian field business simply as ped.business. The function randomTrue(0.15) returns true in an average of 15 percent of cases, which means an average of 15 percent of our model's passengers will travel in business class. Call this function when a new pedestrian is created by the pedSource block. In the pedSource properties area, click the arrow to expand the **Actions** section, and then enter the following code in the **On exit** box: setupPassenger(ped);



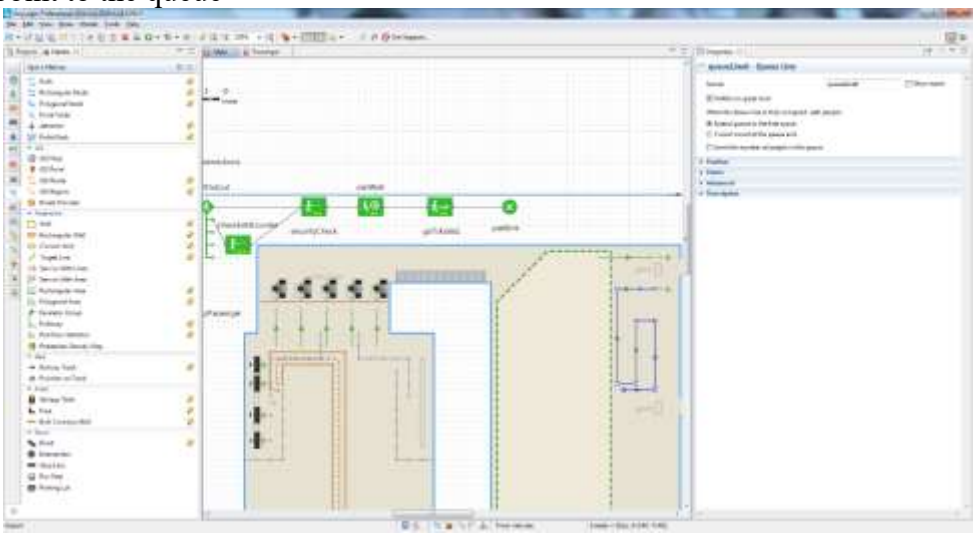
9. Here we call our function setupPassenger for the newly-created pedestrian. This pedestrian is passed to the function via its argument. Draw two services with lines for the upper gate, one for business class and one for economy passengers. Draw a Service with Line, defining the priority line (point service, 1 service, 1 line). Name this service business1



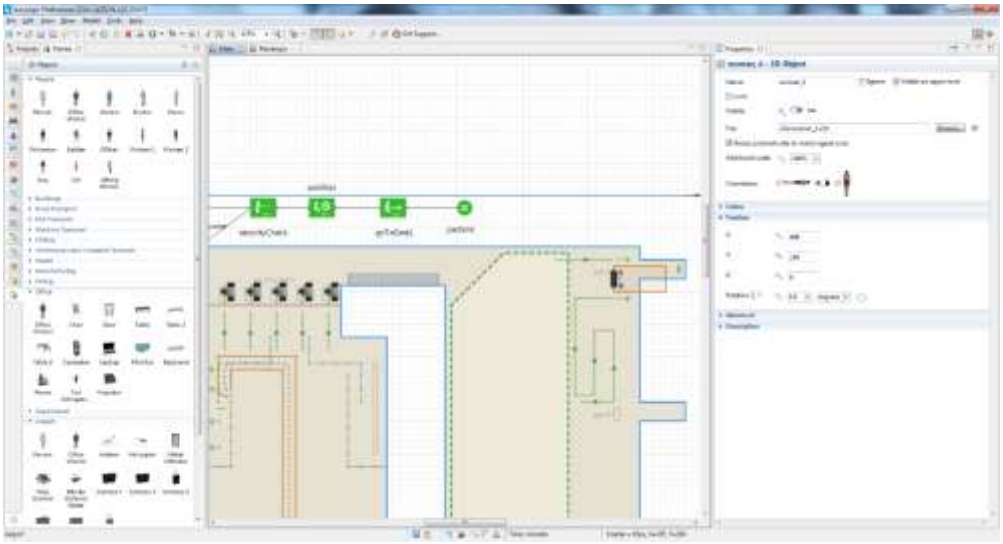
10. Add one more Service with Line, and name it economy1.



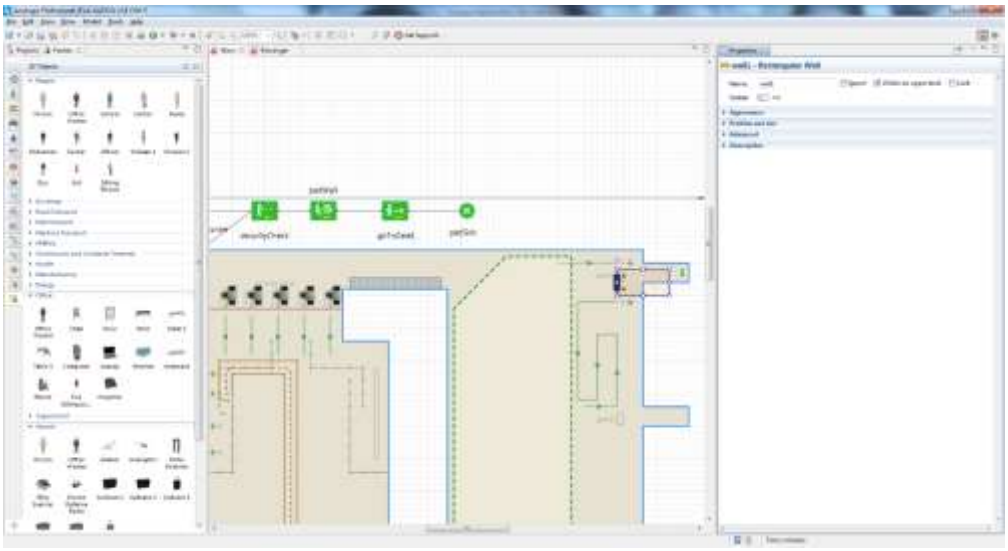
11. Add more point to the queue



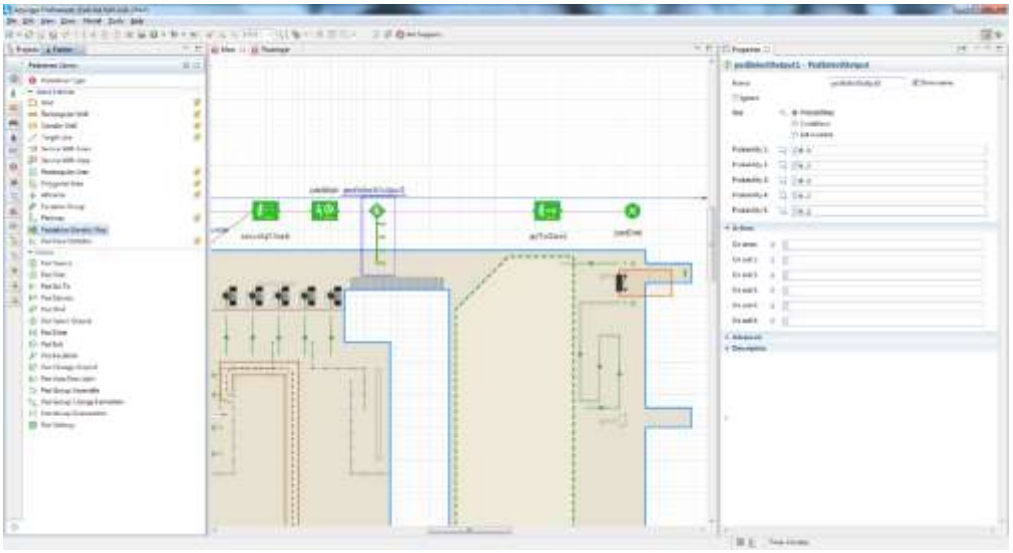
12. Draw an area at the gate with the **Rectangular Wall** element, from the apce markup and then add a table from 3d office & rotate it 90 degree as per display and add two 3D woman models at the table.



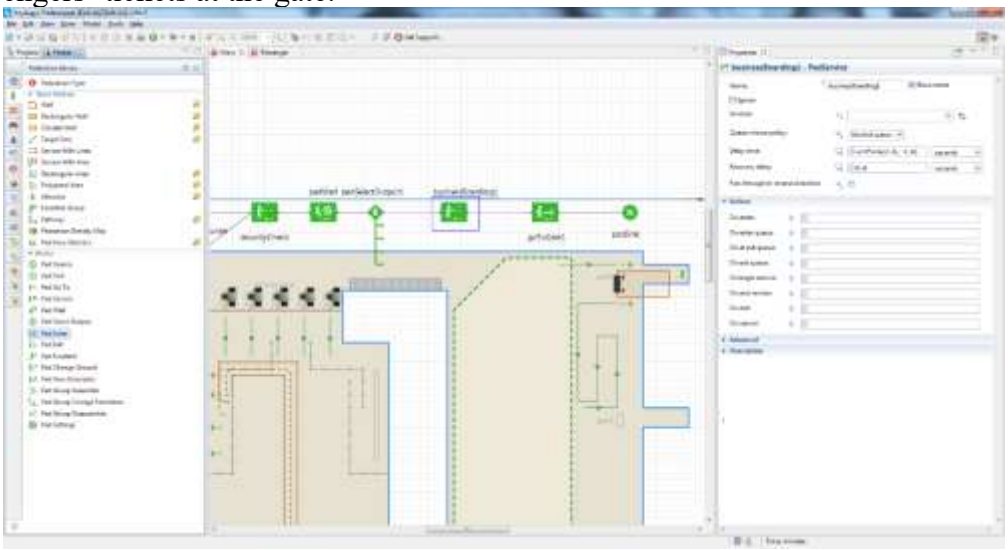
13. In the wall's properties, set **Visible** to **no** to make this wall invisible at the model runtime.

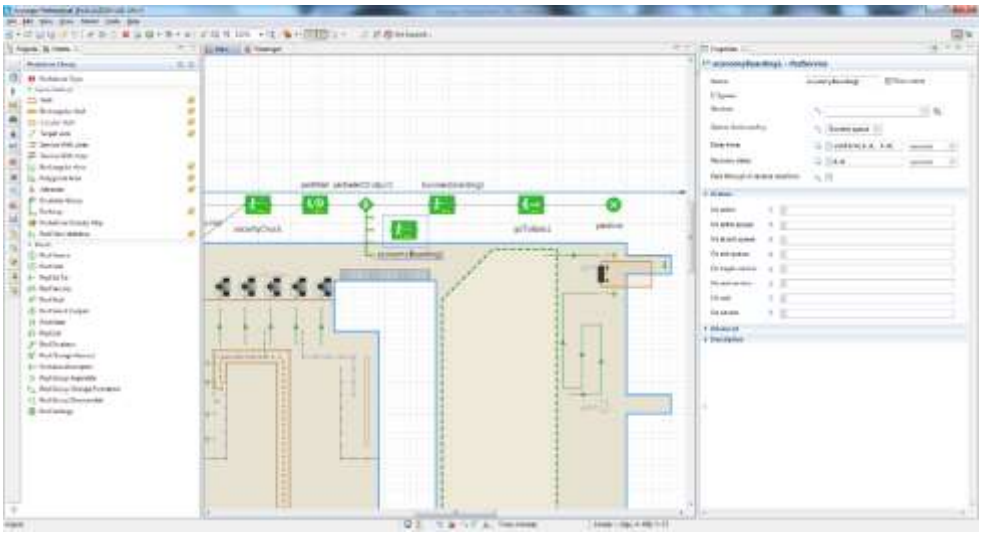


14. Insert the blocks into the flowchart between the pedWait and goToGate1 objects. Add **PedSelectOutput** to route business class and economy passengers to different lines. As shown below:

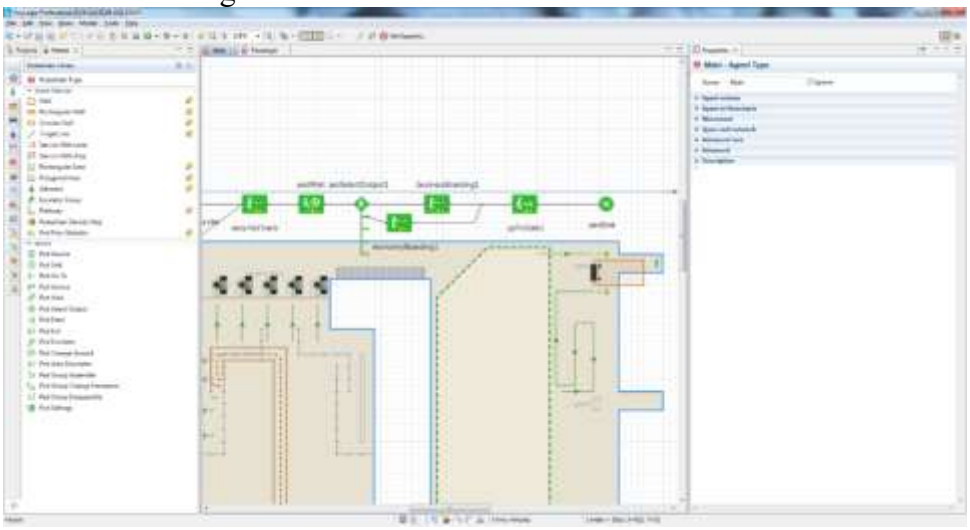


15. Add two **PedService** blocks: businessBoarding1 and economyBoarding1 to simulate the process of checking passengers' tickets at the gate.

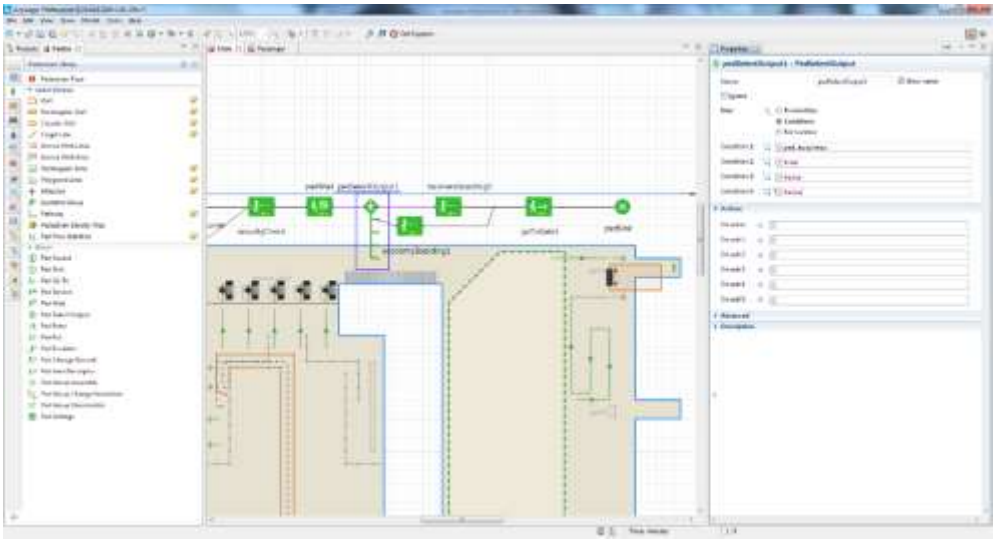




16. Connect it as shown in the figure below

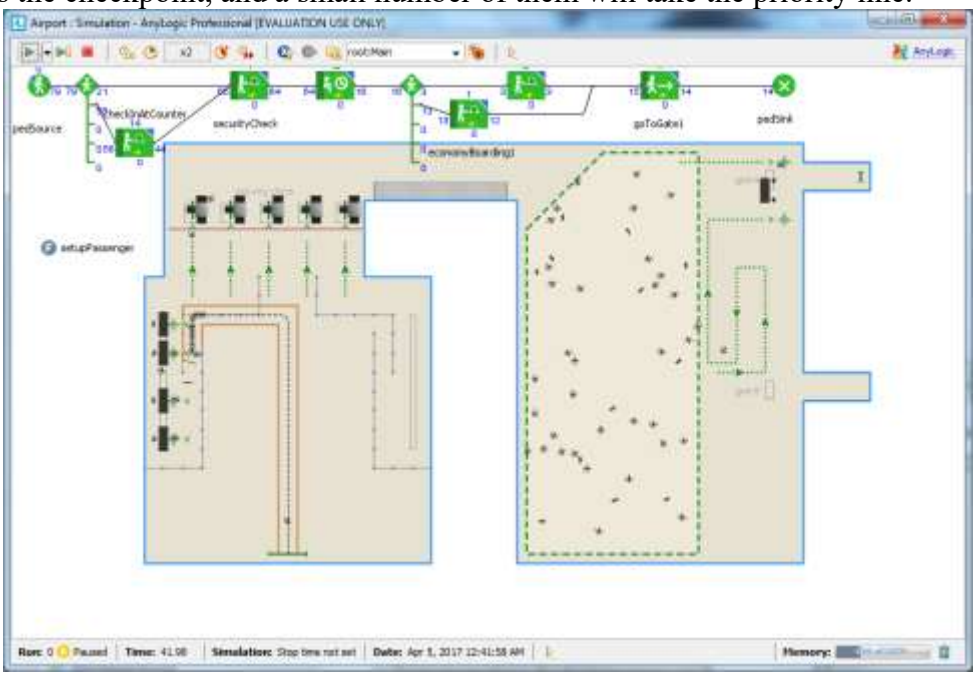


17. Since the **PedSelectOutput** block routes business class and economy passengers to different lines, select **Use**: **Conditions**, and then type ped.business in the **Condition 1** box. This expression will return true for all business class passengers, which means they will follow the upper flowchart branch and join the priority line. After you set up the conditions for the block's next output ports (true, false, false), the model will direct all other passengers to the second output port.



18. For the **PedService** block businessBoarding1, choose business1 as **Services**. Since it takes between two to five seconds to check a passenger's ticket, you can slightly change the **Delay time**. For

economyBoarding1, set **Services**: economy1, adjust the **Delay time**. Run the model. You'll see passengers pass the checkpoint, and a small number of them will take the priority line.



Practical 7

Aim : Verify and validate a model developed like bank model or manufacturing model.

Bank Office Model Scenario :
We will create a simple service system of a bank department, consisting of an automatic teller machine and teller lines. ATM provides people with a quick self-service for cash. More complex transactions, e.g. paying bills, are completed by tellers, allowing customers more time without inconveniencing those customers looking for quick cash.
Here we are going to create the simplest queuing system, consisting of a source of agents, queue, delay and final sink object.

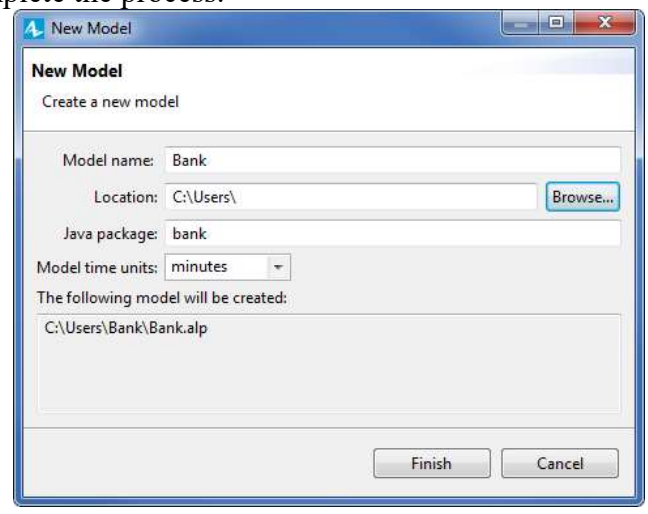
- **Source** block generates agents. It is usually used as a starting point of the process flow. In our example, it models customer arrival.
- **Queue** block models queues. In this model it simulates a queue of customers waiting for the moment they can start accessing ATM services.
- **Delay** here simulates the delay associated with the service at ATM.
- **Sink** block indicates the end of the flowchart and discards the incoming agents.

Functions used :

- triangular() function - is the standard AnyLogic random number generator. AnyLogic provides also other random number distributions, like normal, Poisson, exponential, etc.
- *3D window* - plays the role of a placeholder for 3D animation. It defines the area on the presentation diagram where 3D animation will be shown at runtime.
- ATM.statsUtilization.mean() -ATM is the name of the **Delay** object we created. Each **Delay** object has statsUtilization data set that collects statistics on the object utilization. The mean() is the function that returns the mean value measured. You can use other functions to get statistical values, such as min() and max().
- queue.statsSize.mean() - statsSize is the data set of type **StatisticsContinuous** that collects the statistics on the **Queue** size.
- Histogram data objects to store statistics on customer's waiting time and time in system. Histogram data objects support standard statistical analysis on the data values being added (calculate mean, minimum, maximum, deviation, variance and mean confidence interval).

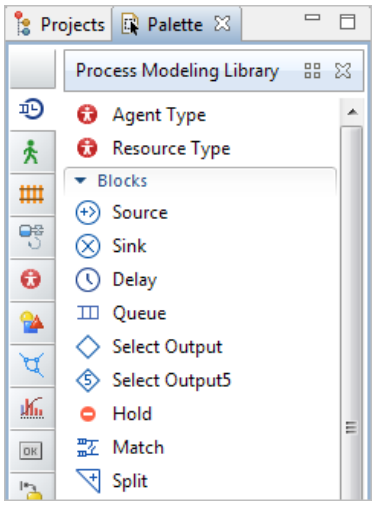
Phase 1. Creating a Simple Model
First, we will create the simplest queueing model simulating how customers are serviced at the ATM.

- Create a new model by selecting **File > New > Model** from the menu, and
- then name it Bank. Select **minutes** as the **Model time units**.
- Click **Finish** to complete the process.

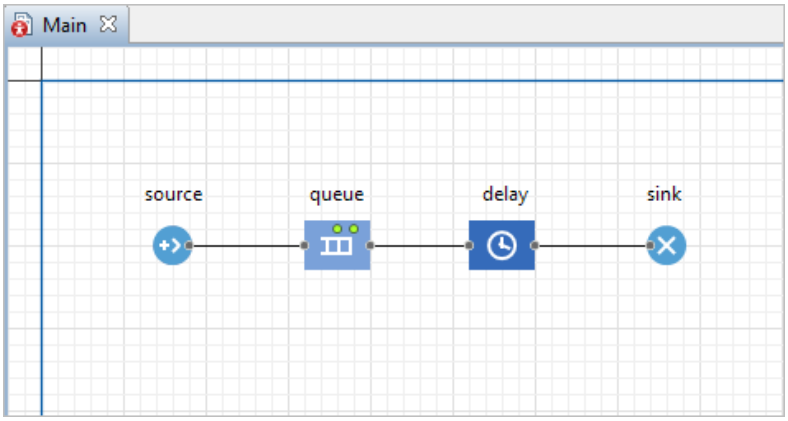


Creating the flowchart defining the process
Create the model flowchart

1. The **Palette** view,display the **Process Modeling Library** palette: Drag source, Sink, Delay and Queue from Blocks



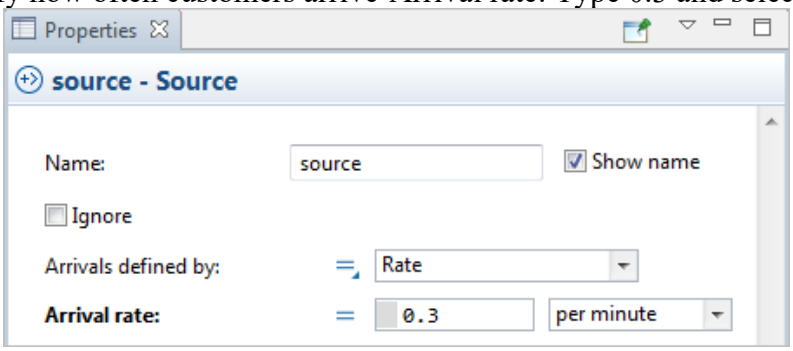
2. Connect them with connectors.



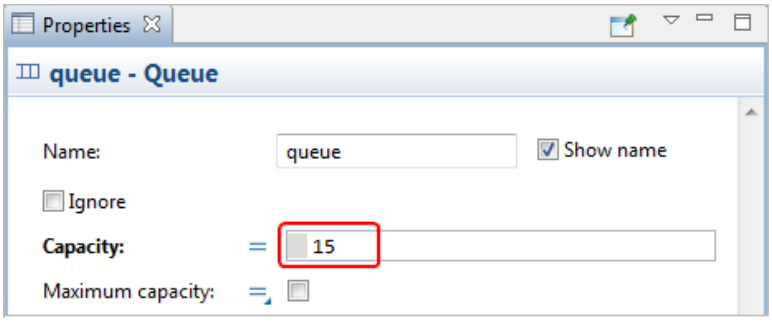
The given flowchart models the simplest queuing system, consisting of a source of agents, delay (and a queue before this delay) and final sink object.

Configure the flowchart blocks

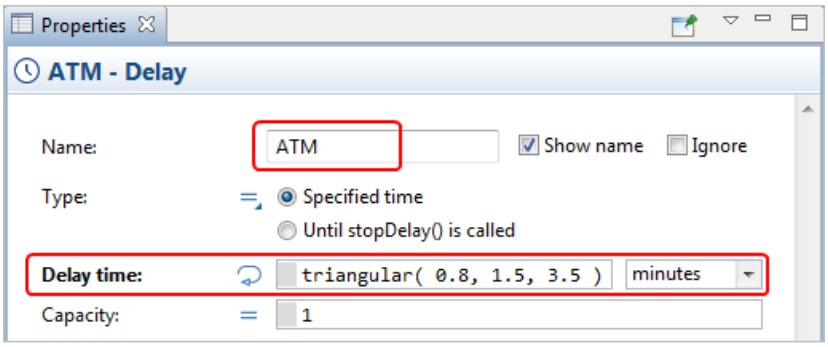
1. Select *source* block. In the **Properties** view,
   - Name: source
   - Specify how often customers arrive-Arrival rate: Type 0.3 and select *per minute*.



2. Modify the properties of the *queue*:
   - Name : queue
   - Capacity : 15 agents. At most 15 customers will wait in a queue.

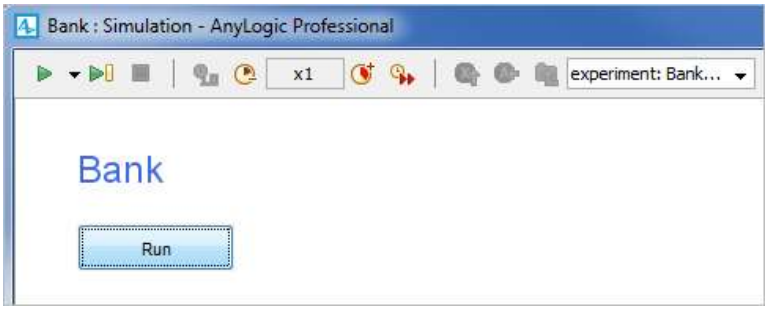


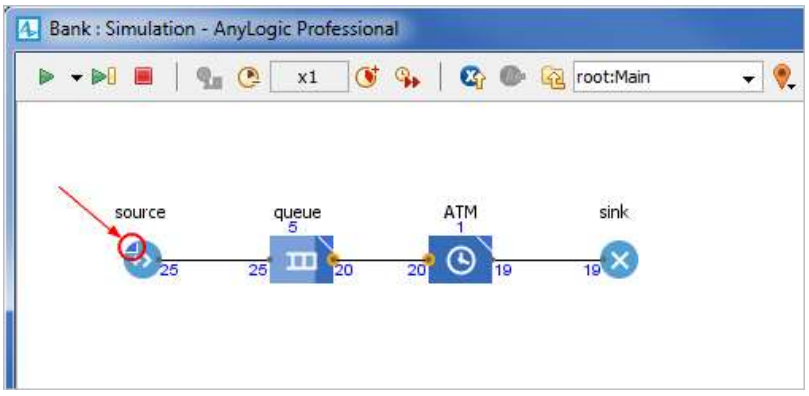
3. Modify the properties of the *delay* :
   - Name : *ATM*. Specify the processing time.
   - Delay Time : triangular(0.8, 1.5, 3.5) and select minutes.[Assume that processing time is triangularly distributed with mean value of *1.5*, min of *0.8* and max value of *3.5 minutes*.]
   - Capacity : 1



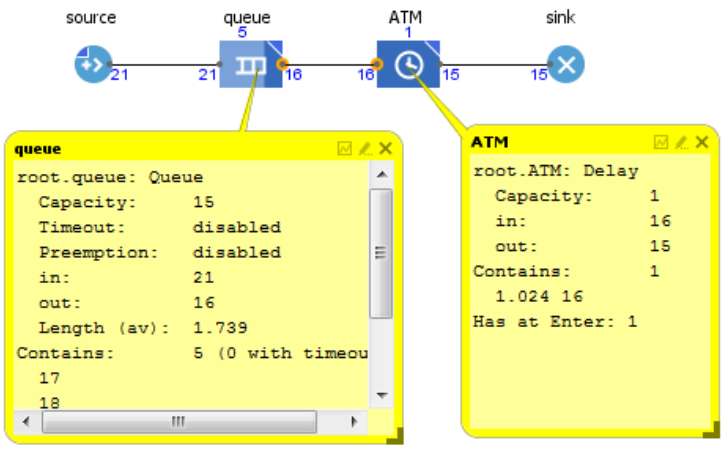
Running the model

- Build
- Run





you can see detailed current object status, for example queue size, number of agents left and so on – completely in graphics. You can inspect flowchart objects to get the detailed information on their current state. Click on the object to open its inspect window. Inspect window show statistics on the object, e.g. **Queue** object's inspect shows the queue capacity, the number of agents passed through either port of the object and also whether the timeout option is enabled for this queue. *Contains* string displays the number of agents currently being in the object along with IDs of these agents.



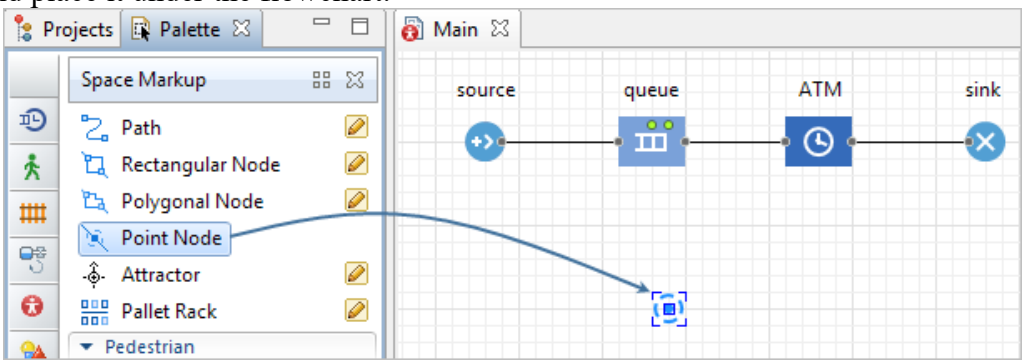
Phase 2. Creating Model Animation

Here we will draw the layout consisting of the ATM and a queue. Then we will animate clients standing in the queue and using the ATM. We also want to visualize the current status of the ATM. [If you have existing image of the layout, you can simply import this picture as the bank layout instead of drawing it by yourself.]1

1. **Adding space markup shapes**

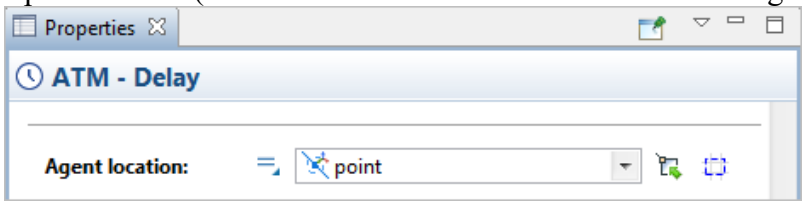
Set up space markup for the ATM

1. Draw the ATM as a point node.
   - Open the **Space Markup** palette in the **Palette** view.
   - Drag the element **Point node** from the **Space Markup** palette into the graphical editor and place it under the flowchart.



2. Select the point node in the graphical editor to open its **Properties** panel. Enter the run-time color expression for the shape in the **Color** property: ATM.size()> 0 ? red : green

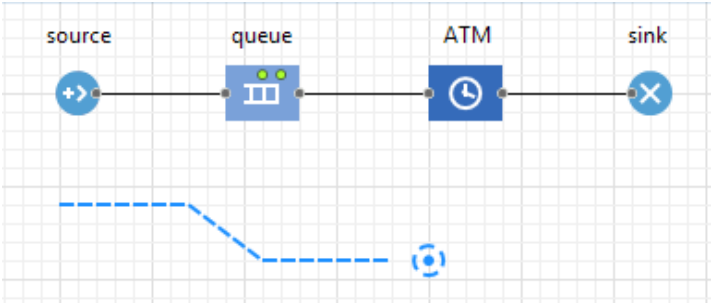
   Note that ATM here is the name of the **Delay** object we created. The expression determines the point node color at run time. The size() function returns the number of agents currently being processed. The color will be red, if a customer is served at this time, and green otherwise.
3. Click the *delay* block, called ATM, to open its **Properties** view:
   - Select the *point* node we have drawn in the **Agent location** option.
     [You can either click the down arrow and select the point node from the list of appropriate objects, or you can click the button, located on the right, to select this space markup shape from the graphical editor (the rest of the elements in the editor will be greyed out).]



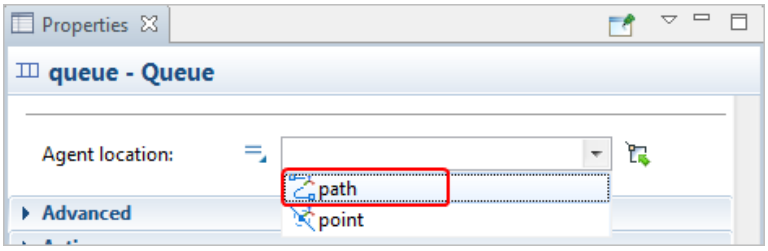
Set up space markup for the queue

1. Draw the queue as a path.

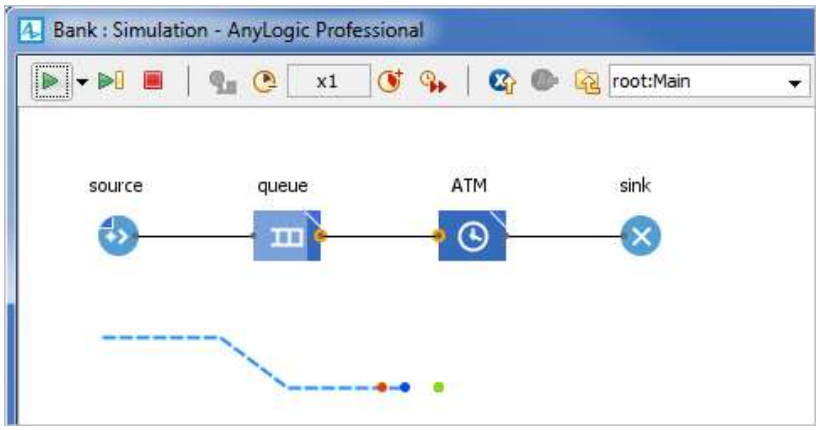
- Open the **Space Markup** palette in the **Palette** view.
- Double-click the element **Path** to switch to the drawing mode.
- Click in the graphical editor to put the first point of the path. Do more clicks to add turning points. Finish drawing with a double-click.
-



2. Click the *queue* block in the flowchart and go to its **Properties** view:
   - Select the *path* we have drawn in the **Agent location** option. You can either click the down arrow and select the path from the list of appropriate objects, or you can click the button, located on the right, to select this space markup shape from the graphical editor (the rest of the elements in the editor will be greyed out).

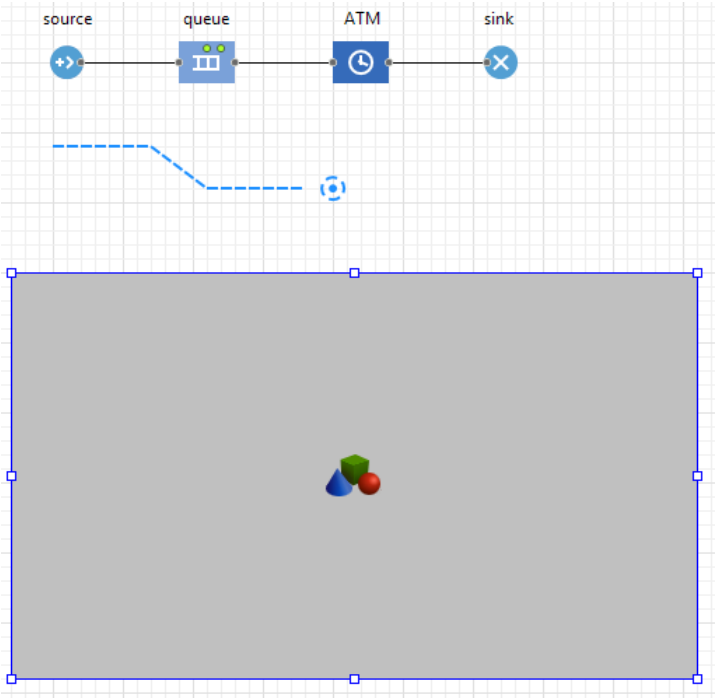


3. Run the Model and observe its behavior.



Adding 3D animation
Add 3D window

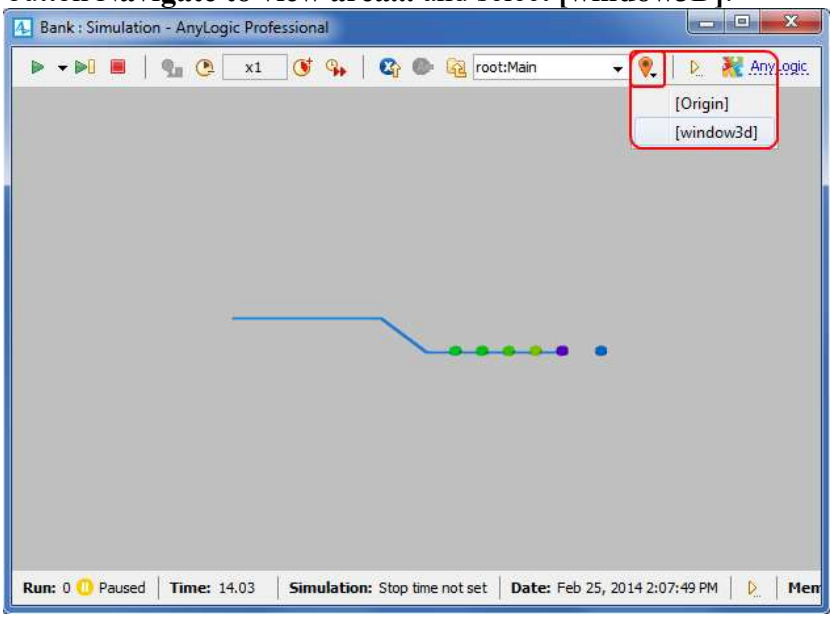
1. Drag the **3D Window** element from the **3D** section of the **Presentation** palette to the graphical editor.
2. The grey area will appear on the screen. Locate it where you want your 3D presentation to be shown at the model runtime:



Navigating through 3D animation

Run your model and observe simple 3D animation.

1. Click the toolbar button **Navigate to view area...** and select **[window3D]**.



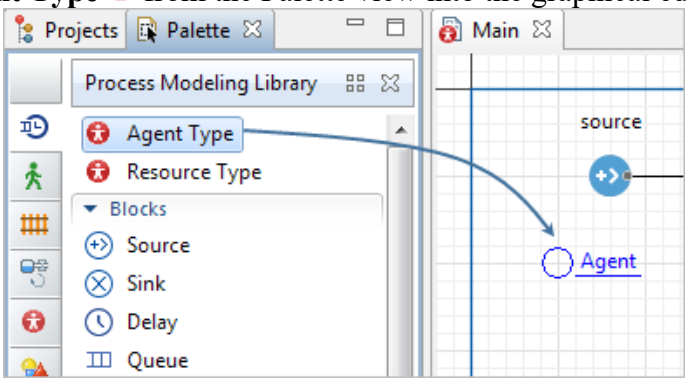
2. Navigate through the 3D scene using the commands described below:

In order to	**Use the mouse like described here**
Move the scene	1. Press the left mouse button in the 3D view and hold the mouse button pressed. 2. Move the mouse in the required direction.
Rotate the scene	1. Press **Alt** key (Mac OS: Option key) and hold it pressed. 2. Click in the 3D scene window and, while holding Alt and the left mouse button down, 3. Move the mouse in the required rotation direction.
Zoom in/out the scene	Scroll the mouse wheel in the 3D window away from / towards you.

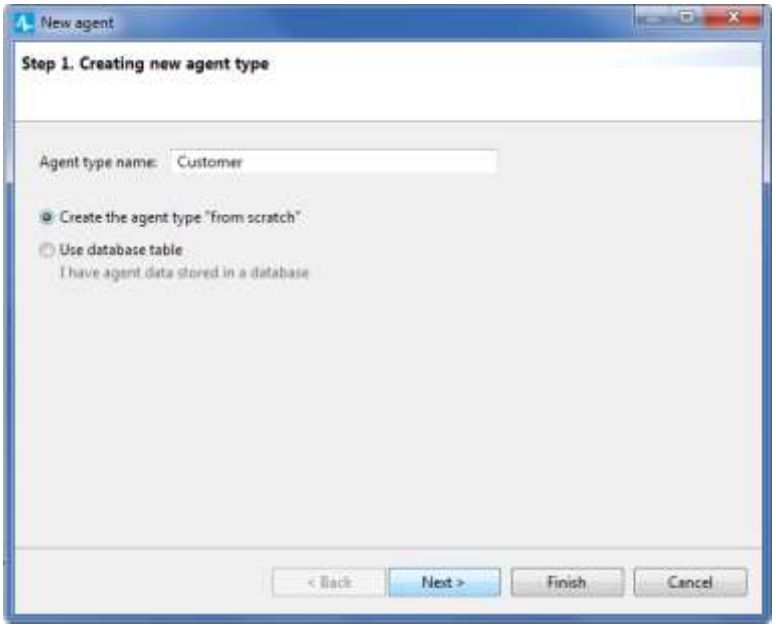
Adding 3D objects

Create a new agent type

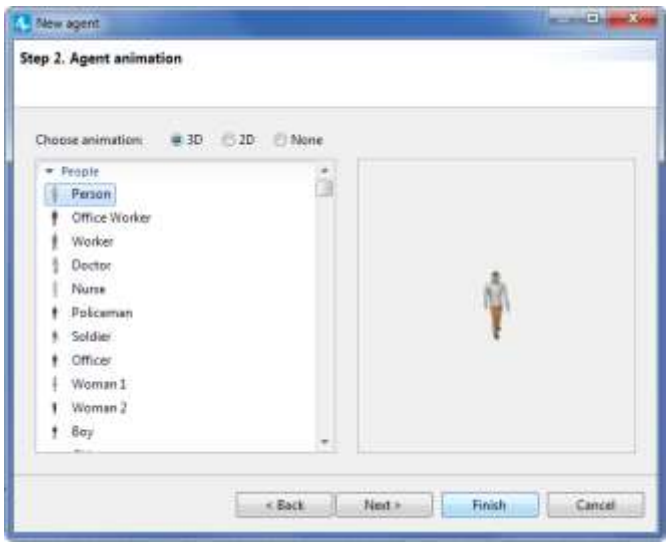
1. Drag the element **Agent Type** from the Palette view into the graphical editor.



2. The **New agent** wizard will open on the **Creating new agent type** step.
   - Enter *Customer* as the **Agent type name**, and
   - leave the **Create the agent type "from scratch"** selected.
   - Press **Next**.



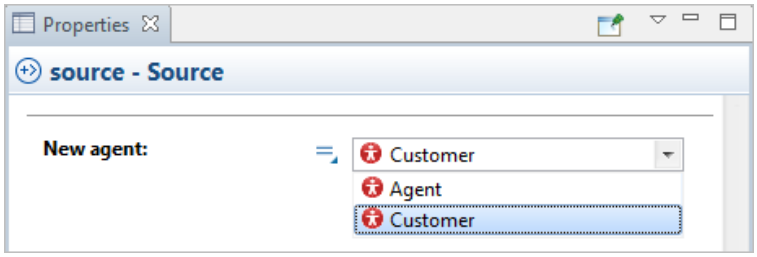
3. In the next step
   - select **3D** as the animation type and
     - select *Person* from the list of the 3D figures. Click **Finish**.



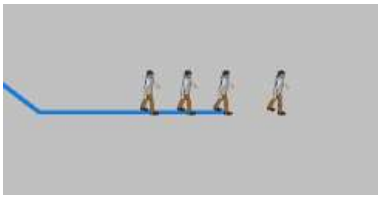
The new Customer diagram will open. You can find the *Person* 3D figure in the axis origin. Switch back to the Main diagram.

Configure flowchart to use the new type

1. On the Main diagram, Select the block *source* go to its properties, Choose Customer in the **New agent** drop-down list.

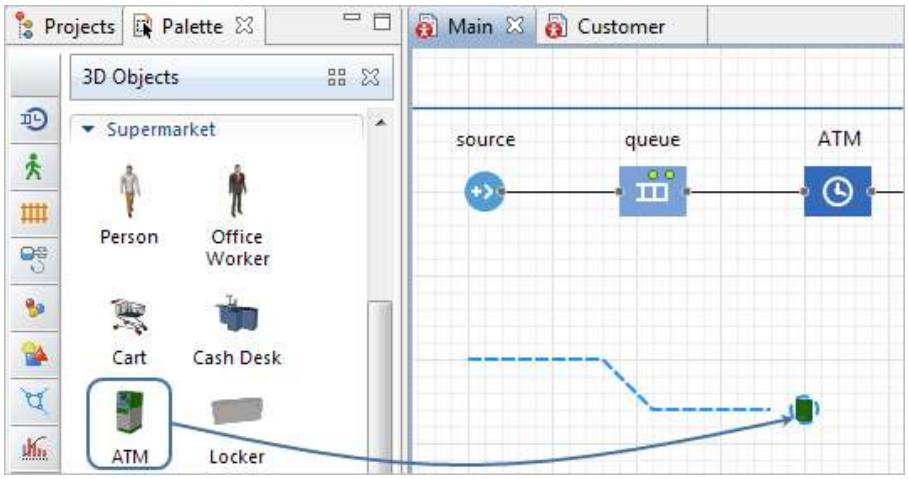


2. Run the model and switch to 3D view to see our customers moving in the queue.

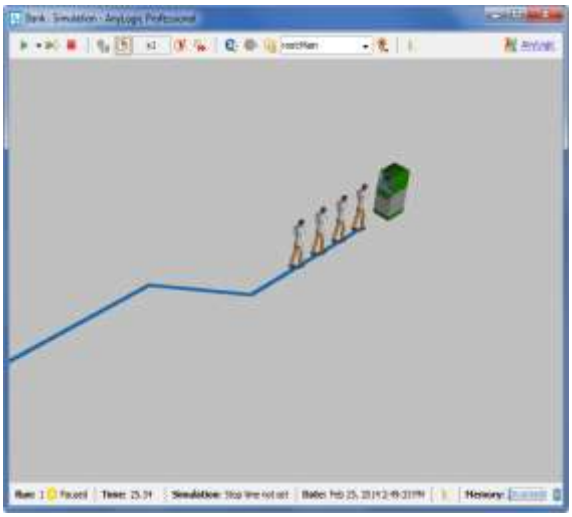


Add an ATM figure

1. Drag the **ATM** 3D figure from the **Supermarket** section from the Palette view onto the point node shape in the graphical editor.



2. If you run the model now and check 3D animation in **window3D** mode, you will notice that our ATM does not face the customers' flow and we need to rotate it.
3. Select the *atm* 3D object in the graphical editor and open the section **Position** in its properties view.
4. Choose *0* degrees from the drop-down list of the **Rotation Z** option.
5. Run the model to double-check that the ATM is facing the customers now.



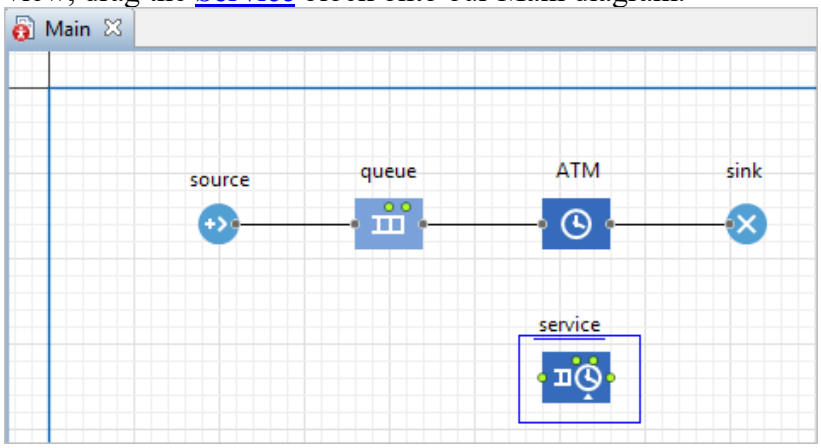
Phase 3. Adding Tellers

Now we will create another part of the system by adding tellers that are working at the bank. Now some clients will come to see tellers, some – to access the ATM. We can model tellers using delays in the same way as we modeled ATM. However, modeling tellers using resources is much more convenient.
Resource is a special unit that can be possessed by an agent. Only one agent can possess a resource at a time; therefore agents compete for resources.

Modifying the flowchart

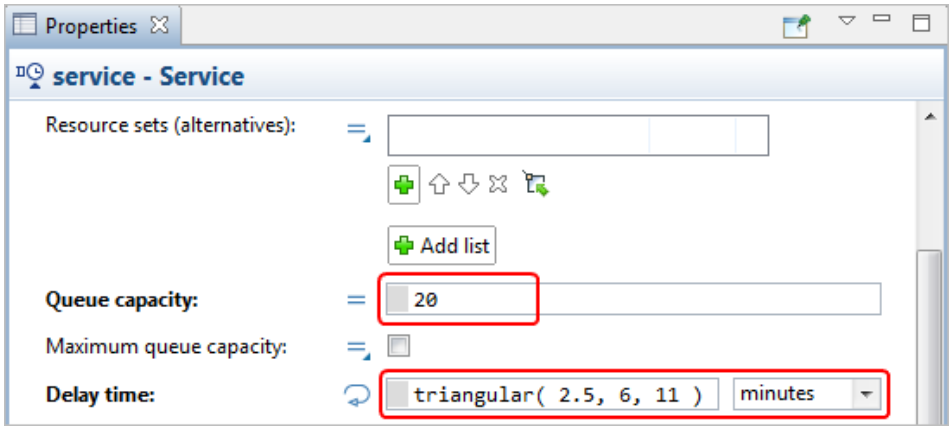
Simulate service

1. From the **Palette** view, drag the **Service** block onto our Main diagram.



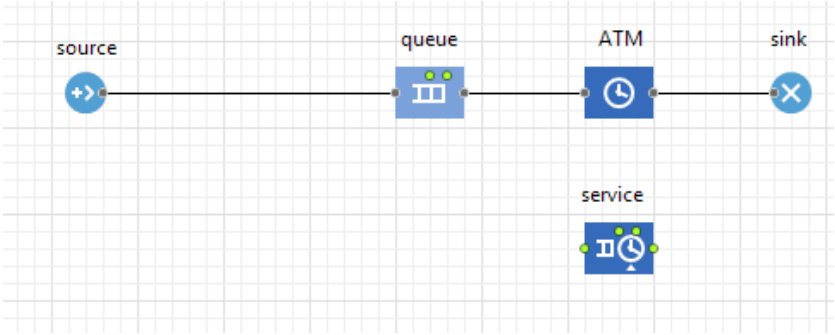
Service seizes resource units for the agent, delays the agent, and releases the seized units.

2. Go to the **Properties** view of the *service* block :
   Modify the object properties: There is one queue for all tellers.
   - Set up **Queue capacity** to be of 20 places.
   - Set **Delay time**: triangular(2.5, 6, 11)[We assume that service time is triangularly distributed with the min value of 2.5, average value of 6, and the max value of 11 minutes.]

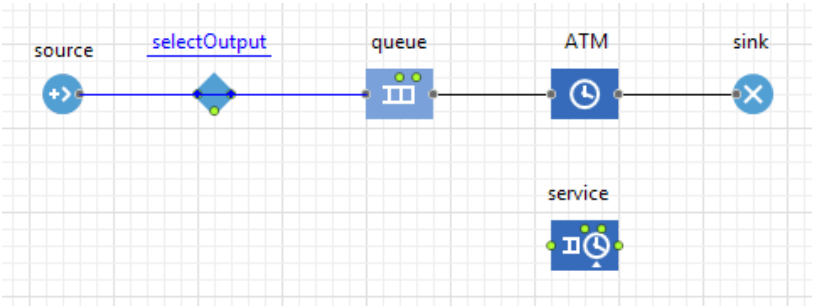


Simulate decision making

1. Move the objects *queue*, *ATM* and *sink* to the right to make space for one new object between *source* and *queue*.

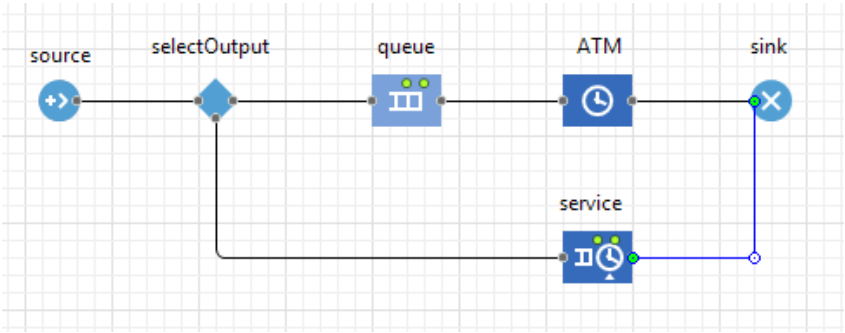


2. From the **Palette** view and add the **SelectOutput** block in the resulting space. When you place the object on the connector, it will automatically get built in.



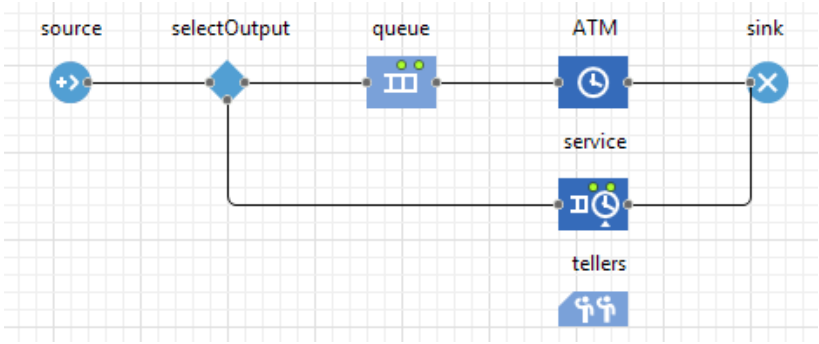
SelectOutput object is a decision making block. The agent arrived at the object is forwarded along one of two output ports depending on the user-defined condition.

3. Select *selectOutput* and go to its **Properties** view:
   - Choose the option *If condition is true* for the **Select True Output** parameter.
   - Make sure that **Condition** is randomTrue(0.5).
     This agent routing condition defines that the number of customers competing for ATM and teller service will be approximately equal.
4. Connect *selectOutput* and *service* with other blocks as shown in the figure:

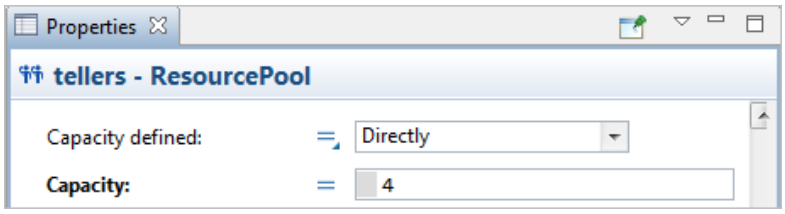


Add resources for the service

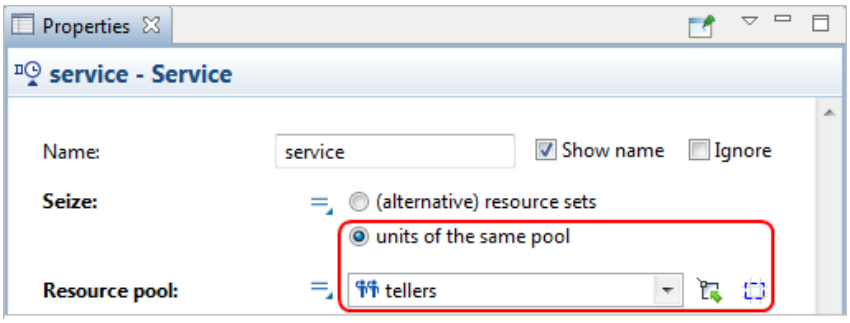
1. From the **Palette** view, drag the **ResourcePool** block onto our Main diagram.
   ResourcePool object is storage for resource units.
2. Place it under *service* and go to its **Properties** view:
   - Name the object *tellers*.



- Specify that this resource object has only four resource units, that means, define its **Capacity**: 4.



3. **ResourcePool** object should be connected to resource seizing and releasing objects (**Service** in our case). So we need to modify the properties of the *service* object :
    - Select *service* in the flowchart to open its properties.
    - Choose the option *units of the same pool* the parameter **Seize**.
    - Then specify the resource pool we have created in the option **Resource Pool**. You can either click the down arrow to select the resource pool object from the drop-down list, or you can click the button, located on the right, to select the object in the graphical editor (all inappropiate objects will be greyed out).



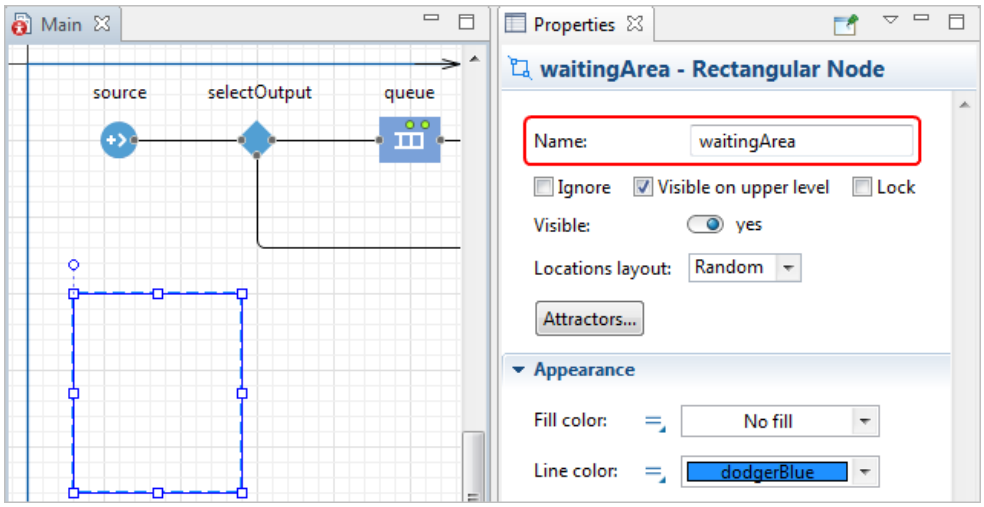
4. Now since the model has changed, we need to alter the model animation as well.

Adding space markup shapes

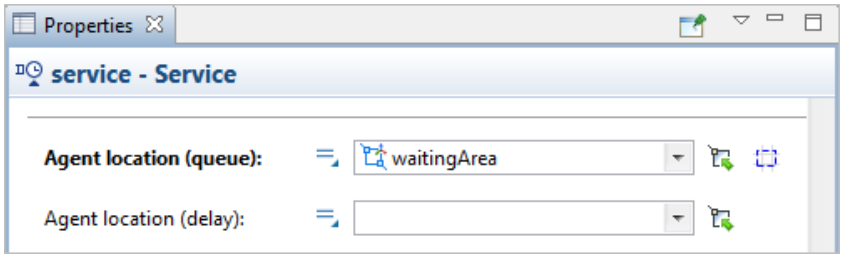
Now we want to draw the area for queueing and a place to get serviced for our clients.

Set up space markup for the queue to tellers

1. This time we will draw a waiting area using a rectangular node.
2. From the Palette view, Double-click the element **Rectangular node** to switch to *the drawing mode*.
3. Click in the graphical editor and drag the rectangle without releasing the mouse button. Release when you have a rectangular node of the required form. You can edit its form later as you need.
4. In it's Prperties:
    - Name the node *waitingArea*.



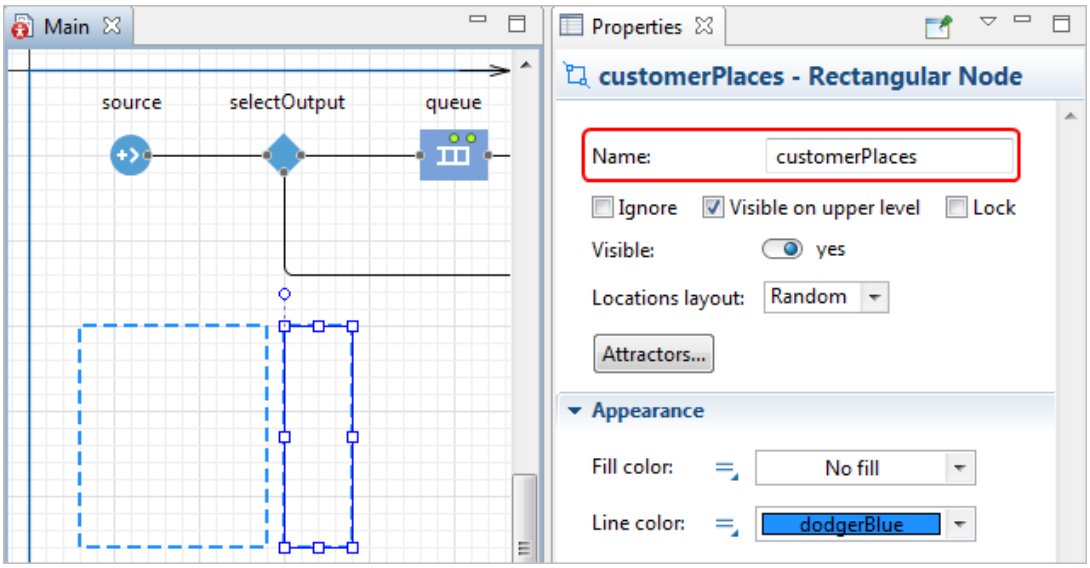
5. Click the *service* block in the flowchart and go to its **Properties** view.
    - Select the node *waitingArea* we have drawn in the **Agent location (queue)** option.
    -



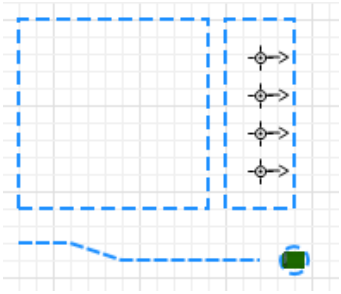
Set up space markup for the customers

The customers need a place to stand somewhere while they are getting serviced by tellers. We will draw an area for this purpose using a rectangular node.

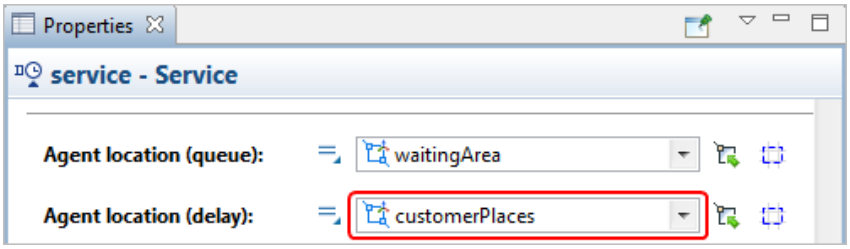
1. From the **Palette** view, double-click the element **Rectangular node** to switch to *the drawing mode*.
2. Drag and release the rectangle without releasing the mouse button.
3. In its Properties :
    - Name the node *customerPlaces*.



4. Select the node *customerPlaces* and click the button **Attractors...** in its properties - attractors to define the customers that are getting service.
5. In the **Attractors** window that will pop-up, specify *4* for the creation mode **Number of attractors** and click **OK**. You will see that attractors appeared in the *customerPlaces* node with the even offset.



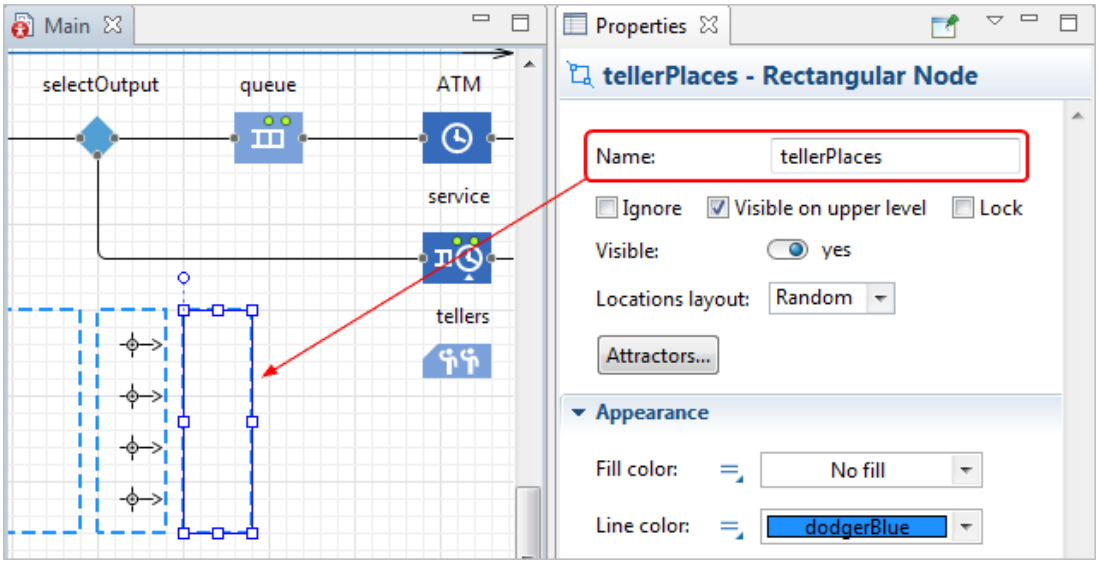
6. Click the *service* object in the flowchart and go to its **Properties** view.
    - Select the node *customerPlaces* have drawn in the **Agent location (delay)** option.



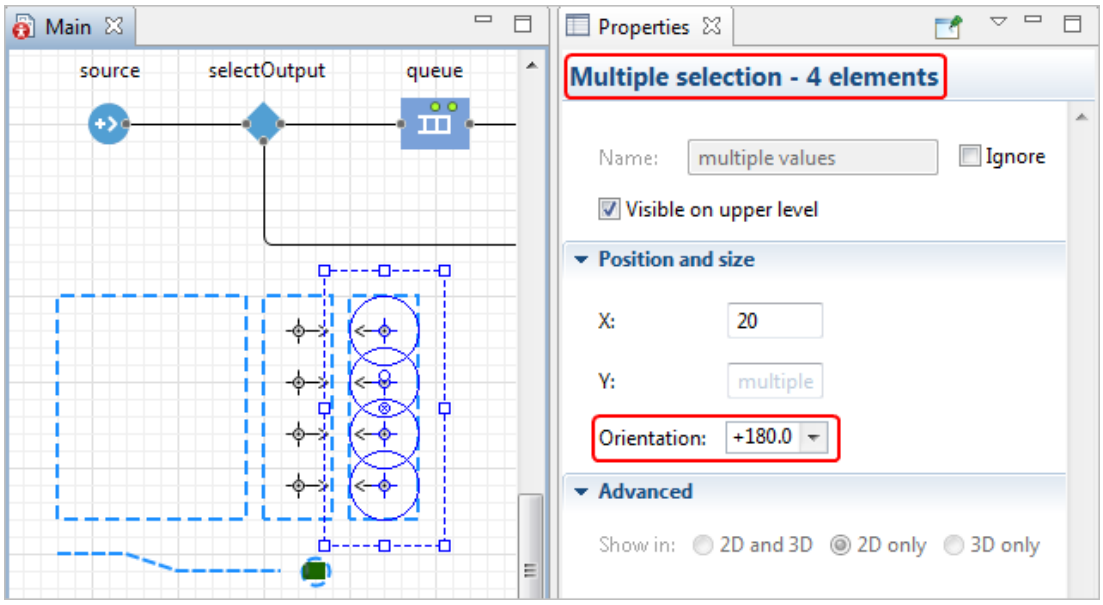
Set up space markup for the tellers

In the previous step, we used a point node to draw the ATM. Since we have 4 tellers this time, we will use a rectangular node to draw this service area.

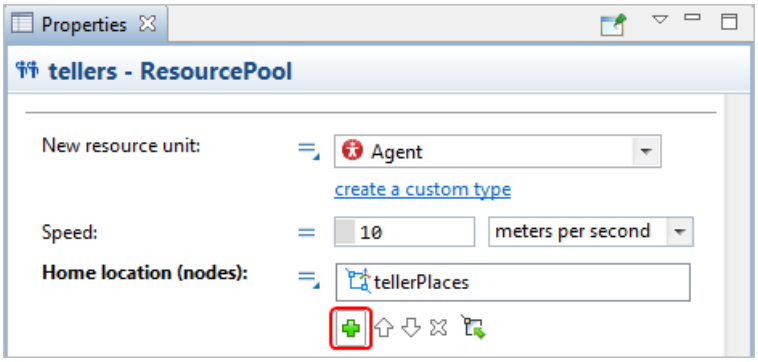
1. From the **Palette** view, double-click the element **Rectangular node** and drag the rectangle .
2. In its Properties : Name the node *tellerPlaces*.



3. Select *tellerPlaces* and click the button **Attractors...** in its properties.
4. In the **Attractors** window that will pop-up, specify *4* for the creation mode **Number of attractors** and click **OK**.
5. You will see that attractors appeared in the *tellerPlaces* node with the even offset, but they are facing wrong direction. Select all attractors by Shift+clicking and go to their properties. In the section **Position and size**, change the **Orientation** parameter to *+180.0*.



6. Click the *tellers* object in the flowchart and go to its **Properties** view:
   Select the node *tellerPlaces* we have drawn in the **Home location (nodes)** parameter.



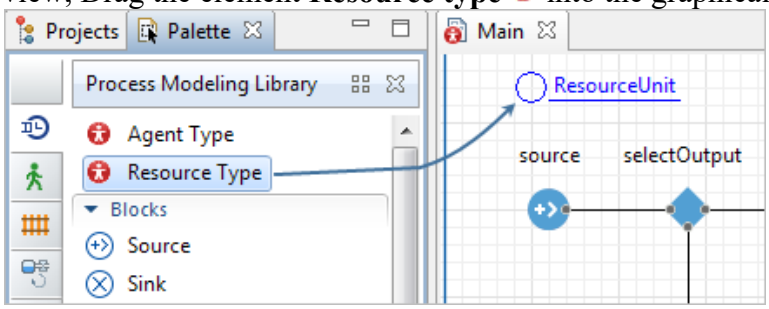
You can run the model now and observe how some customers are getting serviced at the ATM and some go to see tellers.

Adding 3D objects

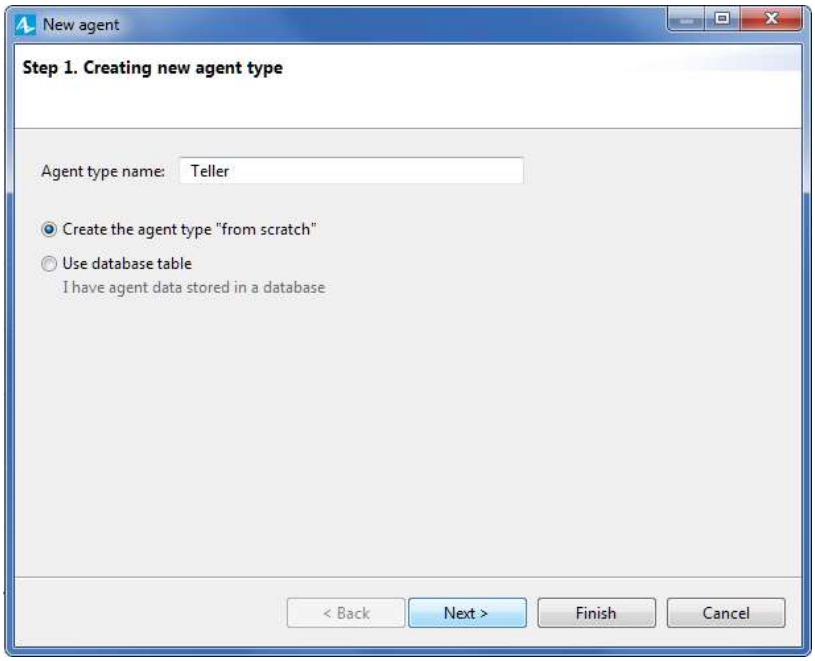
It is time to add teller 3D objects to our model. We will create a new resource type to animate tellers.

Create a new resource type

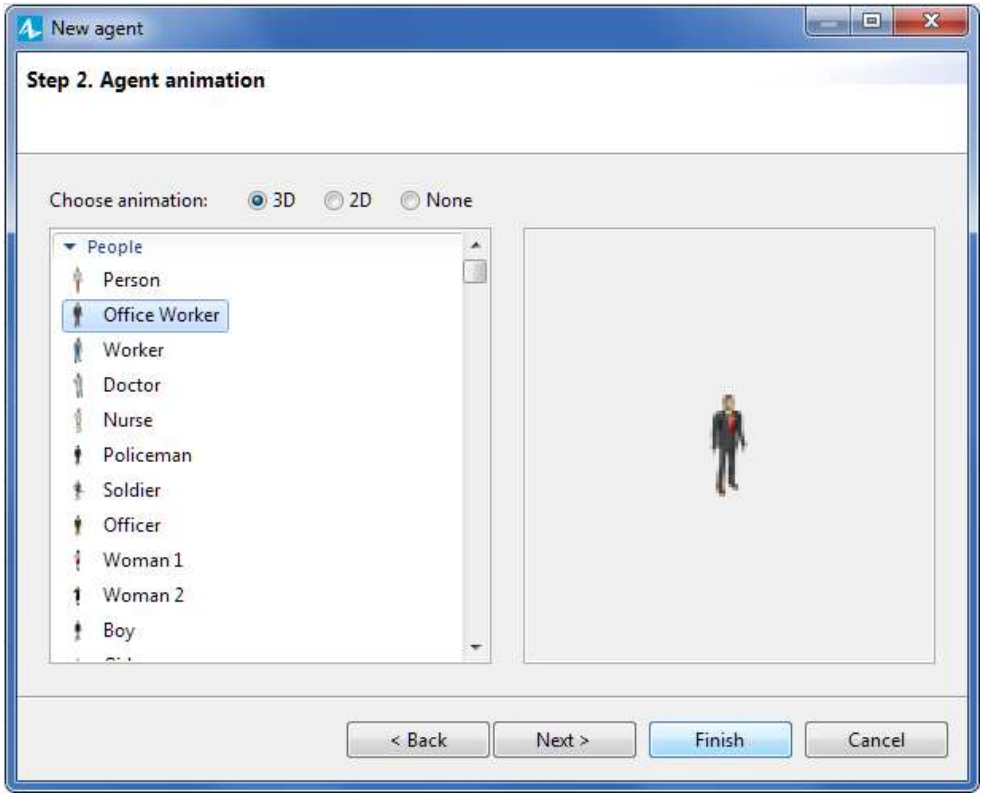
1. From the Palette view, Drag the element **Resource type** into the graphical editor.



2. The **New agent** wizard will open on the **Creating new agent** step.
   - Enter *Teller* as the **Agent type name** and
   - leave the **Create the agent type "from scratch"** selected.
   - Press **Next**.



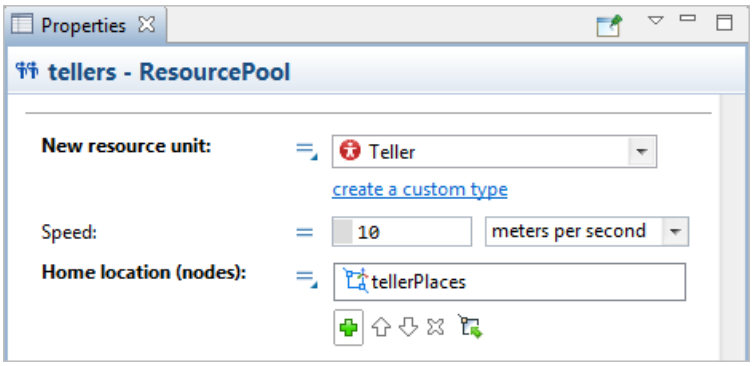
3. In the next step
    - select **3D** as the animation type and
    - select *Office worker* from the list of the 3D figures.
    - Click **Finish**.



4. The new Teller diagram will open. You can find the *Office worker* 3D figure in the axis origin. Switch back to Main diagram.

Configure flowchart to use the new resource type

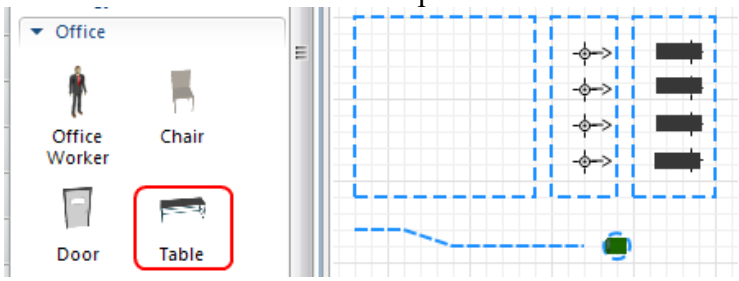
1. Goto block *tellers* properties:
    - specify Teller as the **New resource unit**.



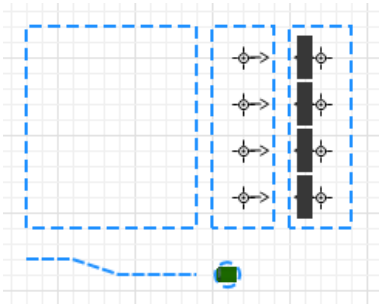
2. Run the model and observe customers and tellers.

Add tables for the tellers

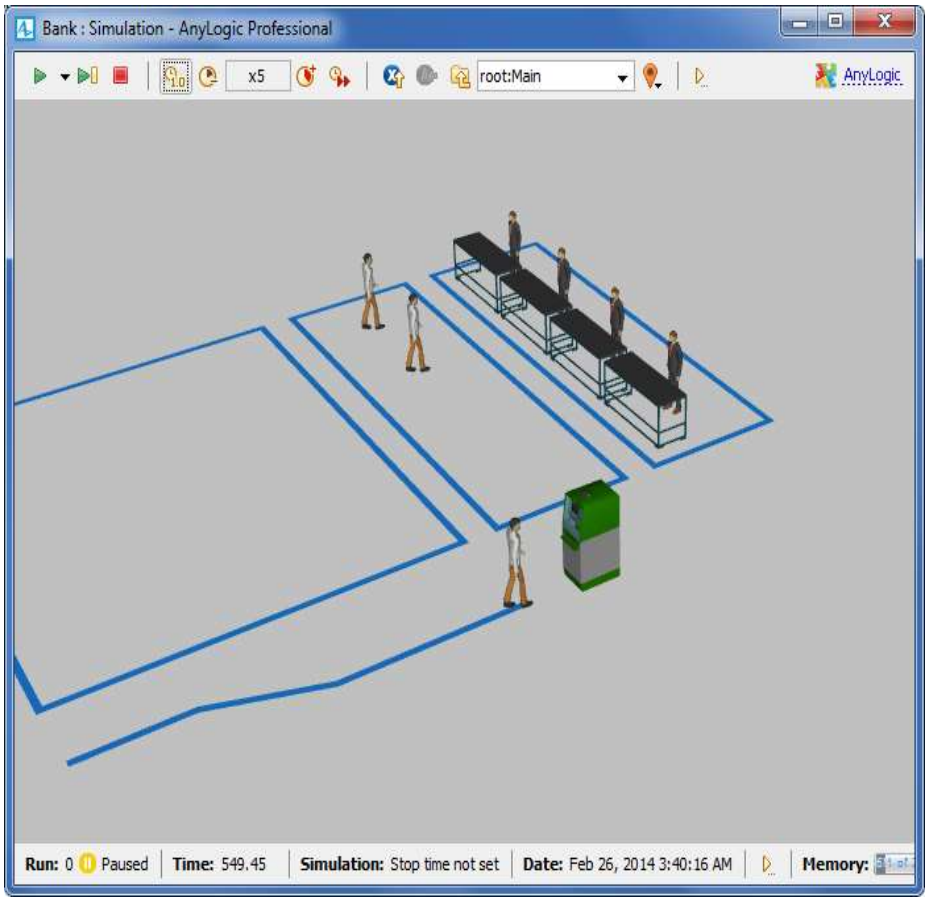
1. From the **Palette** view, Drag four **Table** 3D figures from the **Office** section of this palette onto the node shape called *tellerPlaces* in the graphical editor.
2. Place them at the attractors since attractors are the places where the tellers stand.



3. You can see that their orientation is wrong. Select all tables by Shift-clicking and go to the **Properties** view.
4. In the section **Position**, change the parameter **Rotation** to *-90.0* degrees.
5. If necessary, rearrange all eight attractors and four tables so that they are reasonably lined up.



Now you can run the model and observe in 3D how some customers go to the ATM and other get service at the tellers tables.



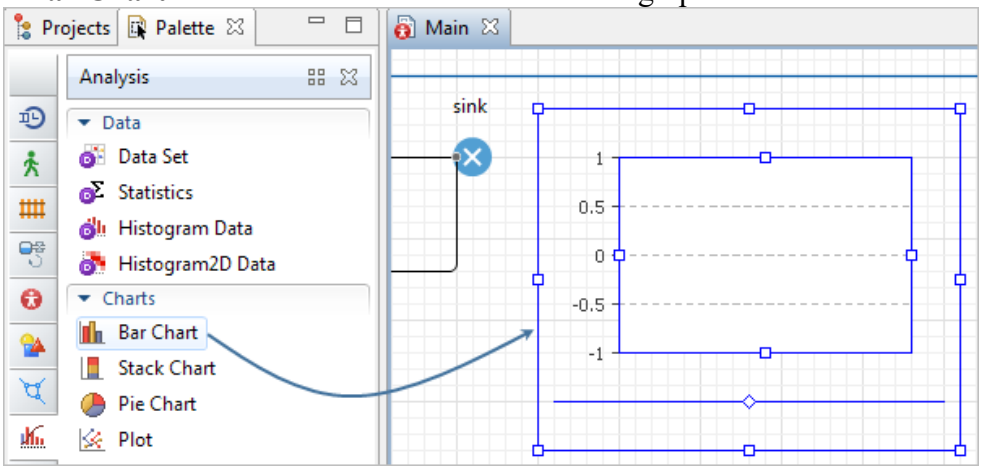
Phase 4. Adding Statistics Collection

To observe how mean ATM utilization and mean queue length change with time.

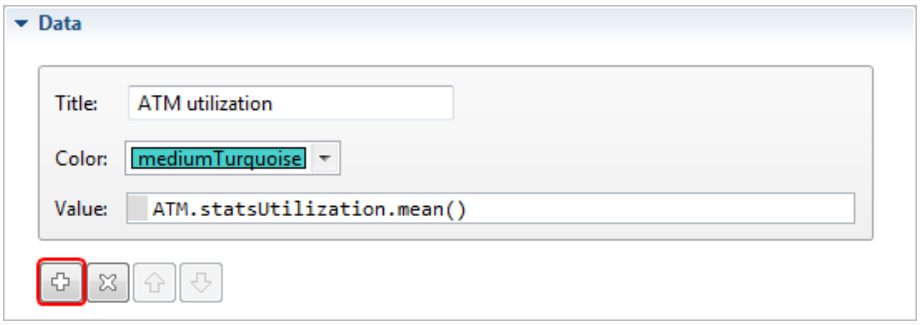
Collecting utilization statistics

Add a bar chart to indicate mean ATM utilization

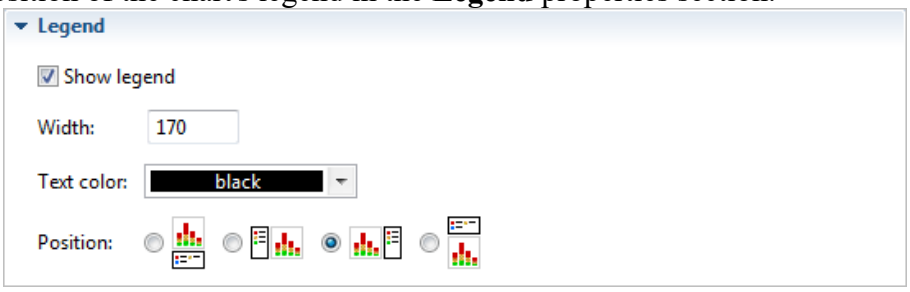
1. Open the **Analysis** palette of the **Palette** view. This palette contains charts and data objects used for collecting data and performing various statistical analysis on them.
2. Drag the **Bar Chart** element from the stencil into the graphical editor.



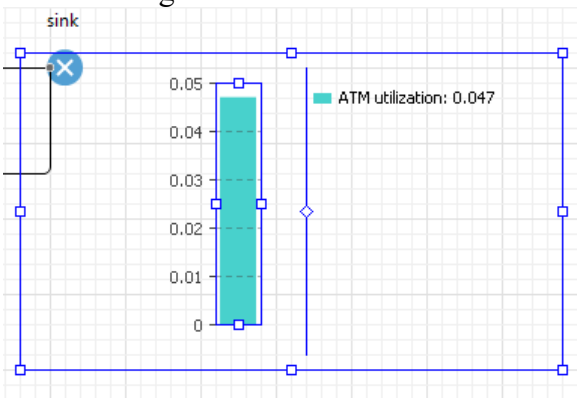
2. Go to the **Data** section in **Properties** of the chart.
   - Click **Add data item** to add data item to be displayed by this chart.
   - Modify the data item's **Title**: *ATM utilization*.
   - Type ATM.statsUtilization.mean() as the **Value** of the data item.



3. Change the position of the chart's legend in the **Legend** properties section.

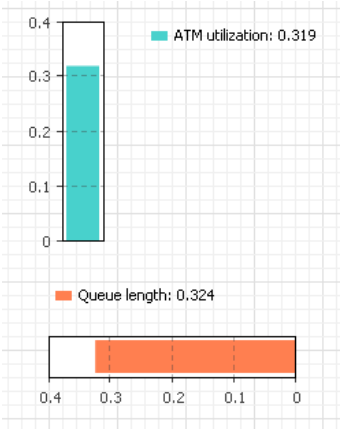


4. Then resize the chart as shown in the figure below:

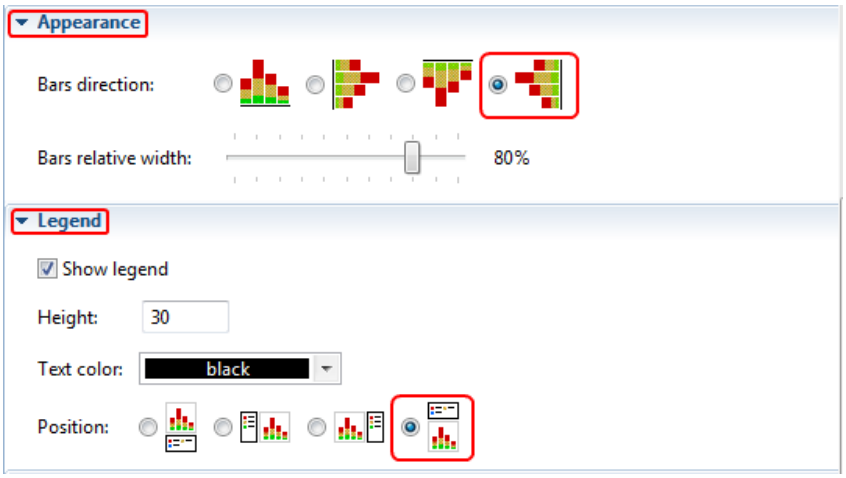


Add a bar chart to indicate mean queue length

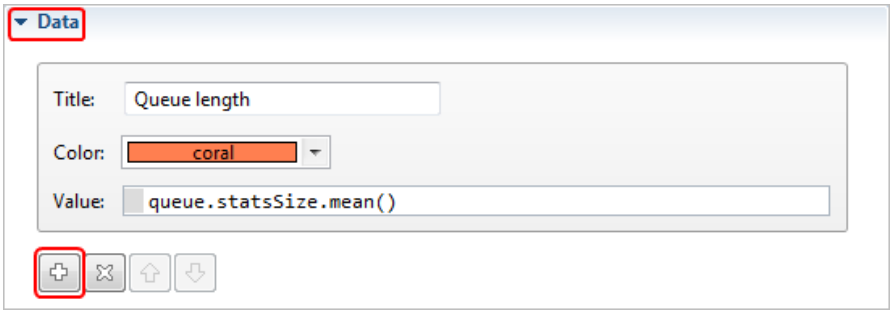
1. Add one more bar chart in the same way. Resize it to look like the one in the figure.



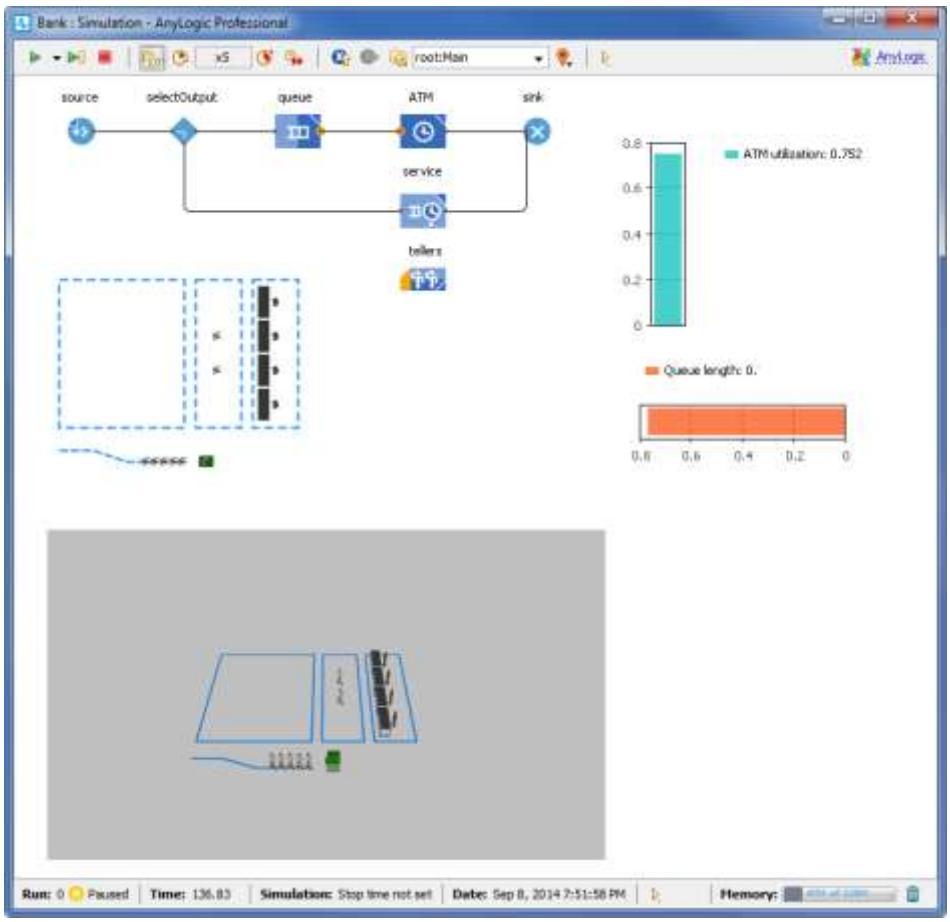
2. Open the **Appearance** section of the **Properties** view and choose the last option from the **Bars Direction** choice to make bars grow to the left and also change the position of the chart's legend in the section **Legend** (like it is shown in the figure below).



3. Add a data item to be displayed by the chart.
   - Set **Title**: *Queue length* and
   - **Value**: queue.statsSize.mean()



4. Run the model and observe the ATM utilization and mean queue length with just created charts.

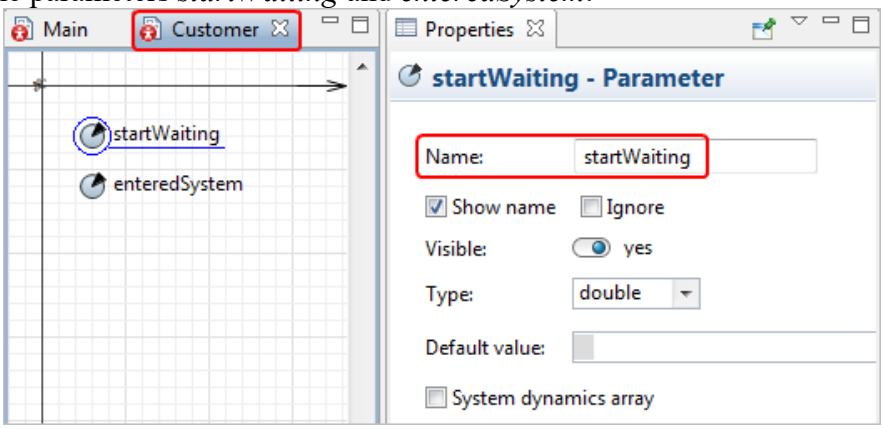


Collecting customer time statistics

We want to know how much time customer spends waiting in ATM queue and the whole time he spends in the bank. We will collect time statistics using AnyLogic analysis data objects and observe the resulting time distributions using histograms. We will use the agent type Customer we have created on the step 2. First we need to add two parameters into our model.

Add parameters

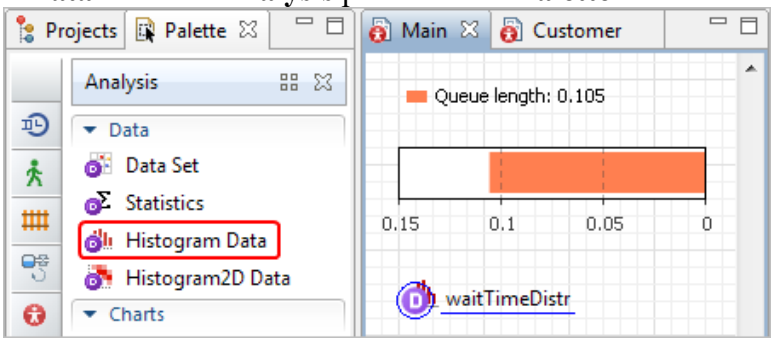
1. Switch to the **Projects** view. Double-click the agent type Customer to open its diagram. We need to create parameteres for the agent type Customer if we want to collect customer statistics.
2. Open the **Agent** palette of the **Palette** view.
3. Drag the element **Parameter** into the diagram of Customer.
   - Leave the type double as it stands by default and
   - name the parameters *startWaiting* and *enteredSystem*.



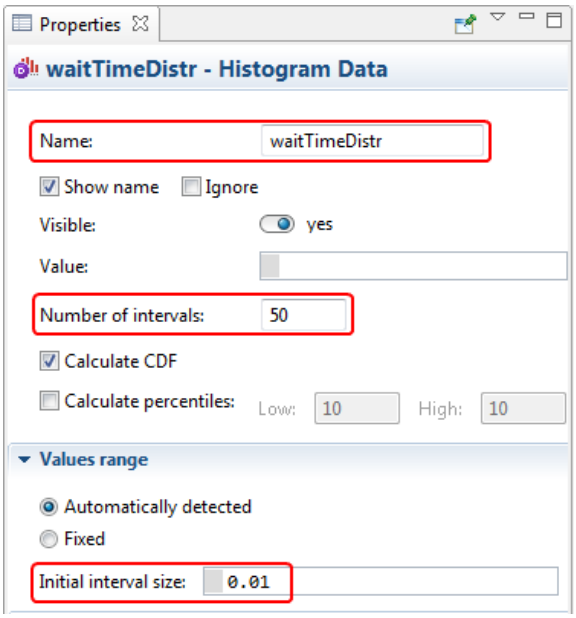
4. We will continue developing our model in the Main diagram.

Add histogram data objects to collect time statistics

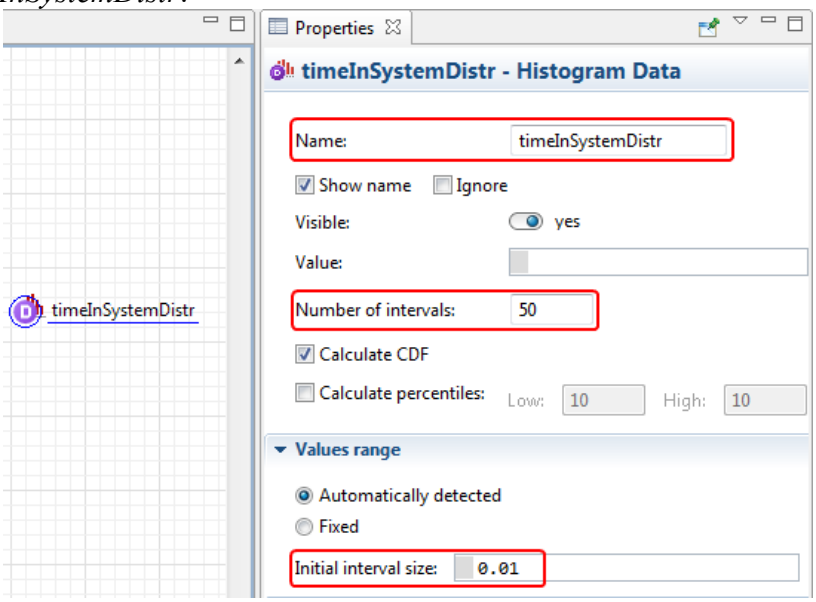
1. Drag the **Histogram Data** from the **Analysis** palette of the **Palette** onto the Main diagram.



2. Set up the properties of the element.
   - Change the **Name** to *waitTimeDistr*.
   - Set the **Number of intervals** equal to *50*.
   - Set the **Initial interval size**: *0.01*.



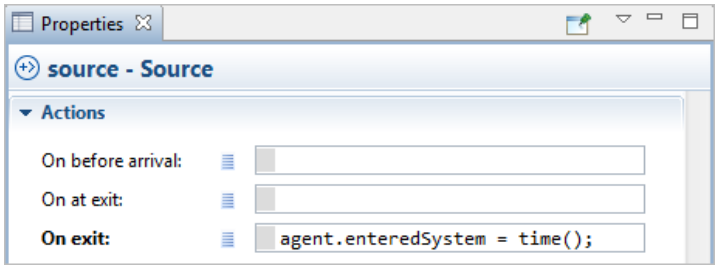
3. Create one more histogram data object. Ctrl+drag histogram data object to create its copy. Change the **Name** to *timeInSystemDistr*.



Now we will modify properties of our flowchart objects.

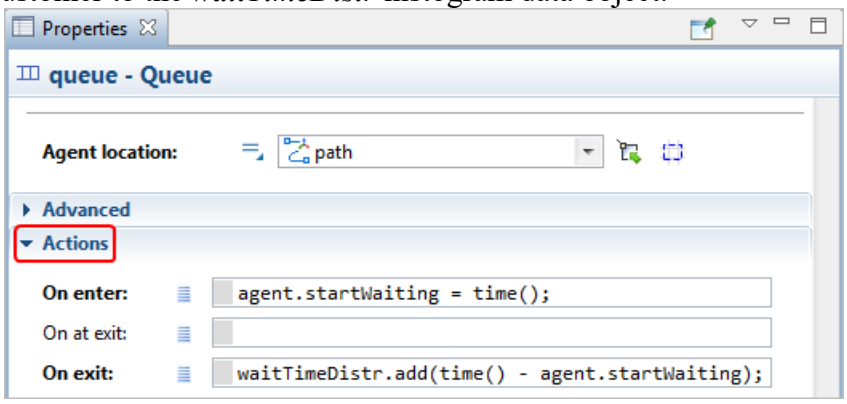
Modify the properties of the flowchart objects

1. Modify *source* properties:
   - Make sure the agent type Customer is specified as **New agent**. This object should continue generating agents of our Customer type.
   - Type agent.enteredSystem = time(); in **On exit** action, located in **Actions** section. This code stores the time when a customer was generated in the Customer's variable enteredSystem. The time() function returns the current model time value.

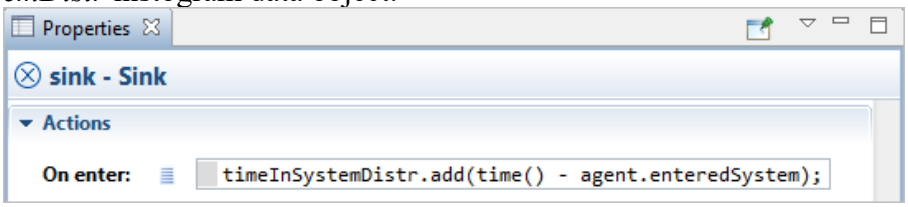


2. Modify *queue* properties:

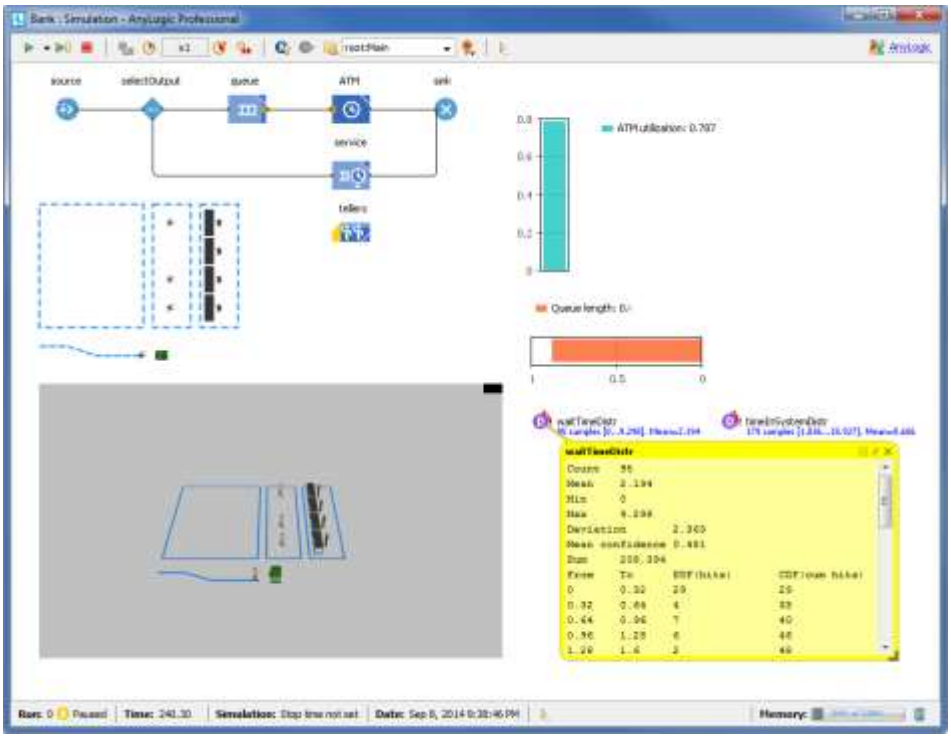
- Type agent.startWaiting = time(); in **On enter** action, located in **Actions** section. This code stores the time when a customer started waiting in the queue in the Customer's variable startWaiting.
- Type waitTimeDistr.add(time() - agent.startWaiting); in **On exit**. This code adds waiting time of the customer to the *waitTimeDistr* histogram data object.



3. Modify *sink* properties:
   - Type timeInSystemDistr.add(time() - agent.enteredSystem); in **On enter** action, located in **Actions** section. This code adds the whole time the customer spent in the bank to the *timeInSystemDistr* histogram data object.



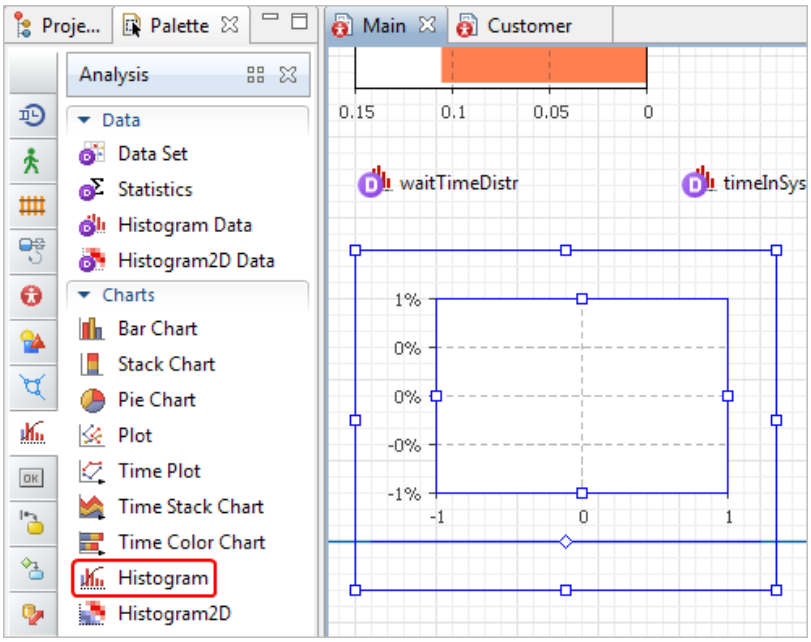
4. Run the model and view the statistics using inspect window of the data set. Open inspect window for data set by clicking on it. Here you can see standard statistical analysis on the data values being added to this data object.



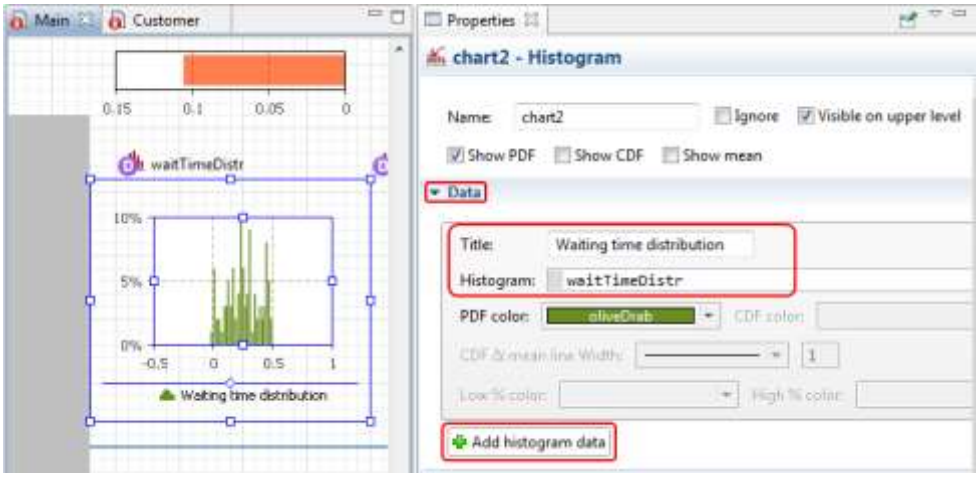
Now we want to display the collected statistics using standard histograms.

Add two histograms to display distributions of customer's waiting time and time in system

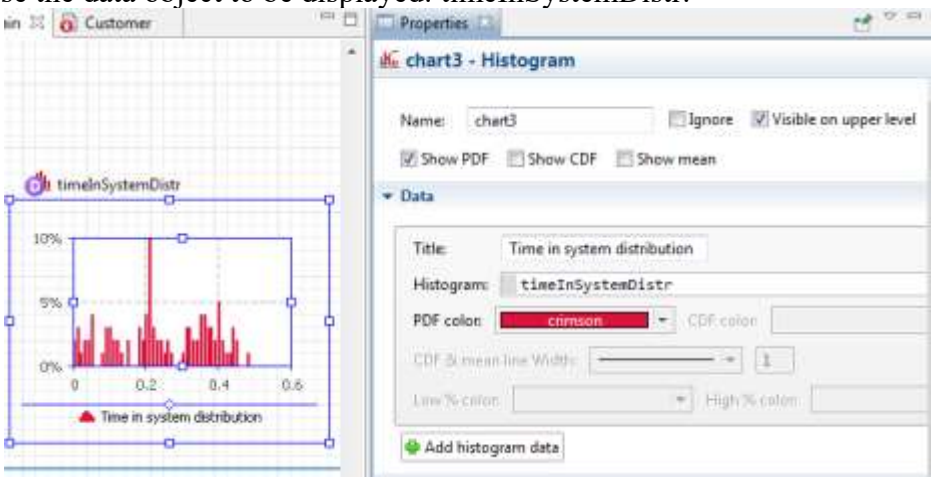
1. Drag the **Histogram** element from the **Analysis** palette of the **Palette** onto the diagram where you want to place the histogram. Resize it if needed.



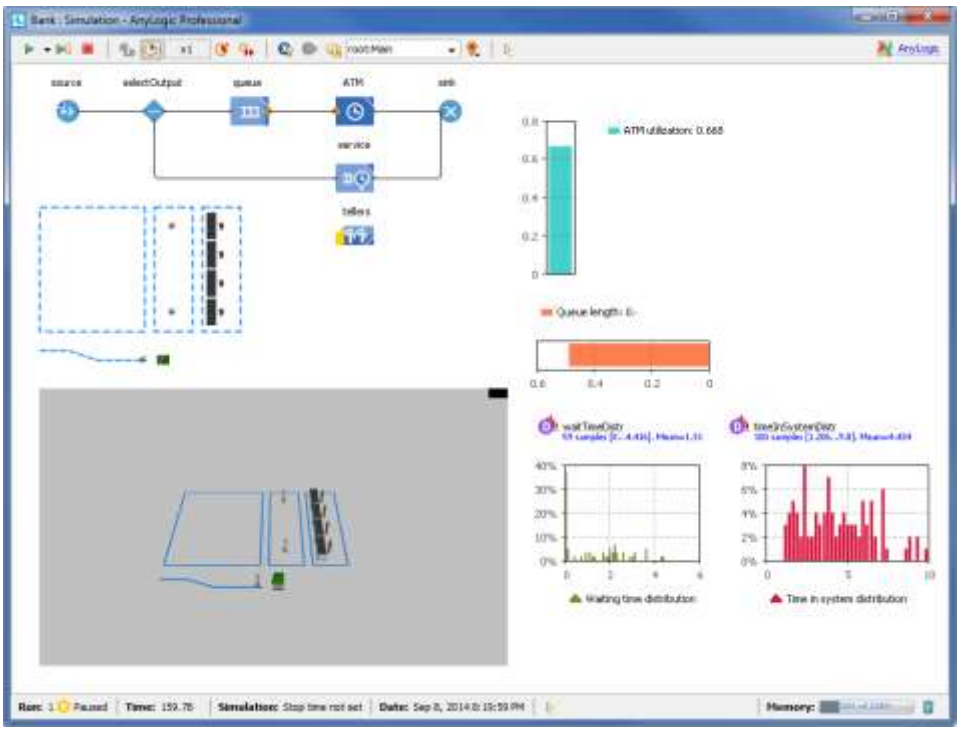
2. Define the data object to be displayed by this histogram.
   - Click the **Add histogram data** button in **Data** section and
   - Change the **Title** to *Waiting time distribution*.
   - Specify the **Histogram** to be displayed: waitTimeDistr
   -



3. Add one more histogram below the existing one.
   - Change the **Title** of the displayed data to *Time in system distribution.*
   - Choose the data object to be displayed: timeInSystemDistr.



4. Run the model. Set virtual time mode and observe distribution of customer's time in system and waiting time.



Practical 8

AIM : Create Defense model to stimulate aircraft behavior.

Air Defense System model Scenario :

We will build a simple model of a radar-based air defense system. All types of agents in this model (bomber aircrafts, radars, missiles, bombs, and buildings) exist and interact in the continuous 3D space. We will use various kinds of movement and space sensing techniques.

Bombers are sent to destroy ground facilities compactly located in a certain area. One aircraft is launched per building. Aircrafts carry bombs. To complete its mission successfully, an aircraft needs to drop a bomb on the building. Having completed the mission, the bomber returns to the base using a higher altitude route. The bomber speed is 600 km/h.

The buildings are protected by the air defense system, which consists of two radars equipped with guided surface-to-air missiles. A radar can simultaneously guide up to two missiles. A missile is launched once the bomber enters the radar's coverage area, which is a hemisphere of 1 kilometer radius around the radar. The missile speed is 900 km/h. The missile explodes once it gets as close as 100 meters to the aircraft. If the missile exits the radar coverage area before it hits the aircraft, it destroys itself.

The model development consists of four phases, with a ready-to-run model at the end of each phase.

Phase 1. Creating assets

In the first phase we will create a new model and populate it with buildings that will be subject to bombing.

Phase 2. Adding bombers

In the previous phase we created a new model, populated it with facilities, defined animation shapes for the facilities, added 3D view and camera to observe the running model and finally added protected area. In this phase we will add bomber aircrafts to our model, define their behavior and mission.

Phase 3. Adding bombs

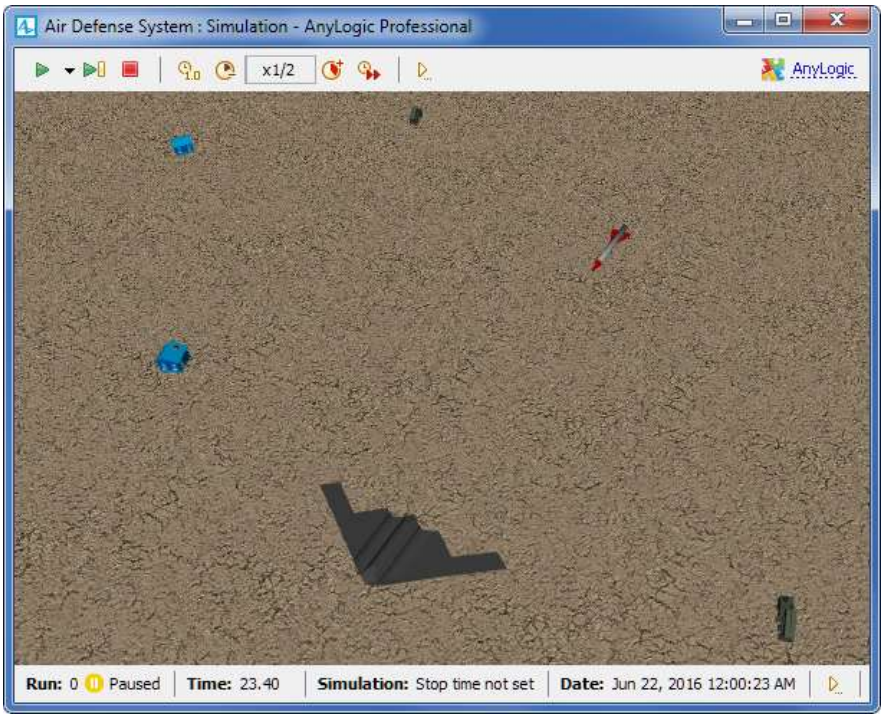
In this phase we will add bombs to our model and learn how to destroy facilities with them. To do that we must create interaction between the bomber and the target building. We will use yet another agent type, Bomb. On approaching the attack distance, the bomber will drop a bomb onto the targeted building, which will change its state to **Destroyed** once the bomb has reached the building.

Although aircrafts carry bombs, the Bomb agents will not be located inside the Bomber agent in the model, but directly inside the Main agent, i.e. at the same level as the bombers and the buildings populations. It will be easier to place bombs in the same space with this model architecture.

Phase 4. Adding air defense system

In this phase we will add radars and radar-guided missiles as agents to our model. The radar will scan the air within its coverage zone every second. Once it detects a bomber and is able to guide yet another missile, it will launch one. The missile will act similarly to the bomb by engaging with the bomber (the radar will pass the bomber to the missile constructor) and exploding once it has got as close as 100 meters to the bomber.

Alternative approach would be to analytically calculate the exact moments of time at which a bomber gets close enough to the target to drop a bomb, enters the radar coverage zone, or when a missile catches up with the bomber. With the given linear or piecewise-linear trajectories and constant velocities of bombers, this would be a task of a medium (high school level) mathematical complexity, which can be done as an exercise. The resulting model will be more accurate and the simulation will be much more efficient. However, the approach we chose (recalculation of the geometric conditions on each time step) is simpler with no analytical skills required at all. It is also more general. It will work regardless of the movementtype.



Solution :

Phase 1. Creating assets

Create a new model

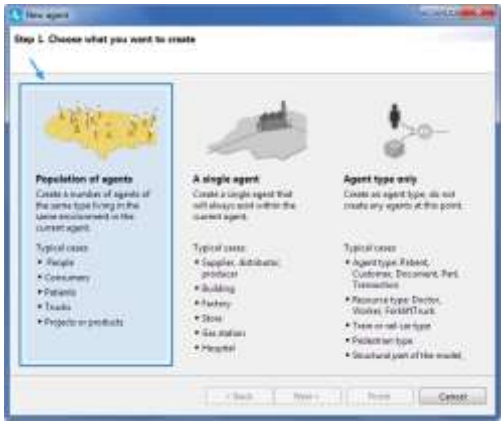
1. Click the **New** toolbar button. The **New Model** dialog box will be displayed.
2. Define the parameters of the new model:
   - Specify the name of the model in the **Model name** field. Type *Air Defense System*.
   - Set the new model location if necessary by clicking the **Browse** button and selecting the required folder.
   - Set **Model time units** to **seconds**.
3. Click **Finish** to create a new model.

We shall start designing our model with creating a new population of agents that will represent 10 facilities.

Create new population of agents

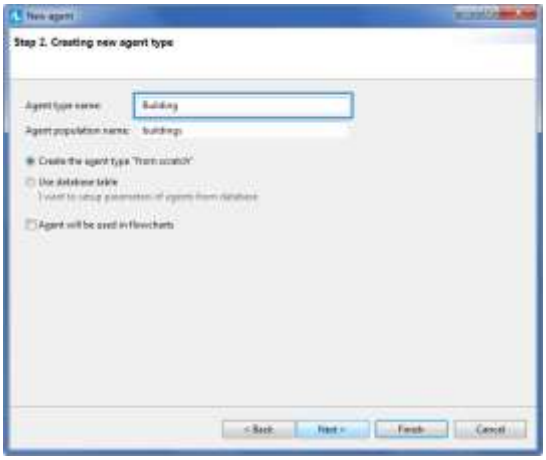
1. Drag the **Agent** element from the **Agent** palette onto the Main diagram. The **New agent** dialog box will be shown.

   Step 1: Choose what you want to create –
   - Click **Population of agents**.
   - Click Next.



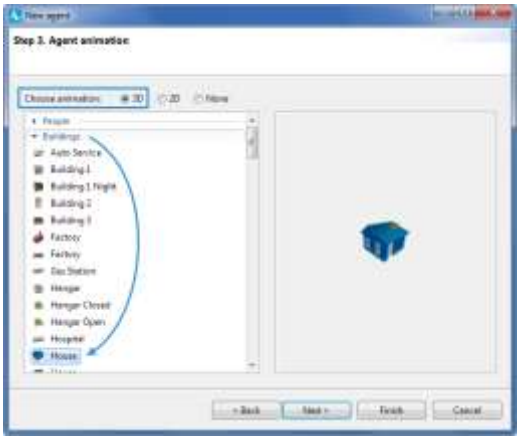
   Step 2: Creating new agent type –
   - **Agent type name** field - Building.
   - The **Agent population name** will be automatically filled.

- Click **Next** to proceed.



Step 3: Agent animation –

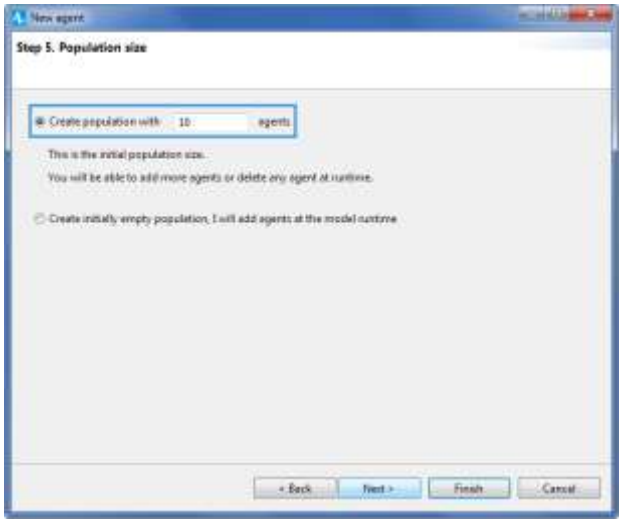
- **Choose animation** parameter set to **3D** mode and
- choose the **House** animation shape in the **Buildings** section.
- Click **Next** to proceed.



Step 4 : We will not set any agent parameters, you may click **Next** to proceed .

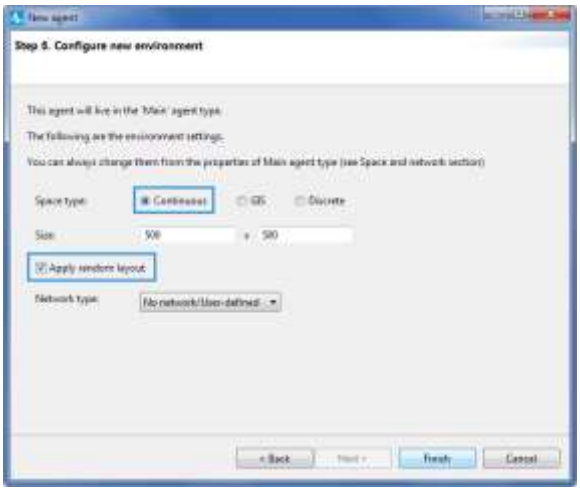
Step 5: Population size –

- Set the **Create population with** parameter to *10*.
- Click **Next** to proceed.

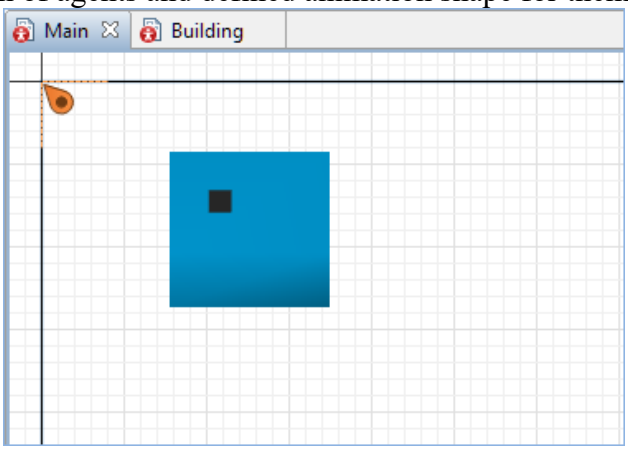


Step 6: New agent –

- **Space type** parameter set to **Continuous** and
- Select the **Apply random layout** option.
- Click **Finish** to save the changes and create the customized population.

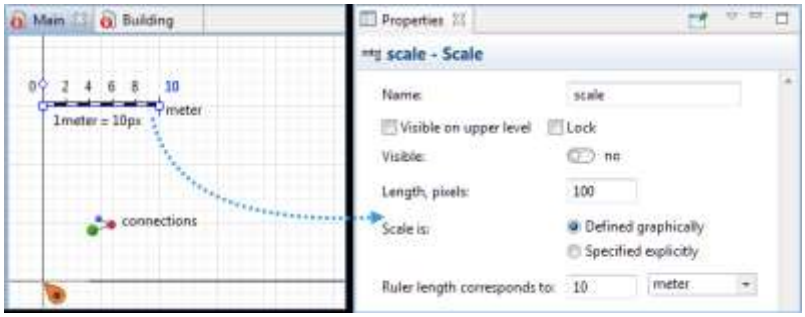


We have created a population of agents and defined animation shape for them.

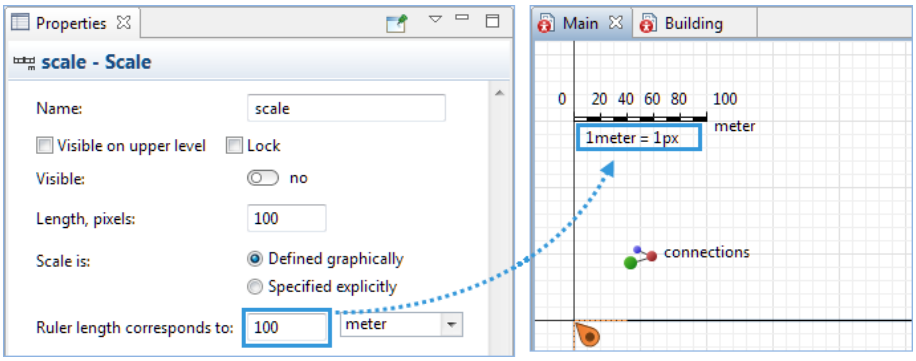


Adjust the scale of the model

1. Click the **Scale** element on the Main diagram to open its **Properties**.

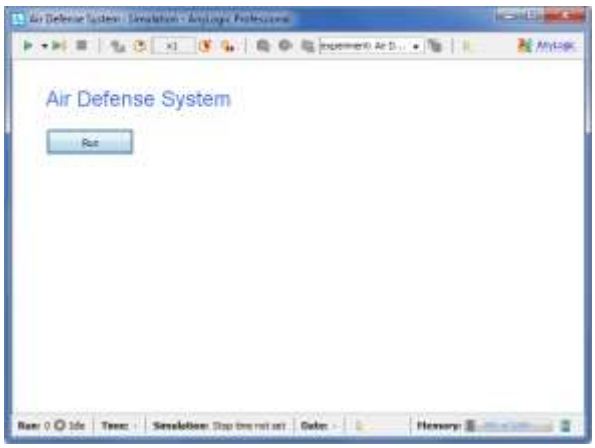


2. Set the **Ruler length corresponds to** parameter to *100*. The scale on the ruler will change.

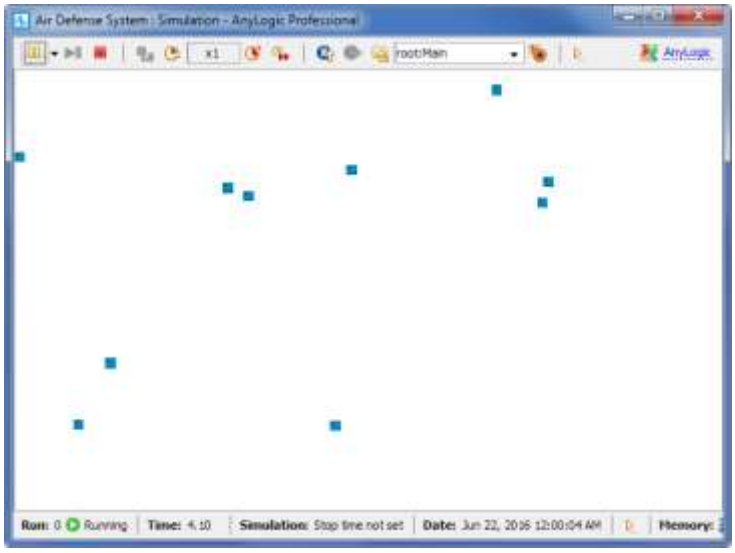


We modified the scale of the model, as a result the building shape changed its size.
We have completed setting up the population of agents and may now run our model for the first time to observe the model behavior.

Run the model

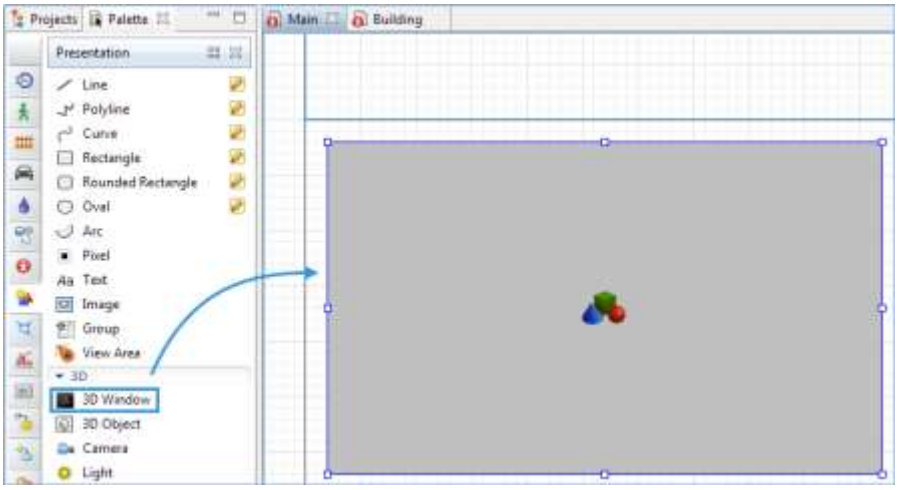


Click **Run** to launch the model. You will see the 10 building shapes randomly distributed in a 500 by 500 pixel space. This is the space size set by default. We do not need to change it as we will customize the location of the facilities.

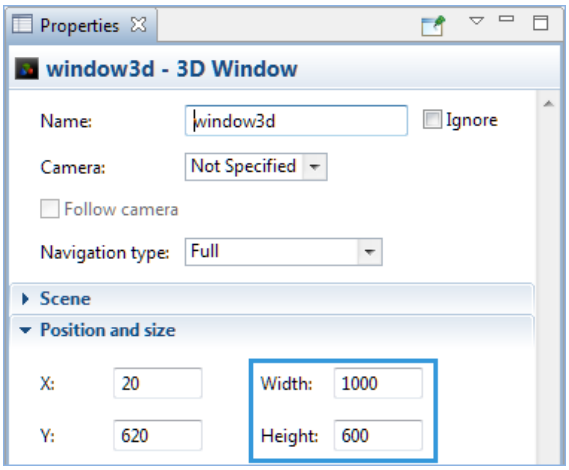


Add 3D window

1. Drag the **3D Window** element from the **3D** section of the **Presentation** palette to the graphical editor. Place the window below the model window frame.



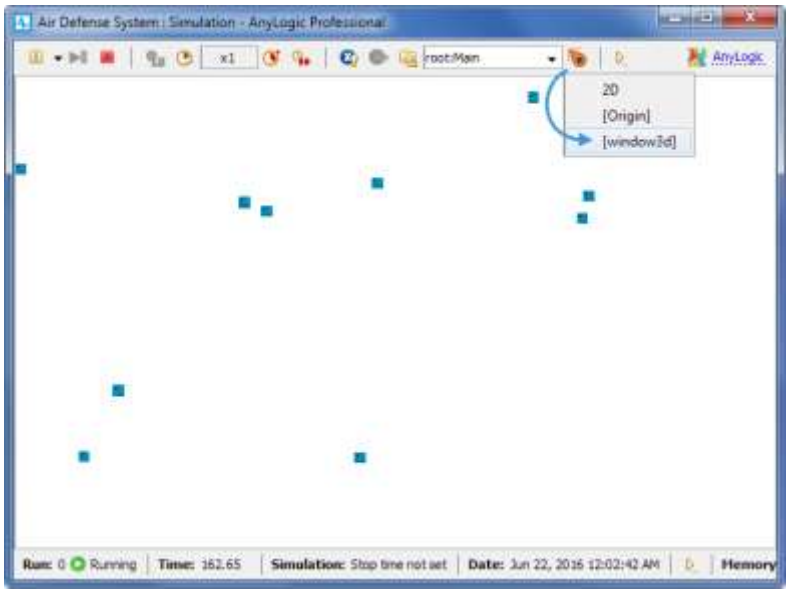
2. Navigate to the **Properties** view of the **3D window** and set its **Width** and **Height** parameters to *1000* and *600* respectively.



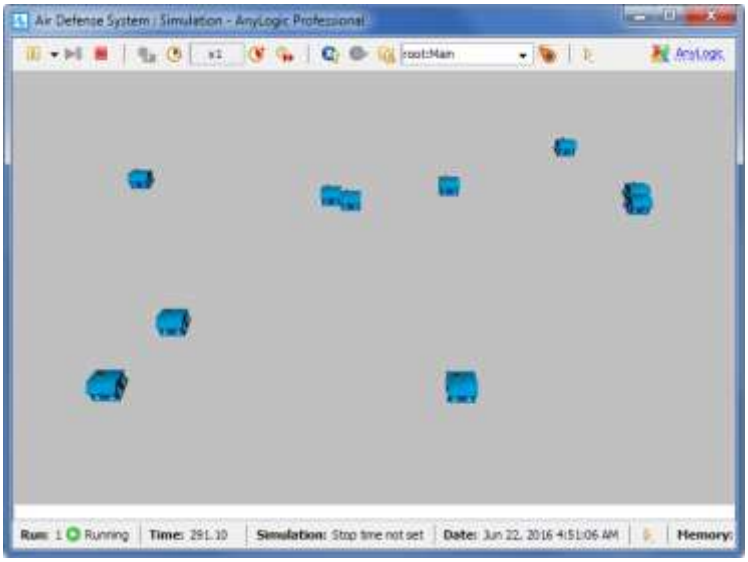
Now we will run the model and switch between the 2D and 3D views in the presentation window.

Compare 2d and 3d views

1. Click **Run** in the toolbar to open the presentation window.
2. Click the **Run** button in the presentation window to launch the model. You will observe the model in the 2D view.
3. Click the **Navigate to view area...** toolbar button and select **[window3D]** from the drop-down list.

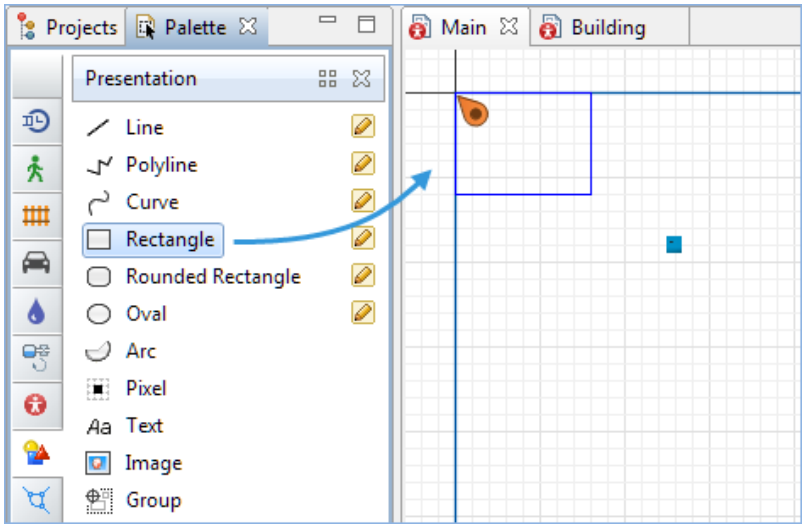


The view mode will change to 3D.

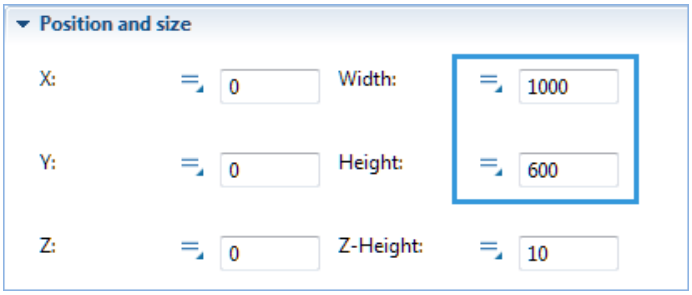


Draw the surface

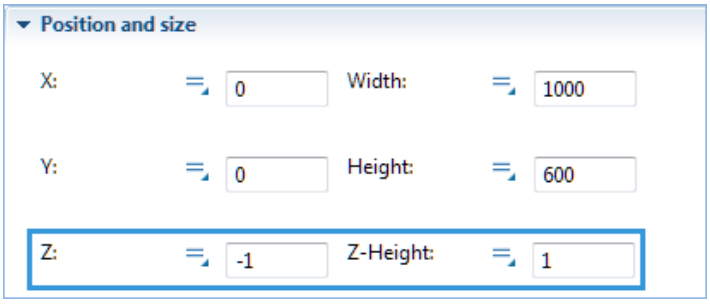
1. Drag a **Rectangle** shape from the **Presentation** palette onto the editor of Main, place its top left corner at (0,0).



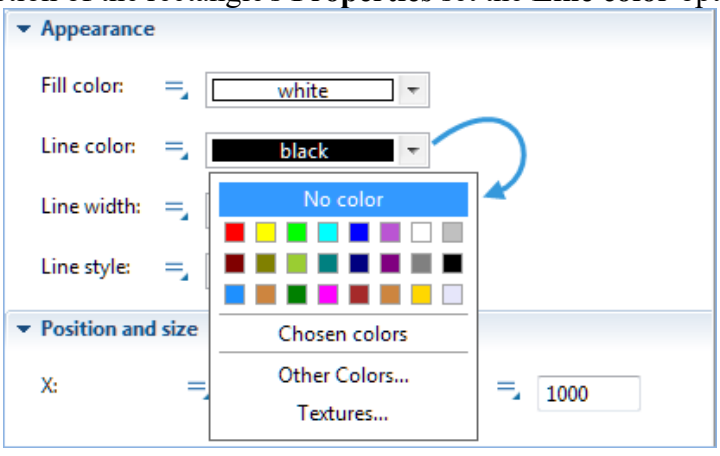
2. Navigate to the **Position and size** section of the rectangle's **Properties** and set its **Width** and **Height** parameters to *1000* and *600* correspondingly, hence the area size is 1000*600 meters.



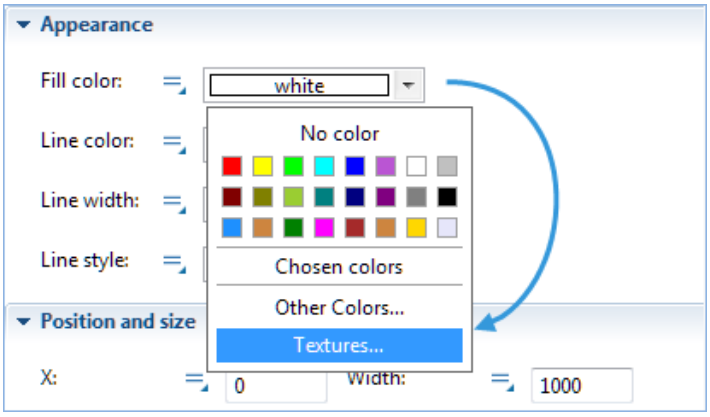
3. Now set the **Z** and **Z-Height** parameters to *-1* and *1* correspondingly.



5. In the **Appearance** section of the rectangle's **Properties** set the **Line color** option to **No color**.



6. Now we will define the texture of the rectangle area. Click the **Fill color** control and select **Textures** to open a dialog box with available textures.

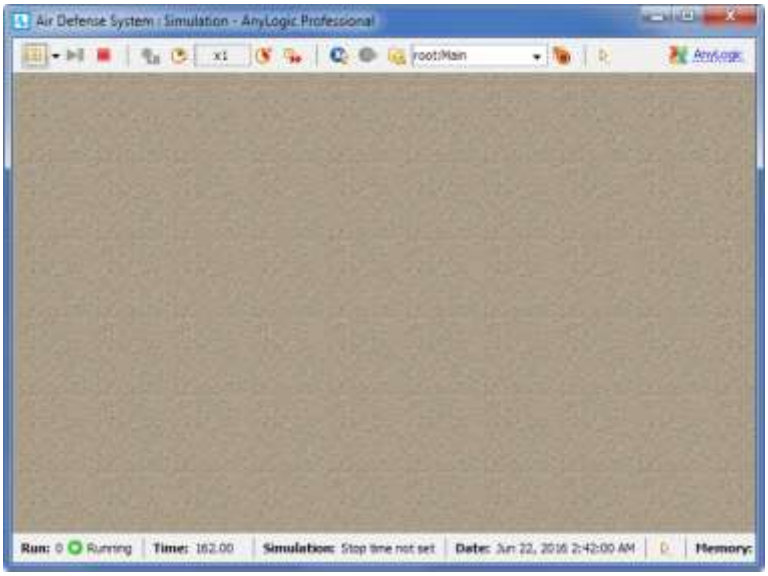


7. Click **Earth** texture to set it as the fill color for the rectangle shape.

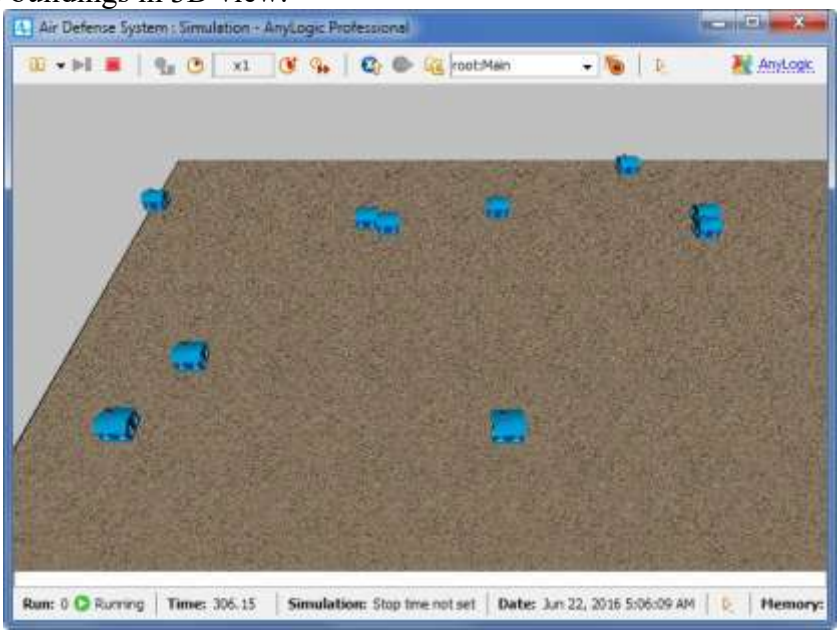
8. Click **OK** to close the dialog box. The rectangle shape will change it appearance.
Now run the model and observe the new layout both in 2D and 3D modes.

Observe the new layout

1. Click **Run** in the toolbar to open the presentation window.
2. Click the **Run** button in the presentation window to launch the model. You will observe the model in the 2D view.
   As you can see, we have the new layout here but no buildings can be seen.



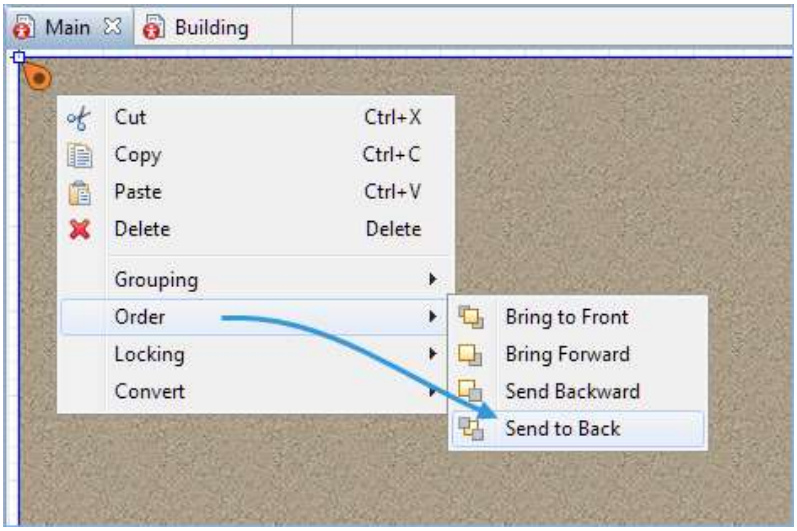
3. Switch to 3D mode by clicking the **Navigate to view area...** toolbar button and selecting **[window3D]** from the drop-down list.
   You can observe buildings in 3D view.



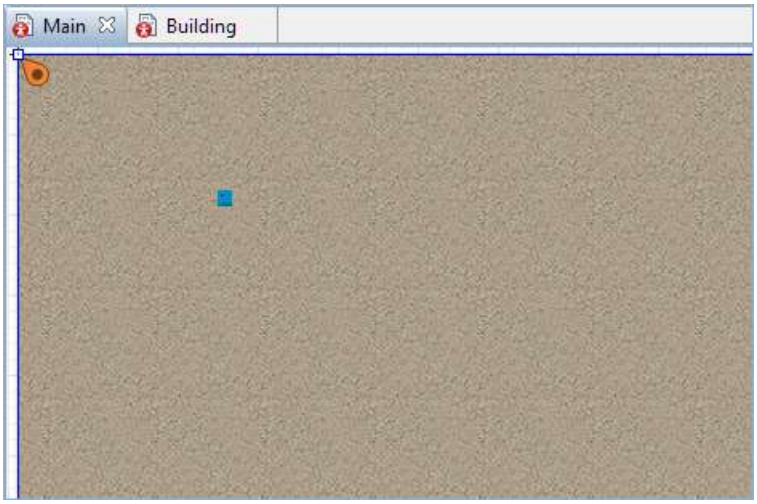
In order to make buildings visible in the 2D view we must reorder the shapes in the design.

Reorder shapes

1. Right-click the rectangle shape to open its context menu. Then click the **Order** menu item and select **Sent to Back** sub-item from the list of available orders.



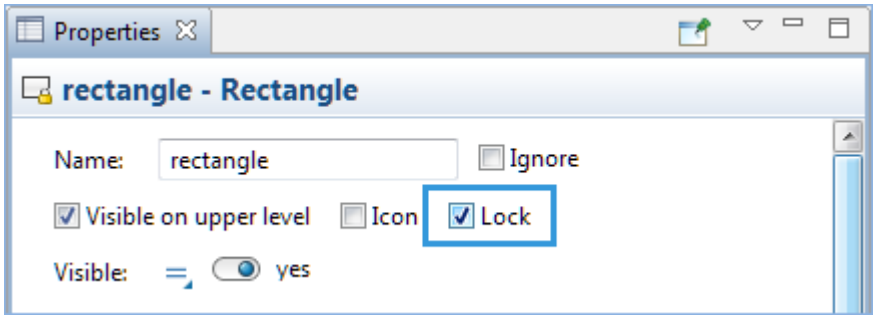
   The 2D building shape will now be visible.



Now the model should have correct visualization in both 2D and 3D view. Prior to running the model to check if everything works like it should, we will lock the **Rectangle** shape and add camera.

Lock the rectangle shape

1. In the **Properties** view select the **Lock** option to lock the shape.
   This will prevent occasional selection of the rectangle while working with other elements on the diagram.



Add a camera

1. Drag the **Camera** element from the **3D** section of the **Presentation** palette onto the graphical editor. You will see the camera icon in the graphical editor.
2. Direct the camera at the buildings presentation shapes (the positioning is described in details in the Camera document).

It is time to run the model and adjust the camera position to the desired one. We will save that position afterwards to have the camera always located at the set position.

Adjust the camera position

1. Run the model and switch to 3D mode.
2. Right-click the 3D model presentation to open a pop-up menu. Navigate to the **Camera** menu item and select **camera** sub-item.
   You will switch to the camera view.
3. Position the camera to have the best possible picture while observing the model in the runtime (for more details on navigating the 3D scene refer here).
4. When done, right-click the 3D model presentation and click **Copy camera's location**.

You may stop the simulation now. We have set up our camera and copied its position, now we need to apply the new camera position.

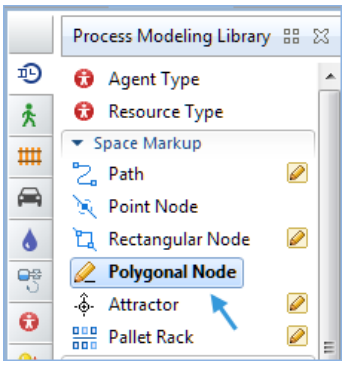
Apply new camera position

1. Click the Camera element to open its **Properties** view.
2. Click the **Paste coordinates from clipboard** button in the **Properties** view. The rotation parameters will change.

In the last step of this phase we will create a protected area, within which the buildings will be placed. The area will further be equipped with the air defense system, eliminating the air attacks on our facilities.

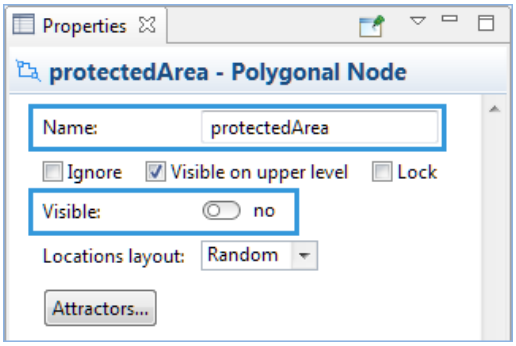
Define protected area

1. Double-click the **Polygonal Node** element in the **Process Modeling Library** palette to activate drawing mode.



2. Click on top of the rectangle to start drawing the protected area.
3. Finish drawing with a double-click on the last node.

4. Navigate to the element's **Properties**, name it *protectedArea* and set its **Visible** parameter to **no**.

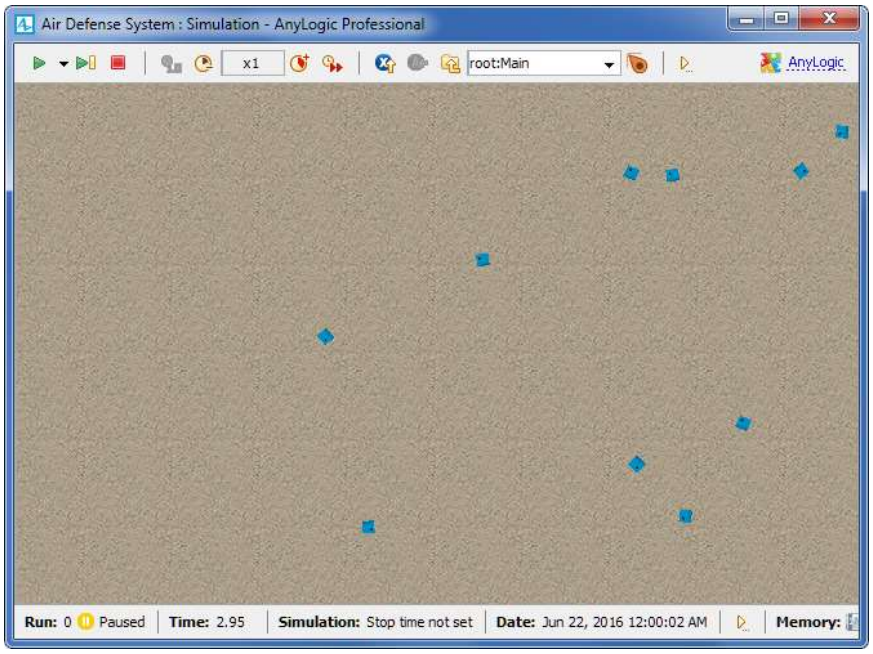


We have created a new area, now we need to locate our facilities within its boundaries.

Relocate the facilities

1. Click the buildings population on the Main to open the **Properties** view.
2. Navigate to the **Initial location** section and set the **Place agent(s)** parameter to **in the node**. The **Node** parameter will appear below.
3. Click the set by default **None** value for the **Node** parameter and select **protectedArea** from the drop-down list of available nodes.
4. Now open the properties of Main and navigate to the **Space and network** section.
5. Set the **Layout type** to **User-defined**.

Finally, we may run the model. The buildings will be located within the area defined by the polygonal node

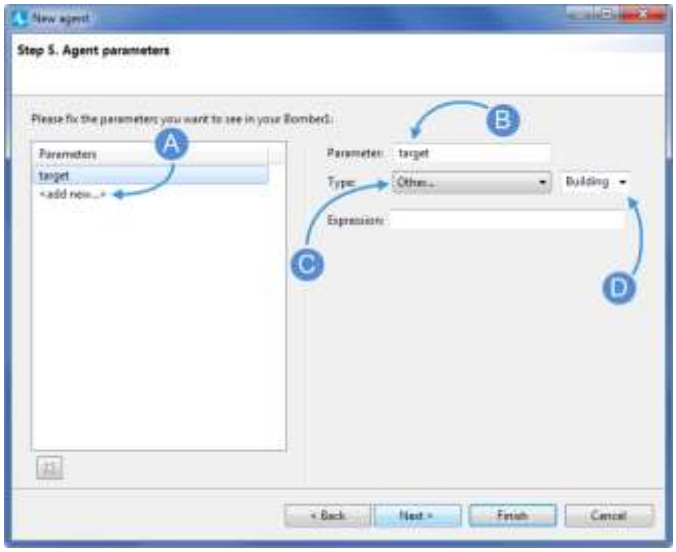


Phase 2. Adding bombers

In the previous phase we created a new model, populated it with facilities, defined animation shapes for the facilities, added 3D view and camera to observe the running model and finally added protected area. Let us start with creating a new population of agents that will represent the bombers.

Create a new agent type

1. Drag the **Agent** element from the **Agent** palette onto the Main diagram. Place it nearby the buildings population to the left of the model animation.
2. Click **Population of agents**. You will be taken to the next step.
3. Specify *Bomber* in the **Agent type name** field on the third step of the agent wizard. Click **Next**.
4. Choose the **Bomber** animation shape in the **Military** section. Click **Next**.
5. Define the population's parameter:
   a. Click **add new...** to create a parameter.
   b. Type *target* into the **Parameter** field.
   c. Set **Type** to **Other** (additional drop-down list will appear next to it).
   d. Select **Building** from the additional drop-down list of the parameter's type (This will be the target building in the bomber mission).
   Click **Next**.

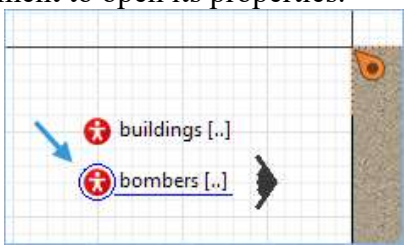


5. Select the **Create initially empty population, I will add agents at the model runtime** option.
6. Finally, click **Finish**.

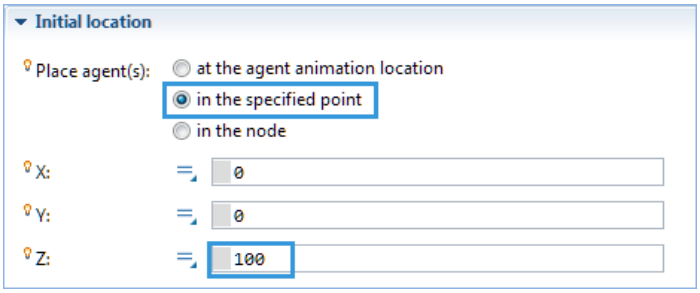
Now we will define the coordinates of the bombers' initial location.

Define initial location

1. Click the bombers population element to open its properties.



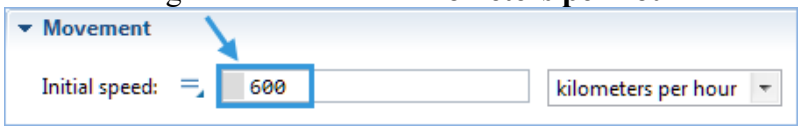
2. Navigate to the **Initial location** section of the element's **Properties**, set its **Place agent(s)** parameter to **in the specified point** to manually specify the point of appearance and then type *100* into the **Z** coordinate's field to specify the altitude the bombers will be flying into the scene at.



The next step is to define the velocity of the aircraft. It is done right here, in the same **Properties** view, just above the **Initial location** section.

Define the velocity of the aircraft

1. Navigate to the **Movement** section and set the **Initial speed** parameter to *800*, then click the speed units drop-down menu to the right of it and select **kilometers per hour**.

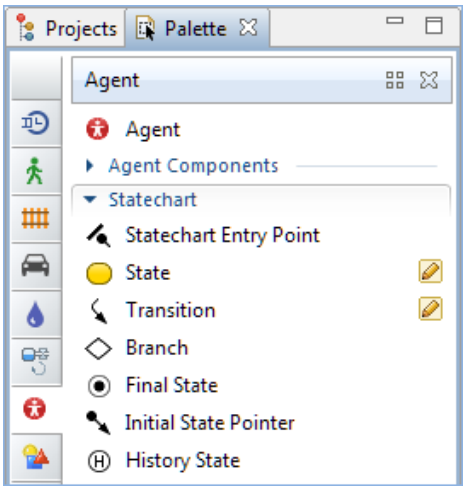


Now that we have created bombers and defined their animation, initial location and target, we can move on to create the initial version of their behavior using the statechart.

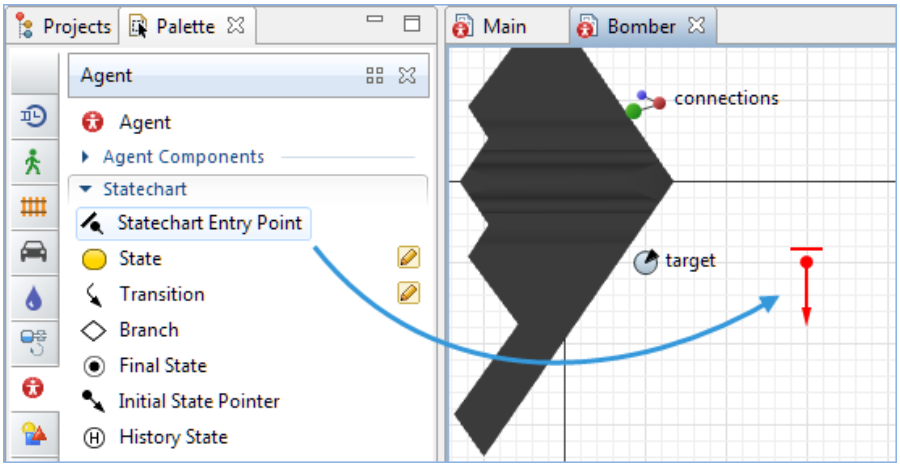
Once created, the bombers will be heading towards the buildings (it is set by the **target** parameter). We will now define the further behavior, which will make the bombers gradually get to a lower altitude and then return to the initial point where they will delete themselves from the model.

Create the behavior defining statechart

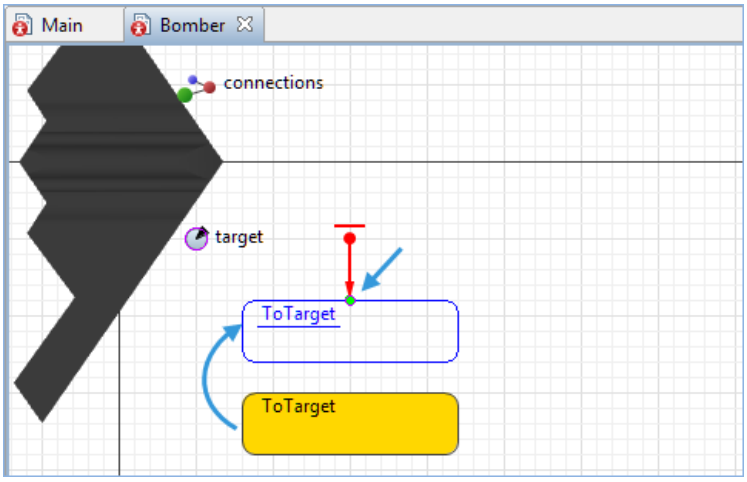
1. Double-click **Bomber** in the **Projects** view to open its diagram.
2. Switch to the **Agent** Palette to use its statechart section.



3. Drag **Statechart Entry Point** element to the graphical editor of the Bomber agent.

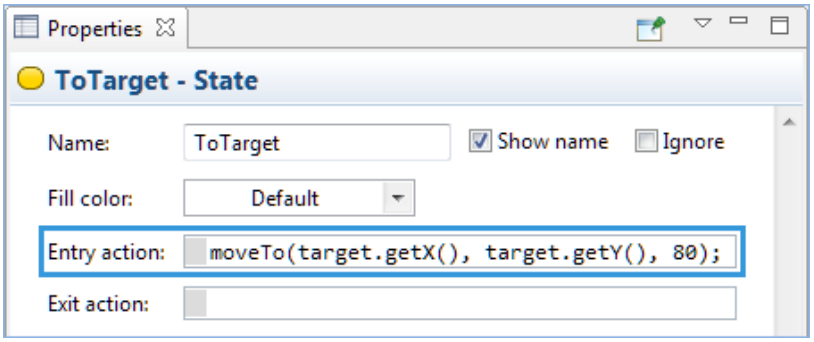


4. Double-click the **State** element to activate the drawing mode and draw a state rectangle below the previously drawn **Statechart Entry Point**.
   Name it *ToTarget* and drag it up to the entry point until you see the green dot indicating that the two elements can be connected.

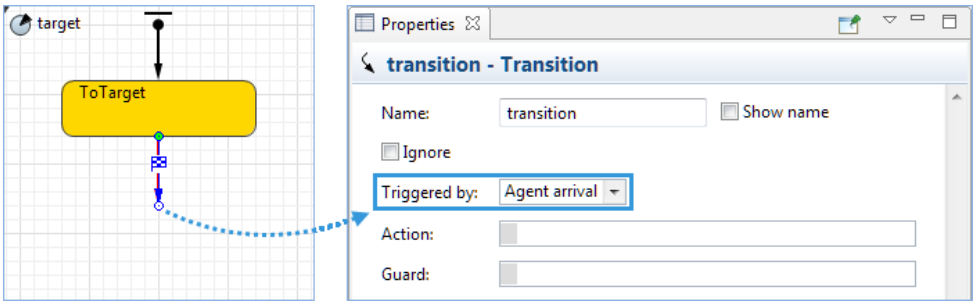


Note: You can alternatively simply click and drag the **State** element from the palette directly to the **Statechart Entry Point** element located on the Bomber diagram. The automatically drawn element can be later resized if needed.

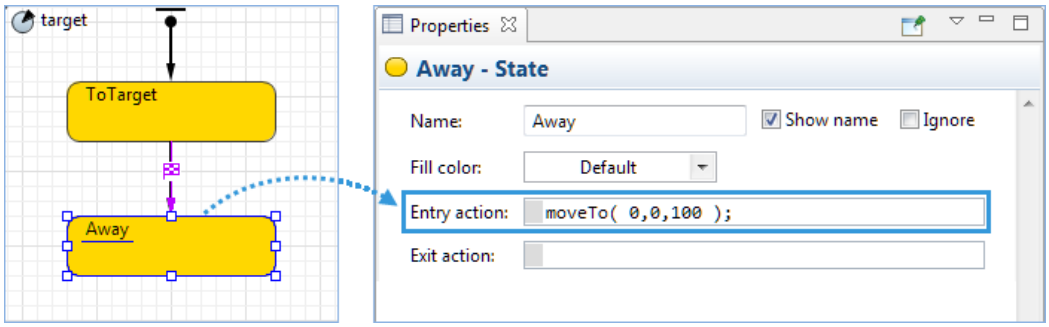
5. Modify the **Entry action** field in the **Properties** view of the *ToTarget* state. Type in the following code to define the coordinates of the bombers' location. By specifying the code we will define the altitude different from the one specified in the **Initial location** section. It will make the bombers get to a lower altitude on their way to the targets, which are represented in our case by buildings.



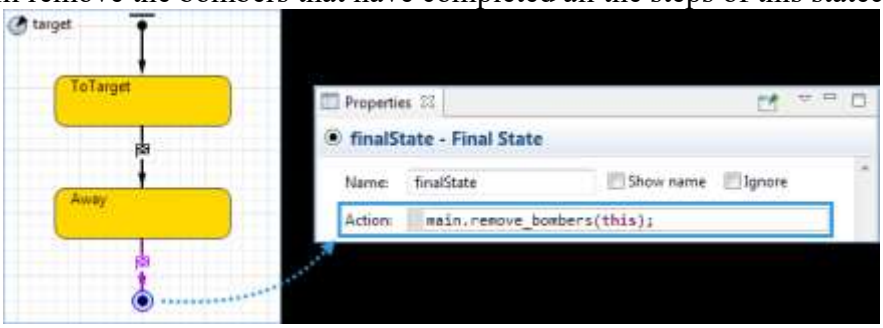
6. Drag the **Transition** element to the *ToTarget* element on the Bomber diagram and connect them. Then navigate to its properties and set the **Triggered by** parameter to **Agent arrival**.



7. Now drag another **State** element to the agent diagram and connect it to the **Transition** element. Name it *Away*. Then navigate to its **Properties** view and type the following code into the **Entry action** field, which will make the bombers get back to their previous altitude once they have reached their targets.



8. Drag another **Transition** element to the agent diagram and connect it to the *Away* element. Then navigate to its properties and set the **Triggered by** parameter to **Agent arrival**.
   The two transition elements on the statechart diagram must be identical.
9. At last drag the **Final state** element to the agent diagram and connect it to the second **State** element. Then navigate to its **Properties** view and type the following code into the **Entry action** field, which will remove the bombers that have completed all the steps of this statechart.

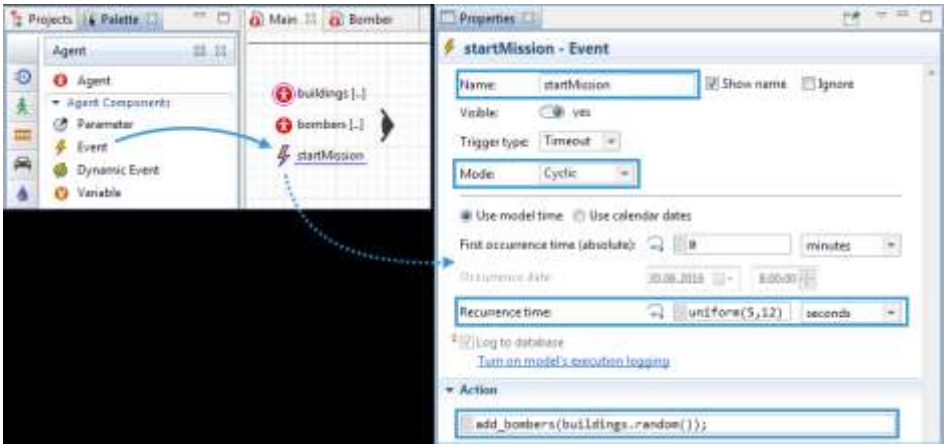


We have completed the behavior defining statechart. As you can see we call the moveTo() function in the **On enter** field of both states to initiate movement of the agent. The transition from one state to another is triggered by the agent arrival. This pattern is very common in the agent based models.
The next step is to create the mission assignment. We will do it with the help of the **Event** element.

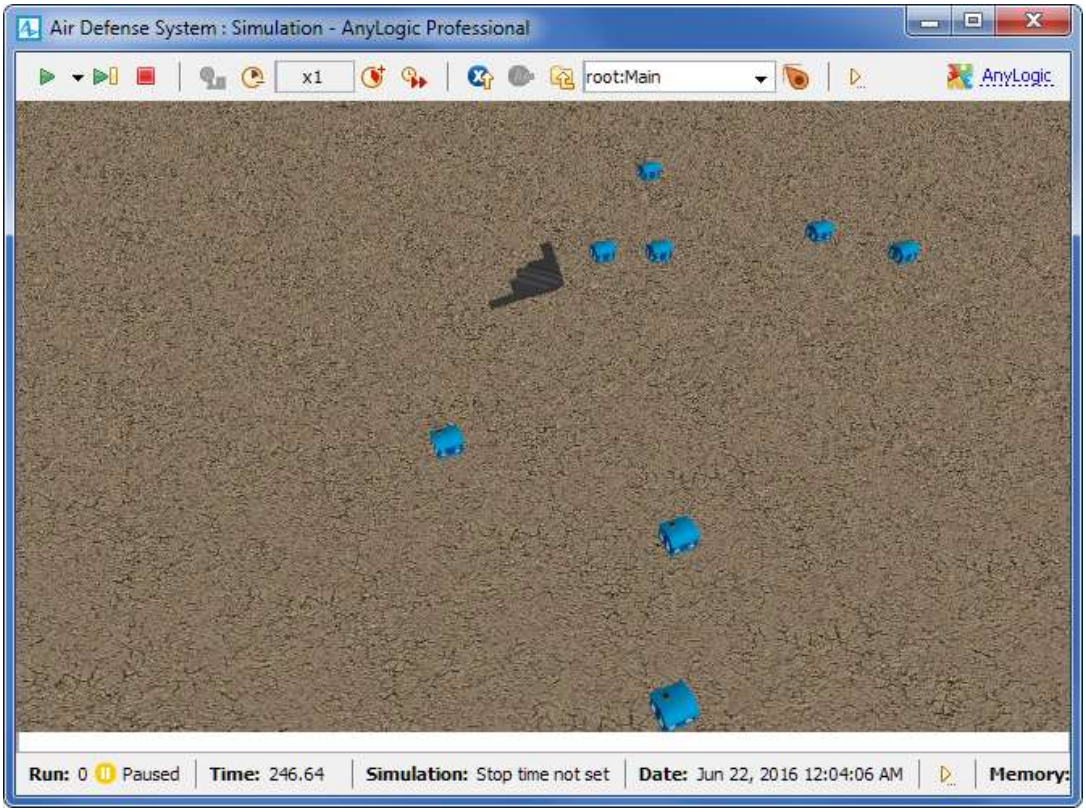
Program mission assignment

1. Switch to the Main diagram of the top level agent.
2. Drag the **Event** element from the **Agent** palette to the Main diagram and place it beside the agents populations.
3. Navigate to its **Properties** and perform the following changes:
   - Name it *startMission*.
   - Set the **Mode** parameter to **Cyclic**.
   - Set the **Reccurence time** parameter's time units to **seconds** and type uniform(5,12) into the parameter's field.
   - Type add_bombers(buildings.random()); into the field of the **Action** section.

The cyclic event periodically looks for a building without a bomber already flying to it. We use iteration across the agent population twice: in the outer loop we iterate across buildings, and in the inner loop we iterate across bombers. If such a building is found, we create a new bomber agent and assign the building to the bomber as the target (as long as the Bomber agent has one parameter **target** of type **Building**, AnyLogic generates a constructor with that parameter).



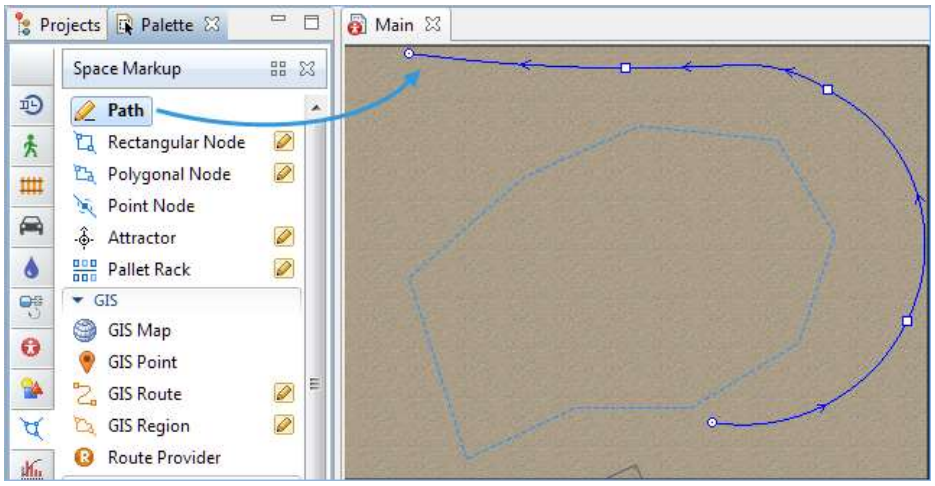
Finally, we can run the model to observe the flight of the bombers live.
Upon creation, a bomber takes direction to the target building, makes an "instant u-turn", and heads back to (0,0). So far, the bombers use straight line trajectories – this is assumed by the moveTo(x, y, z) method. We will further draw the 3D "escape trajectory" for the return route and set the bombers to follow that trajectory on their way back. We will use another version of the method: moveTo(main.exitNode);.



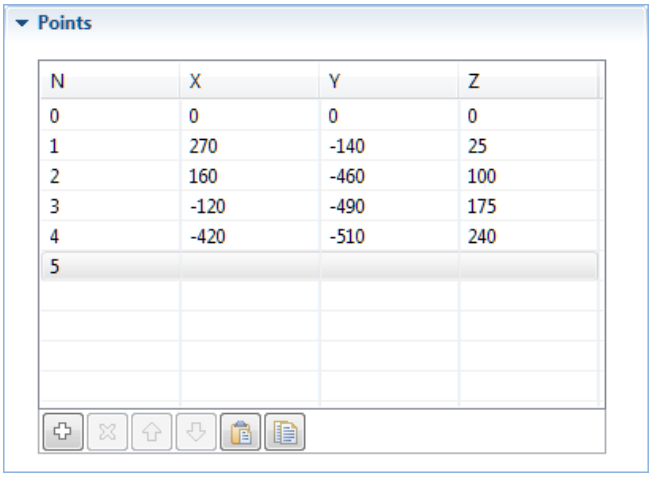
Now we will create escape route that the aircrafts will be following to avoid being hit by a missile.

Draw the 3D escape route

1. Open the **Space Markup** palette, double-click the **Path** element to activate the editing mode and draw a path, as shown in the figure.

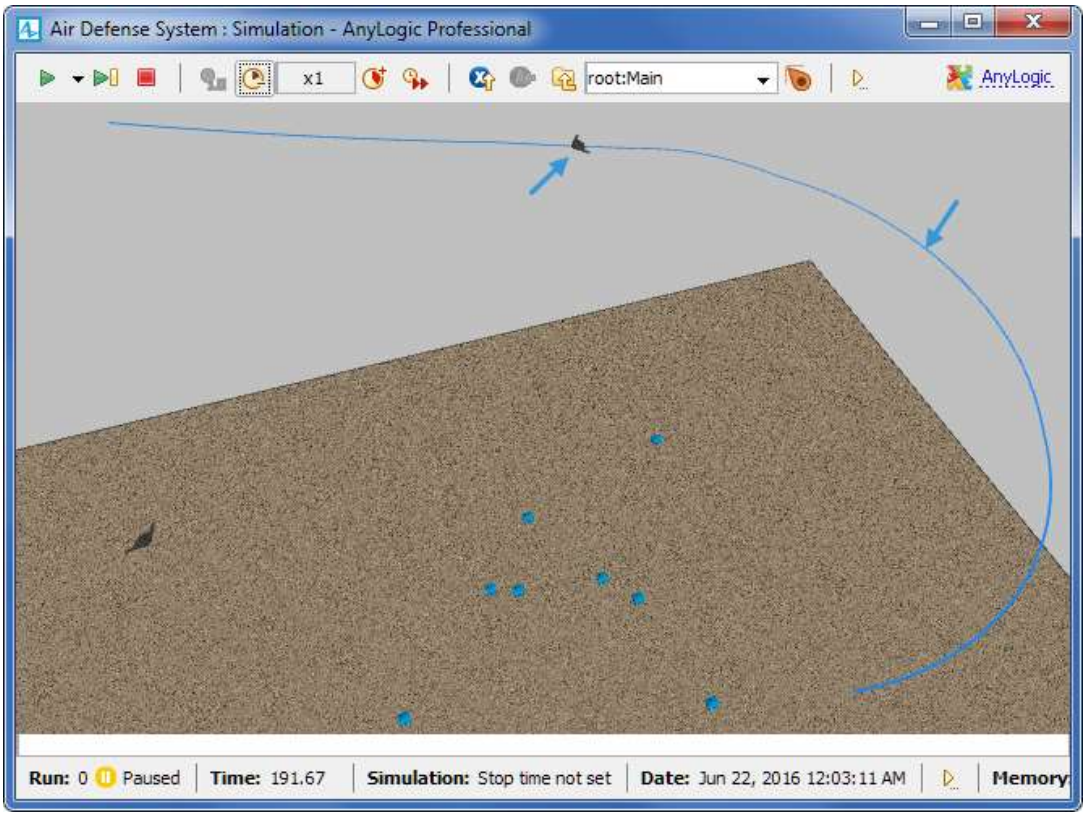


2. Navigate to its properties and name the path *escapeRoute*.
3. Set the **Z** parameter of the **Position** section to *100* (this will be the base Z-coordinate of the polyline).
4. Open the **Points** section of the path properties and modify the individual Z coordinates of the points approximately, as shown in the figure. The idea is to have the initial section of the path at about the same altitude as the bomber attack altitude.



5. Add **Point Node** from the **Space Markup** palette to the end of the path, make sure the node connects to the path. Name it *exitNode*. Set its **Z** parameter of the **Position and size** section to *340*.
6. Open the diagram of the Bomber agent, click the *Away* state to navigate to its properties and change the **Entry action** to: moveTo(main.exitNode);
   We need to put the prefix main before the exitNode because this graphical object is located not inside the Bomber agent, but one level up, in Main.

Run the model. See how the bombers return to the base using the defined by the path route.
The *escapeRoute* is visible at runtime. We will now set the **Visible** parameter of both the *escapeRoute* and the *exitNode* to **no**, to hide both of them.

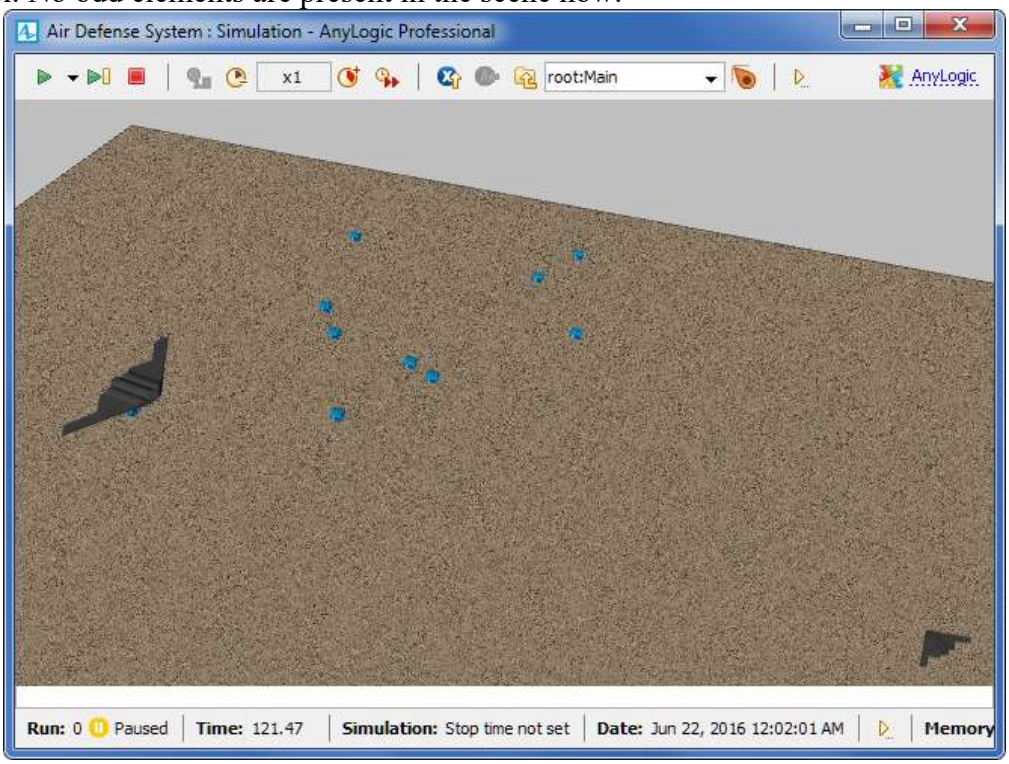


Hide escapeRoute during runtime

1. Click the *escapeRoute* to select it.
2. Navigate to its **Properties** view and click the **Visible** parameter's toggle button to switch the visualization state to **no**.
   The *escapeRoute* will not be visible during runtime now.

In the same way hide the *exitNode*.

Run the model. No odd elements are present in the scene now.



Phase 3. Adding bombs

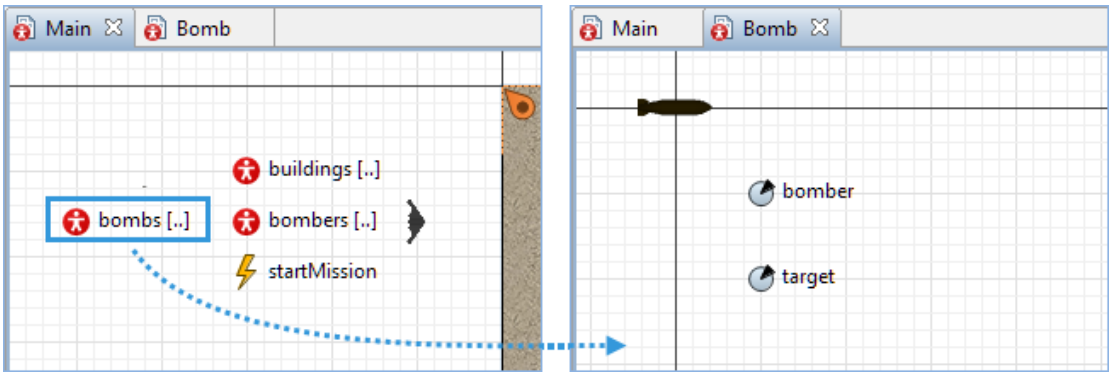
In the previous phases we created a new model, populated it with facilities, bomber aircrafts, defined behavior of aircrafts and their mission, specified animation shapes, added 3D view, camera and protected area.

Let us start with creating the Bomb agent type and program its interaction with the building:

Create Bomb agent

1. Drag the **Agent** element from the **Agent** palette onto the Main diagram. Place it beside the bombers population to the left of the model animation.
2. Click **Population of agents**. You will be taken to the next step.
3. Specify *Bomb* in the **Agent type name** field on the third step of the agent wizard. Click **Next**.
4. Choose the **Bomb** animation shape in the **Military** section. Click **Next**.
5. Create two parameters:
   - Click **add new...** to create a parameter.
   - Type *target* into the **Parameter** field.
   - Set **Type** to **Other** (additional drop-down list will appear next to it).
   - Select **Building** from the additional drop-down list of the parameter's type (This will be the target building in the bomber mission).
   - In the same way create parameter **bomber** of type **Bomber**.
   - Click **Next**.
6. Select the **Create initially empty population, I will add agents at the model runtime** option.
7. Finally, click **Finish**.

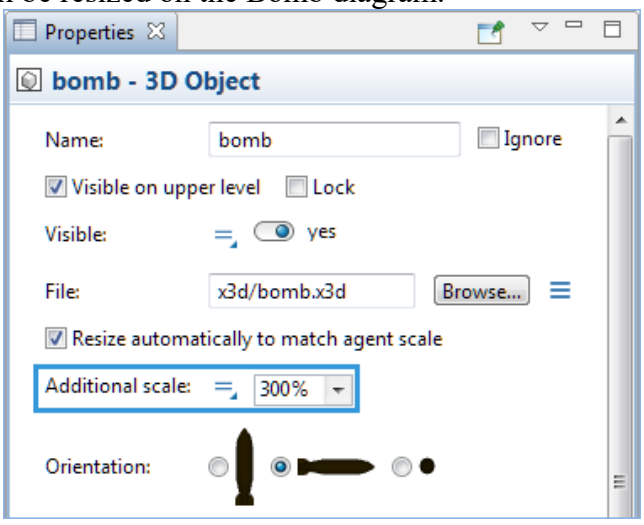
The new agent population will be created with two parameters inside. You can observe them by double-clicking the bombs population on the Main diagram.



Now we need to adjust the scale of the animation shape to perfectly fit it into our environment.

Adjust scale of the animation shape

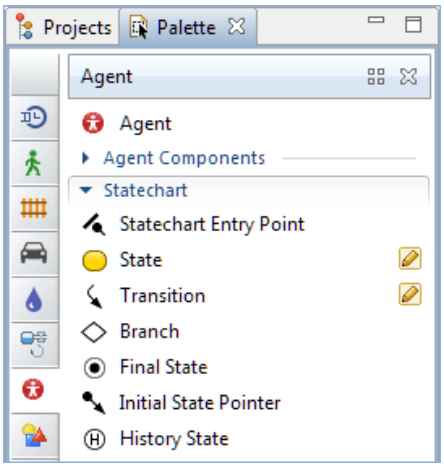
1. Click the black 2D animation shape of the bomb in the previously opened Bomb diagram to open its **Properties** view.
2. Set the **Additional scale** parameter to 300%.
   The animation shape will be resized on the Bomb diagram.



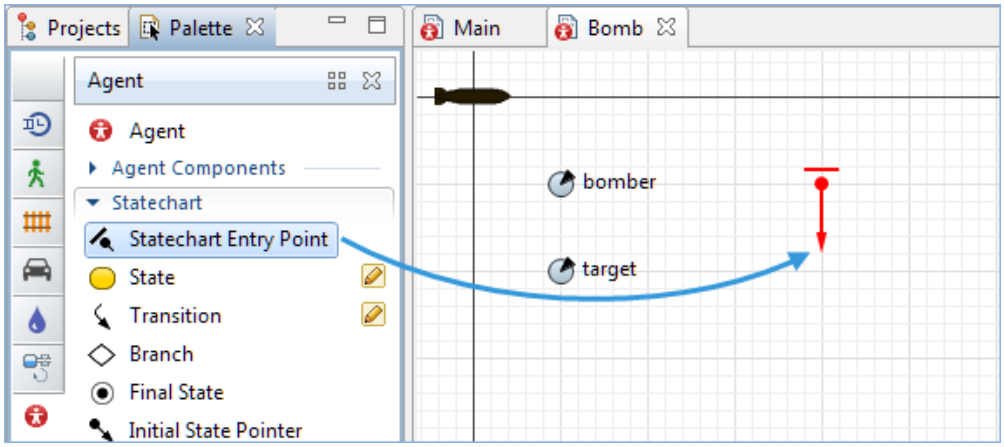
Now we will make a bomb and the targeted building interact with each other by means of a statechart. The bomb will send the "You are destroyed" message to the building on hitting it, destroying itself immediately afterwards.
Start creating the statechart in the Bomb diagram that you are currently in.

Create the interaction between the bomb and the building

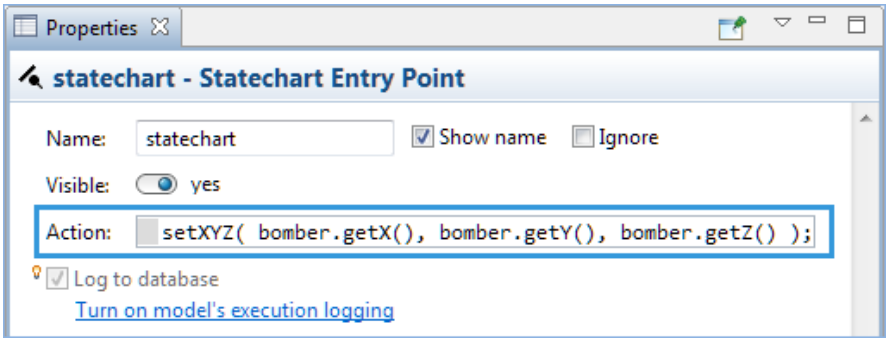
1. Switch to the **Agent** palette to use its statechart section.



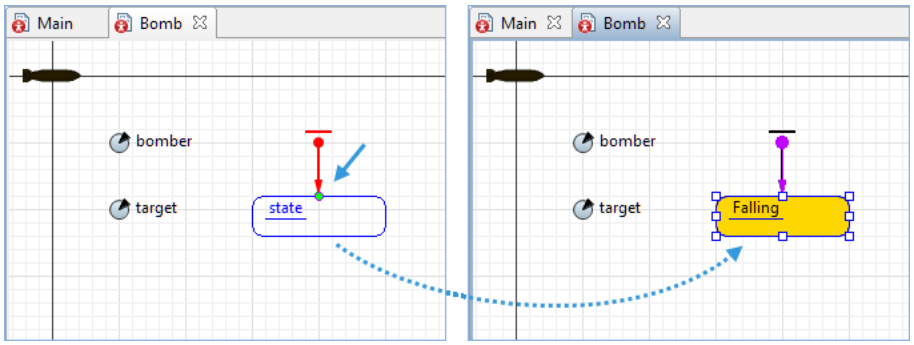
2. Drag **Statechart Entry Point** element to the graphical editor of the Bomb agent.



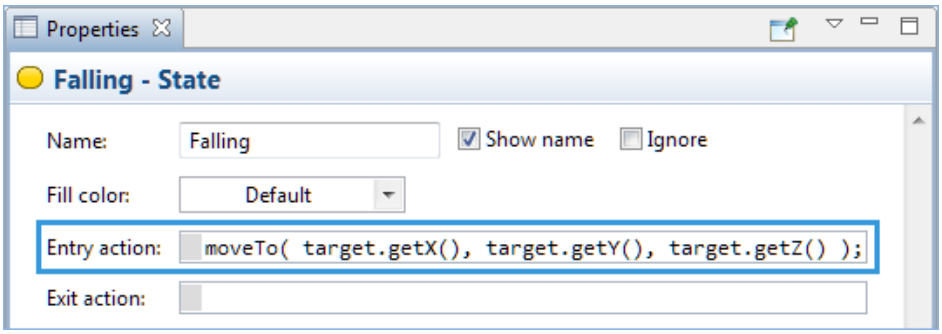
3. Modify the **Action** parameter in the **Properties** of the **Statechart Entry Point** element by typing in the following code:
   setXYZ(bomber.getX(), bomber.getY(), bomber.getZ());



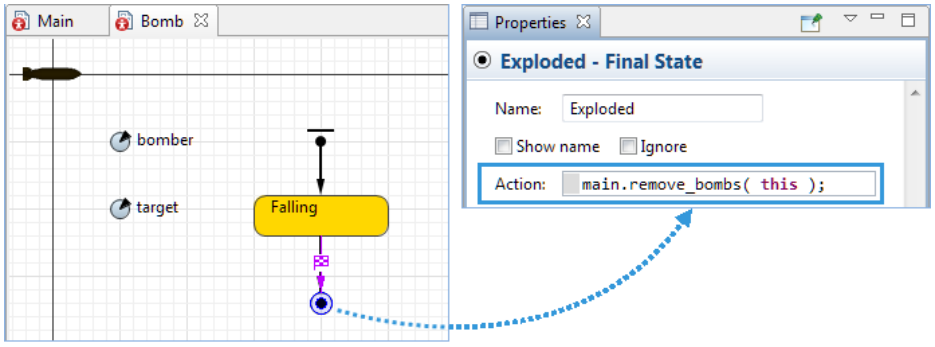
4. Now click and drag the **State** element from the **Statechart** section of the **Agent** palette up to the previously drawn **Statechart Entry Point** element on the Bomb diagram. Release the mouse button when the green dot appears. It indicates that the two elements will be connected. Name it *Falling*.



5. Type the following code into the **Entry action** parameter of the element's **Properties**:
   moveTo(target.getX(), target.getY(), target.getZ());



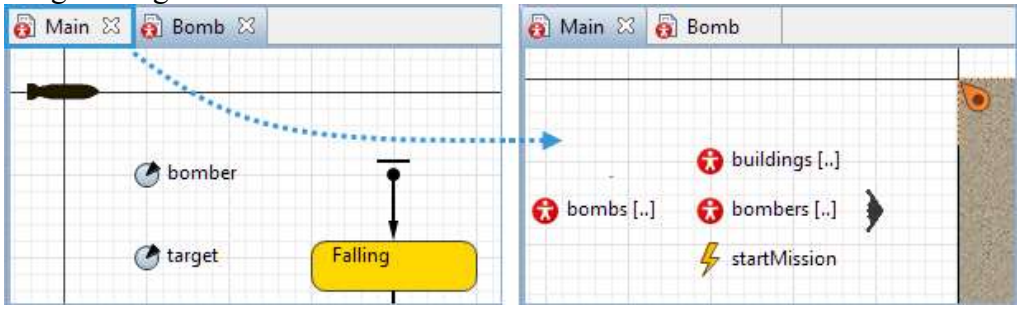
6. Now drag the **Transition** element to the *Falling* element on the Bomb diagram and connect them. Navigate to its properties and set the **Triggered by** parameter to **Agent arrival**.
7. Drag the **Final state** element to the agent diagram and connect it to the **Transition** element. Type the following code into the **Action** field of its **Properties** to destroy the current bomb once the message has been delivered:
   main.remove_bombs(this);



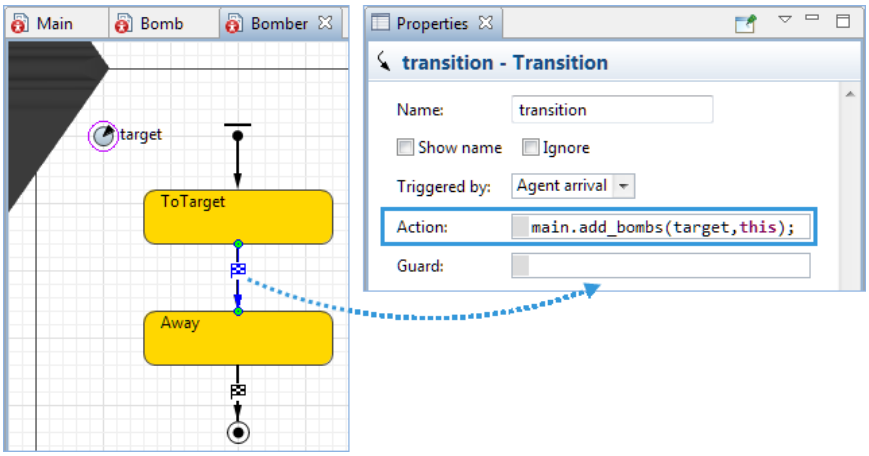
We have completed the statechart and the only missing piece that is left is the bomber's decision to drop a bomb. We will do it exactly upon arrival to the point directly above the target building (in the action of the transition going from *ToTarget* and Away states).

Define bomb dropping

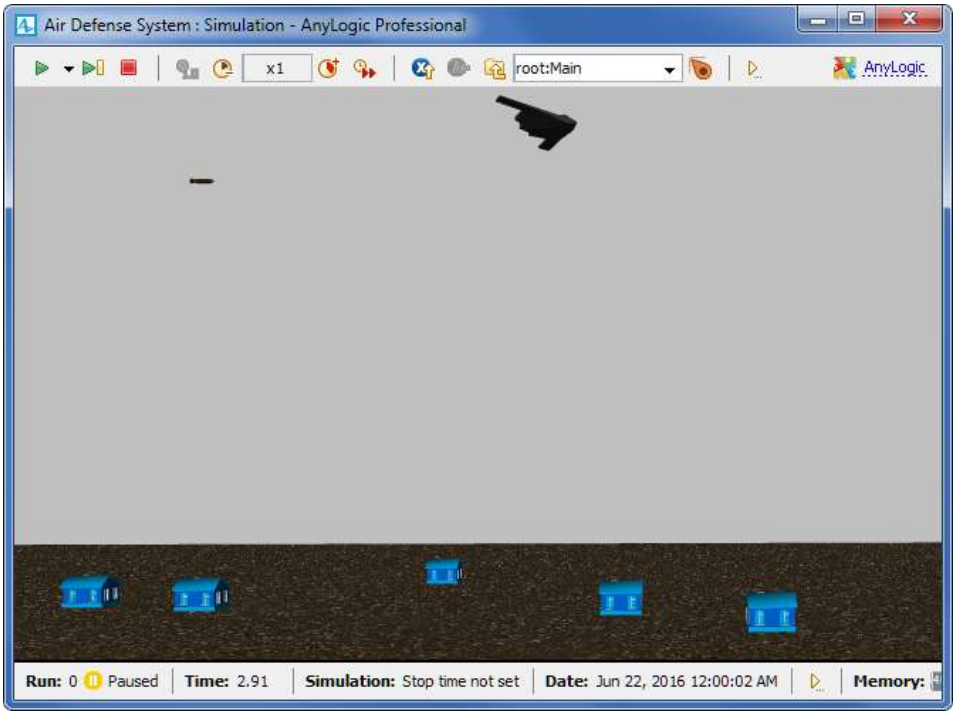
1. Click the Main tab to switch to the top level agent diagram. We will further navigate to the Bomber agent diagram from Main.



2. Double-click bombers to open its diagram.
3. Click the **Transition** element to open its properties and modify the **Action** field by typing in the following code:
   main.add_bombs(target,this);
   Note: On creating a new bomb, the bomber passes itself (this) and the target building to the bomb constructor parameters.



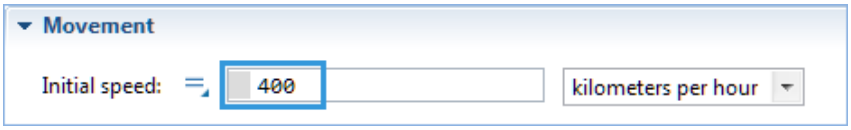
Let us run the model now and watch the bombs being dropped on the buildings in the 3D view mode.



As you can see, the bombs are falling down unnaturally slowly, that is because the initial speed of the bombs is set to the default 10 meters per second.

Define the speed of the bombs

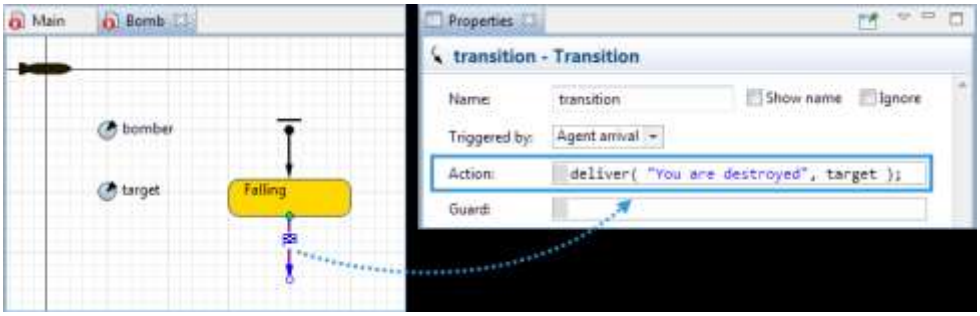
1. Click the bombs population on the Main diagram to navigate to its **Properties**.
2. Set the **Initial speed** parameter of the **Movement** section to *400*, then click the speed units drop-down menu to the right of it and select **kilometers per hour**.
3.



If you run the model now, you will see that bombs are falling faster in the 3D view mode.
The next step is to show the result of a bomb hitting the building, i.e. to implement the building's reaction to the bomb explosion. The state of being hit will be transferred to the building by a message inside the bomb.

Transfer the message from the bomb to a building

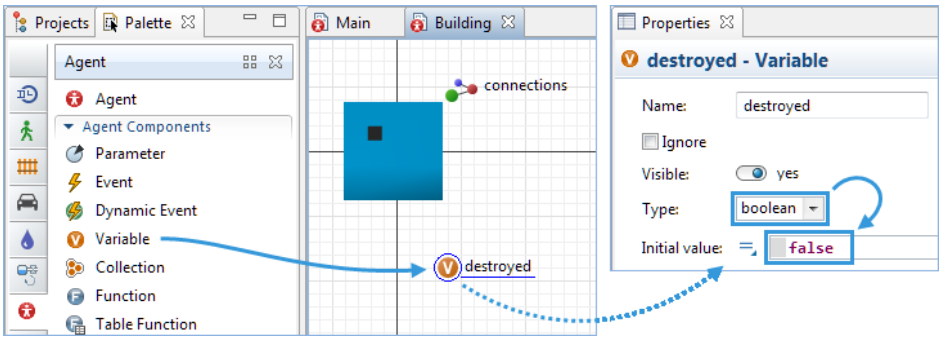
1. Double-click the bombs population on the Main to open its agent's diagram.
2. Click the **Transition** element on the opened Bomb diagram and type the following code in the **Action** field of its properties:
   Type deliver("You are destroyed", target);
   We are using the deliver()method, which delivers the message instantly (within the same event) instead of the send() method, which does it in a separate event (still at the model time though). This is because the message sent by the send() method will be discarded once the sender agent ceases to exist.



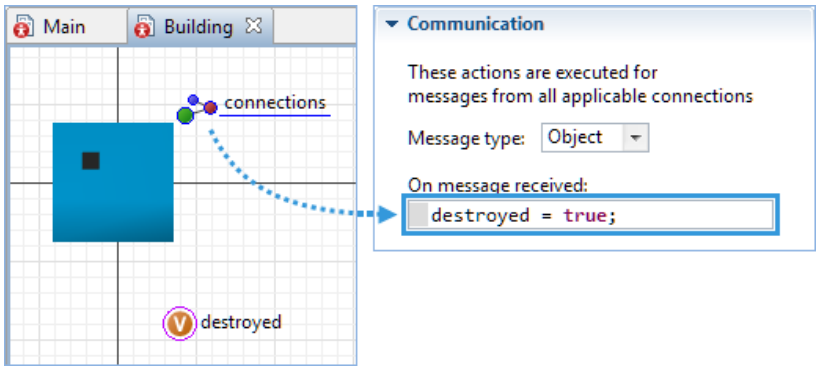
Now we will add a Boolean flag destroyed and some animation to the building hit by the bomb.

Modify the Building agent

1. Navigate to building's properties by double-clicking the buildings on the Main.
2. Click and drag the **Variable** element from the **Agent Components** section of the **Agent** palette onto the diagram and name it *destroyed*.
   Navigate to its **Properties** view and perform the following modifications:
   - Set **Type** parameter to **boolean**
   - Type false into the **Initial value** field

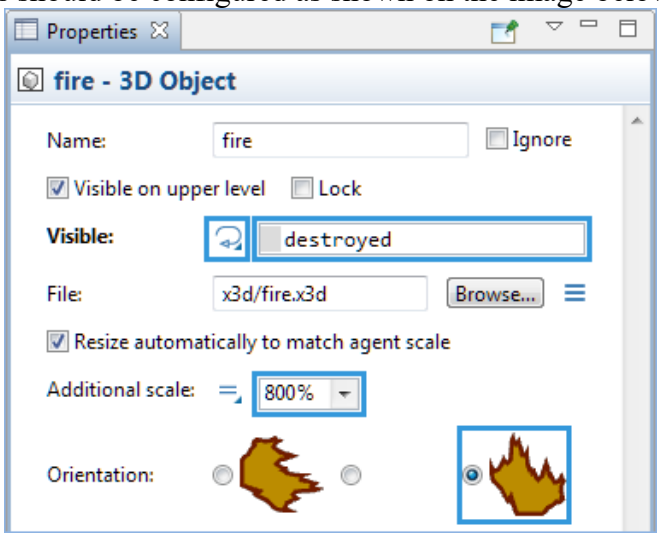


3. Click the connections element on this diagram to open its properties and type the following code into the **On message received** field of the **Communication** section:
   destroyed = true;
   We are not analyzing the content of the message because a bomb is the only message source for our building for now.

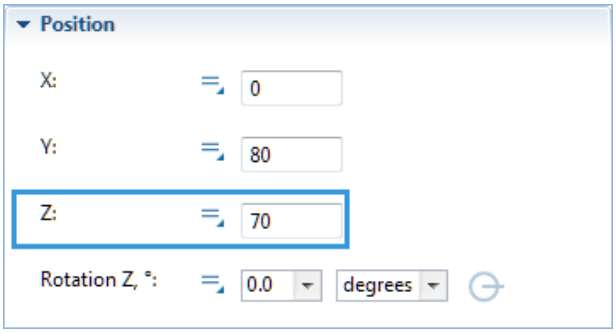


4. Navigate to the **3D objects** palette and drag **Fire** object to the Building diagram. A pop-up message will appear "3D Object's size will change automatically to match with the agent's scale". Click **Yes**.
5. Navigate to **Properties** view of the object and perform the following changes:
   - Click the icon to the right of the **Visible** parameter to switch to the code field, then type destroyed into it.
     It will make the fire appear around the house once it has been bombed.
   - Set the **Additional scale** parameter to *800*.
   - Set the **Orientation** parameter to vertical.

   In the end the properties should be configured as shown on the image below.



6. Move on to the Position section and set the Z parameter to 70.

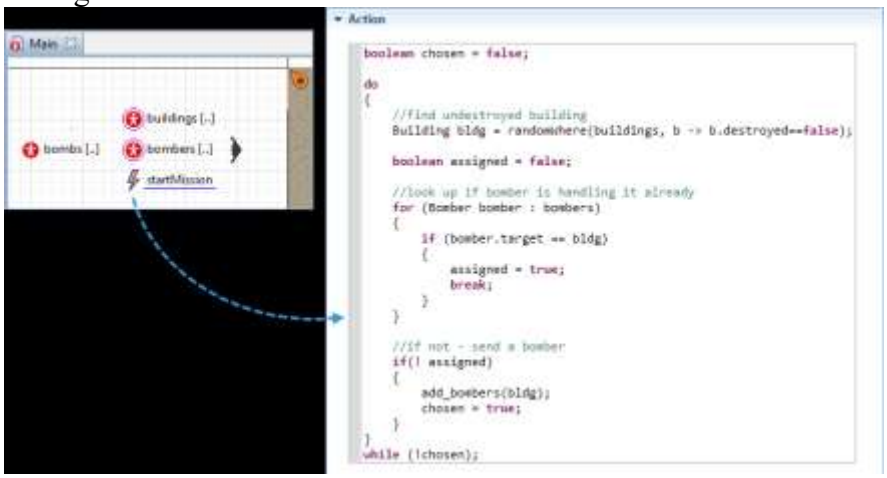


Run the model to observe the building catching on fire when hit.

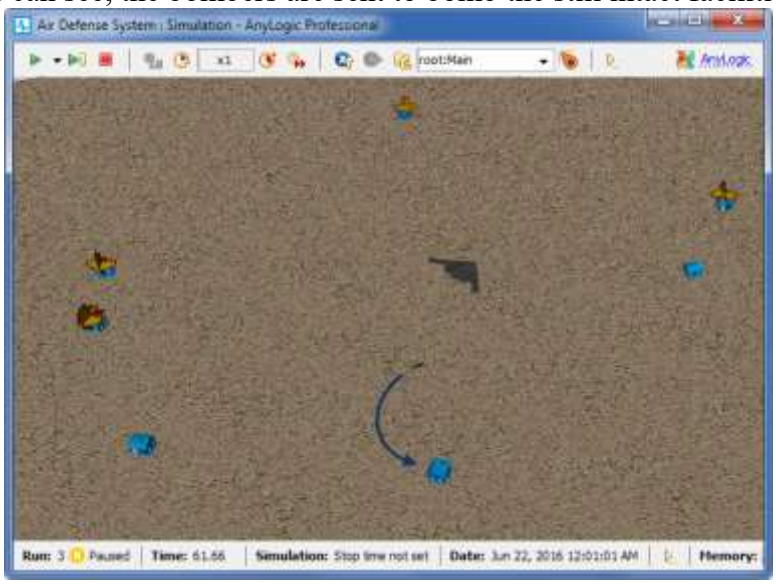
You must have noticed that, although all buildings are destroyed by the very first bomber attack, the new bombers are, nevertheless, sent to the target area. This happens because the mission assignment is done regardless of the state of the target assets. We will fix it now.

Modify mission assignment

1. Navigate to the Main and click the startMission event to open its **Properties** view.
2. Type in the following code into the box of the Action section:



Run the model. As you can see, the bombers are sent to bomb the still intact facilities.



Phase 4. Adding air defense system

In the previous phases we created a new model, populated it with facilities, bomber aircrafts, defined behavior of aircrafts and their mission, specified animation shapes, added 3D view, camera protected area, added bombs to our model and learned how to destroy facilities with them.
Let us start creating our defense system.

Create the first radar

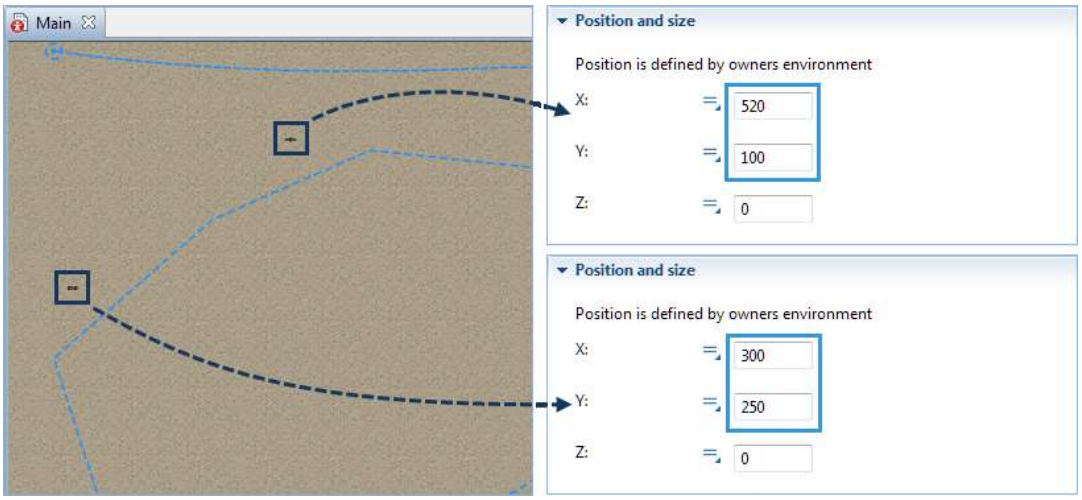
1. Drag the **Agent** element from the **Agent** palette onto the Main diagram. Place it beneath the bombers population to the left of the model animation.
2. Click **A single agent**, because we do not plan to have a population of radars. We will create two individual radars in the model and specify individual parameters to each of them.
3. Specify *Radar* in the **Agent type name** field and *radar1* in the **Agent name** field. Click **Next**.
4. Choose the **Patriot** animation shape in the **Military** section. Click **Next**.
5. Create a *range* parameter:
    - Click **add new...** a new parameter will be created.
    - Type *range* into the **Parameter** field.
    - Leave the **Type** set to **double** and specify *1000* in the **Expression** field (1000 pixels equals 1 km).
    - Click **Next**.
    - Uncheck the **Apply random layout** checkbox if it is checked.
    - Click **Finish**.

Our first radar is created. Proceed to creating the second one.

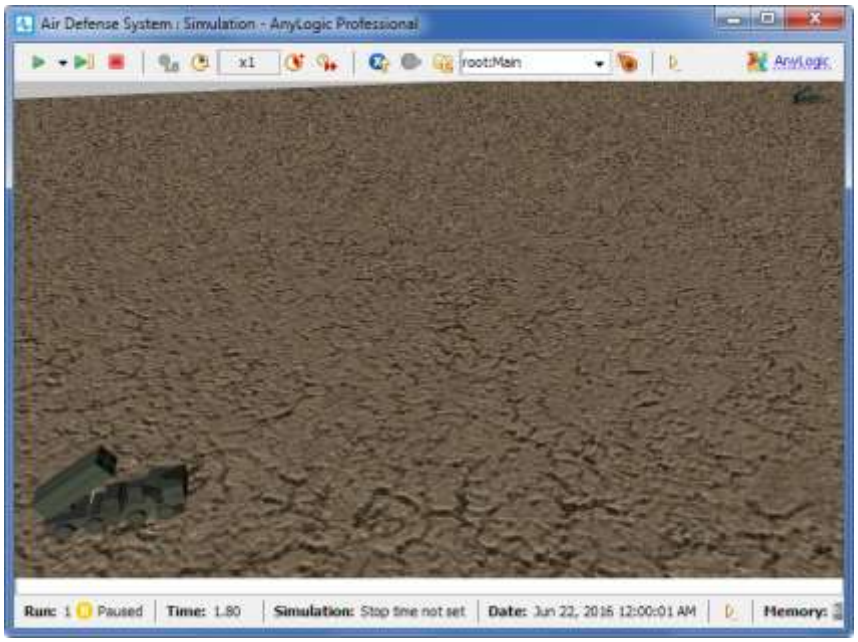
Create the second radar

1. Drag the **Agent** element from the **Agent** palette onto the Main diagram. Place it beneath the radar1 agent.
2. Click **A single agent**.
3. Select the **I want to use an existing agent type** option. Click **Next**.
4. Select **Radar** as the agent type that will be used. Specify *radar2* in the **Agent name** field. Click **Finish**.

Both our radars are ready. Now drag their animations on the Main to the protected area and place them at (300, 350) and (520, 100) respectively as for instance. You may place your radars as you wish by either dragging them over the graphical diagram or specifying the positions in the **Position and size** section of the agent's properties.



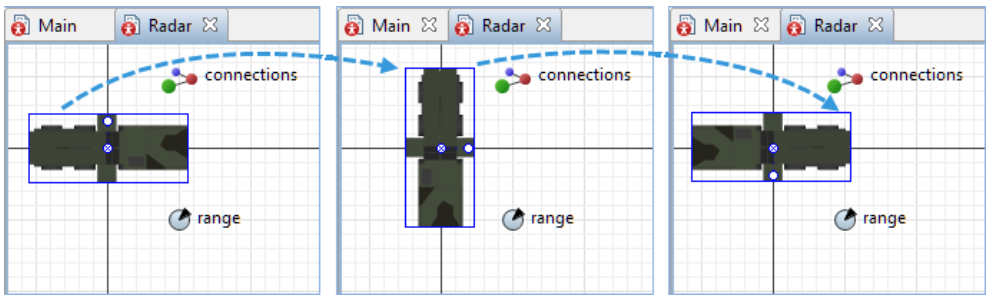
Run the model and make sure the radars are located on the bomber's route. You will see that the shapes are not headed towards the bombers.



Turn the animation of the radars to make them head towards the bombers.

Adjust the radars direction

1. Open radar1 diagram by double-clicking it on the *Main*.
2. Click the **patriot** animation to select it.
3. Hover your mouse over the top circle on the shape. The mouse cursor will change to a rotation icon.
4. Click and hold the mouse button. Drag it clockwise to rotate the shape 180 degrees to make the radar head the opposite way.



Now we will create the population of agents representing the missiles.
It will be very similar to the bomb with the limited lifetime from launch to explosion and it will contain two parameters:

- the radar parameter of type Radar.
- the target parameter of type Bomber.

The missile will be periodically adjusting its trajectory to catch up the moving target.

Create the Missile agent type

1. Drag the **Agent** element from the **Agent** palette onto the Main diagram. Place it to the left of the two radars populations.
2. Click **Population of agents**. You will be taken to the next step.
3. Specify *Missile* in the **Agent type name** field. Click **Next**.
4. Choose the **Missile** animation shape in the **Military** section. Click **Next**.
5. Create two parameters:
    - Click **add new...** to create a parameter.
    - Type *radar* into the **Parameter** field.
    - Set **Type** to **Other** (additional drop-down list will appear next to it).

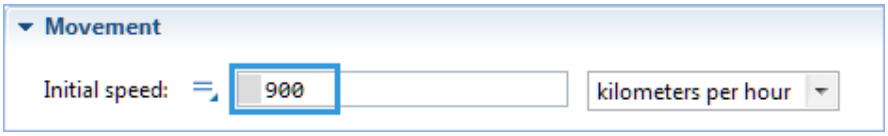
- Select **Radar** from the additional drop-down list of the parameter's type (This will be the target building in the bomber mission).
- In the same way create parameter **target** of type **Bomber**.
- Click **Next**.

6. Select the **Create initially empty population, I will add agents at the model runtime** option.
7. Finally, click **Finish**.

We have created the new population. Now we will define certain parameters of the agents within this population.

Specify the speed of the missiles

1. Click the missiles population on the Main diagram to open its **Properties**.
2. Navigate to the **Movement** section and set the **Initial speed** parameter to *900*. Make sure that the speed is expressed in **kilometers per hour**.



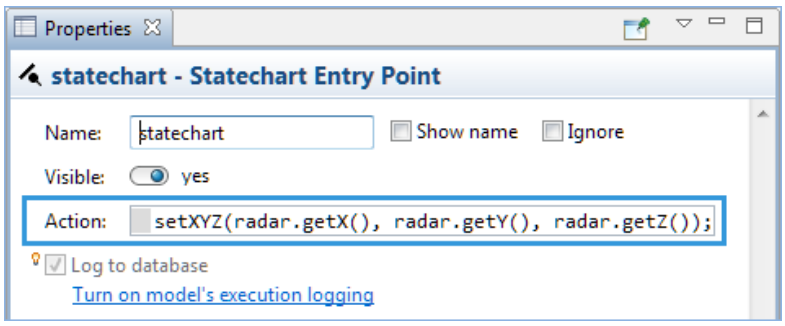
Now we should open the Missile agent diagram to adjust the animation scale and define the agent's behavior.

Adjust scale of the animation shape

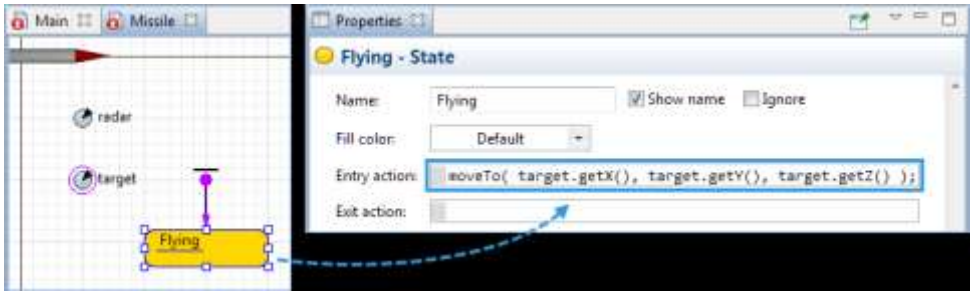
1. Open the Missile agent diagram by double-clicking the missiles population element.
2. Click the 2D animation shape of the missile to open its **Properties** view.
3. Set the **Additional scale** parameter to 400%. It will make the missile visible and easy to observe during model simulation in the 3D mode.

Define the behavior of the missiles

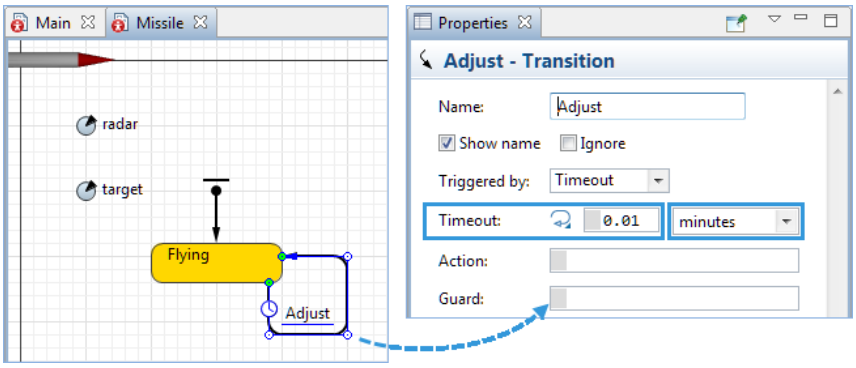
1. Drag **Statechart Entry Point** element to the graphical editor of the Missile agent. Navigate to its **Properties** view and type the following code into the Action field:
   setXYZ(radar.getX(), radar.getY(), radar.getZ());



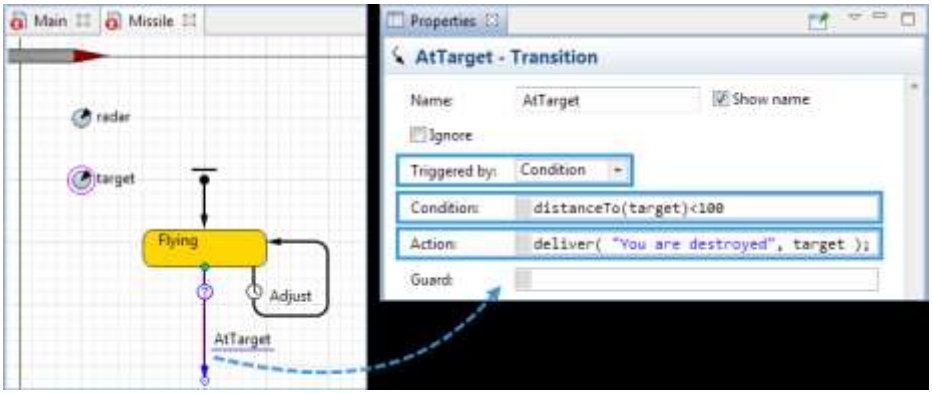
2. Now click and drag the **State** element from the **Agent** palette to the **Statechart Entry Point** element on the Bomb diagram. Name it *Flying*.
   Type the following code into the **Entry action** field:
   moveTo(target.getX(), target.getY(), target.getZ());



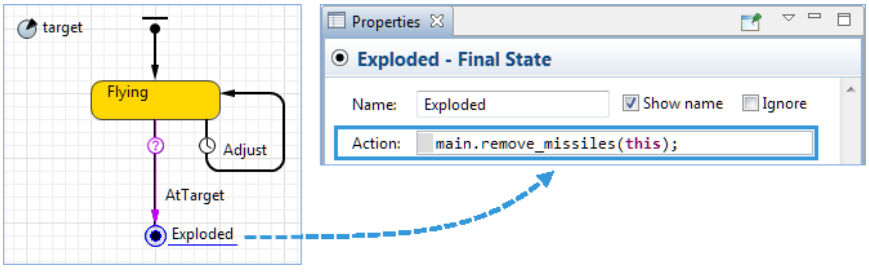
3. Now drag the **Transition** element to the diagram and connect it to the Flying state as shown on the picture below. Name it *Adjust* and set its **Timeout** parameter to *0.01 minutes*.
   It makes sense to discuss the missile trajectory adjustment. The *Adjust* transition is executed every *0.01* minutes. It has no action, but it makes the missile statechart re-enter the *Flying* state. Therefore, the entry action of the *Flying* state is also executed every *0.01* minutes, making the missile head to the current position of the bomber.
   This kind of navigation is far from being ideal, but it will give us a nice curved trajectory of the missile. The condition of the *AtTarget* transition will also be re-evaluated each *0.01* minutes. The same is true for the *OutOfRange* transition, which is responsible for checking if the missile leaves the radar coverage area and can no longer be guided.



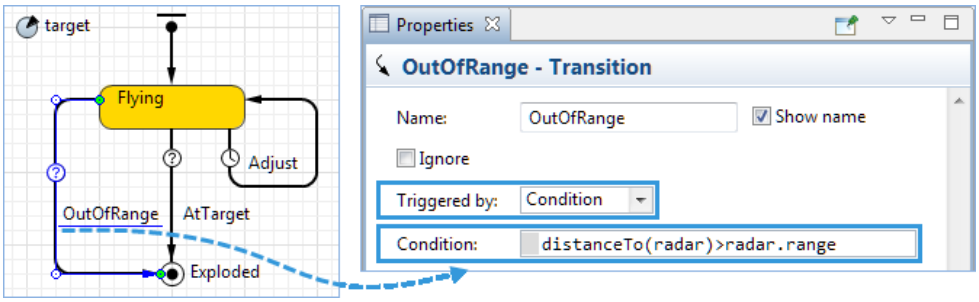
4. Now drag the **Transition** element to the *Flying* element on the Missile diagram and connect them. Name it *AtTarget* and navigate to its properties and perform the following changes:
   - Set the **Triggered by** parameter to **Condition**.
   - Type the following code into the **Condition** field to specify the distance to the bomber at which the missiles will explode:
     distanceTo(target)<100
   - Type the following code into the **Action** field to specify the message the missile will transfer to the bomber:
     deliver("You are destroyed", target);



5. Drag the **Final state** element to the statechart that we are designing and connect it to the *AtTarget* element. Name it *Exploded* and type the following code into the **Action** field of its **Properties**:
   main.remove_missiles(this); - to destroy teh current missile once the message has been delivered



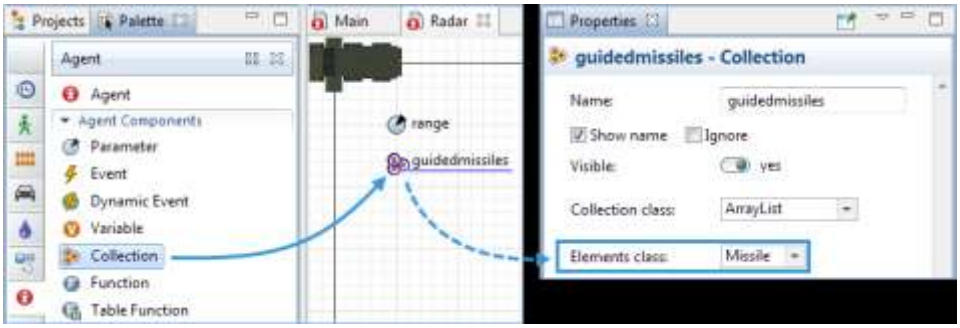
6. Now drag the another **Transition** element to the *Flying* element. Name it *OutOfRange* and navigate to its properties and perform the following changes:
    - Set the **Triggered by** parameter to **Condition**.
    - Specify the distance to the bomber at which the missile will explode by typing the following code into the **Condition** field:
      distanceTo(radar)>radar.range



The next step is to program the launch and the guiding processes of the missiles

Program the missile launch and guiding

1. Create a Collection inside the Radar. Name it *guidedmissiles* and set the **Elements class** to **Missile**.
   This collection will contain the missiles currently guided by the radar.



2. Click somewhere within the Radar diagram's empty space to open the agent's properties and add the following code to the **On step** field:

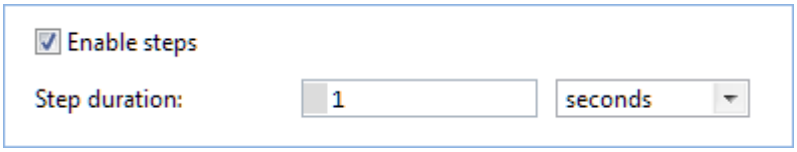
On step:

```
//for all bombers in the air
for( Bomber b : main.bombers ) {
    //if can't have more engagements, do nothing
    if( guidedmissiles.size() >= 2 )
        break;
    //if within engagement range
        //already engaged by another missile?
    if( distanceTo( b ) < range ) {
        boolean engaged = false;
        for( Missile m : main.missiles ) {
            if( m.target == b ) {
                engaged = true;
                break;
            }
        }
        if( engaged )
            continue; //proceed to the next bomber
        //engage (create a new missile)
        Missile m = main.add_missiles( this, b );
        guidedmissiles.add( m ); //register guided missile
    }
}
```

The radar iterates across all the bombers in the model. If a bomber without a missile is found, a new missile is launched. Zone scanning is done on every time step, which is defined in the **Properties** view of the top level agent.

Enable steps

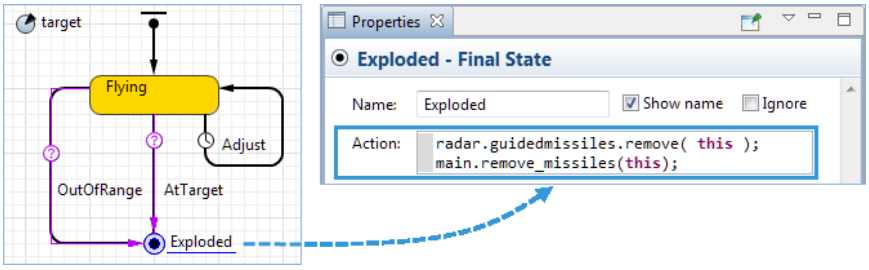
1. Navigate to the properties of the Main diagram and select the **Enable steps** option in the **Space and network** section. Leave the set by default *1 seconds*.



Now we need to let the radar know that the missile reached the aircraft and exploded so that the radar could launch and guide another missile.

Adjust missile behavior

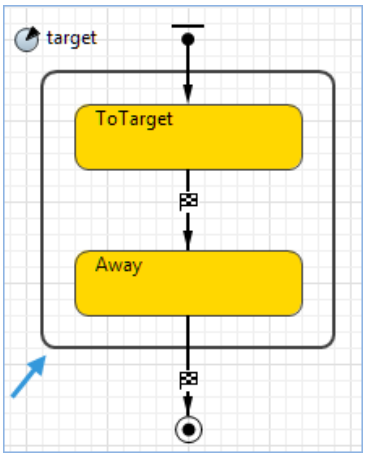
1. Navigate to the Main diagram and double-click missiles to open the agent's diagram. Add one more line to the **Action** parameter of the *Exploded* statechart element:
   radar.guidedmissiles.remove(this); - to deregister itself from the radar



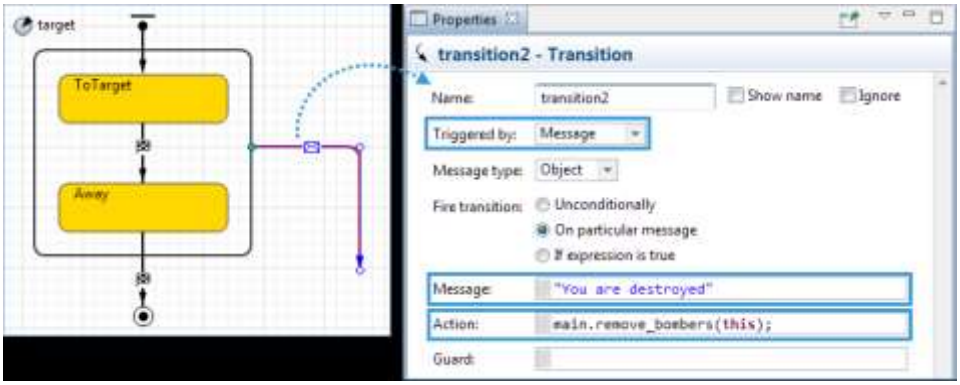
With the final step of this tutorial we will program the bomber's reaction to the missile explosion.

Modify Bomber's statechart

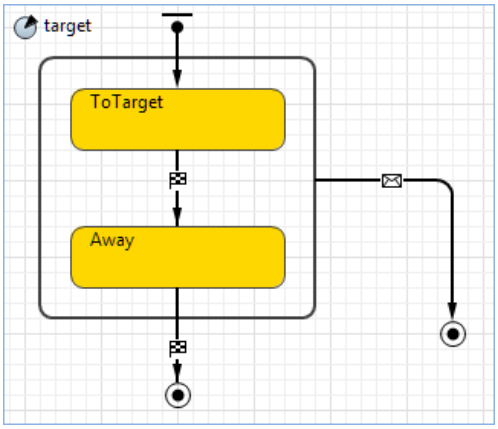
1. Double-click the **State** element in the **Statechart** of the **Agent** palette to activate its drawing mode.
2. Click and drag your mouse over the drawn statechart to enclose it into the new element as shown on the picture below.



3. Now drag the **Transition** element to the diagram and connect it to the previously drawn **State** element. Navigate to its properties and:
   - Set the **Triggered by** parameter to **Message**.
   - Specify "You are destroyed" in the **Message** field.
   - Add main.remove_bombers(this); to the **Action** field.



4. Finally add the **Final state** element to the agent diagram and connect it to the previously added **Transition** element.
   The state should look like this:



Run the model. You will see that a missile explodes on reaching the attack distance. The bombers disappear once the missiles heading to them explode.