Devlog 2 — November 1, 2025. UI Improvements and Fuzzing

This week was focused on UI-related tasks: creating custom themes, adding customizable font families and sizes, improving several views related to the hiring flow, and implementing skill fuzzing and the obfuscation/removal of values the player shouldn't normally see.

Font Selection and Theming

I'd been thinking about adding font selection for a while but considered it unnecessary for the time being. However, since all the testers mentioned the need for text scaling, I decided to the opportunity to implement both together.

Qt's widget scaling works very well out of the box, so that part went smoothly. Merging different stylesheets, however, was finicky, causing the themes from pyqtdarktheme to break down. Instead of trying to fix them, I built simple custom (in-house?) light and dark themes. They were quite easy to set up, and I made some of the views that used custom colors (like the e-mails) use theme-appropriate ones.

I also went through every widget with visibility issues from the original design and fixed them. The trickiest one was the custom QRangeSlider from superqt, which allows a dual-handle slider for min and max values in the talent filter. Making it visible again was surprisingly painful, but it's now working fine.

The spin boxes, on the other hand, still have usability issues. The clickable area on their buttons is tiny despite their large appearance. I'll fix that in time. Testers will surely point out others I've missed.

By popular demand, I also updated the notification system to follow the theme and font settings. Notifications now fade after five seconds instead of three. Eventually, I plan to let players customize notification behavior; how long they stay visible and how comprehensive they are.

Miscellaneous UI Work

I also tackled a smaller visual issue: the social media icons stretching with the window resolution. They no longer stretch horizontally, but vertical stretching remains a problem. The only fix I've found so far involves live recalculations during window resizing to keep the icons' 1:1 aspect ratio, which makes resizing clunky.

Restricting the main window to 'safe' resolutions would solve it, but that would be an extreme and very stupid solution for several reasons. I'll keep looking for a better approach.

Hiring Views Overhaul

The most interesting work this week was the overhaul of the hiring-related views. This felt like a big step in the right direction. I implemented nearly everything mentioned in the first devlog, except custom filters, which are still in the pipeline.

The Talent Tab was completely redone. It was trying to do too much, when the spreadsheet-like view mostly requires all the view real-state. So now it focuses solely on the table. Eventually, when more skills and attributes are added, it might get heavy again, but I plan to add a feature allowing players to hide unwanted columns, which should keep it manageable.

The Talent Profile window has also been fully redesigned. It's my first time using QDockWidgets, and they're incredibly powerful, especially when you can save and load layouts. My only concern is that they might overwhelm players; videogames don't usually give this much control over UI layout, but with a good default setup and considering the niche nature of this game, it should be fine. I plan to use it as a model for other customizable views, possibly including the main window.
The profile now includes the hiring-for-multiple-roles view that was previously in the Talent Tab. From there, you can open the Scene Planner for any selected role, reducing a few clicks when adjusting scenes to fit specific talents.

Going back and forth between the Design and Casting phases is a lot more robust now, thanks to the work in the Scene Planner.
This received several major improvements. It's now modeless and supports multiple instances open at once. You can also hire by role directly from it. The new hiring view shares the same table model as the Talent Tab, which is great for consistency for the player, and code reuse for me. It also inherited the 'filter by role' logic from the old Talent Tab, making integration nearly plug-and-play. Currently, it's still modal, and you can't open a Talent Profile from it. While this isn't ideal, I want to test it for a while. It separates this hiring method, meant for a quick 'I just want to shoot this scene', while the other offers a more detailed approach. That said, I'm open to changing it down the line.

I also added a Scene Summary view to the planner, making it easier to see the overall scene structure without having to look at each widget. However, since it's embedded in the planner, you still need to navigate back and forth or open the planner to check details. I'm considering turning it into its own dialog, perhaps with a scene selector combo box.
The Shot Scene Details dialog (from the Scenes Tab) now uses this same summary component, for which I had to use a tabbed layout, which is kinda clunky. Moving this summary into its own standalone dialog would likely improve this too. The main view of the details dialog has also been redone to instead show detailed financial data and viewer group interest in the scene.

Skill Fuzzing and Obfuscation

I'm much less happy with the skill fuzzing implementation. It introduced noticeable slowdowns in talent population and filtering, since these processes now rely on strings derived from numbers instead of using said numbers directly.

It made the original talent filter refactor (that updated the Talent Tab in real time) impossible to bear due to how slow it was, but it being made modeless and not closing on 'Apply' will hopefully be enough for now.
It's possible I'm doing this inefficiently, but if I can't optimize it in the next couple of months, I'll probably scrap it. The performance hit isn't worth it, even if it adds a bit of uncertainty to the gameplay. Other systems, and especially performance improvements, should make up for it.
Alongside fuzzing, I've obfuscated or hidden various player-facing values: decimal numbers, talent affinities, tag qualities, and viewer group sentiments now need to be 'discovered' (still a work in progress). There are likely a few more I'm forgetting. I'll probably do one of these on player-visible data soon.

Refactoring

Finally, I've started refactoring logic, data access, and UI connections. The goal is to simplify overly 'smart' views that handle too much logic themselves and to make the system easier to unit test. Once I have a clearer understanding of what this actually is, I might cover the refactor in more detail in the next devlog.