Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gregor Bajt

Razvoj aplikacije za spodbujanje učenja igranja kitare

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Sebastijan Šprager

Ljubljana, 2018

COPYRIGHT. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preučite problematiko učenja igranja kitare in zasnujte aplikacijo, ki bo uporabnika pri tem spodbujala. Preučite obstoječe mehanizme igrifikacije in možnost njihove uporabe za učinkovito vadbo in spodbujanje uporabnika pri učenju kitare. Pri realizaciji uporabite ustrezne tehnike digitalnega procesiranja signalov. Za implementacijo uporabite prosto dostopno programsko opremo. Izdelano rešitev ustrezno ovrednotite.

Zahvaljujem se svoji družini, ki mi je omogočila prijetno okolje za pisanje diplome in pa svojemu mentorju Sebastijanu Špragerju za vso pomoč.

Kazalo

Kazalo odsekov kode

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
DAW	digital audio workstation	digitalno zvočno delovno okolje
VST	virtual studio technology	virtualna studijska tehnologija
DFT	discrete Fourier transformation	diskretna Fourierova transformacija
FFT	fast Fourier transformation	hitra Fourierova transformacija
MDA	mechanics, dynamics and aesthetics	mehanike, dinamike in estetike

Povzetek

Naslov: Razvoj aplikacije za spodbujanje učenja igranja kitare

Avtor: Gregor Bajt

Učenje kitare zahteva trud, ki ga mnogi niso pripravljeni vložiti. Čeprav bi se radi naučili igranja kitare, jih odvrne vsa potrebna vadba, ki jo predvidevajo klasične metode učenja. Učenje kitare torej zahteva zadostno raven motiviranosti. V diplomski nalogi predstavimo alternativen način učenja kitare. Namesto klasičnih pristopov se kitarist začetnik uči skozi igro. Elementi s področja igrifikacije, kot so točkovanje in različni nivoji, kitarista spodbujajo in mu višajo motivacijo za nadaljnje učenje. Kitarista se med igranjem snema, s pomočjo frekvenčne analize pa se njegovo igranje analizira in oceni. V diplomski nalogi predstavimo razvoj take aplikacije. Namenjena je tistim, ki bi se radi naučili igranja kitare, pa morda za to nimajo dovolj motivacije oz. potrpežljivosti.

Ključne besede: aplikacija, učenje, kitara, motivacija, igrifikacija, procesiranje signalov.

Abstract

Title: Developing an application for stimulation of learning to play guitar

Author: Gregor Bajt

Learning to play guitar takes effort which many are not prepared to invest. They are put off by all the necessary practise needed by using classic methods of learning guitar, even though they want to learn how to play. Learning how to play guitar therefore requires a sufficient amount of motivation. An alternative method of learning to play guitar is presented in this diploma. Unlike classic methods, the beginner guitarist learns through play. Elements from the field of gamification, such as points-based grading and the usage of different levels, are used to encourage and motivate the player for further practice. The player is recorded during play. His performance is analysed and graded with the use of frequency analysis. The development of such an application is presented in this diploma. It is developed for those who wish to learn to play guitar but haven't got enough motivation or patience.

Keywords: application, learning, guitar, motivation, gamification, signal processing.

Poglavje 1

Uvod

V preteklosti se je z glasbo ukvarjal malokdo. Glasbeni inštrumenti so bili redki in dragi, glasbeno znanje pa dolgo časa ni bilo dokumentirano. Tako so se z glasbo ukvarjali samo poklicni glasbeniki oz. premožni sloj, ki si je lahko inštrumente in inštrukcije privoščil. V današnji dobi je glasbeno znanje lahko dostopno v knjižnicah, na svetovnem spletu ter pri inštruktorjih glasbe, glasbeni inštrumenti pa so zaradi masovne proizvodnje dovolj ugodni za vsakogar.

Med bolj ugodne, predvsem pa popularne inštrumente zagotovo spada kitara. Za popularnost je poskrbela glasba iz druge polovice 20. stoletja, ko so se pojavile prve rock zvezde. Njihov glorificiran in razuzdan način življenja je mnoge mlade prepričal v nakup kitare v upanju, da bodo tudi sami nekoč uspeli in bili slavni kitaristi.

Med učenjem mnogi obupajo. Ne zavedajo se, da so rock zvezde do slave prišli predvsem s trudom. Večino svojega življenja so posvetili glasbi oz. vadbi svojega inštrumenta. Malcom Gladwell v svoji knjigi Outliers [4] omenja pravilo 10.000 ur, ki pravi, da je za naziv mojstra potrebnih 10.000 ur vadbe. Večina kitaristov začetnikov ni pripravljena vložiti toliko časa in truda. Tehnične vaje same po sebi niso dovolj zabavne, počasen napredek pa demotivira začetnike. Klasični pristopi k učenju kitare z glasbenimi inštruktorji in monotonimi vajami niso za vsakogar.

Gre za problem motiviranosti, ki pa ga lahko rešimo z uporabo značilnosti iz sveta iger. S tem se ukvarja področje igrifikacije (ang. gamification). Gre za relativno novo področje, ki preučuje elemente iger, raziskuje njihove psihološke učinke in jih uporablja v drugačnih kontekstih, kot na primer spletno nakupovanje in izobraževanje.

Za izboljšanje motiviranosti učenja igranja kitare lahko torej uporabimo elemente igrifikacije. Za to potrebujemo učni pripomoček, s katerim lahko uporabnik vadi igranje kitare in v katerega lahko implementiramo elemente igrifikacije. Ta učni pripomoček naj bo aplikacija, ki snema in ocenjuje igranje uporabnika.

Samo izdelavo aplikacije nam omogoča uporaba igralnega pogona. Gre za programsko opremo, namenjeno predvsem olajšavi izdelave iger. Razvijalec se lahko osredotoči na vsebinski del projekta, saj imajo igralni pogoni ponavadi mnogo tehničnih funkcionalnosti že implementiranih.

Teoretično podlago dobimo s področja procesiranja signalov. Frekvenčna analiza zvoka nam omogoča, da posnete signale analiziramo in jih ovrednotimo. Relevantni algoritmi in metode za procesiranje signalov so že implementirane v igralnem pogonu.

Ideja rešitve ni nova. Na tržišču že obstaja implementacija s sličnim konceptom – Rocksmith. Gre za računalniško igro, narejeno za akustične, električne in bas kitare, ki uporabnikom pomaga pri učenju igranja skladb. Naša rešitev prav tako uči igranje skladb, vendar so te razporejene po težavnosti, na začetku pa je na voljo zgolj najlažja. Rocksmith za svojo delovanje potrebuje kabel za povezavo kitare z računalnikom, medtem ko ga naša implementacija ne. Naša rešitev je torej bolj priročna s poudarkom na napredovanju.

V diplomski nalogi predstavimo razvoj aplikacije, ki z uporabo frekvenčne analize, igralnega pogona in elementov igrifikacije poskuša rešiti predstavljen problem. V prvem delu diplomske naloge najprej predstavimo področje igrifikacije, nato naredimo pregled procesiranja signalov, nazadnje pa opišemo uporabljeno programsko opremo. V drugem delu diplomske naloge predstavimo razvoj aplikacije ter opišemo njene funkcionalnosti.

Poglavje 2

Igrifikacija

Igrifikacija (ang. gamification) je relativno nov pojem. Prvič ga je uporabil angleški zasnovatelj iger (ang. game designer) Nick Pelling, šele v drugi polovici leta 2010 pa je postal popularen. Eden izmed najbolj vplivnih idejnih vodij s področja igrifikacije je Sebastian Deterding. Leta 2011 je v svojem delu Gamification: Using Game-design Elements in Non-gaming Contexts [3] napisal, da se igrifikacija uporablja kot splošni neformalni izraz za uporabo elementov računalniških iger v ne-igričarskih (ang. non-gaming) sistemih za izboljšanje uporabniške izkušnje in sodelovanja. Septembra istega leta je skupaj s sodelavci v publikaciji predlagal, da se igrifikacijo definira kot uporabo elementov zasnove iger (ang. game design) v ne-igričarskih kontekstih. [2] Mesec prej sta Zichermann in Cunningham izdala knjigo z naslovom Gamification by Design, kjer sta pojem definirala kot proces razmišljanja iger in igričarskih mehanik za sodelovanje uporabnikov in reševanje problemov. [8] Leta 2013 je Zichermann s pomočjo Linderja posodobil svojo definicijo, ki tokrat vključuje ideje programov zvestobe. Igrifikacijo sta definirala kot implementacijo konceptov zasnovanja iz iger, programov zvestobe in vedenjske ekonomije za zagon uporabniškega sodelovanja. [9] Zichermannovo originalno definicijo je posodobil tudi Kapp, ki je poudaril igrifikacijo kot orodje za učenje. Meni, da ima mnogo tehnik igrifikacije temelje v izobraževalni psihologiji. Kapp igrifikacijo definira kot uporabo mehanik, izgleda in razmišljanja

iger za sodelovanje ljudi, motivacijo dejanj, promoviranje učenja in reševanje problemov. [5] Werbach in Hunter potrdita Deterdingovo originalno definicijo s svojo. Pravita, da je igrifikacija uporaba elementov in tehnik zasnove iger v ne-igričarskih kontekstih. [7]

2.1 Elementi iger

2.1.1 Nivojski model

Deterding izpostavlja, da elementi iger niso nujno vezani na igre, saj jih lahko najdemo tudi v ne-igričarskih okoljih. Vsi dosedanji elementi iger naj bi spadali v enega izmed petih nivojev abstrakcije, ki so prikazani v tabeli 2.1.

Nivo	Opis	Primeri
Vzorci zasnove vmesnika iger	Pogoste in uspešne komponente dizajna interakcije in rešitve za obstoječe probleme v kontekstu, tudi implementacije prototipov	Značke, lestvice, nivoji
Vzorci zasnove in mehanik iger	Pogosti deli zasnove iger, vezani na samo igranje	Časovne omejitve, omejitve dobrin
Načela in hevristike zasnove iger	Priporočila pristopov za reševanje problemov dizajna ali analize rešitve	Jasni cilji, pestrost igralnih načinov
Modeli iger	Konceptualni modeli komponent iger ali igralnih izkušenj	MDA; izziv, fantazija, zanimanje
Metode zasnove iger	Procesi in načela specifični za zasnovo iger.	Testiranje, zasnova s poudarkom na igro

Tabela 2.1: Tabela petih nivojev abstrakcije

2.1.2 Ogrodje MDA

Zichermann in Cunningham predstavita svoj opis elementov iger na ogrodju MDA, ki ime dobi po svojih treh komponentah: mechanics (mehanike), dynamics (dinamike) in aesthetics (estetike). Te poskušajo opisati povezavo med izdelavo in porabo iger. Estetika vzbuja čustven odziv igralca. Mehanike so tisti elementi, iz katerih zasnovatelj iger izdela igro. Dinamika je odziv, ki nastane, ko igralec deluje na mehanike in ko mehanike delujejo druga na drugo. Igralec lahko na podlagi dinamike sklepa na pravila igre.

2.1.3 Piramida elementov iger

Werbach in Hunter pravita, da so elementi iger v hierarhiji. Vsi elementi naj bi spadali v eno izmed sledečih kategorij: dinamike, mehanike in komponente. Ti pojmi niso ekvivalentni tistim iz ogrodja MDA. Priporočata, da se sistem, ki uporablja igrifikacijo, gradi od zgoraj navzdol, začenši z dinamiko, ki je najvišje v hierarhiji.

Pod dinamiko spadajo: omejitve, čustva, zgodba, napredek in odnosi. Njihov namen je dajanje motivacije. Dinamike se kažejo skozi mehanike.

Mehanike so glagoli igre, zadolženi za poganjanje igralčevega igranja skozi izzive, priložnosti, tekmovanja, sodelovanja, povratne informacije, nabiranje surovin, nagrade, transakcije in zmagovalne okoliščine. Izražajo se preko komponent.

Komponente so samostalniki igre. To so: dosežki, avatarji, značke, zbirke, boj, odklepanje vsebin, obdarovanje, nivoji, točke, družbeni grafi, ekipe in navidezne dobrine.

2.1.4 Seznam najpogostejših elementov iger

Točke

Točke služijo več namenom. Igralcu in zasnovatelju ponujajo povratne informacije. So kazalci napredka, določajo pogoje za zmago in spodbujajo tekmovanje. Nagrade in nivoji so lahko pogojeni z določenim številom točk.

Zichermann in Cunningham razlikujeta pet različnih tipov točkovanja: izkustvene točke, odkupljive točke, spretnostne točke, točke karme in točke slovesa. Izkustvene točke so igralcu dodeljene za izvajanje zaželenih dejanj. Nimajo zgornje meje, ni jih mogoče izgubiti niti obnoviti, ne uporablja se jih za nakupovanje in v večini primerov predstavljajo osnovo za razvrstitev igralcev na lestvico najboljših. Odkupljive točke se lahko zamenja za navidezne produkte, so osnova za navidezno ekonomijo igre. Spretnostne točke se dodelijo kot nagrada za specifična dejanja in prikazujejo igralčevo uspešnost v izvajanju le-teh. Točke karme si podelijo igralci med sabo, ko se želijo zahvaliti za pomoč ter tako spodbujajo družabnost. Točke slovesa predstavljajo mero zanesljivosti imetnika točk.

Značke

Werbach in Hunter značke definirata kot "vizualno predstavitev nekega dosežka". Tako točke kot značke vzbujajo potrebo po zbiranju, s tem da so značke bolj vizualno privlačne. Antin in Churchill razlikujeta pet motivacijskih značilnosti značk iz perspektive socialne psihologije: uporabnikom odkrijejo cilje, pokažejo jim možne načine igranja, pomagajo pri oceni slovesa, služijo kot statusni simbol in omogočajo poistovetenje skupin z enakimi izkušnjami.

Lestvice

Lestvice najboljših prikazujejo dosežke igralcev v padajoči ureditvi, kar v velikem številu primerov motivira igralce, da se povzpnejo po lestvici, lahko pa ima nasproten učinek. Lahko združujejo več vrst točkovanj, lahko pa

obstaja več lestvic, vsaka s svojim tipom točkovanja. Lahko je neskončna, ali pa igralca postavi na sredino ter prikaže zgolj nekaj igralcev nad in pod njim. Zichermann in Linder sta opazila vedno večjo uporabo Facebookovega družabnega grafa, s pomočjo katerega lahko igre prikažejo igralčev uspeh glede na uspeh njegovih prijateljev.

Nivoji

Nivoji so izraziti koraki igralčevega napredovanja. Vsebujejo vsaj en izziv. Sledečim nivojem ponavadi narašča težavnost, hkrati pa prinašajo nove nagrade.

Izzivi

Izzivi so konkretni primeri predčasno definiranih nalog, izraženi preko mehanik. Včasih prispevajo zgodbi in ponavadi prinašajo nagrade. Zichermann in Cunningham svetujeta, naj zasnovatelji iger redno dodajajo nove izzive, saj igralcu prinašajo globino in pomen.

Avatarji

Avatarji so nedvoumne grafične predstavitve igralčevega lika. Igre igralcem ponavadi ponujajo možnost urejanja izgleda svojega lika. Avatarji so sposobni učinkovati na dojemanje igralca v resničnem svetu, še posebno, če so podobni izgledu igralca.

Spopadi

Spopadi so bitke ali dvoboji krajšega trajanja, ki vključujejo jasne okoliščine zmage in mehanike za tekmovanje.

2.2 Psihologija igrifikacije v izobraževanju

Cilj uporabe elementov iger v ne-igričarskem okolju je lahko izboljšanje sodelovanja, motiviranje dejanj, spodbujanje učenja, reševanje problemov ali pomoč nadzorovanju vedenja. Mnogo psiholoških teorij skuša razložiti, kako uresničiti te cilje z uporabo elementov iger.

2.2.1 Instrumentalno pogojevanje

Skinner je z uporabo nagrajevanja in kaznovanja naučil živali, da izvajajo specifična dejanja. Odkril je, da se lahko z različnimi urniki spodbujevanja vzbudi različna vedenja. Ta spodbujevanja se lahko uporabijo v zasnovanju iger za spodbujevanje dolgoročnega sodelovanja igralca. Ti bodo izvajali določena dejanja bolj pogosto, če se za uspešno izvedbo dejanja dodeli točke, značke, ali nagrade drugih vrst.

2.2.2 Teorija samoodločanja

Behaviorizem, ki vključuje instrumentalno pogojevanje, je teorija, ki se za motivacijska orodja osredotoča na zunanje nagrade. Teorija samoodločanja, ki uporablja kognicijski pristop, pa pravi, da je notranja motivacija, ki izhaja iz posameznikovega cenjenja nečesa, močnejša. Teorijo sta izdelala Ryan in Deci. Razlikujeta tri prirojene potrebe, ki omogočajo notranjo motivacijo, ko so izpolnjene. Te so: avtonomija (občutek nadzora), sposobnost, (občutek mojstrstva) in potrebe po izzivu ter povezanost (z drugimi). Ryan, Rigby in Przybylski so teorijo samoodločanja uporabili na računalniških igrah, kjer so ugotovili, da sta zadovoljstvo z igro in namen igranja v prihodnosti tesno povezana z občutenjem avtonomije, sposobnosti in povezanosti. Sistemi, ki uporabljajo igrifikacijo, naj bodo torej zgrajeni tako, da ugajajo tem trem potrebam. Sposobnost je mogoče dosegati s prikazom točk in ponudbo različnih nivojev. Potrebo avtonomije lahko dosežemo s ponujanjem mnogih možnosti izbire. Povezanost lahko vzbudimo z možnostjo deljenja dosežkov na družabnih omrežjih in z lestvicami, ki prikazujejo imena igralcev.

2.2.3 Zunanja in notranja motivacija

Ko se že naravno zanimive naloge nagradijo z zunanjimi nagradami, se prvotna notranja motivacija lahko zamenja z zunanjo. Pri načrtovanju sistemov z igrifikacijo je treba poskrbeti, da notranja motivacija ni spremenjena, kar pa ne pomeni, da so zunanje nagrade neuporabne, saj so idealne za dolgočasne, ponavljajoče se naloge. Notranja motivacija ni nasprotna zunanji, ponavadi sta prisotni obe hkrati. Za primer vzemimo točke, ki so navadno mišljene kot zunanja nagrada. Točke lahko prikazujejo uporabnikov nivo obvladovanja in tako okrepijo njegov občutek sposobnosti ter ustvarijo notranjo motivacijo. Sistemi z igrifikacijo naj torej uporabljajo obe vrsti nagrad.

2.3 Uporaba v aplikaciji

V okviru diplomske naloge smo uporabili točke, značke in nivoje. Točke se kot merilo uspešnosti prikažejo po vsaki zaigrani skladbi. Uporabniku višajo notranjo motivacijo, saj lahko meri svoj napredek ter dobi občutek mojstrstva. Predvsem pa so točke pomembne kot zunanja nagrada. Tema diplomske naloge predvideva, da uporabnik nima dovoljšnje notranje motivacije za učenje igranja kitare, točke pa so same po sebi nagrada za vadbo.

Značke (zvezdice) imajo podobno funkcijo kot točke. Uporabniku omogočajo enostaven in estetsko bolj privlačen prikaz napredka. Za razliko od točk je število zvezdic najboljšega rezultata vedno prikazano.

Aplikacija ima več različnih nivojev (skladb) različnih težavnosti. Vsaka skladba ima možnost spreminjanja tempa, kar dodatno razdeli težavnost. Odklepanje težjih nivojev je odvisno od zbranega števila značk.

Točke, značke in nivoji skupaj zadovoljujejo potrebo sposobnosti, ki je del teorije samoodločanja. Zadovoljitev te potrebe je eden izmed pogojev za omogočanje notranje motivacije. Potreba po avtonomiji za diplomsko nalogo ne pride v poštev, saj za izpolnitev potrebuje več pristopov oz. načinov igranja, pravilna izvedba vsake skladbe pa je samo ena. Zadovoljitev potrebe po povezanosti zaradi ohranitve enostavnosti aplikacije nismo implementirali.

Poglavje 3

Procesiranje signalov

V tem poglavju predstavimo procesiranje signalov s poudarkom na digitalnem procesiranju. To področje predstavlja teoretično podlago tehničnih zahtev aplikacije, bolj specifično – analizo zvoka. Metode digitalnega procesiranja signalov nam omogočajo zaznavanje fundamentalne frekvence predvajanega zvoka, kar se v aplikaciji uporabi pri ugotavljanju pravilnosti igranih tonov.

3.1 Splošen opis

Področje procesiranja signalov zajema analizo, izdelavo in spremembo signalov. Signali so na široko definirani kot informacije o vedenju atributov nekega pojava [6], kot so zvok, slike in biološke meritve. Med primere uporabe spadajo tehnike za boljši prenos signalov, boljšo izkoriščenost prostora in poudarjanje specifičnih komponent merjenega signala. Področje procesiranja signalov se deli na več kategorij: analogno procesiranje signalov, neprekinjeno časovno procesiranje signalov, diskretno časovno procesiranje signalov, digitalno procesiranje signalov in nelinearno procesiranje signalov. Za temo diplomske naloge je relevantno zgolj digitalno procesiranje signalov.

3.2 Digitalno procesiranje signalov

Digitalno procesiranje signalov je uporaba digitalnega procesiranja za izvedbo raznih operacij. Procesirani signali so zaporedja števil, ki predstavljajo vzorce zvezne spremenljivke v domeni časa, prostora, ali frekvence. V primerjavi z analognim procesiranjem ima nekatere prednosti kot na primer zaznavanje in popravljanje napak pri prenosu signala in stiskanje podatkov.

Vzorčenje

Za digitalno analizo in manipulacijo analognega signala je potrebna digitalizacija s pretvornikom iz analognega v digitalni signal. Vzorčenje se izvede v dveh korakih. Najprej se signal razdeli na enakomerne intervale časa. Vsak interval predstavlja eno meritev amplitude. Nato se vsako meritev amplitude aproksimira z vrednostjo iz končne množice.

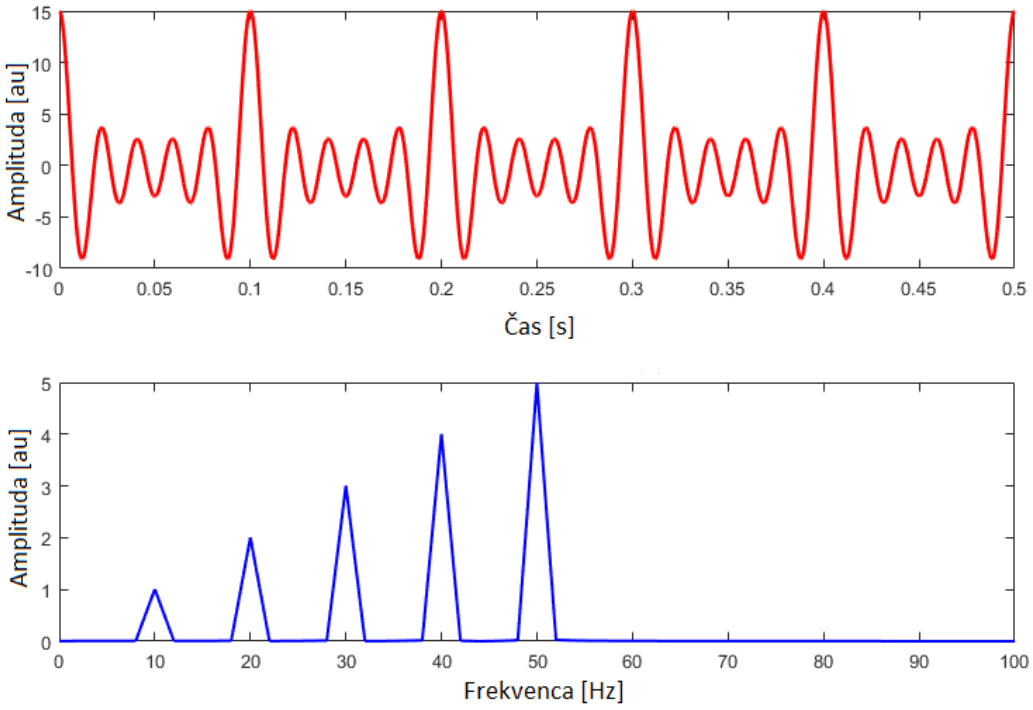
Nyquistov teorem (izrek vzorčenja) pravi, da lahko signal natančno obnovimo iz vzorcev, če je frekvenca vzorčenja večja od dvakratnika najvišje frekvence signala.

Frekvenčna domena

Frekvenčna domena je le ena izmed več domen, s katerim se ukvarja področje digitalnega procesiranja signalov. Signale se v frekvenčno domeno ponavadi pretvori s Fourierovo transformacijo, ki pretvori informacije signala v velikostno in fazno komponento vsake frekvence. Fourierovo transformacijo se ponavadi pretvori v močnostni spekter, kjer je velikost vsake frekvenčne komponente kvadrirana. S preučevanjem spektra lahko določimo katere frekvence so prisotne v vhodnem signalu in katere niso.

3.3 Fourierova transformacija

Fourierova transformacija razgradi funkcijo časa oz. signal v frekvence, ki ga sestavljajo. Rezultat je predstavitev v frekvenčni domeni. Ena izmed posplošenih transformacij je diskretna Fourierova transformacija , ki je uporabljena za Fourierovo analizo. Na sliki 3.1 je prikazan signal $\sum_{n=1}^{5} n \times cos(n \times \omega \times t)$, $\omega = 10 \times 2\pi$ in njegova diskretna Fourierova analiza.



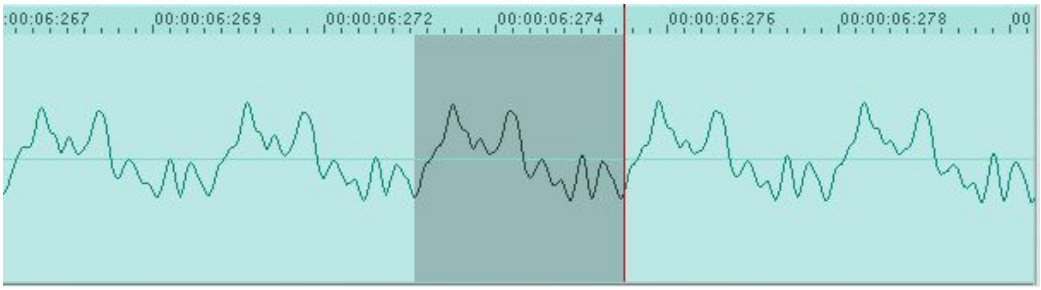
Slika 3.1: Signal in njegova diskretna Fourierova analiza

DFT transformira zaporedje N kompleksnih števil $x_0, x_1, ..., x_{N-1}$ v zaporedje kompleksnih števil $X_0, X_1, ..., X_{N-1}$; $X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$

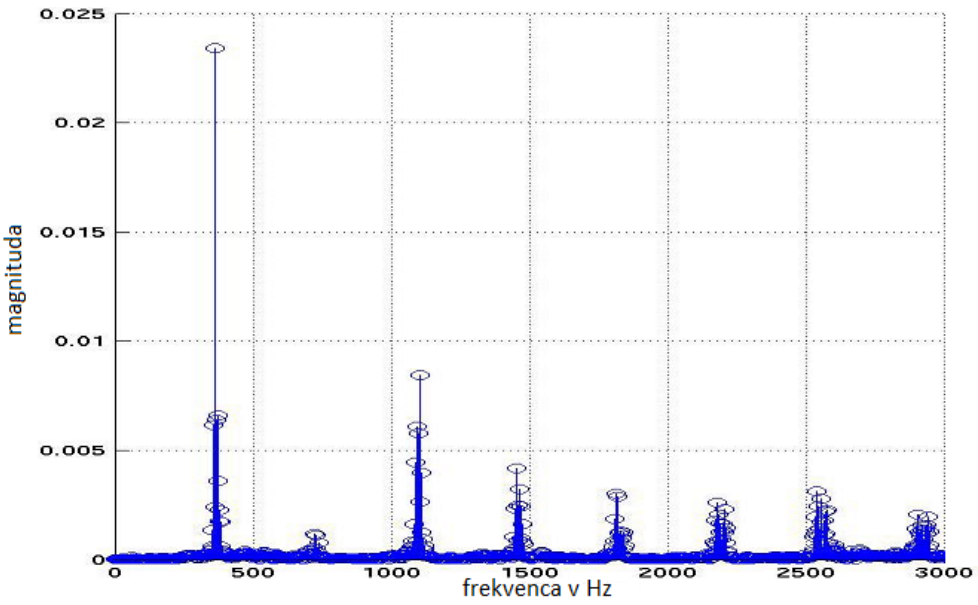
Hitra Fourierova transformacija (FFT) je algoritem, ki izračuna DFT nekega zaporedja oz. njegovega inverza. Časovno zahtevnost DFT, ki je $O(n^2)$, zmanjša na $O(nlogn)$, kjer je n enak velikosti podatkov. Obstaja več FFT algoritmov, najpopularnejši pa je Cooley-Tukey algoritem, ki rekurzivno razbija DFT velikosti $N = N_1 N_2$ v manjše DFT velikosti N_1 in N_2.

3.4 Uporaba FFT za zaznavanje tonov

FFT nam omogoča razgradnjo signala v frekvence, ki ga sestavljajo. Z analiziranjem frekvenc lahko določimo višino tona v signalu. Zvok lahko predstavimo kot skupek sinusoidnih valovanj različnih frekvenc (slika 3.2).



Slika 3.2: Prikaz valovanja tona F#4



Slika 3.3: Prikaz frekvenc tona F#4

Če želimo razbrati ton, moramo poiskati temeljno frekvenco (ang. fundamental frequency). Temeljna frekvenca je definirana kot najnižja frekvenca nekega valovnega signala. Toni imajo v svojem signalu poleg temeljne frekvence še celoštevilske večkratnike temeljne frekvence (slika 3.3). Frekvenca tona je najnižja izmed teh frekvenc. Za poimenovanje tona glede na frekvenco lahko pogledamo v tabelo tonov in njihovih frekvenc. Tabela 3.1 ima na levi stranici naštete vse tone, na zgornji pa šest oktav, ki so poimenovane po Helmholtzovem sistemu. Vrednosti so frekvence v Hertzih.

Ton	Velika	Mala	Enočrtna	Dvočrtna	Tričrtna	Štiričrtna
A	55.00	110.00	220.00	440.00	880.00	1760.00
A♯/B♭	58.27	116.54	233.08	466.16	932.33	1864.66
B/C♭	61.74	123.47	246.94	493.88	987.77	1975.53
B♯/C	65.41	130.81	261.63	523.25	1046.50	2093.00
C♯/D♭	69.30	138.59	277.18	554.37	1108.73	2217.46
D	73.42	146.83	293.66	587.33	1174.66	2349.32
D♯/E♭	77.78	155.56	311.13	622.25	1244.51	2489.02
E/F♭	82.41	164.81	329.63	659.26	1318.51	2637.02
E♯/F	87.31	174.61	349.23	698.46	1396.91	2793.83
F♯/G♭	92.50	185.00	369.99	739.99	1479.98	2959.96
G	98.00	196.00	392.00	783.99	1567.99	3135.96
G♯/A♭	103.83	207.65	415.30	830.61	1661.22	3322.44

Tabela 3.1: Tabela tonov in njihovih frekvenc v Hz

Poglavje 4

Programska oprema

4.1 Unity

Unity je igralni pogon, ki se uporablja predvsem za razvoj video iger in simulacij. V pogonu je mogoče razviti igro tako v 2D kot v 3D grafiki. Podpira 3 različne programske jezike za pisanje skript: JavaScript, C# in Boo. Razvijalcem ponuja razne storitve kot na primer Unity Ads (možnost prikazovanje oglasov v igri), Unity Collaborate (omogoča delo večih razvijalcev na istem projektu) in Unity Multiplayer (vgrajena funkcionalnost za medmrežni večigralski način). Unity spada med najbolj popularne igralne pogone, saj omogoča izvoz iger na veliko število platform. Vključene so praktično vse novejše igralne konzole in najpopularnejši mobilni operacijski sistemi. Največja prednost igralnega pogona Unity je obsežna dokumentacija in velika skupnost razvijalcev. Je tudi prvi profesionalni igralni pogon, katerega osnovna različica je na voljo zastonj, v kolikor izdelek ne prinese več kot 100.000 dolarjev letnega zaslužka.

Unity predstavlja tehnično ogrodje diplomskega dela. Uporabili smo ga za implementacijo uporabniškega vmesnika. Prav tako smo uporabili vgrajene funkcionalnosti za zajem in analizo zvoka. Za diplomsko delo smo uporabili verzijo 5.5.0f3.

4.2 MonoDevelop

MonoDevelop je integrirano razvojno okolje, priloženo k Unityju, ki združuje lastnosti urejevalnika besedil s funkcionalnostjo razhroščevanja kode in drugih nalog za upravljanje projekta. Podpira naslednje operacijske sisteme: Linux, Windows in Mac OS X. Omogoča uporabo večih različnih programskih jezikov, za C# pa še dodatno ponuja samodejno dokončanje kode. Vso kodo diplomskega dela smo napisali v MonoDevelopu.

4.3 FL Studio

FL Studio je okolje za glasbeno produkcijo (DAW – Digital Audio Workstation). Omogoča snemanje, kreiranje in manipuliranje zvočnih posnetkov z raznimi efekti. Ima grafični vmesnik za intuitivno kompozicijo večih posnetkov v skladbi. Na voljo ima veliko dodatkov kot so novi zvoki, urejevalnik zvoka, video predvajalnik, virtualni efekti, sintetizatorji ipd.

Na voljo je v treh različicah: Fruity Edition, Producer Edition in Signature Bundle. Razlika med različicami je v ceni ter v dodatnih funkcionalnostih, ki jih prinaša višja cena, kot so na primer virtualni inštrumenti (VST – Virtual Studio Technology). V okviru diplomske naloge smo uporabili preizkusno verzijo, ki pa ji manjkajo nekatere funkcionalnosti. Najočitnejša je možnost odpiranja shranjenih projektov, vendar je dovolj funkcionalna za potrebe diplome. V FL Studiu smo posneli vse skladbe za aplikacijo kot tudi testne primere.

Poglavje 5

Implementacija

5.1 Zaznavanje tonov

Ena izmed glavnih lastnosti aplikacije je zajem zvoka in njegova analiza. Igra ob igranju snema igralca in si podatke zapisuje v seznam. Ob koncu igranja primerja seznam z originalnim zapisom skladb.

Ko igralec pritisne na gumb *Practice*, se najprej sproži odštevanje v nastavljenem tempu. Ko se odštevanje konča, se začne snemanje. 16-krat na sekundo se kliče funkcija *SaveNote*.

```
1 public void SaveNote() {
2
3   frequency = GetFrequency ();
4   string note = GetNote (frequency);
5   Songs.song1.Add (note);
6 }
```

Odsek kode 5.1: Funkcija SaveNote

Funkcija *SaveNote* (odsek 5.1) najprej kliče funkcijo *GetFrequency*, ki izračuna frekvenco trenutnega odseka zvoka, nato funkcijo *GetNote*, ki frekvenci dodeli ime tona. Nazadnje ime tona doda na seznam *song1*, ki predstavlja igrano skladbo.

```
public static string GetNote(float frequency) {

  string result = "";
  int idx = 0;
  int length = toni.Length;
  float napaka = 0f;
  float precision = 0.02973f;

  while (result == "") {

    if (idx >= length) {

      result = "note not detected";
      break;
    }
    napaka = frekvence [idx] * precision;

    if (idx == 0) {
      if (frequency < frekvence [idx] - napaka) {

        result = "note not detected";
        break;
      }
    }

    if (Mathf.Abs(frekvence[idx] - frequency) < napaka) {

      result = toni[idx];
      break;
    }
    idx++;
  }
  return result;
}
```

Odsek kode 5.2: Funkcija GetNote

Funkcija *GetNote* (odsek 5.2) za vhodno spremenljivko prejme float vrednost *frequency*, ki predstavlja zaznano frekvenco. Iterira skozi tabelo fre-

kvenc in gleda, če zaznana frekvenca pade v interval. V kolikor je pogoj izpolnjen, se za rezultat vrne istoležeči element iz tabele tonov. V kolikor je zaznana frekvenca manjša od najmanjše ali večja od največje frekvence v tabeli frekvenc, se za rezultat vrne niz "note not detected".

Interval je velik za vsoto polovice frekvenčne razdalje do prejšnjega nižjega in polovice frekvenčne razdalje do naslednjega višjega tona. Na ta način je vsaki zaznani frekvenci, ki pade v interval od najnižjega do najvišjega možnega igranega tona na kitari s standardno uglasitvijo (EADGBe), dodeljen svoj ton. Tako lahko uporabnik vadi, četudi njegova kitara ni popolnoma pravilno uglašena. Na nekaterih cenejših inštrumentih (zaradi slabše izdelave preček) lahko nekateri toni zvenijo napačno navkljub pravilni uglasitvi. Tak razpon intervala pomaga le, če napačen ton ni razglašen v tolikšni meri, da preide v frekvenčni interval drugega tona.

```
1  public static string[] toni = {
2    "E2",  "F2",  "F#2", "G2",  "G#2",
3    "A2",  "A#2", "B2",  "C3",  "C#3",
4    "D3",  "D#3", "E3",  "F3",  "F#3",
5    "G3",  "G#3", "A3",  "A#3", "B3",
6    "C4",  "C#4", "D4",  "D#4", "E4",
7    "F4",  "F#4", "G4",  "G#4", "A4",
8    "A#4", "B4",  "C5",  "C#5", "D5",
9    "D#5", "E5",  "F5",  "F#5", "G5",
10   "G#5", "A5",  "A#5", "B5",  "C6"
11 };
```

Odsek kode 5.3: Tabela toni

V tabeli *toni* (odsek 5.3) so kot nizi zapisani vsi toni, ki jih s standardno uglasitvijo lahko zaigramo na kitari.

```
public static float[] frekvence = {
    82.407f,  87.307f,  92.499f,  97.999f,  103.826f,
    110f,     116.541f, 123.471f, 130.813f, 138.591f,
    146.832f, 155.563f, 164.814f, 174.614f, 184.997f,
    195.998f, 207.652f, 220f,     233.082f, 246.942f,
    261.626f, 277.183f, 293.665f, 311.127f, 329.628f,
    349.228f, 369.994f, 391.995f, 415.305f, 440f,
    466.164f, 493.883f, 523.251f, 554.365f, 587.330f,
    622.254f, 659.255f, 698.456f, 739.989f, 783.991f,
    830.609f, 880f,     932.328f, 987.767f, 1046.502f
};
```

Odsek kode 5.4: Tabela frekvence

V tabeli *frekvence* (odsek 5.4) so kot realna števila zapisane vse frekvence istoležnih tonov iz tabele toni.

```
public static float GetFrequency() {

  float frequency = 0f;
  float[] data = new float[arraySize];
  posnetek.GetSpectrumData (data, 0, FFTWindow.BlackmanHarris);
  float max = 0.0f;
  int index = 0;
  for (int j = 1; j < arraySize/2; j++) {
    if (max < data [j]) {

      max = data [j];
      index = j;
    }
  }

  frequency = index * (sampleRate / 2) / arraySize;
  return frequency;
}
```

Odsek kode 5.5: Funkcija GetFrequency

Funkcija *GetFrequency* (odsek 5.5) najprej kliče vgrajeno funkcijo *GetSpectrumData*, ki je vgrajena rešitev za hitro Fourierovo transformacijo. Funkcija vrne podatke spektra igranega zvoka, ki jih zapiše v tabelo *data* velikosti 8192. Celoten spekter frekvenc od 0 Hz do 44100 Hz (frekvenca vzorčenja) se tako razdeli na 8192 odsekov. Rezultat hitre Fourierove transformacije je na polovici zrcaljen, zato se v naslednjem koraku iterira zgolj skozi polovico tabele *data*. Najvišja možna zaznana frekvenca je tako polovica frekvence vzorčenja – 22050 Hz. Na podlagi teh podatkov lahko vidimo, da je ločljivost 2.69 Hz (zaokroženo na dve decimalki), kar izračunamo po formuli: $ločljivost = (frekvenca\ vzorčenja/2)/število\ odsekov$.

Najmanjša frekvenca, ki jo lahko zaznamo, je torej 2.69 Hz. Da dobimo vse ostale frekvence do polovice frekvence vzorčenja, uporabimo formulo: $frekvenca = indeks \times (frekvenca\ vzorčenja/2)/število\ odsekov$.

Tako lahko izračunamo vse frekvence od najnižje (2.69 Hz) do najvišje (22050 Hz). Funkcija *GetFrequency* iterira skozi prvo polovico tabele in išče element z najmočnejšim signalom.

Funkcija *GetSpectrumData* je vgrajena funkcija pogona Unity in je implementacija hitre Fourierove transformacije. Funkcija sprejme tri vzhodne spremenljivke. Prva je tabela realnih števil, kamor shrani podatke spektra. Velikost tabele mora biti enaka potenci števila dve. Druga vhodna spremenljivka je celo število, ki označuje zaporedno številko kanala, iz katerega naj funkcija vzorči. Zadnja vhodna spremenljivka je tipa FFTWindow, ki pove, katero vrsto okna za hitro Fourierovo transformacijo se uporabi pri vzorčenju. Na voljo so sledeči tipi oken, navedeni po naraščajoči kompleksnosti: pravokotno, trikotno, Hammingovo, Hanningovo, Blackmanovo in Blackman-Harissovo. Aplikacija uporablja Blackman-Harissovo okno, ker glede na testiranje ponuja najboljšo natančnost zaznavanja brez opazne upočasnitve.

5.2 Priprava skladb

Za aplikacijo smo izbrali tri skladbe različnih težavnosti. Ker je aplikacija prototip, smo posneli priredbe glavnih melodij, tako da posnetki niso predolgi, bistvo pa je vseeno zajeto.

Prva in najenostavnejša je Kuža pazi. Je uglasbitev ponarodele otroške pesmi avtorja Janeza Bitenca. Je zelo enostavna, saj je melodija v celoti spesnjena iz treh različnih tonov. Tudi ritem je enostaven – vsi toni imajo enako dolžino trajanja. Pogosto je ena izmed prvih skladb, ki se jo nauči glasbenik začetnik.

Druga skladba je Čuk se je oženil. Je uglasbitev slovenske narodne pesmi, katere melodija je bolj zapletena od prve. Uporablja šest različnih tonov. Njen ritem je tudi kompleksnejši, saj vsebuje tone različnih trajanj.

Tretja in zadnja skladba je Plug in Baby skupine Muse. Melodija je osnovana na harmonični molovi lestvici. Prvi del spominja na Tokato in fugo v D molu skladatelja Johanna Sebastiana Bacha. Spada med najbolj znane moderne rokovske melodije [1]. Je najbolj zahtevna izmed izbranih treh skladb, saj vsebuje bistveno več tonov z neizprosnim tempom. Zahteva tudi nekatere tehnike igranja, ki niso primerne za začetnike, zato niso vključene v igranje skladbe.

Skozi tri skladbe narašča težavnost, in sicer v smislu števila različnih tonov. Aplikacija je prototip, zato ima zgolj dve lažji in eno težjo skladbo. Prava aplikacija bi jih morala imeti več na voljo ter z večjim naborom težavnosti. Cilj vključitve zadnje skladbe je prikaz robustnosti oz. združljivosti aplikacije z bolj zapletenimi skladbami.

5.2.1 Prikaz skladb v kodi

Posnete skladbe služijo kot posnetek ciljne izvedbe. Igralcu omogočajo, da dobi idejo, kako naj njegovo igranje zveni. V kodi je posamezna skladba predstavljena kot tabela nizov (odsek 5.6).

```
public static string[] cuksejeozenilNotes = {
  "C", "D", "E", "F",
  "G", "X", "G", "X",
  "A", "X", "A", "X",
  "G", "X", "X", "X",
  "A", "X", "A", "X",
  "G", "X", "X", "X",
  "F", "F", "F", "F",
  "E", "X", "E", "X",
  "D", "X", "D", "X",
  "G", "X", "X", "X",
  "F", "F", "F", "F",
  "E", "X", "E", "X",
  "D", "X", "D", "X",
  "C", "X", "X", "X",
};
```

Odsek kode 5.6: Tabela cuksejeozenilNotes

V tabeli so po vrsti zapisani vsi toni, ki sestavljajo skladbo. Oktava posameznih tonov ni zapisana, posledično se od uporabnika ne zahteva igranja v specifični oktavi. Črka X nam pove, da se na tej točki ne zaigra novega tona, ampak da zveni prejšnji igrani ton.

5.3 Ocenjevanje točnosti

```
public static void compareSelector(int song) {

    int noteSize = 1;

    if (song == 0) {

      songArray = (string[]) kuzapaziNotes.Clone();
    }
    else if (song == 1) {

      songArray = (string[]) cuksejeozenilNotes.Clone();
    }
    else if (song == 2) {

      songArray = (string[]) pluginbabyNotes.Clone();
    }

    int currentTempo = PlayerPrefs.GetInt("tempo", 60);

    if (currentTempo == 60) {

      noteSize = 16;
    }
    else if (currentTempo == 80) {

      noteSize = 12;
    }
    else if (currentTempo == 120) {

      noteSize = 8;
    }

    compareAdvanced(noteSize);
  }
```

Odsek kode 5.7: Funkcija compareSelector

Ko igralec zaključi z igranjem skladbe, se kliče funkcija *compareSelector* (odsek 5.7). Za vhodno spremenljivko sprejme celo število, ki predstavlja zaporedno številko trenutno obravnavane skladbe. Na podlagi tega se določi primerjalno tabelo (spremenljivka *songArray*). Primerjalna tabela je ena izmed tabel, kjer so po vrsti zapisani vsi toni določene skladbe. Zatem se določi spremenljivka *noteSize* glede na nastavljen tempo. Nato se kliče funkcijo *compareAdvanced*.

Funkcija *compareAdvanced* (odsek 5.9) iterira skozi seznam *song1*, kjer so zapisani vsi zaznani toni igralca. Seznam se razdeli na odseke dolžine *noteSize*, ki predstavlja časovno dolžino enega tona glede na nastavljen tempo. Vsi elementi odseka se primerjajo z enim tonom tabele *songArray*.

```
for (int i = noteSize*4 + 8; i < length; i++) {
  if (songArray [noteIdx] == "X") {

    note = tempNote;
  }
  else {

    note = songArray [noteIdx];
    tempNote = note;
  }
  if (song1 [i].Length > 2 && song1[i] != "note not detected") {

    interval = 2;
  }
  else {

    interval = 1;
  }
  if (song1 [i].Substring (0, interval) == note) {

    tempScore++;
  }
  tempNoteSize--;
```

Odsek kode 5.8: Funkcija compareAdvanced – iteracija skozi interval

Algoritem z iteracijo začne na poziciji $noteSize \times 4 + 8$, kar upošteva začetno odštevanje pred začetkom igranja, saj se snemanje zaradi časa inicializacije mikrofona začne že prej. Najprej preveri, ali je trenuten ton v primerjalni tabeli nov ali zveni prejšnji ton. Zatem iz trenutnega zaznanega tona odstrani podatke o oktavi. V kolikor pride do enakosti med zaznanim tonom in tistim iz primerjalne tabele, se za ena poveča števec *tempScore*. Ko se preveri celoten interval dolžine *noteSize*, se oceni pravilnost točnosti celotnega intervala. Če je pravilnih vsaj polovica zaznanih tonov, se rezultatu prišteje 10 točk, interval pa je smatran kot pravilno zaigran ton.

```
if (tempNoteSize == 0) {
    if (tempScore >= noteSize / 2) {

      score += 10;
      noteData [noteDataIndex] = "good";
    }
    else {

      noteData [noteDataIndex] = "bad";
    }
    tempScore = 0;
    noteDataIndex++;
    noteIdx++;
    tempNoteSize = noteSize;
    if (noteIdx >= songArray.Length) {

      break;
    }
  }
}
```

Odsek kode 5.9: Funkcija compareAdvanced – konec intervala

5.4 Ocenjevanje tempa

Aplikaciji smo implementirali enostavno zaznavanje in ocenjevanje tempa. Algoritem za vsak zaigran ton ugotovi, ali je bil zaigran prehitro, prepozno, ali pravilno. Za to poskrbi del funkcije *compareAdvanced*.

```
if (firstInterval) {

  if (song1[i].Substring(0, interval) != note) {

    int previousInterval = 0;
    if (song1[i - 1].Length > 2 && song1[i - 1] != "note not
   detected") {

      previousInterval = 2;
    }
    else {

      previousInterval = 1;
    }
    string previousNote = song1[i - 1].Substring(0,
   previousInterval);

    if (song1[i].Substring(0, interval) == previousNote) {

      if (tempoData[tempoDataIndex] != "rush") {

        dragging = true;
      }
    }
  }
  tempoNote = song1[i].Substring(0, interval);
  firstInterval = false;
}
```

Odsek kode 5.10: Funkcija compareAdvanced – prvi ton intervala

Algoritem preveri in si shrani prvi zaznan ton v intervalu. V kolikor je enak zadnjemu zaigranemu iz prejšnjega intervala ter ni pravilen za trenuten

interval, se privzame, da je (bo) ton zaigran prepozno, razen v primeru, ko je trenuten interval že določen kot prehiter.

```
else {

  lagCounter++;

  if (song1 [i].Substring (0, interval) != tempoNote) {

    newNoteCounter++;
    tempoNote = song1 [i].Substring (0, interval);

    if (newNoteCounter == 1) {

      if (dragging) {

        if (lagCounter <= lagSlack) {

          tempoData [tempoDataIndex] = "good";
          tempoScore += 10;
        }
        else {

          tempoData [tempoDataIndex] = "drag";
        }
      }
      else {

        if (noteSize - lagCounter > lagSlack) {

          if (tempoDataIndex + 1 < tempoData.Length) {

            tempoData [tempoDataIndex + 1] = "rush";
          }
        }
      }
    }
```

Odsek kode 5.11: Funkcija compareAdvanced – prvič zaznan nov ton

Po prvem zaznanem tonu se preverja, ali je bil v intervalu zaznan nov ton. Če se zazna prvič in je privzeto zamujanje, se tu interval določi kot zamujen. Spremenljivka *lagCounter* (v nadaljevanju števec zakasnitve) meri velikost zamujenega dela intervala. Če je števec zakasnitve manjši od spremenljivke *lagSlack* (v nadaljevanju dopustna napaka), je interval določen kot dober, rezultatu pa se prišteje 10 točk. Dopustna napaka je velikosti 2, kar predstavlja približno desetinko sekunde. Uporabnik lahko torej zamuja oz. prehiteva približno desetinko sekunde, da se tempo intervala smatra kot pravilen.

Če se prvič zazna nov ton in ni privzeto zamujanje, se tu naslednji interval določi kot prehiter, v kolikor je razlika med števcem zakasnitve in dolžino intervala večja od dopustne napake.

```
else if (newNoteCounter == 2) {
  if (dragging) {
    if (noteSize - lagCounter > lagSlack) {
      if (tempoDataIndex + 1 < tempoData.Length) {
        tempoData [tempoDataIndex + 1] = "rush";
      }
    }
  }
}
```

Odsek kode 5.12: Funkcija compareAdvanced – drugič zaznan nov ton

Če se drugič zazna nov ton in je privzeto zamujanje, se tu naslednji interval določi kot prehiter, v kolikor je razlika med števcem zakasnitve in dolžino intervala večja od dopustne napake.

```
if (lagCounter == 8) {

  if (tempoData [tempoDataIndex] != "rush" && tempoData [
   tempoDataIndex] != "drag") {

    if (dragging) {

      tempoData [tempoDataIndex] = "drag";
    }
    else {

      tempoData [tempoDataIndex] = "good";
      tempoScore += 10;
    }
  }
}
```

Odsek kode 5.13: Funkcija compareAdvanced – konec intervala

Če trenuten interval v zadnji iteraciji intervala nima določenega tempa, se določi kot dober, razen v primeru, ko je privzeto zamujanje.

Algoritem predvideva, da uporabnik zaigra toliko tonov, kolikor jih je v skladbi. To pomeni, da če na primer uporabnik tekom časovne dolžine enega tona zaigra dva različna tona, bo prvi uporabljen za oceno trenutnega časovnega intervala, drugi pa za oceno naslednjega. Algoritem do konca časovnega intervala ignorira vse preostale zaigrane tone. Razlog take implementacije je enostavnost aplikacije oz. težavnost problema zaznavanja tempa, saj ne moremo vedeti, ali uporabnik določen ton zaigra ponesreči ali ga zaigra namerno.

5.5 Točkovanje rezultata

Po koncu iteracije se izračuna rezultat. Vsak pravilno zaigran interval prinese 10 točk. Točke za tempo in pravilnost tonov se štejejo posebej. Najprej se izračuna delež pravilnosti posamezne kategorije. Končni točkovni rezultat je utežena vsota obeh deležev.

```
1  float maxScore = Recorder.songNoteSums[PlayerPrefs.GetInt("
       currentsong", 0)] * 10;
2
3  float maxTempoScore = Recorder.songNoteSums[PlayerPrefs.GetInt("
       currentsong", 0)] * 10;
4  int songIdx = PlayerPrefs.GetInt ("currentsong", 0);
5  string songName = playerStats.songNames [songIdx];
6
7  float tempoScoreResult = tempoScore / maxTempoScore;
8  float scoreResult = score / maxScore;
9  float tempoUtez = 0.5f;
10 float noteUtez = 1 - tempoUtez;
11 float finalScore = noteUtez * scoreResult + tempoUtez *
       tempoScoreResult;
12
13 int songTempo = PlayerPrefs.GetInt ("tempo", 60);
14 songName += songTempo;
15 int starCount = PlayerPrefs.GetInt (songName, 0);
```

Odsek kode 5.14: Računanje rezultata

Glede na končni rezultat se dodelijo zvezdice. Če je rezultat 75 odstotkov ali več, se dodelijo 3 zvezdice. V primeru 50 odstotkov ali več se dodelita 2 zvezdici, v primeru 33 odstotkov pa se dodeli 1 zvezdica.

```
if (finalScore >= 0.75f) {

  if (starCount < 3) {

    PlayerPrefs.SetInt (songName, 3);
  }
  stars.Set (3);
}
else if (finalScore >= 0.5f) {

  if (starCount < 2) {

    PlayerPrefs.SetInt (songName, 2);
  }
  stars.Set (2);
}
else if (finalScore >= 0.33f) {

  if (starCount < 1) {

    PlayerPrefs.SetInt (songName, 1);
  }
  stars.Set (1);
}
else {

  stars.Set (0);
}
```

Odsek kode 5.15: Dodeljevanje zvezdic

Poglavje 6

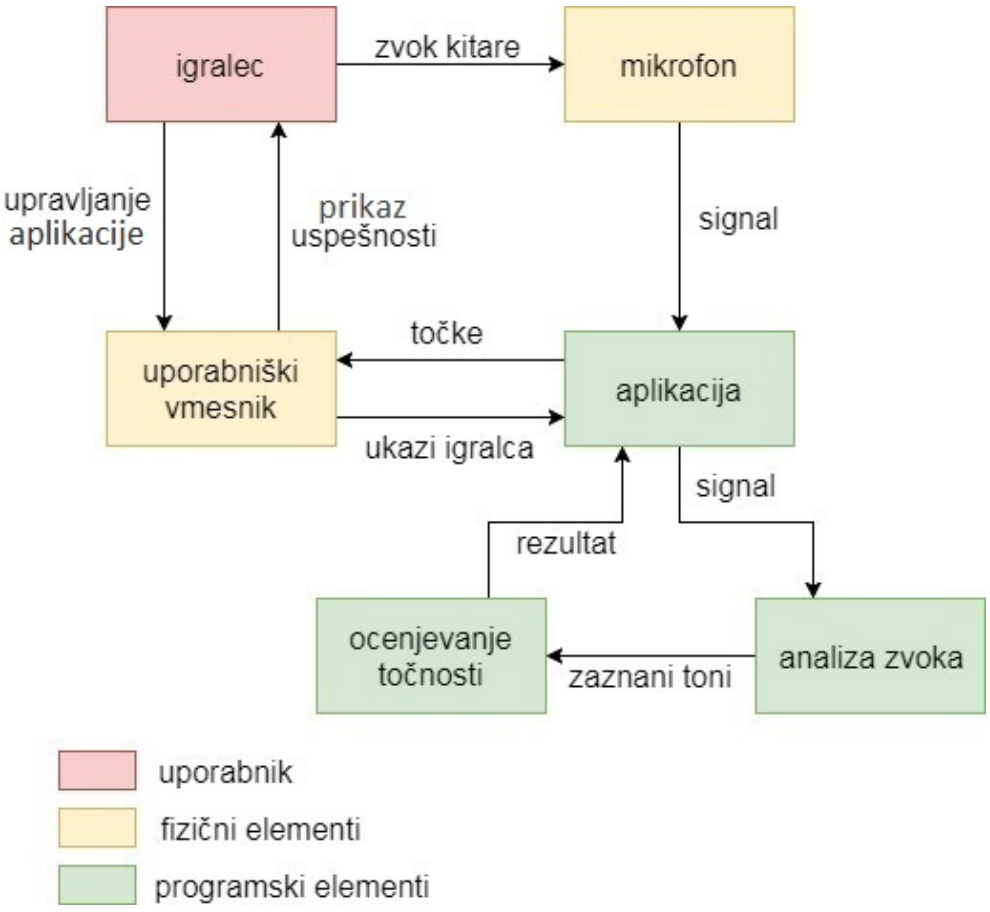
Delovanje aplikacije

Za uspešno implementacijo elementov igrifikacije potrebujemo funkcionalnosti zaznavanja in ocenjevanja pravilnosti igranih tonov. Za zaznavanje potrebujemo mikrofon, za ocenjevanje pa algoritem, ki primerja igran zvok z referenčno skladbo ter glede na uspešnost dodeli rezultat. Za uporabnika je ključen tudi prikaz in dostopnost funkcionalnosti preko uporabniškega vmesnika.

Zaradi narave aplikacije mora biti njeno delovanje dovolj hitro, predvsem med zaznavanjem zvoka. Rešitve ne smejo bistveno vplivati na hitrost snemanja in prikazovanja tonov, saj to lahko poslabša uporabnikov smisel za ritem. Samo zaznavanje mora biti dovolj natančno, da lahko razlikuje med višinama dveh sosednjih tonov v tonski lestvici. Elemente igrifikacije moramo jasno prikazati, da lahko vplivajo na motivacijo.

6.1 Realizacija rešitve

Na sliki 6.1 je prikazan konceptualni diagram rešitve, ki prikazuje različne nivoje in elemente aplikacije. Diagram je razdeljen na 3 dele: uporabnik, fizični del in programski del.



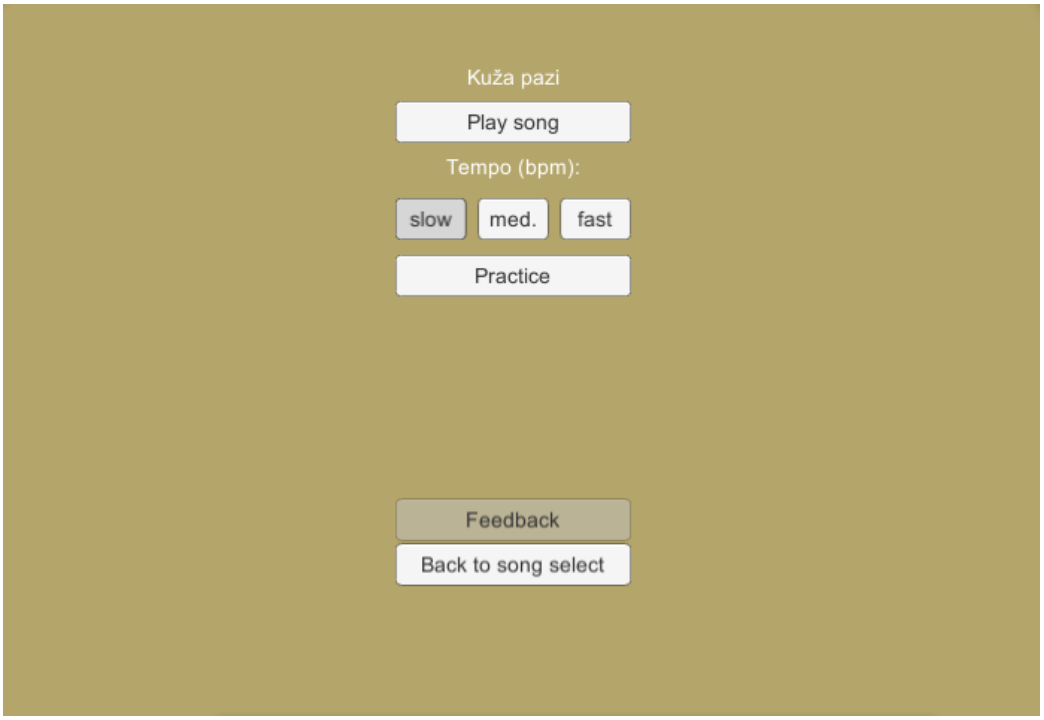
Slika 6.1: Konceptualni diagram rešitve

Aplikacija iz zvočnega signala razbere zaigrane tone, ki jih shranjuje in ob koncu igranja uporabi za ocenjevanje točnosti igranja. Uspeh prikaže s številom točk, več točk pomeni boljši rezultat.

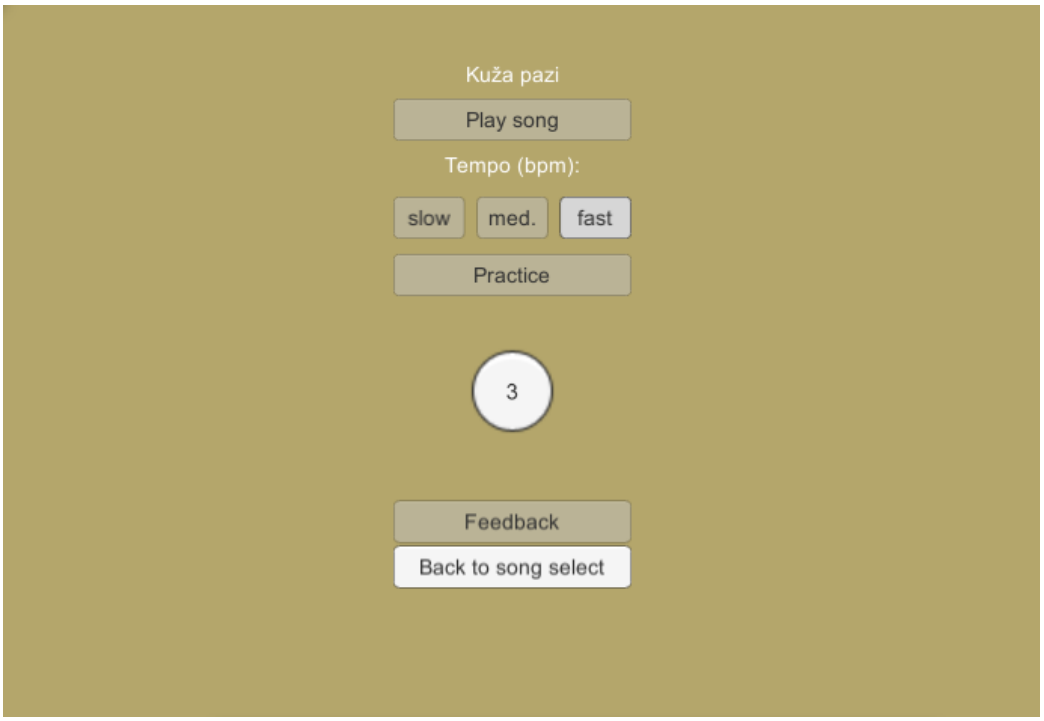
Igralec ima v aplikaciji na voljo izbiro vadbe večih skladb. Te so urejene po težavnosti od najlažje do najtežje. Vse razen prve skladbe so na začetku zaklenjene. Za njihov dostop mora igralec zbrati dovoljšnjo število zvezdic, ki jih lahko dobi s pridobitvijo dovoljšnjega števila točk. Najboljši dosežen rezultat, merjen s številom zvezdic, se izpiše pod posamezno skladbo. Število zvezdic se za vsak tempo hrani posebej.

Slika 6.2: Meni za izbiro skladbe

Ko si uporabnik izbere željeno skladbo izmed tistih, ki so mu na voljo, ima na voljo nove možnosti. Gumb *Play song* predvaja trenutno skladbo. Uporabnik tako dobi predstavo, kako naj bi dobro odigrana skladba zvenela. Pod gumbom *Play song* so trije gumbi, ki nastavijo željen tempo za vadbo skladbe. Na voljo je počasen (slow), srednji (medium) in hiter (fast) tempo. Gumb *Practice* najprej sproži odštevanje. Pod gumbom *Practice* se pojavi krog, v katerem se začne odštevanje v izbranem ritmu. Odštevanje ima tudi zvočno spremljavo, ki uporabniku pomaga ujeti ritem. Gumb *Feedback*, ki je pred igranjem zaklenjen, prikaže pregled igranja. Gumb *Back to song select* uporabnika vrne na meni za izbiro skladbe.

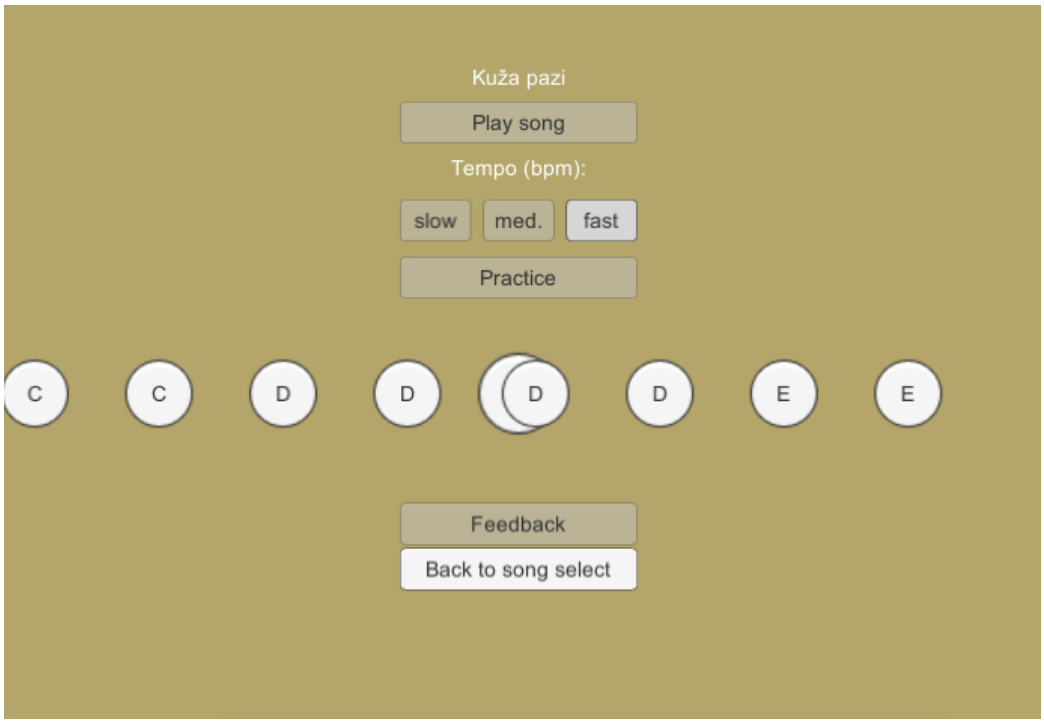


Slika 6.3: Trenutno izbrana skladba



Slika 6.4: Odštevanje

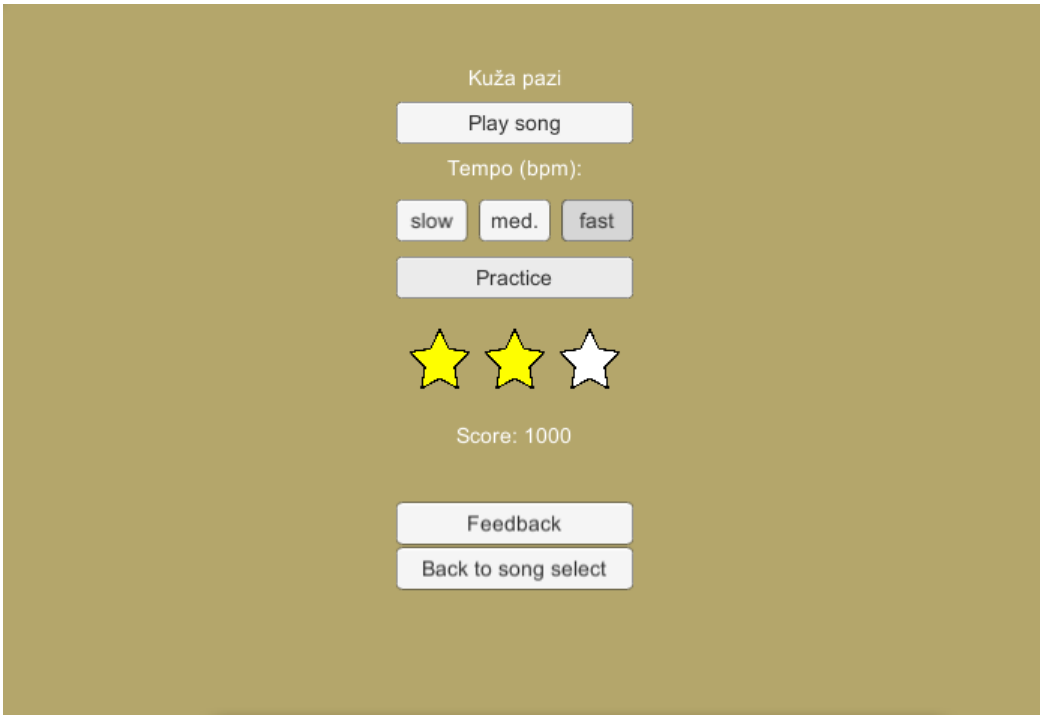
Med odštevanjem se na desni strani začnejo pojavljati krogci z imeni tonov, ki se premikajo iz desne proti levi. Uporabnik mora zaigrati napisane tone na krogcih takoj, ko prekrijejo krog na sredini. Krogci se pojavljajo v frekvenci nastavljenega tempa in se nehajo pojavljati, ko je skladbe konec. Konec skladbe oznani zvočno opozorilo.



Slika 6.5: Med igranjem skladbe

Po zvočnem opozorilu se pod gumbom *Practice* pojavi rezultat v obliki točk in zvezdic. Točke uporabniku pomagajo prikazati uporabnikov napredek, hkrati ga pa ženejo k izboljšanju rezultata. Določeno število dobljenih točk prinaša pridobitev zvezdic. Možno je dobiti do tri zvezdice. Zvezdice imajo podobno funkcijo kot točke, tj. da prikažejo uporabnikovo uspešnost in napredek. Za razliko od točk so zvezdice bolj vizualno privlačne in so boljši pokazatelj ravni obvladovanja skladbe. Obvladovanje vsake skladbe je jasno grafično prikazano s številom dodeljenih zvezdic v obliki deleža od treh

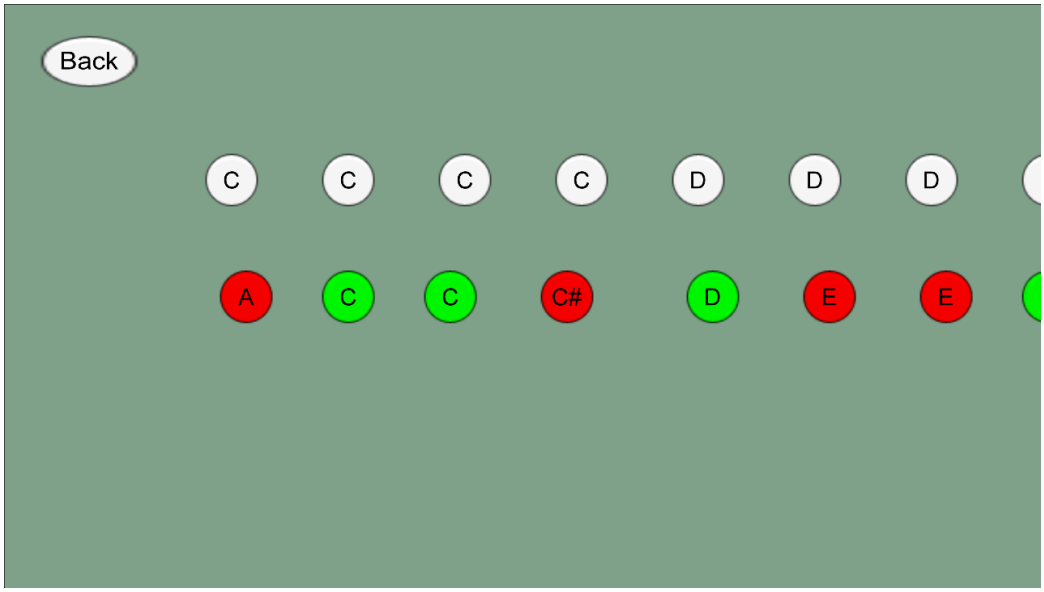
možnih. Točke niso predstavljene kot delež celote, ampak kot utežena vsota števila pravilno zaigranih tonov in pravilnosti tempa.



Slika 6.6: Ob koncu igranja skladbe

Po prikazu rezultata ima uporabnik možnost klika na gumb *Feedback*. Klik na gumb pokaže pregled igranja, kjer je grafično predstavljeno vrednotenje njegovega igranja.

V zgornjem levem kotu je gumb *Back*, ki uporabnika vrne na zaslon za vadbo skladb. V pregledu igranja sta razporejeni dve vrsti s toni. Zgornja vrsta predstavlja po vrsti razvrščene pravilne tone izbrane skladbe. Spodnja vrsta predstavlja po vrsti razvrščene zaznane igrane tone uporabnika. Za vsak takt je prikazan po en ton. Če aplikacija ni zaznala nobenega tona, je uporabljena črka X. Uporabnik se po vrsti lahko pomika s klikom in potegom miške.



Slika 6.7: Pregled igranja

V kolikor je v spodnji vrsti ton enak istoležečemu tonu iz zgornje vrste, je spodnji ton obarvan z zeleno in je smatran kot pravilen. V kolikor tona nista enaka, je spodnji ton obarvan z rdečo ter smatran kot napačen.

Če je ton iz spodnje vrste označen kot prepočasen, je v vrsti zamaknjen v desno relativno na istoležeči ton iz zgornje vrste. Če je ton iz spodnje vrste označen kot prehiter, je v vrsti zamaknjen v levo relativno na istoležeči ton iz zgornje vrste. Če je ton igran s pravilnim tempom, potem je ton navpično poravnan z istoležečim tonom iz zgornje vrste.

6.2 Vrednotenje aplikacije

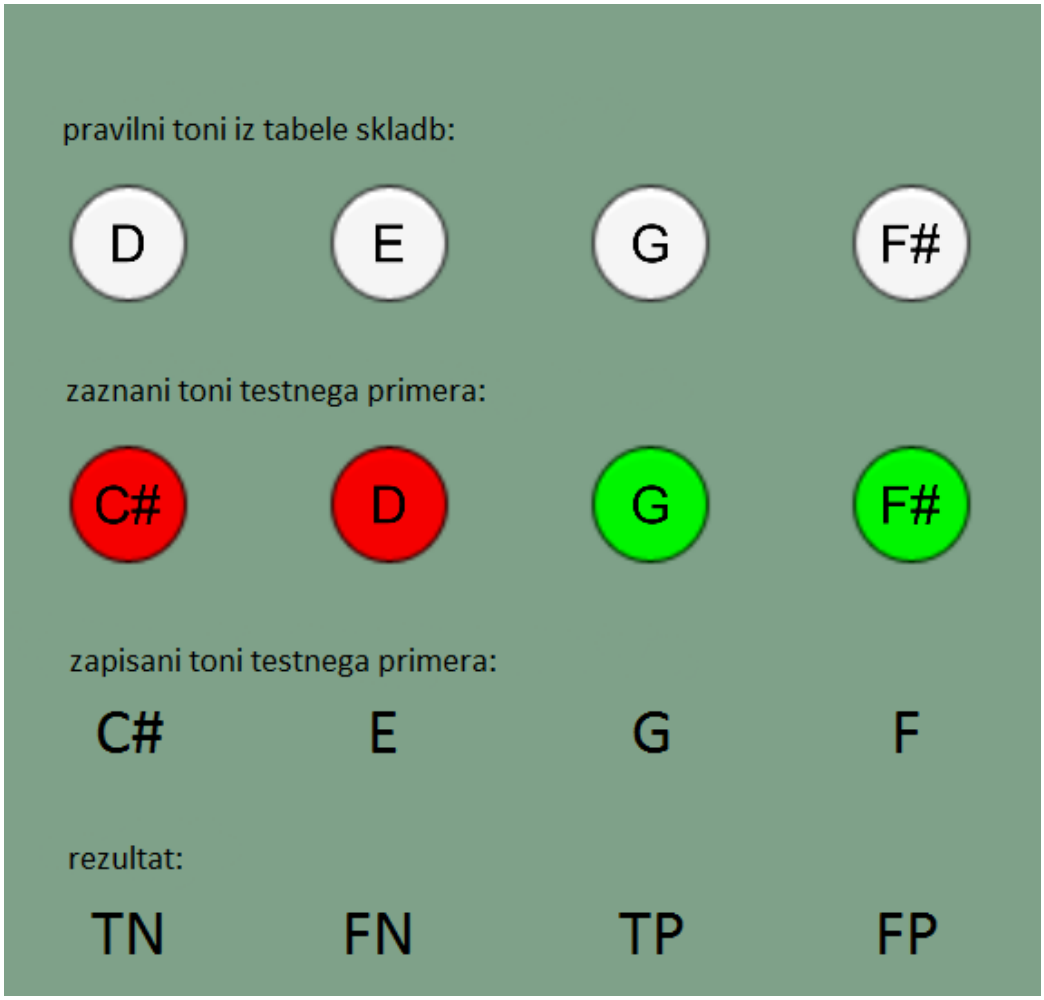
Za testiranje smo zapisali in posneli 9 testnih primerov – po en primer za vsako kombinacijo treh skladb ter treh stopenj ritma. Posnete skladbe imajo vnaprej določene napake – nekatere tone smo načrtno napačno zaigrali. Skladbe smo posneli v programu FL Studio, za snemalno napravo pa smo uporabili odjemnik (ang. pickup), ki preko vhoda USB neposredno snema v program brez zunanjih motenj.

Testiranje je potekalo tako, da smo vsak testni primer predvajali preko zvočnika, katerega zvok je zajemal mikrofon, pred tem pa smo v aplikaciji zagnali pripadajoče ocenjevanje skladbe.

Za vsak zaigran ton so možni 4 tipi rezultata:

- True positive (TP) se zgodi, ko sta tako predviden kot zaznan rezultat pravilna – aplikacija deluje pravilno.
- True negative (TN) se zgodi, ko sta tako predviden kot zaznan rezultat napačna – aplikacija deluje pravilno.
- False positive (FP) se zgodi, ko je predviden napačen rezultat, zaznan pa je pravilen – aplikacija ne deluje pravilno.
- False negative (FN) se zgodi, ko je predviden pravilen rezultat, zaznan pa je napačen – aplikacija ne deluje pravilno.

Po zaključku predvajanja smo v pregledu igranja preverili rezultate. Za vsak testni primer smo ročno primerjali zaznane tone z zapisanimi, nato pa smo preverili rezultate ocenjevanja aplikacije, ki primerja zaznane tone s pravilnimi iz tabel skladb.



Slika 6.8: Primerjava tonov

Rezultati

Tabela 6.1: Matrika zamenjav (ang. confusion matrix)

		pravilno	
		P	N
zaznano	P	352 TP	0 FP
	N	1 FN	67 TN

Izračunali smo sledeče vrednosti:

- občutljivost (ang. sensitivity ali true positive rate (TPR)):

$$TPR = TP/P = TP/(TP + FN)$$

$$TPR = 352/(352 + 1)$$

$$TPR = 0.99718$$

- specifičnost (ang. specifity ali true negative rate (TNR)):

$$SPC = TN/N = TN/(TN + FP)$$

$$SPC = 67/(67 + 0)$$

$$SPC = 1$$

- preciznost (ang. precision ali positive predictive value (PPV)):

$$PPV = TP/(TP + FP)$$

$$PPV = 352/(352 + 0)$$

$$PPV = 1$$

- natančnost (ang. accuracy (ACC)):

$$ACC = (TP + TN)/(TP + FP + FN + TN)$$

$$ACC = (352 + 67)/(352 + 0 + 1 + 67)$$

$$ACC = 0.99762$$

- F-score:

$$F = 2PPV \times TPR/(PPV + TPR)$$

$$F = 2 \times 1 \times 0.99718/(1 + 0.99718)$$

$$F = 0.99859$$

Vrednotenje rezultatov

Rezultati so skoraj popolni. Občutljivost nam pove, da so skoraj vsi predvideni pravilni toni tudi zaznani pravilno. Specifičnost je popolna, kar pomeni, da so vsi predvideni napačni toni zaznani kot napačni. Preciznost je prav tako popolna – vsi toni zaznani kot pravilni so tudi predvideni kot pravilni. Natančnost je skoraj popolna. Pove nam, da so skoraj vsi zaznani toni pravilno zaznani. F-score je skoraj popoln. Izračunan je kot harmonična sredina preciznosti in občutljivosti.

Najverjetnejši razlog za takšne rezultate je način zaznavanja tonov. Ton je smatran kot pravilen, če je zaznana frekvenca znotraj intervala tega specifičnega tona. Ta interval je dolg pol razdalje do naslednjega tona v obe smeri. Posledično ni od najnižjega do najvišjega možnega tona nobene vrzeli – vsaka frekvenca bo zagotovo dodeljena enemu izmed tonov. Ideja takšne implementacije je, da ima uporabnik lahko nepopolno uglašeno kitaro. Prav tako obstaja možnost, najpogosteje pri cenejših inštrumentih, da toni na nekaterih prečkah zaradi slabe izdelave zvenijo napačno, čeprav je kitara pravilno uglašena. V primeru manjšanja dopustnega intervala za napako bi bilo zaznavanje sicer bolj natančno, ob tem pa bi se uspešnost metrik posledično znižala.

Dodaten razlog je dejstvo, da se pri ocenjevanju pravilnosti ne upošteva pravilnosti oktave igranih tonov. Uporabnik lahko igra v katerikoli oktavi, dokler je ime tona pravilno.

Zadnji možen razlog je zvok testnih primerov, ki smo jih posneli z magnetnim odjemnikom na kitari. Zvok je drugačen od navadnega, saj se signal posname brez vpliva zvočnih motenj, kar morda vpliva na zaznavanje tonov.

6.3 Možne izboljšave

Tekom poglobljene študije in realizacije rešitve so se razkrile nove možnosti za dodatne funkcionalnosti, v kolikor bi se nadaljeval razvoj aplikacije. Obstoječa rešitev predstavlja dobro izhodišče za njihovo izvedbo.

Zaznavanje frekvenc

Metoda za določevanje frekvence pravilno zazna igran ton, vendar ne vedno v pravi oktavi. Ta pomanjkljivost je izničena z dejstvom, da se med vadbo pojavljajo toni brez zapisane oktave. Uporabnik lahko ton zaigra enako uspešno v katerikoli oktavi. Ta rešitev je zadovoljiva za skladbe, ki so uglasbene v eni oktavi. Problem nastane pri skladbah, ki uporabljajo več oktav, saj uporabnik na podlagi prikazovanja tonov ne more vedeti, v kateri oktavi mora zaigrati ton. Problem delno rešujemo z možnostjo predvajanja trenutne skladbe. Problem lahko rešili z boljšo implementacijo zaznavanja frekvenc.

Prikazovanje tonov

Med vadbo skladb se prikazujejo pravilni toni, torej aplikacija predvideva, da uporabnik zna prikazane tone na kitari najti in zaigrati. Problem bi lahko rešili z novo funkcionalnostjo aplikacije, ki bi uporabnika naučila potrebno znanje. Za vsako skladbo bi lahko dodali gumb, ki pokaže kje in kako se na kitari zaigra tone, ki se pojavijo v skladbi. Alternativno bi lahko spremenili sistem prikazovanja tonov, da namesto njih prikazuje, katero prečko na kateri struni je potrebno zaigrati. Tako bi razbremenili uporabnika učenja teorije.

Elementi igrifikacije

Aplikacija uporablja tri elemente igrifikacije: točke, značke in nivoje (izzive). Aplikacija bi bila bolj učinkovita, če bi uporabljala več elementov. Najočitnejše bi bilo dodajanje več nivojev oz. skladb. Te bi bili različnih težavnosti, tako da bi uporabnik imel boljši občutek napredka. Zaradi točkovanja bi bilo smiselno implementirati lestvico najboljših igralcev. Uporabnik bi poleg želje po napredku imel še željo po premagovanju ostalih. Z integracijo družbenih omrežij bi se lahko primerjalo rezultate z znanci, kar bi povečalo povezanost in navezanost. Lahko bi omogočili ustvarjanje igralčevega lika, ki bi v morebitnih lestvicah predstavljal uporabnika.

Poglavje 7

Sklepne ugotovitve

Za diplomsko nalogo smo razvili aplikacijo za spodbujanje učenja igranja kitare. Opisali smo področja igrifikacije, njegove elemente, psihološke vplive in primere uporabe v izobraževanju. S tega področja smo vzeli principe in elemente, ki rešujejo problem motivacije pri učenju igranja na kitaro. Zatem smo opisali procesiranje signalov s poudarkom na digitalnem procesiranju in na pripadajoče metode za pretvorbo signalov v frekvenčno domeno, ki rešujejo tehnični vidik problema diplomske naloge. Sledil smo z opisom razvoja in delovanja aplikacije, na koncu pa smo jo ustrezno ovrednotili z namenom prikaza zanesljivosti delovanja.

Aplikacijo smo razvili kot učni pripomoček za kitariste začetnike, ki bi se radi učili igranja na kitaro, vendar jih od cilja odvrača vadba inštrumenta. Aplikacija se osredotoča zgolj na vadbo igranja skladb. Uporabnike motivira za vadbo z uporabo elementov igrifikacije kot so točkovanje, značke in nivoji. Testirali smo uspešnost tehnične izvedbe.

Poleg testiranja na več uporabnikih ostanejo še možne izboljšave. Tema diplome se zaradi obsežnosti področja procesiranja signalov ne osredotoča na tehnično izvedbo zaznavanja tonov. Posledično je zaznavanje pomanjkljivo, zato so tu mogoče izboljšave. Med slednjimi spada tudi boljša uporabniška izkušnja, kakor tudi številčnejša uporaba različnih elementov igrifikacije. Izdelali smo diplomsko nalogo, ki je dobro izhodišče za nadaljno nadgradnjo.

Literatura

[1] Plug in baby - wikipedia article. Dosegljivo: `https://en.wikipedia.org/wiki/Plug_In_Baby`. [Dostopano: 4. 9. 2017].

[2] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 9–15, New York, NY, USA, 2011. ACM.

[3] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification: Using game-design elements in non-gaming contexts. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 2425–2428, New York, NY, USA, 2011. ACM.

[4] Malcolm Gladwell. *Outliers: the story of success.* New York: Little, Brown and Company, 2008.

[5] K. M. Kapp. *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education.* San Francisco, CA: Wiley, 2012.

[6] Roland Priemer. *Introductory Signal Processing.* World Scientific, 1991.

[7] D. Werbach K., Hunter. *For the Win: How Game Thinking Can Revolutionize Your Business.* Philadelphia: Wharton Digital Press, 2012.

[8] Cunningham C. Zichermann G. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. Sebastopol: O'Reilly Media, 2011.

[9] J. Zichermann G., Linder. *The Gamification Revolution: How Leaders Leverage Game Mechanics to Crush the Competition*. McGraw-Hill, 2013.