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
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

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

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

```rust
#[wasm_bindgen(start)]
fn entry_point() { … }
```
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
    \[\text{Arc<Mutex<VecDeque<T>>}>\]
Porting dependencies: time and gilrs

- **instant crate.** `web_time` also works

  ```rust
  #[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

SEGAX
PORTABLE
VIDEO GAME
SYSTEM
Demo (FOSDEM exclusive)

https://anisse.astier.eu/f24
Tricks: console_log

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

```rust
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`

```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
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 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/c6e4101236708890381414f48804201b
Sources

- 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