Building for the Web (WASM)

About this tutorial

This tutorial is free and open source, and all code uses the MIT license - so you are free to do with it as you like. My hope is that you will enjoy the tutorial, and make great games!

If you enjoy this and would like me to keep writing, please consider supporting my Patreon.

Web Assembly is a cool system that lets you run code compiled from non-web-based languages and run them in the browser. It comes with a few limitations:

- You are sandboxed, so you can't access much in the way of files on the user's computer.
- Threads work differently in WASM, so normal multi-threading code may not work without help.
- Your rendering back-end is going to be OpenGL, at least until WebGL is finished.
- I haven't written code to access files from the web, so you have to embed your resources. The tutorial chapters do this with the various `rltk::embedded_resource!` calls. At the very least, you need to use `include_bytes!` or similar to store the resource in the executable. (Or you can help me write a file reader!)

WASM is the tool used to make the playable chapter demos work in your browser.

Building for the Web

The process for making a WASM version of your game is a little more involved than I'd like, but it works. I typically throw it into a batch file (or shell script) to automate the process.

The Tools You Need

First of all, Rust needs to have the "target" installed to handle compilation to web assembly (WASM). The target name is `wasm32-unknown-unknown`. Assuming that you setup Rust with `rustup`, you can install it by typing:

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

You also need a tool called `wasm-bindgen`. This is a pretty impressive tool that can scan your web assembly and build the bits and pieces need to make the code run on the web. I use the command-line version (there are ways to integrate it into your system - hopefully that will be the topic of a future chapter). You can install the tool with:

```
cargo install wasm-bindgen-cli
```

Note: You'll have to reinstall `wasm-bindgen` when you update your Rust toolchain.

Step 1: Compile the program for WASM

I recommend performing a `release` build for WASM. The debug versions can be *huge*, and nobody wants to wait while an enormous program downloads. Navigate to the root of your project, and type:

```
cargo build --release --target wasm32-unknown-unknown
```

The first time you do this, it will take *a while*. It has to recompile all the libraries you are using for web assembly! This creates files in the `target/wasm32-unknown-unknown/release/` folder. There will be several folders of build information and similar, and the important files: `yourproject.d` (debug information) and `yourproject.wasm` - the actual WASM target. (Replace `yourproject` with the name of your project)

Step 2: Determine where to put the files

For the sake of simplicity, I'm going to use a target folder named `wasm`. You can use whatever you like, but you'll need to change the names in the rest of these instructions. Create the folder inside your root project folder. For example, `mkdir wasm`.

Step 3: Assemble web files

Now you need to use `wasm-bindgen` to build the web infrastructure required to integrate with the browser.

```
wasm-bindgen target\wasm32-unknown-unknown\release\yourproject.wasm --out-dir wasm
--no-modules --no-typescript
```

If you look inside the `wasm` folder, you will see two files:

- `yourproject.js` - JavaScript bindings for your project
- `yourproject_bg.wasm` - A modified version of the `wasm` output including the bindings required by the JavaScript file.

I typically rename these files to `myblob.js` and `myblob_bg.wasm`. You don't have to do that, but it lets me use the same template HTML each time.

Step 4: Create some boilerplate HTML

In your `wasm` folder, you need to make an HTML page to host/launch your application. I use the same boilerplate each time:

```html
<html>
  <head>
    <meta content="text/html;charset=utf-8" http-equiv="Content-Type" />
  </head>
  <body>
    <canvas id="canvas" width="640" height="480"></canvas>
    <script src="./myblob.js"></script>
    <script>
      window.addEventListener("load", async () => {
        await wasm_bindgen("./myblob_bg.wasm");
      });
    </script>
  </body>
</html>
```

Step 5: Host it!

You can't run WASM from a local file source (presumably for security reasons). You need to put it into a web server, and run it from there. If you have web hosting, copy your `wasm` folder to wherever you want it. You can then open the web server URL in a browser, and your game runs.

If you *don't* have web hosting, you need to install a local webserver, and serve it from there.

Help Wanted!

I'd love to integrate this into `cargo web` or similar, to provide a simple process for compiling and serving your games. I haven't made this work yet. If anyone would like to help, please head over to My Github and get in touch with me!