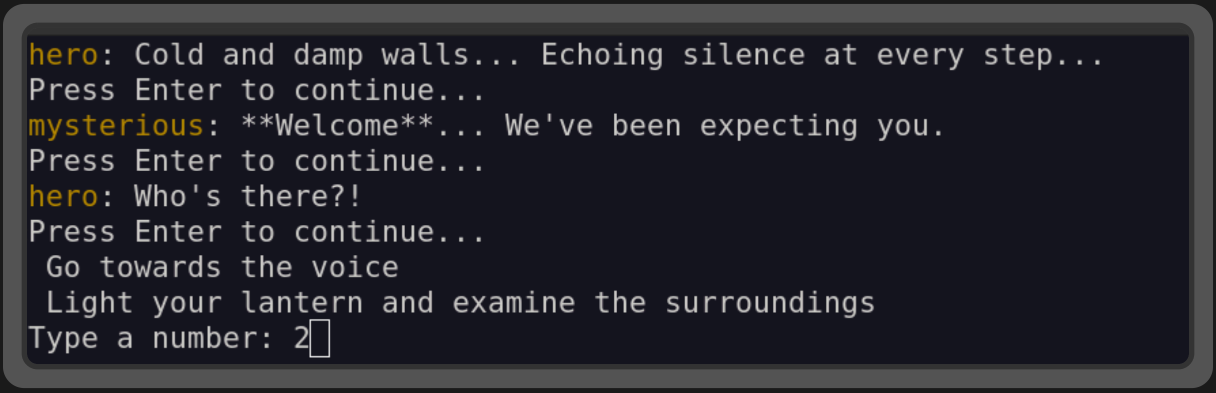
Why Text Based UIs Always Better

I think everybody who started to learn programming, (at least once) have done a text based novel like game where you choose from selections and the story progress.

Something like this,



```
hero: Cold and damp walls... Echoing silence at every step...
Press Enter to continue...
mysterious: **Welcome**... We've been expecting you.
Press Enter to continue...
hero: Who's there?!
Press Enter to continue...
 Go towards the voice
 Light your lantern and examine the surroundings
Type a number: 2
```

Figure 1: Terminal Game

But constructing this kind of things as embedded in the source code is always messy. Then I thought I should create some sort of a data structure or something before started to use **goto**'s everywhere.

And I decided to make a **DSL** (Domain Specific Language) to make my life easier. It would be a markup language, and I could add any kind of functionality any time I want.

So I created this

📖 story

```
begin novel <Hero>

    begin dialog <intro>
        <Narrator>: <A Hero's Tale>
        goto <entrance>
    end dialog

...
```

📖 story
```
    begin dialog <entrance>
        <elder>: <My child, you've finally arrived... The ((Ancient Temple))
        awaits you.>
        <elder>: <Inside, the **Crystal of Fate** is hidden. Only one with a
        pure heart can find it.>
        begin choice
            -> <Enter the temple> goto <temple>
            -> <Ask the elder more questions> goto <question>
        end choice
    end dialog

    begin dialog <question>
        <hero>: <Tell me more about this crystal's power.>
        <elder>: <It reads ~minds~, shows the ~future~. But it's not a power
        everyone can wield.>
        begin choice
            -> <Take the risk, enter the temple> goto <temple>
            -> <Give up, return to the village> goto <return>
        end choice
    end dialog

end novel
```

And I notice that it was very intuative to write. I was really enjoying. This was the time I noticed I can write DSLs for any kind of application or interface.

---

a DSL for 3D Modeling

As a programmer whenever I wanted to make a game just for fun, I encounter with the problem of creating game assets. I tried to learn **Blender** multiple times, but it wants me more time everytime I started over. So as a programmer who interests in 3D graphics and shaders, I decided to make a DSL that generate 3D models on the fly. So raymarching and SDF was perfectly suited for this job. I really recommend this article from Inigo Quilez to learn more about them.

So I of course made a DSL called SDFL and compiles down to GLSL

Here is an example program and output.

📐 SDFL

```

def eyes() {
  local(
    children: [
      smoothSubtraction(
        child1: sphere(
          position:(-1,3.6,1.4),
          radius: 0.2
        ),
        child2: sphere(
          position:(-1,3.5,1),
          radius: 0.5
        ),
        smooth_transition: 0.1
      ),
      smoothSubtraction(
        child1: sphere(
          position:(1,3.6,1.4),
          radius: 0.2
        ),
        child2: sphere(
          position:(1,3.5,1),
          radius: 0.5
        ),
        smooth_transition: 0.1
      )
    ]
  )
}
```

This is the function definition that creates the eyes of the 3D SDF model.

And this is the main scene.

📐 SDFL

```
scene(
  background: (0.9, 0.1, 0.2),
  camera: camera(
    position: (0, 3, 8)
  ),
  children: [
    plane(
      height: 0
    ),
    box(
      position: (10, 5, -15),
      size: (1, 6, 1)
    ),
    rotateAround(
      position: (0,3,0),
      rotation: (0, -20, 0) * time() * 4,
      child: smoothUnion(
        child1: rotateAround(
          position: (0,2,0),
          rotation: (0, 0, 0),
          child: smoothSubtraction(
            child2: sphere(
              position:(0,2,0),
              radius: 2
            ),
            child1: rotateAround(
            position: (0,2,0),
            rotation: (0, -90, 0),
            child: torus(
                position:(1,2,0),
                radius:0.8,
                thickness:0.58
              ),
            ),
            smooth_transition: 0.1
          ),
        ),
        child2: eyes(),
        smooth_transition: 0.2
      )
    )
  ]
)
```

Here the output:

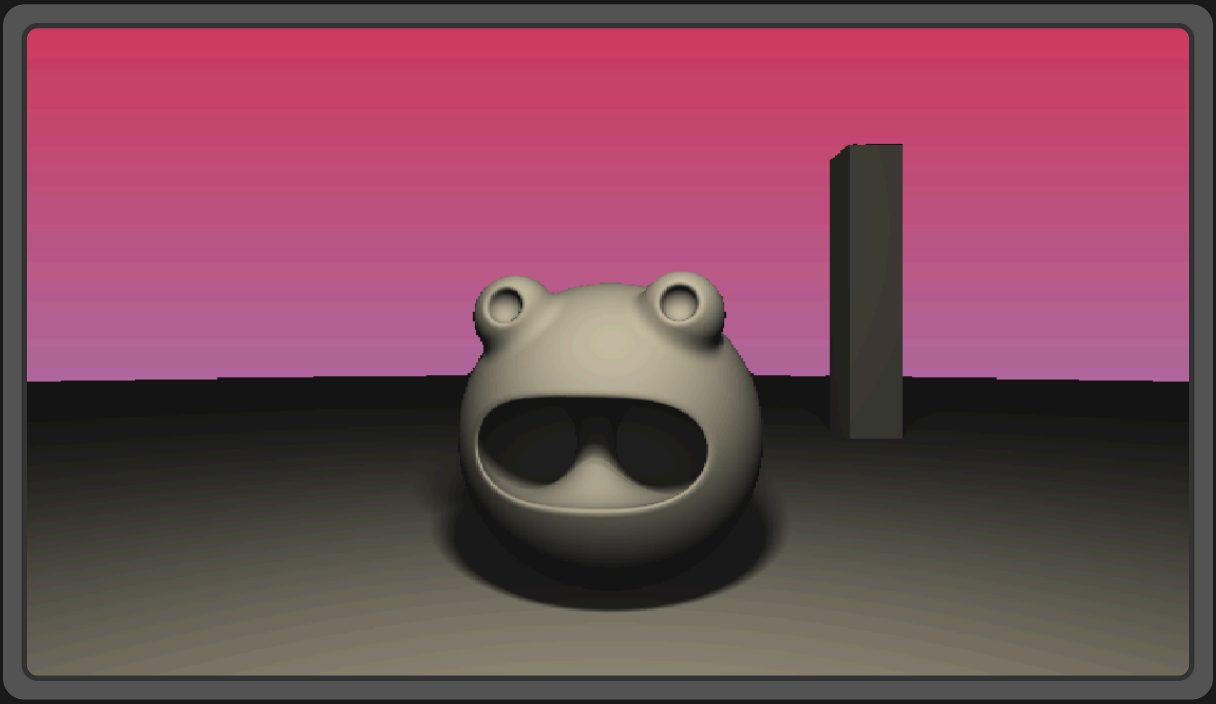


Figure 2: SDF 3D Model

And we can export **.obj** files as well!

Here the repo: "https://github.com/yasinxdxd/sdfl"

If you give me a star ⭐ make me really happy.

Really? DSL for everything?

But wait, shouldn't we be using classical **GUI**'s instead? Are they somehow **harmful**? Are DSL-based UIs going to be the next big thing? Well, not exactly.

The truth is, DSLs are already everywhere. They've quietly become an essential part of how we work with computers, each one designed to solve a specific problem elegantly. From querying databases to defining infrastructure, from styling web pages to describing hardware circuits. DSLs have carved out their own niches where they excel.

Here are some examples of DSLs you’ve probably already encountered:

DSL Name	**Description**
SQL	Query and manage data in relational databases.
HCL	Define cloud and physical infrastructure-as-code (Terraform).
YAML	Human-readable hierarchical data serialization (configuration).
Markdown	Simple text format for structured documents and web content.
CSS	Define the visual styling of web pages (colors, layout, fonts).
Regex	Compact language for defining text search and manipulation patterns.
Gherkin	Natural language used to define executable software tests (BDD).
DOT	Define nodes and edges to automatically generate diagrams and graphs.
OpenSCAD	The “Programmer’s CAD.” Defines 3D models using Constructive Solid Geometry (CSG).
CadQuery	A Python library/DSL for creating complex 3D models using a fluent API (like a chain of CAD operations).
JSCAD	A JavaScript-based Code-CAD DSL for 3D modeling, often used in a web browser.
Curv	A programming language for creating 2D and 3D geometric art using mathematics (like Shadertoy).
LilyPond	A text-based language for generating high-quality musical scores and sheet music.
TikZ/PGF	A LaTeX package/DSL for creating high-quality vector graphics (diagrams, charts) mathematically.
VHDL / Verilog	Hardware Description Languages (HDLs) used to program and describe electronic circuits and logic.

Conclusion

Text-based UIs and DSLs aren't about replacing GUIs, they're about choosing the right tool for the job. When you need precision, version control, reproducibility, and the ability to automate, text shines. DSLs take this further by giving you a clean, purpose-built syntax that's both human-readable and machine-executable.

For my text-based terminal game, creating a simple DSL meant the difference between spaghetti code full of nested conditionals and a clean, declarative structure that anyone could read and modify. And also for my 3D modeling language **SDFL** allow me define complex 3D objects using readable function calls and transformations instead of wrestling with raw shader code.

Sometimes, the best interface is the one you can `git diff`.

So next time you're building something interactive, ask yourself:

"Does this really need buttons and windows, or would a few well-chosen words do the trick?"

- yasin