# The story behind Stelvio (a new SPG solving program)

When I (Reto Aschwanden) first started composing proof games in the early 2000s, Natch had already been around for a while and soon after Euclide appeared on the scene. Around 2005 I met Etienne Dupuis, the author of Euclide, once in Paris and he gave me a birds-eye view of how Euclide worked. I was fascinated by the complexity of the algorithmic problem at hand. At the same time, I imagined the massive amount of work it would take to create my own program, and I came to the conclusion that sadly, for lack of time, I would never try.

Over ten years later, on a foggy November day, I started nonetheless. It was clear to me from the outset that programming this was not just a means to an end, it was the end in itself. I enjoyed thinking about this complex programming puzzle. I had seen the Euclide code shortly around 2005 but did not dive deep into it at the time, and so I had to start from scratch. But that was also a deliberate choice: I did not want to copy what the other solving programs did, I'd rather come up with my own algorithm. In doing so, I might miss out on some really good ideas that Euclide and Natch contained, but on the upside, I had greater chances of coming up with novel approaches. If you see a solution to a difficult problem somewhere, this automatically steers your thoughts in that direction. If you do not have such a solution at hand, then you are forced to find your own. It will take longer, it might by more painful, but at the same time, you could possibly come up with something completely different. In the end, just creating a copy of Euclide/Natch is of use to no one. The only thing that I copied is that Stelvio has the same four phases as Euclide: Initial analysis, strategy seeking, strategy analyzing and strategy playing.

This project turned out to be a very long one, and at around 2019, I even thought that I would probably abandon altogether. The issue was that I only worked on it a few days at a time and then not at all for several months. This way of going at it was of course extremely inefficient and error-prone. Programming something like this is not like filling in Excel sheets. Every time I restarted, I had to think myself back into the program. Because of this and for other reasons, I took quite a few wrong turns and had to rewrite thousands of lines of code. The problem is, that the code needs to be extremely fast, otherwise it is useless. On the other hand, the code should also be clear, since it is already inherently complex and nobody wants added complexity on top of that. These two requirements bite each other, since making everything as clear as possible also makes it slow. A side note for all the software professionals: The old wisdom that you should make a program correct first and improve performance later does not apply here. If the core algorithmic ideas are not good ones that perform well, no amount of profiling and fine-tuning will fix that. You'll have to restart from scratch in such a case. In order to avoid such a scenario, I always included performance considerations when writing code. That includes tolerable amounts of redundancy here and there, something I usually avoid at all costs in other software endeavours.

A bit unnerving in the whole process was the lingering uncertainty if the program would ever be of any use. Before you have implemented most of the logic, you are pretty much left in the dark of how it will eventually perform. And there are many design choices you have to make early on, when you do not really have a clue what is best.

One very uncommon trait of this project is the fact that it was certain that the goal would never change: Solve orthodox SPGs as fast as possible. No new chess rules would emerge. No other requirements would ever pop up. In other software projects, the goal is a moving target, sometimes fast moving.

In 2022, I had some months off from work, and finally I had a decent stretch of time to try to finish a first version of Stelvio. And that is what I did, at least up to the point where I could give it to some testers. While testing, Michel Caillaud came up with many good ideas for improving the strategy analysis logic, and I subsequently refined that area. Very helpful in the whole testing process was a PDB export that Gerd Wilts generously supplied. Many bugs were found because of it, reaching from trivial mistakes, to stupid ones but also to really subtle ones. I exported the WinChloe SPGs as well and in the end I was equipped with almost 3300 SPGs that were said to be correct. Not all of them are useful for testing though, as some take a very long time to solve. But around 2800 can reasonably be used for test cases.

At the time I also got hold of a disused server from the place I work at. Definitely not the newest and hottest hardware you can find, but at least a lot of it. And hey, it was for free, so I'm certainly not complaining! This server is now heating up my basement with its 56 CPUs and 0.5 TB RAM. Especially at the end of development, when I was cleaning up the code, this server was invaluable to test my adjustments.

In order to get some kind of benchmark for performance, I started to test many problems with Euclide and dump the solving times into a database. Then I did the same for Stelvio. As a result I was able to compare the solving times of the two programs. The differences are fascinating: Sometimes Euclide is faster than Stelvio, but at least for my limited test set, it is mostly the other way around. The differences are sometimes huge, and I should analyze the cases where Stelvio is too slow in the future. Stelvio is obviously missing something in those cases. But I also know of many cases where Stelvio is finished quickly and Euclide gets nowhere within hours. All in all I am pleased with Stelvios performance so far, given that for this first release, I've not yet implemented all the optimizing ideas that I want to. Especially the realm of strategy analyzing and playing need further improvement. But those ideas can be added later on, when a first release has found to be stable.

This software project is easily the most complex one I've ever been involved in. No contest. One of my most demanding intellectual endeavours ever - and very satisfying at the same time.

## Some technical notes

For all geeks out there. Stelvio is written in Java, which might suprise some, as a JVM based language might not seem the best fit at first glance. I am convinced that the language choice does not have a big performance impact, as long as the language meets a few requirements. One of them is being able to access massive amounts of memory (hundreds of gigabytes are easily useful, even terabytes), which is not a problem in Java, but there are limitations for applications running in a browser environment (at least at the moment, as far as I know).

I'm not concerned with SPGs which are solved in very short time. Any time below 5 seconds is fast, and I do not care if it is 0.2 seconds or 4.5 seconds. So any performance penalty that comes with slower ramp up time because of compiling the code by the JIT compiler at the start does not matter much in my opinion. Pretty quickly, all the important methods will have been compiled anyway, as they are executed millions if not billions of times. And if you really want, you can even force the compilation up front. On the upside, Java comes with garbage collection and other nice features, together with a huge tool set. And I professionally write code in Java, so coding in this language feels natural and I probably made fewer mistakes because of it. The learning curve to start programming Stelvio was basically nonexistent, which would not have been the case for any other language.

What came as a slight surprise is the raw speed that today’s hardware provides. I was a bit blurred by slow applications like Word, Excel or some browser based applications. But when an application is programmed with performance in mind, then the possible speed is fascinating. Even on my run-of-the-mill notebook, Stelvio can play up to 3 million moves a second depending on various factors. I find this quite remarkable, considering that every move comes with a few checks like "is this piece pinned", "are the required squares empty", "am I in check", "do I have enough free moves to play this", "do we attain a new position or one that we attained before" etc.

I made some performance comparisons with Euclide and a few with Natch. The differences are massive, often orders of magnitude. For that reason, anything less than one order of magnitude can be considered basically equivalent and down to implementation details. Differences by one order of magnitude or more cannot be due to language choice or other minor details. These kinds of differences result in one program seeing a way to shortcut the calculation drastically that another does not. In the end, it is the algorithm that matters not the language it is written in.