

Examensarbete 30p: Nejime

Henrik Berglund (FEND22)

Abstrakt

Nejime är en app för att skapa musik i en webbläsare. Appen är en mycket simplifierad version av programmet LSDj på Gameboy.

Innehållsförteckning

1	Inledning	3
2	Bakgrund	3
3	Syfte	3
4	Mål	4
5	Utvecklingsmiljö	4
6	Krav	4
7	Planering	4
8	Teknisk beskrivning	4
8.1	Kort beskrivning av appen	4
8.2	Globalt state	5
8.3	Indelning av komponenter: song, patterns, phrases	6
8.4	Hur man använder appen	6
8.4.1	Tangentbordskommandon	7
8.5	Instrument	7
8.6	Datastruktur för variabler	7
8.7	Spara/ladda sångdata	8
8.7.1	Komprimering av data (ta bort istället för — etc)	8
9	Utmaningar	8
10	Framtid av appen	8

1 Inledning

Denna uppsats handlar om mitt examensarbete där jag gjorde en webbapp kallad Nejime^{*1} som är till för att göra musik i en browser. Appen är en mycket simplifierad version av programmet LSDj på Gameboy som är en tracker^{*2}.

2 Bakgrund

Jag är intresserad av musik och tänkte att det kunde vara intressant att göra ett examensarbete som är relaterat till musik. Sedan tidigare kände jag till programmet LSDj på Gameboy och tänkte att jag kunde göra en webbsida med en mycket simplifierad version av det programmet.

Jag valde att skapa appen med Svelte 5 som är den senaste versionen av Javascriptramverket Svelte som kommer ha en officiell release troligt senare detta år. Svelte är känt för en simpel HTML och Javascript-liknande syntax och för att vara ett kompilerat ramverk vilket betyder att det ger väldigt bra prestanda. Jag har använt Svelte tidigare i skolarbete men fick använda React på LIA och de sista kurserna så jag ville testa Svelte igen och kände att det var en perfekt timing att testa Svelte 5 som tydligen ska göra vissa saker (t.ex state utanför komponenter) lättare.

3 Syfte

Jag har flera syften med examensarbetet:

- Skapa ett projekt som senare kan bli ett större sidoprojekt. Jag förväntade mig inte att programmet skulle vara helt perfekt efter sex veckor utan att det skulle bli en grund som senare kan förbättras och utvecklas vidare. Speciellt genom opensource för jag kan tänka mig att andra kan bli intresserade av projektet när det har blivit mer komplett.
- Börja lära mig Svelte 5.
- Utmana mig själv genom att göra något som inte bara är en vanlig webbsida. Jag har aldrig gjort något liknande i komplexitet tidigare vilket betyder att det var svårt men också väldigt givande.

^{*1} Japanska 音締め, att stämma ett instrument, speciellt shamisen. Direkt översatt “dra åt ljud”.

^{*2} En tracker är mjukvara för att skapa musik där användaren placerar ljud i rutnät. Varje rad representerar en tidsenhet och varje kolumn representerar en ljudkanal.

4 Mål

Mitt främsta mål var att göra en app för att enkelt skapa musik i en webbläsare. Fokuset med denna första version var att det skulle vara smidigt och snabbt att använda och att enkelt göra musikskisser och testa olika idéer.

5 Utvecklingsmiljö

Min utvecklingsmiljö var [Svelte 5](#) och appramverket [SvelteKit](#) vilket använder [Vite](#) som är känt för snabba uppdateringar efter kodändringar.

Jag använder VSCode som fungerar perfekt för mig för webbutveckling.

6 Krav

Nejime var mitt eget projekt så jag kunde sätta egna krav. Jag hade bara två enkla krav, att man ska kunna göra musik som kan spelas upp och att appen ska kännas enkel och smidig att använda.

7 Planering

Istället för att spendera mycket tid att skapa krav, user stories och tasks som jag kände var litet overkill när man är själv valde jag att ha en todofil med hjälp av en VSCode-extension [Todo+](#) där jag kunde lägga till saker som behöver göras och markera när de är klara. Detta gjorde att jag kunde spendera mer tid att faktiskt koda som var något jag kände behövdes eftersom det för mig var ett svårt projekt.

En annan anledning var att jag inte visste hur långt jag skulle hinna och det kändes onödigt att teorisera för mycket om saker jag kanske inte hann göra ändå. Todofilen gjorde att jag enkelt kunde göra små fokuserade tasks som jag kunde göra, sedan markera när de var klara och lägga till nya todos för att utöka funktionaliteten.

8 Teknisk beskrivning

8.1 Kort beskrivning av appen

Appen är uppbyggd av sidor och komponenter. Det finns två sidor eller routes (definierade av `+page.svelte`), en index-route och en dynamisk route för att ladda sångdata från länkar. Appen använder biblioteket [smplr](https://github.com/danigb/smplr) för att spela upp ljud genom Web Audio API.

Sidan importerar och visar en komponent som heter `Nejime.svelte`. Denna komponent är en ruta som alltid visas som hanterar funktioner som `playback` med `spacebar` och `navigation` mellan olika `view-states` som `song`, `pattern` och `phrase`. När `view-statet` ändras visas komponenten för det statet, t.ex `Song.svelte` för `song`, `Pattern.svelte` för `pattern` osv.

`Song` innehåller `patterns`. `Patterns` innehåller `phrases`. `Phrases` innehåller `notes`. Man kan tänka sig att `song` är en hel sång med alla kanaler och alla `patterns` för varje kanal, `pattern` är ett musikstycke uppbyggt av flera `phrases` och varje `phrase` kan ha flera noter.

Detta gör appen väldigt effektiv för att testa olika musikaliska idéer för man kan enkelt ändra ordningen av `phrases` i ett `pattern`. T.ex om man har `phrases` “01, 02, 03” i ett `pattern` kan man i ett annat `pattern` lägga in ordningen som “03, 02, 01” för att få en ny melodi.

När man öppnar sidan får man `defaultstatet` som är `song`. Detta är det mest “utzoomade” statet som visar alla ljudkanaler samtidigt. Efter att ha skapat ett `pattern` genom att trycka på `F` kan man byta till `pattern` genom att hålla in `S` och trycka på högerpil. Där kan man lägga in en `phrase` genom att trycka på `F` och sedan byta till `phrase` genom att hålla in `S` och trycka på högerpil. Där kan man lägga in noter för sin `phrase`. Man kan gå tillbaka till andra `view-states` genom att hålla in `S` och trycka på vänsterpil.

När man vänjer sig går det väldigt snabbt att navigera mellan olika states.

8.2 Globalt state

Istället för att skicka det som behövdes för varje `view-state` till komponenten genom `props` valde jag att ha `globalt state` istället. Detta betyder att jag kan använda eller ändra värdet för variablerna överallt.

Filen `globalState.svelte.ts` definierar dessa states. Varje state har två delar, en deklaration:

```
let activeScreenState = $state("song");
```

och en funktion:

```
export function createActiveScreenState() {
  return {
    get value() {
      return activeScreenState;
    },
    set value(newState) {
      activeScreenState = newState;
    }
  };
}
```

`$state()` är en så kallad [Rune](#) i Svelte 5 vilket skapar en reaktiv variabel.

Funktionen innehåller en getter och setter som returnerar eller sätter värdet. Man kan även lägga till egna metoder om man vill. Det viktiga här är att statet (här `$state("song")`) inte är i funktionen vilket betyder att jag kan köra den här funktionen i vilken komponent som helst för att få tillbaka samma globala värde. I normala webbsidor är det nog vanligare att man har statet i funktionen vilket gör att man kan ha flera olika states, men för mig passade det bättre med globala states.

För att använda statet sen är det bara att skriva `let activeScreenState = createActiveScreenState();` och sedan `activeScreenState.value` för att få värdet.

Jag har en hel del states vilket gör filen ganska lång. Det finns nog något smartare sätt att göra det men jag gillade hur explicit alla deklarationer blev. De behöver heller inte vara i samma fil, jag kommer nog dela upp dem bättre senare.

8.3 Indelning av komponenter: song, patterns, phrases

Som jag nämnde tidigare finns det flera view-states som song, pattern och phrase. Song-komponenten innehåller flera PatternSelector-komponenter som egentligen är button-element. Pattern innehåller PhraseSelector-komponenter och Phrase innehåller NoteSelector-komponenter. Man skulle nog kunna kombinera dessa komponenter på något smart sätt men jag kände starkt för att ha saker så lätta att förstå för mig själv som möjligt även om det betydde att filer duplicerade kod.

Pattern innehåller t.ex också TransposeSelector-komponenter till höger för att transponera[^Höja/sänka noterna till annan tonart.] en hel phrase. NoteSelector innehåller noter istället för hexnummer. Det finns vissa sådana här skillnader mellan komponenterna men det finns säkert refaktoreringsmöjligheter.

Det finns även samma indelning mellan view-states. Om man spelar upp **song** spelas patterns upp för alla kanaler (rader) samtidigt. Om man spelar upp **pattern** spelas alla phrases för kanalen vars pattern man öppnade, dvs bara en kanal åt gången. Om man spelar upp **phrase** spelas alla noter upp för frasen man öppnade. Uppspelningen är loopande så att den startar om från början efter den sista noten. Detta betyder att man kan lägga till och ändra noter medan man spelar upp en phrase/pattern vilket gör det enkelt att experimentera.

8.4 Hur man använder appen

Genom att skapa patterns och phrases med noter kan man skapa en sång. Man börjar genom att trycka **F** för att skapa ett nytt pattern, sedan gå in i det, skapa phrases, gå in i dem och lägga till noter. Genom att repetera denna process för flera kanaler och kolumner kan man göra sången längre och mer komplicerad. När man är nöjd med sin sång kan man trycka på **Save**-knappen för att få en länk så man kan gå tillbaka till den senare.

Poängen med denna app är att det ska vara enkelt och snabbt när man förstår hur det fungerar. Eftersom tangentbordskommandon är konsekventa borde det gå snabbt att lära sig.

8.4.1 Tangentbordskommandon

För att navigera kan man använda **piltangenter**.

För att lägga till pattern/phrases/noter kan man trycka på **F**.

För att ta bort pattern/phrases/noter kan man hålla in **D** och trycka på **F**.

För att ändra värdet av pattern/phrases/noter kan man hålla in **F** och trycka på **piltangenter**.

8.5 Instrument

Jag använder **smplr** att spela upp ljud i appen. **Smplr** använder sig av så kallade samples som är ljudfiler med en viss namnstruktur indelade per instrument. Dessa samples kommer från en så kallad soundfont som är en samling av ljud, man kan tänka sig som en .zip fil för ljud. Den som jag använder är defaultsoundfonten som heter Musyng Kite och stödjer General Midi-instrumenten[^Lista kan ses här https://www.cs.cmu.edu/~music/cmsip/readings/GMSpecs_Patches.htm], alltså 128 stycken instrument.

Jag har också själv ett repo sedan tidigare där jag laddade upp trumljud. Därför har jag just nu två typer av instrument i appen, “soundfont” och “soundfontdrums”.

Originalversionen av trackern stödjer bara vissa sorters ljud eftersom den är baserat på hårdvaran inuti en Gameboy, men min version använder mjukvara så jag kan egentligen ha hur många sorter instrument som helst. Jag började med dessa två sorter men kan enkelt lägga till andra sorters instrument från [Smplr](#) eller kanske senare chiptune-liknande ljud med hjälp av Web Audio API.

Ett problem jag hade tidigt var att jag försökte spara instrumenten i **Smplr** i objekt när jag ville spara sången vilket betydde att det tog alldeles för mycket plats. Nu sparar jag istället sorten och en hexkod och laddar instrumenten när jag laddar sången istället.

8.6 Datastruktur för variabler

För de flesta states som inte bara är t.ex strängar eller siffror, dvs t.ex song, patterns och phrases, används objekt. Jag kände att det blir enklare istället för t.ex en array för jag slipper hålla reda på ordningen, istället är det bara att se om en key är definierad och isåfall använda värdet, annars räknas det som tomt.

Detta gör det också enklare att spara sången med allt data eftersom jag kan bara skapa ett nytt objekt och lägga in allt som behövs för att representera sången där.

8.7 Spara/ladda sångdata

Som sagt sparar jag sången genom att skapa ett stort objekt. Sedan komprimerar jag detta objekt med `lz-string` till en sträng som sedan kan bli en länk. För att ladda sången behöver man bara öppna länken som blir den dynamiska routen som tar strängen och dekomprimerar den och laddar sången igen. Jag tror detta fungerar bra för enkla sånger men tror också att det kan bli för mycket data för långa sånger[^Maxgränsen för en länk i t.ex Chrome är 2083 tecken.]. På grund av detta kan jag behöva tänka om datastrukturen eller göra en egen komprimering innan `lz-string` på något sätt.

Annars om man bara vill spara lokalt skulle det fungera att använda `localStorage` för sångerna men det skulle inte gå att skicka dem som länkar då.

En annan möjlig lösning för att kunna dela sånger är att använda någon sorts databas där användare kan spara sina sånger. Det skulle dock betyda att man behöver någon sorts auth så att man inte kan skriva över andras sånger.

Jag vet inte exakt vilken lösning som är bäst men tror att `localStorage` där man kan spara flera sånger är den enklaste lösningen att implementera för lokalt användande.

8.7.1 Komprimering av data (ta bort istället för — etc)

9 Utmaningar

Sjuk

Svårt att testa

10 Framtid av appen

Lägga till funktioner som saknas, t.ex kommandon för arpeggion/volym

förbättra komprimering/datastruktur så man kan spara långa/komplicerade sånger

lägga till stöd för mobil

multiplayer?

11 Avslutning och reflektion

Svårt men kul