TetrisCV

Tetris realizado en processing con controles gestuales

Yago García Rodríguez

UO257178

TetrisCV

Hardware necesario

Para la ejecución de este proyecto solo es necesario tener en posesión un ordenador capaz de ejecutar una aplicación realizada en Processing (al menos 512MB de memoria RAM) y una webcam con una calidad aceptable.

Limitaciones importantes

Será necesario un ambiente con una luminosidad adecuada y sobretodo, uniforme para que los colores no cambien con respecto a la posición en la que se encuentran.

Es recomendable que el fondo de la imagen sea blanco y que el color elegido para controlar las piezas sea único. Es decir que en la escena solo este ese objeto con ese color.

Problemas encontrados:

Originariamente el juego estaba pensado para ser controlado simplemente con la mano, es decir, según la posición de la mano, poder controlar las piezas. Esto finalmente no fue posible, ya que no se encontró ninguna "Haar Cascade" de una palma o un puño que funcionara correctamente para OpenCV. Posteriormente se probó con una "Haar Cascade" de la cara. Esta funcionaba muchísimo mejor, pero, aun así, estaba lejos de ser una experiencia "deseable". Tener que mover la cabeza y a veces el cuerpo entero para permitir jugar al juego se convertía en una verdadera odisea, ademas de que al mínimo giro de cabeza, esta, dejaba de ser detectada.

Para solucionar estos problemas se decidió optar por una solución drásticamente distinta: En concreto se trata de que el jugador sostenga una pieza o material de color llamativo que permita mover la pieza al igual que lo haría con la mano.

Código:

```
//Filas y columnas del tablero.
int filas, columnas;
//Objeto figura
Figura figura;
// Cada cuadrado del tablero será un objeto de tipo cuadrado
Cuadrado[][] cuadrados;
int tiempoInicio, tiempoActual;
boolean acabado = false;
boolean preparado = false;
boolean jugadorOk = false;
boolean pausa = false;
boolean seHaMovido = false;
color colorPrincipal;
int manoX = 0;
int manoY = 0;
int thresholdColor = 20;
boolean hayColor = false;
boolean situado = false;

void settings() {
  size(400,800);
}

void setup()
{

    String[] args = {"TwoFrameTest"};
    Movimiento sa = new Movimiento();
    PApplet.runSketch(args, sa);
    background(255, 204, 0);
    textSize(32);
    fill(0,0,127);
```

```
    //se permiten distintos tañamos de pantalla sin problema
  //size(400, 800);
  //Numero de filas que se pintan,
  filas = height / 25;
  //Numero de columnas
  columnas = width / 25;
  //Array que contendrá todos los cuadrados del tablero.
  cuadrados = new Cuadrado[columnas][filas];
  for (int i = 0; i < cuadrados.length; i++)
  {
    for (int j = 0; j < cuadrados[i].length; j++)
    {
      cuadrados[i][j] = new Cuadrado(i, j, false);
    }
  }
  //Nueva forma
  figura = new Figura();
}
void draw()
{
  background(50);
   if(preparado == false) {
    textSize(32);
    fill(0,0,0);
    text("El juego está cargando", 20, 200);
  }
  else if(preparado && !jugadorOk){
    textSize(20);
    fill(0,0,0);
    text("Pulse el color a seguir\nSitue la mano en el cuadro central\nPulse E para empezar",
15, 200);
  }
  else if(preparado && jugadorOk && pausa){
```

```
  textSize(32);
  fill(0,0,0);
  text("Menú de pausa", 50, 200);
  text("Presiona P para seguir", 20, 250);
 } else if(preparado && jugadorOk && !pausa){
if (frameCount % 10 == 0)
{
  figura.abajo();
}

for (int j = 0; j < cuadrados[0].length; j++)
{
  boolean libre = true;
  for (int i = cuadrados.length - 1; i >= 0; i--)
  {
    cuadrados[i][j].show();
    //Si el cuadrado esta lleno no pasa la forma
    if (cuadrados[i][j].isForma == false)
    {
      libre = false;
    }
  }
  //Si está libre la pintamos
  if (libre)
  {
    for (int k = j; k > 0; k--)
    {
      for (int l = 0; l < cuadrados.length; l++)
      {
        cuadrados[l][k].isForma = cuadrados[l][k-1].isForma;
        cuadrados[l][k].col = cuadrados[l][k-1].col;
      }
    }
```

```
    }
  }
  figura.show();
  if(acabado)
    acabado();
  }
}
void acabado(){
   textSize(60);
    fill(127,0,0);
    text("Perdiste", 75, 200);
    textSize(30);
    fill(0,0,127);
    text("Presiona j para reiniciar", 40, 260);
}
void reiniciar(){
    for (int i = 0; i < cuadrados.length; i++)
  {
    for (int j = 0; j < cuadrados[i].length; j++)
    {
      cuadrados[i][j] = new Cuadrado(i, j, false);
    }
  }
  draw();
}
void keyPressed() {
  if (key == 'j' && acabado) {
    reiniciar();
  }
  if (key == 'e' && preparado && hayColor && situado) {
    jugadorOk = true;
  }
  if (key == 'p' && preparado && jugadorOk) {
```

```
    pausa = !pausa;
  }
}

//Clase Cuadrado
class Cuadrado
{
  //Saber si hay figura en el cuadrado o no
  boolean isForma;
  //Poscion del cuadrado en el tablero
  int x, y;
  //rellenaremos todos los cuadrados del mismo color excepto los que tengan figura
  color col;


  Cuadrado(int x, int y, boolean isForma)
  {
    this.x = x;
    this.y = y;
    this.isForma = isForma;
    col = color(0);
  }

  void show()
  {
    strokeWeight(1);
    if (isForma == true)
    {
      strokeWeight(3);
    }
    stroke(0);
    fill(col);
    rect(x * 25, y * 25, 25, 25);
```

```
  }
}

//Clase forma, hay diversos tipos de forma todas requieren de al menos cuatro
//cuadrados del tablero.
class Figura
{
  Cuadrado[] cuadradosForma = new Cuadrado[4];
  //Constructor por defecto, devuelve una nueva forma ( aleatoria )
  Figura()
  {
    nuevaFormaAleatoria();
  }
  //Envía la forma al fondo
  boolean abajo()
  {
    //Si choca no se mueve mas
    boolean valido = true;
    for (int i = 0; i < 4; i++)
    {
      if (cuadradosForma[i].y >= filas-1 || cuadrados[cuadradosForma[i].x]
[cuadradosForma[i].y+1].isForma)
      {
        valido = false;
        break;
      }
    }

    for (int i = 0; i < 4 && valido; i++)
    {
      cuadradosForma[i].y++;
    }

    if (valido == false)
```

```
  {
   for (int i = 0; i < 4; i++)
   {
    int ejex = cuadradosForma[i].x;
    int ejey = cuadradosForma[i].y;
    if(ejex <0 || ejey <0){
        acabado = true;
    } else{
    cuadrados[ejex][ejey].isForma = true;
    cuadrados[ejex][ejey].col = cuadradosForma[i].col;
   }}
   nuevaFormaAleatoria();
  }
  return valido;
}

void nuevaFormaAleatoria()
{
  //Todas las formas tienen al menos un cuadrado lleno..
  cuadradosForma[0] = new Cuadrado(columnas/2, 0, true);
  //Generamos aleatorio
  int type = floor(random(7));
  //Según el aleatorio se generará un tipo de forma
  switch(type)
  {
  case 0:  //Cuadrado
    cuadradosForma[1] = new Cuadrado(columnas/2 - 1, 0, true);
    cuadradosForma[2] = new Cuadrado(columnas/2 - 1, -1, true);
    cuadradosForma[3] = new Cuadrado(columnas/2, -1, true);
    break;
  case 1: //J
    cuadradosForma[1] = new Cuadrado(columnas/2 - 1, -1, true);
    cuadradosForma[2] = new Cuadrado(columnas/2 - 1, 0, true);
```

```
  cuadradosForma[3] = new Cuadrado(columnas/2 + 1, 0, true);
  break;
case 2: //Forma de L
  cuadradosForma[1] = new Cuadrado(columnas/2 - 1, 0, true);
  cuadradosForma[2] = new Cuadrado(columnas/2 + 1, -1, true);
  cuadradosForma[3] = new Cuadrado(columnas/2 + 1, 0, true);
  break;
case 3: //Forma de I
  cuadradosForma[1] = new Cuadrado(columnas/2 - 2, 0, true);
  cuadradosForma[2] = new Cuadrado(columnas/2 - 1, 0, true);
  cuadradosForma[3] = new Cuadrado(columnas/2 + 1, 0, true);
  break;
case 4: //S
  cuadradosForma[1] = new Cuadrado(columnas/2 - 1, 0, true);
  cuadradosForma[2] = new Cuadrado(columnas/2, -1, true);
  cuadradosForma[3] = new Cuadrado(columnas/2 + 1, -1, true);
  break;
case 5: //T
  cuadradosForma[1] = new Cuadrado(columnas/2, -1, true);
  cuadradosForma[2] = new Cuadrado(columnas/2 - 1, -1, true);
  cuadradosForma[3] = new Cuadrado(columnas/2 + 1, -1, true);
  break;
case 6: //Parecida a la Z
  cuadradosForma[1] = new Cuadrado(columnas/2 + 1, 0, true);
  cuadradosForma[2] = new Cuadrado(columnas/2, -1, true);
  cuadradosForma[3] = new Cuadrado(columnas/2 - 1, -1, true);
  break;
}
for (int i = 0; i < 4; i++)
{
  //Los nuevos son blancos
  cuadradosForma[i].col = color(255);
  //Ponemos los cuadrados a true porque hay forma
```

```
    cuadradosForma[i].isForma = true;
  }
 }

 void gira()
 {
  if(!seHaMovido){
  boolean valido = true;
  //Giramos a partir del 1 ya que siempre hay un cuadrado de cada figura que no se mueve
  for (int i = 1; i < 4; i++)
  {
   int nuevox = (cuadradosForma[i].x - cuadradosForma[0].x); //Giro
   int nuevoy = (cuadradosForma[i].y - cuadradosForma[0].y); //Giro
   int x = cuadradosForma[0].x + nuevoy;
   int y = cuadradosForma[0].y - nuevox;
   if (x < 0 || x >= columnas || y < 0 || y >= filas || cuadrados[x][y].isForma == true)
   {
    //choque
    valido = false;
   }
  }
  //Giramos a partir del 1 ya que siempre hay un cuadrado de cada figura que no se mueve
  for (int i = 1; i < 4 && valido; i++)
  {
   int dx = (cuadradosForma[i].x - cuadradosForma[0].x); //Giro
   int dy = (cuadradosForma[i].y - cuadradosForma[0].y); //Giro
   int x = cuadradosForma[0].x + dy;
   int y = cuadradosForma[0].y - dx;
   cuadradosForma[i].x = x;
   cuadradosForma[i].y = y;
  }
  seHaMovido = true;
  }
```

```
}

  void derecha()
  {

    boolean valido = true;
    for (int i = 0; i < 4; i++)
    {
      if (cuadradosForma[i].y >= 0 && (cuadradosForma[i].x >= columnas - 1 ||
cuadrados[cuadradosForma[i].x+1][cuadradosForma[i].y].isForma))
      {
        //choque
        valido = false;
        break;
      }
    }

    for (int i = 0; i < 4 && valido; i++)
    {
      //Movemos a la derecha
      cuadradosForma[i].x++;
    }
    seHaMovido = true;

  }


  void izquierda()
  {

    boolean valido = true;
    for (int i = 0; i < 4; i++)
    {
      //choque
```

```
    if (cuadradosForma[i].y >= 0 && (cuadradosForma[i].x <= 0 ||
cuadrados[cuadradosForma[i].x-1][cuadradosForma[i].y].isForma))
    {
      valido = false;
      break;
    }
  }

  for (int i = 0; i < 4 && valido; i++)
  {
    //Movemos a la izquierda
    cuadradosForma[i].x--;
  }
  seHaMovido = true;
  }



 void suelo()
 {
 //  if(!seHaMovido){
   //Si choca no se mueve mas
   boolean valido = true;
   for (int i = 0; i < 4; i++)
   {
    if (cuadradosForma[i].y >= filas-1 || cuadrados[cuadradosForma[i].x]
[cuadradosForma[i].y+1].isForma)
    {
      valido = false;
      break;
    }
  }

  for (int i = 0; i < 4 && valido; i++)
```

```
    {
      cuadradosForma[i].y++;
    }

    if (valido == false)
    {
      for (int i = 0; i < 4; i++)
      {
        int ejex = cuadradosForma[i].x;
        int ejey = cuadradosForma[i].y;
        if(ejex <0 || ejey <0){
            acabado = true;
        } else{
        cuadrados[ejex][ejey].isForma = true;
        cuadrados[ejex][ejey].col = cuadradosForma[i].col;
      }}
      nuevaFormaAleatoria();
    }
    seHaMovido =true;
    //}
  }

  void show()
  {
    for (int i = 0; i<4; i++)
    {
      cuadradosForma[i].show();
    }
  }
}
```

```
import gab.opencv.*;
```

```
import processing.video.*;
import java.awt.*;

public class Movimiento extends PApplet {

  Capture video;


  public void settings() {
    size(640,360);
  }

  void setup() {
    video = new Capture(this, 640,360);
    tiempoActual = 0;
    tiempoInicio = millis();
    video.start();
  }

  void draw() {
    video.loadPixels();
    preparado = true;
    //scale(0.5);

    image(video, 0, 0 );

    float mejorColor = 500;
  //Recorremos todos los pixeles del video y comprobamos uno a uno la distancia de nuestro color con el del pixel, el que mas se
  //Acerque de todos será el mejor color, siempre y cuando el mejor color tenga una distancia menor de la variable threshold con el original
  for (int x = 0; x < video.width; x ++ ) {
    for (int y = 0; y < video.height; y ++ ) {
      int localizacion = x + y*video.width;
```

```
    color ColorPixelActual = video.pixels[localizacion];
    float r1 = red(ColorPixelActual);
    float g1 = green(ColorPixelActual);
    float b1 = blue(ColorPixelActual);
    float r2 = red(colorPrincipal);
    float g2 = green(colorPrincipal);
    float b2 = blue(colorPrincipal);

    // Funcion euclidea que determina la distancia entre dos colores
    float d = dist(r1, g1, b1, r2, g2, b2);
    //Si es el mejor color guardamos la distancia.
    if (d < mejorColor) {
      mejorColor = d;
      manoX = x;
      manoY = y;
    }
  }
}
//Pintamos localización del mejor color
if (mejorColor < thresholdColor) {
  strokeWeight(4.0);
  stroke(0);
  ellipse(manoX, manoY, 16, 16);
}

    if(manoX>0 && manoX<=640 && manoY<=120) {
        figura.gira();

      }
    else if(manoX>220 && manoX<=420 && manoY>=240) {
        figura.suelo();

      }
```

```
    else if(manoY>0 && manoY<=360 && manoX<=240) {
      figura.izquierda();

    }
    else if(manoY>0 && manoY<=360 && manoX>=420) {
      figura.derecha();

    }
    else{
      seHaMovido = false;
      situado = true;
    }
  //Lineas de campo
   line(0, 120, 640, 120);
   line(220, 240, 420, 240);
   line(220, 120, 220, 360);
   line(420, 120, 420, 360);
  }

  void captureEvent(Capture c) {
    c.read();
  }
  void mousePressed() {
  // Guardar el color en el que hacemos click
  int localizacion = mouseX + mouseY*video.width;
  colorPrincipal = video.pixels[localizacion];
  hayColor=true;
}
}
```

El Código podría ser dividido en 3 partes.

1. La primera podría ser la lógica del juego (modos: pausa, juego, acabado, inicio…)

2. Otra sería la clase Cuadrado: El tablero estaría dividido en cuadrados, estos cuadrados pueden estar o no ocupados por una pieza y tienen una posición en el tablero.

3. Pieza: clase fundamental, incluye todos los movimientos. Se genera aleatoriamente a sí misma.

4. Movimiento: Según la posición de la pieza en la cámara controlará los movimientos de la pieza.

Mejoras:

- Mejorar la precisión de la lectura de colores.
- Añadir puntuación.
- Hacer que sea controlable con la mano y no con un objeto
- Quizás no es el juego idóneo para ser controlado con gestos.
- Añadir musica
- Piezas de colores.

Información relevante:

Puede seguirse el avance del proyecto a través del siguiente repositorio de GitHub:

- `https://github.com/yagogr98/TetrisCV.git`