Código do Projeto

```
//variáveis globais
boolean mouseInsideCanvas = false;
int numNuvens = 110;  //numero de nuvens
int numNotas = 40;    //numero de notas
int numRaios = 16;    //numero de raios
int numEstrelas = 120; //numero de estrelas



// controle de tempo / ciclo dia-noite
boolean ehDia = true;
boolean emTransicao = false;
float tempoPassado = 0;
float duracaoCicloBase = 240;  // 4 minutos = 240s
float transicaoTempo = 0;
float velocidadeTransicao = 0.01;
float proximaMudancaTempo;


//nuvens
float[] nuvensX = new float[numNuvens];
float[] nuvensY = new float[numNuvens];
float[] nuvensTam = new float[numNuvens];
float[] nuvensVel = new float[numNuvens];
float[] nuvensTrans = new float[numNuvens];
float velocidadeBase = 2;


//estrelas
float[] estrelasX = new float[numEstrelas];
```

```
float[] estrelasY = new float[numEstrelas];
float[] estrelasTam = new float[numEstrelas];
float[] estrelasBrilho = new float[numEstrelas];
float[] estrelasPulso = new float[numEstrelas];
boolean[] estrelasHover = new boolean[numEstrelas];

//efeito do click
boolean efeitoClique = false;
float cliqueX, cliqueY;
float cliqueRaio = 0;
float cliqueMaxRaio = 500;
float cliqueVelocidade = 15;

//notas musicais
float[] notasX = new float[numNotas];
float[] notasY = new float[numNotas];
float[] notasAngulo = new float[numNotas];
float[] notasVel = new float[numNotas];
int[] notasTipo = new int[numNotas];
float[] notasTamanho = new float[numNotas];
float[] notasRotacao = new float[numNotas];
int[] notasFonte = new int[numNotas];
boolean[] notasAtivas = new boolean[numNotas];
float[] notasOndulacao = new float[numNotas];
float[] notasTempoVida = new float[numNotas];
float[] notasAceleracao = new float[numNotas];
float[] notasTargetX = new float[numNotas];
float[] notasTargetY = new float[numNotas];
```

```
int proximaNotaIndex = 0;

//comboio de notas pipi
int trailLength = 300;
PVector[] mouseTrail = new PVector[trailLength];
int trailIndex = 0;

// e um array paralelo para cada nota guardar seu índice de início no trail:
int[] notasTrailStart = new int[numNotas];

//liras posição
float[] lirasX = new float[4];
float[] lirasY = new float[4];
float[] lirasEnergia = new float[4];
int[] lirasProxNota = new int[4];

//imagens
PImage[] notasImagens = new PImage[5];  //5 tipos de notas

//posições fixas (ilha e sol)
float movimentoIlhaAmplitude;
float ilhaX, ilhaY;
float solX, solY;
float luaX, luaY;
float rotacaoAstro = 0;

//controle de frames
int intervaloNovaNotaBase = 15;  //gerar notas mais frequentemente
```

```
int proximaNotaEm = intervaloNovaNotaBase;

//cores
color corCeu1, corCeu2;
color corCeuNoite1, corCeuNoite2;
color corIlhaPrincipal, corIlhaGrama1, corIlhaGrama2, corIlhaPraia;
color corTemplo, corTemploSombra, corTemploDetalhe;
color corTemploNoite, corTemploDetalhesNoite;
color corSol1, corSol2, corSolRaios;
color corLua1, corLua2, corLuaRaios;

//------------------------------------------------------------------------------------------
//------------------------------

void setup() {
  size(1000, 600);
 frameRate(60);
 proximaMudancaTempo = duracaoCicloBase;  //240s = 4min

 //comboio
 for (int i = 0; i < trailLength; i++) {
   mouseTrail[i] = new PVector(width/2, height/2);
 }
 for (int i = 0; i < numNotas; i++) {
   notasTrailStart[i] = 0;
 }
```

```
//definir cores
corCeu1 = color(100, 180, 255);
corCeu2 = color(60, 120, 220);
corCeuNoite1 = color(20, 30, 70);
corCeuNoite2 = color(5, 10, 30);

corIlhaPrincipal = color(160, 140, 100);
corIlhaGrama1 = color(130, 200, 100);
corIlhaGrama2 = color(140, 220, 120);
corIlhaPraia = color(210, 190, 140);

corTemplo = color(250, 240, 200);
corTemploSombra = color(200, 190, 160);
corTemploDetalhe = color(240, 230, 190);

corTemploNoite = color(180, 180, 200);
corTemploDetalhesNoite = color(200, 200, 255, 180);

corSol1 = color(255, 240, 100);
corSol2 = color(255, 215, 30);
corSolRaios = color(255, 240, 30);

corLua1 = color(240, 240, 255);
corLua2 = color(220, 220, 240);
corLuaRaios = color(230, 230, 255, 150);

//posições fixas
ilhaX = width / 2;
```

```
ilhaY = height / 2;
solX = width - 120;
solY = 120;
luaX = 120;
luaY = 120;
movimentoIlhaAmplitude = 5;

//carregar imagens das notas (5 tipos)
for (int i = 0; i < 5; i++) {
 try {
  notasImagens[i] = loadImage("nota" + (i+1) + ".png");
  if (notasImagens[i] == null) throw new Exception("Imagem não encontrada");
 } catch (Exception e) {
  //criar nota substituta se a imagem não existir ou der erro
  notasImagens[i] = createImage(40, 40, ARGB);
  notasImagens[i].loadPixels();

  notasImagens[i].updatePixels();
 }
}

//iniciar nuvens
for (int i = 0; i < numNuvens; i++) {
 nuvensX[i] = random(-200, width * 1.5);
 nuvensY[i] = random(height);
 nuvensTam[i] = random(60, 200);
 nuvensVel[i] = random(0.3, 1.2);
 nuvensTrans[i] = random(150, 255);
```

```
}

//iniciar estrelas
for (int i = 0; i < numEstrelas; i++) {
  estrelasX[i] = random(width);
  estrelasY[i] = random(height * 0.8);
  estrelasTam[i] = random(1, 4);
  estrelasBrilho[i] = random(150, 255);
  estrelasPulso[i] = random(TWO_PI);
  estrelasHover[i] = false;
}

//posições das liras
lirasX[0] = ilhaX - 200;
lirasY[0] = ilhaY;
lirasX[1] = ilhaX + 200;
lirasY[1] = ilhaY;
lirasX[2] = ilhaX - 100;
lirasY[2] = ilhaY + 50;
lirasX[3] = ilhaX + 100;
lirasY[3] = ilhaY + 50;

//iniciar energia das liras
for (int i = 0; i < 4; i++) {
  lirasEnergia[i] = 0;
  lirasProxNota[i] = int(random(20, 50));
}
```

```
  //iniciar notas
  for (int i = 0; i < numNotas; i++) {
    notasAtivas[i] = false;
    notasTipo[i] = int(random(5));  // 5 tipos de notas
    notasTamanho[i] = random(20, 50);
    notasRotacao[i] = random(TWO_PI);
    notasOndulacao[i] = random(TWO_PI);
    notasTempoVida[i] = 0;
    notasAceleracao[i] = random(0.01, 0.05);
  }

  noStroke();
  imageMode(CENTER);
}

//------------------------------------------------------------------------------------------
//------------------------------

void draw() {
  //comboio armaneza posição
  mouseTrail[trailIndex].set(mouseX, mouseY);
  trailIndex = (trailIndex + 1) % trailLength;

  //atualiza velocidade de animação
  float velocidadeBase = mouseInsideCanvas ? 1.0 : 2.0;
  //incrementa o tempo decorrido
  float deltaTime = 1.0 / frameRate;
  tempoPassado += deltaTime * velocidadeBase;
```

```
//inicia transição se chegou na hora
 if (tempoPassado >= proximaMudancaTempo && !emTransicao) {
  emTransicao = true;
  ehDia = !ehDia;
}

// processa transição suave
if (emTransicao) {
  if (ehDia) {
    // passou para DIA: diminui de 1 → 0
    transicaoTempo -= velocidadeTransicao * velocidadeBase;
    if (transicaoTempo <= 0) {
      transicaoTempo = 0;
      emTransicao = false;
      proximaMudancaTempo = tempoPassado + duracaoCicloBase;
    }
  } else {
    //passou para NOITE: aumenta de 0 → 1
    transicaoTempo += velocidadeTransicao * velocidadeBase;
    if (transicaoTempo >= 1) {
      transicaoTempo = 1;
      emTransicao = false;
      proximaMudancaTempo = tempoPassado + duracaoCicloBase;
    }
  }
}
```

```
  // desenha cena completa (fundo, sol/lua, nuvens, notas, etc.)
  criarFundoGradiente();
  if (transicaoTempo > 0.1) desenharEstrelas();
  if (transicaoTempo < 0.95) desenharBrilhoSol(1 - transicaoTempo);
  if (transicaoTempo > 0.05) desenharBrilhoLua(transicaoTempo);
  atualizarNuvens();  desenharNuvens();
  desenharIlha();
  if (transicaoTempo < 0.95) desenharSolEraios(1 - transicaoTempo);
  if (transicaoTempo > 0.05) desenharLuaEraios(transicaoTempo);
  processarEfeitoClique();
  atualizarLiras();
  desenharNotasMusicais();

  float tempoRestante = max(proximaMudancaTempo - tempoPassado, 0);
  int minR = int(tempoRestante) / 60;
  int segR = int(tempoRestante) % 60;
  String txt = nf(minR, 2) + ":" + nf(segR, 2);

  textSize(64);
  textAlign(CENTER, TOP);
  stroke(0);
  strokeWeight(6);
  fill(255);
  text(txt, width/2, 10);
  noStroke();
}

//--------------------------------------------------------------------------------
```

```
//------------------------------

//detecção confiável de mouse dentro/fora do canvas
void mouseEntered() {
  mouseInsideCanvas = true;
}

void mouseExited() {
  mouseInsideCanvas = false;
}

//-----------------------------------------------------------------------------------------
//------------------------------

void criarFundoGradiente() {
  //interpolação entre céu diurno e noturno baseado na transição
  for (int y = 0; y < height; y++) {
    float fator = map(y, 0, height, 0, 1);

    //cores do céu durante o dia
    color corDiaCeu = lerpColor(corCeu1, corCeu2, fator);

    //cores do céu durante a noite
    color corNoiteCeu = lerpColor(corCeuNoite1, corCeuNoite2, fator);

    //interpolação entre dia e noite
    color corAtual = lerpColor(corDiaCeu, corNoiteCeu, transicaoTempo);
```

```
  stroke(corAtual);
  line(0, y, width, y);
 }
 noStroke();
}

//-----------------------------------------------------------------------------------------
//------------------------------

void desenharEstrelas() {
 float opacidadeFactor = constrain(map(transicaoTempo, 0.1, 0.4, 0, 1), 0, 1);

 for (int i = 0; i < numEstrelas; i++) {
  //verificar se o mouse está sobre a estrela
  float distToStar = dist(mouseX, mouseY, estrelasX[i], estrelasY[i]);
  float limiarDistancia = 20; // Distância para detectar hover

  if (mouseInsideCanvas && distToStar < limiarDistancia) {
   estrelasHover[i] = true;
  } else {
   if (estrelasHover[i]) {
    estrelasHover[i] = false;
   }
  }

  //calcular tamanho e brilho da estrela
  float tamanhoBase = estrelasTam[i];
  float brilhoBase = estrelasBrilho[i];
```

```
//efeito de pulsação
float pulso = sin(frameCount * 0.05 + estrelasPulso[i]) * 0.5 + 0.5;

//tamanho com hover
float tamanhoFinal = estrelasHover[i] ?
                     tamanhoBase * 3 + pulso * 2 :
                     tamanhoBase + pulso;

//brilho com hover
float brilhoFinal = estrelasHover[i] ?
                    255 :
                    brilhoBase + pulso * 30;

//opacidade baseada na transição dia/noite
float opacidade = brilhoFinal * opacidadeFactor;

//desenhar estrela com efeito de brilho
fill(255, 255, 255, opacidade);
ellipse(estrelasX[i], estrelasY[i], tamanhoFinal, tamanhoFinal);

//adicionar brilho para estrelas em hover
if (estrelasHover[i]) {
  for (int j = 1; j <= 3; j++) {
    float tamBrilho = tamanhoFinal * (1 + j * 0.7);
    float opBrilho = opacidade * (1 - j * 0.2);
    fill(255, 255, 255, opBrilho);
    ellipse(estrelasX[i], estrelasY[i], tamBrilho, tamBrilho);
```

```
    }

    //raios da estrela quando hover
    pushMatrix();
    translate(estrelasX[i], estrelasY[i]);
    rotate(frameCount * 0.01);

    for (int r = 0; r < 4; r++) {
     rotate(PI/4);
     fill(255, 255, 255, opacidade * 0.5);
     rect(0, 0, tamanhoFinal * 4, tamanhoFinal * 0.5, tamanhoFinal);
    }
    popMatrix();
   }
  }
}

//-----------------------------------------------------------------------------------------
//------------------------------
// Atualizações

void atualizarLiras() {
 float movimentoY = sin(frameCount * 0.01) * movimentoIlhaAmplitude;

 for (int i = 0; i < 4; i++) {
  //aumentar energia da lira
  lirasEnergia[i] += 1 * velocidadeBase;
```

```
  //criar nota quando energia suficiente
  if (lirasEnergia[i] >= lirasProxNota[i]) {
    criarNovaNota(i);
    lirasEnergia[i] = 0;
    lirasProxNota[i] = int(random(30, 80));
  }

  //efeito visual de energia
  if (lirasEnergia[i] > lirasProxNota[i] * 0.8) {
    float pulso = sin(frameCount * 0.2) * 10;
    //cor do pulso baseada no ciclo (azulado à noite, dourado de dia)
    color corPulso = lerpColor(color(255, 255, 200, 100), color(200, 220, 255, 100),
transicaoTempo);
    fill(corPulso);
    ellipse(lirasX[i], lirasY[i] + movimentoY, 50 + pulso, 50 + pulso);
  }
 }
}

void atualizarNuvens() {
 int nuvensVisiveis = ehDia ? numNuvens : int(numNuvens * 0.3); // Menos nuvens
à noite

 for (int i = 0; i < nuvensVisiveis; i++) {
  //movimento padrão com a velocidade ajustada
  nuvensX[i] += nuvensVel[i] * velocidadeBase;

  //ajustar movimento baseado no efeito de clique
  if (efeitoClique) {
```

```
    float dx = nuvensX[i] - cliqueX;
    float dy = nuvensY[i] - cliqueY;
    float dist = sqrt(dx*dx + dy*dy);

    if (dist < cliqueRaio) {
     float forca = map(dist, 0, cliqueRaio, 5, 0.5);
     float angulo = atan2(dy, dx);
     nuvensX[i] += cos(angulo) * forca;
     nuvensY[i] += sin(angulo) * forca;
    }
   }

   //reposicionar nuvens que saem da tela
   if (nuvensX[i] > width + nuvensTam[i]/2) {
    nuvensX[i] = -nuvensTam[i];
    nuvensY[i] = random(height);
    nuvensTam[i] = random(60, 200);
    nuvensVel[i] = random(0.3, 1.2);
   }
  }
}

//-----------------------------------------------------------------------------------------
//------------------------------

void processarEfeitoClique() {
 if (efeitoClique) {
  cliqueRaio += cliqueVelocidade;
```

```
  //desenhar onda de clique
  noFill();
  stroke(255, 255, 255, map(cliqueRaio, 0, cliqueMaxRaio, 150, 0));
  strokeWeight(3);
  ellipse(cliqueX, cliqueY, cliqueRaio*2, cliqueRaio*2);
  noStroke();

  if (cliqueRaio > cliqueMaxRaio) {
    efeitoClique = false;
  }
 }
}

//-------------------------------------------------------------------------------------
//------------------------------
// ---------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
-
void criarNovaNota(int liraIndex) {
 int tentativas = 0;
 while (tentativas < numNotas) {
  if (!notasAtivas[proximaNotaIndex]) {
    notasFonte[proximaNotaIndex] = liraIndex;
    notasX[proximaNotaIndex] = lirasX[liraIndex];
    notasY[proximaNotaIndex] = lirasY[liraIndex] - 20;
    notasVel[proximaNotaIndex] = random(1.0, 2.5);
    notasAngulo[proximaNotaIndex] = random(PI/4, PI*0.6);
    notasAtivas[proximaNotaIndex] = true;
```

```
    notasTempoVida[proximaNotaIndex] = 0;
    notasTipo[proximaNotaIndex] = int(random(5));  //5 tipos de notas
    notasTargetX[proximaNotaIndex] = mouseX;
    notasTargetY[proximaNotaIndex] = mouseY;
    break;
   }

   proximaNotaIndex = (proximaNotaIndex + 1) % numNotas;
   tentativas++;
  }

  proximaNotaIndex = (proximaNotaIndex + 1) % numNotas;
  notasTrailStart[proximaNotaIndex] = trailIndex;
}



//-------------------------------------------------------------------------------------------
//------------------------------
//Desenhar
void desenharNuvens() {
  //quantidade de nuvens visíveis baseada no ciclo dia/noite
  int nuvensVisiveis = int(map(transicaoTempo, 0, 1, numNuvens, numNuvens *
0.3));

  for (int i = nuvensVisiveis-1; i >= 0; i--) {
   //ajustar opacidade das nuvens baseada no ciclo dia/noite
   float opacidadeBase = map(transicaoTempo, 0, 1, nuvensTrans[i], nuvensTrans[i]
* 0.6);
   float opacidade = opacidadeBase;
```

```
    //nuvens ficam mais transparentes próximas ao sol/lua
    float distSol = dist(nuvensX[i], nuvensY[i], solX, solY);
    float distLua = dist(nuvensX[i], nuvensY[i], luaX, luaY);

    if (distSol < 200 && transicaoTempo < 0.5) {
      opacidade = map(distSol, 0, 200, 50, opacidade);
    }

    if (distLua < 200 && transicaoTempo > 0.5) {
      opacidade = map(distLua, 0, 200, 50, opacidade);
    }

    //cor das nuvens baseada no ciclo (brancas de dia, azuladas à noite)
    color corNuvemDia = color(255, 255, 255, opacidade);
    color corNuvemNoite = color(200, 210, 255, opacidade);
    color corNuvem = lerpColor(corNuvemDia, corNuvemNoite, transicaoTempo);

    desenharNuvem(nuvensX[i], nuvensY[i], nuvensTam[i], corNuvem);
  }
}

void desenharNuvem(float x, float y, float t, color corNuvem) {
  if (x < -t || x > width + t || y < -t || y > height + t) return;

  fill(corNuvem);
  ellipse(x, y, t, t * 0.6);
  ellipse(x + t * 0.3, y - t * 0.2, t * 0.8, t * 0.6);
```

```
  ellipse(x - t * 0.3, y - t * 0.2, t * 0.8, t * 0.6);
  ellipse(x - t * 0.1, y + t * 0.1, t * 0.7, t * 0.5);
}

void desenharBrilhoSol(float intensidade) {
  float pulsacao = sin(frameCount * 0.02) * 20;

  for (int i = 20; i > 0; i--) {
    float alpha = map(i, 0, 20, 0, 180) * intensidade;
    float tamanho = map(i, 0, 20, 350 + pulsacao, 140);
    fill(corSol1, alpha);
    ellipse(solX, solY, tamanho, tamanho);
  }
}

void desenharBrilhoLua(float intensidade) {
  float pulsacao = sin(frameCount * 0.015) * 15;

  for (int i = 20; i > 0; i--) {
    float alpha = map(i, 0, 20, 0, 150) * intensidade;
    float tamanho = map(i, 0, 20, 300 + pulsacao, 120);
    fill(corLua1, alpha);
    ellipse(luaX, luaY, tamanho, tamanho);
  }
}

void desenharSolEraios(float intensidade) {
  //calcular intensidade dos raios baseado na distância do mouse
```

```
 float intensidadeRaios;
 if (mouseInsideCanvas) {
  float distancia = dist(mouseX, mouseY, solX, solY);
  intensidadeRaios = map(constrain(distancia, 0, 500), 0, 500, 4.0, 0.8);
 } else {
  intensidadeRaios = 1.5;
 }

 intensidadeRaios *= intensidade; // Aplicar intensidade baseada no ciclo dia/noite
 float pulsacao = sin(frameCount * 0.05) * 5;

 //sol
 for (int r = 50; r > 0; r -= 4) {
  float brilho = map(r, 0, 50, 255, 160) * intensidade;
  fill(255, brilho, 30, 220 * intensidade);
  ellipse(solX, solY, r * 2 + pulsacao, r * 2 + pulsacao);
 }

 //raios de sol
 fill(corSolRaios, 150 * intensidade);
 pushMatrix();
 translate(solX, solY);
 rotacaoAstro += 0.008 * velocidadeBase;
 rotate(rotacaoAstro);

 for (int i = 0; i < numRaios; i++) {
  pushMatrix();
```

```
    rotate(radians(i * (360/numRaios)));

    float raioLength = 150 * intensidadeRaios;
    raioLength += sin(frameCount * 0.1 + i) * 15;

    fill(corSolRaios, 150 * intensidade);
    triangle(0, -12, raioLength, 0, 0, 12);

    popMatrix();
  }
  popMatrix();

  //centro do sol
  fill(255, 255, 200, 255 * intensidade);
  ellipse(solX, solY, 60 + pulsacao/2, 60 + pulsacao/2);

  fill(255, 255, 255, 200 * intensidade);
  ellipse(solX, solY, 35, 35);
}

void desenharLuaEraios(float intensidade) {
  float pulsacao = sin(frameCount * 0.03) * 4;

  //lua
  fill(corLua1, 255 * intensidade);
  ellipse(luaX, luaY, 70 + pulsacao, 70 + pulsacao);

  //detalhes da lua (crateras)
```

```
  fill(corLua2, 200 * intensidade);
  ellipse(luaX - 15, luaY + 10, 20, 20);
  ellipse(luaX + 20, luaY - 5, 15, 15);
  ellipse(luaX - 5, luaY - 18, 12, 12);

  //brilho central
  fill(255, 255, 255, 180 * intensidade);
  ellipse(luaX, luaY, 40, 40);

  //raios da lua
  pushMatrix();
  translate(luaX, luaY);
  rotate(-rotacaoAstro * 0.5); //rotação mais lenta que o sol

  // 8 raios principais da lua
  for (int i = 0; i < 8; i++) {
    pushMatrix();
    rotate(radians(i * 45));

    float raioLength = 120 + sin(frameCount * 0.08 + i) * 10;

    fill(corLuaRaios, 100 * intensidade);
    triangle(0, -8, raioLength, 0, 0, 8);

    popMatrix();
  }
  popMatrix();
}
```

```
void desenharIlha() {
  float movimentoY = sin(frameCount * 0.01) * movimentoIlhaAmplitude;

  //definir cores de iluminação baseadas no ciclo dia/noite
  color corSombraAtual = lerpColor(color(255, 240, 180, 40), color(150, 170, 255, 30), transicaoTempo);
  color corGramaAtual1 = lerpColor(corIlhaGrama1, color(70, 100, 140), transicaoTempo);
  color corGramaAtual2 = lerpColor(corIlhaGrama2, color(50, 80, 120), transicaoTempo);
  color corIlhaAtual = lerpColor(corIlhaPrincipal, color(80, 90, 130), transicaoTempo);
  color corPraiaAtual = lerpColor(corIlhaPraia, color(120, 130, 180), transicaoTempo);

  //sombra da ilha
  for (int i = 15; i > 0; i -= 3) {
    float alpha = map(i, 0, 15, 0, 40);
    fill(red(corSombraAtual), green(corSombraAtual), blue(corSombraAtual), alpha);
    ellipse(ilhaX, ilhaY + movimentoY, 600 + i*10, 320 + i*5);
  }

  //ilha principal
  fill(corIlhaAtual);
  ellipse(ilhaX, ilhaY + movimentoY, 580, 300);

  //camadas da relva
  fill(corGramaAtual1);
  ellipse(ilhaX, ilhaY - 20 + movimentoY, 560, 270);
```

```
  fill(corGramaAtual2);
  ellipse(ilhaX, ilhaY - 40 + movimentoY, 400, 180);

  //area da terra
  fill(corPraiaAtual);
  ellipse(ilhaX, ilhaY - 30 + movimentoY, 300, 100);

  //templo
  desenharTemplo(movimentoY);

  //caminhos na ilha
  stroke(corPraiaAtual);
  strokeWeight(12);
  line(ilhaX, ilhaY - 40 + movimentoY, ilhaX - 150, ilhaY + 20 + movimentoY);
  line(ilhaX, ilhaY - 40 + movimentoY, ilhaX + 150, ilhaY + 20 + movimentoY);
  noStroke();

  //liras
  desenharLira(lirasX[0], lirasY[0] + movimentoY, 40);
  desenharLira(lirasX[1], lirasY[1] + movimentoY, 40);
  desenharLira(lirasX[2], lirasY[2] + movimentoY, 30);
  desenharLira(lirasX[3], lirasY[3] + movimentoY, 30);
}

void desenharTemplo(float movimentoY) {
  //cores do templo com base no ciclo dia/noite
  color corTemploAtual = lerpColor(corTemplo, corTemploNoite, transicaoTempo);
```

```
 color corSombraTemploAtual = lerpColor(corTemploSombra, color(100, 100, 140, 100), transicaoTempo);
 color corDetalheTemploAtual = lerpColor(corTemploDetalhe, corTemploDetalhesNoite, transicaoTempo);

 rectMode(CENTER);

 //efeito de iluminação noturna do templo
 if (transicaoTempo > 0.3) {
  float intensidadeBrilho = constrain(map(transicaoTempo, 0.3, 0.8, 0, 1), 0, 1);
  float pulsoBrilho = sin(frameCount * 0.05) * 10;

  // Aura do templo à noite
  for (int i = 0; i < 3; i++) {
   float tamAura = 170 + i*20 + pulsoBrilho;
   fill(200, 220, 255, 20 * intensidadeBrilho);
   ellipse(ilhaX, ilhaY - 80 + movimentoY, tamAura, tamAura * 0.8);
  }
 }

 //sombra do templo
 fill(corSombraTemploAtual);
 rect(ilhaX + 5, ilhaY - 75 + movimentoY, 125, 85, 20);

 //estrutura principal do templo
 fill(corTemploAtual);
 rect(ilhaX, ilhaY - 80 + movimentoY, 120, 80, 20);

 //colunas
```

```
  fill(corDetalheTemploAtual);
  rect(ilhaX - 45, ilhaY - 80 + movimentoY, 15, 80);
  rect(ilhaX + 45, ilhaY - 80 + movimentoY, 15, 80);
  rect(ilhaX - 15, ilhaY - 80 + movimentoY, 15, 80);
  rect(ilhaX + 15, ilhaY - 80 + movimentoY, 15, 80);

  //telhado do templo
  fill(corTemploAtual);
  triangle(ilhaX - 65, ilhaY - 120 + movimentoY,
        ilhaX + 65, ilhaY - 120 + movimentoY,
        ilhaX, ilhaY - 160 + movimentoY);

  //ornamento dourado brilha mais à noite
  color corOrnamento = lerpColor(color(255, 215, 0), color(255, 230, 150),
transicaoTempo);
  fill(corOrnamento);
  ellipse(ilhaX, ilhaY - 135 + movimentoY, 15, 15);

  //luz interior do templo à noite
  if (transicaoTempo > 0.5) {
   float intensidadeLuzInterna = constrain(map(transicaoTempo, 0.5, 0.9, 0, 1), 0,
1);
   float pulsoLuz = sin(frameCount * 0.1) * 5;

   //iluminação interior
   fill(255, 240, 200, 60 * intensidadeLuzInterna);
   rect(ilhaX, ilhaY - 90 + movimentoY, 80 + pulsoLuz, 50 + pulsoLuz, 10);

   //pontos de luz nas colunas
```

```
   fill(255, 240, 150, 180 * intensidadeLuzInterna);
   ellipse(ilhaX - 45, ilhaY - 60 + movimentoY, 6 + pulsoLuz/2, 6 + pulsoLuz/2);
   ellipse(ilhaX + 45, ilhaY - 60 + movimentoY, 6 + pulsoLuz/2, 6 + pulsoLuz/2);
   ellipse(ilhaX - 15, ilhaY - 60 + movimentoY, 6 + pulsoLuz/2, 6 + pulsoLuz/2);
   ellipse(ilhaX + 15, ilhaY - 60 + movimentoY, 6 + pulsoLuz/2, 6 + pulsoLuz/2);
  }
}

void desenharLira(float x, float y, float tamanho) {
  //cores da lira baseadas no ciclo dia/noite
  color corLiraCorpo = lerpColor(color(255, 215, 0), color(230, 220, 150),
transicaoTempo);
  color corLiraInterior = lerpColor(color(200, 170, 0), color(180, 180, 100),
transicaoTempo);
  color corLiraBase = lerpColor(color(220, 190, 0), color(200, 200, 100),
transicaoTempo);
  color corAura = lerpColor(color(255, 255, 200, 30), color(200, 220, 255, 30),
transicaoTempo);

  //aura da lira
  float energiaPulso = sin(frameCount * 0.1) * 10;
  fill(corAura);
  ellipse(x, y, tamanho * 2.5 + energiaPulso, tamanho * 2.5 + energiaPulso);

  //corpo principal
  fill(corLiraCorpo);
  beginShape();
  vertex(x, y);
  bezierVertex(x - tamanho/2, y - tamanho/2, x - tamanho, y - tamanho*1.5, x -
tamanho/2, y - tamanho*2);
```

```
  vertex(x + tamanho/2, y - tamanho*2);
  bezierVertex(x + tamanho, y - tamanho*1.5, x + tamanho/2, y - tamanho/2, x, y);
  endShape(CLOSE);

  //interior da lira
  fill(corLiraInterior);
  beginShape();
  vertex(x, y);
  bezierVertex(x - tamanho/2.2, y - tamanho/2.2, x - tamanho/1.1, y - tamanho*1.4,
x - tamanho/2.2, y - tamanho*1.9);
  vertex(x + tamanho/2.2, y - tamanho*1.9);
  bezierVertex(x + tamanho/1.1, y - tamanho*1.4, x + tamanho/2.2, y - tamanho/2.2,
x, y);
  endShape(CLOSE);

  //barra central
  fill(corLiraCorpo);
  rect(x, y - tamanho, tamanho/10, tamanho*1.6);

  //cordas
  color cordasCor = lerpColor(color(255, 255, 220), color(220, 230, 255),
transicaoTempo);
  stroke(cordasCor);
  strokeWeight(1);
  for (int i = 1; i <= 3; i++) {
    float espacamento = tamanho / 4;
    line(x - tamanho/2 + i*espacamento, y - tamanho*1.8, x - tamanho/2 +
i*espacamento, y - tamanho*0.2);
  }
```

```
  noStroke();

  //base da lira
  fill(corLiraBase);
  arc(x, y, tamanho/2, tamanho/2, 0, PI);

  //brilho extra à noite
  if (transicaoTempo > 0.5) {
    float brilhoIntensidade = map(transicaoTempo, 0.5, 1.0, 0, 1);
    float brilhoPulso = sin(frameCount * 0.2) * 3;

    fill(255, 255, 240, 100 * brilhoIntensidade);
    ellipse(x, y - tamanho, tamanho/3 + brilhoPulso, tamanho/3 + brilhoPulso);
  }
}
//-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
void desenharNotasMusicais() {
  for (int i = 0; i < numNotas; i++) {
    if (notasAtivas[i]) {
      notasTempoVida[i] += velocidadeBase;

      if (mouseInsideCanvas) {
        //atualizar alvos para movimento menos suave
        notasTargetX[i] = lerp(notasTargetX[i], mouseX, 0.1);
        notasTargetY[i] = lerp(notasTargetY[i], mouseY, 0.1);
        //int offset = int(notasTempoVida[i] * 0.5);
        //int idx = (notasTrailStart[i] + offset) % trailLength;
```

```
  //PVector target = mouseTrail[idx];
  //notasTargetX[i] = target.x;
  //notasTargetY[i] = target.y;

  //seguir o mouse com física natural
  float dx = notasTargetX[i] - notasX[i];
  float dy = notasTargetY[i] - notasY[i];
  float distancia = sqrt(dx*dx + dy*dy);

  //ajusta velocidade baseada na distância
  float easing = map(distancia, 0, 500, 0.05, 0.2);
  notasAceleracao[i] = min(notasAceleracao[i] * 1.01, 0.5);

  //aplica movimento com aceleração - velocidade normal com mouse dentro
  notasX[i] += dx * easing * notasAceleracao[i] * 3.0;
  notasY[i] += dy * easing * notasAceleracao[i] * 3.0;

  //atualiza rotação para acompanhar movimento
  notasRotacao[i] = atan2(dy, dx) + HALF_PI;

 } else {
  //movimento ondulado para cima com física melhorada
  notasAceleracao[i] = max(notasAceleracao[i] * 0.99, 0.01);
  notasVel[i] += notasAceleracao[i] * 0.1;
  notasY[i] -= notasVel[i];

  //movimento lateral em onda mais complexo
  float waveFactor = sin(notasOndulacao[i] + notasTempoVida[i] * 0.05) * 1.8;
```

```
  notasX[i] += waveFactor;

  //rotação variável
  notasRotacao[i] += 0.02 + waveFactor * 0.01;
}

//verificar se está fora da tela ou se expirou
if (notasY[i] < -50 || notasY[i] > height + 50 ||
    notasX[i] < -50 || notasX[i] > width + 50 ||
    notasTempoVida[i] > 300) {
  notasAtivas[i] = false;
  continue;
}

//efeito de fade out e resize
float alpha = 255;
float tamanhoAtual = notasTamanho[i];

if (notasTempoVida[i] > 250) {
  alpha = map(notasTempoVida[i], 250, 300, 255, 0);
  tamanhoAtual *= map(notasTempoVida[i], 250, 300, 1, 0.5);
} else if (notasTempoVida[i] < 20) {
  //efeito de surgimento
  alpha = map(notasTempoVida[i], 0, 20, 0, 255);
  tamanhoAtual *= map(notasTempoVida[i], 0, 20, 0.5, 1);
}

//cor do brilho das notas baseado no ciclo dia/noite
```

```
    color corBrilhoNotas = lerpColor(color(255, 255, 200, 80), color(200, 220, 255, 80), transicaoTempo);

    //efeito de brilho pulsante
    if (notasTempoVida[i] % 30 < 15) {
      float pulso = sin(notasTempoVida[i] * 0.2) * 5;
      fill(corBrilhoNotas);
      ellipse(notasX[i], notasY[i], tamanhoAtual * 1.5 + pulso, tamanhoAtual * 1.5 + pulso);
    }

    //cor das notas baseada no ciclo dia/noite
    float tingimentoNotas = transicaoTempo * 30; // Quão azuladas ficam à noite

    //desenhar nota
    pushMatrix();
    translate(notasX[i], notasY[i]);
    rotate(notasRotacao[i]);

    //aplicar azulado à noite nas notas
    tint(255 - tingimentoNotas, 255 - tingimentoNotas/2, 255, alpha);
    image(notasImagens[notasTipo[i]], 0, 0, tamanhoAtual, tamanhoAtual);
    noTint();
    popMatrix();
   }
  }
}

//-----------------------------------------------------------------------------------------
```

```
//------------------------------


//tratamento de evento de clique
void mousePressed() {
 if (mouseX > 0 && mouseX < width && mouseY > 0 && mouseY < height) {
  //inicia efeito de afastamento das nuvens
  efeitoClique = true;
  cliqueX = mouseX;
  cliqueY = mouseY;
  cliqueRaio = 0;


  //criar algumas notas extras no local do clique
  for (int i = 0; i < 5; i++) {
   int liraAleatoria = int(random(4));
   lirasEnergia[liraAleatoria] = lirasProxNota[liraAleatoria];  //força a criação de
notas
  }


  //interagir com estrelas faz as estrelas próximas brilharem mais
  if (transicaoTempo > 0.5) { // Se for noite
   for (int i = 0; i < numEstrelas; i++) {
    float dist = dist(mouseX, mouseY, estrelasX[i], estrelasY[i]);
    if (dist < 100) {
     estrelasBrilho[i] = min(estrelasBrilho[i] + (100 - dist), 255);
     estrelasTam[i] = min(estrelasTam[i] * 1.5, 10);
    }
   }
  }
```

```
  }
}

//-------------------------------------------------------------------------------------
//------------------------------
//Hacks

//forçar a mudança de ciclo com tecla 'c'
void keyPressed() {
  if (key == 'c' || key == 'C') {
    //inverte dia/noite
    ehDia = !ehDia;
    //sinaliza que estamos em transição
    emTransicao = true;
    //define de onde a transição parte
    transicaoTempo = ehDia ? 1 : 0;
    //reset o próximo ciclo para daqui a duracaoCicloBase segundos
    proximaMudancaTempo = tempoPassado + duracaoCicloBase;
  }
}



//-------------------------------------------------------------------------------------
//------------------------------

//reconfigura a animação se a janela for redimensionada
void windowResized() {
  if (width > 0 && height > 0) {
```

```
  //atualizar posições quando o tamanho da janela mudar
  ilhaX = width / 2;
  ilhaY = height / 2;
  solX = width - 120;
  solY = 120;
  luaX = 120;
  luaY = 120;

  //atualizar posições das liras
  lirasX[0] = ilhaX - 200;
  lirasY[0] = ilhaY;
  lirasX[1] = ilhaX + 200;
  lirasY[1] = ilhaY;
  lirasX[2] = ilhaX - 100;
  lirasY[2] = ilhaY + 50;
  lirasX[3] = ilhaX + 100;
  lirasY[3] = ilhaY + 50;

  //reposicionar estrelas
  for (int i = 0; i < numEstrelas; i++) {
    estrelasX[i] = random(width);
    estrelasY[i] = random(height * 0.8);
  }
 }
}
```