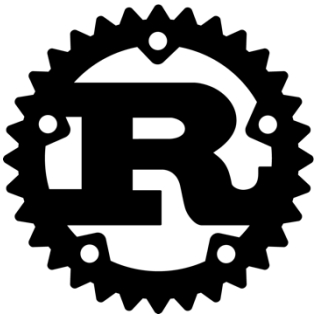

Rust Case Study:

Chucklefish taps Rust to Bring Safe Concurrency to Video Games

How Rust is providing a stable future for Chucklefish's games

Rust at Chucklefish

Chucklefish, an independent game studio based in London, publishes hit video games like Stardew Valley and Starbound. Now, the company is developing its next game, code-named **Witchbrook**, using the Rust systems programming language instead of C++. Why the switch? Two main reasons: to get better performance on multiprocessor hardware and to have fewer crashes during game play.

Rust is known for enabling fearless concurrent programming, the ability to run code safely on multiple processors at the same time.

The team at Chucklefish is excited about Rust because they're seeing fewer crashes and bugs than with C++ without sacrificing portability or requiring garbage collection. After working with Rust on Witchbrook, the Chucklefish engineers decided to also use Rust to build a scalable web service for matchmaking in the game Wargroove, planned for release later in 2018.

Screenshot of Chucklefish's upcoming game, Witchbrook

Challenges with Existing Solutions for Game Programming

Most video games today are written in C++. Whether they're 2D or 3D, video games have many large data structures to hold the massive information necessary for the game to render the graphics. The game needs to cache that data for quick access and, paradoxically, also change the data frequently to be responsive. If a game is written in a garbage-collected language like C# or Java, the game experience can be noticeably affected by pauses for

garbage collection. Chucklefish used C++ for previous games, but growing difficulties during development and maintenance led the company to explore alternatives for their next game.

The first difficulty is that C++ is complex, even for experts, and dealing with that complexity was taking up an inordinate amount of developers' time. Chucklefish's developers had deep experience with modern C++ best practices. "We were 100% all-in on modern C++," says Catherine West, Technical Lead on Witchbrook and Starbound. But using the most modern features wasn't enough to tame the complexity of C++. West elaborates, "We triggered undefined behavior all the time despite knowing quite a lot about how not to trigger undefined behavior. We regularly encountered crashes due to iterator invalidation, expired references, uninitialized values, destructor order, data races, and exception unsafety."

What took up even more time than crashes were intermittent problems and unexpected behavior, as debugging these required tracing through a large code base. Often, after several hours of work, the root cause of these problems would be identified as unintentional undefined behavior. The amount of time spent on these issues had become unacceptable.

> **"I was losing performance because the easy and safe thing to do was *the slow thing*." - Catherine West, Lead Programmer**

While the undefined behavior manifested in catastrophic ways, it usually only affected a small subset of users. The second and biggest motivator for Chucklefish to switch away from C++ was a need to write correct, fast concurrent, and parallel code in order to improve performance for everyone. Modern consoles have less powerful CPU cores but have multiple cores to make up the difference. Previous Chucklefish games had performance issues because they were written as a single core program first. Once the performance problems came to light, it was extremely difficult to rewrite those programs so they would scale correctly across multiple processors. West recalls trying to fix structural performance problems while maintaining memory safety: "I was losing performance because I was invoking copy constructors needlessly. I was losing performance due to allocation, and I was losing performance because the easy and safe thing to do was *the slow thing*." Chucklefish needed their next game to use multiple cores, and do so safely, to compete on consoles.

How Rust Solves Complexity and Enables Safe Concurrent Code

The Rust type system brought immediate relief from the pain of dealing with complexity in C++. Compared to using C++ with the type system and standard library APIs that have been added on to C++ over the years, the Chucklefish developers found that the types and APIs in Rust felt natural and required much less ceremony and maintenance of custom wrapper code. Knowing that safe Rust has no undefined behavior reduced fear during development. Using Rust meant no more long debugging sessions, in which multiple team members spent hours trying to figure out what caused an intermittent problem.

Rust made it possible for the team at Chucklefish to create a game that takes advantage of multiple cores without introducing crashes. West relates one experience of refactoring Witchbrook's engine to run systems in parallel with the help of Rust's compiler: "I didn't do absolutely everything correctly; there were some logic bugs with the systems that I didn't order correctly. I still had to write some extra code to ensure that component- and resource -locking would not deadlock with other systems without having to pay close attention to the lock order, and this was to be expected. But, with those caveats out of the way, it more or less *worked on the first try*."

West now strongly prefers writing parallel and concurrent code in Rust. "Rust is one of the few languages that really gives you a large amount of confidence that your parallel and concurrent code is anywhere near correct." This confidence translates to a fast, robust game. Developers can spend their time getting the game logic correct, rather than debugging subtle data races. If the Chucklefish team had had Rust's safety guarantees available earlier, it would have been possible to make Starbound fast on consoles with less development effort than they're currently expending on porting.

Additional Rust Benefits

By switching to Rust, Chucklefish solved the complexity and concurrency problems they were having with C++. They also saw some additional, unexpected benefits in using the new language. In the past, the Chucklefish developers were hesitant to add third-party dependencies to their C++ codebase, due to the tooling effort required and cross-platform concerns for any code introduced. Rust's package manager and build tool, Cargo, made adding a dependency to Chucklefish's build a breeze, even on the wide variety of operating systems their games run on. Adding a pure Rust library as a dependency was trivial to do, even on consoles. In the past, adding a single library had taken days of work to successfully integrate on all supported platforms.

Rust unexpectedly eliminated another developer timesink for Chucklefish: dealing with cross-platform *differences*. "Having multiple competing compilers is supposed to be a benefit of C++, but most of the time when we would write code, it would break on whatever architecture we weren't actually developing on," West said. "We had Continuous Integration that would build on the three desktop architectures, and, more than weekly, we would write code that broke on one or more of them." Even after taking into account the ten days needed to customize Rust for the Xbox, PS4, and Nintendo Switch consoles, Chucklefish saved time previously spent debugging cross-platform idiosyncrasies.

Now that the team has familiarity with Rust, they're looking to leverage Rust's advantages in other areas as well. The system to match players for their Wargroove game needed to be scalable, as well as secure, reliable, and fast. Rather than bring in a web-focused language like Node.js, Chucklefish chose to use Rust for this project and found it to be a good fit. The new webservice is using high quality open source libraries that are available and help make building a scalable web service in Rust straightforward.

Rust: A Solid Choice

Overall, Chucklefish is pleased with their choice to write their next game in Rust, even though the project is not yet complete. Throughout the development process, the company has seen performance improvements. They've also experienced the benefits of Rust's safety features, which have resulted in much less time spent on development and debugging tasks. West concludes, "It's not only important that it be possible to do a good design in a given language, but that the language actively encourage it by making the bad design painful. Rust does a fantastic job of this."