
ggez



What is this?

docs **passing** docs **passing** license MIT downloads **295k** downloads **295k** discord chat **32 online**

ggez is a Rust library to create a Good Game Easily.

The current version is 0.9.3.

More specifically, ggez is a lightweight cross-platform game framework for making 2D games with minimum friction. It aims to implement an API based on (a Rustified version of) the LÖVE game framework. This means it contains basic and portable 2D drawing, sound, resource loading and event handling, but finer details and performance characteristics may be different than LÖVE.

ggez is not meant to be everything to everyone, but rather a good base upon which to build. Thus it takes a fairly batteries-included approach without needing a million additions and plugins for everything imaginable, but also does not dictate higher-level functionality such as physics engine or entity component system. Instead the goal is to allow you to use whichever libraries you want to provide these functions, or build your own libraries atop ggez.

Features

- Filesystem abstraction that lets you load resources from folders or zip files
- Hardware-accelerated 2D rendering built on the [wgpu](#) graphics API
- Loading and playing .ogg, .wav and .flac files via the [rodio](#) crate
- TTF font rendering with [glyph_brush](#).
- Interface for handling keyboard and mouse events easily through callbacks
- Config file for defining engine and game settings
- Easy timing and FPS measurement functions.
- Math library integration with [mint](#).
- Some more advanced graphics options: shaders, instanced draws and render targets

Non-Features (i.e. things to add from elsewhere if needed)

- Physics
- Animation (check out keyframe; it works pretty well with ggez (source))
- GUI
- Assets manager
- AI
- ECS
- Networking

Supported platforms

- Fully supported: Windows, Linux, MacOS
- Not officially supported but might work anyway: Android, iOS, Web

For details, see docs/BuildingForEveryPlatform.md

If you want to run ggez (up to 0.7 as of now) on Android, iOS or the web using WebAssembly right now, take a look at good-web-game.

Who's using ggez?

Check out the projects list!

Usage

ggez requires rustc \geq 1.42 and is distributed on crates.io. To include it in your project, just add the dependency line to your `Cargo.toml` file:

```
1 ggez = "0.9.3"
```

ggez consists of three main parts: A `Context` object which contains all the state required to interface with the computer's hardware, an `EventHandler` trait that the user implements to register callbacks for events, and various sub-modules such as `graphics` and `audio` that provide the functionality to actually get stuff done. The general pattern is to create a struct holding your game's data which implements the `EventHandler` trait. Create a new `Context` object with default objects from a `ContextBuilder` or `Conf` object, and then call `event::run()` with the `Context` and an instance of your `EventHandler` to run your game's main loop.

See the API docs for full documentation, or the examples directory for a number of commented examples of varying complexity. Most examples show off a single feature of ggez, while [astroblasto](#) and [snake](#) are small but complete games.

Getting started

For a quick tutorial on ggez, see the Hello ggez guide in the [docs/](#) directory.

Examples

See the [examples/](#) directory in the source. Most examples show off a single feature of ggez, while [astroblasto](#) is a small but complete Asteroids-like game.

To run the examples, just check out the source and execute `cargo run --example` in the root directory:

```
1 git clone https://github.com/ggez/ggez.git
2 cd ggez
3 cargo run --example 05_astroblasto
```

If this doesn't work, see the FAQ for solutions to common problems.

Basic Project Template

```
1 use ggez::{Context, ContextBuilder, GameResult};
2 use ggez::graphics::{self, Color};
3 use ggez::event::{self, EventHandler};
4
5 fn main() {
6     // Make a Context.
7     let (mut ctx, event_loop) = ContextBuilder::new("my_game", "Cool
8         Game Author")
9         .build()
10        .expect("aieeee, could not create ggez context!");
11
12    // Create an instance of your event handler.
13    // Usually, you should provide it with the Context object to
14    // use when setting your game up.
15    let my_game = MyGame::new(&mut ctx);
16
17    // Run!
18    event::run(ctx, event_loop, my_game);
19 }
20 struct MyGame {
21     // Your state here...
22 }
```

```
23
24 impl MyGame {
25     pub fn new(_ctx: &mut Context) -> MyGame {
26         // Load/create resources such as images here.
27         MyGame {
28             // ...
29         }
30     }
31 }
32
33 impl EventHandler for MyGame {
34     fn update(&mut self, _ctx: &mut Context) -> GameResult {
35         // Update code here...
36         Ok(())
37     }
38
39     fn draw(&mut self, ctx: &mut Context) -> GameResult {
40         let mut canvas = graphics::Canvas::from_frame(ctx, Color::WHITE
41         );
42         // Draw code here...
43         canvas.finish(ctx)
44     }
45 }
```

Implementation details

ggez is built upon [winit](#) for windowing and events, [rodio](#) for sound, and a 2D drawing engine implemented with [wgpu](#). It is entirely thread-safe (though platform constraints mean the event-handling loop and drawing must be done in the main thread), and portable to Windows and Linux.

ggez is pure Rust™.

Help!

Sources of information:

- The FAQ has answers to common questions and problems.
- The API docs, a lot of design stuff is explained there.
- Check out the examples.

If you still have problems or questions, feel free to ask! Easiest ways are:

- Open an issue on the Github issue tracker
- Say hi on our new Discord server

-
- Or ask the wise people on the unofficial Rust Discord server, the Rust Gamedev server or the good-web-game Discord server

License: MIT