

Studio 1: Folio 2

First name: Qiannyu

Last name: Ma

Masters of Animation, Games and Interactivity (MAGI),
School of Design,
RMIT University,
Melbourne, Australia

s3735632@rmit.student.edu.au

Abstract

I am interested in motion sensing interaction because it releases people from input hardware. Without the limitation of this equipment, people could interact with games or program by their hand gesture and body motion directly. I like the way which reduces the limitation between audience and interactive stuff and increase the immersion. So I would like to take this chance to learn and experiment motion sensing interaction and connect this interactive method with my artwork. In the end, I will create an interactive scene or image

I used Processing program software which is powerful for graphics interaction to try simple interaction with the mouse. Then I will connect Kinect sensor with Processing to achieve motion sensing interaction.

Authors Keywords: motion sensing, interaction, scene/image

Introduction

I am going to create a motion sensing interactive scene which is same as folio 1. Audiences could interact with my scene using their body directly without any input equipment.

Motion sensing interaction is not a new technology in the era when technology is developed and iterated by leaps and bounds. However, it still takes an important place in the interaction field because it reduces the limitation between people and interactive stuff and increases acceptability.

Actually, connect motion sensing interaction with VR is the most popular way now which make audiences immerse in scenes totally and reality. However, I am not going to use VR this time because people need to wear or hold VR equipment when they interact with VR scenes. I hope that free people from equipment and back to the natural and original interaction way—using hand and body.

Several years ago, I saw a motion sensing football shooting game in a shopping mall.

No matter the age and gender, all customers would like to try the simple game which does not require complicated operation and devices, the only thing they need to do is kicking their legs. So 2 or 3 years old children can play, 70 or 80 years old people can play, the women wear high-heeled shoes can play. I think this example exactly show that motion sensing interaction reduces the limitation between people and interactive stuff and increase acceptability.

In folio 2 I mainly focused on 2 part: code and design. In coding part, I learned how to connect Kinect sensor with Processing at first and then coded some motion sensing interaction functions in Processing with Kinect sensor. In design part, I pay much attention to thinking about what kind of scene I should have, and what kind of interaction suit for the scene.



Related Work (1/3)

Through research, I find that motion sensing interaction could be applied in many fields for example education, business, and entertainment. I want to make something fun and interesting so I think my work belongs to entertainment field.

Here are some great examples of using motion sensing interaction in the entertainment field which inspire me a lot.



Funky Forest Interactive Ecosystem

Funky Forest is a wild and interactive ecosystem where children manage the resources to influence the environment around them. Children could create new plants by certain body pose.



Quantum Space

When people walk to screen, particles would form their figures which could change as their motion. Designer wants to offer audiences an experience which feels like communicate with the universe and immerse in "digital meditation".

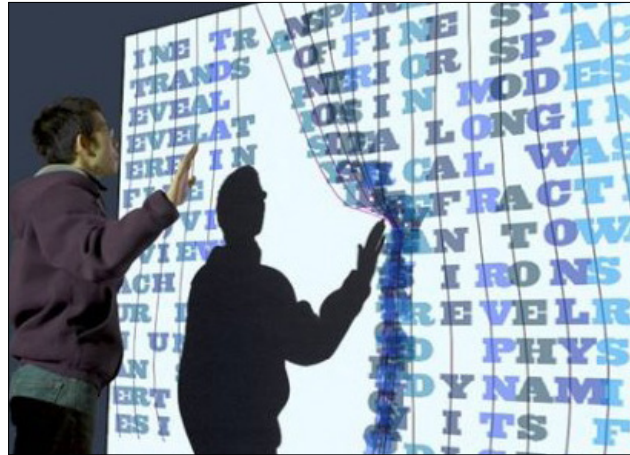
Related Work (2/3)

Motion sensing interaction & Entertainment



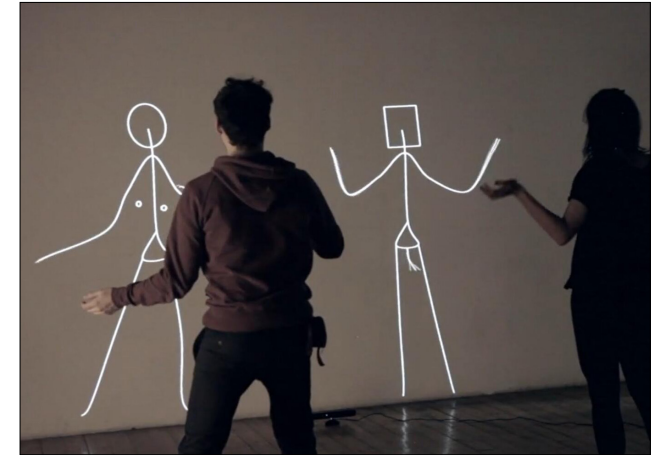
The world of letters

Letters fall down incessantly and people could block letters' way by their body. In this way, create a fresh scene and interaction between people and letters which cannot achieve in the real world.



Text curtain

The curtain is formed by text lines. People could move the text lines by hand. The text curtain is very creative which could audience freshness.



EGO

The interactive installation EGO re-stages and reverses the process of alienation by enhancing and deforming the mirror image by the movements of the users.

Inspiration from related work

The examples of Motion sensing interaction & Entertainment inspire me a lot. Most Motion sensing interaction & Entertainment examples show an incredible world where people could see and do what they cannot do in the real world. Use natural and original interaction way (motion sensing interaction) and the incredible interactive content to bring the audience a playful and immerse experience is what I want to achieve in my project.

Analyze of related work

Comparing these example, I find some helpful point:

1. Audience's figure could be or not be a part of the scene on screen.

For example, in Funky Forest, children's figure would not show on screen; in the world of letters, audience figure is shown on screen.

2. Audiences could interact with objects in scene or all scene.

For example in the world of letters, audiences just interact with the letters, the background is non-interactive. In Quantum Space, the whole scene is interactive, every particle could transform as audiences' motion.

3. Interaction result could be transformation, displacement and creating new objects.

Transformation: Text curtain
Displacement: The world of letters
Creating new objects: Funky Forest
Creating new object& transformation& displacement: Quantum Space, Fitting room, Be the bird.

4. Whole body movement, hand movement, specific body pose, and specific gesture are mainly used to interact.

How it works

There are a number of ways to achieve motion sensing interaction, here are some methods applied mainly in industry:

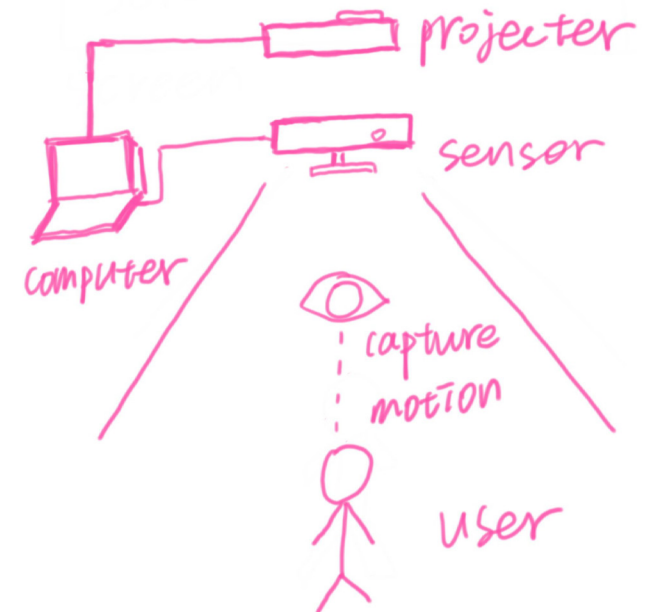
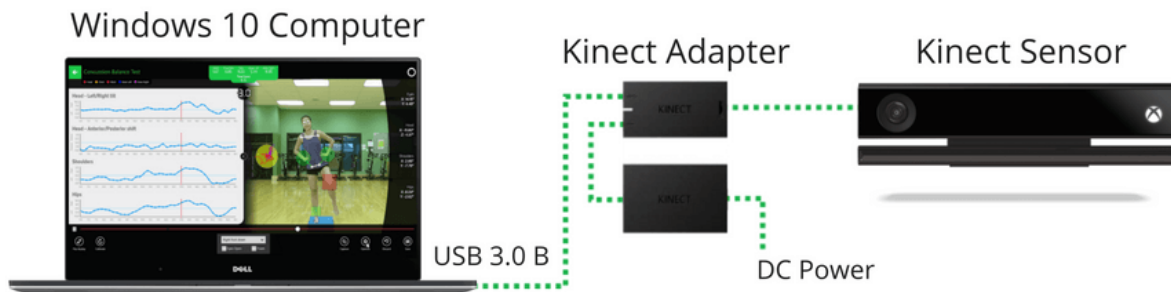
Kinect sensor + Processing

Kinect sensor + Unity

Leap motion + Processing

Leap motion + Unity

I am going to use Kinect sensor + Processing as the main way to achieve motion sensing interaction. There are 3 reasons: 1. Kinect could capture whole body motion rather than just hand part like Leap motion; 2. As I want to create a simple interaction sense rather than a game, using processing is a more simple and effective to achieve the project; 3. There are many learning resources online about interactive coding with processing and Kinect, which is easier for me to start with these tutorials.



Reflect Folio 1 and plan Folio 2

In folio 1, I learned some basic functions of processing, achieved simple mouse interaction in processing and generally know how Kinect sensor works with Processing, which is an important foundation of achieving motion sensing interaction with processing and Kinect sensor.

In folio 2, I need to:

Step 1. Test Kinect.

Step 2. Motion sensing interactions.

Step 3. Refine or redesign scene.

Step 4. Add interactions in my scene. Think about how people interact with my scene and what kind of interactions suit my scene, then achieve them.

Step 5. Create challenges. For example put some objects at a higher place which attract people to jump to catch them, in order to make the scene more playful.

Step 6. Test and Fix bugs (whole process task)

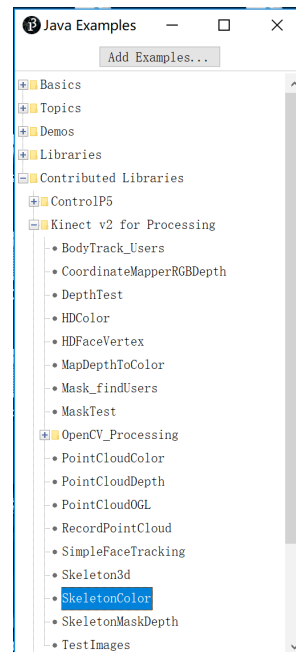
I will explain each step detailedly in the following.

Step 1. Test Kinect

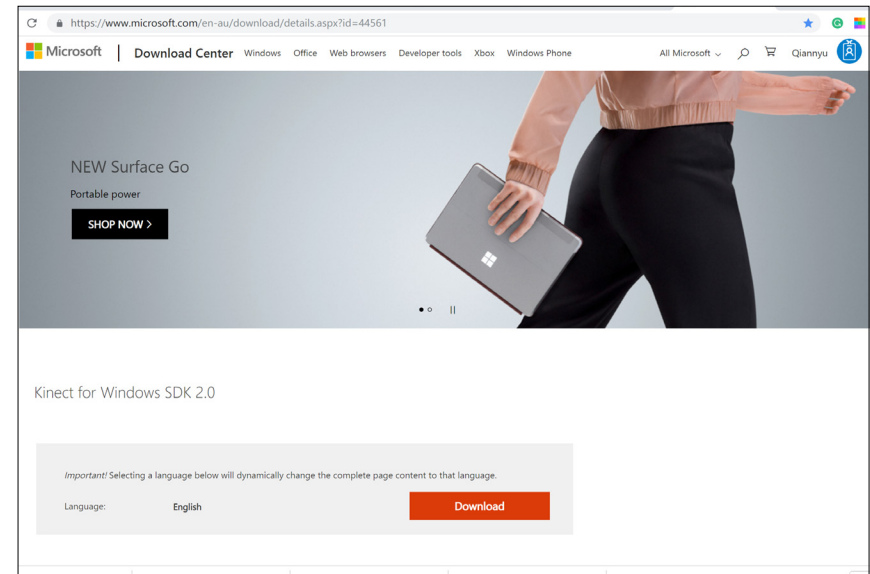
The first step is connecting the Kinect sensor with my computer and test it with processing. There are some things need to prepare in order to make Kinect work with processing.

Windows 8 and Windows Embedded Standard 8
Processing 3+
Microsoft Kinect SDK 2.0
Kinect V2 library

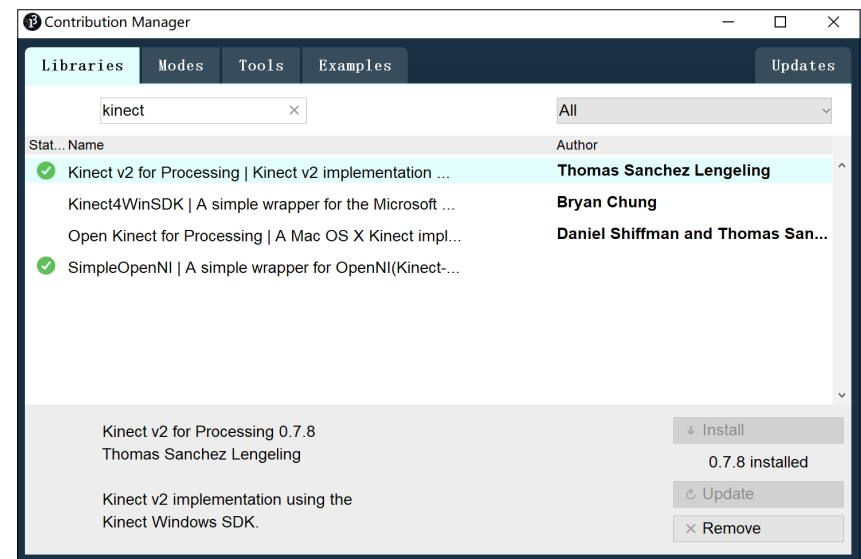
luckily, my Kinect works well with my computer and processing. The good news is that Kinect V2 library provides some example codes help me get into it. So I run some examples and want to see what would happen.



Examples list



Microsoft Kinect SDK 2.0 download page



Kinect V2 library install page (inside Processing)

Step 1. Test Kinect

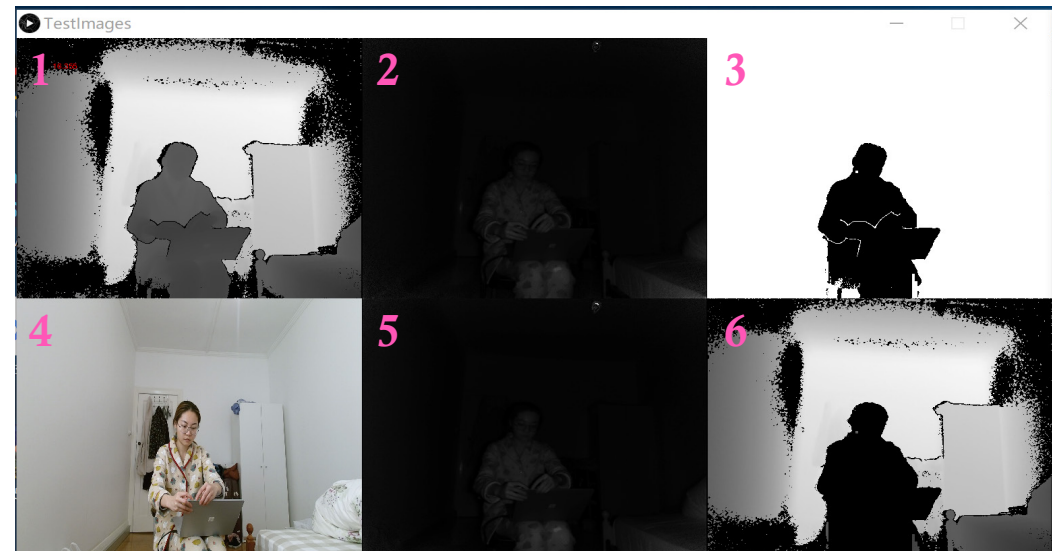
TestImages | Processing 3.4

文件 编辑 速写本 调试 工具 帮助



TestImages

```
5 KinectPV2, Kinect for Windows v2 library for processing
6
7 Test all Images
8 */
9
10 import KinectPV2.*;
11
12 KinectPV2 kinect;
13
14 public void setup() {
15     size(1536, 848);
16     kinect = new KinectPV2(this);
17     kinect.enableColorImg(true);
18     kinect.enableDepthImg(true);
19     kinect.enableInfraredImg(true);
20     kinect.enableInfraredLongExposureImg(true);
21     kinect.enableBodyTrackImg(true);
22     kinect.enableDepthMaskImg(true);
23
24     frameRate(60);
25
26     kinect.init();
27 }
28
29 public void draw() {
30     background(0);
31
32     image(kinect.getColorImage(), 0, 424, 512, 424);
33     image(kinect.getDepthImage(), 0, 0, 512, 424);
34
35     image(kinect.getInfraredImage(), 512, 0, 512, 424);
36     image(kinect.getInfraredLongExposureImage(), 512, 424, 512, 424);
37
38     image(kinect.getBodyTrackImage(), 512*2, 0, 512, 424);
39     image(kinect.getDepthMaskImage(), 512*2, 424, 512, 424);
40
41     fill(255, 0, 0);
42     text(frameRate, 50, 50);
43 }
```



1.1 Different images tsest

This is the result of the Image test example in Kinect PV2 library. This example shows 6 image wich Kinect sensor provides for developers.

- 1.Depth image
- 2.Infrared image
- 3.Body track image
- 4.Color image
- 5.Infrared long exposure image
- 6.Depth mask image

Body track image helps me a lot in following steps because it only recognizes people figures. I could create user silhouette and merge it into my scene.

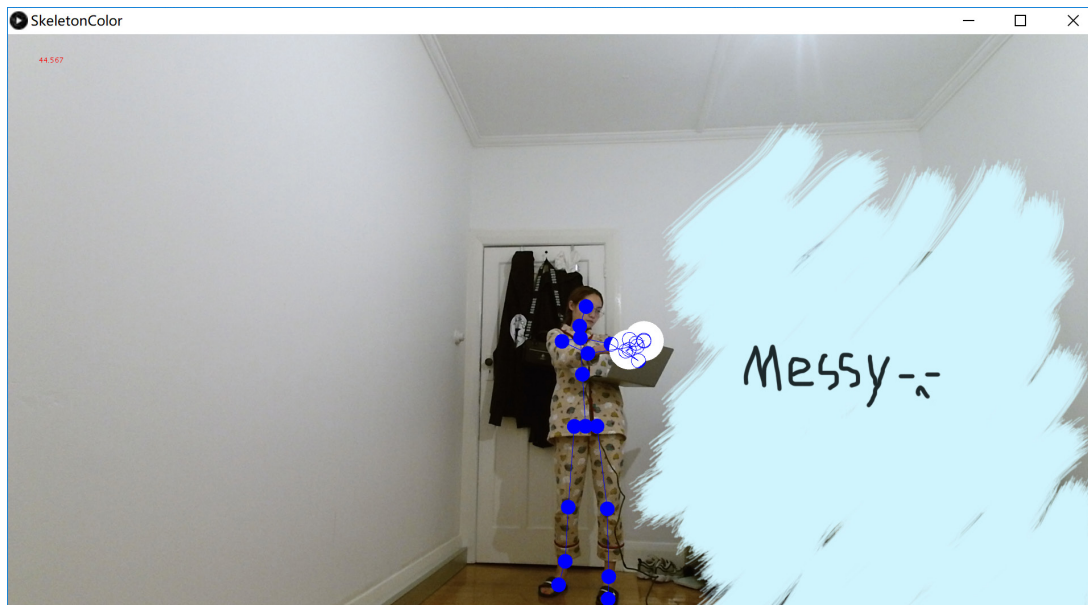
Step 1. Test Kinect

```
11 import KinectPV2.KJoint;
12 import KinectPV2.*;
13 KinectPV2 kinect;
14
15 void setup() {
16   size(1920, 1080, P3D);
17
18   kinect = new KinectPV2(this);
19
20   kinect.enableSkeletonColorMap(true);
21   kinect.enableColorImg(true);
22
23   kinect.init();
24
25   void draw() {
26     background(0);
27
28     image(kinect.getColorImage(), 0, 0, width, height);
29
30     ArrayList<KSkeleton> skeletonArray = kinect.getSkeletonColorMap();
31
32     //Individual JOINTS
33     for (int i = 0; i < skeletonArray.size(); i++) {
34       KSkeleton skeleton = (KSkeleton) skeletonArray.get(i);
35       if (skeleton.isTracked()) {
36         KJoint[] joints = skeleton.getJoints();
37
38         color col = skeleton.getIndexColor();
39         fill(col);
40         stroke(col);
41         drawBody(joints);
42
43         //draw different color for each hand state
44         drawHandState(joints[KinectPV2.JointType_HandRight]);
45         drawHandState(joints[KinectPV2.JointType_HandLeft]);
46
47       }
48     }
49
50     fill(255, 0, 0);
51     text(frameRate, 50, 50);
52 }
```

```
96 //draw joint
97 void drawJoint(KJoint[] joints, int jointType) {
98   pushMatrix();
99   translate(joints[jointType].getX(), joints[jointType].getY(), joints[jointType].getZ());
100   ellipse(0, 0, 25, 25);
101   popMatrix();
102 }
103
104 //draw bone
105 void drawBone(KJoint[] joints, int jointType1, int jointType2) {
106   pushMatrix();
107   translate(joints[jointType1].getX(), joints[jointType1].getY(), joints[jointType1].getZ());
108   ellipse(0, 0, 25, 25);
109   popMatrix();
110   line(joints[jointType1].getX(), joints[jointType1].getY(), joints[jointType1].getZ(), joints[jointType2].getX(), joints[jointType2].getY(), joints[jointType2].getZ());
111 }
112
113 //draw hand state
114 void drawHandState(KJoint joint) {
115   noStroke();
116   handState(joint.getState());
117   pushMatrix();
118   translate(joint.getX(), joint.getY(), joint.getZ());
119   ellipse(0, 0, 70, 70);
120   popMatrix();
121 }
122
123 //draw hand state
124 void handState(int handState) {
125   switch(handState) {
126     case KinectPV2.HandState_Open:
127       fill(0, 255, 0);
128       break;
129     case KinectPV2.HandState_Closed:
130       fill(255, 0, 0);
131       break;
132     case KinectPV2.HandState_Lasso:
133       fill(0, 0, 255);
134       break;
135     case KinectPV2.HandState_NotTracked:
136       fill(255, 255, 255);
137       break;
138   }
139 }
```

```
53
54 //DRAW BODY
55 void drawBody(KJoint[] joints) {
56   drawBone(joints, KinectPV2.JointType_Head, KinectPV2.JointType_Neck);
57   drawBone(joints, KinectPV2.JointType_Neck, KinectPV2.JointType_SpineShoulder);
58   drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.JointType_SpineMid);
59   drawBone(joints, KinectPV2.JointType_SpineMid, KinectPV2.JointType_SpineBase);
60   drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.JointType_ShoulderRight);
61   drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.JointType_ShoulderLeft);
62   drawBone(joints, KinectPV2.JointType_SpineBase, KinectPV2.JointType_HipRight);
63   drawBone(joints, KinectPV2.JointType_SpineBase, KinectPV2.JointType_HipLeft);
64
65   // Right Arm
66   drawBone(joints, KinectPV2.JointType_ShoulderRight, KinectPV2.JointType_ElbowRight);
67   drawBone(joints, KinectPV2.JointType_ElbowRight, KinectPV2.JointType_WristRight);
68   drawBone(joints, KinectPV2.JointType_WristRight, KinectPV2.JointType_HandRight);
69   drawBone(joints, KinectPV2.JointType_HandRight, KinectPV2.JointType_HandTipRight);
70   drawBone(joints, KinectPV2.JointType_WristLeft, KinectPV2.JointType_ThumbRight);
71
72   // Left Arm
73   drawBone(joints, KinectPV2.JointType_ShoulderLeft, KinectPV2.JointType_ElbowLeft);
74   drawBone(joints, KinectPV2.JointType_ElbowLeft, KinectPV2.JointType_WristLeft);
75   drawBone(joints, KinectPV2.JointType_WristLeft, KinectPV2.JointType_HandLeft);
76   drawBone(joints, KinectPV2.JointType_HandLeft, KinectPV2.JointType_HandTipLeft);
77   drawBone(joints, KinectPV2.JointType_WristRight, KinectPV2.JointType_ThumbLeft);
78
79   // Right Leg
80   drawBone(joints, KinectPV2.JointType_HipRight, KinectPV2.JointType_KneeRight);
81   drawBone(joints, KinectPV2.JointType_KneeRight, KinectPV2.JointType_AnkleRight);
82   drawBone(joints, KinectPV2.JointType_AnkleRight, KinectPV2.JointType_FootRight);
83
84   // Left Leg
85   drawBone(joints, KinectPV2.JointType_HipLeft, KinectPV2.JointType_KneeLeft);
86   drawBone(joints, KinectPV2.JointType_KneeLeft, KinectPV2.JointType_AnkleLeft);
87   drawBone(joints, KinectPV2.JointType_AnkleLeft, KinectPV2.JointType_FootLeft);
88
89   drawJoint(joints, KinectPV2.JointType_HandTipLeft);
90   drawJoint(joints, KinectPV2.JointType_HandTipRight);
91   drawJoint(joints, KinectPV2.JointType_FootLeft);
92   drawJoint(joints, KinectPV2.JointType_FootRight);
93   drawJoint(joints, KinectPV2.JointType_ThumbLeft);
94   drawJoint(joints, KinectPV2.JointType_ThumbRight);
95 }
```

Completed code of skeleton recognise function



1.2 Skeleton recognise

This is another example in Kinect PV2 library. This example provides the skeleton recognition function.

It is very powerful because it provides the coordinates of joints. I could create interaction between objects and joints for example catch objects when I get these joints' coordinates.

Step 2. Motion sensing interactions

```
new3_2x2_6_original_Animation_bg | Processing 3.4
文件 编辑 速写本 调试 工具 帮助

new3_2x2_6_original_Animation_bg carrot drop eye falling

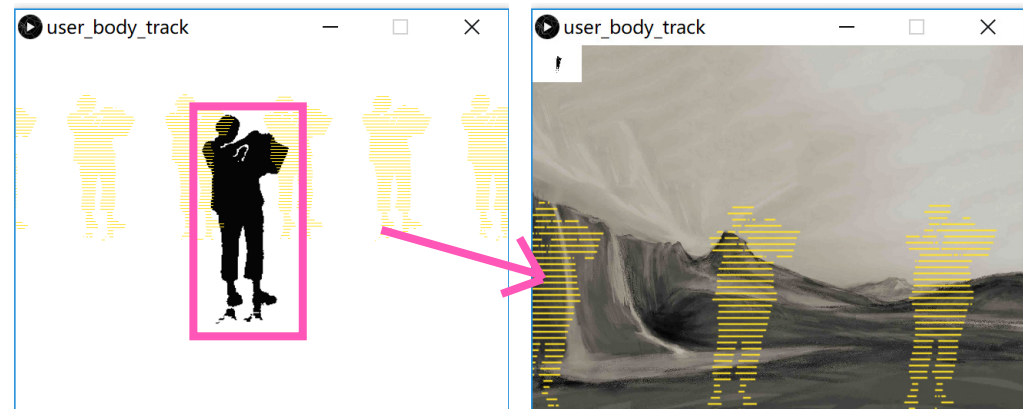
421 //copy human image
422 void copyImage(){
423   PImage n=kinect.getBodyTrackImage();
424   for (int i=0; i<512*424; i++)
425   {
426     if(n.pixels[i]!=color(255))
427     {
428       imgHuman.pixels[i]=color(255,222,33,500);
429     }
430   }
431   else
432   {
433     imgHuman.pixels[i]=color(0,0,0,0);
434   }
435 }
436 imgHuman.updatePixels();
437 }
438
439 //draw the body
440 void drawBody(KJoint[] joints) {
441
442   //Single joints
443   drawJoint(joints, KinectPV2.JointType_HandLeft);
444   drawJoint(joints, KinectPV2.JointType_HandRight);
445   drawJoint(joints, KinectPV2.JointType_Head);
446   drawJoint(joints, KinectPV2.JointType_FootLeft);
447   drawJoint(joints, KinectPV2.JointType_FootRight);
448 }
449
450 //draw a single joint
451 void drawJoint(KJoint[] joints, int jointType) {
452   pushMatrix();
453   translate(joints[jointType].getX()*C+80, joints[jointType].getY()*K, joints[jointType].getZ()*K);
454   fill(255);
455   noStroke();
456   ellipse(0, 0, 20, 20);
457   popMatrix();
458 }
459 }
```

After testing Kinect and making sure it works well, I started to try basic motion sensing interactions.

2.1 User Silhouette

The first thing I did was creating user silhouette. I want that users could see themselves in my scene, which feels like they are in my scene. In this way, the immersion could be created.

Although Kinect provides Body track image, it could not be used directly in my scene because it is non-transparent and cover my scene completely. So I wrote a new function called copyImage which could only copy the black pixels from Body track image to my scene. In this way, the user silhouette shows on my scene.



Copy black pixels from body track image, give change black pixels to yellow and past in my scene

Step 2. Motion sensing interactions



2.2 Catch falling bubbles by body

I achieved endless falling bubbles in folio 1, so I tried to make the bubbles interactive first. If the bubbles fall on the user silhouette they would stop falling, go up and change color, which looks like users could catch these bubbles by their body.

My user silhouette is in yellow which means all pixels of the user silhouette are yellow. So the key to achieving this function is that judge if the bubbles encounter the yellow pixels.

```
Fruit_user_body_track2
3 PImage backgroundimg;
4 PImage imgHuman;
5 Bubble bubble[]=new Bubble[20];
6
7 /*-----*/
8
9 class Bubble{
10 float X,Y,VX,VY, split=0,angle=0;
11 boolean dead=false;
12 Bubble(float X, float y, float vx, float vy){
13 X=X;Y=y; VX=vx; VY=vy; dead=false;
14 }
15
16 void draw(){
17 if(dead){
18 fill(255,222,33);
19 arc(X+split,Y,50,50,angle,angle+PI,CHORD);
20 arc(X-split,Y,50,50,angle+PI,angle+PI*2,CHORD);
21 }
22 else fill(255);
23
24 ellipse(X,Y,50,50);
25 }
26
27 void update(){
28 X+=VX;
29 Y+=VY;
30 VY+=1.98;
31 if(X<-100||X>1300||Y<-100||Y>860){
32 X=random(1280); Y=random(100); VX=random(30)-15; VY=-random(30); dead=false;
33 }
34 if(dist(X,Y,mouseX,mouseY)<50){
35 dead=true;
36 angle=atan2(mouseY-pmouseY, mouseX-pmouseX);
37 }
38 if(X>0 && X<1280 && Y>0 && Y<760 && Kinect.getBodyTrackImage().pixels[int(X)+int(Y)*240]!=color(255)){
39 dead=true;
40 VY=-20;
41 }
42 }
43 }
44 }
```

Write a class called Bubble which gives endless falling bubbles.

Write Catch function in bubble update function

```
Fruit_user_body_track2
48
49 void setup() {
50 size(1280,960);
51 Kinect = new KinectPV2(this);
52 Kinect.enableBodyTrackImg(true);
53
54 imgHuman=createImage(640,480, ARGB);
55
56 Kinect.init();
57
58 for(int i=0; i<20; i++){
59 bubble[i]=new Bubble(random(1280), random(400), random(30)-15,-random(30));
60 }
61
62 void draw() {
63 background(255);
64 image(Kinect.getBodyTrackImage(), 0,0,1280,960);
65
66 for(int i=0; i<20; i++){
67 bubble[i].draw();
68 bubble[i].update();
69 }
70 }
71
72
73 void copyImage(){
74 PImage n=Kinect.getBodyTrackImage();
75 //now.loadPixels();
76 for (int i=0; i<900*240; i++)
77 {
78 if(n.pixels[i]!=color(255))
79 {
80 imgHuman.pixels[i]=color(255,222,33,500);
81 }
82 else
83 {
84 imgHuman.pixels[i]=color(0,0,0,0);
85 }
86 }
87 imgHuman.updatePixels();
88 }
89 }
```

Call update function in main function.

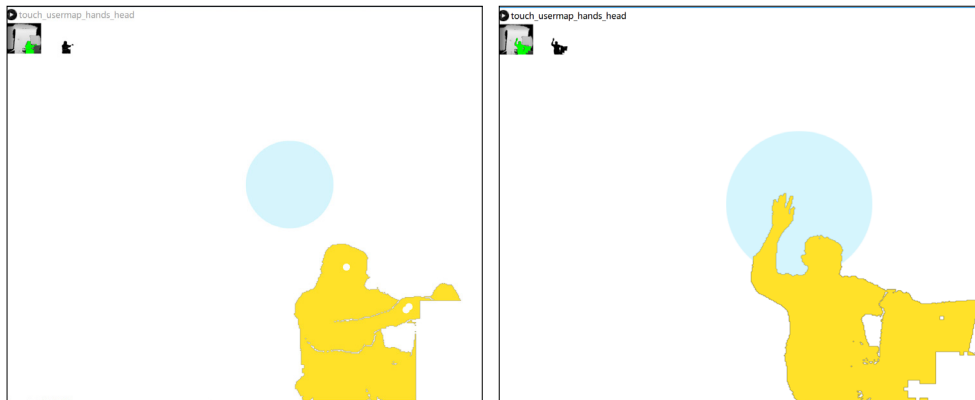
Step 2. Motion sensing interactions

2.3 Use left hand as trigger

Use left hand as a trigger is one of the most necessary and basic functions in motion sensing interaction.

As I mentioned before, Kinect PV2 library provides the skeleton recognize function. So Processing could know the coordinates of left hand real-time. On this basis, the key to achieving this function is judging if the distance between the left hand and object short enough. I set a range 100. If the distance shorter than 100 pixels, the object would change color which means the hand trigger something.

Not only left hand, but all the joints which could be recognized by Kinect could be used as triggers.



```
touch_usermap_hands_head | Processing 3.4
文件 编辑 速写本 调试 工具 帮助

运行

touch_usermap_hands_head Fruit0 carrot
65 ArrayList<KSkeleton> skeletonArray = kinect.getSkeletonDepthMap();
66
67 for (int i = 0; i < skeletonArray.size(); i++) { //individual joints
68   KSkeleton skeleton = (KSkeleton) skeletonArray.get(i);
69   //if the skeleton is being tracked compute the skeleton joints
70   if (skeleton.isTracked()) {
71     KJoint[] joints = skeleton.getJoints();
72     image(kinect.getDepthMaskImage(), 0, 0, 102.4, 84.8); //small depth map
73     image(imgHuman,0,0,512*2.8,424*2.5); // big user map
74     drawBody(joints); // hands head
75     handLeft.set(joints[KinectPV2.JointType_HandLeft].getX()*2.8, joints[KinectPV2.JointT
76     handRight.set(joints[KinectPV2.JointType_HandRight].getX()*2.8, joints[KinectPV2.Joir
77     head.set(joints[KinectPV2.JointType_Head].getX()*2.8, joints[KinectPV2.JointType_Heac
78
79     if(PVector.dist(PVvolcano,handLeft)<400){
80       image(volcano, 600, 250,500,500);
81       image(imgHuman,0,0,512*2.8,424*2.5);
82     }
83
84   }
85   /*-----*/
86
87   /*-----*/
88
89 }
90 }
91 }
92 }
93 }
94 }
95 }
```

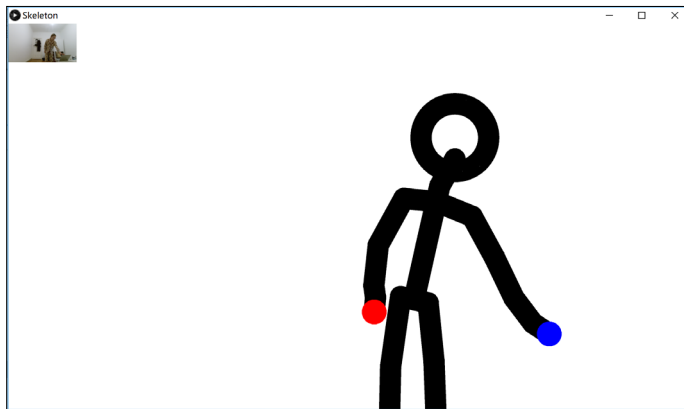
The key part of this function

Step 2. Motion sensing interactions

2.4 Matchstick Men

Inspired by skeleton recognize function, I came up with an idea that I would probably create a matchstick man by simplifying and changing joints and skeletons' shapes.

I thickened all the bones and deleted unnecessary joints just keeping head, left hand, right hand, left foot and right foot. And enlarge head joint making the head joint become a big circle. In this way, the skeleton looks like matchstick men. I think it is an interesting way to cartoon users figures.

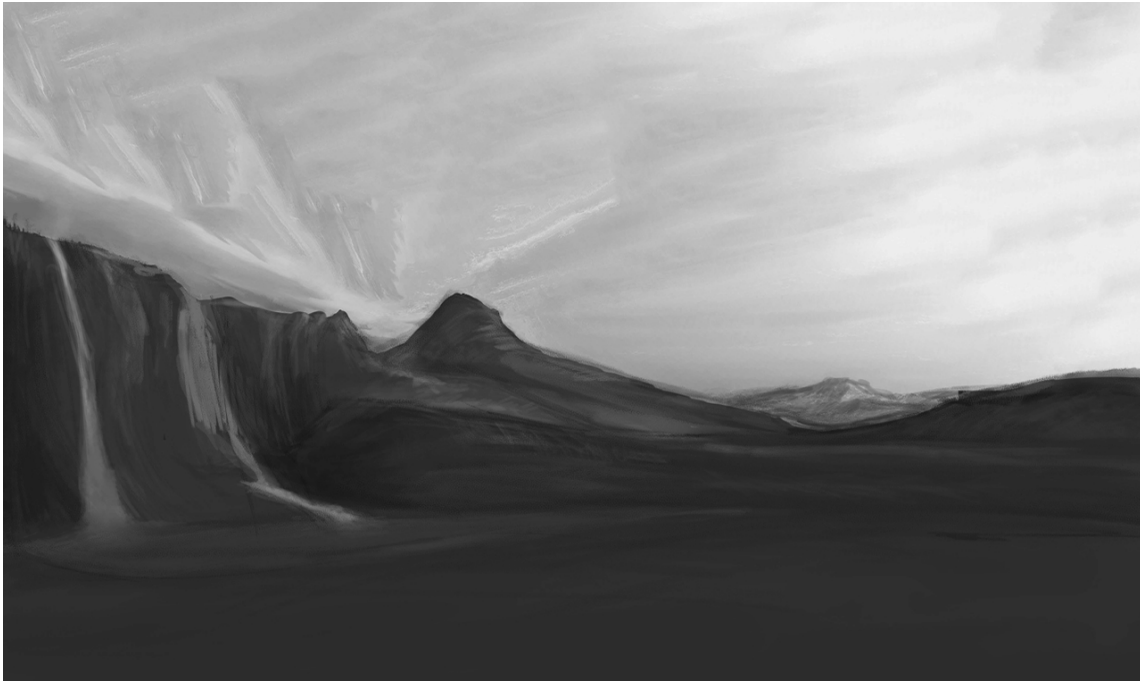


The key part of this function is that change the shape of bones and joints based on skeleton recognize function.



```
Skeleton | Processing 3.4
文件 编辑 速写本 调试 工具 帮助

Skeleton
82 }
83
84 //draw joint
85 void drawJoint(KJoint[] joints, int jointType) {
86   pushMatrix();
87   translate(joints[jointType].getX(), joints[jointType].getY(), joints[jointType].getZ(
88   fill(0);
89   stroke(0);
90   ellipse(0, 0, 5, 5);
91   popMatrix();
92 }
93
94 //draw head
95 void drawHead(KJoint[] joints, int jointType) {
96   pushMatrix();
97   translate(joints[jointType].getX(), joints[jointType].getY(), joints[jointType].getZ(
98   fill(255);
99   stroke(0);
100  ellipse(0, -60, 190, 190);
101  //strokeWeight(100);
102  popMatrix();
103 }
104
105 //draw left hand
106 void drawLefthand(KJoint[] joints, int jointType) {
107   pushMatrix();
108   translate(joints[jointType].getX(), joints[jointType].getY(), joints[jointType].getZ(
109   fill(255,0,0);
110   stroke(255,0,0);
111   ellipse(0, 0, 10, 10);
112   popMatrix();
113 }
114
115 //draw right hand
116 void drawRighthand(KJoint[] joints, int jointType) {
117   pushMatrix();
118   translate(joints[jointType].getX(), joints[jointType].getY(), joints[jointType].getZ(
119   fill(0,0,255);
120   stroke(0,0,255);
121   ellipse(0, 0, 10, 10);
122   popMatrix();
123 }
124
```



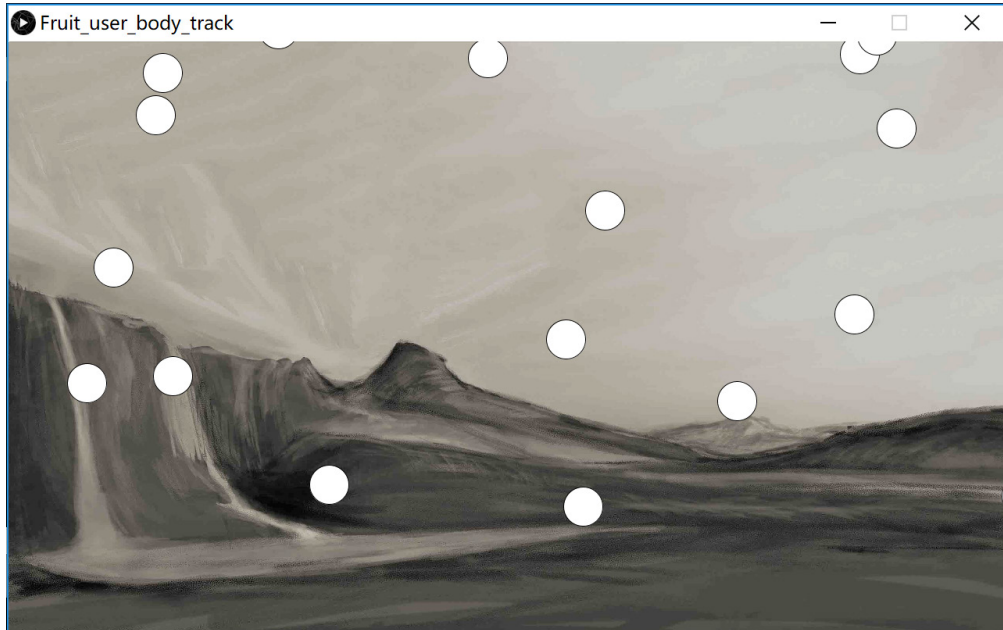
Step 3. Redesign scene

Version 1

From the concept in folio 1, I draw the mountain background. I wanted to create an open field which make people feel free and relax. People could interact with falling objects in this background.

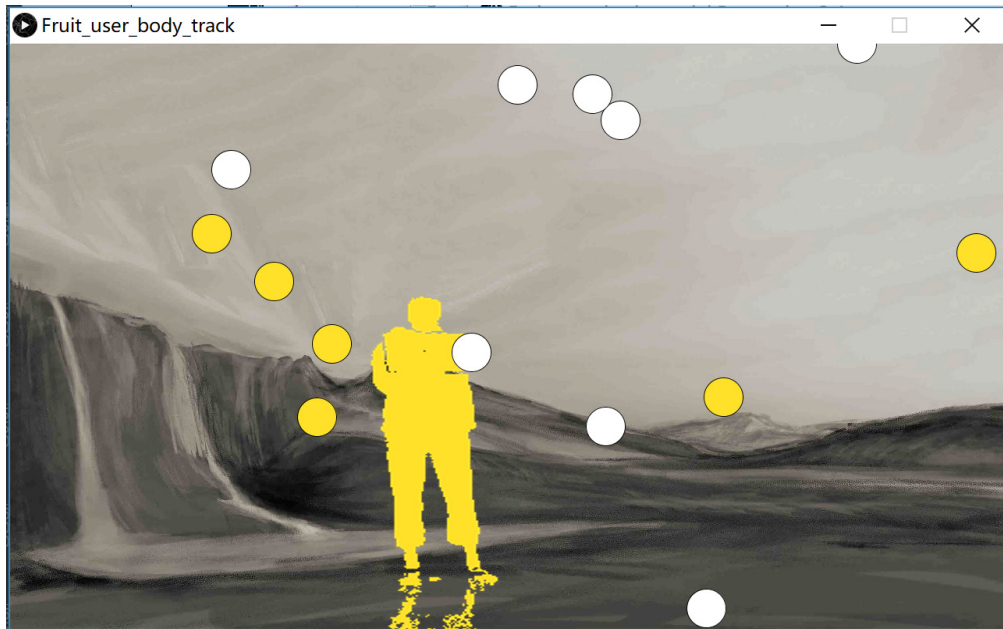


Concept in folio 1



Step 3. Redesign scene

However, when I add the user silhouette on the background it looks a little bit awkward. The background is kind of realistic so I do not think it matches the user silhouette's style.





Step 3. Redesign scene

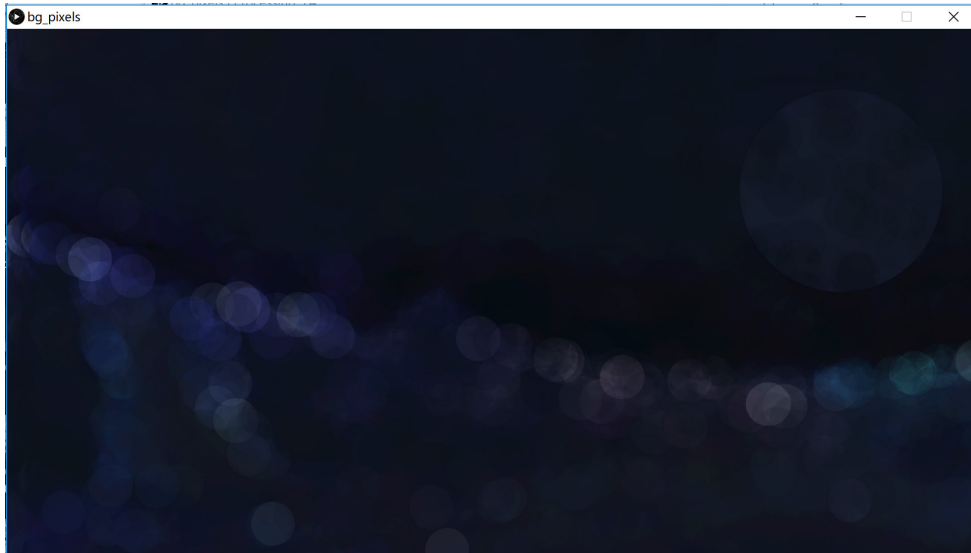
Version 2

So I edited my drawing in PS. I added a filter called Glowing Edges on it and draw some light colors on the bright edges. In this way, the background is more dreamlike and virtual. I think this style is more suitable for the user silhouette.

And I change some parameters of the image in PS to find the better version.



Then I found that the background is too still to match the dynamic user silhouette. So I made the background dynamic and interactive in Processing. I used lots of pixels which update every second to compose this image in Processing. In this way, the image looks like dynamic.



When mouse at the bottom of the window, image looks like this.

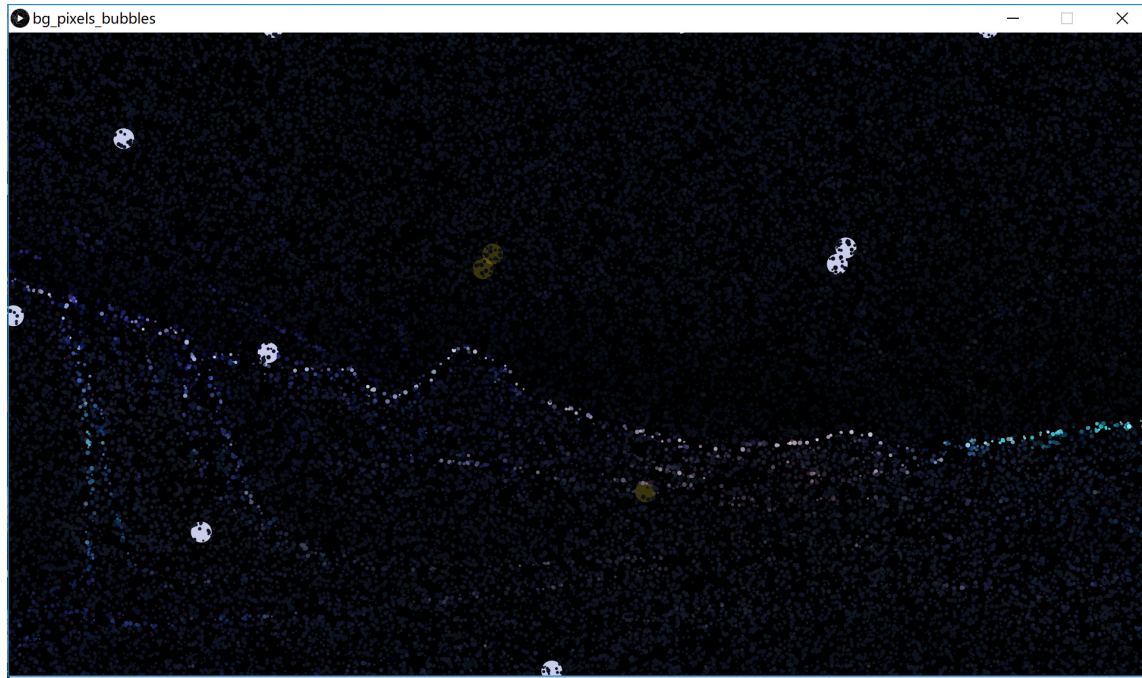


When mouse at the top of the window, image looks like this.

Step 3. Redesign scene

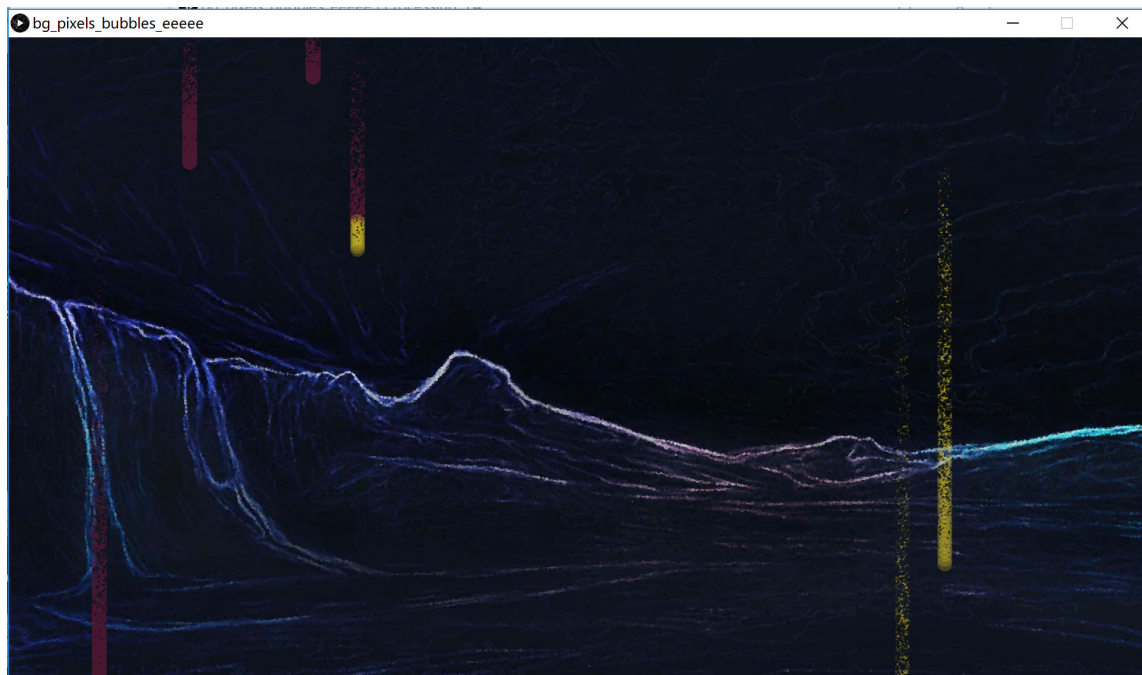
```
1 PImage mountain_brightness_change4;
2 float ex=1650;
3 float ey=320;
4 float sz=400;
5 float a=255;
6
7 void setup(){
8   size(1920,1080);
9   mountain_brightness_change4=loadImage("mountain_brightness_change9.jpg");
10  background(0);
11 }
12
13 void draw(){
14
15   for (int i=0; i<700-mouseY*0.23; i++){
16     float x=random(1920);
17     float y=random(height);
18
19     color c=mountain_brightness_change4.get(int(x),int(y));
20     fill(c,150-mouseY/10);//150---20
21     noStroke();
22     ellipse(x+100-mouseX/10,y,random(2+mouseY*0.08,6+mouseY*0.08),random(2+mouseY*0.08,10));
23   }
24
25   sun();
26 }
27
28 void sun(){
29   ellipse(ex,ey,sz,sz);
30   noStroke();
31   fill(255,222,33,a);
32 }
```

And I also add some interaction for this image in Processing. Users could control the pixels' size by mouse. When mouse at the bottom of the window, pixels are big and the image is blurry. As mouse going up, pixels become smaller and smaller and the image becomes focused.



Step 3. Redesign scene

Also tried different object falling style in this background.



Version 3

The design work above-mentioned is based on the concept in folio 1. One day when I used emoji to chat with my friends, a new idea came to my mind. I think it will be interesting that using emoji things to create an emoji world and allow people to interact with emojis.

I am excited about this idea because emoji is very popular in the world. "Emoji originating on Japanese mobile phones in 1999 and became increasingly popular worldwide in the 2010s after being added to several mobile operating systems. They are now considered to be a large part of popular culture in the west. In 2015, Oxford Dictionaries named the Face with Tears of Joy emoji the Word of the Year" (Wikipedia, 2018).

Emoji exist in the virtual world. It will be attractive and creative that if people could interact and play with emoji in the real world. Comparing with the mountain background, people would have more emotional resonance with my emoji world because people use or know emoji more or less in their life.

So I created an emoji world using emoji elements for example emoji faces, buildings, plants. I show the emojis I used beside.

Step 3. Redesign scene



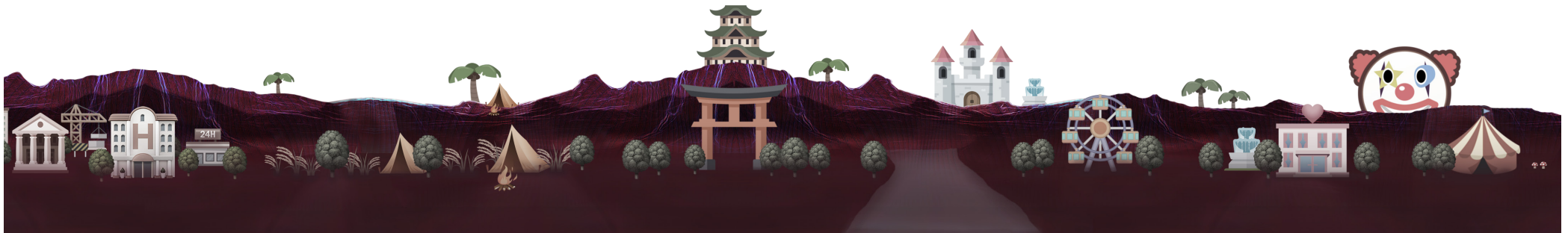
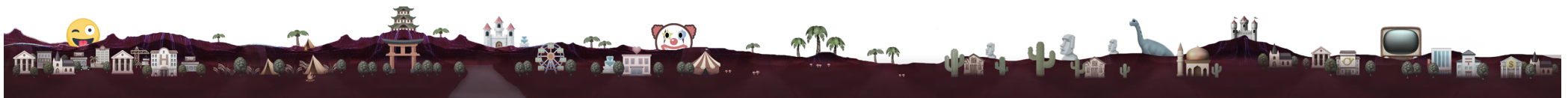


My emoji world
Final scene design

Step 3. Redesign scene

Step 3. Redeisgn scene

I also created a very long scene which includes city, countryside, amusement park, wild field. It would be probably used if I add change scene function.



Step 4. Add interactions in the scene

I really like my emoji world ideas, so I decided to use the emoji world as the scene in the final project. Then I started to think about what kind of interactions suit the emoji world and how to achieve them.

4.1 Wear and take off emoji faces

The first interaction idea come to my mind is that allow users to wear emoji faces on heads in the emoji world. In step 2 I tried to use left hand as a trigger which is similar to wear emoji faces function.

As for wear emoji face function, I set the head joint as a trigger and set a range. If the distance between the head joint's coordinate and the emoji's coordinate shorter than the range, user could wear on the emoji. For users, they walk to emoji faces then they could wear them.

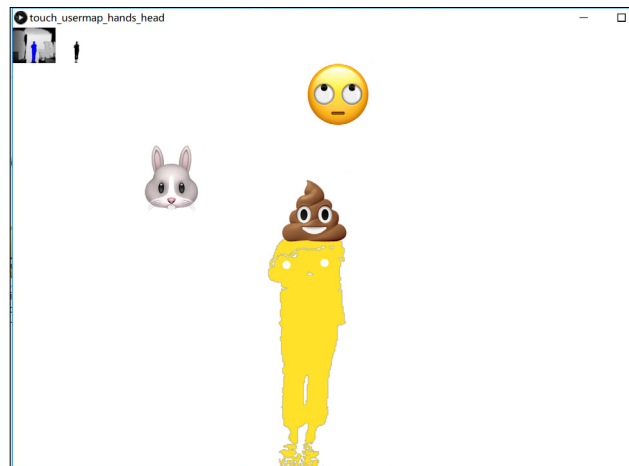
As for taking off emoji face function, I set both left and right hands joints as triggers and set a range. If the distance between the left hand coordinate, right hand coordinate, and the head joint coordinate shorter than the range, user could take off the emoji. For users, when put both left and right hands in front of their face they could take off the emoji.

```
touch_usermap_hands_head | Processing 3.4
文件 编辑 速写本 调试 工具 帮助

touch_usermap_hands_head Fruit0 carrot

79  /*-----*/
80  if(PVector.dist(PVface,head)<200){
81    if( (PVector.dist(handLeft,head)<25) && (PVector.dist(handRight,head)<25)){
82      PVface.x=700;
83      PVface.y=0;
84      // PVface.y+=5;
85      image(face, PVface.x, PVface.y);
86    } else{
87
88      image(face,joints[KinectPV2.JointType_Head].getX()*2.8-80,joints[KinectPV2.JointType_Head].getY()*2.5-80);
89      PVface.x=joints[KinectPV2.JointType_Head].getX()*2.8-80;
90      PVface.y=joints[KinectPV2.JointType_Head].getY()*2.5-80;
91    }
92  }
93  else {
94    // PVface.y+=5;
95    image(face,PVface.x,PVface.y);
96    //if(PVface.y>1060) PVface.y=0;
97  }
98  /*-----*/
```

The key part of wear on function



Interactive part of body: Head

Step 3. Add interactions in the scene

4.2 Fall emojis

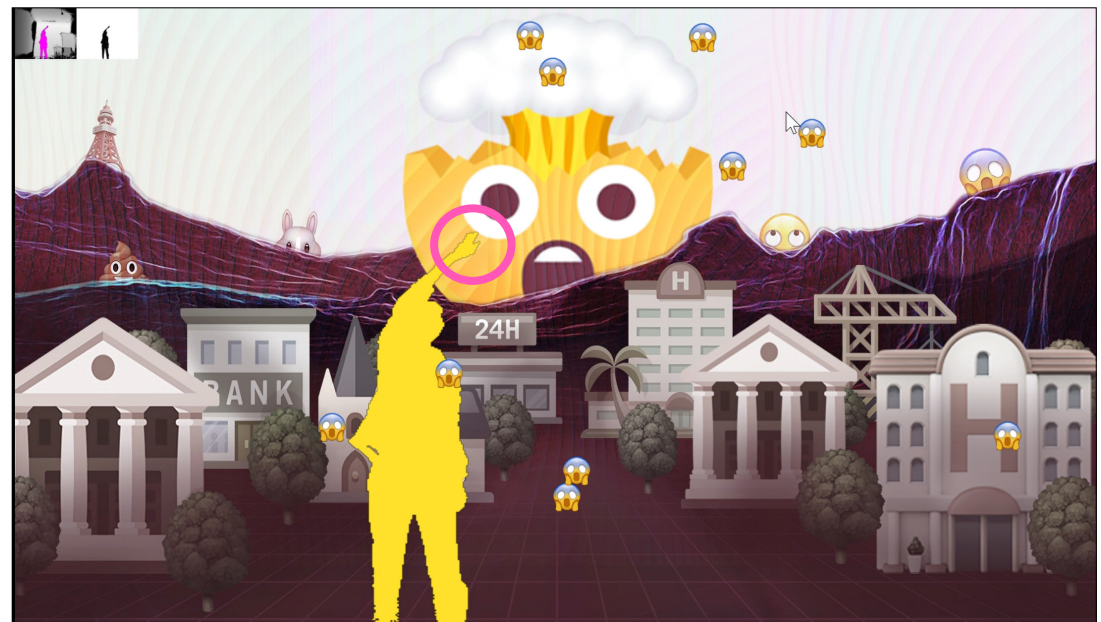
Touch the biggest emoji face, small emojis will fall down. This function is similar to falling bubbles function which I achieved in step 2.

```
new3_2x2_6_original_Animation_bg carrot drop eye falling poo
1 class Falling{
2   float X,Y,W,H,VX,VY;
3   //boolean dead=false;
4   Falling(float x, float y, float w, float h, float vx, float vy){
5     X=x;Y=y; VX=vx; VY=vy; W=w; H=h; //dead=false;
6   }
7
8
9
10  void display(){
11    PImage img;
12    img=loadImage("0.png");
13    image(img, X, Y, W, H);
14    X+=VX;
15    Y+=VY;
16    VY+=1;
17    if(X<0||X>1800||Y<0||Y>900){
18      X=900; Y=200; VX=random(30)-15; VY=-random(30);
19    }
20  }
21
22 }
23
24 /*-----*/
25
```

Define a class called Falling which control carrots falling.

```
new3_2x2_6_original_Animation_bg carrot drop eye falling poo
356 /*-----*/
357
358 /*-----falling-----*/
359 if(PVector.dist(PVvolcano, handLeft)<100){
360   image(volcano, -40, -57,1926,1085.97);
361   image(imgHuman,80,0,512*C,424*K);
362   for(int a=0; a<10; a++) fruit[a].display();
363   image(cover,-40, -57,1926,1085.97);
364 }
365
366 /*-----*/
367
```

Call the Falling class in main founction. In this way carrot could fall down in my scene.



Interactive part of body: Left hand

Step 4. Add interactions in the scene

Then I refine this function.

4.2.1. When people touch the biggest emoji face it will change to exploding head.

4.2.2. Controlling falling emoji's types by wearing different emoji faces

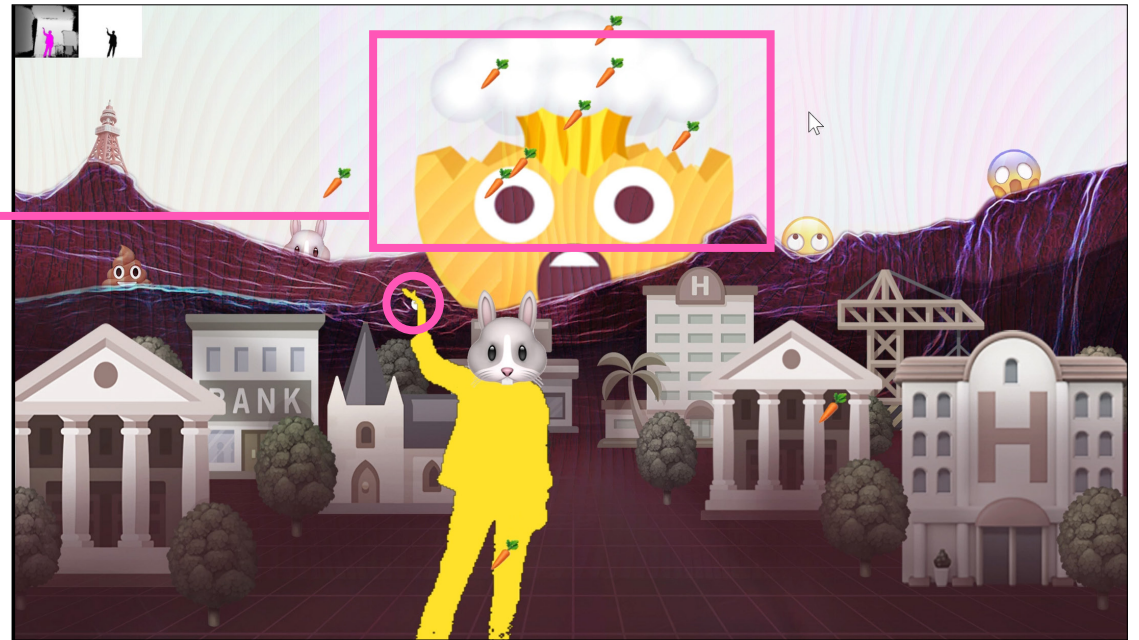
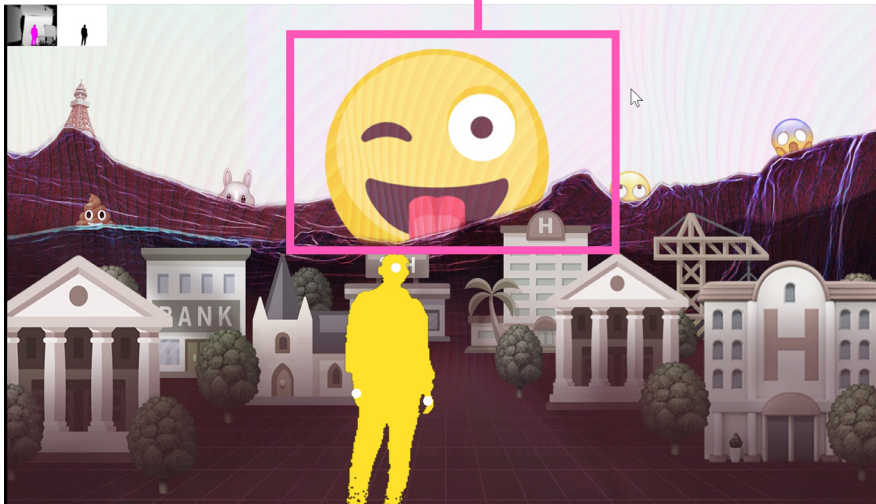
The key to controlling falling emoji's types by wearing different emoji faces is that set different falling emoji function inside different wear emoji face function. For example, I set fall carrot function inside wear rabbit face function, which means wear rabbit is the precondition of fall carrot. So when people wear rabbit and touch the biggest emoji, they will see carrots fall down.

```
new_2x2_6_original_Animation_bg carrot drop eye fruit poo
195 /-----rabit-----rabit-----x/
196
197 if(PVector.dist(PVrabit2,head)<150){
198     Pickup.play();
199
200     PVface3.x=1280;
201     PVface3.y=319;
202     image(face3, PVface3.x-80, PVface3.y-80,1,1);
203     PVpool.x=177;
204     PVpool.y=399;
205     image(pool, PVpool.x-80, PVpool.y-80,1,1);
206     PVface4.x=1600;
207     PVface4.y=235;
208     image(face4, PVface4.x-80, PVface4.y-80,1,1);
209
210     if ( (PVector.dist(handLeft,head)<30) && (PVector.dist(handRight,head)<30)){
211         PVvolcano2.x=900;
212         PVvolcano2.y=250;
213         PVrabit2.x=473;
214         PVrabit2.y=280;
215         image(rabit2, PVrabit2.x-80, PVrabit2.y-80);
216     } else{
217         image(rabit2,joints[KinectPV2.JointType_Head].getX()*C,joints[KinectPV2.JointType_Head].getY()*K-90);
218         PVrabit2.x=joints[KinectPV2.JointType_Head].getX()*C;
219         PVrabit2.y=joints[KinectPV2.JointType_Head].getY()*K-90;
220
221         if (PVector.dist(PVvolcano2,handLeft)<200){ //falling carrot
222             timerabbit--;
223             if (timerabbit>0){
224                 PVvolcano2.x=joints[KinectPV2.JointType_HandLeft].getX()*C+80;
225                 PVvolcano2.y=joints[KinectPV2.JointType_HandLeft].getY()*K;
226             } else {PVvolcano2.x=900; PVvolcano2.y=250; timerabbit=600;}
227
228             image(volcano, -40, -57,1926,1085.97);
229             image(imgHuman,80,0,512*C,424*K);
230             image(rabit2,joints[KinectPV2.JointType_Head].getX()*C,joints[KinectPV2.JointType_Head].getY()*K-90);
231             PVrabit2.x=joints[KinectPV2.JointType_Head].getX()*C;
232             PVrabit2.y=joints[KinectPV2.JointType_Head].getY()*K-90;
233
234             for(int ca=0; ca<10; ca++) carrot[ca].display();
235             image(cover,-40, -57,1926,1085.97);
236
237         }
238     }
239 }
```

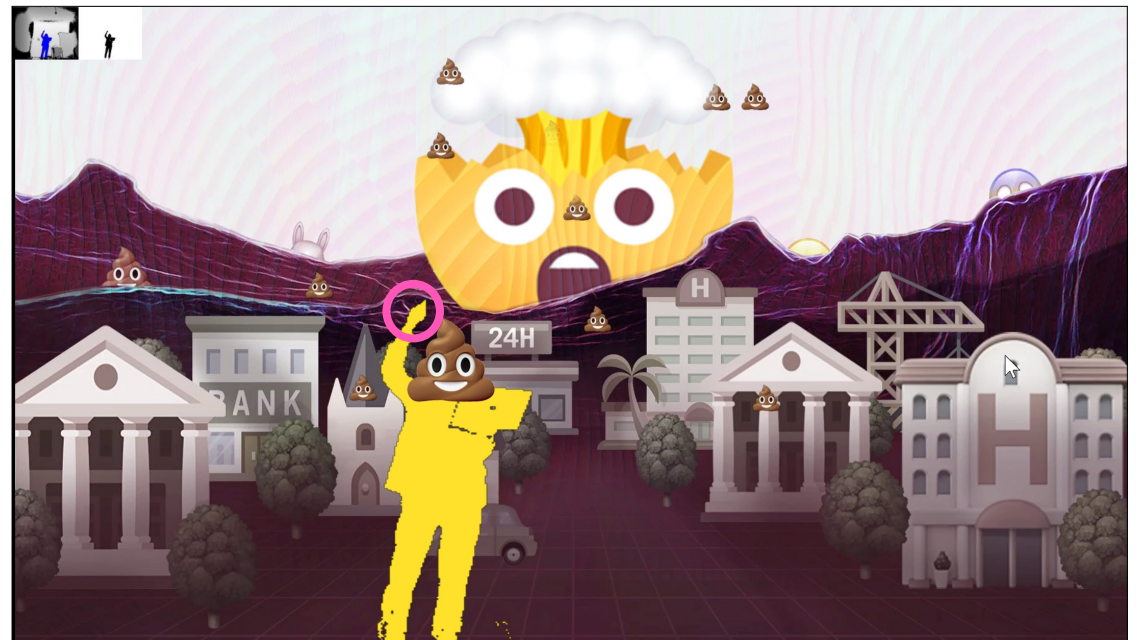
Wear and take off rabbit face function

Touch the biggest emoji then falling carrot function

The face will change to exploring head when people touch it



Wear rabbit, carrots fall. Interactive part of body: Left hand.



Wear poo, poo fall. Interactive part of body: Left hand.

Step 4. Add interactions in the scene

Step 4. Add interactions in the scene



4.3 Catch falling emojis

I also added catch falling emojis function which is similar to catch falling bubbles function achieved in step 2. In this way, people could interact with these falling emojis.

Interactive part of body: Whole body.



Step 4. Add interactions in the scene

4.4 Get on and get off the bus

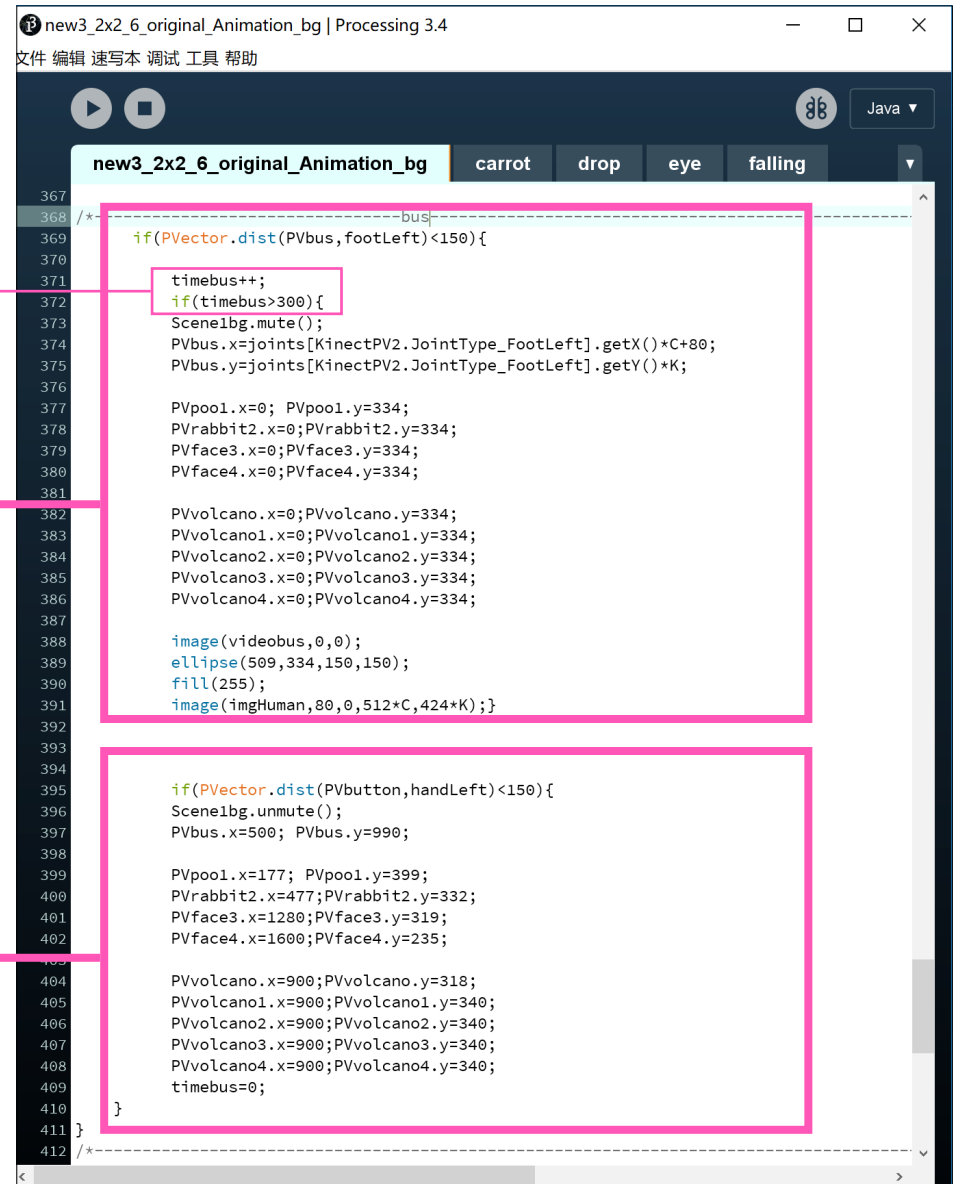
I decided to add change scene function because I want to show more about the emoji world rather than just one scene. So I added a bus stop in the first scene. When people stand beside the bus stop more than 5 seconds, people could get on the bus. Through the bus window, people could see the landscape outside the bus is changing. It is like have a journey in the emoji world.

The code of getting on and getting off the bus is on the right.

Stay in the range for 5 seconds, you could get on the bus

Get on the bus

Get off the bus



```
new3_2x2_6_original_Animation_bg | Processing 3.4
文件 编辑 速写本 调试 工具 帮助

new3_2x2_6_original_Animation_bg carrot drop eye falling

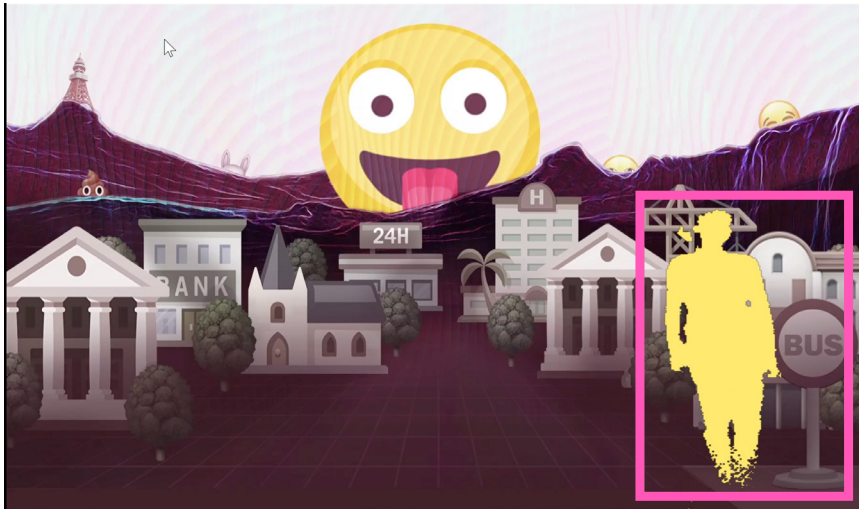
367
368 /*-----bus-----*/
369 if(PVector.dist(PVbus,footLeft)<150){
370
371     timebus++;
372     if(timebus>300){
373         Scene1bg.mute();
374         PVbus.x=joints[KinectPV2.JointType_FootLeft].getX()*C+80;
375         PVbus.y=joints[KinectPV2.JointType_FootLeft].getY()*K;
376
377         PVpoo1.x=0; PVpoo1.y=334;
378         PVrabbit2.x=0;PVrabbit2.y=334;
379         PVface3.x=0;PVface3.y=334;
380         PVface4.x=0;PVface4.y=334;
381
382         PVvolcano.x=0;PVvolcano.y=334;
383         PVvolcano1.x=0;PVvolcano1.y=334;
384         PVvolcano2.x=0;PVvolcano2.y=334;
385         PVvolcano3.x=0;PVvolcano3.y=334;
386         PVvolcano4.x=0;PVvolcano4.y=334;
387
388         image(videobus,0,0);
389         ellipse(509,334,150,150);
390         fill(255);
391         image(imgHuman,80,0,512*C,424*K);}
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411 }
412 /*-----*/

if(PVector.dist(PVbutton,handLeft)<150){
    Scene1bg.unmute();
    PVbus.x=500; PVbus.y=990;

    PVpoo1.x=177; PVpoo1.y=399;
    PVrabbit2.x=477;PVrabbit2.y=332;
    PVface3.x=1280;PVface3.y=319;
    PVface4.x=1600;PVface4.y=235;

    PVvolcano.x=900;PVvolcano.y=318;
    PVvolcano1.x=900;PVvolcano1.y=340;
    PVvolcano2.x=900;PVvolcano2.y=340;
    PVvolcano3.x=900;PVvolcano3.y=340;
    PVvolcano4.x=900;PVvolcano4.y=340;
    timebus=0;
}
```

Step 4. Add interactions in the scene



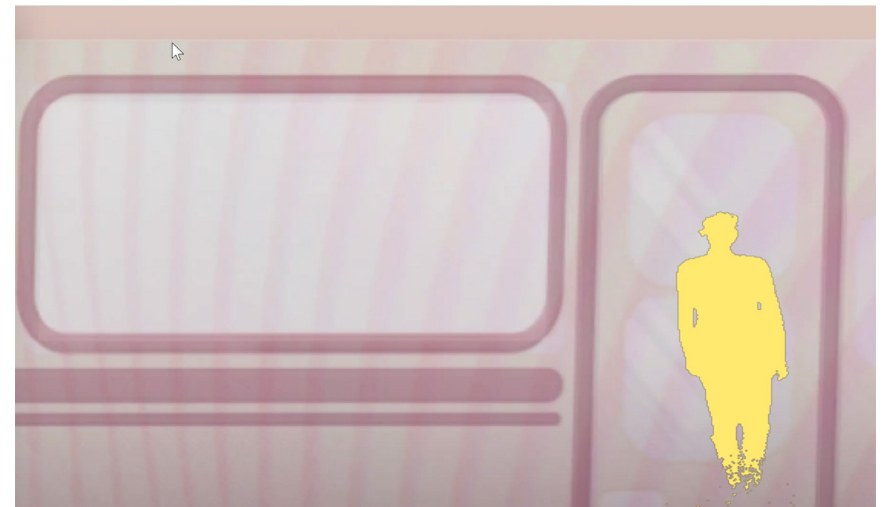
Here is a bus station. Interactive part: Feet.



Bus is coming



On the bus



Transition

Step 4.

Add interactions in my scene

```
new3_2x2_6_original_Animation_bg carrot drop eye fruit poo
13
14 import ddf.minim.*;
15 Minim minim;
16 AudioPlayer Scene1bg;
17 AudioSample Pickup;
18
19 Movie video;
20 Movie videobus;
21 KinectPV2 kinect;
```

Import minim library which is a powerful sound library. define two sound variables one for background sound, one for short sound effect.

```
new3_2x2_6_original_Animation_bg carrot drop eye fruit poo
40
41 -----
42 void setup(){
43   size(1800,1060,P3D);
44
45   minim = new Minim(this);
46   Pickup = minim.loadSample("pickup.wav");
47   Scene1bg=minim.loadFile("pickup3.wav");
48   Scene1bg.loop();
49
50   video=new Movie(this, "saturday0.6.mov");
51   video.loop();
52   videobus=new Movie(this, "busbusbusnew.mov");
53   videobus.loop();
```

Load sound files. And loop background sound.

```
new3_2x2_6_original_Animation_bg carrot drop eye fruit poo
159
160 -----
161 if(PVector.dist(PVpool,head)<150){
162   sound--;
163   if(sound>0){
164     Pickup.trigger();
165
166     PVface3.x=1280;
167     PVface3.y=280;
168     image(face3, PVface3.x-80, PVface3.y-80,1,1);
169     PVrabbt2.x=477;
170     PVrabbt2.y=280;
```

Put play sound function in wear emoji face function.

4.5 Sound

Background sound

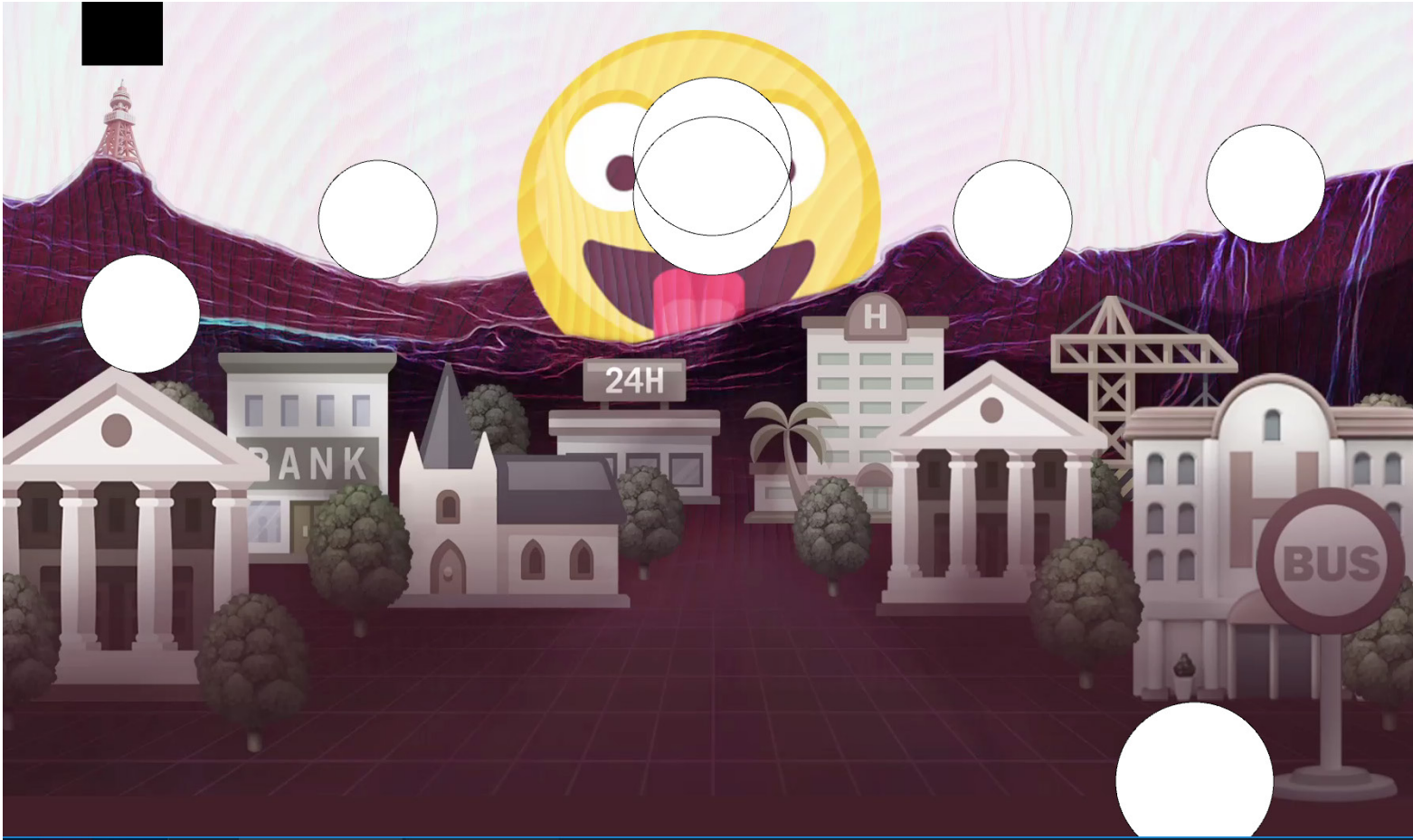
I chose chill and relax background sound which makes people can help dancing when they wear emoji face.

wear emoji face sound effect

I add a short sound effect in wear emoji face function. So when people pick up emoji face they would not only get funny visual feedback but also sound feedback.

The key code of the sound function on the left.

Step 5. Challenges



These white circles represent interactive areas.

In the end, I adjusted all the interactive area in my scene. I set 2, 3 and 4 interactive areas a little bit high which engage people to jump to pick up the emoji faces.

And the falling emojis would disappear in several seconds after people touch the biggest face at the middle. If people touch the biggest face again, the small emojis will fall again. I think this way would engage people to interact with the biggest face more and avoid that people pay too much attention to the small falling emojis and miss other interesting interactions in my scene.

In studio 1, I chose a totally new topic, motion sensing interaction, for myself. It brings people immersion and free people from devices. So people could interact with our design more naturally and freely.

In folio 2, I designed an emoji would scene and achieved some motion sensing interactions in my scene with Processing and Kinect sensor.

The list of motion sensing interactions in my scene

1. Wear 4 emoji faces with sound effect.

Trigger: user's HEAD close to one of the 4 wearable emoji faces enough

2. Take of emoji faces after wearing.

Trigger: user's LEFT HAND overlaps RIGHT HAND in front of his/her head.

3. Make the biggest face explore small emoji stuff.

Trigger: user's LEFT HAND close to the biggest emoji face enough.

4. Catch small emoji stuff.

Trigger: catch small emoji stuff by the whole body.

5. Get on the bus.

Trigger: user stands close to the bus stop enough in 5 seconds.

6. Get off the bus.

Trigger: user's LEFT HAND close to the stop button enough.

Conclusion

During the process of making this project, I generally know what motion sensing interaction is and how it works and how to using Processing and Kinect sensor to achieve simple motion sensing interaction. It is not the only way to achieve motion sensing interaction, but I think Precossing+Kinect sensor is friendly to beginners like me because that there are lots of tutorial online, Processing support Kinect well and the Processing language is not very hard to understand. I am still in a very begging stage of using Processing and Kinect and there are lots of things I want to learn in future, for example, make creative and abstract user's figure like this.



As for my project, I think there is still a lot of potentials to improve more. For example, beautify the user silhouette, add some interactions after people getting on the bus and make it support multi-user not well, which did not achieve because of the tight schedule and my poor programming skill. I will go further in Processing and hope I could make this project more complete in future.

Thank you!