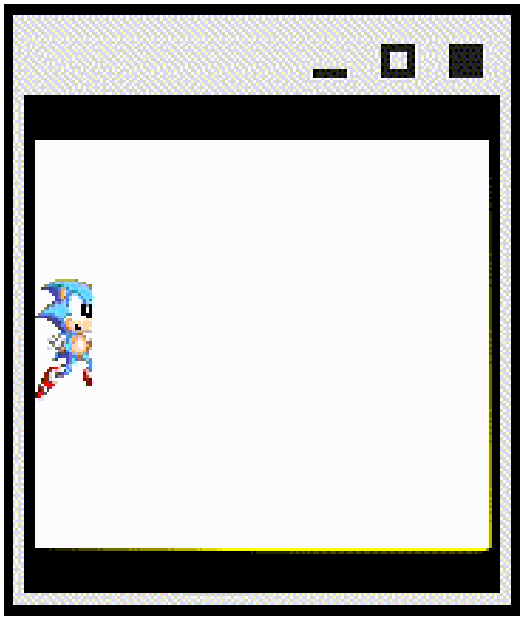
WASM 101: Porting a SEGA Game Gear emulator to the browser

whoami

- Anisse Astier
- @Aissen@treehouse.systems
- https://anisse.astier.eu
- Learning Rust for 5 years
- Started a SEGA Game Gear emulator to learn more Rust

My emulator

- gears : for SEGA Game Gear
- Written in Rust, core depends only on std
- Works on native:



What is WebAssembly

- WASM: a text and bytecode format
- A “compile once run everywhere” architecture
- In the browser: sandboxed and secure like js, close to native performance
- Multiple use cases

WEBASSEMBLY


GAME GEAR
POWER
SEGA
PORTABLE
VIDEO GAME
SYSTEM
x01
BUILDING

Compiling: WASM startup kit

- Install deps

```
$ rustup target add wasm32-unknown-unknown
```

```
$ cargo install wasm-bindgen
```

- Build

```
$ cargo build --target=wasm32-unknown-unknown
```

```
$ wasm-bindgen --out-dir dist --target web
target/wasm32-unknown-unknown/release/crate_name.wasm
```

- Run served with an HTTP server

WASM dev tools (pick one)

- With wasm-server-runner:

```
$ cargo run --target wasm32-unknown-unknown
```

- With cargo-run-wasm:

```
$ cargo run-wasm
```

- With `trunkrs.dev`, this becomes:

```
$ trunk build
```

- With `wasm-pack`:

```
$ wasm-pack build --target web
```

WASM dev tools comparison

Tool	Barebone to one-stop build solution	Works with library or binary crate	Runs Binaryen's wasm-opt	Comment
wasm-bindgen	1	both	no	needs sync between cli and lib
wasm-server-runner	2	binary	no	Can replace cargo runner
cargo run-wasm	2	binary	no	Recommends adding a workspace command and cargo alias
wasm-pack	4	library	yes	Geared toward the webpack and npm ecosystem
trunk	5	both	yes	Build driven by source html, supports multiple asset types

Entry point

- Can use binary (simpler)
- Or a library with:

```
#[wasm_bindgen(start)]
fn entry_point() { … }
```


GAME GEAR
POWER
SEGA
PORTABLE
VIDEO GAME
SYSTEM
x02
PORTING

Porting dependencies

- For desktop “UI”, picked only WASM-capable dependencies:
  - `pixels`
  - `winit`
  - `cpal`
  - `gilrs`
- How hard can it be ?

Porting dependencies: pixels and winit

- `pixels` depends on `wgpu`
  - enable `wgpu` crate **`webgl`** feature
- `pixels` init is different:
  - `winit` on canvas
  - async:
    - `wasm_bindgen_futures::spawn_local(init());`

Porting dependencies: cpal and audio

- `cpal` needs `wasm-bindgen` crate feature
- Browsers want interaction for `AudioContext` init
- Callbacks and `mpsc::channel`
  - std `mpsc channel` relies on `Condvar`, unsupported on `wasm32`
  - Wrote custom minimal channel `Arc<Mutex<VecDeque<T>>>`

Porting dependencies: time and gilrs

- `instant` crate. `web_time` also works

```
#[cfg(target_arch = "wasm32")]
use instant::{Duration, Instant};
#[cfg(not(target_arch = "wasm32"))]
use std::time::{Duration, Instant};
```

- `gilrs`: no action, but browser Gamepad API is not mature enough: different buttons than native

Porting bugs

- Too much `usize`, led to overflows on `wasm32`
  - Caught by `overflow-checks = true` config in release
  - Use u64 or u32 when necessary and add casts


GAME GEAR
POWER
SEGA
PORTABLE
VIDEO GAME
SYSTEM
x03
DEMO

Demo (FOSDEM exclusive)

https://anisse.astier.eu/f24


GAME GEAR
POWER
SEGA
PORTABLE
VIDEO GAME
SYSTEM
x04
TRICKS

Tricks: console_log

- Logging is possible with console_log crate
- Integrates well with `log` crate
- Panic hook with console_error_panic_hook

```
console_log::init_with_level(log::Level::Info).unwrap();
```

Tricks: cargo config

- `RUSTFLAGS=--cfg=web_sys_unstable_apis`
- In `.cargo/config.toml`:

```
[target.wasm32-unknown-unknown]
rustflags = [ "--cfg=web_sys_unstable_apis" ]
```

Tricks: rust-analyzer

- At the root level of your crate, configure `.cargo/config.toml`

```
[build]
target = [
    "x86_64-unknown-linux-gnu",
    "wasm32-unknown-unknown",
]
```

- Won’t work in a workspace member, must be at the root
- `cargo run` will now always need a `--target` argument

```
error: only one `--target` argument is supported
```


GAME GEAR
POWER
SEGA
PORTABLE
VIDEO GAME
SYSTEM
x05
FEEDBACK

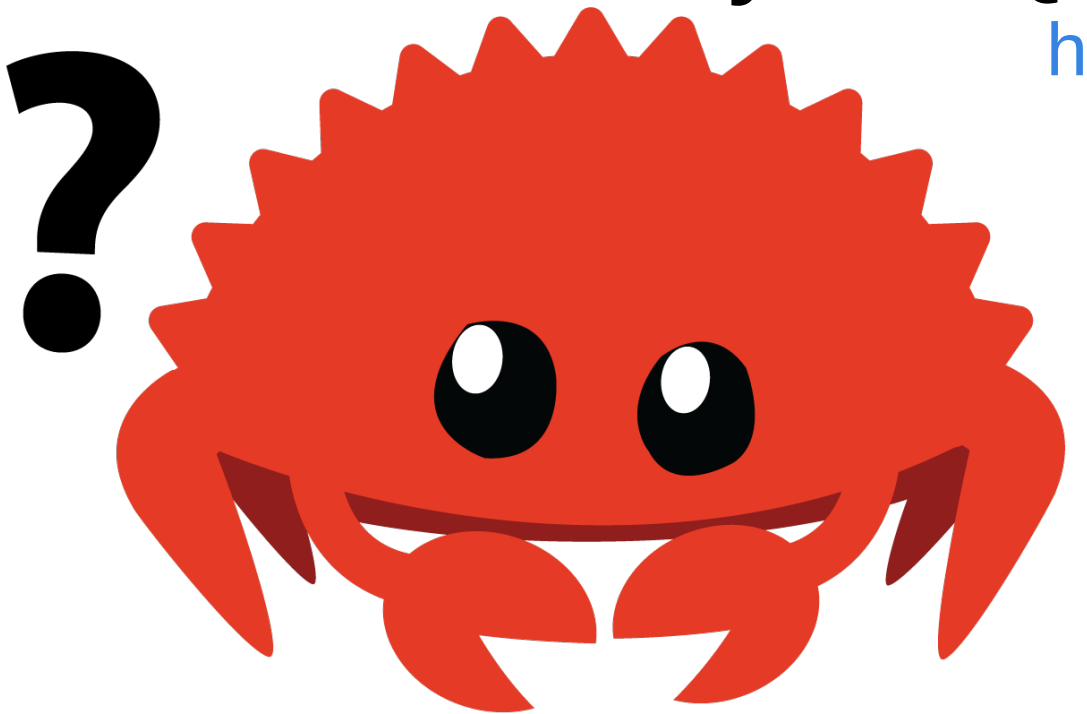
WASM+Rust feedback

- Very easy to port standalone code to WASM
- No app architecture change to port to web
- Total port took a few hours
- Custom code added for:
  - init
  - DOM interaction

To go further

- Leptos, or a another web framework is fundamental for non-ugly DOM access.
- A “real” UI with egui or slint.
  - Blurry fonts
- Minification / WASM size
- Performance measurements

Thank you! Questions ?



https://anisse.github.io/gears

https://github.com/anisse/gears

Links

- https://github.com/anisse/gears
- Wasm-bindgen guide: https://rustwasm.github.io/wasm-bindgen/
- Pixels crate: https://github.com/parasyte/pixels
- Romhack to generate level screens: https://gist.github.com/anisse/c6e410123670889038 1414f48804201b

Sources

- Game gear by Evan-Amos - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=13317134
- Web Assembly Logo by Carlos Baraza, https://github.com/carlosbaraza/web-assembly-logo
- Ferris with WASM hat and rustwasm logo, https://github.com/rustwasm/rustwasm.github.io
- Pixel ferris from the pixels crate, https://github.com/parasyte/pixels
- Ferris question by Karen Rustad Tölva, https://github.com/aldeka/rustacean.net/tree/master/site/more-crabby-things/rustbook-emotes