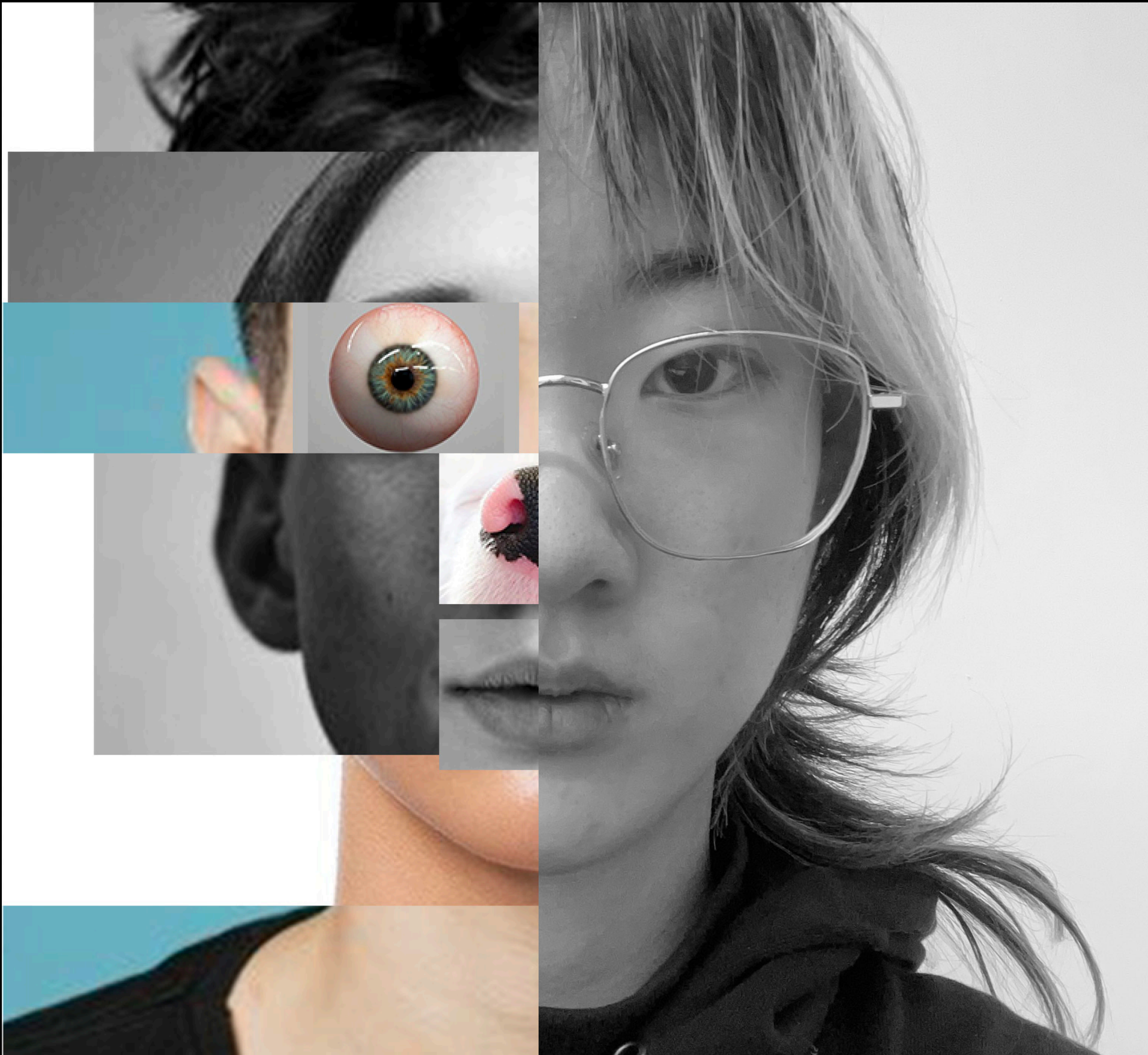




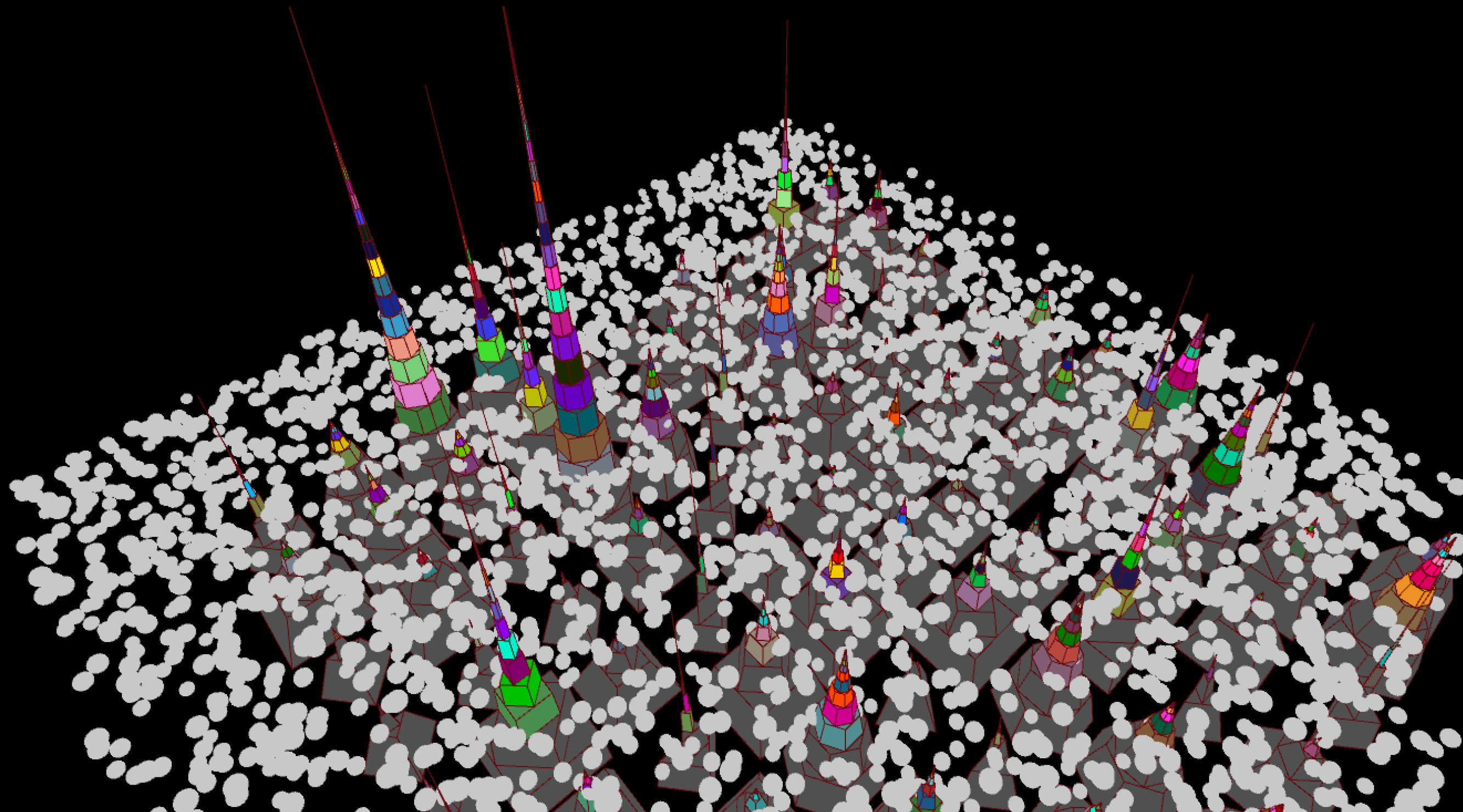
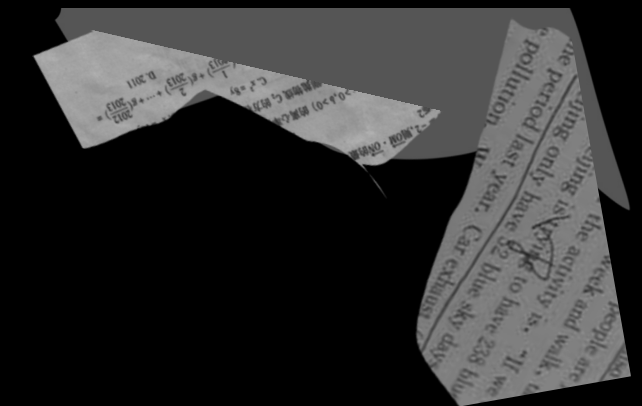
MEDIART206 Computer Graphics

# Portfolio of Jiaxin Wang

|                         |       |                 |
|-------------------------|-------|-----------------|
| UNDERAGE                | ..... | PARAMETERIZE.   |
| Surrealism Generator    | ..... | TRANSFORMATION. |
| WHY BRACES ???          | ..... | REPETITION.     |
| Doggies Want Bone       | ..... | SIMULATION.     |
| ADHD Data Visualization | ..... | VISUALIZATION.  |

# UNDERAGE

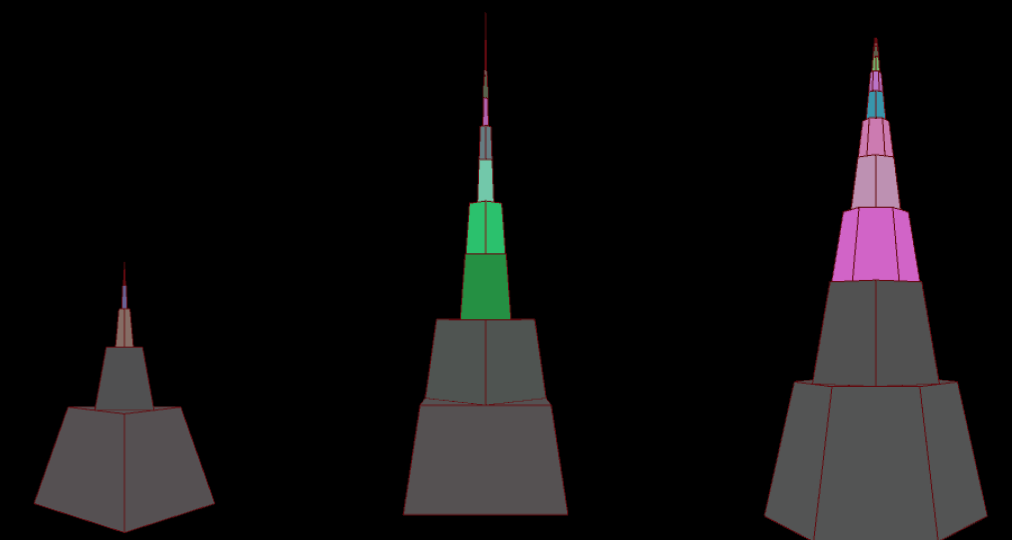
Parametric Design



## Concept Statement

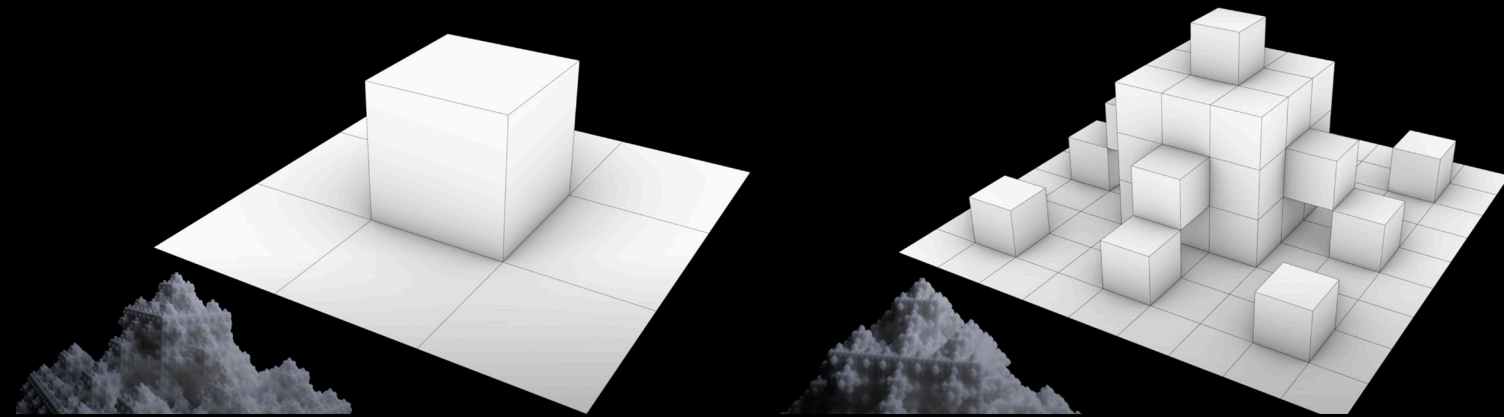
UNDERAGE is a sketch that visualizes the muffled and confused underage life growing up in an East Asia environment. Personally speaking, the fact that Chinese style education uses several “important” exams to split up kids’ childhood and puberty, is like chopping up their uniqueness and using one single standard to determine their success. However, whether a person lives a valuable life is by no means determined by a single life stage, but by every -thing that he or she experiences during the entire lifespan.

I visualize this idea using simple geometry structures in this sketch. Every stack of frustums is a symbol of a person. It can either shrink its size to grow higher in one level, or choose to be patient but maintain the potential to grow higher later. Some of them stay in the gray shadow of childhood forever, while others grow into colorful towers.



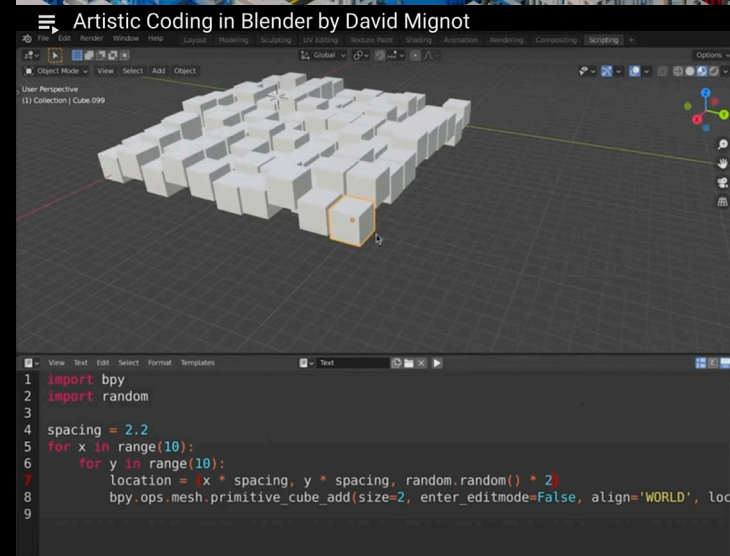
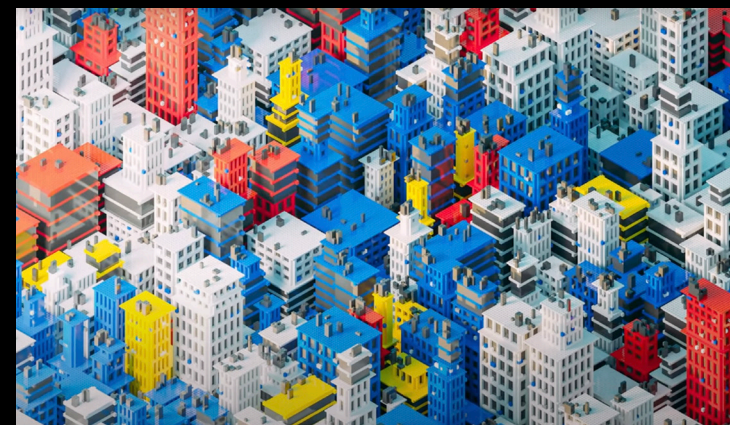


# Visual Research



I want my parametric system to engage recursion and 3D geometry shapes. This fractal shape gives me idea for my own recursive parametric design.

<https://www.youtube.com/watch?v=S1kpqOqSttA&t=331s>



About suppressed and muf- fled emotions, visual elements like black and white images, blurry gray colors are usually used. Therefore, I decided to use gray as the basic color from the view of an underage kid.

I drew inspirations from the artist David Mignot. He uses python in Blender to generate architecture models. I find the repetitive city landscape creates the effect that I hope to achieve: each building is different while similar to each other, allowing me to show the differences between parameterized objects.

# Code & Generative Rules

## Visual Objects

The visual objects that I want to generate are towers made up of layers of frustums. There are three different cross-section shapes: rectangles, triangles, and hexagons. The top surface of it has the same shape but is smaller. The bottom few tiers has a gray color, while the top tiers have random pastel colors.

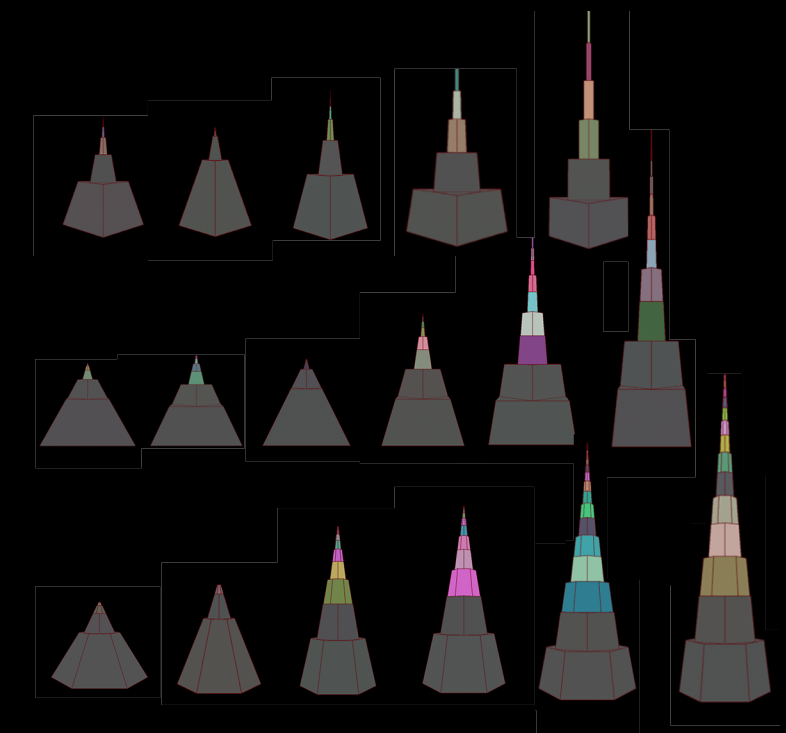
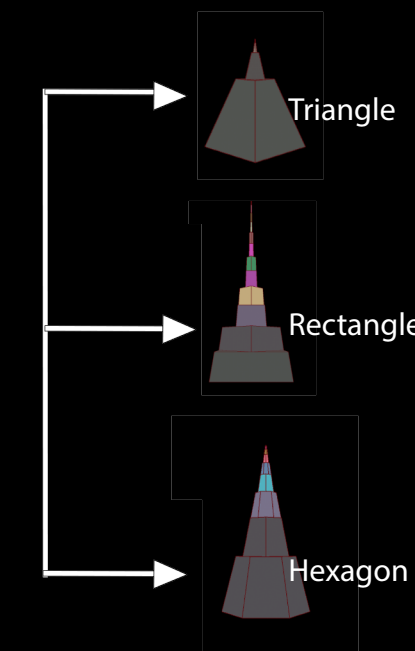
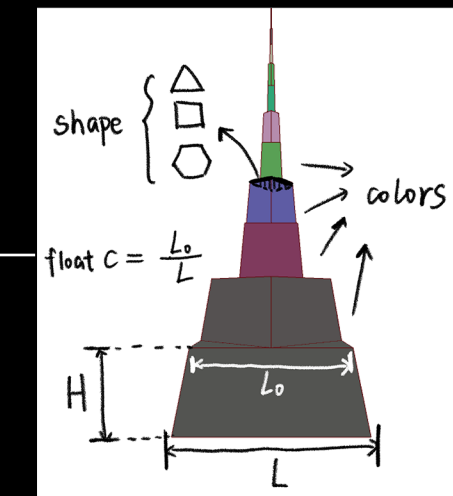
**L** Side length of the “base shape”, the rectangle/triangle/hexagon on the bottom

**H** Height of the bottom tier frustum

**c** A coefficient, representing: side length of the top surface polygon/side length of the bottom surface polygon

**Shape** Shape of the polygon

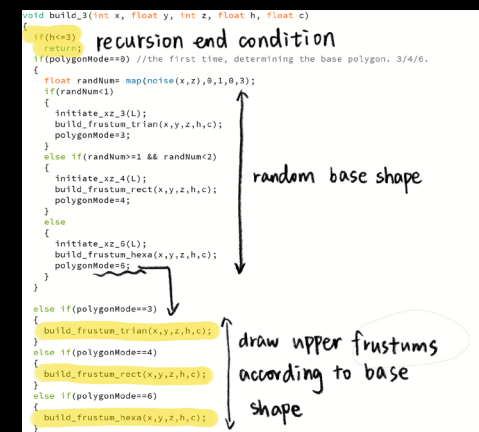
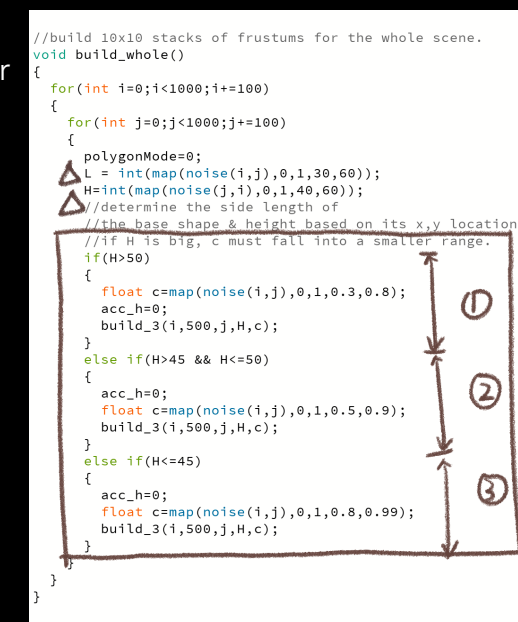
**Color**



## Relationships & Scopes

There is a tradeoff between higher H and bigger c: H all fall in the range between 40 and 60. I separate different H into 3 intervals: (50,60), (45,50], and (40,45]. The bigger H value is, the smaller the c value range has.

This symbolizes that even though some people can seem to develop better in a certain life stage, he/she is meanwhile limiting his/her potential to grow.



Recursion

I used a recursive function to draw the frustums. Draw the bottom one first (with a random shape out of triangles/rectangles/hexagons), and then draw the upper ones with the same shape. When the height of a upper frustum is less than 3, the function stops calling itself, and the stack stops building.

# Code & Functions

```
float nx1=c*(triangles[0].x+triangles[1].x)/2;
float nz1=c*(triangles[0].y+triangles[1].y)/2;
float nx2=c*(triangles[1].x+triangles[2].x)/2;
float nz2=c*(triangles[1].y+triangles[2].y)/2;
float nx3=c*(triangles[2].x+triangles[0].x)/2;
float nz3=c*(triangles[2].y+triangles[0].y)/2;

triangles[0].x=nx1;
triangles[1].x=nx2;
triangles[2].x=nx3;
triangles[0].y=nz1;
triangles[1].y=nz2;
triangles[2].y=nz3;

color pickColor()
{
    int r,g,b;
    if(acc_h<60)
    {
        r=int(random(80,85));
        g=int(random(80,85));
        b=int(random(80,85));
    }
    else if(acc_h>=60 && acc_h<100)
    {
        r=int(random(60,150));
        g=int(random(60,150));
        b=int(random(60,150));
    }
    else if(acc_h>=100 && acc_h<250)
    {
        r=int(random(0,255));
        g=int(random(0,255));
        b=int(random(0,255));
    }
    else
    {
        r=int(random(0,255));
        g=int(random(0,255));
        b=int(random(0,255));
    }
    return color(r,g,b);
}
```

To draw the tower, I used several functions.

1. To draw the frustum above the base one, find the midpoints of each edge and connect them as the bottom surface of the upper frustum.
2. use beginShape(), endShape(), and vertex() to outline the frustum.
3. pickColor() picks the color of each layer of frustum based on its height.

```
fill(pickColor());

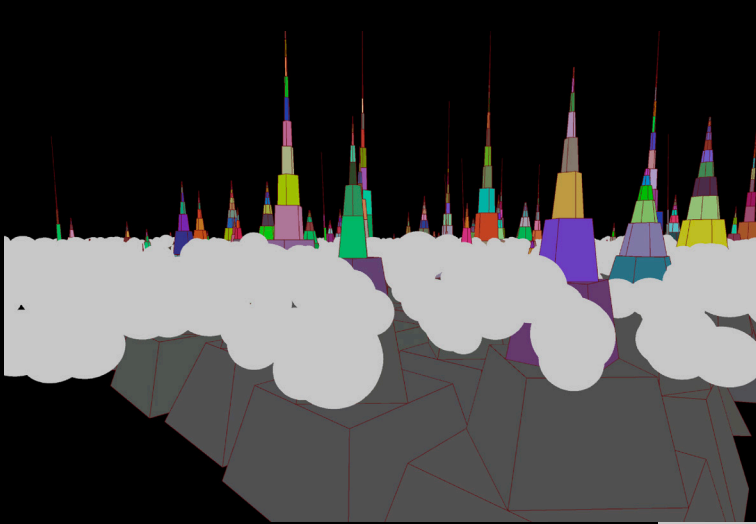
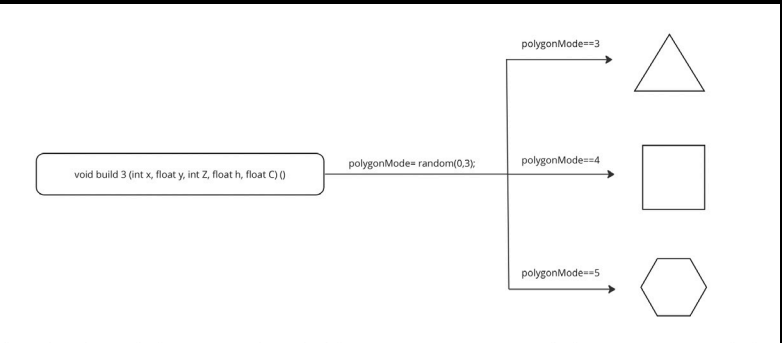
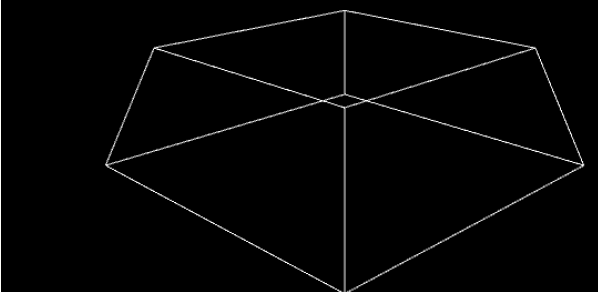
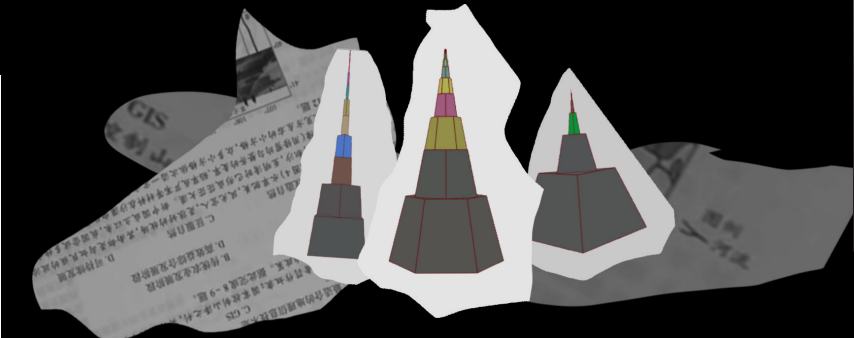
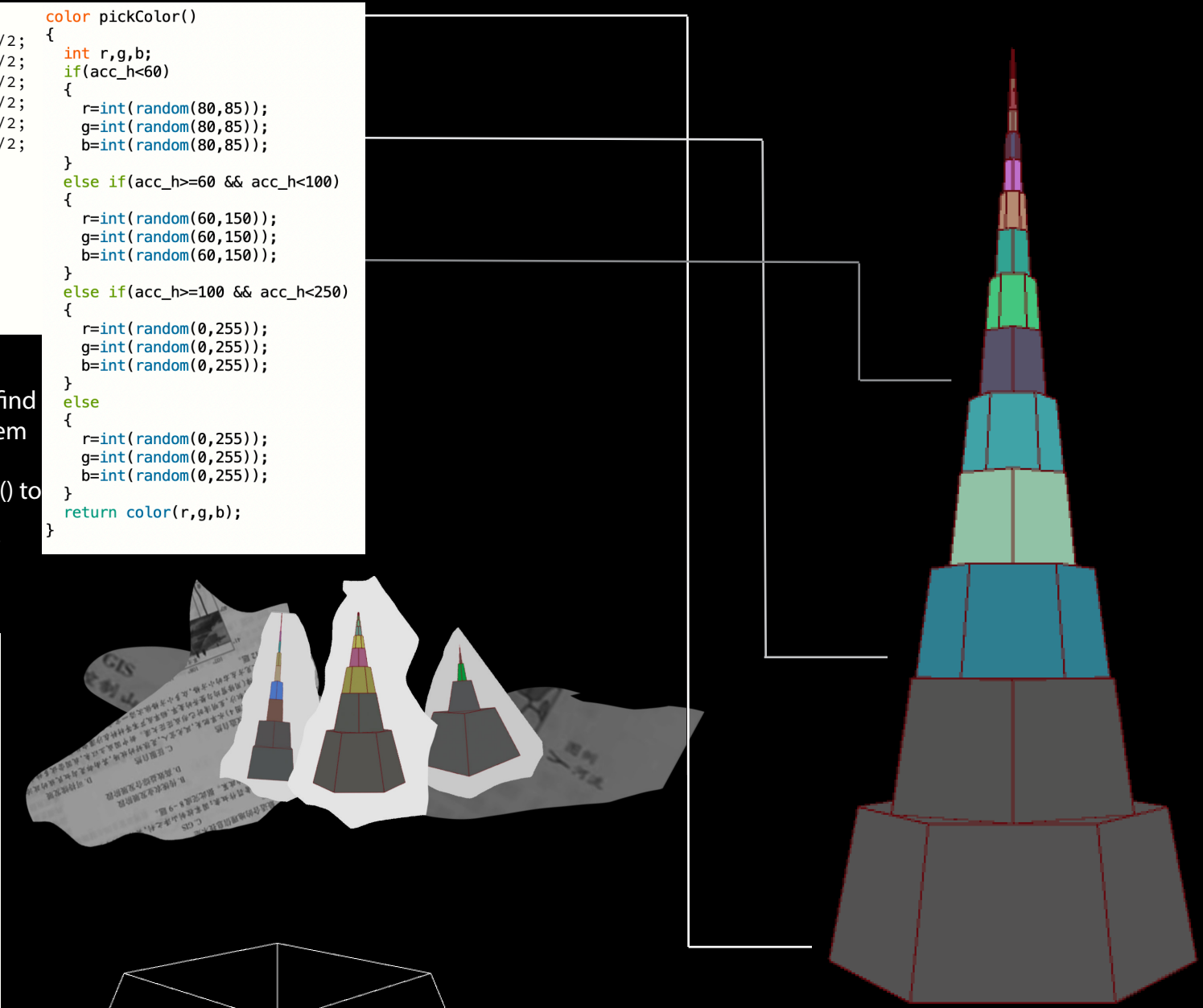
beginShape();
vertex(triangles[0].x,0,triangles[0].y);
vertex(triangles[1].x,0,triangles[1].y);
vertex(triangles[1].x*c,-h,triangles[1].y*c);
vertex(triangles[0].x*c,-h,triangles[0].y*c);
endShape(CLOSE);

beginShape();
vertex(triangles[1].x,0,triangles[1].y);
vertex(triangles[2].x,0,triangles[2].y);
vertex(triangles[2].x*c,-h,triangles[2].y*c);
vertex(triangles[1].x*c,-h,triangles[1].y*c);
endShape(CLOSE);

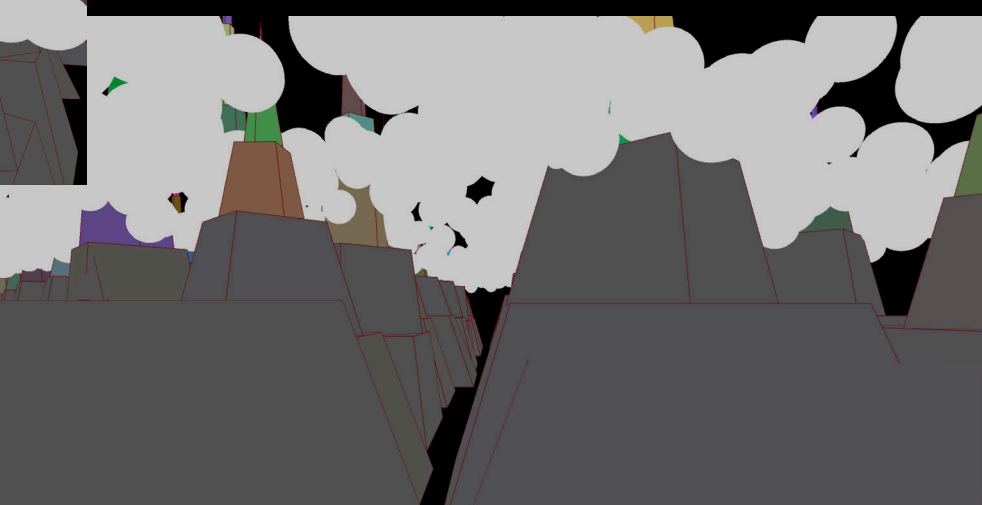
beginShape();
vertex(triangles[2].x,0,triangles[2].y);
vertex(triangles[0].x,0,triangles[0].y);
vertex(triangles[0].x*c,-h,triangles[0].y*c);
vertex(triangles[2].x*c,-h,triangles[2].y*c);
endShape(CLOSE);

beginShape();
vertex(triangles[0].x,0,triangles[0].y);
vertex(triangles[1].x,0,triangles[1].y);
vertex(triangles[2].x,0,triangles[2].y);
endShape(CLOSE);

beginShape();
vertex(triangles[0].x*c,-h,triangles[0].y*c);
vertex(triangles[1].x*c,-h,triangles[1].y*c);
vertex(triangles[2].x*c,-h,triangles[2].y*c);
endShape(CLOSE);
popMatrix();
```



To create a layer of fog around the top of the gray layer, I added a particle system. I hope that viewers see from the bottom of the sketch when it first start to run, so that their view should be covered up by the fog. This symbolizes that during childhood and puberty, my eyesight was limited so that I couldn't see what life truly is in adulthood, so it created a lot of anxiety and suppressed emotions.



The particles have their instance variables. They randomly appear across the whole area. They are affected by the gravitation force from the mouse, so that when they are close to the mouse, they are drawn to where (mouseX, mouseY) is.

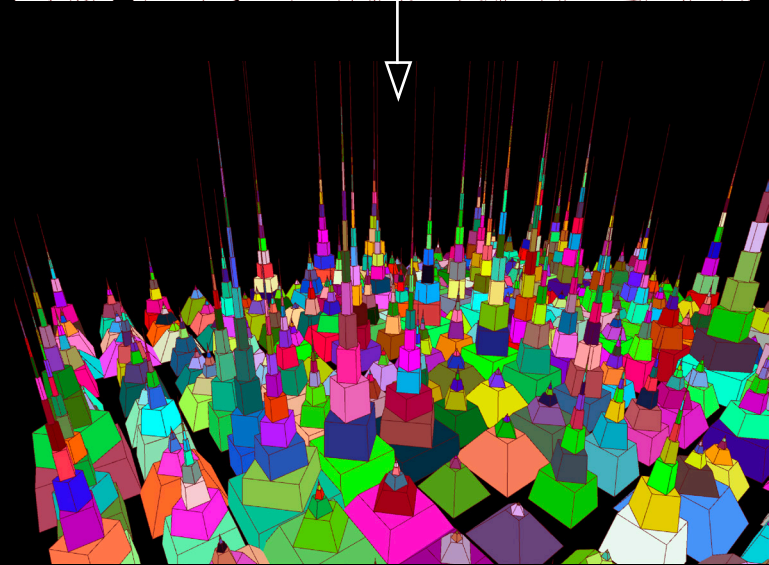
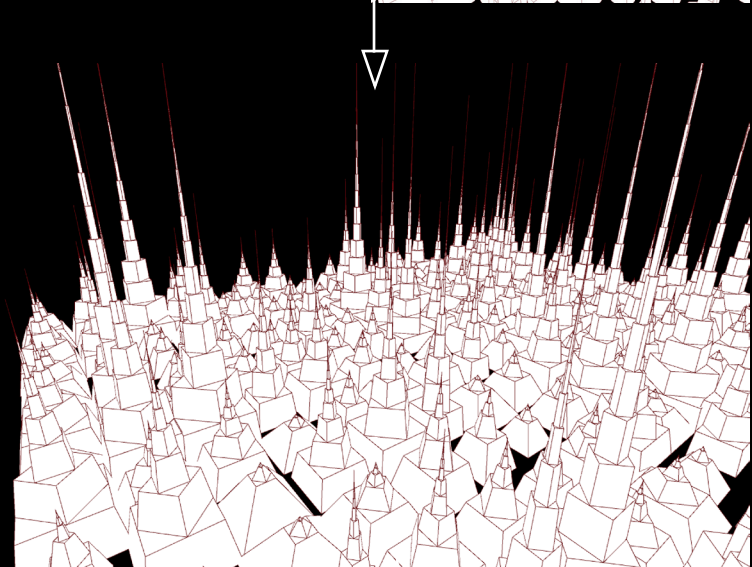
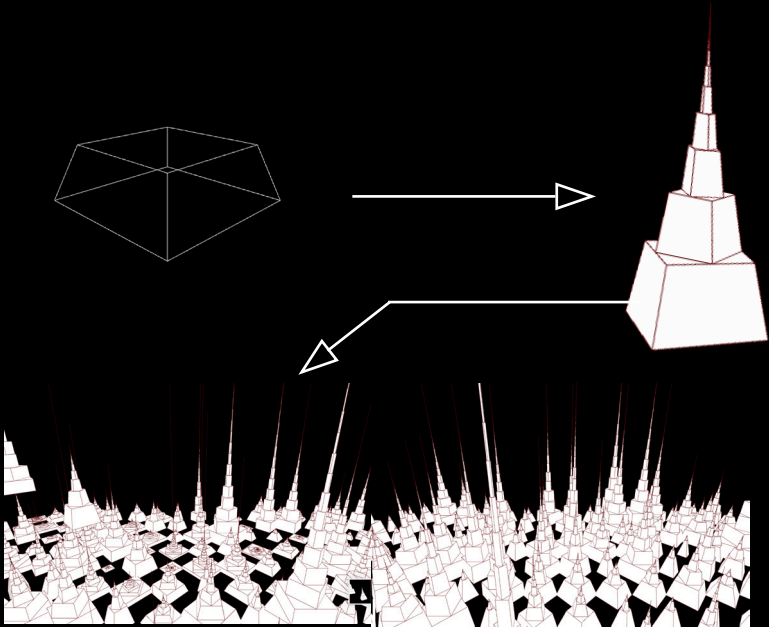
```
class Particle
{
    float xPos, zPos,yPos;
    float size;
    float speedX;
    float speedY;
    float speedZ;
    float col;

    Particle()
    {
        xPos=random(-200, 1000);
        zPos=random(-200, 1000);
        yPos=random(500-100-10,500-100+10);
        speedX=random(-0.2, 0.2);
        speedY=random(-0.2, 0.2);
        speedZ=random(-0.01,0.01);

        size=random(6, 10);
        //col=255; Previously yellow one.
        col=0;
    }
}
```

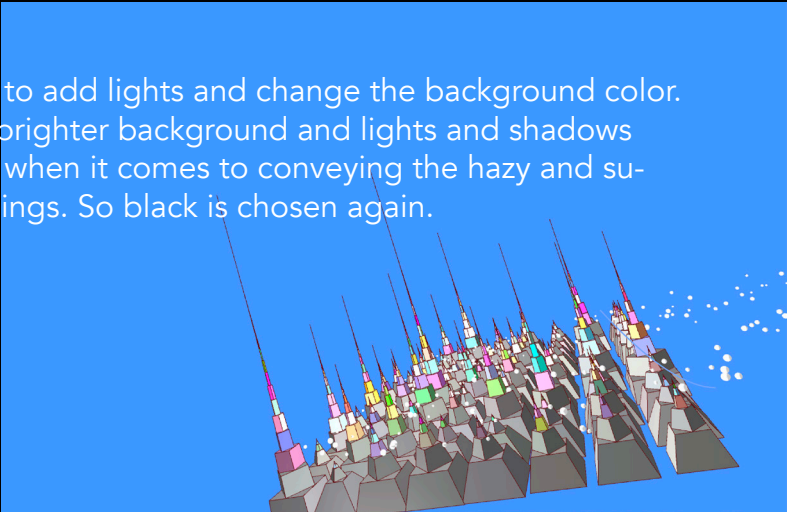


# Iterative Process

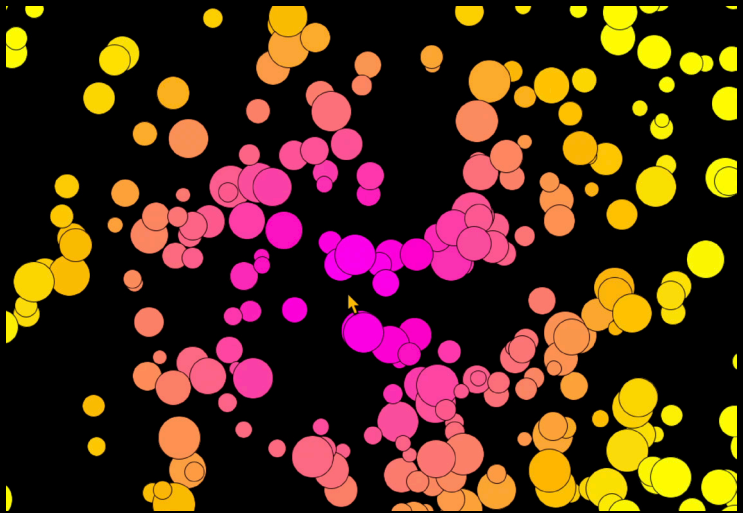
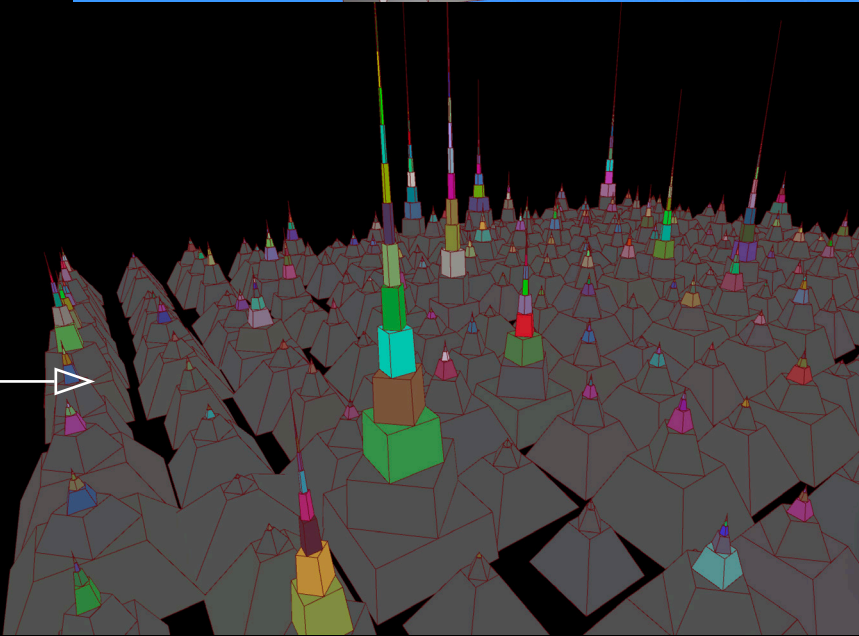


From one stack to 100 stacks, from white to random colors to parameterized colors, and finally adding mouse interaction (camera position changes according to mouseX and mouseY).

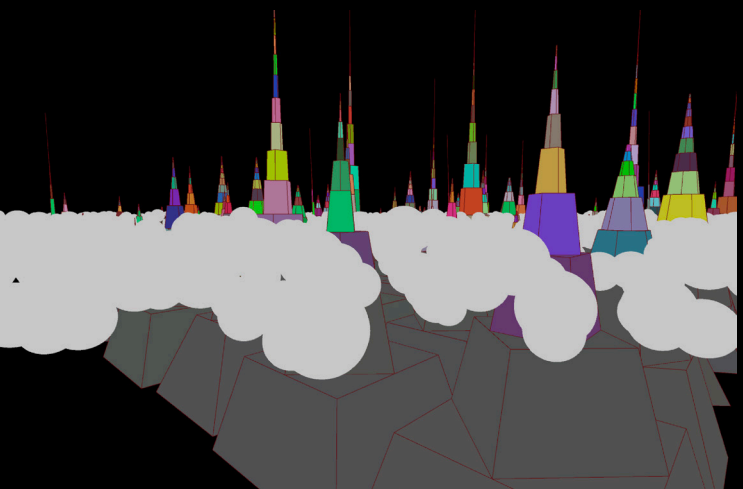
I attempted to add lights and change the background color. However, a brighter background and lights and shadows work poorly when it comes to conveying the hazy and suppressed feelings. So black is chosen again.



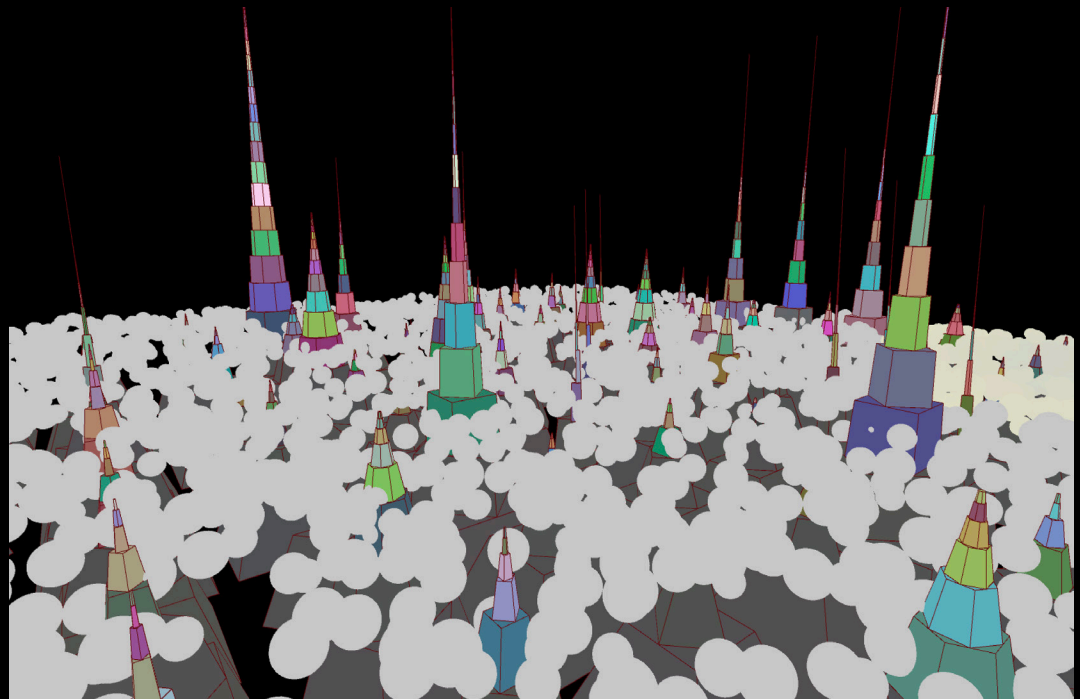
Adding particle system



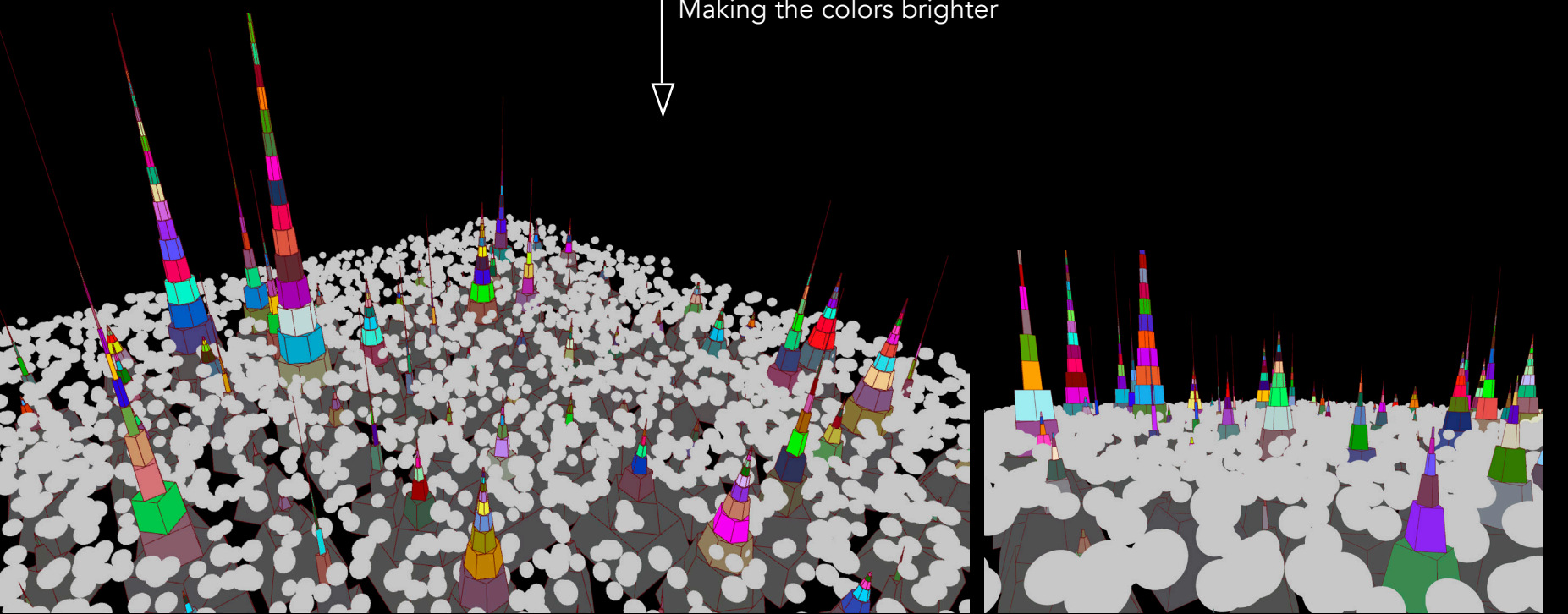
Result



2D Prototype & 3D Prototype



Making the colors brighter



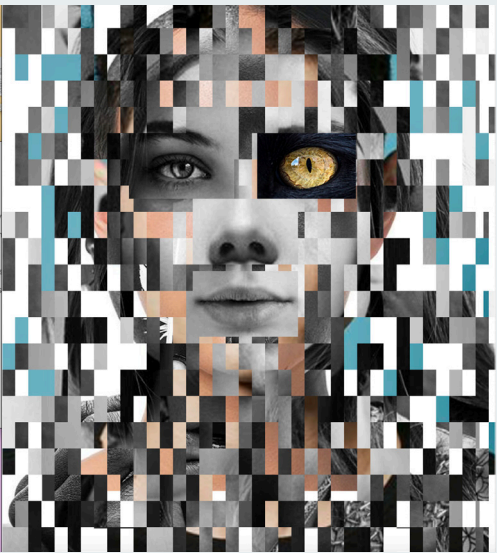
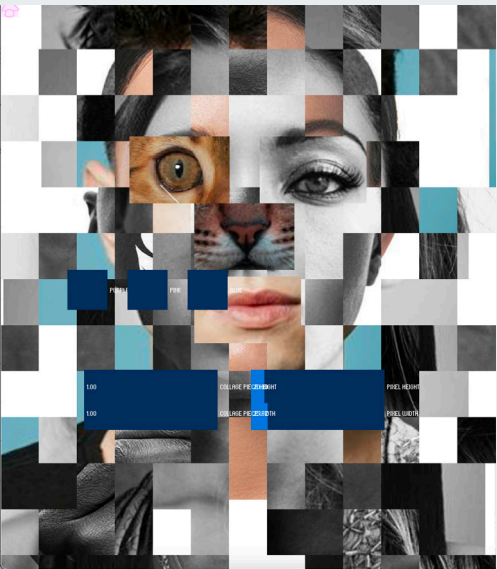
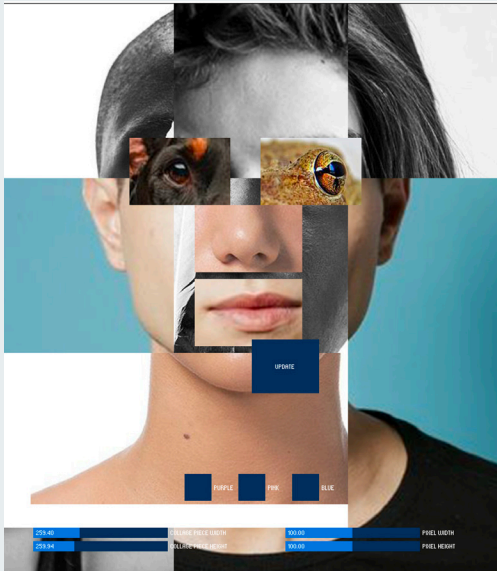
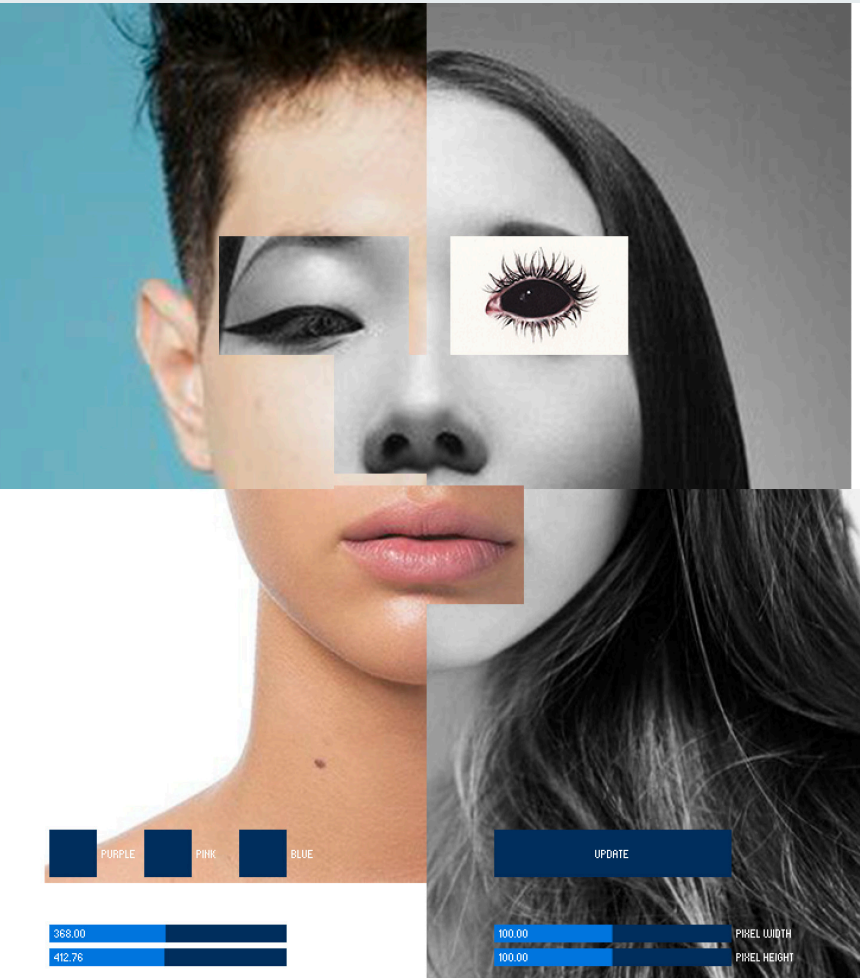


# Surrealism Generator

## Transformation

## Concept Statement

The purpose of designing this project is to help surrealism artists, find inspirations. Drawing inspiration from Exquisite Corpse, which is a common way of looking for inspirations among Surrealism artists, this sketch can generate endless different forms of dreamy and absurd portraits with the user interacting with it.

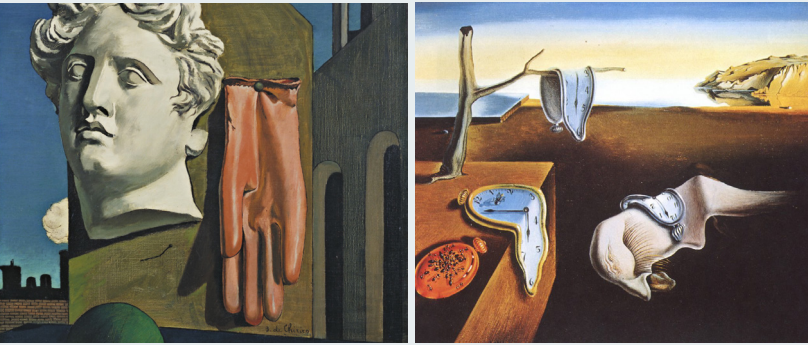


## Visual Research

The original idea of this project comes from the “Exquisite Corpse”. It is a game where artists each draw a body part and then conceal and pass it to the next person. It gained popularity among Surrealism artists during the 1920s, since it is a good way to generate inspirations collaboratively.



Cadavre Exquis with Valentine Hugo, André Breton, Tristan Tzara, Greta Knutson 1933.



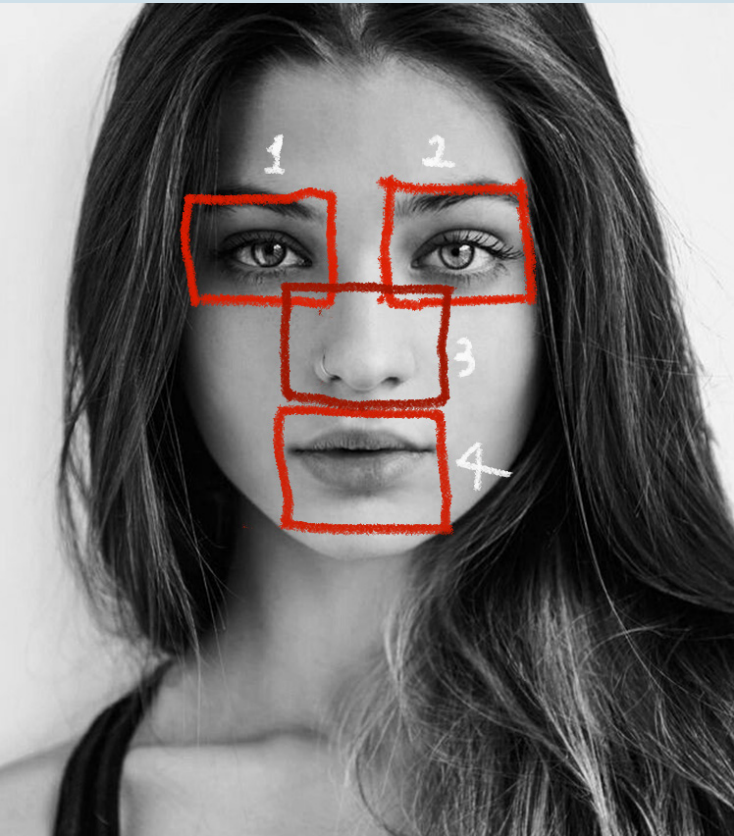
By researching some famous surrealism paintings, there is this characteristic that there are lots of recombination of parts of body or objects.

Furthermore, since I specifically hope to make my project in a more Pop-surrealism style, I looked for famous Pop-surrealism artists' works. Different from more traditional Surrealism, Pop-surrealism comes with more pastel colors such as pink, creating a high contrast between the childlike visual style and the dark themes.





# Generative Rules



First, I cut out 4 areas from a portrait's face, which are left eye, right eye, nose, and mouth respectively. I found portraits of people online and manually make sure that their eyes, noses, and mouths all fit into the location of these 4 rectangles.

To make it more fun, I found other kinds of organs, such as animal eyes and noses, illustrations, eyeballs, which can also go into these areas.

Each time the image updates, the 4 regions change into other parts which are pictures stored in the database.

There's 50% chance of getting another human's organ, and 50% chance of getting a random organ from animals or illustrations.



```
if (randBool1()) //50% probability of drawing from another portrait
{
    int which=int(random(0, 7));
    copy(portraits[which], sx, sy, w, h, sx, sy, w, h);
}
else //50% probability of drawing from a animal eye/nose/mouth.
{
    if (i==0) //left eye
    {
        int which=int(random(0, 6));
        copy(left[which], 0, 0, w, h, sx, sy, w, h);
    }
    else if (i==1) //right eye
    {
        int which=int(random(0, 6));
        copy(right[which], 0, 0, w, h, sx, sy, w, h);
    }
    else if (i==2) //nose
    {
        int which=int(random(0, 5));
        copy(nose[which], 0, 0, w, h, sx, sy, w, h);
    }
    else
    {
        int which=int(random(0, 6));
        copy(portraits[which], sx, sy, w, h, sx, sy, w, h);
    }
}

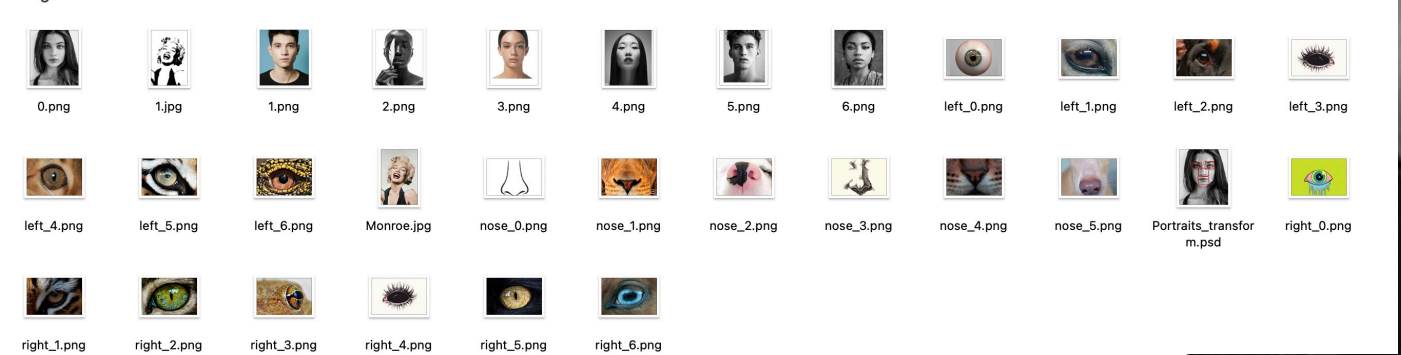
void change_organ_2()
{
    int sx, sy;
    int w=160;
    int h=100;
    for (int i=0; i<4; i++)
    {
        if (i==0)
        {
            sx=193;
            sy=200;
        }
        else if (i==1)
        {
            sx=388;
            sy=200;
        }
        else if (i==2)
        {
            sx=290; //310
            sy=300;
        }
        else
        {
            sx=290;
            sy=410;
        }
    }
}
```



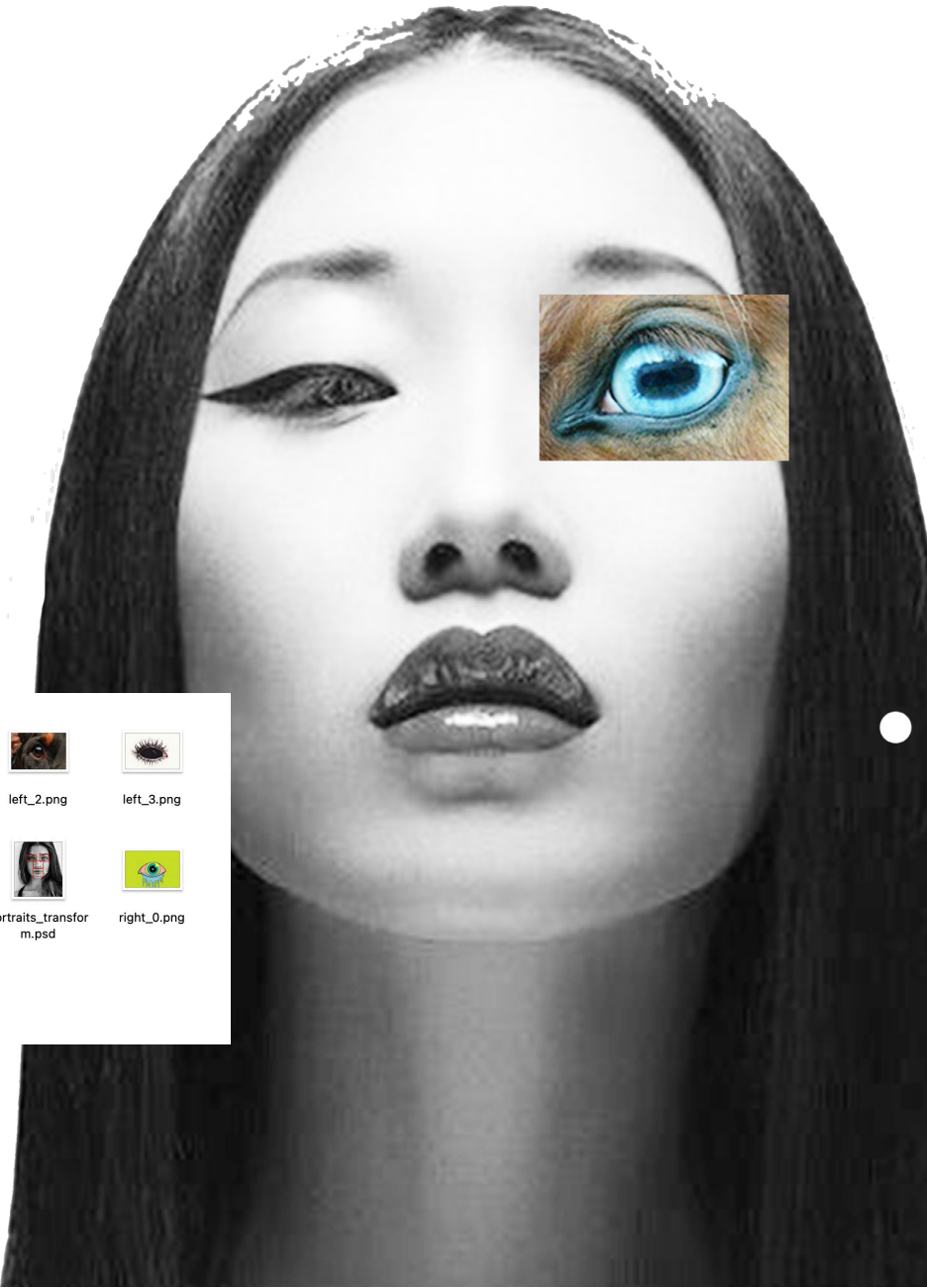
And then, transform the whole screen depending on the user's input:

In each piece, randomly choose a picture from the database, and copy this part of that picture to the screen.

The width of each piece.

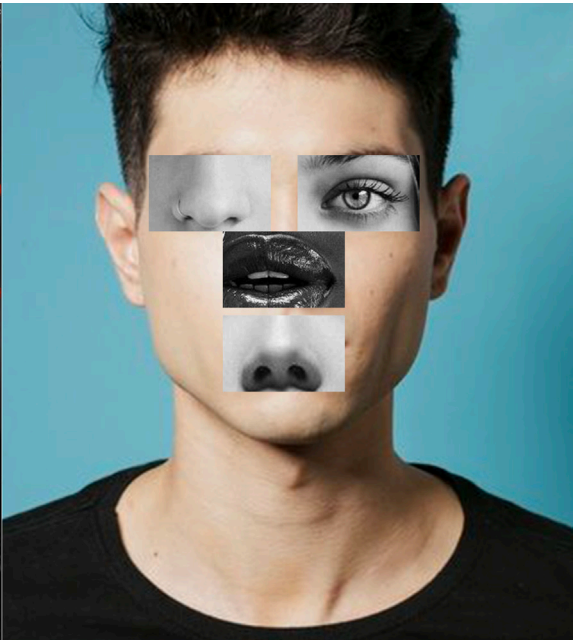


The height of each piece.

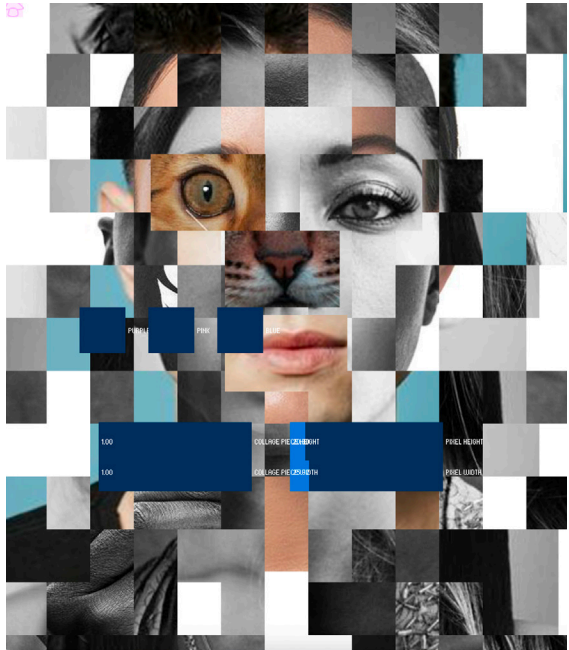




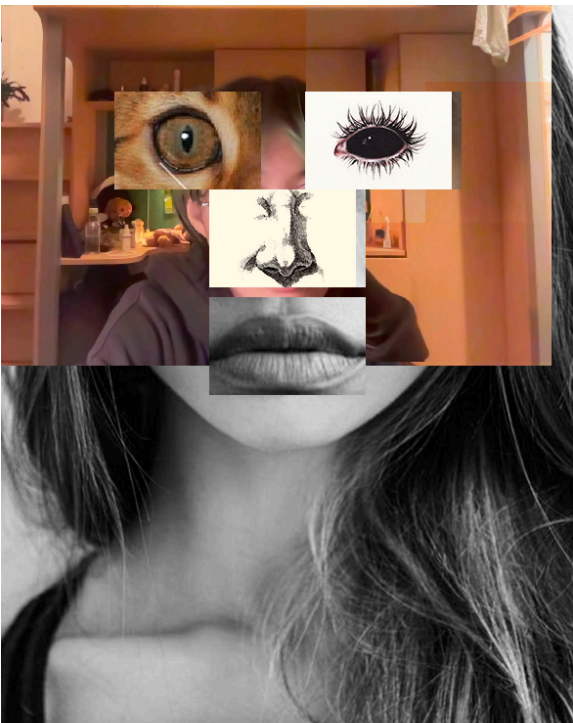
# Iterative Process



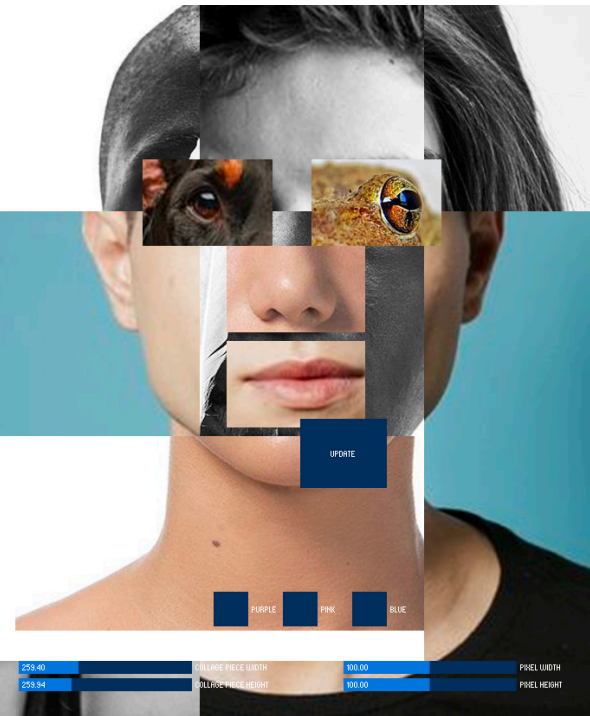
At first, I wanted the nose/eyes/mouth to switch places, but it doesn't look ideal.



Making the user interface.

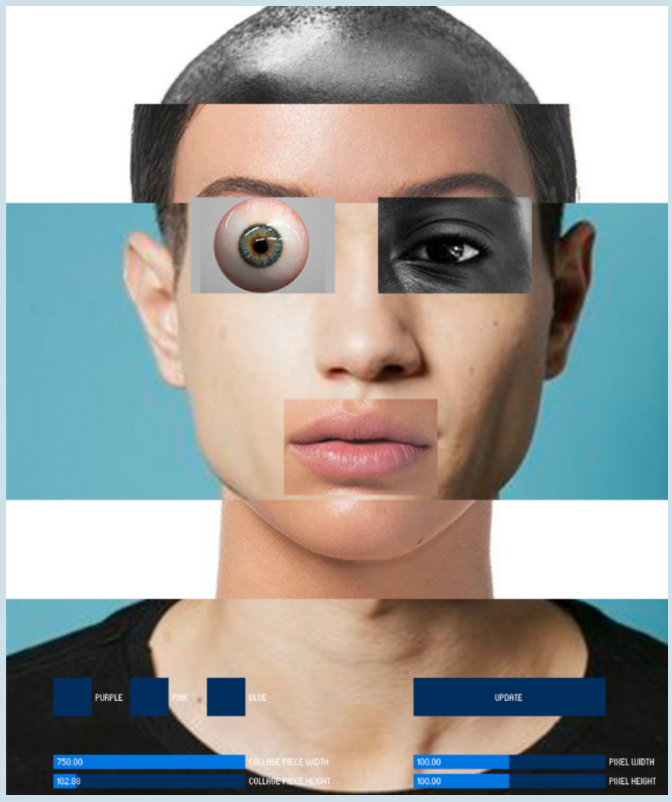
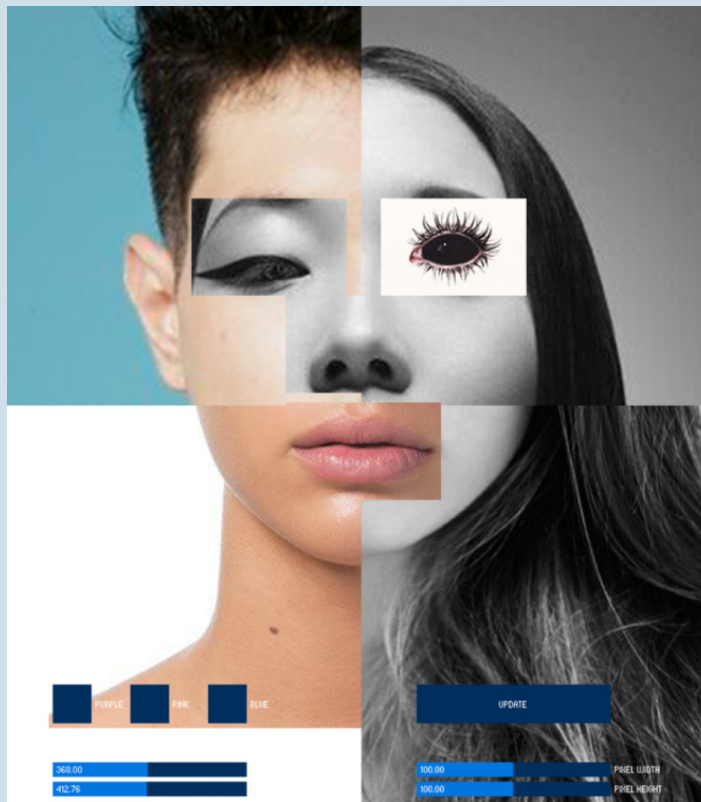
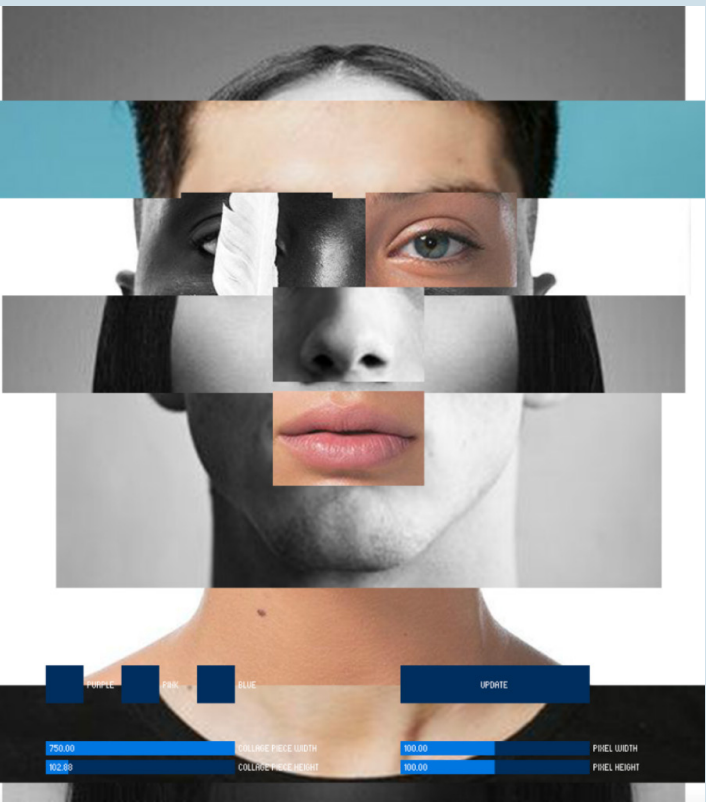
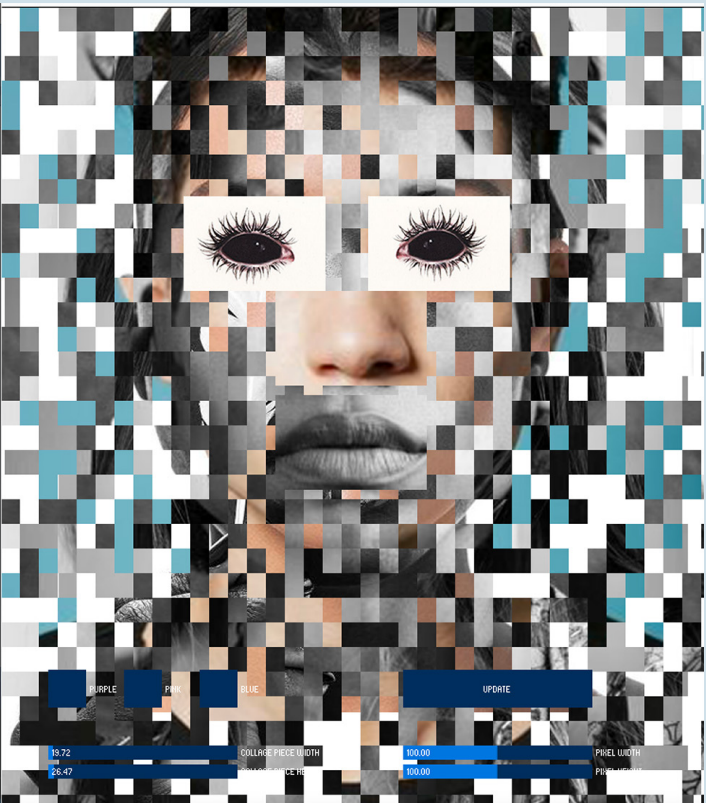


Adding camera to interact also looks bad.



Final result.

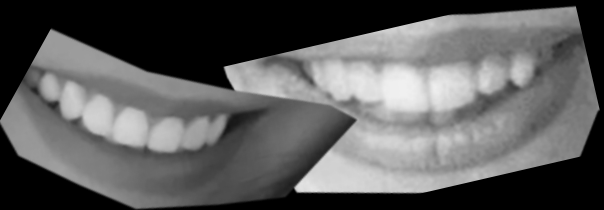
# Final Outcome





# Game: WHY BRACES ???

## Repetition



Concept Statement

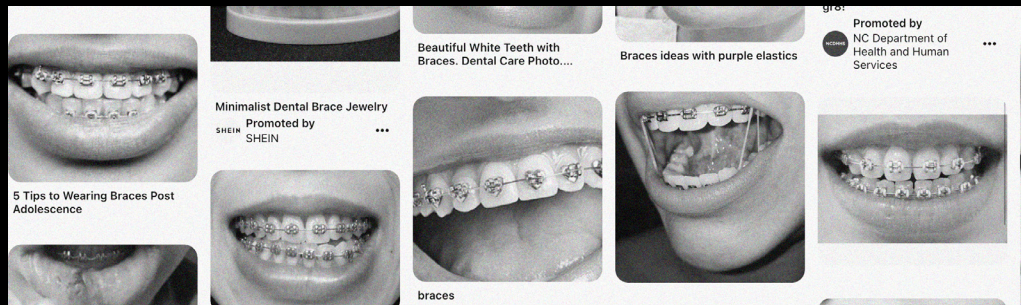
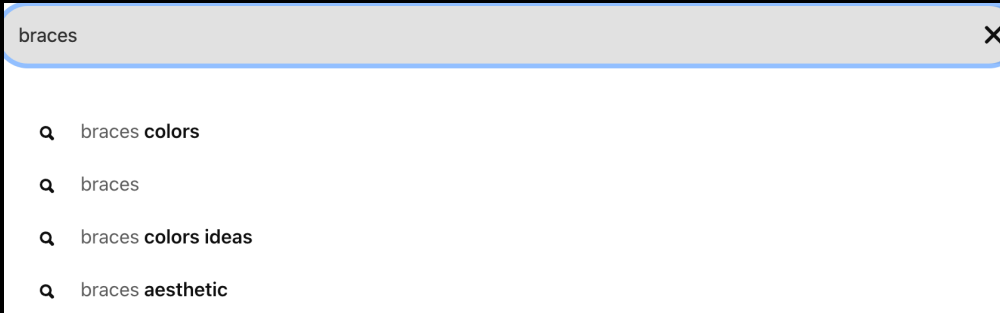
The concept of this game is that braces are symbols of suffering which is caused by the ideas imposed on individuals by the society.

Personally, braces embody the pressure of the social norms that are forced upon varying groups of people. Children are made to get orthodontics treatments because of their parents' will, while parents are also compelled to do so due to society's narrow beauty standard. As the ideas of beauty, health, and how to look normal and fit in are continuously imposed on parents, children, and other adults, more people naturally think of braces as something hopeful and positive, without knowing that this is not even their own ideas.

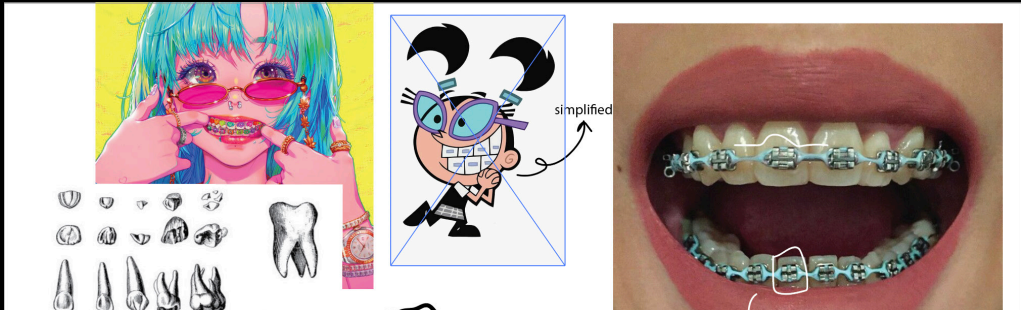
This game visualizes the pain and sense of oppression during orthodontic treatments, as an expression of rebellion against the social notions and suffering which is forced upon us.

Visual Research

When searching about braces, a lot of pictures online are presented cute, beautiful, and happy. Most searched keywords include "braces colors", "braces aesthetic". It occurred to me that a lot of people considers braces themselves beautiful, or simply seek to make the plain iron item look better.



<https://www.pinterestpin/489907265731489722/>

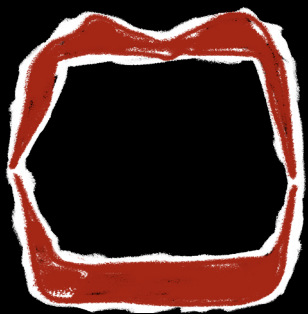


God Save the Queen, by Jamie Reid, 1977.

In order to present a rebellious sense, I researched about punk style images. I noticed that an iconic feature of punk art is the use of collage. For example, one famous artwork, God Save the Queen, transforms the picture and removes the eye and mouth of the queen.



# Generative Rules



## Grid

The grid system is made up of 36 squares, 6 rows and 6 columns. The game board size is 1000x1000 pixles, and the grid system size is 800x800 pixles, placed in the center of the game board. There is a hand-drawn mouth in a scratchy style surrounding the grid system. The visual style corresponds the punk atmosphere, while the mouth indicates where this game takes place.

## Module

In each module, I separated the unit into a smaller 5x5 grid. With this coordinate system, I used beginShape(), endShape(), and curveVertex() to outline the shape of a tooth and a piece of braces (which are both symmetrical).

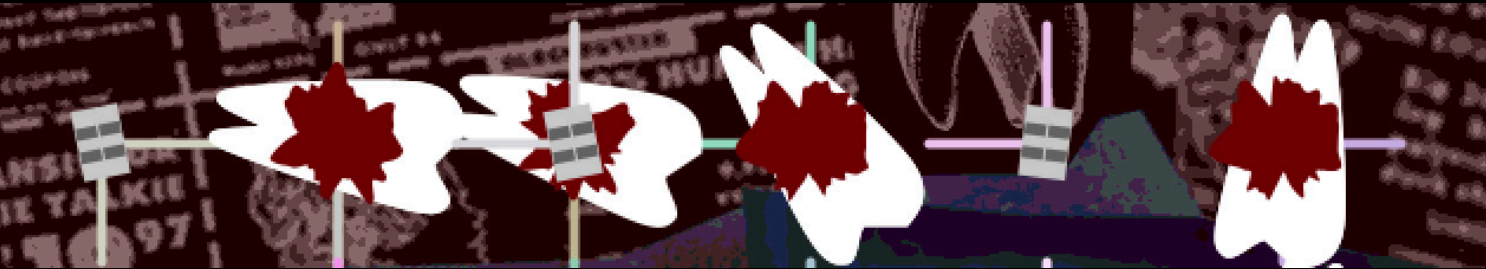
```
stroke(255);
fill(255);
beginShape();
curveVertex(int(-0.75*s_unit),int(-2.5*s_unit));
curveVertex(0,-2*s_unit);
curveVertex(int(0.75*s_unit),int(-2.5*s_unit));
curveVertex(int(1.5)*s_unit,int(-1.5*s_unit));
curveVertex(int(0.5*s_unit),int(2.5*s_unit));
curveVertex(int(0*s_unit),1.5*s_unit);
curveVertex(int(-0.5*s_unit),int(2.5*s_unit));
curveVertex(int(-1.5*s_unit),int(-1.5*s_unit));
curveVertex(int(-0.75*s_unit),int(-2.5*s_unit));
curveVertex(0,-2*s_unit);

endShape(CLOSE);
```

Code drawing the outline of a tooth.



```
for(int j=200;j<800;j+=unit)
{
    int recordX=int((i-200)/unit);
    int recordY=int((j-200)/unit);
    if(randomBool())
    {
        //draw_teeth(i,j,random(0,TWO_PI));
        draw_teeth(i,j,map(noise(i,j),0,1,0,TWO_PI));
        record[recordX][recordY][0]=1;
        if(randomBool())
        {
            draw_blood(i,j,2,45);
            record[recordX][recordY][1]=1;
        }
        if(randomBool())
        {
            draw_braces(i,j,random(0,TWO_PI));
            record[recordX][recordY][2]=1;
        }
    }
    if(record[recordX][recordY][2]==0)
    {
        draw_braces(i,j,random(0,TWO_PI));
        record[recordX][recordY][2]=1;
    }
}
```

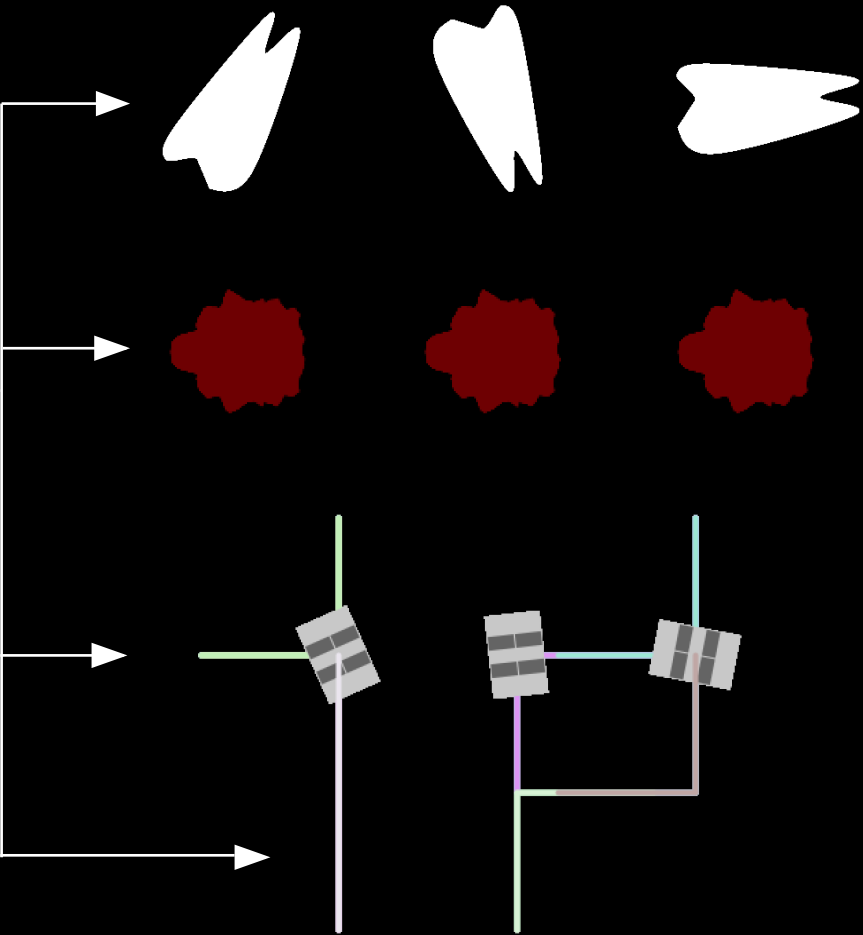


## Generative Repetition

There are four components in each module, each containing some generative features:

- 1. Tooth: rotating angles.
- 2. Blood stain: shape.
- 3. Braces: rotating angles.
- 4. Braces wires: colors, directions.

With the beginning of each round of game, each unit randomly appears its own braces, tooth, and blood stain. There must be a piece of braces in each unit. There is a 20% probability of drawing a teeth in a unit, and if a tooth is drawn, there is a 20% chance of drawing a bloodstain on this tooth.



```
float phaseX=0;
float phaseY=0;
void draw_blood(int X, int Y,int minEd,int maxEd)
{
    float noiseMax=5; //to be changed

    pushMatrix();
    translate(X+int(unit/2),Y+int(unit/2));
    noStroke();
    fill(100,10,10);

    beginShape();
    for(float a=0;a<TWO_PI;a+=0.1)
    {
        float x_offset=map(cos(a+phaseX),-1,1,0,noiseMax);
        float y_offset=map(cos(a+phaseY),-1,1,0,noiseMax);

        phaseX+=0.001;
        phaseY+=0;

        float r = map(noise(x_offset,y_offset),0,1,minEd,maxEd);
        //float r = map(noise(x_offset), 0, 1, 100, 200);

        float x = r*cos(a);
        float y = r*sin(a);

        curveVertex(x,y);

        // x_offset+=0.005;
        // y_offset+=0.003;
    }
    endShape(CLOSE);
    popMatrix();
}
```

### Teeth angles

The functions that draws the teeth has 3 parameters:  
`void draw_teeth(int x, int y, float angle)`

The angle which the tooth rotates around the center, is defined by a two-dimensional noise() function. Since the two parameters of noise() are the x and y coordinate of the tooth module, the angle doesn't change as the game proceed.

e.g.: `draw_teeth(i, j, map( noise(i, j), 0,1, 0, TWO_PI ) );`

When drawing the blood stains on the teeth, the minimum radius of the blob is 2, and the maximum is 45, as the large difference makes it look like blood splashing.



# Game Rules

The game mechanism is as following:

- In each module, when the player operates and it needs to change:
- 1. If there is **only a braces** drawn: draw an extra tooth beneath it.
  - 2. If there is **a tooth and a braces**: remove the braces.
  - 3. If there is **only a tooth**: make no changes.
4. Randomly add new braces wires in random directions.
5. Randomly draw bloodstain on the tooth that appears.

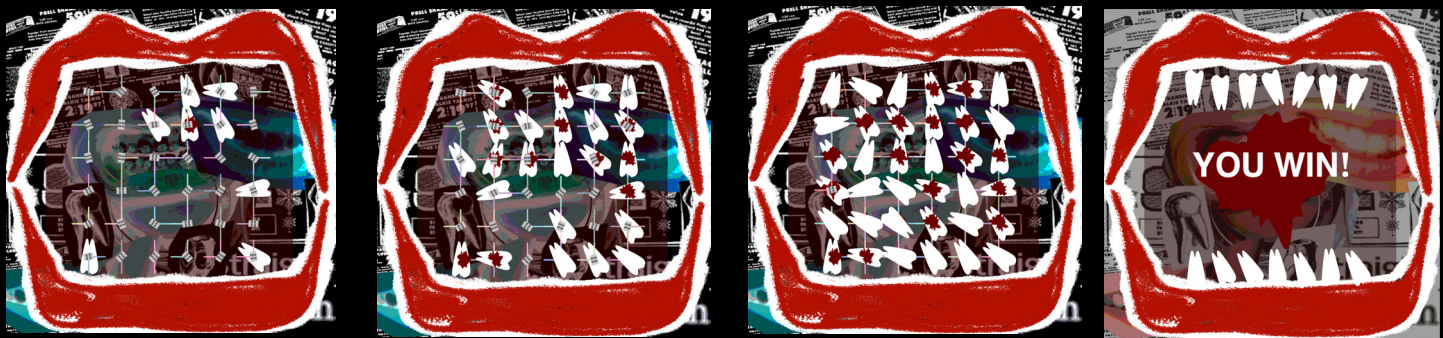
The interactions:

- Players press the keys **1 2 3 Q W E** to interact with the game.
- 1** : controls the 1st and 5th rows of the grid
  - 2** : controls the 2nd, 4th and 6th rows of the grid
  - 3** : controls the 3rd, 6th rows of the grid
  - Q**: controls the 1st and 5th columns of the grid
  - W**: controls the 2nd, 4th and 6th columns of the grid
  - E**: controls the 3rd, 6th columns of the grid

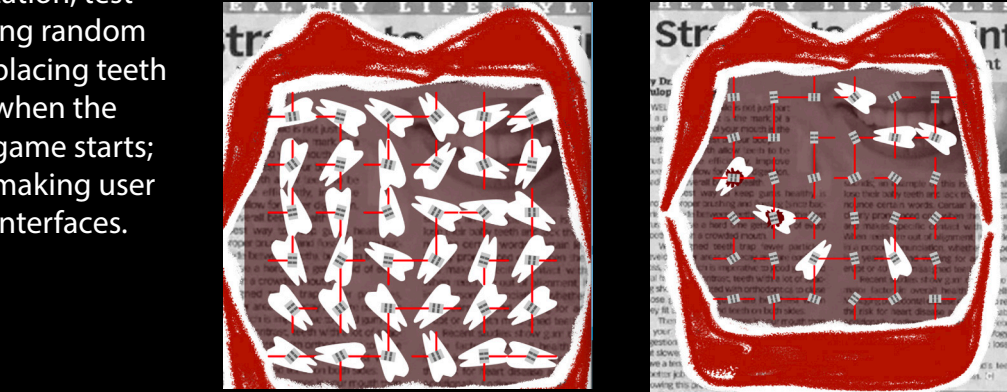
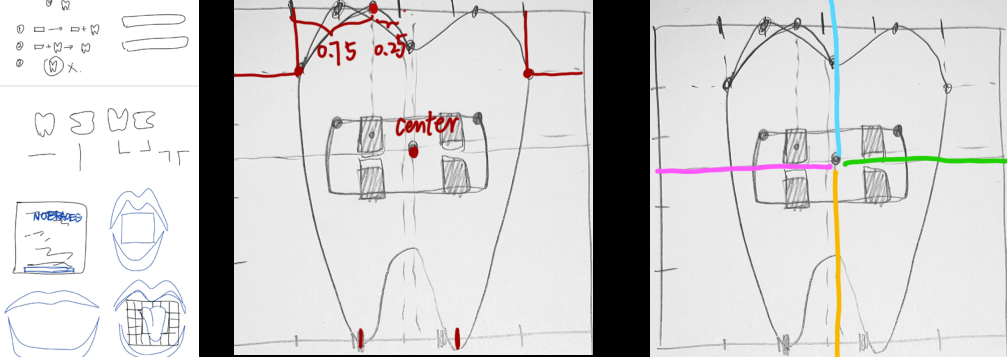
When pressing each key, each module in the corresponding rows or columns has a 80% chance to be changed. When it needs to change, it changes according to the rules explained above.

```
void update1() //1 and 5 in a row.
{
    for (int i=200; i<800; i+=unit)
    {
        for (int j=200; j<800; j+=4*unit)
        {
            int recordX=int((i-200)/unit);
            int recordY=int((j-200)/unit);
            if (record[recordX][recordY][0]==0 && record[recordX][recordY][2]==1 && randomBool2()) //only braces:
            {
                //draw_teeth(1,j,random(0,TWO_PI));
                draw_teeth(1, j, map(noise(1, j), 0, 1, 0, TWO_PI));
                record[recordX][recordY][0]=1;
                if (randomBool3()) //!!! Randomly draw blood.
                {
                    draw_blood(1, j, 2, 45);
                    record[recordX][recordY][1]=1; //Blood_drawn = true.
                }
                draw_braces(1, j, random(0, TWO_PI));
            }
            else if (record[recordX][recordY][0]==1 && record[recordX][recordY][2]==1 && randomBool2()) //tooth and braces.
            {
                //draw_teeth(1,j,random(0,TWO_PI));
                draw_teeth(1, j, map(noise(1, j), 0, 1, 0, TWO_PI)); //Now: use teeth to cover up braces and blood.
                record[recordX][recordY][2]=0;
                record[recordX][recordY][1]=0; //Blood_drawn = false.
                if (randomBool3()) //!!! Randomly draw blood.
                {
                    record[recordX][recordY][1]=1; //Blood_drawn = true.
                    draw_blood(1, j, 2, 45);
                }
            }
        }
    }
}
```

# Final Outcome



# Iterative Process



# WHY BRACES

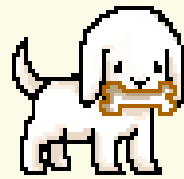
**1 2 3**  
**Q W E** Try pressing these buttons on your keyboard.

# HINT

To win the game, you need to eliminate all the braces on the teeth.

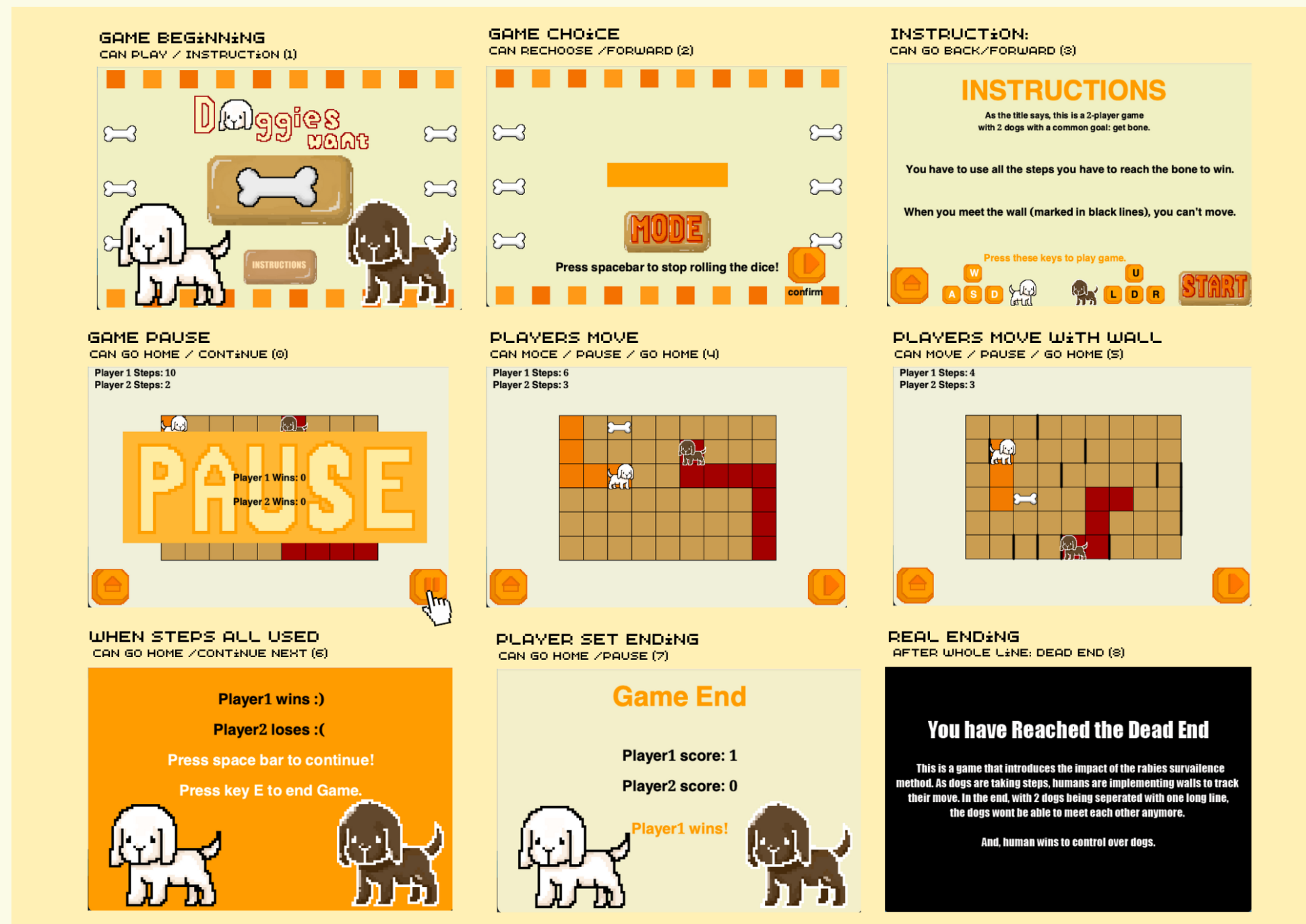


# Doggies Want Bone



## Game Design

Collaboration with Yuri Park



## Concept Statement

Doggies Want Bone is a two-player pixel game. It is in fact 3 games combined into one game: the 2 players' relationship are randomly determined, so they play with different rules in the game. There are 3 kinds of relationships, or game modes, can be Collaboration, Competition, and Predation. In collaboration, players need to meet each other; in competition, players compete to reach the bone placed somewhere on the grid first; in predation, one player try to catch the other, all with one condition: use exactly how many steps they are assigned. In short, the core rule of this game is that the players need to use up all their steps to win the game.

The concept of this game is to discuss the relationships between dogs and dogs, and dogs and humans. When one round is finished, the players go to the next round, where the computer generates random restrictions based on the players' paths in the previous round. The players can't pass the restrictions now. This is a metaphor for human building walls based on dogs' usual paths to prevent rabies. When there is one wall that completely separates the 2 dogs, they reach the hidden ending, where humans win over dogs. This can also be a metaphor for other social events, where stronger powers can deeply influence weaker ones.

## Generative Rules

### Inspiration

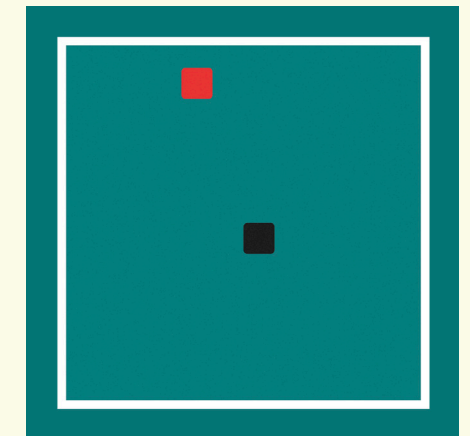
We are inspired by 2 differnt things:

1. Stray dogs' relationship with humans.
2. Maze game.

For the game theme and visuals, we are first inspired by human-dog relationships: When a human is chasing a stray dog, the dog doesn't know the intention of the human: he/she could want to just share food, or could want to harm the dog. We originally intended to have two players each playing the human and dog respectively, and randomly generate their gameplay relationship. We hope to use food to indicate their relationship, however, we found that this idea isn't very reasonable for dogs and humans. Therefore, we decided to change it into 2 dogs: there are 3 kinds of relationships, each represented by whether they are holding a bone or not.

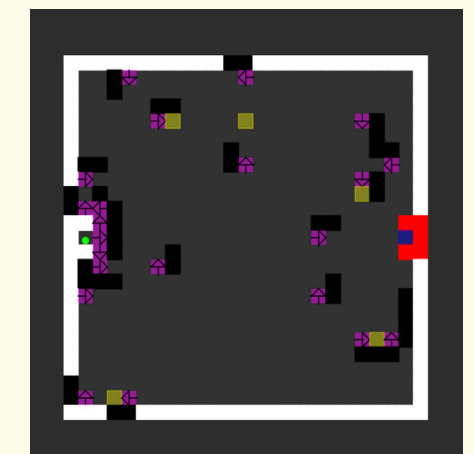


Isabella, keyMoveMaze



oscar, snake

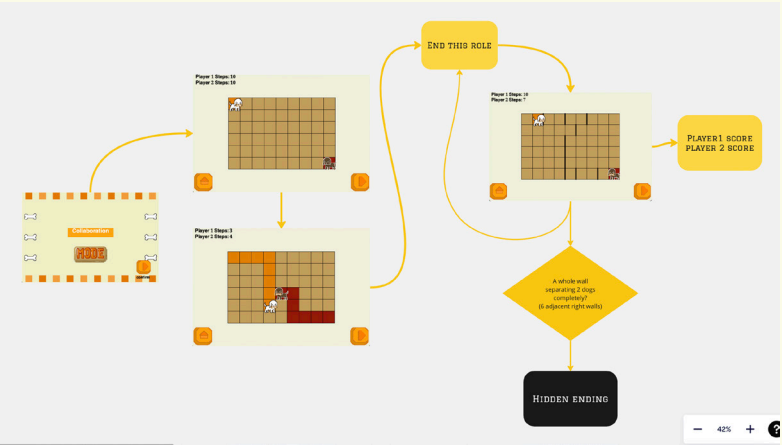
For the game mechanism, we are inspired by some classic pixel games and maze games. We investigated the following games on OpenProcessing: <https://openprocessing.org/sketch/1879889> Snake game, in which the player moves by pixel. <https://openprocessing.org/sketch/1670736> Two-player game. <https://openprocessing.org/sketch/1378209> 2D game grid, with restrictions. <https://openprocessing.org/sketch/1936690> Making maze. The player leaves marks on the path that is covered.



Glenn Noland, different kind of platformer



Starting Point & Objectives



The game starts with random game mode generator: the players click the “MODE” button, and then the 3 different game modes start rolling. When they press the space bar, the animation stops and shows them their chosen game mode (Collaboration, Competition, and Predation).

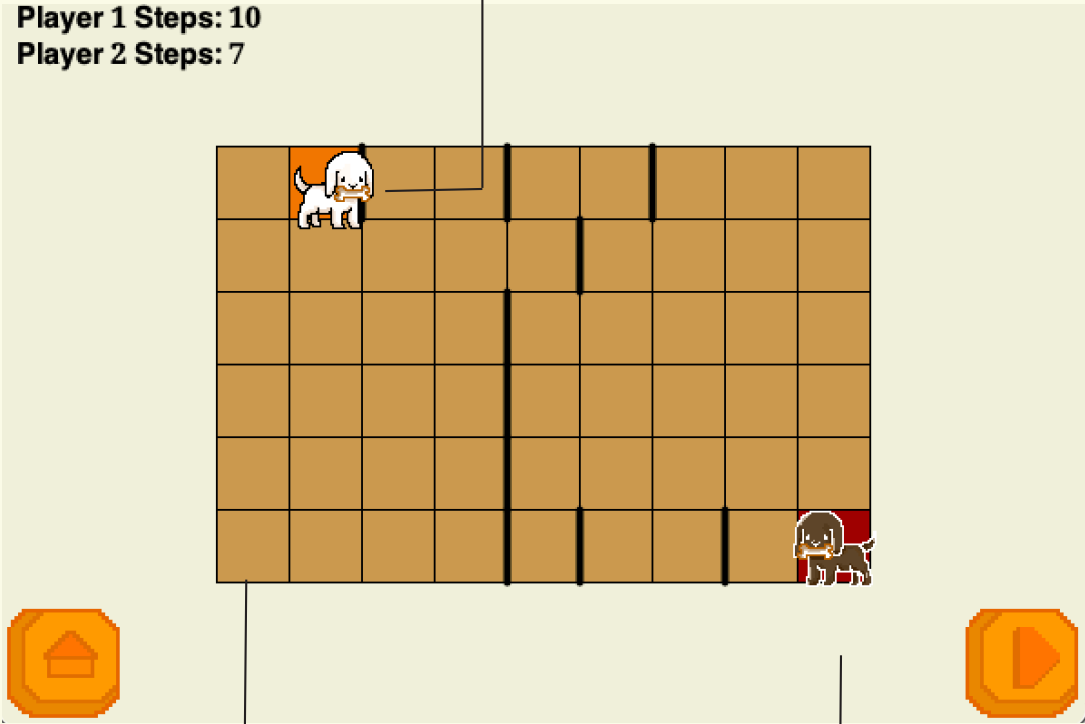
Aftering confirming the mode, the players starts the game with each player standing on a corner of the grid. Depending on the game mode, the play-ers need to either meet, compete for a bone, or try to catch/stay away from the other player.

When they both use up all their steps, the current round of game ends and records which player wins. The winner accumulates 1 point. Going onto the next round, the computer generates random restrictions based on the players’ paths in the pre-vious round.

After each round, the players can either choose to end game, and the player with more scores win; or they can keep on playing until they reaching the hidden ending.

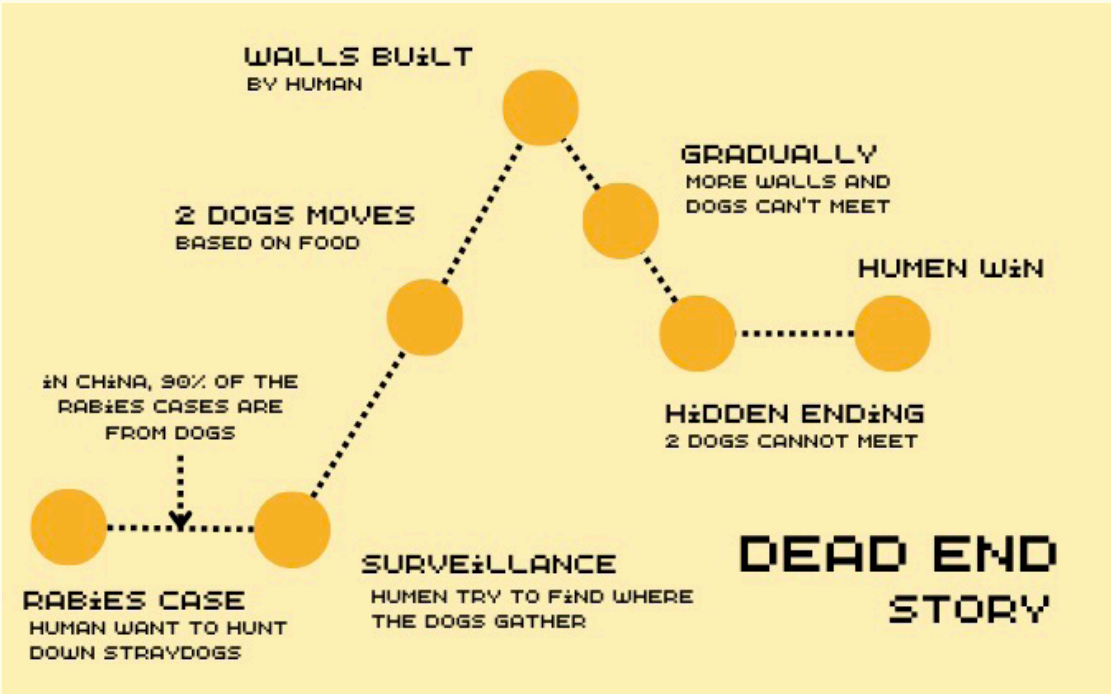


Game Rules



Players can go back to their previous steps. The step count will increase back up.

Narrative Arc



Loops & Code

In this project, Yuri coded the game mechanics (Recording player’s position variables, stepped paths, making corresponding grid, etc.); I coded the game user interfaces, drew the pixel art illustrations, and integrated the build walls game mechanics into the UI/UX code.

```
void resetBoard() {
    arrayCopy(tiles,previousTiles);
    boolean[][] previousStep = new boolean[9][6];

    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            boolean temp=previousTiles[i][j].steppedOn;
            previousStep[i][j]=temp;
        }
    }

    println("-----Stepped On:-----");
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            print(previousStep[i][j]);
            print(" ");
        }
        println();
    }

    // Reset players to random positions
    int randomCoLP1 = floor(random(1, 4));
    int randomRowP1 = floor(random(6));
    int randomCoLP2 = floor(random(5, 9));
    int randomRowP2 = floor(random(6));
    int randomCoLP3 = floor(random(1, 9));
    int randomRowP3 = floor(random(6));

    player1.x = 120 + (randomCoLP1 * 40) + 20;
    player1.y = 80 + (randomRowP1 * 40) + 20;
    player2.x = 120 + (randomCoLP2 * 40) + 20;
    player2.y = 80 + (randomRowP2 * 40) + 20;
}

void chooseGameMode_rollDice()
{
    fill(#ffa613);
    noStroke();
    rect(width/2-100,height/2-40,200,40);

    fill(0);
    textFont(GameModeFont);
    textAlign(CENTER);

    modeImgCnt++;
    if(modeImgCnt%10==0)
    {
        modeImgIdx=(modeImgIdx+1)%3;
        if(modeImgIdx==0)
            text("Collaboration",width/2,height/2-15);
        else if(modeImgIdx==1)
            text("Competition",width/2,height/2-15);
        else
            text("Predation",width/2,height/2-15);
    }

    fill(0);
    textFont(instrFont1);
    textAlign(CENTER);
    text("Press spacebar to stop rolling the dice!",width/2,340);
}

ArrayList<Character> moveHistory = new ArrayList<Character>();

void handlePlayerSnapMovement() {
    println("At Start - Move History Size: " + moveHistory.size());
    int startX = 140;
    int startY = 120;
    int endX = startX + 40 * 8;
    int endY = startY + 40 * 4;

    Character move = null;

    int col = floor((player1.x - 120) / 40);
    int row = floor((player1.y - 80) / 40);

    if (moveHistory.size() > 0) {
        println("Within Condition - Move History Size: " + moveHistory.size());
        println("Within Condition - Player steps: " + player1.steps);
        Character lastMove = moveHistory.get(moveHistory.size() - 1);

        // Assuming there are some other checks for other directions, you can add them here

        if ((key == 'D' || key == 'd') && lastMove == 'A') {
            if (player1.x < endX && !tiles[col][row].rightWall) {
                player1.x += 40;
                move = 'X';
            }
        }

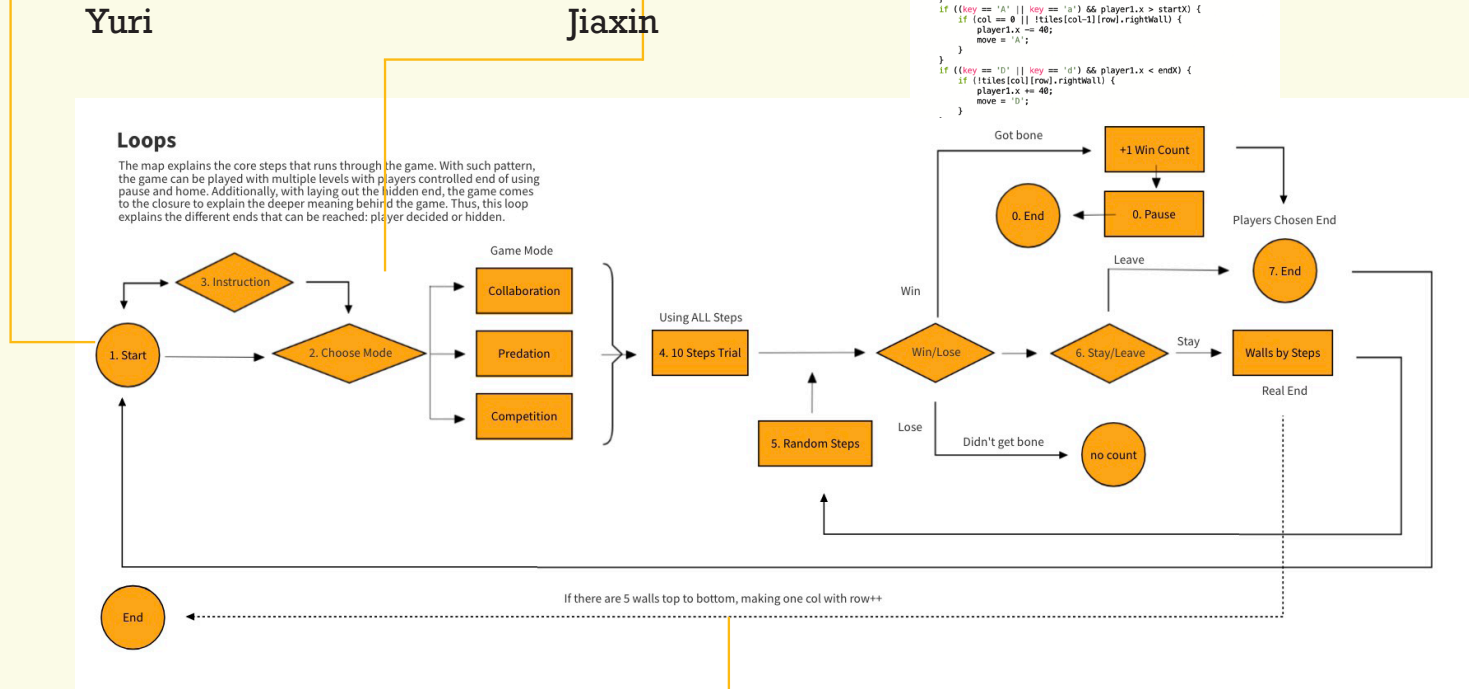
        if ((key == 'A' || key == 'a') && lastMove == 'D') {
            if (player1.x > startX && col == 0 || !tiles[col-1][row].rightWall) {
                player1.x -= 40;
                move = 'D';
            }
        }

        if ((key == 'U' || key == 'u') && player1.y > startY) {
            player1.y -= 40;
            move = 'U';
        }

        if ((key == 'S' || key == 's') && player1.y < endY) {
            player1.y += 40;
            move = 'S';
        }

        if ((key == 'L' || key == 'l') && player1.x < startX) {
            if (col == 0 || !tiles[col-1][row].rightWall) {
                player1.x -= 40;
                move = 'L';
            }
        }

        if ((key == 'R' || key == 'r') && player1.x < endX) {
            if (!tiles[col][row].rightWall) {
                player1.x += 40;
                move = 'R';
            }
        }
    }
}
```



```
void draw() {
    if(page.equals("start"))
        startPage();

    else if(page.equals("Instructions"))
        instructionPage();

    else if(page.equals("chooseGameMode"))
        chooseGameMode();

    else if(page.equals("chooseGameMode_rollDice"))
        chooseGameMode_rollDice();

    else if(page.equals("chooseGameMode_chosen"))
        chooseGameMode_chosen();

    else if(page.equals("game_start"))
        setup_game();
        setup_transition();
        page="game";
}

void drawTiles() {
    int startX = 120;
    int startY = 80;
    int rectWidth = 40;
    int rectHeight = 40;
    strokeWeight(1);
    stroke(0);

    for (int row = 0; row < 6; row++) {
        for (int col = 0; col < 9; col++) {
            stepMode tile = tiles[col][row];

            // Set tile fill color based on the tile's steppedBy attribute
            if (tile.steppedBy.equals("player1")) {
                fill(#e38100);
            } else if (tile.steppedBy.equals("player2")) {
                fill(#951a0c);
            } else {
                fill(#c3915d);
            }
            strokeWeight(1); // Thin grid
            rect(startX + col * rectWidth, startY + row * rectHeight, rectWidth, rectHeight);
        }
    }

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 6; j++) {
            stepMode tile = previousTiles[i][j];

            strokeWeight(4); // Thicker walls

            if (tile.rightWall) {
                line(startX + (i + 1) * rectWidth, startY + j * rectHeight,
                    startX + (i + 1) * rectWidth, startY + (j + 1) * rectHeight);
            }
        }
    }
}

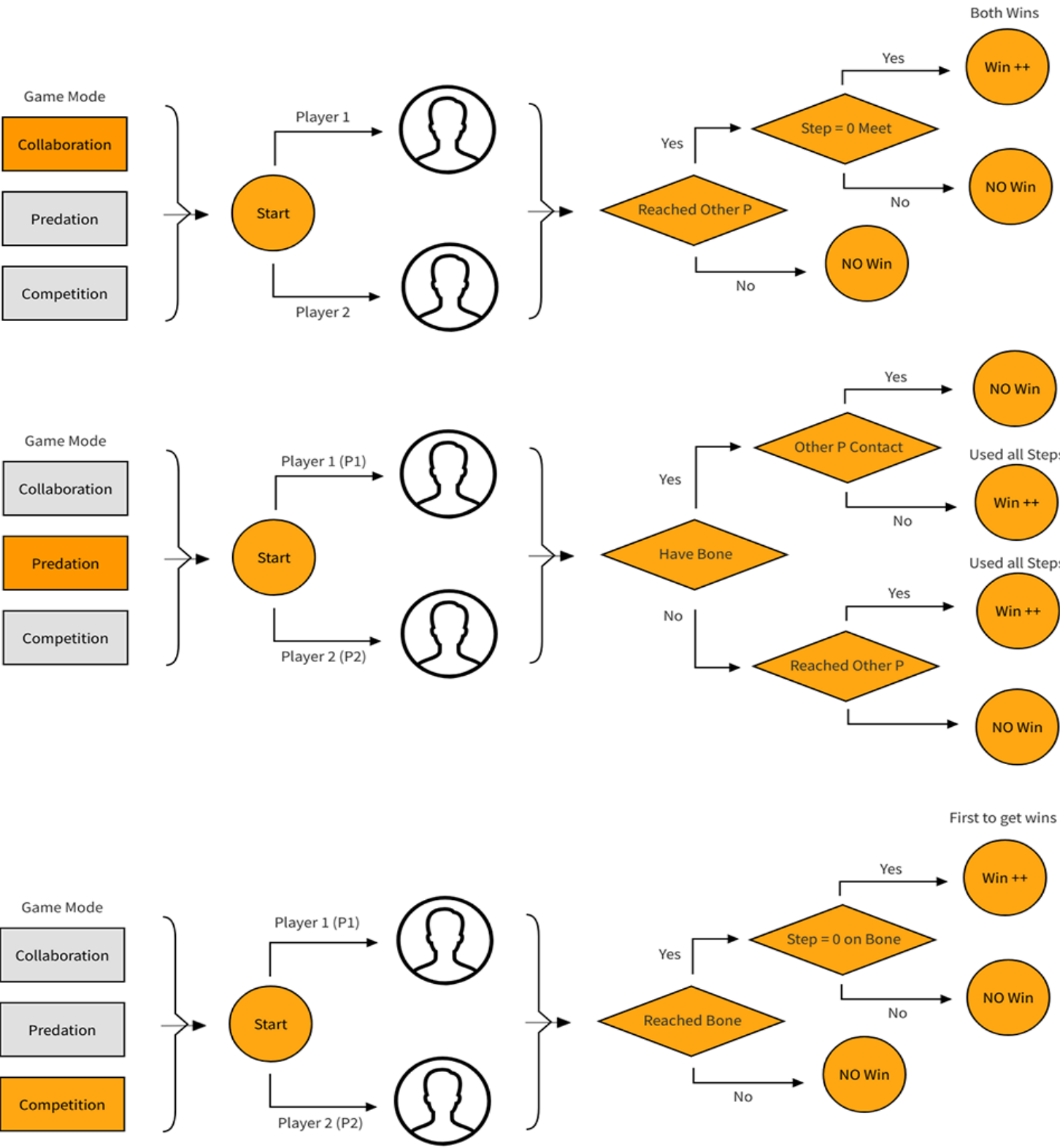
boolean gameOver()
{
    for(int col=0; col<9; col++)
    {
        println("col="+col);
        for(int row=0; row<6; row++)
        {
            if(tiles[col][row].rightWall==true)
            {
                if(row==5)
                {
                    println("this is true");
                    return true;
                }
            }
            else
            {
                print(tiles[col][row]);
                print(" ");
                continue;
            }
        }
    }
    return false;
}
```



# Core Game Mechanics

## Game Mechanics

The map explains the core steps that runs through the game. With such pattern, the game can be played with multiple levels with players controlled end of using pause and home. Additionally, with laying out the hidden end, the game comes to the closure to explain the deeper meaning behind the game. Thus, this loop explains the different ends that can be reached: player decided or hidden.



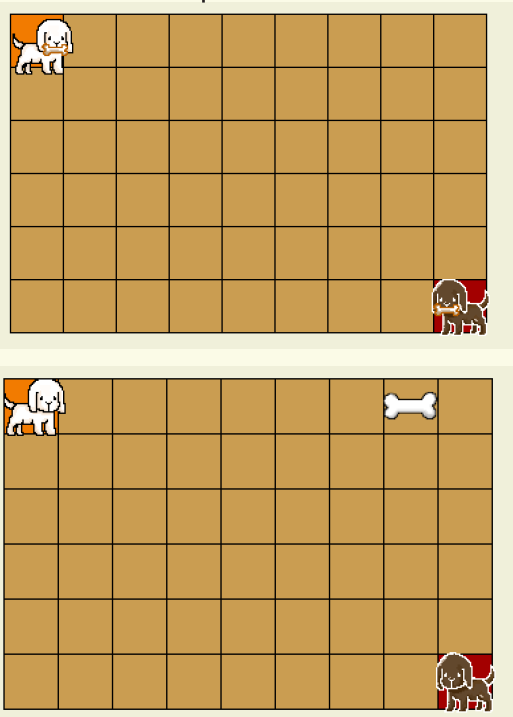
# User Tests

## Collaboration

When 2 players both have 0 steps left && 2 players are standing on 2 tiles next to each other, both of them win.

```
if(gameMode=="Collaboration")
{
    placeWhiteDog(wDog_bone,player1.x-20,player1.y-20);
    placeBlackDog(bDog_bone,player2.x-23,player2.y-25);

    if(player1.steps==0 && player2.steps==0)
    {
        if(abs(player1.x-player2.x+player1.y-player2.y)==40)
        {
            p1win=true;
            p2win=true;
            page="end";
            endGame();
        }
        else
        {
            page="end";
            endGame();
        }
    }
}
```



## Competition

When one player has 0 steps left && stands on the bone before the other player, this player wins this round.

```
else if(gameMode=="Competition")
{
    placeWhiteDog(wDog,player1.x-20,player1.y-20);
    placeBlackDog(bDog,player2.x-23,player2.y-25);
    placeBone(boneLocation);

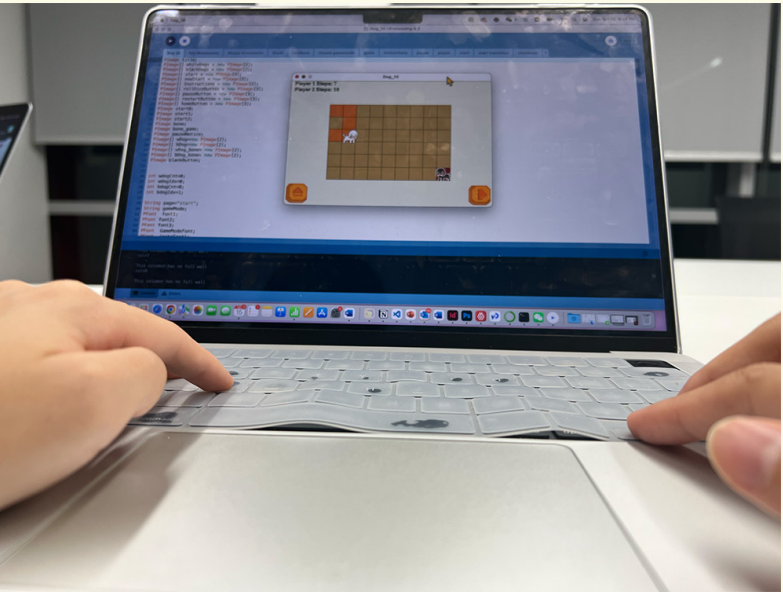
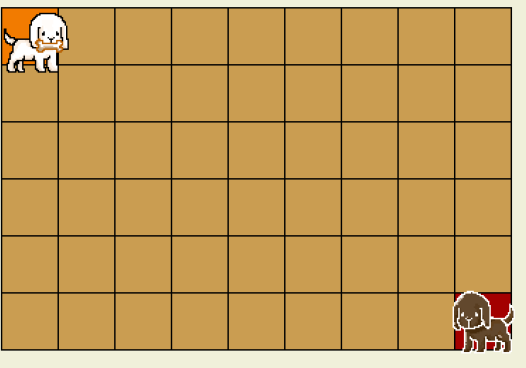
    if(player1.steps==0 && player1.x==boneLocation.x && player1.y==boneLocation.y && !(player2.x==boneLocation.x && player2.y==boneLocation.y))
    {
        p1win=true;
        p2win=false;
        page="end";
        endGame();
    }
    if(player2.steps==0 && player2.x==boneLocation.x && player2.y==boneLocation.y && !(player1.x==boneLocation.x && player1.y==boneLocation.y))
    {
        p1win=false;
        p2win=true;
        page="end";
        endGame();
    }
    //Players both use up their steps, but neither reach the bone.
    if(player1.steps==0 && player2.steps==0 && !(player2.x==boneLocation.x && player2.y==boneLocation.y) && !(player1.x==boneLocation.x && player1.y==boneLocation.y))
    {
        //Print in if player1.steps==0 && player2.steps==0
        {
            //Print in if player2.x==boneLocation.x && player2.y==boneLocation.y;
            //Print in if player1.x==boneLocation.x && player1.y==boneLocation.y;
            page="end";
            endGame();
        }
        if (p1win) {
            player1Wins++;
        }
        if (p2win) {
            player2Wins++;
        }
    }
}
```



## Predation

```
else
{
    placeWhiteDog(wDog_bone,player1.x-20,player1.y-20);
    placeBlackDog(bDog,player2.x-23,player2.y-25);

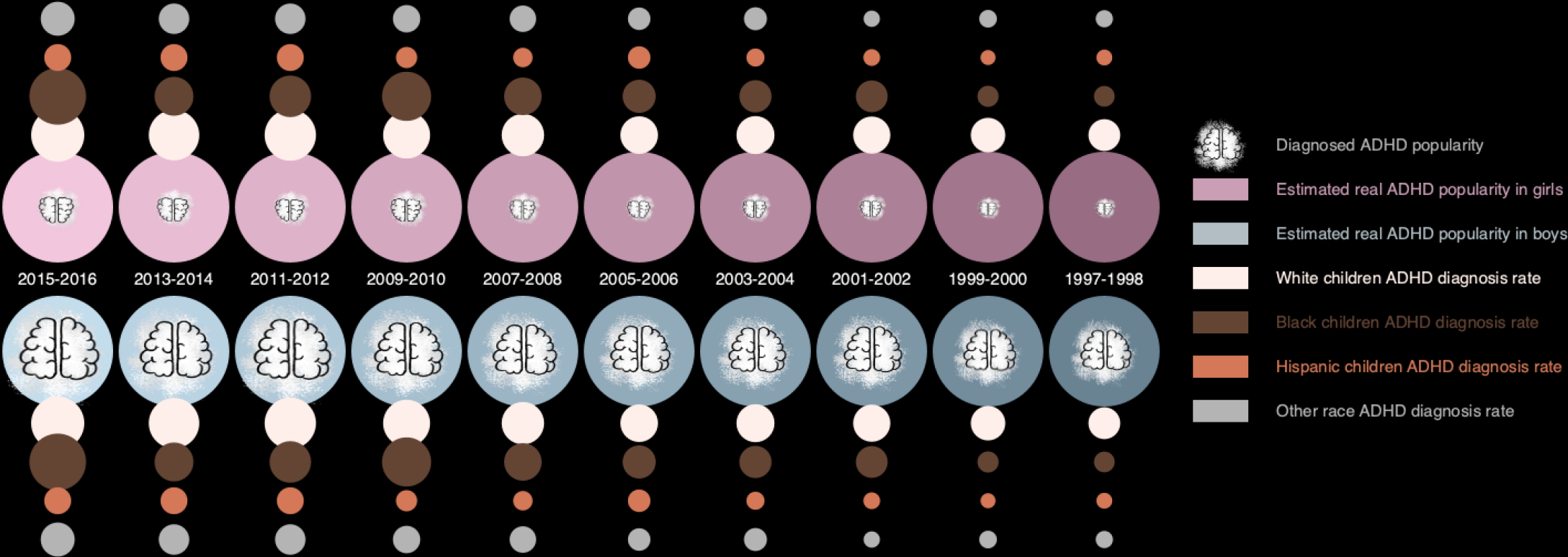
    if(abs(player1.x-player2.x+player1.y-player2.y)==1)
    {
        p1win=false;
        p2win=true;
        page="end";
        endGame();
    }
    else if(player1.steps==0 && player2.steps==0)
    {
        p1win=true;
        p2win=false;
        page="end";
        endGame();
    }
}
```





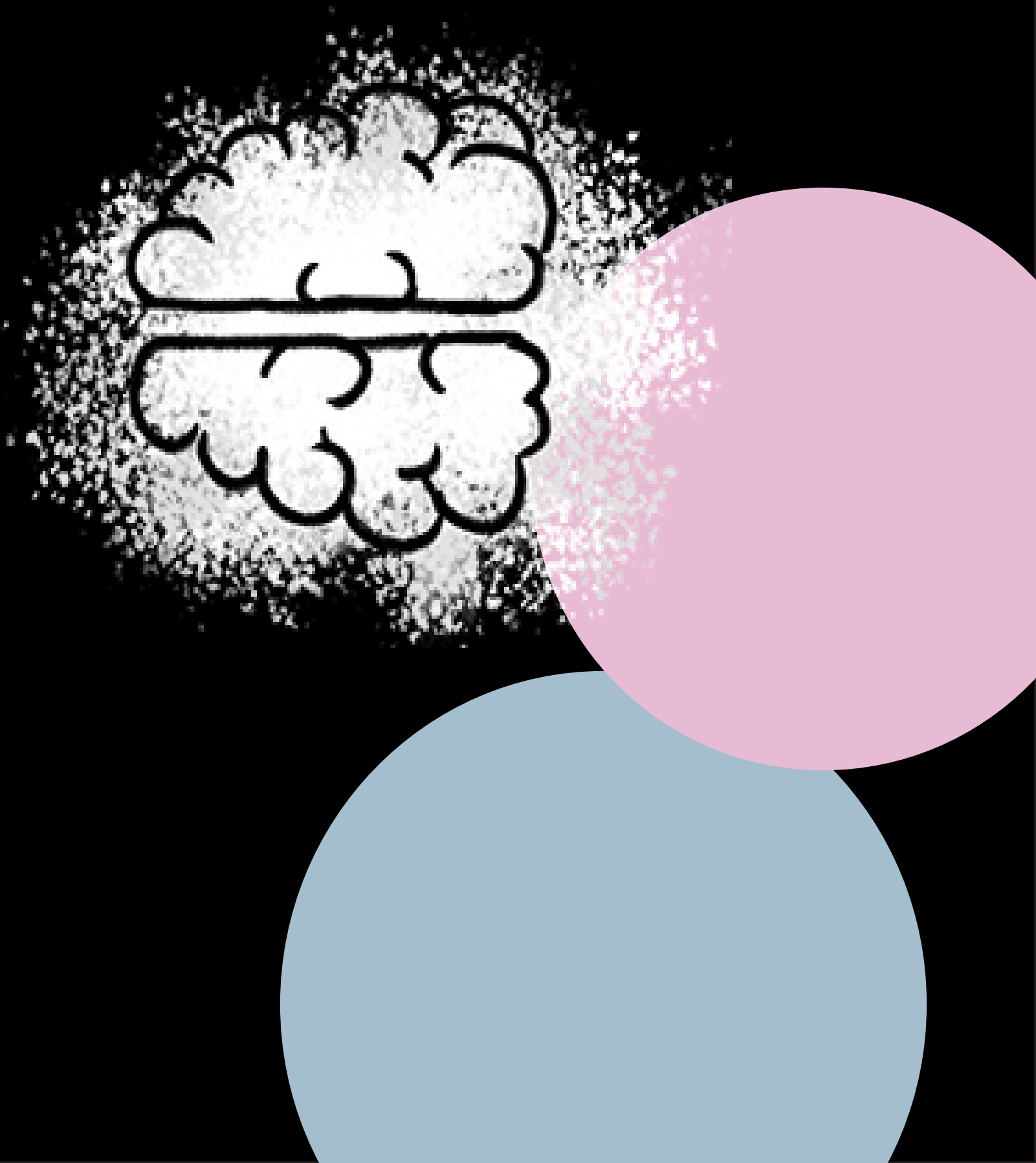
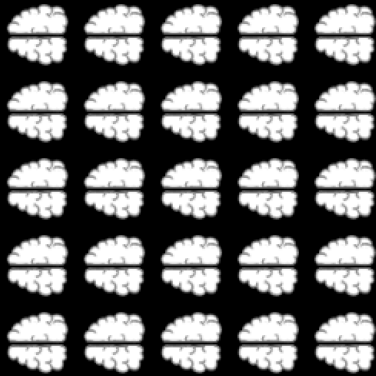
# ADHD Data Visualization

## 01 ADHD Ratios Among General Public



## 02 Data Among ADHDers

Among the children who have ADHD 51.5 percent had behavioral or conduct problems, 32.7 percent had anxiety problems, 16.8 percent had depression, 13.7 percent had autism spectrum disorder.





# Concept Statement

ADHD is not an illness stranger to the public, however, the diagnosis of ADHD is not something as easy as it may seem. In fact, the diagnosis rate of ADHD is deeply influenced by different factors, such as gender, ethnicity, family economic conditions, etc. In fact, a lot of people are left in the darkness never knowing that they have ADHD, especially for minority groups, which further causes them problems.

In this project, I visualize the ADHD data among children 2-17 years old, so it shows people directly the problem of ADHD diagnosis and the other problems that are closely linked to ADHD.

# Generative Rules

## Dataset & Source

The data I use comes from the CHADD website: <https://chadd.org/about-adhd/general-prevalence/>.

The data that I focused on is primarily from the analysis of data from the National Health Interview Survey (NHIS), (Xu et al, 2018). By cleaning the data and extracing what I need, I get the following 2 .csv files:

| Age            | 2015-2016 | 2013-2014 | 2011-2012 | 2009-2010 | 2007-2008 | 2005-2006 | 2003-2004 | 2001-2002 | 1999-2000 | 1997-1998 |
|----------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 2-17 years old | 14.0/6.3  | 12.8/5.8  | 13.6/5.4  | 12.2/5.7  | 11.3/4.8  | 10.6/4.2  | 10.2/4.3  | 10.3/4.0  | 9.4/3.4   | 9.0/3.1   |

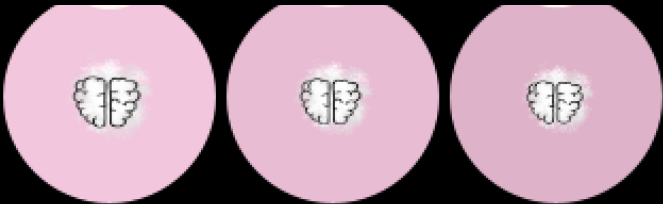
| Ethnicity          | 2015-2016 | 2013-2014 | 2011-2012 | 2009-2010 | 2007-2008 | 2005-2006 | 2003-2004 | 2001-2002 | 1999-2000 | 1997-1998 |
|--------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| Non-Hispanic White | 12        | 11.4      | 11.6      | 10.6      | 9.6       | 8.5       | 8.6       | 8.4       | 7.8       | 7.2       |
| Non-Hispanic Black | 12.8      | 8.8       | 9.4       | 11.1      | 8.5       | 7.5       | 7.3       | 7.2       | 4.8       | 4.7       |
| Hispanic           | 6.1       | 6.1       | 6.1       | 4.8       | 4.4       | 5.1       | 4.1       | 3.8       | 3.4       | 3.6       |
| Others             | 7.7       | 6.9       | 6.9       | 6.2       | 6         | 5.1       | 5.2       | 3.7       | 4         | 3.9       |

I also used the data which is about the treatment and co-occurring sytoms.

- **Treatment used** by children ages 2-7 with a **current diagnosis** of ADHD:
  - Two out three were taking medication (*62 percent*).
  - Less than half received behavioral treatment in the past year (*46.7 percent*).
  - Nearly one out of three received a combination of medication and behavioral treatment in the past year (*31.7 percent*).
  - Nearly one out of four had not received any treatment (*23 percent*).
- **Severity of ADHD** among children ages 2-17:
  - 14.5 percent had severe ADHD
  - 43.7 percent had moderate ADHD
  - 41.8 percent had mild ADHD
- **Co-occurring conditions** (children ages 2-17):
  - Two out of three children (*63.8 percent*) had at least one co-occurring condition.
  - Half of all children (*51.5 percent*) had behavioral or conduct problems.
  - One out of three children (*32.7 percent*) had anxiety problems.
  - One out of six children (*16.8 perccent*) had depression.
  - About one out of seven children (*13.7 percent*) had autism spectrum disorder.
  - About one out of 80 children (*1.2 percent*) had Tourette syndrome.
  - One in a hundred adolescents (*1 percent*) had a substance abuse disorder.

## Visualization Strategy

My visualization strategy is inclined to functional and abstract, but I also added decorative and figurative parts, which is basically the brain illustration.



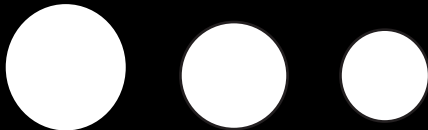
## Mapping Methods

I use the size of ellipses to represent how high the prevalence of diagnosed ADHD in a certain group of people.

```
void vis2()//Ethnicity
{
  noStroke();
  for(int row=0;row<table1.getRowCount();row++)
  {
    TableRow Row=table1.getRow(row);
    if(row==0)
      fill(0);
    else if(row==1)
      fill(0);
    else if(row==2)
      fill(0);
    else if(row==3)
      fill(0);
    //fill(0);
    for(int col=1;col<11;col++)
    {
      float num= Row.getFloat(col);
      ellipse(5*col*distance,midLine+10+row*distance_y,num*4,num*4);
      ellipse(5*col*distance,midLine-10-120-row*distance_y,num*4,num*4);
    }
  }
}
```

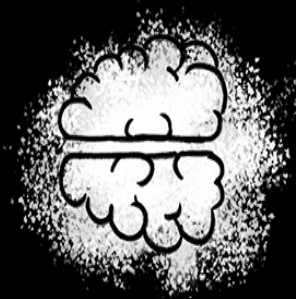
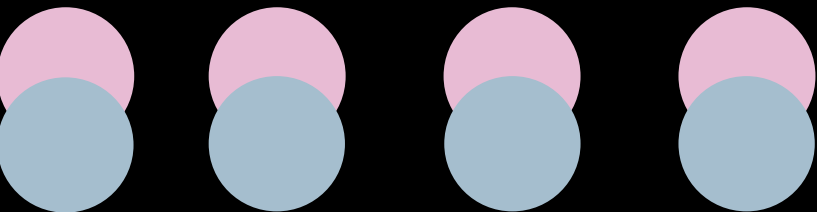
Get the diagnosis ratio from the csv table.

Let the number mutply by 4 to be the radius of thr circle.



It is estimated that 20% of the people who have ADHD are really diagnosed. Therefore, I use the larger circles to represent the estimated real ADHD population. If the whole human population doesn't change dramatically, it is very likely that the real ADHD population throughout years also don't change. Therefore, the radius of the ellipses are the same.

To make viewers instinctively know the gender differences, I use blue to represent boys and pink to represent girls. Although this might be stereotypical, it is a simple color difference that conveys the message.

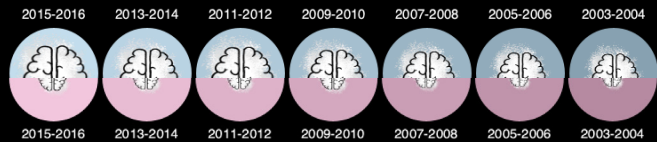


```
brain0=loadImage("Brain_gray_up.png");
brain1=loadImage("Brain_gray_down.png");

String num= Row.getString(col);
String[] nums=split(num, "/");

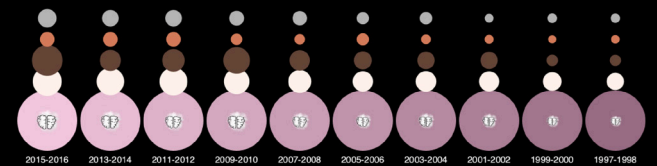
brain0.resize(int(28*float(nums[0])/4),int(28*float(nums[0])/4));
brain1.resize(int(28*float(nums[1])/4),int(28*float(nums[1])/4));
```

Get the prevalence of ADHD in boys and girls respectively 1997-2016 from the first csv file (the number before "/" represent boys while the one after represent girls. Adjust the parameters, so that the size of the brain fits well into the larger circles with the right ratio.



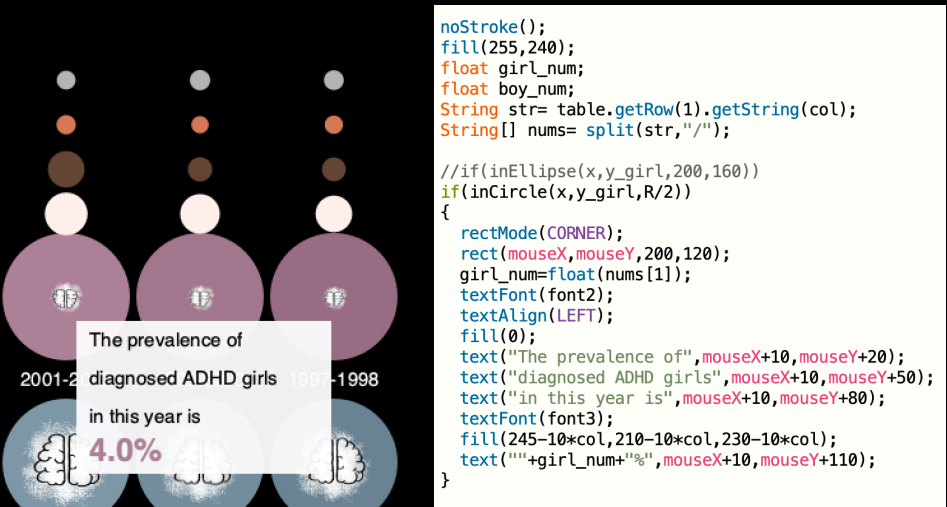
By comparing the diagnosed ADHDer and the whole ADHD population, and comparing the diagnosed population between booy and girls, it is apparent that females have a way lower diagnosis rate than males. The situation improved throughout the years, but there is still obvious gaps between different genders.

Similarly, I visualized the ADHD diagnosis rate among different ethnicities. By using the colors corresponding to different races, I map out the data across the years.

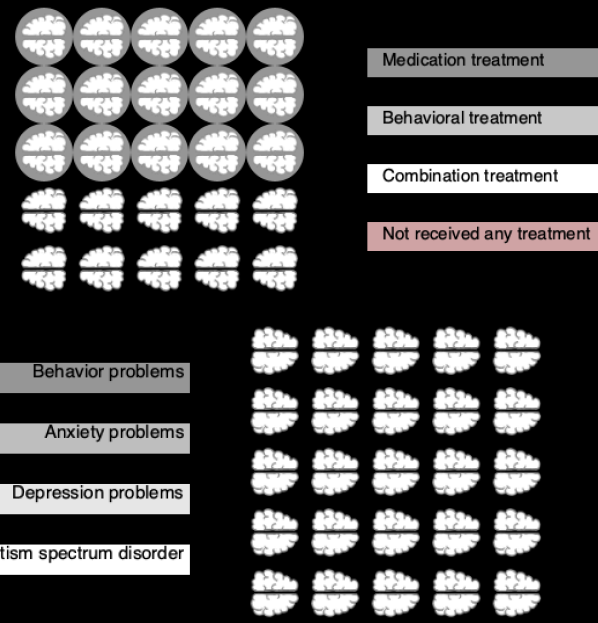




Interaction



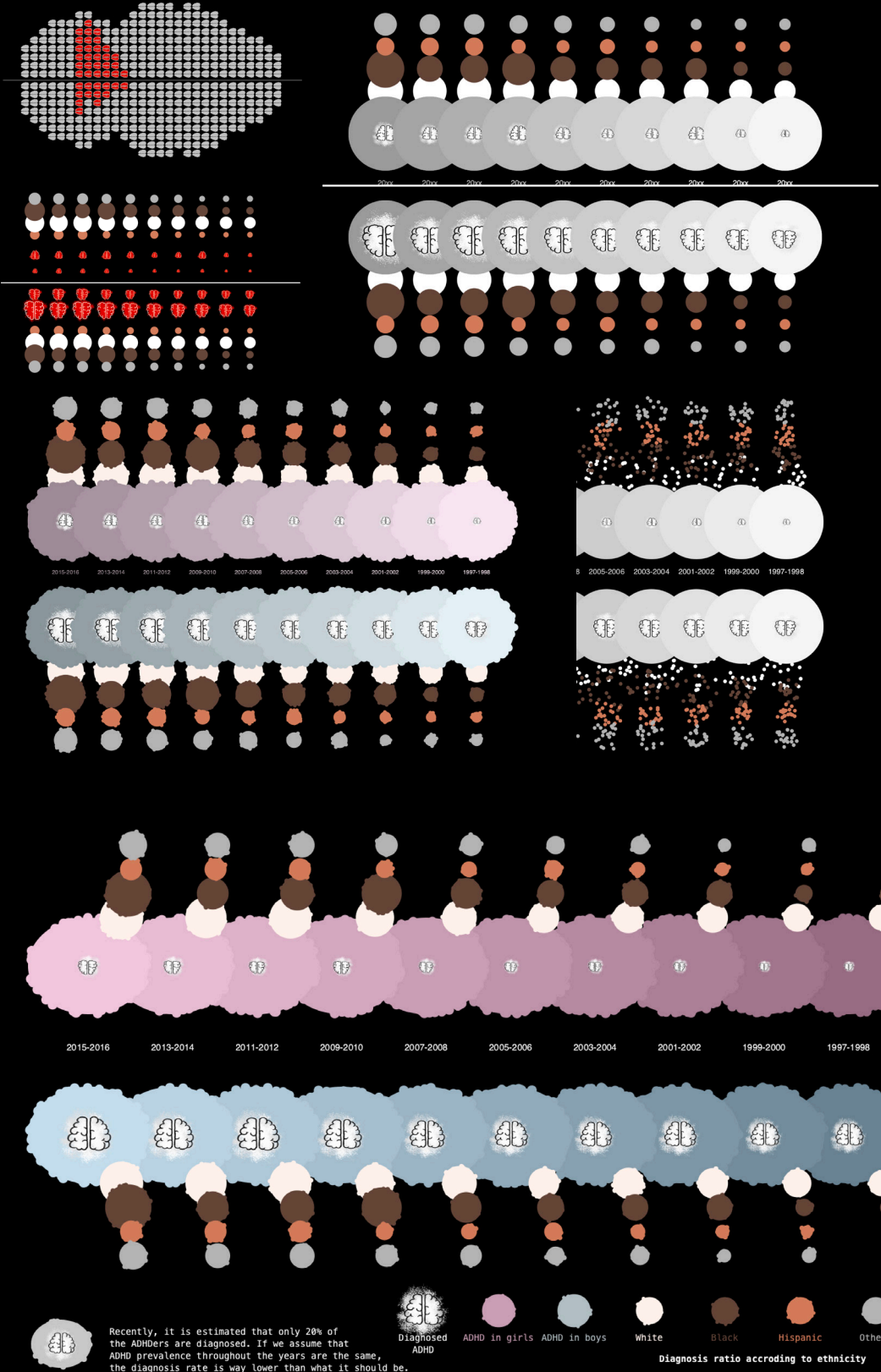
```
void coFill()
{
  //first button.
  if(mouseX>o_sx-4*o_dx-40 && mouseX<o_sx-4*o_dx-40+button_x && mouseY>o_sy+10 && mouseY<o_sy+button_y)
  {
    for(int i=0;i<5;i++)
    {
      for(int j=0;j<2;j++)
      {
        noStroke();
        fill(150);
        ellipse(o_sx+i*o_dx+30,o_sy+j*o_dy,50,50);
        image(brain2, o_sx+i*o_dx+30,o_sy+j*o_dy);
      }
    }
    ellipse(o_sx+30,o_sy+2*o_dy,50,50);
    image(brain2, o_sx+30,o_sy+2*o_dy);
    ellipse(o_sx+o_dx+30,o_sy+2*o_dy,50,50);
    image(brain2, o_sx+o_dx+30,o_sy+2*o_dy);
  }
}
```



For the “Data Among ADHDers”, I used the number of brains highlighted to visualize the rate data. I placed 25 brain illustrations. For instance, there are 62 percent of the diagnosed ADHDers are using medication treatment, then 15 brains are highlighted to represent the 62%. There are corresponding buttons on the side of the brains. When the mouse is hovered over the button, the corresponding data is visualized among the brains.

I use different colors for different groups. Generally, the lower the ratio is, the brighter the gray color is. I specially used red for “Not received any treatment”, because it is different from the other treatment data, and to raise awareness from people.

Iterative Process



Final Outcome

