Making an enthusiast-grade rhythm game in Godot without looking like a fool

By EIREXE

EIRTeam

Who am I

- I’m Álex Román AKA EIREXE, I’m from Spain, I work at a self funded indie studio called EIRTeam
- I make a rhythm game called Project Heartbeat, and I am working on a new game that I will reveal soon, also in Godot
- I am very very stupid, so take everything I say with a gigantic grain of salt
- I used to work at a racing simulation company, I now work independently

EIRTeam

What this talk is about

- Summary of most struggles I found while working on PH (yes that's the official acronym)
- This includes things that may not only apply to rhythm games, but other general things
- I also talk about some engine customizations, these are usually PR'd so check if they have been merged (so far none of these have)
- This is about making an enthusiast grade rhythm game AKA one people would actually play at the arcades, if you are simply making a music game going to certain extents to reduce latency is unnecessary

EIRTeam

Let’s make a rhythm game!

- But:
  - We are lazy
  - We have no money
  - We have no music licenses
  - We suck at designing games
  - We are not a studio (I’m the only person to work on this game)
  - Of course we are not:
    - SEGA
    - Bemani
    - Harmonix/Viacom/Whoever owns Guitar Hero now
  - We are Spaniards :)

EIRTeam

Let’s make it a clone game!

- Clone games are a tradition, so why not do it again?
- Commercial clones can be succesful! See:
  - League of Legends (DOTA/AoS)
  - Stardew Valley (Harvest Moon)
  - Rockband (Guitar Hero)
  - Candy Crush (Bejeweld)
  - Osu! (Ouendan)
  - The various GTA clones

EIRTeam

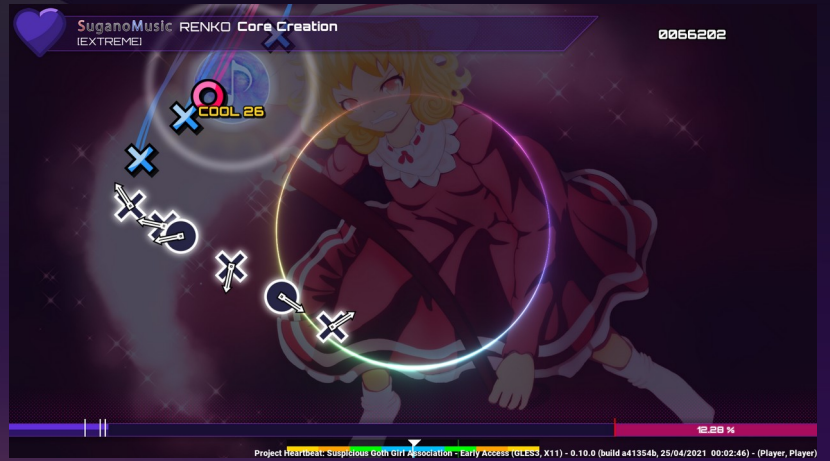
We need a redeeming factor

- Of course making a clone won’t sell, it can’t be a 1:1 copy (that would also be grounds for litigation)
- We need something to make our game unique and have people play it instead of the game we are cloning

EIRTeam

Project Heartbeat TL;DR

- Rhythm game with heavy focus on UGC
- It’s a Project DIVA clone
- Notes come flying off the screen and the user presses them when they are over the “target”
- We don’t have the buget SEGA does, so we have to not have background animations like they do
- Made some changes to mechanics to make more sense

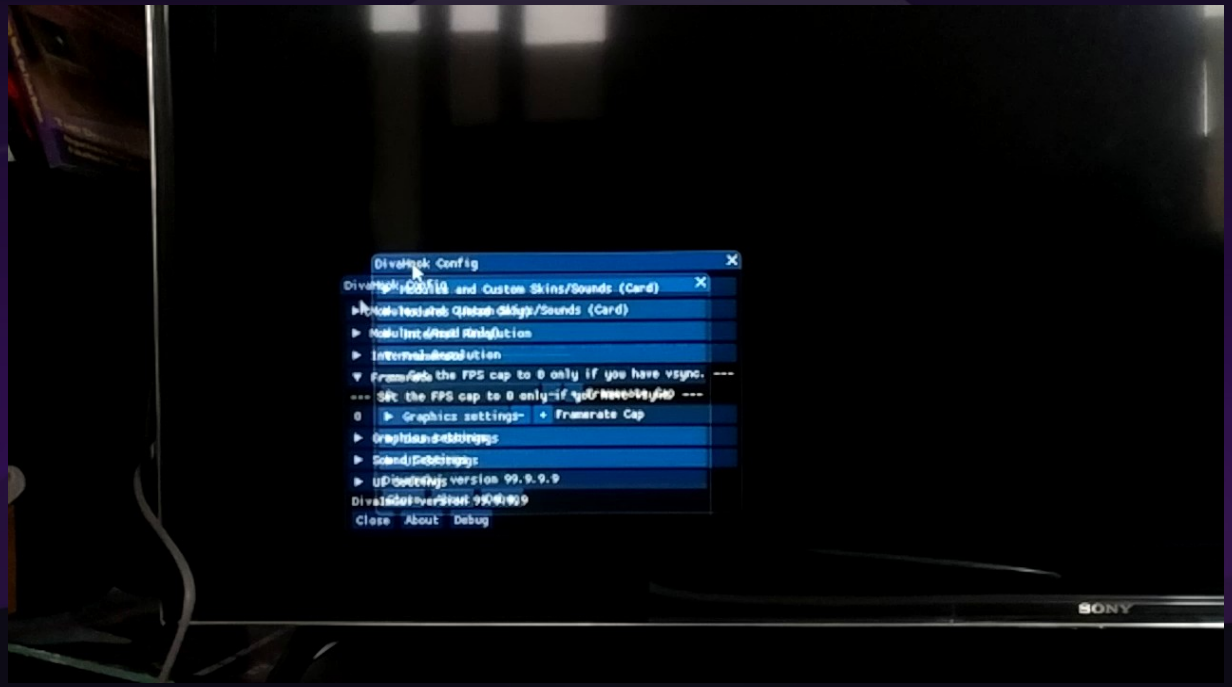


NVIDIA are morons

- NVIDIA Optimus drivers have latency and frame pacing issues with OpenGL, only way to solve it is extremely hacky
- This involves rendering the game offscreen in opengl and presenting it using DX (this is how the project diva community fixed the issue)
- Another requirement is to use exclusive fullscreen to bypass the compositor :(
- Clearly this a pretty big deal for Rhythm Games

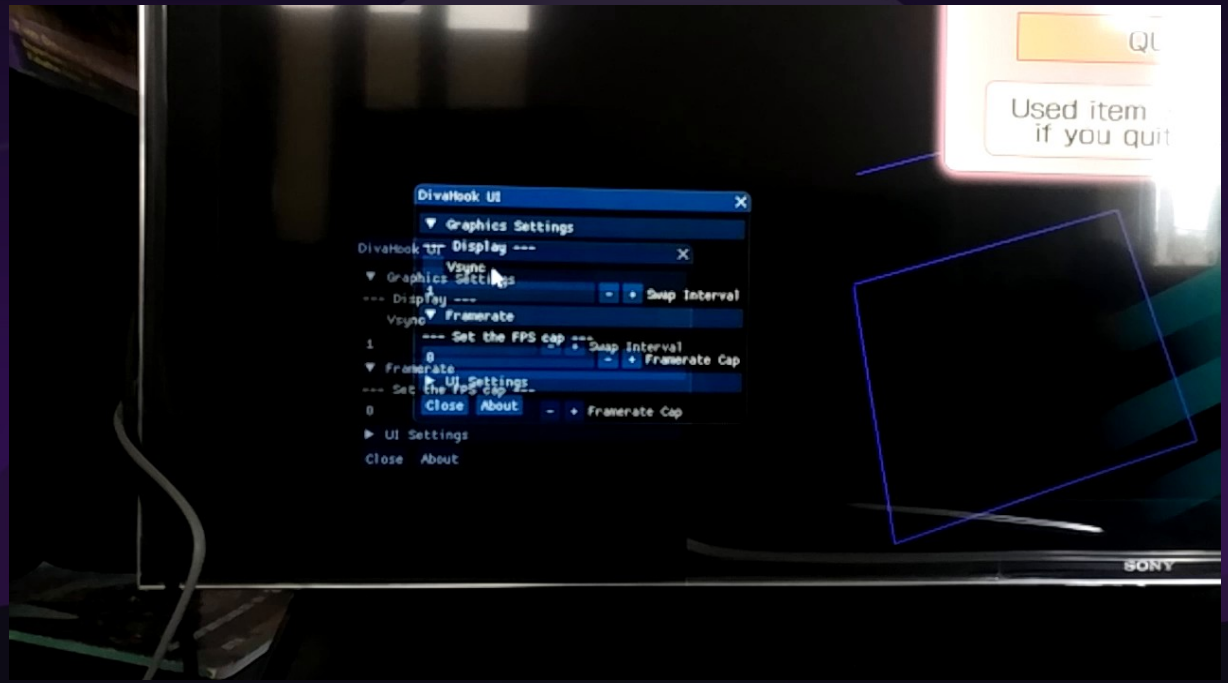
EIRTeam

Don’t believe me? Check this out



EIRTeam

Now, check OpenGL



Might as well use ANGLE

- AMD and NVIDIA are so utterly incompetent their OpenGL drivers on Windows are a disaster of biblical proportions for different reasons (AMD’s performance vs NVIDIA’s noncompliance), so one might as well use ANGLE and kill two birds with one stone

EIRTeam

Integrating ANGLE

- ANGLE is an OpenGL implementation with various backends including Vulkan and DX11 amongst others.
- Already done for UWP, for desktop port, check this PR
- Works fine, includes some fixes by iFire to make it work 1:1 to native OGL
- Can counter-intuitivelly give a big free performance improvement
- … Or you can use the mesa OGL to DirectX 12 driver, that works too (use mesa_glthread=true), this also somehow lowers power consumption massively, need to investigate more

EIRTeam

Latency shenanigans

- PH needed two things:
  - To be able to have low latency (because we make noises on button presses)
  - To be able to predict output latency to compensate for it

EIRTeam

How digital audio works

- Digital audio is stored as samples, might be interleaved or not, but usually should be (LRLRLRLR layout vs LLLRRR)
- In Godot, OGGs work this way
- Chunks or groups or samples are sent to the sound hardware
- Add to this that godot mixes chunks on another thread and you got a recipe for disaster
- And let’s not even talk about display lag

EIRTeam

How 2 predict when our sound will play

- Godot audio is multi threaded, so we can predict when it will next be mixed (sent to the sound card) by using AudioServer.get_time_to_next_mix()
- You can get the output latency from the audio driver by using AudioServer.get_output_latency()

EIRTeam

Output latency target

- In the Project Settings the output latency option gives Godot a target size for mixing chunks, on Windows this is dictated by WASAPI, so it will pick the next biggest latency setting it can
- This is slightly broken as of 3.3… more on that later

EIRTeam

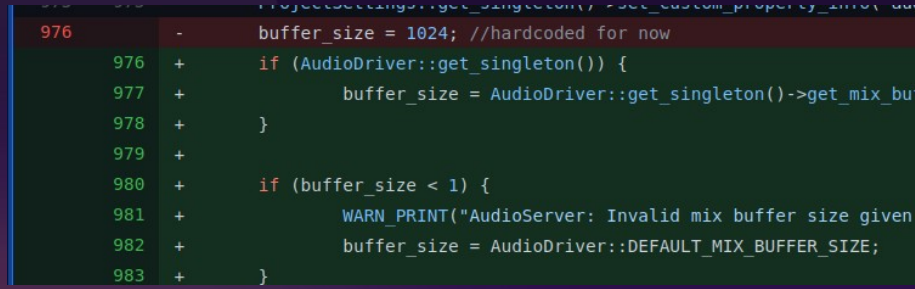
Syncing methods

- Check the official documentation page on this
- TL;DR it’s a software vs hardware clock situation
  - Software clock drifts but it will be more accurate (and might be outright broken on some systems for some reason, particularly on those with low core counts or under heavy load)
  - Hardware clock won’t drift but it might be less accurate
  - I’d recommend using the Hardware clock method if you can get away with it
  - Give your users an offset setting to change how early/late the notes are.

EIRTeam

Things that are broken in godot atm

- Godot's internal buffer doesn't change! //hardcoded for now… or forever!
- WASAPI has a new low latency mode on Windows 10, so let's use this (exclusive mode is the next best thing, but users will bite if you use this since only the game can then output audio)
- See benjarmstron's PR1 and PR2 for fixes to this, thanks!



EIRTeam

Video playback

- Godot’s video decoding is deprecated, use godot-videodecoder with FFMPEG (but don’t use H264 or MPEG-LA will bite your ass)
- Godot’s stock video decoding code might not be ideal, since you can’t seek while paused this can lead to desync on modest systems, use this PR
- I recommend seeking BEFORE you make the stream start playing, that way the decoder has time to catch up.

EIRTeam

Menu and serialization system

- A system for menus and transitions was written, this makes use of the power of Godot nodes to make a menu that can be used with both controller and keyboard+mouse
- PH uses a custom JSON-based format for… everything, this includes a custom serialization and Deserialization system
- These two components are public as part of the EIRTeam Game Framework, which you can get here

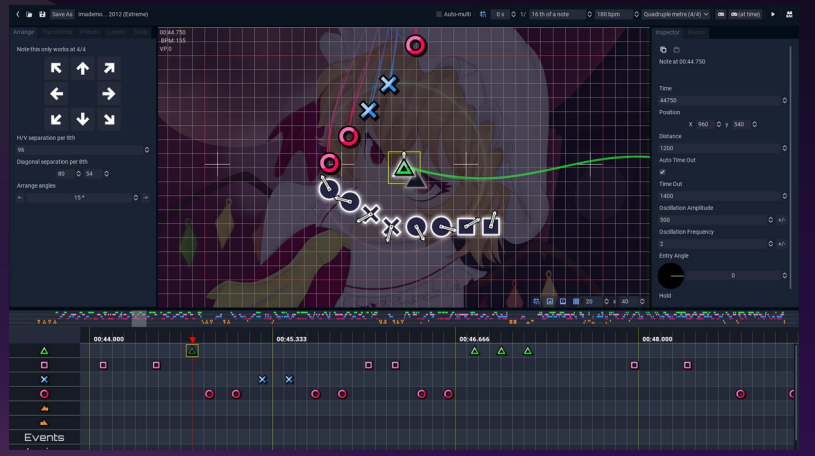
Tie everything to timing

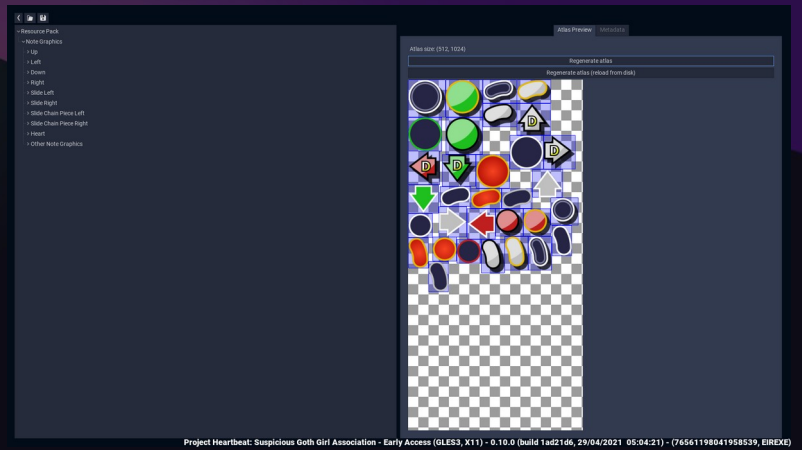
- I see a lot of newcomers to rhythm game development make things such as note hit detection be based on sprite collisions, these just aren't the right tools for the job, they aren't robust enough
- You need these things if you wanna be taken seriously as an enthusiast rhythm game

```
func _process_game(_delta):
	_sfx_debounce_t += _delta
	var latency_compensation = UserSettings.user_settings.lag_compensation
	if current_song.id in UserSettings.user_settings.per_song_settings:
		latency_compensation += UserSettings.user_settings.per_song_settings[current_song.id].lag_compensation

	if audio_stream_player.playing and (not editing or previewing):
		if UserSettings.user_settings.timing_method == HBUserSettings.TIMING_METHOD.SOUND_HARDWARE_CLOCK \
				or UserSettings.user_settings.timing_method == HBUserSettings.TIMING_METHOD.SOUND_HARDWARE_CLOCK_FALLBACK:
			var t = audio_stream_player.get_playback_position() +  - AudioServer.get_output_latency()
			if UserSettings.user_settings.timing_method == HBUserSettings.TIMING_METHOD.SOUND_HARDWARE_CLOCK:
				t += AudioServer.get_time_since_last_mix()
			t *= audio_stream_player.pitch_scale

			if t > time:
				time = t
				time -= latency_compensation / 1000.0
			time = max(time, 0)
		else:
			# Obtain current time from ticks, offset by the time we began playing music.
			time = (OS.get_ticks_usec() - time_begin) / 1000000.0
			time = time * audio_stream_player.pitch_scale
			# Compensate for latency.
			time -= time_delay

			# User entered compensation
			time -= latency_compensation / 1000.0

			# May be below 0 (did not being yet).
			time = max(0, time)



		if not editing:
			var end_time = audio_stream_player.stream.get_length() * 1000.0
			if current_song.end_time > 0:
				end_time = float(current_song.end_time)
			if time*1000.0 >= end_time and not _finished:
				_on_game_finished()
	for i in range(timing_points.size() - 1 - (timing_points.size() - last_hit_index), -1, -1):
		var group = timing_points[i]
		if group is NoteGroup:
			# Ignore timing points that are not happening now
			var time_out = group.precalculated_timeout
			if time * 1000.0 < (group.time - time_out):
				break
			if time * 1000.0 >= (group.time - time_out):
				_process_note_group(group)
```

Making tools

- Godot is great for tools (the editor itself is made using mostly the same API we have in GDScript)
- The API's reflection capabilities make it ideal to make tools with
- I recommend reading the engine's code for inspiration





EIRTeam

If you had an unlimited budget

- A custom Input stack with timestamped inputs would be a good point to start from, would be relatively simple to implement in Godot
- I heard Miniaudio (a library) might yield less latency than Godot, but I don’t really buy this

EIRTeam

Using other engines

- While this whole talk might have seemed like I’m dissing Godot, I have to say Godot isn’t any worse than the general-purpose alternatives at being a rhythm game:
  - High quality Unity rhythm games use their own input and audio stacks anyways
  - Clone hero devs reported needing C++ code for I/O performance, yuck
- GDScript isn’t as catastrophically slow as you might be lead to believe by people who don’t like it, particularly since heavy lifting is done in the C++ code, this is even better with statically typed Gdscript in 4.0

EIRTeam

Summary

- I think godot is a great tool to make rhythm games with, these kinds of games are technical by nature, so a lot of effort has to be put into making them work nice, and they are not simple
- Things that would be ugly native appendages in other engines (such as audio stack changes) in Godot can simply be done by editing the code, which is a plus

EIRTeam