SER5 Uspeh ni naključen

Srednja elektro-računalniška šola Maribor

DEKORATIVNI LED PANEL

Aplikativni inovacijski predlogi in projekti

Šola: SREDNJA ELEKTRO-RAČUNALNIŠKA ŠOLA MARIBOR

Avtorja: MARK AHMAN, DARKO PLIBERŠEK

Mentorica: HELENA STERNAD

Maribor, 2023

KAZALO

1. UVOD 1
2. PROBLEMI IN REŠITVE 2
2.1 LED diode 2
2.2 Hitrost osveževanja 3
2.3 Šestkotni piksli 3
2.4 Izbira elektronike 4
3. NAČRT 5
3.1 Lesena plošča 5
3.2 LED trakovi 5
3.3 Plastično satovje 6
3.4 Plastični robovi 7
3.5 Akrilno steklo 8
4. IZDELAVA 10
4.1 Plošča in lepljenje trakov 10
4.2 Električna vezava trakov 10
4.3 Robovi in stranski trakovi 12
4.4 Satovje 13
4.5 Prosojno ohišje 13
5. PROGRAMIRANJE 16
5.1 Zasnova programa 16
5.2 Funkcija za prikaz slike 18
5.3 Funkcija za zlivanje slike čez rob 19
5.4 Funkcija za upravljanje z WiFi-jem 20
5.5 Spreminjanje in upravljanje efektov 22
5.6 Efekti 24

5.6.1 Diagonalni mavrični val ... 24
5.6.2 Radialni mavrični val ... 26
5.6.3 Bledeče točke ... 28
5.6.4 Prikazovalnik frekvenčnega spektra ... 30
6. PRIMERJAVA S PODOBNIMI IZDELKI ... 32
7. DRUŽBENA ODGOVORNOST ... 33
8. ZAKLJUČEK IN SKLEPI ... 34
9. VIRI IN LITERTURA ... 35

KAZALO SLIK

Slika 1: Prikaz vezave LED trakov ... 2
Slika 2: Razdalja med LED trakovi ... 3
Slika 3: Načrt - lesena plošča ... 5
Slika 4: Načrt - LED trakovi ... 6
Slika 5: Načrt - posamezen del satovja ... 6
Slika 6: Načrt - sestavljeno satovje ... 7
Slika 7: Načrt - Robovi ... 7
Slika 8: Načrt - Sestavljen panel brez stekla ... 8
Slika 9: Načrt - kotni deli ohišja ... 8
Slika 10: Načrt - sestavljeno ohišje ... 9
Slika 11: Načrt - sestavljen panel ... 9
Slika 12: Izdelava - lepljenje trakov ... 10
Slika 13: Izdelava - vezava trakov ... 11
Slika 14: Izdelava - vezava trakov od blizu ... 11
Slika 15: Izdelava - stranski trakovi ... 12
Slika 16: Izdelava - pritrjeni robovi in dopolnjena vezava ... 12
Slika 17: Izdelava - panel s satovjem ... 13
Slika 18: Izdelava - lepljenje stranic ohišja ... 14
Slika 19: Izdelava - sestavljeno ohišje ... 14

Slika 20: Končan panel 15
Slika 21: Definicije, globalne spremenljivke in začetne funkcije 17
Slika 22: Funkcija za prikaz slike 18
Slika 23: Funkcija za zlivanje slike čez rob 19
Slika 24: Ukaz za dodelitev funkcije jedru 0, z minimalno prioriteto 20
Slika 25: Funkcija za upravljanje z WiFi-jem 21
Slika 26: Prikaz delovanja spremembe efekta preko spletnega brskalnika 22
Slika 27: Funkcija loop() 23
Slika 28: Funkcija za iteracijo efekta do spremembe 23
Slika 29: Efekt diagonalnega mavričnega vala in QR koda do posnetka 24
Slika 30: Koda efekta diagonalnega mavričnega vala 25
Slika 31: Efekt radialnega mavričnega vala in QR koda do posnetka 26
Slika 32: Koda efekta radialnega mavričnega vala 27
Slika 33: Efekt bledečih točk in QR koda do posnetka 28
Slika 34: Koda efekta bledečih točk 29
Slika 35: Efekt frekvenčnega prikazovalnika in QR koda do posnetka 30
Slika 36: Koda efekta prikazovalnika frekvenčnega spektra 31

KAZALO TABEL

Tabela 1: Primerjava spodobnimi izdelki 32

POVZETEK

Namen inovacijskega predloga je izdelati LED panel, ki se pritrdi na prazno belo steno, kot moderna alternativa slikam in sobi tako doda poseben čar z dinamičnimi svetlobnimi efekti. Predstavljena bo izdelava samega izdelka s sodobnimi tehnikami, kot so 3D tiskanje in lasersko izrezovanje ter programiranje nekaterih svetlobnih efektov v programskem okolju Arduino. Cilj je narediti izdelek čim cenejši z uporabo standardne elektronike in doma narejenih delov; čim tanjši, da je čim bolj vizualno in prostorsko nemoteč ter čim lažji, da za steno ne povzroča prevelikih obremenitev. Izdelek bo odstopal od podobnih produktov na trgu z uporabo pikslov v obliki šestkotnikov in svetlobnih učinkov, ki reagirajo na glasbo, upravljati pa ga bo mogoče preko aplikacije za telefon. Meniva, da je izdelek zanimiv in privlačen za ljubitelje svetlobnih učinkov.

Ključne besede: LED, panel, dekoracija, moderno, svetlobni učinki

SUMMARY

The purpose of this innovation proposal is to create an LED panel that can be mounted on a blank white wall, offering a modern alternative to traditional paintings while adding a unique charm to the room with its dynamic lighting effects. The paper will cover the production process using modern techniques such as 3D printing and laser cutting, as well as programming various lighting effects through the Arduino development environment. The objective is to make the product as affordable as possible, using readily available components and homemade parts, and to make it as slim and lightweight as possible to minimize its visual and spatial impact and reduce stress on the wall. This product will distinguish itself from similar items on the market by utilizing hexagonal pixels and music-reactive lighting effects, which can be controlled through a mobile application. We believe that this product will be appealing and engaging for lighting effects enthusiasts.

Key words: LED, panel, decoration, modern, lighting effect

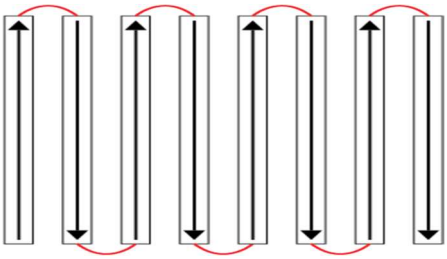
1. UVOD

Belo obarvane stene izgledajo moderno in sobi dajejo občutek prostornosti, vendar jih je treba okrasiti, da ne izgledajo dolgočasno. Večina ljudi se odloči za umetnine ali dekorativne police, nekateri pa si želijo nekaj bolj dinamičnega in svetlečega. Seveda na tržišču že obstajajo znani dekorativni LED izdelki, na primer Nanoleaf, vendar so pogosto precej dragi in nobeden ni takšen, kot sva si ga zamislila. Želela sva izdelek, ki izgleda kot en samostojni LED panel, je čim bolj neopazen ko je izklopljen, ima široko izbiro prilagodljivih svetlobnih učinkov - tudi takšne ki reagirajo na glasbo, se ga da nadzorovati z mobilnim telefonom in je cenovno ugoden. Ker na tržišču takšnega nisva našla, sva se odločila, da ga bova naredila sama.

2. PROBLEMI IN REŠITVE

2.1 LED diode

Ideja izdelka je da sliko prikaže v dveh dimenzijah, torej v bistvu matrika led lučk. Prikazovalnik vsebuje 51×25 lučk, izmed katerih mora vsaka znati prikazati tri barve. Če bi uporabili standardne RGB LED diode, ki morajo biti z mikrokrmilnikom povezane s tremi žicami, bi torej potrebovali kar 3.825 povezav, kar pa je seveda bistveno preveč. Iz tega namena uporabimo »adresibilne« LED trakove, ki omogočajo individualno nadziranje teoretično neomejenega števila lučk z eno samo podatkovno povezavo. Te trakove uporabimo kot stolpce matrike in jih na koncih povežemo v eno verigo tako, da efektivno še vedno služijo kot en sam neprekinjen trak (kot prikazuje slika 1), mikrokrmilnik pa s pomočjo funkcije ugotovi kako mora podatke poslati, da se prikažejo kot smiselna slika.



Slika 1: Prikaz vezave LED trakov

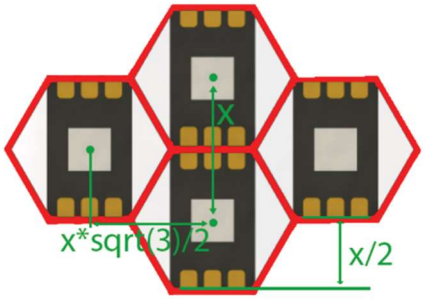
Vir: Lasten vir

2.2 Hitrost osveževanja

Adresibilni LED trakovi so dobra izbira, ker potrebujejo le eno podatkovno povezavo, vendar imajo problem, da pri večjem številu lučk potrebujejo nekaj časa, da vse posodobijo. Sodeč po podatkovnem listu čipov na izbranem adresibilnem traku (WS2812B) se podatki prenašajo s hitrostjo 800kb/s, vsaka lučka pa potrebuje 24 bitov, da prikaže željeno barvo (8 bitov za vsak barvni kanal). Tako lahko izračunamo, da se čas osveževanja poveča za 30 µs na lučko ($\frac{24b}{800.000b/s} = 0{,}00003s$). Če želimo zagotoviti, da lahko izdelek prikaže 60 sličic na sekundo za čim bolj gladke animacije, lahko torej uporabimo največ 555 lučk na trak ($\frac{\frac{1s}{60}}{0{,}00003s} = 555{,}\overline{55}$). Ker potrebujemo 51×25 torej 1.275 lučk, lahko problem rešimo tako, da trakove razdelimo na 3 verige po 17 trakov (425 lučk). Če želimo da ima panel lučke tudi po stranicah rabimo vse skupaj torej štiri verige LED trakov.

2.3 Šestkotni piksli

Ker sva želela narediti izdelek bolj unikaten in zanimiv, sva se odločila, da bo prikazovalnik sestavljen iz šestkotnih pikslov. To dosežemo tako, da lučke postavimo v plastično satovje , na vrh pa damo zamegljeno akrilno steklo, ki svetlobo razbije v obliko šestkotnika. Zaradi geometrije mreže šestkotnikov, moramo zagotoviti, da je vsak drugi LED trak zamaknjen za pol lučke in da so trakovi za $\frac{\sqrt{3}}{2}$ -krat bolj narazen, kot lučke na trakovih (slika 2).



Slika 2: Razdalja med LED trakovi

Vir: Lasten vir

2.4 Izbira elektronike

Za osvetlitev sva izbrala **WS2812B** adresibilne LED trakove, zaradi relativno nizke cene in podpore iz strani Arduino knjižnice **FastLED**, ki je naredila programiranje veliko lažje. Za mikrokrmilnik sva izbrala ESP32 zaradi visoke zmogljivosti in vgrajene povezljivosti z WiFi-jem.

3. NAČRT

Pred začetkom izdelave je bil celoten izdelek 3D modeliran s pomočjo programa Fusion 360. To je bilo pomembno, ker so nekateri deli 3D natisnjeni in ker je zelo pomagalo pri predstavi kako bi naj končni izdelek izgledal.

3.1 Lesena plošča

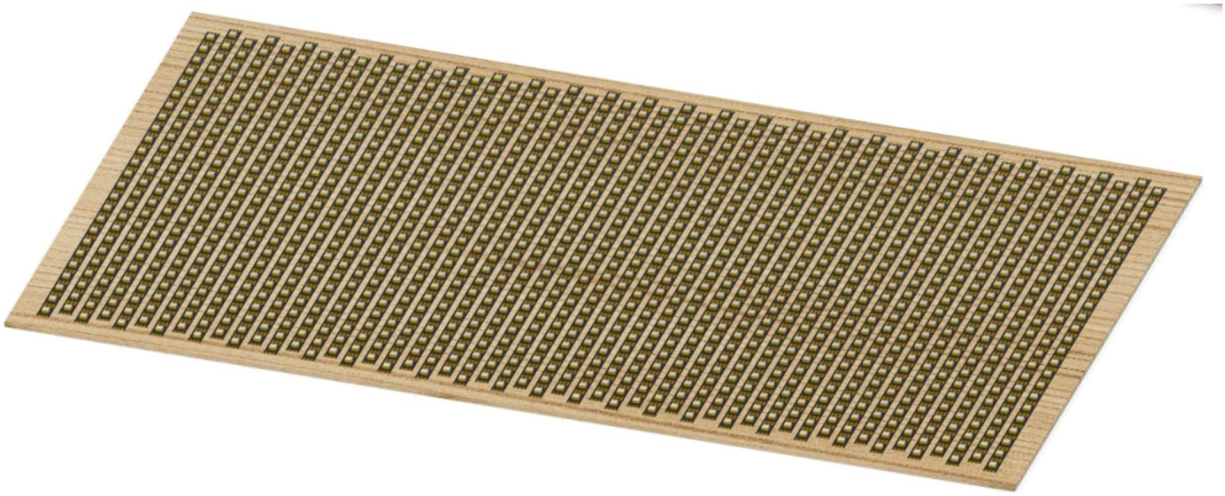
Osnova izdelka je 4×455×770 mm velika lesena plošča, na katero so pritrjeni LED trakovi in ostali deli (slika 3).

Slika 3: Načrt - lesena plošča

Vir: Lasten vir

3.2 LED trakovi

Na ploščo so prilepljeni LED trakovi, kar je prikazano na sliki 4. Kot že prej omenjeno je zelo pomembno, da pazimo na razmak med trakovi.

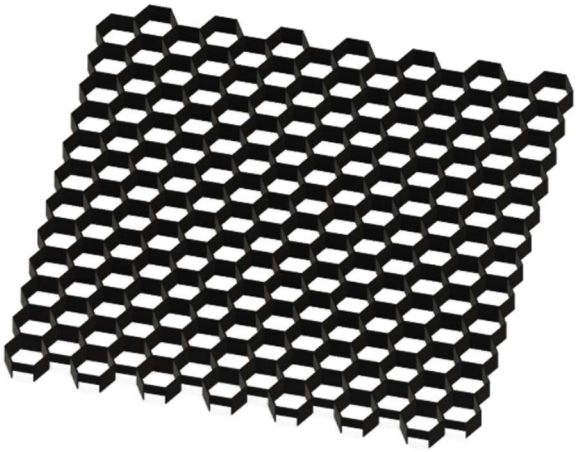


Slika 4: Načrt - LED trakovi

Vir: Lasten vir

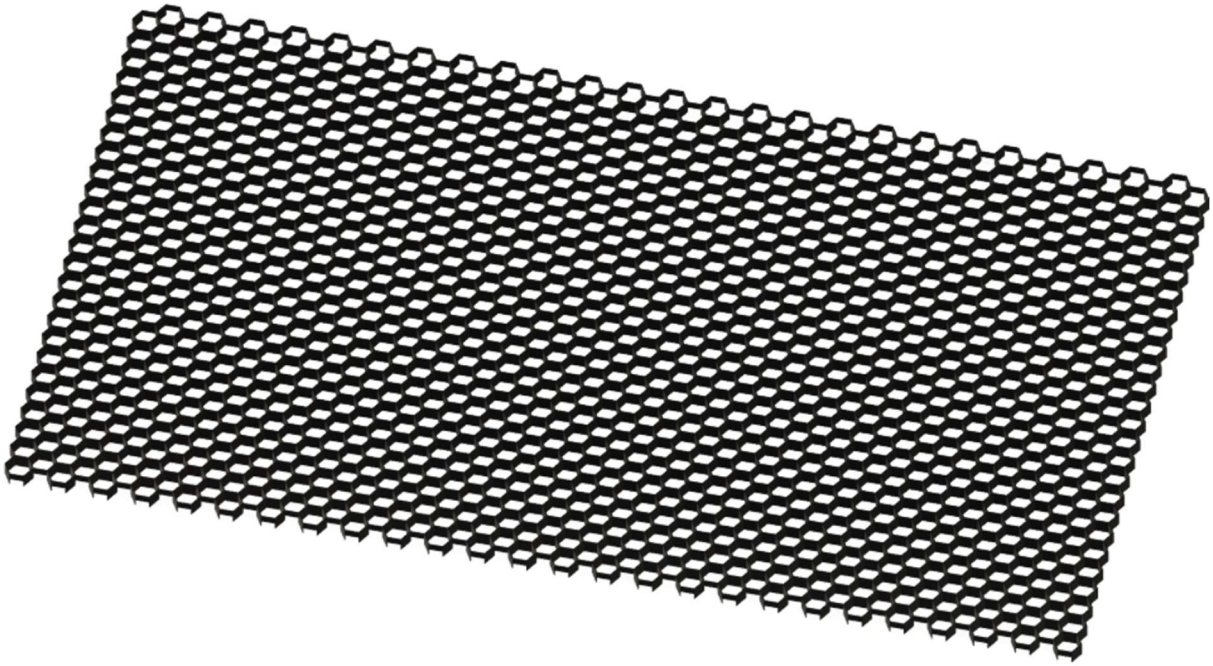
3.3 Plastično satovje

Da se svetloba lučk razbije v šestkotnike, morajo biti lučke obdane z neprosojnim satovjem, ki je bilo kasneje 3D natisnjeno. Oblikovano je tako, da pride vsaka dioda LED trakov v eno celico, robovi med celicami pa so debeli le 0.8 mm, zaradi manjše porabe materiala. Celice na zgornjem in spodnjem robu satovja imajo odprtino za napajalne in podatkovne žičke. Zaradi precejšne velikosti mora biti satovje razdeljeno na 8 delov, ki se po tiskanju prilepijo skupaj. Načrt prikazujeta sliki 5 in 6.



Slika 5: Načrt - posamezen del satovja

Vir: Lasten vir



Slika 6: Načrt - sestavljeno satovje

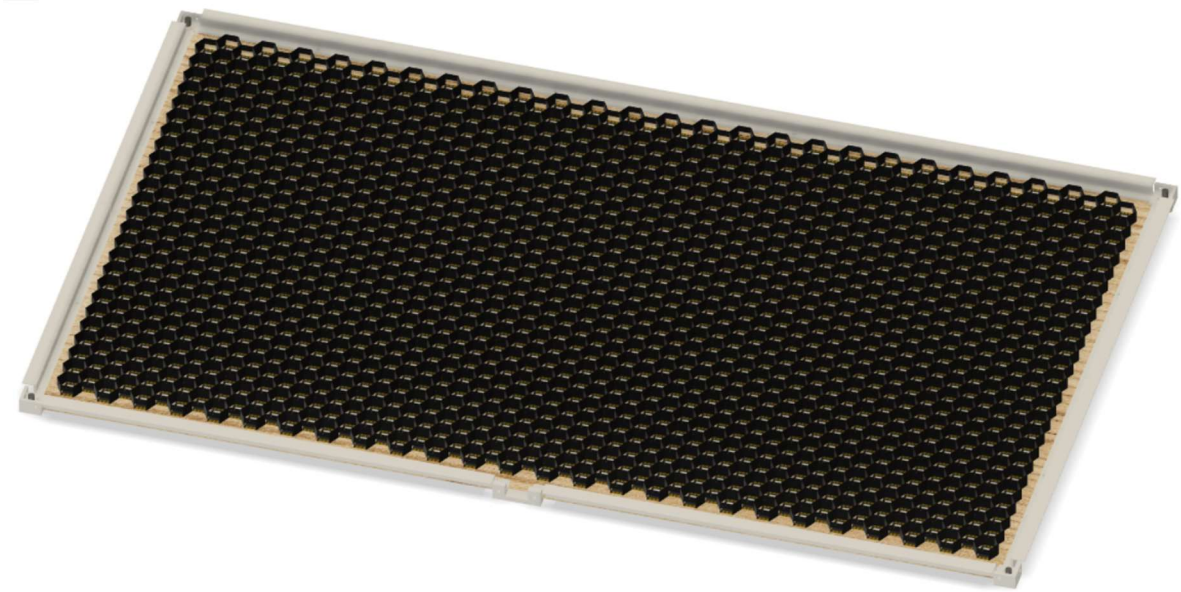
Vir: Lasten vir

3.4 Plastični robovi

Na robove plošče so prilepljeni 3D natisnjeni deli, na katere prilepimo še stranske LED trakove. Na zgornji strani imajo strešico, ki poskrbi, da je svetloba vidna le na stranicah panela, ne pa tudi na zgornji strani. Prav tako na kote plošče namestimo dele, ki poskrbijo da lahko akrilno steklo v naslednjem koraku pritrdimo na celotno konstrukcijo. Nazadnje še na sredino spodnje stranice namestimo kos, skozi katerega bodo potekali napajalni in podatkovni kabli. Načrt prikazujeta sliki 7 in 8.

Slika 7: Načrt - Robovi

Vir: Lasten vir

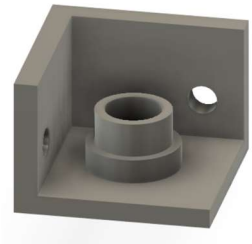


Slika 8: Načrt - Sestavljen panel brez stekla

Vir: Lasten vir

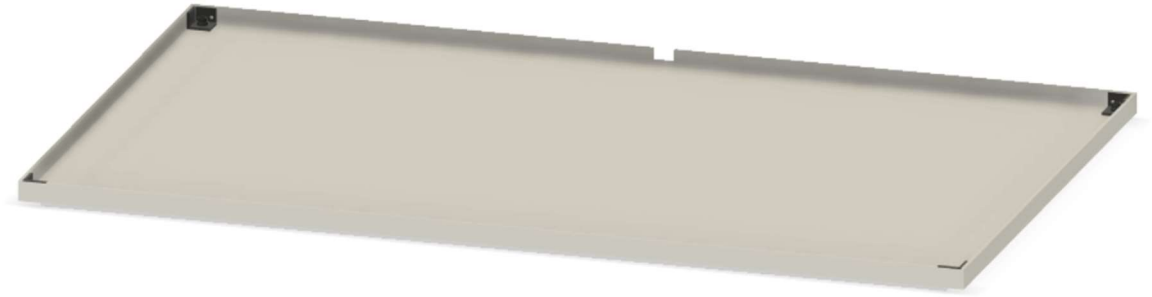
3.5 Akrilno steklo

Zadnji kos sestavljanke je ohišje iz akrilnega stekla. Sestavljeno je iz petih kosov 3 mm debelega, lasersko izrezanega zamegljenega pleksi stekla – zgornja površina (796×481 mm), dve krajši stranici (481×14 mm) in dve daljši stranici (790×14 mm), izmed katerih ima ena na sredini odprtino za kable. Ti so pritrjeni v obliko škatle s pomočjo štirih plastičnih delov, ki pomagajo tudi pri namestitvi ohišja na sam panel (slika 9 in 10). Na sliki 11 je prikazan sestavljen panel.



Slika 9: Načrt - kotni deli ohišja

Vir: Lasten vir



Slika 10: Načrt - sestavljeno ohišje

Vir: Lasten vir

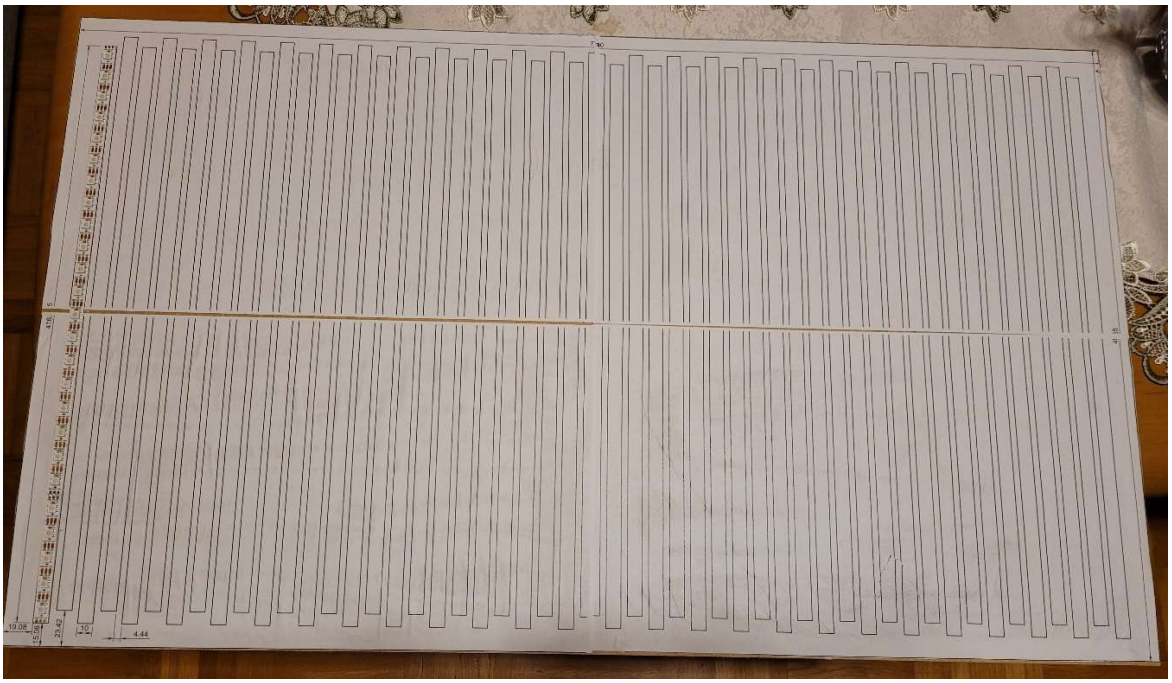
Slika 11: Načrt - sestavljen panel

Vir: Lasten vir

4. IZDELAVA

4.1 Plošča in lepljenje trakov

Začela sva z 4 mm debelo leseno ploščo, ki je bila izrezana na 770 × 455 mm. Za lažje lepljenje trakov je bil na štiri A3 liste natisnjen načrt postavitve. Na ploščo so bili nato z lesnim lepilom nalepljeni listi papirja, na papir pa so bili med črte prilepljeni LED trakovi. Paziti je bilo treba, da so bili trakovi nalepljeni tako, da se je smer poteka podatkov izmenjevala (slika 12).



Slika 12: Izdelava - lepljenje trakov

Vir: Lasten vir

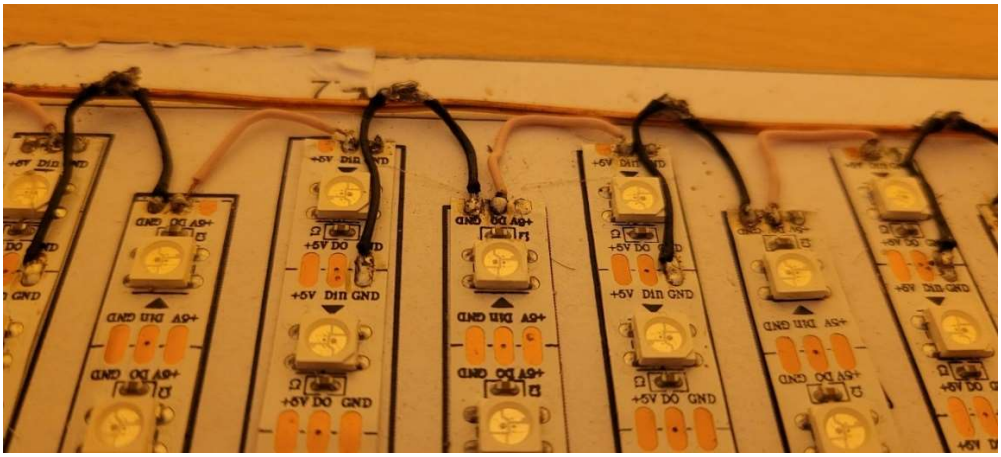
4.2 Električna vezava trakov

Da dobijo vse lučke enako električnega toka in svetijo z enako močjo, so bili LED trakovi vezani vzporedno namesto zaporedno. To je bilo doseženo tako, da je bil vzdolž obeh dolgih stranic pritrjen bakreni vodnik. Nato so bili vsi pozitivni priključki trakov s tankimi žičkami prispajkani na vodnik na eni strani, vsi negativni pa na vodnik na drugi strani. Podatkovni priključki so bili prispajkani zaporedno, kot je prikazano v poglavju 2.1, razen med 16. in 17. ter med 34. in 35. stolpcem, da so bili trakovi ločeni v tri verige, kot je obrazloženo v poglavju

2.2. Kasneje, ko so bili dodani tudi stranski trakovi, so bile podatkovne linije vseh štirih verig trakov povezane na štiri digitalne priključke mikrokrmilnika. Napajalna vodnika sta bila vsak s po dvema žičkama iz nasprotnih strani povezana na debelejši kabel, ta pa na 5V DC napajalnik.

Slika 13: Izdelava - vezava trakov

Vir: Lasten vir

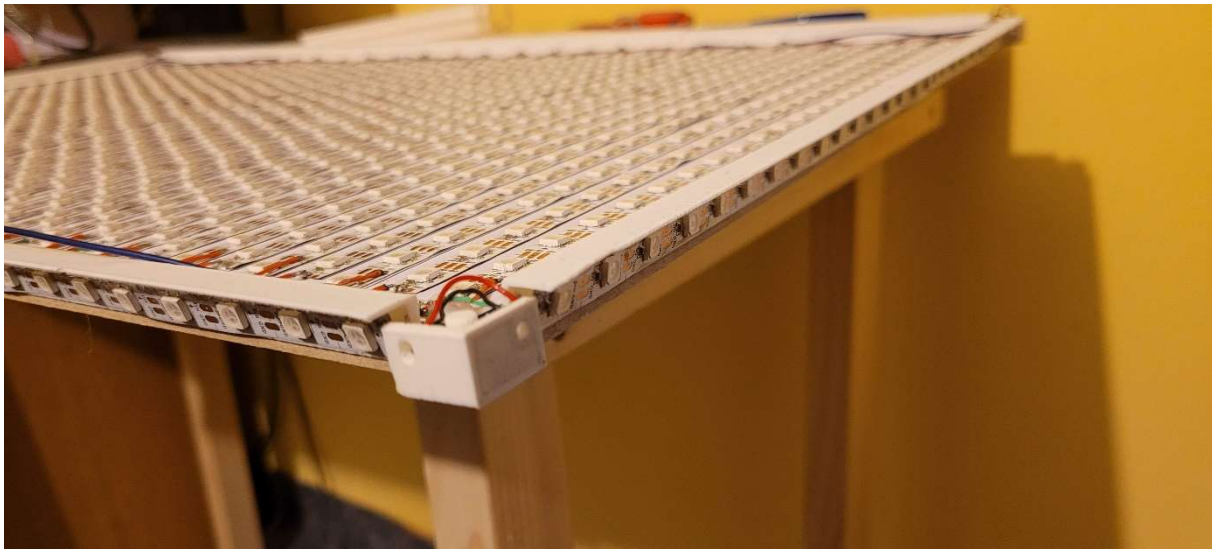


Slika 14: Izdelava - vezava trakov od blizu

Vir: Lasten vir

4.3 Robovi in stranski trakovi

3D natisnjeni plastični robovi so bili pritrjeni na ustrezna mesta na plošči (slika 16), nanje pa so bili prilepljeni še stranski LED trakovi (slika 15), ki so bili vezani zaporedno; napajalni priključki pa so bili na napajanje povezani na obeh koncih, za čim bolj enakomerno porazdelitev toka.



Slika 15: Izdelava - stranski trakovi

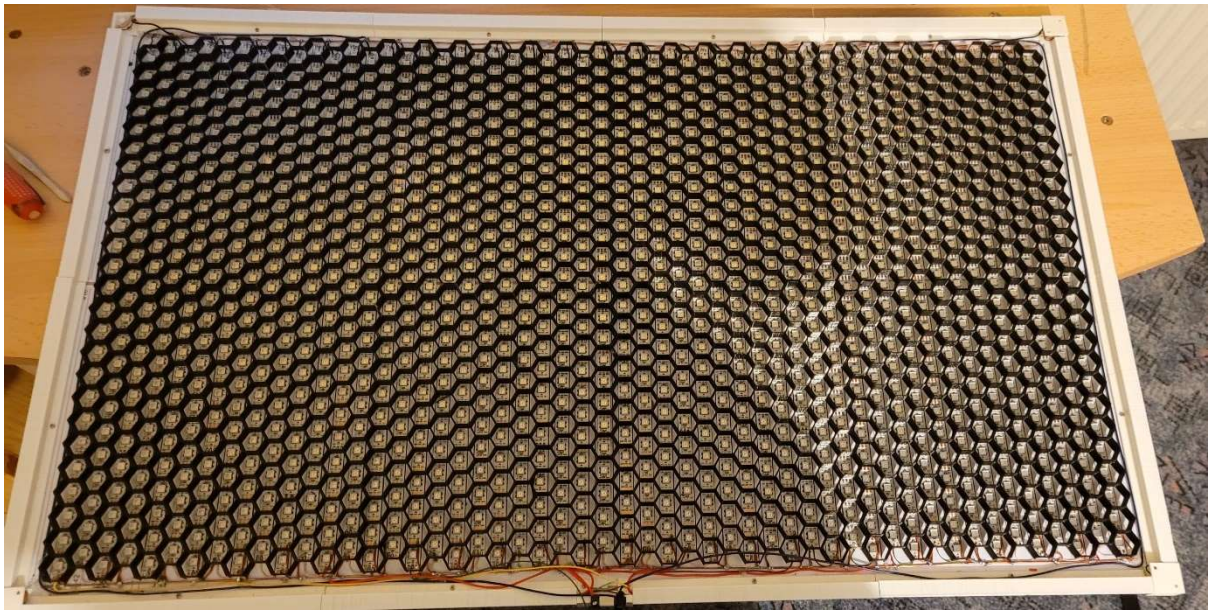
Vir: Lasten vir

Slika 16: Izdelava - pritrjeni robovi in dopolnjena vezava

Vir: Lasten vir

4.4 Satovje

Natisnjeni so bili še posamezni deli satovja in prilepljeni v eno enoto. Celotno satovje je bilo položeno na ploščo, tako da so bile lučke pozicionirane na sredini celic. Lepljenje ali privijanje ni potrebno, saj satovje na mestu drži ohišje (slika 17).

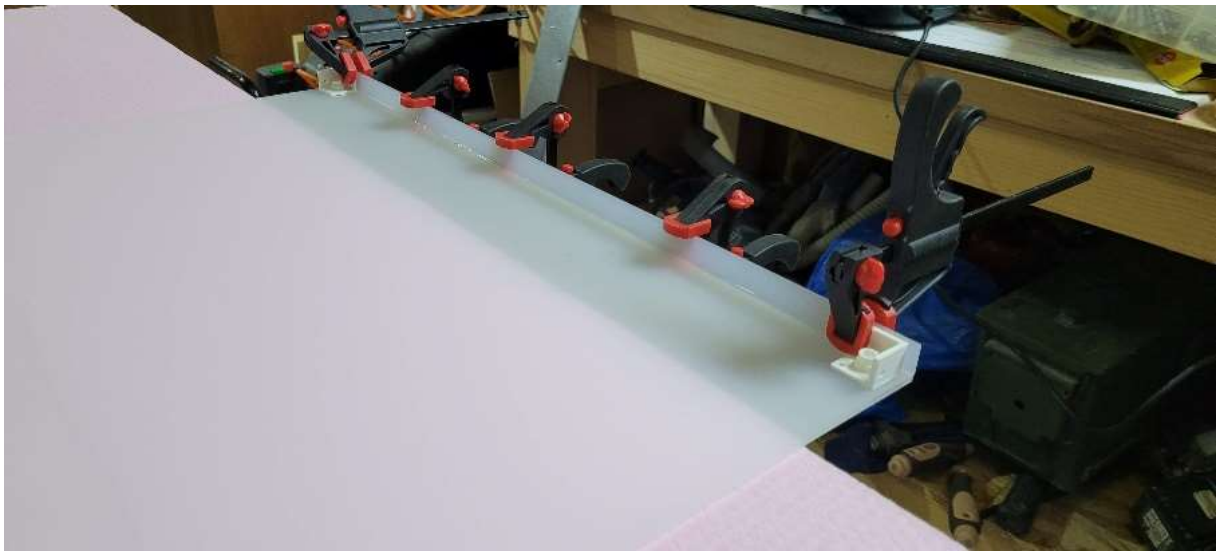


Slika 17: Izdelava - panel s satovjem

Vir: Lasten vir

4.5 Prosojno ohišje

Za konec je bilo iz zamegljenega akrilnega stekla lasersko izrezanih pet delov po dimenzijah omenjenih v poglavju 3.5. V štiri kote največjega dela, so bili 3 mm stran od robov pritrjeni pomožni plastični deli, na le te pa tudi ostali štiri kosi pleksi stekla. Ohišje je bilo nato položeno čez ploščo, kar pa je bilo precej težavno, saj pri izdelavi ni bilo puščene dovolj tolerance. Ohišja ni treba priviti, saj ga na mestu drži pritisk stranskih robov.

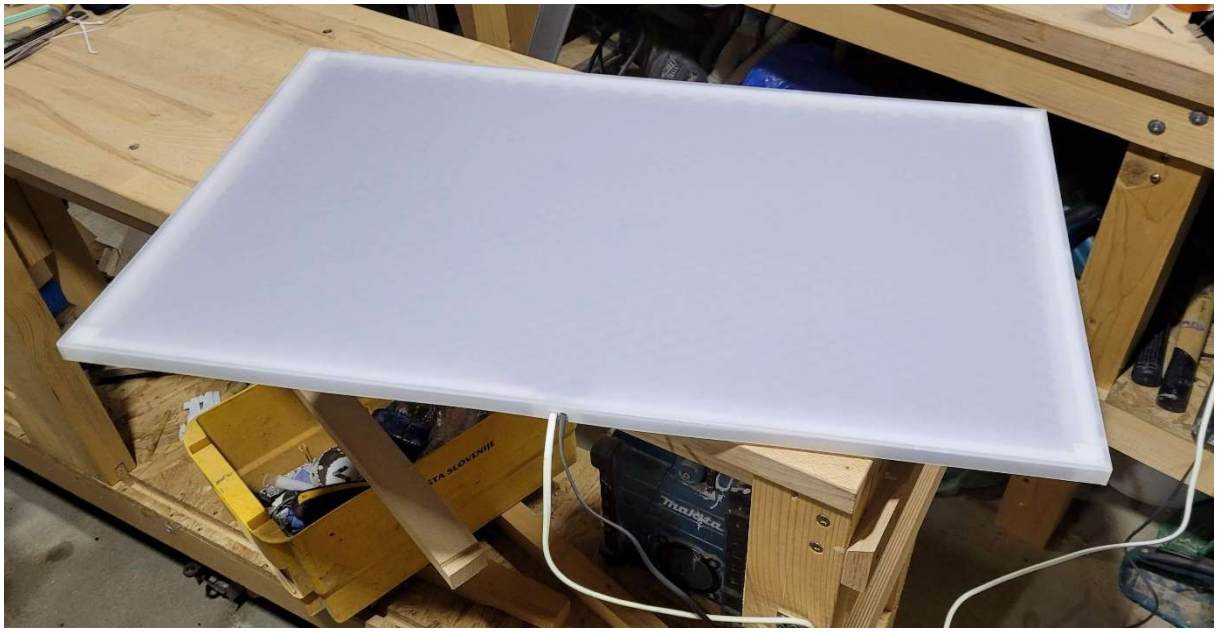


Slika 18: Izdelava - lepljenje stranic ohišja

Vir: Lasten vir

Slika 19: Izdelava - sestavljeno ohišje

Vir: Lasten vir



Slika 20: Končan panel

Vir: Lasten vir

5. PROGRAMIRANJE

5.1 Zasnova programa

Programiranje je potekalo v razvojnem okolju Arduino. Za lažjo berljivost je projekt razdeljen med več datotekami. Glavna datoteka vsebuje le vključitve knjižnic in ostalih datotek, ter standardni funkciji setup() in loop(). V prvi vključitveni datoteki startup.h se nahajajo vse definicije, globalne spremenljivke in objekti potrebni za delovanje knjižnic ter funkcije, ki se izvedejo le enkrat ob zagonu programa. V drugi vključitveni datoteki tasks.h, se nahajajo funkcije, ki se izvajajo več krat med delovanjem programa. V zadnji vključitveni datoteki effects.h so shranjeni razredi za izvajanje svetlobnih efektov.

Podatki za prikazovanje slike na sprednjem delu plošče so hranjeni v 2D polju oziroma matriki, kjer prva dimenzija predstavlja stolpce, druga pa vrstice pikslov na panelu ter element na indeksu [0][0] predstavlja zgornji levi piksel panela. Podatki za prikazovanje na stranskih LED trakovih so hranjeni kot navadno polje, kjer indeksi predstavljajo diode na traku po vrsti z začetkom na polovici spodnje stranice (levo od odprtine za kable) in smerjo poteka v smeri urinega kazalca okoli panela. Vsak element obeh struktur je pravzaprav sam po sebi polje, dolžine tri in tipa uint8_t (osem bitno pozitivno celo število), kot narekuje uporabljena knjižnica za upravljanje LED trakov - FastLED. Elementi le-tega predstavljajo tri komponente željene barve – bodisi v RGB bodisi v HSV barvnem modelu. Obe prej omenjeni strukturi sta globalni spremenljivki, saj ju uporablja večina funkcij v programu.

Navodila za izvajanje efektov so strukturirana kot razredi, s potrebnimi spremenljivkami za delovanje efekta, konstruktorom ki nastavi nekatere spremenljivke in postavi prvo sliko efekta ter funkcijo, ki ob vsakem klicu »iterira« efekt oziroma nastavi naslednjo sliko. Razred efekta mora za pravilno delovanje programa vedno imeti prej omenjeno javno funkcijo iterate(), ki ne vrne ali vzame nobene vrednosti; javno logično spremenljivko HSVmode, ki funkciji za prikaz slike pove, da so podatki, ki jih efekt piše v strukturi za hranjenje podatkov slike v HSV barvnem modelu, če je vrednost spremenljivke logična 1, oziroma da so v RGB modelu, če je vrednost logična 0; ter javno pozitivno celo-številčno spremenljivko iterationDelay, ki pove, koliko časa mora miniti med vsako iteracijo efekta v milisekundah. Vsak efekt ima zaporedno številko, določeno v glavni funkciji loop(), lahko pa ima tudi neke parametre, ki jih navedemo ob nastavljanju efekta in mu določajo različne lastnosti, na primer hitrost, barve, smer itd.

S tem, da so efekti shranjeni kot razredi dosežemo, da so spremenljivke potrebne za delovanje deklarirane le, ko se ta efekt izvaja; z obvezno funkcijo in spremenljivkama pa, da so deklaracija, iteracija ter prikaz efekta vedno izvedene na enak način, kar zelo skrajša kodo v glavnem programu, kot bo vidno v nadaljevanju.

Spodaj je prikazana vsebina datoteke startup.h (slika 21), torej definicije, globalne spremenljivke in začetne funkcije.

```
1   //_____________________________________________________________Definicije in globalne spremenljivke
2   //Priključki:
3   #define LEDsec1Pin 14       //Pin za LEDice na prvem delu matrice
4   #define LEDsec2Pin 27       //Pin za LEDice na drugem delu matrice
5   #define LEDsec3Pin 26       //Pin za LEDice na tretjem delu matrice
6   #define LEDsidePin 25       //Pin za LEDice na strani plošče
7   #define WiFiStatusLEDPin 2  //Pin za lučko, ki se vklopi ko je vzpostavljena povezava z WiFi-jem
8   #define AudioPin 33         //Pin za glasbo
9
10  //LED lučke:
11  #define LEDcols 51          //Število stolpcev
12  #define LEDrows 25          //Število vrstic
13  #define LEDsec1Cols 16      //Število stolpcev v prvem delu matrice
14  #define LEDsec2Cols 18      //Število stolpcev v drugem delu matrice
15  #define LEDsec3Cols 17      //Število stolpcev v tretjem delu matrice
16  #define LEDsideSec1LEDs 21  //Število LEDic na prvi polovici spodnje stranice
17  #define LEDsideSec2LEDs 26  //Število LEDic na levi stranici
18  #define LEDsideSec3LEDs 45  //Število LEDic na zgornji stranici
19  #define LEDsideSec4LEDs 26  //Število LEDic na desni stranici
20  #define LEDsidesec5LEDs 21  //Število LEDic na drugi polovici spodnje stranice
21  #define LEDtotalSideLEDs LEDsideSec1LEDs+LEDsideSec2LEDs+LEDsideSec3LEDs+LEDsideSec4LEDs+LEDsidesec5LEDs  //skupno število stranskih LEDic
22  #define LEDtype WS2812      //Tip LED traka
23  #define LEDvoltage 5        //Napetost
24  #define LEDmaxCurrent 5000  //Najvecje dovoljeno število milliamperov za vsak trak
25
26  //Objekti za LED trakove:
27  CRGB LEDsec1[LEDsec1Cols * LEDrows];  //FastLED array za prvi del LED matrice
28  CRGB LEDsec2[LEDsec2Cols * LEDrows];  //FastLED array za prvi del LED matrice
29  CRGB LEDsec3[LEDsec3Cols * LEDrows];  //FastLED array za prvi del LED matrice
30  CRGB LEDside[LEDtotalSideLEDs];       //FastLED array za stranice
31
32  //WiFi:
33  #define NWssid "[redacted]" //Ime omrežja
34  #define NWpass "[redacted]"   //Geslo omrežja
35  #define WiFiRCT 10000         //Na koliko časa poskuša vzpostaviti povezavo
36  unsigned long WiFiLCA = 0;    //Čas zadnjega poskusa povezave
37  TaskHandle_t HandleWiFiTask;  //Da lahko handleWiFi funkcijo izvedemo na drugem jedru
38  WiFiServer server(80);        //Objekt za WiFi server, da lahko sprejemamo HTTP zahteve
39
40  //Efekti:
41  #define MaxEffectParameters 10                              //Maksimalno število parametrov za efekte
42  bool effectChanged = false;                                 //Če se je trenutni efekt spremenil
43  int currentEffectNum = 0;                                   //Številka renutnega efekta
44  Array<int, MaxEffectParameters> currentEffectParameters = {}; //Dinamični array parametrov za trenutni efekt
45  unsigned long lastEffectIterationTime = 0;                  //Čas zadnje iteracije efekta
46  uint8_t gridData[LEDcols][LEDrows][3] = {0};                //Glavni 2D array za podatke LED matrice
47  uint8_t sideData[LEDtotalSideLEDs][3] = {0};                //Glavni array za podatke stranskih LED lučk
48
49  //FFT:
50  #define FFTsamples 256            //Število vzorcev za Fourierjevo tranformacijo
51  #define FFTsampleFrequency 20000  //Frekvenca vzorčenja
52  #define VisualizerBands 17        //Število prikazanih frekvenčnih skupin
53  double vReal[FFTsamples];         //Array za realne komponente Fourierjeve transformacije
54  double vImag[FFTsamples];         //Array za imaginarne komponente Fourierjeve transformacije
55  arduinoFFT FFT = arduinoFFT(vReal, vImag, FFTsamples, FFTsampleFrequency);  //FFT objekt
56
57  //_____________________________________________________________Začetne funkcije:
58  //LED trakovi:
59  void initLEDstrips()        //Funkcija za inicializacijo LED trakov
60  {
61    FastLED.addLeds<LEDtype, LEDsec1Pin, GRB>(LEDsec1, LEDsec1Cols * LEDrows);  //Doda prvi del LED matrice
62    FastLED.addLeds<LEDtype, LEDsec2Pin, GRB>(LEDsec2, LEDsec2Cols * LEDrows);  //Doda prvi del LED matrice
63    FastLED.addLeds<LEDtype, LEDsec3Pin, GRB>(LEDsec3, LEDsec3Cols * LEDrows);  //Doda prvi del LED matrice
64    FastLED.addLeds<LEDtype, LEDsidePin, GRB>(LEDside, LEDtotalSideLEDs);       //Doda stranice
65    FastLED.setMaxPowerInVoltsAndMilliamps(LEDvoltage, LEDmaxCurrent);          //Nastavi najvecjo dovoljeno porabo
66    FastLED.clear();                                                            //Sprazni LED arraye
67    FastLED.show();                                                             //Izklopi vse LEDice
68  }
```

Slika 21: Definicije, globalne spremenljivke in začetne funkcije

Vir: Lasten vir

5.2 Funkcija za prikaz slike

Ker so podatki slike sprednjega dela plošče shranjeni v matriki, trakovi pa rabijo podatke v obliki navadnega polja, poleg tega pa imajo še izmenično smer toka podatkov in so razdeljeni na tri verige, potrebujemo funkcijo, ki prebere podatke iz matrike in jih pošlje trakom tako, da se prikažejo kot željena slika.

V prvem delu funkcije se definira polje dolžine skupnega števila lučk na sprednji strani, kamor se bodo shranili podatki, kot da če bi imeli eno samo verigo. Nato se izvede zanka, ki gre skoz vse stolpce matrike. Če je številka stolpca soda, se podatki matrike v polje prepišejo od spodaj navzgor, če je liha pa od zgoraj navzdol.

V drugem delu funkcije se podatki zapišejo v FastLED polje ustrezne verige trakov, glede na njihovo definirano dolžino. Zapišejo se tudi podatki za prikaz stranskih lučk. Pomembno je če se podatki zapišejo v RGB ali HSV modelu, zato pred pisanjem preverimo vrednost spremenljivke, ki jo funkcija dobi kot parameter. Nato lahko sliko preprosto prikažemo z klicem funkcije FastLED.show().

```
void updateLEDs(bool HSVmode)                //Funkcija, ki podatke prikaže na panelu.
{
  uint8_t stripData[LEDcols * LEDrows][3];            //1D array za linerane podatke matrice
  int count = 0;                                      //Števec

  //Pretvori 2D podatke v 1D glede na postavitev trakov:
  for(int x = 0; x < LEDcols; x++)                    //Gre skoz vse stolpce
    if(x % 2 == 0)                                      //Če je stolpec sodi
      for(int y = LEDrows - 1; y >= 0; y--, count++)      //Gre po vrsticah od spodaj navzgor
        for(int i = 0; i < 3; i++)                          //Gre skoz vse tri elemente
          stripData[count][i] = gridData[x][y][i];            //Prepiše podatek iz 2D v 1D array
    else                                                //Če je stolpec lihi
      for(int y = 0; y < LEDrows; y++, count++)           //Gre po vrsticah od zgoraj navzdol
        for(int i = 0; i < 3; i++)                          //Gre skoz vse tri elemnte
          stripData[count][i] = gridData[x][y][i];            //Prepiše podatek iz 2D v 1D array

  //Prikaže podatke iz 1D arraya na ustreznem traku:
  count = 0;
  if(HSVmode)                                         //Zapiše podatke na ustrezni trak za HSV model
  {
    for(int i=0; i < LEDsec1Cols * LEDrows; i++, count++) LEDsec1[i]=CHSV(stripData[count][0], stripData[count][1], stripData[count][2]);
    for(int i=0; i < LEDsec2Cols * LEDrows; i++, count++) LEDsec2[i]=CHSV(stripData[count][0], stripData[count][1], stripData[count][2]);
    for(int i=0; i < LEDsec3Cols * LEDrows; i++, count++) LEDsec3[i]=CHSV(stripData[count][0], stripData[count][1], stripData[count][2]);
    for(int i=0; i < LEDtotalSideLEDs; i++) LEDside[i]=CHSV(sideData[i][0], sideData[i][1], sideData[i][2]);
  }
  else if(!HSVmode)                                   //Zapiše podatke na ustrezni trak za RGB model
  {
    for(int i=0; i < LEDsec1Cols * LEDrows; i++, count++) LEDsec1[i]=CRGB(stripData[count][0], stripData[count][1], stripData[count][2]);
    for(int i=0; i < LEDsec2Cols * LEDrows; i++, count++) LEDsec2[i]=CRGB(stripData[count][0], stripData[count][1], stripData[count][2]);
    for(int i=0; i < LEDsec3Cols * LEDrows; i++, count++) LEDsec3[i]=CRGB(stripData[count][0], stripData[count][1], stripData[count][2]);
    for(int i=0; i < LEDtotalSideLEDs; i++) LEDside[i]=CRGB(sideData[i][0], sideData[i][1], sideData[i][2]);
  }
  FastLED.show();                                     //Prikaže podatke
}
```

Slika 22: Funkcija za prikaz slike

Vir: Lasten vir

5.3 Funkcija za zlivanje slike čez rob

Pri nekaterih efektih bi morda želeli, da bi bilo dogajanje na stranskih trakovih nepovezano z dogajanjem na sprednji strani panela (npr. sprednja stran simulira padanje dežja, stranice pa vsake toliko zabliskajo da simulirajo strelo), pri nekaterih pa to nima smisla. Stransko osvetljavo lahko vseeno izkoristimo tako, da stranice matrike preslikamo na stranske trakove in tako ustvarimo efekt, da se slika zlije čez rob in osvetli tudi steno, tako kot pri nekaterih televizorjih.

Problem je, da število lučk na straneh ni enako številu lučk na robovih matrike, sploh zgoraj in spodaj, saj so po širini trakovi na sprednji strani dalje narazen, kot lučke na njih (poglavje 2.3). To bomo rešili tako, da bomo matematično poiskali indeks na robu matrike, ki je najbližje določenem indeksu na stranskem traku z upoštevanjem dolžine obeh polij. Najprej izračunamo na koliko odstotkih dolžine traku je določena lučka z formulo $\frac{i \times 100}{x-1}$, kjer je i indeks lučke, x pa število lučk na traku. Dobljeno vrednost nato pomnožimo z številom lučk na soležnem robu matrike - 1, da dobimo »idealni indeks« na robu matrike pomnožen z 100. To nato delimo z 100 in zaokrožimo, da dobimo indeks lučke, ki je najbližja lučki na stranskem traku.

V funkciji gremo skoz vse lučke na vseh stranicah in kopiramo podatke od najbližje lučke na robu matrike. Paziti je treba pri spodnji stranici, saj je ena polovica stranice na začetku verige druga polovica pa na koncu.

```
void expandEffectToSide()   //Funkcija, ki vrednosti lučk na robovih matrike razširi na stranske trakove:
{
  int leftSideOffset = LEDsideSec1LEDs;                       //Zamik do prve lučke na levi strani
  int topSideOffset = leftSideOffset + LEDsideSec2LEDs;       //Zamik do prve lučke na zgornji strani
  int rightSideOffset = topSideOffset + LEDsideSec3LEDs;      //Zamik do prve lučke na desni strani
  int bottomSideOffset = rightSideOffset + LEDsideSec4LEDs;   //Zamik do prve lučke na spodnji strani
  //Leva stran:
  for(int i = 0; i < LEDsideSec2LEDs; i++)  //Za vse lučke na levi strani
  {
    int matchWith = (LEDrows-1)-round(float(i*100/(LEDsideSec2LEDs-1)*(LEDrows-1))/100);  //Ugotovi katera lučka na matrici je najbližja
    for(int j = 0; j < 3; j++) sideData[i+leftSideOffset][j] = gridData[0][matchWith][j]; //Prekopira podatke
  }
  //Desna stran:
  for(int i = 0; i < LEDsideSec4LEDs; i++)  //Za vse lučke na desni strani
  {
    int matchWith = round(float(i*100/(LEDsideSec4LEDs-1)*(LEDrows-1))/100);  //Ugotovi katera lučka na matrici je najbližja
    for(int j = 0; j < 3; j++) sideData[i+rightSideOffset][j] = gridData[LEDcols-1][matchWith][j];  //Prekopira podatke
  }
  //Zgornja stran:
  for(int i = 0; i < LEDsideSec3LEDs; i++)  //Za vse lučke na zgornji strani
  {
    int matchWith = round(float(i*100/(LEDsideSec3LEDs-1)*(LEDcols-1))/100);  //Ugotovi katera lučka na matrici je najbližja
    for(int j = 0; j < 3; j++) sideData[i+topSideOffset][j] = gridData[matchWith][0][j];  //Prekopira podatke
  }
  //Spodnja stran:
  for(int i = 0; i < LEDsideSec3LEDs; i++)  //Za vse lučke na spodnji strani
  {
    int matchWith = (LEDcols-1)-round(float(i*100/(LEDsideSec3LEDs-1)*(LEDcols-1))/100);  //Ugotovi katera lučka na matrici je najbližja
    int realIndex;        //Spremenljivka v kateri je dejanski indeks lučke v verigi, z upoševanjem da je veriga razpolovljena
    if(i<(LEDsideSec3LEDs-LEDsideSec1LEDs)-1) realIndex = i + bottomSideOffset;   //Če je indeks manj kot dolžina verige
    else realIndex = i - (LEDsideSec3LEDs-LEDsideSec1LEDs);                       //Če je indeks večji kot dolžina veriga
    for(int j = 0; j < 3; j++) if(realIndex<LEDtotalSideLEDs) sideData[realIndex][j] = gridData[matchWith][LEDrows-1][j]; //Prekopira podatke
  }
}
```

Slika 23: Funkcija za zlivanje slike čez rob

Vir: Lasten vir

5.4 Funkcija za upravljanje z WiFi-jem

S tem da mikrokrmilnik povežemo na WiFi omrežje dosežemo, da lahko željeni efekt nastavimo brezžično, na primer preko aplikacije za mobilni telefon; pa tudi da lahko kodo posodobimo brez, da je treba mikrokrmilnik vsakič povezati na računalnik preko kabla. Ker ima ESP32 dva jedra, bomo naredili funkcijo, ki upravlja vse v zvezi z WiFi-jem in se nenehno izvaja ter jo dodelili neuporabljenemu jedru 0, ostalemu programu pa tako pustili, da se nemoteno izvaja na primarnem jedru 1.

Za povezavo na omrežje in dobivanje HTTP zahtev, preko katerih bo mogoče spreminjati efekte, bomo uporabili knjižnico WiFi, za brezžično posodabljanje kode pa knjižnico ArduinoOTA.

Uporabili bomo tudi knjižnico Array, ki omogoča uporabo polja z dinamično dolžino, saj bomo vanj shranjevali željene parametre efekte, ki pa jih ni enako veliko pri vseh efektih.

Najprej v funkciji deklariramo spremenljivko, v katero bomo shranili zadnjo dobljeno HTTP zahtevo, da preprečimo, da se ista zahteva interpretira večkrat, v primeru da je bila večkrat poslana. Vse ostalo damo v večno zanko, saj želimo, da se ves čas izvaja. Na začetku vedno preveri ali je mikrokrmilnik povezan z WiFi omrežjem. Če ni se poskuša povezati, v kolikor je minila definirana količina časa od zadnjega poizkusa povezave. Če povezava uspe, zažene OTA, ki skrbi za brezžično posodabljanje in server, ki skrbi za dobivanje HTTP zahtev. Če je povezava že vzpostavljena, najprej pokliče funkcijo OTA.handle(), ki preverja če želimo naložiti novo kodo in če želimo, za to avtomatsko poskrbi. Nato preveri če je bila prejeta nova HTTP zahteva in če je bila jo pretvori v niz. Če zahteva ni enaka prejšnji, jo preuči in če je ustrezno formatirana (»*ip*/setEffect/*številka_efekta*«ALI»*ip*/setEffect/*številka_efekta*/*parameter1*&*parameter2*&*parameter3*…«) ugotovi številko željenega efekta in njegove parametre, če smo jih vnesli. Podatke shrani v ustrezni spremenljivki in vklopi spremenljivko, ki glavnemu programu pove, da je bil efekt spremenjen. Da funkcijo dodelimo jedru 0, v glavni funkciji setup() izvedemo spodnji ukaz.

```
xTaskCreatePinnedToCore(handleWiFi, "HandleWiFi", 8000, NULL, tskIDLE_PRIORITY, &HandleWiFiTask, 0);
```

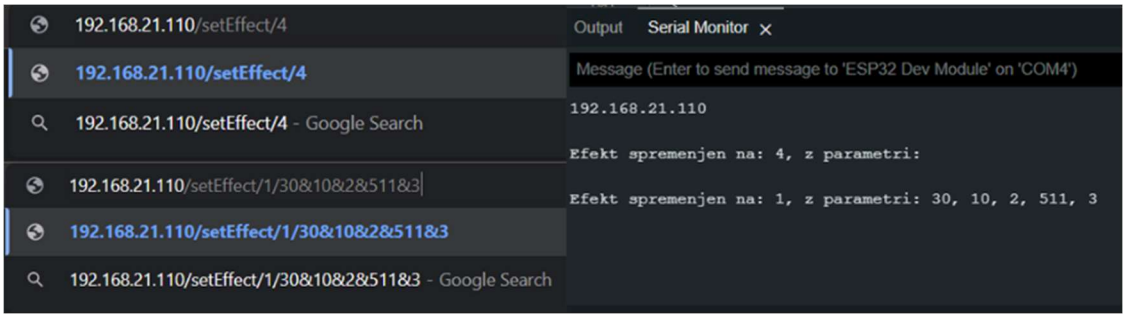
Slika 24: Ukaz za dodelitev funkcije jedru 0, z minimalno prioriteto

Vir: Lasten vir

```
void handleWiFi(void *)       //Funkcija za upravljanje z WiFi-jem
{
  String previousRequest = "";                        //Prejšnja obdelana HTTP zahteva
  while(true)                                         //Večna zanka
  {
    if(WiFi.status() == WL_CONNECTED)                     //Če je WiFi povezan
    {
      digitalWrite(WiFiStatusLEDPin, HIGH);                   //Vklopi WiFi lučko
      ArduinoOTA.handle();                                    //Preveri če želimo program brezžično posodobiti
      WiFiClient client = server.available();                 //Preveri HTTP zahteve
      if(client)                                              //Če smo dobili HTTP zahtevo
      {
        String request = client.readStringUntil('\r');            //Prebere HTTP zahtevo
        client.flush();
        if(!request.equals(previousRequest))                      //Če zahteva ni enaka prejšnji
        {
          if(request.indexOf("/setEffect/") == 4)                     //Če želimo spremeniti efekt
          {
            int thirdSlash = request.indexOf("/", 15);                    //Mesto tretje poševnice v zahtevi
            int secondSpace = request.indexOf(" ", 4);                    //Mesto drugega presledka v zahtevi
            currentEffectNum = request.substring(15, secondSpace).toInt();//Spremeni številko trenutnega efekta
            currentEffectParameters.clear();                              //Sprazni array za parametre efekta
            if(thirdSlash < secondSpace)                                  //Če zahteva vsebuje parametre
            {
              int parCount = 1;                                           //Spremenljivka za število zahtevanih parametrov
              for(int i = thirdSlash + 1; i < secondSpace; i++)           //Za vsaki znak od tretjega '/' do drugega ' '
                if(request.charAt(i) == '&') parCount++;                      //Če je znak '&' poveča število parametrov
              int currentParStart = thirdSlash + 1;                       //Začetno mesto trenutnega parametra
              for(int i = 0; i < parCount; i++)                           //Za vsak parameter
              {
                int currentParVal = request.substring(currentParStart).toInt(); //Prebere vrednot trenutnega parametra
                currentEffectParameters.push_back(currentParVal);               //Doda trenutni parameter v array
                currentParStart = request.indexOf("&", currentParStart + 1) + 1;//Spremeni mesto trenutnega parametra
              }
            }
            effectChanged = true;                                         //Označi da se je efekt spremenil
            previousRequest = request;                                    //Posodobi zadnjo zahtevo
          }
        }
      }
    }
    else                                                  //Če WiFi ni povezan
    {
      digitalWrite(WiFiStatusLEDPin, LOW);                    //Izklopi WiFi lučko
      if(millis() > WiFiLCA + WiFiRCT || millis() < WiFiLCA)  //Če je že minilo dovolj časa, da ponovno poskusi z povezavo
      {
        WiFiLCA = millis();                                       //Posodobi zadnji čas povezave
        WiFi.begin(NWssid, NWpass);                               //Poskuša vzpostaviti povezavo z omrežjem
        if(WiFi.waitForConnectResult() == WL_CONNECTED)           //Če je povezava uspela
        {
          ArduinoOTA.begin();                                         //Začne "OTA"
          server.begin();                                             //Začne server
        }
      }
    }
    delay(100);                                           //Počaka 100 milisekund
  }
}
```

Slika 25: Funkcija za upravljanje z WiFi-jem

Vir: Lasten vir



Slika 26: Prikaz delovanja spremembe efekta preko spletnega brskalnika

Vir: Lasten vir

5.5 Spreminjanje in upravljanje efektov

Zaradi uporabe razredov je spreminjanje in upravljanje trenutnega efekta, ki se prikazuje precej preprosto. V glavi funkciji loop() program z pogojnim stavkom najprej preveri, kateri efekt želimo da se prikaže in ustvari objekt razreda efekta s tem številom. Nato vstopi v zanko, ki se izvaja dokler se efekt ne spremeni. V njej iterira efekt, ga prikaže in počaka da mine dovolj časa za ponovno iteracijo. Ker je postopek pri vseh efektih popolnoma enak, ga lahko zapakiramo v funkcijo, za še krajšo in bolj berljivo loop() funkcijo.

```
void loop()
{
  effectChanged = false;    //Izklopi spremenljivko za spremembo efekta
  switch (currentEffectNum) //Switch za številko trenutnega efekta
  {
    case 1:                        //Če je številka trenutnega efekta 1
    {
      DiagonalRainbowWave effect;         //Ustvari objekt za diagonalni mavrični val
      iterateEffectUntilChanged(effect);  //Pokliče funkcijo za iteriranje efekta do spremembe
      break;
    }
    case 2:                        //Če je številka trenutnega efekta 2
    {
      RadialRainbowWave effect;            //Ustvari objekt za radialni mavrični val
      iterateEffectUntilChanged(effect);   //Pokliče funkcijo za iteriranje efekta do spremembe
      break;
    }
    case 3:                        //Če je številka trenutnega efekta 3
    {
      FadingSpots effect;                 //Ustvari objekt za bledeče točke
      iterateEffectUntilChanged(effect);  //Pokliče funkcijo za iteriranje efekta do spremembe
      break;
    }
    case 4:                        //Če je številka trenutnega efekta 4
    {
      SpectrumVisualizer effect;          //Ustvari objekt za frekvenčni prikazovalnik
      iterateEffectUntilChanged(effect);  //Pokliče funkcijo za iteriranje efekta do spremembe
      break;
    }
    default:                       //Če je številka trenutnega efekta karkoli drugega
    {
      FastLED.clear();                    //Sprazni arraye LED trakov
      FastLED.show();                     //Izklopi LED trakove
      while (!effectChanged) delay(100);  //Čaka dokler se ne vklopi spremenljivka za spremembo efekta
      break;
    }
  }
}
```

Slika 27: Funkcija loop()

Vir: Lasten vir

```
template <class Effect>
void iterateEffectUntilChanged(Effect effect)   //Funkcija za iteracijo trenutnega efekta do spremembe efekta
{
  while(!effectChanged)   //Dokler se efekt ne spremeni
  {
    //Če je minilo dovolj časa za ponovno iteracijo efekta:
    if(millis() > lastEffectIterationTime + effect.iterationDelay || millis() < lastEffectIterationTime)
    {
      lastEffectIterationTime = millis(); //Posodobi čas zadnje iteracije
      effect.iterate();                   //Iterira trenutni efekt
      updateLEDs(effect.HSVmode);         //Posodobi LED trakove
    }
  }
  return;
}
```

Slika 28: Funkcija za iteracijo efekta do spremembe

Vir: Lasten vir

5.6 Efekti

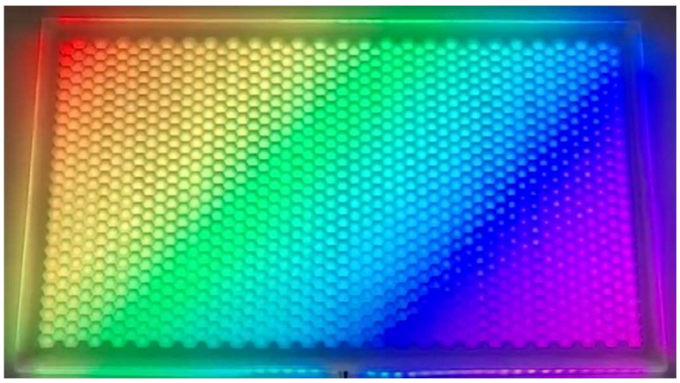
5.6.1 Diagonalni mavrični val

Prvi efekt prikaže mavrico ki se premika diagonalno od enega izmed kotov. Po vrsti ima naslednje parametre: hitrost, število mavric čez matriko, izvorni kot (1-zl, 2-zd, 3-sl, 4-sd) in če je prikazan tudi na straneh. Privzeti parametri so 25 (milisekund med vsako iteracijo), 1 (mavrica), 1 (zgornji levi kot), 1 (se prikaže tudi na straneh panela).

Deluje tako, da ob konstrukciji najprej izračuna, kakšno razliko v barvi mora imeti vsaka naslednja diagonala. Nato gre skozi vse elemente matrike, jim izračuna Manhattansko razdaljo od izvornega kota (vsi elementi v eni diagonali imajo enako Manhattansko razdaljo) in jim nastavi barvno vrednost primerno za to diagonalo. S tem nariše mavrico ki poteka diagonalno od izbranega kota. Ob vsaki iteraciji nato pomanjša barvno vrednost vsakega elementa za enako število, kot je razlika barve med diagonalami, kar ustvari iluzijo, da se mavrica premika stran od izvornega kota.

Spodaj je statična slika (slika 29) efekta ob konstrukciji in QR koda do video posnetka, ki prikaže delovanje efekta z različnimi kombinacijami parametrov. Vljudno vas pozivava k ogledu za boljšo predstavo delovanja.

URL posnetka: https://www.youtube.com/watch?v=Ymdjr39LeBg



Slika 29: Efekt diagonalnega mavričnega vala in QR koda do posnetka

Vir: Lasten vir

```
class DiagonalRainbowWave  //Efekt diagonalnega mavričnega vala
{
  private:
    int hueShift;     //Razlika barve med diagonalami
    bool showSide;    //Če je efekt prikazan tudi na strani

  public:
    unsigned int iterationDelay;  //Čas med vsako iteracijo
    const bool HSVmode = true;    //Podatki v HSV barvnem modelu

    DiagonalRainbowWave()   //Konstruktor
    {
      //Privzete vrednosti:
      iterationDelay = 25;
      float rainbowNum = 1;   //Število mavric čez matriko
      int originCorner = 1;   //Izvorni kot
      showSide = true;

      //Če ima array za parametre efekta validno dolžino nastavi vrednosti po njem:
      if(currentEffectParameters.size() == 4)
      {
        iterationDelay = currentEffectParameters[0];
        rainbowNum = currentEffectParameters[1];
        originCorner = currentEffectParameters[2];
        showSide = bool(currentEffectParameters[3]);
      }

      hueShift = 256 / (LEDcols + LEDrows - 1) * rainbowNum;  //Izračuna spremembo barve med diagonalami
      int originX, originY;                                   //Koordinate izvorne točke:
      if(originCorner == 1) originX = 0, originY = 0;
      else if(originCorner == 2) originX = LEDcols - 1, originY = 0;
      else if(originCorner == 3) originX = 0, originY = LEDrows - 1;
      else originX = LEDcols - 1, originY = LEDrows - 1;

      //Nastavi začetne vrednosti luč:
      for(int x = 0; x < LEDcols; x++)
      {
        for(int y = 0; y < LEDrows; y++)
        {
          int distance = (abs(originX - x) + abs(originY - y)); //Manhattanska razdalja od izvorne točke
          gridData[x][y][0] = distance * hueShift % 256;
          gridData[x][y][1] = 255;
          gridData[x][y][2] = 255;
        }
      }
      for(int i = 0; i < LEDtotalSideLEDs; i++) sideData[i][0] = 0, sideData[i][1] = 0, sideData[i][2] = 0;
    }

    void iterate()  //Funkcija za iteracijo efekta
    {
      //Vsaki lučki zmanjša barvno vrednost za hueShift
      for(int x = 0; x < LEDcols; x++)
        for(int y = 0; y < LEDrows; y++)
          gridData[x][y][0] = (gridData[x][y][0] - hueShift + 255) % 255;

      if(showSide) expandEffectToSide();  //Če želimo efekt prikazati tudi na strani ga razširi na stranske lučke
    }
};
```

Slika 30: Koda efekta diagonalnega mavričnega vala

Vir: Lasten vir

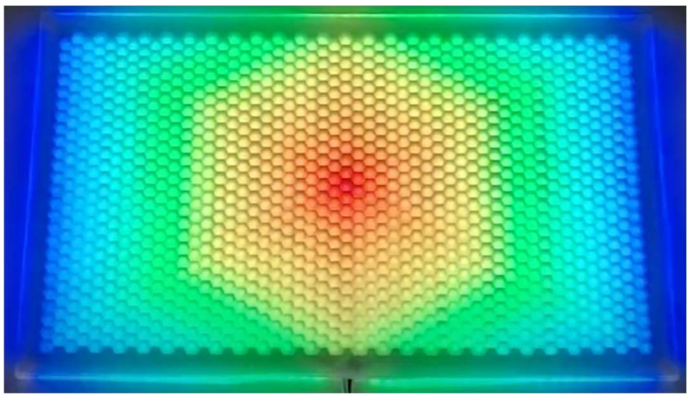
5.6.2 Radialni mavrični val

Drugi efekt je podoben prejšnjemu, le da mavrica potuje iz poljubne točke in se širi v krogu (oziroma šestkotniku), namesto po diagonali. Po vrsti ima naslednje parametre: hitrost, število mavric čez matriko, x koordinata izvorne točke, y koordinata izvorne točke, če potuje iz točke navzven ali od zunaj proti točki in če je prikazan tudi na straneh. Privzeti parametri so 25 (milisekund med vsako iteracijo), 1 (mavrica), 25 (srednji stolpec), 12 (srednja vrstica), 0 (navzven) in 1 (se prikaže tudi na straneh panela).

Deluje tako, da ob konstrukciji najprej izračuna, kakšno razliko v barvi mora imeti vsak naslednji krog. Nato gre skozi vse elemente matrike in jim izračuna Manhattansko razdaljo od izvorne točke. Razdaljo izračuna na aksialni način v šestkotnih mrežah z formulo najdeno na spletni strani [https://www.redblobgames.com/grids/hexagons/#distances], kar nam v bistvu da številko kroga okoli izvorne točke . Nato nastavi barvo elementa na barvo primerno za ta krog. S tem nariše mavrico, ki poteka od izvorne točke navzven. Ob vsaki iteraciji nato pomanjša barvno vrednost vsakega elementa za enako število, kot je razlika barve med krogi, kar ustvari iluzijo, da se mavrica premika navzven od izvorne točke - oziroma ga poveča da mavrica potuje navznoter.

Spodaj je statična slika (slika 31) efekta ob konstrukciji in QR koda do video posnetka, ki prikaže delovanje efekta z različnimi kombinacijami parametrov. Vljudno vas pozivava k ogledu za boljšo predstavo delovanja.

URL posnetka: https://www.youtube.com/watch?v=8Xpwc22vBgM



Slika 31: Efekt radialnega mavričnega vala in QR koda do posnetka

Vir: Lasten vir

```
class RadialRainbowWave     //Efekt radialnega mavričnega vala
{
  private:
    int hueShift;     //Razlika barve med krožnicami
    bool reverse;     //Če gre efekt v nasprotni smeri
    bool showSide;    //Če je efekt prikazan tudi na strani

  public:
    unsigned int iterationDelay;  //Čas med vsako iteracijo
    const bool HSVmode = true;    //Podatki v HSV barvnem modelu

    RadialRainbowWave()   //Konstruktor
    {
      //Privzete vrednosti:
      iterationDelay = 25;
      float rainbowNum = 1;     //Število mavric čez matriko
      int originX = LEDcols/2;  //X koordinata izvorne točke
      int originY = LEDrows/2;  //Y koordinata izvorne točke
      reverse = false;
      showSide = true;

      //Če ima array za parametre efekta validno dolžino nastavi vrednosti po njem:
      if(currentEffectParameters.size() == 6)
      {
        iterationDelay = currentEffectParameters[0];
        rainbowNum = float(currentEffectParameters[1]) / 10;
        originX = currentEffectParameters[2];
        originY = currentEffectParameters[3];
        reverse = bool(currentEffectParameters[4]);
        showSide = bool(currentEffectParameters[5]);
      }

      hueShift = 256 / (LEDcols + LEDrows - 1) * rainbowNum * 2;  //Izračuna spremembo barve med krožnicami
      int originAxialY = originY - (originX + (originX & 1)) / 2; //Izračuna Y koordinato izvorne točke v aksialnem načinu

      //Nastavi začetne vrednosti lučk:
      for(int x = 0; x < LEDcols; x++)
      {
        for(int y = 0; y < LEDrows; y++)
        {
          int axialY = y - (x + (x & 1)) / 2; //Izračuna Y koordinato trenutne točke v aksialnem načinu
          int distance = (abs(originX-x)+abs(originX+originAxialY-x-axialY)+abs(originAxialY-axialY))/2;//Axialna Manhattanska razd
          gridData[x][y][0] = distance * hueShift % 256;
          gridData[x][y][1] = 255;
          gridData[x][y][2] = 255;
        }
      }
      for(int i = 0; i < LEDtotalSideLEDs; i++) sideData[i][0] = 0, sideData[i][1] = 0, sideData[i][2] = 0;
    }

    void iterate()    //Funkcija za iteracijo efekta
    {
      //Smer valovanja:
      int direction = -1;
      if(reverse) direction = 1;

      //Vsaki lučki zmanjša/poveča barvno vrednost za hueShift
      for(int x = 0; x < LEDcols; x++)
        for(int y = 0; y < LEDrows; y++)
          gridData[x][y][0] = (gridData[x][y][0] + (hueShift * direction) + 255) % 255;

      if(showSide) expandEffectToSide();  //Če želimo efekt prikazati tudi na strani ga razširi na stranske lučke
    }
};
```

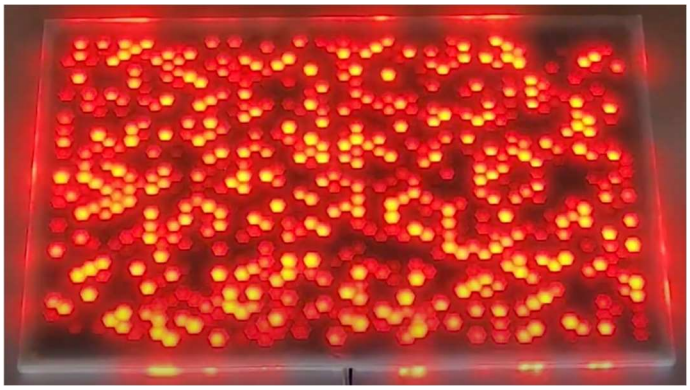
Slika 32: Koda efekta radialnega mavričnega vala

Vir: Lasten vir

5.6.3 Bledeče točke

Tretji efekt na vsake toliko časa z polno močjo prižge novo naključno lučko na panelu, nato pa vse prižgane lučke malo zatemni in to ves čas ponavlja. Po vrsti ima naslednje parametre: koliko časa vsaka lučka ostane vklopljena, število istočasno prižganih lučk, če se barva točk spreminja, barva točk (če je spreminjanje barv izklopljeno) ALI čas da preklopi med vsemi barvami (če je spreminjanje barv vklopljeno) in če je efekt prikazan tudi na straneh. Privzeti parametri so 1000 (milisekund da se lučka izklopi), 600 (lučk naenkrat), 1 (mavrični način), 10000 (milisekund, da gre skoz celotno mavrico) in 1 (efekt se prikaže tudi na straneh panela).
Deluje tako, da ob konstrukciji na podlagi lučk, ki morajo biti hkrati vklopljene in kako dolgo mora biti vsaka lučka vklopljena izračuna na koliko iteracij mora vklopiti novo lučko oziroma koliko lučk mora vklopiti na vsako iteracijo. Nato poskrbi, da je svetlost vseh lučk nastavljena na 0 (jih izklopi) in če uporabnik želi, da se barva ne spreminja, jih nastavi na željeno barvo. Ob vsaki iteraciji nato preveri če mora vklopiti novo lučko / koliko jih mora vklopiti, nato pa vse lučke ki so vklopljene, zatemni za dano vrednost. Če je vklopljen mavrični način, spremeni barvo vsem lučkam.
Spodaj je statična slika (slika 33) efekta med delovanjem in QR koda do video posnetka, ki prikaže delovanje efekta z različnimi kombinacijami parametrov. Vljudno vas pozivava k ogledu za boljšo predstavo delovanja.

URL posnetka: https://www.youtube.com/watch?v=aL_ew3iGdbQ



Slika 33: Efekt bledečih točk in QR koda do posnetka

Vir: Lasten vir

```
class FadingSpots   //Efekt bledečih točk
{
  private:
    int onTime;              //Koliko časa lučka ostane vklopljena
    int spotsNum;            //Število iztočasno prižganih točk
    int hueShift;            //Sprememba barve med iteracijami
    int cyclesPerNewSpot;    //Število iteracij med novo točko / število novih točk na iteracijo
    int trackCycles = 0;     //Spremenljivka za beleženje iteracij
    bool rainbow;            //Če točje spreminjajo barvo
    bool showSide;           //Če je efekt prikazan tudi na strani
    const int dimBy = 5;     //Za koliko se lučke zbledijo med vsako iteracijo

    void generateNewSpot()  //Funkcija, ki ustvari novo točko
    {
      while(true)             //Večna zanka
      {
        int x = random(LEDcols), y = random(LEDrows); //Ustvari naključni koordinati
        if(gridData[x][y][2] == 0)                    //Če je lučka na teh koordinatah izklopljena
        {
          gridData[x][y][2] = 255;                      //Vklopi lučko z polno močjo;
          break;                                        //Prekine zanko
        }
      }
    }

  public:
    unsigned int iterationDelay;  //Čas med vsako iteracijo
    const bool HSVmode = true;    //Podatki v HSV barvnem modelu

    FadingSpots()   //Konstruktor
    {
      //Privzete vrednosti:
      onTime = 1000;
      spotsNum = 600;
      rainbow = true;
      int hueSetting = 10000;  //Barva, če je spreminjanje barv izklopljeno / čas da gre skoz vse barve, če je
      showSide = true;

      //Če ima array za parametre efekta validno dolžino nastavi vrednosti po njem:
      if(currentEffectParameters.size() == 5)
      {
        onTime = currentEffectParameters[0];
        spotsNum = currentEffectParameters[1];
        rainbow = bool(currentEffectParameters[2]);
        hueSetting = currentEffectParameters[3];
        showSide = bool(currentEffectParameters[4]);
      }

      int onCycles = (240 / dimBy);       //Število iteracij, da se lučka izklopi
      iterationDelay = onTime / onCycles; //Izračuna čas med iteracijami
      if(spotsNum < onCycles)             //Če je število točk manj kot število iteracij za izklop lučke
        cyclesPerNewSpot = onCycles / spotsNum; //Izračuna število iteracij na novo točko
      else                                //Če je število točk več kot število iteracij za izklop lučke
        cyclesPerNewSpot = spotsNum / onCycles; //Izračuna število novih točk na iteracijo

      //Nastavi začetne vrednosti luč:
      if(rainbow)
      {
        hueShift = 255 / (hueSetting / iterationDelay);
        if(hueShift == 0) hueShift = 1;
        for(int x = 0; x < LEDcols; x++)
          for(int y = 0; y < LEDrows; y++)
            gridData[x][y][0] = 0, gridData[x][y][1] = 255, gridData[x][y][2] = 0;
      }
      else
      {
        for(int x = 0; x < LEDcols; x++)
          for(int y = 0; y < LEDrows; y++)
            gridData[x][y][0] = hueSetting % 256, gridData[x][y][1] = 255, gridData[x][y][2] = 0;
      }
      for(int i = 0; i < LEDtotalSideLEDs; i++) sideData[i][0] = 0, sideData[i][1] = 0, sideData[i][2] = 0;
    }

    void iterate()    //Funkcija za iteracijo efekta
    {
      if(spotsNum < (240 / dimBy))  //Če je število točk manj kot število iteracij za izklop lučke
      {
        if(trackCycles == 0) generateNewSpot(); //Ustvari novo točko, če je minilo dovolj iteracij
        trackCycles = (trackCycles + 1) % cyclesPerNewSpot;
      }
      else for(int i = 0; i < cyclesPerNewSpot; i++) generateNewSpot(); //Drugače ustvari določeno število novih točk

      //Gre skoz vse lučke
      for(int x = 0; x < LEDcols; x++)
      {
        for(int y = 0; y < LEDrows; y++)
        {
          if(gridData[x][y][2] > 0)   //Če je lučka vklopljena
          {
            if(gridData[x][y][2] <= 15) gridData[x][y][2] = 0;  //Če ima svetlost 15 ali manj jo popolnoma izklopi
            else gridData[x][y][2] -= dimBy;                    //Drugače jo zatemni za določeno vrednost
          }
          if(rainbow) gridData[x][y][0] = (gridData[x][y][0] + hueShift) % 256; //Če je vklopljeno spreminjanje barv poveča barvo
        }
      }

      if(showSide) expandEffectToSide();  //Če želimo efekt prikazati tudi na strani ga razširi na stranske lučke
    }
};
```

Slika 34: Koda efekta bledečih točk

Vir: Lasten vir

5.6.4 Prikazovalnik frekvenčnega spektra

Četrti in zadnji implementirani efekt prikazuje frekvenčni spekter glasbe, ki se predvaja. Trenutno mikrokrmilnik zvočni signal pridobi preko kabla, v prihodnosti pa bi lahko sistem deloval preko BlueTooth-a ali WiFi-ja.

Deluje tako, da ob vsaki iteraciji z frekvenco 20kHz prebere analogno vrednost priključka z zvočnim kablom 256-krat, nato pa na podatkih izvede hitro Fourierjevo transformacijo. To nam vrne relativne intenzivnosti različnih frekvenc, ki jih nato razporedimo v 17 pasov. Te prikažemo na panelu, kot stolpce z različnimi višinami.

Spodaj je statična slika (slika 35) efekta med delovanjem in QR koda do video posnetka, ki prikaže delovanje efekta z različnimi pesmimi. Vljudno vas pozivava k ogledu za boljšo predstavo delovanja.

URL posnetka: https://www.youtube.com/watch?v=BidUexYdS3k

Slika 35: Efekt frekvenčnega prikazovalnika in QR koda do posnetka

Vir: Lasten vir

Koda za izvajanje Fourierjeve transformacije in prikazovanje frekvenčnega spektra na LED matrici je bila navdihnjena po GitHub repozitoriju uporabnika s-marley:
[https://github.com/s-marley/ESP32_FFT_VU]

```
class SpectrumVisualizer    //Efekt frekvenčnega prikazovalnika
{
  private:
    unsigned int amplitude;                    //Amplituda zvočnega signala
    unsigned int noise;                        //Količina šuma v zvočnem signalu
    unsigned int FFTsamplePeriod;              //Čas med FFT vzorčenjem
    unsigned long lastSampleTime = 0;          //Čas zadnjega vzorčenja
    int freqBand[VisualizerBands] = {0};       //Vrednosti frekvenčnih pasov
    int oldBarHeight[VisualizerBands] = {0};   //Višine stolpcev v prejšnji iteraciji

    void performFFT()     //Funkcija, ki izvede Fourierjevo transformacijo
    {
      for(int i = 0; i < FFTsamples; i++)    //Za število vzorcev
      {
        while(micros() < lastSampleTime + FFTsamplePeriod && micros() > lastSampleTime){} //Počaka do časa naslednjega vzorčenja
        lastSampleTime = micros();                 //Posodobi čas zadnjega vzorčenja
        vReal[i] = analogRead(AudioPin);           //Realno komponento analogno vrednost zvočnega priključka
        vImag[i] = 0;                              //Imaginarno komponento nastavi na 0
      }
      //Izvede Fourierjevo transformacijo:
      FFT.DCRemoval();
      FFT.Windowing(FFT_WIN_TYP_HAMMING, FFT_FORWARD);
      FFT.Compute(FFT_FORWARD);
      FFT.ComplexToMagnitude();
    }

    void updateFreqBands()    //Funkcija, ki posodobi vrednosti frekvenčnih pasov
    {
      performFFT();              //Izvede Fourierjevo transformacijo
      for(int i = 0; i < VisualizerBands; i++) freqBand[i] = 0; //Nastavi vrednosti vseh pasov na 0
      //Doda realne komponente ustreznim frekvenčnim pasovom:
      if(VisualizerBands == 17) //Če je 17 pasov:
      {
        for (int i = 1; i < (FFTsamples/2); i++)
        {
          if (vReal[i] > noise)
          {
            int band;
            if (i == 1) band = 0;
            else if (i == 2) band = 1;
            else if (i == 3) band = 2;
            else if (i == 4) band = 3;
            else if (i == 5) band = 4;
            else if (i > 5 && i <= 7) band = 5;
            else if (i > 7 && i <= 9) band = 6;
            else if (i > 9 && i <= 12) band = 7;
            else if (i > 12 && i <= 17) band = 8;
            else if (i > 17 && i <= 25) band = 9;
            else if (i > 25 && i <= 30) band = 10;
            else if (i > 30 && i <= 40) band = 11;
            else if (i > 40 && i <= 55) band = 12;
            else if (i > 55 && i <= 80) band = 13;
            else if (i > 80 && i <= 100) band = 14;
            else if (i > 100 && i <= 125) band = 15;
            else if (i > 125) band = 16;
            freqBand[band] += (int)vReal[i];
          }
        }
      }
    }

  public:
    unsigned int iterationDelay;  //Čas med vsako iteracijo
    const bool HSVmode = true;    //Podatki v HSV barvnem modelu

    SpectrumVisualizer()    //Konstruktor
    {
      //Privzete vrednosti:
      iterationDelay = 5;
      amplitude = 1000;
      noise = 2500;

      //Če ima array za parametre efekta validno dolžino nastavi vrednosti po njem:
      if(currentEffectParameters.size() == 3)
      {
        iterationDelay = currentEffectParameters[0];
        amplitude = currentEffectParameters[1];
        noise = currentEffectParameters[2];
      }

      FFTsamplePeriod = round(1000000 * (1.0 / FFTsampleFrequency));  //Izračuna čas med FFT vzorčenjem

      //Nastavi začetne vrednosti luč:
      for(int x = 0; x < LEDcols; x++)
      {
        int hue = (255/VisualizerBands)*(x/(LEDcols/VisualizerBands));  //Izračuna barvo stolpca
        for(int y = 0; y < LEDrows; y++)
        {
          gridData[x][y][0] = hue, gridData[x][y][1] = 255; gridData[x][y][2] = 0;
        }
      }
      for(int i = 0; i < LEDtotalSideLEDs; i++) sideData[i][0] = 0, sideData[i][1] = 0, sideData[i][2] = 0;
    }


    void iterate()    //Funkcija za iteracijo efekta
    {
      updateFreqBands();                       //Posodobi vrednosti frekvenčnih pasov
      int barHeight[VisualizerBands] = {0};    //Višine stolpcev
      for(int i = 0; i < VisualizerBands; i++)  //Za število frekvenčnih pasov
      {
        //Izračuna višino stolpca:
        barHeight[i] = freqBand[i] / amplitude;
        if(barHeight[i] > LEDrows-1) barHeight[i] = LEDrows-1;
        barHeight[i] = ((oldBarHeight[i] * 1) + barHeight[i]) / 2;
        oldBarHeight[i] = barHeight[i];
      }

      //Nariše stolpce:
      for(int x = 0; x < LEDcols; x++)
      {
        for(int y = 0; y < LEDrows; y++)
        {
          int invertedY = LEDrows - 1 - y;
          if(barHeight[x/(LEDcols/VisualizerBands)] >= y)
          {
            gridData[x][invertedY][2] = 255;
            if(barHeight[x/(LEDcols/VisualizerBands)] == y) gridData[x][invertedY][1] = 0;
            else gridData[x][invertedY][1] = 255;
          }
          else gridData[x][invertedY][2] = 0;
        }
      }
    }
};
```

Slika 36: Koda efekta prikazovalnika frekvenčnega spektra

Vir: Lasten vir

6. PRIMERJAVA S PODOBNIMI IZDELKI

Tržišče stenske LED dekoracije še ni zelo razvito. Praktično edini najinemu podoben produkt, ki sva ga našla je Nanoleaf in njegovi ponaredki. Za razliko od najinega izdelka, Nanoleaf uporablja več manjših ploščic, po navadi trikotne oblike, ki jih uporabnik sestavi v željeno obliko, kar pa ustvari popolnoma drugačne učinke. Razlikuje se tudi po tem, da porabi veliko več prostora in je od najinega izdelka bistveno dražji, saj odvisno od števila uporabljenih ploščic stane od nekaj sto do nekaj tisoč eurov. Najin produkt je tudi edini, ki za prikaz glasbeno reaktivnih efektov uporablja direktno povezavo preko kabla namesto mikrofona, kar zelo poboljša natančnost in odzivnost prikaza takih efektov.

Tabela 1: Primerjava spodobnimi izdelki

Primerjave	Najini Izdelek	Nanoleaf
Prilagodljivi svetlobni učinki	Da	Da
Brezžično upravljanje	Da	Da
Zasede veliko prostora	Ne	Da
Reagira na glasbo	Da	Omejeni učinki in samo preko mikrofona
Cenovno dostopen	Da	Ne

Vir: Lasten vir

7. DRUŽBENA ODGOVORNOST

Pri izdelavi izdelka sva skušala čim bolj zmanjšati vpliv na okolje. Osnova izdelka je lesena plošča, notranji deli so 3D natisnjeni iz plastike PLA, ki je narejena iz naravnih in obnovljivih materialov ter biorazgradljiva, ohišje pa iz akrilnega stekla, ki se ga da reciklirati. Za osvetljavo so uporabljene LED lučke, ki imajo daljšo življenjsko dobo in manjšo porabo energije. Meniva, da bi morala vsa podjetja spoštovati okolje in naravo ter narediti njihove produkte kar se da okolju prijazne.

8. ZAKLJUČEK IN SKLEPI

Meniva da je bila izdelava izdelka uspešna, saj nama je uspelo izpolniti zadane osnovne cilje projekta:

- izdelek je za svoj namen primerno velik – 796×481 mm,
- je kljub velikosti zelo tanek in relativno lahek – 1.7 mm in približno 2 kg,
- je zaradi tankosti in bele barve na beli steni vizualno nemoteč,
- ga je mogoče upravljati brezžično,
- zna prikazati tudi zvočno reaktivne efekte in
- je proti podobnim LED dekorativnim izdelkom bolj poceni – skupna cena delov 162 €, kar bi bilo z množično izdelavo mogoče še dodatno zmanjšati.

Ima pa izdelek definitivno tudi nekaj možnih izboljšav in pomanjkljivosti, ki bi jih bilo potrebno odpraviti preden bi bil pripravljen za na tržišče:

- Izdelati bi bilo potrebno še aplikacijo za upravljanje,
- povezavo na WiFi bi bilo potrebno implementirati na način, da bi se izdelek lahko povezal na katerokoli omrežje, ki bi ga uporabnik izbral preko aplikacije,
- povezava z izdelkom za pošiljanje zvoka bi lahko potekala preko WiFi-ja ali BT-ja, namesto preko kabla,
- mikrokrmilnik bi se lahko nahajal znotraj ohišja izdelka, da bi ga bilo treba povezati le z napajalnim kablom, ki bi se lahko priklopil v stilu polnilca za prenosni računalnik in
- potrebno bi bilo najti boljši način za pritrditev izdelka na steno, saj je trenutno potrebno odstraniti ohišje in ga skozi leseno ploščo priviti na steno, kar pa za uporabnika ni idealno.

Kljub pomanjkljivostim meniva, da je inovacijski predlog zanimiv in nekaj, kar bi z veseljem uporabila za dekoracijo v lastnih domovih.

9. VIRI IN LITERTURA

1) Formula za računanje razdalj v šestkotnih mrežah. 2023. (15. 2. 2023). Dostopno na: https://www.redblobgames.com/grids/hexagons/#distances
2) Inspiracija za programiranje frekvenčnega prikazovalnika. 2023 (15. 2. 2023). Dostopno na: https://github.com/s-marley/ESP32_FFT_VU
3) Podatkovni list za WS2812B. 2007. (15. 2. 2023). Dostopno na: http://www.world-semi.com/Certifications/WS2812B.html
4) Videoposnetki efektov:
   Inovacijski predlog - LED panel: Diagonalni val:
   https://www.youtube.com/watch?v=Ymdjr39LeBg
   Inovacijski predlog - LED panel: Radialni val:
   https://www.youtube.com/watch?v=8Xpwc22vBgM
   Inovacijski predlog - LED panel: Bledeče točke:
   https://www.youtube.com/watch?v=aL_ew3iGdbQ
   Inovacijski predlog - LED panel: Frekvenčni prikazovalnik:
   https://www.youtube.com/watch?v=BidUexYdS3k

PRILOGA

3D model:

https://a360.co/3wvKsKd

Celoten program:

https://drive.google.com/file/d/1x7tFA_gWzxqK6ArOKzpC7s5nMSVX7gX4/view?usp=sharing