Dynamic music and sound techniques for video games

2023-12-10 • edited • 8 mins read

#audio #foretrack #icarus #loops #music #playdate #sounds #yoyozo

The only aspect of game development I've not attempted myself is the music. I mostly use royalty free music of Japanese origin (just because I dig their vibe, man) as in the case of *Sparrow Solitaire* or *Fore! Track* or in rare cases I pay friends (like the amazing Jamie Hamshere) to write music specifically for a game as in the case of *YOYOZO*. Maybe one day that will change, but until then I'm enjoying gaining more understanding and control of the music in my games. Whilst I develop games for Playdate these techniques are general enough to apply anywhere.

The main way I make the music into more than a static track is to apply a dynamic, reactive, or adaptive effect in one way or another. In this blog post I'll go into how I've achieved this. Please note this is by no means an exhaustive list, rather it's just the ones I have personally used.

Dynamic BPM

I use this method in *YOYOZO* because it uses "chip tune" music data representing songs composed by my friend Jamie Hamshere using *Playdate Pulp*. A playback engine for this data, written by Pulp creator Shaun Inman, works beautifully when integrated into games written using *Lua* and the *Playdate SDK*. I added hook to allow me to set the BPM at any point to any value. The end result is that the BPM of the music scales from 130 to 135 as your score increases. As you improve at the game you'll notice the music speed up ever so slightly along with an increase in tension and anxiety.

Of course, it's possible to do this in a pre-recorded song stored as a digital music file, but it's much more difficult for that to respond to the what the player does in the game. An example that takes an interesting approach to this is the track "Sunny Day" from the game Vib Ribbon, and indeed the rest of its soundtrack, where tempo changes over the duration of each song.

Infinite Variations

Another technique I use in *YOYOZO*, again made possible because I can modify the music data and playback parameters in real time. With this one I cycle the values for the instrument voices pseudo-randomly so that the track plays once as it was programmed and then morphs slightly for each subsequent playback. The track is quite minimal and repetitive in Steve Reich, Philip Glass or John Adams sort of way, so there are automated variations wandering around the original arrangement work really well. Perhaps the ultimate implementation of this approach is *Wii Play's* Tanks game.

For sound effects, I vary the playback sample rate to change the pitch of sound effects. This prevents the same sound effect becoming monotonous. Two examples might be *Lara Croft* in the first *Tomb Raider* game, groaning the same way every time she climbs up a platform compared with the rich variety of sounds when *Mario* walks on different surfaces.

Blending/Fading/Balance

Another idea I had was to fade or blend two tracks as the player makes progress in the game. But how to find two tracks that can be cross-faded in a way that always makes sense? Of course you have them composed, but what about in music that already exists? If only there was an easy way to find such tracks!

There is: stereo pairs! You'd be surprised at how different the left and right channels can sound whilst obviously being the same tune. Of course this means that are output audio will be mono but for me on *Playdate* that's just fine. I use this method in *Fore! Track*.

The idea is to adjust the balance of the two parts of the audio, at once point you're playing just the left audio across both outputs, then you adjust the balance to play a mix of both, at the other end of the scale you'd be playing just the right audio. Unfortunately the *Playdate SDK* currently has no API to easily adjust balance, so I had to program a method myself. First, I convert to the destination format which is for me ADPCM using adpcm-xq. Once the files are in this format I split the stereo pair into two files, one for the left channel and one for the right channel. Converting to the destination format before splitting ensures that the two are exactly the same length in terms of samples/bytes.

In the game I fade between the two as the score/chain increases, which has the effect of subtly changing the instrumentation of the tune. It's one of those things that most people wouldn't notice, but that once you know about it you can't miss it. Sadly, I don't have an easy way to demo this in a video or sound file.

For sound effects, you might consider panning to increase immersion and guide the players visual focus through use of audio. In *YOYOZO* I pan certain sounds relative to the location of the ball, certain other sounds relative to the location of the player, and there are global sound effects that are not panned at all.

Progressive Loops

Digital audio is a different beast. I always try to find a time that fits the game, going so far as to audition many hundreds of tracks and creating playlist of songs far ahead of ever making a game or even having and idea for a game. I try to find tracks that will loop well and not get annoying, which is easier said than done. If I can't find a track that loops well, there's another way.

You can use PyMusicLooper to analyse a digital audio track and spit out information about ranges that loop nicely, along with a percentage indicating how good it considers the loop. In other words you can

identify and extract a loop from digital audio files that sound like they could loop. Of course, can't identify loops in tracks that aren't repetitive or consistent in their structure. You might get *PyMusicLooper* to split the file into into three sections (intro, loop, outro) or just export the loop information as a text file to use in your game. Which I choose depends on how much of the file I want to use.

For an example, in my game ICARUS I'm using a file that gives the vibe I wanted in the game and sounded like it contained some loops even though it was not provided as a looping song. *PyMusicLooper* reported that it contains a dozen or so possible loops of varying quality.

LOOP	START	END	DURATION	MATCH
0	**3.437**	**30.755**	**27.318**	**94.87%**
1	0.023	27.341	27.318	94.79%
2	7.279	34.598	27.319	94.63%
3	6.850	34.168	27.318	93.95%
4	8.127	35.445	27.318	93.92%
5	22.221	52.953	30.732	93.80%
6	25.635	56.366	30.731	93.23%
7	11.970	39.288	27.318	93.17%
8	6.850	35.875	29.025	92.13%
9	**6.850**	**54.660**	**47.810**	**91.98%**
10	20.515	52.953	32.438	91.54%
11	10.263	37.581	27.318	91.26%
12	22.221	54.660	32.439	91.11%
13	23.928	68.325	44.397	90.92%
14	39.300	70.031	30.731	90.82%
15	**12.399**	**70.449**	**58.050**	**90.78%**

My goal was to find three loops of increasing length and with a high percentage loop quality. After some experimentation and listening, I decided on loops 0, 9, and 15 (table only shows the top 15 loops from this track, even though their percentage loop match are not 100% they still sound like good loops, so selecting these loops was a case of finding three of suitable length and content. *PyMusicLooper* will let you audition the loops directly, so there's no need to use an audio editor.

Using the *Playdate SDK* I can do `setRange()` on the audio track to change the playback range and the music will loop between those new points when the playhead reaches the end of the range. For this reason, this method does not provide immediate results so is better used to signify a large change in progress as the delay until the change is noticed will be an unknown amount of time. But when the change does kick in it's a really nice surprise!

The final result sees the game start by playing loop 1 (synth and drums) and then as the player gets makes some good progress I switch to loop 2 (synth, drums, guitar licks), and finally as they pass a

certain threshold I switch to loop 3 (synth, drums, guitar licks into guitar solo). This provides music that sounds very dynamic with little effort. You could even drop back to the shorter loops if the player lost a life, missed a target, and so on. Again, there's no real way of me demoing this as it's something that will become apparent through play, and the final result is just more of the song!

For sound effects I use the same approach as above. As an example, in *Fore! Track* there is a clapping sound effect after the player gets the ball in a hole. This is a long sound effect but I play three increasingly long sections of it as the player's chain increases (number of successive holes-in-one). It starts off as a short clap, increases to a longer more enthusiastic clap, and finally it starts with a whoop and continues to enthusiastic clap. I have a separate sound effect for the end game cheer that plays over the top of the full clap, resulting in a raucous end of game celebration.

I'd love to hear about other methods of achieving dynamic music and sound in video games. Feel free to reach out to me on social media!

Further reading

- *YOYOZO* (or, how I made a *Playdate* game in 39KiB)
- Easter egg emoji: converting pixels into particles
- Dynamic music and sound techniques for video games

Elsewhere

- 2023-11-22 — Hacker News
- 2023-11-22 — Tildes

--

Originally published: 2023-12-09

--

Comments: @gingerbeardman

PREVIOUS POST

Easter egg emoji: converting pixels into particles

tag cloud: 1bit 1bitwoodblocks 3ds 60fps amiga animation apple appletv appstore arcade archaeology art artist atarist audio automation bandkun basiliskii becker cars controllers credits critique dailydriver deneba dreamcast dsiware electronics emulation extension famicom foretrack game gamedev gameplay gba golf graphics guide hack hanafuda hardware history hypercard internetarchive ios itchio japanese koei koikoi macintosh macos mahjong maps marchintosh markdown medium megadrive midjourney mmm mod modes msx music nes nintendo nintendo64 nintendods openscad palmos patreon pc pc98 physics piece pixelart playdate playstation polarium preservation proxy recommendations rendering reverseengineering review rss save scans script sfx snes software solitaire sonyreader sparrowsolitaire system7 thoruyamamoto translation tv twitter ultrapaint ux videogame videogames wii wiiware windows workflow wwdc x68000 yoyozo