LED SCULPTURE

Design Report

Team members:

Potturu Sai Venkat Kushal

Ashutosh

Karthik Balaji (EP18B033)

Gnanesh Naik B (EE18B133)

Nuggu Phani Harsha (ED18B047)

Poloju Kruthik Sai (ED18B020)

TABLE OF CONTENTS

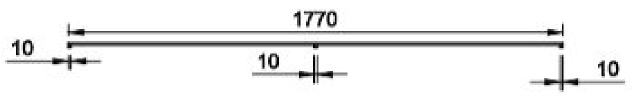
1. ABSTRACT:

LED Sculpture is one of a kind Techno-Entertainment project which is used to show 3D patterns, spectrum analyzer pattern and more. There are a total of 12 x 12 x 8 = 1152 green LEDs (10mm diffused) which are arranged in a cuboid, supported by steel frames. The anodes of the LEDs are connected along the vertical columns and the cathodes of the LEDs are connected along with the horizontal 12 x 8 layers. The whole LED network is controlled by a PCB that holds 14 shift registers, which control the LEDs by taking input from an Arduino Mega board. The Arduino reads an SD card that holds the data for showing animations and sends it to the shift registers.

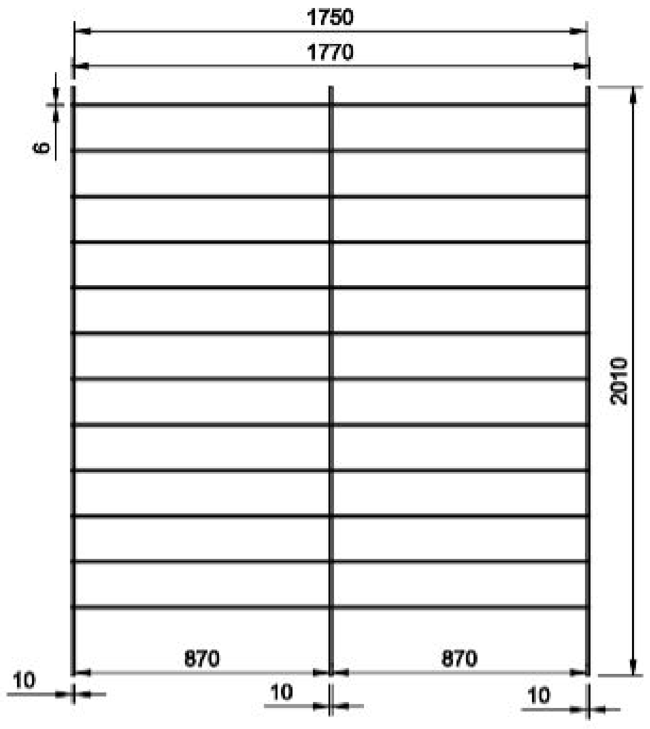
2. MECHANICAL MODULE:

The mechanical structure of the led display is cuboidal in shape and has modular functionality accommodating 1152 LEDs. The structure is made up of mild steel rods of two different kinds, 100mm^2 and 36mm^2 of square cross-sectional area, and 1-inch x1-inch L-angle rods of 3mm material thickness. The structure is made modular so as to aid in the transportation of structure from one place to another. There are two different sets of components in this modular design of the structure, one being vertical layers which are 8 in number and the horizontal layers which are 2 in number which holds the above vertical layers together.

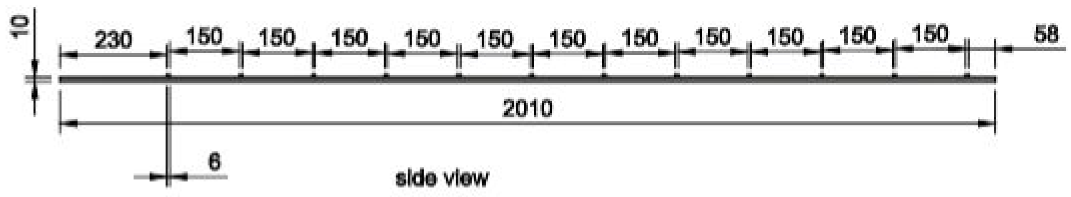
2.1 Vertical layers:



Top view



Front view

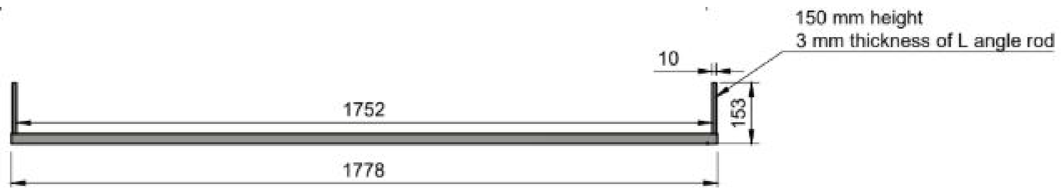


Side view

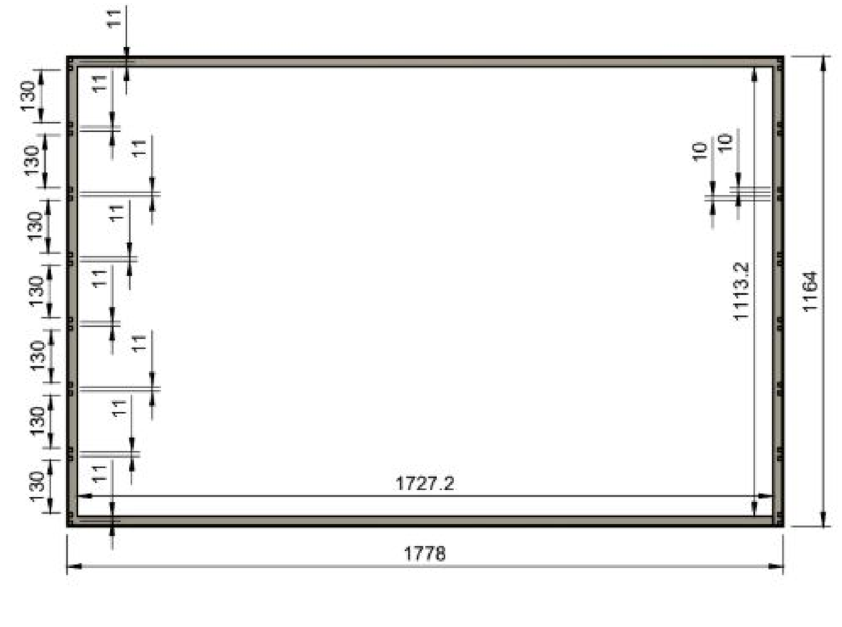
The eight vertical layers mentioned above are utilized to fix the array LEDs in the desired spatial orientation. Each vertical layer is created using 12 horizontal rods of $36mm^2$ square cross-sectional area of length 177 cm and 3 vertical rods of $100mm^2$ square cross-sectional area. Horizontal rods are welded to vertical rods uniformly with 15 cm distance between them leaving 5 cm at the top and 23 cm at the bottom. The spacing between the horizontal rods is taken as 15 cm to ensure that each LED can be differentiated from another LED as an individual pixel when an image is shown on the structure by taking dispersion effect into consideration. The length of the horizontal rod is also calculated with similar criteria and the thickness of vertical rods is also taken into account to have a good amount of surface for welding. The extra length of rods with

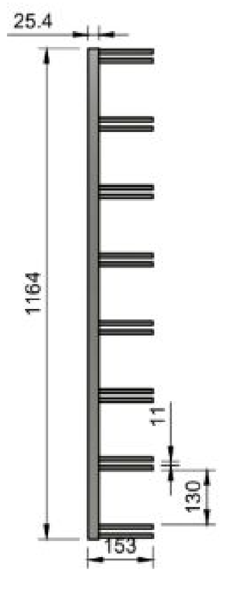
100mm^2 square cross-sectional area in every frame at the top and bottom are used to fit the layers to a base unit and a top unit to hold the layers together and withstand the structure from vibrations and bending. The structure is designed such that the structure is assembled after completing electrical connections on the vertical layers and then layers are mounted into top and base units to make the final structure.

2.2 Top and bottom units:



Front view





Top view

Side view

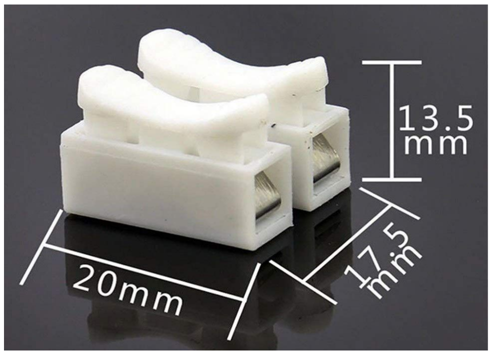
The base rectangular frame is made using 1-inch x 1-inch L-angle rods of 3mm material thickness. L-angle rods are chosen since all the joints welded and L-angle rods provide enough area for welding to make joints stronger. A pair of opposite sides are designed to hold the vertical layers.

For bottom unit, a pair of 10 mm side square cross-section rods of length 15 cm are arranged with a gap of 11 mm between them to accommodate the extra length of $100mm^2$ square cross-sectional rods, left at the bottom of the vertical layers. 8 such pairs are welded to 1-inch x1-inch L-angle rods parallelly on both sides of the rectangle frame of length 1.164m. 13 cm distance is left between adjacent pairs of $100mm^2$ square cross-sectional rods to ensure a distance of 15 cm between two vertical layers by taking a thickness of $100mm^2$ square cross-section rods into account. 8 wheels are joined to the rectangle frame at four vertices and midpoints of four sides to give mobility to structure.

For the top unit, instead of 15cm $100mm^2$ square cross-sectional rods, 5cm $100mm^2$ square cross-sectional rods were used and the rest of the dimensions are maintained the same as a bottom unit.

2.3 Wiring and LED’s

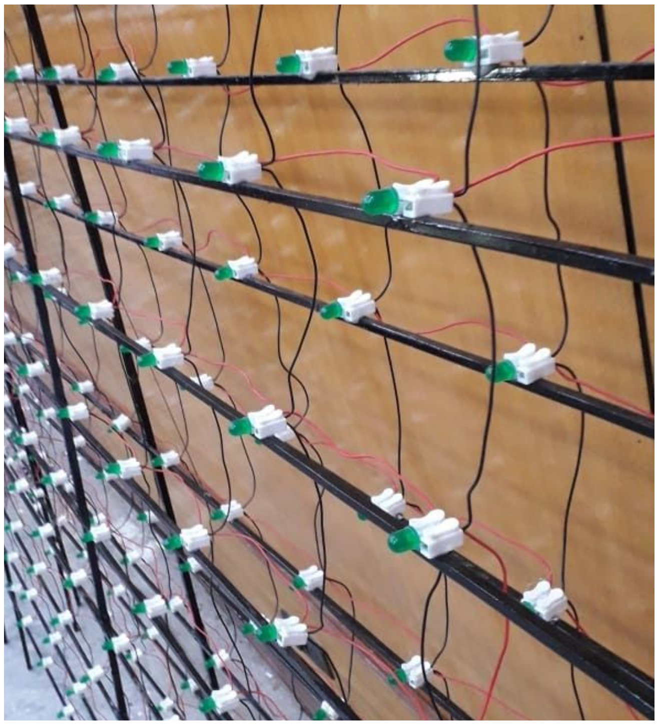
Push-Pin connectors are used as the connection between the wires and LEDs, in the first step LED’s were placed in push-pin connectors, cathodes of LED on the right side of push-pin connector and anode in left. Push-pin connectors with LED’s are stuck to vertical layers of structure using standard Araldite, the gap in between these push-pin connectors with LED’s is maintained to be 14.5cm, in this fashion each horizontal rod in the vertical layer is stuck with 12 push-pin connectors with LED’s and entire vertical layer contains a total of 144 push-pin connectors with LED’s.



Push-pin connector

In the second step, all cathodes or right sides of push-pin connectors that are on the same horizontal rods are connected, in the same fashion, all anodes on the left side of the push-pin connectors in vertical layers are connected, which are in the same vertical line. These layers are placed in the bottom unit as a part of the physical structure, all the cathodes which make a horizontal layer are shorted using single-strand wire by connecting cathodes at ends of different vertical layers.

The top unit is mounted on these layers, from the bottom horizontal layer, all left side of the push-pin connectors are connected with extra length single-strand wires, which in turn are connected to PCB. In the same way, extra length single-strand wire is connected to one of the cathodes of shorted horizontal layers, another end of these single-strand wires are connected to PCB.



Pic of connections in the structure

3. ELECTRICAL MODULE:

The electrical module deals with the components that can be operated using code to drive the LEDs. It has 2 main parts:

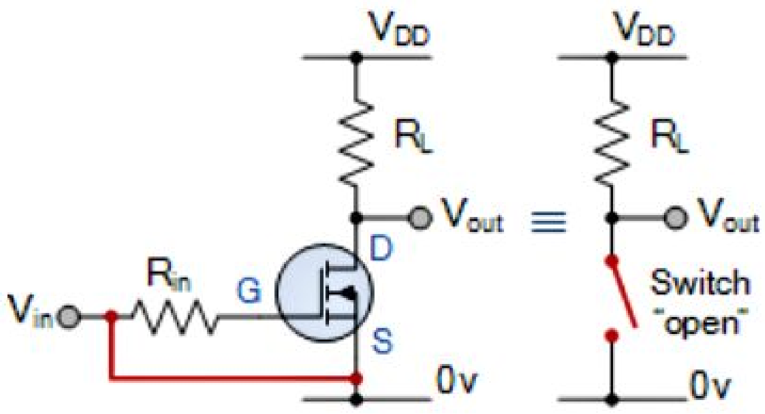
- Switch circuit
- PCB

3.1 Designing the Switch Circuit:

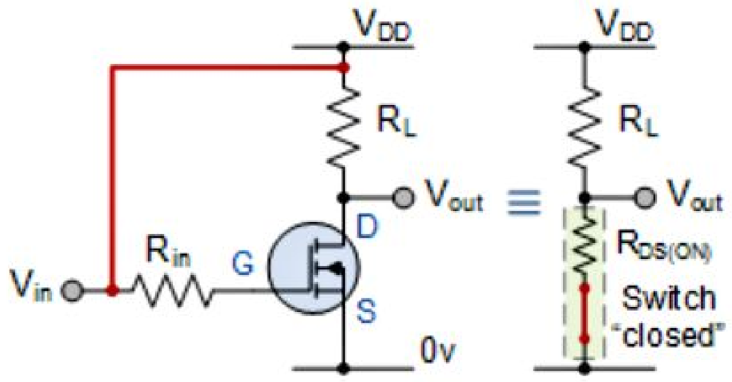
To implement multiplexing, we need to use a switching method in order to turn the horizontal layers ON and OFF quickly and precisely, meaning there shouldn't be any residual voltage when the switch is OFF. For this we came up with the idea of using MOSFETs (Metal Oxide Semiconductor Field Effect Transistor) for the following reasons:

- They can handle large amounts of currents, which is required since they need to drain the current of 96 LEDs.
- The voltage required to close and open the MOSFET is around 2V, which can be easily supplied by shift register output.

nMOS 'OFF' characteristics:



nMOS 'ON' characteristics:



The switching action of the LEDs along the horizontal layers is done using the switching circuit given below, consisting of MOSFETs. There are 12 such switching circuit modules, which are connected to the output pins of 2 shift registers which were given the name cathode shift register 1 and 2. All 8 output pins of the cathode shift register 1 and the first 4 output pins of cathode shift register 2 are connected to the common Gates of the switching module, and the common Drains are connected to the horizontal layers (see Figure 1).

To turn 'OFF' a layer, we must give high voltage to the cathode (reverse bias), and to turn 'ON' a layer, we need to give low voltage to the cathode (forward bias). This must be done irrespective of the voltage at the anode columns. Hence, we need to give 5V at the gate to turn ON a layer, and 0V to turn OFF a layer.

The original circuit consisted of only one nMOSFET for each layer, in which the source was connected to GND along with a 1-2 kΩ resistor connected to VCC. But after testing the circuit on a breadboard, we found that there is some backflow of current flowing through the LEDs even when we want to turn OFF the layer, which is undesirable. The circuit was tested after removing the resistor, but this was discarded too because the nMOSFET was getting overheated due to the short between VCC and GND Hence, we came up with this new switching circuit design consisting of both nMOS and pMOS connected as below:

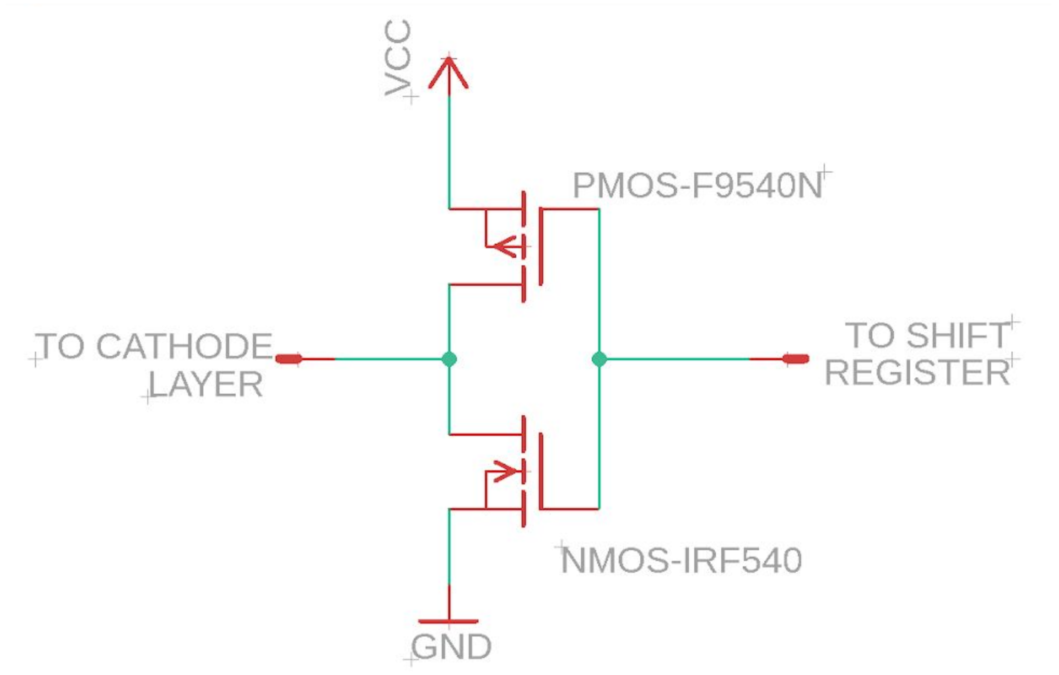


Figure 1 Switching Circuit Module

nMOS IC: IRF540

pMOS IC: F9540N

NOTE: For pMOS, the cutoff and saturation characteristics are the reverse of that for nMOS, i.e., the MOS will turn 'on' when V_{GS} is low, and it will turn 'off' when V_{GS} is high.

We used 12 of these nMOS and pMOS ICs and connected them in a GCB (General Circuit Board) as the PCB we designed earlier didn't have room to accommodate the extra 12 pMOS ICs. We used jumper wires to connect the nMOS pins from the PCB to the ones in the GCB.

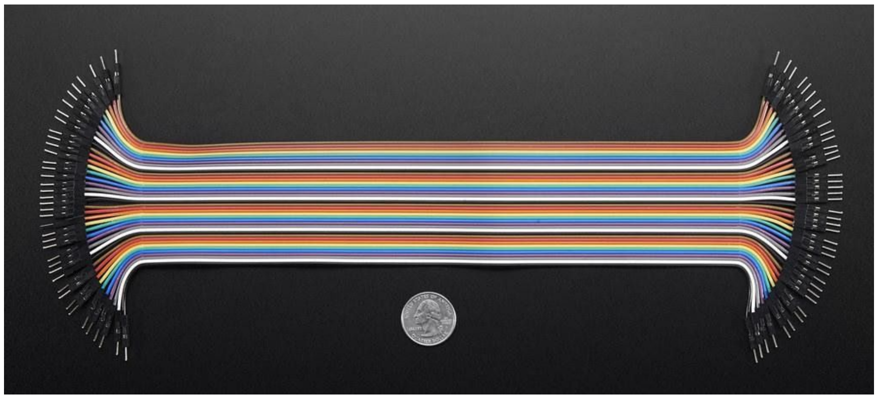


Figure 2 Jumper wires (male-male)

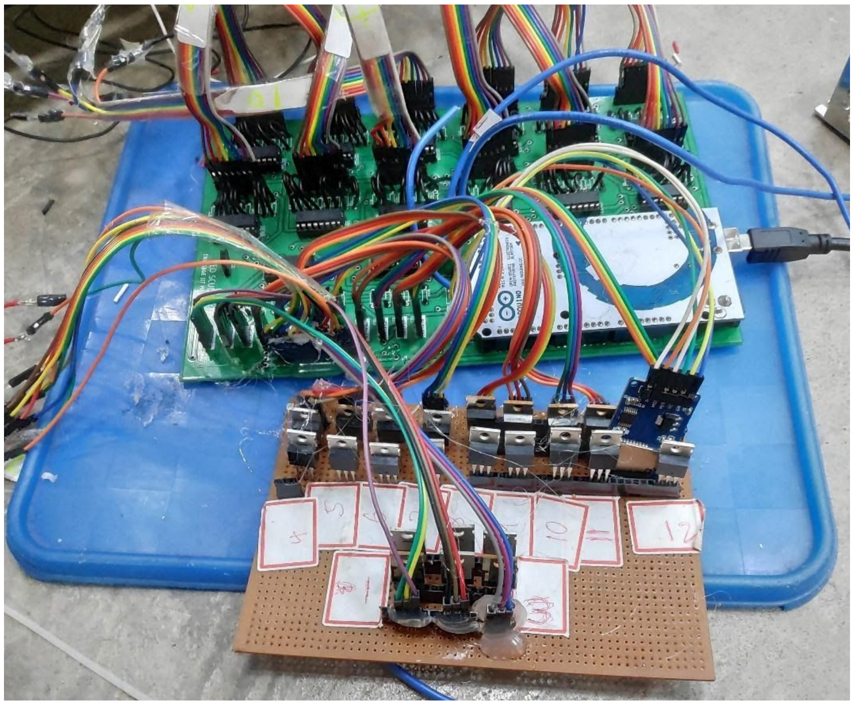


Figure 3 Image of PCB connected to GCB

3.2 Circuit Design- PCB

We designed the PCB in Autodesk Eagle. There are totally 14 Shift Registers (IC: 74HC595) out of which 12 are used for controlling the anodes (12 x 8 = 96 output pins in the shift registers = 96 anode column connections) and 2 for controlling the horizontal cathode layers (2 x 8 = 16 output pins in the shift registers, out of which only 12 are used). An Arduino Mega 2560 was used to send data to the shift registers, so it was given a dedicated space in the PCB. 6 pin slots were given for the SD card reader as well.

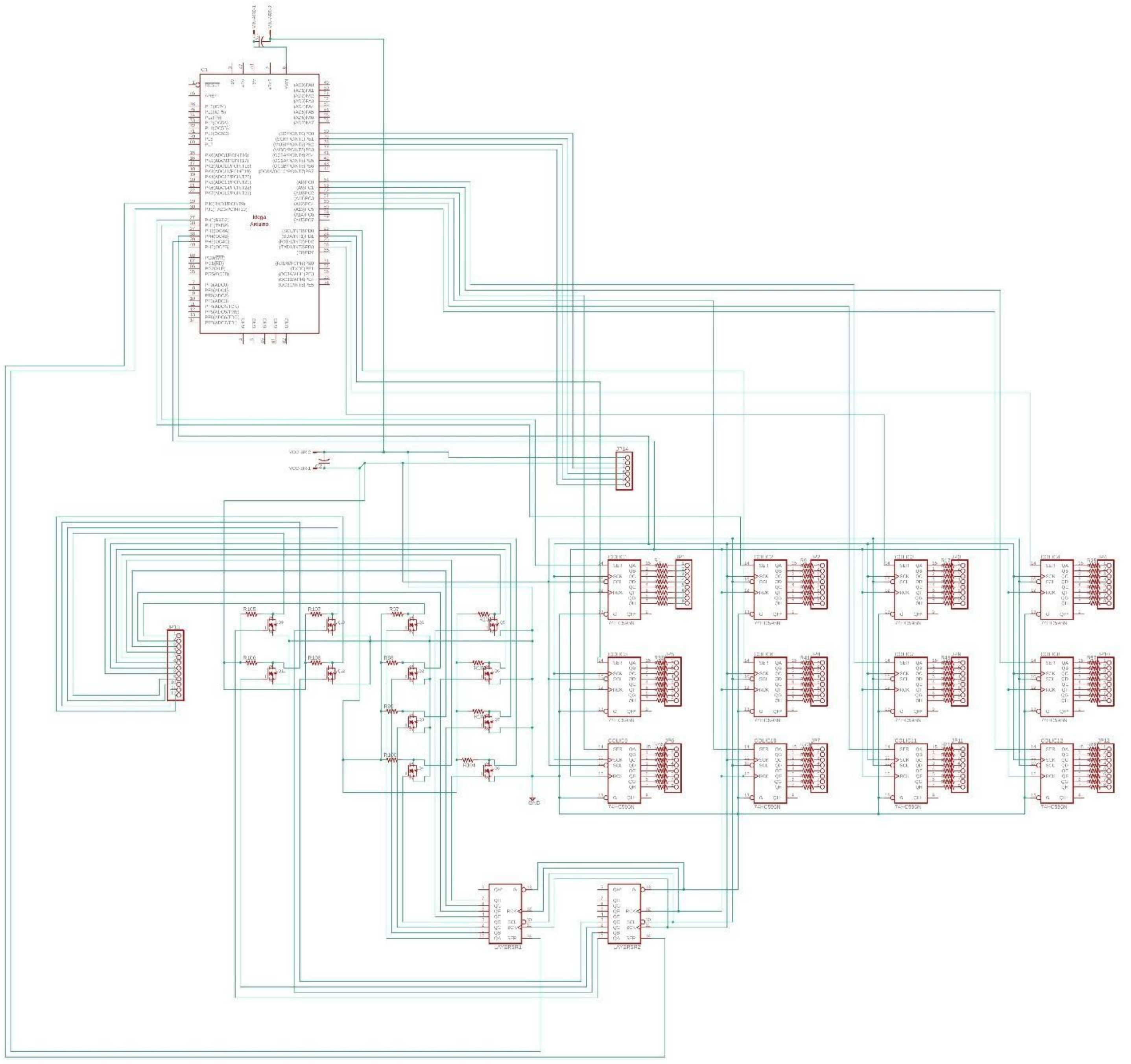


Figure 4 Screenshot of PCB Schematic

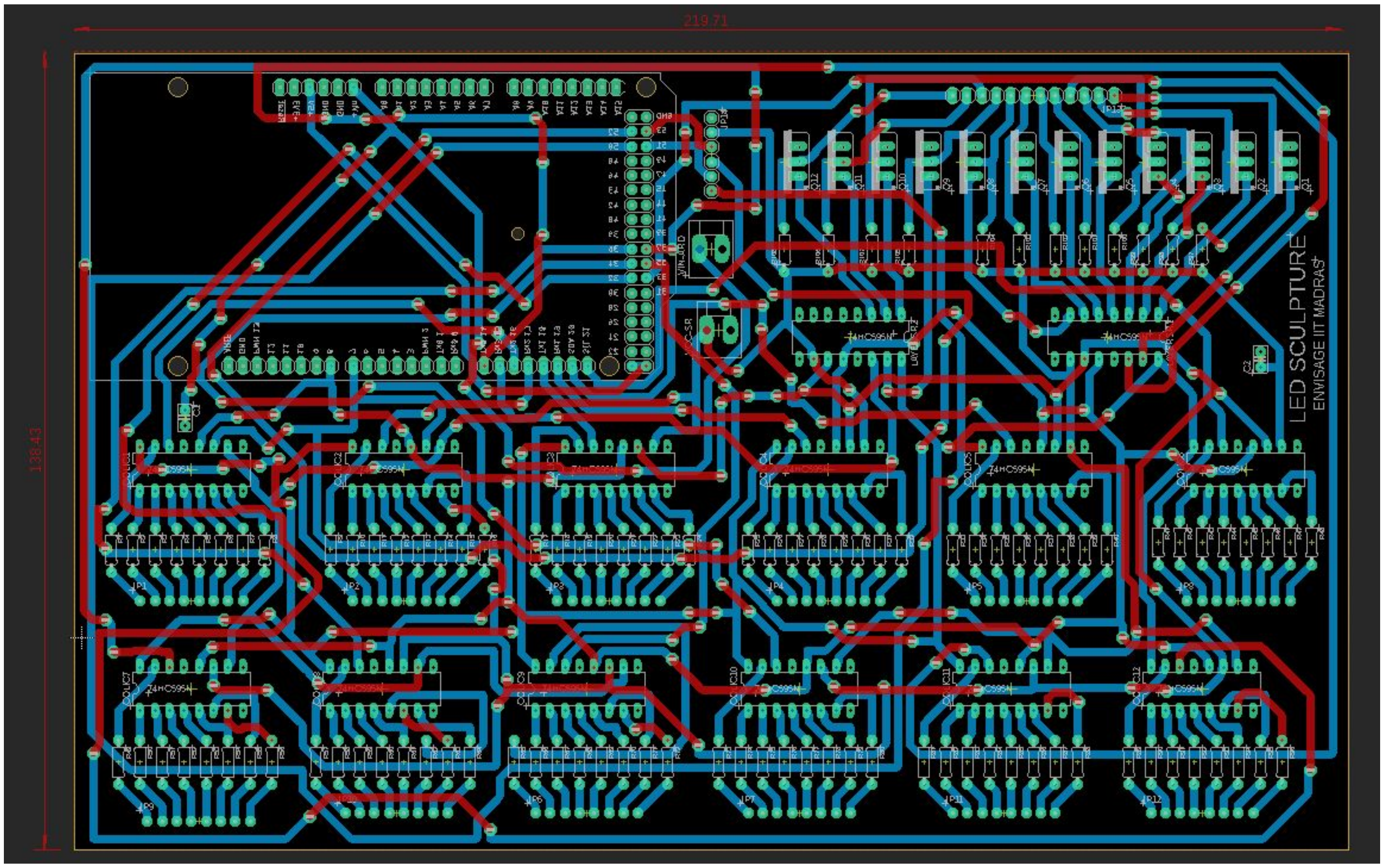


Figure 5 Screenshot of PCB board design

It was planned to use resistors of value 100-200Ω along with each anode connection, but later while testing on a breadboard it was found that the LEDs didn't get overloaded and glowed the brightest when we didn't use any resistors. Hence we had to short circuit all the 96 two pin slots on the PCB which were given for the resistors.

We soldered 16-pin IC bases to the PCB to fix the shift registers in them, 12 sets of 1 x 8 berg pins (female) to hold the male-male jumper cables, which were then soldered with single-stranded wires connected to the anode columns. A set of 1 x 12 berg pins (female) was soldered in the PCB (see the top right corner in the image) to hold the 12 jumper cables connected to the 12 horizontal cathode layers. 1 x 6 berg pins (female) were kept for placing the SD card reader module. We had to connect the SD card reader module to the PCB using male-female jumper cables, since the pins in the Arduino Mega board that were assigned to the

MISO, MOSI and SCK pins in the SD card reader module were connected in wrong order while designing the PCB.

The Arduino Mega board was placed on the PCB using male berg pins. 12 sets of 1 x 3 female berg pins were soldered on the PCB to fix the male-male jumper cables connecting it to the GCB containing the MOSFETs.

4. GENERATING THE CONTROL SIGNALS:

The shift registers which control the LEDs receive the data through Arduino. In this project, to show images, patterns, and spectrum, the coding has been made in Arduino, Python, and Matlab. Further, some of the patterns which were hardcoded and shown are done using a Java link which has been mentioned in the references.

To display 3D images in the 12 x 12 x 8 LED cube, A method called multiplexing was used, which is based on the persistence of vision. An image is displayed one horizontal layer at a time and repeated 20 times (or more depending on the frame rate of the animation) so that our brain perceives it as a 3D image.

4.1 Arduino Code:

This is the main code that receives the data to be displayed in the sculpture. In this code, the dataPin array has the control of all the data pins. There are code lines that are written to communicate with the serial monitor. These are written for debugging purposes. Once all the required pins in the shift register are made to output mode, the file is read. While each line is being read and stored, the data is converted into hex digits and sent to a data string (datastr) which is responsible for sending data to the corresponding shift registers. From there, data goes to cathodes and anodes and the required LED's glow. In this code, ShiftOut is a function that shifts all the characters or data present in each line (96 bits) at a time. This helps in reducing delays and makes the code run faster.

The code is presented below:

```
#include <SD.h>
#include <SPI.h>
char a[29];
File A;                         //Define Files
int pinCS = 53;         //Used to access data from SD card
int latchPin = 8;        //Pin connected to ST_CP of 74HC595
int clockPin = 7;       //Pin connected to SH_CP of 74HC595

int dataPin[] = {14,15,16,17,18,19,20,21,37,36,35,34,33,32};

// First two data pins are for  cathodes, next 12 data pins are for anodes

void setup()
{
  Serial.begin(9600);
  pinMode(pinCS, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  for (int i=0 ; i<15;i++)
  {
    pinMode(dataPin[i], OUTPUT);
  }

  // SD Card Initialization
  if (SD.begin())
  {
    Serial.println("SD card is ready to use.");
  }
  else
  {
    Serial.println("SD card initialization failed");
    return;
  }
   // Reading the file
   A = SD.open("text.txt");      //'text' is the name of the file to be read and send to
Arduino.
}

void loop()
{
  byte datastr[14];
  char ch;
  int b[27];

  for(int k=0;k<14;k++)
  {
    datastr[k]=0;
  }
  for (int p=0;p<27;p++)
  {
    b[p]=0;
  }
  if (A)
  {
     Serial.println("Read:");
     while (A.available())
       {
            ch = A.read(a,29);
            Serial.println(a);
            for (int i=0;i<27;i++)
             {
                if (a[i]>='0' && a[i]<='9')
                {
                    b[i]=(a[i]-'0');
                }
                else if (a[i]>='A' && a[i]<='F')
                {
                   b[i]=(a[i]-'A'+10);
                }
                Serial.print(b[i]);
             }
             datastr[0]=(int)((b[0]<<4)+b[1]);
             datastr[1]=(int)b[2];

             for(int s=2;s<14;s++)
             {
              datastr[s]=(int)((b[(2*s)-1]<<4) +(b[(2*s)])) ;
             }
             Serial.println();

            //lighting the leds
            digitalWrite(latchPin,LOW);
            ShiftOut(dataPin,clockPin,datastr);
            digitalWrite(latchPin,HIGH);
            Serial.println("file closed once");
       }
    }
    else
    {
      Serial.println("error opening file");
    }
     A.close();
}

 // This is a Shiftout function which shifts data to all the output pins of 12 anode shift
registers (12x8= 96 bits) in one call. It shifts all the LSB bits first.

void ShiftOut(int myDataPin[], int myClockPin, byte myDataOut[])
{
   digitalWrite(myClockPin, LOW);
  for (int i=0; i<=7; i++)
  {
    for (int j=0;j<=13;j++)
    {
      digitalWrite(myDataPin[j],(((1<<(7-i)) & (myDataOut[j]))>>(7-i)));
    }
      digitalWrite(myClockPin, HIGH);
      digitalWrite(myClockPin, LOW);
  }
}
```

4.2 Matlab Code:

Fast Fourier Transform(FFT) is a signal processing technique that is used in converting music into its frequency spectrum. In this code, A music file is taken in .wav format. It is then converted into its frequency domain using fftshift function. Then, appropriate frequencies of the music are taken to be shown in the sculpture. Since the amplitudes are continuous, it is divided into groups of ranges and is stored in a matrix, thereby making it into discrete values. Now, in order to make the Spectrum look better, Frequencies that have high amplitudes are shown at the center and the low amplitudes ones at the edges. Then this 2-D matrix is extended to all the 8 layers. Now, this data is sent to a text file which is given to Arduino. The last part of the code shows the bar graph which changes for each discrete time interval thereby acts as a moving picture. It is written only for verification purposes and amplitudes in it are shown in increasing order from left to right.

The code is presented below:

```
clc;
clear all;
close all;
load handel.mat
filename='taki.wav';
clear y;
[y,Fs] = audioread('taki.wav');
dt=1/Fs;
t = 0:dt:(length(y)*dt)-dt;
n=2205;
y_index=1;
h=zeros([round(length(y)/n) 12]);
Y=y(:,1);
file=fopen('taki.txt','wt');
for r = 1:941     %time = 2205 units total widths = (2077883)//2205 = 942. For clear
image,run till 941.
       X=fftshift(fft(Y(r*n:(r+1)*n),n));
       X=abs(X);
       l=length(X);
       X=X(1:(round(l/2)+1));
       x_index=1;
       for c = [1 100 250 300 330 350 450 500 650 775 950 1000]
           if c==1
               d=1;
           end
           h(y_index,x_index)= max(X(d:c));
           x_index=x_index+1;
       end
       y_index=y_index+1;

       cc=zeros([12 12]);  %cc is a 2-D matrix which divides amplitude into 1's and 0's so as
to show the same output in the sculpture.
```

```
       for i = 1:12
           if (h(r,i)>=0.1)
               cc(12,i)=1;
           else
               cc(12,i)=0;
           end
           if (h(r,i)>=0.3)
               cc(11,i)=1;
           else
               cc(11,i)=0;
           end
           if (h(r,i)>=0.5)
               cc(10,i)=1;
           else
               cc(10,i)=0;
           end
           if (h(r,i)>=1.25)
               cc(9,i)=1;
           else
               cc(9,i)=0;
           end
           if (h(r,i)>=1.75)
               cc(8,i)=1;
           else
               cc(8,i)=0;
           end
           if (h(r,i)>=2)
               cc(7,i)=1;
           else
               cc(7,i)=0;
           end
           if (h(r,i)>=3.5)
               cc(6,i)=1;
           else
               cc(6,i)=0;
           end
           if (h(r,i)>=4.6)
               cc(5,i)=1;
           else
               cc(5,i)=0;
           end
           if (h(r,i)>=6)
               cc(4,i)=1;
           else
               cc(4,i)=0;
           end
           if (h(r,i)>=8)
               cc(3,i)=1;
           else
               cc(3,i)=0;
           end
           if (h(r,i)>=20)
               cc(2,i)=1;
           else
               cc(2,i)=0;
           end
           if (h(r,i)>=30 && h(r,i)<=800)  %This is an approximate maximum to prevent
additional errors.
               cc(1,i)=1;
           else
               cc(1,i)=0;
           end
       end

       m=zeros([12 12]);   % m is another 2-D matrix which changes positions of columns in cc
matrix to make the fft look better.
       nn=zeros([12 12 8]);

           m(:,1)=cc(:,1);
           m(:,2)=cc(:,3);
           m(:,3)=cc(:,5);
           m(:,4)=cc(:,7);
           m(:,5)=cc(:,9);
           m(:,6)=cc(:,11);
           m(:,7)=cc(:,12);
           m(:,8)=cc(:,10);
           m(:,9)=cc(:,8);
           m(:,10)=cc(:,6);
           m(:,11)=cc(:,4);
           m(:,12)=cc(:,2);

       nn(:,:,:)=cat(3,m,m,m,m,m,m,m,m);  %nn is a 3-D matrix which extents the fft to all
the 8 layers.
       abcd=zeros([12 12]);
       tempo=0;
        for j=1:4                                                   % 4 times to make sure a
stable picture of each pattern is shown.
            for col=1:12
                for row=1:12
                    for lay=1:8
                        tempo=(nn(row,col,lay)*(2^(lay-1)))+tempo;
                    end
                    abcd(row,col)=tempo;
                    tempo=0;
                end
            end
            for v=1:8

fprintf(file,'%03X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X\n',2^(v+3),abcd(v,:));
            end
            for w=9:12

fprintf(file,'%03X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X\n',2^(w-9),abcd(w,:));
            end
       end
end
 fclose(file);

for o = 1:size(h,1)      % This 'for loop' is to check the output in a bar graph and written
for verification purpose.
        bar(h(o,:));
        ylim([0,15]);
        xlim([0,13]);
        pause(0.1);
end
```

4.3 Python Code:

Sending data to Arduino:

The data is sent to the Arduino one horizontal layer (96 bits) at a time. The 96 bits are converted into hexadecimal form, which amounts to 24 nibbles (1 nibble = 1 hex = 4 bits). A header, consisting of 3 hexes, is added to the beginning of the line to identify which horizontal layer the line's data belongs to. There are a total of 12 x 8 = 96 bits of data in one horizontal layer, which is 24 hexes/24 nibbles. Along with the header, each line would contain 27 nibbles of data for controlling one horizontal layer.

These strings are then stored in a text file in order, which is then stored in an SD card, from which the Arduino reads the data and sends it to the shift registers.

Order of headers:

The headers are for identifying the horizontal layer to which a line of data belongs to. This is done by defining a 12-bit string, and defining the first layer to be: '100000000000', the second as: '010000000000' and so on. These strings are then converted to their hex formats.

Hence, the logical order of the headers should be: `['800', '400', '200', '100', '080', '040', '020', '010', '008', '004', '002', '001']`. However, the order of we used is `['010', '020', '040', '080', '100', '200', '400', '800', '001', '002', '004', '008']`. This is because of the fact that in the Arduino code, the data is sent MSBFIRST into the shift registers. Thus, if 800 is sent into the cathode shift registers, the data received by the cathodes will be: 00000001 0000 (first two nibbles into the first shift register, the third nibble into the second shift register), which will correspond to the eighth cathode layer, not the first. Thus, we had to change the order of the headers because of this.

Generating data for animations using Python:

We are using the python library NumPy to create 3D and 4D 'ndarrays' (matrices) using the function np.zeros(), which initializes all the elements of the matrix to zero. The data for any animation can be divided into frames, which are 3-dimensional still images which when played fast creates the illusion of a moving object/animation. The data for each frame consists of 12 x 12 x 8 = 1152 binary digits/bits, which are stored temporarily in the variable ‘cube’.

For animations generated using python directly, this ‘cube’ object gets operated upon by user-defined functions, thus changing its values which creates new frames. These frames are then stored in a 4D matrix, which is essentially just a collection of 3D 12x12x8 matrices. Some animations were generated in hard-code manner, using the java link: http://have.funoninter.net/LEDCube/, from which the data was extracted and stored in text files, in the manner of one frame in one line. The text files are read and the data is stored directly in the corresponding 4D matrices.

After this, the data from the 4D matrices are read one horizontal layer at a time, converted to hexadecimal format, and stored in a string.

The code is presented below:

```
import numpy as np

'''defining all the variables used'''

# cube is a dummy 3D matrix used for storing a single frame of an animation/pattern.
cube = np.zeros(shape=(12, 12, 8), dtype=int)

# the following are 4D matrices, which are a collection of 3D matrices. The first index
represents the number of frames in that animation.
cube_xz = np.zeros(shape=(12, 12, 12, 8), dtype=int)
cube_xy = np.zeros(shape=(12, 12, 12, 8), dtype=int)
cube_yz = np.zeros(shape=(8, 12, 12, 8), dtype=int)
cube_xy_ff = np.zeros(shape=(24, 12, 12, 8), dtype=int)
cube_xz_ff = np.zeros(shape=(24, 12, 12, 8), dtype=int)
cube_yz_ff = np.zeros(shape=(16, 12, 12, 8), dtype=int)
cube_rainfall = np.zeros(shape=(40, 12, 12, 8), dtype=int)
cube_xzboing = np.zeros(shape=(24, 12, 12, 8), dtype=int)
cube_xyboing = np.zeros(shape=(24, 12, 12, 8), dtype=int)
cube_yzboing = np.zeros(shape=(16, 12, 12, 8), dtype=int)
cube_shrink_grow = np.zeros(shape=(64, 12, 12, 8), dtype=int)
cube_woop = np.zeros(shape=(10, 12, 12, 8), dtype=int)
cube_shuffle = np.zeros(shape=(48, 12, 12, 8), dtype=int)
cube_arrow = np.zeros(shape=(40, 12, 12, 8), dtype=int)
cube_edgefill = np.zeros(shape=(36, 12, 12, 8), dtype=int)
cube_helix = np.zeros(shape=(50, 12, 12, 8), dtype=int)
cube_sine = np.zeros(shape=(12, 12, 12, 8), dtype=int)
cube_socure = np.zeros(shape=(12, 12, 12, 8), dtype=int)
cube_socureletters = np.zeros(shape=(54, 12, 12, 8), dtype=int)
cube_heart = np.zeros(shape=(10, 12, 12, 8), dtype=int)
cube_pac = np.zeros(shape=(56, 12, 12, 8), dtype=int)
cube_pokeball = np.zeros(shape=(16, 12, 12, 8), dtype=int)
cube_snake = np.zeros(shape=(68, 12, 12, 8), dtype=int)
```

```python
cube_dice = np.zeros(shape=(20, 12, 12, 8), dtype=int)
cube_countdown = np.zeros(shape=(40, 12, 12, 8), dtype=int)
cube_bouncing = np.zeros(shape=(40, 12, 12, 8), dtype=int)
cube_ending = np.zeros(shape=(70, 12, 12, 8), dtype=int)
cube_loading = np.zeros(shape=(15, 12, 12, 8), dtype=int)
cube_crab = np.zeros(shape=(14, 12, 12, 8), dtype=int)
cube_gangnam = np.zeros(shape=(17, 12, 12, 8), dtype=int)
cube_running = np.zeros(shape=(12, 12, 12, 8), dtype=int)
cube_ind = np.zeros(shape=(13, 12, 12, 8), dtype=int)
# strings to store the data to be written in the text file
strxy = ''
strxz = ''
stryz = ''
strxyff = ''
stryzff = ''
strxzff = ''
str_rain = ''
strxzboing = ''
strxyboing = ''
stryzboing = ''
strshrink = ''
strwoop = ''
strshuffle = ''
strarrow = ''
stredgefill = ''
strhelix = ''
strsine = ''
strsocure = ''
strsocureletters = ''
strheart = ''
strpac = ''
strpokeball = ''
strsnake = ''
strdice = ''
strcountdown = ''
strbounce = ''
strending = ''
strloading = ''
strcrab = ''
strgangnam = ''
strrunning = ''
strtaki = ''
str_ind = ''

# opening the FFT data sent from MATLAB and storing it in a separate string
taki = open(r'C:\Users\karth\OneDrive\Desktop\taki.txt', 'r')
takilist = taki.readlines()
for ctr in takilist:
    strtaki += ctr

'''Defining the functions in which the animation/pattern data is generated. At the end of each
function, all the frame-data is stored in a 4D matrix. In some functions like pokeball(),
pacman(), etc, the hard code data in the text file is read and stored in a 4D matrix.'''

def xz():
    for i in range(12):
        cube[:, :, :] = 0
        cube[i, :, :] = 1
        cube_xz[i, :, :, :] = cube
```

```python
def xy():
    for i in range(12):
        cube[:, :, :] = 0
        cube[:, i, :] = 1
        cube_xy[i, :, :, :] = cube

def yz():
    for i in range(8):
        cube[:, :, :] = 0
        cube[:, :, i] = 1
        cube_yz[i, :, :, :] = cube

def xzboing():
    for i in range(12):
        cube[:, :, :] = 0
        cube[i, :, :] = 1
        cube_xzboing[i, :, :, :] = cube
    for i in reversed(range(12)):
        cube[:, :, :] = 0
        cube[i, :, :] = 1
        cube_xzboing[23-i, :, :, :] = cube

def xyboing():
    for i in range(12):
        cube[:, :, :] = 0
        cube[:, i, :] = 1
        cube_xyboing[i, :, :, :] = cube
    for i in reversed(range(12)):
        cube[:, :, :] = 0
        cube[:, i, :] = 1
        cube_xyboing[23-i, :, :, :] = cube

def yzboing():
    for i in range(8):
        cube[:, :, :] = 0
        cube[:, :, i] = 1
        cube_yzboing[i, :, :, :] = cube
    for i in reversed(range(8)):
        cube[:, :, :] = 0
        cube[:, :, i] = 1
        cube_yzboing[15-i, :, :, :] = cube

def xy_ff():
    cube[:, :, :] = 0
    for i in range(12):
        cube[:, i, :] = 1
        cube_xy_ff[i, :, :, :] = cube
    for i in range(12):
        cube[:, i, :] = 0
        cube_xy_ff[i, :, :, :] = cube

def xz_ff():
    cube[:, :, :] = 0
    for i in range(12):
        cube[i, :, :] = 1
        cube_xy_ff[i, :, :, :] = cube

    for i in range(12):
        cube[i, :, :] = 0
        cube_xy_ff[i, :, :, :] = cube
```

```
def yz_ff():
    cube[:, :, :] = 0
    for i in range(8):
        cube[:, :, i] = 1
        cube_xy_ff[i, :, :, :] = cube

    for i in range(8):
        cube[:, :, i] = 0
        cube_xy_ff[i, :, :, :] = cube

def rainfall():
    cube[:, :, :] = 0
    for i in range(40):
        if i >= 1:
            for j in reversed(range(1, 12)):
                cube[:, j, :] = cube[:, j - 1, :]

        cube[:, 0, :] = np.random.randint(2, size=(12, 8))
        cube_rainfall[i, :, :, :] = cube

def shrink_grow():
    for i in range(8):  # (0,11,0)
        cube[:, :, :] = 0

        cube[0, i:12, 0] = 1
        cube[11-i, i:12, 0] = 1
        cube[11-i, i:12, 7-i] = 1
        cube[0, i:12, 7-i] = 1

        cube[0:12-i, i, 0] = 1
        cube[0, i, 0:8-i] = 1
        cube[0:12-i, i, 7-i] = 1
        cube[11-i, i, 0:8-i] = 1

        cube[0:12-i, 11, 0] = 1
        cube[0, 11, 0:8-i] = 1
        cube[0:12-i, 11, 7-i] = 1
        cube[11-i, 11, 0:8-i] = 1

        cube_shrink_grow[i, :, :, :] = cube

    for i in reversed(range(8)):  # (0,11,0)
        cube[:, :, :] = 0

        cube[0, i:12, 0] = 1
        cube[11-i, i:12, 0] = 1
        cube[11-i, i:12, 7-i] = 1
        cube[0, i:12, 7-i] = 1

        cube[0:12-i, i, 0] = 1
        cube[0, i, 0:8-i] = 1
        cube[0:12-i, i, 7-i] = 1
        cube[11-i, i, 0:8-i] = 1

        cube[0:12-i, 11, 0] = 1
        cube[0, 11, 0:8-i] = 1
        cube[0:12-i, 11, 7-i] = 1
        cube[11-i, 11, 0:8-i] = 1
        cube_shrink_grow[15-i, :, :, :] = cube

    for i in range(8):  # (0,11,7)
        cube[:, :, :] = 0

        cube[0, i:12, i] = 1
        cube[11-i, i:12, i] = 1
        cube[11-i, i:12, 7] = 1
        cube[0, i:12, 7] = 1

        cube[0, i, i:8] = 1
        cube[11-i, i, i:8] = 1
        cube[0:12-i, i, i] = 1
        cube[0:12-i, i, 7-i] = 1

        cube[0, 11, i:8] = 1
        cube[11-i, 11, i:8] = 1
        cube[0:12-i, 11, i] = 1
        cube[0:12-i, 11, 7-i] = 1

        cube_shrink_grow[16+i, :, :, :] = cube

    for i in reversed(range(8)):  # (0,11,7)
        cube[:, :, :] = 0

        cube[0, i:12, i] = 1
        cube[11-i, i:12, i] = 1
        cube[11-i, i:12, 7] = 1
        cube[0, i:12, 7] = 1

        cube[0, i, i:8] = 1
        cube[11-i, i, i:8] = 1
        cube[0:12-i, i, i] = 1
        cube[0:12-i, i, 7-i] = 1

        cube[0, 11, i:8] = 1
        cube[11-i, 11, i:8] = 1
        cube[0:12-i, 11, i] = 1
        cube[0:12-i, 11, 7-i] = 1

        cube_shrink_grow[16+15-i, :, :, :] = cube

    for i in range(8):  # (0,0,0)
        cube[:, :, :] = 0
        cube[0, 0:12-i, 0] = 1
        cube[0, 0:12-i, 7-i] = 1
        cube[11-i, 0:12-i, 0] = 1
        cube[11-i, 0:12-i, 7-i] = 1

        cube[0:12-i, 0, 0] = 1
        cube[0:12-i, 0, 7-i] = 1
        cube[0, 0, 0:8-i] = 1
        cube[11-i, 0, 0:8-i] = 1

        cube[0:12-i, 11-i, 0] = 1
        cube[0:12-i, 11-i, 7-i] = 1
        cube[0, 11-i, 0:8-i] = 1
        cube[11-i, 11-i, 0:8-i] = 1

        cube_shrink_grow[32+i, :, :, :] = cube
    for i in reversed(range(8)):  # (0,0,0)
        cube[:, :, :] = 0
        cube[0, 0:12 - i, 0] = 1
        cube[0, 0:12 - i, 7 - i] = 1
        cube[11 - i, 0:12 - i, 0] = 1
        cube[11 - i, 0:12 - i, 7 - i] = 1

        cube[0:12 - i, 0, 0] = 1
        cube[0:12 - i, 0, 7 - i] = 1
        cube[0, 0, 0:8 - i] = 1
        cube[11 - i, 0, 0:8 - i] = 1

        cube[0:12 - i, 11 - i, 0] = 1
        cube[0:12 - i, 11 - i, 7 - i] = 1
        cube[0, 11 - i, 0:8 - i] = 1
        cube[11 - i, 11 - i, 0:8 - i] = 1

        cube_shrink_grow[32+15-i, :, :, :] = cube

    for i in range(8):  # (0,0,7)
        cube[:, :, :] = 0
        cube[0, 0:12-i, i] = 1
        cube[0, 0:12-i, 7] = 1
        cube[11-i, 0:12-i, i] = 1
        cube[11-i, 0:12-i, 7] = 1

        cube[0, 0, i:8] = 1
        cube[11-i, 0, i:8] = 1
        cube[0:12-i, 0, i] = 1
        cube[0:12-i, 0, 7] = 1

        cube[0, 11-i, i:8] = 1
        cube[11-i, 11-i, i:8] = 1
        cube[0:12-i, 11-i, i] = 1
        cube[0:12-i, 11-i, 7] = 1

        cube_shrink_grow[48+i, :, :, :] = cube

    for i in reversed(range(8)):  # (0,0,7)
        cube[:, :, :] = 0
        cube[0, 0:12 - i, i] = 1
        cube[0, 0:12 - i, 7] = 1
        cube[11 - i, 0:12 - i, i] = 1
        cube[11 - i, 0:12 - i, 7] = 1

        cube[0, 0, i:8] = 1
        cube[11 - i, 0, i:8] = 1
        cube[0:12 - i, 0, i] = 1
        cube[0:12 - i, 0, 7] = 1

        cube[0, 11 - i, i:8] = 1
        cube[11 - i, 11 - i, i:8] = 1
        cube[0:12 - i, 11 - i, i] = 1
        cube[0:12 - i, 11 - i, 7] = 1

        cube_shrink_grow[48+15-i, :, :, :] = cube

def woop_woop():
    for i in range(5):
        cube[:, :, :] = 0
        cube[i:12-i, i, i] = 1
        cube[i:12-i, i, 7-i] = 1
        cube[i:12-i, 11-i, i] = 1
        cube[i:12-i, 11-i, 7-i] = 1

        cube[i, i:12-i, i] = 1
        cube[11-i, i:12-i, i] = 1
        cube[i, i:12-i, 7-i] = 1
        cube[11-i, i:12-i, 7-i] = 1

        cube[i, i, i:7-i] = 1
        cube[11-i, i, i:7-i] = 1
        cube[i, 11-i, i:7-i] = 1
        cube[11-i, 11-i, i:7-i] = 1

        cube_woop[i, :, :, :] = cube

    for i in reversed(range(5)):
        cube[:, :, :] = 0
        cube[i:12-i, i, i] = 1
        cube[i:12-i, i, 7-i] = 1
        cube[i:12-i, 11-i, i] = 1
        cube[i:12-i, 11-i, 7-i] = 1

        cube[i, i:12-i, i] = 1
        cube[11-i, i:12-i, i] = 1
        cube[i, i:12-i, 7-i] = 1
        cube[11-i, i:12-i, 7-i] = 1

        cube[i, i, i:8-i] = 1
        cube[11-i, i, i:8-i] = 1
        cube[i, 11-i, i:8-i] = 1
        cube[11-i, 11-i, i:8-i] = 1

        cube_woop[9-i, :, :, :] = cube

def shuffle_plane():
    for i in range(12):
        cube[:, :, :] = 0
        cube[:, 0, :] = 1
        cube[:, 11, :] = 1
        cube[0, :, :] = 1
        cube[11, :, :] = 1

        cube[i, :, :] = 1
        cube_shuffle[i, :, :, :] = cube

    for i in range(12):
        cube[:, :, :] = 0
        cube[:, 0, :] = 1
        cube[:, 11, :] = 1
        cube[0, :, :] = 1
        cube[11, :, :] = 1

        cube[:, i, :] = 1
        cube_shuffle[12+i, :, :, :] = cube

    for i in reversed(range(12)):
        cube[:, :, :] = 0
        cube[:, 0, :] = 1
        cube[:, 11, :] = 1
        cube[0, :, :] = 1
        cube[11, :, :] = 1

        cube[i, :, :] = 1
        cube_shuffle[24+11-i, :, :, :] = cube

    for i in reversed(range(12)):
        cube[:, :, :] = 0
        cube[:, 0, :] = 1
        cube[:, 11, :] = 1
        cube[0, :, :] = 1
        cube[11, :, :] = 1

        cube[:, i, :] = 1
        cube_shuffle[36+11-i, :, :, :] = cube

def arrow():
    s = '0000000000000000000000000000000000000001000001100011000011100111111111' \
           '00011111111100001100001110000100000110000000000000000000000000000000000000'
    cube2 = np.zeros(shape=(12, 40), dtype=int)
    for i in range(12):
        for j in range(12):
            cube2[i, j] = int(s[12*i+j])

    for i in range(40):
        cube[:, :, 0] = cube2[:, 0:12]
        cube[0, :, :] = cube2[:, 12:20]
        cube[:, :, 7] = cube2[:, 20:32]
        cube[11, :, :] = cube2[:, 32:40]
        cube2[:, i-1] = cube2[:, i]
        cube_arrow[i, :, :, :] = cube

def edgefill():
    file = open(r'C:\Users\karth\OneDrive\Desktop\edgefill.txt', 'r')
    # opening the file containing the hard code data generated using the website
    s = file.readlines()
    for i in range(36):
        s[i] = s[i].rstrip()
    for i in range(36):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_edgefill[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    file.close()

def helix():
    s =
'000011000000000000000000000000000000000000000000000000000000000000000000000000000000000000110
000' \

'000000110000000000000000000000000000000000000000000000000000000000000000000000000000000011000
000' \

'000000000000000000001000000000000000000000000000000000000000000000000000001000000000000000000
000' \

'000000000000000000000000000000000100000000000100001000000000001000000000000000000000000000000
000' \
'000000000000000000000000001000000000001000000000000000000100000000000100000000000000000000000
000' \

'000000000000000100000000000000000000000000000000000000000000000000000000000000001000000000000
000'
    cube[:, :, :] = 0
    for i in range(6):
        for j in range(8):
            for x in range(12):
                cube[x, i, j] = int(s[12*8*i + 12*j + x])
                cube[x, 6+i, j] = int(s[12*8*i + 12*j + x])

    for j in range(50):
        for i in reversed(range(12)):
            cube[:, i, :] = cube[:, i-1, :]
        cube_helix[j, :, :, :] = cube

def sine():
    file = open(r'C:\Users\karth\OneDrive\Desktop\sine.txt', 'r')
    s = file.readlines()
    for i in range(12):
        s[i] = s[i].strip() # to remove all the newlines which can't be converted into
integer.
    for i in range(12):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_sine[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    file.close()

def socure():
    file = open(r'C:\Users\karth\OneDrive\Desktop\socure.txt', 'r')
    s = file.readlines()
    for i in range(8):
        s[i] = s[i].rstrip()
    for i in range(8):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_socure[i, x, y, z] = int(s[i][12*12*z + 12*y + x])
    file.close()
    cube_socure[8, :, :, :] = cube_socure[9, :, :, :] = cube_socure[10, :, :, :] =
cube_socure[11, :, :, :] \
        = cube_socure[7, :, :, :]

def socure_letters():
    cube[:, :, :] = 0
    s =
'000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000' \

'000000000000000000000001100011000110100101110011100000000000000000000000000010010100101000100
101001' \

'010000000000000000000000000000010000100101000100101001010000000000000000000000000000001100100
1010001' \

'001011100111000000000000000000000000000000101001010001001011000100000000000000000000000000000
10010100' \
'101000100101010010000000000000000000000000000000110001100011001100100101110000000000000000000
000000000' \

'000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000' \
        '000000000000000000000000000000000000000000000'
    cube2 = np.zeros(shape=(12, 54), dtype=int)
    for i in range(12):
        for j in range(54):
            cube2[i, j] = int(s[12*i + j])

    for j in range(54):
        for i in range(12):
            cube[i, :, 0] = cube2[:, i]
            cube2[:, j] = cube2[:, j-1]
        cube_socureletters[j, :, :, :] = cube

def heart():
    file = open(r'C:\Users\karth\OneDrive\Desktop\heart.txt', 'r')
    s = file.readlines()
    for i in range(10):
        s[i] = s[i].rstrip()
    for i in range(10):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_heart[i, x, y, z] = int(s[i][12*12*z + 12*y + x])
    file.close()

def pacman():
    file = open(r'C:\Users\karth\OneDrive\Desktop\pacman.txt', 'r')
    s = file.readlines()
    for i in range(14):
        s[i] = s[i].rstrip()
    for i in range(14):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_pac[i, x, y, z] = int(s[i][12*12*z + 12*y + x])
    for i in range(14):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_pac[14+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(14):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_pac[28+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(14):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_pac[42+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])

    file.close()

def pokeball():
    file = open(r'C:\Users\karth\OneDrive\Desktop\pokeball.txt', 'r')
    s = file.readlines()
    for i in range(16):
        s[i] = s[i].rstrip()
    for i in range(16):
        for x in range(12):
            for y in range(12):
                cube_pokeball[i, x, y, 0] = int(s[i][12*y + x])
    file.close()

def gangnam():
    file = open(r'C:\Users\karth\OneDrive\Desktop\gangnam.txt', 'r')
    s = file.readlines()
    for i in range(6):
        s[i] = s[i].rstrip()
    for i in range(17):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_gangnam[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])

    file.close()

def running_man():
    file = open(r'C:\Users\karth\OneDrive\Desktop\runningman.txt', 'r')
    s = file.readlines()
    for i in range(12):
        s[i] = s[i].rstrip()
    for i in range(12):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_running[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])

    file.close()

def snake():
    file = open(r'C:\Users\karth\OneDrive\Desktop\snake_game.txt', 'r')
    s = file.readlines()
    for i in range(62):
        s[i] = s[i].rstrip()
    for i in range(62):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_snake[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    file.close()

def dice():
    file = open(r'C:\Users\karth\OneDrive\Desktop\shaastra_dice.txt', 'r')
    s = file.readline()
    cube2 = np.zeros(shape=(12, 20), dtype=int)
    for i in range(12):
        for j in range(20):
            cube2[i, j] = int(s[12*i + j])

    cube[:, :, :] = 0

    for i in range(20):
        cube2[:, i-1] = cube2[:, i]
        for j in range(12):
            cube[j, :, 0] = cube2[:, j]
        cube_dice[i, :, :, :] = cube

def countdown():
    s5 = '0000000000000000000000000001111100000001000000000001000000000001' \
         '11100000000000010000000000010000000100010000000011100000000000000000000000000000'
    s4 = '000000000000000000000000000100100000000100100000000100100000000' \
         '100100000000111111000000000100000000000100000000000100000000000000000000000000000'
    s3 = '000000000000000011100000000100010000000000010000000000010000000011' \
         '100000000000010000000000010000000100010000000011100000000000000000000000000000'
    s2 = '000000000000000001100000000010010000000000010000000000010000000001' \
         '100000000010000000000010000000000010000000000011110000000000000000000000000000'
    s1 = '000000000000000000100000000001100000000010100000000000100000000000100' \
         '000000000100000000000100000000000100000000011110000000000000000000000000000'

    c5 = np.zeros(shape=(12, 12), dtype=int)
    c4 = np.zeros(shape=(12, 12), dtype=int)
    c3 = np.zeros(shape=(12, 12), dtype=int)
    c2 = np.zeros(shape=(12, 12), dtype=int)
    c1 = np.zeros(shape=(12, 12), dtype=int)

    for i in range(12):
        for j in range(12):
            c5[j, i] = int(s5[12*i + j])
            c4[j, i] = int(s4[12*i + j])
            c3[j, i] = int(s3[12*i + j])
            c2[j, i] = int(s2[12*i + j])
            c1[j, i] = int(s1[12*i + j])

    for i in reversed(range(8)):
        cube[:, :, :] = 0
        cube[:, :, i] = c5
        cube_countdown[7-i, :, :, :] = cube
    for i in reversed(range(8)):
        cube[:, :, :] = 0
        cube[:, :, i] = c4
        cube_countdown[14-i, :, :, :] = cube
    for i in reversed(range(8)):
        cube[:, :, :] = 0
        cube[:, :, i] = c3
        cube_countdown[21-i, :, :, :] = cube
    for i in reversed(range(8)):
        cube[:, :, :] = 0
        cube[:, :, i] = c2
        cube_countdown[28-i, :, :, :] = cube
    for i in reversed(range(8)):
        cube[:, :, :] = 0
        cube[:, :, i] = c1
        cube_countdown[35-i, :, :, :] = cube

def cube_bounce():
    file = open(r'C:\Users\karth\OneDrive\Desktop\cube_bouncing.txt', 'r')
    s = file.readlines()
    for i in range(40):
        s[i] = s[i].rstrip()
    for i in range(40):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_bouncing[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    file.close()

def stack():
    for j in range(12):
        cube[:, 11-j, :] = 1
        for i in range(12):
            cube[:, :, 0:12-j] = 0
            cube[:, i, :] = 1
            cube[i, :, :, :] = cube

def ending():
    file = open(r'C:\Users\karth\OneDrive\Desktop\ending.txt', 'r')
    s = file.readlines()
    for i in range(7):
        s[i] = s[i].rstrip()

    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[7+i, x, y, z] = int(s[6-i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[14+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[21+i, x, y, z] = int(s[6-i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[28+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[35+i, x, y, z] = int(s[6-i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[42+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[49+i, x, y, z] = int(s[6-i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[56+i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    for i in range(7):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ending[63+i, x, y, z] = int(s[6-i][12 * 12 * z + 12 * y + x])
    file.close()


def loading():
    file = open(r'C:\Users\karth\OneDrive\Desktop\loading.txt', 'r')
    s = file.readlines()
    for i in range(12):
        s[i] = s[i].rstrip()
    for i in range(12):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_loading[i, x, y, z] = int(s[i][12*12*z + 12*y + x])
    file.close()


def crab():
    file = open(r'C:\Users\karth\OneDrive\Desktop\crab.txt', 'r')
    s = file.readlines()
    for i in range(14):
        s[i] = s[i].rstrip()
    for i in range(14):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_crab[i, x, y, z] = int(s[i][12*12*z + 12*y + x])
    file.close()


def indicator():
    file = open(r'C:\Users\karth\OneDrive\Desktop\indicator.txt', 'r')
    s = file.readlines()
    for i in range(13):
        s[i] = s[i].rstrip()
    for i in range(13):
        for x in range(12):
            for y in range(12):
                for z in range(8):
                    cube_ind[i, x, y, z] = int(s[i][12 * 12 * z + 12 * y + x])
    file.close()

'''Calling all the functions to generate the data for the animations.'''
xz()
xy()
yz()
xy_ff()
xz_ff()
yz_ff()
rainfall()
xzboing()
xyboing()
yzboing()
shrink_grow()
woop_woop()
```

```
shuffle_plane()
arrow()
edgefill()
helix()
sine()
socure()
snake()
crab()
socure_letters()
heart()
pacman()
pokeball()
dice()
countdown()
cube_bounce()
ending()
loading()
running_man()
gangnam()
indicator()

''' Conversion of the 3D image data into hex data'''

header = ['010', '020', '040', '080', '100', '200', '400', '800', '001', '002', '004', '008']
# This array contains the headers (in hexadecimal format) which need to be added at the
beginning
# of each line. Each line has 27 nibbles (1 nibble = 1 hexadecimal digit), the first 3 nibbles
is
# the header, the remaining 24 are the anode data.

''' The following for loops read the matrices frame by frame and convert the binary data into
hex and then store it in the corresponding string'''

for k in range(12):
    for a in range(10): # repeating each frame 10 times
        for q in range(12):
            strxy += header[q] # adding the header in the beginning
            strxz += header[q]
            for p in range(12):
                # str1 and str2 are dummy variables used to store anode data in one layer
                # syntax for joining elements from a 1D array into a string
                str1 = ''.join(str(e) for e in cube_xy[k, p, q, :])
                str2 = ''.join(str(e) for e in cube_xz[k, p, q, :])
                strxy += "{0:0>2X}".format(int(str1, 2)) # syntax for converting binary string
to hex
                strxz += "{0:0>2X}".format(int(str2, 2))
            strxy += '\n'
            strxz += '\n'

for k in range(8):
    for a in range(10):
        for q in range(12):
            stryz += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_yz[k, p, q, :])
                stryz += "{0:0>2X}".format(int(str1, 2))

            stryz += '\n'

for k in range(24):
    for a in range(10):
        for q in range(12):
            strxyff += header[q]
            strxzff += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_xy_ff[k, p, q, :])
                str2 = ''.join(str(e) for e in cube_xz_ff[k, p, q, :])
                strxyff += "{0:0>2X}".format(int(str1, 2))
                strxzff += "{0:0>2X}".format(int(str2, 2))

            strxyff += '\n'
            strxzff += '\n'

for k in range(16):
    for a in range(10):
        for q in range(12):
            stryzff += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_yz_ff[k, p, q, :])
                stryzff += "{0:0>2X}".format(int(str1, 2))

            stryzff += '\n'

for k in range(24):
    for a in range(10):
        for q in range(12):
            strxyboing += header[q]
            strxzboing += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_xyboing[k, p, q, :])
                str2 = ''.join(str(e) for e in cube_xzboing[k, p, q, :])
                strxyboing += "{0:0>2X}".format(int(str1, 2))
                strxzboing += "{0:0>2X}".format(int(str2, 2))

            strxyboing += '\n'
            strxzboing += '\n'

for k in range(16):
    for a in range(10):
        for q in range(12):
            stryzboing += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_yzboing[k, p, q, :])
                stryzboing += "{0:0>2X}".format(int(str1, 2))

            stryzboing += '\n'

for k in range(40):
    for a in range(10):
        for q in range(12):
            str_rain += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_rainfall[k, p, q, :])
                str_rain += "{0:0>2X}".format(int(str1, 2))

            str_rain += '\n'

for k in range(64):
    for a in range(8):
        for q in range(12):
            strshrink += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_shrink_grow[k, p, q, :])
                strshrink += "{0:0>2X}".format(int(str1, 2))
            strshrink += '\n'

for times in range(2): # repeating the whole animation 2 times
    for k in range(10):
        for a in range(10):
            for q in range(12):
                strwoop += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_woop[k, p, q, :])
                    strwoop += "{0:0>2X}".format(int(str1, 2))
                strwoop += '\n'

for times in range(1):
    for k in range(48):
        for a in range(10):
            for q in range(12):
                strshuffle += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_shuffle[k, p, q, :])
                    strshuffle += "{0:0>2X}".format(int(str1, 2))
                strshuffle += '\n'

for k in range(40):
    for a in range(10):
        for q in range(12):
            strarrow += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_arrow[k, p, q, :])
                strarrow += "{0:0>2X}".format(int(str1, 2))
            strarrow += '\n'

for k in range(36):
    for a in range(10):
        for q in range(12):
            stredgefill += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_edgefill[k, p, q, :])
                stredgefill += "{0:0>2X}".format(int(str1, 2))
            stredgefill += '\n'

for times in range(2): # repeating the whole animation twice
    for k in range(50):
        for a in range(10):
            for q in range(12):
                strhelix += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_helix[k, p, q, :])
                    strhelix += "{0:0>2X}".format(int(str1, 2))
                strhelix += '\n'

for times in range(6): # repeating the whole animation 6 times
    for k in range(12):
        for a in range(10):
            for q in range(12):
                strsine += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_sine[k, p, q, :])
                    strsine += "{0:0>2X}".format(int(str1, 2))
                strsine += '\n'

for k in range(12):
    for a in range(40):
        for q in range(12):
            strsocure += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_socure[k, p, q, :])
                strsocure += "{0:0>2X}".format(int(str1, 2))
            strsocure += '\n'

for k in range(54):
    for a in range(10):
        for q in range(12):
            strsocureletters += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_socureletters[k, p, q, :])
                strsocureletters += "{0:0>2X}".format(int(str1, 2))
            strsocureletters += '\n'

for times in range(6): # repeating the whole animation 6 times
    for k in range(10):
        for a in range(10):
            for q in range(12):
                strheart += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_heart[k, p, q, :])
                    strheart += "{0:0>2X}".format(int(str1, 2))

                strheart += '\n'

for k in range(56):
    for a in range(20):
        for q in range(12):
            strpac += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_pac[k, p, q, :])
                strpac += "{0:0>2X}".format(int(str1, 2))

            strpac += '\n'

for k in range(15):
    for a in range(21):
        for q in range(12):
            strpokeball += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_pokeball[k, p, q, :])
                strpokeball += "{0:0>2X}".format(int(str1, 2))
            strpokeball += '\n'

for k in range(62):
    for a in range(12):
        for q in range(12):
            strsnake += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_snake[k, p, q, :])
                strsnake += "{0:0>2X}".format(int(str1, 2))
            strsnake += '\n'
```

```
for k in range(20):
    for a in range(10):
        for q in range(12):
            strdice += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_dice[k, p, q, :])
                strdice += "{0:0>2X}".format(int(str1, 2))
            strdice += '\n'

for k in range(40):
    for a in range(10):
        for q in range(12):
            strcountdown += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_countdown[k, p, q, :])
                strcountdown += "{0:0>2X}".format(int(str1, 2))
            strcountdown += '\n'

for k in range(40):
    for a in range(10):
        for q in range(12):
            strbounce += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_bouncing[k, p, q, :])
                strbounce += "{0:0>2X}".format(int(str1, 2))

            strbounce += '\n'
f = 0
for k in range(70):
    if k in range(14): # This is done to make the animation to run faster after every 14
frames
        f = 10
    elif k in range(14, 28):
        f = 8
    elif k in range(28, 42):
        f = 7
    elif k in range(42, 56):
        f = 6
    elif k in range(56, 70):
        f = 4
    for a in range(f):
        for q in range(12):
            strending += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_ending[k, p, q, :])
                strending += "{0:0>2X}".format(int(str1, 2))
            strending += '\n'

for k in range(15):
    for a in range(25):
        for q in range(12):
            strloading += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_loading[k, p, q, :])
                strloading += "{0:0>2X}".format(int(str1, 2))
            strloading += '\n'

for k in range(14):
    for a in range(20):
        for q in range(12):
            strcrab += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_crab[k, p, q, :])
                strcrab += "{0:0>2X}".format(int(str1, 2))
            strcrab += '\n'

for times in range(4): # repeating the whole animation 4 times
    for k in range(17):
        for a in range(15):
            for q in range(12):
                strgangnam += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_gangnam[k, p, q, :])
                    strgangnam += "{0:0>2X}".format(int(str1, 2))

                strgangnam += '\n'

for times in range(2): # repeating the whole animation twice
    for k in range(12):
        for a in range(20):
            for q in range(12):
                strrunning += header[q]
                for p in range(12):
                    str1 = ''.join(str(e) for e in cube_running[k, p, q, :])
                    strrunning += "{0:0>2X}".format(int(str1, 2))

                strrunning += '\n'

for k in range(13):
    for a in range(25):
        for q in range(12):
            str_ind += header[q]
            for p in range(12):
                str1 = ''.join(str(e) for e in cube_ind[k, p, q, :])
                str_ind += "{0:0>2X}".format(int(str1, 2))
            str_ind += '\n'

'''Opening text file in write mode and writing the strings in order'''

f = open(r"C:\Users\karth\Documents\textfile.txt", "w")
f.writelines([str_ind, strloading, strcountdown, strxyboing,
              strxzboing, stryzboing, strshuffle, stredgefill, strtaki, str_rain, strshrink,
strwoop,
              strhelix, strsine, strheart, strpac, strpokeball, strrunning, strgangnam,
strsnake, strbounce, strcrab,
              strending, strsocure])

f.close()
```

5. FUTURE PLANS:

6. REFERENCES: