Chapter 1

Quick Start

This chapter gives a quick introduction on launching and getting a first try of Roassal. The chapter assumes you have (i) some basic notion of programming and (ii) have a connection to internet.

1.1 Basic Installation

From scratch (novice)

Installing Roassal is fairly easy. It is just the matter of downloading a couple of files. You simply need to download two .zip files and unzip them. The files are available form:

- http://bit.ly/lastMoose weights about 25 Mb and contains 3 files: .image, .changes, and a source file.
- A virtual machine is also needed. You can get it from http://pharo.org/download. Pick the virtual machine according to your platform and operating system. The virtual machine weights less than 5 Mb.

These files are also available from the website http://AgileVisualization.com. Create a folder and unzip the two downloaded archives. And *voila*, Roassal, Moose, and Pharo are installed. Your filesystem will not be polluted: the folder contains everything you need.

Loading from Pharo (advanced)

In case you are a Pharo user and want to integrate Roassal in your working developping environment, you may be interested in loading Roassal directly

from its repository. Here is the Gofer script to execute in a Playground:

```
Gofer it
   smalltalkhubUser: 'ObjectProfile' project: 'Roassal2';
   configurationOf: 'Roassal2';
   loadStable.
```

Roassal is known to work on Pharo 4.0. Version of Roassal dating from January 2015 is known to work on Pharo 3.0.

1.2 Running Roassal

Dragging-and-dropping the file moose.image on the virtual machine will open the Moose programming environment. Moose is a platform for data and software analysis. Roassal is the visualization engine part of Moose. Moose and Roassal are written in the Pharo programming language.

1.3 First visualization

Open a workspace from the World menu and type the following (or copy if you have using an online version of the book):

```
v := RTView new.
shape := RTEllipse new size: 20; color: Color lightBlue.
shape := shape + RTLabel.

es := shape elementsOn: (1 to: 20).
v addAll: es.

RTEdgeBuilder new
   view: v;
   objects: (1 to: 20) from: [ :i | i // 2 ].

RTTreeLayout on: es.
es @ RTDraggable.
v
```

Press the open button. Alternatively, select the whole content of the playground with Ctrl-A or Cmd-A and press Ctrl-O or Cmd-O. You should obtain Figure 1.1. The script begins with creating an empty view. A shape is then defined. From the shape graphical elements are created. Each circle element represents a number. Elements are then added to the view. Edges are built as follows: for each number i between 1 and 20, an edge is created from the element representing i // 2 and i. The expression a // b returns the

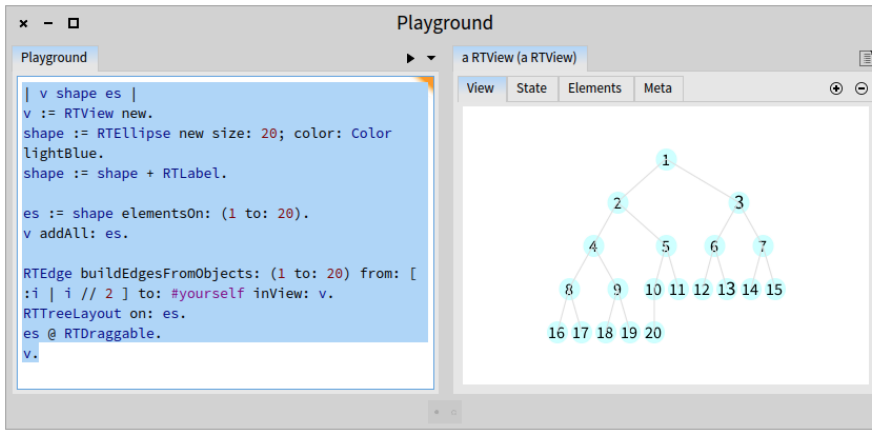

Figure 1.1: Connecting numbers.

quotient a and b, *e.g.*, 9 // 4 = 2 and 3 // 2 = 1. Elements are then ordered as a tree layout. Each element is draggable with the mouse.

1.4 Geographical CSV data

Comma-separated values (CSV) is a file format close to what spreedcheat are manipulated. Roassal has facilities to extract data from CSV files. The following example shows earthquakes over the last 30 days:

```
tab := RTTabTable new input: (ZnEasy
   get: 'http://earthquake.usgs.gov/earthquakes/feed/v0.1/summary/2.5_month.csv')
       contents usingDelimiter: $,.
tab removeFirstRow.
tab replaceEmptyValuesWith: '0' inColumns: #(2 3 4 5).
tab convertColumnsAsFloat: #(2 3 4 5).
b := RTMapLocationBuilder new.
b shape circle
   size: [ :v | 2 raisedTo: (v - 1) ];
   color: (Color red alpha: 0.3).
tab values do: [ :row | b addPoint: row second @ row third value: row fifth ].
b build.
```

You should obtain a picture ressembling at Figure 1.2.

The class ZnEasy is used to retrieve the data from the US Earthquake Hazards Program (http://earthquake.usgs.gov/earthquakes/feed/v0.1/). The retrieved data looks like the following lines:

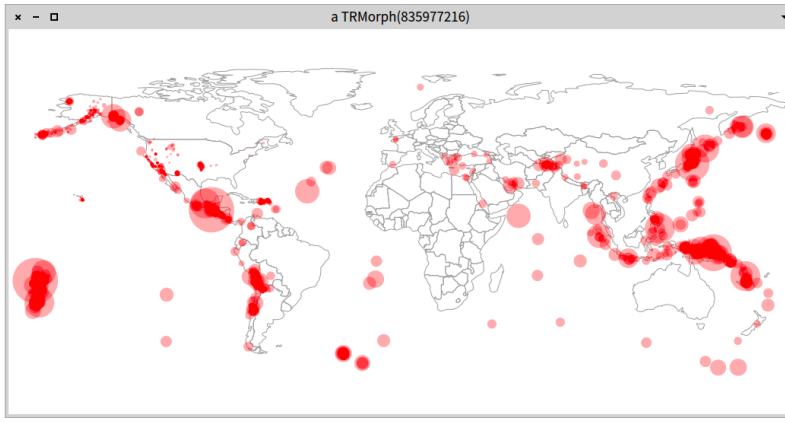

Figure 1.2: Seisms on Earth the last 30 days.

```
'DateTime,Latitude,Longitude,Depth,Magnitude,MagType,NbStations,Gap,Distance,
    RMS,Source,EventID,Version
2014-07-31T12:56:24.800+00:00,38.741,-122.714,1.2,0.9,Md,8,112,,0.04,nc,
    nc72268926,1406813044917
2014-07-31T12:49:08.000+00:00,67.655,-162.002,15.5,3.5,ml,,,,0.65,ak,
    ak11344820,1406813786856
2014-07-31T12:49:01.000+00:00,59.713,-142.589,99.9,1.4,ml,,,,5.95,ak,
    ak11344819,1406813766175'
```

The class RTTabTable is made to convert and extract values from the data. We first remove the header ('DataTime,Latitude,...'). Empty values are converted with '0'. Portion of the data matching ',,' is transformed into ',0,'. Since the table is composed of string characters, we need to convert them as numerical floats. Column 2, 3, 4, and 5 are converted into float numbers (*i.e.*, decimal numbers, such as 4.5). We create a RTMapLocationBuilder, which will be feed with the extracted values. We create a shape, as a transparent circle. Radius of each circle represents an earthquake of magnitude m is given by 2^{m-1}. This formular si rather arbitrary but it gives a descent output.

1.5 Time line

Time lines are useful abstractions to communicate evolution of some properties or metrics against the time. Using the same example, the following script produces a time line of the earthquakes (Figure 1.3):

```
tab := RTTabTable new input: (ZnEasy
```

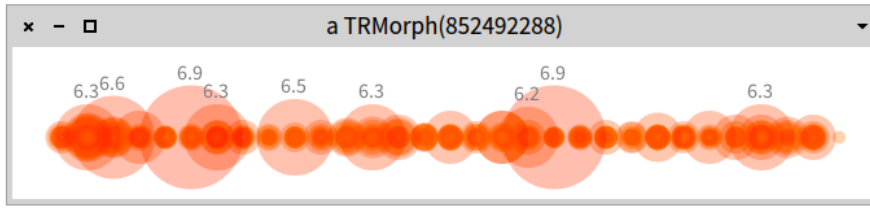

Figure 1.3: Timeline of earthquakes.

```
    get: 'http://earthquake.usgs.gov/earthquakes/feed/v0.1/summary/2.5_month.csv')
        contents usingDelimiter: $,.
tab removeFirstRow.
tab convertColumn: 1 to: [ :s | (DateAndTime fromString: s) julianDayNumber ].
tab convertColumnsAsFloat: #(5).

v := RTView new.
es := RTEllipse elementsOn: tab values.
v addAll: es.
es @ RTPopup.

RTMetricNormalizer new
    elements: es;
    normalizeColor: #fifth using: { Color orange . Color red };
    alphaColor: 0.3;
    normalizeX: #first min: 0 max: 600;
    normalizeSize: #fifth min: 0 max: 80 using: [ :mag | 2 raisedTo: (mag - 1) ].

es @ (RTLabelled new text: [ :row | row fifth > 6 ifTrue: [ row fifth ] ifFalse: [ '' ] ]).
v @ RTDraggableView.
v
```

The first column is converted into the julian day number. This is useful to order the earthquakes later on. The fifth column is then converted as a float. An ellipse is associated to each entry of the table. The size of each ellipse and its colors is deduced from the fifth column (earthquake magnitude). The first column, the Julian Day Number, is used to horizontally orders the ellipse. Each earthquake with a magnitude greater than 6 has a title.

The **Julian Day Number** (JDN) is the number of elapsed days since the beginning of a 7,980 years cycles. This number is frequently used by astronomer. The starting point for the first Julian cycle began on January 1, 4713 BC.

1.6 Charting

Roassal has a sophisticated mechanism to draw charts. *Charter* takes as input a set of data points, which could be anything, and draw from them.

The following example illustrates Ebola outbreaks:

```
"Preparing the data"
tab := RTTabTable new input: (ZnEasy get: 'http://bit.ly/EbolaCSV') contents
    usingDelimiter: $,.
tab removeFirstRow.
tab replaceEmptyValuesWith: '0' inColumns: #(10 11).
tab convertColumnsAsInteger: #(10 11).
tab convertColumnsAsDateAndTime: #(3 4).
data := tab values reversed.

"Charting the data"
b := RTGrapher new.
b extent: 400 @ 200.

ds := RTStackedDataSet new.
ds interaction fixedPopupText: [ :row | row value at: 12 ].
ds dotShape ellipse
    color: (Color blue alpha: 0.3);
    size: [ :row  | (row at: 11) / 5 ].
ds points: data.
ds connectColor: Color blue.
ds y: [ :r | r at: 10 ].
ds highlightIf: [ :row | (row at: 10) > 100 ] using: [ :row | row third year ].
b add: ds.

b axisX noLabel; numberOfTicks: tab values size.
b axisY noDecimals.
b build.
```

Figure 1.4 gives the output of the script. Vertically is the number of fatalities. Horizontally the event.

The CSV data is first obtained from http://mapstory.org using the bit.ly URL shortener. MapStory is a wonderful website with many short and illustrative stories. Outbreaks are contained in the file in a reverse chronological order, which is why we first order it properly. Each point is a row in the table. The use of fixedPopupText: gives a popup for each data point when the mouse is above the data point. The column 12 of a row is a description of the outbreak.

Each data point is a circle. Its size reflects the value of the column fatality rate (% of death per contamination). The column 10 indicates the number of fatalities. This value is used on the Y-axis. Only major events are labelled with the year (the third column correspond to the year of the outbreak).

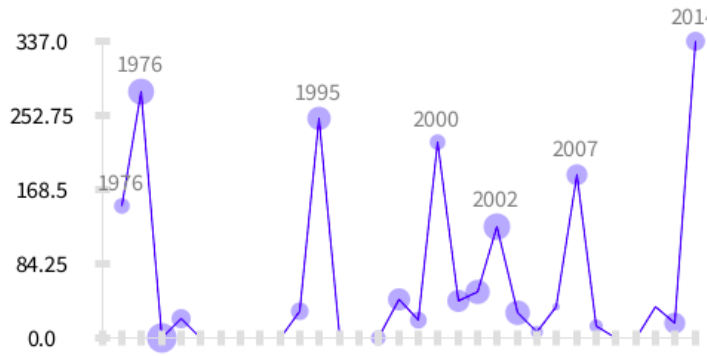

Figure 1.4: Ebola fatalities.

Charter offers several visualization types. Double bar chart are effective for small dataset and with two metrics to be represented. Consider the following example that illustrates the distribution of people in North American states (Figure 1.5).

```
tab := RTTabTable new input: (ZnEasy get: 'http://bit.ly/CensusGov') contents
       usingDelimiter: $,.
tab removeFirstRow.
tab convertColumnsAsInteger: #('POPESTIMATE2013' 'POPEST18PLUS2013').

b := RTDoubleBarBuilder new.
b pointName: [ :row | row at: (tab indexOfName: 'NAME') ].
"Remove the first line, the sum"
b points: tab values allButFirst.
b bottomValue: [ :row | ((row at: (tab indexOfName: 'POPESTIMATE2013')) / 1000)
       asInteger ]
   titled: 'Pop estimate (x 1000)'.
b topValue: [ :row | ((row at: (tab indexOfName: 'POPEST18PLUS2013')) / 1000)
       asInteger]
   titled: 'Pop +18 estimate (x 1000)'.
b build
```

Data are obtained from http://census.gov, a wonderful source of social data. During the data processing, columns are accessed from the column name. Rows are then provided to the builder b. The first line is removed since its it contains the value for the United State in the whole, which would distort the visualization.

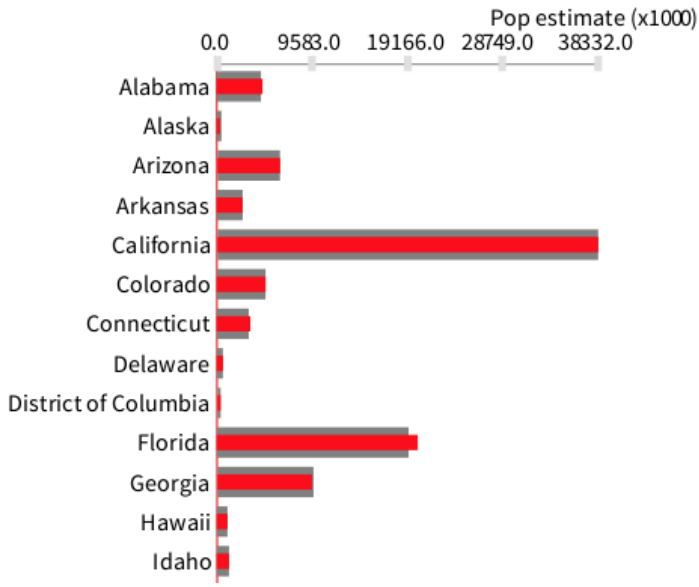

Figure 1.5: Population state in the USA.