Chapter 1

Painting with Trachel

Any sophisticated visualization boils down primitives graphical elements. Ultimately, no matter how complex a visualization is, circles, boxes, labels, lines are the bricks to convey information.

Before jumping into Roassal, it is important to get a basic understanding on using primitive graphical elements. Trachel is a low-level API to draw primitive graphical elements. This chapter briefly describes Trachel.

1.1 Core of Trachel

Trachel is composed of the following components:

- *Canvas*: a container of shapes. Adding a shape to the canvas makes it visible and responsive to user events.
- *Shape*: a visual entity, such as colored line, circle, box, and text.
- *Event*: a user action typically involving mouse and keyboard.
- *Camera*: a model that describes the visible portion of the canvas. A camera has a position and a scaling factor.
- *Viva*: infrastructure to animate shapes.

1.2 Using the Canvas

A canvas is a set of graphical shapes that may react to events and may be shown using the camera. As the first example, open a playground from the World menu and enter the code:

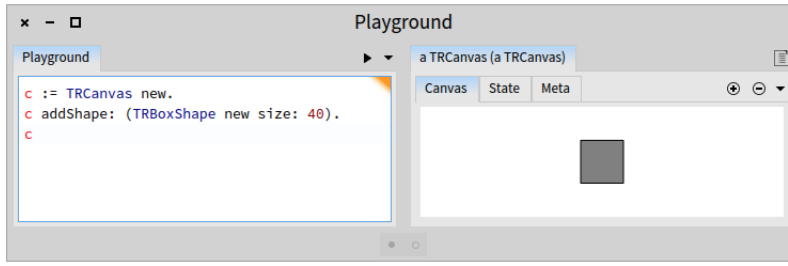

Figure 1.1: Instantiating canvas and a shape.

```
c := TRCanvas new.
c addShape: (TRBoxShape new size: 40).
c
```

Execute the code by pressing the **Do it all and go** button (the same option is accessible from right clicking on the whole selection, or pressing Cmd-G or Ctrl-G). Figure 1.1 shows what you are currently seeing.

A canvas is simply created by instantiating the class TRCanvas. A shape is added to the canvas by sending the message addShape: with a shape as argument to a canvas. In the example given above, the TRBoxShape describes a box which has a size of 40 pixels. Since we have not specified a color of a shape, gray is used. Gray is the default color for most shapes.

A shape may be removed from a canvas by simply sending the remove message to shape.

```
c := TRCanvas new.
shape := TRBoxShape new size: 40.
c addShape: shape.
shape when: TRMouseClick do: [ :event | event shape remove. c signalUpdate ].
c
```

On the example given above, the shape is removed from the canvas by clicking on it. This example uses events, which will be described below. The message signalUpdate is sent to the canvas to refresh the window. This message has to be sent whenever the canvas is modified *after* being opened. The message addShape: does not need to be followed by signalUpdate since the canvas is opened after the script execution.

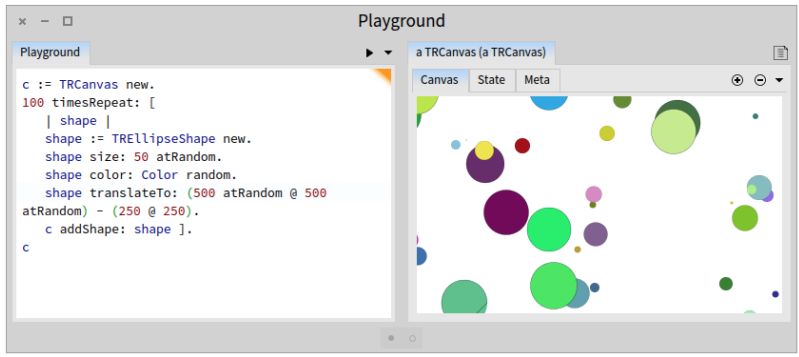

Figure 1.2: Random size and colors.

1.3 Adding shapes

Size and colors of shapes is defined by sending size: and color: to a shape. Consider the following example:

```
c := TRCanvas new.
100 timesRepeat: [
   | shape |
   shape := TREllipseShape new.
   shape size: 50 atRandom.
   shape color: Color random.
   shape translateTo: (500 atRandom @ 500 atRandom) - (250 @ 250).
   c addShape: shape ].
c
```

Consider the script given in Figure 1.2.

Several shapes are available. The most commonly used are TRBoxShape, TREllipseShape, TRArcShape, TRBezierShape, TRBitmapShape, TRLineShape, TRPolygonShape, TRSVGPath.

In addition, a new shape may be easily defined. This is useful in case you need to answer a particular need that cannot be easily fulfilled otherwise.

All Trachel shapes answer the following messages:

- translateTo: to move a shape to a new position
- translateBy: to move a shape using a step, specified as a point
- topPosition, leftPosition, bottomPosition, rightPosition returns the position of a shape taken from a particular side.

- topPosition:, leftPosition:, bottomPosition:, rightPosition: set the position based on a given position and a side of the shape. Note that these methods may be invoked only on boxes and ellipses.

- extent:, extent to set and query the dimension of the shape. The result of extent is a point for which the x component represents the width and the y the height.

Most of the methods listed above requires a point as argument. Each trachel shape is described by a matrix underneath. This affine matrix contains the position, scale factors and rotations values.

The color of a shape may be set using colors:. The border color using strokePaint: and the border width using strokeWidth:.

1.4 Fixed Shapes

Shapes visible in a canvas depends on the position and altitude of the camera. Some shapes may not be subject to the camera: some objects may remain fixed even if the camera moves. This is the case for a menu button for example. Consider the following script (Figure 1.3):

```
c := TRCanvas new.

100 timesRepeat: [
   shape := TREllipseShape new.
   shape
      color: Color random;
      size: 30 atRandom;
      translateTo: (400 atRandom @ 400 atRandom).
   c addShape: shape ].

button := TRLabelShape new text: 'move'.
c addShape: button.
button setAsFixed.
button translateBy: 30 @ 30.

button when: TRMouseClick do: [ :evt |
   c camera translateBy: 4 atRandom @ 4 atRandom.
   c signalUpdate ].
c
```

The code given above creates a new canvas, and adds one hundred ellipses, each having a random size, color, and position. The variable button holds a label. By sending the message setAsFixed to it, the label remains located on the top left corner. Clicking on it slightly moves the camera. All

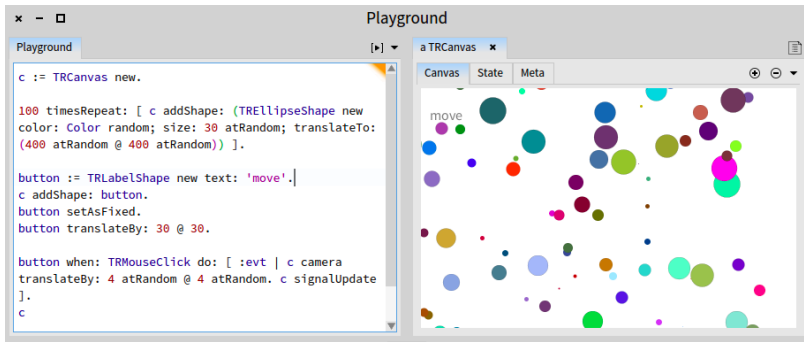

Figure 1.3: Fixed shape.

the circles moves accordingly. The label button remains at the same position, since it is fixed.

1.5 Scaling and rotating

Several operations may be performed on a shape.

- scaleBy: makes the elements scale.
- rotateByDegrees: incrementally rotates the shape
- rotateToDegrees: set the rotation of a shape

1.6 Callback and event handling

A *callback* refers to a piece of code that is trigged from an event. Such events are typically mouse movements, mouse clicks or keyboard strokes. Callbacks are registered using the message when:do:. The first argument is an event class (subclass of TREvent). Several events classes are available to cover common user actions. The example given above associates a callback to the object button: the block provided as second argument is evaluated when clicking on the Trachel shape.

1.7 Roassal generates Trachel shapes

Trachel is a simple object model on top of which Roassal is built. In practice, Trachel is rarely directly used to visualize data. Instead, Roassal offers expressive operations that create and manipulate Trachel shapes.

A visualization is updated by directly modifying Trachel shapes. Most Roassal objects are used when building the visualization. Once the visualization is built and displayed, then Roassal objects are pretty much useless if no interaction is embedded.