Interviewer
My first question is, what have been your motivations for learning Rust and choosing to use it?

Participant
I guess, I don't know, it's a few things for me. One is like my kind of earliest background in programming stuff came from like a security perspective. So I did a lot of reverse engineering and disassembly, kind of taking stuff apart. Early work was in like binary exploitation stuff. I feel like that gives you a very good appreciation for the importance of memory safety. So I think very early on, like that promise alone made me interested in looking at Rust. I think I first really started paying attention a little bit more closely, maybe five years ago or so, determined it was kind of getting there, but not quite ready. And about three years or so ago, I had been kind of playing with it in the background just to keep my eye on it. And a friend of mine had a project that seemed like a particularly good fit, which is a game that he's been running for a really long time. It's an ancient C, C++ code base. And so it was kind of a neat opportunity to slot Rust in, make it a nice three binary or three language binary and go from there. So I really started playing with it for that purpose and then kind of spent a couple of years working on that with him in addition to some other projects.

18:1 Early... Memory Safety

18:2 And a friend o... Game Development

Interviewer
Which other projects?

Participant
Just kind of my own personal tooling. I use some Rust tooling just to manage databases of personal information, bookmarks, a little link shortener. Just kind of toy web backend services that I can use for my stuff just to keep the muscles strong.

18:3 Just... Web Application Devel

Interviewer
Yeah. Nice. Nice. So it's like a web backend implemented in Rust and then are you compiling to WebAssembly or something like that or is it just the backend is separate implemented in Rust and then you're accessing it through a typical web interface then?

Participant
Almost entirely through CLI tools.

Interviewer
Oh, gotcha.

Participant
Big on CLI, but served over HTTP. And then just a couple maybe static pages being generated. I store all my images in B2, it's like an S3 compatible object storage. I just have a hundred line Rust program that

18:4 B... Personal Tools & Toy P

loops through and renders a template and small static site gen type stuff.

Personal Tools & Toy P

Interviewer
Gotcha. And then this game, tell me more about the way the code bases are interacting. Which part is Rust handling? Which part is the C, C++ handling?

Participant
Yeah. So the game is a MUD, I don't know if you're familiar with these. So MUD is multi-user dungeon. They're old telnet text-based MMORPGs effectively. If you've played World of Warcraft or something, you can really think of it as having all the same mechanics. You're walking around in the world, you've got gear, you're fighting mobs, you've got quests with a bunch of stuff to go on, different regions you can travel to. Really just think World of Warcraft mechanics, but text-based and scaled down. But so single server where a bunch of people are all online and interacting, persistent online game world. And so I guess I got involved with this project for a few reasons. They're starting to have some performance issues, which was a little bit confusing for them because this game has been going for over 20 years and hardware has gotten much, much faster and they've still managed to end up having performance problems. So I think we set out with the goal of fixing a few performance problems and getting a bunch of the core data and data structures into a more usable format. So I think the first big thing that we did in Rust was basically move the definitions of all the core games data structures from the C headers into Rust headers. There's very little C++, so we can kind of ignore the C++ aspect, but moved those, did bindgen first, moved those from C headers into Rust headers, wrapped them all with some procedural macros and basically kept the ABI level stability. So the in-memory representation is the same, but we made those accessible to Rust and put Serde serializers and deserializers on top of them and deleted all the custom, like they had a custom .dat format that was like writing out keys and parsing values and just like every single data type had like a 300 line parser for a custom file format kind of thing. And we just wiped all of that out and just used Serde to dump it to JSON in and out kind of stuff. And so once all the kind of stuff was loaded and unloaded that way, we were able to start doing a bunch of kind of really just putting a little bit of a declarative language around some of the common data structures the game might use. So really the game almost entirely used collections of stuff as just silly linked lists because C is complicated to do any other data structures in. So like if they had a list of entities that each had an ID on them and you needed to do a lookup by ID, they would every single time traverse the linked list and check each one for ID and go to the next one and that sort of stuff. So we identified a few common patterns for those things and kind of made it so that since all of the structure definitions are inside a procedural macro, we've got some compile time reflection type stuff. And so you could just annotate and say this field is basically a unique index on this struct and we generate all the stuff to keep a static hash map in memory and when you allocate a new one, it gets in there. When you save, we update the hash map kind of thing. And so really the thing that's in Rust is kind of those core data structure

18:5 So the game is a MU...
Game Development

18:6 They're starti...
Rust is ergonomic
Rust Performs Well

18:7 So I think t...
bindgen
Macros

definitions and then a bunch of kind of code gen on top of them to add new indices and stuff.

Interviewer
Gotcha. Gotcha. So then where do you use unsafe Rust in the projects that you mentioned?

19 **Participant**
Well so all of the data structures kind of retain their CABI compatibility. So that means that amongst other things like if you've got the character's data type, the character is in a faction. And so they have the char data type has a pointer to the faction that they're in. And the serialization, deserialization previously would just kind of populate that pointer at runtime and so we auto-gen-ed repopulating that pointer when everything is loaded, like did a nice topological sort to actually just not have to explicitly encode the dependencies but allow that to be auto-determined. But so all of these things are used and referenced all over the place in the C code base, right? So the Rust side is allocating them. But then so anytime you want to use them from Rust, like if you want to have serde serialize and deserialize, you need a reference. You only have a raw pointer. So there's going to be unsafe in there somewhere. And so the tricky part of this whole thing has been coordinated like when Rust code is running, like when is it safe to get a reference to this thing based on the rest of the code base that's in C knowing that it could be doing anything at any time.

18:... Allocation in Rust

18:8 And so... Difficult FFI

24

25 **Interviewer**
Gotcha. So then is all allocation happening in Rust and being exposed to C or is there a mix of both would you say?

26

27 **Participant**
It's 99% in Rust at this point. There's maybe a couple random allocs in the C code floating around somewhere, but all of the core data structures were moved to Rust and all of their allocators are based on Rust now.

18:50... Allocation in Rust

28

29 **Interviewer**
Gotcha. So if you're maintaining this ABI compatibility and you're passing pointers across the FFI boundary, how like have you had issues with C using or copying pointers in a way that breaks Rust's like aliasing semantics for borrows?

30

31 **Participant**
As far as I can tell, I have not had like aliasing based undefined behavior. But like I fully believe that there exists some somewhere in this code base and it's been impossible to, well, like luckily I haven't had any kind of heisenbugs that point to the kind of pointer aliasing or anything like that. But it's huge, like it's, you know, there's 30 plus gigs of content in memory. So it's like, and mostly small objects and strings, like there's tons of stuff I did to pull that down, like the string interner and stuff. So like

18:9 As far as I can... Unsafe is Diffi...to Und

there's just a lot of stuff there. And if we break the aliasing guarantees and like, you know, had a torn write or something, it would probably take me quite a while to find it if it did happen. But I've seen no indication of it so far.

18:9 A... Unsafe is Diffi...to Und

Interviewer
Gotcha, gotcha. Okay. So then just to sort of summarize, you have your core logic for the game in C, but you were facing performance issues. So you chose to rewrite the data structure handling and allocation and like serialization and deserialization in Rust. And then that gives you this situation where you have Rust's sort of memory model surrounding the, like it being essentially like more than 90% of the memory that is allocated for this, but then it's just exposed directly to C for when it's actually used for the game. Is that an accurate summary?

Participant
Yep.

Interviewer
Gotcha. Okay. Were there any other uses of unsafe in some of the other projects that you mentioned or is this mostly for the game?

Participant
I think the game is probably where I've used it the most. I've got like one microcontroller that has some unsafe Rust in it. And I think maybe, I think there's maybe one pattern that I commonly use unsafe for or previously did before OnceCell was stabilized. Now that OnceCell is stabilized, I feel less inclined to use that pattern all over the place. But I think in my other projects, the main place outside the microcontroller I think is really just doing the, like, I'm going to initialize something one time at program startup, leak it and put it into like static memory and just like you can assume for the rest of the lifetime of the program that it's going to be there without actually doing a check, which that extra check you can't dodge with like lazy static or whatever else. They're always doing the check to see like, have I initialized this to be safe? And so like occasionally I do the thing where you just like put the static and leak it and unsafe it into a static mute and then access it the rest of the time. But you know, it's like on one hand the canonical example of unsafe in the book, it's like here's an example of when you need to use unsafe and like it shows exactly that. So like I have high confidence that that pattern is right, but like it feels weird to use unsafe because it's like we do have OnceCell and like unless you can prove that you're wrecking performance with it, you probably shouldn't.

18:... Operating & E...dded S
18:14 I'm going to initi... Increase Performance
Static Variables
18:13 So I... Preference for Safety

Interviewer
Gotcha. So would you generally, like if you had the development time and effort, would you go through and replace these with OnceCell? Or do you feel like you'll be fine leaving it as is?

43 **Participant**
I mean, for my personal stuff, it's like there's very little risk. So I'm kind of fine leaving it. But often if I'm touching a code base anyways and I come across stuff, I'm usually inclined to remove it. Pardon me. I just need to plug my laptop in really quick.

18:... Audience-specific Rea

44

45 **Interviewer**
I know you're, you're totally good. Okay. So then in the way that unsafe is used within this game engine, I guess with unsafe in like just an arbitrary Rust code base, you usually have your sort of code that's unsafe and then you have this like safe interface surrounding it. Does that generally happen in this application with the game or is it somewhat different because it's like you're just exposing things unsafely and to see where the main logic is going on?

46

47 **Participant**
No, I think it's a lot more of the former. You know, I think like, so the longer term goal of this project is really to be able to start writing some more of the game logic in Rust and to actually kind of natively use some Rust data structures instead of kind of like, you know, we talked about the hash map. That's kind of hacked on as like a, we keep a global hash map and expose some like accessor functions to the C world, but like actually storing everything in a hash map might like be a useful thing or like having those, you know, having Rust data types kind of associated with the ones that are, have to retain their CABI compatibility. So really for each of these types, there's kind of a layer that is meant to hide the unsafe from everyone else because I think, you know, this game, the team working on it is full of not programmers. It's mostly folks who were interested in the game 15 years ago and have kind of been involved. So a lot of the C is kind of copy pasted and cargo colted. There's not really, there's no sense of like programming fundamentals there. So like, I didn't want anyone else to ever have to touch unsafe Rust. I figured like, I'll take that, put a safe barrier on it. And so on the Rust side, like there's kind of a layer that exposes standard guard objects. Like you get from a Mutex or RwLock that contains all of the logic to protect those accesses versus the rest of the C code kind of thing.

18:16 So really for eac... Exposing a Safe API

18:17 An... Exposing a Safe API
Mutex<T>
RwLock<T>

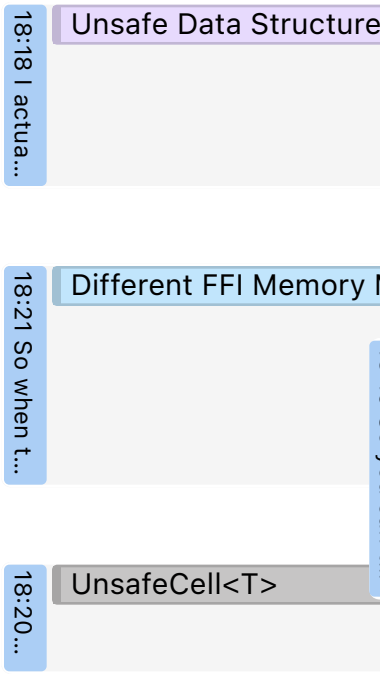
48

49 **Interviewer**
Gotcha. Okay. Okay. So from, from the, if I'm operating from the Rust perspective and developing on this application, unless you're changing one of these core, like guard objects to access some fundamental data structure, you generally be using just like safe functions that would coordinate through the F1/19/24, 11:10 AMFI to C, would that be correct?

50

51 **Participant**
Yeah. And, and this is, I guess there's been a little bit of like philosophically, what's the right way to do this. Cause like, you know, there's a bunch of functions that are called into from C. And so they're, they're giving a raw pointer, like here's a raw pointer to the character. And obviously we don't know if that raw pointer is valid. And so like, do we, do

we put the unsafe there or not for like the user into like getting the guard object? And what I settled on was kind of a little bit of maybe a nasty hack, but actually it just kept a, a, because I control the allocators and deallocators, I actually just keep a set of the addresses that are valid and have been allocated and associate those with like the type of the object. And so internally the, the kind of methods that get you a guard object are able to check, like, is this pointer actually valid? By saying like, is it one that we allocated and do the types match? And then we keep kind of a per, well, it's a bit of a hierarchical locking scheme, but we kind of keep, you can either obtain a guard object through, we do a bit of like a global lock for the C code base. So when the main C engine's code is running, it obtains like a global lock. And that's like the hint that like the C could be holding any of the pointers and doing stuff right now. So you can't alias. And if that lock is not held, then we kind of fall back to like RwLocks or Mutexes that, that allow you granular locking to the objects on the Rust side. So that the Rust stuff can actually do multi-threaded parallel stuff by like doing granular locking and the C can just not worry about locking at every individual object, it just locks the global thing and blocks all the Rust for that time. Unless you're kind of in the stack of the C main program, in which case you're allowed to kind of do whatever, I just expose a guard with UnsafeCells so that you're allowed to alias.



Interviewer
Okay. And then is there a reason why, like, architecturally that you chose to have that global lock on the C side? Like, is it like motivated by the functionality of the game or like hypothetically, let's say you were like a multi-threaded programming wizard and every single like implementation of every single thread thing was perfect. Would you also have granular locking in the C side?

Participant
I think just because like the bulk of the logic was in C and it was really easy to like put the global lock around like the core, like this is actually the game servers tick and we'll just lock around there, that it really just made sense to do that. Like I would probably fall back to pretty granular locking, though like even in this project I'd say that the granular locking has already proved to be a little bit of a performance bear on the Rust side of things. And so like I've started peeling back some of the granular locking and doing like maybe a per data type lock where possible. So you lock all of the characters at the same time, since usually you're kind of iterating over the characters as a group kind of thing. That makes like getting the mutable references in Rust a little bit harder, but so far we've kind of, the very little bit of game logic that I've implemented in the Rust side has mostly been limited to like let's actually read a bunch of data and then like expose it to the player. So very little mutation happening on the Rust side which makes things a little bit easier because I can actually just you know have references to alias since we never have a mutable one.

Interviewer
Gotcha, gotcha. Okay, yeah. And then, oh what was my question, tip of

my tongue. Oh yeah, just quick clarification. Is the game started from Rust or is it started from C?

Participant
That flipped relatively recently. I think main is now in Rust where it was previously in C. I think I moved the main in Rust because we have like there's a tokio runtime doing some stuff in there like the game now connects to Discord and so like the in-game chat is mirrored to Discord and so like we've got a tokio runtime on the side with some channels that we use to like throw stuff back and forth. So I just moved, like I took the C main, oh well also like I redid the config system so like put the config in Rust instead of whatever was the nasty C like parsing of get up stuff. So like through clap and whatever at that and then toml config. So main is definitely in Rust now and just calls out to C at some point, but it's definitely been both ways over the lifetime of the project.

18:22 That flipped rel...
Rewrite it in Rust

Interviewer
Do you feel like there were any challenges that are sort of unique to having it be one way or the other?

Participant
Not particularly, it seemed about the same like I don't think I had to update the linkers or anything like it was it just ended up being a convenient pragmatic switch. I guess the, I remember now the biggest challenge to like the number one thing that made me actually move the main to Rust was setting up some unit tests against some of the C code, but they didn't have a testing framework in C so I actually just used like the standard Rust testing framework which needs to be able to control main because the test framework compiles a bunch of individual binaries where like main is the body of the test and so if you've got main in C then your link fails because you've got two mains. So I think that that was really the big reason for moving it to Rust was just so that I could stop doing linker shenanigans to get testing working.

Interviewer
Gotcha, gotcha, okay. So now I guess a bit more specific in terms of different data structures that you use. You mentioned using Box, UnsafeCell, and Rc/Arc to contain raw pointers or to do like in cases where you'd be doing raw pointer use or like raw pointer conversion and in the survey. Could you go through each of those three and sort of talk about where they fit within the code base?

Participant
I guess maybe starting with with Box. Yeah I think Box is really just like at kind of the core of the allocator for all of these game objects like none of them ever live on the stack. They're all kind of heap allocated game objects so mostly those are actually just like Box new and immediately leak because we're just going to keep the raw pointers around and we just keep raw pointers as well as the like locks or whatever on top of

18:23 Yeah I thi...
Allocation in Rust
Box<T>

them. So really the allocator is basically Box new and leak and the the deallocation is just you know Box from and let the normal drop implementation handle it.

18:23... Allocation in Rust
Box<T>

Interviewer
Gotcha, gotcha. And then so like when you're doing this conversion do you need to record any additional information associated with pointers to make sure that the allocation and deallocation is happening safely? Like I remember you mentioning something about like associating particular pointers with type information.

Participant
Yeah so I mean I guess we have I have that information for other purposes so I use it in the deallocator and that's really just like when we allocate I basically I do the nasty thing and I cast the pointer to an integer and store it in a set of integers that represent the pointers that are allocated for that type and so on deallocate you just you know cast the pointer back make sure that it's still in that set and then you can you know know that you can safely do the Box from for for deallocating kind of thing you clean up from that set at that time.

18:24 when we allo... Box<T>
Pointer Arithmetic

Interviewer
Gotcha, so have you had any issues where like you're you've freed something by doing Box from and like the the type information wasn't correct or maybe it was just like not allocated with Box new so you got like an error or has that usually been fine when you when you do that process?

Participant
I guess well so I provide basically some slightly nicer error messages that are basically like if if that if I get a pointer to free and it doesn't pass the normal checks which is like it needs to be in this hash map I'll check the hash maps for the other types and say like you've you've screwed up on the C side somewhere because this is always from the C side and I could say you've like screwed up somewhere and you've like you're trying to free this type with a free function for this other type and like I throw an error message it's like I don't know how you did this but you did or like if you call free and the pointer is not in any of the sets I can just tell you this was a double free most likely and like you've got a problem somewhere in the C code that is causing this double free and so like as soon as I implemented that it took like I don't know two months or something of a bunch of folks playing whack-a-mole like actually just going through the C codebase and finding all of the weird places where they were like keeping dangling pointers around and stuff so like it we certainly found a lot of issues like right away

18:26 I gues... Runtime Assertion

18:25 I throw an error messa... Double Free
Temporal Memory Bug

Interviewer
Yeah, gotcha, and then are none of these, I guess because you, you can, it seems like you have this invariant that once you allocate one of these

things that pointer is going to stay the pointer to that object? Or are there cases where you have to reallocate something and then you need to change its pointer in the type table?

Participant
No, they're all kind of, all of the types are fixed in size, so there's never any sort of reallocation and they're all Boxed and Pinned so they never move on the heap. So we kind of, I just assume that the pointers are stable everywhere.

18:27 No,...
Box<T>
Contract or Invariant
Pin<T>

Interviewer
Uh-huh, gotcha, gotcha. So then for unsafe cell, I remember you mentioned that being used in the core, like stack of the C interpreter where you have like a guard and then you use unsafe cell so that you have like multiple interior mutability. Could you talk a bit more about that use case or just any other place where you're using any of the cell types?

Participant
Yeah, I think it's really, I think UnsafeCell is the only one that I really use. And that's really just used in the case that like, you have the kind of global, so in the case that you're running the C main game tick, and you've got the global lock and your C code has somewhere along the way called into Rust now, right? So up, if you walk up the stack somewhere, you're gonna find a C frame that represents the game and it's holding that lock, but you need a reference to something in Rust. That's where like, we kind of assume that, well, the C side may be holding a pointer to it too, so like you may be aliasing somewhere on the C side, but we just know that we do have this global lock and the C code is the only thing there. The C code itself is not threaded. And so like we use UnsafeCell to just give you a temporary like unsafe reference to those objects, only in the case that like the kind of, the C code has that global lock. And so it just implies that somewhere up the stack you're being called from C.

18:28 So up, if you wa...
UnsafeCell<T>

Interviewer
Gotcha, gotcha, okay. What motivated you to introduce the UnsafeCell in this case? Was there just like necessary to do the constraints of Rust's type system or was there a particular judgment that you made in implementing this where you thought, yeah, no, we need to have an UnsafeCell here for soundness?

Participant
I think it really came like, as I was in the very early stages of like figuring out the whole memory model for this thing, I think I had a conversation with someone there's an unofficial Rust community discord that's pretty popular. There's a channel in there called Dark Arts that lots of folks kind of sit around and talk about like the minutia of safety and soundness. And I think a conversation there really helped me understand that like you need that UnsafeCell in order to basically tell the compiler like, you must assume that this pointer can alias. And that's like something in that

18:29 I think I had...
Engaging with...st Com

conversation clicked which was just like unsafe cell means like you can dereference a pointer and it can alias. And like, that's the fundamental thing that's different about UnsafeCell. And so like, I think that's where that came from was it was just a very early conversation there and kind of understanding that like, oh yeah, like because these things alias all over the place in the C world, like we do.

Interviewer
Gotcha, gotcha, okay. Yeah, I think we actually recruited from that Discord. Yeah, no, that makes sense. So I guess more in terms of the FFI stuff now specific, oh wait, no, there's one last one, Rc/Arc. Yeah, where does Rc/Arc come up?

Participant
So Rc/Arc I use for kind of the very granular locking on the, for like the pure Rust side. So like if you don't have that global lock from the C side, you're in Rust. That means you could be doing multi-threaded stuff and everything. And so we basically keep a per object Arc, RWLock that lets you either get, that lets you get access to those things in a safe way. And so that's just like a per object. Actually, I cheat a little bit in the allocation. I allocate the space for the arc just in front of the object itself so that I don't have to do any sort of lookup. Like I take the pointer, I allocate arc and then allocate those things next to each other in memory. And so like the pointer that we get from C is always the start of the C object itself, but actually it's the interior of a struct that has the Rc/Arc in front. So I can just subtract off the pointer and grab the arc once I've made sure that the pointer is valid kind of thing.

18:30 So Rc/...
Arc<T>/Rc<T>
RwLock<T>

18:31 Actually, I cheat...
Easier or More Ergono
Pointer Arithmetic

Interviewer
Okay, gotcha. So wait, are you doing that allocation then? Like are you creating that allocation through Box? Or are you using like a custom allocator in the place?

Participant
No, still through Box. Like there's a struct that is, you know, rep or C with the Rc/Arc upfront and then, you know, is generic over all the game types.

Interviewer
Okay, gotcha. So you'd just be allocating this rep or C struct. And then when you expose that struct to C, you take a pointer past the arc, but then you know that if you have it as that particular type, you can do the offset and safely access the arc because it's still there at the front of the allocation.

Participant
Yeah, exactly.

101 **Interviewer**
Gotcha, are there any other like tricks that are similar like that with pointer arithmetic that you have to do in that code base?

102

103 **Participant**
I think the only other like weird pointer arithmetic kind of unsafe stuff I do is really like, I've got like a bit of a compile time reflection system built around these objects. So like, there's a, you know, for the admins of the game, they've got basically a way to peek and poke at objects through the game itself. And so like, you should be able to do like edit character number and like get a table of all the fields and like a printed version of what they are and say like, I wanna edit this character's name or whatever and do that stuff. And like, there's basically a hacky like V table of the like, these are the field names and their types and their offsets from the struct so that we can kind of generically have the parsing for all the types and just say like, you're writing to the spot kind of thing. Oh, sorry, there you go. Oh no, my gut says that there's a better way to do that. It was one of the very earliest things I wrote. So like, there's probably a better way that I just don't know about yet or didn't know about at the time and I haven't thought about it.

18:34 So like, there's a, y... — Pointer Arithmetic, Unsafe Data Structure

18:32 Oh... — No Other Choice

104

105 **Interviewer**
Gotcha. It seems like in general, that seems like a common architectural pattern that you've done in this code base is that whenever you have to do a raw pointer conversion, either like just recording the type of a raw pointer or recording the necessary offsets to access like fields, you have this sort of in-memory table that you do that is your source of truth for when something is safe. Is that a, would you say that perspective is accurate?

106

107 **Participant**
Yeah, I think so.

108

109 **Interviewer**
Gotcha.

110

111 **Participant**
It feels like given like having hundreds of thousands of lines of C that you can't reasonably audit, like it feels like keeping that runtime information around somewhere is the only way to like even start to think about this thing being sound.

18:33 It f... — C++ is Difficult

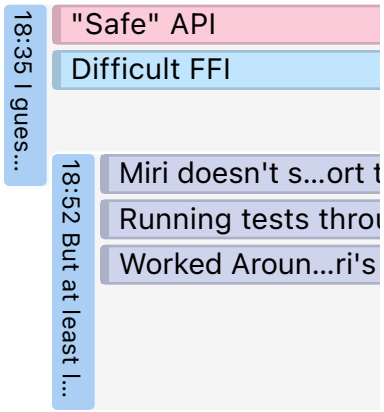
112

113 **Interviewer**
Gotcha, gotcha. Yeah, I guess how, what, so like in general, could you summarize your confidence level about soundness using these techniques?

114

115 **Participant**

I guess I feel reasonably confident that in isolation, the abstractions are mostly sound. I'm not at all confident that they're actually sound in practice with how they're used because the C code base is so freaking large that like it's doing something weird somewhere that probably breaks my guarantees. But at least like a few of the kind of primitives where I could like, so for the kind of global lock piece of things, I kind of prototyped that outside the game and ran it through Miri and made sure that like it looked okay with Miri, but obviously like Miri doesn't do FFI boundaries or anything, so like it kind of, I can't actually do it in the broader scope of the full game or anything, but at least tried to like take a piece of it and prove soundness that way.



Interviewer
Okay, gotcha. Have you used Miri, like are there any other areas where you've used Miri and like, have you had any particular errors that you found with Miri when you've done that sort of like factoring out parts of the code base?

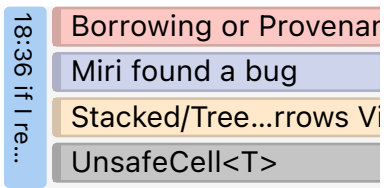
Participant
I think that's really the only place that I've personally run Miri. I've done like, I'm just interested in this stuff, so like there's a project somewhere out there that just runs Miri on a bunch of the like the 10,000 most popular crates and they like post soundness issues and say like, hey, you might want to go and like contribute fixes for these soundness issues upstream, so I kind of follow that and occasionally go like figure out why people's stuff is breaking just so that I feel like I have a better model, but I haven't found much of a case to use it myself.

Interviewer
Gotcha, and when you did that experiment factoring out that part, did Miri give you any issues for it or was it all good?

Participant
Sorry, could you repeat that really quick?

Interviewer
Oh, sorry, you mentioned there is part of your code base that you factored out and ran through Miri. Did Miri find any issues with that or was it all good?

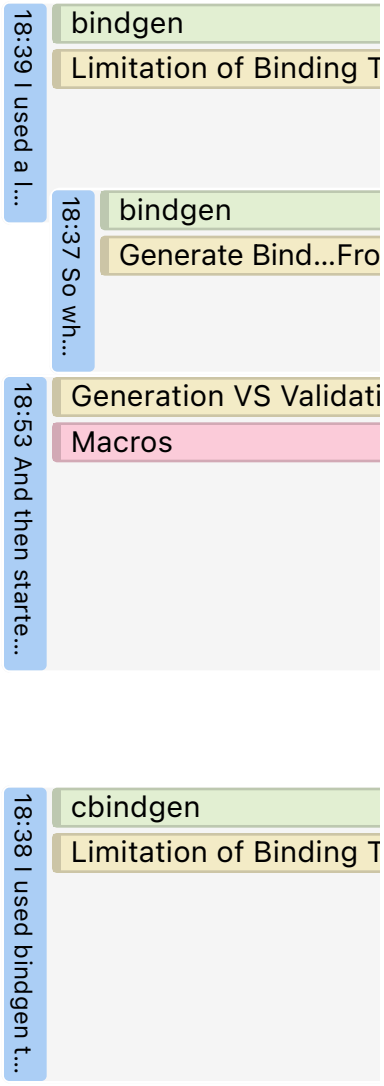
Participant
I think I found one or two issues along the way. It, I don't know, it's been a few years now, but it probably, if I remember correctly, actually it may have been finding an issue with Miri and then going back to the Dark Arts channel and saying why the hell is this not working that led me to realizing I needed that extra little UnsafeCell versus just kind of grabbing references from the raw pointers just to prove the aliasing was okay.



129 **Interviewer**
Gotcha, gotcha, so then with some of the FFI stuff that you've been doing, you mentioned in the survey that you've used, like you've manually written bindings, but you've also used bindgen and cbindgen and PyO3. Could you talk a bit about your experience in writing bindings to other languages, either just manually or using those tools, like what worked great about the tools, what was challenging, what were the problems that you had in sort of getting those two code bases to link up and connect together?

130

131 **Participant**
Yeah, so for this project in particular, I used a lot of, or tried to use bindgen very early to just start exposing like bits of the C engine to Rust. And, you know, bindgen seems to work great if you've got really, really simple headers doing no sort of weird macro trickery and just like very common stuff. And that's to say, it didn't work at all in this code base, really. So what I ended up doing, I kind of spent a bunch of time very early actually refactoring some of the headers and just pulling just functions and just a subset of the data types that I wanted to touch in Rust into like a separate set of shared headers and running bindgen on just those. And like sharing those headers with the Rust and C-side. But that was, so that worked okay for a little while. And then started, as I started getting into like the actual core game data structures, it started falling apart, which is why we ended up like, I left the bindgen stuff there, stopped doing the core game data structures via bindgen and actually just hand rewrote all of them in Rust. And I did that for the added benefit of being able to put a proc macro around them and do a bunch of the code gen. But like, if I could have just bindgen them all and then run a RegEx and put the thing in front that way, it would have been nicer. Luckily, like for this one via my friend, he had an army of the kind of junior programmers. So like I did like three or four of the core game data structures and said like, this is what it looks like to move the definitions. And I just let them spend a week like manually retyping everything. I used bindgen to expose all the Rust stuff back to the C-side or sorry, cbindgen rather. To expose that way. And cbindgen almost entirely works. I had two big problems and complaints with it. One is that it requires nightly and we were otherwise able to be on stable. And so it like caused a bunch of tooling thrash to have to hit nightly there. And the other issue is actually that cbindgen doesn't seem to be capable of handling like a loop of dependent structure definitions. So like the character contains this, contains this and like they've got pointers that form a circle in their definitions of those data structures. And so like, I actually, I have a regex that runs that pulls that makes a typedef for the structs and moves those all up to the front just on the C-BindGen output. And it's like gross and nasty, but it works really well.



132

133 **Interviewer**
Gotcha, okay, okay. Wait, so you have like a cyclic structure going on there. That sounds like it might be something tricky to get working with Rust where you really can't make cyclic things with safe references. I guess it's not, it's got a chain of like pointers that are cyclic, right?

Participant
So like a character may be inside a faction and a faction has memberships and a membership points to the character that is the member. And so like technically those structure definitions from like a C-Sense all depend on one another because it says I've got a pointer to this other type in it. But they're not quite like the circular references as you might think of in Rust.

18:40 So like a cha...
Multiple / cyclic aliasin

Interviewer
Gotcha, gotcha, okay. And then with Python and the PyO3 bindings, is there like a Python component to the game engine or is that something different?

Participant
No, that's been for my own playing around and edification. So like I wrote some bindings to a couple Rust libraries myself in PyO3 and then as my day job is doing like big data…And so like as all of the researchers, the computational scientists, biologists and stuff like they're all Python, like everything is Python. I kind of hate Python just by virtue of having spent too much time writing it in the last 10 years probably. But the ecosystem there has just been like so many of the core libraries have been being rewritten in Rust that I've been keeping up quite a lot. So like pydantic is now Rust under the scene, pullers is the data frame library we've started to use all over the place has Rust under the covers. Like there's enough packages that are like Rust with Python packages are now just Rust underneath with that light barrier that it's been useful to keep on top of it.

...
For Fun

18:42 I kind of hate Pyth...
Rewrite it in Rust

Interviewer
Gotcha, gotcha, okay. So I guess I have some broad questions now just about all of your experiences with unsafe Rust and then also in particular the FFI stuff. So with your C to Rust FFI usage, and you've already discussed, I think, some of these types of issues before, like you're sort of working with Rust where you have this very strict set of rules about memory and referencing, and then you also are exposing these structs to see where you have like a completely different and very like much more loose set of restrictions on what you can do with memory. Have there been any problems that have arisen for you just because of those differences in the memory models that have been particularly challenging when you're working with this type of like mixed language code base?

Participant
Yeah, I think like, I guess the biggest challenge has really been that like all of those core game data structures we talked about kind of needed to retain their like C compatible in memory representation, and oftentimes that means that like, so, you know, if we're refactoring like the factions and you've got the membership list, that's really like maybe a hash map of the member name to like the membership info, you can't put that in the actual faction because it doesn't mesh with like the C ABI there. And so

we've kind of had to like bifurcate the logic a little bit and say like, you know, all of the old stuff, all of the stuff that existed before we started this rough stuff is like C ABI compatible. And then we kind of, I guess using that same trick that we talked about with the arc, we're like actually this structure that we allocate is the C ABI structure. We've got the Rc/Arc up front and then we do an additional struct that is a Rust, like a pure Rust struct that doesn't have the C, the rep or C requirements on it. And we attach that to like the end. And so you can kind of from the like, you can now write a function in Rust that takes the C raw pointer to like the character object and then gets you some pure Rust data types and does some operations there. And so like that's kind of, I guess the bifurcation is really that like, there's a bunch of stuff that's like C in memory representation compatible and all of that stuff you can kind of access from either C or from Rust. And then there's a bunch of stuff that's only Rust compatible. And the only way to actually access or do anything with it is to just like call across the FFI boundary to Rust and deal with it. And so like that causes a little bit of confusion for some of the newer folks, because they don't realize at first that actually like this character struct has like this Rust extension struct packed onto the end of it. And they just look and see the C version and look at all the fields available and they're like, I don't see this field that I thought was here. And it's like, well, you can't hit it from C, you kind of have to go to the other language if you want to touch that data kind of thing.

18:43 And so like that's kind of, I gues...

Different FFI Memory I

Interviewer
Yeah, gotcha. And have there been any particular like bugs with your unsafe code that you found unique or interesting?

Participant
I think most of the biggest bugs came kind of as I was finishing the like safety and guardrails, right? So like the first versions of the deallocators didn't have the kind of table to check and make sure that like this pointer is valid. And that caused kind of interesting, difficult to track down crashes because you crash in Rust and then it unwinds and you've got the like FFI boundary stuff. So like those crashes, and I think in the very early days, I didn't realize actually that you had to kind of prevent unwinding across the FFI boundary. And so like the like first two weeks I had a bunch of these really hard to track down crashes that like, man, I can't tell at all what's going on here. And it turned out it was cause I like unwinding up the C stack. So I was getting a bunch of nonsense or like crashing in the process trying to unwind the C stack kind of thing.

18:44 And t...

FFI & Binding Bug

Unwinding Acr...FI Bou

143 **Interviewer**
Gotcha, gotcha. Okay, interesting. And then are there any problems that you face with this code base that you feel like your current development tools can't solve or that there just aren't tools that can solve and that you'd want to exist?

150

151 **Participant**
I mean, I think being able to prove soundness across the FFI boundary

Wants Solution

would be really nice. Like I haven't tracked, but I think Miri is making progress towards doing that in the last year or so, but I haven't tracked it too much lately. But I think that would be one of the largest one is just like a version of Miri that could go across that FFI boundary. I also, I did try briefly at one point to like do like UBSan and ASan and like the kind of LLVM sanitizers. Like doing those in a mixed C, C++ Rust program turned out to just be too much of a pain for me to bother with. So like any sort of like tooling to make running those sanitizers in multi-language binaries a little bit easier would be amazing.

18:45 I m... Wants Solution

18:46 I also, I di... Laziness Fights To... Wants Solution

Interviewer
Gotcha, was that like false positive errors type stuff or was it just getting to run it all?

Participant
Just getting it to run it all. It's very finicky.

Interviewer
Yeah, that's been my experience too, actually, yeah. So I guess that's all the questions that I have. I think you've covered everything that I could think of. Really appreciate your time. It's been awesome talking to you. I guess, do you have any questions for me?

Participant
I'm just really curious in like, I don't know, your own words really quickly. Like what is your goal of this project as a whole?

Interviewer
Sure, so I'm finishing up the second year of my PhD program and what I'm interested in is formal methods for verifying programs in particular in Rust that deal with things like the Rust memory model. And right now I'm trying to figure out what the problems are with unsafe code, like to figure out what people need to prove essentially. So with this interview study, there are sort of two components. There's one more quantitative component where I'm doing some analysis of code base, but with this part, it's like I'm just talking to as many developers who work with unsafe regularly as I can to get a sense of the issues they face so that when I do work on more like formal methods projects, I make sure that I'm like solving actual problems that people have. So yeah, that's sort of the spiel.

Participant
I guess, just out of curiosity, because like one of the things that's been, I guess, difficult for me to ascertain or really like keep a pulse on is like, there's kind of all of these efforts going on in the ecosystem to further formalize the memory model of it so there's like the efforts of doing stacked borrows that give us a better bar checker and a more formal model and stuff there. And like, I feel, I feel reasonably confident after a lot of testing and, you know, a few mere things like I think that my code is sound under the current Rust memory model as it exists today, but my gut

18:47 there's kind... Shifting Ground

says that as soon as like if stacked borrows became the new memory model, like it probably blows up everything. So like, how are you thinking about like formalizing against what exists today versus what's coming versus, you know, we don't know if stacked borrows is actually gonna work out?

18:... Shifting Ground

Interviewer
That's a tricky thing. Because like even now stacked borrows is like, now we have tree-borrows, right? And then not only is there like one version of the borrow checker, there's also like the new like non-lexical lifetimes version 2.0 that's coming with Polonius.

Participant
Yeah, the Polonius work is fascinating for sure. I think that the community doesn't really like, I think from a formal perspective, what I can do best is abstract away some of the details about like what a lifetime is in particular, assuming that the compiler will have some like version of it for me. I think it is a challenge though. It's like, that's where sort of the, I think the most like the biggest rough edge of the Rust ecosystem is in terms of like just where all the new stuff is happening.

18:48... Shifting Ground

Interviewer
You're totally right. It's like there is not a single like memory model for Rust, it seems like. So yeah, and I think what the memory model is for Rust and how do we effectively prove that is, I think one of the themes of the stuff I'm working on too. But yeah, no, your perspective is my perspective on it and in terms of everything shifting, but yeah.

Participant
Cool, well, great to hear that you're aware of and on top of it and I wish you luck. Sounds like a fun project.