Chiptune glasba

Orodje za chiptune glasbo

Žiga Elsner,
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
Ljubljana, Slovenija
2012

Primož Bajželj,
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
Ljubljana, Slovenija
2012

Matej Mežik,
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
Ljubljana, Slovenija
2012

***Povzetek* — Včasih nas nostalgija lahko ponese nekaj let ali celo desetletje v preteklost. Chiptune je glasba, ki se je pojavila več kot desetletje v preteklosti in sega celo v prejšnje tisočletje. V nadaljevanju so opisane lastnosti chiptune glasbe, in njena uporabnost. Na kratko je omenjen tudi standard WAV, ki je bil uporabljen za sam razvoj orodja, ki je napisano v programskih jezikih Java in Processing, z namenom ustvarjanja svoje chiptune glasbe s pomočjo vnaprej prednastavljenih vzorcev. Orodje omogoča predvajanje ustvarjene glasbe, vgrajen ima tudi tokovnik (streamer) in možnost vizualizacije predvajane glasbe.**

Ključne besede - chiptune, ustvarjanje glasbe, predvajalnik, vizualizator, tokovnik

***Abstract* — Nostalgia can sometimes takes us a few years or even a decade ago. Chiptune is music that has emerged over the past decade and dates back to the previous millennium. The following describes the characteristics of chiptune music, and its usefulness. We briefly mentioned the WAV standard, which was used for the development tool itself, which is written in programming languages Java and Processing, in order to create their own chiptune music using preset samples. Tool created to play music, it also has a built streamer and visualization of music played.**

Index terms; chiptune, music composing, player, visualisator, streamer

I. UVOD

Chiptune (chip music) je sintetizirana elektronska glasba, pogosto ustvarjena z zvoki, ki jih proizvajajo zvočni čipi 8 bitnih računalniških sistemov (Commodore 64) in igralnih konzol (Nintendo Entertainment System in Gameboy). Z drugimi besedami, gre za način ustvarjanja novih glasbenih izdelkov s klasičnimi, nostalgičnimi zvoki starih video iger in računalniške strojne opreme.

Po trditvah iz vira (Wikipedia - Chiptune, 2012) se je ta glasbena zvrst rodila na Japonskem v poznih 70-tih in zgodnjih 80-tih letih, kar niti ni tako čudno, saj je bila Japonska kultura v tem času v neverjetnem porastu z video in računalniškimi igrami. Poleg tega so računalniki in igralne konzole postale zelo poceni in dostopne in je lahko vsakdo z malo umetniške žilice ustvaril čudovito glasbo.

Čeprav je bila včasih njena izdelava namenjena zgolj video igram, jo danes s pridom uporablja tako glasbena (elektronska glasba) kot filmska industrija (npr. film iz leta 2010, Scott Pilgrim vs. the World). Še posebno pa je ta glasba postala popularna med glasbenimi skupnostmi (communities) v zadnjih 10-tih letih, ki pridno pišejo tovrstno glasbo, in jo proti plačilu ponujajo na spletu. Naj omenimo le tri večje spletne strani, in sicer 8-bit collective (http://8bc.org/), 8-bit peoples (http://www.8bitpeoples.com/) in Bleep Street (http://www.bleepstreet.com/). Slednja ponuja profesionalno chiptune glasbo, ki so jo ustvarili trenutno najboljši svetovni chiptune glasbeniki. Nekaj te glasbe sicer lahko najdemo tudi na strani iz vira (Audio Catch, 2012).

O njeni popularnosti pričajo tudi številni Chiptune dogodki, kot so Philadelphijski 8-static (http://8static.com/), Baltimorski Byte Nyte (http://bytenyte.com/) in pa mednarodna festivala Blip Festival (http://blipfestival.org/2012/) in Chip-Con (http://www.chip-con.org).

Iz chiptune glasbe sta se razvili tudi dve podzvrsti, to sta bitpop in nintendocore. Bitpop je modernejša zvrst chiptune glasbe. Od chiptune se razlikuje predvsem v načinu produkcije in uporabe modernejše produkcijske opreme (npr. sintesizer). Nintendocore pa je zvrst glasbe, ki združuje agresivnejše stile (rock , metal, punk, …) z chiptune glasbo.

II. KAKO USTVARITI CHIPTUNE GLASBO?

Čeprav je bila včasih ta glasba prvotno narejena na sistemih z enim samim zvočnim čipom (od tod tudi ime), jo je danes možno ustvariti na več možnih načinov, lahko s programsko opremo (Chiptune trackers, Chiptune Plug-ins, Samples) ali s strojno opremo (Gameboy).

A. *Chiptune sledilniki*

Audio sledilniki (trackers) so programska orodja, ki omogočajo programiranje glasbe korak za korakom. Renoise je le eno izmed bolj znanih komercialnih orodij. Več orodij lahko najdete na http://en.wikipedia.org/wiki/List_of_audio_trackers.

B. *Chiptune vtičniki*

Obstaja tudi vrsta vtičnikov (plugins) za različne DAW (Digital Audio Workstations) sisteme (Pro Tools, Live, Garageband, FL Studio) s katerimi lahko ustvarimo Chiptune oblike signalov. Prednost tega pristopa je, da se glasbo ustvarja samo z MIDI podatki. Eden boljših vtičnikov prihaja iz rok japonske chiptune skupine YMCK, imenovan Magical 8-bit Plug (http://www.ymck.net/magical8bitplug/). Vtičnik prihaja s 5 oblikami signalov: square, triangle, 25% pulse, 12.5% pulse in noise. Ponuja tudi osnovne sintesizer kontrole za filter: attack, decay, sustain, release, sweep.

C. *Vzorci*

Zadnja možnost bi bila uporaba paketov posnetih vzorcev (samples) chiptune zvokov in inštrumentov. Kar nekaj paketov se najde na spletu. Te vzorce se nato uporabi v katerem izmed audio trackers ali DAW orodjih. Slaba stran tega načina bi bila pomanjkanje fleksibilnosti pri generiranju zvoka v realnem času. Prednost pa hiter dostop do chiptune zvokov. Način, ki je še posebej pisan na kožo tistim, ki bi se mogoče radi začeli amatersko ukvarjati s produkcijo chiptune glasbe, pa ne vedo kje začeti.

D. *Gameboy*

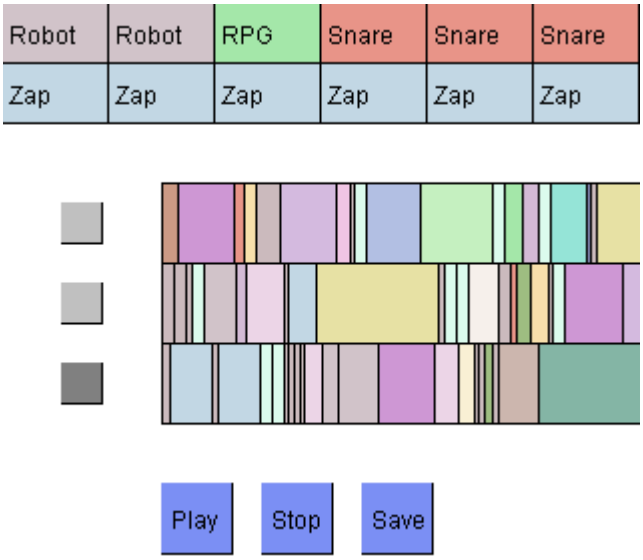
Kljub načinom omenjenim zgoraj pa obstaja edino en "pravi" način kreiranja te glasbe. Kako drugače kot ravno z eno od napravic, ki so prve na svetu znale producirate tovrsten zvok. Ena izmed njih je ravno Nintendov GameBoy, ki je danes najbolj popularna metoda kreiranja. Predvsem po zaslugi teh naprav je chiptune glasba postala tako popularna.

III. PREDVAJALNIK, TOKOVNIK IN VIZUALIZACIJA

Za seminarsko nalogo smo se odločili, da izdelamo orodje, s katerim bo mogoče sestaviti poljubno zaporedje izbranih chiptune vzorcev in jih zvezno predvajati ter shraniti na disk. Ob tem smo si zamislili še vizualizacijo, ki v obliki zabavnih efektov prikazuje like na zaslonu, sinhronizirano glede na vrsto predvajanega vzorca. Poleg tega pa je možno sestavljeno glasbo pretakati na oddaljen vir, v ta namen smo izdelali tudi tokovnik. Cilj seminarske naloge je bil, da izdelamo za uporabnika čimbolj enostavno orodje, s katerim si bo na zabaven način pomagal priti do željene chiptune glasbe. Za predstavitev vzorcev smo uporabili standard WAV.

IV. PREDVAJALNIK

Sam predvajalnik (slika 1) je napisan kot Java Applet, ki obenem omogoča predstavitev na spletu.



Slika 1: Delna slika predvajalnika

V zgornjem delu (slika 1) so v obliki mreže predstavljeni možni chiptune vzorci, ki jih lahko z desnim klikom tudi predvajamo. V primeru, da želimo predvajan vzorec uporabiti v izbranem zaporedju, ga lahko prenesemo v izbran kanal s klikom nanj. Na voljo so trije kanali, ki jih lahko uporabnik poljubno izbira s pomočjo gumbov, ki se nahajajo pred njimi. Ko uporabnik izbere poljuben vzorec, ga lahko ponovi s pomočjo gumba "*Add*", ki omogoča tudi vnos števila ponovitev. Za naključno generiranje zaporedja vzorcev, ima uporabnik na voljo tudi gumb "*Rand*", s katerim lahko napolni celoten kanal z naključno izbranimi vzorci. Čiščenje kanala je omogočeno s pomočjo gumba "*Del*", ki odstrani vse trenutne vzorce v izbranem kanalu. Kot smo že zgoraj omenili, so na voljo trije kanali, ki jih lahko uporabnik izbere po želji in jih napolni z vzorci. V primeru, da več kanalov hkrati vsebuje vzorce, se le-ti predvajajo hkrati. To omogoča zlivanje kanalov, ki bo opravljeno med samim predvajanjem. Za predvajanje izbranih vzorcev je na voljo gumb "*Play*", s katerim tudi aktiviramo vizualizacijske efekte. Prekinitev predvajanja je mogoča s pritiskom na gumb "*Stop*". Poleg tega je na voljo tudi shranjevanje trenutne sheme vzorcev v eno samo WAV datoteko, kar omogoča gumb "*Save*".

Prednost tega orodja je enostavna in zelo intuitivna uporaba, saj je namenjen tudi uporabnikom, ki niso ravno vešči upravljanja naprednih orodij.

A. *Programska rešitev predvajalnika*

Aplikacija je zgrajena na osnovi Appleta, ki je namenjen objavi na spletu. Glavni razred, ki predstavlja grafični vmesnik je *Chiptune*, kjer se pripravijo vsi grafični elementi, vključno z mrežo vzorcev. Prav tako se v tem razredu ustvarijo kontrolni gumbi, ki uporabniku omogočajo kontrolo nad aplikacijo. Razred *ChiptuneSample* je namenjen zgolj predstavitvi posamičnega vzorca (ime, datoteka, itd.).

Razred *Track* predstavlja posamezen kanal, vključno z vsemi že izbranimi vzorci. Prav tako je zadolžen za predvajanje vzorcev, zato je implementiran tako, da ob predvajanju vsak

kanala teče v svoji niti (neodvisno od ostalih). Razred *Player* predstavlja povezavo med glavno aplikacijo in kanali, saj skrbi za zagon in ustavljanje niti, ki predstavljajo kanale. V bistvu se v tem razredu izvedejo ukazi samo v primeru, da uporabnik želi predvajati izbrane vzorce na kanalih ali v primeru, da želi prekiniti predvajanje. Za predvajanje vzorcev smo uporabili razrede *AudioInputStream* (branje datoteke), *AudioFormat* (format predvajane datoteke) in *SourceDataLine* (kot izvor predvajanega vzorca). Poleg njih smo potrebovali še nekaj ne tako pomembnih razredov, predvsem za dostop do določenih konstant ali statičnih metod.

Za potrebe shranjevanja 3-kanalne glasbe iz izbranih vzorcev smo razvili razred *WavAppender*, ki s pomočjo že obstoječih razredov pripravijo podatke iz vseh treh kanalov in jih shranijo v WAV datoteko. Med pomembnejše spadajo *AudioInputStream*, *SequenceInputStream*, *AudioSystem* in *ByteArrayInputStream*. Shranjevanje poteka na nivoju vseh treh kanalov (če niso prazni), zato smo se pri podatkih oz. vzorcih spustili na nivo byte-ov, s katerimi lahko nato operiramo za samo združevanje kanalov. Združevanje poteka s pomočjo izbranih deležov vsakega byte-a. Na primer, da imamo v nekem trenutku na vseh treh kanalih določene byte, jih je potrebno med seboj pravilno porazdeliti. Mi smo se odločili tako, da smo napisali metodo *merge()*, ki je sposobna združiti dva toka podatkov (iz dveh kanalov). Ker imamo na voljo tri kanale, to stirmo v dveh stopnjah.

Zavedali smo se, da že obstajajo določene knjižnice, ki znajo manipulirati z zvočnimi zapisi. Takšna je na primer knjižnica *JSyn* z vgrajenimi določenimi elementi, ki se zelo pogosto uporabljajo v audio industriji. Vendar prav to, da smo se spustili na nivo byte-ov, je dokaz, da chiptune glasba res ni tako kompliciranа in se jo da razumeti, potrebno je le kanček zanimanja.

B. Lastnosti standarda WAV

WAV standard je nastal leta 1991 pod okriljem organizacij Microsoft in IBM, in sicer z namenom shranjevanja zvočnih zapisov. V Windows OS se WAV uporablja za shranjevanje zvoka bodisi s pomočjo komprimiranja ali brez komprimiranja. Kompatibilen je z večino glavnih operacijskih sistemov (Macintosh, Windows, Linux), kar je povzročilo, da je standard zelo razširjen in posledično tudi uporabljen. Tako bi lahko rekli, da je WAV postal de facto standard za shranjevanje nekomprimiranega zvoka. Dodatne informacije lahko razberemo iz virov (Wikipedia - WAV, 2012) in (Trček, 2011) .

WAV standard običajno uporablja za kodiranje linearno pulzno modulacijo (Linear Pulse Code Modulation – LPCM), pri kateri lahko uporablja dva kanala po 44.100 vzorcev na sekundo s 16-imi biti na vzorec. Zvok lahko zapišemo v WAV datoteko tudi tako, da ni komprimiran, tako je kvaliteta zvoka maksimalna, s tem pa se povečuje tudi velikost datotek, ki niso primerne za posredovanje preko spleta.

Datoteka WAV je običajno sestavljena iz RIFF zapisov, ki vsebujejo najprej glavo (header), nato pa ji sledijo še podatkovni kosi (chunks). Lahko se zgodi, da datoteka vsebuje samo en RIFF zapis z glavo in enim podatkovnim kosom, ki je (običajno) sestavljen iz dveh pod-kosov (sub-chunks). Eden izmed njiju opredeljuje format zapisa, drugi pa predstavlja osrednje podatke (vzorce).

V. TOKOVNIK

Točenje (streaming) je način prenosa pri katerem končni uporabnik lahko neprestano dostopa do določenega vira katerega ponudnik ponuja možnost pretoka. Pri tem velja, da se lahko sam prikaz začne še preden je prenesen celoten vir – torej predvajanje filma se lahko začne še predno imamo cel film na računalniku. Primeri različnih točenj (streaming) so npr. radio, televizija ali cene delnic.

Za pretok po omrežju je najbolj uporabljen referenčni model TCP/IP, ki velja za de facto standard. Razdeljen je na 4 sloje in v primerjavi s starejšim OSI modelom odstrani prezentacijski in sejni sloj ter združi fizično in povezavno plast. Višje plasti vedno uporabljajo funkcije nižjih plasti.

- Naloga fizične plasti je, da skrbi za fizični prenos podatkov.
- Mrežna plast skrbi za transparentno pošiljanje podatkov med mrežami. Dostava pri tem ni zagotovljena, niti vrstni red dostave. Povezava s povezavnim slojem je protok ARP, ki skrbi za pretvorbo med MAC in IP naslovi.
- Prenosna plast skrbi za povezavni in brez-povezavni način delovanja. Tu najdemo protokola TCP in UDP.
- Aplikacijska plast služi uporabi standardnih in nestandardnih aplikacij kamor spada tudi naša aplikacija

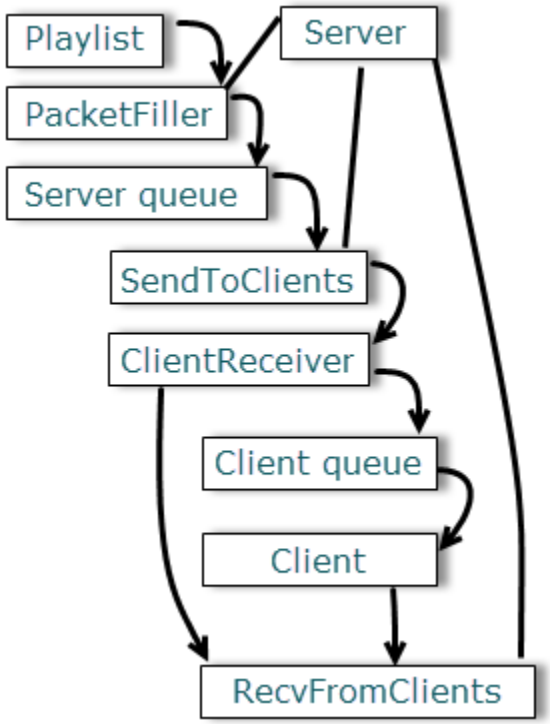
Streamanje se lahko izvaja na različnih protokolih, kot sta npr. UDP ali TCP. Za namene zvoka in videa pa tudi obstaja namenski standard RTP v povezavi z RTCP in drugi kar lahko bolj obsežno razberemo na (Wikipedia - Streaming media, 2012).

TCP se smatra za zanesljiv protokol s čimer so vsi paketi dostavljeni in so v pravem vrstnem redu. Pri tem se sicer lahko pojavi težava ob večjih motnjah na omrežju, saj se mora paket takrat še enkrat ali večkrat poslati s čimer se streamanje ustavi. Zato se v takem primeru ustvari nek pomnilnik, ki to preprečuje - s tem se sicer ustvari določen zamik.

Druga možnost pa je UDP, ki velja za nezanesljiv protokol, kar pomeni, da ne zagotavlja, da bo paket zagotovo prišel na cilj kot tudi ne, da bodo prišli v pravem vrstnem redu. Te težave se potem rešujejo na nivoju aplikacije, kjer npr. pazimo, da pakete pred predvajanjem uredimo po vrsti.

Primeri drugih tokovnikov so Adobe Flash Player, Windows Media Player, RealPlayer, Quick time in drugi. Različni ponudniki ponavadi nudijo tako svoje rešitve tokovnikov in vsaj še 2 od prej naštetih za primer internetnega radia. Sami smo se odločili za izdelavo lastnega tokovnika, ki bo deloval s protokolom UDP.

A. *Opis programske rešitve*



Slika 2: Prikaz sestave programske rešitve

Slika 2 prikazuje sestavo programske rešitve, ki je opisana v nadaljevanju.

B. *Paket*

Preko medmrežja pošiljamo serializiran objekt razreda Packet, ki služi za prenos zvoka in formata. Med drugim je v paketu tudi nekakšen ID oz. zaporedna številka s katerim se preverja ali je to res naslednji paket. Ob sami konstrukciji se objekt z zvokom oz. formatom pretvori v zapis bajtov, ki se uporabljajo za kasnejše pošiljanje.

1) *Zvok*

Komponenta AudioInputStream nam omogoča, da pretok ne zgolj preusmerimo v izhod za zvok, ampak tudi v tok bajtov do katerih nato lahko dostopamo. S pomočjo vsebine formata nato lahko vse skupaj razdelimo na določene dele, ki jih nato lahko uporabimo za pošiljanje preko tokovnika. Pri tem je pametno, da delom dodelimo pametno velikost, kar lahko na strani klienta vpliva na kakovost zvoka. Pri tem moramo upoštevati izgube, kjer je dobro, da so paketi čim manjši (zaradi primera izgub oz. zamujanja paketov je na zvoku manj izgube, če so paketi manjši) ter število poslanih paketov, kjer je bolje, da so paketi čim večji (na ta način se vse skupaj pohitri).

2) *Format*

Razred AudioInputStream poleg zvoka vsebuje tudi informacijo o formatu v razredu AudioFormat. Sam razred AudioFormat ni serializiran, kar pomeni, da ga ne moramo direktno pretvoriti v bajte ter poslati preko omrežja. Zaradi tega smo izdelali poseben razred, ki omogoča serializiranje podatov in s tem tudi pretvorbo ter prenos po omrežju.

C. *Klient*

Klient je sestavljen iz dveh delov:

- Razred Client za inicializacijo in obdelavo paketov
- Razred ClientReceiver za prejemanje paketov

1) *Inicializacija*

Klient ima nalogo, da poleg inicializacije ostalih objektov inicializira tudi nit razreda ClientReceiver, ki služi, da se prejemanje paketov dogaja ločeno neodvisno od ostalega dela. Poleg tega pa ima tudi zelo pomembno nalogo, da pošlje določeno začetno kodo s čimer označi strežniku, da želi prejemati zvok.

2) *Prejemanje paketov*

Prejemanje paketov se dogaja v ločeni niti, kar nam omogoča ločeno delovanje predvajanja in prejemanja paketov. Na ta način se paketi poskušajo, najhitreje prejemati, kar je ključno za hitrost predvajanja. Sami prejeti paketi se dodajajo v določeno vrsto, ki je v drugi niti spraznjena. Poleg tega ima ta nit tudi nalogo, da v primeru, če je prejeti paket format, prejetje le tega sporoči strežniku.

3) *Obdelava paketov*

Prvi prejeti paket je paket formata. Le ta je v niti za prejemanje paketov potrjen strežniku, da ne pride do ponovnega pošiljanja. V tej niti pa se ob prejemu formata nastavi izhod na dobljen format, s čimer je v nadaljevanju možen vpis podatkov za zvok.

Naslednji paketi so paketi z zvokom, kjer je kot prvo potrebno preverjanje ali gre res za naslednji zaporedni paket. Vsi paketi so v osnovi uvrščeni v neko tabelo (array) s katere se vsakič vzame najstarejši naslednji paket. Ob tem obstaja tudi možnost, da kakšen paket pride z zamudo, kar pomeni, da ta paket ne bo nikoli predvajan oz., da bo izbrisan ob naslednji iteraciji.

Poleg tega ima klient v tem delu nalogo, da sporoča ali je morda preveč zasičen. Ob določenem presegu minimalnega števila strežniku sporoči naj začne hitreje pošiljati pakete oz. ob presegu maksimalnega števila, da naj jih začne pošiljati počasneje. S tem se prepreči, da v primeru, ko strežnik pošilja oz. klient hitreje prejema pakete kot jih klient lahko predvaja, da bi število paketov preraslo v neskončnost.

D. *Strežnik*

Strežnik v osnovi sestoji iz treh glavnih niti, ki služijo za pravilno funkcioniranje aplikacije:

- Nit razreda PacketFiller, ki skrbi za dodajanje novih paketov
- Nit razreda RecvFromClients, ki služi za prejemanje iz strani klientov
- Nit razreda SendToClients, ki služi za pošiljanje klientom

1) *Dodajanje novih paketov*

Za pridobivanje novih paketov služi poseben razred Playlist, katera med drugim omogoča tudi dodajanje novih datotek oz. imenikov. Po odprtju posamezne datoteke se s

pomočjo razreda AudioInputStream kot prvo pridobi format. Za zbiranje paketov služi posebna vrsta kamor je paket formata v nadaljevanju tudi uvrščen. Formatu sledi branje posameznih delov zvoka, ki se prav tako v obliki paketa zapišejo v prej omenjeno vrsto. Po koncu določene datoteke ima razred Playlist tudi nalogo, da odpre novo datoteko, kjer obstaja glavna možnost razširitve, ki bi npr. omogočala glasovanje za naslednjo skladbo.

2) Prejemanje s strani klientov

Zaradi različnih potreb komunikacije strežnik tudi prejema pakete iz strani klientov. Sami klienti so identificirani z IP naslovom, kar bi v primeru poslušanja z istega IP naslova prineslo težave. V nadaljevanju bi bila tako potrebna dodatna identifikacija, ki ne bi delovala zgolj na IP naslov, ampak bi se npr. klientu iz strani strežnika dodelila določena številka. V uporabi so naslednje naloge:

- Strežnik v osnovi ne ve komu mora pošiljati pakete, kar pomeni, da mu mora le to sporočiti klient. Znotraj te zanke zato strežnik prejema pakete in v primeru zahtevka za priključitev začne pošiljati pakete klientu.
- Dodatno se v tej zanki tudi prejema ali je klient uspešno prejel format. Potrditev le tega je potrebna, da ga strežnik ne pošilja ponovno, kar je popolnoma nepotrebno in bi zgolj obremenjevalo linijo.
- Zaradi napačne hitrosti pošiljanja se lahko na strani klienta pojavi problem nasičenja ali da mu primanjkuje paketov za predvajanje. V obeh problemih zaradi tega lahko pride do neželenega učinka počasnega igranja ali prekinjanja. V prvem primeru zaradi preobremenitve, medtem ko pri drugem ta učinek lahko nastane, ker se paketi prepočasi predvajajo. Zaradi tega ima strežnik tudi dodatno nalogo, da spremlja, če mu klient pošlje zahtevek za počasnejše ali hitrejše pošiljanje. Vse skupaj je pri tem omejeno na določeno minimalno in maksimalno vrednost, saj bi v nasprotnem primeru nekdo to lahko izkoristil kot varnostno luknjo ter zaradi zasičenosti klientov povzročil zaustavitev predvajanja pri le-teh.

3) Pošiljanje klientom

Za vsakega priključenega klienta se v tej niti pošiljajo paketi. Posamezen paket se vzame iz vrste, kjer so paketi nato pa se le-ta pošlje klientom. V primeru, ko se za klienta ugotovi, da ni potrdil formata se mu ga nemudoma pošlje.

VI. VIZUALIZACIJA

Za vizualizacijo smo se odločili, da uporabimo najbolj primitiven način predstavitve vzorcev, ki ima kot rezultat neko zgodbo trenutno predvajanega zaporedja vzorcev. Namen je bil, da uporabnik s pomočjo vizualizacije začuti glasbo tudi na vizualen način in se ob tem tudi zabava. Zato smo za objekte nalašč izbrali malo bolj otroške like, s katerimi želimo privabiti tudi najmlajše, ki bi se radi srečali z začetki ustvarjanja glasbe in so pri tem čimbolj izvirni.

Za vizualizacijo smo uporabili programski jezik Processing, ki velja trenutno za najbolj uporabljenega in ga lahko najdemo na viru (Processing, 2012). Za vsak vzorec smo določili primerno sliko in efekt, ki se pojavi na zaslonu za časovni interval, ki je približno enak dolžini predvajanega vzorca. Za določene like na zaslonu smo pripravili tudi njihovo značilno gibanje. Nekateri uporabljajo sinusno premikanje, drugi linearno, dodali pa smo tudi efekt utripa. Največ problemov smo imeli pri časovni sinhronizaciji prikazovanja likov na zaslonu, ki smo jo rešili z merjenjem časa prvega prikaza na zaslon in dolžino trajanja vzorca. Glede na to, smo tudi ustrezno pohitrili oz. upočasnili premikanje likov preko zaslona. Nekaj težav je bilo tudi s pričetkom predvajanja lika, in sicer v smislu, kdaj začeti predvajati, saj smo želeli vizualizacijo uskladiti z glasbo.

Slika 3: Vizualizacija (posnetek)

Obstaja več načinov vizualizacije, mi smo uporabili zabavnejši način prikaza. Lahko bi se odločili za nekatere že znane načine, ki uporabljajo kompleksne matematične operacije, kot je npr. FFT. Lahko bi izbrali vizualizacijo, ki bi upoštevala določene karakteristike zvoka, kot so višina tona, dolžina tona, glasnost, itd. S tem bi dosegli bolj sinhrono in harmonično predstavitev glasbe.

A. Programski jezik Processing

Programski jezik *Processing* je v osnovi namenjen izključno vizualizacji poatkov. Z njim na primer lahko večjo količino podatkov ustrezno predstavimo publiki in s tem omogočimo lažje razumevanje.

Zgoraj omenjeni jezik je zelo dinamičen, prilagodljiv in zaradi tega zelo razširjen in priljubljen med programerji in ostalimi uporabniki. Ekipa, ki skrbi za razvoj tega jezika, med drugim nudi tudi orodja za lažje programiranje, zaradi česar, ga lahko uporabljajo tudi grafični navdušenci, ki niso ravno navdušeni nad postavljanjem razno raznih razvojnih okoljih. Tako je na voljo preprosta aplikacija oz. orodje, kjer je možno samo z uporabo jezika *Processing* napisati program, ki nam bo po navodilih vizualiziral izbrane podatke. Sama sintaksa jezika pa je zelo podobna sintaksi Javi, ki mislimo, da je trenutno najbolj razširjen programski jezik.

Na drugi strani pa so razvijalci poskrbeli, da je jezik *Processing* moč uporabiti v razno raznih razvojnih okoljih, ki bolj naprednim programerjem omogoča, da uporabljajo že izbrana razvojna orodja, katerih so vajeni uporabljati. Sami smo izkusili možnost integracije jezika *Processing* v razvojno okolje *Eclipse*, in sicer je potrebno z uradne spletne strani (http://processing.org/) prenesti knjižnjico *core.jar* in jo vključiti v sam projekt.

Osnovni koncept jezika je takšen, da je potrebno napisati dve funkciji, prva je *setup()*, ki se pokliče le na začetku in v kateri naj bi se nastavili parametri, kot so ozadje ali velikost platna (canvas). Druga funkcija pa je *loop()*, kot samo ime pove, se kliče ciklično in je namenjena izvajanju dinamiki vizualizacije. V tej funkciji se opravljajo razni premiki, transformacije, preoblikovanja, barvanja, ipd. zato je namenjena predvsem pripravi posebnih efektov.

Izris likov in transformacije temljijo izključno na virtualnem koordinatnem sistemu, ki ima izhodiščno točko (0,0) v zgornjem levem kotu. Vrednosti na X osi naraščajo v smeri proti desni, vrednosti na Y osi (pozor) pa naraščajo proti dnu. Ukazi za izris osnovnih likov, kot so krog (elipsa), kvadrat (pravokotnik) in trikotnik, so zelo enostavni. Primer kode za izris elipse, ki bo imela središče v točki (50,40), širino 40 točk in višino 20 točk izgleda takole:

```
ellipseMode(CENTER);
ellipse(50,40,40,20);
```

Najprej določimo na kakšen način bomo določili izhodiščno točko, tako smo jo določili kot središčno (CENTER), ki je nastavljena že kot privzeta. Obstaja možnost, da nastavimo izhodiščno točko na zgornji levi kot (CORNER), kar pomeni, da predstavljata prva dva parametra ob klicu funkcije *ellipse()* koordinate točke v koordinatnem sistemu, kjer bo zgornji levi kot narisane elipse (gledano kot pravokotnik).

S pomočjo transformacij, ki so podprte s strani jezika *Processing*, je mogoče ustvarjati posebne efekte. Podprte so 2D transformacije, s katerimi pa je z manjšo dopolnitvijo moč pripraviti objekt, ki bo gledalcu viden kot 3D objekt. Najbolj osnovni transformaciji v jeziku *Processing* sta premik in zasuk. Primer kode za premik izhodišča koordinatnega sistema v točko (50, 40) in zasuk le-tega za 90° bi izgledala takole:

```
translate(50,40);
rotate(radians(90));
```

Sam jezik ima tudi podporo za manipulacijo s slikami, kar smo izkoristili v seminarski nalogi. Zelo enostavno je prikazovati slike in jih premikati po zaslonu. Za prikaz slike je potrebno sliko najprej naložiti, to storimo s pomočjo funkcije *loadImage()*, šele nato jo lahko s funkcijo *image()* prikažemo.

VII. ZAKLJUČEK

Z rezultatom smo sicer zadovoljni, saj smo dosegli svoj namen in zadovoljili pričakovanja. Ob razvoju so se nam porajale nove ideje, ki bi jih lahko vključili v orodje in s tem zelo izpopolnili uporabnost le-tega. V primeru realizacije idej, bi lahko privabili še drugo vrsto populacije, ki niso ravno glasbeni navdušenci.

Proti koncu razvoja orodja smo ugotovili, da bi lahko nekatere stvari naredili drugače oz. jih izboljšali. Tako bi lahko samo arhitekturo orodja v celoti spremenili, in sicer tako, da bi osnovna aplikacija izhajala iz programskega jezika Processing, preko katerega bi lahko krmilili tudi samo predvajanje vzorcev. S tem bi poenostavili sinhronizacijo vizualnih efektov in naredili še bolj elegantno in prikupno orodje. Prav tako bi lahko omogočili poljubno dodajanje kanalov, tako da ne bi bili omejeni samo s tremi. Mogoče bi lahko vključili tudi brisanje posameznega ali zadnjega vzorca iz kanala. Za boljšo uporabniško izkušnjo bi lahko poskrbeli s funkcijo povleci-spusti, ki bi zadovoljila tudi zahtevnejše uporabnike.

Narejen tokovnik sicer ni popoln, a lahko služi kot osnova za nadaljnje delo. Ob sami izdelavi smo se sicer dotaknili glavnih težav, ki se pojavijo s tokovnikom, kjer pa se pokaže, da je pomembno, da pravilno obravnavamo lastnosti protokola, da je zvok čimbolj kvaliteten. Zanimiva je predvsem relativno enostavna možnost nadgradnje za glasovanje, ki ni velikokrat prisotna pri različnih drugih tokovnikih.

Z množično uporabo računalnikov je prišla tudi doba, ko je postal računalnik najbolj univerzalen instrument za ustvarjanje glasbe vseh zvrsti. S tem so se povečale tudi potrebe po orodjih, ki uporabnikom pomagajo priti do željenega cilja, kot je npr. ustvarjanje sintetične glasbe. Naše orodje je namenjeno prav takšnim željam oz. zahtevam, saj je tovrstna glasba v neverjetnem razmahu.

VIII. VIRI

[1] *Audio Catch*. (25. April 2012). Prevzeto 25. April 2012 iz Audio Catch: http://www.audiocatch.com

[2] *Processing*. (15. Maj 2012). Prevzeto 15. Maj 2012 iz Processing Language: http://procesing.org

[3] *Wikipedia - Chiptune*. (20. April 2012). Prevzeto 20. April 2012 iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Chiptune

[4] *Wikipedia - Streaming media*. (20. April 2012). Prevzeto 20. April 2012 iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Streaming media

[5] *Wikipedia - WAV*. (20. April 2012). Prevzeto 20. April 2012 iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/WAV

[6] Trček, D. (2011). Računalniška zvočna produkcija, kopije prosojnic. FRI UL.