quicksilver

quicksilver

crates.io v0.4.0 docs passing dependencies 3 of 12 outdated

A simple 2D game framework written in pure Rust, for both the Web and Desktop

Maintenance Status

I've posted an update on my website about Quicksilver. To keep a long story short: **Quicksilver is no longer actively developed.** For now I will continue to triage bugs and pull requests and (maybe) fix small bugs.

Alpha Notice

This version of Quicksilver is currently working its way through alpha! There is still work to do on the API and on bugfixes, as well as waiting on an upstream library for audio support. Please feel free to use this version and **provide feedback!** If you run into bugs or want to give feedback on API decisions, please open an issue.

A quick example

Create a rust project and add this line to your `Cargo.toml` file under `[dependencies]`:

```
quicksilver = "0.4"
```

Then replace `src/main.rs` with the following (the contents of quicksilver's `examples/01_square.rs`):

```
// Example 1: The Square
// Open a window, and draw a colored square in it
use quicksilver::{
    geom::{Rectangle, Vector},
    graphics::Color,
    run, Graphics, Input, Result, Settings, Window,
};
```

```rust

fn main() {
    run(
        Settings {
            title: "Square Example",
            ..Settings::default()
        },
        app,
    );
}

async fn app(window: Window, mut gfx: Graphics, mut input: Input) ->
    Result<()> {
    // Clear the screen to a blank, white color
    gfx.clear(Color::WHITE);
    // Paint a blue square with a red outline in the center of our
        screen
    // It should have a top-left of (350, 100) and a size of (150, 100)
    let rect = Rectangle::new(Vector::new(350.0, 100.0), Vector::new
        (100.0, 100.0));
    gfx.fill_rect(&rect, Color::BLUE);
    gfx.stroke_rect(&rect, Color::RED);
    // Send the data to be drawn
    gfx.present(&window)?;
    loop {
        while let Some(_) = input.next_event().await {}
    }
}
```

Learning Quicksilver

A good way to get started with Quicksilver is to read and run the examples which also serve as tutorials. If you have any questions, feel free to open an issue or ask for help in the Rust Community Discord from other Quicksilver users and developers.

Made with Quicksilver

Version 0.4

- Libraries
  - Lenscas: Silver_Animation - An animation system
  - Lenscas: Mergui - A simple GUI system
  - johnpmayer: quicksilver-utils-async - Tasks, timers, and net code
  - johnpmayer: quicksilver-utils-ecs - Entity Component System integrations

- Games
  - alec-deason: Pixel Imperfect

Version 0.3

- Documentation / Tutorials
  - tomassedovic: quicksilver-roguelike
- Games
  - WushuWorks: I am the Elder God
  - codec-abc: RustyVolley
  - rickyhan: Kingston Crabfight Simulator
  - robotcaleb: Replay
  - rsribeiro: Evil Alligator
  - nycex: Axosnake
  - Leinnan: Slavic Castles
  - Lenscas: Arena keeper
- Libraries
  - Lenscas: Mergui - A simple GUI system

Want to add your project? Feel free to open an issue or PR!

Building and Deploying a Quicksilver application

Quicksilver should always compile and run on the latest stable version of Rust, for both web and desktop.

Make sure to put all your assets in a top-level folder of your crate called `static/`. *All* Quicksilver file loading-APIs will expect paths that originate in the static folder, so `static/image.png` should be referenced as `image.png`.

Linux dependencies

On Windows and Mac, all you'll need to build Quicksilver is a recent stable version of `rustc` and `cargo`. A few of Quicksilver's dependencies require Linux packages to build, namely `libudev`, `zlib`, and `alsa`. To install these on Ubuntu or Debian, run the command `sudo apt install libudev-dev zlib1g-dev alsa libasound2-dev`.

Deploying for desktop

If you're deploying for desktop platforms, build in release mode (`cargo build --release`) and copy the executable file produced (found at "target/release/") and any assets you used (image files, etc.) and create an archive (on Windows a zip file, on Unix a tar file). You should be able to distribute this archive with no problems; if there are any, please open an issue.

Deploying for the web

If you're deploying for the web, first make sure you've installed the cargo web tool. Then use `cargo web deploy` to build your application for distribution (located at `target/deploy`).

If you want to test your application locally, use `cargo web start --features quicksilver/stdweb` and open your favorite browser to the port it provides.

wasm-bindgen support Quicksilver has recently gained experimental support for `wasm-bindgen`, under the `web-sys` feature. The workflow is not currently documented here, but it should be the same as using any other library with `wasm-bindgen`.

Optional Features

Quicksilver by default tries to provide all features a 2D application may need, but not all applications need these features.

The optional features available are: - easy logging (via log, simple_logger, and web_logger) - gamepad event generation (via gilrs) - saving (via gestalt) - font rendering (via elefont) and TTF parsing (via rusttype)

Each are enabled by default, but you can specify which features you actually want to use.

Supported Platforms

The engine is supported on Windows, macOS, Linux, and the web via WebAssembly.

Mobile support would be a future possibility, but likely only through external contributions.