

Building a Software Instrument in C++ Using HISE - An Analysis

Tileyard Education London
University of Wales Trinity Saint David

Student Name: Jan Trafas
Student Number: 22514
Module Code: ADAD7005_CPL0922
Assignment Code: D2b
Word count: 4011

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been previously submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and research and that all the assistance received in preparing this thesis and sources have been acknowledged.

A handwritten signature in black ink, appearing to read 'Jan Trafas', located in the bottom right corner of the page.

Table of Contents

Table of Contents	2
List of Figures	3
Introduction	3
Build Analysis	4
Sample Creation for Kroeger	4
HISE Project - Experiential Learning	6
Kroeger - Feature Overview	11
Saturator Module	11
Reverb Panel Functionality	13
Chorus Panel Functionality	14
ADSR Panel	15
Keyboard Controls	16
Meters Panel	17
Result	19
Room For Improvement	19
Beta Testing	20
Conclusion	22
References	23

List of Figures

Fig. 1 Equaliser emphasising piano harmonics of a middle C on pink noise.

Fig. 2 EQs stacked.

Fig. 3 HISE: Attack knob selected in the interface designer window on the right (green box) and highlighted on the left (in green) where the option to duplicate it was not working.

Fig. 4 HISE: Examples of filmstrip images of a two-strip animation of a button (left) and a ten-strip animation of the movement of a switch (below).

Fig. 5 HISE: Component Specific Properties, slider style set to knob despite it being a horizontal fader which proved to be the solution to the custom filmstrip problem.

Fig. 6 HISE: Vacuum Tube texture two-piece filmstrip ready to be used for a button in the saturator module of Kroeger.

Fig. 7 HISE: Vacuum Tube Texture scale factor having to be set to 0.035 in order for the filmstrip image to fit in the interface designer.

Fig. 8 HISE: Audio & MIDI Settings buffer size being set to 512 samples in order for the interface building process to run smoother.

Fig. 9 HISE: Saturator module graphics showing amount knobs, gain offset and the vacuum tubes.

Fig. 10 HISE: Illustration of an exemplary signal flow diagram of the saturator module.

Fig. 11 HISE: Reverb Unit on Kroeger.

Fig. 12 HISE: Chorus unit on Kroeger.

Fig. 13 HISE: ADSR Panel controlling the envelope of the sound.

Fig. 14 HISE: Highlighted Status LED Button in the component list. Highlighted panel in GUI designer (centre). Parameter Properties showing linkedto: OnOff_Button (right).

Fig. 15 HISE: Module Tree: Transposer location.

Fig. 16 Transposer dial set to -24 semitones.

Fig. 17 Lines of code that made it possible for the output meters to work.

Fig. 18 HISE: The design of the output meters placed on a leather strip panel on the device.

Introduction

An executive from *Google* when asked on their opinion on the C++ programming language responded saying: "They're oversold, and used far too broadly." and "How do we have stuff like this [get to be] the standard way of computing that is taught in schools and is used in industry?" calling the language "very bureaucratic" where "Every step must be justified to the compiler". (Pike, 2010). That is the bad fame that C++ has brought upon itself over the years of frustrated developers complaining about its complexity. This essay attempts at making a case for the utility of C++ by showcasing an open-source software called *HISE* which allows anybody willing to learn a single computer programme to operate in the programming language without ever noticing they are coding or running scripts. This work also sets out to showcase that an easy-to-use instrument can be built in *HISE*.

The central theme of this paper is a software instrument called *Kroeger* which was entirely built inside *HISE* by its first-time user. This written work begins with an analysis of the sound design and build process of the instrument, to then follow by scrutinising all of its features and the methods behind how they work. The reflective part of this essay dwells on the results achieved by completing this project, from analysing its potential on the software instrument market, through looking at its advantages and areas for improvement, to feedback from beta-testing the instrument on users who use similar devices on a daily basis. The feedback from beta-testing returns a result that answers the question of whether an easily-comprehensible interface can be built inside the *HISE* environment.

Build Analysis

Sample Creation for Kroeger

The technique used to obtain samples for the instrument requires the use of a signal generator set to output 'pink noise' and an equaliser that emphasises the harmonics that normally sound in a middle C note when it is played on a piano.



Fig. 1 Equaliser emphasising piano harmonics of a middle C on pink noise.

Pink noise is a type of static noise that implements the nature of human hearing into its characteristic. As it contains no typical melodic content that can be distinguished as musical it needed altering with an equaliser to obtain the desired middle C note, which has a frequency of roughly 263Hz. The result of this operation on the pink noise is a set of resonant frequencies appearing in the spectrum which start to peak out above the rest of the hum, which in the example above has been additionally controlled with both a low-pass and high-pass filter.

Because the loudness limit of the boosts is 30dB when using bells in FabFilter Pro Q2, the clean tone-to-noise ratio after applying just one of those equalisers was still not sufficiently high and a lot of what at this point seemed like artefacts, remained. To achieve a cleaner note, four of those exact processors with the same setting were used and stacked in-line as shown in *figure two*.

The gain plugin at the end of the channel strip chain was put in place to offset the volume gain which happened due to boosting multiple resonant frequencies with four equalisers in a row.

After achieving a sound that matched the tone of a piano just ever so slightly, because the high-frequency content has been removed with multiple equalisers using high-pass filters, a bit crush effect was used which added a small amount of *top-end* to the sound and made it unique.



Fig. 2 EQs stacked.

HISE Project - Experiential Learning

HISE, which stands for *Highly Integrated Software Environment* is an open-source software framework designed for building audio plugins and standalone applications. It provides a graphical interface for creating custom digital signal processing modules and musical instruments, with a modular architecture that allows for easy customisation and expansion. *HISE* is written in C++ and was the software used to build *Kroeger*. (Hart, 2017).

Kroeger works as a sampler instrument based on own-produced sounds, so the first step at creating it was mapping the samples correctly. Because of the fact that achieving the correct EQ notches for all the harmonics in every one of the twelve semitones in a chromatic scale was incredibly time-consuming a decision was made to only create samples for one octave and then transpose the whole octaves up and down the keyboard. Then began the interface design, starting with a red box, soon a vintage stylistic began establishing itself in the project. Looking back at it from the perspective of a finished instrument, the purpose behind this was not as well thought-out as it was accidental; the vintage stylistic matched the retro-sounding samples which sounded like this because of the inability to create a higher quality timbre matching that of a real piano. Using the website *WebKnobman* (available at: g200kg.com Accessed: 24 March 2023) textures of sliders, knobs, potentiometers, faders and buttons were selected, styled and implemented into the project. The first challenge in making the instrument was learning the software that is *HISE* which is not the most intuitive to use but it atones for it by being quick to understand and highly adjustable.

The method used here, which is an experiential learning approach has numerous positives, including:

- Efficiency - learning in this way can be more efficient than traditional theoretical learning because it is focused on specific skills and knowledge needed for a particular job or task.
- Practicality - it allows immediate application of what has been learned to the real specific environment.
- Confidence - as experience and knowledge are gained through this type of learning, building confidence in one's abilities and becoming more self-sufficient is an immediate and direct result.

At first, the creation and adding of functionality seemed to be a mundane task that needed to be repeated over and over for similar potentiometers to work in similar, but not identical ways. It would have been easier if duplication of components would have

had been possible. As it turns out it was, but because of a bug in the software or perhaps a feature of *HISE* the typical copy-and-paste keyboard commands would not work. The solution was selecting the component that needed to be copied in the interface designer, using the mouse cursor with a left click, instead of highlighting it in the component list. After that, the key command Cmd + D could be used to duplicate any selected entity. This is a great example of a learning experience that turned out to save a lot of time once the functionality of *HISE* became clearer.



Fig. 3 HISE: Attack knob selected in the interface designer window on the right (green box) and highlighted on the left (in green) where the option to duplicate it was not working.

Another feature which was found to be problematic in the development of the device were custom textures used for some of the sliders. In order to be able to speak about this issue, some terms need be explained. In the context of the HISE scripting language, *numstrips* are the number of individual frames or images that make up a custom filmstrip. A *filmstrip* is a sequence of images used in graphical user interfaces (GUIs) to create animations or to display different visual states of a control (such as a button). In HISE, custom filmstrips can be created for use in plugins or instruments, and *numstrips* refer to the number of individual images in that filmstrip. For example, if a custom filmstrip has ten images to display a button animation, the *numstrips* value would be ten.



Fig. 4 HISE: Examples of filmstrip images of a two-strip animation of a button (left) and a ten-strip animation of the movement of a switch (below).



When adding a custom texture and saving the preset to overwrite previous progress and record the most recent completed actions, seemingly at random, the custom *filmstrips* would disappear and the default textures for sliders and buttons would show. When asked on the *HISE* forum, one user responded:

Every time you add a new component HISE needs to re compile, this is normal behaviour, you don't need to change the strip number just click compile after you add or remove components.

(Soundavid, HISE Forum, 16 March 2023)

After taking a while to figure out what *compiling* was, since the workflow used was not code-based but rather graphic UI-lead, this piece of advice has proven to be incorrect. The problem persisted and the solution found by trial and error was setting the slider type from *vertical* to *knob* which is counterintuitive since the texture of a fader is far from looking like a knob, but this is what eradicated the problem anyway.

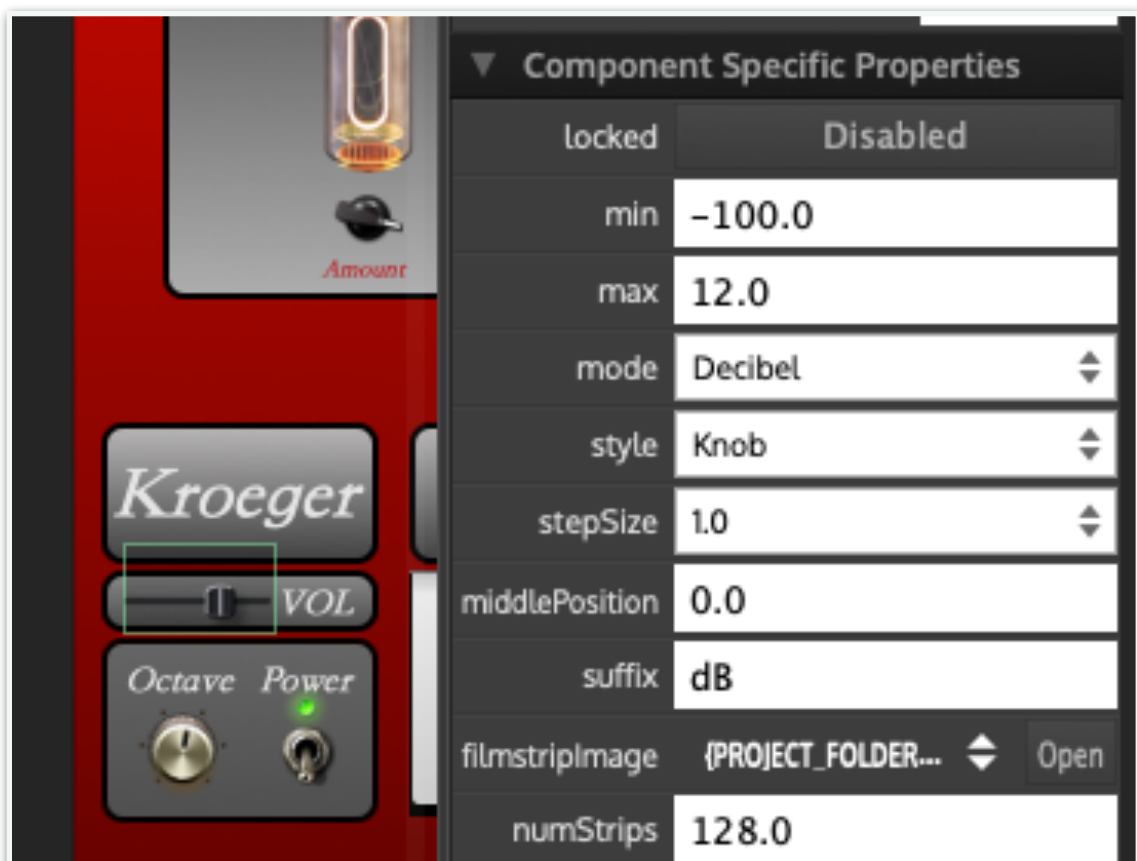


Fig. 5 HISE: Component Specific Properties, slider style set to *knob* despite it being a horizontal fader which proved to be the solution to the custom *filmstrip* problem.



While learning more about how the *HISE* environment works and becoming more comfortable with its functions, the author of the instrument has decided to attempt designing their own buttons. Building the *saturator* module required a matching vintage-looking style of valves, so vacuum tubes were chosen for the design of buttons used to enable all three individual *saturator* modules. Using *Canva* high quality graphics of the vacuum tubes were obtained. By observing the design of the *filmstrips* downloaded from *Knobman* the principle was adapted to designing own vacuum tube switches. As the graphic only had a picture of the tube while it was on, there was a need to create the texture for when it was off. For this, the saturation was dropped, as colour is only visible when it reflects light (Shevell, 2003), and a black shape following the filament of the bulb was put-in. The contrast was dropped and the exposure brought down to create a realistic look of the lamp being off. As the default position of the switch is always off, the corresponding image belonged at the top of the *filmstrip* and the on-position-texture stayed at the bottom.

Fig. 6 HISE: Vacuum Tube texture two-piece filmstrip ready to be used for a button in the *saturator* module of Kroeger.

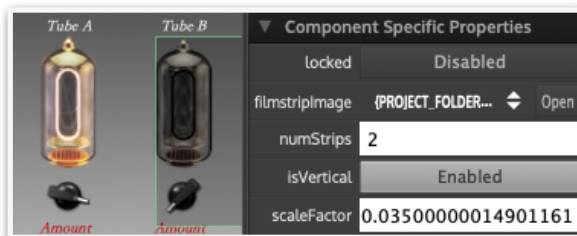


Fig. 7 HISE: Vacuum Tube Texture scale factor having to be set to 0.035 in order for the *filmstrip* image to fit in the interface designer.

This texture contained images of a great quality which were not compressed or resized for this project. Because the *scale factor* setting only shrinks the image in terms of its vector, the resolution remains the same and thus is as CPU-heavy as if the image was scaled to its full size.

This became a problem as more features were being added, the instrument device began to run more and more slowly, where new windows would open with significant delay and buttons would respond, sometimes seconds, after pressing them. The solution that was found to have stopped this behaviour was increasing the buffer size for audio in the programme settings of *HISE*. In the future, more attention should be paid to image sizes and aimless high-resolution imagery which does not improve the overall design of a device will be avoided.

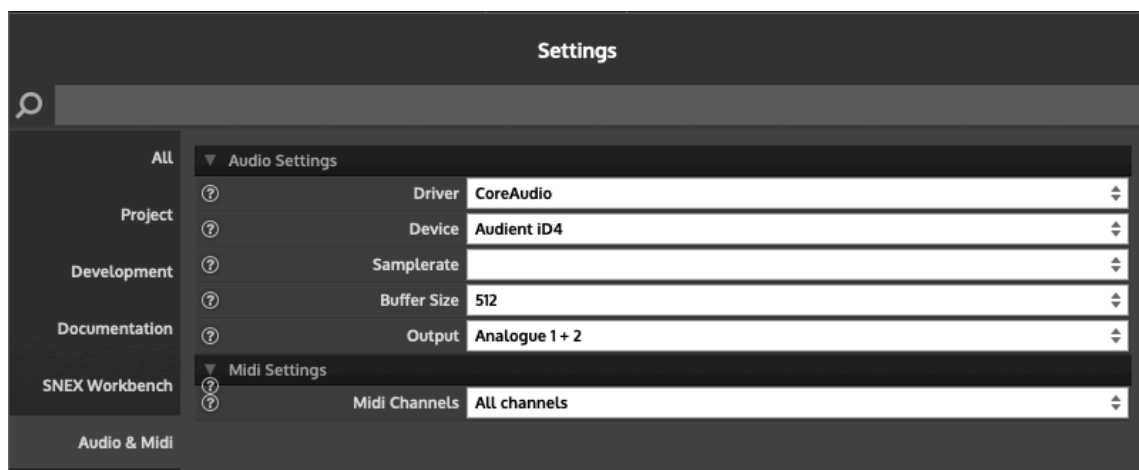


Fig. 8 HISE: Audio & MIDI Settings buffer size being set to 512 samples in order for the interface building process to run smoother.

Kroeger - Feature Overview

Kroeger has many features of vintage electronic pianos from the nineteen-seventies, (Borden, 2015) paired with modern units for a unique experience created for users who value the vintage sound but do not want to miss the more recent capabilities that software instruments have been offering to the world of digital music creation in the most recent years. The retro-fitted features include the sound of the samples themselves, which as a result of the inability to closely mimic a piano sound, has an undeniably aged character to it. (Beentjes, 2020) The inclusion of the saturator module that utilises vacuum tubes is another addition users can possibly value very highly. The ADSR (Attack, Decay, Sustain, Release) module allows for shaping the envelope of the built-in sample and the reverb and chorus units let the user build their tone on top of that.

Saturator Module

The *amount* knobs under each valve each have a different limit, thirty per cent for *Tube A*, seventy for *Tube B* and one hundred for *Tube C*. In this way, identical software saturation processors were turned into different devices providing varying levels of distortion. The fact that each tube can work on its own, as well as in combination with any amount of neighbouring bulbs is an attempt at an isolated power supply type of electrical design, where multiple devices are not co-dependent and can operate individually. This creates a limitation however, which is that the order of the saturation processors, when more than one is enabled, cannot be changed, and will always remain the same, as shown in figure ten.

When looking at the module illustration images it must be disclaimed that the signal flow diagram is only a representation of what an electric circuit would look like if *Kroeger* included actual, real, analogue vacuum tubes. It is helpful in understanding the design choices behind the saturator module on the device.



Fig. 9 HISE: Saturator module graphics showing amount knobs, gain offset and the vacuum tubes.

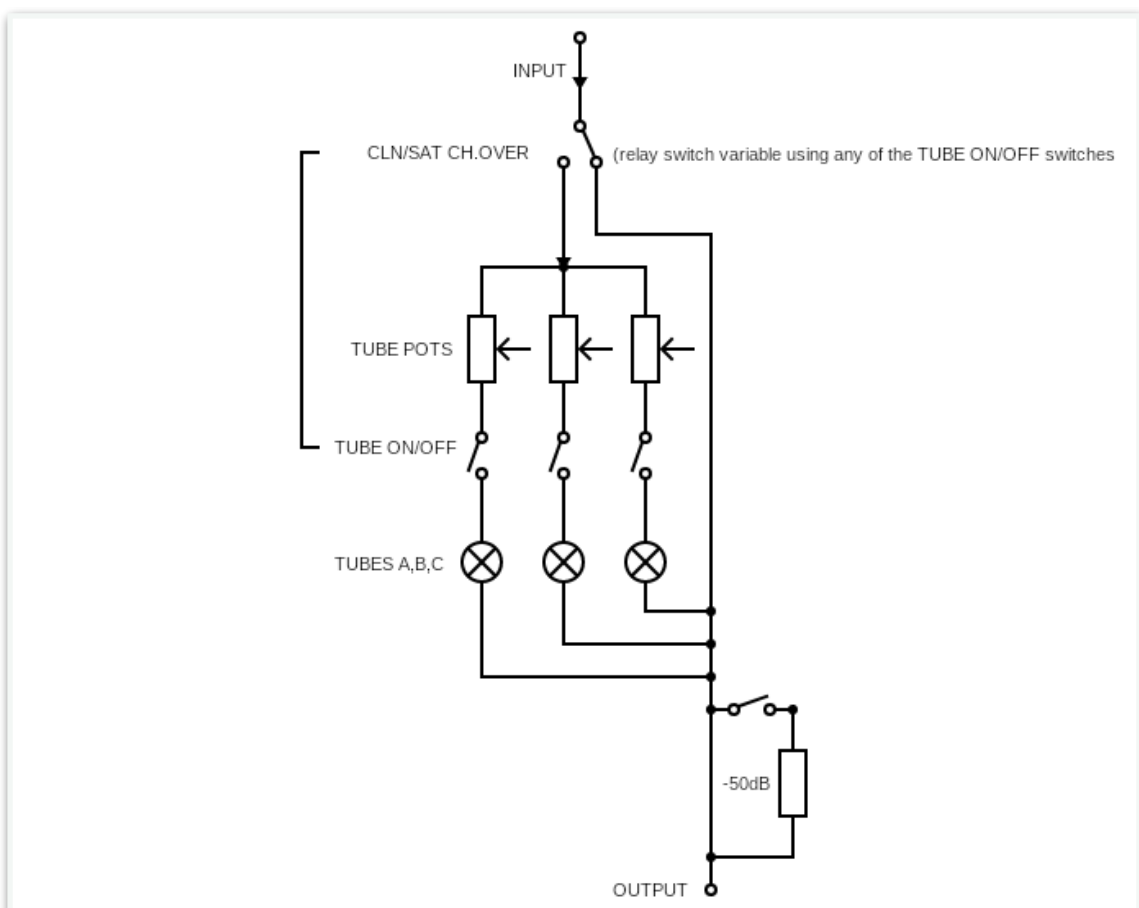


Fig. 10 HISE: Illustration of an exemplary signal flow diagram of the saturator module. Includes:

- CLEAN/SATURATED CHANGEOVER operated with any of the TUBE ON/OFF switches.
- TUBE POTENTIOMETERS placed on individual tube circuits.
- GAIN OFFSET switch at end of chain removing 50dB of volume.
- Presented order of tube circuits joining the main circuit before the output.

Reverb Panel Functionality

The reverb panel is connected to the *Simple Reverb* effects unit under the sampler path of the instrument design tree. Its simple design resembles the world's first reverbs. Those were spring or plate-based however, this one leans towards the chamber/hall area. (Sound On Sound, 1997)

All there is to it is a *Size* control dial which as self-explanatory as it is, actually extends the decay time on the effect unit. The *Wet* dial adjusts how much of the affected signal is joined with the unaffected and the last control is a power switch to enable or bypass the reverb unit. The reverb in this instrument does not just serve the function of placing the instrument in a room, but during beta-testing and playing Kroeger it has proven itself to be a great sound design tool for shaping the otherwise standard sound. Paired with the saturation tubes many pads; drone-like, eerie and atmospheric sounds can be achieved.



Fig. 11 HISE: Reverb Unit on Kroeger.

Chorus Panel Functionality

The chorus being one of the most popular effects used on keyboards and pianos, in the decades to which Kroeger is trying to teleport its user to, had to make its appearance at the end of the effects panel above the keyboard. It is routed to affect the clean sound after the ADSR processor but before the reverb when all is turned on, to preserve the naturality of the reverb. It has a *Rate* dial, which adjusts the frequency of the chorus. The *Width* knob controls the stereo width of the effect and *Feedback* controls how much of the signal is fed back into the effect. With that dial turned all the way up and paired with a long reverb the sound begins to morph into a never-ending siren of a feedback loop which also makes the chorus hold multiple sound-design capabilities.



Fig. 12 HISE: Chorus unit on Kroeger.

ADSR Panel

The ADSR panel used to control the envelope of the sound is probably the most powerful tool of the entire interface. It allows the user to go from a plucked piano sound, to a padded-out synth-like swell. The controls have been marked with different suffixes, as per their assigned parameters. The *Attack* knob displays milliseconds, and so do the *Decay* and the *Release*. The *Sustain*, however shows decibels, because this control sets the level of the sound's sustain, it is the decay time that is responsible for its duration.



Fig. 13 HISE: ADSR Panel controlling the envelope of the sound.

Keyboard Controls

The keyboard control panel contains three components; the *Octave* potentiometer, the *Volume* horizontal fader and a vertical *Power* switch. Powering off the device also switches the status LED to the red colour, indicating the device is off and no sound will be produced. A status LED inside *HISE* is done using a button, that is not usable on its own and is tied to another button - in this case the On/Off switch.



Fig. 14 HISE: Highlighted *Status LED Button* in the component list. Highlighted panel in GUI designer (centre). Parameter Properties showing *linkedto: OnOff_Button* (right).

Transposer

The transposer knob is a simple MIDI transposer device plugged straight into the sampler in the *Module Tree*. It is covered with a custom texture of a panning potentiometer due to the fact that transposition can be viewed as a left-to-right motion along the keyboard, just like panning a signal in stereo from the left to the right channel.

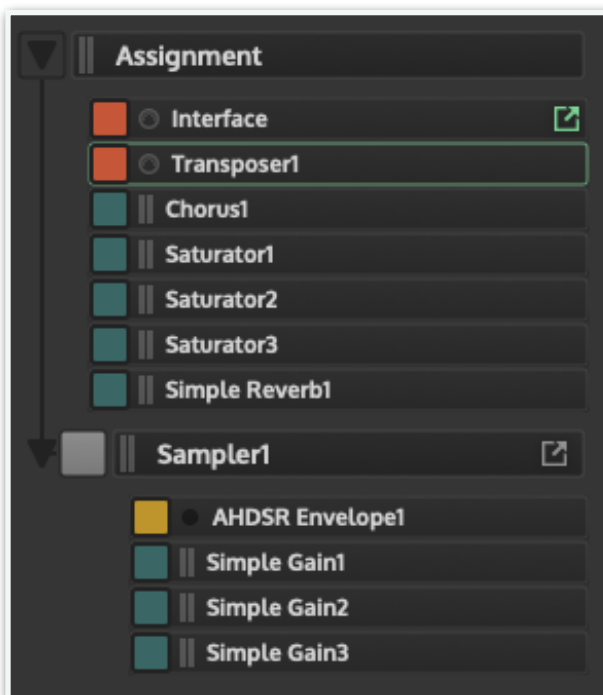


Fig. 15 HISE: Module Tree: Transposer location.

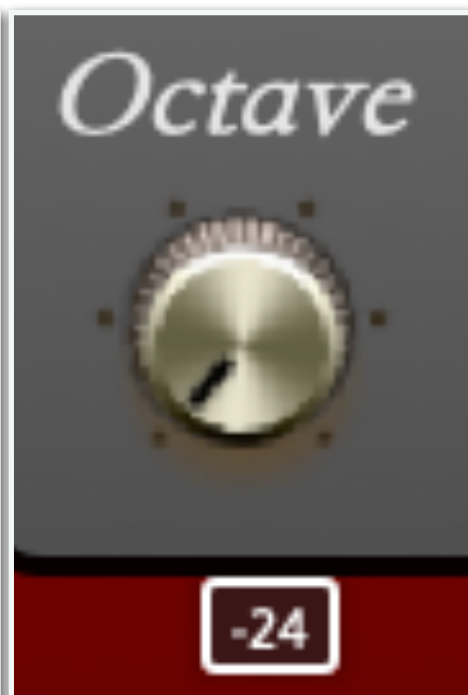


Fig. 16 Transposer dial set to -24 semitones.

Meters Panel

The output meters posed an issue to create at first and that is why the initial design of *Kroeger* did not include them at all. After careful and thorough research through the *HISE* forum a solution was found - sliders. The idea is to take the values of both of the stereo master outputs which *HISE*'s engine already stores, and input them as the value to which two corresponding sliders would be set at any given time.

Being the only element in the whole design of *Kroeger* that required coding, the functioning output meters are the largest accomplishment of this task.

```
//meters//

const var LeftMeter =
Content.getComponent("LeftMeter");
const var RightMeter =
Content.getComponent("RightMeter");

reg timer = Engine.createTimerObject();
timer.setTimerCallback(function()
{

LeftMeter.setValue(Engine.getMasterPeakLevel(0));
RightMeter.setValue(Engine.getMasterPeakLevel(1));
});

timer.startTimer(30);
```

Fig. 17 Lines of code that made it possible for the output meters to work.

Having overcome this issue, textures that matched the design of the rest of the instrument had to be implemented. The *VU Meters* that can be seen in figure fifteen are working meters with their needles moving accordingly with the master outputs of the left and right channels. The problem with this design is that the way the code was written does not allow for the output values to be displayed in decibels, and the needles respond as if the *VU Meters* were PPM meters, which is not at all in their nature. This is a design flaw that can definitely be assessed and fixed in later development of *Kroeger* or while working on other instruments.



Fig. 18 HISE: The design of the output meters placed on a leather strip panel on the device.

With all the features combined and working together, Kroeger was complete. Any more development and it would lose its basic and rudimental look and feel which is supposed to be one of its unique selling points. The development of the device does not stop here but for the purpose of this project and to the best of the author's abilities, the known resources of knowledge have been exhausted by this point.

Result

The finished product is a functional, playable instrument which fits a stylistic closely related to the vintage electronic pianos of the nineteen seventies. (Borden, 2015).

When it comes to competing on the market of plugins and virtual instruments it objectively does not stand a chance against the big companies making VST keyboards, like the industry leaders *Arturia* or *Kontakt*. If the goal was ever to perform on a level that is anywhere near one of those, a whole team of developers would have to be hired with tasks assigned to each of them and a few years taken into account to create such a device. Working on an instrument alone however has its upsides. The most important one being that it is a great educational tool to utilise the C++ programming language as well as to channel one's creativity and patience fuelled by inspiration to develop something new. As mentioned before, the do-it-yourself quality of *Kroeger* deems it appropriate even to sell, because the market for anything DIY or indie-made is now at its ultimate peak. (Zuydgeest, 2022).

One might ask what the direct and indirect results of completing such a project are. The answer is; a personal experience with a subjective opinion built based on the troubles which were possible to overcome and problems which have not been managed. *Kroeger* stands as an example of what can be accomplished as a first project, made inside *HISE*.

Room For Improvement

Obvious areas for improvement include initial sample quality, sampler mapping, interface design depth, controls' sensitivities and graphics of custom textures and panels. Obtaining better quality samples would be the key move here, because they are the building blocks of an instrument of this type. Either a better, more educated version of the sound design technique that was used to obtain individual notes for this project or a completely different one using synthesis to achieve interesting sounds may be the way to go about creating new sounds. With graphic design, improvement seems to be slightly further and almost out of reach as of now, but *Kroeger's* creator developing design and animation skills is not a completely unattainable scenario. Treating *Kroeger* as a demo is the most appropriate approach as even though it is a finished and working sampler instrument, it could be optimised to run smoother in many ways. The fact that *HISE* is a young platform only having been established in two thousand and seventeen by *Hart Instruments* shows that it will develop and grow as is now its community on the official forum page.

Beta Testing

Before offering *Kroeger* for users to test, some assumptions on what the results of the tests might be can be derived from knowledge about its design compiled with market research into other software instruments.

Some users may appreciate the simplicity of *Kroeger's* design and interface, as it allows for quick and easy sampling and manipulation of sounds. Others may find the limited controls and lack of advanced features to be frustrating or limiting when creating music. Additionally, users may have varying opinions on the quality of the samples used in the instrument, as well as the overall sound and character of the instrument. Some may appreciate the vintage electronic piano sound that *Kroeger* aims to emulate, while others may be pushed away and would rather have access to more modern and versatile sounds, especially in the case of purchasing the instrument.

Kroeger paired with a MIDI keyboard was given to two users from two opposite sides of the spectrum when it comes to knowledge about music production. *Subject A* is an educated musician, producer and sound engineer and feedback from them is valuable from a technical standpoint and can be implemented by using features of *HISE* or *C++* that are yet to be researched by the author of *Kroeger*. *Subject B* however, is not at all a potential customer and does not belong to the target audience of this instrument's release. Feedback from them should provide important cues for the design of the graphic user interface, whether it is rudimentary easy to use or if its features require too much trial and error attempts due to the complexity of its design.

Beta - Testing Results

Category	Subject A	Subject B
Enjoyability	7	8
Ease of Use	9	6
Quality of Sound	6	7
Graphic Design	9	5
Unambiguity of Features	7	3
General Rating	7.6	5.8

Fig. 19 Table: Results of beta-testing *Kroeger*.

The results displayed above mark a guideline on how further development of *Kroeger* and similar devices should proceed. The more musically-educated *Subject A* highlighted features like *Ease of Use* and *Graphic Design* highly which seems to be an indicator of a well-designed interface, but for *Subject B* these were one of the lowest-scoring points. *Subject B* also graded *Unambiguity of Features* the lowest out of all the categories, which would not be a concern given that they have had virtually no experience in music production, however *Subject A* also indicated a problem by not giving that category a full mark. What one can derive from this *beta test* is that as simple as the design of *Kroeger* might seem to the builder, there are some labelling problems on the interface. The saturator module for example, for the experienced producer that is *Subject A* the vacuum tubes pose no questions and they immediately figure out what the module is supposed to do. For inexperienced users, this would be a confusing and unclear feature.

Subject A has indicated that the feature of *Kroeger* that needs the most work is the sample behind it, by rating the *Quality of Sound* metric a six out-of-ten. This points to what has been discussed earlier when speaking of the sound design part of the build. The inability of the instrument's developer to achieve a high quality sound has been exploited and implemented as a sort of feature and characteristic of the instrument only because of the limitations that were faced, not due to a conscious choice being made. This can not stay as a rule for building plug-ins in the future and must be addressed by the developer if they are willing to compete in the virtual instrument market.

Conclusion

In summary, Kroeger project resulted in the creation of a functional sampler instrument. Once it is acknowledged that the instrument was built by a single person as their first complete attempt at using the *HISE* platform and C++ programming language, the matter of it competing with larger instrument-making brands on the market becomes insignificant. The instrument's design is closely related to vintage electronic pianos from the 1970s and features a simple user interface with basic controls. Kroeger's simple build and design can be seen as a unique selling point and makes the instrument hold its place in a potential market for indie-made equipment and software. What is more important however, is the educational value the project has provided to its author. The process behind this project established that experiential learning is the fastest and most efficient way of gaining skills for the author.

Kroeger project not only showcases its own areas for improvement which can all be addressed and visited when creating another instrument in the future, but it also serves as a great educational tool and demonstrates the potential for creating new instruments using the *HISE* platform.

Another point made during the reflection upon the progress made with this device is that C++ as a language is useful for building software, in which coding and scripting are not necessary for building instruments. Since components inside *HISE* operate as stable chunks of code which is written in the background of every action that is completed inside the software, the user never or very rarely has to deal with any programming language at all. This opens doors for many beginners in the area of instrument development and allows for creative flow to stay unobstructed by the issues normally faced with C++.

References

Beentjes, D. (2020) Classic Keys: The Wurlitzer Electronic Piano, Abbey Road Institute. Available at: <https://abbeyroadinstitute.com/miami/blog/classic-keys-the-wurlitzer-electronic-piano/> (Accessed: 24 March 2023).

Borden, D. (2015) Music For Amplified Keyboard Instruments, Pitchfork. Available at: <https://pitchfork.com/reviews/albums/20265-music-for-amplified-keyboard-instruments/> (Accessed: 24 March 2023).

Dallon Adams, R. (2020) Tech Republic: The 4 Most Hated Programming languages: Experts Pile On Javascript, C++ and more. Available at: <https://www.techrepublic.com/article/the-4-most-hated-programming-languages-experts-pile-on-javascript-c-and-more/> (Accessed: 24 March 2023).

Disley, A.C., Howard, D.M. and Hunt, A.D. (2006) 'Timbral description of musical instruments', International Conference on Music Perception and Cognition, pp. 61–68. Available at: https://www.researchgate.net/profile/Andy-Hunt-2/publication/242557004_Timbral_description_of_musical_instruments/links/0deec5293be34ded21000000/Timbral-description-of-musical-instruments.pdf.

Fab (2022) Viscount Legend '70s Review – Vintage E. Piano Meets Modular, Audioe News Room. Available at: <https://audionewsroom.net/2022/03/viscount-legend-70s-review-vintage-piano-meets-modular.html> (Accessed: 24 March 2023).

g200kg (no date) WebKnobMan: Knob Gallery. Japan. Available at: <https://www.g200kg.com/en/webknobman/gallery.php> (Accessed: 24 March 2023).

Hart, C. (2017) HISE: The Open Source Framework for Sample Based Instruments. Available at: <http://hise.audio> (Accessed: 24 March 2023).

Industry Talk (1997) The History Of The Hammond, Sound On Sound. Available at: <https://www.soundonsound.com/people/history-hammond> (Accessed: 24 March 2023).

Jackson, J. (2010) 'Google executive frustrated by Java, C++ complexity', CIO, July. Available at: https://www2.cio.com.au/article/print/354210/google_executive_frustrated_by_java_c_complexity/.

Kalev, D. and Pike, R. (2010) InformIT. Available at: <https://www.informit.com/articles/article.aspx?p=1623555> (Accessed: 24 March 2023).

OpenAI (2023) OpenAI: ChatGPT 3. Available at: <https://chat.openai.com/> (Accessed: 24 March 2023).

Regen, J. (2012) Keyboard Magazine: Vintage Vibe Electric Pianos. Available at: <https://web.archive.org/web/20131103122218/http://www.keyboardmag.com/article/vintage-vibe-electric-pianos/150018> (Accessed: 24 March 2023).

Reid, G. (2023) Vintage Vibe Deluxe 73, Sound On Sound. Available at: <https://www.soundonsound.com/reviews/vintage-vibe-deluxe-73> (Accessed: 24 March 2023).

Shevell, S.K. (2003) 'Color Appearance', in *The Science of Color*. Elsevier, pp. 149–190. doi:10.1016/B978-044451251-2/50005-2.

Zuydgeest, R. (2022) NEW HABITS The Online Shopping Boom of DIY Products Has Stabilized At The Pandemic Peak Level, USP Marketing Consultancy. Available at: <https://www.usp-research.com/new-habits-the-online-shopping-boom-of-diy-products-has-stabilized-at-the-pandemic-peak-level/> (Accessed: 24 March 2023).