How to Build a Combined Agent Based / System Dynamics Model in AnyLogic

Tutorial

Contents

Introduction

AnyLogic allows you to build a simulation model using multiple methods: System Dynamics, Agent Based and Discrete Event (Process-centric) modeling. Moreover, you can combine different methods in one model: put agents into an environment whose dynamics is defined in SD style, use process diagrams or SD to define internals of agents, etc, etc. Any kind of mixed architecture is possible due to flexible object-oriented AnyLogic modeling language.

The choice of model architecture (how to partition the model into components, what to aggregate, which granularity to use, what behavior is best mapped to a process diagram, and what – to a statechart, etc) belongs mainly to the domain of art and intuition of the modeler and is outside the scope of this short tutorial. The goal of this tutorial is to show step-by-step how to build a combined AB+SD model in AnyLogic using one particular architecture. We will highlight the “points of interaction” of agents and system dynamics and try to show that model elements belonging to different approaches live a single space of AnyLogic model and can easily access each other. Having completed this tutorial you should be able to build various multi-method models with more confidence and efficiency.

The tutorial details level assumes some familiarity with AnyLogic model development environment: the instructions in most cases are of the type: “Create a state PotentialUser” rather than “Open the Model palette, click on the State item, then click on the canvas and enter PotentialUser in the edit box”.

The problem definition

We will build a model of a consumer market and a supply chain.

The assumptions we make about the market are similar to ones of the classical models of product/innovation diffusion, e.g. of Bass model with discards and replacements. We will however consider two competing products instead of one.

- There are two alternative products A and B manufactured by different (and competing) companies. The products are equivalent, i.e. can replace each other. The product prices are equal and therefore do not matter.
- Consumers (there are **Total Population = 1000** of them) initially are not using any products but all are potentially interested (are potential users).
- Consumers are sensitive to advertizing and to word of mouth.
- Advertizing generates the demand for a product among the potential users. **Advertizing Effectiveness = 0.011** is the percent of potential users that become ready to buy a particular product (A or B) during a day. Both companies do advertizing.
- Consumers contact each other. A consumer contacts on average a **Contact Rate = 5** other people per day.
- During those contacts the users of products may influence potential users. If a user of e.g. A contacts a potential user, the latter will want to buy A with probability **Adoption Fraction = 0.015**, same for B.
- Any product discards in **Discard Time = uniform(17,23)** days and generates the immediate need to buy a replacement of the same brand.
- If a person wants to buy e.g. A, but A is not available for **Maximum Waiting Time = 2** days, he becomes ready to buy anything that is available (A or B), same for B.

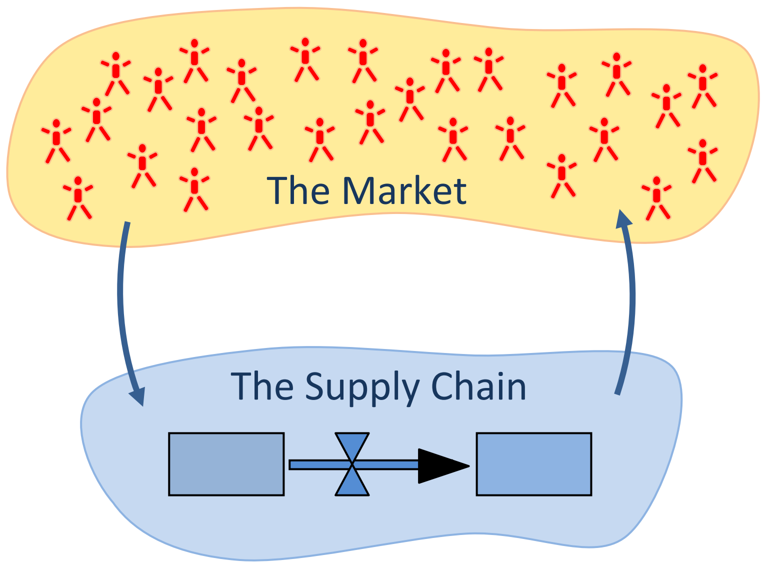
Each company (A and B) has its own supply chain that delivers products to the end consumers. The supply chains are very simple and work as follows:

- The product can be purchased by a consumer only from the retailer stock, initially holding a certain amount (**Initial Retailer Stock = 100**) of product.
- The product is manufactured by a producer. A producer makes **Production Rate** products per day, and this rate may vary, e.g. it can be adjusted according to the demand (which is known to the producer)
- The finished products are delivered to the retailer within **Delivery Time = 2** days.

The output of the model should include the market shares for A and B, the demand (i.e. the number of people who want to buy while the product(s) are not available) and the inventory levels in the supply chains.

The plan

We will model the consumer market in Agent Based way: each consumer will be an agent. The supply chains for both products A and B will be done using System Dynamics. Please note that the problem definition allows many other choices – this is just one of them.



As you obviously know, the best way to develop simulation models is to do it iteratively, i.e. in multiple phases with a runnable model at the end of each phase. In our case it makes sense to have the following order:

1. Start with the market model and just one product (A)
    a. Build the behavior model of an individual consumer
    b. Populate the market with the consumers
    c. Assume the product is available
2. Add supply chain for the product A
3. Add product B

Model development

Phase 1. Create 1000 agents with empty behavior

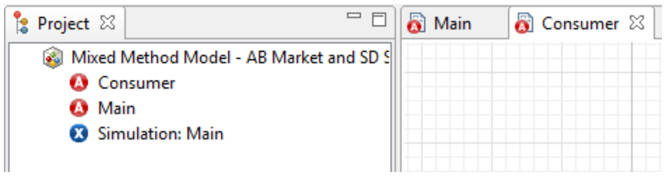
In this phase we will create a new AnyLogic project, define an Active Object class for our consumers, draw a simple animation of a consumer, define, create and visualize a population of 1000 consumers. The consumers will have no behavior at this time, so the model will be doing nothing.

1. Create a new project **Mixed Method Model - AB Market and SD Supply Chain**

   A new project always has one Active Object class **Main**, whose editor opens automatically. Main will be the top level of our model where we will define the interaction between the agent based and system dynamics parts. We will leave it blank for now and start with developing the model of a consumer.

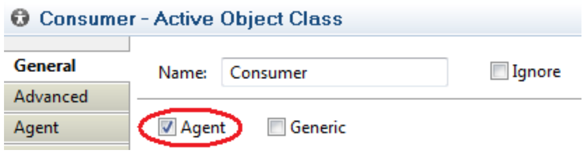
2. Create a new Active Object class **Consumer**

   We need a separate class for consumers, in which we will define the consumer behavior. Having that behavior encapsulated in a class enables us to create multiple instances of consumers. The editor and properties of the class Consumer open when you create it.



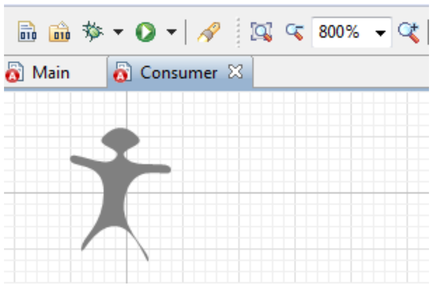
3. In the properties of the class Consumer specify that Consumer is an Agent

   This will enable some useful AnyLogic services for agents, for example communication, space, layout, etc. The icon of the active object class Consumer changes.



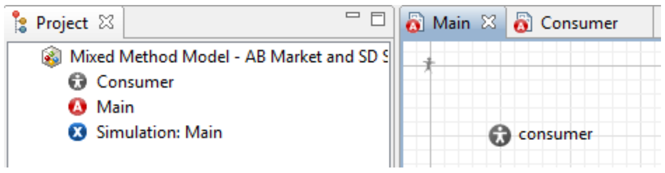
4. In the editor of Consumer draw a visual representation of a consumer using e.g. a Curve shape near the coordinate origin. You may want to use maximum zoom for fine positioning of the curve points. Set line color of the curve to No Color and fill color to **gray**

   Each consumer will be visualized by such shape. The color of the shape as well as its other properties can be dynamically set up by each individual consumer.



5. Drag the class Consumer from the project tree to the editor of Main

This way we specify that instance of Consumer is embedded in the top-level object Main. The icon and the presentation of consumer appear in the editor of Main.



6. Move the presentation of the embedded consumer to the position (50,250) on Main

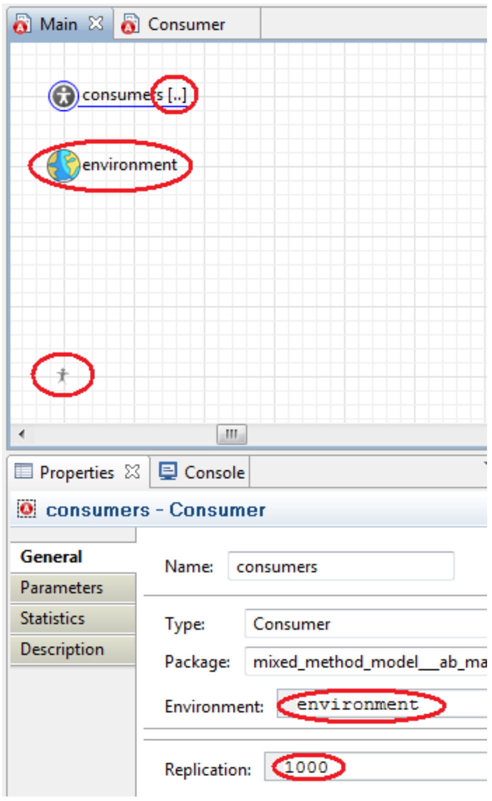
This will be the top left corner of the rectangular space where we will later on visualize the consumer population. The other model elements will be located above it.

7. Set the Replication property of the embedded consumer to **1000**. Change the name of the embedded object from consumer to **consumers**

This way we tell AnyLogic that inside Main there will be not one but 1000 consumers. **consumers** will be the name of that population. The square brackets **[..]** appear near the name to indicate the replication.

8. Add environment object from the Model palette to Main. Set the property Environment of the consumer population to **environment**

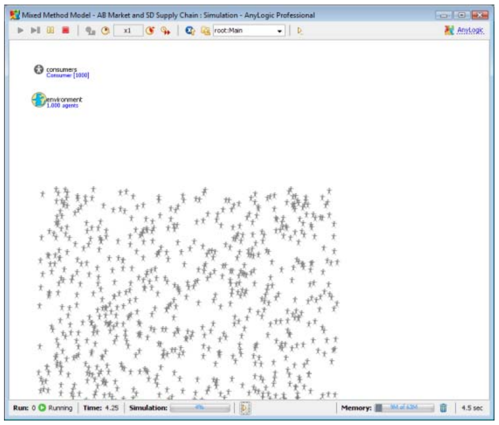
Environment takes care of space, layout, network, communication of agents. In our case we need environment to lay out the agent presentations and to model word of mouth by agent-to-agent interaction.



Name: consumers
Type: Consumer
Package: mixed_method_model__ab_ma
Environment: environment
Replication: 1000

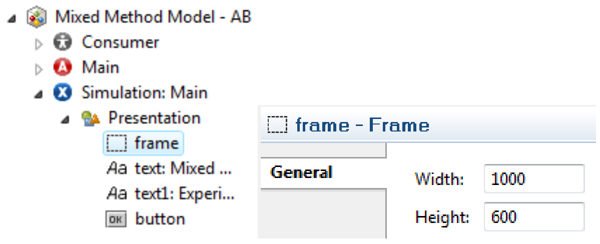
9. Run the model

You should be able to see the picture similar to the one below. There is no dynamics in the model, so nothing changes over time. In the next step we will adjust the window size and the agent space borders.



10. In the project tree expand the experiment item **Simulation:Main** and then expand its Presentation subitem. Click on Frame to view its properties. Set the width of the frame to **1000** and Height to **600**.

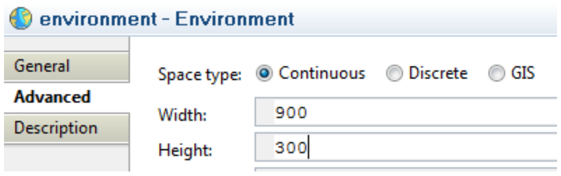
This is the way you specify the initial window size of the simulation.



Width: 1000
Height: 600

11. In the editor of Main click environment. In the Advanced page of its properties set space width to 900 and space height to 300.

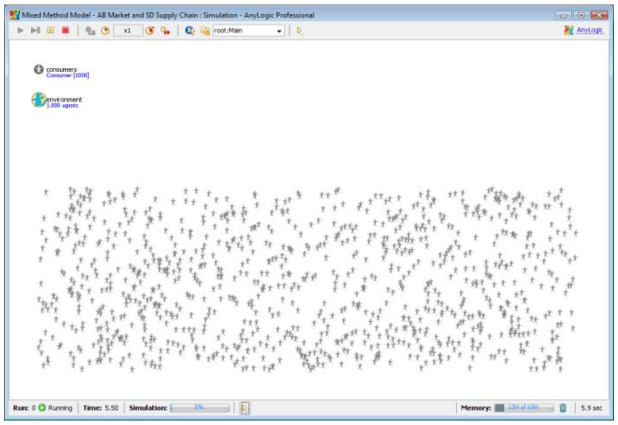
Now the agents will be visualized in a rectangular space that fits the window.



Space type: [✓ Continuous] Discrete GIS
Width: 900
Height: 300

12. Run the model again

The layout should look better now.

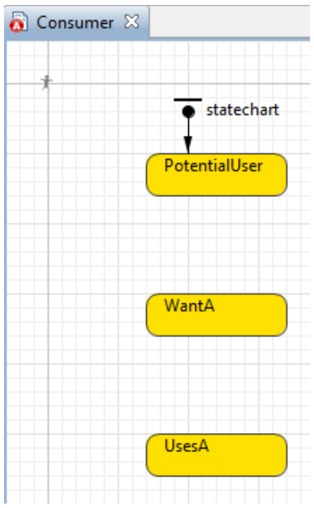


Phase 2. Add simple state-based behavior to the agents

Based on the problem definition, we will define the behavior of a consumer as a sequence of three states: PotentialUser, WantA, UsesA. We will assume that the product is always available, so the transition from WantA to UsesA will be unconditional and immediate. The effect of advertizing will be modeled as stochastic delay associated with a transition from PotentialUser to WantA. No product discards and no agent-to-agent communication will be added in this phase.

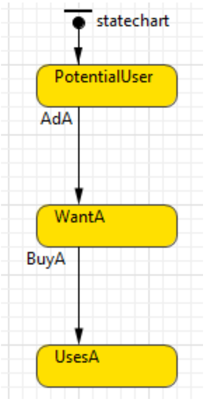
1. In the editor of Consumer draw three states (top down): **PotentialUser**, **WantA** and **UsesA**. Draw the statechart entry point pointing to the state PotentialUser.

The states in the statechart are alternative: the agent (in our case – the consumer) can only be in one state at a time. The statechart entry point marks the initial state, which obviously is PotentialUser. The name of the statechart entry point is the name of the statechart. In general, an Active Object in AnyLogic may have multiple statecharts, but now we need only one.



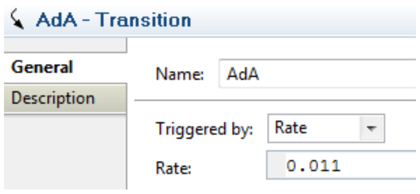
2. Draw transition from PotentialUser to WantA and call it **AdA**. Draw transition from WantA to UsesA and call it **BuyA**. Check the checkbox Show name for each transition and adjust the positions of labels in the editor

Transitions define how the object changes its states. A transition may be triggered by e.g. a message arrival, a condition, or time. The transition AdA will model the effect of advertizing, and BuyA – the event of purchasing the product A.



3. Specify the following properties for the transition AdA: Trigger type: **Rate** and Rate: **0.011**

Transition of rate type in AnyLogic in this case is the same as transition triggered by a stochastic timeout distributed exponentially. When the statechart enters the state PotentialUse, a draw from the exponential distribution is made and the timeout is setup. Therefore, each consumer will have different time of adoption because of advertizing, so that on average 1.1% of potential users will want to buy the product in one time unit (day).


AdA - Transition

General | Description

Name: AdA

Triggered by: Rate

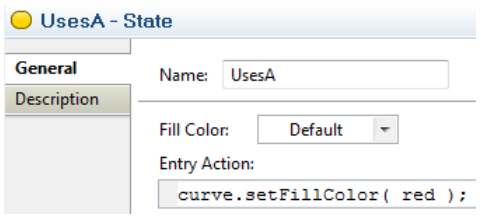
Rate: 0.011

4. Specify the following properties for the transition AdA: Trigger: **timeout** and Timeout: **0**.

As we assume that the product is always available, anybody who wants the product can immediately buy it – therefore once the consumer statechart enters the state WantA, it will proceed to state UsesA with zero delay.

5. In the Entry action of the state WantA write: **curve.setFillColor(pink);**
   In the Entry action of the state UsesA write: **curve.setFillColor(red);**

This way we will change the color of the consumer presentation to visualize the change of its state. As you can see, all model and presentation elements of the active object class are accessible one from another. Note that if you have used other kind of shape to draw the presentation of Consumer, you may need to use other name for the shape.


UsesA - State

General | Description

Name: UsesA

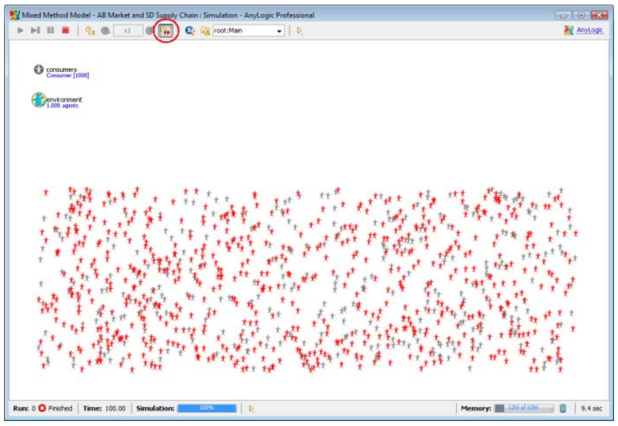
Fill Color: Default

Entry Action:

```
curve.setFillColor( red );
```

6. Run the model. After a while you may press the Virtual time button to speed up the model.

You should be able to see how the population of consumers gradually turns red – this is the effect of advertizing. Although each consumer goes through the state WantA, in which he should be pink, you will not be able to see it as he does not stay in that state and immediately proceeds to UsesA. The model will stop at time 100 when some consumers are users and some are not.



7. Open the properties of experiment Simulation:Main. On the General page set **Random seed (unique simulation runs)**.In the Model time page set Stop time: **Never**.

   As our model is stochastic (you may remember that the source of stochasticity is the rate transition AdA in the consumers' statecharts), the simulation results will depend on the random number generation. By setting the Random seed you tell Anylogic to use different sequences of random numbers for each run. By setting the Stop time to never you tell AnyLogic to execute the model infinitely long (or until there is nothing to execute).

8. Run the model again

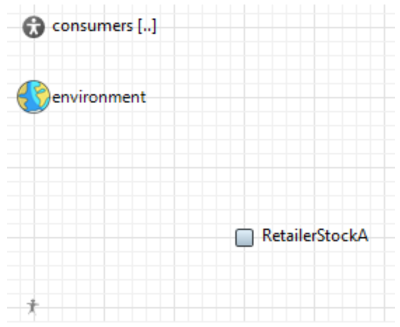
   You will see that eventually all consumers will buy the product.

Phase 3. Add the retailer stock with some initial amount of product A

In this phase we will add the first (actually the end) element of the supply chain for product A: the retailer stock. The (System Dynamics) stock will have some initial amount of product, so some consumers will be able to buy the product. We will modify the consumer behavior so that the transition from WantA to UsesA will only be possible if the there is at least one unit of product on stock. As a result of a consumer taking this transition the amount of available product will be decremented. This is the first point of interaction of the Agent Based and the System Dynamics components in the model.

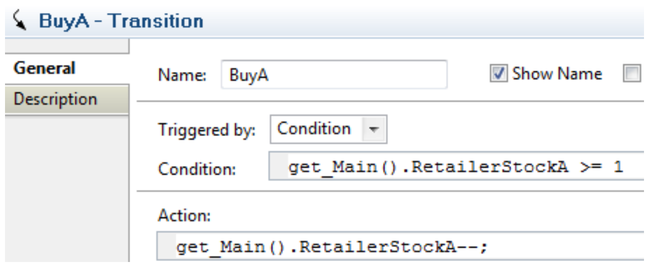
1. Open the editor of Main and add a stock variable **RetailerStock**. Set its initial value to **100**.

   This is the first System Dynamics element in the model. We define it in Main – on the same level where we have embedded the population of agents-consumers.



2. Go to the editor of Consumer and change the properties of the transition BuyA: it should now have trigger type **Condition**, the Condition: **get_Main().RetailerStockA >= 1** and Action: **get_Main().RetailerStockA--;**

   This is how we implement the AB-SD interaction: when in the state WantA a consumer constantly monitors the stock RetailerStockA; when the stock contains at least one product, the transition is taken and, as a result, one unit of product is deducted from the stock. Please note that, as the stock is located one level up regarding the consumer behavior (at the Main object that contains the consumer), we need first get to Main and only then we can access the stock – that is why we use the prefix **get_Main()**, which brings us to the container of Consumer of type Main.



BuyA - Transition

General
Description

Name: BuyA ☑ Show Name ☐

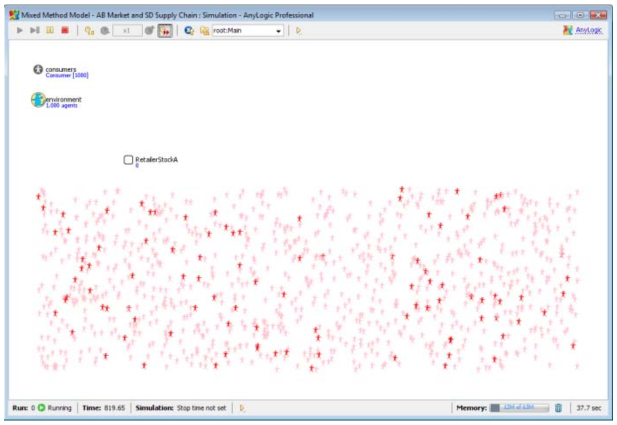
Triggered by: Condition

Condition:
```
get_Main().RetailerStockA >= 1
```

Action:
```
get_Main().RetailerStockA--;
```

3. Run the model in virtual time (fast) mode.

You will see that 100 consumers will be able to buy A, while all others will eventually turn pink as the retailer is out of stock.

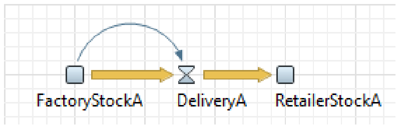

Mixed Method Model - AB Market and SD Supply Chain : Simulation - AnyLogic Professional
consumers
environment
RetailerStockA

Phase 4. Add SD supply chain for the product A

We will add the remaining part of the supply chain for A: production and delivery. The supply chain will be a pure System Dynamics construct starting with Production flow into Factory stock and then through the Delivery flow to the Retailer stock. We will first make the production rate constant and then – variable, depending on the demand for the product. This will be a second point of interaction between the AB and the SD parts of the model. To feed the demand into the SD part we will add a statistics item to the consumer population: it will calculate the number of consumers that are in the state WantA. The production rate will be recalculated once a day according to that number.

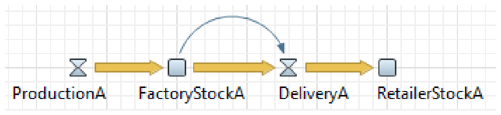
1. In the editor of Main add the stock **FactoryStockA** to the left of **RetailerStockA**. Draw flow from FactoryStockAm to RetailerStockA (by double-clicking the source stock and clicking at the sink stock). Call the created flow **DeliveryA**. Set the formula for DeliveryA = **FactoryStockA / 2**

This way we model the delivery delay of 2 days from the factory to the retailer stock.



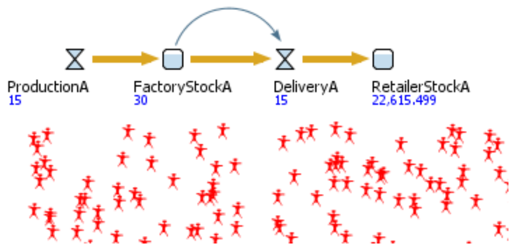
2. Add the flow variable **ProductionA** to the left of FactoryStockA and let it flow into FactoryStockA. Let the ProductionA = **15**.

We will assume a constant production rate for now.



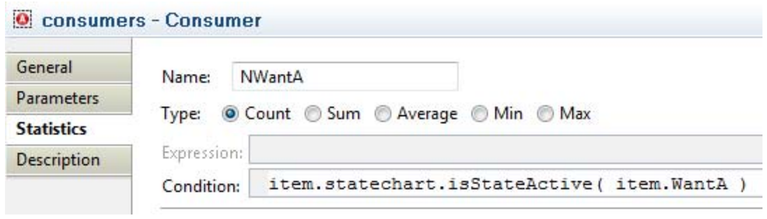
3. Run the model

You will see that now the demand can be fully satisfied, and moreover, the amount of produced product is much greater than needed. Obviously, when everybody has purchased the product, the supply chain continues to grow the retailer stock.



4. In the editor of Main click on the icon of the embedded consumers and open the Statistics page of their properties. Add the statistics item with name **NWantA**. Leave the type of statistics **count**. In the condition field write: **item.statechart.isStateActive(item.WantA)**

As our consumer market is completely disaggregated, we need to iterate through all agents to find out how many of them want to buy the product, i.e. are in the state WantA. The statistics of type count does exactly that: it iterates through the population and counts how many agents satisfy the given condition. In the condition expression "item" represents the current agent, "statechart" is the name of the consumer's statechart, "isStateActive" is a standard method of statechart, and "WantA" is the name of the state defined within the agent, that is why it needs the prefix "item.".



Name: NWantA

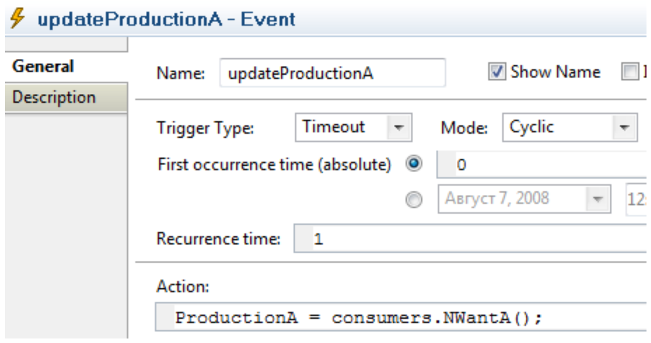
Type: [✓ Count] Sum Average Min Max

Expression:

Condition: `item.statechart.isStateActive( item.WantA )`

5. In the editor of Main add event nearby the flow ProductionA. Call it **updateProductionA**. In the properties of the event specify Trigger type: **timeout**, Mode: **Cyclic**, Recurrence time: **1**, Action: **ProductionA = consumers.NWantA();**

By adding such event we build yet another link between the AB and SD models: now at the beginning of each day the production rate will be modified according to the number of consumers who are willing to buy the product. There could be better formulas than "rate = demand", but that's the simplest one. NWantA() is the statistics method we have defined in the consumer population in the previous step.


updateProductionA - Event

General
Description

Name: updateProductionA
Show Name ✓

Trigger Type: Timeout
Mode: Cyclic
First occurrence time (absolute) ● 0
○ Август 7, 2008 12
Recurrence time: 1

Action:

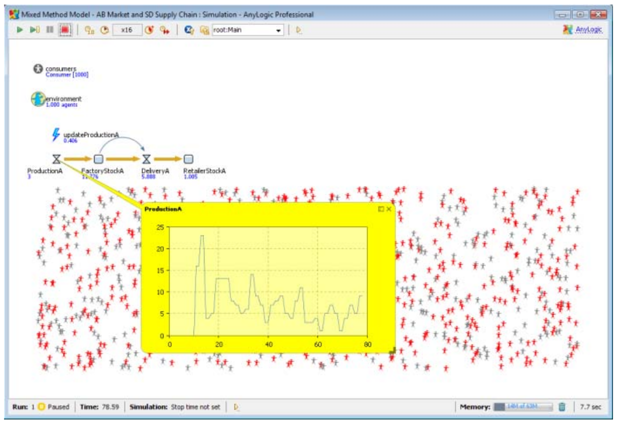
```
ProductionA = consumers.NWantA();
```

6. In the formula for ProductionA remove “15”

Now the production rate is fully controlled by the event updating it, so formula must be removed.

7. Run the model. Click on the ProductionA to bring up its inspect window and then click switch it to chart mode by clicking the small chart icon in the upper right corner.

You should be able to see that now the production rate oscillates, which is typical for the supply chains with their inevitable delays.

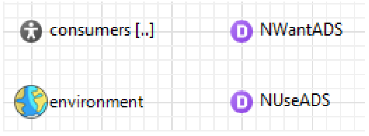


Phase 5. Visualize the dynamics of demand and user base

In this phase we will visualize the dynamics of unsatisfied demand generated by the market and the number of users. As we are modeling the market in a completely disaggregated way, we need to add another statistics items to the population of consumers, namely the one counting the number of agents that are in UsesA state. Also, if we want to see the history of these outputs, we need to create a couple of datasets and a chart, e.g. a time stack chart.

1. In the editor of Main click on the consumers and go to their Statistics page. Add the second statistics item **NUseA** of “count” type with expression **item.statechart.isStateActive(item.UsesA**) (which you can copy from the item NWantA and modify).
2. Stay in Main and add two datasets from the Analysis palette: **NWantADS** and **NUseADS**.

To be able to visualize the history of a certain value, we need to keep that history in a dataset object.

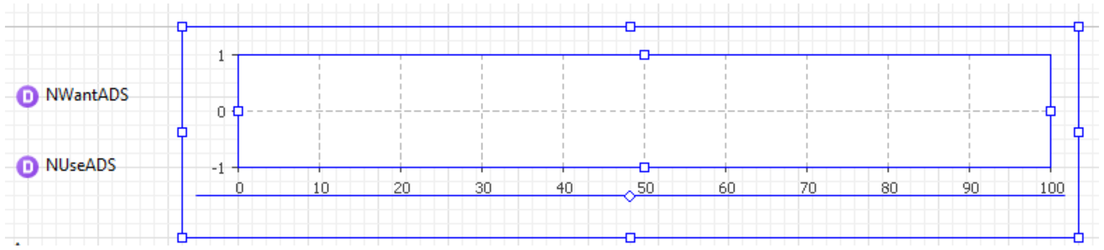


3. In the properties of NWantADS specify the Vertical axis value: **consumers.NWantA()**, for NUseADS correspondingly: **consumers.NUseA()**.

Datasets in AnyLogic are arrays of pairs (x,y). As horizontal axis value (x) is time by default, the data added to the e.g. NWantADS will be of type (t, demand at time t). By default the capacity of a dataset is 100 samples.

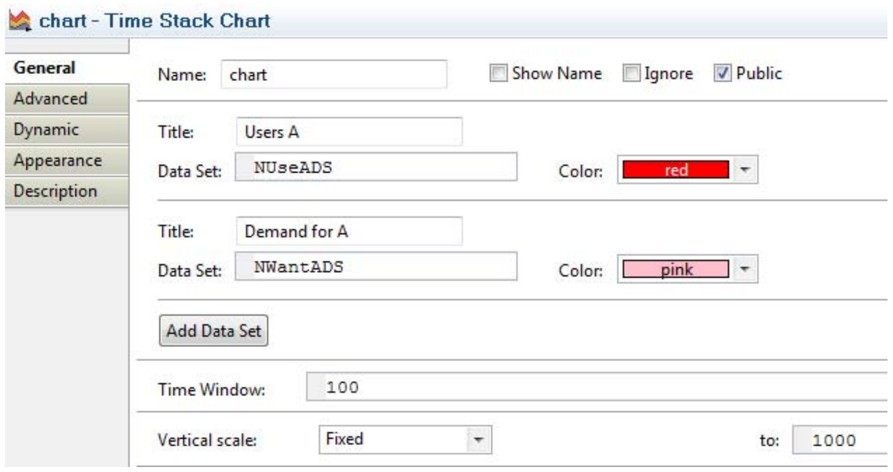
4. From the Analysis palette drop Time stack chart to the editor of Main

The best way to visualize the dynamics of fractions of the population in time is time stack chart.



5. In the properties of the chart add two data sets: NUseA with title **Users A** color **red** and NWantADS with title **Demand for A** and color **pink**. Set the Vertical scale of the chart as **Fixed** to **1000**.

This defines what will be shown in the chart. By default the time window of the chart is 100 and it will automatically update the datasets each 1 time unit.



Name: chart ☐ Show Name ☐ Ignore ☑ Public

Title: Users A
Data Set: NUseADS Color: red

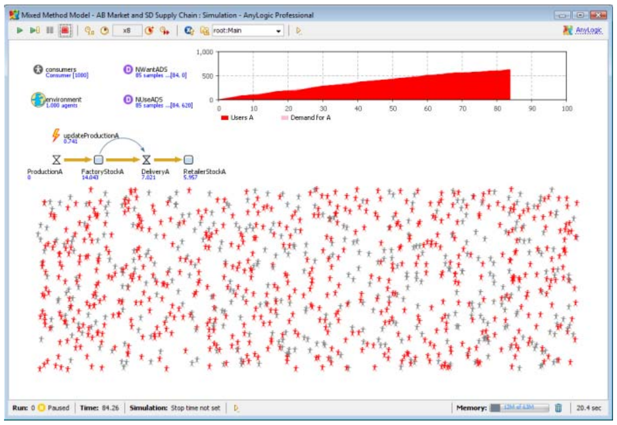
Title: Demand for A
Data Set: NWantADS Color: pink

Time Window: 100

Vertical scale: Fixed to: 1000

6. Run the model

You will see how the user base for A grows. The supply chain works well and there is only a small slightly oscillating unsatisfied demand.

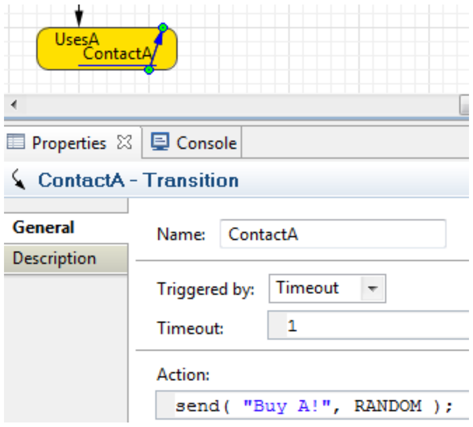


Phase 6. Add agent communication to model word of mouth

In this phase agents will start talking to each other. As we are interested in users of the product talking to potential users, we will set up a cyclic transition in the state UsesA. The transition will be taken periodically and on each occurrence the agent-user will be sending a message to a random other agent saying that product A is good. If another agent is a potential user (i.e. is in the state PotentialUser), he will react to such a message by changing its state to WantA. Obviously, we will achieve this be adding a transition from PotentialUser to WantA triggered by the corresponding message.

1. Go to the editor of Consumer and draw a transition inside the state UsesA such that it starts and ends on the borders of the state and all its segments are inside the state. Call it **ContactA**. Set the trigger type of the transition to **Rate** with rate = **5 * 0.015**. Set its Action to **send("Buy A!", RANDOM);**

   The internal transition is a cyclic transition that does not take the statechart out of the state in which it is defined. In this case the transition ContactA will occur with the specified rate. The value of rate (5+0.015) needs to be further explained. We could model all contacts straightforwardly, i.e. 5 contacts per day, in which case the rate would be 5. But we know that not all contacts are successful, i.e. convince potential users to buy A. Therefore here we model only a “successful fraction” of contacts and make them more rare by multiplying by 0.015. In the action of the transition the consumer chooses another random consumer (not necessarily the potential user!) and sends him a text message “Buy A!”.



ContactA - Transition

General
Description

Name: ContactA

Triggered by: Timeout

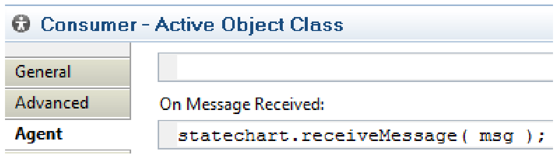
Timeout: 1

Action:

```
send( "Buy A!", RANDOM );
```

2. Go to the Agent page of the properties of the Consumer class. In the On message received field write **statechart.receiveMessage(msg);**

In the previous step we have set up the agents-users to periodically contact other agents by sending them the message "Buy A!". In the section On message received we can specify the reaction on the incoming messages. The code forwards the message to the statechart, where we will specify one additional transition in the next step.

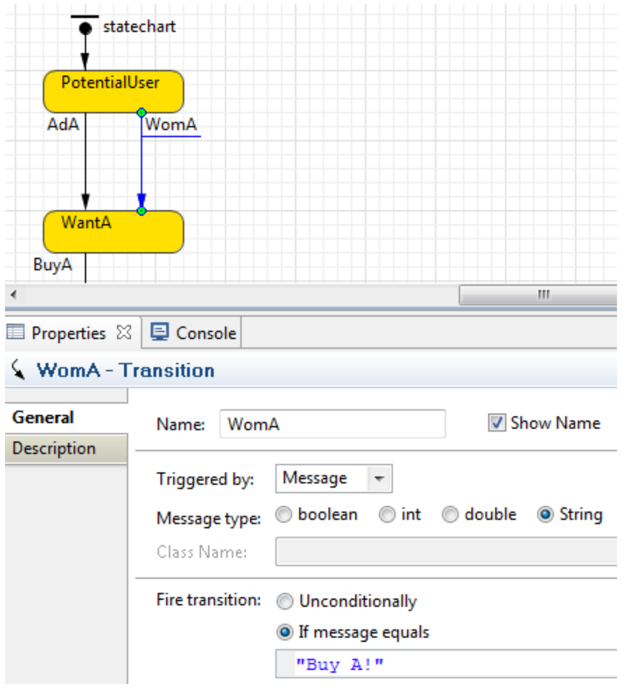

Consumer - Active Object Class

On Message Received:

```
statechart.receiveMessage( msg );
```

3. Add one more transition to the statechart of the Consumer: from PotentialUser to WantA and call it **WomA**. Set the trigger of the transition to **Message**, the type of the message: **String**, and let the transition to fire **If message equals "Buy A!"**.

This is the last step in modeling the word of mouth. The message that comes from another agent is forwarded the statechart, and, if the statechart is in the state PotentialUser, causes an immediate transition to WantA. In all other states the message is ignored.



WomA - Transition

Name: WomA [✓ Show Name]

Triggered by: Message

Message type: boolean int double [✓ String]

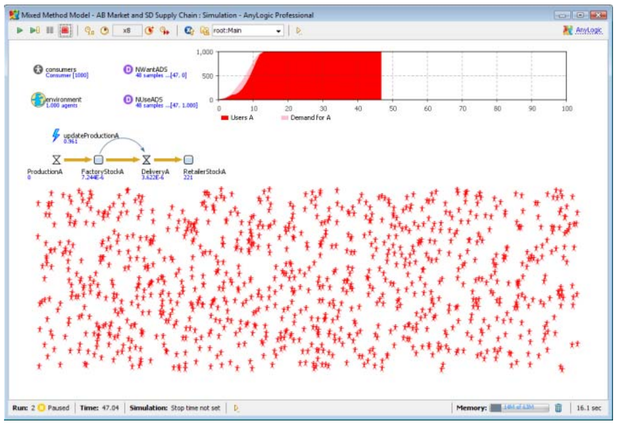
Class Name:

Fire transition: Unconditionally [✓ If message equals]

```
"Buy A!"
```

4. Run the model.

You should see that the market saturation is achieved a lot faster now. The chart shows the well-known S-shaped curve of the product adoption. The unsatisfied demand during the peak of interest to the product is quite significant.

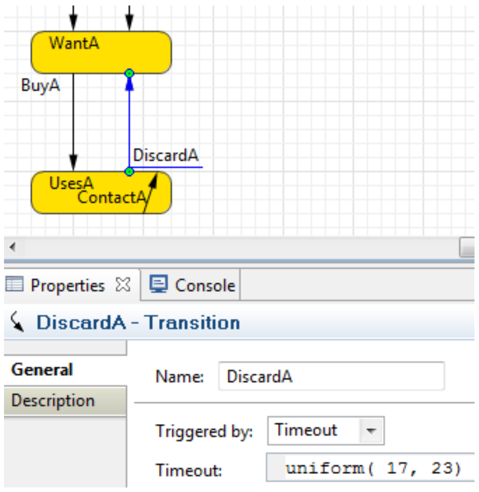


Phase 7. Add product discards

This is a very simple phase. As the product is discarded after a certain period and the user needs to buy a replacement, we will add a transition from UsesA to WantA triggered by a constant timeout Discard Time. Having defined such transition we restrict the sojourn time in the state UsesA by Discard Time.

1. In the statechart of a consumer add a transition from UsesA to WantA, call it **DiscardA**. Set the trigger to **Timeout** with value **uniform(17, 23)**.

When the statechart enters the state UsesA, a sample of stochastic timeout is evaluated and countdown is started for the transition DiscardA. The time of using a particular product item (i.e. the time spent in UsesA) therefore will be distributed uniformly from 17 to 23. Please note that the transition ContactA will not be resetting that timeout as it is an internal transition. After taking DiscardA the consumer will enter WantA state, which means he immediately is willing to buy a replacement.



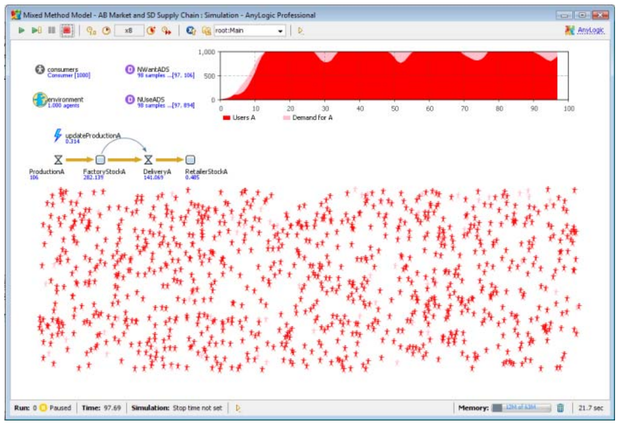
Name: DiscardA

Triggered by: Timeout

Timeout: `uniform( 17, 23)`

2. Run the model

After the market is saturated (everybody is using the product), you will observe the periodic shortages in the product supply caused by the product expiration at the majority of users. The supply chain as it is set up now is not capable of satisfying these peaks of demand.

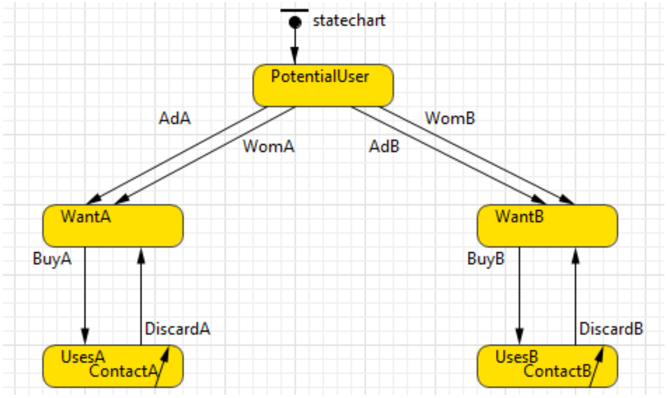


Phase 8. Add product B to the model

Now we can add the product B that competes with product A. We will do it by simply copying the supply chain for A and renaming its elements. The product-specific part of the consumer's statechart will also be copied so that PotentialUser will have two alternatives: transition to WantA or to WantB. At this time we will not model the maximum waiting time for a product. Two new statistics items and two datasets will be needed to show the dynamics of the market.

1. In the editor of Consumer select two states WantA and UsesA and all transitions (in other words, select all statechart elements except for the state PotentialUser and the statechart entry point. Make a copy of the selection and arrange the transitions as shown in the screenshot below. Rename "A1" that appears in the endings of the copied elements to "**B**".
   In the transitions WomB and ContactB change "Buy A!" to **"Buy B!"**.
   In the properties of the transition BuyB change RetailerStockB to **RetailerStockB**.
   In the Entry actions of states WantB and UsesB change the colors to **lightBlue** and **blue**.

   In this step we have defined an alternative path of the consumer behavior. We assume the same advertizing effectiveness for product B (0.011) and same contact rate and adoption fraction (obviously we can change any of those parameters). Note that the new statechart structure allows only initial choice of product by a potential user; once the initial choice has been made, there is no way for a consumer to switch.



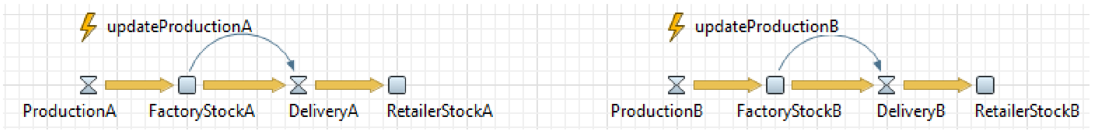
2. In the editor of Main click on the consumers and go to the Statistics page. Add two more statistic items: **NWantB** and **NUseB** with conditions

item.statechart.isStateActive(item.WantB) and **item.statechart.isStateActive(item.UsesB)** correspondingly

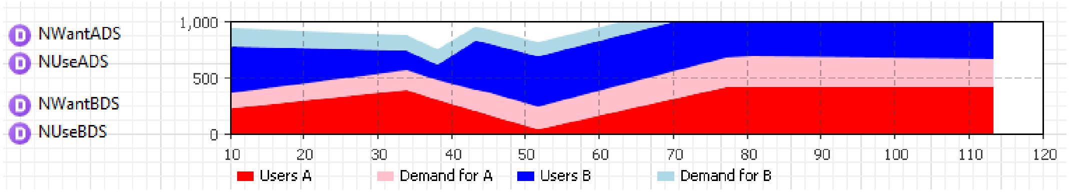
These statistics items will count the demand and the number of users of the product B.

3. In the editor of Main select the whole supply chain for product A (all SD elements) and also the event updateProductionA, copy it and paste to the right. Rename:
event updateProductionA1 to **updateProductionB,** stock ProductionA1 to **ProductionB**, etc.
Change the action of updateProductionB to **ProductionB = consumers.NWantB();**
Change A to B in all formulas in the supply chain for product B.

Needless to say, all that copying and renaming does not look elegant. And indeed, AnyLogic offers much better way of replicating pieces of model: you can pack the supply chain and the corresponding event into a new Active Object class, parameterize it with e.g. demand, and instantiate it twice in Main: once for A and once for B. For simplicity however we are not doing it in this tutorial.

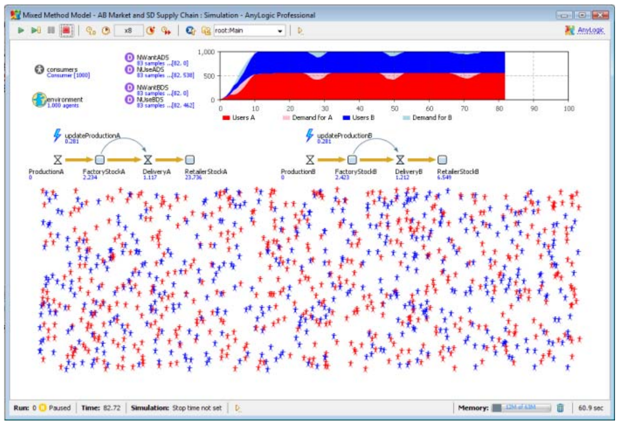


4. In the editor of Main make a copy of the two datasets. Change the names of the new items to **NWantBDS** and **NUseBDS**. Change their values to **consumers.NWantB()** and **consumers.NUseB()**. Add these datasets to the chart with titles “Demand for B” and “Users B”.



5. Run the model

You will see the symmetric diffusion dynamics for A and B. Please note that the market share is defined only by the initial adoption and cannot change later on – because of the consumer’s statechart structure.

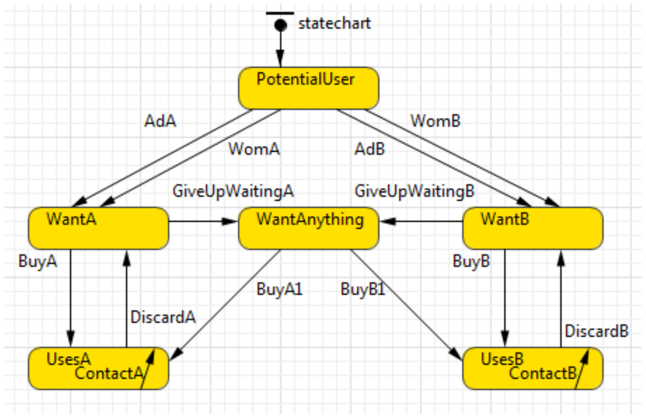


Phase 9. Add product switching triggered by waiting time

According to our problem definition, if a consumer waits too long for a particular product, he becomes impatient and will buy whatever is available (A or B), so product switching is possible. The consumer statechart can easily be modified to capture this: we should add two timeout transitions from WantA and WantB to a new state WantAnything. From that state, in turn, there will be two alternative condition-type transitions to UsesA and UsesB triggered by the purchase of the corresponding product.

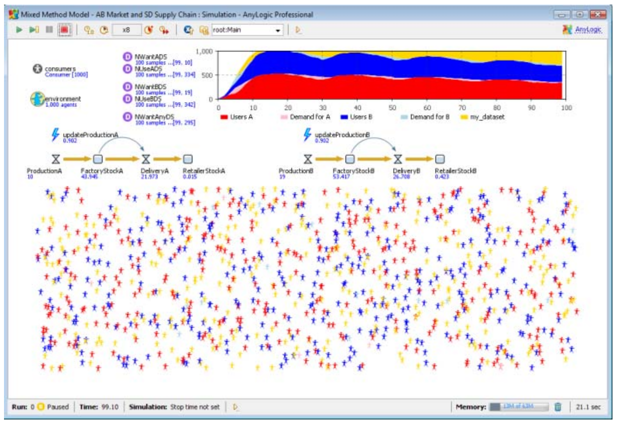
1. Add a new state **WantAnything** in the middle of the consumer's statechart. Draw two transitions from WantA and from WantB to the new state. Call them **GiveUpWainingA** and **GiveUpWaitingB**. Both transitions should be of Timeout type with timeout **2**. Make a copy of the transition BuyA (it will be called **BuyA1**) and place it from WantAnything to UsesA. Similarly, create a copy of BuyB. Leave the triggers of those transitions as they are. In the Entry action of WantAnything write **curve.setFillColor(gold);**

The timeout transitions restrict the waiting time for a particular product to 5. Once the consumer gets to the state WantAnything, he starts monitoring availability of both products and will buy whatever becomes available first.



2. In the editor of Main click on the consumers and add yet another statistics item **NWantAny** with condition **item.statechart.isStateActive(item.WantAnything)**. Add the corresponding dataset **NWantAnyDS** with Vertical axis value **consumers.NWantAny()** and add that dataset to the chart with **gold** color.
3. Run the model

You will see that the unsatisfied demand grows over time. This is caused by the wrong demand estimations made by the supply chains: the production rate for each product depends only on the number of consumers that are waiting for that particular product and does not take into account those who are in the state WantAnything.

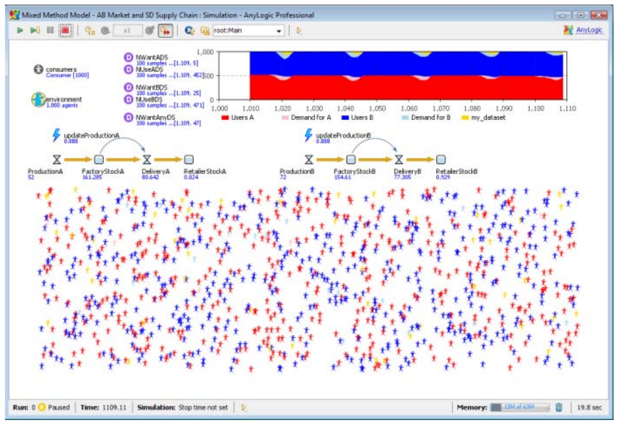


4. Modify the actions of events updateProductionA and updateProductionB in the following way: write **ProductionA = consumers.NWantA() + consumers.NWantAny();** and **ProductionB = consumers.NWantB() + consumers.NWantAny();**.

   This way we take into account the demand of those consumers who are ready to buy any product.

5. Run the model

   Now the supply chains work better, although there are slight periodic shortages. You can also see that the market shares fluctuate a little bit, i.e. the product switching occurs. But the balance between A and B stays around 50/50 because the model is completely symmetric.



Experimenting with the model

At this time the model fully reflects the problem definition and you can do various experiments with it.

For example, you can play with the supply chain policies and explore how they affect the resulting market shares. It might make sense to monitor not only the demand satisfaction but also the inventory level at the stocks (which, obviously, you wish to keep at minimum level).

The supply chain can be modeled using “discrete event” or “process” technology (with AnyLogic Enterprise Library blocks) instead of the SD. Then it will be a lot easier to model the delivery time and granularity.

Another interesting set of experiments can be done with the consumer behavior and consumer population. You can add a social network into the model where contacts can only occur between people who know each other. This will obviously affect the diffusion process. You can add heterogeneity to the consumers at the parameters level (e.g. different sensitivity to advertising and to word of mouth, different level of brand loyalty), or at the behavior pattern level (e.g. some consumers may become frustrated with the product, spread negative information, etc.).