Features of Rust programming language for building drone network models

Anton Tyshchenko

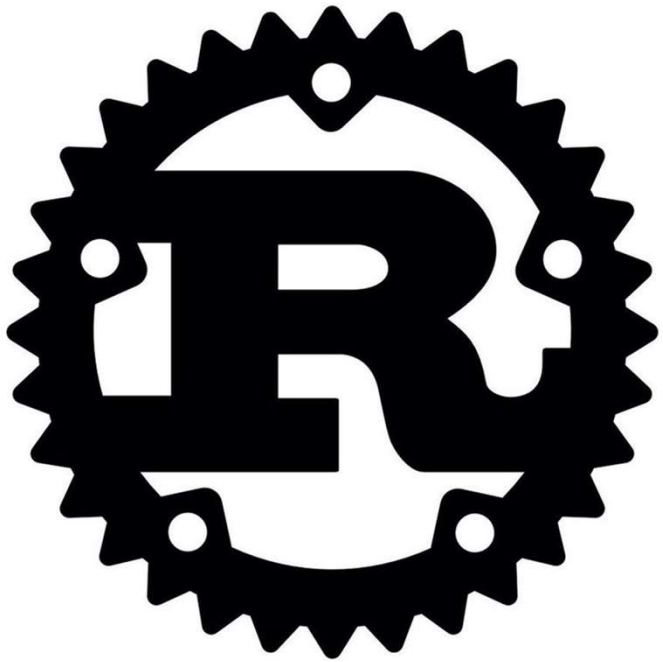
Plan

1. Rust
2. Program design
3. Code analysis
4. Conclusion

Rust. What?

A programming language that is

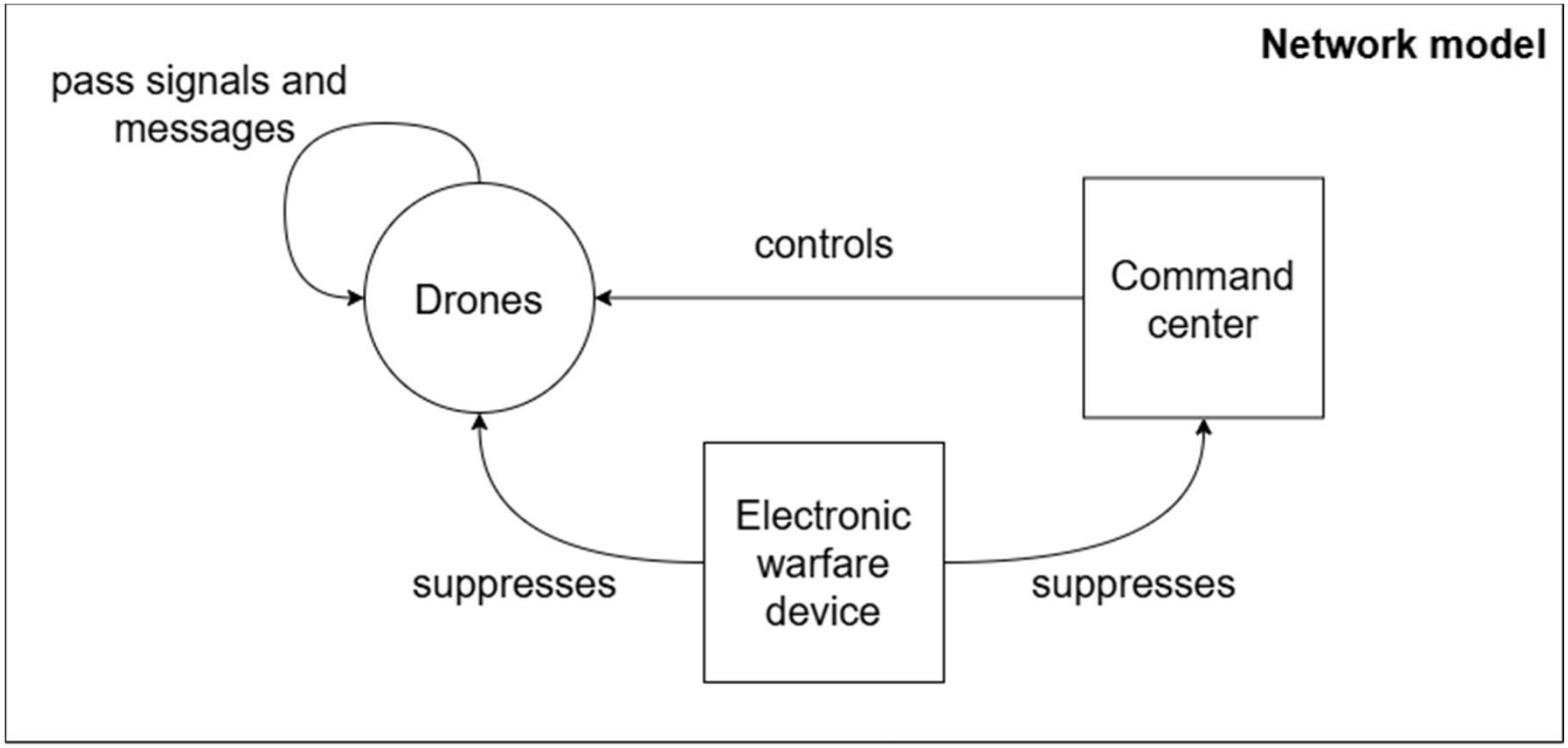
- **performant**
- **reliable**
- **productive**



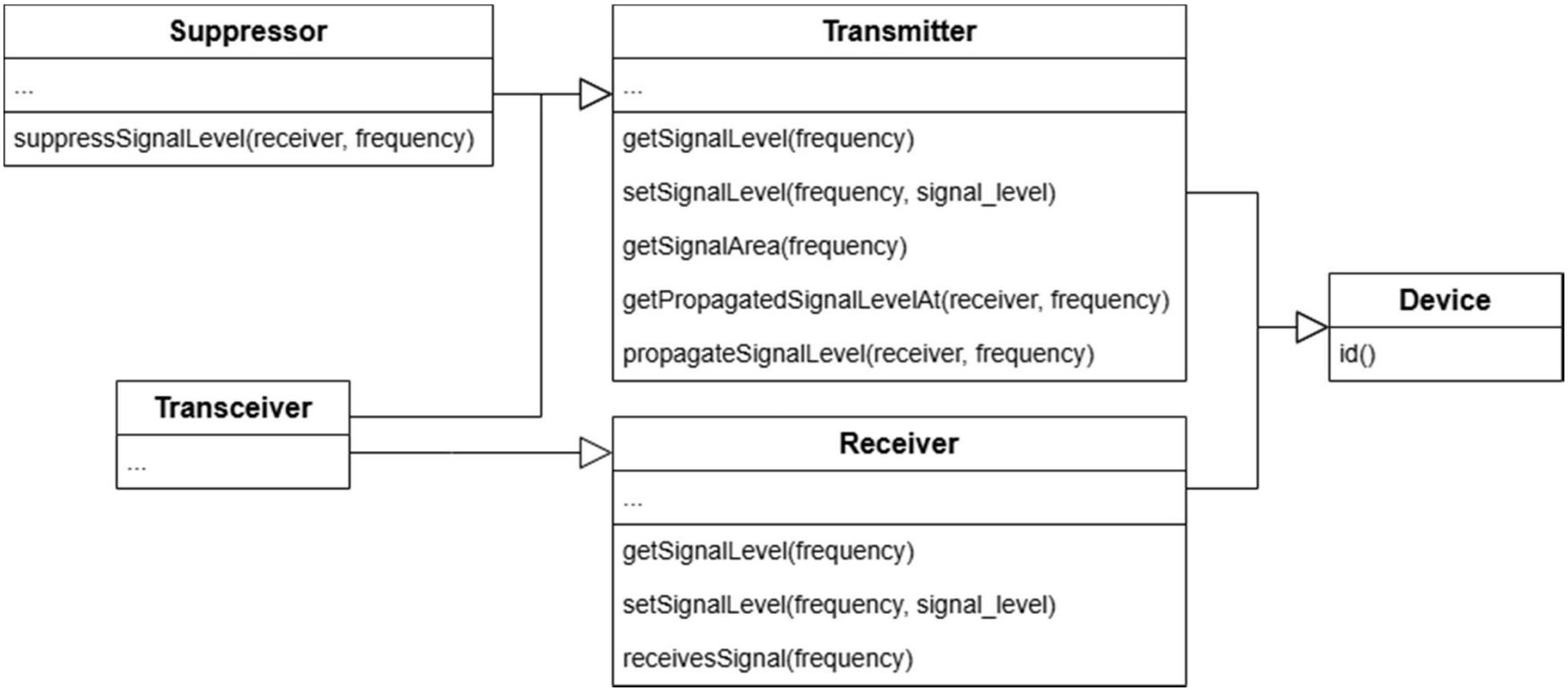
Rust. Why?

- Building a complex, safe and fast system
  - Rust is a <u>systems</u> programming language
- User- and developer-friendly CLI
  - *clap* crate
- Support of various graph operations
  - *petgraph* and *rustworkx-core* crates
- Visualizing results
  - Bevy, *plotters* crate, OpenGL, Vulkan…
- Easy package management
  - *cargo*

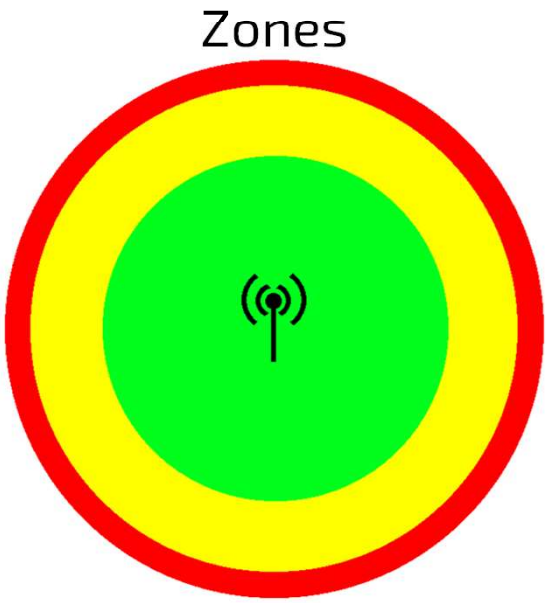
Program design. Model behavior

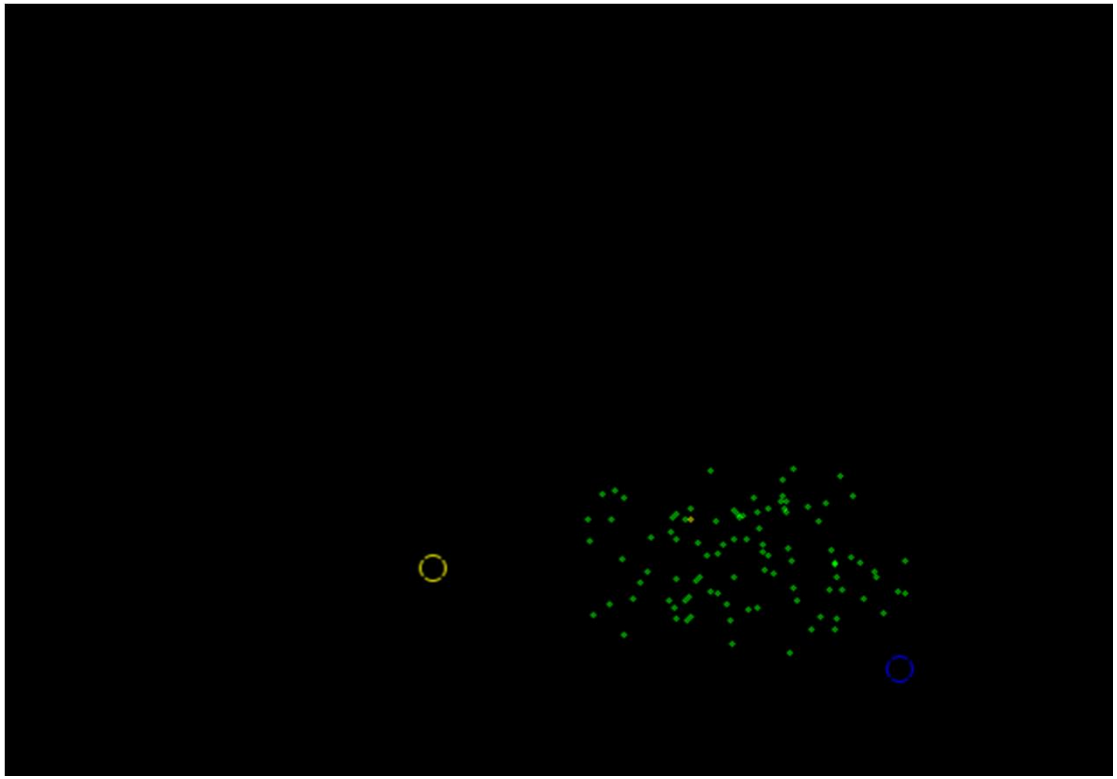


Program design. Device interfaces



Program design. Signal levels



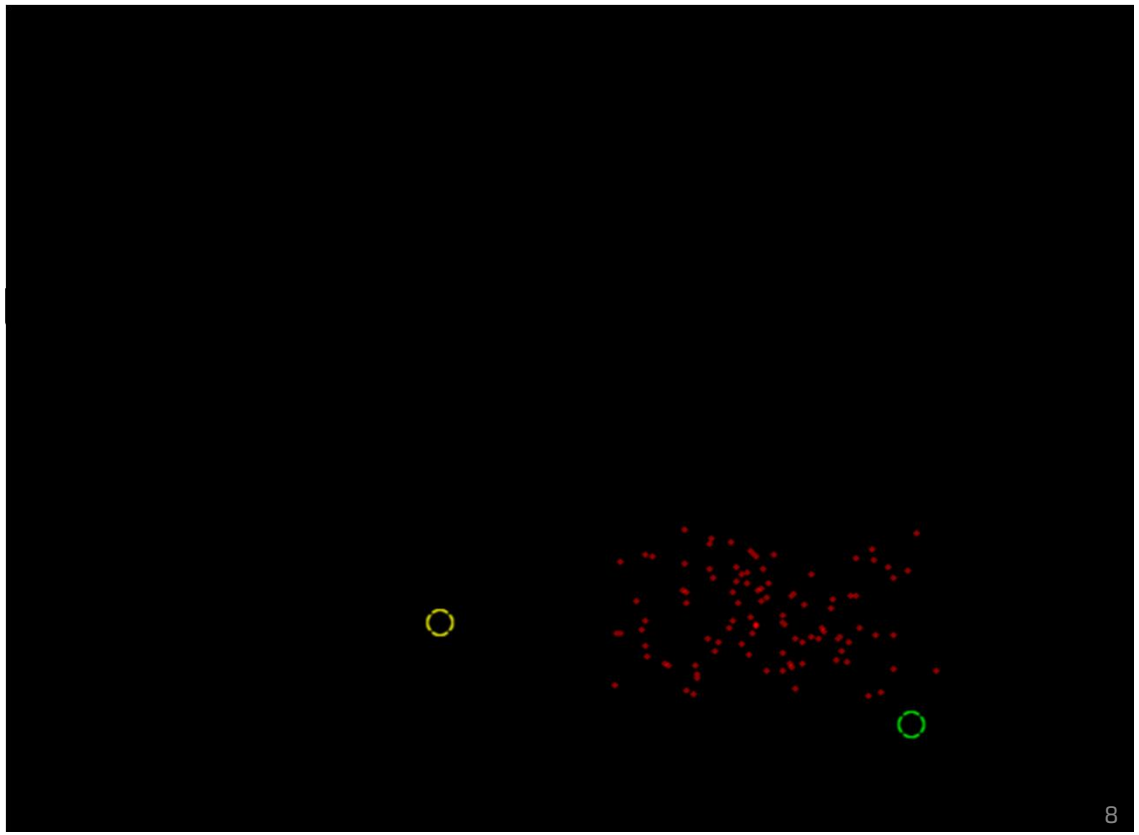


Program design. Network models

Complex network

- **has** command delays
- **continuous** signal level calculation

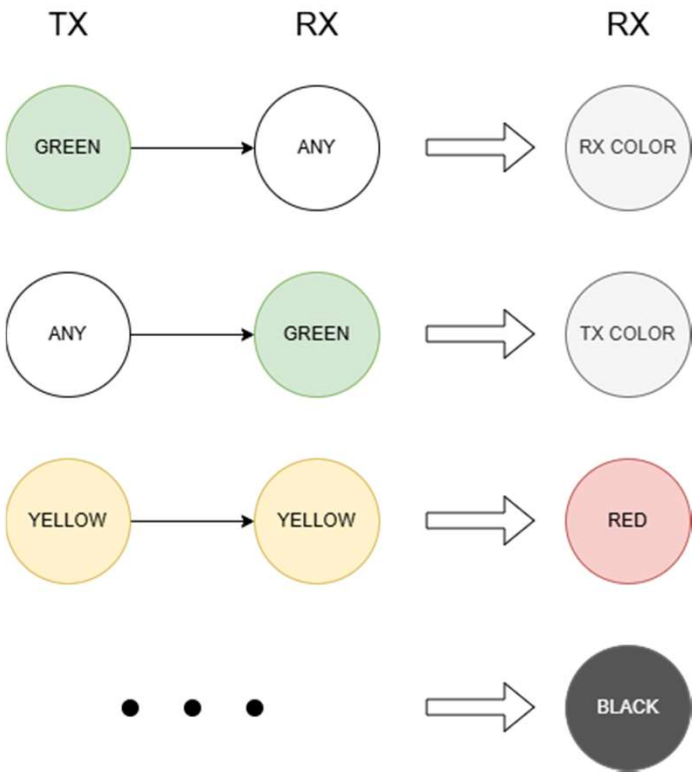
$$S_{rx} = S_{tx} \cdot k \cdot \left(\frac{\lambda}{d}\right)^2$$



Program design. Network models

Cellular automaton

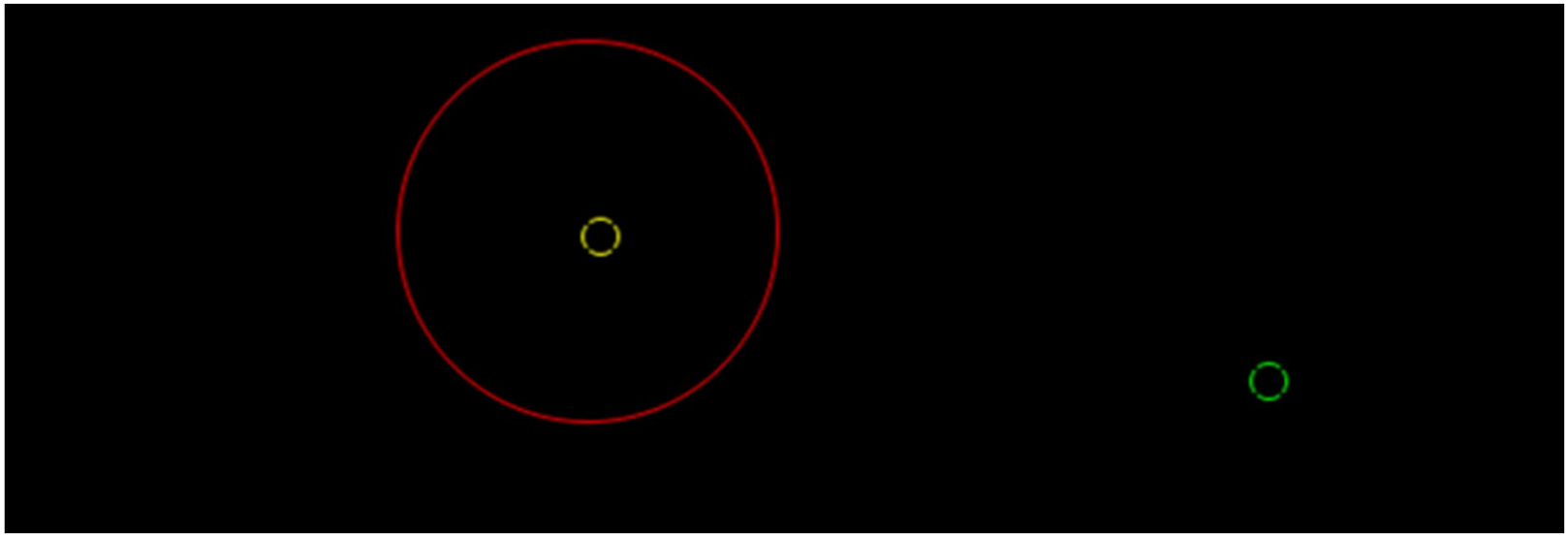
- **color-based** signal level calculation



Program design. Network models

Cellular automaton

- signal level changes happen with some **probability**



Code analysis. Traits

```
trait Receiver: Device {
    fn get_signal_level(&self, frequency: Hertz) -> &SignalLevel;
    fn set_signal_level(
        &mut self,
        frequency: Hertz,
        signal_level: SignalLevel);
    fn receives_signal(&self, frequency: Hertz) -> bool;
    fn receive_signal(
        &mut self,
        frequency: Hertz,
        signal_level: SignalLevel);
    fn receive_message(
        &mut self,
        frequency: Hertz,
        message: &Message);
}
```

Code analysis. Enums (1)

```
enum SignalLevel {
    Black(SignalStrength),
    Red(SignalStrength),
    Yellow(SignalStrength),
    Green(SignalStrength)
}
```

Code analysis. Enums (2)

```
impl SignalLevel {
    fn receive_by_color(&self, tx_signal_level: Self) -> Self {
        if tx_signal_level.is_green() {
            *self
        } else if self.is_green() {
            tx_signal_level
        } else if tx_signal_level.is_yellow()
          && self.is_yellow() {
            RED
        } else {
            BLACK
        }
    }
    /* other methods */
}
```

Code analysis. Borrow checker

```
error[E0502]: cannot borrow `drone_map` as mutable because it is also
borrowed as immutable
    --> src/device.rs:1107:22
     |
1106 | let drone1 = drone_map.get(&id1).unwrap();
     |              --------- immutable borrow occurs here
1107 | let drone2 = drone_map.get_mut(&id2).unwrap();
     |              ^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here
1108 |
1109 | dbg!(&drone1, &drone2);
     |      ------- immutable borrow later used here
…
```

Conclusion

What does Rust have to offer?

- Varios **crates**
- OOP with **traits**
- States with **enums**
- Safety with **borrow checker**

References

1. Rust Team (2018). Rust Programming Language. [online] Rust-lang.org. Available at: https://www.rust-lang.org/.
2. clap-rs (2025). GitHub - clap-rs/clap: A full featured, fast Command Line Argument Parser for Rust. [online] GitHub. Available at: https://github.com/clap-rs/clap [Accessed 17 Mar. 2025].
3. petgraph (2025). GitHub - petgraph/petgraph: Graph data structure library for Rust. [online] GitHub. Available at: https://github.com/petgraph/petgraph [Accessed 17 Mar. 2025].
4. Qiskit (2019). rustworkx/rustworkx-core at main · Qiskit/rustworkx. [online] GitHub. Available at: https://github.com/Qiskit/rustworkx/tree/main/rustworkx-core [Accessed 17 Mar. 2025].
5. bevyengine.org. (n.d.). Bevy - A data-driven game engine built in Rust. [online] Available at: https://bevyengine.org/.
6. plotters-rs (2019). GitHub - plotters-rs/plotters: A rust drawing library for high quality data plotting for both WASM and native, statically and realtimely 🦀📈🚀. [online] GitHub. Available at: https://github.com/plotters-rs/plotters [Accessed 17 Mar. 2025].
7. glium (2016). GitHub - glium/glium: Safe OpenGL wrapper for the Rust language. [online] GitHub. Available at: https://github.com/glium/glium [Accessed 17 Mar. 2025].
8. vulkano-rs (2025). GitHub - vulkano-rs/vulkano: Safe and rich Rust wrapper around the Vulkan API. [online] GitHub. Available at: https://github.com/vulkano-rs/vulkano [Accessed 17 Mar. 2025].
9. Wikipedia. (2020). Free-space path loss. [online] Available at: https://en.wikipedia.org/wiki/Free-space_path_loss.
10. rust-lang (2024). GitHub - rust-lang/book: The Rust Programming Language. [online] GitHub. Available at: https://github.com/rust-lang/book.

Thank you for your attention!