Τεχνολογία Πολυμέσων

2η Προγραμματιστική Εργασία
"SimpleSynth"
!!!ΟΛΟΚΛΗΡΩΘΗΚΕ!!!

Γεώργιος Καρέττας (p3130087@aueb.gr)

Περιεχόμενα.

Οδηγίες Χρήσης:

ΣΥΜΒΑΤΟ ΜΕ FIREFOX, CHROME

Δείτε την εφαρμογή:

1. Κατευθυνθείτε στην ηλ. διεύθυνση https://gtkall.github.io/SimpleSynth/

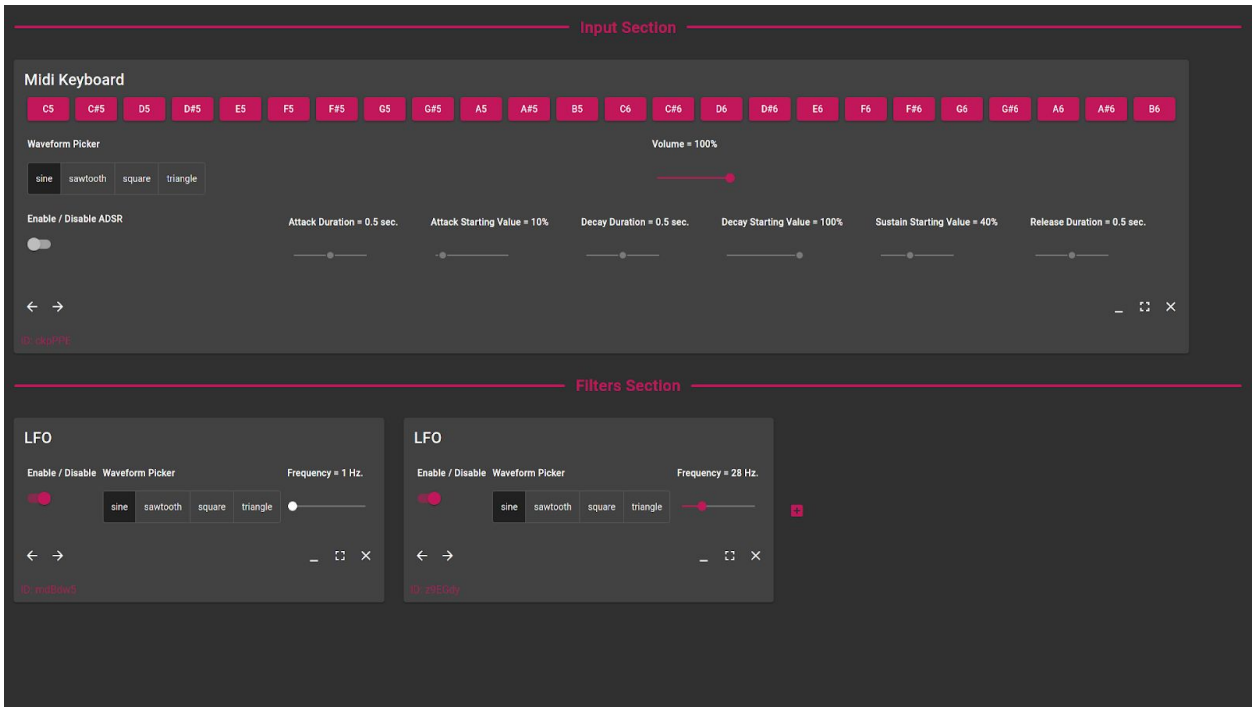
2. Επιλέξτε Input (Midi Keyboard ή Oscillator)
3. Επιλέξτε φίλτρα
    a. Μπορείτε να προσθαφαιρέσετε φίλτρα και να αποκρύψετε τις ρυθμίσεις τους, αν θέλετε, μέσω των πλήκτρων ελέγχου κάτω δεξιά σε κάθε παράθυρο
    b. Μπορείτε επίσης να τα μετακινήσετε αριστερά ή δεξιά στην σειρά, με real-time εναλλαγή και ενημέρωση του rack.
4. Πατήστε ένα πλήκτρο από το Midi Keyboard ή εκκινήστε τον Oscillator

Δείτε / τρέξτε τον κώδικα:

1. Κατευθυνθείτε στην ηλ. Διεύθυνση https://github.com/Gtkall/SimpleSynth
2. ΣΗΜΑΝΤΙΚΟ!!! Επιλέξτε το branch “release”
3. Διαβάστε το README.md για περεταίρω οδηγίες

Σενάρια Χρήσης:

1. MIDI πιάνο με “παλμό”:

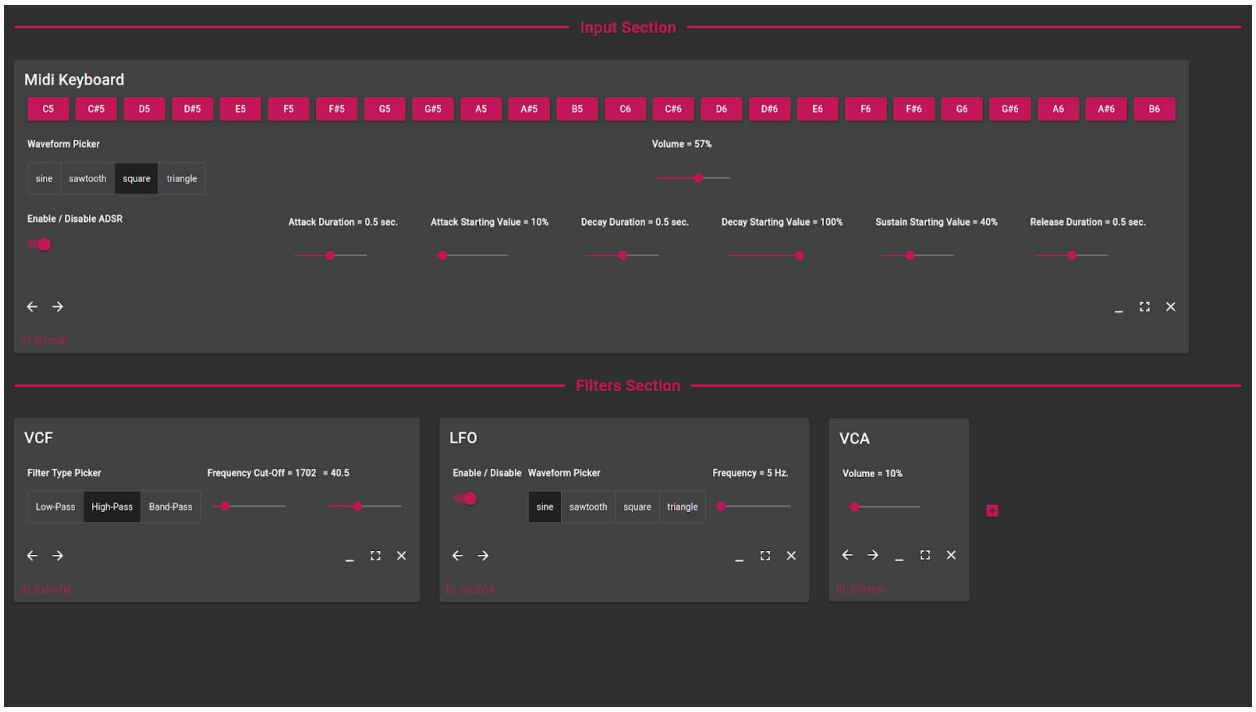


Θα χρειαστείτε:

- Input: Midi Keyboard
- Filters: LFO με μικρή συχνότητα → LFO με μεγάλη συχνότητα

Ο ήχος που παράγεται μοιάζει με απόκοσμο ήχο διαστημοπλοίου από ταινία των ‘60s.

2. Έναν “τρομακτικό” midi piano ήχο



Θα χρειαστείτε:

- Input: Midi Keyboard με ενεργοποιημένο το ADSR
- Filters: VCF high-pass → LFO με μικρή συχνότητα → VCA για την ένταση
- *(Ακολουθήστε την φωτογραφία για ακριβείς συχνότητες)*

Ο ήχος μοιάζει με ένα τρομακτικό synth πιάνο που παίζουν στις κλασικές ταινίες τρόμου.

Τεκμηρίωση:

Feature Case Study 1: Dynamic Window System

Για το “modular” κομμάτι του “Modular Synth” χρειάστηκε να υλοποιηθεί ένα window system το οποίο θα ήταν πλήρως agnostic ως προς το περιεχόμενο του, καθώς και θα παρείχε τα απαραίτητα references προς τα πάνω στην ιεραρχία, για περεταίρω επεξεργασία.

Υλοποίηση:

Για την υλοποίηση χρησιμοποιήθηκε το Dynamic Component Creation κομμάτι της Angular. Συγκεκριμένα, το **SynthesizerComponent** περιέχει ως παιδί του στο view ένα **WorkspaceComponent**, στο οποίο και δίνει την λίστα με τα φίλτρα τα οποία τελευταίο θα μπορεί να δημιουργήσει. Στην συνέχεια, όταν ο χρήστης επιλέξει ένα φίλτρο, δημιουργείται ένα παιδί τύπου **WindowComponent** το οποίο περιέχει το δυναμικά δημιουργημένο φίλτρο.

Το **WorkspaceComponent** δέχεται events κλεισίματος ή αλλαγής θέσης των **WindowComponent** που βρίσκονται μέσα, και το ίδιο παρέχει την -κάθε φορά- ανανεωμένη λίστα με component references στο **SynthesizerComponent**.

Στη συνέχεια, το **SynthesizerComponent** χρησιμοποιεί μεθόδους της abstract κλάσης στην οποία όλα τα φίλτρα υπακούν, για να τα “ενώσει” το ένα με το άλλο.

Feature Case Study 2: The “MultiAudioNode abstract class

Για να είμαστε σίγουροι πως όλα τα φίλτρα ενώνονται μεταξύ τους σωστά ανά πάσα στιγμή, κάθε φίλτρο και είσοδος είναι παιδιά της abstract class **MultiAudioNode**.

Υλοποίηση:

Η κλάση αυτή περιέχει 2 analyzer nodes, τα οποία ονομάζονται _inputNode και _outputNode. Επίσης, η κλάση παρέχει getters και setters. Το **AudioNodeLike** είναι ένα interface που υλοποιεί η abstract class **NodeLikeComponent**, και παρέχει τις δηλώσεις των μεθόδων *connectInputTo(node)* και *connectOutputTo(node)*.

Η **NodeLikeComponent** είναι κλάση-γονέας όλων των components που φτιάχνουμε, και οι μέθοδοι της είναι που χρησιμοποιούνται από το **SynthesizerComponent** για να ενώνονται τα components.

Τεχνολογίες που χρησιμοποιήθηκαν:

Για την υλοποίηση της εργασίας χρησιμοποιήθηκε το Web Audio API.

Λογισμικό που χρησιμοποιήθηκε:

Για την υλοποίηση:

- Angular 11
- Typescript 4
- Angular Material 11
- Git

Περιβάλλον Ανάπτυξης:

- Microsoft Code
- Windows 10
- Firefox 83 / Chrome 86

Hosting:

- Github Pages

Πηγές Πληροφόρησης:

Η πληροφόρηση μου κατά την ανάπτυξη έγινε χάρη στην ευγενική χορηγία της Mozilla Foundation, που μας παρέχει με το καταπληκτικό MDN.

Όσον αφορά το framework, Η Google και η κοινότητα της μας παράσχει λεπτομερείς πληροφορίες και οδηγούς για κάθε στάδιο ανάπτυξης του κώδικα μέσω Angular, στο angular.io.

Προβλήματα που παρουσιάστηκαν:

> Ένα πολύ σημαντικό πρόβλημα που παρουσιάστηκε έχει να κάνει με την αδυναμία του Web Audio API να παρέχει ολοκληρωμένες λύσεις για time-based audio sequencing.

Συγκεκριμένα, υπάρχει ένα bug στο ADSR του midi piano, το οποίο λογικά έχει να κάνει με τους κύκλους της μηχανής της Javascipt.

Η λύση που χρησιμοποιήθηκε ήταν το να χρησιμοποιηθεί η *setTimeout()* σε χρόνο υπολογισμένο από το Audio Context που παρέχεται από το Web Audio API. ωστόσο, όπως θα παρατηρήσετε, υπάρχουν audio bugs που δημιουργούνται λογικά από την διαφορετικού χρόνου επίλυση της *setTimeout()* σε σχέση με τις μεθόδους που χρησιμοποιούνται για audio sequencing (όπως η *linearRampToValueAtTime()* που ανήκει στο Web Audio API). Παρόλα αυτά, η λύση δεν πετυχαίνει στο 100% των περιπτώσεων.

> Ένα άλλο πρόβλημα έχει να κάνει με ένα γνωστό bug της Angular. Συγκεκριμένα, η δυναμική δημιουργία instance ενός component μέσω factories δεν μας αφήνει να περάσουμε είσοδο στον Constructor. Επιπλέον, αν περάσει η πληροφορία ως assignment σε input field αμέσως μετά, η Angular δεν προλαβαίνει να χτίσει το lifecycle hook, καθώς γίνεται ασύγχρονα.

Η λύση που χρησιμοποιήθηκε είναι να περιμένουμε για μισό δευτερόλεπτο μέσω της *setTimeout()*, και μετά να περάσουμε το input που χρειάζεται για το initialization του οποιουδήποτε component

(Σημείωση: το συγκεκριμένο bug βρίσκεται αρκετό καιρό σε P3, εν-δυνάμει P2 προτεραιότητα…)