Initial Idea:

I have planned to produce a sensory deprivation chamber, which can be installed in a public place, in which a user will have their sense of sight restricted and in which they must navigate and explore surreal sonic environments using only their senses of touch and hearing. The chamber will be manufactured to a standard that will be suitable for members of the public to use and should immerse the user in a sonically stimulating way. The product will be designed using SuperCollider and Arduino software and should contain many inputs and means of user interaction so that the user can sufficiently feel as though they have control over their environment.

Software Design:

When my idea is broken down into its essential characteristics I have arrived at the conclusion that I need to create:

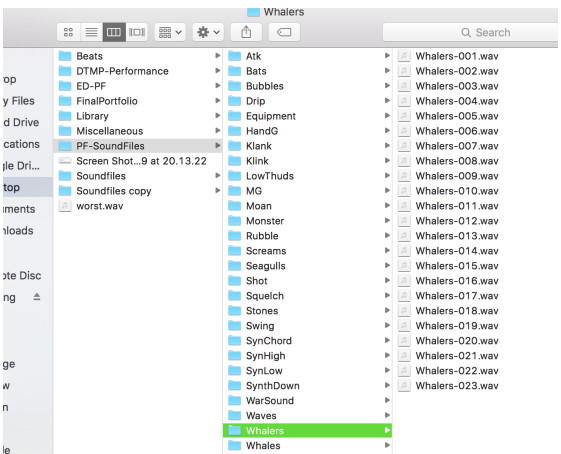
1) a sampler player
2) a selection of samples
3) a backing track sound environment
4) a means of scheduling events
5) a means of translating physical input to audio output

Sample Player - The sample player I have created comes in the form of a Synth Definition in SuperCollider that I have named Samplet1:

```
//Samplet1
SynthDef(\Samplet1, {
   arg ich = 0, obs = 0, gate = 0, bufnum = 0, pan = 0, amp = 1, rate = 1, lpfreq = 20000, hpfreq = 0,spos = 0;
   var source, env;
   env = EnvGen.ar(Env.asr(releaseTime:0.8), gate);
   source = PlayBuf.ar(2, bufnum, rate*BufRateScale.ir(bufnum),startPos:spos, doneAction:2 );
   source = Balance2.ar(source[0],source[1], pan, amp);
   source = LPF.ar(source,lpfreq);
   source = HPF.ar(source,hpfreq);
   Out.ar(obs,source*env);
} ).add;
```

This SynthDef will allow me to create a synth that will play out a sample using the PlayBuf object. By setting the argument for bufnum in the PlayBuf object, I can assign a specific buffer number to a synth created using this SynthDef which means that I can create a new synth for every sample that I want played and I can also set things like the panning, amplitude and even control a high or low pass filter on that specific synth. The use of the envelope env is to remove any clicks or pops that may occur when the synth has finished playing. The doneAction:2 tells the synth to free itself after it completes one cycle of the buffer. So I now have a sample player.

Sample selection - I have created many folders full of different samples to be used based on which preset is being played. These are all sound files that I have brought into Reaper to edit and



process for the exact purpose of being used in particular preset environments. In SuperCollider loading a sample usually means that you must copy and paste the entire path name to the file into your code but using the Dictionary object I am able to load all of these folders into supercollider at once and have much easier access to the files by simply referencing their symbol and then choosing a number starting from 0 which represents the order of the files in the folder e.g. :

```
z = Dictionary.new;
PathName("/Users/edwardgoodwin/Desktop/PF-SoundFiles/").entries.do{
    arg subfolder;
    z.add(
        subfolder.folderName.asSymbol ->
        Array.fill(
            subfolder.entries.size,
            {
                arg i;
                Buffer.read(s,subfolder.entries[i].fullPath);
            }
        )
    );
};
```

z is the name of the dictionary and files inside it can be accessed using this syntax:

```
z[\Shot][0].play;
```

This plays the first file contained in the folder named 'Shot' in the z dictionary.
Using this means of accessing my samples allowed me to keep all my folders in one place and not have rewrite a new pathname for every different folder of samples that I created. If the location of all of the sound files changes, the PathName to the folder only has to be updated rather than the pathnames of every single file. I picked up this method of loading buffers from tutorial videos by Eli Fieldsteel on YouTube (**https://www.youtube.com/watch?v=oR4VZy2LJ60**).

Sound Environment - For my sound environments I composed a few presets using the Digital Audio Workstation Reaper. I used a combination of self recorded sounds and some samples I found using **http://freesound.org/**. When creating these environments or skeleton tracks as I refer to them, I was simultaneously testing the sounds that I felt would work well as samples to be activated during them and adding these samples into my folder to be used in the dictionary.

When creating these skeleton tracks my aim was to create a series of events or storylines in which the pieces would follow a certain order and therefore call for changes in the environment and input from the user. The events in the pieces are supposed to transition the user from one area to another or from a particular situation to another. When these situations/areas change in the piece, the samples available to user must also change. In this way I have taken inspiration from video games. There is an overall fixed storyline to the pieces but what each user experiences whilst moving through the storyline is unique to them depending on when they choose to interact with the system or which button to press.

I tried to increase this sense of variation and randomness in each different users unique experience by having many possible variations of the samples be available at any time. This was achieve by using the ".choose" object in SuperCollider as seen below:

```
z[\Whales][[0,2,4,6].choose].bufnum
```

This means that any time this particular piece of code is evaluated, a random sample is chosen out of either the first, third, fifth or seventh sample in the "Whales" folder. Therefore if the user hits the same button twice they may not necessarily hear the same sample again.

Event Scheduling - In order to create a varying, immersive environment that the user can be continually stimulated by I needed to create a means of varying the samples available to the user over the course of the piece. I needed to create a time schedule that would change the sample banks of sounds that related to the changing skeleton tracks. To do this I had to plan the events of every section of the pieces and also take down the timings of important events or times when things changed. When I had done this for all the pieces, I created a schedule:

```
//scheduling
SystemClock.sched(0, {( ~t = 1).postln; //Whale sounds ocean
   (~u = 1);nil });
SystemClock.sched(56, {( ~t = 2).postln; nil }); //bubbles
SystemClock.sched(66, {( ~t = 3).postln; //inside Whale - equipment squelch - reverb
   (~u = 2); nil });
SystemClock.sched(86, {( ~t = 4).postln; nil }); // moans, squelch, drip
SystemClock.sched(152, {( ~t = 5).postln; nil }); // ^ and stomach whalers, seagulls, waves
SystemClock.sched(168, {( ~t = 6).postln; nil }); //cannons
SystemClock.sched(192, {( ~t = 7).postln; nil }); //seagulls
SystemClock.sched(200, {( ~t = 8).postln; nil });// cheers, all the rest
SystemClock.sched(205, {( ~t = 0).postln;
   (~u = 0); nil });// dead
```

This schedule starts a system clock at the start of every piece and sets a global variable ~t to a certain value. ~t represents the different sections of a piece. The system clock schedules the events within it at a time given at the start in seconds. e.g. when the code is first evaluated it sets ~t to equal one straight away (at 0 seconds in).

Using this global variable ~t I created a number of nested If statements to decide which samples are assigned to each button at a specific time:

```
//Button0
if( header == \b0, {

   if(val == 1 && ~t == 1,{
      s.sendMsg(\s_new,\Samplet1,4000,0,g1.nodeID, \gate, 1,\bufnum, z[\Whales][[1,2,3,4].choose].bufnum, \amp,
rrand(0.3,0.6), \pan, rrand(-1,-0.8), \rate, rrand(0.5,0.9));
      //end of synth1
   },{
      if(val==1 && ~t==2,{
         s.sendMsg(\s_new,\Samplet1,4000,0,g1.nodeID, \gate, 1,\bufnum, z[\Bubbles][[0,1,2].choose].bufnum,
\amp, rrand(0.3,1), \pan, rrand(-1,-0.8), \rate, rrand(0.4,1.4));
         //end of synth2
      },{
         if(val==1 && ~t==3,{
            s.sendMsg(\s_new,\Samplet1,4000,0,g1.nodeID, \gate, 1,\bufnum, z[\Equipment]
[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14].choose].bufnum, \amp, rrand(0.3,0.6), \pan, rrand(-1,-0.8), \rate, rrand(0.6,1.4));
            //end of synth3
         },{
            if(val==1 && ~t==4,{
               s.sendMsg(\s_new,\Samplet1,4000,0,g1.nodeID, \gate, 1,\bufnum, z[\Moan]
[[0,1,2,3,4,5].choose].bufnum, \amp, rrand(0.6,1), \pan, rrand(-1,-0.8), \rate, rrand(0.4,1.1));
               //end of synth4
            },{
```

(if val == 1 then then button is being pressed, so depending on what ~t is equal to at that time will decide which sample bank is loaded into to synth Samplet1)

As seen above, when the synth is set, so too are its parameters. Therefore I use this opportunity to set the original panning, rate and amplitude for each sample in these statements too. This is Button0 which is to be located at the back of the left wall, therefore the panning has been set to originate between -1 and -0.8 this is a means of making the user recognise the samples that

they are controlling, as the button location will correspond to where they are hearing the sample originate.

Also above is a reference to the global variable ~u. This variable is used to control the parameters of a reverb SynthDef which takes incoming signal from the microphone that will be attached to to headphones being used in the enclosure in this particular preset. This can be seen in the PF-Whale.scd document.

Physical Input - Audio Output - My means of communicating between the physical components of my enclosure and the code in SuperCollider will be via an Arduino Nano chip and the Arduino Software IDE.

Included in this folder will be a Fritzing schematic showing the physical circuitry I am using in order to connect physical inputs in the form of push buttons and force sensing resistors to an Arduino Nano chip. Also included in this folder will be an Arduino file containing the code I will be using in my project. This code will show how 4 force sensing resistors and 10 push buttons are being recognised by the Arduino chip and how their values are being evaluated and passed into SuperCollider via the Serial Port.

```
~ino = ArduinoSMS("/dev/tty.wchusbserial1420",9600); //Aduino communication to SC through Serial Port

  //Arduino Action//
  ~ino.action = {|msg|
     var msgsplit = split(msg,$:);
     var header = msgsplit[0].asSymbol;
     var val = msgsplit[1].asFloat;
```

The Arduino action object reads the incoming messages from supercollider via the Serial Port and splits them into headers and values. The headers are then used to recognise which button or force sensing resistor is being read and its values can be used to set parameters of my choosing inside the SuperCollider code.