Experiment 05: Evolutionary Impressions

Chengkun Li

Computational Media, University of California - Santa Cruz

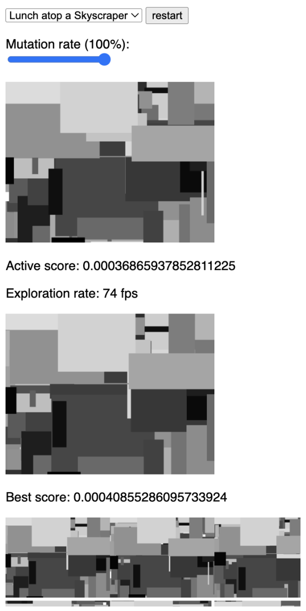
CMPM 147: Generative Design

Prof. Wes Modes

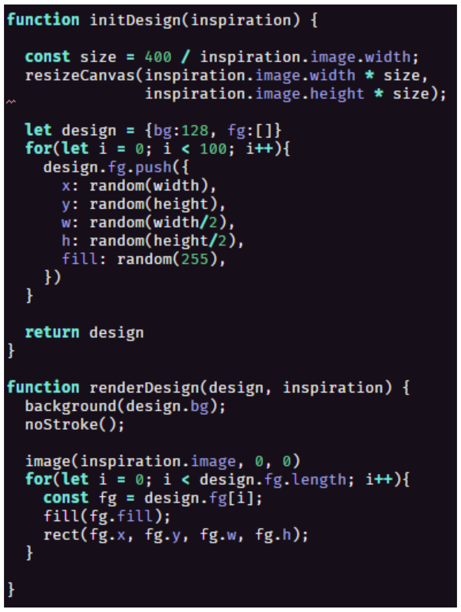
5/08/2025

Step 1 - Imitate:

At the beginning, I referred to Professor Wes's code examples to understand the assignment, and then completed initDesign() and renderDesign() according to the examples and prompts in the slides. I would like to thank Raven Cruz and Jackie Sanchez for their solutions, which helped me solve the canvas size issue. I changed the image canvas size to 400 and set the background color to gray. For the foreground, I placed 100 randomly sized and colored rectangles to fill the canvas. The image below is what I got.

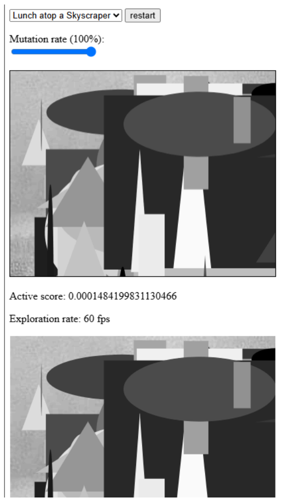


```
Mutation rate (100%):
Active score: 0.0003686593785281125
Exploration rate: 74 fps
Best score: 0.00040855286095733924
```

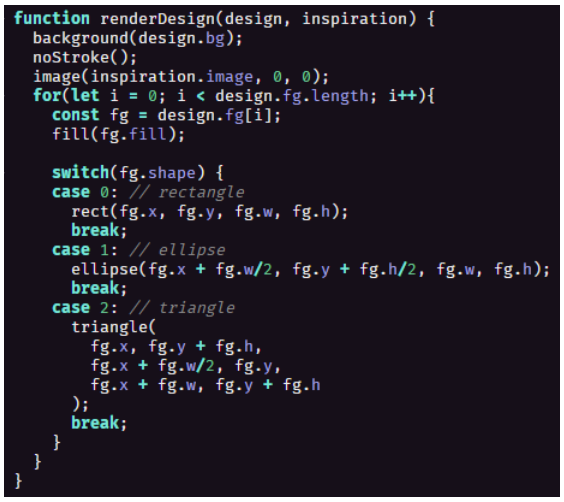


```
function initDesign(inspiration) {

  const size = 400 / inspiration.image.width;
  resizeCanvas(inspiration.image.width * size,
               inspiration.image.height * size);

  let design = {bg:128, fg:[]}
  for(let i = 0; i < 100; i++){
    design.fg.push({
      x: random(width),
      y: random(height),
      w: random(width/2),
      h: random(height/2),
      fill: random(255),
    })
  }

  return design
}

function renderDesign(design, inspiration) {
  background(design.bg);
  noStroke();

  image(inspiration.image, 0, 0)
  for(let i = 0; i < design.fg.length; i++){
    const fg = design.fg[i];
    fill(fg.fill);
    rect(fg.x, fg.y, fg.w, fg.h);
  }

}
```

After that, I added different shapes according to the prompts in the slides. I switched the rectangle into different cases and used random(3) to get random shapes. In this way, the shape of the mutation will not be too simple.



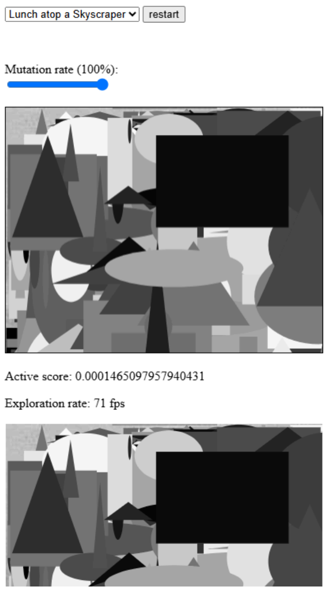
```
Mutation rate (100%):
Active score: 0.0001484199831130466
Exploration rate: 60 fps
```



```
function renderDesign(design, inspiration) {
  background(design.bg);
  noStroke();
  image(inspiration.image, 0, 0);
  for(let i = 0; i < design.fg.length; i++){
    const fg = design.fg[i];
    fill(fg.fill);

    switch(fg.shape) {
    case 0: // rectangle
      rect(fg.x, fg.y, fg.w, fg.h);
      break;
    case 1: // ellipse
      ellipse(fg.x + fg.w/2, fg.y + fg.h/2, fg.w, fg.h);
      break;
    case 2: // triangle
      triangle(
        fg.x, fg.y + fg.h,
        fg.x + fg.w/2, fg.y,
        fg.x + fg.w, fg.y + fg.h
      );
      break;
    }
  }
}
```

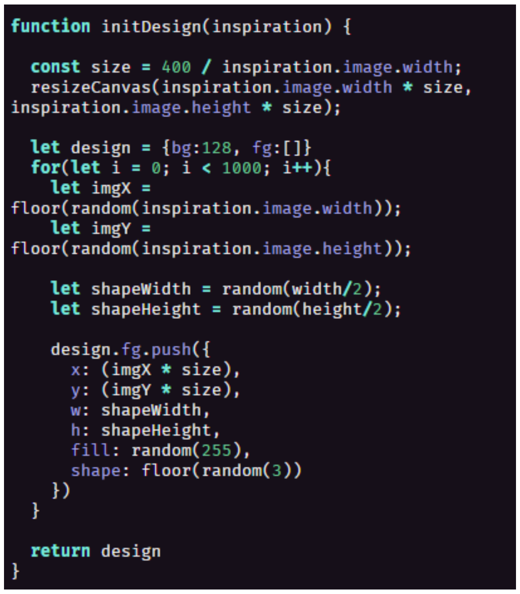
Step 2 - Integrate:

At this point, I noticed that the number of mutation shapes was not enough to cover the entire canvas, so some areas were gray, which means there was no shape in that area. Therefore, I changed the 100 shape to the 1000 shape.



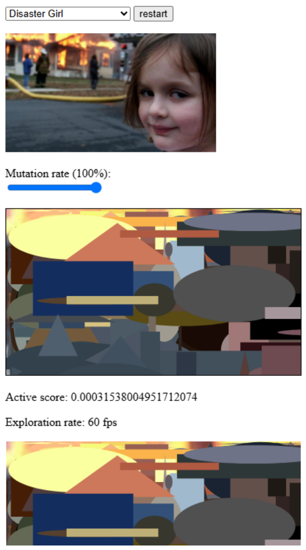
Active score: 0.0001465097957940431

Exploration rate: 71 fps



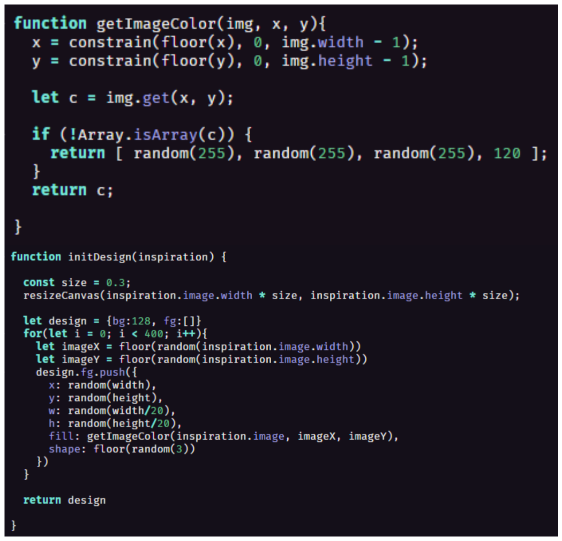
```
function initDesign(inspiration) {

  const size = 400 / inspiration.image.width;
  resizeCanvas(inspiration.image.width * size,
inspiration.image.height * size);

  let design = {bg:128, fg:[]}
  for(let i = 0; i < 1000; i++){
    let imgX =
floor(random(inspiration.image.width));
    let imgY =
floor(random(inspiration.image.height));

    let shapeWidth = random(width/2);
    let shapeHeight = random(height/2);

    design.fg.push({
      x: (imgX * size),
      y: (imgY * size),
      w: shapeWidth,
      h: shapeHeight,
      fill: random(255),
      shape: floor(random(3))
    })
  }

  return design
}
```

At first, I didn't know why the shape had no color, but then I realized it was because of fill: random(225). Then I searched for p5.js examples on how to get the color in the image. Finally, I found the slow_izzm's example, the get() Reference, and Raven Cruz and Jackie Sanchez's solutions. Then, I combined these ideas and made the getImageColor() function. And the image below is what I got.



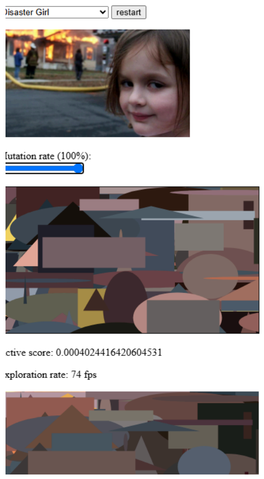
Active score: 0.00031538004951712074

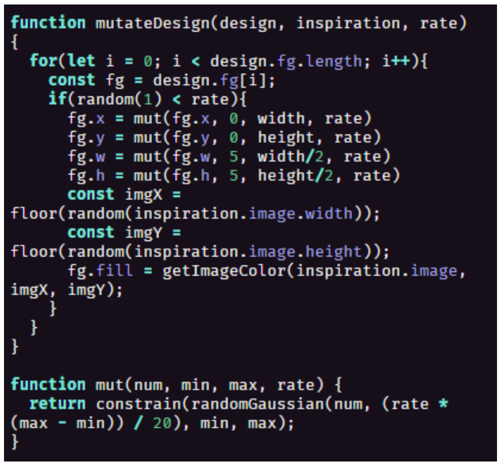
Exploration rate: 60 fps



```
function getImageColor(img, x, y){
  x = constrain(floor(x), 0, img.width - 1);
  y = constrain(floor(y), 0, img.height - 1);

  let c = img.get(x, y);

  if (!Array.isArray(c)) {
    return [ random(255), random(255), random(255), 120 ];
  }
  return c;

}

function initDesign(inspiration) {

  const size = 0.3;
  resizeCanvas(inspiration.image.width * size, inspiration.image.height * size);

  let design = {bg:128, fg:[]}
  for(let i = 0; i < 400; i++){
    let imageX = floor(random(inspiration.image.width))
    let imageY = floor(random(inspiration.image.height))
    design.fg.push({
      x: random(width),
      y: random(height),
      w: random(width/20),
      h: random(height/20),
      fill: getImageColor(inspiration.image, imageX, imageY),
      shape: floor(random(3))
    })
  }

  return design

}
```

I followed the slides and the professor's code and completed the mutateDesign() function. After that, I got the shape that can be randomly mutated. The image below is what I got.

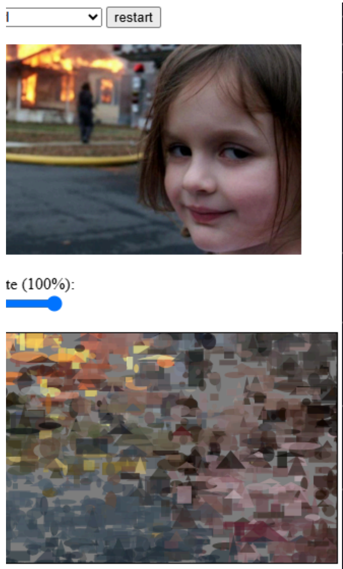


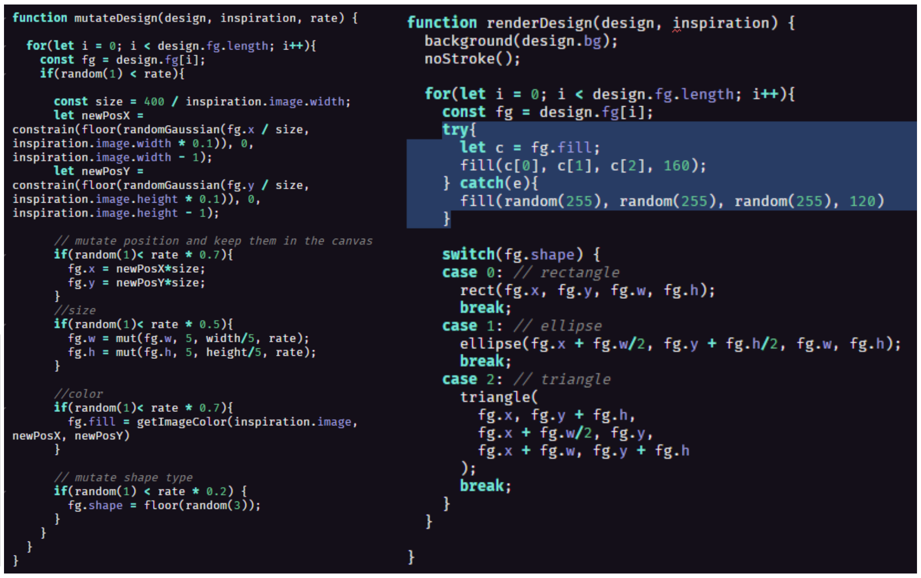


```
function mutateDesign(design, inspiration, rate)
{
  for(let i = 0; i < design.fg.length; i++){
    const fg = design.fg[i];
    if(random(1) < rate){
      fg.x = mut(fg.x, 0, width, rate)
      fg.y = mut(fg.y, 0, height, rate)
      fg.w = mut(fg.w, 5, width/2, rate)
      fg.h = mut(fg.h, 5, height/2, rate)
      const imgX =
floor(random(inspiration.image.width));
      const imgY =
floor(random(inspiration.image.height));
      fg.fill = getImageColor(inspiration.image,
imgX, imgY);
    }
  }
}

function mut(num, min, max, rate) {
  return constrain(randomGaussian(num, (rate *
(max - min)) / 20), min, max);
}
```

However, the mutated shape is now too large and has no relation to the original image at all, or even a new picture. For this reason, I resized the shape again. In addition, I also noticed the problem of shape transparency. Because there are too many mutated shapes, they will cover other shapes, making it impossible for the mutation process to get close to the original image.

For these issues, I spent some time modifying the renderDesign() and mutateDesign(). For the above problem, I modified the transparency of the shape in renderDesign(). To get valid colors, I set the transparency to 160 so that it would not completely cover the color of the shape behind. To prevent invalid colors, I set them to random colors with a transparency of 120. This change will not cause any conflicts or bugs in the mutation process.

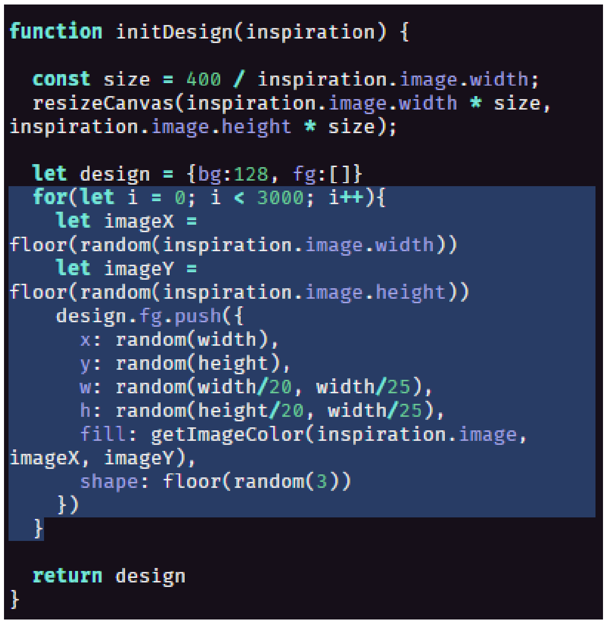




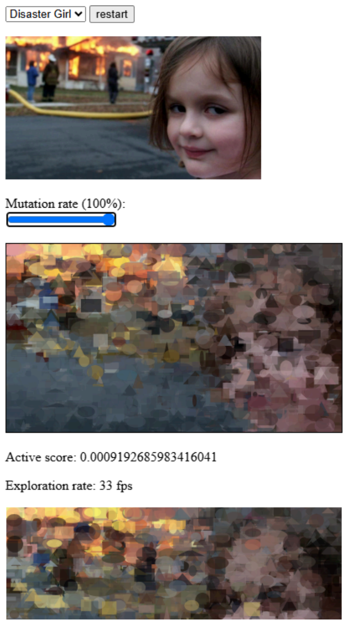
```
function mutateDesign(design, inspiration, rate) {

  for(let i = 0; i < design.fg.length; i++){
    const fg = design.fg[i];
    if(random(1) < rate){

      const size = 400 / inspiration.image.width;
      let newPosX =
constrain(floor(randomGaussian(fg.x / size,
inspiration.image.width * 0.1)), 0,
inspiration.image.width - 1);
      let newPosY =
constrain(floor(randomGaussian(fg.y / size,
inspiration.image.height * 0.1)), 0,
inspiration.image.height - 1);

      // mutate position and keep them in the canvas
      if(random(1)< rate * 0.7){
        fg.x = newPosX*size;
        fg.y = newPosY*size;
      }
      //size
      if(random(1)< rate * 0.5){
        fg.w = mut(fg.w, 5, width/5, rate);
        fg.h = mut(fg.h, 5, height/5, rate);
      }

      //color
      if(random(1)< rate * 0.7){
        fg.fill = getImageColor(inspiration.image,
newPosX, newPosY)
      }

      // mutate shape type
      if(random(1) < rate * 0.2) {
        fg.shape = floor(random(3));
      }
    }
  }
}
```

```
function renderDesign(design, inspiration) {
  background(design.bg);
  noStroke();

  for(let i = 0; i < design.fg.length; i++){
    const fg = design.fg[i];
    try{
      let c = fg.fill;
      fill(c[0], c[1], c[2], 160);
    } catch(e){
      fill(random(255), random(255), random(255), 120)
    }

    switch(fg.shape) {
    case 0: // rectangle
      rect(fg.x, fg.y, fg.w, fg.h);
      break;
    case 1: // ellipse
      ellipse(fg.x + fg.w/2, fg.y + fg.h/2, fg.w, fg.h);
      break;
    case 2: // triangle
      triangle(
        fg.x, fg.y + fg.h,
        fg.x + fg.w/2, fg.y,
        fg.x + fg.w, fg.y + fg.h
      );
      break;
    }
  }

}
```

After the above adjustments, I realized that there were still a few too few shapes, so I increased the number of shapes again to fill the canvas.



```
function initDesign(inspiration) {

  const size = 400 / inspiration.image.width;
  resizeCanvas(inspiration.image.width * size,
inspiration.image.height * size);

  let design = {bg:128, fg:[]}
  for(let i = 0; i < 3000; i++){
    let imageX =
floor(random(inspiration.image.width))
    let imageY =
floor(random(inspiration.image.height))
    design.fg.push({
      x: random(width),
      y: random(height),
      w: random(width/20, width/25),
      h: random(height/20, width/25),
      fill: getImageColor(inspiration.image,
imageX, imageY),
      shape: floor(random(3))
    })
  }

  return design
}
```



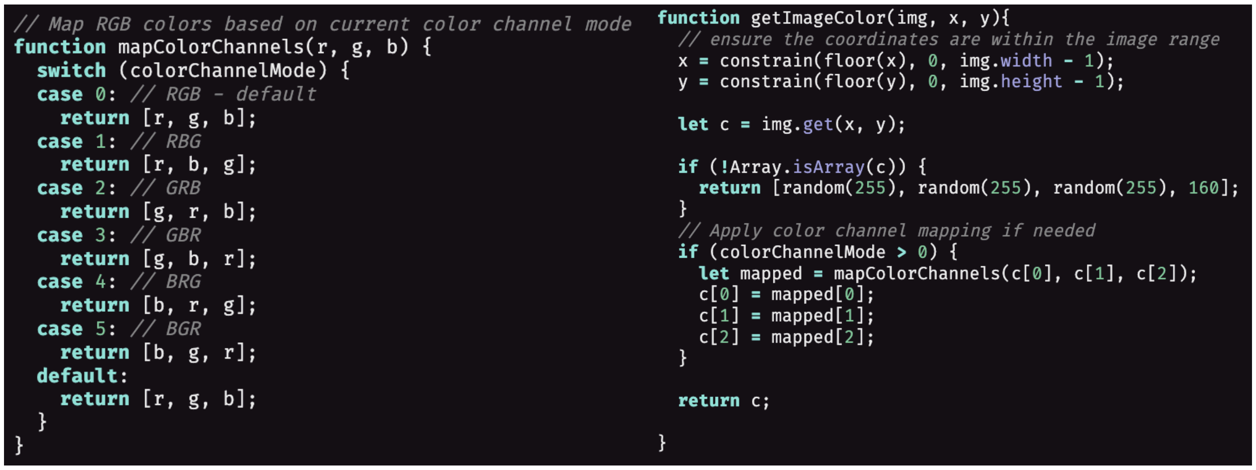
Mutation rate (100%):

Active score: 0.0009192685983416041

Exploration rate: 33 fps

Step 3 - Innovate:

Based on the slow_izzm's example and the get() Reference. I thought if I could add a button to switch the color order—for example, RGB, RBG, GRB, GBR, BGR, BRG.

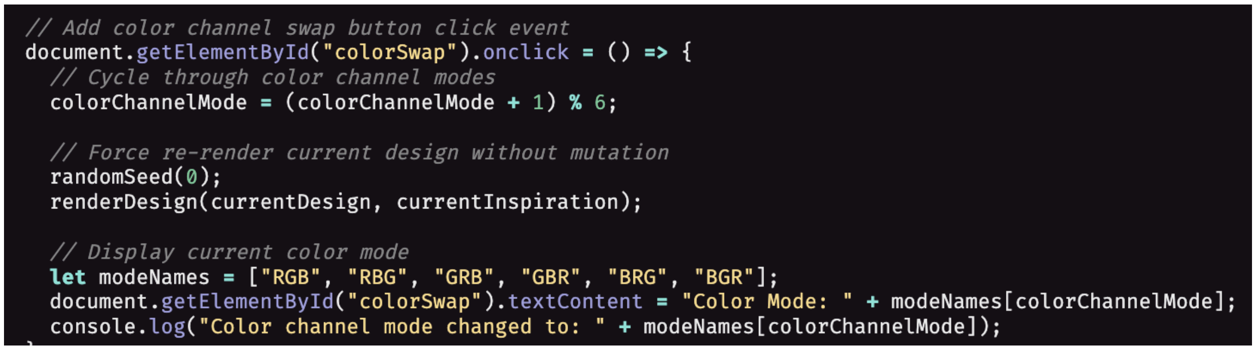


```
// Map RGB colors based on current color channel mode
function mapColorChannels(r, g, b) {
  switch (colorChannelMode) {
    case 0: // RGB - default
      return [r, g, b];
    case 1: // RBG
      return [r, b, g];
    case 2: // GRB
      return [g, r, b];
    case 3: // GBR
      return [g, b, r];
    case 4: // BRG
      return [b, r, g];
    case 5: // BGR
      return [b, g, r];
    default:
      return [r, g, b];
  }
}
```

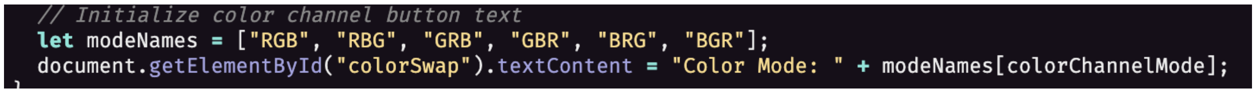
```
function getImageColor(img, x, y){
  // ensure the coordinates are within the image range
  x = constrain(floor(x), 0, img.width - 1);
  y = constrain(floor(y), 0, img.height - 1);

  let c = img.get(x, y);

  if (!Array.isArray(c)) {
    return [random(255), random(255), random(255), 160];
  }
  // Apply color channel mapping if needed
  if (colorChannelMode > 0) {
    let mapped = mapColorChannels(c[0], c[1], c[2]);
    c[0] = mapped[0];
    c[1] = mapped[1];
    c[2] = mapped[2];
  }

  return c;

}
```

To make this idea concrete, I wrote a mapColorChannels() helper that takes an r, g, b triple and reorders it based on the current mode index. A global constant array stores the six possible permutations, and each button press simply increments the index. For example, in RGB order, the red, green, and blue values remain the same, but in RBG order, the blue and green values are swapped. Implementing this function requires careful planning to ensure that the color transition is seamless and does not produce visual artifacts. I also modified the getImageColor() function to integrate it with mapColorChannels() to ensure that the colors retrieved from the canvas are correctly mapped to the selected channel order.

In addition to the above code, I also added the code in the following picture to preload() and setup() in sketch.js. This code is used for debugging and letting users know what mode they are currently switching to.



```
// Add color channel swap button click event
document.getElementById("colorSwap").onclick = () => {
  // Cycle through color channel modes
  colorChannelMode = (colorChannelMode + 1) % 6;

  // Force re-render current design without mutation
  randomSeed(0);
  renderDesign(currentDesign, currentInspiration);

  // Display current color mode
  let modeNames = ["RGB", "RBG", "GRB", "GBR", "BRG", "BGR"];
  document.getElementById("colorSwap").textContent = "Color Mode: " + modeNames[colorChannelMode];
  console.log("Color channel mode changed to: " + modeNames[colorChannelMode]);
```



```
  // Initialize color channel button text
  let modeNames = ["RGB", "RBG", "GRB", "GBR", "BRG", "BGR"];
  document.getElementById("colorSwap").textContent = "Color Mode: " + modeNames[colorChannelMode];
```

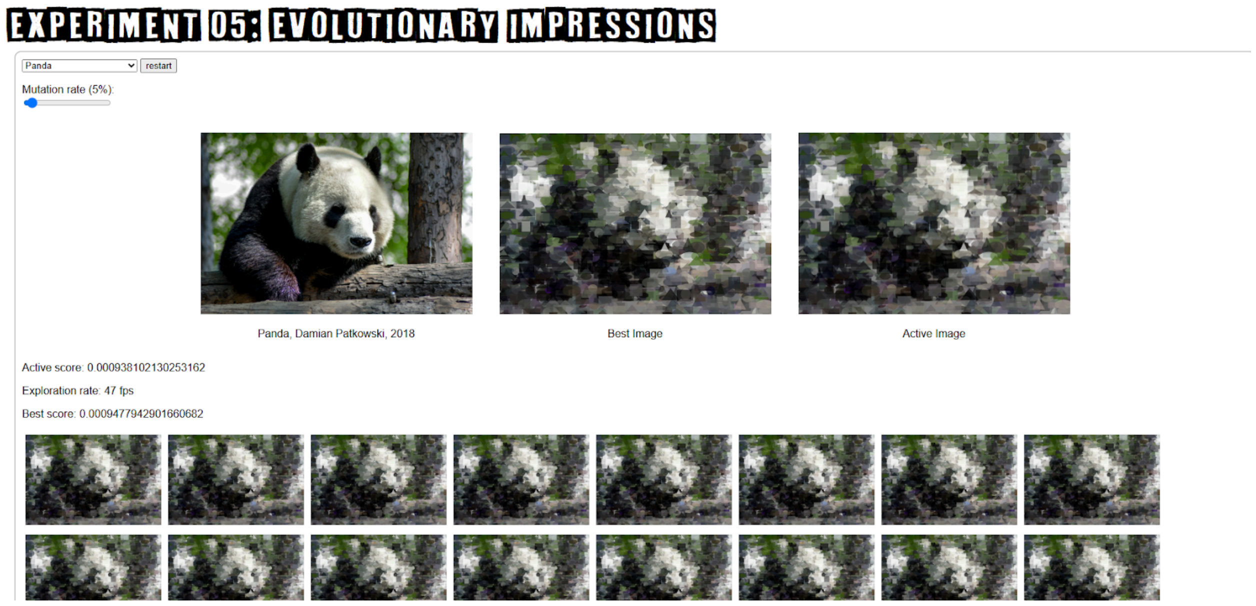
At this point, I believe this experiment is finished. The whole process can be said to be very painful, but I am quite satisfied with the final result. However, what makes me dissatisfied is that the Active score has not been significantly improved. And you are welcome to take a look at my code, website, and Glitch.

Reflection:

Overall, it's okay. The process of constant mutation reminds me of anime EVA and this video. It looks very cool to me. During the debugging process, I also realized that as long as the mutation shape is small enough and there are enough of them, the Active score can be improved. Of course, it also requires enough computer power, otherwise, the mutation process will crash and freeze.

Self Evaluation Rubric						
Completion: Did you complete the assignment and did you complete it on time?	Submitted on time	Up to 1 day late	Up to 2 days late	Up to 3 days late	4 days late or more	Do you need to clarify? I completed it on time.
	X	☐	☐	☐	☐	
Effort: Did you put in earnest effort in execution and attention to detail?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? While writing the code, I took some screenshots and wrote my thoughts at the time.
	☐	X	☐	☐	☐	
Results: How well did you accomplished the assignment goals? Was it complete, with minimal errors, correct output, and good style?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? The assignment was completed without errors. At least, there were no errors during the run. I also have some comments in the file.
	☐	X	☐	☐	☐	
Reflection: How much did you reflect on the strengths and weaknesses of the work and what you'd like to improve?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? I feel okay with my final results. However, I feel like I should have added a few more shapes.
	☐	X	☐	☐	☐	
Reflection on section above.						

Screenshots:



Mutation rate (5%):

Panda, Damian Patkowski, 2018

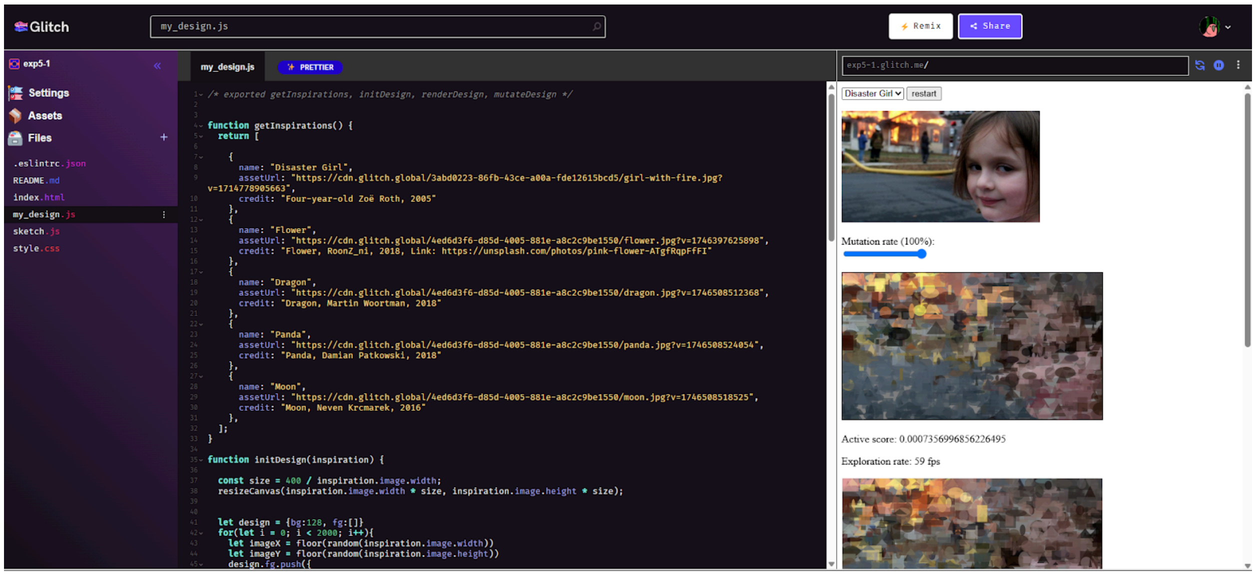
Best Image

Active Image

Active score: 0.000938102130253162

Exploration rate: 47 fps

Best score: 0.0009477942901660682



```
/* exported getInspirations, initDesign, renderDesign, mutateDesign */


function getInspirations() {
  return [

    {
      name: "Disaster Girl",
      assetUrl: "https://cdn.glitch.global/3abd0223-86fb-43ce-a00a-fde12615bcd5/girl-with-fire.jpg?
v=1714778905663",
      credit: "Four-year-old Zoë Roth, 2005"
    },
    {
      name: "Flower",
      assetUrl: "https://cdn.glitch.global/4ed6d3f6-d85d-4005-881e-a8c2c9be1550/flower.jpg?v=1746397625898",
      credit: "Flower, RoonZ_ni, 2018, Link: https://unsplash.com/photos/pink-flower-ATgfRqpFfFI"
    },
    {
      name: "Dragon",
      assetUrl: "https://cdn.glitch.global/4ed6d3f6-d85d-4005-881e-a8c2c9be1550/dragon.jpg?v=1746508512368",
      credit: "Dragon, Martin Woortman, 2018"
    },
    {
      name: "Panda",
      assetUrl: "https://cdn.glitch.global/4ed6d3f6-d85d-4005-881e-a8c2c9be1550/panda.jpg?v=1746508524054",
      credit: "Panda, Damian Patkowski, 2018"
    },
    {
      name: "Moon",
      assetUrl: "https://cdn.glitch.global/4ed6d3f6-d85d-4005-881e-a8c2c9be1550/moon.jpg?v=1746508518525",
      credit: "Moon, Neven Krcmarek, 2016"
    },
  ];
}

function initDesign(inspiration) {

  const size = 400 / inspiration.image.width;
  resizeCanvas(inspiration.image.width * size, inspiration.image.height * size);


  let design = {bg:128, fg:[]}
  for(let i = 0; i < 2000; i++){
    let imageX = floor(random(inspiration.image.width))
    let imageY = floor(random(inspiration.image.height))
    design.fg.push({
```

Mutation rate (100%):

Active score: 0.0007356996856226495

Exploration rate: 59 fps