

# Process & Decision Documentation

## Project/Assignment Decisions

### Side Quests and A4 (Individual Work)

I made the decision to adjust the Y values for the start and instruction button constants in the mousePressed function. In my start screen file, after adjusting the spacing between the title and buttons to fit the image of the dice, I noticed that my buttons were no longer working. To fix this, I realized that I also had to adjust the Y values for the buttons inside the mousePressed function. Once I changed them to the new values I had picked in the drawStart function, the buttons started to work again.

#### *GenAI Documentation*

**Date Used:** Monday, February 2<sup>nd</sup>, 2026 and Tuesday, February 3<sup>rd</sup>, 2026

**Tool Disclosure:** ChatGPT 5.2

**Purpose of Use:** I used GenAI to generate the images of dice for the new storyline of my interactive game and to figure out how to display a different image result for wins and losses every time the screen is called.

**Summary of Interaction:** The tool was able to produce 8 cartoon-style images: The first was of two dice, the second was of one die, and the other six were of 6 dice, each showing the different possible results to roll on a die. It also showed me how to create arrays of images and how to call a random result from those arrays every time the win or the lose screen is called.

**Human Decision Point(s):** In generating these images, I had to adjust my prompts several times so that the tool could generate the appropriate visual for my game. The engine was giving me images of dice with blank faces or with the wrong number of pips. When figuring out the logic for the arrays, the GenAI first made it so that a random image was displayed every frame, and then when I tried to fix it, it made it so that the same random result was shown every time. I had to ask the engine to ensure that a different, but single random result is called every time the win or lose screen is called.

**Scope of GenAI Use:** GenAI did not contribute to the dice idea of the game. I used the class example's logic and screen layout, I came up with the storyline idea of rolling dice on my own, and I used my own pre-existing knowledge about preloading images and displaying them to add the generated images to the game. GenAI was used to generate

images and to write the code for the image arrays and for choosing a random image from the arrays. GenAI was not used to write the README file or this Process Document.

**Limitations or Misfires:** The tool didn't effectively follow my prompts for image generation. The GenAI was not creating an image for any more pips than 2, despite my prompt clearly stating that I wanted the face to show 3 pips. I had to change the wording of my prompt several times to create the rest of the images.

## Appendix

Generate a image of two cartoon style dice on a black background



Generate an image of a single cartoon style die on a black background



Generate 3 separate images of a cartoon-style die on a black background. In the first image, the face shows 1, in the second image the face shows 3, in the third image, the face

shows 5.



Generate an image of a cartoon-style die where the face shows 1



Generate the exact same image, but the face shows 2 instead



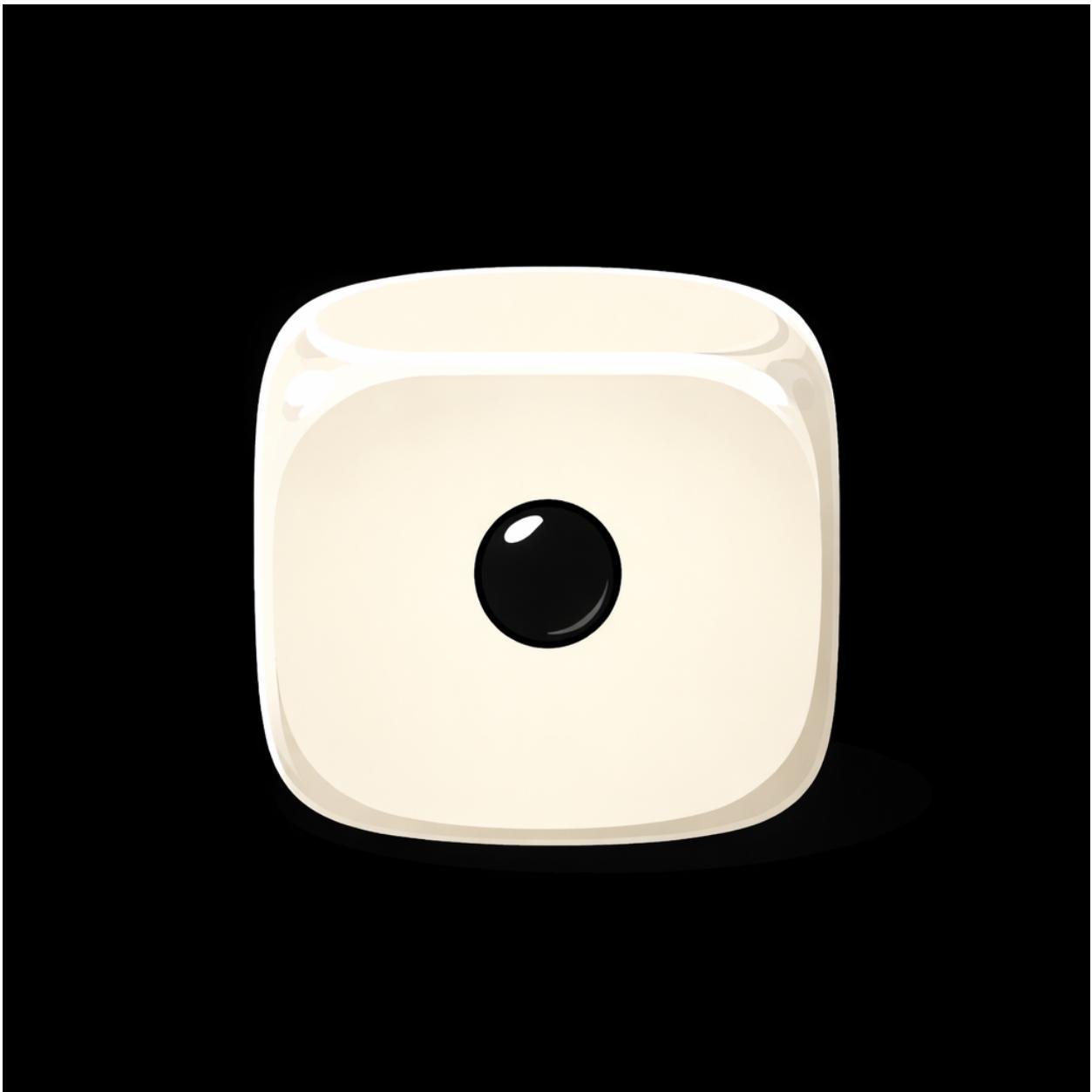
Generate an image of a cartoon-style die on a black background where the front face shows 1



Make the face showing one be facing the front



Generate the image so that no other face is visible except the front face with one on it



Generate the exact same image but with two dots on the front face instead of one, continue to make sure that no other face is visible except the front face



Generate the exact same image but with three dots on the front face instead of two, continue to make sure that no other face is visible except the front face



The front face should have THREE dots



Generate the exact same image but with three dots on the front face instead of two, continue to make sure that no other face is visible except the front face



that is two dots, generate the exact same image with THREE PIPS



generate the same style image, but with three pips on the front face instead of two



Generate the same style image, but with four pips on the front face instead of two



Generate the same style image, but with five pips on the front face instead of four



Generate the same style image, but with six pips on the front face instead of five



Generate the same style image, but with six pips on the front face



How to choose a random image from an array of images in p5.js

Easy one 😊 p5.js makes this super clean.

### Basic idea

1. Put your images in an array
  2. Pick a random index (or use `random()` directly)
  3. Display that image
-

## Example: choosing a random image

```
let images = [];
let chosenImg;

function preload() {
  images[0] = loadImage("img1.png");
  images[1] = loadImage("img2.png");
  images[2] = loadImage("img3.png");
}

function setup() {
  createCanvas(400, 400);

  // Option 1 (recommended): p5's random()
  chosenImg = random(images);

  // Option 2 (manual index)
  // let index = floor(random(images.length));
  // chosenImg = images[index];
}

function draw() {
  background(0);
  image(chosenImg, 0, 0, width, height);
}
```

---

## Change the image each time you click

```
function mousePressed() {
  chosenImg = random(images);
}
```

---

## Change image every few seconds

```
function draw() {
  background(0);
  image(chosenImg, 0, 0, width, height);

  if (frameCount % 120 === 0) { // every ~2 seconds at 60fps
    chosenImg = random(images);
  }
}
```

---

## Common mistakes to avoid

- **✗ Loading images **outside** preload()**
- **✗ Using random(images.length) without floor()**
- **✗ Calling random() every frame when you only want one pick**

---

If you want:

- weighted randomness
- non-repeating images
- random image **once per sketch reload**
- random die faces (🎲 relevant to your dice image)

tell me what behavior you want and I'll tailor it exactly for your project

I don't want to call random every frame, I want one pick

Got it 👍 — you want **one random choice**, picked **once**, not changing every frame.

Here are the **correct + clean ways** to do that in p5.js.

---

### ✓ Best practice: pick once in `setup()`

```
let images = [];
let chosenImg;

function preload() {
  images.push(loadImage("img1.png"));
  images.push(loadImage("img2.png"));
  images.push(loadImage("img3.png"));
}

function setup() {
  createCanvas(400, 400);

  // ONE random pick
  chosenImg = random(images);
}

function draw() {
  background(0);
  image(chosenImg, 0, 0);
}
```

👉 `setup()` runs **once**, so `random()` is called **once**.

---

## ✓ Alternative: use a random index (explicit)

```
function setup() {  
  createCanvas(400, 400);  
  
  let index = floor(random(images.length));  
  chosenImg = images[index];  
}
```

Same result, slightly more verbose.

---

## ✗ What NOT to do

```
function draw() {  
  chosenImg = random(images); // ✗ this runs 60 times per second  
}
```

That's why images keep changing.

---

## 🎯 If you want “random once per page refresh”

You're already done — `setup()` only runs when the sketch loads.

---

## 🎲 Bonus (dice-style logic, no images)

If you ever want the **number** once (e.g. 1–6):

```
let roll;  
  
function setup() {  
  roll = floor(random(1, 7)); // 1 to 6  
}
```

---

If you want:

- one random image **per click**
- one random image **per reload but saved**
- **no repeats**
- or you're mapping dice numbers → images

tell me which one and I'll wire it exactly for your project.

can it be done without setup?

Yes — **100%**, and this is actually a common + valid pattern 

You can do a **one-time random pick without setup()** by using a flag or by doing it at variable declaration time.

Here are the **clean options**, from best → simplest.

---

### Option 1: Pick once using a flag in `draw()` (most explicit)

```
let images = [];
let chosenImg;
let picked = false;

function preload() {
  images.push(loadImage("img1.png"));
  images.push(loadImage("img2.png"));
  images.push(loadImage("img3.png"));
}

function draw() {
  if (!picked) {
    chosenImg = random(images);
    picked = true; // prevents future picks
  }

  background(0);
  image(chosenImg, 0, 0);
}
```

-  `random()` runs **once**
  -  **No** `setup()`
  -  **Very clear intent**
- 

### Option 2: Pick once using a `null` check (clean + minimal)

```
let images = [];
let chosenImg = null;

function preload() {
  images.push(loadImage("img1.png"));
  images.push(loadImage("img2.png"));
  images.push(loadImage("img3.png"));
}
```

```
function draw() {
  if (chosenImg === null) {
    chosenImg = random(images);
  }

  background(0);
  image(chosenImg, 0, 0);
}
```

This works because once `chosenImg` is set, it never changes.

---

### ⚠ Option 3: Pick at global scope (only works *after* preload)

```
let images = [];
let chosenImg;

function preload() {
  images = [
    loadImage("img1.png"),
    loadImage("img2.png"),
    loadImage("img3.png")
  ];
}

function draw() {
  if (!chosenImg) {
    chosenImg = random(images);
  }

  image(chosenImg, 0, 0);
}
```

❗ You **cannot** safely call `random(images)` outside `draw()`/`setup()` because `preload()` must finish first.

---

### ✗ What will break

```
// ✗ images not loaded yet
let chosenImg = random(images);
```

This runs **before** `preload()`.

---

## Rule of thumb

- **Once per sketch load** → `setup()` OR `draw() + guard`
  - **Once per interaction** → `mousePressed()`
  - **Every frame** → `draw()` (no guard)
- 

If this is for a **dice project**, image sequencing, or a generative art assignment, tell me what constraint your prof gave you — I'll tailor it so it still passes grading 😊

do these give you a different option everytime the code runs?

Yes 😊 — **all of the approaches I showed you will give a different option each time the sketch runs** (i.e. every page refresh / restart).

Here's *why*, and one small caveat.

---

## Why you get a different result each run

p5.js automatically **seeds its random number generator** using the current time when the sketch starts.

So when you do:

```
chosenImg = random(images);
```

on sketch load, the RNG seed is different → **different pick**.

This is true whether you pick in:

- `setup()`
- `draw()` (with a guard)
- `mousePressed()`

As long as it runs **once per load**, you'll get a new random result each run.

---

## When it would NOT change

If you do this:

```
randomSeed(42);  
chosenImg = random(images);
```

Then you'll get the **same image every time**, because you forced the seed.

That's sometimes useful for:

- reproducible generative art
  - debugging
  - assignments that require consistency
- 

## Quick summary

Method	Different each run?
random() once	 Yes
draw() + guard	 Yes
setup()	 Yes
randomSeed(fixed)	 No
random() every frame	 Changes constantly

## Dice context (ties to your images)

If you're mapping dice faces:

```
let roll;  
  
function draw() {  
  if (!roll) {  
    roll = floor(random(1, 7)); // 1-6  
  }  
}
```

- 👉 Every reload = new roll
- 👉 Every frame = same roll

Perfect dice behavior.

---

If you want:

- **same roll per session**
- **same roll per day**
- **seeded but still “random-feeling”**
- **keyboard-triggered re-roll**

say the word — this is classic generative design stuff and you’re already doing it right.

when using `null`, it is the same roll every time

Good catch — and you’re right to question it.

**Using `null` itself does NOT cause the same roll every time.** If you’re seeing the same result on every reload, something else is fixing the randomness.

Let’s pin it down 

---

## First: the `null` pattern is NOT deterministic

This code **should** give a different result every reload:

```
let roll = null;

function draw() {
  if (roll === null) {
    roll = floor(random(1, 7));
  }

  text(roll, 100, 100);
}
```

So if it’s always the same number, one of the following is happening.

---



## Most common causes of “same roll every time”

 `randomSeed()` is set somewhere (even indirectly)

Check your code for **any** of these:

```
randomSeed(1);
```

```
randomSeed(42);  
randomSeed(someNumber);
```

Even something like this in another file will lock it:

```
randomSeed(0);  
noiseSeed(0);
```

→ **Fix:** remove it, or move it to debugging only.

---

## 2 You're reseeding every frame (subtle but deadly)

✗ This will force the same value forever:

```
function draw() {  
    randomSeed(1); // ✗ resets RNG every frame  
  
    if (roll === null) {  
        roll = floor(random(1, 7));  
    }  
}
```

---

## 3 You're testing in an environment that hot-reloads state

Some setups (p5 editor, Vite, live servers) **don't fully reset JS state** on refresh.

→ Try:

- Hard refresh (Cmd + Shift + R)
  - Opening in a new tab
  - Restarting the sketch editor
- 

## ✓ Bulletproof solution (no setup, no seed issues)

If you want something that is **guaranteed** to run once and never again:

```
let roll;  
  
function draw() {  
    if (frameCount === 1) {  
        roll = floor(random(1, 7));  
    }  
}
```

```
background(0);
textSize(64);
fill(255);
text(roll, 100, 100);
}
```

- ✓ Runs exactly once
  - ✓ Different every reload
  - ✓ No null, no flags, no `setup()`
- 

## Why `frameCount === 1` is so reliable

- `frameCount` starts at 1
- It's controlled by p5 internally
- Impossible to "accidentally" lock randomness

This is a **very common generative-art pattern**, especially in school assignments.

---

## TL;DR

- ✗ `null` is **not** the problem
- ✗ A hidden `randomSeed()` almost certainly is
- ✗ `frameCount === 1` is the safest one-shot trigger

If you want, paste your exact sketch and I'll spot the line that's freezing the roll instantly ☺

```
let chosenWrongImage = null; function drawLose() { background(0); fill(255, 0, 0);
textAlign(CENTER, CENTER); // Main message textSize(40); text("YOU LOSE!", width / 2,
250); if (chosenWrongImage === null) { chosenWrongImage = random(wrongImages); }
image(chosenWrongImage, width / 3, 300, 250, 250); // Instruction text textSize(20);
text("Click or press R to return to Start.", width / 2, 550); } // -----
// Mouse input for lose screen // ----- // Any mouse click returns the player
to the start screen // (no buttons needed for this simple end state) function
loseMousePressed() { currentScreen = "start"; } // -----
// Keyboard input for lose screen // ----- // R is commonly used for "restart" in games
function loseKeyPressed() { if (key === "r" || key === "R") { currentScreen = "start"; } }
```

Perfect — thanks for pasting the code.

This explains **exactly** why you're seeing the **same image every time** 👍

## Short answer

Your logic is correct **inside drawLose()**, but  
👉 `chosenWrongImage` is never reset when you leave and re-enter the lose screen.

So the first time you lose, it gets set...  
and every future loss reuses that same value.

---

## What's actually happening

```
let chosenWrongImage = null;
```

This runs **once when the sketch loads**, not each time you lose.

Then:

```
if (chosenWrongImage === null) {  
  chosenWrongImage = random(wrongImages);  
}
```

- ✓ Works the **first** time you lose
- ✗ Never runs again, because `chosenWrongImage` is no longer null

Switching screens does **not** reset variables in p5.

---

## ✓ Correct fix (recommended): reset when entering the lose screen

Wherever you switch to the lose screen (something like this):

```
currentScreen = "lose";
```

### Reset the variable at that moment:

```
chosenWrongImage = null;  
currentScreen = "lose";
```

That way:

- Each loss → new random image
- Still only one pick per lose screen
- No flicker
- No `setup()` needed

---

## ✓ Alternative: reset when leaving the lose screen

Since you already have restart handlers:

```
function loseMousePressed() {  
    chosenWrongImage = null;  
    currentScreen = "start";  
}  
  
function loseKeyPressed() {  
    if (key === "r" || key === "R") {  
        chosenWrongImage = null;  
        currentScreen = "start";  
    }  
}
```

Same effect, just reset on exit instead of entry.

---

## 🧠 Why your intuition was right

- `null` logic ✓ correct
- `random()` ✓ correct
- Bug was **screen-state lifecycle**, not randomness

This is a **classic game-state gotcha**, especially in p5 projects.

---

## 🎯 Pro tip (clean architecture)

For screen-based games, randomness should usually happen:

- **when entering a screen**
- not inside the screen's draw loop

You already did 90% of this right — just needed the reset.

If you want, I can help you refactor this into a clean `enterLoseScreen()` pattern that your prof will love 😊