OpenVic II

Dev Diary 2

Doing it Properly

Dev Diary 2

Doing it Properly

A common pitfall that is easy to fall into as a dev team with a highly publicised project is to rush to a solution, looking for quick and easy results that fall apart under closer inspection. Therefore, the team is taking a professional approach, employing industry-standard techniques our team members are familiar with from their extensive experience in software engineering. Our first goals are; focusing on outlining requirements, goals, and building our software architecture, while we begin the basics of what will become OpenVic2.

Every software project starts with an objective or vision of a product, for us it's an open-source recreation of Victoria II. But while any amateur can knock together code, get some nice visuals quickly and convince themselves they're not far from a final product, professionals start with requirements. Whether it's user stories, use cases, requirements or a project schedule with inch pebbles approaching milestones, every project looking to design complex software will define a set of functionality as discrete, documented units. In a single phrase a requirement is "a single piece of software functionality", big or small, complex or simple. When we write requirements and we link them together, we build up a picture of what the software will look like over time. It's a good way to start off your design and funnily enough, that tends to come next when making software.

Doing it Properly

But why use requirements? Well over time software projects get big, sometimes so big that lines of code numbers in the millions. Tracking what bits work and what doesn't becomes seriously hard work, but it's easier if you can follow requirements linked in a program. Line them up against tests for them and bingo, failed tests tell you where stuff isn't working. X, Y, Z requirements failed, look for the code where they are documented as comments and you're right there on the problem. Compartmentalisation, documentation and traceability. It's just writing software like everyone does but with more mindful planning and more readable, more maintainable code.

So far we're at circa ~685 requirements and counting, of which 230 are in-production, functional requirements. By the finish of the project we'll easily be into a high 5-figure number. It's a serious bit of work and a lot of bureaucracy to manage, but in the end it's the best way to run large software projects relying on high-performance, complex code.

Dev Cycle 1 Requirements

Doing it Properly

So, why is this our priority? The team believes that in order to truly and definitively lift OpenVic2 into the modern standard in games, no half measures should be taken when it comes to the speed and stability of the product. We aren't aiming for a product that is kind of more stable with some buggy backwards compatibility with an arguably more modern feature set, we are aiming for the best of what we believe is achievable within a reasonable timeframe. Mounting architectural instability is a serious issue in many complicated software products, and it's an issue we are determined to snuff out with good planning, design, and standards for what we make.

Speaking of what we have made, our teams have hardly been idle while this planning phase takes place, the art and music teams have been hard at work drawing up concept art, UI style prototype, upscaled and updated RGO icons, and mixing professional music to an industry standard (which we will talk about more in a soon to come dev diary)

On the final page are some examples of this amazing work and the many work in progress pieces of modelling, music, concept art, and UI prototypes we have. I hope you like them!

Well that's all I have for you today folks, I look forward to seeing you all in the next dev diary!
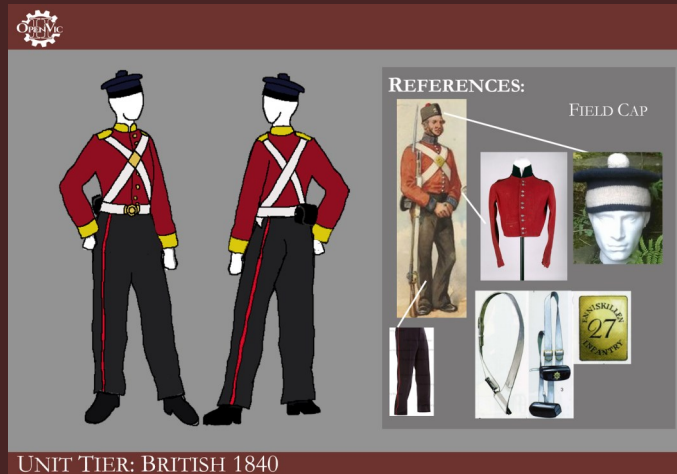
Doing it Properly

Loading Screen by PeuPeu

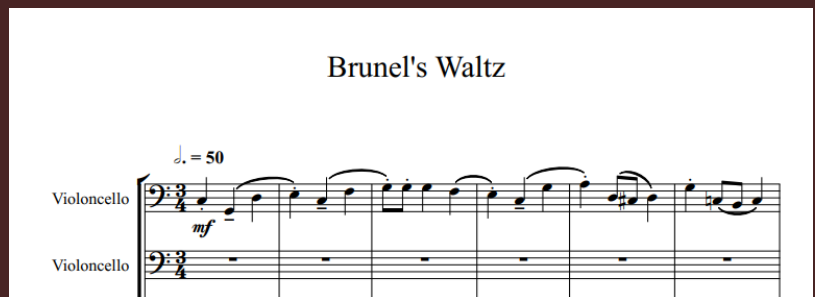Iron Resource Icon by JunkJen

Button Icons by qazdr6

Base for Unit Model by Bon Marche

Uniform Reference by Enigmatic

Catylist's take on Johan's Waltz