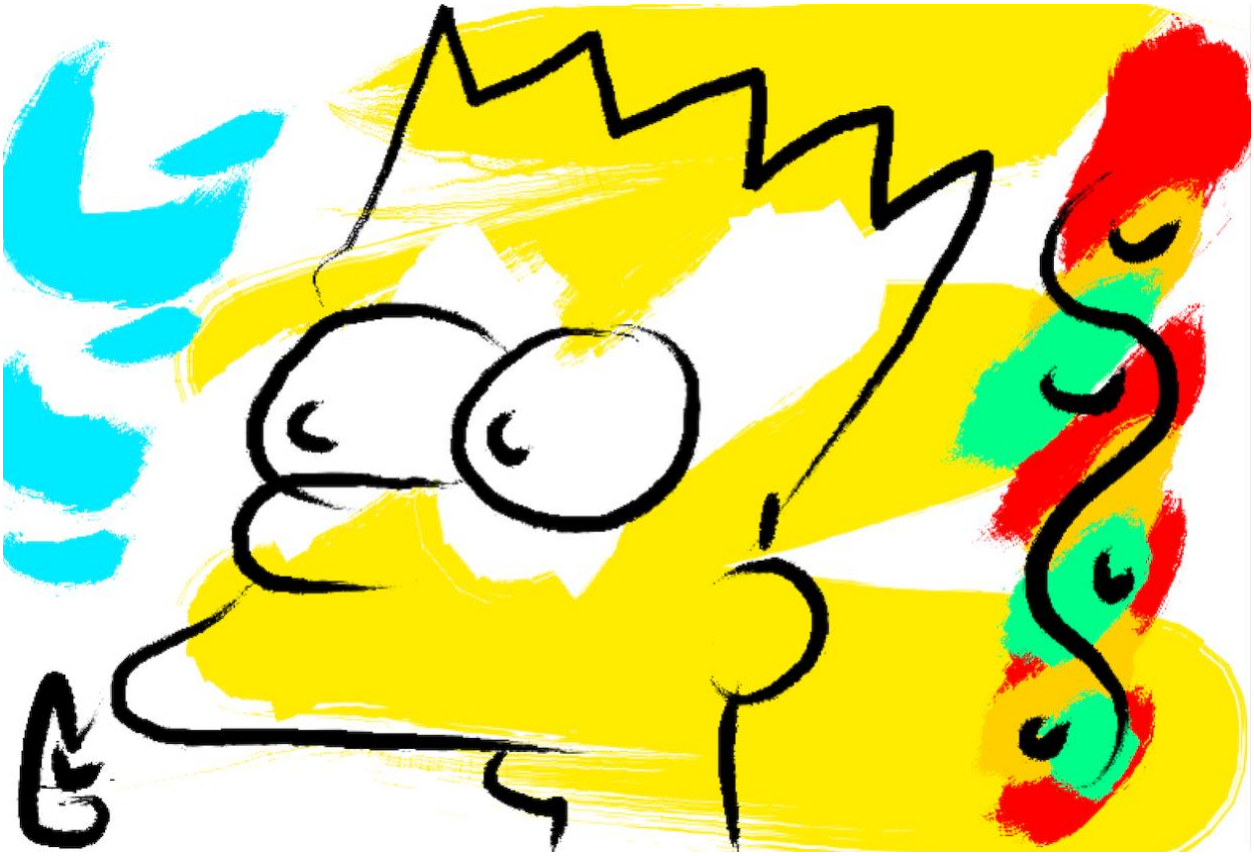
Paint With Music by Victor Garcia
User Manual

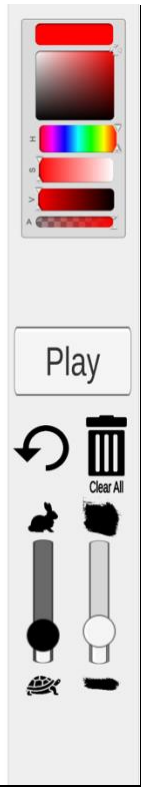


Thank you for using my program and I hope you have fun making beautiful drawings and music. In this manual I will be walking you through how to start using my program, and the core design elements of the program.

Open Program:

To begin you must download the correct version of my program for your system Mac/Windows. The download

links will be hosted on my website
https://beetorcreates.wordpress.com/music-147/
Once you have downloaded the program open it and let your click allow on your computer if prompted. You will them be greeted to a blank canvas and a toolbar as shown below



Painting:
From here you are free to start painting.
You will notice as you paint those sounds will be made and the frequency will grow the higher you paint vertically.

```
float note_on = (pointsV[i].y + 5) * 2.4f + 40;//- 69f;
```

```
        TheSynth.GetComponent<PlayAllSounds>().PlayNote(note_on, c);
```

My code checks the y position of your mouse and maps it to a range from -5 to 5. I then add 5 to that value then multiply 2.4 so I can get a range from 0 to 24 (I want a total of 24 notes to play) then add 40 to that value so our final Midi Notes would be in a range from 40 to 64.
I chose this range as a nice sounding range to hear when you place your lines. I didn't want to add more notes because it would become much harder to place certain notes. Also, the midi notes aren't restricted to whole numbers so it's a much more fluid sound.

However, there is another way to influence the frequency of the sound by changing the color.

```
for(int i = 0; i < AllNotes.Count; i ++){

        note_on += AllNotes[i] +
        map(AllColors[i].r + AllColors[i].g + AllColors[i].b, 0.0f, 3.0f, -12f, 24f);
       note_on = (note_on / (float)(AllNotes.Count ));
```

This loop goes through all the current playing notes and averages their frequency. It also goes through all the colors of the brushstrokes that are currently playing and averages their how much red, green, and blue the color

has and increase/lowers the pitch down 1 octave or up 2 octaves

UI: On the right side you will see a color picker. The color you chose will change the color of your next brush stroke and change the pitch of the note you place up 2 octaves if you draw a white color or down 1 octave if you draw a black color, and everything in-between. You will not be able to hear those influences until you click play so keep that in mind. This is a design decision as I will like the final song to be more of a surprise.

Below a Play button you will also see an Undo button that undo only your last brush stroke. It can't undo more than that so be mindful.

Next to the Undo Button is a trash can Button that will reset the entire scene and clear all your settings and brushstrokes. So only use this when you are ready to start a new drawing.

Below the delete buttons are two sliders. The left slider controls the speed in which the song will be played. The right slider controls the width of your brush.

Play:
As you draw your brush stroke is recorded and all the xy values are stored in a list in the order in which you drew them. When you finished drawing you can click the giant play button which will move a black line across the screen from left to right and trigger every brush stroke that encounters it. Each brush stroke has a script associated with it and has a trigger function that plays the y values that that brush stroke has in the order that you have placed it.

```
void Triggered(){
                if( i < pointsV.Count){

             float note_on = (pointsV[i].y + 5) * 2.4f + 40;

            TheSynth.GetComponent<PlayAllSounds>().PlayNote(note_on, c);
               i++;
               }
               else{
                  isTriggered = false;
                  i = 0;

               }

 }
```

You can also stop playing the song by pressing play again. Try and see what song you can make by putting dots on the screen

Sound:
I had a lot of help in implementing the sound from [pixlpa / Unity-Synth-Experiments Public](). But basically the way sound is produced in Unity is through the OnAudioFilterRead function that lets us takes assign 44100*Channels number with our sound data and it then produces that sound.

```
void OnAudioFilterRead(float[] data, int channels) //Send our Audio Data to Unitys Audio Source
{
    for (var i = 0; i < data.Length; i += 2) // 2 Channels
    {
        float s1 = Run() * volume; // gets our wave and mulitplies it by volume/gain
        data[i] = s1; // It then sets that to channel 1
        data[i + 1] = s1; // then channel 2
    }
}
```

We first have to create our sound data to send through so here we have a basic square and triangle wave mixed together as a base for our synthesized sound.

```
//really basic 2 part square wave and triangle generator
public float StraxRun()
{
    float step2 = harmonic * step;
    px1 += step;
```

```
        px1 = px1 - Mathf.Floor(px1);
        px2 += step2;
        px2 = px2 - Mathf.Floor(px2);
        return 1.0f - ((Mathf.Floor(px1+0.5f)*2f-1f) * (1f - osc2Mix) + (Mathf.Abs(1f-(px2*2f))*2f-1f) * osc2Mix);
//SquareWave and Triangle Wave
    }
```

Step is our just the frequency that we calculated from the midi note we sent it divided by the sampleRate(44100)

```
public void SetNote(float note) //turns our midi not into frequency
    {
        float freq = 440.0f * Mathf.Pow(2.0f, 1.0f * (note - 69f) / 12.0f);
        step = freq / sampleRate;
    }
```

We also send our wave through a filter to shape our sound

```
public float FilterRun(float filterIn)
    {
        float fc = Mathf.Clamp (cutoff+lope.current*envelope+lfo.Run()*cutoff, 0.0005f, 1.0f);
        float res = Mathf.Clamp (resonance, 0f,1f);
        float input = filterIn;
        float f = fc * 1.16f;
        float fb = res * (1.0f - 0.15f * f * f);
        input -= WaveShape(out4) * fb;
        input *= 0.35013f * (f * f) * (f * f);
        out1 = input + 0.3f * in1 + (1 - f) * out1; // Pole 1
        in1 = input;
        out2 = out1 + 0.3f * in2 + (1 - f) * out2;  // Pole 2
        in2 = out1;
        out3 = out2 + 0.3f * in3 + (1 - f) * out3;  // Pole 3
        in3 = out2;
        out4 = out3 + 0.3f * in4 + (1 - f) * out4;  // Pole 4
        in4 = WaveShape(out3);
```

```
    return out4;
}
```

And here is our keyOn and KeyOff functions that turn our sound on and off with our attack and release values.

```
public void KeyOn()
{
    delta = 1.0f / (attack * sampleRate);
}

public void KeyOff()
{
    delta = -1.0f / (release * sampleRate);
}
```

Future Fixes:

My biggest problem that I couldn't fix was making chords sound nice. The way I had my program set up in the beginning was that every single line you drew had its own audio source so when you drew two lines next to each other horizontally they would be playing a chord like you would on a piano. However, there was a problem with the way Unity audio works which makes a chord sound like a garbled mess. So, I had to do the next best thing which was to average all the frequencies that are playing at the same time and play that note. It sounds way better but there isn't much you can do musically besides playing a lot of single notes.

My second problem was changing the type of sound. I had coded a way that changes the values of sound like attack, distortion, volume, and release based on the color. However, it just caused problems and did not sound good, so I scraped the idea. However, I am really proud with what I created and I hope you had some fun playing around with it!