

NEON BEAT

A Free Blind Test game
from A to Z:
when FOSS animates
your parties !

Maxence Fiorina
Raphaël Roy
Alexis Lothoré



What is Neon Beat ?

- a blind test game for parties with friends
- a multi-disciplinary hobbyist project
- a still-evolving implementation
- a completely open-source system
- actually a new revision, following multiple iterations

The team



Raphaël : Backend



Maxence : Frontends, NBPC



Alexis : Buzzers, Controller

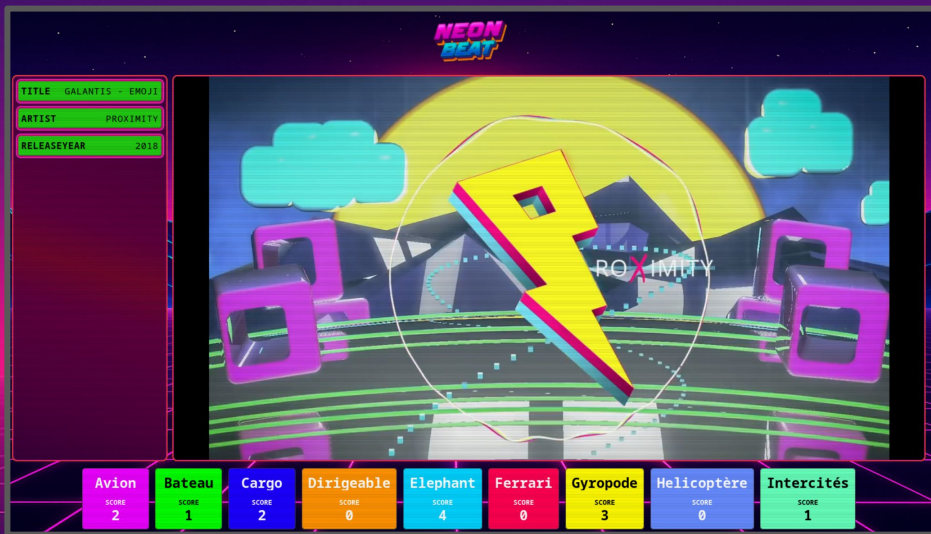


Benjamin : V2 buzzers casing



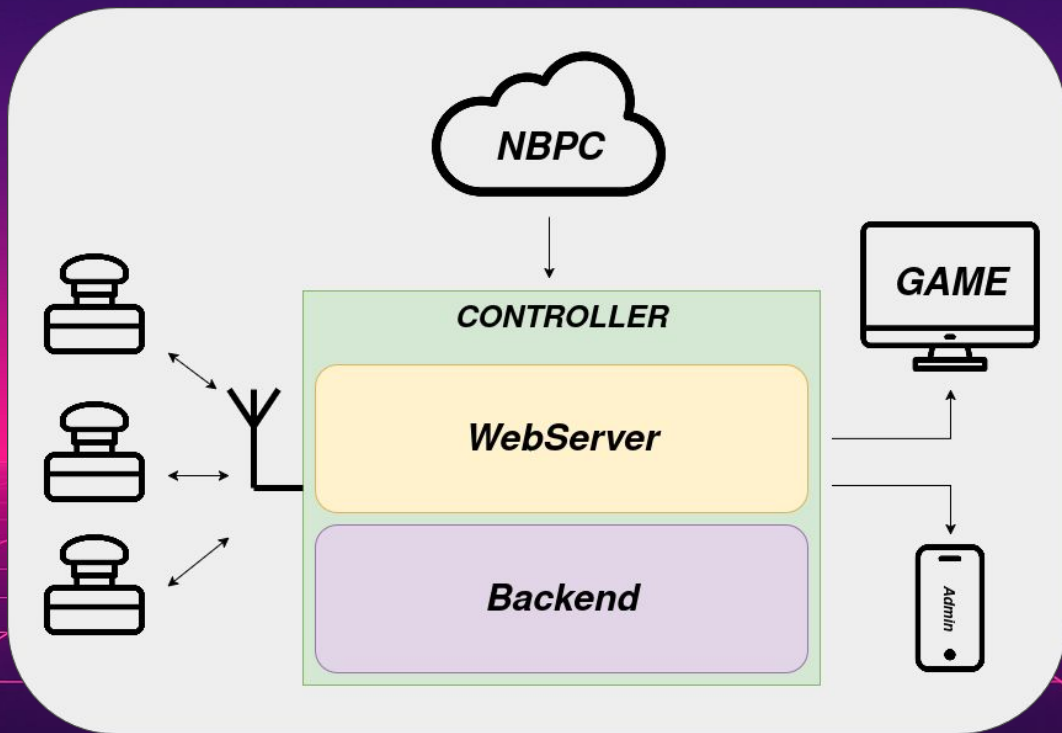
Nicolas : OST

Basic features



- Game configuration:
 - playlist
 - teams: names, scores
 - buzzers pairing
- Players interface: scores, song in progress, buzz notifications
- Controlled by a "game master" on a dedicated interface

Internal architecture



The game controller



- Raspberry Pi 4 + USB WiFi dongle
 - before: Raspberry Pi OS + ansible
 - now: custom Buildroot-based distribution
- Hosts all technical features needed for game handling:
 - wireless access point for buzzers
 - web server: players page, game master page
 - game engine (backend)
 - database: CouchDB
 - (players page display over HDMI, coming soon)



The core: the game engine

- Finite state machine
- Real Time communication : WebSocket (buzzers) & HTTP (frontends) with REST + SSE
- Decoupled from the selected database (CouchDB, MongoDB)
- Reliability (developed in Rust, auto-reconnect, clean shutdown, degraded mode)



Documentation

neon-beat-back 0.8.3 OAS 1.1 openapi.json	
health	Health check endpoints
sse	Server-sent events streams
GET	/sse/admin Stream admin-only events, establishing or validating the admin token.
GET	/sse/public Stream realtime public events to connected frontends.
buzzers	WebSocket operations for buzzer devices
game	Game bootstrap operations
admin	Administrative controls for running games.
POST	/admin/game/answer Validate or reject the currently submitted answer.
POST	/admin/game/end Mark the game as finished and perform cleanup.
POST	/admin/game/fields/flag Flag a point or bonus field as discovered for the current song.
POST	/admin/game/next Advance to the next song in the running game.
POST	/admin/game/pause Pause the current game flow, freezing timers and buzzers.
POST	/admin/game/resume Resume a previously paused game.
POST	/admin/game/reveal Explicitly reveal the current song's answer to participants.
POST	/admin/game/start Begin a game session and publish the first song to admins.
POST	/admin/game/stop Stop the game early and return final team standings.
GET	/admin/games Retrieve all games known to the system for administration purposes.
POST	/admin/games Generate a game using an existing playlist as the source material.
POST	/admin/games/with-playlist Create a bespoke game definition under admin control.
GET	/admin/games/{id} Retrieve a game by its ID.
DELETE	/admin/games/{id} Delete a persisted game by its identifier.
POST	/admin/games/{id}/load Load and activate a stored game for continued play.
GET	/admin/playlists Retrieve playlists eligible for generating new games.

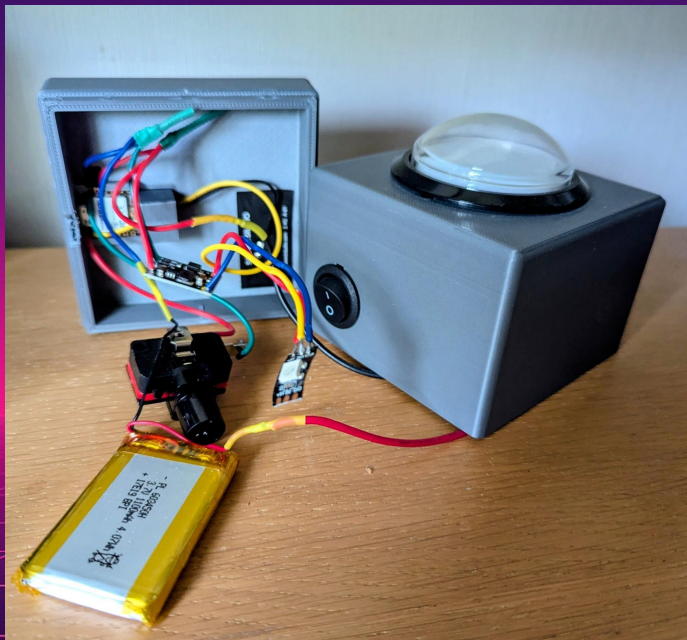
- Crucial support for :
 - discussions and draft for new features
 - proper documentation for an open source project
- Needed to properly defines frontiers with other system components: buzzers, players page, admin page
- Multiple parts:
 - OpenAPI/Swagger: self-generated REST API
 - Markdown: communication protocols
 - Mermaid: markdown-compatible diagrams



The players view

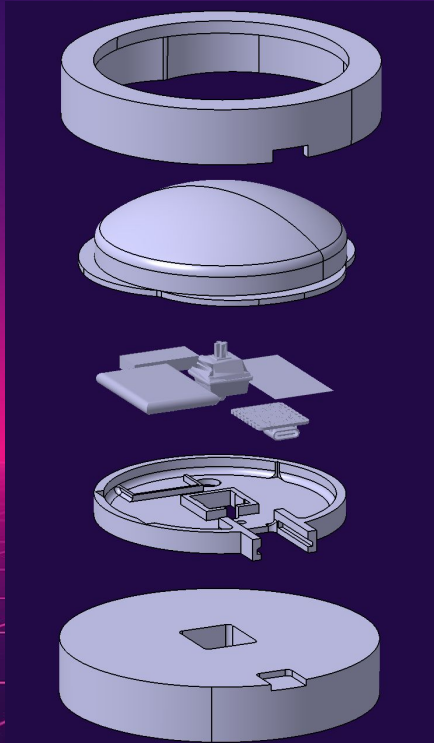
- Single Page Application (SPA) built with [React.js](#)
- Embedded Youtube iFrame
- Stateless: it is mirroring the game engine state
- Song countdown handled on client side

Buzzers



- esp32/esp32c3
 - C-based firmware (Rust rewrite in progress)
 - based on esp-idf
- 3.7V 320mAh LiPo battery
- "arcade" button
- on/off switch
- ws2812 led (serial protocol)
- 3d printed casing
 - designed with FreeCAD

V2 buzzers

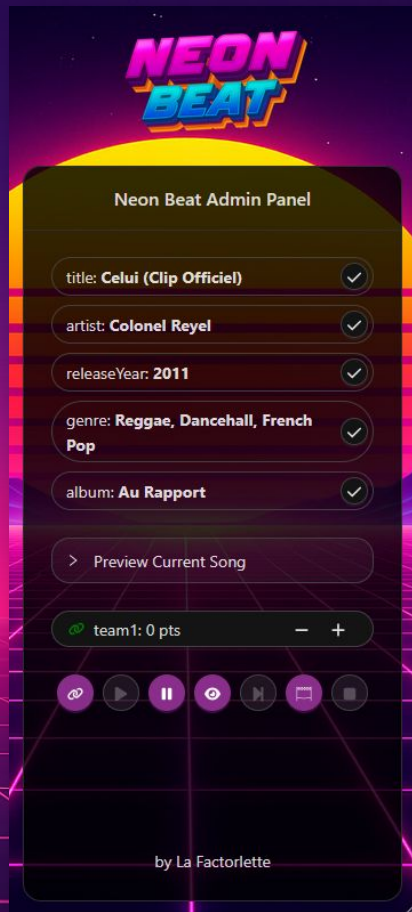


- Wooden casing, painted, varnished
- Buzzer cap casted in epoxy resin
 - epoxy resin + mica powder mix
 - 3d printed parent mold
 - Final mold in silicone
- 3D printed internal body
 - evolutionary design: the internal body can be swapped
 - coming soon: custom electronic board



The admin view

- Single Page Application (SPA) built with [React.js](#)
- Full control on the game state
- Mobile-first interface
- Playlist import based on JSON format





Game preparation: Neon Beat Playlist Creator

- Single Page Application (SPA) built with [React.js](#)
- Relies on API Youtube (registered app)
- Youtube playlists fetching
- Any song item (used as answers or hints) can be customized
- Song metadata can be enriched with LLMs
- Genericity thanks to OpenAI standard interface: compatible with multiple LLMs
- Playlist exported in JSON, to be imported in the admin interface

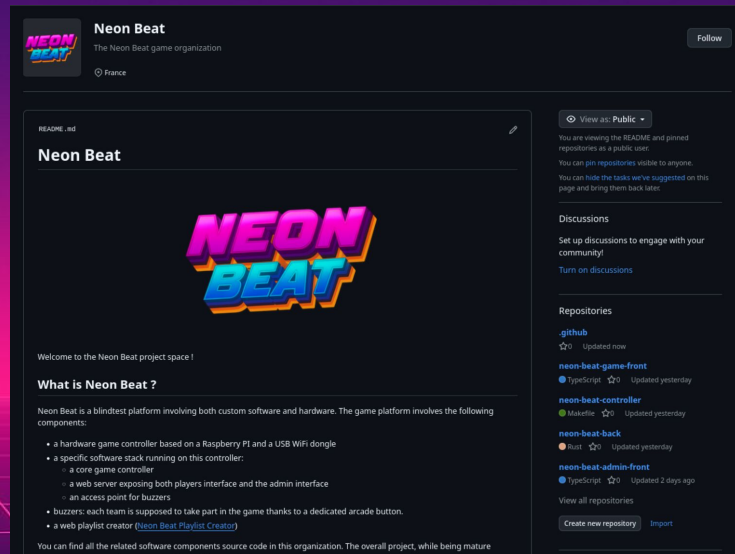


Next steps

- V2 buzzers
- new songs/playlists sources in NBPC
- virtual buzzers (smartphones)
- generic game mode: quiz mode
- dedicated hardware console for the game master !
- any suggestion welcome !



<https://github.com/neon-beat>





Ready for a demo game ?

Capitole du Libre