Agile Visualization with Pharo

Crafting Interactive Visual Support Using Roassal

—

Alexandre Bergel

Apress®

Agile Visualization with Pharo

Crafting Interactive Visual Support Using Roassal

Alexandre Bergel

Apress®

Agile Visualization with Pharo: Crafting Interactive Visual Support Using Roassal

Alexandre Bergel
Santiago, Chile

ISBN-13 (pbk): 978-1-4842-7160-5 ISBN-13 (electronic): 978-1-4842-7161-2
https://doi.org/10.1007/978-1-4842-7161-2

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: James Markham
Coordinating Editor: Mark Powers
Copyeditor: Kezia Endsley

Cover designed by eStudioCalamar

Cover image by Wirestock

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

Alexandre Bergel, Ph.D., is a professor (associate) at the Department of Computer Science (DCC), at the University of Chile and is a member of the Intelligent Software Construction laboratory (ISCLab). His research interests include software engineering, software performance, software visualization, programming environments, and machine learning. He is interested in improving the way we build and maintain software. His current hypotheses are validated using rigorous empirical methodologies. To make his research artifacts useful not only to stack papers, he co-founded Object Profile.

About the Technical Reviewer

Jason Whitehorn is an experienced entrepreneur and software developer and has helped many companies automate and enhance their business solutions through data synchronization, SaaS architecture, and machine learning. Jason obtained his Bachelor of Science in Computer Science from Arkansas State University, but he traces his passion for development back many years before then, having first taught himself to program BASIC on his family's computer while in middle school. When he's not mentoring and helping his team at work, writing, or pursuing one of his many side-projects, Jason enjoys spending time with his wife and four children and living in the Tulsa, Oklahoma region. More information about Jason can be found on his website: `https://jason.whitehorn.us`.

CHAPTER 1

Introduction

Computers are a formidable extension of the human brain: a computer liberates us from performing boring and repetitive tasks. Data visualization is a wonderful field where computers nicely complement what the brain excels at.

Conveying information through interactive visualizations is both a sophisticated engineering process and an art. When crafting a visualization, many decisions have to be made based on a carefully evaluated design aspect or a personal intuition. Either way, being able to quickly experiment with a new idea is key. Agile Visualization is about leveraging creativity by reducing the cost associated with building visualizations.

Visualizing data is probably the easiest part of the field of data visualization. Numerous books and sophisticated libraries exist for that very purpose. One of challenges of data visualization is to identify the right abstractions to build a visualization that is reusable, composable, extensible, navigable, and produced at a very low cost. Roassal is a visualization engine for Pharo and Smalltalk that leverages the experience of crafting and using data visualization.

Roassal is written in the Pharo programming language. All the examples provided in this book are therefore made for Roassal and are written in the Pharo programming language.

Since there is no better way than programming to craft a visualization, readers are expected to have some programming experience to fully enjoy Agile visualization. This book is written for a large audience, and it provides the necessary technical background as a starter for programming with Pharo.

Agile Visualization

Agile visualization promotes the creation of a visualization based on very short, incremental steps. A data analysis is carried out by building a number of visualizations, many of which are simply attempts and have a very short usage time. Reducing the creation time of a visualization to a few seconds or minutes greatly increases the number of different paths the data scientist can take to solve a given problem.

A. Bergel, *Agile Visualization with Pharo*, https://doi.org/10.1007/978-1-4842-7161-2_1

Jackson Pollock, a famous American painter, once said: “Splatter painting celebrates spontaneity, improvisation, and a highly physical approach to making art.” Colors are thrown, mixed, and removed at will. This metaphor may be considered as a guiding line of Agile visualization. Similar to Agile programming, feedback should always occur a short time after the inception of a visualization.

The Pharo Programming Language

Roassal is developed in the Pharo programming language. All the source code provided in this book is written in Pharo. Pharo is a dynamically typed programming language, sharing some flavors with Smalltalk, Python, and Ruby. Pharo is easy to install, learn, and use. Pharo has a very simple syntax, which means that the code is understandable as soon as you have some programming knowledge. Pharo is both a programming language and a powerful environment. This book provides a light introduction to the syntax of Pharo and presents an overview on how to use and extend its environment.

If you do not know Pharo, here are a few pieces of advice. Pharo is easy to learn and use. It comes with fantastic programming tools to make you intimately interact with *objects*; an object being an elementary computational and logical unit. Resist the natural tendency to map your knowledge and expectations into Pharo. Embracing the way of thinking with objects is rich and enlightening. The Pharo programming environment is now your new friend, and plenty of great adventures will soon come.

The `https://pharo.org/download` website gives a very detailed instruction set to install Pharo. Installing Pharo is just a matter of a couple of clicks.

The Roassal Visualization Engine

Roassal is a visualization engine developed in Pharo. Roassal offers a simple API to build a visualization and maps arbitrary domain-specific objects to visual elements. This mapping process is at the core of Agile visualization and is extensively discussed in the book.

Roassal has a great list of features that make it share some similarities with other visualization engines, including D3.js and Matplotlib. However, Roassal naturally produces interactive visualizations to directly explore and operate on the represented domain objects. Furthermore, Roassal is integrated in the Pharo environment, which leverages the experience of building visualization.

At the moment this book is written, Roassal is the most commonly used visualization engine in Pharo and in the Smalltalk communities. Although alternatives are possible, including the Morphic framework or directly drawing in a `Form` object, Roassal offers a large set of pluggable and composable tools.

Roassal License

Roassal is distributed under the MIT License, which means that you can:

- Use Roassal for commercial purposes. Roassal can be freely distributed in a business-friendly manner, without any monetary payment due to the authors of Roassal.
- Modify the original source code of Roassal and distribute it as a separate project.

The MIT License is one of the most permissive software licenses available. However, the MIT License imposes two restrictions:

- You cannot hold the original authors of Roassal liable for any damage caused by using Roassal.
- You also cannot claim Roassal is your original work. Derivatives are okay as long as the original authors get credit and their name stays on the license.

Contributing to the Development of Roassal

Roassal is the result of more than ten years of hard work made by the authors of Roassal and the Pharo community. We encourage you to contribute to Roassal as well. It is easy to become a contributor of Roassal. There are many different ways to do so:

- If you find a bug or an opportunity for improvement, you can open an issue in the GitHub repository of Roassal that describes the issue.
- If you can improve the codebase of Roassal, define a pull request.

The GitHub repository of Roassal is available at `https://github.com/ObjectProfile/Roassal3`.

Accompanying Source Code

All the source code provided in this book is kept in the following Git repository: https://github.com/bergel/AgileVisualizationAPressCode.

It's better to use the provided material instead of trying to manually transcript the code given in the book.

Want to Have a Chat?

If you want to discuss Roassal, need help, or simply have a friendly chat, there are a number of ways to get in touch. Roassal has its own channel on the Pharo Discord server. The Pharo community extensively uses Discord to communicate. You can join the Pharo Discord server by following the instruction provided at https://pharo.org/community. After joining it, you can jump on to the channel `#roassal` and say "Hi!". A number of friends' channels are also in the Pharo Discord server:

- `#roassal-scriptoftheday` regularly provides short script that's ready to be executed. These scripts typically illustrate a particular aspect of Roassal.
- `#roassal-commit` reflects the activity of the Roassal GitHub repository by listing the commits made in the repository.

Adding the hashtag `#Roassal` and the `@Roassal1` users to your tweets is also a great way to advertise ideas. You can also use the mention `@PharoProject` since the Pharo community will surely have an interest in your post. Finally, you can reach me by email at alexandre.bergel@me.com.

Book Overview

Agile Visualization is divided into 15 chapters, each targeting a specific topic in the field of visualizing and interacting with data in Pharo:

- Chapter 2 provides a tour of Roassal by presenting a few visualizations.
- Chapter 3 is an introduction to the Pharo programming language.

- Chapter 4 discusses the notion of Agile visualization.
- Chapter 5 presents the relevant components of Roassal.
- Chapter 6 explains the Roassal Canvas.
- Chapter 7 lists the shapes offered by Roassal.
- Chapter 8 describes the line builder.
- Chapter 9 details the composition mechanism of shapes.
- Chapter 10 presents the normalizers and scales offered by Roassal.
- Chapter 11 highlights the most relevant interactions supported by Roassal.
- Chapter 12 lists a number of commonly used layouts.
- Chapter 13 describes the integration of Roassal in the Inspector framework of Pharo.
- Chapter 14 applies visualizations to explain the behavior of reinforcement learning, a machine learning algorithm.
- Chapter 15 details how visualizations can be automatically generated from a GitHub repository using GitHub Actions.

As with any successful open source project, Roassal is driven by active community effort. The positive aspect of being a successful open source project is that Roassal is evolving every day (literally). The negative aspect is that documentation quickly becomes obsolete. The book is written in a way that deep technical aspects are not discussed while general concepts are largely presented. These general concepts are much more stable over time.

Who Should Read This Book?

This book is designed to satisfy various facets of a large audience:

- *Data scientists* –Readers familiar with Pharo will learn the essential components of the Roassal visualization engine. After reading the book, you will be able to apply visualization techniques to any domain data. Many examples are provided with the book and they can be used as a guide or as templates for analyses.

- *Designers of visualization engine* – Designing and implementing a visualization engine is an incredibly difficult task. Roassal went through three complete rewrites to reach its current status. Readers who want to implement an engine may definitely find valuable resources regarding the design and implementation of a visualization engine.

Acknowledgments

The book is the result of multiple and long-lasting collaborations. First of all, I would like to thank the Lam Research company. Lam Research's team has always been supportive, both psychologically and financially. Thanks CH and Chris! You made this book a reality.

Many people in the Moose, Pharo, and ESUG communities have deeply contributed to what is presented in the book. Your enthusiastic support and trust in what I do has always been invaluable.

I also would like to thank you, yes you, the reader, for your questions, support, bug fixes, contribution, and encouragement.

I am also deeply grateful to the following people for their contributions (in no particular order): CH Huang, Chris Thorgrimsson, Milton Mamani, Ted Brunzie, Tudor Gîrba, Renato Cerro, Stéphane Ducasse, Yuriy Tymchuk, Natalia Tymchuk, Juraj Kubelka, Juan Pablo Sandoval Alcocer, Vanessa Peña, Ronie Saldago, Alvaro Jose Peralta, Pablo Estefo, Igor Stasenko, Faviola Molina, Ricardo Jacas, Daniel Aviv Notario, Sergio Maass, Serge Stinckwich, Bui Thi Mai Anh, Nick Papoulias, Johan Fabry, Nicolai Hess, Miguel Campusano, Peter Uhnák, Martin Dias, Jan Blizničenko, Samir Saleh, Nicolai Hess, Leonel Merino, Volkert, Pierre Chanson, Andrei Chis, Thomas Brodt, Mathieu Dehouck, Miguel Campusano, Onil Goubier, Thierry Goubier, Esteban Maringolo, Alejandro Infante, Philippe Back, Stefan Reichhart, Ronie Salgado, Steffen Märcker, Nour Jihene Agouf, Pavel Krivanek, and Esteban Lorenzano.

CHAPTER 2

Quick Start

This chapter provides an overview of Roassal, as well as a few examples. The chapter does not go into detail since that is something that will occur in forthcoming chapters. Many short code snippets are provided and briefly described. You can copy and paste these snippets directly into Pharo and each one illustrates a particular aspect of the Roassal platform.

It is important to keep in mind that this chapter is just a tour of Roassal and Pharo. If you are not familiar with Pharo, you might find the amount of code given here a bit confusing. The next chapter serves as an introduction of the Pharo language and explains many aspects of the language syntax.

All the code provided in this chapter is available at `https://github.com/bergel/AgileVisualizationAPressCode/blob/main/01-02-QuickStart.txt`.

Installation

Roassal is the framework for the Pharo programming language. As such, the first step to try the examples given in this chapter is to install Pharo. The Pharo website provides all the necessary instructions to do so (`https://pharo.org/download`). Pharo can be installed directly via the command line or using the Pharo launcher. Both ways are equivalent and you may prefer one over the other based on your personal workflow.

Once you have installed Pharo, you need to open the Playground, which is a tool offered by Pharo to execute code. You can think of the Playground as a UNIX terminal. The Playground is opened from the top toolbar, as shown in Figure 2-1.

A. Bergel, *Agile Visualization with Pharo*, https://doi.org/10.1007/978-1-4842-7161-2_2

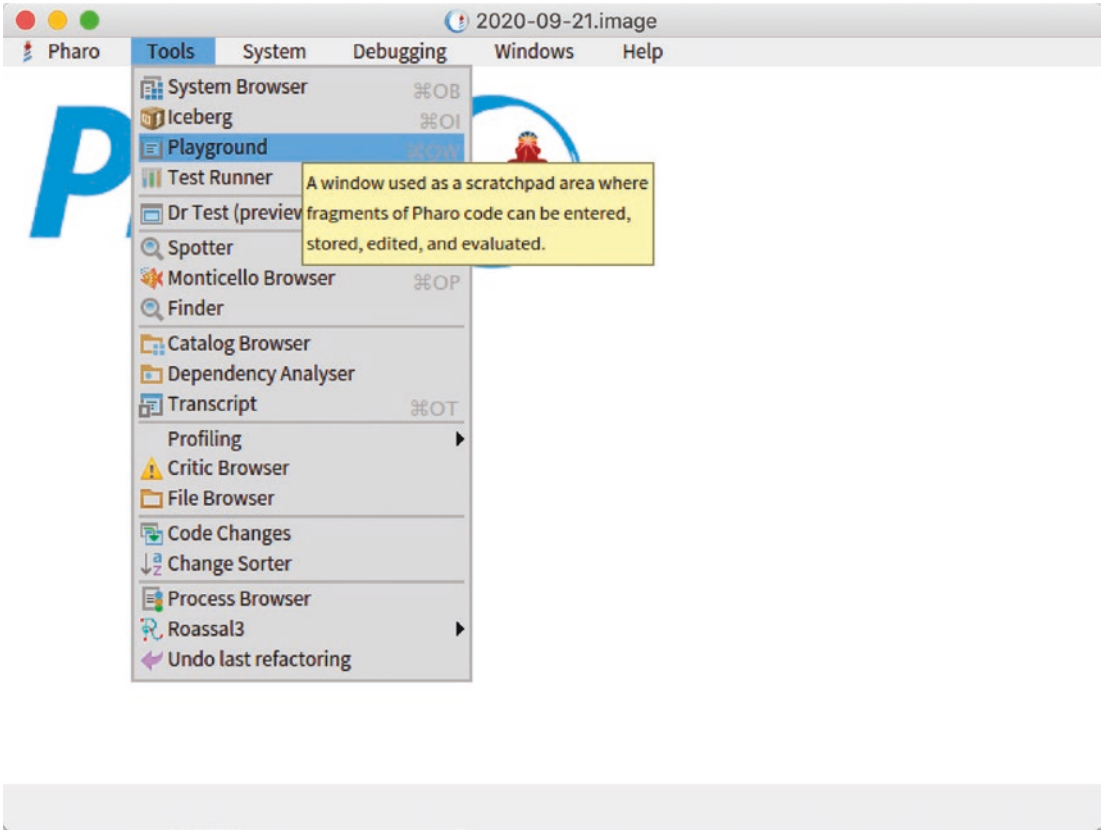

Figure 2-1. *Opening the Playground from the World menu*

After selecting Playground from the menu, you'll see a window in which you can type any Pharo instructions. Just type the following instructions (or copy and paste them if you are reading an electronic version of this book):

```
[ Metacello new
    baseline: 'Roassal3';
    repository: 'github://ObjectProfile/Roassal3';
    load: 'Full' ] on: MCMergeOrLoadWarning do: [ :warning | warning load ]
```

You can execute the code by pressing Cmd+D (if you are a macOS user) or Ctlr+D (for Windows and UNIX users), or by pressing the green Do It button.

Loading Roassal should take a few seconds, depending on your Internet connection. Once it's loaded, you may want to save your Pharo environment (a.k.a , the image in the Pharo jargon) by choosing Pharo➤Save from the toolbar menu. You are now ready to try your first visualization.

First Visualization

Most of the visualizations in this book are written as short scripts, directly executable in the Playground. You can open a new Playground or simply reuse an already open Playground (e.g., the one you used to load Roassal). In that case, you should erase the contents of the Playground and paste in the new script.

Evaluate the following example in the Playground (see Figure 2-2).

```
c := RSCanvas new.
1 to: 100 do: [ :i |
    c add: (RSLabel new model: i) ].

RSLineBuilder line
    shapes: c nodes;
    connectFrom: [ :i | i // 2 ].

RSClusterLayout on: c nodes.
c @ RSCanvasController.
c open
```

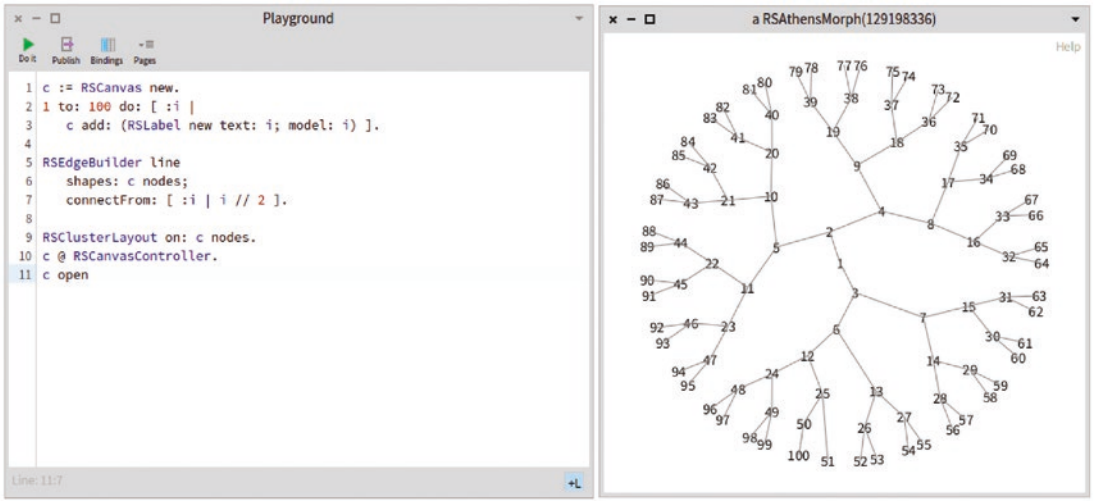

Figure 2-2. *Connecting numbers*

The script begins by creating a canvas using the `RSCanvas` class. The script adds 100 labels to the canvas, each representing a number between 1 and 100. Lines are built as follows: for each number `i` between 1 and 100, a line is created from the element representing `i //2` and `i`. The expression `a //b` returns the quotient between a and b, e.g., `9 //4 = 2` and `3 //2 = 1`. Nodes are then ordered using a cluster layout.

As a short exercise, you can replace 100 with any other value. You can also replace the `RSClusterLayout` class with `RSRadialTreeLayout`, `RSTreeLayout`, or `RSForceBasedLayout`.

Visualizing the Filesystem

You will reuse the previous visualization to visualize a filesystem instead of arbitrary numbers. Pharo offers a complete library to manipulate files and folders. Integrating files into a Roassal visualization is easy. Consider the following script:

```
path := '/Users/alexandrebergel/Desktop'.
extensions :=
    { 'pdf' -> Color red . 'mov' -> Color blue } asDictionary.
allFilesUnderPath := path asFileReference allChildren.
c := RSCanvas new.

allFilesUnderPath do: [ :aFile |
    | s color |
    s := RSEllipse model: aFile.
    color := extensions at: aFile path extension
                        ifAbsent: [ Color gray ].
    s color: color translucent.
    s @ RSPopup @ RSDraggable.
    c add: s ].

RSNormalizer size
    shapes: c nodes;
    from: 10; to: 30;
    normalize: [ :aFile | aFile size sqrt ].
```

```
RSLineBuilder line
    shapes: c nodes;
    connectFrom: #parent.

RSClusterLayout on: c nodes.
c @ RSCanvasController.
c open
```

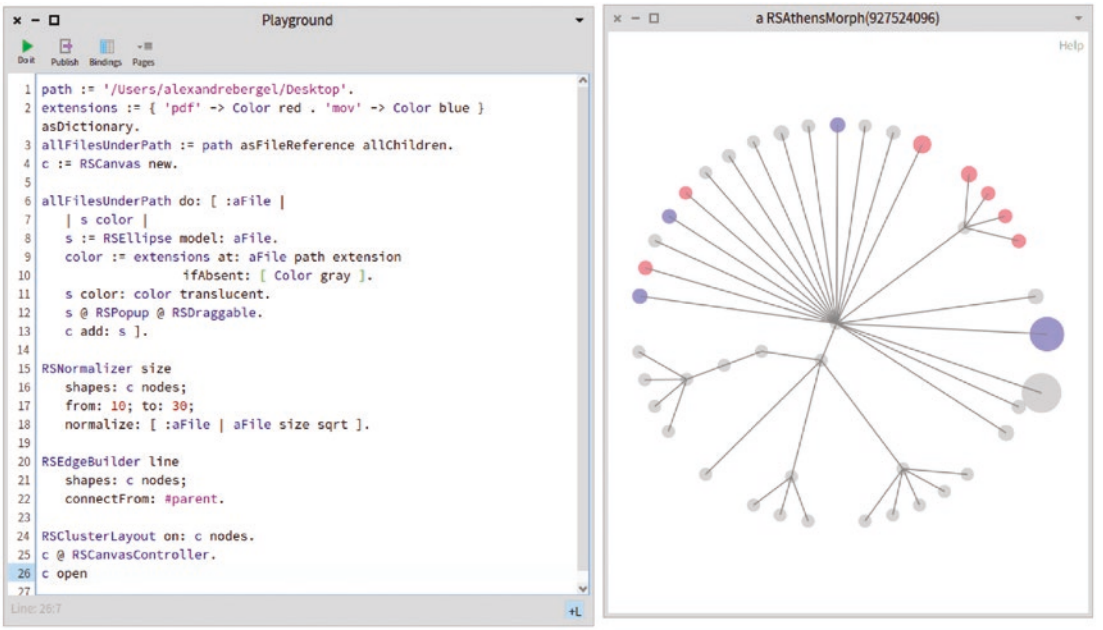

Figure 2-3. *Visualizing the filesystem*

Figure 2-3 shows the contents of the path `/Users/alexandrebergel/Desktop`, which correspond to the contents of the desktop on macOS. The `path` variable contains a location on your filesystem. Obviously, you need to change the path to execute the script. Note that indicating a large portion of the filesystem may significantly increase the computation time since recursively fetching file information is time-consuming. The `path asFileReference` expression converts a string indicating a path as a file reference. `FileReference` is a Pharo class that represents a file reference, typically locally stored on hard disk. The `allChildren` message gets all the files recursively contained in the path. The visualization paints files whose names end with `.pdf` in red and paints all videos files ending with `.mov` in blue.

Compared to the previous example, this visualization uses a normalizer to give each circle a size according to the file size. The size varies from 10 to 30 pixels, and it uses a square root (sqrt) transformation to cope with disparate sizes.

As an exercise, you can extend the color schema for specific files located on your filesystem. Color rules must follow the pattern `'pdf'-> Color red` and must be separated by a period character.

Charting Data

Roassal offers a sophisticated library to build charts. Consider the following example, showing the high of the COVID-19 pandemic during its first 250 days (see Figure 2-4).

```
url := 'https://raw.githubusercontent.com/ObjectProfile/',
        'Roassal3Documentation/master/data/',
        'covidDataUntil23-September-2020.txt'.

rawData := OpalCompiler evaluate: ((ZnEasy get: url) contents).
countries := rawData collect: #first.
allData := rawData collect: #allButFirst.
color := NSScale category20.

chart := RSChart new.
chart extent: 400 @ 400.
chart colors: color.
allData do: [ :data | chart addPlot:(RSLinePlot new y: data) ].
chart xlabel: 'Days since epoch' offset: 0 @ 20.
chart ylabel: 'Contaminated' offset: -60 @ 0.
chart title: 'Coronavirus confirmed cases'.
chart addDecoration: (RSHorizontalTick new fontSize: 10).
chart addDecoration: (RSVerticalTick new integerWithCommas; fontSize: 10).
chart ySqrt.
chart build.

b := RSLegend new.
b container: chart canvas.
countries with: chart plots do: [ :c : p |
    b text: c withBoxColor: (chart colorFor: p) ].
```

```
b layout horizontal gapSize: 30.
b build.
b canvas open
```

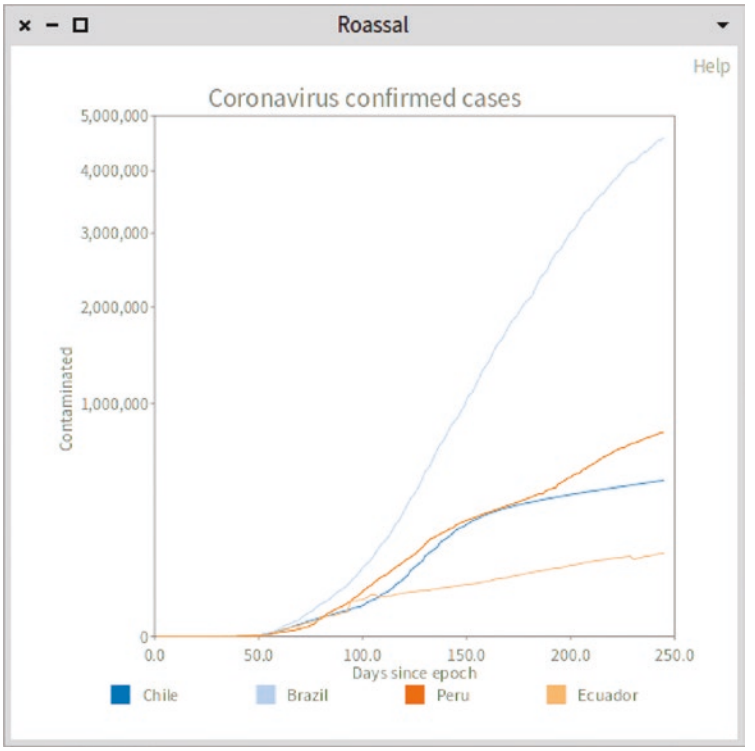

Figure 2-4. *The first 250 days of the pandemic in 2020*

The `url` variable points to a dataset that contains the first 250 days of the COVID-19 pandemic in 2020 caused by the SARS-CoV-2. Note that we split the URL to accommodate the book formatting. The data stored in the `covidDataUntil23-September-2020.txt` file is a simple serialization of the data to be rendered by the charter. The `RSChart` class is the entry point of the charting library. Plots are added to a chart and a few decorations are added. A legend is located below to associate curves with countries.

Sunburst

A sunburst is a visualization designed to represent hierarchical data structure. Consider the following example (Figure 2-5).

```
sb := RSSunburstBuilder new.
sb sliceShape withBorder.
sb sliceColor: [ :shape | shape model subclasses isEmpty
                    ifTrue: [ Color purple ]
                    ifFalse: [ Color lightGray ] ].
sb explore: Collection using: #subclasses.
sb build.
sb canvas @ RSCanvasController.
RSLineBuilder sunburstBezier
    width: 2;
    color: Color black;
    markerEnd: (RSEllipse new
        size: 10;
        color: Color white;
        withBorder;
        yourself);
    canvas: sb canvas;
    connectFrom: #superclass.
sb canvas open
```

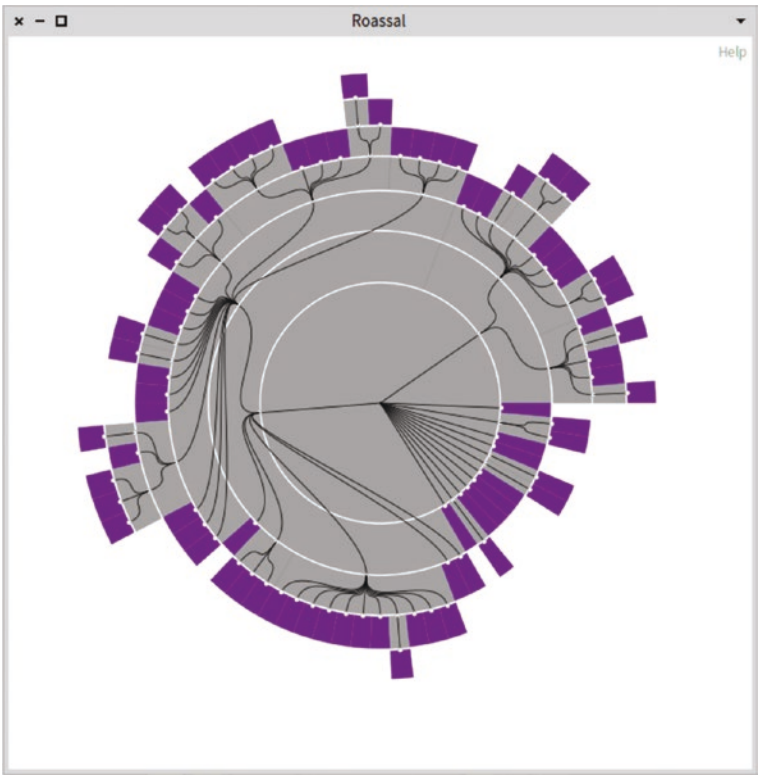

Figure 2-5. *Visualizing the collection class hierarchy using a sunburst*

This sunburst is a software visualization. Each arc represents a class, and the nesting indicates class inheritance, which is highlighted with Bezier lines.

Graph Rendering

Roassal offers a wide range of tools to manipulate and render graphs. Consider the following script (see Figure 2-6).

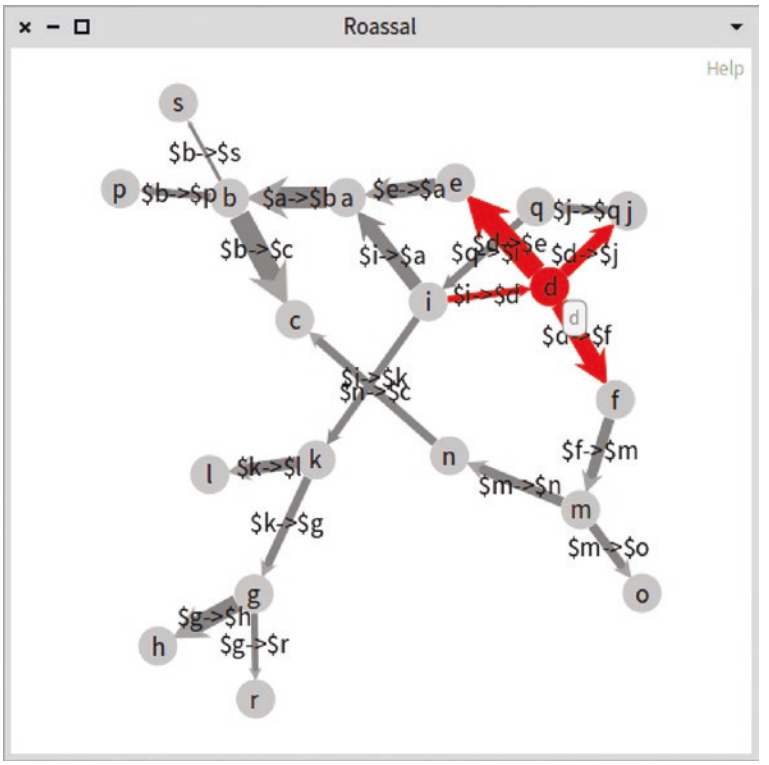

Figure 2-6. *Visualizing a graph*

```
nodesModel := $a to: $s.
edges := #( #( $a $b 30 ) #( $b $s 1 ) #( $b $p 4 ) #( $b $c 30 )
            #( $d $e 30 ) #( $d $f 20 ) #( $d $j 10 ) #( $e $a 15 )
            #( $f $m 8 ) #( $g $h 20 ) #( $g $r 3 ) #( $i $a 14 )
            #( $i $k 4 ) #( $i $d 3 ) #( $j $q 5 ) #( $k $l 10 )
            #( $k $g 5 ) #( $m $n 7 ) #( $m $o 6 ) #( $n $c 5 )
            #( $p $b 5 ) #( $q $i 4 ) ).

graph := Dictionary new.

nodesModel do: [ :aNode |
    graph at: aNode put: Set new ].

edges do: [ :edge |
    fromNode := edge first.
    toNode := edge second.
    (graph at: fromNode) add: toNode ].
```

```
canvas := RSCanvas new.
nodes := RSCircle models: (nodesModel) forEach: [:circle :model | circle
size: 20; color: Color veryLightGray. ].

nodes @ RSDraggable; @ RSPopup.
canvas addAll: nodes.

highlightable := RSHighlightable new.
highlightable highlightColor: Color red.
highlightable withEdges.
nodes @ highlightable.

lb := RSLineBuilder line.
lb canvas: canvas.
lb makeBidirectional.
lb moveBehind.
lb objects: nodesModel.
lb connectToAll: [ :aNumber | graph at: aNumber ].
canvas lines do: [ :line | | edge length |
    edge := edges detect: [ :e |
        e first = line model key
            and: [ e second = line model value ] ].
    length := edge third sqrt * 2.
    line width: length.
    line attachPoint: (RSBorderAttachPoint new
        endOffset: length).
    line markerEnd: (RSShapeFactory arrow size: length * 2).
    line markerEnd offset: length / -5.
     ].
(canvas nodes, canvas lines) @ (RSLabeled new
    in: [ :lbl |
        lbl location middle.
        lbl shapeBuilder labelShape color: Color black ];
    yourself).
RSForceBasedLayout new charge: -500; doNotUseProgressBar; on: nodes.
canvas @ RSCanvasController.
canvas open
```