Chapter 5. System dynamics

The intention of this chapter is not to teach system dynamics modeling (there are other excellent books serving this purpose), but rather to explain how to build and run system dynamics models in AnyLogic.

The process of building a system dynamics model in AnyLogic does not differ much from the process used in Vensim™, Powersim™, or STELLA™. AnyLogic supports:

- Stock and flow diagrams with automatic consistency checking
- Arrays (subscripts) with enumeration- and range-type dimensions
- Table functions (lookup tables)
- Delays and other SD-specific functions
- Units and unit checking

AnyLogic simulation engine includes numerical solver for differential, algebraic, and mixed equations.

The experiment capabilities include general simulation, interactive simulation (flight simulators and games), compare runs, sensitivity analysis, calibration and optimization, and Monte-Carlo.

In addition to this traditional toolset, AnyLogic offers several important extensions:

- You can build modular, hierarchical, and object-oriented system dynamics models. System dynamics components can be packaged into AnyLogic agents, parameterized, organized in various structures, and reused.
- System dynamics can be combined with discrete event and agent-based modeling. AnyLogic naturally supports the interaction of stock and flow structures with events, statecharts, process flowcharts, and agent populations.
- AnyLogic simulation engine is a hybrid engine designed for efficient and accurate simulation of continuous dynamics being interrupted by a large number of discrete events.
- Animation capabilities are much richer than those in any other SD modeling framework and include 2D and 3D animation, interactive UI, and business graphics.
- AnyLogic models are standalone, 100% Java applications, and therefore run on any platform; they can be published in AnyLogic Cloud. This means you can easily deliver or share models with end users, who will be able to run them without installing any software.

5.1. How to draw stock and flow diagrams

Elements of stock and flow diagramming can be found in the **System Dynamics** palette, see Figure 5.1. If the user stays within the pure system dynamics modeling paradigm, these elements, perhaps along with the charts from the **Analysis** palette will be all he needs.

Drawing stocks and flows

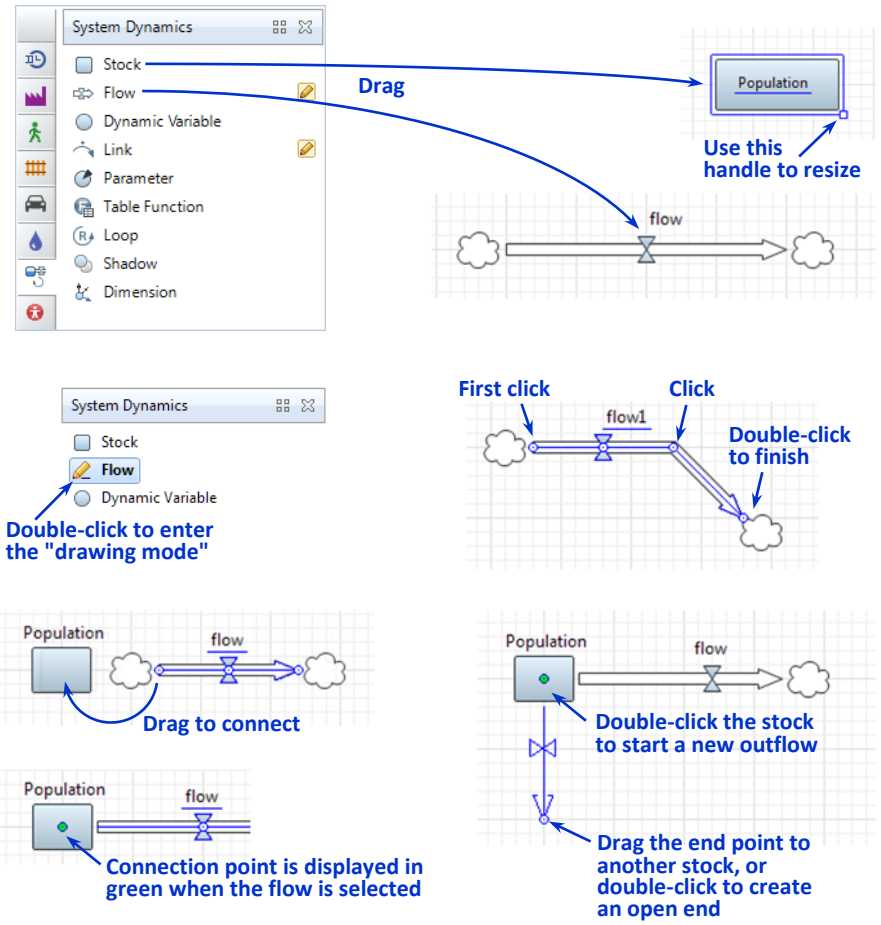


Figure 5.1 Drawing stocks and flows

To draw a stock:

1. Drag the **Stock** element from the **System Dynamics** palette into the graphical editor.
2. Enter the stock name (the name can be moved and changed at any time).
3. Resize the stock shape by dragging its bottom right corner.

To draw a flow from a stock:

1. Double-click the stock and drag the flow out of it. Each subsequent click adds a point to the flow polyline.
2. To finish the flow, click on the target stock or double-click to create an open end (a "cloud").

To draw a flow not connected to any stocks:

1. Drag the **Flow** element into the graphical editor. A straight, open-ended flow "from cloud to cloud" is created.

To connect a flow to a stock:

1. Drag its end point onto the stock. The connected end point turns green.

To draw a flow with multiple segments (a polyline flow):

1. Double-click the **Flow** element in the **System Dynamics** palette. The flow tool switches to "drawing mode".
2. Draw the flow by clicking at each point. Double-click at the end point to finish.

To add a new point (segment) to an existing flow:

1. Double-click the flow where you want to create a new point.

The color of stocks and flows can be customized by using the **Color** control in the stocks and flows' properties.

Drawing variables, dependency links, polarities, and loop types

In addition to dynamic variables that are not stocks or flows (also called auxiliary variables in other system dynamics tools), you can consider using parameters and "constant" variables, which serve as inputs to the feedback loop structures.

To create a variable:

1. Drag the **Dynamic Variable** element from the **System Dynamics** palette into the graphical editor.
2. Enter the variable name (the name can be moved and changed at any time).

To create a dependency link from a variable:

1. Double-click the variable and drag the link.
2. Click to finish.

Links from flows cannot be drawn by double-clicking them. You must create a link first, and then connect it.

▶ **To create a "standalone" link:**

1. Drag the **Link** element from the **System Dynamics** palette into the graphical editor.

▶ **To connect a link to a stock, flow, parameter, or dynamic variable:**

1. Drag the end point of the link to the element. The connection is shown as a green point.

▶ **To create a loop icon:**

1. Drag the **Loop** element from the **System Dynamics** palette into the graphical editor.
2. Choose the direction and type of loop in the loop properties.

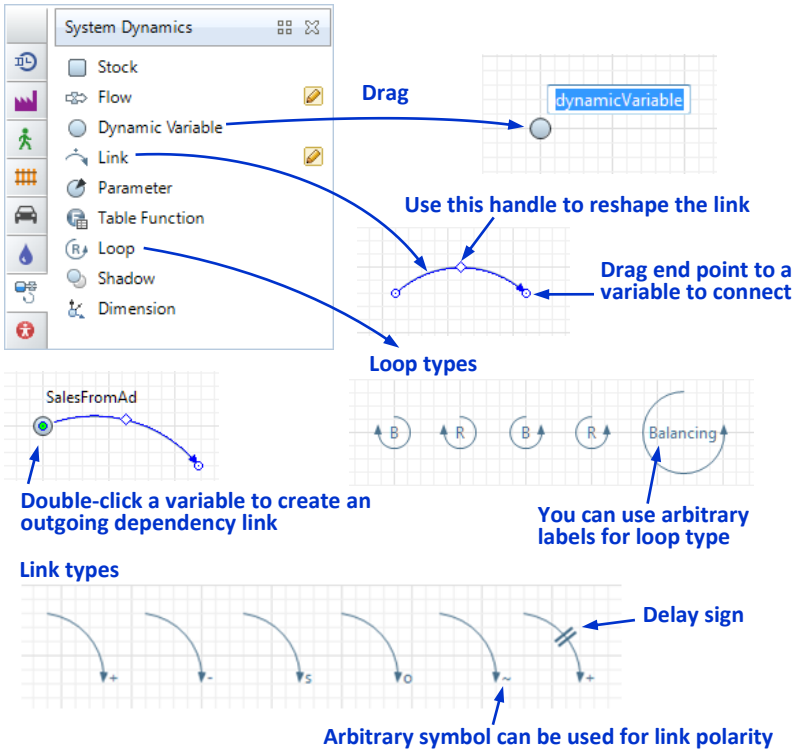


Figure 5.2 Drawing variables, dependency links and loops

AnyLogic offers a choice of link polarity symbols, see Figure 5.2. A link can also be decorated with the delay sign. Loop signs can be customized as well. The colors of variables, links, and loops, and link line width can be changed in these elements' properties.

Unlike in some other graphical notations, in AnyLogic table functions are not graphically linked to variables.

Naming conventions for system dynamics variables

System dynamics variables should obey AnyLogic naming conventions. Those are a bit different than in other system dynamics tools.

Variable names cannot contain spaces or line breaks. We recommend using mixed case, with the first letter of each word capitalized. Underscores ("_") are also allowed.

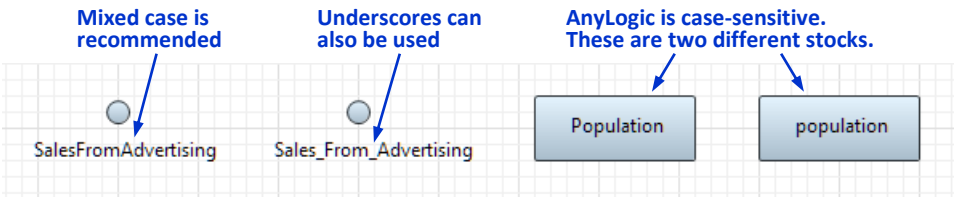


Figure 5.3 Names of system dynamics variables

Layout of large models. "Sectors" and shadow variables

Traditional SD frameworks offer the following way of laying out large models: the model is partitioned into pieces called *sectors*, each focusing on a particular aspect – for example, Housing, Business, Tax, and Labor. The diagram for each sector is drawn separately from the others, and variables used in multiple diagrams are multiplied as well, so that there are *no graphical links between sectors*. For each such variable, there is one "original" instance in one of the sectors and "shadow" copies in other sectors.

In AnyLogic, you can partition the model into components in an object-oriented way by using agents exposing the "input" and "output" dynamic variables as part of their interface. However, the traditional "sector" method is still available. For that, AnyLogic supports view areas and *shadow variables*. In Figure 5.4, the population model is partitioned into a Housing sector and a Population sector. The "interface" between the sectors includes two variables:

- The stock **Houses** from the Housing sector is used in the Population sector.
- The variable **HouseholdsToHousesRatio** from the Population sector is used in the Housing sector.

Correspondingly, the Housing sector has a shadow copy of **HouseholdsToHousesRatio**, and vice versa. You can distinguish the shadow from the original by the angle brackets around its name: **<HouseholdsToHousesRatio>**.

To create a shadow variable:

1. Drag the **Shadow** element from the **System Dynamics** palette into the graphical editor.
2. Select the "original" variable from the list.

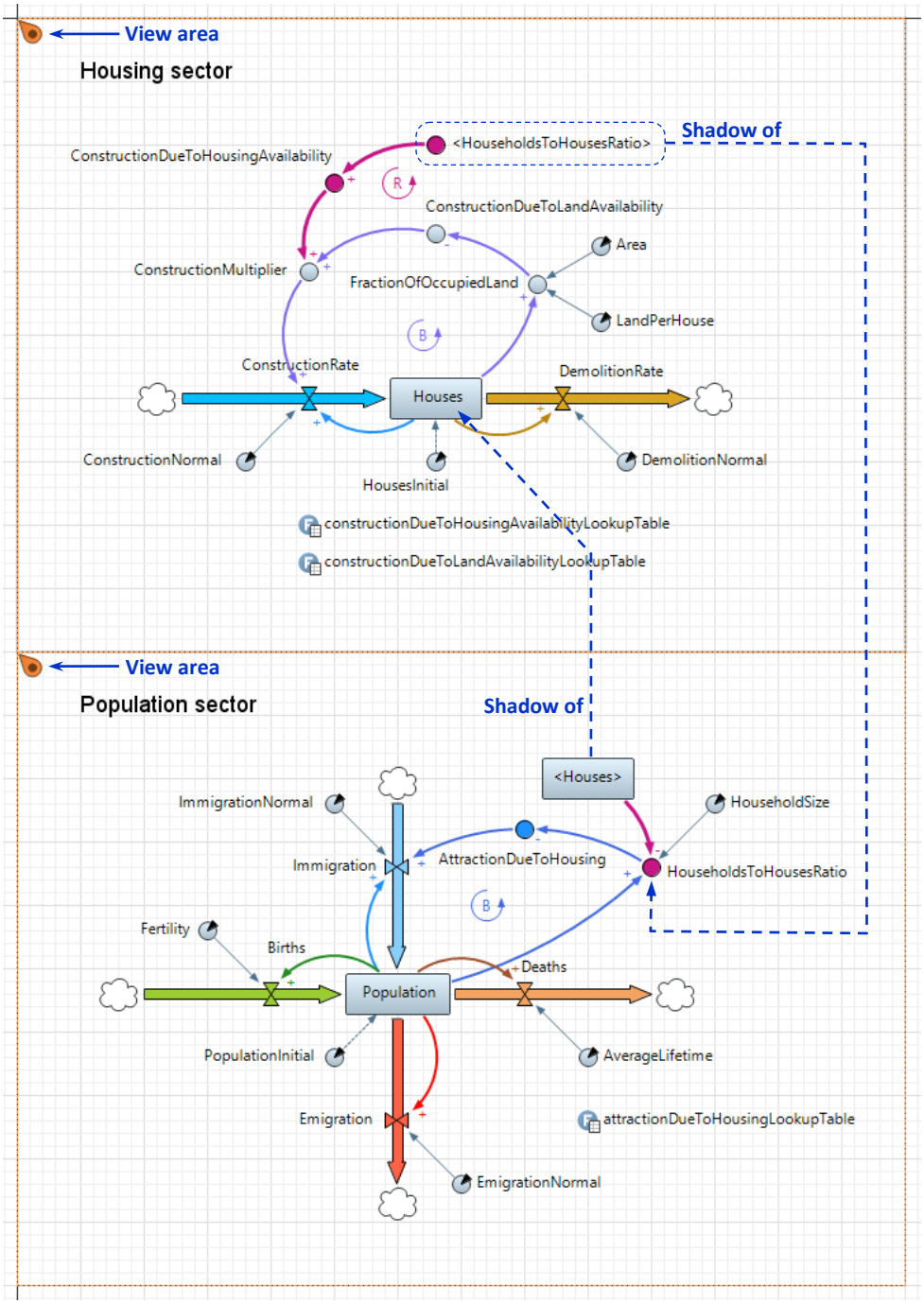


Figure 5.4 Shadow variables and view areas are used to create "sectors"

5.2. Equations

In AnyLogic, equations for stocks are constructed automatically from the graphical structure, and optionally can be typed manually. Equations for flows and other variables are entered in their properties and checked against the existing dependency links.

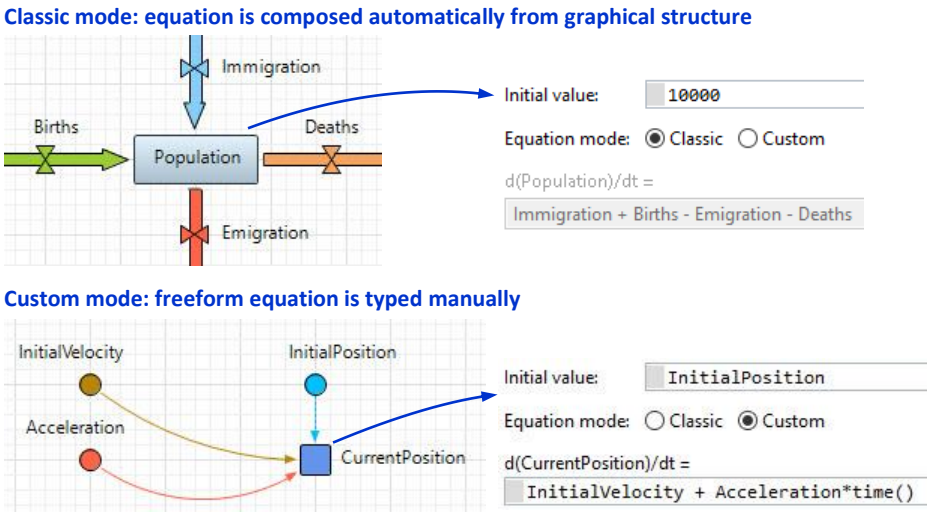


Figure 5.5 Classic and custom modes of stock equations

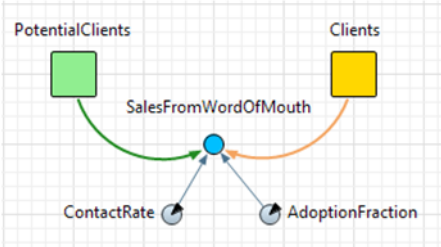
The default mode for stock equation is classic (automatic). In this mode, each incoming flow is added to the stock derivative expression, and each outgoing flow is subtracted. Therefore, the resulting equation is a linear combination of flows always conforming with the graphical structure, see Figure 5.5.

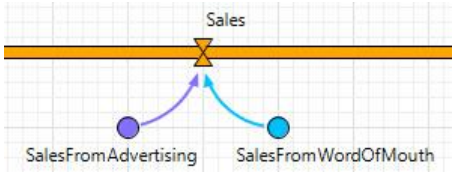
If you switch the radio button **Equation mode** in the stock properties to **Custom** mode, you will be able to manually type a freeform expression. The expression can then contain arbitrary arithmetic operations and function calls, see the bottom case in Figure 5.5. This mode is used more frequently in dynamic (physical) systems models than in system dynamics.

The expression for stock derivative typed in custom mode is still checked for conformance with the graphics – namely, for each instance of a variable in the expression there should be an incoming link or flow from that variable to the stock, and vice versa.

Equations for flows and other variables are freeform. They are typed manually and checked for conformance with the graphical structure in the same manner, see Figure 5.6. If discrepancy is detected, a problem item appears in the **Problems** view

and a small red sign is displayed next to the equation. To view the error description, move the mouse cursor over the sign.





Correct

Sales=

```
SalesFromAdvertising + SalesFromWordOfMouth
```

Correct: equation conforms with links

SalesFromWordOfMouth=

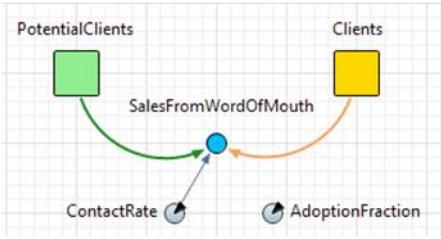
```
Clients * ContactRate * AdoptionFraction *
PotentialClients / ( PotentialClients + Clients )
```

Error: the link from the parameter AdoptionFraction is not reflected in the equation

SalesFromWordOfMouth=

```
Clients * ContactRate *
PotentialClients / ( PotentialClients + Clients )
```

AdoptionFraction is not used but expected



Error: the parameter is used in the equation, but the link is missing

SalesFromWordOfMouth=

```
Clients * ContactRate * AdoptionFraction
PotentialClients / ( PotentialClients + Clients )
```

AdoptionFraction is used but not expected

Figure 5.6 Equations of flows and other variables

When typing equations, it makes a lot of sense to use AnyLogic code completion. You do not have to type the name of a variable completely (which may be time-consuming, as the names in system dynamics tend to be long). Instead, you can type the first letters of the name and press Ctrl+space (Mac OS: Option-space). AnyLogic will display a list of suggestions – variables and functions whose names contain the typed letters. You then just need to choose the intended item, see Figure 5.7.

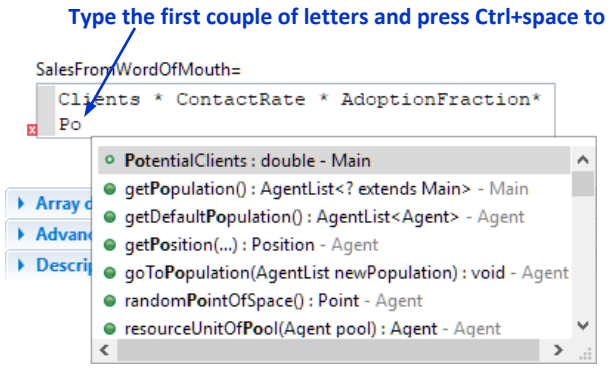


Figure 5.7 Code completion

To rename a system dynamics variable:

1. Select the variable.
2. Double-click the variable name in the graphical editor and enter the new name, or type the new name in the variable properties.
3. Press Ctrl+Enter (Mac OS: Command-Enter) to perform the model refactoring.

After renaming a system dynamics variable, you should press Ctrl+Enter before leaving the name editor. This will rename all occurrences of the variable in all expressions throughout the model. If you do not press Ctrl+Enter, the variable name will change, but its occurrences will not be renamed.

Using Java in SD equations

Manually typed equations are in fact Java of numeric type **double**. As such, they can contain function calls and conditional operators, as well as reference arbitrary elements in the model. This is an example of the conditional operator:

```
Level < 10 ? MaximumRate : NormalRate
```

The above expression evaluates to **MaximumRate** when the **Level** is lower than 10 and to **NormalRate** otherwise. This is equivalent to the Vensim™ IF THEN ELSE function.

You can define custom Java functions implementing analytical or general algorithmic dependencies and use them in SD equations. The functions should return the Java type **double**. For example, suppose a dependency of a certain form is used in multiple places across the model. You can create a Java function with one or several arguments – say, **SmoothReverseProportional()** – and use it in multiple equations:

```
MaxBirthRate * SmoothReverseProportional( Crowding )
```

The following expression references the agent population **consumers** located at the same level as the system dynamics diagram, and uses the statistics function **NoWaiting()** defined in that population:

```
BaseRate + consumers.NoWaiting() * DemandCoefficient
```

Assume the system dynamics diagram is located in an agent, which is embedded in **Main**, and that there is another agent, **manufacturing**, embedded in **Main** as well, with a queue named **stock** inside. To calculate, for example, the total cost of agents in **stock**, you should type in the system dynamics model:

```
main.manufacturing.stock.size() * CostPerItem
```

As you can see, the use of Java in equations helps you to link the system dynamics with other elements in the model, in particular with agents and discrete event elements. For more information on how to navigate in the hierarchical model with Java.

Keep in mind, however, that linkage of *two system dynamics models located in different agents* should not be done with Java expressions. Instead, you should explicitly link the models using external variables and graphical connectors.

You should also refrain from using stochastic functions – functions that return a different random value each time they are called, like **exponential()** or **logistic()** – in the system dynamics equations.

"Constant variables" and parameters

Flows and dynamic variables (but not stocks) can be declared *constants* by selecting their **Constant** property. The expression typed in the value field will be treated as an initial value, and will not be evaluated on each numeric integration step. You can, however, change the value of such a variable by explicitly assigning a new value.

Parameters, in this sense, are similar to constant variables. They have no dynamic equations and have default values that can be changed during the model runtime. In addition, parameters are a part of the agent interface (they are visible from outside) and can be linked to the parameters of outer and inner agents.

In Figure 5.8, the dynamic variable **Acceleration** is marked as constant. Its initial value is 1. After 10 seconds have been simulated, the event **ChangeAcceleration** will occur and change the value of **Acceleration** to 2. This, by the way, is yet another way of linking a discrete element of the model with the system dynamics.

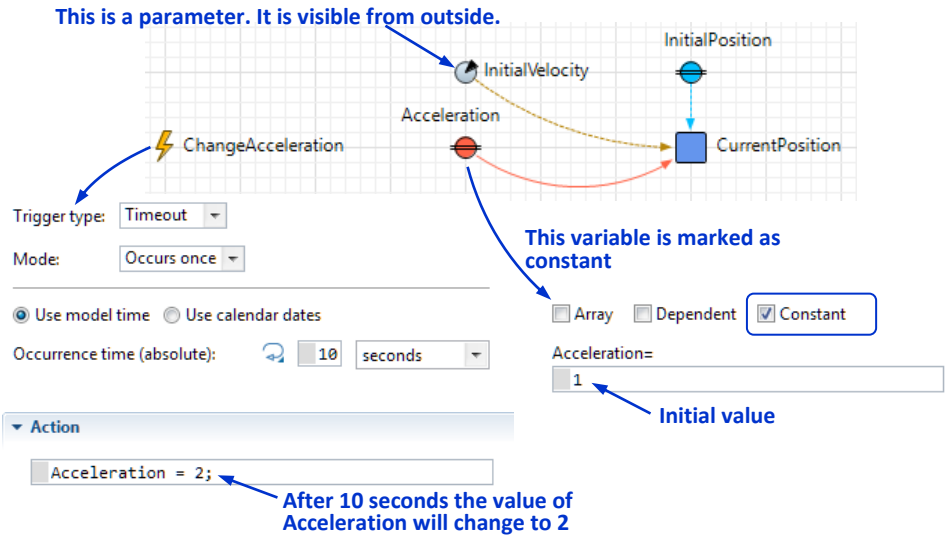


Figure 5.8 Constant variable and event changing its value

Units and unit checking

Each value in the model is measured in certain *units* (or is unitless). You can define new base units; specify units for variables, parameters, and functions; and check if the units are consistent with the formulas. Although these steps are optional and do not affect the simulation results, they are important for keeping track of the physical meaning of the variables and for model verification purposes. Units may be obvious for some variables and not so obvious for others. The next section includes an example of unit specification.

To specify units for a variable, parameter, or function:

1. Expand the **Advanced** section of the variable/function properties and select the checkbox **System dynamics units**.
2. Type the expression for the unit in the field on the right.

The expression can contain the predefined unit **time**, any unit that has previously been defined in the model, or a new unit. The expression may also include the ***** and **/** operations. If you leave the expression field blank, the variable will be considered as unitless. Some typical examples of units are shown in Figure 5.9.

To perform unit checking throughout the model:

1. Select **Tools | Check System Dynamics Units** from the main menu. The results are displayed in the **Problems** view.

The unit inconsistencies are shown as errors. The instances of a variable with undefined units (one with the checkbox **System dynamics units** not selected) in an expression of a variable with defined units is signaled as a warning, see Figure 5.9.

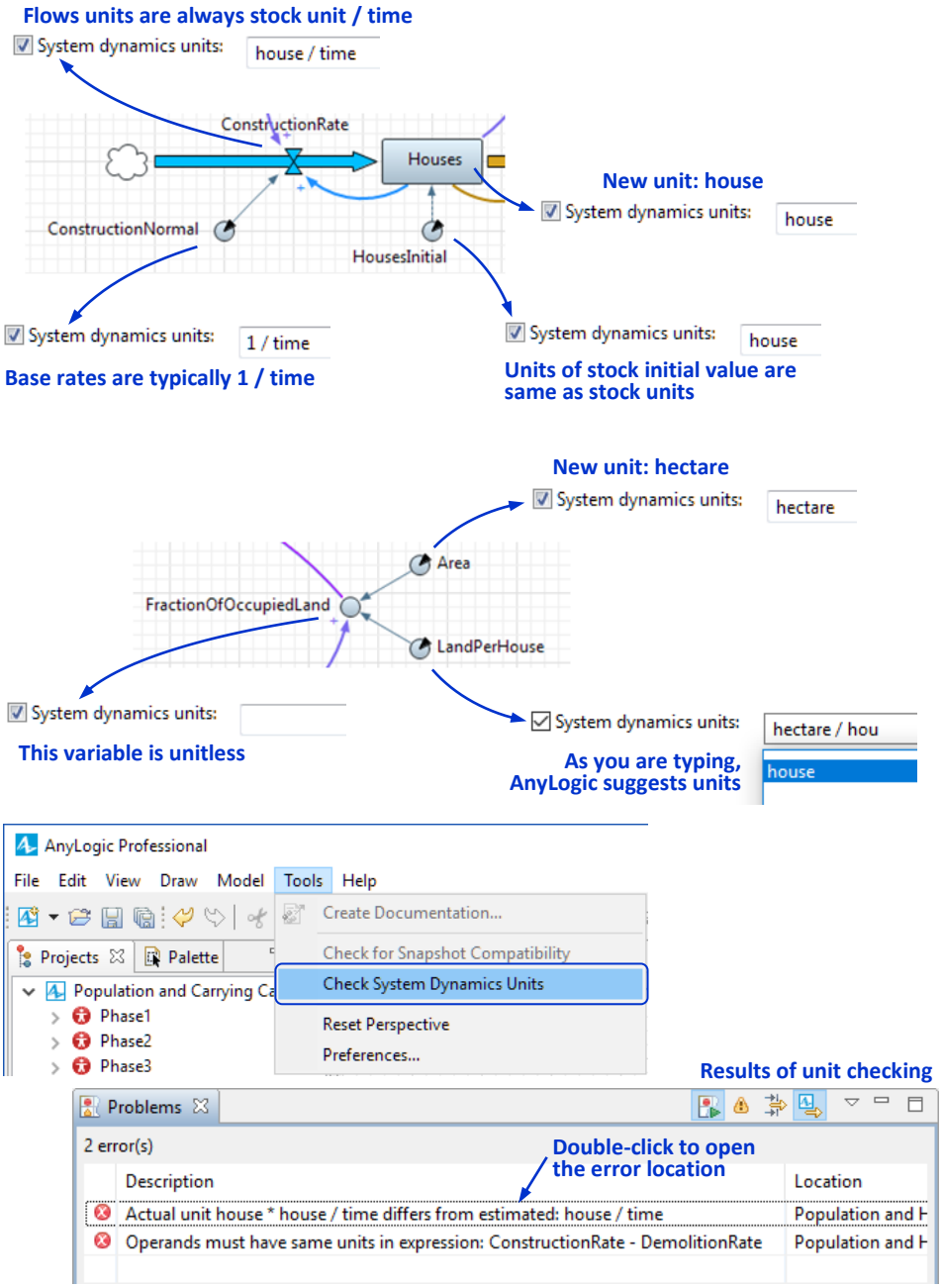


Figure 5.9 Units and unit checking

5.3. Example: Population and carrying capacity

Now, we will build a simple model of population dynamics in a habitat with limited carrying capacity.

> "The *carrying capacity* of any habitat is the number of organisms … it can support and is determined by the resources available in the environment and the resource requirements of the population." (Sterman, 2000)

In our simple model, we will assume the carrying capacity is constant.

Phase 1: Unlimited resources. Positive feedback. Exponential growth

There will be just one stock in the model – the **Population** stock. We will assume a closed system without immigration and emigration, so **Births** is the only inflow to the stock, and **Deaths** is the only outflow. In the first version of the model, we will assume environmental resources are unlimited. Therefore, the birth rate (the average number of new individuals produced by an individual per year) is constant and is at its maximum level. So is the average lifetime.

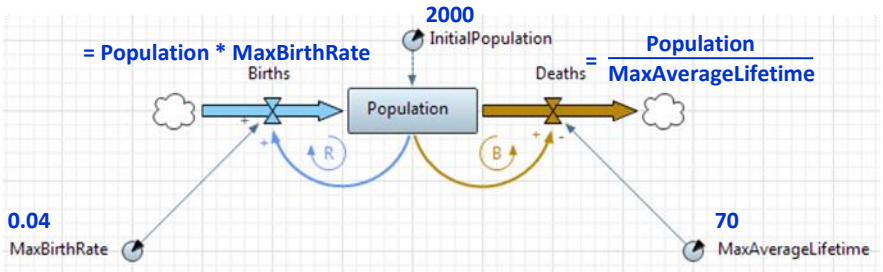


Figure 5.10 The population model. Unlimited resources

▶ **Create the stock and flow diagram:**

1. Create a new model.
2. Open the **System Dynamics** palette and drag the **Stock** into the graphical editor. Change its name to **Population** and resize the stock so that the name can be placed inside.
3. Double-click the stock and drag out the flow to the right. Double-click to finish. Name the outflow **Deaths**.
4. Drag the **Flow** from the **System Dynamics** palette and place it so that its arrow links to the stock. This will be the **Births** inflow.
5. Drag the **Parameter** from the same palette and place it above the stock. Name the parameter **InitialPopulation**. Set the default value of the parameter to 2000.

6. Drag a **Link** so that its beginning connects to the parameter. Connect the end of the link to the **Population** stock.

The connections are indicated as green dots when the link is selected. When a variable is selected, the connected flows and links are highlighted in the color magenta.

7. Select the stock and type **InitialPopulation** in the **Initial value** field of the stock properties.

You do not have to type all the letters of a name. Just start typing and press Ctrl+Space (Mac OS: Option-space) to open the code completion popup window. Then select the name you are looking for from the list.

8. Create two more parameters: **MaxBirthRate**, with the default value 0.04, and **MaxAverageLifetime**, with the default value 70. Place the parameters near the **Births** and **Deaths** flows correspondingly, as shown in Figure 5.10.
9. Go to the the properties of the **Births** flow and type the formula for births: **Population * MaxBirthRate** (remember to use code completion).

Notice the error message that appears once you finish typing the formula: "Population, MaxBirthRate is used but not expected". This message signals the inconsistency between the graphics and the formulas, namely the fact that the influence of the **Population** stock and the **MaxBirthRate** parameter on the **Births** rate is not "confirmed" graphically.

10. Create the two missing links in the same manner as described in step 6. The error message disappears.
11. Similarly, enter the formula for the **Deaths** flow: **Population / MaxAverageLifetime**. Draw the corresponding links.

The first version of the model is ready to run. But first, let us discuss the meaning of the formulas. The formula for **Births** means that each individual produces another one, on average, every 1/0.04 = 25 years (notice that a couple produces twice as many), so each year there will be **Population * 0.04** births. The formula for **Deaths** means the following: an individual will "leak out" of the **Population** stock, on average, in 70 years, so its "individual outflow" is 1/70 per year. Given that there are multiple individuals, their total immediate outflow is **Population * (1/70)** = **Population / 70** deaths per year.

▶ Run and explore the model:

12. Run the model. You can see the current values of the variables displayed underneath their names. The values change as the model is simulated.
13. Click the **Births** flow to open its **Inspect** popup window. Hover the mouse over the window title bar and click the chart icon to display the time chart of the variable.
14. Do the same for the **Population** stock and the **Deaths** flow. Watch the dynamics of the model until the time is approximately 100 years, then click the **Pause** button. The picture looks like that shown in Figure 5.11.

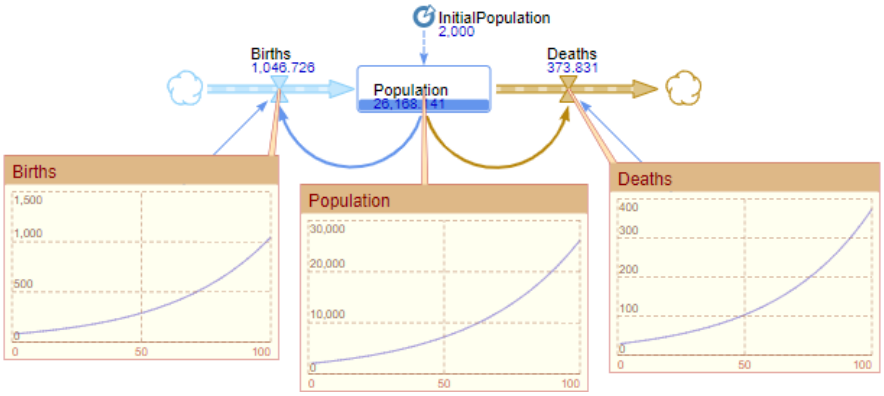


Figure 5.11 You can view the time plot of a dynamic variable in the Inspect window

While the model is executed in such a "slow" mode, you can play with the parameter values using the runtime editors in the **Inspect** window.

15. Open the **Inspect** window of the **MaxAverageLifetime** parameter and double-click the displayed value to make it editable. Change the value to 12 years and continue the simulation. Watch the change in the system dynamics; this is explained later.

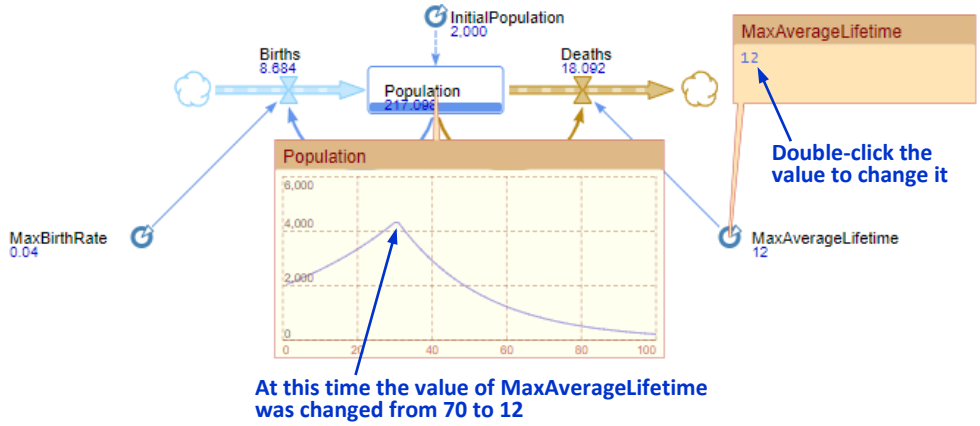


Figure 5.12 The balancing feedback dominates after the parameter change

By default, an AnyLogic model runs in a *scale to real time* (which, in our case, is one model year per second) and has *no specific stop time*. This is not typical for system dynamics modelers, who are used to instant calculations for a limited simulated time period. You can change both settings in the simulation experiment properties.

16. Select the **Simulation** experiment in the **Projects** tree and expand the **Model time** section of its properties.
17. Set the **Execution mode** to **Virtual time (as fast as possible)**.
18. Set the **Stop time** to 100.
19. Run the model again. This time, the simulation is performed almost instantly, and the time charts are available right away.

The charts in the popup windows are useful for rapid exploration of the model. However, you must re-open and re-adjust them in each model run. For the most interesting variables, you can add "permanent" charts.

▶ **Add the time plot of the key variables:**

20. Close the running model and go back to the editor.
21. Open the **Analysis** palette and drag the **Time Plot** underneath the diagram.
22. In the **Data** section of the time plot properties, click the  **Add** button and type **Population** in the **Value** field. Set the title of the item to **Population**, as well.
23. In the **Legend** section of the chart properties, choose the position of the legend on the right of the chart.
24. Ctrl+drag the time plot to create another one below, see Figure 5.13.
25. Delete the **Population** item from the second plot and add **Births** and **Deaths** items.

26. Add the third item to the second plot. Set the value of the item to **Births – Deaths** and the title to **Net Birth Rate**.
27. Run the model and view the plots.

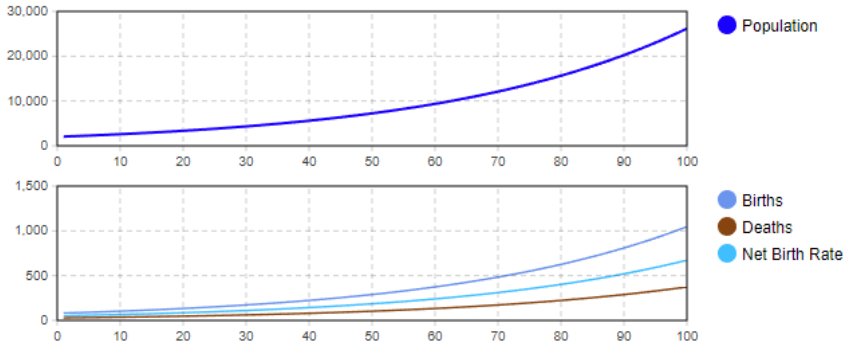


Figure 5.13 Time plots of the key system variables. Exponential growth

We observe *exponential growth* of the population. This exponential growth arises from *positive* (*self-reinforcing*) *feedback*. The larger the population, the more individuals are born, which further increases the size of the population. In our model, this positive feedback interacts with a *negative* (*balancing*) *feedback*: the larger the population, the more people die. Under the default parameter values, the net birth rate is positive, which means the reinforcing loop dominates and results in exponential growth. When you were experimenting with the value of the average lifetime, you saw the configuration where the balancing feedback dominated, see Figure 5.12.

A good rule for system dynamics modeling is to always indicate the link polarities and loop types.

▶ **Set link polarities and indicate the loop types:**

28. Select links one by one and choose their polarities.
29. Open the **System Dynamics** palette and drag the **Loop** icon inside the **Population** -> **Births** -> **Population** loop, as shown in Figure 5.10. Change the loop type to **R** (reinforcing).
30. Create another loop icon for the **Population** -> **Deaths** -> **Population** loop (this time the loop is balancing). Change the loop direction to **Counterclockwise**.

Customizing the dataset collection

In AnyLogic, you can customize the collection of datasets for dynamic variables that are displayed in the **Inspect** windows and in the "permanent" charts. By default, there is a dataset associated with each dynamic variable. The dataset is updated at

every time unit and contains the 100 latest samples. When you open the variable time plot in the **Inspect** window, this is the dataset that shows up.

To change the settings of the dataset auto-collection for dynamic variables:

1. Open the properties of the agent type where the variable is located (the settings are individual for each agent type) and expand the **Advanced** section.
2. If you want to collect the datasets, select the **Create datasets for dynamic variables** checkbox and choose the recurrence time and the dataset length, see Figure 5.14.

The reason for not collecting the datasets is the impact on model performance and memory, which may only matter for very large models.

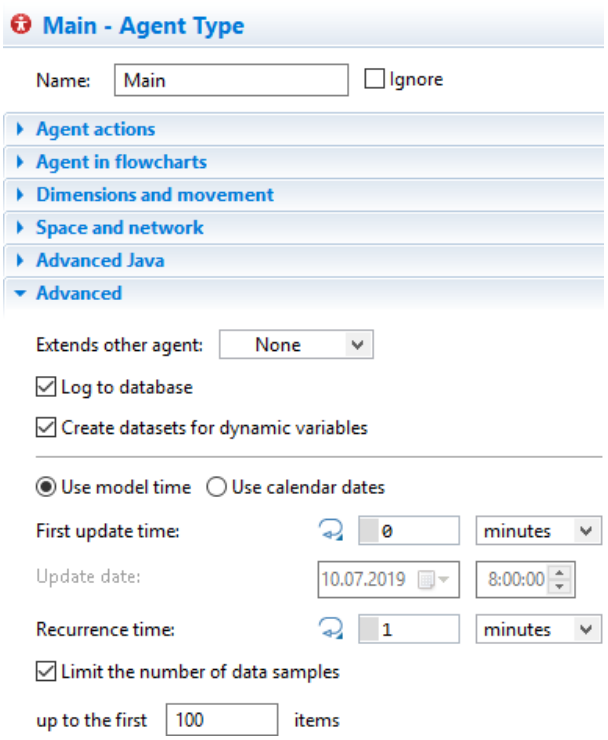


Figure 5.14 Customizing the dataset collection for every dynamic variable

The charts that are added to the model from the **Analysis** palette have their own settings, which can be changed in the chart's **Data update** properties section, see Figure 5.15.

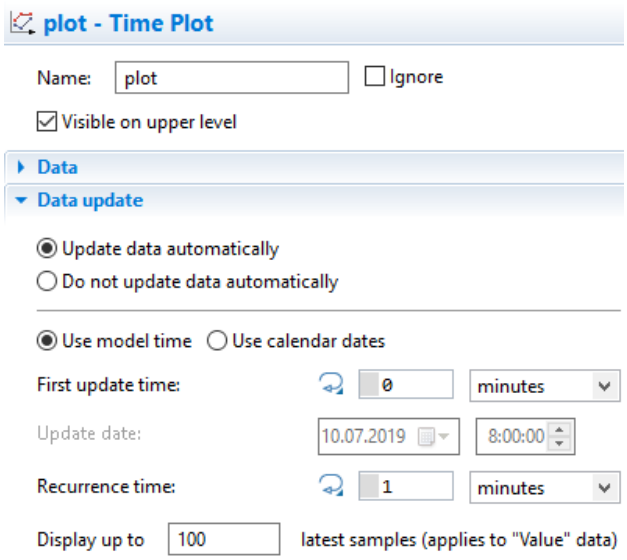


Figure 5.15 Customizing the dataset collection by charts

Phase 2: Crowding affects lifetime. Negative feedback. S-shaped growth

Now we will introduce carrying capacity to our model. We will assume the environment can only support 5,000 individuals, and as the population gets closer to that number, lifetime reduces dramatically. When the population is significantly below the carrying capacity, lifetime is at its maximum level. How do we incorporate such dependency into the model? There are two ways:

- We can put together an analytic formula that has the desired shape.
- We can create a *table function* (*lookup table*) where we can build dependency of an arbitrary shape.

In this phase we will use a table function, and in the next phase we will use a formula to create a similar dependency for the birth rate.

The value, which will be the input to the table function, is the population relative to carrying capacity. We will create the corresponding variable and call it **Crowding**.

Update the feedback structure:

1. Create a new parameter **CarryingCapacity** with the default value 5000.
2. Create a new dynamic variable **Crowding**, which equals **Population / CarryingCapacity**.
3. Create the links from **CarryingCapacity** and from the **Population** stock to the **Crowding** variable (these links should be dragged from the palette). Set up the link polarities.

4. Create another variable **AverageLifetime**. This will be the current average lifetime, which changes dynamically as the crowding grows.
5. Add the **Table Function** from the **System Dynamics** palette and name it **EffectOfCrowdingOnLifetime**.
6. Enter the formula for **AverageLifetime**: **MaxAverageLifetime * EffectOfCrowdingOnLifetime(Crowding)**.
7. Change the formula of the **Deaths** flow to **Population / AverageLifetime**.
8. Correct the link from **MaxAverageLifetime**: it should now point to **AverageLifetime**.
9. Draw the missing links: double-click the **Crowding** variable and place the link arrow on **AverageLifetime**. Then, double-click **AverageLifetime** and create a link to **Deaths**. Set up the polarities.

In AnyLogic, links are drawn only between variables, but *not between functions and variables*. Therefore, although the function **EffectOfCrowdingOnLifetime** is used in the formula of **AverageLifetime**, the link between them is not drawn.

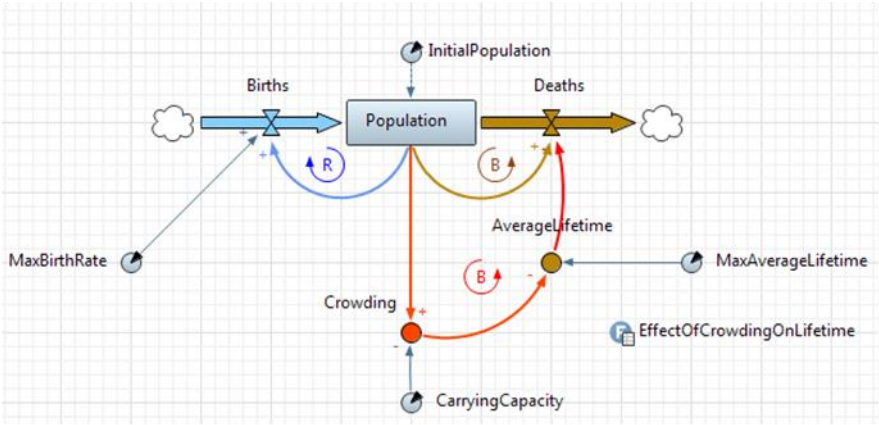


Figure 5.16 The feedback structure, updated to include the effect of crowding on lifetime

Now we need to fill in the argument/value table in the table function. On a real project, you could use historical data. In our exercise, we will make up the function using common sense. Given the formula for **AverageLifetime**, the value range of the function is 0..1 because **AverageLifetime** cannot be negative or be greater than **MaxAverageLifetime**. We will assume that until the population reaches half of the carrying capacity, lifetime remains at its maximum level, at which point it goes down and reaches almost 0 when crowding is at the level of 1.5 (we could use 0 exactly, but then we should have protected the formula of **Deaths** from division by 0).

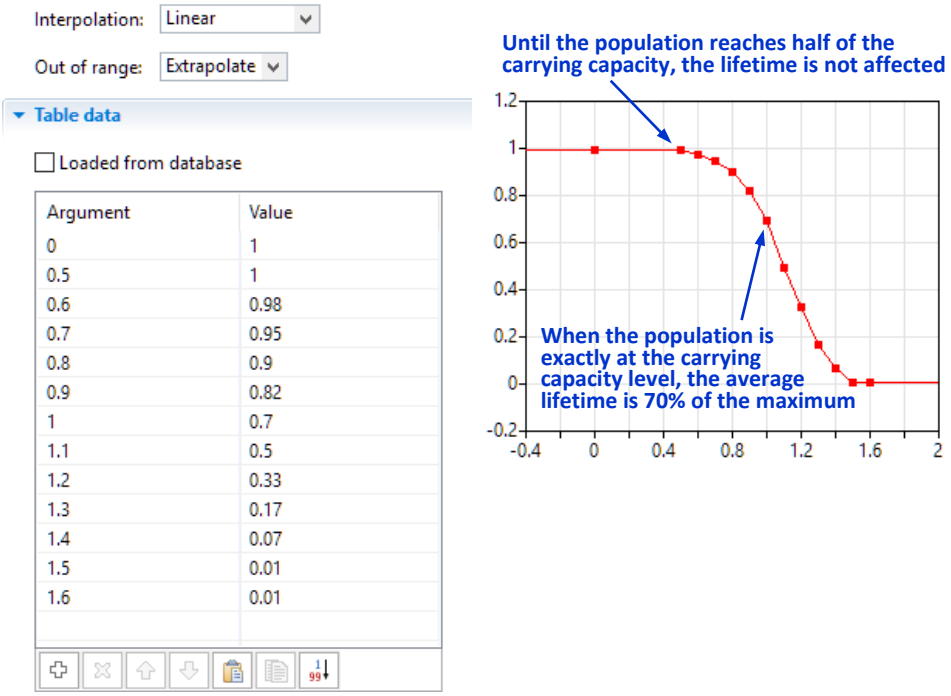


Figure 5.17 The table function EffectOfCrowdingOnLifetime

10. Expand the **Table data** section of the table function properties and fill the table, as shown in Figure 5.17.
11. Set the **Out of range** option to **Extrapolate**. This is necessary because the argument of the function may potentially exceed the last value of 1.6, and we should tell the function how to handle it. The graph reflects the current inter- and extrapolation.
12. Run the model and see the new dynamics.

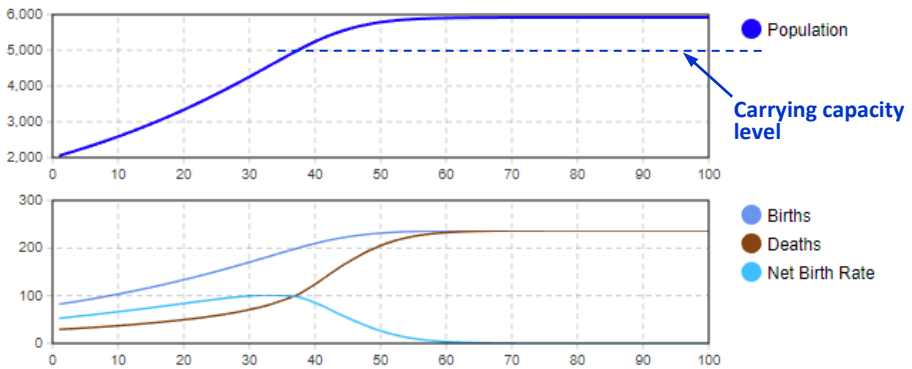


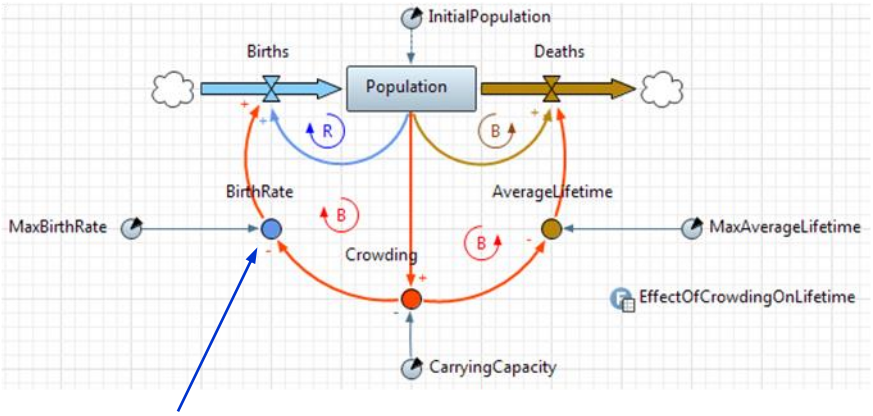
Figure 5.18 The S-shaped behavior of the updated population model

The time charts of the updated model are shown in Figure 5.18. The unlimited exponential growth has changed to *S-shaped growth*. This reflects the fact that the initially dominating positive feedback through births is gradually suppressed by the negative feedback through the effect of crowding on lifetime, which dominates as the system reaches equilibrium. The "power" of that second feedback is defined by the table function. Notice that equilibrium is reached not at the level of the carrying capacity, but above it. Our model, however, is not yet complete, for we have not yet defined the effect of crowding on births.

Phase 3: Crowding affects births

We will assume that crowding affects the birth rate in a similar way to how it affects lifetime: the more crowded the habitat, the fewer individuals born. This time, we will use an analytic function of shape similar to the table function in Figure 5.17. The function and its graph are shown in Figure 5.19. The formula structure has no specific relation to the population dynamics; it just has been chosen because of its graph shape.

AnyLogic supports exponential, logarithmic, power, and many other mathematical functions due to its rich Java heritage.



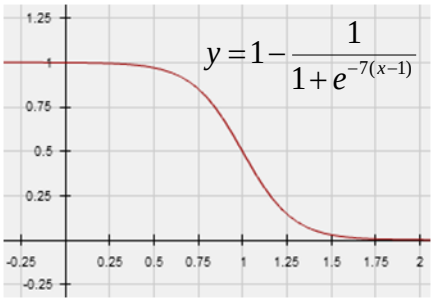


Figure 5.19 The effect of crowding on births is modeled by an analytic function

Update the feedback structure to include the effect of crowding on births:

1. Create a new variable **BirthRate** and change the formula of **Births** to **Population * BirthRate**.
2. Type the formula for **BirthRate**, as shown in Figure 5.19.
3. Update the links structure according to the new dependencies.
4. Run the model.

The behavior is similar to the previous one (see Figure 5.18), but now the equilibrium is reached near the carrying capacity level: at about 4,980 individuals. Births and deaths stabilize at the level of ~100 per year, and average lifetime at about 50 years.

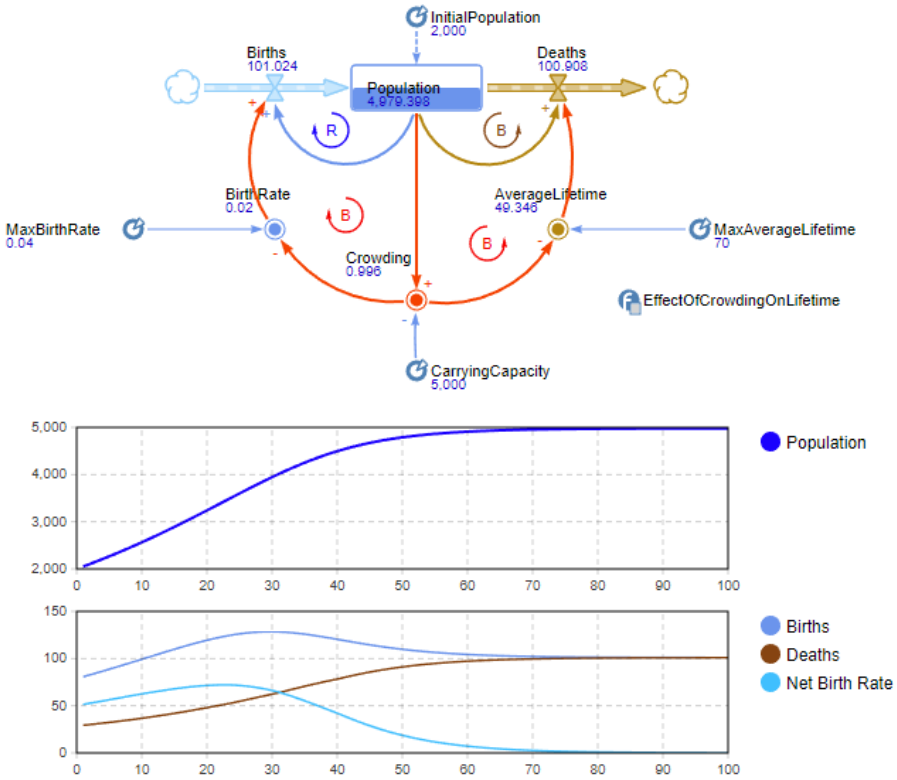


Figure 5.20 The S-shaped behavior of the population model with multiple negative feedbacks

You may wonder why the population size does not stabilize exactly at the level of carrying capacity. The answer is that the *parameter* **CarryingCapacity**, which has a value of 5,000, is not explicitly incorporated into the model as the goal, but affects the behavior via the two functions. The *actual carrying capacity* in our model is 4,980, as defined by the *combination of the parameter and the functions*. You can calibrate the shapes of the functions to achieve the exact value.

Phase 4: Negative feedback with delay. Overshoot and oscillation

Now assume that resource shortage does not affect births immediately, but instead after a certain time delay (this may happen due to some social or physiological reasons). We will incorporate the **delay()** function into the formula of the **BirthRate**.

Add delay to the effect of crowding on births:

1. Add a new parameter, **MaturationDelay**, with the default value 15, and a link from this parameter to the **BirthRate**.
2. Modify the formula of the **BirthRate**, as shown in Figure 5.21.

3. Add the "delay indication" to the link from **Crowding** to **BirthRate** (this is done in the link properties). This is an optional but recommended action.
4. Run the model.

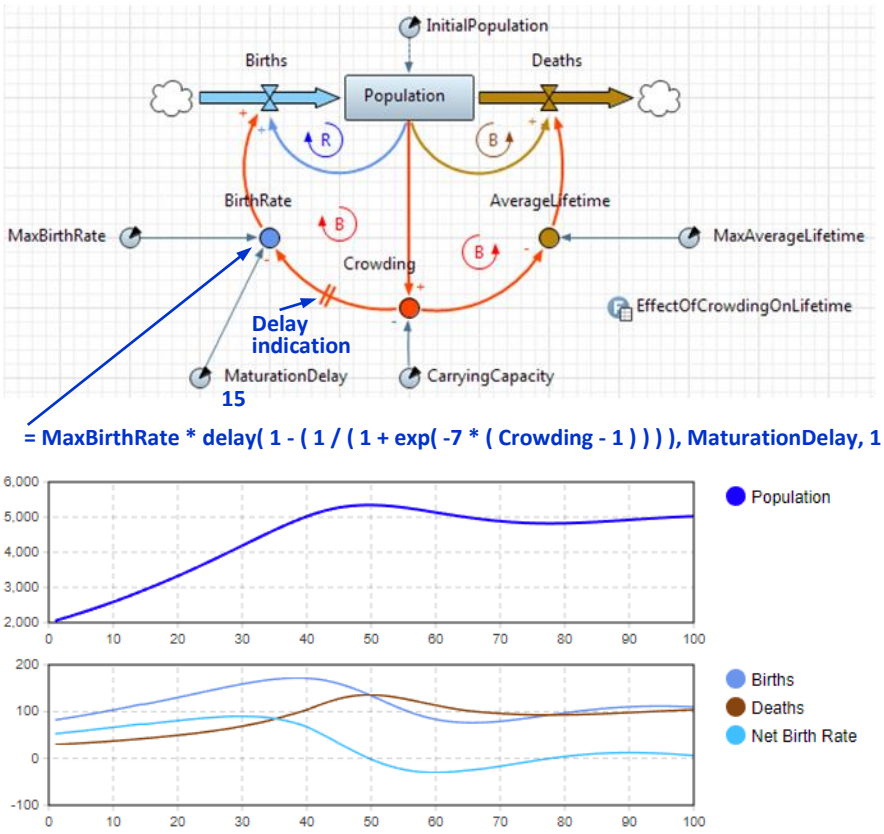


Figure 5.21 Delay in the negative feedback causes oscillation

"Time delays in the negative loops lead to the possibility that the state of the system will overshoot and oscillate around the carrying capacity" (Sterman, 2000), which we observe in our model (Figure 5.21).

The function **delay(x, T, x0)** reproduces the exact behavior of its argument **x** with the delay of **T** time units. The last argument, **x0**, is the value provided as output before the value of **x T** time units ago is known – that is, during the interval **0..T**, see Figure 5.22.

Besides this type of delay, AnyLogic supports several other types, including first and third order delays.

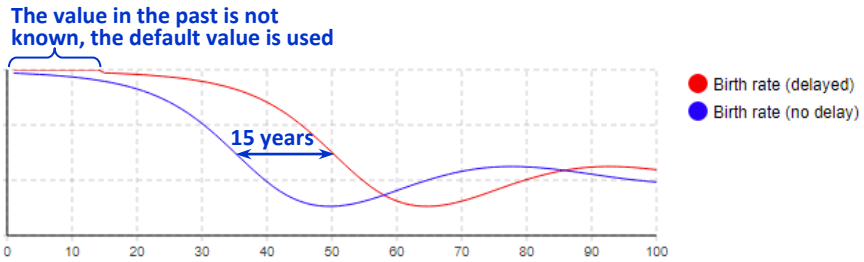


Figure 5.22 The birth rate with and without delay

Specifying units and performing unit checking

We will start with the **Population** stock. The stock contains individuals, so its unit is **individual** (this will be a new custom unit in our model). The **Births** and **Deaths** flows add and remove individuals to/from the stock, and are therefore measured in number of individuals per time unit. Their units are **individual / time**, where **time** is a pre-defined unit in AnyLogic.

This holds for all stocks and flows: a flow to/from a stock is always measured in *<stock unit>* **/ time** because a flow is a part of the stock derivative, i.e. d(Stock) / dt.

The variable **AverageLifetime** and the parameter **MaxAverageLifetime** are naturally measured in **time** units. Given that the unit of **Deaths** is **individual / time**, this is consistent with the formula for **Deaths**: **Population / AverageLifetime**.

Consider the formula for **AverageLifetime**:

AverageLifetime = MaxAverageLifetime * EffectOfCrowdingOnLifetime(Crowding)

What are the units of the table function **EffectOfCrowdingOnLifetime()**? The meaning of the function is to provide a coefficient reducing **MaxAverageLifetime** to a lower value that is currently applicable. The coefficient does not change the units of measurement, so the table function is unitless.

CarryingCapacity is the maximum population that can be supported by the environment, so its unit of measurement is **individual**. **Crowding** is unitless; it is the fraction of the carrying capacity currently occupied.

MaturationDelay is naturally measured in **time** units. The only two variables that may seem to have not so intuitive units are the **MaxBirthRate** and **BirthRate**. At first glance, variables with such names should be measured in **individual / time**. However, **BirthRate** does not flow directly into the **Population** stock: in the formula of the **Births** flow it is a multiplier to population meaning "births per individual per time unit". Therefore, the units for **BirthRate** (and consequently for **MaxBirthRate**) are **1 / time**.

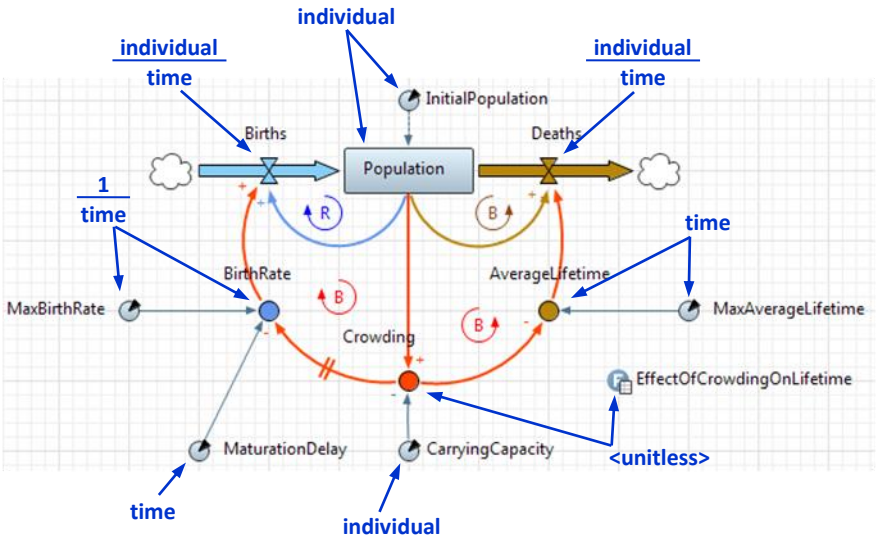


Figure 5.23 Units in the population model

5.4. Conducting experiments in AnyLogic Cloud

AnyLogic Cloud is an online service where you can export the models and share them with members of your own team, clients, and the wider community of AnyLogic users. AnyLogic Cloud is scalable by design, and its high computing capabilities can be leveraged for parallel execution of numerous iterations in multiple-run experiments.

Advanced charts and graphs available in the cloud platform will help you create easily customizable dashboards for complex analytics and configure them to suit your end users.

To illustrate how to upload models and perform experiments in AnyLogic Cloud, we will use a simple model of new product diffusion. The stock and flow diagram for that model is shown in Figure 5.24. Briefly: a company starts selling a new product in the market of a known constant size. Consumers are sensitive to advertising and word-of-mouth. The product has an unlimited lifetime and does not generate repeated purchases. A consumer needs only one product. The model forecasts the sales dynamics.

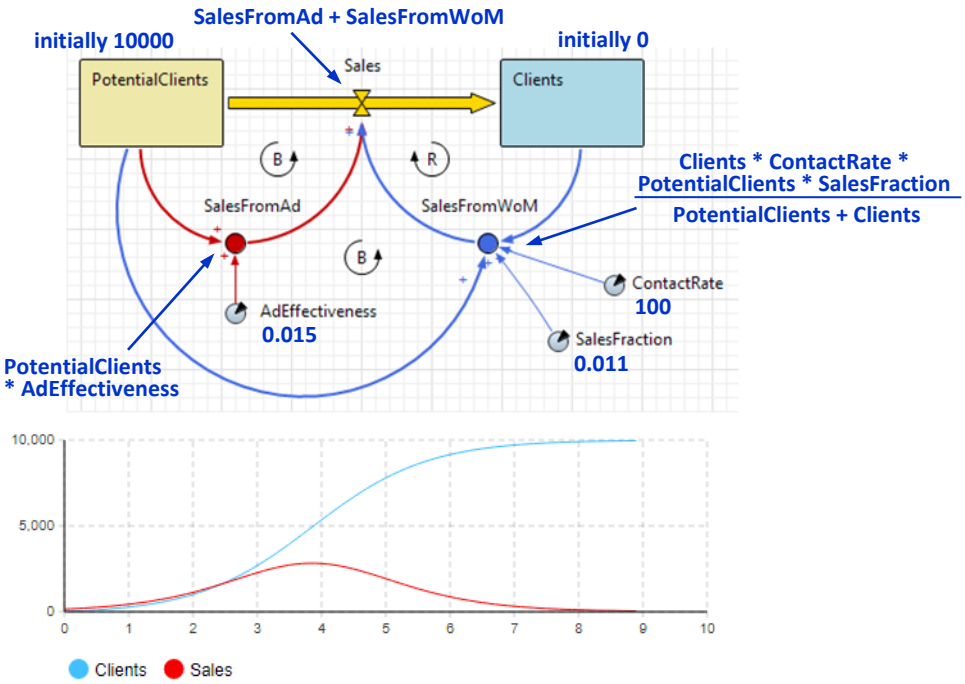


Figure 5.24 The model of new product diffusion and its behavior with the default parameters values

The model exhibits the S-shaped growth caused by the initial dominance of the reinforcing word of mouth feedback, gradually taken over by the balancing market saturation feedback.

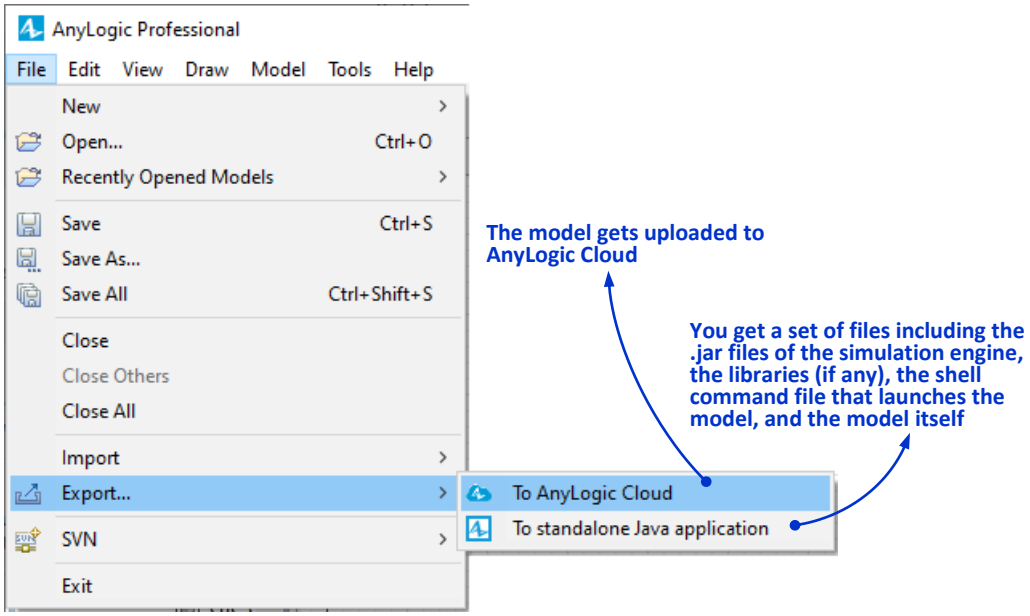


Figure 5.25 AnyLogic model export options

Export model to AnyLogic Cloud

1. Go to the **File** menu and select the **Export... | To AnyLogic Cloud** option as displayed in Figure 5.25 to open the model export wizard.
2. The first page of the wizard is the login page. If you do not have an AnyLogic Cloud account yet, you can create one by following the link on the page. Otherwise, enter the information you have provided at registration and click the **Log in** button.
3. On the next page of the wizard choose the icon for the model. This icon will help you distinguish this model from the others in AnyLogic Cloud at a glance.
4. Decide whether you want the model source files to be available to download for other AnyLogic Cloud users and set the access rights. If you choose to **Provide public access to this model**, the model will be listed in the **Public models** section of AnyLogic Cloud. Otherwise, you will be able to control the access to the model and share it with the selected users via a special link to the model page.
5. On the next page of the wizard specify the model's category (**Test models**), application area (**Market research**), simulation methods (**System Dynamics**), and tags. Tags are needed to improve the search among the uploaded models. For example, for our model the tag could be "Bass Diffusion".
6. Click **Finish** to export the model to AnyLogic Cloud.

7. When the export is finished, AnyLogic Cloud will automatically open in your default browser displaying the model page.

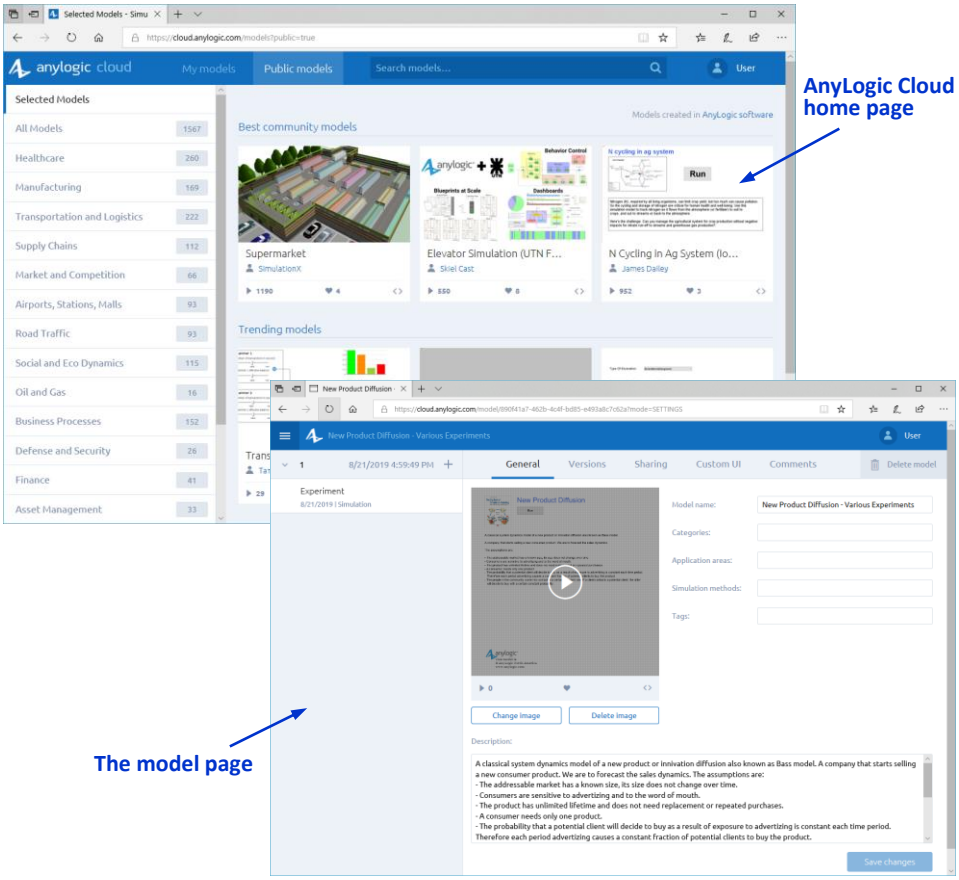


Figure 5.26 The New Product Diffusion exported to AnyLogic Cloud

8. Click the gray widget with the model icon to run the model and watch the animation. If you want to skip the animation and simply obtain the simulation results, select the experiment in the sidebar and click the **Run** toolbar button.

If on export you have denied the public access to the model, but still want to share it with specific people, the sharing settings can be adjusted accordingly.

Configure the model's sharing settings

9. Switch to the **Sharing** tab of the model's page.
10. Type in the e-mail address of the person you want to share the model with and specify the access level. Choose **User** to grant the person view-only

permissions to the model, or **Developer** to grant full ownership permissions. Click the **Invite** button.

11. You can make the model available via the direct link. This does not require the end user's registration in AnyLogic Cloud.

Safety is paramount when you work online, and many companies and industries must follow strict operating guidelines in this regard. The AnyLogic Private Cloud, that could be hosted at your data center, offers different subscription options, if you want to integrate the models in your company workflows while retaining complete control over your sensitive data and its processing.

The exported model runs on any platform and the simulation animation can be easily published on the web via YouTube-style embedding.

▶ **To embed the model animation into a webpage**

12. Display the model tile either in the model page or in AnyLogic Cloud home page.
13. Click the <> icon in the bottom right corner of the tile.
14. Copy the HTML fragment displayed in the dialog box and paste it in your HTML page where you want the model animation to appear.

Only public model animations can be embedded this way, and the animation will always display the simulation of the latest model version.

To develop and run simulations programmatically and build fully customized web interfaces to your models, and query experiment results, you can use the JS, Java, or Python Cloud APIs.

Example 5.1: New product diffusion - compare runs

We will perform the compare runs experiment by manually varying the **ContactRate** parameter and comparing the model behavior. Namely, we will compare the growth of the client base and the sales rate.

First, we need to create the datasets that will contain the history of the key variables at the end of a simulation run.

▶ **Create datasets for the key variables:**

1. Right-click the **Clients** stock and choose **Create Data Set** from the context menu. Place the dataset to the right of the stock and flow diagram.
2. In the dataset properties, select **Update data automatically** with **Recurrence time** 0.1. Given that the simulated period is 10 time units, we will have 100 samples.
3. Do the same for the **Sales** flow.

Before exporting the model to AnyLogic Cloud, where we will create the experiment, we should define the model's inputs and outputs. The inputs expose the model's parameters and allow us to change their values. The outputs are charts and other metrics which will make the model output available for observation and, in our case, for comparison.

Define the inputs/outputs and export the model to AnyLogic Cloud:

4. Double-click the **Run Configuration** item in the **Projects** tree to open the **Run Configuration** editor.
5. In the **Run Configuration** editor, drag the **ContactRate** parameter from the tree on the left to the **Inputs** section. In the same manner add **ClientDS** and **SalesDS** datasets to the **Outputs** section. AnyLogic Cloud will use these datasets to create two charts – one for **Clients** and one for **Sales**.
6. In the **Properties** view, click the **Export model to AnyLogic Cloud** link.

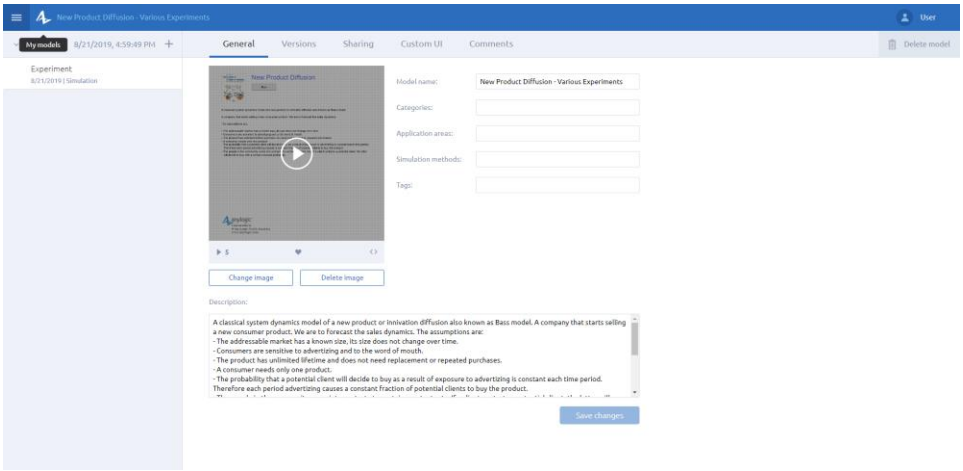


Figure 5.27 Model page in AnyLogic Cloud

In AnyLogic Cloud, the sidebar displays the version number of the exported model with a timestamp. The ⌄ control expands/collapses the version's experiments. By default, each exported model will contain one simulation experiment.

7. Open the experiment's dashboard by clicking the **Experiment** item in the sidebar.

The **Inputs** section of the dashboard displays the controls to modify the values of the parameters we have specified as inputs at export. The **Outputs** section is blurred since the experiment hasn't been run yet and we don't have the outputs data, see Figure 5.28. Once the experiment has been run, this section will contain the charts with the experiment's results.

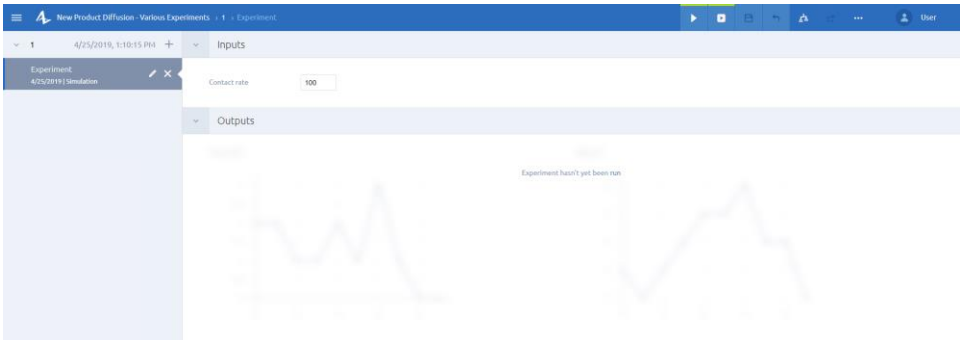


Figure 5.28 Experiment page in AnyLogic Cloud before the first run

To run the experiment, click the **Run** toolbar button. After the run has been finished, the **Outputs** section will display the **Sales** and **Clients** charts.

Compare the model runs in AnyLogic Cloud:

8. Click the **More** control and select **Copy experiment** from the drop-down menu. Change the **Contact rate** parameter value in the **Inputs** section and run the newly created experiment.
9. Create more copies of the experiment, change the parameter value for each of them and run them one by one.
10. Click the **Compare** control. In the sidebar, select the checkboxes next to the experiments you want to compare, see Figure 5.29. The charts displaying the results of all these runs will appear in the **Outputs** section. Each curve in a chart will correspond to a simulation run.
11. The legend is displayed above the chart: each input value is highlighted in the same color as the corresponding curve in the chart.
12. To discard the comparison results, click the **Exit Comparison** control.

Figure 5.29 Compare runs experiment in AnyLogic Cloud

Example 5.2: New product diffusion - sensitivity analysis

Our next experiment type is sensitivity analysis. We will explore how sensitive our model is to advertising effectiveness. We will assume you have the model with the two datasets (**ClientsDS** and **SalesDS**) already created.

Create a sensitivity analysis experiment:

1. Before exporting the model to AnyLogic Cloud, expose the **AdEffectiveness** parameter just like we have done it in the previous example and remove the **ContactRate** parameter from the **Inputs** section.
2. In the export wizard select this model from the **Model** drop-down list to upload a new version of it instead of creating another model.
3. After export, on the AnyLogic Cloud model page, the new version will appear in the sidebar. It will contain the same number of simulation experiment copies as we have previously created to compare model runs. You can delete them since they are obsolete.
4. Click the **New experiment** link on the sidebar. Change the experiment's name to **Sensitivity Analysis**, set the type of the experiment to **Variation** and click **Add**. This experiment will perform multiple parallel model runs varying the input value in the specified range with the specified step.
5. Click the **Dashboard Editor** link in the **Inputs** section of the experiment's dashboard. In the dashboard editor, change the type of **Ad effectiveness** parameter's control to **Varied in range**.
6. Click the  **Save dashboard** control in the sidebar to return to the experiment's dashboard. Specify that the parameter is varied from 0 to 0.2 with the step of 0.01, see Figure 5.30.
7. Click the  **Run** control and view the charts.

AnyLogic Cloud performs a series of runs varying the **AdEffectiveness** parameter. Upon completion, all simulation outputs are displayed in the charts. To see the data for a specific point on the chart, hover the mouse cursor over this point, see Figure 5.30.

Additionally, at the top of every chart there is a toolbar with instruments for panning and zooming, downloading the chart in PNG format, autoscaling, resetting the axes to the default range and displaying the spike lines.

You can export the results of the experiment to a MS Excel file by clicking the  **Export to Excel** toolbar button.

Figure 5.30 Sensitivity analysis experiment in AnyLogic Cloud

5.5. Other types of experiments. Interactive games

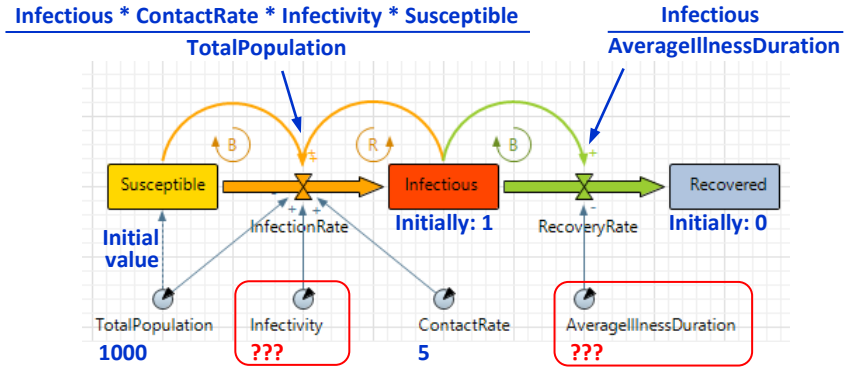
Example 5.3: Epidemic model – calibration

For the first two examples we will use a classic epidemic model *SIR*, or Susceptible Infectious Recovered ("Compartmental models in epidemiology". n.d.), see Figure 5.31. We are modeling a spread of contagious disease in a population where everybody is susceptible and, having been infected, can transmit the disease to other individuals. The disease has a finite duration and, after recovering, an individual becomes immune. The model has two parameters we will focus on:

- **Infectivity** – the probability of disease transmission during contact of an infected individual with a susceptible individual.
- **AverageIllnessDuration** – the average period an individual remains infectious after being infected.

These parameters cannot be measured (at least with reasonable efforts) directly in the real world, in contrast with **TotalPopulation** (which we assume we know) and

ContactRate (which we assume we can measure). Before we use the model, we need to find out the values of the immeasurable parameters. One way of doing that is to *calibrate* the model – use historical data of a similar case and adjust the parameter values so that the model output reproduces the historical data as closely as possible.



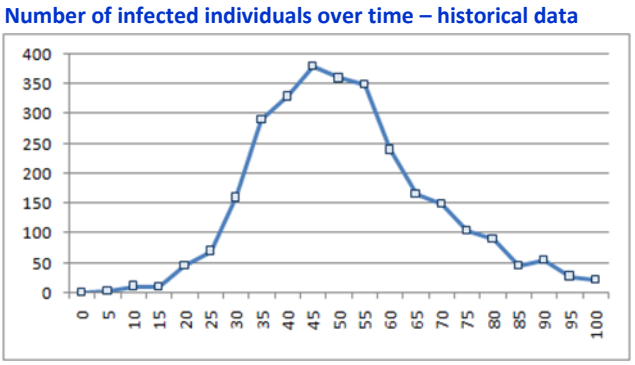


Figure 5.31 The SIR epidemic model and the historical data

The historical data is shown in Figure 5.31 as the number of infected people over time in a population of a thousand people. To calibrate the model, we will use the OptQuest optimizer built into AnyLogic, which will iteratively run the model, compare its output with the historical curve, change the parameter values, perform another run, etc., until it decides it has done its best (or the iteration limit is reached).

Create the SD model:

1. Create the SIR model, shown in Figure 5.31. Use some values for **Infectivity** and **AverageIllnessDuration**, say 0.08 and 12.
2. In the **Model time** properties section of the default simulation experiment (the one that has been created automatically), set **Stop** to **Stop at specified time** and **Stop time** to **100**.

3. Run the model (run the default simulation experiment) to make sure it works as expected and the dynamics of **Infectious** stock is bell-shaped.

Before we do the calibration, we need to create a dataset in the model that will keep the model output at the end of a simulation run.

Create a dataset from the stock

4. Right-click the **Infectious** stock and choose **Create Data Set** from the context menu. A dataset **InfectiousDS** is created.
5. In the dataset properties, switch the update mode to **Update data automatically**.

Create a calibration experiment:

6. Right-click the model (top-level) item in the **Projects** tree and choose **New | Experiment** from the context menu.
7. In the first page of the wizard, choose the **Calibration** experiment type and click **Next**.
8. In the **Parameters and Criteria** page of the wizard, in the **Parameters** table change the **Type** of the parameters we are calibrating from fixed to continuous. Set the range of the **Infectivity** to 0 .. 0.1 and the range of **AverageIllnessDuration** to 3 .. 30.
9. In the **Criteria** table, fill in one row, as shown in Figure 5.32. Click **Finish**.

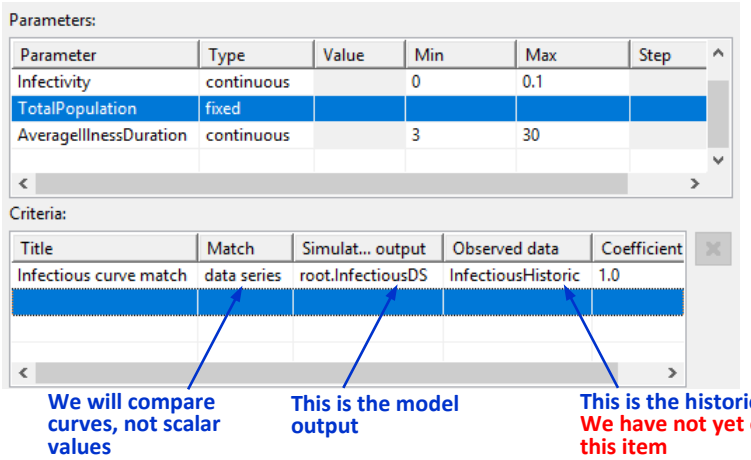


Figure 5.32 Settings of the calibration experiment

The calibration experiment with the default UI has been created. If you compile the model, you will get the error "InfectiousHistoric cannot be resolved to a variable". This is because we have not created the item in the model that contains the historical data, although we have already indicated that this item will be used.

Create a table function and import the historical data:

10. Open the editor of the **Calibration** experiment and resize the charts to fit them inside the blue frame.
11. Drag the **Table Function** from the **System Dynamics** palette to the graphical editor. Change the name of the function to **InfectiousHistoric**. Leave the default **Interpolation** and **Out of range** settings.
12. Expand the **Table data** section of the table function properties and provide the historical data, as shown in Figure 5.33. If you have the data in a spreadsheet or a text editor, you can copy the data on the clipboard and click the **Paste from clipboard** button below the table.

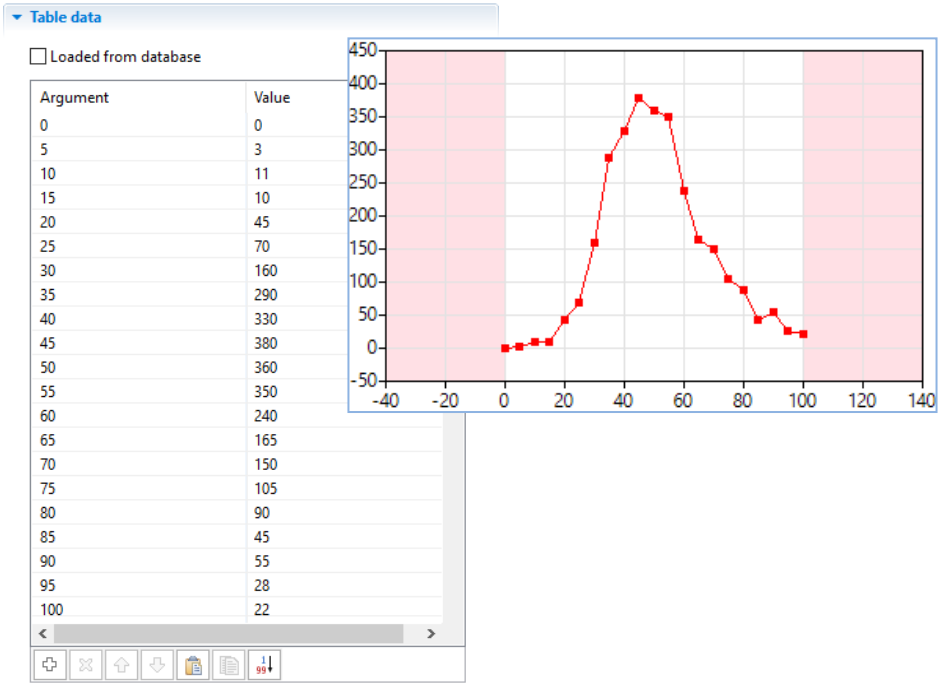


Figure 5.33 The table function InfectiousHistoric

13. Run the **Calibration** experiment. Watch the progress of calibration. It is beautiful!

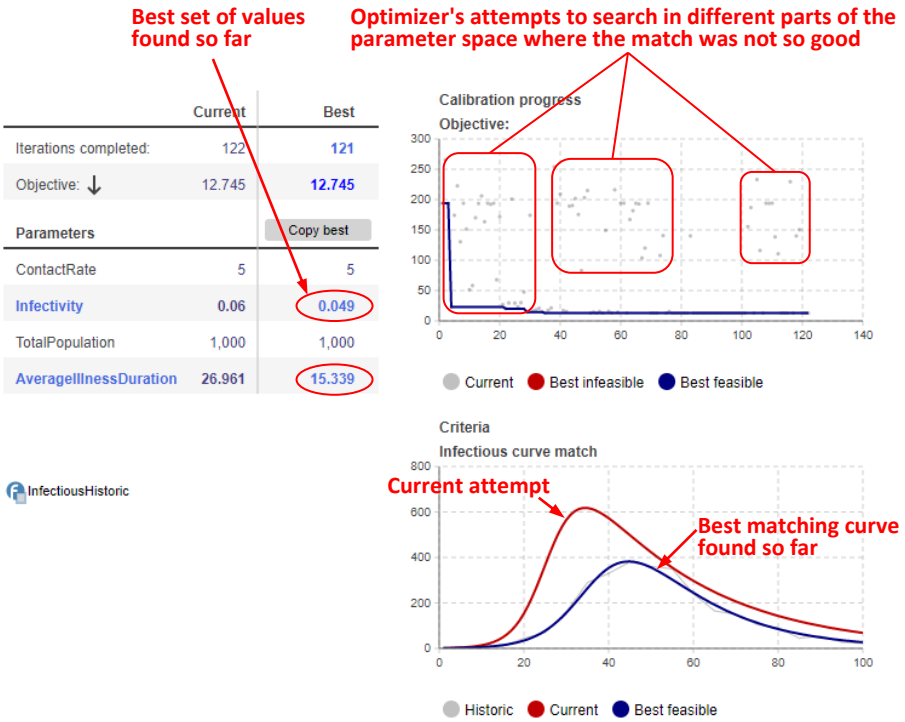


Figure 5.34 Calibration experiment in progress (iteration 122 of maximum 500)

The values of **Infectivity** and **AverageIllnessDuration** found in the calibration experiment are about 0.049 and 15.34. You can copy the values from the calibration experiment (see the **Copy best** button in Figure 5.34) and paste them to the simulation or any other experiment by clicking **Paste from clipboard** in the **Parameters** section of the experiment properties.

Example 5.4: Epidemic model - instant charts

A typical scenario in system dynamics modeling is the instant re-simulation of the model as the user changes the parameters and the output charts immediately reflecting the change. You can create such a scenario based on the standard experiment compare runs.

We will expose the three parameters of the model (**Infectivity**, **ContactRate**, and **AverageIllnessDuration**) to the user as three sliders in the model window. The chart in the same window will show the simulation results immediately after the user has changed the parameters.

Prepare the model and create a compare runs experiment:

1. In the **Main** agent of the SIR model, add datasets for all stocks. Do it in the same way we did it for the **Infectious** stock in the previous example. You should get **SusceptibleDS**, **InfectiousDS**, and **RecoveredDS**.
2. In the properties of the datasets, select **Update data automatically**.
3. In the **Value editor** properties section of the parameters **Infectivity**, **ContactRate** and **AverageIllnessDuration**, choose the **Slider** control type. Set the slider ranges:
   for **Infectivity**: from 0.01 to 0.20
   for **ContactRate**: from 1 to 20
   for **AverageIllnessDuration**: from 3 to 30.
4. Right-click the model item in the **Projects** tree and choose **New | Experiment** from the context menu. In the wizard choose the **Compare Runs** experiment type, change its name to **InstantCharts**, and click **Next**.
5. In the **Parameters** page of the wizard, choose the three parameters you are going to vary (all except for **TotalPopulation**) and move them to the **Selection** list. Click **Next** and, on the next page, click **Finish**. A new experiment is created, and you can see its default UI with three sliders, see Figure 5.35.
6. Delete the unnecessary graphics, as shown in the same Figure 5.35.
7. In the **InstantCharts** experiment, create three datasets with the same names as the datasets in the **Main** agent: **SusceptibleDS**, **InfectiousDS**, and **RecoveredDS**. In the properties of the datasets, deselect the option **Use run number as horizontal axis value**. These three datasets will be used to copy the data from the latest simulation run.
8. Create a plot (drag the **Plot** element from the **Analysis** palette) and add to it the three datasets which requires switching each data item in the plot properties from **Value** to **Data set**. Adjust the line colors as you wish.

Figure 5.35 Auto-created and modified UI of the compare runs experiment

Now we have the controls to vary the parameters and the chart that is ready to display the simulation results. To complete the model, we only need to:

- Modify the slider actions so that each time the slider is moved, a new simulation run is performed.
- Upon completion of a simulation run, re-fill the datasets displayed in the plot with the most recent model output.

Set up the slider actions and add an action executed after each simulation run:

9. Add the function call **run();** at the end of the action of each slider.
10. Expand the **Java actions** section of the **InstantCharts** experiment properties and type this code in the **After simulation run** field:
    ```
    SusceptibleDS.fillFrom( root.SusceptibleDS );
    InfectiousDS.fillFrom( root.InfectiousDS );
    RecoveredDS.fillFrom( root.RecoveredDS );
    ```
    This code deletes the current contents of the datasets at the experiment level and copies there the content of the datasets in the model (**root** refers to the top-level agent of the model, **Main**).
11. Run the **InstantCharts** experiment and use the sliders to change the parameters. The plot instantly displays the simulation output.

Note that the **run()** function in the compare runs experiment performs a simulation run in the fastest possible mode with animation turned off.

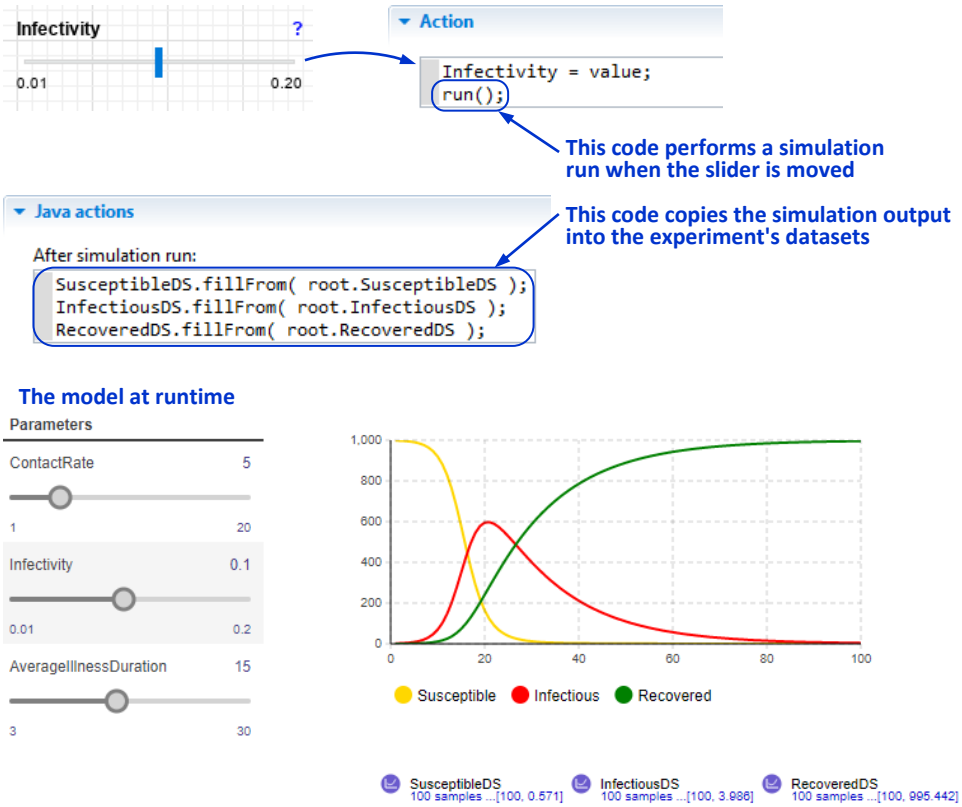


Figure 5.36 The last steps in experiment setup. The completed model at runtime

Example 5.5: Stock management game

We will now build a *game*, or a *flight simulator* – an interactive model that exposes its output to the user as the simulation is being performed and allows the user to make decisions; for example, to change parameters in a running model and observe the effect.

Typically, such games work in step-pause mode: the model executes for a certain period, then it is put in the paused state, at which time the user observes the output, makes decisions, and starts the simulation for the next period. This may make sense when the real system is also controlled only at certain regular points in time – for example, each month or each quarter. In addition to that scenario, AnyLogic allows you to build games where the model executes in a scale to real time, and the user can make changes continuously, at arbitrary moments. We will first implement the latter scenario, and then add steps and pauses.

The model will be a very simple supply chain, see Figure 5.37. The user is to manage the stock by controlling the order rate. The sales rate is exogenous and will be modeled as randomly changing.

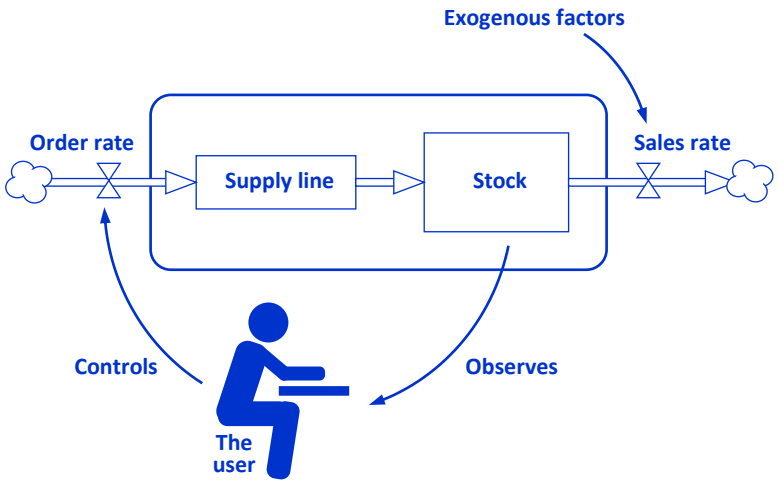


Figure 5.37 Architecture of the stock management game

▶ **Create the system dynamics model:**

1. Create a new model. In the **New Model** wizard, set **days** as the **Model time units**.
2. Create the stock and flow diagram, as shown in Figure 5.38.

Note that both **OrderRate** and **Demand** are marked as constants (see Section 5.2): therefore, they only have initial values, and no equations. These two "variables" will

be controlled from outside the system dynamics model: **Demand** will be changed by a recurrent discrete event, and **OrderRate** will be controlled by the user.

The equation of **SalesRate** uses the conditional operator. While there is product in stock, it sells at the **Demand** rate, otherwise nothing is sold.

3. Add the event **ExogenousDemandChange** and set it up to occur at every time unit (day). In the **Action** field of the event, type the following code:
```
Demand = max( 0, Demand + uniform( -1, 1 ) );
```

This code increases or decreases the value of the **Demand** variable by a random amount, uniformly distributed between -1 and 1. The **max()** function protects **Demand** from falling below zero.

Do not confuse time unit and time step. The time unit is the unit of simulated time and can correspond to an hour, day, week, etc. This correspondence is established in the properties of the model. The time step is a micro-step of numerical methods in the AnyLogic simulation engine. It is set up in the **Advanced** properties of the experiment and is typically much smaller than the time unit (say, 0.001).

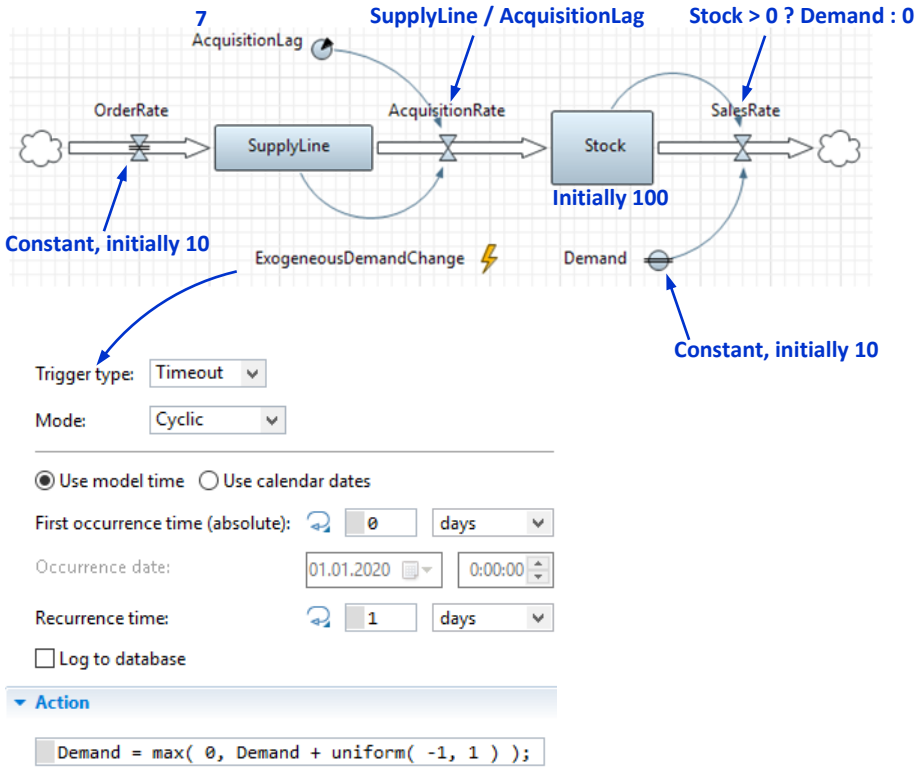


Figure 5.38 The stock management model without the controlling feedback

4. Add a **Time Stack Chart** from the **Analysis** palette. Add a single data item with the expression **max(0, Stock)** (**Stock** still can fall slightly below zero due to numeric errors.). Set the chart's **Time window** to 1,000 and the number of samples to display to 1,000 as well.
5. Select the **Simulation** experiment in the **Projects** tree and expand the **Model time** section of its properties. Set the **Execution mode** to **Real time with scale** 50. This is a better speed for our game.
6. Run the model and see how the stock changes over time.

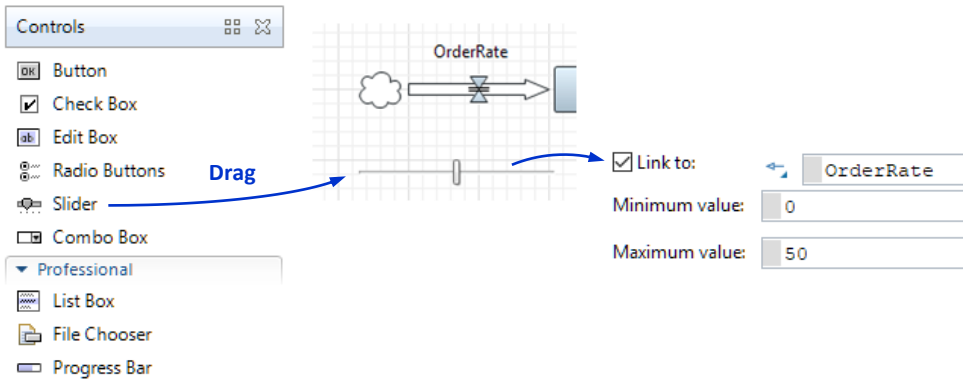


Figure 5.39 Slider – an element of the user interface

Close the control loop: create a slider that varies the order rate:

7. Drag the **Slider** from the **Controls** palette and link it to the **OrderRate** variable. Set the slider range to 0..50, see Figure 5.39.

If you link a slider to a dynamic variable *with an equation* (the one that is not marked as constant), the slider's attempts to change the variable will be unsuccessful – the equation will immediately override the value assigned by the slider.

8. Run the model and try to keep the stock within a certain interval – say, between 100 and 500.

In our current setup, the user can see the entire stock and flow diagram with all the variable values. This is not always desirable, and it is up to the game designer to decide which variables are to be exposed to the user. In the next step, we will create a separate interface page that will contain only the selected model output and the control elements.

Create a separate interface page using view area

9. Drag the **View Area** from the **Presentation** palette and place it at (0, 600). This will be the top left corner of the user screen. Give the view area the name **viewUserScreen** and the title "User screen".

As the model window size is 1000 by 600 pixels (these settings can be checked and changed in the properties of the frame), the Y-coordinate of the view area, 600, ensures the user screen does not intersect with the model screen.

10. Move the chart and the slider to the user screen, i.e., below the Y coordinate 600.
11. In the slider properties, click the button **Add labels**.

12. In the **Agent actions** section of the **Main** agent properties, type the following code in the **On startup** field:
    `viewUserScreen.navigateTo();`
    This will display the user screen right at the beginning of the simulation.
13. In the **Window** section of the **Simulation** experiment properties, deselect the checkbox **Enable zoom and panning**. This will prevent the user from occasionally moving the canvas.
14. Run the model. Now you can only see the stock time chart and the slider.

You can still switch to the model screen by choosing **[Origin]** from the **Select view area to navigate** drop-down list in the developer panel. To disable the developer panel, go to the **Window** section of the simulation experiment properties and clear the **Enable developer panel** checkbox.

As a final phase of this exercise, we will implement the step-pause game mode. The step duration will be 50 days. To schedule the pause at the end of a 50-day period, we will use a cyclic event and the AnyLogic engine function **pauseSimulation()**. To indicate that the user has finished making decisions and to run the simulation again, we will use a button and the function **runSimulation()**.

▶ **Implement the step-pause game mode:**

15. Expand the **Model time** section of the **Simulation** experiment properties and change the **Execution mode** to **Virtual time (as fast as possible)**. (If you want the charts to progress with some animation effects, you can stay within the real time mode with a higher scale.)
16. Open the editor of the **Main** agent and drag the **Button** element from the **Controls** palette. Place the button on the user screen and change its label to "Done." Type the following condition in the **Enabled** field of the button:
    `getEngine().getState() == Engine.PAUSED`
    and type this function call in the **Action** field:
    `runSimulation();`
17. Copy the condition in the **Enabled** field of the button and paste it to the **Enabled** field of the slider. These conditions ensure that the user will only be able to change the order rate when the model is in the paused state.
18. Drag the **Event** element from the **Agent** palette. Set the event mode to **Cyclic**, the **First occurrence time (absolute)** to 50 days, and the **Recurrence time** to 50 days. In the **Action** field of the event, type: `pauseSimulation();`
19. Run the model. The model now works in step-pause mode, allowing the user to change the order rate once every 50 days.

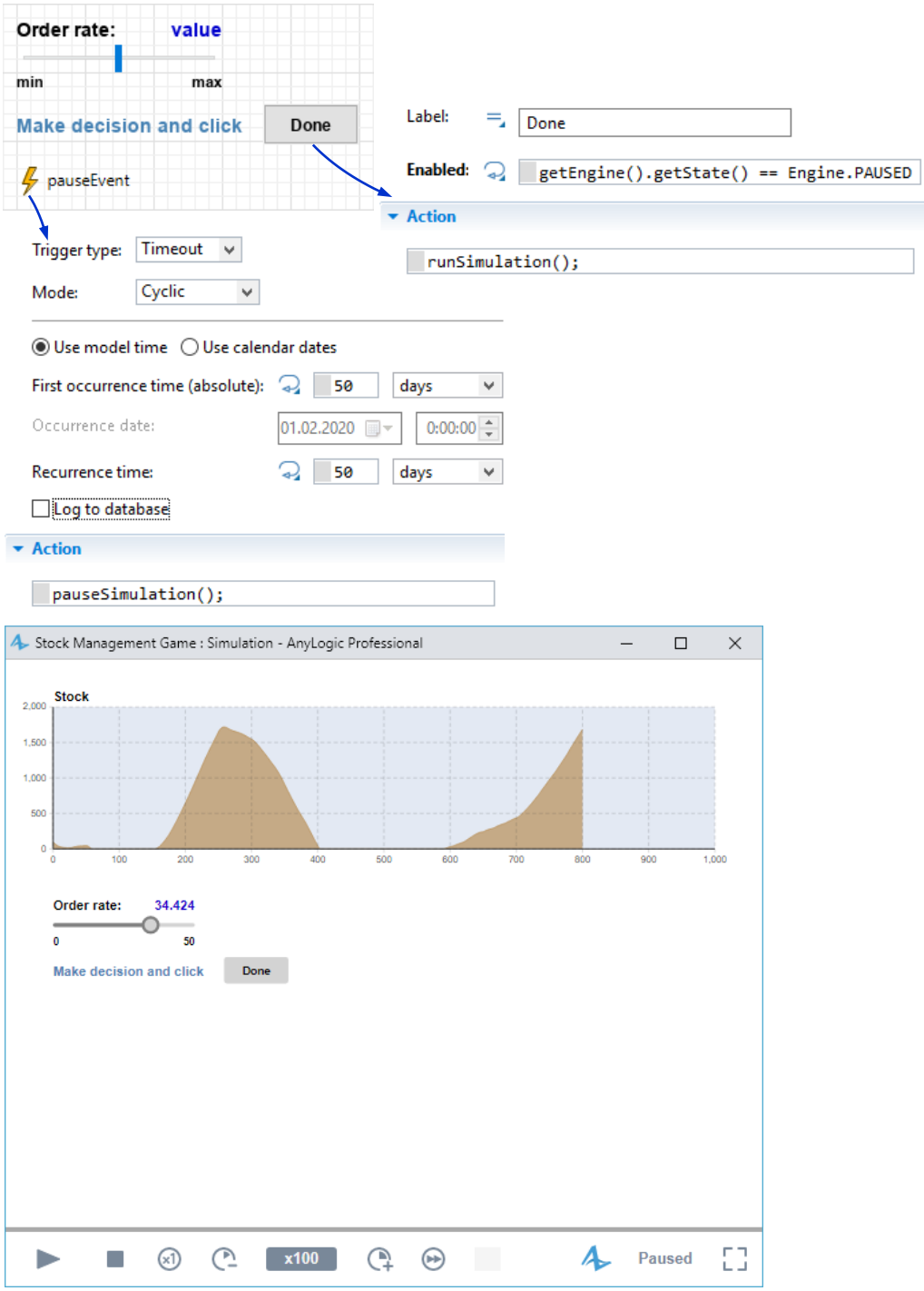


Figure 5.40 Implementation of the step-pause game mode