LEVEL 0

INTRODUCTION

If you have spent some time working in Nyquist, you may have noticed that it doesn't exactly seem to be made for traditional music composition. Although you have many options when it comes to generating sounds and manipulating them, you may find it a bit harder to build a full song, from start to finish, and have a coherent result at the end, without writing hundreds of lines of scores. The present guide tries to make this task a bit easier. Trying to make some kind of exhaustive guide to composition in Nyquist would be particularly complicated: each music genre has it codes, its own patterns, and composing a piano score is a fundamentally different thing from composing a modern rap beat. This is the reason why we will only focus on one very specific genre here: chiptune. Although many of the functions or techniques explained here are applicable to a multitude of other genres, our main focus will be to get all the tools needed to compose chiptune (with some pop music inspirations as well).

A GENRE ?

Chiptune isn't exactly a genre: it's more of an specific choice of instruments and sounds that you can use to do pretty much anything you want. More precisely, retro synthesizers made with very simple waveforms. If you listen to chiptune tracks, they will inevitably remind you of 80's or 90's video game soundtracks: the memory capacities of chips was extremely limited at the time, so these very basic waveforms was the only instruments that could be used to create melodies. This also means that a lot of the focus of chiptune goes to the melodies you generate rather than the refinement of the sounds you make them with: impressing people with your flawless sound design will be particularly hard when the only tool you have is a single strident sawtooth wave, so the only thing you can reliably count on is the melody itself: perfect for a composition-based toolkit!

Because chiptune is more of a sound aesthetic than a genre, a lot of very different music tracks have used it. While most traditional chiptune music would be considered underground, Kesha's « Tik Tok » is simply a chiptune melody put over classic early 2010's pop drums, and Daft Punk's « Veridis Quo » is highly chiptune-inspired as well. This more popular, musically open-ended side of chiptune is what this Nyquist guide hopes to help you re-create. The present guide will show the creation of a chiptune song step-by-step, while giving you all the tools you need to create your own. But, before, we'll go over some music fundamentals: if you're not a musician, the next section should get you up to date on the couple of concepts we'll need.

LEVEL 1

NOTES AND SCALES

The first section of this guide aims to review some fundamental music theory concepts necessary for composition, not only in chiptune but in most modern genres as well. Music theory is a particularly rich and complex subject that would take a lifetime to tackle correctly, so we will make two choices:

- First, we'll focus on modern, wide-appeal music genres. Not only are we all familiar with at least some of them, but modern popular music also has the advantage of being harmonically and rhythmically "simpler" than many alternatives such as jazz, art rock, jazz, classical music, jazz, 1950's German microtonal songs and jazz.
- Then, we're going to make lots of generalizations.

The most central concept to composing music is the note. A note corresponds to a particular sound frequency in Hz. Although there are many ways of demarcating notes around the world, the most common system in western music is a twelve-note system, which is designated by letters:

C Cs / Df D Ds / Ef E F Fs / Gf G Gs / Af A As / Bf B

These twelve notes form what we call an octave, which can be designated by a number. For example, a4 is a note corresponding to a sound wave of 440 Hz, b4 is another note on the same octave at 494 Hz, and a3 is the same note on a different octave, at 220 Hz. The fundamental frequency of the human voice is typically around 165 to 255 Hz for a female voice, compared to 90 to 155 Hz for a male voice.

However, if you try to compose songs using all the notes in all the octaves, there is a good chance that the result will be unpleasant. Indeed, the vast majority of songs that you have heard and will hear during your life have scales. A scale is a group of notes, usually 7, that "work well together." There are a very large number of different scales, and many of them overlap. Scales are also the reason why there are only 7 letters for 12 notes: the letters A, B, C, D, E, F designate the 7 notes of the C major scale. When you compose a modern song, pretty much all the notes you will use will be part of a single scale.

To designate a scale, we associate one of its notes, called the "fundamental", with a mode. Mode is a somewhat abstract concept. Fundamentally, the mode is an instruction manual that explains how to find the other 6 notes in your scale: a mode is a list of intervals, and you can find all the notes of a given scale by simply following the intervals starting from the fundamental. For example, the Major mode is defined by the intervals: 2, 2, 1, 2, 2, 2, 1. If you apply these intervals starting from C, you get C, D, E, F, G, A , B: this is our C major scale.

Now that you have your 7 favorite notes, you may be tempted to play them all at once. You will then be bitterly disappointed, because although it will sound better than when you played twelve notes at the time, the result will not be exactly harmonious either. In fact, scales are just bags of notes: to have a harmonious result, you have to pick certain notes from this bag, generally between 2 and 5, very often 3, to form a chord. A chord is a set of notes that (finally) sounds good when played all at once. A chord is also referred to by its root note and its key, plus some other elements that are outside the scope of this guide.

It's About Emotion

The usefulness of modes may not seem obvious at first, but it becomes much more interesting when you realize that certain modes can evoke certain emotions!

For example, (massive over-simplification incoming), we tend to say that the major mode (or Ionian) sounds powerful and bright while the minor mode (or Dorian) sounds more melancholic.

Among the less common modes, the Phrygian mode is often associated with a certain tension and the Lydian mode with a mystical sensation. Keep in mind that the major and minor modes are sufficient to cover much of the composition, and emotional spectrum, of modern popular music.

For example, the song « I Want It All » by Queen is built on the B minor scale, which contains the notes B, C♯, D, E, F♯, G and A. The song repeats a sequence of four chords, comprised of notes that are all drawn from this scale:

Bm, which contains B, D and F#
G, which contains G, B and D
A, which contains A, C# and E
Bm again

A chord sequence that you repeat over and over is called a chord progression, and some are so popular that they seem instantly familiar. Try playing Am F C G on a piano: it will probably remind you of some song you know, even if you and I probably won't have the same songs in mind.

Now do musicians who compose songs think about all of this? Do they first choose a scale, then draw from it for their chords? For most of them, absolutely not: most musicians don't even think about what scale they are composing in. They find the chords by ear, and follow what they think "sounds good" from experience. This approach will often work, but scales and notes are important for two reasons: they allow you to understand why what you're playing works, and if you're new to composing, they're a great place to start!

LEVEL 2

MAKING MELODIES

Now that we know how to choose our notes and chords, it's time to talk melody. If you have already done some projects with Nyquist it is very likely that you have already made melodies, so I will mainly focus on what are the additional tools that the Chiptune Guide provides, and how you can use them. First of all, you have access to 12 synthesizers, which you can use by simply calling them like your usual instrument, except both parameters, pitch and duration, are optional.

```
>> chip-keys-1(pitch: c3,
duration: 2)

>> chip-pad-3(pitch: df2,
duration: 1.2)
```

There are 3 types of synthesizers, which are all called following the chip-*type*-*number* format:
- Keys: short sounds adequate for fast, defined melodies. Keys are numbered from 1 to 7
- Lead: sustained sounds adequate for slower melodies. There are only 2 leads.
- Pad: sustained, slow-attack sounds adequate for background supporting melodies or chords. Pads are numbered from 1 to 3.

You can use Nyquist's `score-gen` to create your melodies, but four tools in this guide could make this creation much simpler by returning all-made Nyquist `scores`. First, the arpeggiator. In music, we call an arpeggio a very simple melody which consists of playing the notes of a chord in a loop, at regular intervals, typically in an order from the lowest to the highest. For example, an arpeggio of the G chord can be obtained by playing the notes g3, b3 and d4 in succession. The `chip-arpeggiator` function allows you to replicate this effect, but also to create your own note patterns. For example, a pattern of {1 2 3 3 2} with notes {c3 d3 e3} will result in the melody c3 d3 e3 e3 d3 on a loop. You can also control the `speed` (i.e. the number of notes per 4-beat period, aka measure) of the arpeggiator.

If you want more randomness in your melody, the `random-melody-maker` should help: given a scale and a chord, it generates a melody that will sound harmonious with this combination. This means you can generate melodies that fit a whole chord progression by calling this function multiple times on a single scale but different chords. The `proba` parameter allows you to control the chance of a note just being a silence, while `speed` is still the number of notes per 4-beat period. In both functions above, `duration` is the duration of a single note.

```
>> chip-arpeggiator(quote(chip-pad-1), notes: {c4 e4 g4 c5},
pattern: {1 2 3}, speed: 6, duration: 20, bpm: 120, measures: 1)

>> random-melody-maker(quote(chip-pad-1), my-scale: {c4 d4 e4 f4
g4 a4 b4}, notes: {c4 e4 g4 c5}, speed: 8, proba: 0.5, bpm:
120, measures: 1, duration: 20)
```

If you want to accompany your arpeggio with a chord pad which will serve as a background, you can use the `chord-player` function, that just plays a full chord at a time. `Humanization` is a parameter that controls the randomness of the strumming: if you set it to a non-zero value, the notes of the chord will not play exactly at the same time, creating a more « human » playing effect.

```
>> chord-player(quote(chip-pad-2), notes: {c4 e4 g4 c5},
humanization: 0.0, duration: 1, bpm: 120, measures: 1)

>> chord-player(quote(chip-keys-3), notes: {cs2 f2}, humanization:
 0.2, duration: 4, bpm: 120, measures: 3)
```

Finally, the `chord-sequencer` function allows you to apply the three functions above to an entire sequence of chords. Simply choose a chord sequence, configure a few parameters, and you have a ready-made arpeggio and accompaniment. 3 pre-recorded chord progressions, `chord-prog1`, `chord-prog2`, `chord-prog3` allow you to have this effect without even having to find your chords.

To use chord-sequencer, you simply have to enter your chord progression, choose a type as your second argument (0 for the chord player, 1 for the arpeggiator, 2 for the random melodies) and then add the parameters you would normally have entered for the corresponding instrument. The chord-sequencer will then return the corresponding score. As with all the function above, not specifying optional (orange) parameters will result in the default ones being applied (for example, the default `speed` of the arpeggiator is 8). All functions mentioned above work with all the provided synths and basses of this guide. If you want to make a full song with a given progression, there's a good chance you may want to use all the functionalities of the chord sequencer. To showcase them better, here are a few different melodies generated with it. Feel free to try them out!

```
set my-chord-prog = {{c4 e4 a4} {d4 f4 a4} {d4 g4 b4} {e4 g4 b4}}

; Create the different melodies

set backing-synth = chord-sequencer(quote(chip-keys-6), 0, my-chord-
prog,  humanization: 0.0, duration: 4, bpm: 120, measures: 1)

set backing-pad = chord-sequencer(quote(chip-pad-1), 0, my-chord-prog,
humanization: 0.0, duration: 2, bpm: 120, measures: 1)

set arpeggio = chord-sequencer(quote(chip-lead-2), 1, my-chord-prog,
pattern: {1 2 3}, bpm: 120, measures: 1, duration: 1.8)

set slow-arpeggio = chord-sequencer(quote(chip-keys-4), 1, my-chord-
prog, pattern: {1 2 3}, bpm: 120, measures: 1, duration: 1.8, speed: 4)

set backing-pattern = chord-sequencer(quote(chip-keys-2), 1, my-chord-
prog, pattern: {1 2 1 2}, speed: 8, bpm: 120, measures: 1)

set random-synth = chord-sequencer(quote(chip-keys-7), 2, my-chord-prog,
bpm: 120, measures: 1, duration: 2, speed: 16, proba: 0.6)
```

LEVEL 3

ADDING DRUMS

Before we talk about drums, let's talk about rhythm and time. The main unit to qualify time in a song is the measure. A measure has a certain number of beats in it, which is just a smaller unit of measure. The overwhelming majority of the songs you have heard in your life have 4 beats in each measure, although if you're an amateur of waltzes (what are you doing here?) there's a chance that 3-beats-by-measure songs come pretty close. The current guide only supports 4-beat measures.

The « speed » of a song can be quantified by the number of beats-per-minute (bpm) they have. It could be tempting to believe that the higher the number of beats per minute, the faster a song will sound. This is sometimes true, but not always. « Lose Yourself » by Eminem has 171 beats per minute, which is much more than DragonForce's « Cry Thunder » with 130 — yet the latter sounds much faster. To keep it short, the feeling of speed associated with a song has more to do with how you place your rhythmic and melodic elements than with the bpm itself.

However, most functions in this toolkit have bpm as an optional parameter, the default being 120, so you may want to get used to what a certain bpm « feels like » for different melodies.

To compose our drums, we will use the a very basic electronic drum-set layout, composed of 6 elements:

- the kick, very low frequency drums that « anchors » the rhythm. Almost all modern music genres put kicks on the 1st beat of a measure, at least.
- The snare, which often happens on the 2nd and 4th beat of a measure
- The hi-hat, a very high-pitched drum that adds rhythm and speed to the pattern
- The cymbal, or crash, that helps initiates the beginning of a pattern and is often placed on the 1st beat of a measure
- The open-hat, that plays a similar role to the hi-hat but has much longer decay happens less frequently
- Various percussions that can be used to add some singularities to the beat

```
>> drum-player(path: "", drum-name: "kick", number: 1)

>> drum-player(path: "", drum-name: "oh", number: 2)
```

To get a drum's `SOUND` in order to play it, simply call the `drum-player` function with the path of the directory where the chiptune drum kit can be found — or an empty string if it's in the same directory as your ChipGuide_Drums.sal file. Then, you can specify the name and number of the drum. There are 3 drums of each kind: kick, snare, hh (hi-hat), crash, oh (open hat), perc (percussion).

Is bass a drum piece ?

We tend to classify most instruments in two categories: the melodic section, and the rhythmic section. In most modern music genres, the bass somehow escapes this classification. It has pitch and needs to be on-tune, and you use it to play *melodies*. Yet, its role is so critical for rhythm that we more often choose to consider it as a drum piece.

In Funk, bass lines are created to play a back-and-forth game with the drum sets and create the groovy feel that characterizes the genre. In the overwhelming majority of modern trap beats, we simply line up the 808's (the basses)

Composing the scores of drum patterns can be a very tedious task. Luckily, 3 pre-made common drum patterns are ready for you to use.

To use a pre-made drum pattern, simply enter the bpm, the path of the drum kit as mentioned above, and the list of which drum numbers you're choosing. The format is {Kick, Snare, HH} for the pattern 1, {Kick, Snare, HH, OH} for the pattern 2 and {Kick, Snare, HH, Perc} for the pattern 3. As an example, if you want to use the first pattern with kick 1, snare 3 and HH 2, you will pass the list {1 3 2} in the `instr_list` parameter.

It is important to note that the drum pattern functions return a list of scores, one for each drum piece: you will want to mix those separately, so it's important to have separate scores to play with. Here's an example of how you would use a drum pattern function. Here, I repeat each drum score twice to match the length of the melodies defined in part 2. Notice how each individual score is one of the elements of the drum pattern function's output. To use a drum pattern, simply call the `drum_pattern_1`, `drum_pattern_2` or `drum_pattern_3` functions.

```
set all-drums-scores = drum_pattern_1(bpm: 120, instr_list: {2 2 1},
path: « /Users/johndoe/nyquist")

set kicksco = score-repeat(nth(0, all-drums-scores), 2)

set snaresco = score-repeat(nth(1, all-drums-scores), 2)

set hhsco = score-repeat(nth(2, all-drums-scores), 2)
```

This Guide also includes 5 basses, which can be used the exact same way as the synths. Their names are `chip-bass-1`, `chip-bass-2`... and so on until `chip-bass-5`.

LEVEL 4

MIXING

Once you have obtained the scores for your melodies and drums, you will need to mix them: make them fit together in a harmonious way. This usually entails two main problems: how to balance the volumes, and how to add effects. For the volume balancing, the process itself is fairly straightforward: you can simply create lists for each instrument with its score and its desired volume as value, I'll show you later what to do with the lists.

```
set mix-backing-synth = list(backing-synth, 1)
set mix-backing-pad = list(backing-pad, 0.003)
set mix-arpeggio = list(arpeggio, 0.8)
set mix-slow-arpeggio = list(slow-arpeggio, 0.008)
set mix-backing-pattern = list(backing-pattern, 3)
set mix-random-synth = list(random-synth, 2)
```

The tricky part is figuring out the values themselves: how loud you want each instrument to be compared to each other? First, because in Nyquist many instruments use fundamentally different waveforms or processes, they may need very different tuning: it is pretty common for an instrument with volume 1 and another with volume 0.03 to sound as loud as each other when put together. In other words, there will be no math in this section: just listening to different sounds, and figuring out what sounds best to you through trial-and-error. A good habit may be to pick your main melody, set its volume to 1, and then try to use a combination of sim and normal multiplication to compare each of your score's volumes with your main melody, adjusting everything along the way. I am by no means a mixing engineer, but after composing on DAW's for many years here would be my two main tips, for Nyquist but also for mixing in general:

- Try your mix on different devices. A good mix relies heavily on which volume balances feel « right » and « natural », and the perfect mix you made with your headphones might sound absolutely terrible the second you put in on a speaker. Before you get anything resembling a final track, make sure you have tested it on headphones, speakers and some simple smartphone audio output.
- Don't spend too much time on your mix at once. After hours listening to the same sounds in a loop, your brain will loose all sense of what feels like « right » or « wrong » volume balances, and you're likely to just undo your own work over and over. If you find yourself spending more than 30min on a mix, it's probably time to take a break and continue next time!

Once you have figured your volumes out, simply add them all to a list and call the `mix-scores` function on the list. This will define a **section** of your track: this could be a chorus, a verse, or anything you want. For example, if you want your verse to have a kick, a snare and an arpeggio, you can call the `mix-scores`

function on a list with these tree elements and it will return your verse as a `SOUND`. For the next parts of your composition, it is critical that all your sections are the same length (e.g. 4 measures) and the same bpm. After balancing the volumes and getting the different parts of your song, you can add effects. In usual DAW's you would add effects instrument-by-instrument. Here, for simplicity, the `effect-mixer` function that adds effects does it **section** per **section**. Namely, it adds four effects: a low pass, a high pass, a delay and a reverb. There are many other effects out there, but these should give you a base. Here is an example of how to use the `effect-mixer` directly on the output of `mix_scores`:

```
set verse = mix-scores(list(mix-backing-pad, mix-arpeggio, mix-
backing-pattern)

set section = effect-mixer(verse, low: pwlv(400, 8, 900), high: 3000,
reverb-strength: 0.1, delay-speed: 8, delay-strength: 0.2, bpm: 120)
```

`low` and `high` determine the cutoff thresholds for the low and high passes, `reverb-strength` and `delay-strength` should be set between 0 and 1 and determine how audible these effects will be. `delay-speed` determines how many times a delayed sound will be heard in a measure, and the bpm parameter allows the function to enforce this speed.

There are a lot of different ways to use these effects, but as guidelines I generally set delay and reverb strengths to 0.1 or 0.2, and simply turn them to 0 when the section has drums in it: these are nice effects to use once in a while, but drowning your mix in delay and reverb may end up making it sound more confusing than appealing to listeners. Both the low and high parameters can be used with envelopes or scalars, the other ones can only be used with scalars.

EQ AND FRENCH RAP

EQ's may be one of the easiest ways to make cool, instantly recognizable effects which I will showcase using exclusively French Rap songs because this is my guide and I'm the only one in charge here:

- A low-pass filter with a cut-off between 400 and 1000 hz will make an instrument sound muttered, distant, almost underwater. By using a low-pass filter with a rising envelope similar to the one shown in the example above, you can create the feeling of a melody progressively closer and clear, which is a great tool to help dynamize your intro and pre-chorus: PNL's song « 91's » has an intro that showcases this wonderfully.
- A high pass filter with a cut-off in the same range will reduce a melody's presence, giving it an almost claustrophobic feel. Releasing such a cut-off at the end of an intro will feel like a sudden breath of air and warmth, like in Ben PLG's « Demain ça ira ».
- Combining a low-pass filter in the low-thousands and a high-pass filter in the mid-hundreds (basically cutting both ends of the frequency spectrums) can make your melody feel like it comes out of an old radio or record, like the guitar at the beginning of Luv Resval's « AZNVR ».

LEVEL 5

MAKING A SONG

You now know how to create sections, but this doesn't tell you how to make a full song.

To get the coding out of the way, you can simply concatenate your sections using the `structure-maker` function. You input a list of sections, and an order list where you determine which sections should arrive after which others. If your section list is something like {intro, verse, chorus}, setting your order list to {1, 2, 3, 2, 3} will give you a classic intro-verse-chorus-verse-chorus song. Make sure you accurately specify the bpm and the number of measures of each section (which should be the same for every section). You can wrap the whole call in a `to-mono()` to make sure that no weird panning effects happen with the drums, and set `autonorm-off` (this is dangerous, please turn it back on afterwards) while multiplying your mix by a very low constant to avoid normalization-linked clipping at playtime. You should also call `effect-mixer` on all sections before putting them together, even the ones where you end up adding no effects, otherwise some alignments problems can occur within `structure-maker`.

```
; Turning off autonorming and giving the final mix a low volume
avoids clipping. Just don't forget to turn it back on!
exec autonorm-off()

; Put everything together

set full-song = to-mono(structure-maker(list(intro-1, intro-2,
 verse-1, verse-2, pre-chorus-1, pre-chorus-2, chorus-1, chorus-2,
 outro), {1 3 4 5 6 7 8 9}, 120, 4) * 0.03)

play full-song
```

The present guide comes with a full start-to-end composition called « Try2chip » that you can also play by running ChipGuide_Compo.sal, and many of the codes snippets you have seen so far directly come from this composition. Feel free to modify this composition as you wish to test some of this guide's functionalities!

Now, all of this doesn't really answer the question of how to make a full song. How do you pick a structure? How do you pick melodies? How do you make sure your composition is engaging? These questions do not have definite answers, and I definitely do not have them. I have been composing music for several years but my experience is mostly limited to rap instrumentals, metal and a bit of synthwave here and there. I have no academic formation nor any degrees in music, nor did I know that the major mode was also called « Ionian » before making this guide. To keep it short, please take my advice with a watermelon-sized grain of salt. But in the very uncertain event that my tips may actually help you, the next page is full of them.

TIPS

- In most genres, the capacity of a song to be engaging resides heavily in contrast. Your chorus will sound much fuller and louder if preceded by a quiet, minimalist pre-chorus — the reverse is also true. Keeping the intensity to the highest during a full song and not boring your listeners is not an impossible task, but it certainly is a difficult one.

- Don't get caught up on looking for the perfect melody or the perfect sound for hours. Starting with something « good enough » and progressively making it better while going along with the flow is faster, more satisfying, and the end result is usually cooler!

- Start with small and easy compositions: making only 30 seconds of engaging music is already a great exercise!

- If you like something in an actual song, try to do the same thing on your own to understand it better and re-use the trick. ~~Remember, it's not plagiarism as long as no one listens to it.~~

- Don't be afraid of repetition. Simple melodies, repeating structures help give a feel of familiarity to your song. There is obviously a balance to be found, but as an aspiring composer, your natural tendency may more likely be on the over-complication side. So don't shy away from simplifying!

- A similar tip to the previous one: avoid instrument overloading. If you feel like an element doesn't add something significant to your composition, it's totally ok and probably preferable to remove it.

- Don't set your bass too loud when you're mixing with headphones, or it's the only thing you will hear when you play your song on a speaker.

- Oddly specific to this guide, but don't over-use the random-melody function. Making one melody or two with it can be fun, but more will just sound incredibly messy

- Don't pay too much care to what is considered good or bad taste in your composition, or too what is considered easy or hard. Have fun composing, it's the best way to get better.

- If one of the functions of this guide does not work as expected, please never contact me because if you couldn't figure it out then I definitely can't. On the other hand, if you want to cover me with praise or, even better, send me some fun composition you made using this guide (I would genuinely love to hear it), you can find my email address on the CMU directory at ID hhelfgot.