Book reviews

Realm of Racket, by Forrest Bice, Rose DeMaio, Spencer Florence, Feng-Yun Mimi Lin, Scott Lindeman, Nicole Nussbaum, Eric Peterson, Ryan Plessner, David Van Horn, Matthias Felleisen and Conrad Barski, MD, No Starch Press, San Franscisco, CA, 2013, £27.49. ISBN-10:1-59327-491-2.
doi: 10.1017/S0956796814000070

Summary

When writing a review, it is best to know your audience. In this case, your humble reviewer suspects that you probably know far more Haskell than I do, spend a bit too much time writing code that builds crazed messaging topologies in Erlang, or perhaps are a Scala programmer who just can't admit that, deep down, you really wish LISP had managed to take over the world. Regardless, if you're not impressed that *Realm of Racket* starts with the Y combinator and ends with writing a networked artificial intelligence (AI) involving flame-thrower wielding squirrels, then you can stop reading now.

Who Wrote Realm of Racket?

This question matters, as the text carries the unofficial tagline 'written by freshmen, for freshmen'. Eight of the 11 authors are first year students who were in process of being introduced to Racket through the text *How to Design Programs*. With light-hearted illustrations throughout and an informal tone, the text is fun to read and pleasant to work through.

Whom is *Realm of Racket* for?

If you've never programmed before, or you have minimal programming experience, and you're interested in writing some interactive programs in Racket with a visual element, I wouldn't start with *Realm of Racket*. Instead, grab the revised version of *How to Design Programs*, and work through the first few chapters, in addition to exploring animation and interactivity as presented in that text. After getting the basics of the Racket language down, and (more importantly), learning to think in a systematic way (which *How to Design Programs* does an *excellent* job of preparing you for), then you'll be ready to dive into *Realm of Racket*.

If, on the other hand, you have spent enough time in another language to make use of some basic control structures (conditionals and loops), atomic data types (numbers, booleans), and list-type data (vectors, arrays, lists, or whatever your language calls them), then this quick dive into Racket should feel just fine.

A Walk through the text

The preface is fun; we get some history of Scheme, and some fun cartoons of Guy Steele and Gerry Sussman inventing the language. This is followed by a pretty rapid succession of chapters that serve primarily to introduce Racket as a programming language. Lists appear (including nesting dolls for CONS cells), structures, and other tools that are necessary for writing a game are covered through Chapter 4½.

Chapter 5 introduces the (big-bang) function, which is the heart of how animation is introduced in both *How to Design Programs* and *Realm of Racket*. The macro looks like this:

```
(big-bang     <state-expression>
                                  (to-draw      <draw-function>)
                                  (on-tick      <tick-function>)
                                  (on-key       <key-function>)
                                  (stop-when    <stop-function>
                                                          <optional-last-scene>))
```

Once you define state or functions to populate each of these pieces, the system takes care of launching your animated environment and managing the evolution of the world's state.

I personally thought Chapter 6 ('Recursion is Easy') was a lot of fun. In particular, it walks the reader through the authoring of *Snake*, the second most popular video game of all time (the first, I declare, would be *Solitaire* and the third being *Tetris*). And while we all like to think that we're incredibly intelligent (and good looking), I have to admit that I have never really sat down and thought about the game of *Snake*. By this I mean that I've never moved beyond the obvious questions (such as, 'why is my wife so much better at this game than me?') to important questions, such as 'how do you represent the snake and evolve its state from one time step to the next, or grow it when it eats something?'

Given the chapter title, I should have seen the solution coming. However, I still think that the structure used and the mechanism for snake growth and animation is a really nice example of the use of recursion. I liked it, both from pedagogic and simplicity-of-solution perspective.

Realm of Racket goes on to introduce lambda in Chapter 7 (with no game, but there is a nice one-page piece illustrating the definition of a simple derivative function) and the 'grizzly details of inheritance' in Chapter 8. (This is a visual pun involving teddy bears; it's wonderful.) Chapter 9 introduces looping constructs (again, no game – just syntax and examples), leading us to our first substantial game, 'Dice of Doom'.

Toward artificial intelligence

What I especially like about the second half of *Realm of Racket* is that it takes us into the realm of AI. When I say 'artificial intelligence', I do not mean the new-fangled, big-data-social-graph-genetic-algorithm kind of AI that all the kids are learning today, but instead I mean classic AI: game trees, search, and a computer program (that you're proud of) that plays Tic-Tac-Toe badly.

In Chapter 10, we're introduced to the idea of game trees, and in Chapter 11, lazy evaluation. (Yes, Haskell programmers, rejoice.) Laziness is introduced through `delay` and `force`, but we don't do anything with it. It is a short chapter, and it sets us up for Chapter 12, where the real work of implementing laziness takes place.

Chapter 12, in addition to being about lazy evaluation, is where the flamethrower-wielding squirrels enter the picture. Not only is this a radical (and welcome) departure from the content of most programming-related texts, it is also the subject of a desktop background that you can download from the book's website, http://realmofracket.com/. I have to admit, this chapter took me back to my graduate studies at Indiana University Bloomington, where this same kind of transformation to laziness took place at various points in my study of AI as well as languages (through the authoring of interpreters… not my work as a graduate student). If you're working through the chapter, you'll see the minimax search algorithm, and be challenged to look other algorithms up on the Internet and consider implementing them for the computer player. I have to admit, I do wish the authors had at least provided a pointer or two; mentioning alpha-beta pruning or A* wouldn't, I don't think, have removed too much agency for the reader, and might put them on the right track as they consider this challenge.

Chapter 13 introduces networking for communication between game instances, and Chapter 14 brings it all together in one final game, 'Hungry Henry'. Finally, with a 'close paren', *Realm of Racket* leaves you with hints of where you might go: object orientation, syntactic meta-programming, and the design of languages and their interpretation.

Conclusions

Realm of Racket is a book written 'by freshmen, for freshmen'. It introduces readers with some programming experience to Racket, a language in the Scheme family. This introduction leverages the years of extensive work that have gone into Racket (the language) and DrRacket (the program development environment), and through the development of games, prepares the reader for thinking seriously about the design of complex, interactive programs. The design of structured, linked data (game trees), lazy evaluation, memorization, and the basics of search and AI await the reader who makes their way through to the end of the text. It isn't quite a textbook, and it certainly isn't 'Game Programming in 21 Days'. It is, however, a wonderful text for delving into a language with a long and rich history.

The text is informal, the illustrations whimsical and fun, and flamethrower-wielding squirrels await the intrepid programmer who makes their way through to the end. Indeed, I wonder if I might have picked up Scheme more easily, back in the day, had a text like this existed for me at the time. It puts important ideas (both regarding the language and more generally regarding interactive application design) in a context that is natural, and it does it well. All of that said, I am easily swayed by things like rodents armed with military-grade weapons; but, when it comes down to it, I can't help but ask: what isn't there to like?

MATT JADUD
Computer Science, Berea College, Berea, KY, USA

Beginning Haskell, by Alejandro Serrano Mena, Apress, New York City, NY, 2014, ISBN-10: 1430262508, 428 pp.

doi:10.1017/S0956796814000112

This book is a modern introduction to Haskell. It is refreshingly up-to-date, covering topics such as the EclipseFP plugin for the Eclipse IDE, view patterns, and even – towards the end of the book, as a taster for going beyond Haskell – dependently typed programming in Idris. (It is worth noting that recent Haskell language extensions available in the Glasgow Haskell Compiler (GHC) do provide alternatives to full-fledged dependent types, so even this chapter could be indirectly useful to a professional Haskell programmer.)

The book assumes, very reasonably, that the reader will be using a recent version of GHC, and that the Haskell 2010 standard (modulo a few tweaks) will constitute the default language environment to work in. Although Haskell programs are usually compiled, the book sensibly starts by introducing the reader to the Haskell interpreter GHCi – an invaluable tool for learning Haskell and for quickly checking one's understanding of a concept or function. The book also introduces the reader to EclipseFP, but it is always presented as a convenient alternative, not as a tool that must be used, which I think is the right approach.

Topics are introduced in a logical order – but that does not mean according to a dry, dusty, strict taxonomy. Interesting asides and useful tips are inserted at relevant points in the text. Priority is given to clear step-by-step exposition over immediately introducing an