I like Game Programming

8 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=G8VDU30EEFW
https://www.youtube.com/watch?v=G8vdu30EEfw

SUMMARY

The speaker discusses their progress in developing a tower defense game using the Odin programming language, emphasizing the creation of a new UI system that employs a stack-based approach for building elements. They also highlight the development of a demo runner that facilitates testing by capturing and comparing UI states, ensuring consistency and ease of debugging.

- *Transitioned from Lua to Odin for game development due to dissatisfaction with Lua and 2D experiences.*
- *Developed a stack-based UI system that allows for flexible element placement and animation control.*
- *Created a demo runner called "demo test" to automate testing of UI elements and layouts.*
- *Implemented a "golden" version comparison feature to ensure UI consistency after changes.*
- *Introduced test frames for mouse movements to simulate user interactions and validate UI behavior.*
- *Aims to achieve deterministic game behavior, ensuring that the same actions yield the same results each time.*
- *Plans to progress towards implementing tower placement and enemy movement in the game.*
- *Currently has around 13,000 lines of code, with a daily output of about 1,000 to 1,500 lines.*

All right, so if you've been following along I've been building a tower defense in Odin. I already built like 80% of it in Lua. I didn't really love the Lua experience or love 2D. So I went with Odin, Odin being a very simple language C like programming language for the love of the game. And so with that I've been building out a UI system. I've decided to change the UI system to be this kind of like stack based element building approach to where every time I want to build out an element I can just simply start this open element process which will allow me to do a bunch of stack based calls and then build the tree as it goes. And then every part of the tree can call further into every single tree and every single open method just leaves open a element such that I'm able to edit where the elements placed and how it's placed on the screen within side say an if statement right here. I can even say how the animations will work on it and all that. So that way a card that's being rendered in one spot can be rendered completely different somewhere else. It kind of makes this really nice beautiful experience that I'm very very happy about. But nonetheless, building your own UI and going through all that was a lot of fun for me. I even by the way just as I kind of like a little side note inside the builder I even built my own flex elements now which means that we're actually having our beautiful own little flex elements that say what what axis they're on. I started to go with unions and actually have subtypes of elements and made a bunch of changes.

But nonetheless, very happy about this experience. Everything is super good. And I really like my UI. I can't believe I built a UI framework I actually kind of like here. But nonetheless, building all that was fantastic. But what's the big problem about building any sort of layout system? Well, the moment you make any sort of change to your game you can impact your layout system in some way in which you did not expect. And I didn't want to have to constantly playing the game, making changes and then like I don't know a week into developing something realize I've screwed up some part of the layout and then try to have to like figure out why the layouts

kind of goofy. Like what has gone wrong with the layout? I don't know. So I built this beautiful little demo runner called demo test. So what it's going to do is it's going to load up all my different demos and just play them one at a time for me cuz I did display equal like I added the display flag. So, I can just see it. See that button was clicked without me. Menus are being expanded.

You can see opacity right here is flowing from the parent to the child and making sure it's multiplicative. You're also seeing like that middle element move. I'm doing fixed position off-axis alignment. Stats table's running. Boom, bada bing, animation, everything running. But, the best part about this is that I can remove that display and it will actually test against golden versions of that. And what do I mean by that? Well, if I go in here and say golden's, I actually get a display that brings up all the golden's. So, this takes all my demos and it records them.

So, every time I press the mouse, it changes in between the scenes on what it saved and it makes sure that every single time I make a change to my program, I should be able to see all of these running correctly and they produce the same results every time. I say, "Hey, I want this many milliseconds to go and I want to be able to take a picture at that amount of milliseconds and see it actually happen." So, that way I can build this game by building a bunch of demos. When I build a demo, I then save it and say, "Hey, this is how many frames I want you to save." Now, this is where things get a little bit clever is I built this idea called a test frame and a test frame is simply a time test frame or a mouse test frame. A mouse test frame is pretty straightforward. It's a duration plus this little mouse state right here. A mouse state, of course, is position and if the left button is down. That means over time I can actually take these little frames and I can play the mouse.

So, if I say, "Hey, take the mouse and I want you to move it to this position over 500 milliseconds." As you can see right here, it's going to lerp it and play it through my program. So, my program actually acts as if a real mouse movement has flown through it and then I can actually get the UI effects, the expands, the scales, the movements, the animations and then I can say, "Hey, at the end of every single one of these frames, I want you to take a picture and save it." So, instead of rendering to the screen, I render to a texture and either paint the texture to the screen if I want to see it or paint that texture to a file, and then I can load that file back up and test it against it every single time I run. So, that means after every single time I run a beautiful little task, I can say, "Hey, agent task file finalize." Go through, runs all my different unit tests, make sure I build for Windows, for Mac, for Darwin, Darwin AMD arm 64, blah blah blah blah blah.

And also make sure all the demos run the exact same. So, that means anytime I've screwed up anything, it's actually going to go and find that out. Because one thing I always have with unit tests, or one thing I really hate about unit tests, is that if you look at any of my unit tests, they're always the same crap where it's just like I you have so many of these like hard-coded values you have to go and make sure that are all actually the same. You have to make sure that you didn't drag like one pixel too far, or else everything breaks. If any of these values change, things just blow up, and it's always so hard to kind of go through like the minutia of every one of those details.

But, if I can mostly prove with unit tests that things are operating the way I want on the small scale, then I can do integration tests or end-to-end tests effectively, where I actually have the mouse move and animations

happen. I can take all the photos, and everything looks beautiful. And, honestly, I've been very happy with this approach. I'm I'm actually loving it very, very much so. And so, that's kind of the update I wanted to talk about a little bit is that that's what I've been exploring for the last week is like, how do I just not shoot myself in the foot? That way, if I decide to vibe out a feature, at least I know it's going to work. Which, by the way, I've been trying to vibe out this beautiful little feature right here, generate a level, and then I want to be able to see the level, and have it generate the exact same every single time. And I want to be able to have all the different display tiles display the exact same way. And to have the different height offsets and everything, because, you know, some of the tiles are a little bit higher than others, some are a little bit lower, like this one's lower than that one. Like, actually see all the differences in tile heights, and make sure that we're still rendering everything the same way. And I'm using all the different tiles, right? Like here's the super green one all the way down to the super red ones. Anyways, it's kind of fun, you know? And so that way I can actually add this to a demo and I can start adding say towers or people walking through and that way it should always produce the same walk, the same towers, the same shots because the game should be deterministic, which means that I should be able to have demos actually walking and doing everything and it should not break. And if anything breaks, I should be able to flip on logging if I've done this all correctly and be like, "Yo, yo, definitely not Fable because yo, Fable, you dead, but not Fable, why don't you go through and just tell me what's wrong?" Like read the logs, understand the pixel differences between the two images and just tell me what changed. And hopefully that means if I make any sweeping changes, I should be able to quickly understand what went wrong. And the nice part is when I have decided that I have I have made some changes and I want to update all of my stuff, I can just go Odin run dot Oh, not that.

There we go. Demo equals test save. And by doing that, it's going to save all new goldens. And by saving all new goldens, I now can make sure that I'm producing the same image or the new updated image. Okay, that's all I really wanted to talk about this time. I'm hoping that next week I should be able to start getting into tower placement and enemy movement is kind of my goal is to have enemies walk along the path, have towers be placed and to be able to fire out arrows. So I actually have cards, tower, kill enemies. I that might be a little too ambitious.

I'll probably pull it back a little bit and just have cards, towers, next level, right? Cuz if I can at least get to that point, then, you know, we're we're getting into some pretty serious progress at this point. And of course, the most useless of all measurements, but I thought I'd let you know. We're up to 13,000 lines of code. Yikes. Okay, I know I'm no Gary Tan. I'm not producing 30,000 a day, okay? I'm producing like 1,000 1,500 a day. I know. Arms weak, palms unsteady, mom's spaghetti.

All right, hey. The name is the Primagen.