Final Project - Project XYO

XYO is an app that helps singers rappers and songwriters create content even if they have limited technical abilities. XYO is a fun simple tool anyone can use, just play through a quick fun quiz and immediately find yourself with a perfectly tailored backing track ready for your vocals.-

“With the technology these days, any idiot can record on Pro Tools on your laptop. All you have to do is plug a microphone into the input jack and anybody can have their own recording studio.”(1) And it’s getting even simpler than that! Gone are the days when only professional artists could afford to create music, as companies are always searching for new ways of making music production simpler and more accessible.

Companies like Splice and LANDR work hard on providing lots of easy to use high quality royalty free samples, and making sure that the customer experience is intuitive and hassle free. But even though they specialise in samples most big companies like to push a range of tools for producers. Splice Studio lets split users back up their projects to the cloud and facilitates collaboration with other artists, as it integrates most popular DAWs. It saves and stores backup copies of the files so the users can mess with the tracks without having to worry about losing older versions of the project. Splice has also designed it’s own DAW: Studio One Professional. It’s a very simple and powerful DAW designed for intuitive end-to-end music production. Splice also boasts a number of useful plugins including big name VST instruments like Serum, Massive, Nexus and Sylenth. It also provides a lot of effect plugins like VahallaRoom and Pro-Q. It is very important for companies like this to be as comprehensive and encompassing as possible in order to make the production experience more pleasurable for the user.

One of the most recent and exciting tools on the market at the moment is Arcade by Output. The plugin is quickly becoming industry standard as its fresh simple to use UI will give any producer the inspiring beginnings of a track in seconds. The plugin is basically a glorified sampler, with a few extra effects and a beautiful design, but it comes with new fresh samples every day that are just what you need to kick the musical creativity into motion.These samples are grouped into packs and they are essentially designed to spark creativity as quickly as possible. This plugin was in particular one of the main inspirations for XYO. The capabilities of Arcade and other similar VSTs demonstrated how the idea for this project could be applied as a tool for producers.

XYO is a music production tool for those artists with lots of passion but minimal technical ability. It is designed to be as easy to use as possible, so that anybody can make a beat, no matter what their understanding of music production is. Using this app artists can simply swipe on what they think sounds good and create a custom loop in seconds.

Project Development

XYO in essence has always aspired to be a quiz app. The user interface is very simple and intuitive, the questionnaire is fun and gathers enough information to be able to present the customer with a personalised product.

The project was initially going to be coded in Open Frameworks, an open source C++ toolkit for creative coding. At the start it seemed that Open Frameworks would be more than capable of handling this project, however the decision of the project being a music based tool had not yet been made. Initially the idea was for the app to be created in Xcode and to uploaded to the app store directly. This process, although not simple, is quite doable as long as you follow the lengthly apple developers protocols and also join the Apple Developer Program as an organisation or an individual. The first thing you have to do to get your app on the app store is create an iOS distribution provisioning profile and distribution certificate, the easiest way to do this is through Xcode, where automatic signing is available. Then you can create an iTunes Connect record for your app, archive and upload the app using Xcode, configure metadata and further details in iTunes Connect, and finally submit the app for review (which usually takes about 3 days to get approved). This process is relatively complicated but once you have it set up it is easy to uploaded updates or more apps. Uploading you app to the apple store is probably the single fastest way of not only make in money from your app but also ensuring good distribution so it can definitely be worth the effort.

Process.

“Gin parties, gin menus, ginvent calendars and even a Ginstitute hotel: the UK’s renewed passion for all things gin is fast creating a whole new industry.”(1) Over the past few years gin seems to be rapidly rising in popularity in the UK as current distillations and infusions are creating hundreds of flavours for customers to enjoy. But with hundreds of gins to choose from, people may struggle to know where to even begin to find exactly the right gin for their tastes.

The original scope for this project was to create an app to help solve this problem. The app would help people find the right gin for them, by presenting them with a light fun questionnaire that would help determine which gins were best suited for them. The code would have a simple logic tree that took the users down the path to the right gin, based on their answers. The UI would be simple and sleek, and just like many existing apps, it would use the swipe left/right system for yes/no answers (especially popular in dating apps). The questions could be as simple as:

“Do you prefer sweet or dry?” “Fruity or floral?” “Berry or citrus?” “Grapefruit or lime?”
At the end of the quiz the app would suggest a list of gins that would contain the desired flavours.
There could be many extensions to this app, for example it could provide purchase links for the gins or suggest nearby pubs or gin bars that sell those gins. There could also be the possibility for the customer to review the gins after they have been tasted. There are already a few apps on the market that dabble in these areas, one that is especially popular right now is Untapped, an app that specialises in craft beer. The app lets you record what beer you’ve had and where you’ve had it and gives you the option to rate, review and even upload a picture of the drink. You can then share all this information with your friends. All these features could be directly transferable to this project, or at least could be considered as features that could potentially be added in the future to increase the potential of the app.
The Oliver Conquest, a gin bar in Whitechapel showed interest in the project, both for its ability to spread awareness and gin knowledge but also for the potential of hooking up the app directly to their system and allowing the customer to purchase a drink directly from them once they had completed the quiz.

One of the hardest to obtain but most crucial parts of this app would be securing an external database of every type of gin (or at least a substantial amount of them). The Oliver Conquest’s menu contained a list of over 300 gins, which seemed more than enough to create a proof of concept. The list denotes the location of origin, %Vol and descriptions of each of those gins. This information would be most useful in CSV format, as it is relatively easy for OpenFrameworks to communicate with this type of data. A huge advantage of having an external database is that it could be modified at any point and the app should sill run effectively but with more accurate data. In order to access the CVS information in OpenFrameworks one of the simplest things to do is to create a buffer and push back each vector string (line of the file) into that buffer. You can then call any line of the buffer as you would with an array and use the string as you wish.
Unfortunately, due to a corrupted computer all the digital copies of the menu were lost and an alternate way of procuring the data was necessary.
Shortly after the gin bar closed down due to unforeseen circumstances and a lot of the aspiration for this app seemed to be devolving. Although there are quite a few free online databases, no extensive gin lists were found and this project was pulled back to the ideas board.

In searching for a new avenue for this project, many different types of data were looked at, from football statistics, to crime statistics to movie scripts. The later of which drew inspiration for a new angle to the project. After finding a script for the pop culture film Star Wars on kaggle.com, a quiz app was quickly designed using the bits of code that had already been written for the gin app. The app essentially analysed the script (which was purposefully in CSV format) removed any scene setting and narrator parts and stored each line in one array and the character who said that line in the corresponding position in another array. A line could then be extracted at random and the user could try and guess who might have said that line. Essentially the app was going to be a solid quiz game and all the mechanics

seemed to work well. The app was able to use big amounts of data (3 films worth of scripts) however there was a short-come. After testing the app numerous times on a few audiences it was discovered that the CSV scripts were very early raw un-edited scripts and therefore contained scenes and characters that were never even filmed. Many of the lines were slightly different to how they were in the movies and it soon became Clear that the quiz would be too difficult for even the most avid Star Wars fans. The scripts were far too long to manually correct, besides that would have defeated the point of using data.

At this point everything was re-evaluated and it was crucial to come up with a theme for this questionnaire app that was a bit more fail-proof. Given the amount of musical background and expertise obtained over the past 3 years it seed obvious to attempt to do something music related. It was decided to make an app that questioned the user about what kind of music they like, and then Automatically produces them a short beat personalised to their preferences. The app was to be a music tool for songwriters/rappers that can't or don't know how to produce their own beats. There would be a short questionnaire on the type of music they want and even an option to choose between samples. The user should then be able to listen to and download a final beat. It could possibly even be expanded on by having more tools to facilitate the songwriting, like being able to change the bpm or the pitch after the quiz, the possibility to record vocals in-app, and even potential lyric-writing facilities (rhyming databases, auto lyric generator, etc.).

Work on XYO began, using Openframeworks and the add-on OfMaximillian. Maximilian is an audio synthesis and signal processing library written in C++. It's cross-platform compatible with MacOS, Windows, Linux and IOS systems. A few west sketches were written and it was discovered that a few buggy elements might make the project difficult to write and the lack of support and little reference material for Maxim was discouraging. It was decided that perhaps a more musically inclined software would do a better job at constructing the mechanics for the audio.

As OF maxim struggled with time and pitch shifting, Max MSP seemed like the most useful tool for the job. Max MSP is a powerful audio-based software easily capable of sample manipulation. Max, also known as Max/MSP/Jitter, is a visual programming language for music and multimedia developed and maintained by San Francisco-based software company Cycling '74. Max MSPs main downfall(in terms of the scope of this project) is it's UI. The lack of personalisation and adjustability and a somewhat clunky, fixed interface limits Max's prospects as a professional looking software developing tool and limits it more to a music creation tool, as a Max patcher is immediately recognisable and the objects can not move around the screen when the project is locked. Also useful too to use alongside max is jitter, a powerful visual tool that works in Max and can interact with max objects. Similarly useful is Max's ability to communicate with other code, in particular JavaScript. Both jitter and JavaScript have the capabilities of creating simple slick UIs, however it felt much more sensible to use JavaScript as this was an area that: a) was an area which had been more covered during studies and b)

had lots more supporting material and references. This also meant that the bulk of the mechanics for the app could be written in traditional code format and max could just be used to deal with what it does best: storing and playing audio. P5.js is common and simple to use JavaScript library that seemed perfect for the job. It's a JS client-side library for creating graphic and interactive experiences, based on the core principles of Processing. Having had plenty of prior experience using P5 it seemed like an obvious tool for the job of creating not only the UI but also most of the app mechanics.

In order to get P5.js and max communicating properly it was important to write the sketch in instance mode. It could then be sent to a jweb object in max, enabling the JavaScript sketch and the max patch to easily communicate with each other.

A lot of the previously written code could be reused for this iteration of the project, however it needed to be converted from C++ to JavaScript. This was a fairly simple process that consisted mainly of just correcting all the syntax (for example turning float and int into var). With search and replace this job was fairly easy and the UI mechanics that were used to make the gin app were suddenly usable in a jweb max object.

One of the reasons max was chosen for this project was also for its ability to export the project as an application. That means that the finished app can be downloaded and used by anyone. The scope for this project is to make music creation accessible and easy so it is very important that it runs as a standalone app.

In the early stages the UI consisted simply of a box that could be dragged around the screen. If the box was let go near the left edge of the screen it would go shooting off the left side of the screen then reappear in the center, and vice versa with the right side. Even in the finished product the UI remains almost the same as in testing, as a very important aspect of this app is its simplicity.

Unfortunately the time restraints and the limitations of not having a team or a budget meant that this project for now remains somewhat underdeveloped. Although the current app is a good proof of concept there are still many features an aspects to be developed on before it can become a viable commercial project. The main limitation for the moment is the selection of samples, which is currently fixed (but not particularly hard coded). Although the app succeeds in creating a fun interactive way for people with low to no production skill or knowledge to produce and feel like they have created their own beat, essentially for the time being the app only makes one customisable track. The contact time with the consumer would be limited as the app has little to no reusability at the moment. Once you have created your loop, apart form going back and changing a few sample selections there's not really anything you can do.
Given more time the next main step would be to expand on the content available in app, as the current selection of samples could be considered one 'pack' but the idea would be to constantly build on the amount of packs available. This is where a

partnership with or funding from a company like Splice would bring access to much more development for XYO. Access to a large database of samples could be huge for this project. If this app could be connected to Splice's servers and have access to all its samples and content it's usefulness would grow exponentially. The code could pull thousands of samples directly from Splice, therefore generating way more interest and usefulness. The app could have hundreds of packs offering lots of variety to the user. This could also be a good way of generating revenue form the app, as some packs could require purchases. The app would then work extremely well as a VST plugin, much in the same way as Arcade it would be a great way to start a session and get the creative juices flowing. But as a difference to Arcade instead of just playing one sample at a time you can construct mini loops within XYO, hear what sounds good together and construct full complex layered loops within seconds. These loops could then be dragged or played into the DAW and used as the basis for a track. There could even be an option to upload your own loops or save favourites into your own custom pack.
Other features for the app could be developed and put in a stage after the loop is finished. There could be a writing/recording screen that could be an extremely simple DAW, which lets you plan out a structure for your loop and maybe edit simple aspects of the loop (like wether the drums are playing). In order for it to be understandable and accessible by everyone (not just experienced producers) it must remain simple and intuitive and probably only have one track as to not confuse users that are not used to seeing DAWs. Another great feature to add at this stage would be lyric writers tools like RhymersBlock, that would have a space for users to type in lyrics and would be able to suggest usable rhymes to make the lyric-writing process easier. There could even be an auto lyric generator function for if the singer can't think of anything to write. The user would then be able to record their vocals directly through the app and the user would have a few simple options add some basic vocal processing effects (compression, reverb, auto tune, etc.). It is important to not go overboard at this stage and not add anything too non familiar or difficult for an non experienced producer or musician to understand like granular synthesis, polly-rhythms or FFT effects.

At this point this version could also work great as a mobile app. It could be a contender to the likes of Smule and Musical.ly bit instead of just covering other people's songs you write your own. There could be options to upload the finished song to SoundCloud, YouTube or even Spotify or iTunes (for that some sort of partnership with a company like CD Baby would be necessary). The app could be highly profitable via a number of methods: pop up adds could appear as the user is swiping through the app, users could be encouraged to purchase a subscription to the app (in order to obtain an add-free experience or gain access to exclusive sample packs), there could be in-app purchases for more packs or bonus skins or features and there could even be a small stream of revenue for royalties (a percentage could be taken from each song uploaded to Spotify).

In order to be able to fully complete this project however a small amount of funding would be required to cover all the expenses and time put into the further development of this app. It would be important to hire the expertise, talents and

advice of someone knowledgable in the field of design and possibly also a programmer who has worked on similar apps before in order to take this project to a professional level.
The two main ways of obtaining finance are by contacting directly established companies than could be interested in the development of the app or by starting a crowdfunding campaign. With a few fancy adverts and the promise of early access and exclusive features you can get people to invest in a subscription to your app or product before its even fully developed.

Code.

The bulk of the logic for this project is contained within the code written in sketch.js. Max seems to run smoothly and integrate well with the P5 Javascript library via the jweb object.However, there is a namespace conflict between Max API binded to the "window" object (accessible from within jweb) and P5js API binded by default to the same object (in so called "global mode"). The P5 used in this project is written in instance mode to circumvent that conflict.

Common syntax in P5 global mode looks a little like this:

```
var x = 100;
var y = 100;

function setup() {
  createCanvas(200,200);
}

function draw() {
  background(0);
  fill(255);
  ellipse(x,y,50,50);
}
```

However in instance mode the correct syntax for the previous sketch would be:

```
var s = function( sketch ) {

  var x = 100;
  var y = 100;

  sketch.setup = function() {
    sketch.createCanvas(200, 200);
  };

  sketch.draw = function() {
    sketch.background(0);
    sketch.fill(255);
    sketch.rect(x,y,50,50);
  };
};
```

```
var myp5 = new p5(s);
```

Essentially what is happening here is the creation of a new object called `myp5` (or any other variable name). We call it via constructor `new p5()`. The code for `function p5()` can be found in the p5.js source. It's not just making a "blank" sketch, but it's passing in an argument called `s` that will serve as the basis for the code of that sketch. `s` then becomes a function that takes one argument, a `sketch` object and attaches properties to that `sketch`. The properties are the necessary functions and variables for a p5 sketch, functions like `setup()` and `draw()`.

The function detectMax() allows the sketch to know wether it is being run in Max MSP. If all communication with the max patch (for example window.max.outlet) only occurs whilst max is present then the code will also run smoothly in a browser, however a command like window.max.outlet(..) if called whilst running index through a browser will cause the sketch to crash.

```
// Make sure things are running smoothly with max
function detectMax() {
	try {
		window.max.outlet('Hello Max!');
		return true;
	} catch (e) {
		console.log('Max, where are you?');
	}
	return false;
}
```

Here s is declared a function to run P5 in instance mode (as explained earlier).

```
// all the main code is written inside function(p) as we are running P5
in instance mode

var s = function(p) {
```

The outcome of detectMax() is stored as a boolean maxIsDetectedt. This may be used later to safely pass messages to max without damaging the sketches ability to run independently.

```
	var maxIsDetected = detectMax();
```

In this sketch the principal graphic is a box that the user can move around the screen. It is important for this box to move fluidly and intuitively as it is the primary focus of the UI. All the measurements in this sketch are scaled to the window size,

meaning that everything should look fine no matter what resolution or aspect ratio it is run at. Unfortunately the max jweb object is not resizable by sending messages so once the patch is locked it is impossible to resize. However it is still useful not having too much hard coded as not only is it good practice to do so but also large parts of the code will still be easily usable for other versions of the project (for example a mobile app). The way the box is coded it has a current x and y (that are the location of the box at any given time) a move x and y that are activated once the user starts dragging the box, and a return speed for x and y which is essentially the force at which the box gets pulled back to the centre or off screen once the mouse is released. Also stated here are the variables that control the rotation of the box at any given time and the string that contains the words that will be displayed on the box.

```
// Bunch of variables for the main box
var dimentions = innerWidth / 5;
var moveX = 0;
var moveY = 0;
var currentX = innerWidth / 2;
var currentY = innerHeight / 2;
var returnSpeedX = 5;
var returnSpeedY = 10;
var rotate = 0;
var words = 'pad';
```

The yes/no mechanics is probably one of the most complex parts of this sketch. It is necessary to record each answer from each round and communicate that to the buffers in the max patch. It is also possible to skip a round or go back a round, in both cases the answer needs to be null.

```
// Bunch of variables for the yes/no mechanichs
var round = 0;
var answer = 0;
var answers = [];
var yes = 0;
var no = 0;
var skip = 0;
var left = 0;
var right = 0;
var counter;
var words = 'pad';
```

The only other element to the sketch is the back button, that again is proportional to the window size. The only necessary information for this button is its x & y points and its dimensions.

```
// Bunch of variables for the return button
var buttdimX = innerWidth / 20;
var buttdimY = innerHeight / 40;
var buttX = innerWidth / 30 + buttdimX;
var buttY = innerHeight - innerWidth / 30 - buttdimY;
```

In this sketch design is quite important. The following variables are used to chance the tint of the texture on the box itself. There is also a variable for the transparency of text so it can be hidden when not necessary.

```
// Bunch of variables for coloures and textures
var saturation = 0;
var hue = 0;
var trans = 0;
var textcol=0;
var img;
var img2;
```

The canvas is resized to fit the window. The resizing of the window also changes many other variables in this sketch as most measurements are proportional to the innerWidth and innerHeight.

```
// resise window
p.windowResized = function() {
      p.resizeCanvas(innerWidth, innerHeight);
}
```

The preload function is initiated before setup, to ensure that all the correct media can be loaded into the sketch. Here images are being taken from github and prepared to be used later on in the sketch.

```
//load images (from github res)
p.preload = function() {
      img = p.loadImage('https://raw.githubusercontent.com/
SamWmusic/xyo/master/plexus-curve-effect-3d-blue-
background_35761-250%202.jpg');
      img2 = p.loadImage('https://raw.githubusercontent.com/
SamWmusic/xyo/master/smaller.png');
}
```

Setup() is a function that runs only once: at the start of the sketch (after preload). It is the perfect place to attribute initial values and modes. It is also where the canvas is usually created. As this sketch is communicating with max it is necessary to remove the unwanted warning text.

```
// setup function
p.setup = function() {

      p.createCanvas(innerWidth, innerHeight);

      // get rid of unwanted overlay text
      if (maxIsDetected) {
            document.getElementsByTagName('body')[0].style.overflow
= 'hidden';
      }
```

The easiest way to store the answers to each question (whether the user swiped left, right or up on each sample) is to store them in an array. The first step is to create an array and fill it with 0s. Later in the cod it will be told on each round what element it should store.

```
            // fill the array with 0s
            for (var i = 0; i <= 10; i++) {
                  answers[i] = 0;
            }
```

For this sketch on most objects centre mode is used. It is easier to work that way if everything needs to be even and proportional. When rect is in centre mode the centre off the rectangle will be drawn at (x,y) rather than the top left corner. The same goes for image and text. For this sketch HSB (hue, saturation & brightness) colour space is used. This way it is easy to radically change colour (hue) using a single variable rather than having to coordinate 3 different variables (like in RGB). In setup it is also useful to declare a font of you don't plan on changing fonts throughout the sketch.

```
            //set up
      p.textAlign(p.CENTER, p.CENTER);
            p.imageMode(p.CENTER);
            p.colorMode(p.HSB);
            p.rectMode(p.RADIUS);
      p.noStroke();
      p.textFont('Helvetica');

        };
```

The mousePressed function runs once every time the mouse is clicked. The if statements in this function are checking if the mouseX and mouseY are within the confines of the back buttons boundaries at the time of the mouse being pressed. If it is then the quiz resets back to the previous round and clears the previous answer.

```
// click back button
      p.mousePressed = function() {

            if (p.winMouseX < buttX + buttdimX && p.winMouseX > buttX -
buttdimX &&
                  p.winMouseY < buttY + buttdimY && p.winMouseY > buttY -
buttdimY && round > 0) {

                  window.max.outlet('clear', round);
                  round--
                  answers[round] = 0;
                  window.max.outlet('answers', 0);
            }
      }
```

Draw is the main function of most sketches, this is where the ‘meat’ of the code happens. Draw() is called 60 times per second (unless the frame rate is modified) and pretty much everything is called three here. Instead of calling background, here image is called, but the effect is the same, to wipe the canvas clear every frame.

```
      p.draw = function() {

// draw background
            p.tint(50)
            p.image(img, innerWidth / 2, innerHeight / 2, innerWidth,
innerHeight);
```

A variable like counter is often used in code to delay something. In this instance it is being used to reset the variable ‘answer’ for a short amount of time. This is to make sure that the variable turns back to 0 for long enough for the max patch to reset a buffer.

```
            counter++;
            if (counter > 10) {
                  answer = 0;
            }
```

One of the most important communications the sketch has to send to max is the position of the box on the screen. This way if the box is on the left side of the screen then max will increase volume of the buffer playing the sample allocated to the left side of the screen and vice versa. The only problem is that the location of the box is stored in the distance from the top left of the screen in pixels. This means than not only are the outputs somewhat confusing and difficult to be transferd into usable variables in max but they also greatly depend on the size of the screen, which could vary. In order to circumvent this problem the easiest thing to do is to sort out the data here in the sketch, with a few of statements and mapping functions we can clean up the variables and have left and right numbers that go from 0 to 127 (the most common range when dealing with audio).

```
// map position of box to volumes L R
            if (moveX == 0) {
                  left = 0;
                  right = 0;

            if (currentX > innerWidth / 2 && currentX < innerWidth * 0.65)
{
                  left = p.map(currentX, innerWidth / 2, innerWidth *
0.65, 0, 127);
                  right = 0;
```

```
            } else if (currentX < innerWidth / 2 && currentX > innerWidth
* 0.35) {
                  right = p.map(currentX, innerWidth * 0.35, innerWidth /
2, 127, 0);
                  left = 0;
            } else if (currentX > innerWidth * 0.65) {
                  right = 0;
                  left = 127;
            } else if (currentX < innerWidth * 0.35) {
                  right = 127;
                  left = 0;
            }
```

Similar to before, here values are being matched to the boxes position. This time it is the hue and saturation values that are being affected. As the box stays in the middle the satire action of the tint is 0, leafing the texture in grayscale, but as it is dragged towards the sides it gets coloured. Also here the transparency of the font is affected.

```
// map position of box to colour of tint
            if (currentX <= innerWidth * 0.65 &&
                  currentX >= innerWidth * 0.35) {
                  hue = 0;
                  trans = 0;
                  if (currentY < innerHeight * 0.35) {
                        saturation = p.abs(p.map(currentY, innerHeight *
0.35, 0, 0, 125));
                        trans = p.abs(p.map(currentY, innerHeight * 0.35,
0, 0, 1));
                  } else {
                        saturation = 0
                  }
            } else if (currentX > innerWidth * 0.65) {
                  hue = 65;
                  saturation = p.abs(p.map(currentX, innerWidth * 0.65,
innerWidth, 0, 225));
            } else if (currentX < innerWidth * 0.35) {
                  hue = 180;
                  saturation = p.abs(p.map(currentX, innerWidth * 0.35, 0,
0, 225));
            }
      if (p.winMouseX < buttX + buttdimX && p.winMouseX > buttX - buttdimX
&&
                  p.winMouseY < buttY + buttdimY && p.winMouseY > buttY -
buttdimY && round > 0) {
                  textcol=80;
            } else{
        textcol=40;
      }
```

Next is the bulk of the actual drawing functions. Here all the visuals are drawn to the canvas. In order to get the rotation centre in the middle of the box it is necessary to first translate to the centre of the screen. After the box has been drawn it is important not to forget to rotate and translate in the opposite way as to cancel out the previous rotation an translation. That way the reference point 0,0 goes back to the top left of the screen.

```
    //set ammout of rotation
           rotate = p.map(moveX, 0, innerHeight / 2, 0, 0.3);


// draw main box
           p.translate(innerWidth / 2 + moveX, innerHeight / 2 + moveY);
           p.rotate(rotate);
    p.tint(hue, saturation, 225);
           p.image(img2, currentX - innerWidth / 2 - moveX, currentY -
innerHeight / 2 - moveY, dimentions * 2, dimentions * 2);
           p.fill(80);
           p.textSize(innerWidth / 15);
           p.text(words, currentX - innerWidth / 2 - moveX, currentY -
innerHeight / 2 - moveY);
           p.fill(80, trans);
           p.textSize(innerWidth / 25);
           p.text('skip', currentX - innerWidth / 2 - moveX, currentY -
innerHeight / 2 - moveY + dimentions / 2);
           p.rotate(-rotate);
           p.translate(-innerWidth / 2 - moveX, -innerHeight / 2 -
moveY);
```

Here the rectangle for the back button is drawn.

```
// draw back button
           p.fill(10);
    p.rect(buttX, buttY, buttdimX, buttdimY, buttdimX/20);
    p.fill(textcol);
    p.textSize(buttdimX/3);
           p.text('back', buttX, buttY);
```

Most of the important logic that moves the box and registers the answers is answered in this next section. The current x and y is affected by move x and y (where most of the maths is done). If the mouse id pressed then the different between the previous mouse position and the current mouse position is added to move x&y. If the box does not get dragged far enough out then *`moveX -= (moveX / returnSpeedX);`* so essentially moveX becomes smaller and smaller until it returns back to 0 and slips back into position. Essentially the maths for the box sliding off screen (left, right or up) works on the same principal but they respectively require an extra variable (yes, no and skip) as they are accelerating towards a specific point rather than to move(0,0). This code also checks to see if the box has fully

exited the canvas. If so then it sends the answer to the array answers[ ] and resets everything.

```
// move the box && bulk of logic
    currentX = moveX + innerWidth / 2;
    currentY = moveY + innerHeight / 2;
           if (p.mouseIsPressed) {
      //move box
                moveX += p.winMouseX - p.pwinMouseX;
                moveY += p.winMouseY - p.pwinMouseY;
                yes = moveX - innerWidth / 2 - dimentions;
                no = moveX + innerWidth / 2 + dimentions;
                skip = moveY + innerHeight / 2 + dimentions;
           } else {
                if (currentX != innerWidth / 2 && currentX <= innerWidth
* 0.65 &&
                     currentX >= innerWidth * 0.35) {
                     if (currentY < innerHeight * 0.35) {
          //push box up
                          skip -= (skip / returnSpeedX);
                          moveY = skip - innerHeight / 2 - dimentions;
                     } else if (moveY != 0) {
          // drag box back to centre
                          moveY -= (moveY / returnSpeedY);
                          moveX -= (moveX / returnSpeedX);
                     }
                     if (currentY + dimentions < 1 && !p.mouseIsPressed)
{
          // if skip then reset
                          moveX = 0;
                          moveY = 0;
                          currentX = innerWidth / 2;
                          currentY = innerHeight / 2;
                          right = 0;
                          left = 0;
                          for (var i = 0; i <= 10; i++) {
                               if (round == i) {
                                    answers[i] = 0;
                               }
                          }
                          window.max.outlet('answers', answers[round]);
                          counter = 0;
                          round++;
                     }
                } else if (currentX > innerWidth * 0.65) {
        // push to  right
                     yes -= (yes / returnSpeedX);
                     moveX = innerWidth / 2 + dimentions + yes;
                     if (currentX - dimentions > innerWidth - 1 && !
p.mouseIsPressed) {
          // reset and give answer
                          moveX = 0;
                          moveY = 0;
                          currentX = innerWidth / 2;
                          currentY = innerHeight / 2;
```

```
                              right = 0;
                              left = 0;
                              for (var i = 0; i <= 10; i++) {
                                    if (round == i) {
                                          answers[i] = 2;
                                    }
                              }
                              window.max.outlet('answers', answers[round]);
                              p.image(img, innerWidth / 2, innerHeight / 2,
innerWidth, innerHeight);
                              counter = 0;
                              round++;
                        }
                  } else if (currentX < innerWidth * 0.35) {
        // push to  left
                        no -= (no / returnSpeedX);
                        moveX = no - innerWidth / 2 - dimentions;
                        if (currentX + dimentions < 1 && !p.mouseIsPressed)
{
          // reset and give answer
                              moveX = 0;
                              moveY = 0;
                              currentX = innerWidth / 2;
                              currentY = innerHeight / 2;
                              right = 0;
                              left = 0;
                              for (var i = 0; i <= 10; i++) {
                                    if (round == i) {
                                          answers[i] = 1;
                                    }
                              }
                              window.max.outlet('answers', answers[round]);
                              counter = 0;
                              round++;
                        }
                  }

           }
```

This simple code is used to display the correct text for the current round.

```
// display text based on round
            if (round == 0) {
                  words = 'pad';
            } else if (round == 1) {
                  words = 'bass';
            } else if (round == 2) {
                  words = 'snare';
            } else if (round == 3) {
                  words = 'kick';
            } else if (round == 4) {
                  words = 'synth';
```

```
            } else if (round == 5) {
                  words = 'synth';
            } else if (round == 6) {
                  words = 'perc';
            } else if (round == 7) {
                  words = 'pad';
            } else if (round == 8) {
                  words = 'guitar';
            } else if (round == 9) {
                  words = 'effects';
            } else if (round == 10) {
                  words = '🎵 enjoy your track 🎵';
            }
```

By keeping this communication with max inside this if statement it avoids calling these functions if max isn't detected. Unfortunately this sketch will still crash if used without max because of window.max.outlet() functions called earlier that were not in an 'if max is detected' statement. It would however be easy to put them in one if it was necessary for this patch to run independently, but as this sketch makes little sense without sound anyway it seems neglegant.

```
// communicate with max
            if (maxIsDetected) {
                  window.max.outlet('status', p.frameCount,
p.mouseIsPressed, p.mouseX);
                  window.max.outlet('position', right, left, moveX);
                  window.max.outlet('round', round, answer);
            }

      };


};

And finally we can declare the

// let's run the sketch in the "instance mode"
var myp5 = new p5(s);
```

Sebastian Bach Quotes. (n.d.). BrainyQuote.com. Retrieved 2019, from BrainyQuote.com Web site: https://www.brainyquote.com/quotes/sebastian_bach_601611

Sarah Butler 18 jan 2018 https://www.theguardian.com/business/2018/jan/18/from-mothers-ruin-to-modern-tipple-how-uk-rediscovered-gin