Chapter 1

Painting with Trachel

Trachel is a low-level API to draw primitive graphical elements.

1.1 Core of Trachel

Trachel is composed of seven components:

- *Shapes* represents graphical elements, such as colored lines, circle, boxes, and texts.
- *Canvas* is a container of shapes. Adding a shape to the canvas makes it visible.
- *Camera* models the visible portion of the canvas. A camera has a position and a scaling factor.
- *Events* covers actions the end-user may perform using the mouse and keyboard.
- *Callback* covers transformations a shape may have.
- *Constraint* defines constraints between one or more set of shapes (e.g., a shape is located on the right hand side of another shape).
- *Focuses* are particular actions to locate the camera on particular position.

Most Graphic libraries comes with a API to draw. A major difference between Trachel and traditional painting API is the camera. Visualizing data requires a flexible handling of the camera, which represent the visible portion of the data. Navigating in a large amount of data is greatly easier using

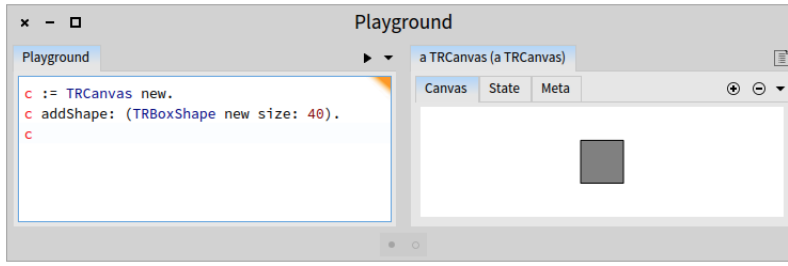

Figure 1.1: Instantiating canvas and a shape.

a camera. Another important point of Trachel is to relieve the programmer from handling the refresh loop for most of the tasks.

1.2 Canvas

As the first example, open a playground from the World menu. And enter the code:

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

Press the **Open** button or right click and select open. You should now see the equivalent of Figure 1.1

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 force a refresh of the window.

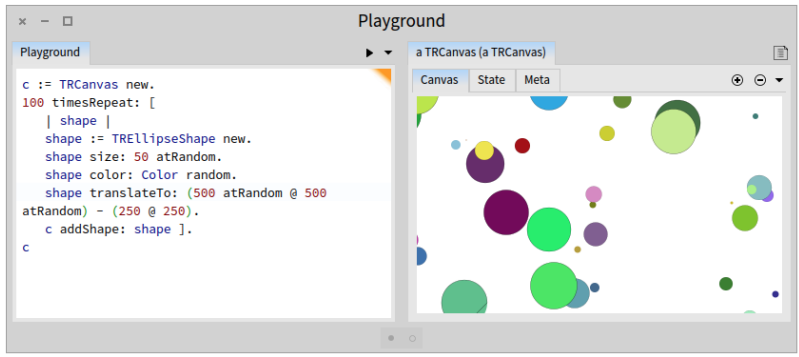

Figure 1.2: Random size and colors.

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 execution of the script.

1.3 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.

The different Trachel shapes available are

- TRBoxShape
- TREllipseShape
- TRArcShape
- TRBezierShape

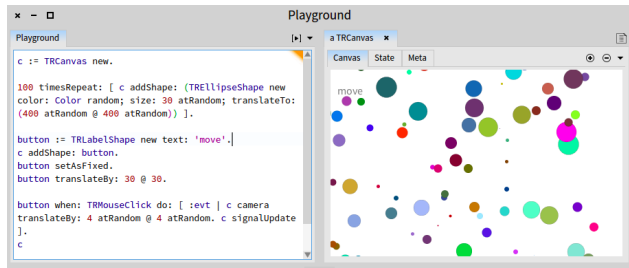

Figure 1.3: Fixed shape.

- TRBitmapShape
- TRLineShape
- TRPolygonShape
- TRSVGPath

1.4 Fixed Shapes

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

```
c := TRCanvas new.

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

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 add one hundred of ellipse, with a random size, color, and position. The variable button creates 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 the circles moves accordingly. The label button remainds fixed.

1.5 Positioning

Each shape has a position. A shape is created at position `0 @ 0`. In the example given previously, the box is created and positioned at `0 @ 0`, which corresponds to the center of the window.

The unit used in the position are pixels. However,

A shape answers to the messages:

- translateBy:
- translateTo:

1.6 Size

Book Todo:

1.7 Scaling and rotating

Book Todo:

- scaleBy:
- rotateByDegrees:
- rotateToDegrees:

1.8 Colors

Book Todo:

1.9 Callback

- when:do:

1.10 Constraints between shapes

The class TRConstraint

```
TRConstraint  class >> stick: aShape onTheTopLeftOf: anotherShape
    | b |
    self move: aShape onTheTopLeftOf: anotherShape.

    b := [ :shape :step | self move: aShape onTheTopLeftOf: anotherShape ].
    anotherShape addCallback: (TRTranslationCallback block: b).
    anotherShape addCallback: (TRExtentCallback block: b)
```

1.11 Why Does Trachel Matter?

Roassal will be introduced in the following chapter.

Updating the visualization should be done by manipulating trachel shapes. Roassal shapes are good to create elements. Once an element has been created, modifying its shape should not result in an update of the visualization. Because there are some case where one do not want this.

Modifying a visualization should be done by modifying trachel shapes. Or building appropriate Roassal interactions.

1.12 Visualizing data

Book Todo:
Give a motivation of having Roassal.