Polyglot Conway

Adrian Kosmaczewski

2021-11-05

My personal project during the pandemic was Conway, a project providing implementations of Conway's Game of Life in as many programming languages as possible.

I like to see how the same idea looks in a different programming language; I like to see the ecosystem around those languages as well: libraries, unit testing, documentation, code formatting and linting, etc.

It started as a teaching tool. I used the first version of it, initially in just TypeScript for the web, Swift for iOS, and Kotlin for Android, to highlight the differences and similarities in syntax and drawing libraries of those three programming languages.

And then I said, why not in other languages? And I started piling them up. At this time there are 20 of them.

In each project I try to use the default package manager, patterns and best practices. For example ~~Conan~~ vcpkg for C++ and C, pip for Python, npm for TypeScript, Composer for PHP, and so on. Modern languages like Go, Rust, and .NET have a built-in package manager, and that's very handy.

In the cases of Common Lisp and Smalltalk I had to first learn the language from scratch before starting. I used this exploration as the source for my article about Smalltalk in De Programmatica Ipsum, published in October 2020.

For the Pascal and FreeBASIC versions I had to remember how to write them, as I had not used any of these languages since the mid nineties.

All the implementations share the same very simple architecture: the world of cells is represented as a hash table whose keys are coordinates, and whose values are cells; cells being those things that can be alive or dead at any time. A world can "evolve" into its next stage of life, applying the rules of life to each cell. Most languages provide something akin to a hash table, except for C, which took me to add one inspired from some code I found online.

All projects share the same unit tests, too, written in the default unit test library provided by each language. For Objective-C, however, I could not find a suitable framework to use, so I just created a separate executable with `assert()` calls

in it. I wanted to use an old Objective-C unit test framework called UnitKit, written by Duncan Davidson (the creator of Tomcat and Ant, by the way) but couldn't make it work.

Most versions only work on the terminal, drawing the grid in pure text. Only the first three original implementations mentioned above (TypeScript, Swift, and Kotlin) have GUI representations, but I have not updated the mobile projects in years, and I don't think they can work anymore. The web app in TypeScript still works. Maybe the C++ could be used for a GUI app with Qt or wxWidgets, maybe. I would love to use the C# code to write a text-based UI using gui.cs by none other than Miguel de Icaza.

The implementation that took me the longer was FreeBASIC, simply because I hated it. It's really a horrible language to write code with.

The ones that I like most are F# and Rust. I just fell in love with these languages.

As another curiosity, I also keep statistics of the lines of code taken by each project, extracted with the cloc tool. It turns out that Pascal and Rust are the longest implementations (around 340 lines) while F#, Python, and Common Lisp are the shortest (around 140 lines).

I plan to add more languages to the project, but I don't really follow a plan. What I have been doing lately, however, is to update some implementations to the latest versions of each language. For example, I've lately updated projects to Python 3.10 and Java 17, and this month I'll do the same with PHP 8.1 (enums!) and C# 9. Sometimes these updates decrease dramatically the total number of lines of code of a project.

The project is in GitLab, free for everyone to use.