Univerzitet u Sarajevu
Elektrotehnički Fakultet
Odsjek za Računarstvo i Informatiku

Seminarski rad iz predmeta Multimedijalni Sistemi

Konverzija Slika u Boji u Zvučni Sadržaj

Studenti:
Muftić *Belma, 1423/17260*
Kulović *Nejra, 1519/17484*
Krupalija *Ehlimana, 1431/17461*

Predmetni nastavnik:
r. prof. dr. Haris Šupić, dipl. ing. el.

Februar, 2019

Definicija Zadatka

U ovom radu opisan je postupak pretvaranja slika u boji u zvučni sadržaj. U ovu svrhu prvo je neophodno izdvojiti boje iz slike (budući da se rezultujuće zvučne datoteke razlikuju na osnovu različitog udjela boja u slici), zatim je potrebno na adekvatan način dodijeliti određeni ton (od mogućih 128) nijansama boja, te kreirati zvučnu datoteku u *.mid* formatu. Kako bi se ovaj postupak demonstrirao, napravljena je *Windows Forms* aplikacija koristeći programski jezik `C#`, koja korisnicima omogućava odabir slike u boji za konverziju, pokretanje postupka konverzije te puštanje rezultujućeg zvučnog sadržaja.

U radu su detaljno opisane sve faze konverzije, uključujući razlaganje slike na *color channels*, mapiranje različitih boja u karakteristične frekvencije zvuka (tonove), odabir audio formata koji je najpogodniji za ovakvu konverziju te sam postupak kreiranja audio *file*-ova u odabranom formatu. Opisan je način implementacije aplikacije koja vrši konverziju, odnosno način na koji se slika razlaže na *color channels*, boje dodjeljuju frekvencijama, te date frekvencije koriste kako bi se kreirao audio *file* koji korisnici zatim mogu slušati.

Na kraju je opisan značaj ovakvog postupka, odnosno njegova moguća primjena u praksi za omogućavanje diferenciranja (i općenito identifikacije) različitih boja za ljude koji nisu u stanju da vide boje (ahromatopsija) ili razlikuju pojedine boje (daltonizam), te preciznost same konverzije.

Izjava o Doprinosu Članova Tima Pri Izradi Seminarskog Rada

Član tima	**Aktivnosti**
Belma Muftić	*Analiza problema*: Definicija teme *Osmišljavanje rješenja*: Pronalazak rješenja za problem pretvaranja *color channels* u oblik pogodan za pretvaranje u zvučni sadržaj *Praktična implementacija*: Dio rastavljanja slike na *color channels* i pretvaranje u ekvivalentne frekvencije zvuka *Pisanje teksta*: Za prethodno navedeni dio problema
Nejra Kulović	*Analiza problema*: Odabir tehnologije za razvoj rješenja *Osmišljavanje rješenja*: Ideja za rješavanje problema dodjeljivanja odgovarajućih frekvencija zvuka dekodiranim bojama *Praktična implementacija*: Aplikacija koja objedinjuje zasebne dijelove za manipulisanje slikom i zvukom te njihovo spajanje i usklađivanje *Pisanje teksta*: Za prethodno navedeni dio problema
Ehlimana Krupalija	*Analiza problema*: Odabir vrste protokola za rezultujuće zvučne *file*-ove *Osmišljavanje rješenja*: Prilagođavanje *.mid* formata primjeni te pretvaranje dodijeljenih frekvencija u audio format *Praktična implementacija*: Dio definisanja dijelova MIDI *file*-a, ubacivanja zvučnog sadržaja i kreiranja *.mid file*-a *Pisanje teksta*: Za prethodno navedeni dio problema

Tabela 1: Opis aktivnosti članova tima pri rješavanju problema

Sadržaj

1 Uvod

1.1 Postavka problema

Problem koji će biti riješen u okviru ovog rada je pronalazak načina da se premosti prepreka koju predstavlja nemogućnost viđenja ili razlikovanja boja. Budući da osobe koji nisu u stanju razlikovati ili vidjeti boje vide slike u boji kao da su *grayscale*, neophodno je izvršiti ekstrakciju onih elemenata koje oni ne vide - boje - te njihovu konverziju u sadržaj koji će moći razlikovati, što je u najvećem broju slučajeva zvuk. Iz tog razloga izvršiti će se postupak konverzije slika u boji u zvučni sadržaj.

U ovu svrhu potrebno je definisati tri glavne faze konverzije:

1. Izdvajanje boja iz slike, odnosno adekvatno razlaganje slike na određeni broj nijansi;
2. Mapiranje pojedinih boja u frekvencije zvuka;
3. Kreiranje audio *file*-a na osnovu informacija o tonovima od kojih se ista treba sastojati.

U nastavku će biti detaljno objašnjene sve faze konverzije, odnosno način ekstrakcije boja iz slike, njihovog mapiranja u različite frekvencije zvuka (koje će predstavljati način razlikovanja boja) te spajanja različitih boja u jedinstven audio *file* koji će zatim biti moguće reproducirati te na taj način putem slušnog organa omogućiti identifikaciju različitih boja.

1.2 Modeli boja

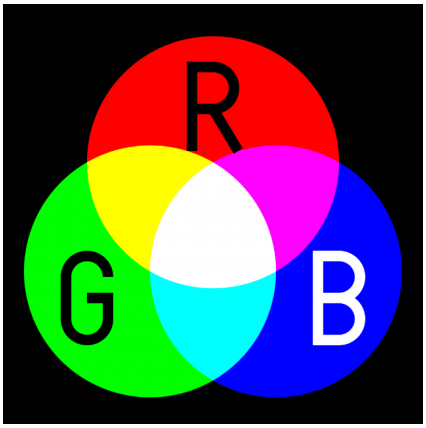
Model boja je način da se predstavi boja, bilo na ekranima ili prilikom printanja [1], tako što se kombinuje skup primarnih boja. Postoje razni modeli, no neki od najznačajnijih su [2]:

- RGB (*Red, Green, Blue*);
- CMYK (*Cyan, Magenta, Yellow, Black*);
- *Grayscale*;
- HSV (*Hue, Saturation, Value*).

1.2.1 RGB model boja

Ovaj model se koristi prilikom prikaza slike na ekranima, te se još naziva i **aditivnim modelom** jer koristi svjetlost da se prikaže boja [2]. Slika se sastoji od tri kanala boja (*color channels*) čije vrijednosti se kreću od 0 do 255 uključivo: crvena, zelena i plava. Kada svi kanali imaju vrijednost 0, tada se kreira crna boja, a kada su vrijednosti 255, tada se kreira bijela boja. "Čista" plava/crvena/zelena boja nastaje kada se vrijednosti ostale dvije boje postave na 0, vrijednost čiste boje na 255, dok se ostali spektar boja dobija kombinacijom ove tri boje [1]. Kao rezultat, moguće je predstaviti **16.7 miliona različitih boja** [2] ovim modelom.

Prikaz RGB slike na digitalnim ekranima se vrši uz pomoć **tri odgovarajuće obojene svjetlosne zrake** koje je potrebno "preklopiti" (*superimpose*) jednu preko druge. Kada nijedna zraka nema intenzitet, onda se kreira crna boja, dok puni intenzitet sve tri boje kreira bijelu boju (analogno numeričkim vrijednostima) [2]. Na Slici 1 prikazan je dijagram RGB modela.



Slika 1: RGB model boja [3]

1.2.2 CMYK model boja

Za razliku od RGB modela, ovaj model je **subtraktivni model**, kako koristi grafičke boje za prikaz, što ga čini pogodnim za printanje. Kanali koji čine ovaj model jesu cijan, purpurno crvena, žuta, te crna, i koriste se tako što se na bijelu pozadinu ”maskiraju” boje, te se tako smanjuje svjetlost koja se reflektuje, zbog čega se i naziva subtraktivnim modelom. Crna se uvodi jer mješavina ostale tri boje ne daje čistu crnu boju [2], a na Slici 2 prikazan je dijagram ovog modela.

Slika 2: CMYK model boja [4]

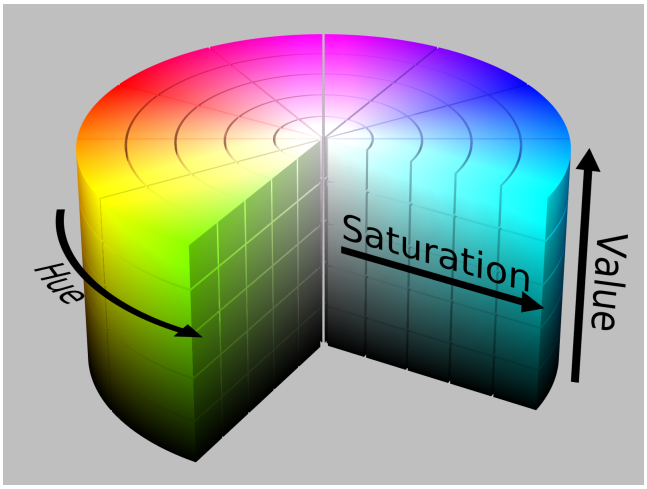
1.2.3 *Grayscale* model boja

Grayscale slika jeste slika koja sadrži samo nijanse sive boje, dakle ovaj model ima **samo jedan kanal** koji nosi informaciju o intenzitetu piksela. Slično kao kod RGB modela, vrijednost 0 predstavlja crnu boju, dok vrijednost 255 daje bijelu boju [2].

1.2.4 HSV model boja

HSV predstavlja trodimenzionalni model boja koji opisuje odnose između boja, te koji se sastoji od tri kanala: **nijansa**, **zasićenost** i **vrijednost**. Na Slici 3 nalazi se 3D prikaz ovog modela, gdje je očito da centar točka predstavlja bijelu boju, krajevi predstavljaju crnu boju, dok se ostale boje nalaze između. Ugao od ose predstavlja nijansu (što znači da vrijednosti idu od 0 do 360), udaljenost od ose predstavlja zasićenost boje, dok je nijasnsa predstavljena

udaljenošću duž ose. Vrijedi napomenuti da pored HSV modela postoji i **HSL model**, gdje se umjesto nijanse nalazi količina svjetlosti boje [2].



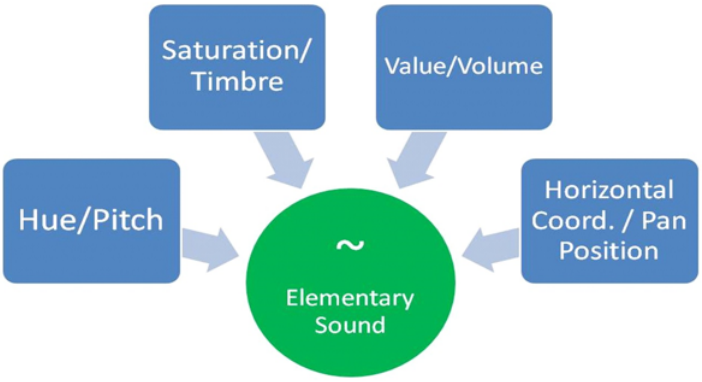
Slika 3: HSV model boja [5]

1.3 Mapiranje pojedinih boja u frekvencije zvuka

Sve pojave i događaji u prirodi uspješno se mogu opisati zakonima fizike, a usku vezu sa ovim zakonima dijele boje i zvukovi. Kao što je već rečeno, boje i zvukovi predstavljaju veoma bitne faktore koji djeluju na proces percepcije okoline od strane posmatrača. Iako svaki doživljaj okoline predstavlja subjektivnu pojavu, razvijene su mnoge metode korištene za analizu i manipulaciju ovim faktorima u cilju proučavanja procesa u kojem određeno živo biće sa čulima sluha i vidi percipira svoju okolinu.

U svrhu proučavanja ovih faktora i njihovog udjela u procesu percepcije, neophodno je bilo pronaći način na koji se oni predstavljaju. Poznato je da se i zvuk i boja mogu predstaviti **odgovarajućim frekvencijama** i upravo su proučavanjem ovih frekvencija izvedene mnoge analize i pronađena rješenja koja olakšavaju percepciju pojedincima sa oslabljenim čulom sluha ili vida. U ovom radu fokus je stavljen na način na koji se određena slika, koja se sastoji od određenog broja piksela, pretvara u zvučni sadržaj.

Ovaj proces konverzije temelji se na predstavljanju boja od kojih je sastavljena posmatrana slika preko nekog od modela boja koji su opisani u prethodnom poglavlju. Nakon toga se vrši mapiranje odabranog modela u odgovarajuću zvučnu frekvenciju. U ovu svrhu, najčešće se koristi **HSV model boja** iz prostog razloga što predstavlja boje koje su najsličnije percepciji ljudskog oka. Kao što je već rečeno, HSV model boja se sastoji od tri kanala: *Hue*, koji opisuje nijansu, *Saturation*, koji opisuje zasićenost, te *Value*, koji opisuje vrijednost. Na osnovu ovoga određen je način na koji se **svaki od ovih kanala mapira u odgovarajuću zvučnu karakteristiku**, što je prikazano na Slici 4 [6].



Slika 4: Preslikavanje HSV modela boja u zvučne parametre [7]

Nijansa (*Hue*) boje određuje količinu proizvedenih vibracija, odnosno **frekvenciju zvuka** (*pitch*). Zasićenost (*Saturation*) određuje **kvalitetu zvuka** (*timbre*), dok je vrijednost (*Value*) vezana za predstavljanje **jačine zvuka** (*volume*). Četvrti parametar prikazan na Slici 4 predstavlja horizontalne koordinate piksela koje se u ovom procesu konverzije preslikavaju u poziciju koja simulira položaj zvučnog izvora u horizontalnoj ravni. Na ovaj način se nastoji osigurati da se promjenom kanala boja u određenom smjeru, mijenja frekvencija zvuka, kvalitet kao i jačina zvuka, te time **preslikati percepciju** kojom ljudi povezuju zvuk sa određenim bojama [8].

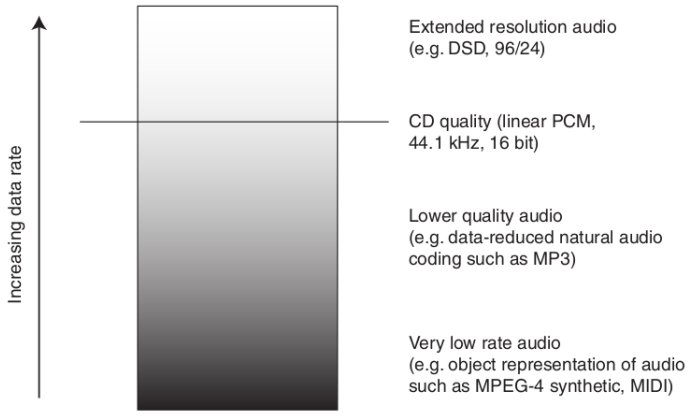
Veza između boje i zvuka oduvijek je bila zanimljiva tema istraživanja mnogim filozofima i naučnicima, a jedan od njih je i **Isaac Newton** [9]. U ovom radu će biti prikazan jedan način implementacije preslikavanja kanala boja u odgovarajuće zvučne parametre.

1.4 Kreiranje zvučnog sadržaja na osnovu boja

1.4.1 Odabir audio formata

Postoji veliki broj audio formata koji omogućavaju čuvanje zvučnih sadržaja nastalih prirodnim ili vještačkim putem, no većina takvih formata zahtijeva **uzorkovanje signala** te veliku preciznost kako bi izlazni *file*-ovi bili dovoljno kvalitetni. S povećanjem broja uzoraka u jedinici vremena povećava se kvalitet zvučnog sadržaja, no u isto vrijeme, povećava se i **memorija potrebna za njihovo čuvanje**.

Prema Nyquistovom teoremu o uzorkovanju, frekvencija uzorkovanja signala mora biti barem dva puta veća od kritične frekvencije prisutne u izvornom signalu kako bi se signal mogao uspješno rekonstruisati. U slučaju zvuka, ljudski slušni sistem ima najvišu frekvenciju mehaničke rezonance u iznosu od **20 kHz**, te iz tog razloga frekvencija uzorkovanja treba biti **barem 40 kHz** kako ne bi došlo do pojave *aliasinga* (pogrešne rekonstrukcije signala zbog nedovoljnog uzorkovanja) [10]. Na Slici 5 prikazan je trend porasta zauzeća memorije pri porastu kvaliteta prirodnih audio sadržaja, te je vidljivo da veći broj audio formata nema dovoljnu frekvenciju uzorkovanja za rekonstrukciju ulaznih zvučnih signala. [11]



Slika 5: Odnos količine podataka i frekvencije uzorkovanja audio signala [11]

U slučaju pretvaranja slika u zvučni sadržaj, nema razloga za korištenje audio formata koji imaju visoku frekvenciju uzorkovanja, odnosno koji su prilagođeni za uzorkovanje ulaznih signala (poput RIFF, PCM ili DSD formata), budući da je manja frekvencija uzorkovanja, tj. manja preciznost pri kvantizaciji, **sasvim dovoljna** kako bi vještački kreiran zvuk bio dovoljno kvalitetan. Jedan od formata koji je pogodan za ovakvu obradu je **MIDI format**, o kojem će biti više riječi u nastavku.

1.4.2 *Musical Instrument Digital Interface* format

MIDI predstavlja protokol kreiran ranih 1980-ih godina kako bi se omogućilo **kontrolisanje muzičkih instrumenata** i svih njihovih aspekata (visina, jačina tona, pritisak na tipku, držanje tipke, ritam, sinkope i sl.), što nijedan prethodni format nije razmatrao. Glavna razlika između standardnih formata za digitalizaciju zvuka i MIDI formata je u tome što se u ovom formatu **ne čuvaju podaci o uzorcima zvučnog signala**, tako da se omogućava kreiranje *file*-ova koji se ne sastoje od velikog broja uzoraka frekvencija zvuka u vremenu.

U ovom formatu, zvuk poprima jednu od **128 karakterističnih frekvencija** (odnosno, frekvencija predstavljenih 16-bitnim vrijednostima) odabranih za predstavljanje muzičkih nota (pri čemu je najniži ton C_0 kodiran heksadecimalnom vrijednošću `00 00`, te ima frekvenciju od 16.35 Hz, a najviši ton B_9 kodiran je heksadecimalnom vrijednošću `FF FF` i ima frekvenciju 15,804.26 Hz [12]). Na ovaj način postiže se da je veličina *file*-ova koji koriste ovaj protokol **mala u odnosu na druge formate**, jer nema potrebe za čuvanjem velikog broja uzoraka po jedinici vremena, zbog čega je za posljedicu i veličina audio *file*-ova koji koriste MIDI format manja u odnosu na druge audio formate. [11]

Posljedica mapiranja svih frekvencija zvuka u 128 karakterističnih frekvencija je **manji kvalitet zvuka**, no na ovaj način omogućava se kreiranje **potpuno vještačkog signala** (čemu je ovaj format primarno i namijenjen), što je veoma teško ili nemoguće izvršiti koristeći druge formate, budući da to nije njihova primarna namjena (već digitalizacija prirodnog zvučnog signala te što vjernije predstavljanje istog).

Još jedna važna razlika između MIDI formata i standardnih formata je u tome što je zvučni signal u standardnim formatima potpuno definisan, te je kao takav **na svim izlaznim uređajima uvijek isti**. MIDI format definiše veliki broj različitih aspekata važnih za izlazni signal i njegove pojedine dijelove, no sam ton (odnosno, karakteristična frekvencija) predstavlja samo **smjernicu** za konačni izlazni signal, što omogućava da se za isti ulazni *file*

dobiju različiti izlazni signali koristeći različite izlazne uređaje. Budući da su izlazni uređaji najčešće simulatori muzičkih instrumenata, na ovaj način postiže se jednostavna tranzicija između njih, te nije potrebno uvoditi dodatne specifikacije kako bi se omogućio ovaj efekat. Ovakvo nešto nemoguće je izvršiti koristeći standardne audio formate. [11]

U Tabeli 2 prikazana je struktura kompletnog MIDI *file*-a, o kojoj će biti riječi u nastavku.

MIDI Header	*Track Header*	*Track Data*
4D 54 68 64 00 00 00 06 00 XX D E	4D 54 72 6B G	[...] 00 FF 2F 00

Tabela 2: Struktura MIDI *file*-a

File-ovi u MIDI formatu sastoje se od sljedećih dijelova [13]:

1. ***MIDI Header***, koji specificira format *file*-a:
   - **Sekcija A**: Označava da je format *file*-a MIDI, te se sastoji od 4 *byte*-a koji imaju heksadecimalnu vrijednost `4D 54 68 64`;
   - **Sekcija B**: Specificira veličinu ostatka MIDI *header*-a, odnosno broja *byte*-a do početka informacija o samom signalu, te se sastoji od 4 *byte*-a koji imaju heksadecimalnu vrijednost `00 00 00 06` (budući da ostali dijelovi *header*-a imaju fiksnu dužinu);
   - **Sekcija C**: Označava tip MIDI formata. Postoje tri moguća tipa: *tip 0* (osnovni tip), *tip 1* (omogućava korištenje do 2^16 različitih *channels* za definisanje različitih signala koji se pokreću u isto vrijeme) i *tip 2* (koji predstavlja više MIDI *file*-ova u jednom kako bi se definisali različiti šabloni zvuka). Ova sekcija sastoji se od 2 *byte*-a (za najčešće korišteni tip 1, heksadecimalna vrijednost ove sekcije je `00 01`);
   - **Sekcija D**: Označava broj *channels* koji će se koristiti u *file*-u, te se sastoji od 2 *byte*-a;
   - **Sekcija E**: Označava brzinu signala, odnosno ritam zvuka, te se sastoji od 2 *byte*-a.

   U Tabeli 3 prikazana je struktura ovog dijela MIDI *file*-ova.

A	B	C	D	E
4D 54 68 64	00 00 00 06	00 XX	YY YY	ZZ ZZ

Tabela 3: Struktura MIDI *header*-a

2. ***Track Header***, koji daje osnovne informacije o samom zvučnom signalu:
   - **Sekcija F**: Označava početak podataka o zvučnom signalu, te se sastoji od 4 *byte*-a koji imaju heksadecimalnu vrijednost `4D 54 72 6B`;
   - **Sekcija G**: Specificira broj *byte*-a u ostatku *file*-a, odnosno veličinu samog zvučnog signala. Budući da se ova sekcija sastoji od 4 *byte*-a, najveća moguća veličina zvučnog signala je 2^32 *byte*-a.

U Tabeli 4 prikazana je struktura ovog dijela MIDI *file*-ova.

F				G			
4D	54	72	6B	XX	XX	XX	XX

Tabela 4: Struktura *track header*-a

3. ***Track Data***, koji sadrži sam signal, odnosno njegove pojedine dijelove (muzičke tonove). Svaki ton sastoji se iz sljedećih dijelova:

   – ***Timestamp***: Označava broj vremenskih jedinica koje trebaju proći prije početka sljedećeg tona, odnosno njegovo kašnjenje. Za čekanja do 127 otkucaja (`7F`) ova sekcija sastoji se od jednog *byte*-a.

   Za sva veća čekanja, dodaje se po jedan *byte* vrijednosti `81` koja se zatim povećava po potrebi (naprimjer, čekanje od 128 otkucaja specificira se kao `81 7F`, pri čemu se ne uvodi novi *byte* dok se ne dostigne vrijednost `FF 7F` i sl.).

   Razlog za ovakvo kodiranje je što je potrebno biti moguće kodirati velika kašnjenja, a u isto vrijeme treba se moći identificirati mjesto na kojem informacija o kašnjenju završava, što ne bi bilo moguće u slučaju da svi *byte*-i mogu poprimiti vrijednosti od `00` do `FF`;

   – ***Status***: Gornja četiri bita ovog *byte*-a označavaju događaj (*event*), odnosno akciju koju je potrebno izvršiti prije početka sljedećeg tona (moguće vrijednosti prikazane su u tabeli 6), dok donja četiri bita označavaju *channel* na koji se sljedeći ton i sve specificirane postavke odnose.

   – ***Pitch***: Predstavlja karakterističnu frekvenciju, odnosno sljedeći ton u zvučnom signalu, te se sastoji od jednog *byte*-a (vrijednosti od `00` do `7F`);

   – ***Volume***: Predstavlja jačinu tona, te se sastoji od jednog *byte*-a (vrijednosti od `00` do `7F`).

U Tabeli 5 prikazana je struktura ovog dijela MIDI *file*-ova.

Timestamp		*Status*		*Pitch*	*Volume*
[...]	XX	Y	Z	AA	BB

Tabela 5: Struktura pojedinih tonova

Statusni bit	Značenje
8	Ugasi ton
9	Upali ton
A	Pritisak tipke instrumenta (*AfterTouch*)
B	Kontroler (simulacija fizičkih osobina uređaja) [11]
C	Promjena programa (simulacija efekata) [11]
D	Jačina pritiska na tipku
E	Visina tona

Tabela 6: Moguće vrijednosti statusnih bita

Važno je napomenuti da se nakon posljednjeg tona mora definisati kraj zvučnog sadržaja ubacivanjem vrijednosti `00 FF 2F 00`. Ovo je jedna od formi *meta events*, odnosno lažnih događaja koji se kodiraju na isti način kao i stvarni događaji koji proizvode tonove, no kao posljedicu nemaju novi ton, već neku drugu akciju. Svi lažni događaji sastoje se iz četiri dijela:

- Prvi *byte* označava da je u pitanju *meta event* i uvijek ima heksadecimalnu vrijednost `FF`;
- Drugi *byte* specificira vrstu *meta event*-a (informacije o samom *file*-u (broj snimka, autor, godina, ime kompozicije i sl.), tempo, MIDI port, itd.) [11];
- Treći dio specificira broj *byte*-a od kojih se sami podaci sastoje;
- Četvrti dio sadrži same podatke.

U Tabeli 7 prikazana je struktura ovog dijela MIDI *file*-ova.

1	2	3	4
FF	XX	YY	[...]

Tabela 7: Struktura *meta event*-a

2 Implementacija rješenja problema

2.1 Manipulacija slikom

Za svrhu pretvaranja boje u odgovarajući ton, kreirane su dvije funkcije: `toNormalizedHue`, te `imageToSound`, koje obje kao parametar primaju putanju do željene slike. Funkcija `toNormalizedHue` prikazana u Listingu 1 pretvara odabranu sliku u niz **pogodan za biranje tona boje**, koji je opisan u narednim poglavljima. Kako je potrebno vrijednosti piksela pretvoriti u cjelobrojne vrijednosti u intervalu [0,128), korišten je **HSV model**, tačnije, kanal nijanse (*hue*), koji je potom skaliran. Bitno je napomenuti da RGB model **nije korišten** zbog kompleksnosti mapiranja boja u dati interval, te da se sa *grayscale* slikama gubi smisao pretvaranja boje u zvuk.

Kako će veće slike rezultirati većim audio zapisom, korištena je ideja dobivena iz **DCT algoritma**: blokovi slike su kodirani u jedinstvenu notu, tako što se uzela prosječna vrijednost piksela. Ovisno od veličine slike, vrijednosti varijable `factor`, koja predstavlja veličinu bloka, variraju **od 8 do 64**. Nakon računanja prosječne vrijednosti blokova, podaci su kopirani u odgovarajući tip, te se šalju na dalje procesiranje.

Listing 1: Funkcija `toNormalizedHue` za pretvaranje boja u tonove

```
1  //helper function to turn image to normalized hue
2  static int[,] toNormalizedHue(string path) //path to image
3  {
4      Bitmap bitmap = new Bitmap(path); //load image
5      int width = bitmap.Width;
6      int height = bitmap.Height;
7      int[,] note = new int[height, width]; //all of the values [0,128) will
           be stored here
8      for (int i = 0; i < height; i++) {
9        for (int j = 0; j < width; j++) {
10         Color pixel = bitmap.GetPixel(j, i);
11                       float hue = pixel.GetHue(); //get Hue of pixel
12                       hue = hue * 127F / 360F; //normalise to [0,128)
13                       note[i, j] = (int)(Math.Floor(hue)); //round to int
14       }
15     }
16     //since large images will produce longer audio files, the factor
           variable is introduced
17     //similar to DCT, we will use blocks of the image to shorten the audio
           file
18     //bigger images will have a bigger factor
19     List<List<int>> notes = new List<List<int>>();
20     int factor = 8;
21     if (width > 2500 || height > 2500) {
22       factor = 64;
23     }
24     else if (width > 1500 || height > 1500) {
25       factor = 32;
26     }
27     else if (width > 1000 || height > 1000) {
28       factor = 16;
29     }
30     //calculating average normalized hue value for blocks size (factor x
           factor)
31     for (int i = 0; i < height; i += factor) {
```

```
32        List<int> temp = new List<int>();
33        for (int j = 0; j < width; j += factor) {
34          double sum = 0;
35        //total is kept track of since it is possible that a whole block would
             not fit in (instead of padding the image)
36                          int total = 0;
37                          for (int k = i; k < i + factor && k < height; k++) {
38            for (int l = j; l < j + factor && l < width; l++) {
39              sum += note[k, l];
40              total++;
41            }
42                          }
43                          temp.Add((int)(Math.Floor(sum / total)) + 1);
44                          sum = 0;
45                          total = 0;
46        }
47        notes.Add(temp);
48      }
49      //copying to the needed return type
50      int[,] returnNote = new int[notes.Count, notes[0].Count];
51      for (int i = 0; i < returnNote.GetLength(0); i++) {
52        for (int j = 0; j < returnNote.GetLength(1); j++) {
53          returnNote[i, j] = notes[i][j];
54        }
55      }
56      return returnNote;
57  }
```

Funkcija `imageToSound` prikazana u Listingu 2 pretvara odabranu sliku u zvučni zapis uz pomoć klase `MIDIFIle` opisane u nastavku. Nakon što se pozove `toNormalizedHue`, dobiveni dvodimenzionalni niz **dodaje za svaki piksel notu** metodom `addNote`, nakon čega se kreira audio zapis uz pomoć metode `createMIDIFile` (u nastavku su objašnjene pomenute metode).

Listing 2: Funkcija `imageToSound` za kreiranje audio *file*-a

```
1  static void imageToSound(string path) //path to image
2  {
3              MIDIFile m = new MIDIFile();
4              int[,] note = toNormalizedHue(path);
5              m.setVolume(127);
6              for (int i = 0; i < note.GetLength(0); i++) {
7                  for (int j = 0; j < note.GetLength(1); j++) {
8                      m.addNote(note[i, j]); //create MIDI file of notes
                           acquired from the hue channel of the image
9                  }
10             }
11             m.createMIDIFile("test"); //create MIDI file
12 }
```

2.2 Manipulacija zvukom

Treća faza konverzije slika u boji u zvuk je **kreiranje MIDI *file*-a** s tonovima koji predstavljaju odgovarajuće boje od kojih se slika sastoji. Kako bi se to omogućilo, kreirana je klasa `MIDIFile`, čiji atributi predstavljaju pojedine dijelove MIDI *file*-a koji se sukcesivno kreira, odnosno koji ima predefinisane dijelove (*header* dijelove), a dio koji se odnosi na same podatke (tj. tonove) inicijalno je prazan. Atributi klase `MIDIFile` prikazani su u Listingu 3.

Listing 3: Klasa MIDIFile i njeni atributi

```
public class MIDIFile
    {
        #region MIDIHeader
        string format = "4D546864"; // specifies that the file format is
            MIDI (MThd)
        string headerSize = "00000006"; // specifies the number of bytes of
                    // the following three parts of the MIDI header (SMF)
        string type = "0001"; // specifies the type of MIDI (0, 1 or 2)
        string tracks = "0001"; // specifies the number of tracks (0 -
            65.536)
        string speed = "0080"; // specifies the speed of music
        #endregion
        #region TrackHeader
        string start = "4D54726B"; // specifies the track start (MTrk)
        string noOfBytes = ""; // specifies the number of bytes in the track
        #endregion
        #region Track
        string data = ""; // specifies the musical notes in the track
        string rhythm = "8100"; // specifies the speed of each note
        string volume = "60"; // specifies the volume of each note
        string end = "00FF2F00"; // specifies the track end
        #endregion
    }
```

Klasa `MIDIFile` omogućava **dodavanje novog tona** na vrlo jednostavan način. Kao parametar se prima cjelobrojna vrijednost između 0 i 127 (koja predstavlja karakterističnu frekvenciju tona), te se zatim provjerava da li je u pitanju prvi ton ili ne. Ukoliko je u pitanju prvi ton, potrebno je poslati signal da se pritišće prva tipka, dok je u svakom drugom slučaju dovoljno dodati samo broj otkucaja koji se čekaju do početka sljedećeg tona. Zatim se dodaje vrijednost koja označava karakterističnu frekvenciju (uz dodatnu provjeru da li karakteristična vrijednost ima dva *byte*-a, te ukoliko nema, drugi *byte* se manuelno dodaje), te trenutno postavljena vrijednost jačine zvuka. Ova funkcionalnost implementirana je u funkciji `addNote`, čija je struktura prikazana u Listingu 4.

Listing 4: Metoda za dodavanje tonova u MIDI *file*

```
// adds a new note (range from 0 to 127) to the track
        public void addNote(int note)
        {
            if (data.Length == 0) data += "800090"; // the first note needs
                                // to send the signal to turn the notes on
            else data += rhythm;
            if (note < 16) data += "0";
            data += note.ToString("X") + volume;
        }
```

Nakon dodavanja svih željenih tonova, potrebno je kreirati *.mid file* koji se sastoji od svih prethodno navedenih dijelova. Nakon dodavanja posljednjeg, "tihog" tona koji signalizira otpuštanje pritiska na tipku, svi dijelovi (*headers* i *data*) sastavljaju se u **jedinstvenu varijablu** koja se sastoji od heksadecimalnih vrijednosti prikazanih u obliku karaktera. Ti karakteri zatim se pretvaraju u binarne vrijednosti (2 po 2 karaktera, kako bi se dobila 16-bitna vrijednost) koje se redom upisuju u **binarni *file* s ekstenzijom *.mid***. Ova funkcionalnost implementirana je u funkciji `createMIDIFile`, čija je struktura prikazana u Listingu 5.

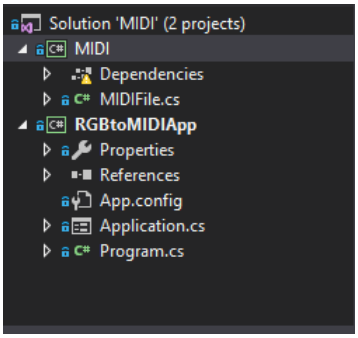
Listing 5: Spajanje svih dijelova MIDI *file*-a i kreiranje *.mid file*-a

```
1         // puts all file parts together and writes to .mid file
2         public void createMIDIFile(string path)
3         {
4             data += rhythm + "B07B00"; // the final note just turns off the
                  music
5             noOfBytes = ((data.Length + end.Length) / 2).ToString("X");
6                   // previously skipped part of TrackHeader
7             while (noOfBytes.Length < 8) noOfBytes = "0" + noOfBytes;
8                   // the noOfBytes part needs to have 8 bytes
9             string file = format + headerSize + type + tracks + speed +
10                          start + noOfBytes + data + end; // putting all
                                parts together
11            // writing to .mid file - hexadecimal code is converted to
                  binary and then written to file
12            var stream = new FileStream(path + ".mid", FileMode.Create,
                  FileAccess.ReadWrite);
13            var twoCharacters = new StringBuilder(); // two bytes are used
                  for every conversion (16 bits - two hex numbers)
14            var singleByte = new byte[1]; // two binary bytes to which the
                  hex numbers will be converted
15            foreach (var character in file)
16            {
17                twoCharacters.Append(character); // adding one hex character
                      to the 16-bit variable
18                if (twoCharacters.Length == 2) // added two characters -
                      reached 16 bits
19                {
20                    singleByte[0] = (byte)Convert.ToByte(twoCharacters.
                          ToString(), 16); // conversion from hex to bin
21                    stream.Write(singleByte, 0, 1); // writing bin to file
22                    twoCharacters.Clear(); // starting over again with new
                          characters
23                }
24            }
25            stream.Close();
26        }
```

2.3 Aplikacija za pretvaranje slika u boji u zvučni sadržaj

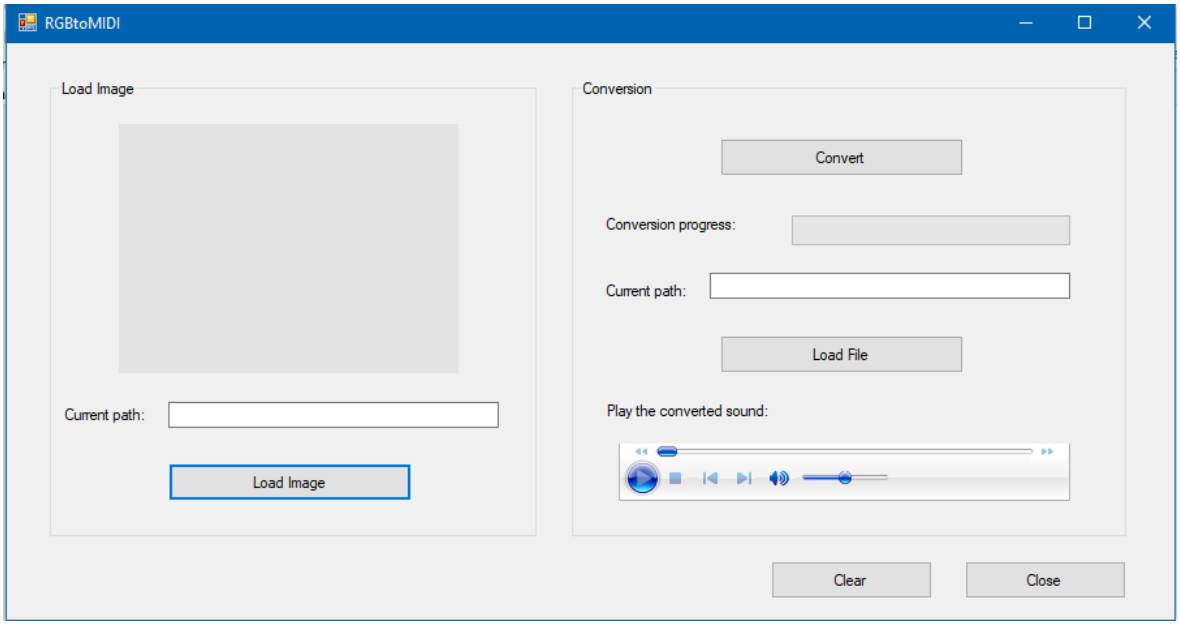
Nakon što su analizirani, opisani i implementirani pojedinačni dijelovi manipulacije nad slikom i zvukom, u ovom dijelu će biti opisano povezivanje svih komponenti u **jednu cjelinu** koju čini *desktop* aplikacija. Aplikacija je implementirana upotrebom *Windows Forms* projekta u okviru *Visual Studio* razvojnog okruženja.

Implementirano rješenje sastoji se od **dva projekta**, kako je prikazano u *Visual Studio Solution Explorer*-u na Slici 6.



Slika 6: Projekti od kojih se aplikacija sastoji

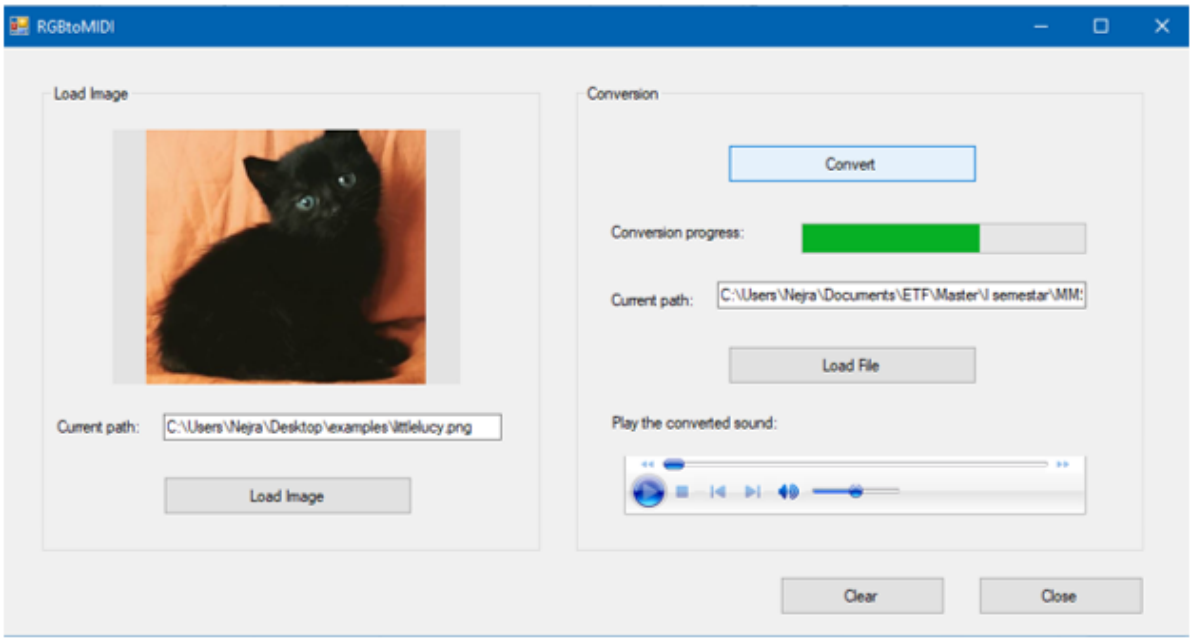
Prvi projekat pod nazivom `MIDI` predstavlja biblioteku klasa (*Class Library*) i sastoji se od metoda pomoću kojih se dodaju note te kreira odgovarajući MIDI *file*. Drugi projekat je ***Windows Forms* projekat** u kojem je dizajniran izgled aplikacije i implementirana logika upravljanja korisničkim kontrolama. Na Slici 7 prikazan je izgled aplikacije.



Slika 7: Izgled aplikacije

Nakon pokretanja aplikacije, korisniku se otvara forma sa odgovarajućim korisničkim kontrolama. Klikom na dugme *Load Image*, korisniku je omogućen **odabir željene slike za konverziju**. Nakon učitavanja slike, klikom na dugme *Convert* vrši se **konverzija odabrane slike u MIDI *file***, pri čemu je putanja do kreirane datoteke prikazana u tekst polju na aplikaciji.

Ako je konverzija uspješno završena, korisniku se omogućava **učitavanje kreiranog *file*-a**. Kreirani MIDI *file* se može reproducirati u samoj aplikaciji pomoću *Windows Media Player* korisničke kontrole. Dok se vrši konverzija, moguće je pratiti progres napretka pomoću *Progress Bar* korisničke kontrole. Progres pri konverziji nakon odabira slike, kao i putanja *file*-a u koji će se konverzija snimiti prikazani su na Slici 8.



Slika 8: Izgled aplikacije tokom izvršavanja procesa konverzije

Pored osnovne implementacije upravljanja događajima na korisničkim kontrolama, za ovaj rad su od najvećeg značaja dijelovi implementacije aplikacije vezani za **objedinjavanje implementiranih komponenti u jednu cjelinu**. Ti događaji se dešavaju klikom na dugme za konverziju i reproduciranjem *file*-a.

U Listingu 6 prikazano je upravljanje klikom na dugme *Convert*. Dakle, postavljaju se odgovarajuće vrijednosti za *Progress Bar* kontrolu, te se poziva **odgovarajuća nit** koja obavlja konverziju odabrane slike u MIDI *file*.

Listing 6: Pokretanje konverzije slike u audio *file*

```
1 private void buttonConvert_Click(object sender, EventArgs e)
2         {
3             progressBar1.Maximum = 100;
4             progressBar1.Step = 1;
5             progressBar1.Value = 0;
6             backgroundWorker1.RunWorkerAsync();
7         }
```

U Listingu 7 prikazane su ostale operacije vezane za pokretanje niti za konverziju slike, pozivanje metode za pomenutu konverziju koja je opisana u jednom od prethodnih poglavlja, kao i ažuriranje *Progress Bar*-a i korisničkih kontrola kada se konverzija završi.

Listing 7: Vršenje konverzije u zasebnoj niti

```
1 //turn hue to sound
2         private void backgroundWorker1_DoWork_1(object sender,
              DoWorkEventArgs e)
3         {
4             var backgroundWorker = sender as BackgroundWorker;
5             MIDIFile m = new MIDIFile();
```

```
6               int[,] note = toNormalizedHue(imagePath);
7               m.setVolume(127);
8               int noOfSteps = note.GetLength(0) * note.GetLength(1);
9               int reportProgressStep = noOfSteps / 100 + 1;
10              int step = 0;
11              for (int i = 0; i < note.GetLength(0); i++)  {
12                  for (int j = 0; j < note.GetLength(1); j++) {
13                      m.addNote(note[i, j]); //create MIDI file of notes
                            acquired from the hue channel of the image
14                      step++;
15                      if (step \% reportProgressStep == 0) backgroundWorker.
                            ReportProgress(step / reportProgressStep);
16                  }
17              }
18              m.createMIDIFile("test"); //create MIDI file
19              backgroundWorker.ReportProgress(100);
20              musicFilePath = Directory.GetCurrentDirectory() + "\\test.mid";
21          }
22
23          private void backgroundWorker1_ProgressChanged_1(object sender,
                ProgressChangedEventArgs e)
24          {
25              progressBar1.Value = e.ProgressPercentage;
26          }
27
28          private void backgroundWorker1_RunWorkerCompleted(object sender,
                RunWorkerCompletedEventArgs e)
29          {
30              textBox1.Text = musicFilePath;
31              string x = textBox1.Text;
32          }
33      }
```

Nakon što se *file* uspješno kreira, implementacija koja omogućava njegovo pokretanje prikazana je u Listingu 8.

Listing 8: Pokretanje reprodukcije MIDI *file*-a

```
1 private void axWindowsMediaPlayerPlay_Enter(object sender, EventArgs e)
2           {
3               axWindowsMediaPlayerPlay.URL = musicFilePath;
4           }
```

Dakle, implementirana *desktop* aplikacija spaja dijelove implementacije u jednu cjelinu i na jednostavan način omogućava pokretanje tih dijelova i prikaz rezultata njihovog rada. Cijela implementacija dostupna je na linku: https://github.com/ehlymana/RGBtoMIDI.

3 Zaključak

Razvoj tehnologije omogućio je jednostavnu i brzu manipulaciju multimedijalnim sadržajima. Postoji veliki broj različitih formata slika te gotovih programskih rješenja i biblioteka koje omogućavaju njihovu analizu, izmjenu ili ekstrakciju onih dijelova koji su važni za određenu primjenu. S druge strane, iako je obrada zvukom manje popularna, postoji veliki broj audio formata koji omogućavaju tretiranje zvuka na različite načine, kompresiju ili kreiranje potpuno vještačkih zvučnih signala, što također može biti korisno i od velike pomoći za različite primjene. Najvažnije je omogućavanje izvršavanja svih ovih obrada **u realnom vremenu**, što otvara vrata velikom broju mogućnosti.

Jedna od takvih mogućnosti je učitavanje slike, njeno razlaganje na *color channels* te njihovo pretvaranje u audio *file* koji je moguće reproducirati gotovo odmah. Na ovaj način omogućava se da boje, koje neki ljudi nisu u stanju vidjeti, budu pretvorene u karakterističnu frekvenciju koja, na isti način kao što oko prepoznaje karakteristične boje, bude prepoznata od strane slušnog sistema. Na ovaj način moguće je ukloniti prepreku koju predstavlja **nemogućnost viđenja boja**, odnosno omogućiti ljudima koji nisu u stanju da razlikuju boje (daltonizam) ili koji vide samo nijanse sive boje (ahromatopsija) da **"čuju" boje**, odnosno da razlikuju boje na osnovu različite frekvencije zvuka koja se proizvodi na osnovu mapiranja boja u muzičke tonove.

Naravno, ovakav sistem nije savršen, jer postoji na milione različitih nijansi koje je ljudsko oko u stanju razlikovati, dok postoji samo 128 različitih muzičkih tonova, čime se većina nijansi zapravo **aproksimira** u jednu od karakterističnih frekvencija. Ovo je posljedica činjenice da vidni i čulni sistemi ne funkcionišu na isti način: u bazelarnoj membrani dolazi do pojave **frekvencijskog maskiranja**, pri čemu se bliske frekvencije detektuju kao jedna, što predstavlja jednu vrstu aproksimacije [10]. Iz ovog razloga i postoji samo 128 različitih muzičkih tonova, zbog čega se nameće zaključak da je vršenje aproksimacija **neizbježno** pri mapiranju boja u frekvencije zvuka.

Iako je potpuno preslikavanje skupa boja u skup zvukova nemoguće, preslikavanje 128 boja je ipak dovoljno detaljan postupak kako bi se stvorio osjećaj razlikovanja boja. Na ovaj način omogućava se pojava **sinestezije**, odnosno nakon određenog vremena korištenja sistema za pretvaranje slika u boji u zvukove, ljudski čulni sistem koji nije u mogućnosti razlikovati boje početi će vršiti automatsku identifikaciju boja koje ne vidi na osnovu zvuka koji čuje. Omogućavanje viđenja boja i u slučaju kada je čulni organ oštećen ili defektan, bez obzira s kolikom aproksimacijom, veliki je uspjeh, jer može predstavljati veliku pomoć ljudima koji nisu u stanju razlikovati ili vidjeti boje, a za realizaciju ovakvog sistema nisu potrebne velike sume novca niti komplikovani postupci.

Reference

[1] J. Farley. (2019, February) A short guide to color models. [Online]. Available: https://www.sitepoint.com/a-short-guide-to-color-models

[2] (2019, February) Understanding color models and spot color systems. [Online]. Available: https://www.designersinsights.com/designer-resources/understanding-color-models

[3] J. Vlahos. (2019, February) Everything you need to know about cmyk color. [Online]. Available: https://www.printi.com/blog/cmyk-color

[4] (2019, February) Color theory. [Online]. Available: http://www.pengadprinting.com/content/color-theory-part-ii-types-color-and-uses-0

[5] C. Forsyth. (2019, February) Hand-coding a color wheel with canvas. [Online]. Available: https://medium.com/\spacefactor\@m{}bantic/hand-coding-a-color-wheel-with-canvas-78256c9d7d43

[6] M. M. et al, *From Color to Sound: Assessing the Surrounding Environment.* Lisabon, Portugal: Universidade Nova de Lisboa, 2012.

[7] S. C. T. Henriques and M. Mengucci. (2019, January) Secção de ciência e tecnologia. [Online]. Available: http://ctp.di.fct.unl.pt/~sc/SeeThroughSound/index.html

[8] D. L. Datteri and J. N. Howard, *The Sound Of Color.* Evanston, Illinois: Conference on Music Perception & Cognition.

[9] I. Newton, *Opticks: or, A Treatise of the Reflections, Refractions, Inflections and Colours of Light. Also two Treatises of the Species and Magnitude of Curvilinear Figures.* United States: Palala Press, 2015.

[10] H. Šupić, *Kompresija Podataka.* Sarajevo, Bosnia and Herzegovina: Univerzitet u Sarajevu, 2016.

[11] F. Rumsey, *Desktop Audio Technology: Digital Audio and MIDI Principles*, ser. Music Technology Series. Oxford, United Kingdom: Focal Press, 2004, vol. IV.

[12] (2019, February) Table of musical notes and their frequencies and wavelengths. [Online]. Available: https://www.liutaiomottola.com/formulae/freqtab.htm

[13] (2019, February) A crash course on the standard midi specification. [Online]. Available: http://www.skytopia.com/project/articles/midi.html