Chapter 12. Presentation and animation: working with shapes, groups, colors

AnyLogic offers a rich set of graphics tools to design the visual 2D and 3D front-ends of your models. The tools include various shapes (rectangles, polylines, text shapes, 2D images, etc.), controls (buttons, sliders, text boxes, etc.), 3D-specific elements (cameras, lights, 3D objects, etc.), and data visualization elements (charts and plots). This chapter explains how to draw 2D shapes and tells about basic animation principles.

AnyLogic is a dynamic simulation tool, and of course the picture that you draw in the AnyLogic graphical editor can be animated. Any property of any shape (such as size, position, or color) can be varied during the model runtime to reflect the dynamics of the model. Shapes can also be dynamically replicated, hidden, shown, or even created and deleted.

Graphics can do more than just decorate the model or visualize its dynamics. The shapes created by the user are accessible to the model elements so that the latter can behave according to the graphical configuration.

You can consider the model elements (process flowcharts, statecharts, events, stock and flow diagrams, etc.) and the graphical elements (shapes, images, groups, etc.) as living in the same space and being able to access and control each other.

12.1. Drawing and editing shapes

Basic shapes are located in the **Presentation** palette and the simplest way to create a shape is drag-and-drop to the graphical editor.

▶ **To create a new shape:**

1. Open the **Presentation** palette.
2. Drag the shape from the palette and drop it to the graphical editor.

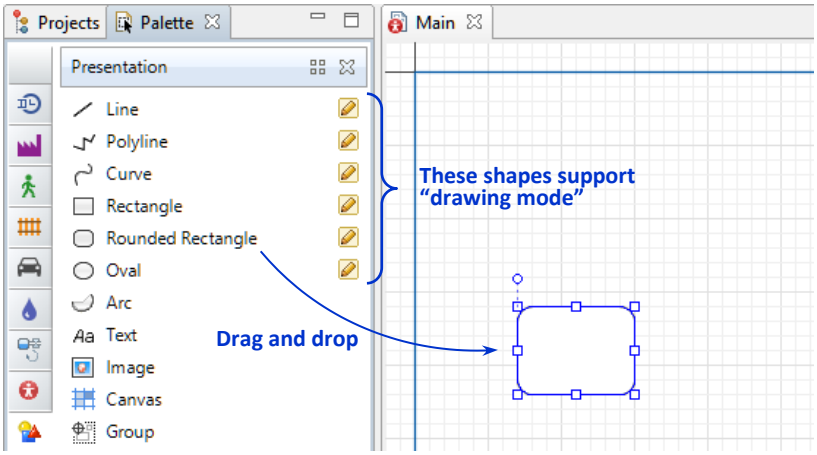

Figure 12.1 The Presentation palette

The new shape created this way has a default size and you can then adjust it as needed using handles. Alternately, you can create a shape by entering the *drawing mode*. The shapes that support drawing mode have a special icon with a pencil on the right-hand side in the palette.

To create a new shape using the drawing mode (except for polyline and curve):

1. Double-click the shape icon the palette.
2. Click in the graphical editor where the upper left (or first) point of the shape should be placed.
3. Hold the mouse button; drag the cursor to the bottom right (or second) point of the shape.
4. Release the mouse button.

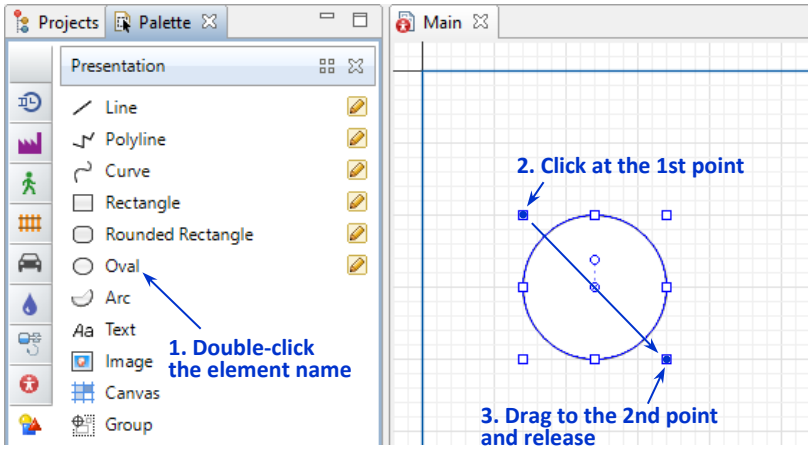

Figure 12.2 Using Drawing mode to draw shapes

You can draw both the oval and the circle using the same element **Oval**. The choice of shape is specified in the property **Type**.

Polylines and curves

Polylines and curves are drawn and edited in a slightly different way.

To draw a polyline or a curve:

1. Double-click the polyline or curve icon in the palette.
2. Click in the graphical editor to place the first point of the polyline.
3. Continue clicking at all but the last point positions.
4. Double-click at the last point position to exit the drawing mode.

To create a closed polyline or curve you should *not* create its first and last points at the same position. Instead, you should check the **Closed** checkbox in its properties. The last segment is then drawn automatically.

Polylines and curves can be edited after creation; you can move, add and delete individual points. To do it, you need to enter the *edit points* mode.

To edit individual points of a polyline or a curve:

1. Select the polyline or curve. The way the polyline (curve) is highlighted should change so that you can see the points. We are now in the "edit points" mode.
2. To add a new point, double-click the segment where the point must be added.
3. To delete a point, double-click it.
4. To move a point, drag it to the new position.

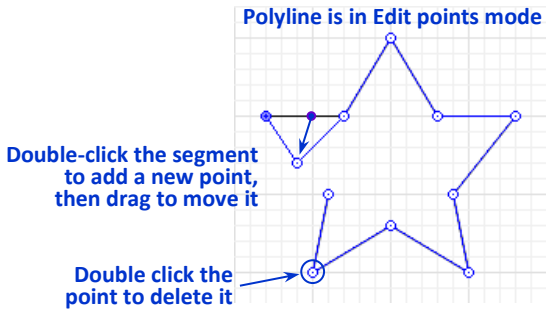

Figure 12.3 Editing points of a polyline

The points of a polyline can also be edited in the table in the **Points** section of the polyline properties.

To rotate a polyline or a curve

1. Right-click the shape and choose **Edit shape** from the context menu.
2. Click the handle that appears in the center of the shape and rotate it to achieve the intended position of the shape.
3. Release the mouse button when the position has been achieved.

When you rotate a polyline or a curve in the graphical editor, the editor just recalculates the point coordinates. There is no way to rotate the shape in design time via the shape properties, and the **Rotation, rad** field in the **Position and size** properties section is only used to achieve the runtime rotation.

Arcs

Arcs have a *start angle* and an *extension angle*. While the size and position of an arc can be edited graphically by using handles, the angles are edited in the **Position and size** section of the arc properties. The angles are counted from 3 o'clock clockwise.

By default, the arc is visualized as a segment of a **Circle**. You can change its **Type** to **Oval** in the arc's properties.

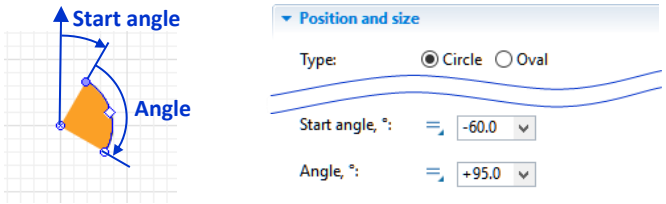

Figure 12.4 Editing arc angles

Text

All the properties specific to the text shape are edited in the **Text** and **Appearance** sections of its **Properties** view.

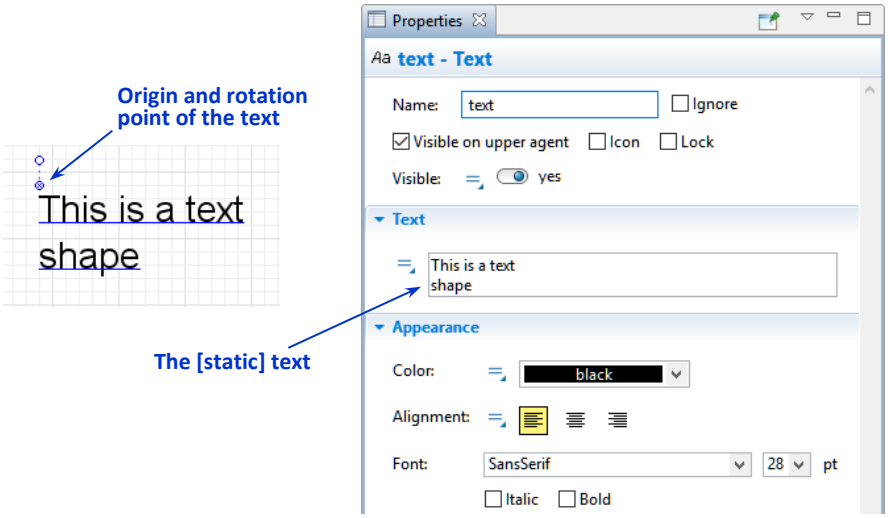

Figure 12.5 Text shape

The text will extend downwards from its origin. Depending on the alignment, it will be left justified to the right of the origin (as shown in Figure 12.5), right justified to the the left of the origin, or centered to the origin. The line spacing is chosen automatically.

The text that is entered in the **Text** field of the properties will show in design time in the graphical editor and at runtime. To have the displayed text change during runtime, switch the **Text** field to the **Dynamic value** mode and provide the appropriate dynamic expression in the field. For more information on dynamic properties, see Section 12.3, “Animation principles. Dynamic properties of shapes".

Images

When you drag the **Image** element from the palette to the graphical editor, you create a placeholder for your images. The placeholder in general may contain several images, and you can switch between them at runtime (at design time however you will see only one). This may be useful when two or more different images correspond to different (and alternative) states or modes of an object.

To add an image:

1. Drag the **Image** element from the **Presentation** palette to the graphical editor. A placeholder for the image is created and the **Open** dialog pops up.
2. Choose the image file and click **Open**. The image appears in the image properties and in the graphical editor.
3. If needed, adjust the size of the image, or check **Original size** checkbox in the image properties to preserve its original size.

4. To add more images (that you intend to switch at runtime with the first one), click **Add image** button in the **Images** list and repeat steps 2-3.
5. To change the order of images, use the arrow controls, see Figure 12.7.

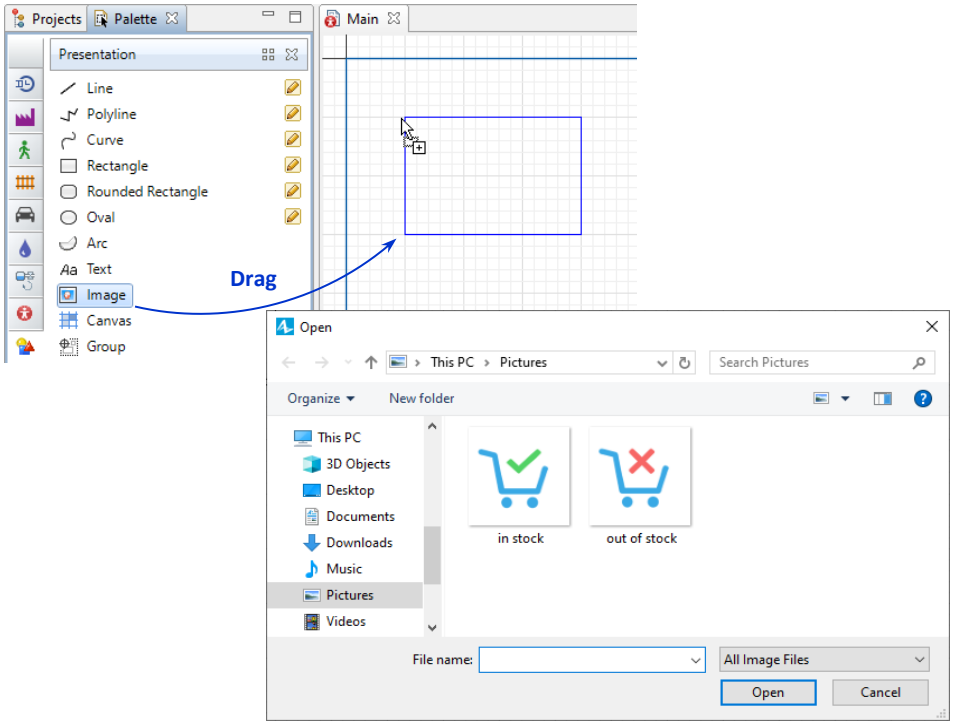

Figure 12.6 Adding an image

The original size will be set for the first image only. The other images with different sizes will be resized to fit that size.

When added, image files become model's *file resources*. Resource files are copied to the model folder, which simplifies model export and distribution. You monitor file resources in the **Resources** branch of the **Projects** tree.

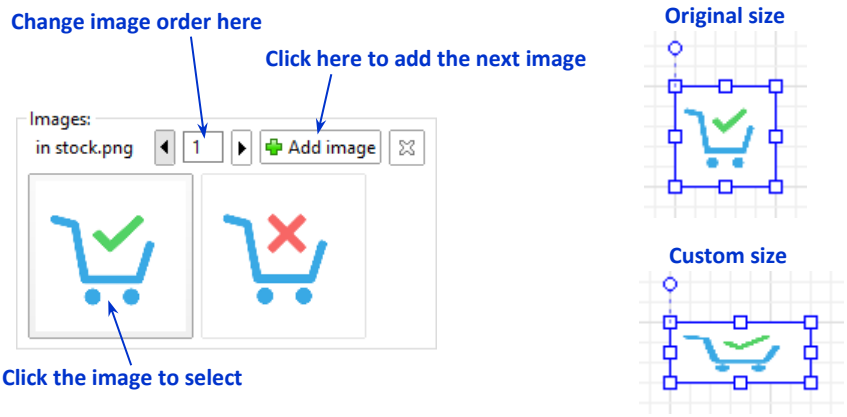

Figure 12.7 Image element with two images added

Canvas

Canvas represents a rectangular area, within which you can draw by means of programming. The shapes you draw on the canvas can form either static or dynamic picture (if drawn and modified at model runtime). This feature allows you to use the canvas element for creating heatmaps. Another typical use of AnyLogic canvas is animation of the spatial distribution (e.g., epidemy, pollution, housing).

To add a canvas:

1. Drag the **Canvas** element from the **Presentation** palette to the graphical editor.
2. You will see a rectangular shape filled with a grid. The just created canvas is transparent. Resize the canvas.

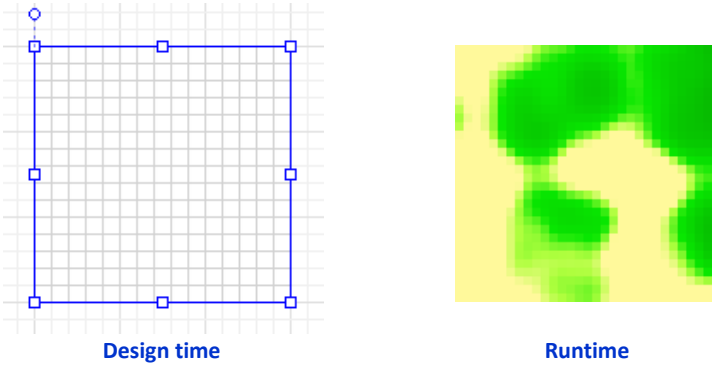

Figure 12.8 Canvas

Note that the grid is used only to depict the canvas area - it does not matter how many cells are drawn.

Now you can draw shapes on the canvas using its functions **fillRectangle()** and **fillCircle()**. You clear the canvas or its specific parts with the canvas functions **clear()** and **clearRectangle()**. You can find the **ShapeCanvas** functions description in *AnyLogic Help* (The AnyLogic Company, 2019).

Typically, you add canvas to a group (see Section 12.2, "Grouping shapes") and place the code updating the canvas in the advanced **On draw** property of this group. The following code dynamically updates **mapCanvas** to draw a vegetation map. The canvas dimensions are 500*500 pixels. We have segmented it into 10 000 cells, and for each animation frame we change the color of the cells where vegetation level has changed.

```
Color c;
for( int i = 0; i < 100; i++ ) {
    for( int j = 0; j < 100; j++ ) {
        c = vegetationToColor( vegetation[i][j] );
        if( !c.equals( vegcolors[i][j]) ) {
            mapCanvas.fillRectangle(i * 5, j * 5, 5, 5, c );
            vegcolors[i][j] = c;
        }
    }
}
```

In contrast to other presentation shapes, any shapes drawn on a canvas become a part of a raster image and cannot be accessed individually. Therefore, using canvas is only recommended if you need to optimize the animation performance.

Z-Order

Although the 2D picture is flat, shapes are drawn in a certain order and shown one above the other. This order is called the *Z-order* because there is a virtual Z-axis perpendicular to the surface of the picture and extending towards the viewer.

▶ To change the Z-order of a shape:

1. Select the shape(s).
2. Choose **Order** from the context menu.
3. Choose the appropriate action:
   Bring to Front places the shape on top of all other shapes.
   Bring Forward moves the shape one step up (swaps it with the shape directly in front).
   Send Backward moves the shape one step down.
   Send to Back places the shape below all other shapes.

Sometimes it is more convenient to use the toolbar buttons instead of the context menu.

If the shape is a member of a group, the ordering applies only within the group. The group appears as a single item in the global shape order.

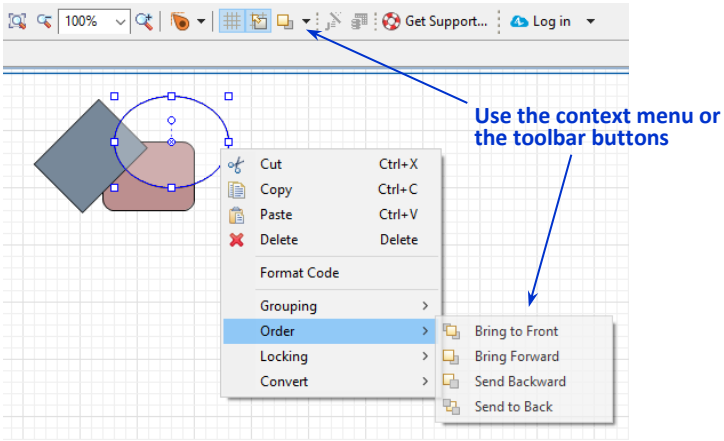

Figure 12.9 Changing the Z-order of a shape

The Z-order of flat shapes defined for 2D animation will not automatically be preserved in 3D animation. Flat shapes with the same Z-coordinate may appear interlaced. To force the Z-order for flat shapes in 3D animation you should assign slightly different Z-coordinates for them.

Selecting hidden shapes

Sometimes shapes are hidden below other shapes. You can still select those shapes by clicking several times on them.

To select a shape that is below other shapes:

1. Click at the position where the hidden shape should be. The top-level shape is selected.
2. Keep clicking at the same position until the shape you are looking for gets selected. To find out which shape is currently selected look at the **Properties** view.

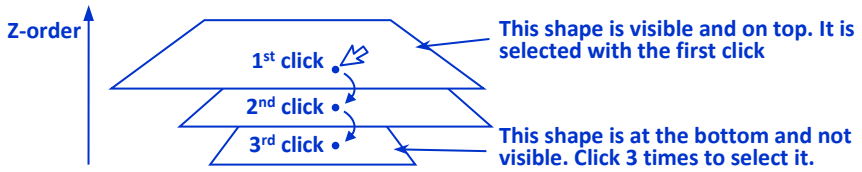

Figure 12.10 Selecting the hidden shapes by clicking

If you think you have completely lost a shape (or just any element), you can always find it in the **Projects** tree, and double-click it there to select the element in the graphical editor.

Coordinates and the grid

Graphics in AnyLogic are created in infinite space in the 2D editor. The X axis is horizontal directed to the right, and the Y axis is directed downwards. The Z axis is directed towards you, the viewer. The X and Y axes are shown as black lines, and when you open the graphical editor the coordinate origin is at the upper left corner of the window. Your graphics as well as the model elements can be located in all four quadrants of the space having positive or negative coordinates. When you run the model, however, only a particular section of the lower right quadrant is shown by default. This section is defined by the model *frame*. The frame begins at the coordinate origin and its default size is 1000 x 600 pixels.

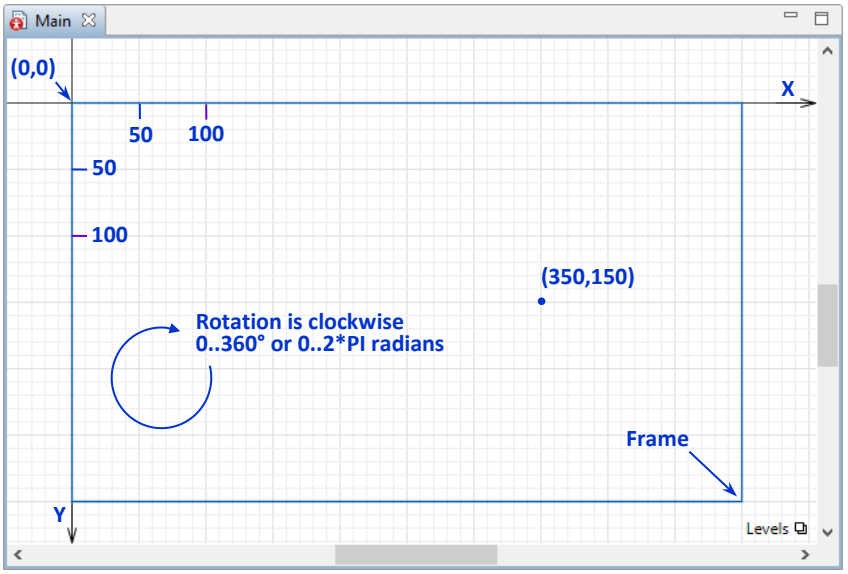

Figure 12.11 Coordinate origin, axes, frame and the grid

To resize the frame:

1. Click the frame to select it.
2. Drag the handle in the bottom right corner of the frame.

The coordinate units correspond to pixels at 100% zoom. To simplify arranging and aligning shapes the graphical editor shows the *grid* and supports *snapping* elements to the grid. At 100% zoom the grid step is 10 and guidelines are shown every 50 pixels. If snap to grid is on, the elements' coordinates and sizes will be equal to

N*10. As you change the zoom, the grid step will also change. For example, at 200% zoom the grid step is 5, and starting from 800% and up the grid step is 1.

To change the zoom:

1. Hold the Ctrl button and rotate the mouse wheel, or use the zoom control section of the toolbar.

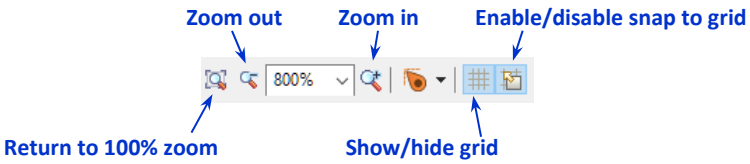

Figure 12.12 Zoom and grid control sections of the toolbar

You can finely adjust the position of your shape without going to large 800% zoom.

To adjust the position of a shape in between the grid lines:

1. Hold the Shift button and use the Arrow keys to move the shape. The shape coordinates will change by one step regardless of the zoom and grid settings.

The coordinate values in the graphical editor (the design-time coordinates) are integers and correspond to pixels at 100% zoom. The runtime coordinates are real numbers (Java type **double**). Therefore, you can position your graphics to within any degree of accuracy by using dynamic properties or the API.

Copying shapes

AnyLogic graphical editor supports cut, copy, and paste operations in the same way any other editor does. You should select the shape(s) first and then use the context menu, toolbar buttons, or main menu commands. You can copy graphics between different agent types or between an experiment and an agent type.

There is another way to create a copy bypassing the clipboard: using Ctrl+drag.

To create a copy of a shape:

1. Select the shape(s).
2. Hold Ctrl button and drag the shape(s) to the new position. The copy is created.

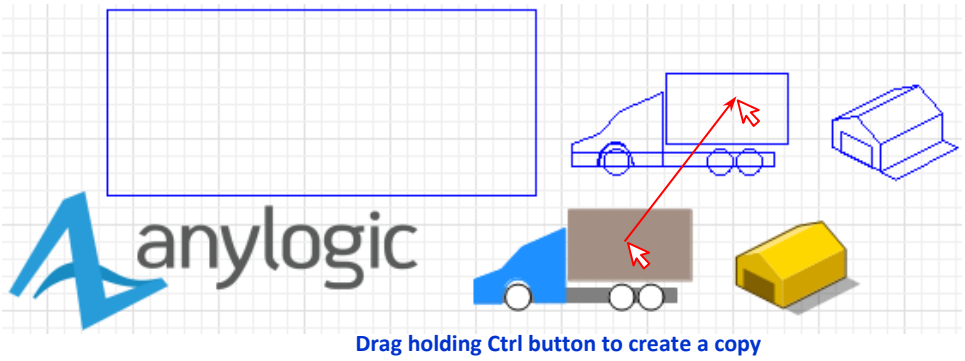

Figure 12.13 Using Ctrl+drag to create a copy of a shape

All these operations (cut, copy, paste via clipboard and copy by Ctrl+dragging) are supported in the **Projects** tree as well.

Locking shapes – preventing selection by mouse

Graphical editing can sometimes be made easier by desensitizing shapes to mouse clicks to avoid selecting the wrong shape. A simple example is when you have a large background image and wish to draw something on top of it. You can temporarily *lock* shapes for that purpose.

To lock a shape:

1. Select the shape.
2. Choose **Locking | Lock Shape** from the context menu.

Once the shape has been locked, it will not react to any mouse actions. However, it still can be selected by clicking on its icon in the **Projects** tree.

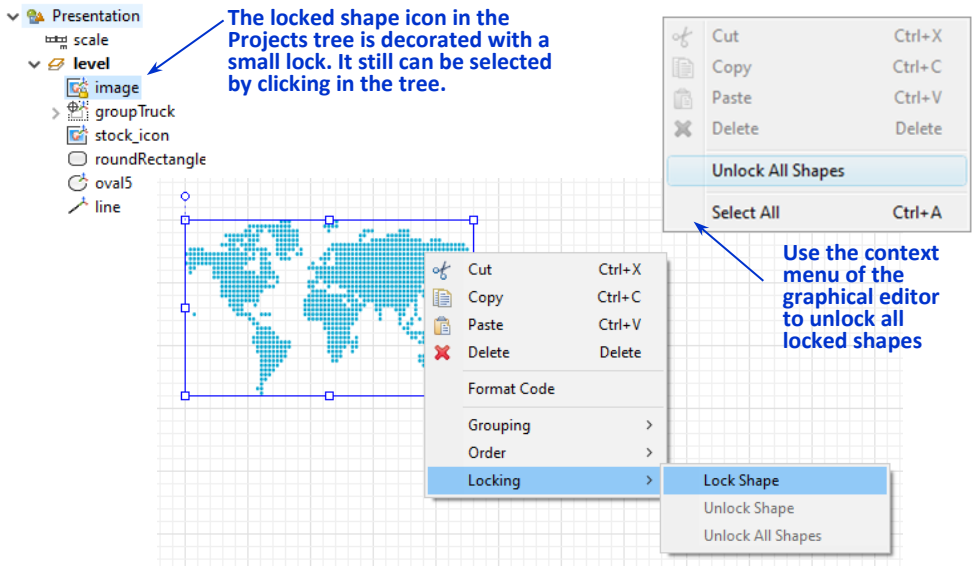

Figure 12.14 Locking and unlocking shapes

To unlock all locked shapes

1. Choose **Unlock All Shapes** from the context menu of the graphical editor (pops up when you right-click in the empty space of the graphical editor).

To unlock a shape

1. Find the shape icon in the **Projects** tree and select it. The locked shape gets selected in the graphical editor as well.
2. Right-click the shape in the graphical editor and choose **Locking | Unlock Shape** from the context menu.

Properties of graphical shapes

Any property of any shape can be edited in the **Properties** view. A field for the shape name, which actually is the name of the corresponding Java object, and several checkboxes explained in Figure 12.15 below, are located at the top of the **Properties** view.

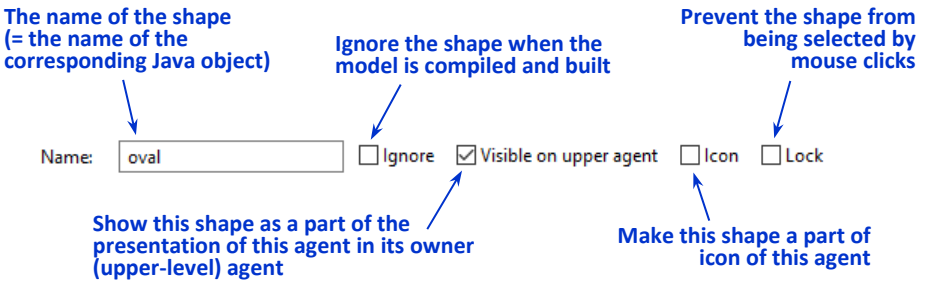

Figure 12.15 General properties common to all shapes

The **Appearance** section contains the most frequently used properties: line and fill color, line style and width, etc.

Most of the shape's properties can be overridden by the dynamic values (see Section 12.3, "Animation principles. Dynamic properties of shapes") or changed using the shape API.

The name of a shape is not normally needed in the graphical editor. You may only wish to display it when it clarifies your model design. Names of shapes are never displayed during runtime.

The **Position and size** section of the properties contains properties that are used to specify the dimensions of the shape, its position and the scale factors.

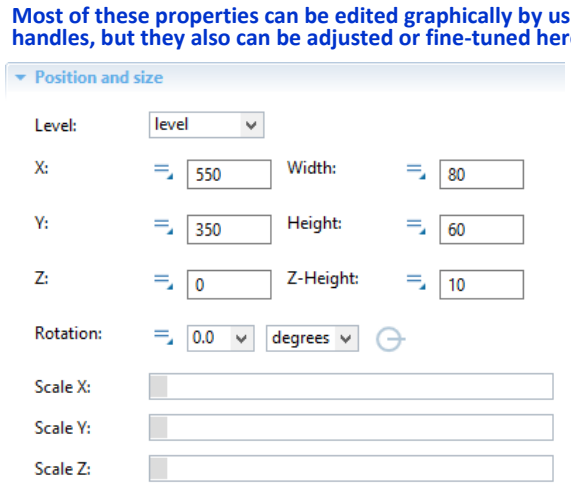

Figure 12.16 Position and size properties of a shape

You can use the **Level** drop-down list to relocate the shape to a different level, while retaining the shape's coordinates.

The position, size, and rotation fields just give you an alternative way of moving and resizing the shape if you do not want to use the graphical editor.

The scale fields provide a place to define the scale factor of the shape in the directions of X, Y, Z axes.

By default, the shapes are displayed both in 2D and 3D. The specifics of the corresponding **Show in** property in the **Advanced** section are explained in Chapetr 14, “3D animation”.

12.2. Grouping shapes

Presentation shapes in AnyLogic can be grouped so that you can work with them as if they were a single shape: select, move, resize, rotate, copy, etc. While a shape is a member of a group, it can still be selected individually and edited. At runtime you can dynamically show, hide, rotate, and move grouped shapes by using the group dynamic properties. Groups are also used to provide a new rotation center and coordinate origin for its members.

You can group:

- Simple shapes (rectangles, ovals, polylines, etc.), text, and images,
- Controls (buttons, sliders, edit boxes, etc.),
- Charts (bar charts, plots, histograms, etc.),
- 3D shapes and figures, 3D views, cameras and lights,
- Other groups,
- Embedded presentations (presentations of embedded agents).

You cannot group non-presentation elements like events, statecharts, variable and function icons, agent populations, flowchart blocks, etc.

To group one or several shapes:

1. Select the shapes (by dragging the selection rectangle around or by Ctrl+clicking each shape).
2. Choose **Grouping | Create a Group** from the context menu.

A group of shapes in AnyLogic has its own “origin”, which in general is not the same as the geometrical center of the grouped shapes – it may even be outside the shapes’ boundary. The location of the group origin is shown as a small circle with a handle when the group is selected. The group origin serves as rotation center for the group and as coordinate origin for the shapes – members of the group.

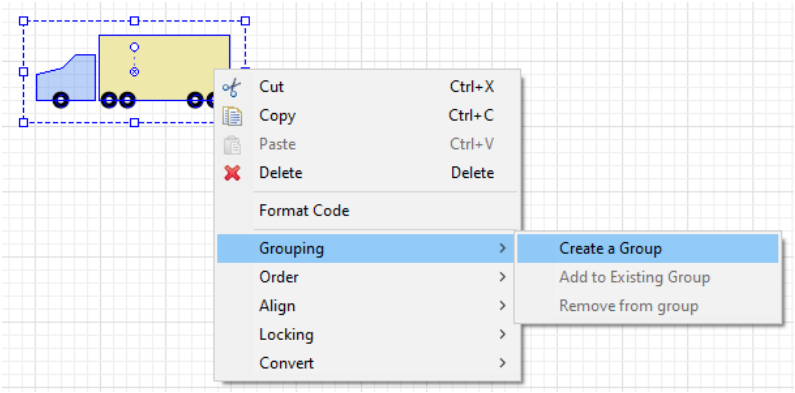

Figure 12.17 Grouping shapes

You may need to adjust the Z-order of shapes after grouping. This is done by selecting individual shapes and bringing them to front or sending them back.

To add a shape to the existing group:

1. Right-click the shape and choose **Grouping | Add to Existing Group**. The available groups will be shown as circles with crosses at their origin points.
2. Click the group where you wish to add the shape.

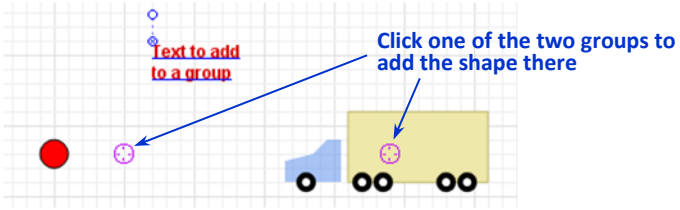

Figure 12.18 Selecting a group to add the shape to

Shapes that reside in different levels cannot be grouped together. If you add a shape from, say, Level 1 to a group that lives on Level 2, the shape will automatically move to Level 2.

To select an individual shape that is a member of a group:

1. Select the group.
2. Click the shape while the group is selected.

Groups in AnyLogic can be nested. If you wish to select the shape that is a member of the inner group, keep clicking the shape until it is selected.

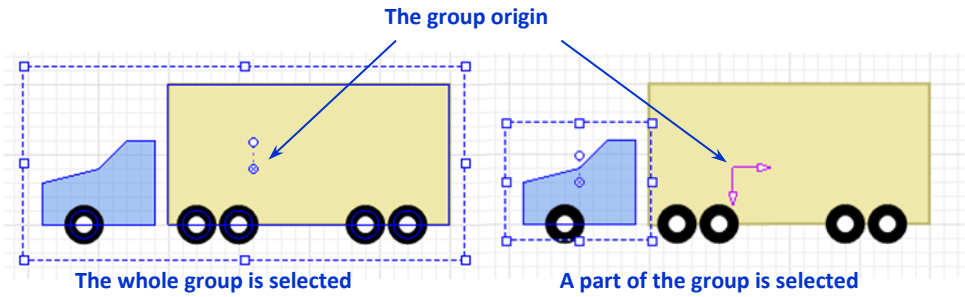

Figure 12.19 A group of shapes in the graphical editor

To rotate a group:

1. Pull the handle at the group "origin".

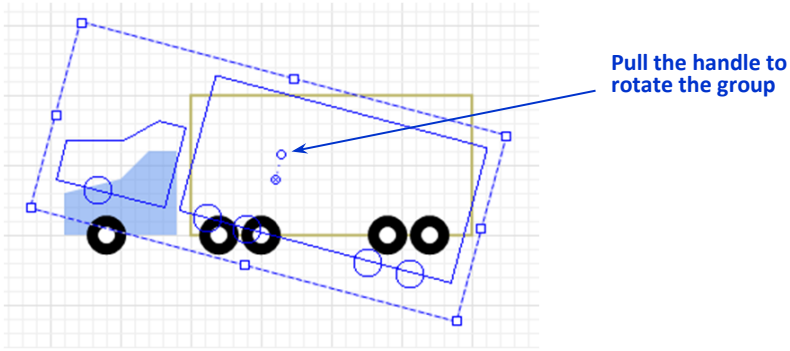

Figure 12.20 Group rotation in the graphical editor

Please note that controls, charts, and 3D views cannot be rotated. If you rotate a group containing such elements, they will only change their location, but orientation will remain the same.

Sometimes the position of the group origin needs to be moved relative to the grouped shapes. This may be needed e.g., to change the rotation and scaling center or coordinate origin of the group.

To move the group origin relative to the grouped shapes:

1. Select the group.
2. Choose **Select Group Contents** from the group context menu.
3. Move the group contents so that the group origin appears in the right position relative to the shapes.
4. Deselect the group contents by clicking anywhere in the graphical editor.
5. If needed, select the group again and move it.

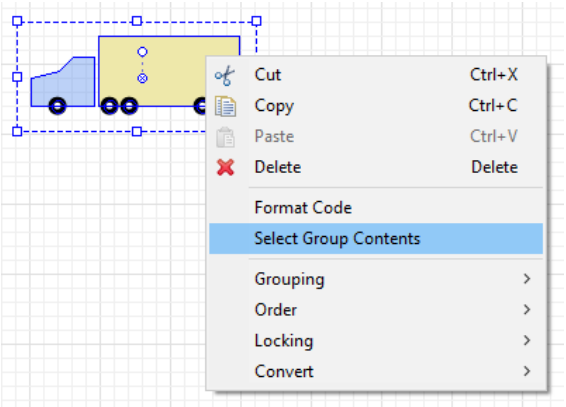

Figure 12.21 Moving the group origin relative to the group contents

A group may be used to set a new rotation point, new scaling center, and new coordinate origin for an individual shape different to the default ones. For example, you may wish to rotate a rectangle around its center, or have its center as its coordinate origin, while the default for both is the upper left corner. You can use the following technique:

▶ **To change the rotation point, the scaling center and coordinate origin of a shape:**

1. Select the shape.
2. Choose **Grouping | Create a Group** from its context menu (a new group containing just that shape is created and selected).
3. Choose **Select Group Contents** from the group context menu.
4. Move the group contents (i.e., the shape) so that the group origin appears in the right position relative to the shape.
5. Deselect the shape, reposition and rotate the whole group if needed.

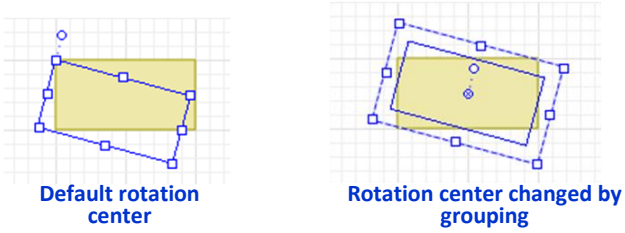

Figure 12.22 Changing the rotation point, scaling center and coordinate origin of a rectangle

Unlike in some other graphical editors such as MS PowerPoint, groups in AnyLogic may be empty containing no shapes or other groups. Such empty groups may be used during runtime to add or remove shapes dynamically. Or they may simply serve as references to specific locations.

To create a new empty group

1. Drag the **Group** element from the **Presentation** palette to the graphical editor.

While the group is empty, it is shown in the editor as a circle with a cross inside. Once you add the first shape to the group, its icon becomes invisible.

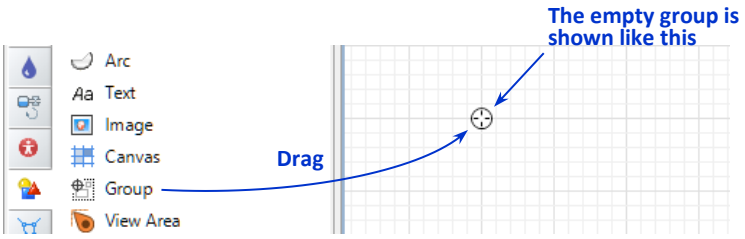

Figure 12.23 Creating an empty group

Pictures that you drag from the **Pictures** palette and drop to the graphical editor are also regular groups of regular AnyLogic shapes. You can edit them as you like, change their internals, or even ungroup them.

Groups are extensively used at runtime to show, hide, rotate or move parts of the presentation. The group origin serves as the new coordinate origin and rotation center for the group members. For example, if the space where a certain object is going to move has its upper left corner at (150, 200) it makes sense to place the group there and to add the object to the group.

To define circular movement around the group origin

1. Drag **Oval** from the **Presentation** palette to anywhere in the graphical editor, make it a circle with radius 10, set **Fill color** to **red**.
2. In the **Position and size** properties section of the circle set **X**: **50 * sin(time())** and **Y**: **50 * cos(time())**.
3. Run the model. The oval is moving on the orbit around the coordinate origin (the upper left corner of the window).
4. Drag a **Group** from the same palette to the point e.g. (200,150).
5. Click the oval and select **Grouping | Add to Existing Group**, then click the group. When the oval is added to the group its icon is shown as a small circle with a handle.
6. Run the model again. The oval is now moving around the group origin.

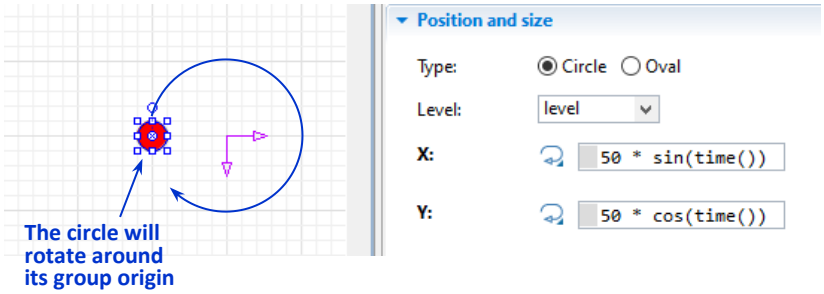

Figure 12.24 Using group as coordinate origin for its member shapes movement

Working with the group contents dynamically using API

AnyLogic Java class for group is **ShapeGroup** (a subclass of **Shape**). Its API allows you to iterate through the group members and add or remove shapes dynamically. Suppose you have a group with name **group21**. The following code iterates through the contents of the group, finds all ovals and outputs their sizes to the model log:

```
for( int i=0; i<group21.size(); i++ ) {
  Object member = group21.get(i);
  if( member instanceof ShapeOval ) {
    ShapeOval oval = (ShapeOval)member;
    traceln( "Oval Rx: " + oval.getRadiusX() + " Ry: " + oval.getRadiusY() );
  }
}
```

Single shapes as well as replicated shapes (see Section 12.4) can be mixed in a group. The function **get()** returns the type **Object** - an object of a subclass of **Shape**, or an object of class **ReplicatedShape**. You need to check (or just cast) the type before doing further work with the group members.

To dynamically add a shape to the group:

1. Call **group.add(shape);**

To dynamically remove a shape from the group:

1. If you know both the shape and the group, call
   group.remove(shape);
   If you know the shape but do not know the group, call
   shape.getGroup().remove(shape);

As you can see, a shape knows its group and returns it when you call **getGroup()** function. The top-level shapes (the ones that do not belong to any user-created group) belong to the ever-present default group **presentation** (or **icon**).

On draw extension point – execute custom code on each frame

Groups have an extension point that may be used to do something on each animation frame during the model runtime. The extension point is called **On draw**. It is located in the **Advanced** section of the group properties.

It may be used for example to dynamically draw shapes on a canvas.

Groups in the project tree

As with any element in AnyLogic, groups and their contents are shown and can be accessed from the **Projects** tree. The tree shows the hierarchy of the grouped shapes and their Z-order. Suppose you have drawn the truck as two groups **groupTrailer** and **groupTractor** grouped in the third group **groupTruck**. Then the tree will show the following hierarchy:

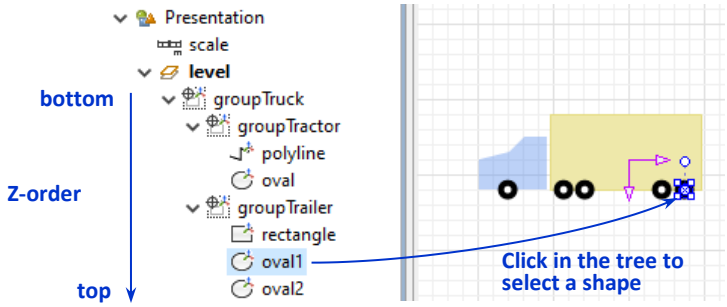

Figure 12.25 Grouped shapes in the Projects tree

Top-level groups for agent presentation and icon

Each agent by default has two top-level groups:

- Top-level **presentation** group contains all presentation shapes.
- Top-level **icon** group contains all shapes marked as **Icon**.

Both **presentation** and **icon** are instances of classes inherited from **ShapeGroup** and support the corresponding API.

12.3. Animation principles. Dynamic properties of shapes

At the elementary level AnyLogic animation is based on the following principles. The graphical shapes have their properties: size, position, rotation, color, image index, text, visibility, etc. These properties can be linked to variables and expressions in the model: values of stocks and flows, states of statecharts, remaining time of events, and so on. When those variables and expressions change their values during the model runtime the shapes change their graphical properties.

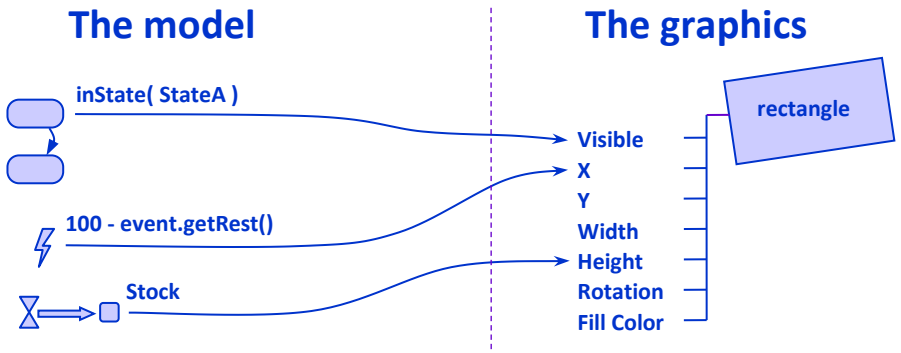

Figure 12.26 The model can control the graphics via dynamic properties of shapes

The agent-based modeling framework, the Process Modeling Library, and other libraries and components of AnyLogic offer higher level tools for model animation. For example, an agent can move along a road on GIS map, pedestrians can ride an escalator, etc. However, the low-level direct access to the shape properties allows you to achieve very sophisticated custom animation effects.

Dynamic properties of shapes

Linking of the model and the graphical properties is done via *dynamic properties* of shapes.

When you enter an expression in the dynamic property field, it will be constantly reevaluated for each animation frame during the model runtime and the value of the corresponding property will change accordingly.

Some properties of the shapes accept only a static value, others can accept either a static value or a dynamic value, and some only accept a dynamic value.

Properties that accept either a static value or a dynamic value are identified by an icon to the left of the property field. If the icon has a small triangle in the lower right corner, it means you can click this icon to switch the property field between the **Value editor** mode (for static values) and the **Dynamic value** mode (for dynamic values).

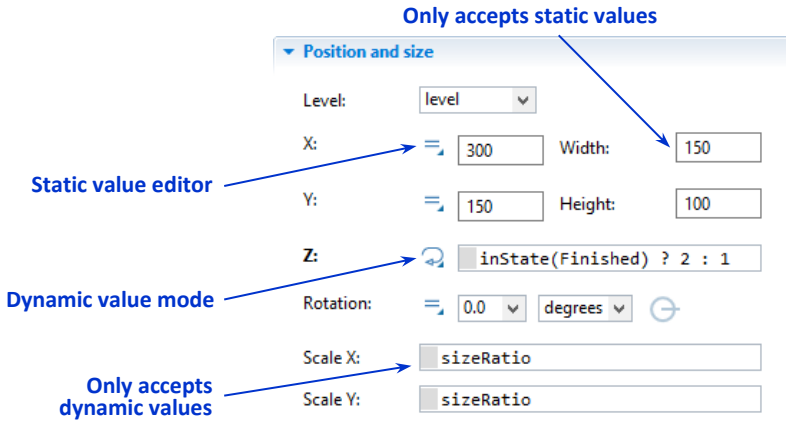

Figure 12.27 Different types of the shape properties

In Figure 12.28 you can see the dynamic properties of a rectangle shape. Most of the fields are quite straightforward and do need comments. In some cases, you need to know special constants to control the property (e.g., the expression for line style should evaluate to one of those: **LINE_STYLE_SOLID**, **LINE_STYLE_DASHED**, **LINE_STYLE_DOTTED**). You can find these values in *AnyLogic Help* (The AnyLogic Company, 2019). When the expression for line (or fill color) evaluates to **null**, the line (or fill) of the shape is not drawn. The expressions for rotation are interpreted as radians, and you can use the constant **PI**, which equals 180°.

The first dynamic property that is evaluated during animation is **Visible**. If the expression for **Visible** evaluates to **false**, the shape is not drawn at all and all other expressions are not evaluated. Therefore, you can specify the expressions in other fields that do not make sense (can raise error) when the shape is not visible and be sure they will not be evaluated.

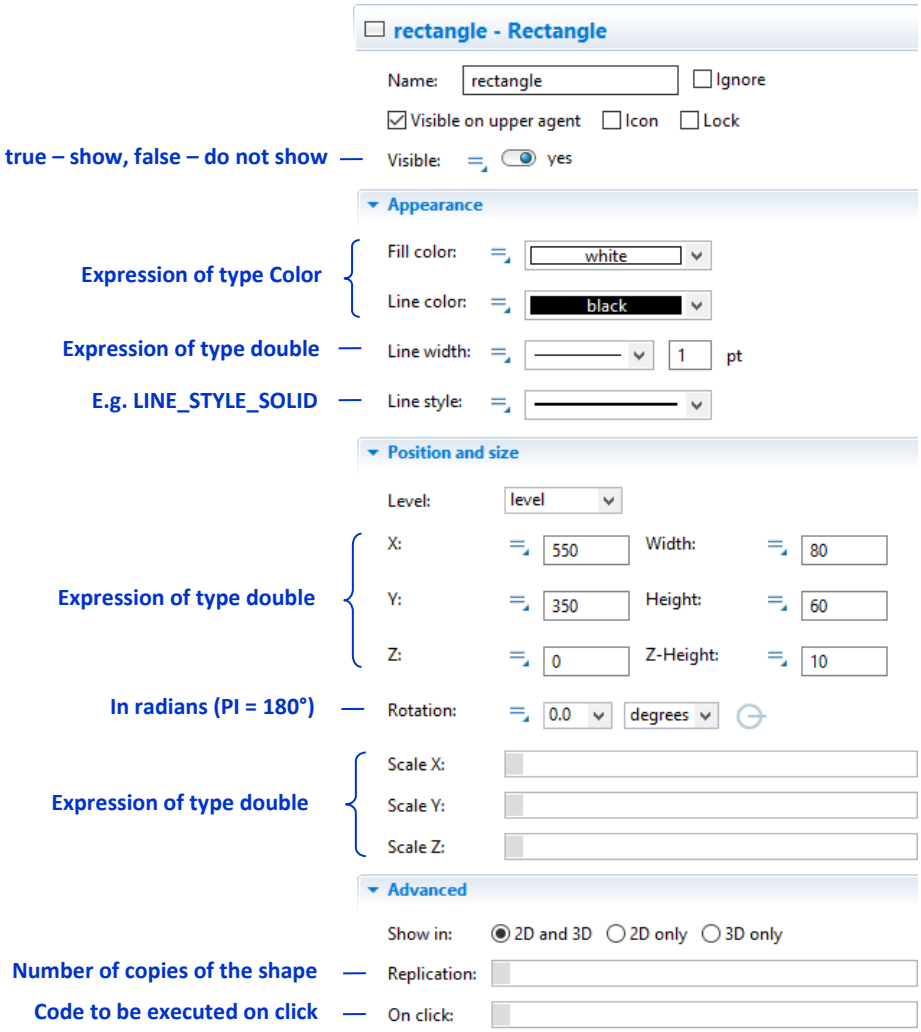

Figure 12.28 Dynamic properties of a rectangle

Example 12.1: Commodity price change animation

We will use a text shape and a little bit of graphics to create a commodity price change indicator. The price itself will be modeled by a variable **price** randomly changing every day. To display the change, we need to remember the last day price – another variable **lastDayPrice** will be used for that. The dynamics of the price will be modeled by a cyclic event **dailyChange** (see Section 8.2).

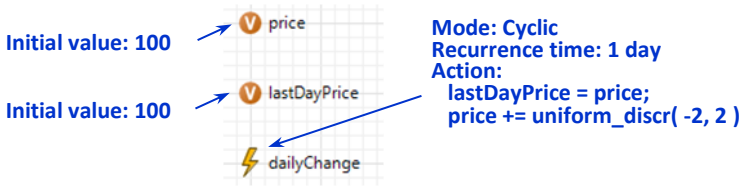

Figure 12.29 A dynamic model of a changing price

Create the dynamic model of the changing price:

1. Create a new model. Set model time units to days.
2. Create a variable **price** by dragging the **Variable** element from the **Agent** palette and setting its name to **price**.
3. In the properties of the variable set its initial value to **100**.
4. Ctrl+drag the **price** variable to create its copy below it. Change the name of the copy to **lastDayPrice** and leave the initial value unchanged.
5. Create an event **dailyChange** by dragging the **Event** element from the **Agent** palette and changing its name.
6. Change the **Mode** of the event to **Cyclic** and leave the default **Recurrence time** (1 day).
7. In the **Action** of the event write the following:

```
lastDayPrice = price;
price += uniform_discr( -2, 2 );
```

8. Right-click the **price** variable and select the **Create Chart > Create Time Plot** option from the context menu.
9. Run the model. Observe the time plot to see how the commodity price changes.

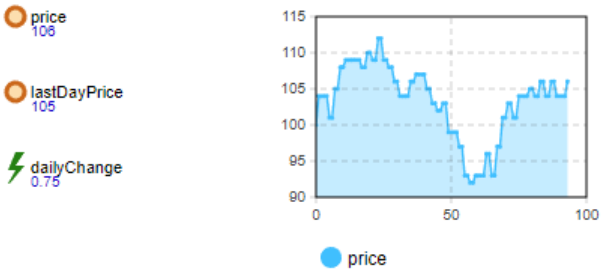

Figure 12.30 The commodity price fluctuations

You should be able to see the price fluctuations. The event **dailyChange** occurs every day and does the following: stores the current value of the **price** to **lastDayPrice** and adds to the **price** a change value randomly chosen from the set {-2, -1, 0, 1, 2} – this what is returned by the function **uniform_discr(-2, 2)**.

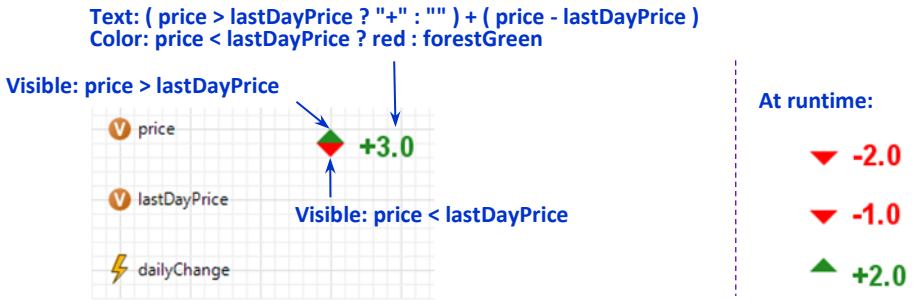

Figure 12.31 The price change animation

Create the price change animation:

10. Use polylines to draw two small triangles as shown in Figure 12.31. A triangle is a 3-point polyline with option **Closed** checked in its properties.
11. Choose the **red** color for the lower triangle and **forestGreen** color for the upper one.
12. For the red triangle, click the **Visible** property's icon to switch to the dynamic value editor. Type **price < lastDayPrice** in the **Visible** field.
13. For the green triangle, switch the **Visible** field to the dynamic value editor and type **price > lastDayPrice** there.
14. Add a text shape on the right of the triangles. In the **Appearance** property section, set the text font size to 20 and the font style to **Bold**.
15. You may write some sample text in the field inside the **Text** property section, e.g. "+2.0" – just to see how it will look like, the actual text will be set in the dynamic value editor.
16. Switch the **Text** field to the dynamic value editor and type there
    (price > lastDayPrice ? "+" : "") + (price - lastDayPrice)
17. In the same way, set the dynamic value for the **Color**:
    price < lastDayPrice ? red : forestGreen
18. Run the model.

Depending on the current and the last day prices, either the green triangle will show, or the red one, or, if the price does not change, none. The text color is red if the price goes down, and green otherwise (i.e., if the price goes up or stays the same). The expression for the dynamic value of the text means the following. The part **(price > lastDayPrice ? "+" : "")** evaluates to either string **"+"** or **""** (empty string). The part **(price - lastDayPrice)** obviously evaluates to the numeric value of the price change. If you add a string and a number, the result is a string containing the text representation of the number.

Example 12.2: Elevator doors animation

Let us create an animation of elevator doors. Suppose the dynamics of the doors is modeled by a statechart with four states: **Open**, **Closed**, **Opening**, **Closing**. The doors are animated as two rectangles. We assume you already know how to draw a statechart with transitions triggered by timeout (see Section 7.3).

Create the statechart:

1. Create a new model. Set the model time units to minutes.
2. Draw a statechart with four states and four transitions as shown in Figure 12.32 below. The name of the statechart should be **doors**.
3. Name the transitions exactly as shown. To let the transition names appear on the screen, select the **Show name** checkbox for all transitions.
4. All four transitions are triggered by timeouts. Specify the following timeouts:

   StartOpening: **2 minutes**

   OpeningCompleted: **30 seconds**

   StartClosing: **2 minutes**

   ClosingCompleted: **30 seconds**

This statechart defines the following behavior of the doors: the doors stay closed for 2 minutes, then open (which takes 30 seconds), then stay open for another 2 minutes, then close again within 30 seconds.

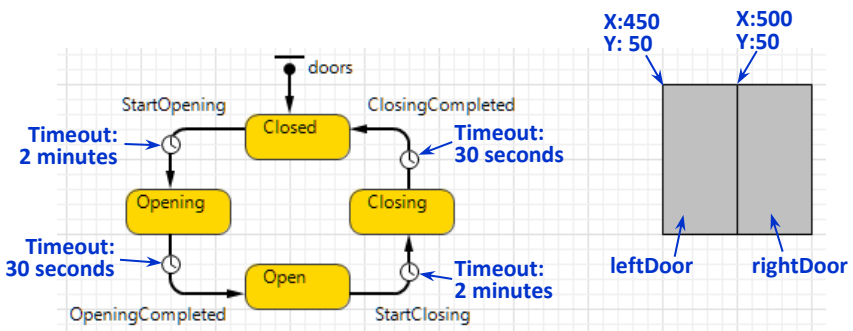

Figure 12.32 The statechart and animation of the elevator doors

Draw the doors:

5. Draw the two rectangles as shown in Figure 12.32. The location and sizes are important and should be exactly as shown.
6. Give them the names: **leftDoor** and **rightDoor**.
7. Expand the **Position and size** section of **leftDoor** properties. Switch the **Width** property to the dynamic value editor and enter the following expression in the **Width** field:

```
inState ( Closed ) ? 50 :
  ( inState ( Open ) ? 0 :
    ( inState ( Opening ) ? 50 * OpeningCompleted.getRest()/0.5 :
      50 * ( 1 - ClosingCompleted.getRest()/0.5 )
    )
  )
```

8. In the **rightDoor** properties, type the following dynamic expressions:
   X: **500 + (50 - leftDoor.getWidth())**
   Width: **leftDoor.getWidth()**
9. Run the model.

The long expression for the width of the left door has the following meaning. If the doors are closed (we test this by checking whether the current active state of the statechart is **Closed**), the width of the **leftDoor** is **50**. Otherwise, if the doors are completely open (state **Open**), the width is **0**. Otherwise, if the doors are opening, we need to set the door width proportional to the *uncompleted fraction* of opening operation. The fraction is calculated as the time to opening completion (**OpeningCompleted.getRest()** returns the remaining time before the corresponding transition is taken) divided by the total opening time (in model time units, **0.5** minutes). The last possible case is when the doors are being closed. Then the door width is set proportional to the *completed fraction* of operation, which is 1 – uncompleted fraction.

The expression is quite long, and we do not want to replicate it for the **rightDoor**. Moreover, something may change later, and we do not want to modify code in two different places. Instead, we will use the fact that the two doors always have the same width, so the width of the **rightDoor** is set equal to the width of the **leftDoor**. Note that the right door not only changes its width, but also its X coordinate, hence the expression for **X**.

If you move the door rectangles you will need to correct the coordinate **500** in the properties, which is undesirable. To create position-independent doors you can group them and use group-relative coordinates. See Section 12.2, "Grouping shapes" for details.

Example 12.3: Stock of money animation

In this example we will visualize a system dynamics stock with a custom graphics. The stock and flow diagram will be very simple: there will be one stock **Money**, one inflow **Income** and one outflow **Expenses**. The graphics will be a bag with a "$" sign on it. The bag will change its size to reflect the value of the stock.

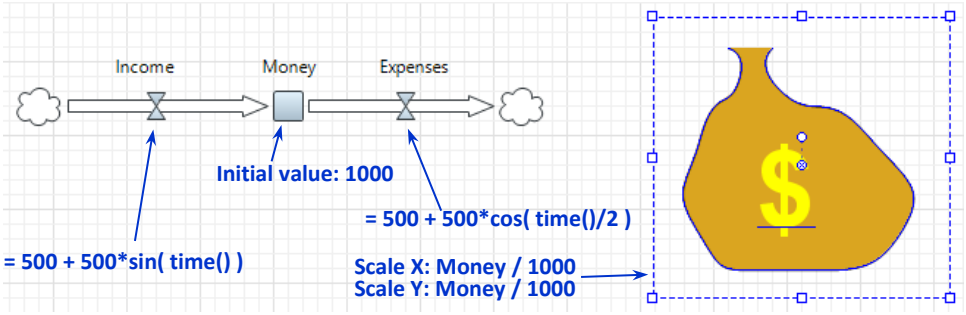

Figure 12.33 Money stock and its custom animation

Follow these steps:

1. Draw a system dynamics diagram as shown in Figure 12.33.
2. Set the initial value of the **Money** stock to **1000**.
3. Set the formula for
   Income: **500 + 500 * sin(time())**
   Expenses: **500 + 500 * cos(time()/2)**
4. Draw a curve in the form of a bag. Set the fill color of the curve to e.g. **goldenRod**.
5. Add a text shape on top of the bag. Set its properties:
   Text: **$**
   Font size: **72 pt**
   Font style: **Bold**
   Color: **yellow**
6. Select both the bag and the text and group them (see Section 12.2, "Grouping shapes" for details).
7. In the **Position and size** section of the group properties set:
   Scale X: **Money / 1000**
   Scale Y: **Money / 1000**
8. Run the model.

You should be able to see how the bag changes its size with oscillations of the stock value. Use the **Money** stock's inspect window, as shown in Figure 12.34, to view its time chart.

? The bag expands and contracts to/from its center. How could we change the bag's animation to appear to lay flat on a table and only grow up and sidewise as it filled?

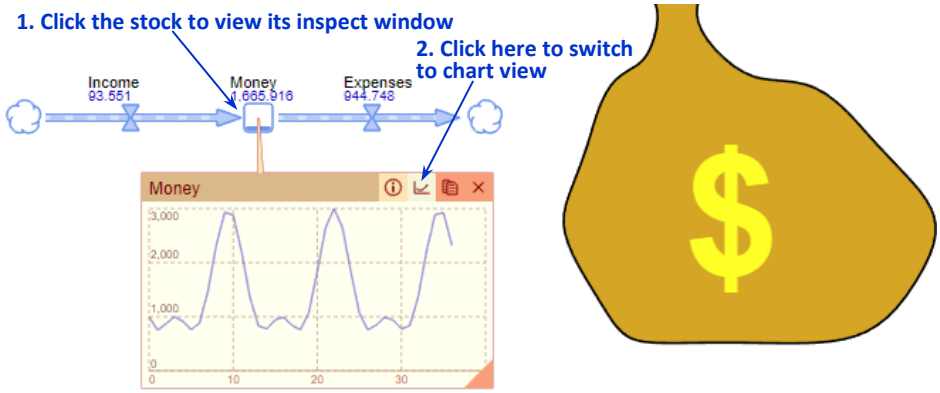

Figure 12.34 The money stock animation

Example 12.4: Missile attack animation

In this example we will use two different images to animate the missile during its flight and after explosion at the target. We will also add a visual effect of explosion purely on animation level. Please prepare two images: one of a missile, another of explosion. It is better if they have the same size or at least the same proportions.

▶ **Follow these steps:**

1. Create an event **hitTarget**. It should be a timeout event that occurs once at time **50**.
2. Add an image element at **(100,150)**
3. Add two images to the image element – the first one with a missile, the second one with explosion. You may or may not use the **Original size** option. The image resulting size should be approximately 100 x 100 pixels.
4. In the **Advanced** section of the image properties set
   Image index: **hitTarget.isActive() ? 0 : 1.**
5. Right-click the image and choose **Grouping | Create a Group**.
6. In the **Position and size** section of the **group** properties switch the **X** field to dynamic value editor and type there:
   500 * (hitTarget.isActive() ? (1 - hitTarget.getRest() / 50) : 1)
7. In the same section of the **group** properties set:
   Scale X: **hitTarget.isActive() ? 1 : 1 + (time() - 50) / 10**
   Scale Y: **group.getScaleX()**
8. Draw the "target" – a thick vertical line from **(550,50)** to **(550, 250)**
9. Run the model.

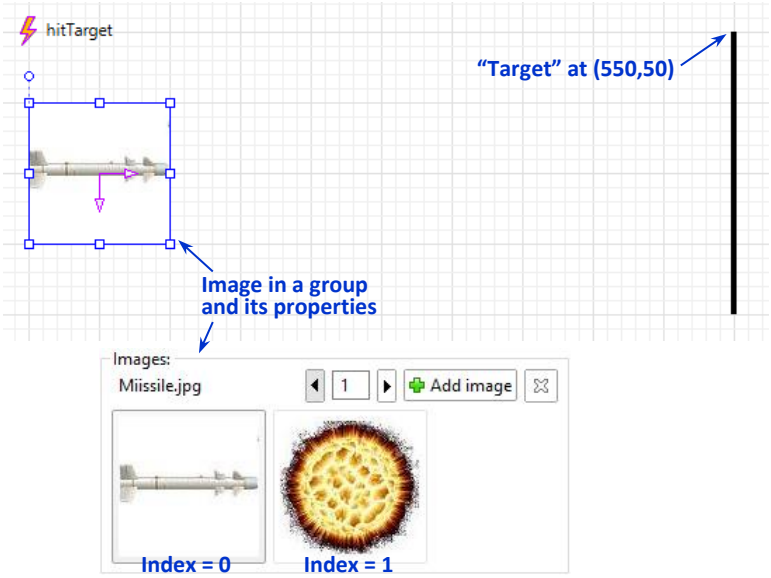

Figure 12.35 Animating missile and explosion with images

The **Image index** property is set to change the image of the missile to the image of the explosion once the missile hits the target – and the **hitTarget** event becomes inactive. Grouping was used to shift the scaling center of the image from its upper left corner to its geometrical center. The flight of the missile is animated by the dynamic **X** property of the group: while **hitTarget** event is active the missile X coordinate is proportional to the fraction of the distance to target (at 500) covered by the missile. After the hit, the group stays at X = 500. Finally, the explosion effect is modeled by enlarging the scale of the group after the hit from 1 to infinity. X and Y scales are the same, so scale Y just refers to scale X.

As you may have noticed, the visual effects in this example are achieved solely by setting the shapes' dynamic properties. At the model level we have just one timeout event that occurs once at time 50. Models designed this way are very efficient because when the animation is off in fast execution modes such as during optimization or parameter variation experiments, no CPU time is spent on animation-related tasks.

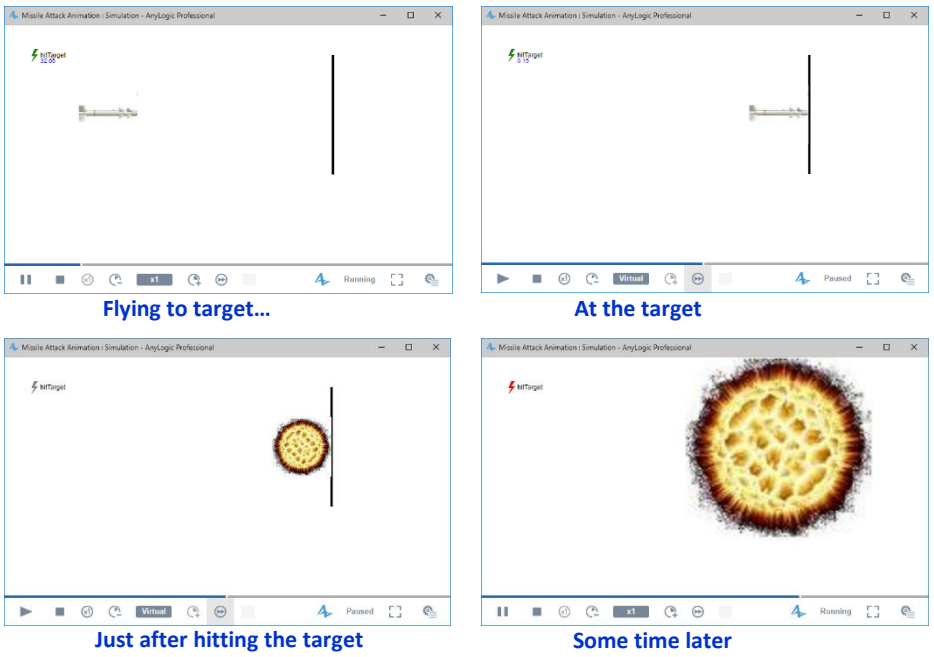

Figure 12.36 Animating missile and explosion with images

Animation frames

2D animation is drawn in *frames*. On the first frame AnyLogic draws shapes one by one according to their Z-order (see Section 12.1) starting from the bottom. When a shape is drawn, its dynamic properties override its static properties. On subsequent animation frames, AnyLogic updates only the shapes whose dynamic properties have been changed. The property **Visible** is evaluated first, and if it evaluates to **false**, the shape is not drawn, and its other properties are not evaluated.

After the frame is drawn, AnyLogic lets the CPU to continue the simulation in between this frame and the time when the next frame is drawn. As a result, there is a fragment of the simulation executed between frames. Some values in the model may change and be reflected in the next frame (through the dynamic properties of shapes).

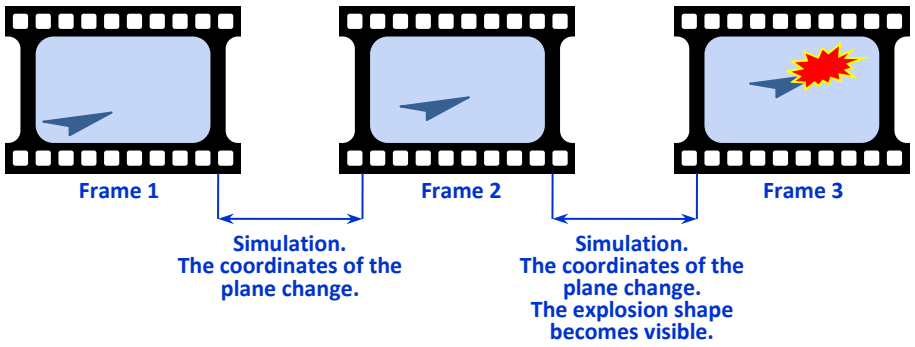

Figure 12.37 Animation frames interlaced with the model simulation

12.4. Replicated shapes

AnyLogic gives you the ability to display an arbitrary (and dynamically changing) number of similar shapes. You need to draw a shape in the graphical editor once and declare it as *replicated shape*. Then you can specify individual property values for every copy of the shape. This is useful for both drawing static regular structures and for animating dynamic collections of similar objects.

Agents in populations and in all AnyLogic libraries (agents and resource units in Process Modeling Library, pedestrians in Pedestrian Library, material items in Material Handling Library, etc.) have their own higher-level animation capabilities and typically you would not use replicated shapes to animate them.

Do not confuse *replicated shapes* with replicated flowchart blocks or simulation replications – these are completely unrelated items.

To declare a shape as replicated:

1. In the **Advanced** section of the shape properties type an expression in the **Replication** field.

The expression should return an integer number. It can be constant or variable. At runtime AnyLogic will evaluate the expression and draw the corresponding number of copies of the shape. Of course, replication only makes sense if you can specify different properties for different copies (otherwise all copies will be identical and drawn one above the other). Once you enter something in the **Replication** field the variable **index** becomes available in all other dynamic properties of this shape. You can check this by placing the cursor into any other field and then pointing your mouse to the little light bulb on the left.

To specify different property values for different copies of a replicated shape:

1. Enter the expression that depends on the **index** variable in the desired dynamic property field.

When drawing the shape copies, AnyLogic will be evaluating the dynamic properties for each copy substituting **index** with 0, 1, 2, … [Replication-1]. If a property depends on **index**, it will have different value for different copies.

The property **Visible** is evaluated *before* the property **Replication**. If **Visible** evaluates to **false**, the shape is not drawn at all regardless the value of **Replication**. Therefore, it is not possible to use **Visible** to display some copies of the shape while hiding the others. To achieve this effect, you can set the line and fill colors to **null** for the shape copies that need to be hidden.

Example 12.5: Drawing seats in a movie theater

We will use replication to draw seats arranged in a rectangular movie theater. We will draw the seat once and then we will replicate it. Different copies of the seat will get different coordinates. In this example the replication will result in a static picture. Later on, we will extend this example to include animation of dynamic seat sales.

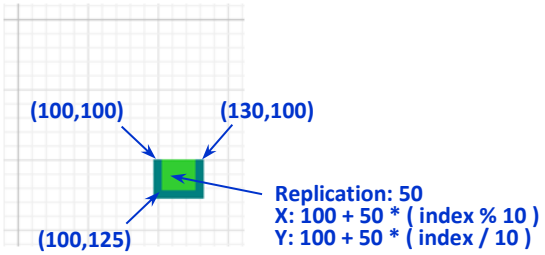

Figure 12.38 A movie theater seat animation – a replicated polyline

Draw a seat animation:

1. Draw an open three-point polyline as shown in Figure 12.38.
2. Expand the **Appearance** section of the polyline properties and set the polyline line width to 6 pt by entering **6** in the **Line width** text box.
3. Set the **Line color** of the polyline to **teal** and the **Fill color** to **limeGreen**.
4. In the **Advanced** section of the properties set **Replication**: **50**.
5. In the **Position and size** section switch both **X** and **Y** properties to dynamic value editing mode by clicking the icon and set:
   X: **100 + 50 * (index % 10)**
   Y: **100 + 50 * (index / 10)**
6. Run the model.

You should see 50 seats arranged in 5 rows, 10 seats per row. The number 50 you entered in the **Replication** field is the number of seat copies. By providing X and Y coordinates depending on the seat **index** you have arranged seats in rows. The **index** is varied from 0 to 49. The expression **index % 10** is the remainder of division of **index** by **10**, i.e. it will take the values 0, 1, …, 8, 9, 0, 1, … 8, 9, etc. We will use that as the *seat number in a row*, hence the X coordinate is the seat number multiplied by **50** to provide some spacing. The 100 is the coordinate of the first seat. Similarly, the expression **index / 10** is the integer division, and it will evaluate to 0 for seats 0-9, to 1 for the seats 10-19, etc. We will use that as a *row number*. For more information on Java arithmetic see Section 10.5.

The dynamic coordinates override the static values. It does not matter where you have drawn a shape in the editor if you have provided any values in the dynamic coordinate fields. Therefore, if we do not add 100 to the dynamic X and Y of the seats, the first seat will be placed at (0,0).

Row and seat numbers are replicated text shapes, see steps 7-10

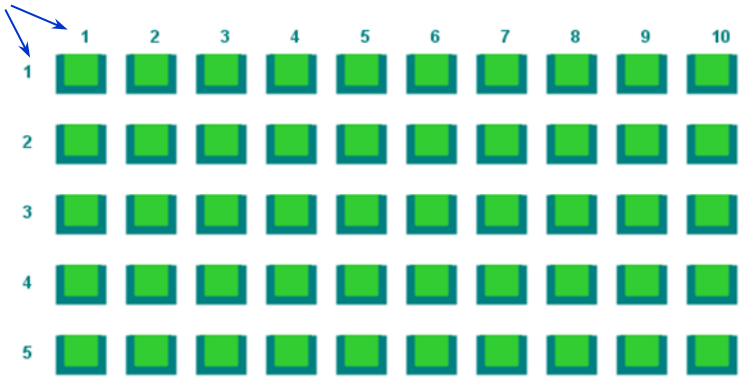

Figure 12.39 Movie theater seats at runtime – 50 instances of a polyline

Draw seat and row numbers:

7. Create two text shapes, set the static text for both to e.g., "1", and place one above and the other – on the left of the seat polyline, as shown in Figure 12.40.
8. For both texts set **1 + index** as the dynamic value for **Text**, set the **Font size** to **12**, select the **Bold** option, and set the **Color** to **teal**.
9. For the upper text set these properties:
   Replication: 10
   X: 115 + 50 * index

10. Similarly set the following properties for the text on the left:
    Replication: 5
    Y: 105 + 50 * index
11. Run the model.

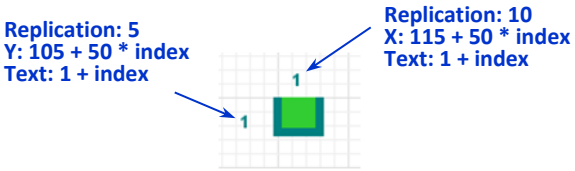

Figure 12.40 Seat and row numbers – replicated text shapes

For these text shapes we have only set one dynamic coordinate (X or Y), which means that the value for the other coordinate will be the same as the static value, i.e. the coordinate in the graphical editor. As all array and collection numbering in Java starts with 0, we need to add 1 to the **index** to have the row and seat numbers starting with 1.

If AnyLogic detects a numeric value in a dynamic expression defined in the **Text** field, it will automatically convert it to text, so you should not worry about it.

Example 12.6: Selling seats in the movie theater

Now we will make the previous example interactive and show how the properties of a replicated shape can be linked to a dynamically changing collection of values. Imagine the picture you have created appears on the ticket agent's screen at the theater box office. The unsold seats show in green. The ticket agent clicks on the seats he sells, and they become red. To display the sold seats, we need to remember them. We will use Java array (see Section 10.6) of Boolean values – one value per seat. **true** will mean sold.

Figure 12.41 Java array of Boolean values

Follow these steps:

1. Open the **Agent** palette and drag the **Variable** element to the graphical editor as shown in Figure 12.41. Name the new variable **sold**.
2. In the properties of the **sold** variable set:
   Type: select **Other...**, type **boolean[]** in the field on right.
   Initial value: **new boolean[50]**

3. Select the seat animation (the polyline) and set its dynamic properties:
   Fill color in the **Appearance** section: **sold[index] ? red : limeGreen**
   On click in the **Advanced** section: **sold[index] = true;**
4. Run the model.
5. As the model runs, click on some arbitrary seats.

The way you have defined and initialized a Java array of Boolean values in this example is common for all kinds of Java arrays: you need to specify the array type in the **Type** field and initialize the array with the corresponding newly created construct.

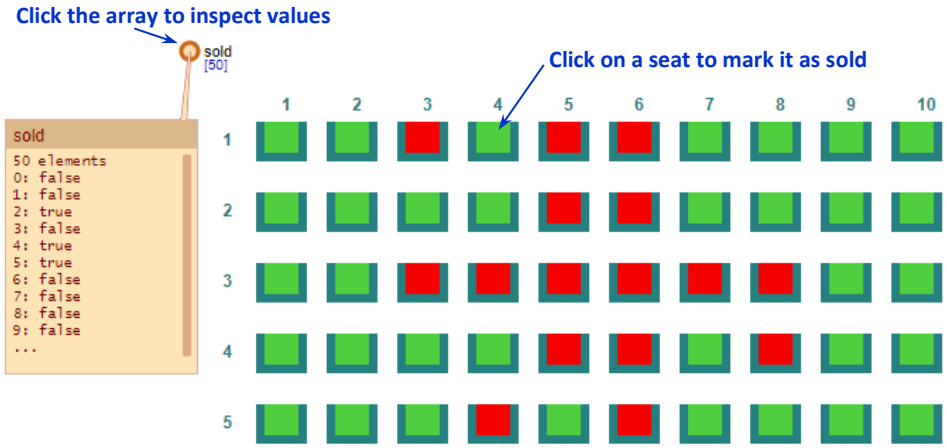

Figure 12.42 Interactive animation of seat selling process

The code you enter in the **On click** field gets executed when the user clicks on the shape (see Section 13.3 for more information about handling clicks). The variable **index** is available in this field and it identifies the seat where the user has clicked. The code **sold[index] = true;** sets the value of the array element corresponding to the seat to **true** (by default the Java boolean variables are initialized as **false**). This way the user action through graphics changes the model variable **sold**. And the fill color of the seat dynamically reflects the status (**red** for sold, **limeGreen** for unsold).

Example 12.7: Drawing a flower

This small entertaining example shows how to use replication combined with rotation to arrange shapes in a ring. We will draw a flower petal and then replicate it around the flower center.

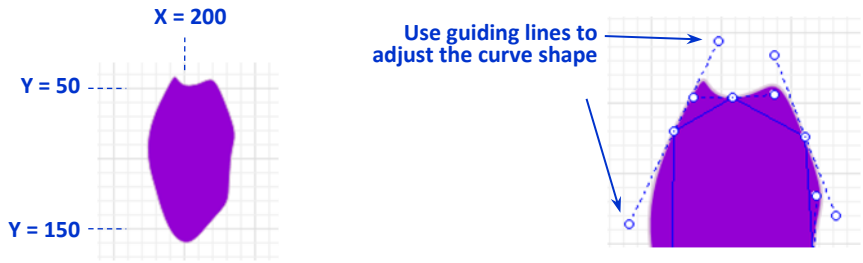

Figure 12.43 A flower petal

Follow these steps:

1. Select the **Curve** shape in the drawing mode (see Section 12.1) and draw a flower petal approximately as shown in Figure 12.43. The center of the petal should be at X = 200, and the bottom – slightly lower than Y = 150.
2. Select the **Closed** checkbox in the curve properties.
3. Set the line color of the curve to **thistle** and the fill color – to **darkViolet**.
4. Right-click the curve and choose **Edit Using Guiding Lines** from the context menu.
5. Use the guiding lines to adjust the curve shape.
6. Right-click the curve and choose **Grouping | Create a Group** from the context menu. The group origin appears at the center of the curve.
7. Drag the group so that its origin is at (200,200).
8. Right-click the group and choose **Select Group Contents**. The curve (as the only member of the group) gets selected.
9. Move the curve upwards to the position where it originally was drawn. The group origin should remain at (200,200).
10. Draw gold and red circles with the center at (200,200) as shown in Figure 12.44. Send them back so that they appear below the petal.
11. Select the group containing the petal curve. In the **Advanced** section of the properties set **Replication**: **12**
12. In the **Position and size** section, switch the **Rotation** field to a dynamic value editor (the label changes to **Rotation, rad**) and type there: **index * PI / 6**
13. Run the model.

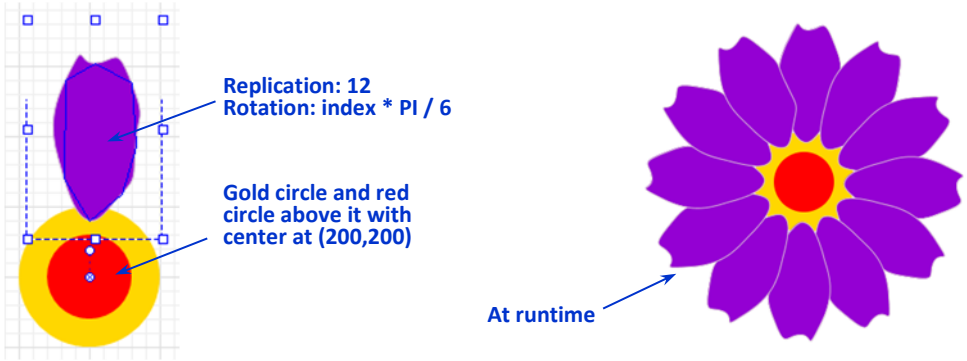

Figure 12.44 Replicating the group with the petal drawing and the resulting runtime picture

You should see a beautiful flower with 12 petals. Grouping in this example was used to provide a new rotation point for the petal shape (this technique is explained in Section 12.2). Each next petal is rotated around (200,200) by the angle **PI / 6** relative to the previous petal.

The copies of a replicated shape are drawn in their natural order, so the last copy always appears on top.

More fun:

14. Change the dynamic rotation value of the group to: **index * PI / 6 + time()**.
15. Select the curve (click the curve while its group is selected) and set the following dynamic value for **Y**: **-40 - 10 * time()**. Here **-40** is the static Y coordinate of the curve in the group (may be different in your case, see the static value of the curve's **Y**).
16. Run the model.

12.5. Shapes' API

Drawing shapes manually using the AnyLogic graphical editor is not the only way to create model graphics and using shapes' dynamic properties and replication is not the only way to control the graphics at runtime. AnyLogic offers you a rich API to access the shape properties, change, group, and ungroup shapes, create new and delete existing shapes.

Any shape is mapped by AnyLogic to a Java object, which exposes its functions to the user. For example, a rectangle shape is an instance of class **ShapeRectangle** and has functions like **getX()**, **getHeight()**, **setFillColor(Color fillColor)**, **contains(double px, double py)**, etc. A group is an instance of class **ShapeGroup** with functions

add(Shape s), **remove(Shape s)**, **size()**, **get(int index)**, and so on. Below you will find some examples of using the shape API.

In some cases, you can achieve the same effect by using either dynamic properties of the shape or the shape API. You should choose whichever way seems more natural and elegant, producing less code or producing cleaner code. For example, if the expression in the dynamic properties is too large because several alternative states are checked, it may make sense to use the API to change the property of the shape explicitly when the state changes.

Example 12.8: Using color to show the current state of a statechart

We will use the function **setFillColor()** to change the color of a shape when the state of a statechart changes. This technique is frequently used (e.g. when you develop animations of agents in agent-based models). In this example the statechart will model a consumer and will have three states: **Addressable**, **OurClient**, **CompetitorsClient**. We will draw a simple animation of a consumer and change its color from the entry actions of the states.

Follow these steps:

1. Create a new model. Set the model time units to months.
2. Use a curve to draw a picture of a consumer, e.g., like the one in Figure 12.45. You may use the guiding lines to adjust the shape.
3. Set the fill color of the curve to e.g., **silver**, and the line color - to **No color**.
4. Draw a statechart with three states as shown.
5. Set the following properties of the transitions:
   Addressable -> **branch** (decision diamond): **Rate 1 per month**
   branch -> **OurClient**: **Conditional**, **Condition randomTrue(0.5)**
   branch -> **CompetitorsClient**: **Default (is taken if all other conditions are false)**
   OurClient -> **CompetitorsClient**: **Rate 1 per month**
   CompetitorsClient -> **OurClient**: **Rate 1 per month**
6. Specify the following **Entry actions** of the states:
   OurClient: **curve.setFillColor(royalBlue);**
   CompetitorsClient: **curve.setFillColor(orangeRed);**
7. Run the model.

You should be able to see how the color of the picture changes from blue to red and back. The model of the consumer is, of course, very simplistic.

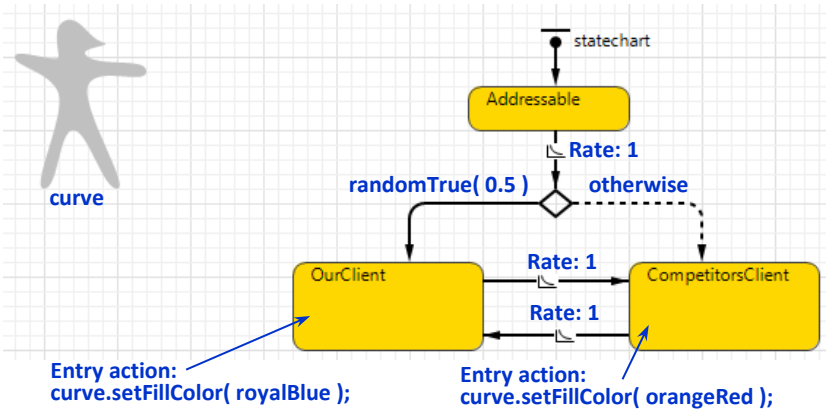

Figure 12.45 The statechart controls the shape's color by calling the shape functions

Example 12.9: Show/hide a callout

In this example we will create a dynamic callout that will be displayed when you click on certain shapes. Such constructs are used to make the models more user-friendly by displaying quick information about certain objects. The callout will be a group of shapes, for which we will dynamically change coordinates, text content, and visibility.

Draw a callout:

1. Use a polyline to draw the callout shape as shown in Figure 12.46. The polyline should be closed.
2. Set the polyline fill color to **khaki**, and line color – to **darkKhaki**.
3. Draw two text shapes on top of the callout as shown. Use the font size 11, **Bold**. The text that reads "Callout text" is black, and the text "Close" should be blue.
4. Give the text "Callout text" the name **calloutText**.
5. Select the polyline and both texts and create a group. The name of the group should be **callout**.
6. Move the group origin to the sharp end of the callout (see Section 12.2).
7. Set the advanced property **On click** of the text "Close" to:
   callout.setVisible(false);
8. Run the model. Click the "Close" text on the callout. The callout should disappear.

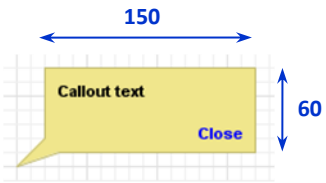

Figure 12.46 The callout

In steps 1-8 we have created the callout group. The group contains the callout background and two text elements. The text **calloutText** will be used later to dynamically display the information. The text "Close" is click-sensitive and is used to hide the callout by calling its function **setVisible(false)**.

▶ **Draw the shapes that will display the callout:**

9. Move the whole group so that its origin (the sharp end) is at (0,0). You may need to right-drag the graphical editor to do it.
10. Draw a red circle e.g. at (100,150). Name it **redCircle**.
11. Set the **On click** advanced property of the **redCircle** to:

```
calloutText.setText( "The red circle" );
callout.setPos( redCircle.getX(), redCircle.getY() );
callout.setVisible( true );
```

12. Ctrl+drag the red circle to create its copy. Set the fill color of the copy to **darkViolet** and the name – to **violetCircle**.
13. In the **On click** advanced property of the **violetCircle** change the code to:

```
calloutText.setText( "The violet circle" );
callout.setPos( violetCircle.getX(), violetCircle.getY() );
callout.setVisible( true );
```

14. Run the model. Click different circles to display the callout.

Both circles are click-sensitive. When you click any of them, they dynamically change the text of the **calloutText** shape inside the **callout** group, change the position of the callout and make it visible.

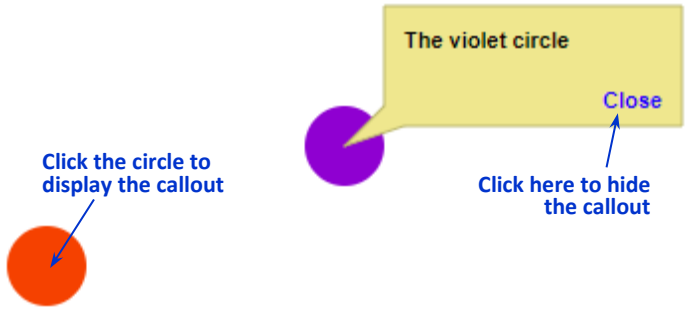

Figure 12.47 The callout at the model runtime

Example 12.10: Find all red circles

We will look through all shapes at the top-level group of the model presentation, find all red circles and add them to a collection. The technique may be useful to configure the model according to the graphics drawn by the user.

Follow these steps:

1. Draw some arbitrary picture in the graphical editor using various shapes. In that picture create several red circles.
2. Drag the **Collection** element from the **Agent** palette to the graphical editor. Name the collection **redCircles**.
3. In the properties of the collection set the **Elements class** to **Other...** and type **ShapeOval** in the field on right.
4. Click the graphical editor to display the agent properties. Expand the **Agent actions** section of the properties and enter the following code in the **On startup** section:

```
for( int i=0; i<presentation.size(); i++ ) {
  Object o = presentation.get(i);
  if( o instanceof ShapeOval ) {
    ShapeOval oval = (ShapeOval)o;
    if( oval.getFillColor() == red )
      redCircles.add( oval );
  }
}
```

5. Run the model and click the **redCircles** collection to inspect it.

The startup code gets executed after the agent and all its graphics are created so you can access the shapes there. The code iterates through all shapes of the top-level group **presentation**. Every shape is tested to be a circle and, if yes, to have red fill color. All red circles are added to the collection, which contains elements of class **ShapeOval** (this is AnyLogic Java class for circles and ellipses).

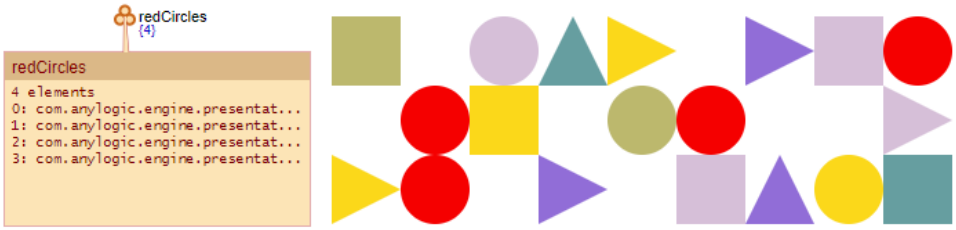

Figure 12.48 The four red circles have been found and added to the collection

Example 12.11: Resize the red circles

Now we will extend the previous example by adding two buttons that will change the size of the red circles.

Follow these steps:

1. Open the **Controls** palette and drag the **Button** element to the graphical editor.
2. Change the **Label** of the button to "**+**".
3. Expand the **Action** section of the button properties and type the following code into the field:

```
for( ShapeOval circle : redCircles ) {
  circle.setRadiusX( circle.getRadiusX() * 1.1 );
  circle.setRadiusY( circle.getRadiusY() * 1.1 );
}
```

4. Ctrl+drag the button to create another similar button below it. Change the label of the second button to "**-**".
5. Modify the code of the second button: replace "**1.1**" by "**0.9**".
6. Run the model. Click "+" and "-" buttons.

In the buttons action code, the sizes of all red circles in the collection are increased or decreased by 10%. The **get...** and **set...** functions of the **ShapeOval** are used.

AnyLogic Java class hierarchy for shapes

The full list of shapes' functions is available in *AnyLogic Help. Advanced Modeling with Java: API reference* (The AnyLogic Company, 2019). In Figure 12.49 below we give the Java class hierarchy for shapes. The classes in boxes represent actual shapes, the ones without are intermediate classes.

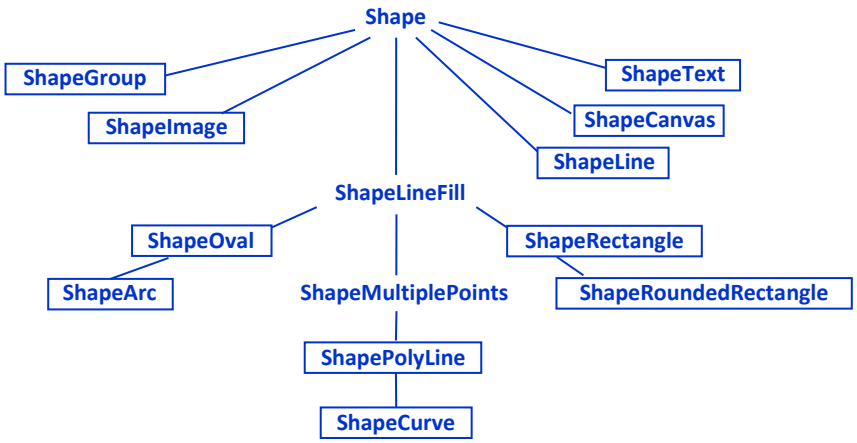

Figure 12.49 AnyLogic Java classes for shapes

12.6. Colors and textures

Static colors and textures of AnyLogic graphical objects (shapes, controls, charts, etc.) are set using the **Colors dialog** and **Color picker**. Dynamic colors (those that may override the static colors at runtime) are defined using expressions in the dynamic properties of shapes (see Section 12.3). You can also set colors using the shape API (see Section 12.5).

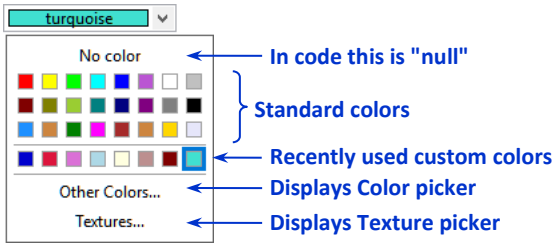

Figure 12.50 Colors dialog

The **Colors dialog** that pops up when you click the button near the color box (see Figure 12.50) contains 24 standard and 8 recently used custom colors. To set a different color or a texture you should choose **Other Colors...** or **Textures...** from the menu.

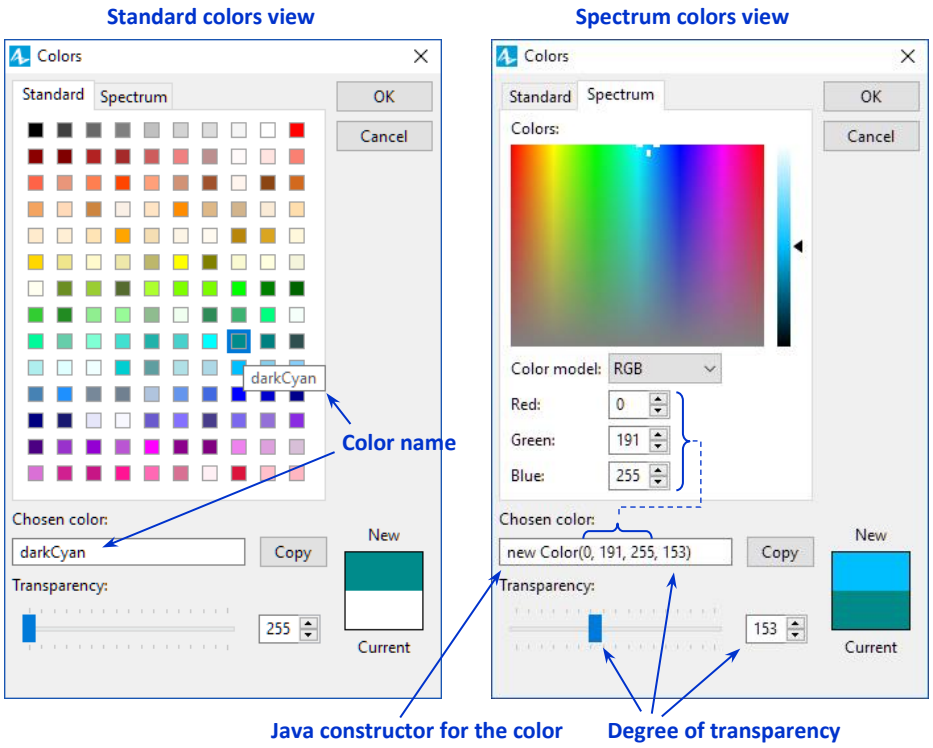

Figure 12.51 AnyLogic color picker

While the color picker saves the chosen color in the shape properties, you can also use it to obtain the Java representation of the color, which you can insert in your expressions and code. The code equivalent for "no color" is **null**. The standard colors have names like **red**, **blue**, **yellow**, **limeGreen**, **gold**, etc. - these are the names of Java constants of type **Color**. For the nonstandard colors you should use constructors (see Section 10.2) like **new Color(139, 14, 191)** or **new Color(0, 255, 0, 128)**. To copy the Java representation of a color, click the **Copy** button nearby.

The word **new** is needed to tell Java that a new color is created. This happens each time the expression with **new...** is evaluated. Therefore, if you are using constant colors in your code, especially in the dynamic properties of shapes (which are evaluated on every frame), it makes sense to define a constant of type **Color** and refer to it in the code. The constant will be created once, so the simulation performance improves.

There are several functions that help to manipulate colors. For example, to obtain a darker or brighter version of a color **color** you can call:

```
color.darker()
```

color.brighter()

Color is the standard Java class and you can find more information on these and other functions in Java class reference (Oracle, 2019). In addition, AnyLogic offers some useful functions, e.g. **spectrumColor()**, **semiTransparent()**, and **lerpColor()**.

Example 12.12: Obtaining nice colors for an arbitrary number of objects

Assume you have several different but similar objects in your model, which you wish to display using different colors that go well together. The function **spectrumColor(index, period)** will return an attractive color with a given **index** out of **period** different colors evenly distributed over the whole spectrum. In the example below 10 (the period) can also be a variable.

Follow these steps:

1. Use a polyline to draw a shape, e.g. a star like shown in Figure 12.52.
2. In the **Advanced** section of the properties of the star set:
   Replication: 10
3. In the **Position and size** section switch the **X** property to the dynamic value editing mode and set **X: 100 + index * 50**
4. In the **Appearance** section set the dynamic value for the **Fill color**:
   spectrumColor(index, 10)
5. Run the model.

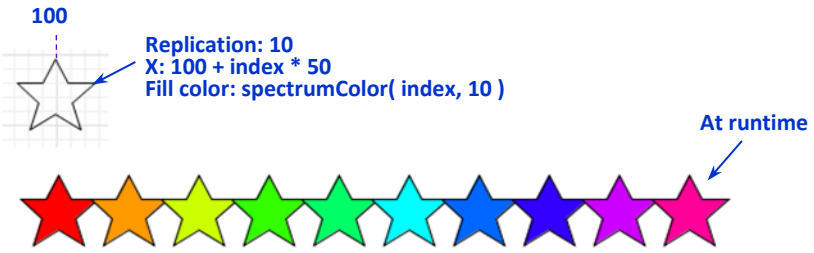

Figure 12.52 Good looking colors obtained using the function spectrumColor()

Transparency

Transparency can be used in many ways to enhance the graphical presentation of your models. For example, a semitransparent color can show a certain (virtual) area on top of a main map or plan. Dynamically changing degree of transparency can reflect a certain property of a model object, e.g., density, readiness, level of emergency, etc. If you have many thin lines which display the links between agents they may look better semitransparent. You can choose a semi-transparent color for parts of your histograms, charts or plots, and so on.

Static transparent colors can be set by using the color picker. The **Transparency** slider changes the degree of transparency from 255 (fully opaque) to 0 (fully transparent). If there is any degree of transparency (i.e., the transparency has other value than 255), the fourth parameter is added to the Java color constructor as shown in Figure 12.51.

There is also an easy way of obtaining the semitransparent colors programmatically: you should call the function **semiTransparent(** *<base color>* **)**, which produces the color of the same hue and transparency equal to 128.

To control the degree of transparency dynamically you should vary the fourth parameter of the color constructor. For example, you can use the expression like **new Color(255, 0, 0, x)** in the dynamic **Fill color** property of a shape where **x** is an integer that varies between 0 and 255, to obtain semitransparent red colors.

Example 12.13: Using transparency to show coverage zone

We will use semitransparent circles to show a hypothetical cell phone coverage zones on top of the map of the USA.

Follow these steps:

1. Drag the **USA Map** element from the **Pictures** palette to the graphical editor.
2. Create several circles (element **Oval** in the **Presentation** palette) and place them over the map approximately at the locations of the following cities: New York, Chicago, San Francisco, Los Angeles, Seattle, Orlando, Houston, and Boston.
3. Adjust the sizes of the circles as shown in Figure 12.53.
4. Select all circles by Ctrl+clicking them.
5. In the **Appearance** section of the circles' properties set the line color to **No color**.
6. Open the **Color picker** (see Section 12.6) for the fill color.
7. Click the red color (the top right square).
8. Move the **Transparency** slider to the position 100 (you can finely adjust the value by using the Arrow keys).
9. Click **OK**.

You will see the semitransparent circles on top of the map. Note that the color is more intense where the circles overlap.

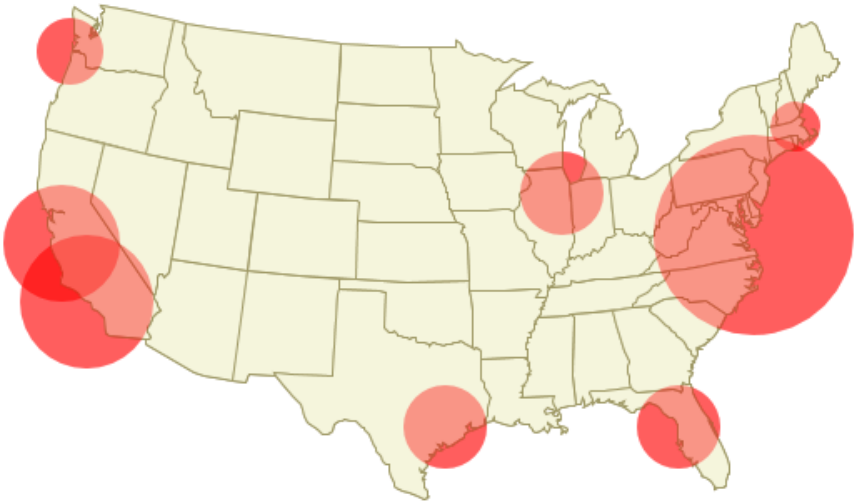

Figure 12.53 Coverage map shown using semitransparent colors

Example 12.14: Show population density using color interpolation

We will use AnyLogic function **lerpColor()**, which performs linear *color interpolation*, to visualize the dynamically changing population density of US states. We will use the USA map that can be found in the **Pictures** palette. The map is a group of state shapes, so we can set the dynamic fill color property for any state individually. We will consider only two states: Texas and California and use system dynamics stock and flow diagram to model the hypothetical migration.

Follow these steps:

1. Drag the **USA Map** element from the **Pictures** palette to the graphical editor.
2. Draw the system dynamics diagram (see Section 5.1) as shown in Figure 12.54. Enter the following values and formulas (the initial values are taken from the (United States Census Bureau, 2019), the migration is of course made up):
   PopulationCA initial value = **36756666**
   LandAreaCA = **155959**
   DensityCA = **PopulationCA / LandAreaCA**
   Migration = **PopulationCA / 10**
   PopulationTX initial value = **24326974**
   LandAreaTX = **261797**
   DensityTX = **PopulationTX / LandAreaTX**
3. Click the US map (this will select the whole group) and then click California to select the California shape.

4. In the properties of the California shape, set the following dynamic expression for the **Fill color**:
   lerpColor(DensityCA/500, white, red).
5. Similarly, set the dynamic fill color for the Texas shape to:
   lerpColor(DensityTX/500, white, red).
6. Run the model.

The system dynamics model assumes 10% of the California population moves to Texas each year, and the population density variables change in the range 0..500 approximately. Let us assign white color to zero density and red color to density 500 persons per square mile. The function **lerpColor(fraction, white, red)** returns **white** if **fraction** <= 0, **red** if **fraction** >= 1, and a color in between white and red if **fraction** is between 0 and 1. Therefore we need to divide the density by 500 to get a value between 0 and 1.

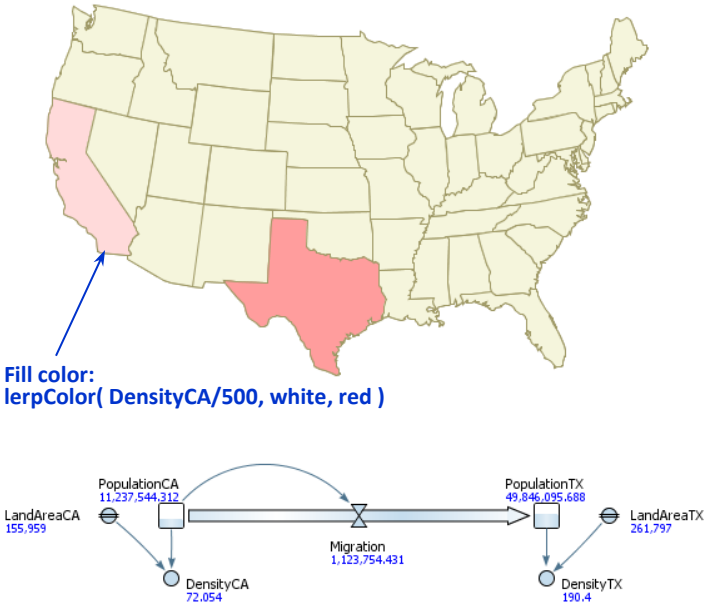

Figure 12.54 The state population density shown by color between white and red