Grezi V3 project proposal

Isaac Mills

Neumont College of Computer Science

ABSTRACT

Grezi is a presentation software that I started developing when I begun high school. I wanted to make a piece of software that could make slideshows that I could actually pay attention to, and get something out of. I wanted to make an easy way to design engaging presentations with lots of movement, and great transitions; with slides that look clean and professional. I also wanted to be able to make these presentations using plain text. Thus: Grezi was born.

At first, it started as a smaller application split between two 2000 line files. When it worked, it worked. But when it didn't, it wasn't able to produce errors of any kind. I tried adding that feature a bit after I'd written it, but the code was just too messy to do that. This led me to re-write Grezi from the ground up into what it is today.

Because I re-wrote the parser using 'tree-sitter', it allowed me to easily add syntax highlighting for the language to my editor, and it allowed me to make a featurful language server with autocomplete, auto-formatting, various goto and rename functions, and more. And by implementing the renderer using 'egui', I was not only able to make a native runtime, but also a WebAssembly runtime allowing me to share presentations with people using a web URL. And because 'egui' is renderer agnostic, I could also render presentations to PNG, PDF, SVG and more using the 'cairo' library.

TL;DR: Ever since I started making it, Grezi has been the only presentation software I've had to use, and it's been in constant development for about a year at this point. At this point it's very featurful, but because of that, the codebase has ballooned in size. I'd like to be able to add the ability to control presentations from my phone, but to do that, I'd have to re-write a bunch of code that I wrote for WebAssembly, for Android, and that would take longer to do than just re-writing the software from the ground up. So that's what I've decided to do,

For my final project, I'd like to re-write Grezi a second time. This time instead of implementing the entire project in one package, I'm going to implement it using multiple separate packages, and I'm also going to make sure my code is much DRYer than it was before. Instead of making my code revolve around many separate functions, I'm going to make my code more object oriented and focus more on making solid types, which implement the methods I need. I'm also going to make sure that all my packages compile to every platform I need them to, with a minimal amount of conditional compilation tags in the actual source code.

I won't be able to re-implement the entire thing by the end of sprint 2, but I should be able to get the parser, renderer, and phone remote implemented by then.

This document goes into more detail about how I'm going to use what I learned in this OOP class to re-write the program

1. K.I.S.S and D.R.Y principles

There are many ways that I could simplify the current version of Grezi. Right now, Grezi's code base is made up of many very large functions that do separate things. For this re-write, I'm going to try to use more funcitons, and associate those functions with types.

2. ArrayLists and basic arrays

My code uses both the Rust 'Vec' type (equivalent to ArrayList), and basic arrays to get the best performance and memory usage out of the parser.

3. Model | View | Controller pattern (MVC)

I plan to split my project into 3 packages 'grezi_parser', 'grezi_egui', and 'grezi_native'.

3.1. grezi_parser

This package will act as the model. It's purpose will be to parse the presentation file format into a model which can be used by the other packages to display the slideshow. I'm making sure to keep the parser separate from the model itself by using a feature flag to disable the parser and it's many dependencies. That way the model can be compiled to platforms that don't support the parser.

3.2. grezi_egui

This package will act as the controller. It's purpose will be to take the output of 'grezi_parser', and draw a slide to an 'egui::Ui' instance. Because 'egui' is renderer agnostic, I don't believe this counts as a view, since the 'egui::Ui' doesn't need to know anything about how the content will be displayed. This package simply allows the instructions for how to display it to be described.

3.3. grezi_native

This package will act as the view. It's purpose will be to create a complete user interface around the slideshow using 'egui'. This package will use 'grezi_egui' to draw the slides, and it will use 'egui' to draw things around the slides such as a speaker view.

4. Memory

Rust is a low level language, and just the parser requires that I use types on the stack, and heap; as well as use values by value, and by reference. I will surley need to do all 4 for the other packages as well.

5. Method call stacks

One thing that I want to implement in this re-write is better, and easier to debug errors. To do that second part, I want to try to capture the state of the parser like so when an error is appended to the list of errors

6. Interfaces

My code will use interfaces throughout. For rendering, each Object that the parser outputs will implement the ‘egui::Widget‘ interface, or something similar. This way all the drawing code is encapsulated to each object. If I have time to implement a language server, an interface to complete auto-complete queries in the editor would also be very nice.

7. Concrete classes

Oh boy will my code implement concrete classes. The output from the paresr will be entirely composed of many concrete classes. Using concrete classes like this, each with a ‘parse‘ function, will both make parsing easier and the output easier to use and render.

8. Static

For syntax highlighting, my parser has a static map of languages and their grammars, that way I don’t need to re-initialize each language’s grammars every time I need to syntax highlight something.