第3回 クリエイティブ・コーディング入門

アニメーションとインタラクション

今日の内容

- マウスを使った描画
  - `mouseX`, `mouseY` で位置に応じた描画
  - `dist()` を使ったサイズ・色の変化
- パーティクルの生成
  - クラスでパーティクルを定義
  - 生成 → 動き → 消える流れ
- パーティクルの動きと印象
  - ゆっくり動く／速く動く表現の違い
  - 操作感・インタラクションへの影響

マウスを使った描画

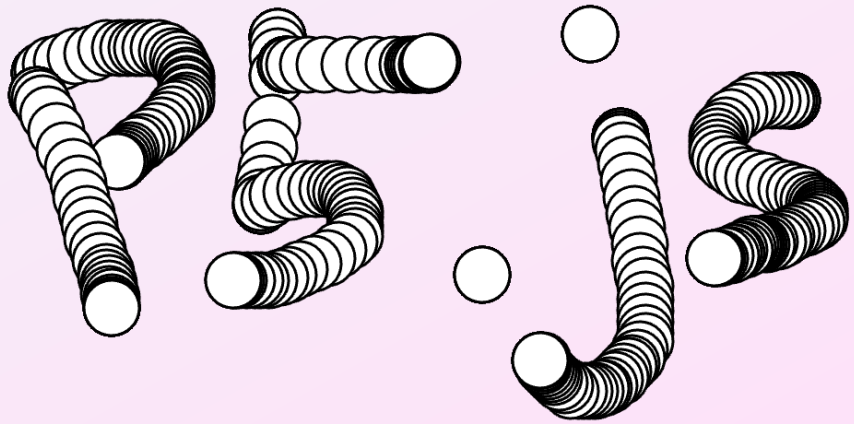
マウス座標で図形を描く

- p5.js では、キャンバス内のマウス位置を取得できる
  - `mouseX`：マウスのX座標
  - `mouseY`：マウスのY座標
- `draw()` 内で `ellipse(mouseX, mouseY, ...)` とすると、マウスカーソルに追従した円を描くことができる（簡易的な「絵かきツール」になる）
- `mouseIsPressed` でクリック状態をブール値で取得できる

例：マウスクリックで描画

```
function setup() {
  createCanvas(600, 400);
}

function draw() {
  if (mouseIsPressed) {
    ellipse(mouseX, mouseY, 20);
  }
}
```



距離 `dist()` を使ったインタラクション

- `dist(x1, y1, x2, y2)`：2点間の距離を返す関数
- 例：
  - 中心から遠いほど大きい円／小さい円
  - マウスの位置からの距離で色を変える

例：中心からの距離で円の大きさを変える

```
function setup() {
  createCanvas(600, 400);
  noStroke();
}

function draw() {
  background(240);

  // キャンバス中心からマウスまでの距離
  let d = dist(width / 2, height / 2, mouseX, mouseY);

  // 距離 d(0〜約350) を サイズ(10〜200) に変換
  let size = map(d, 0, 350, 10, 200);

  fill(100, 150, 255);
  ellipse(width / 2, height / 2, size);

  fill(0);
  textAlign(LEFT, TOP);
  text("distance: " + nf(d, 1, 1), 10, 10);
}
```

例：マウスに近いほど明るくなる円

```
function setup() {
  createCanvas(600, 400);
  colorMode(HSB, 360, 100, 100);
  noStroke();
}

function draw() {
  background(0, 0, 15);
  let d = dist(mouseX, mouseY, width / 2, height / 2);

  // 距離を 0〜100 の明るさにマッピング（遠いと暗く）
  let b = map(d, 0, 300, 100, 20);
  b = constrain(b, 20, 100);

  fill(200, 80, b);
  ellipse(width / 2, height / 2, 200);
}
```

パーティクルの生成

パーティクル

- たくさんの「粒（particle）」が集まった表現
  - 小さな円・点・線などの集合
- 各パーティクルは
  - 位置（position）
  - 速度（velocity）
  - 寿命（lifespan）
    を持ち、時間とともに動いて消えていく

パーティクルのクラスを定義する

1. クラスの定義：パーティクルの雛形を作る（`class Particle { ... }`）
2. クラスの初期設定：`constructor()` で初期位置や速度、寿命を設定
3. `update()` で毎フレーム位置や寿命を更新
4. `display()` で描画
5. 配列 `particles[]` にたくさん入れて、`for` で回す

例：パーティクル・クラス

```
class Particle {
  constructor(x, y) {
    this.x = x; this.y = y;
    this.vx = random(-1, 1); this.vy = random(-1, 1);
    this.e_size = random(5, 20); this.life = 255;
  }
  update() {
    this.x += this.vx; this.y += this.vy; this.life -= 1;
  }
  display() {
    noStroke(); fill(255, this.life);
    ellipse(this.x, this.y, this.e_size);
  }
  isDead() {
    return this.life <= 0;
  }
}
```

パーティクルを動かすメイン部分

- 毎フレーム、全てのパーティクルに対して
  - `update()` → `display()`
- 寿命が尽きたものは配列から削除

```
function setup() {
  createCanvas(600, 400);
}

function draw() {
  background(20);

  // 新しいパーティクルを中央から追加
  particles.push(new Particle(width / 2, height / 2));

  // 全てのパーティクルを更新・描画
  for (let p of particles) {
    p.update();
    p.display();
  }

  // 後ろから順に消えてるものを削除
```

3. マウスに反応するパーティクル

マウス位置からパーティクルを出す

- パーティクルの「発生源（エミッタ）」を
  キャンバス中央 → マウスの位置に変更すれば
  - マウスを動かすだけで「光の軌跡」などの表現ができる
- クリックしているときだけ生成すると
  - スプレーブラシのようなインタラクションになる

例：マウス位置から出るパーティクル

```
let particles = [];

class Particle {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.vx = random(-1, 1);
    this.vy = random(-2, 0);
    this.life = 255;
  }

  update() {
    this.x += this.vx;
    this.y += this.vy;
    this.life -= 4;
  }

  display() {
    noStroke();
    fill(100, 200, 255, this.life);
    ellipse(this.x, this.y, 8);
  }

  isDead() {
    return this.life <= 0;
  }
}

function setup() {
  createCanvas(600, 400);
  background(0);
}

function draw() {
  // クリックしている間パーティクルを追加
  if (mouseIsPressed) {
    for (let i = 0; i < 3; i++) {
      particles.push(new Particle(mouseX, mouseY));
    }
  }

  // 半透明背景で残像っぽく
  fill(0, 30);
  rect(0, 0, width, height);

  for (let p of particles) {
    p.update();
    p.display();
  }

  for (let i = particles.length - 1; i >= 0; i--) {
    if (particles[i].isDead()) {
      particles.splice(i, 1);
    }
  }
}
```

マウスの動きの速さで変化させる

- `pmouseX`, `pmouseY`：1フレーム前のマウス位置
- `dist(mouseX, mouseY, pmouseX, pmouseY)` で
  → どれだけ「速く」動いたかの目安になる
- これを使って
  - 速く動かしたときだけ大きいパーティクル
  - 速いと色が変わる
    などのインタラクションが作れる

例：マウス速度でパーティクルのサイズを変える

```
let particles = [];

class Particle {
  constructor(x, y, s) {
    this.x = x;
    this.y = y;
    this.vx = random(-1, 1);
    this.vy = random(-1, 1);
    this.size = s;
    this.life = 255;
  }

  update() {
    this.x += this.vx;
    this.y += this.vy;
    this.life -= 4;
  }

  display() {
    noStroke();
    fill(255, 200, 100, this.life);
    ellipse(this.x, this.y, this.size);
  }

  isDead() {
    return this.life <= 0;
  }
}

function setup() {
  createCanvas(600, 400);
  background(0);
}

function draw() {
  // マウスの移動距離から速度を求める
  let speed = dist(mouseX, mouseY, pmouseX, pmouseY);
  let s = map(speed, 0, 50, 5, 40);
  s = constrain(s, 5, 40);

  if (mouseIsPressed) {
    particles.push(new Particle(mouseX, mouseY, s));
  }

  fill(0, 40);
  rect(0, 0, width, height);

  for (let p of particles) {
    p.update();
    p.display();
  }

  for (let i = particles.length - 1; i >= 0; i--) {
    if (particles[i].isDead()) {
      particles.splice(i, 1);
    }
  }
}
```

4. パーティクルの動きと印象の違い

動き方による印象の違い

- **ゆっくり・なめらか**な動き
  - 落ち着いた・やわらかい・幻想的
  - バックグラウンドや待機画面などに向いている
- **速く・激しい**動き
  - 元気・緊張感・エネルギッシュ
  - ゲームのエフェクトやアラート表現に向いている
- 速度だけでなく
  - 軌道（まっすぐ / ジグザグ / ランダムウォーク）
  - 重力や風（加速度）の有無
    も印象を大きく変える

例1：ゆっくり漂うパーティクル

- 下方向にゆっくり落ちる → 「ふわっとした」印象

```
let particlesSlow = [];

class SlowParticle {
  constructor() {
    this.x = random(width);
    this.y = random(-50, 0);
    this.vx = random(-0.3, 0.3);
    this.vy = random(0.2, 0.8);
    this.size = random(10, 30);
  }

  update() {
    this.x += this.vx;
    this.y += this.vy;
    if (this.y > height + 20) {
      this.y = random(-50, 0);
      this.x = random(width);
    }
  }

  display() {
    noStroke();
    fill(200, 230, 255, 180);
    ellipse(this.x, this.y, this.size);
  }
}

function setup() {
  createCanvas(600, 400);
  for (let i = 0; i < 60; i++) {
    particlesSlow.push(new SlowParticle());
  }
}

function draw() {
  background(10, 20, 40);
  for (let p of particlesSlow) {
    p.update();
```

例2：すばやく弾けるパーティクル

- 急に飛び出してすぐ消える → アクションや爆発のような印象

```
let burst = [];

class BurstParticle {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    let angle = random(TWO_PI);
    let speed = random(3, 8);
    this.vx = cos(angle) * speed;
    this.vy = sin(angle) * speed;
    this.life = 255;
  }

  update() {
    this.x += this.vx;
    this.y += this.vy;
    this.life -= 10;
  }

  display() {
    noStroke();
    fill(255, 200, 50, this.life);
    ellipse(this.x, this.y, 6);
  }

  isDead() {
    return this.life <= 0;
  }
}

function mousePressed() {
  // クリックした場所から一気に噴き出す
  for (let i = 0; i < 80; i++) {
    burst.push(new BurstParticle(mouseX, mouseY));
  }
}

function draw() {
  background(0);
  for (let p of burst) {
    p.update();
    p.display();
  }
  for (let i = burst.length - 1; i >= 0; i--) {
    if (burst[i].isDead()) {
      burst.splice(i, 1);
```

まとめ

- マウス座標（`mouseX`, `mouseY`）や `dist()` を使うと
  → 位置に応じたインタラクション表現ができる
- パーティクルは
  - 「位置・速度・寿命」を持つ小さな点の集合
  - クラスと配列で管理すると扱いやすい
- 動きのパラメータ（速度・方向・重力など）を変えることで
  - 「落ち着いた」「激しい」「軽い」「重い」など
    インターフェースの印象をコントロールできる

演習

1. マウスの動きに応じて
    - 大きさ・色・透明度が変わる「ブラシ」を作る
2. パーティクルクラスを使って
    - マウスクリックで噴き出すエフェクトをデザインする
3. 動きのパラメータ（速度・重力・寿命など）を変えて
    - 「落ち着いた」「激しい」など、印象が変わるパターンを2種類以上つくる

windowresizeによるキャンバスサイズの自動調整

fullscreenの利用

- keypressed関数を使う
- key == f などで特定のキー入力を処理を紐付け