TETRIS GAME

PROYECTO

CONTENIDO

- OBJETIVOS
- CODIGO
- PROYECTO FISICO
- RESULTADOS Y CONCLUSIONES

OBJETIVOS

OBJETIVOS

Programar el videojuego Tetris mediante el microcontrolador STM32L432, junto con un sensor flexible, joystick y pulsadores para controlar el juego en una pantalla display


CODIGO

VOID SETUP

```
oid setup() {

 pinMode(2, INPUT_PULLUP);
 pinMode(3, INPUT_PULLUP);
 pinMode(4, INPUT_PULLUP);
 pinMode(5, INPUT_PULLUP);

 Serial.begin(9600);
 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
   Serial.println(F("SSD1306 allocation failed"));
   for(;;);
 }

 display.setRotation(1);
 display.clearDisplay();

 display.drawBitmap(3, 23, mantex_logo, 64, 82,  WHITE);
 display.display();
 delay(2000);
 display.clearDisplay();

 drawLayout();

 display.display();

 randomSeed(analogRead(0));
 nextType = random(TYPES);
 generate();

 timer = millis();
```

LIBRERIAS

```
1  #include <Wire.h>
2  #include <Adafruit_GFX.h>
3  #include <Adafruit_SSD1306.h>
4
5  #define WIDTH 64 // OLED display width, in pixels
6  #define HEIGHT 128 // OLED display height, in pixels
7
8  Adafruit_SSD1306 display(128, 64, &Wire, -1);
9
```

VOID LOOP

```
163 void loop() {
164   Serial.println(analogRead(A0));
165
166   if(millis() - timer > interval){
167     checkLines();
168     refresh();
169
170     if(nextCollision()){
171       for(short i = 0; i < 4; i++)
172         grid[pieceX + piece[0][i]][pieceY + piece[1][i]] = 1;
173       generate();
174     }else
175       pieceY++;
176
177     timer = millis();
178   }
```

FUNCIONES

```
204   if(analogRead(A1)<400){
205     int val=(analogRead(A1)-400)*0.5;
206       if(!nextHorizontalCollision(piece, 1)){
207         pieceX=pieceX+1;
208         refresh();
209       }
210   }
211 if(!digitalRead(2)){
212     if(b2){
213       if(!nextHorizontalCollision(piece, 1)){
214         pieceX++;
215         refresh();
216       }
217       b2 = false;
218     }
219   }else{
220     b2 = true;
221   }
```

Movimiento a la derecha

```
234   if(!digitalRead(9)){
235     if(b3){
236       if(rotation == getMaxRotation(currentType) - 1 && canRotate(0)){
237         rotation = 0;
238       }else if(canRotate(rotation + 1)){
239         rotation++;
240       }
241
242       copyPiece(piece, currentType, rotation);
243       refresh();
244
245       b3 = false;
246       delayer = millis();
247     }
248   }else if(millis() - delayer > 50){
249     b3 = true;
250   }
```

Funcion para bajar rapido

```
if(!digitalRead(9)){
  if(b3){
    if(rotation == getMaxRotation(currentType) - 1 && canRotate(0)){
      rotation = 0;
    }else if(canRotate(rotation + 1)){
      rotation++;
    }

    copyPiece(piece, currentType, rotation);
    refresh();

    b3 = false;
    delayer = millis();
  }
}else if(millis() - delayer > 50){
  b3 = true;
}
```

Cambio de forma

FUNCIONES

```
void checkLines(){
  boolean full;
  for(short y = 17; y >= 0; y--){
    full = true;
    for(short x = 0; x < 10; x++){
      full = full && grid[x][y];
    }
    if(full){
      breakLine(y);
      y++;
    }
  }
}
```

```
void breakLine(short line){
  for(short y = line; y >= 0; y--){
    for(short x = 0; x < 10; x++){
      grid[x][y] = grid[x][y-1];
    }
  }
  for(short x = 0; x < 10; x++){
     grid[x][0] = 0;
  }
  display.invertDisplay(true);
  delay(50);
  display.invertDisplay(false);
  score += 10;
}
```

```
void refresh(){
    display.clearDisplay();
    drawLayout();
    drawGrid();
    drawPiece(currentType, 0, pieceX, pieceY);
    display.display();
}
```

```
void drawGrid(){
  for(short x = 0; x < 10; x++)
    for(short y = 0; y < 18; y++)
      if(grid[x][y])
        display.fillRect(MARGIN_LEFT + (SIZE + 1)*x, MARGIN_TOP + (SIZE + 1)*y, SIZE, SIZE, WHITE);
}
```

```
boolean nextHorizontalCollision(short piece[2][4], int amount){
  for(short i = 0; i < 4; i++){
    short newX = pieceX + piece[0][i] + amount;
    if(newX > 9 || newX < 0 || grid[newX][pieceY + piece[1][i]])
      return true;
  }
  return false;
}
```

FUNCIONES

```
boolean nextCollision(){

  for(short i = 0; i < 4; i++){
    short y = pieceY + piece[1][i] + 1;
    short x = pieceX + piece[0][i];
    if(y > 17 || grid[x][y])
      return true;
  }

  return false;
}

void generate(){
  currentType = nextType;
  nextType = random(TYPES);
  if(currentType != 5)
    pieceX = random(9);
  else
    pieceX = random(7);
  pieceY = 0;
  rotation = 0;
  copyPiece(piece, currentType, rotation);
}
```

```
void copyPiece(short piece[2][4], short type, short rotation){
  switch(type){
  case 0: //L_l
    for(short i = 0; i < 4; i++){
      piece[0][i] = pieces_L_l[rotation][0][i];
      piece[1][i] = pieces_L_l[rotation][1][i];
    }
    break;
   case 1: //S_l
    for(short i = 0; i < 4; i++){
      piece[0][i] = pieces_S_l[rotation][0][i];
      piece[1][i] = pieces_S_l[rotation][1][i];
    }
    break;
   case 2: //S_r
    for(short i = 0; i < 4; i++){
      piece[0][i] = pieces_S_r[rotation][0][i];
      piece[1][i] = pieces_S_r[rotation][1][i];
    }
    break;
   case 3: //Sq
    for(short i = 0; i < 4; i++){
      piece[0][i] = pieces_Sq[0][0][i];
      piece[1][i] = pieces_Sq[0][1][i];
    }
    break;
```

```
short getMaxRotation(short type){
  if(type == 1 || type == 2 || type == 5)
    return 2;
  else if(type == 0 || type == 4)
    return 4;
  else if(type == 3)
    return 1;
  else
    return 0;
}

boolean canRotate(short rotation){
  short piece[2][4];
  copyPiece(piece, currentType, rotation);
  return !nextHorizontalCollision(piece, 0);
}

void drawLayout(){
  display.drawLine(0, 15, WIDTH, 15, WHITE);
  display.drawRect(0, 0, WIDTH, HEIGHT, WHITE);
  drawNextPiece();
  char text[6];
  itoa(score, text, 10);
  drawText(text, getNumberLength(score), 7, 4);
}

short getNumberLength(int n){
  short counter = 1;
  while(n >= 10){
    n /= 10;
    counter++;
  }
  return counter;
}
```

FUNCIONES

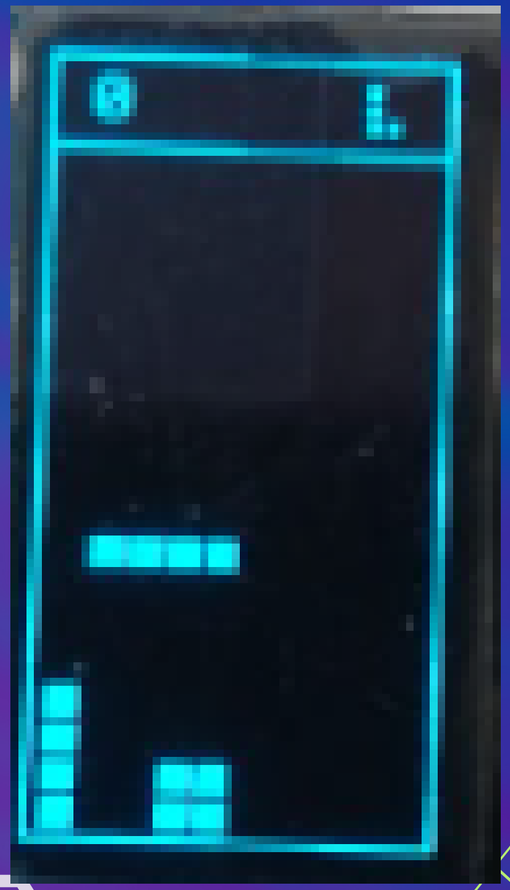
```
void drawText(char text[], short length, int x, int y){
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(x, y);
  display.cp437(true);

  for(short i = 0; i < length; i++)
    display.write(text[i]);
}
```

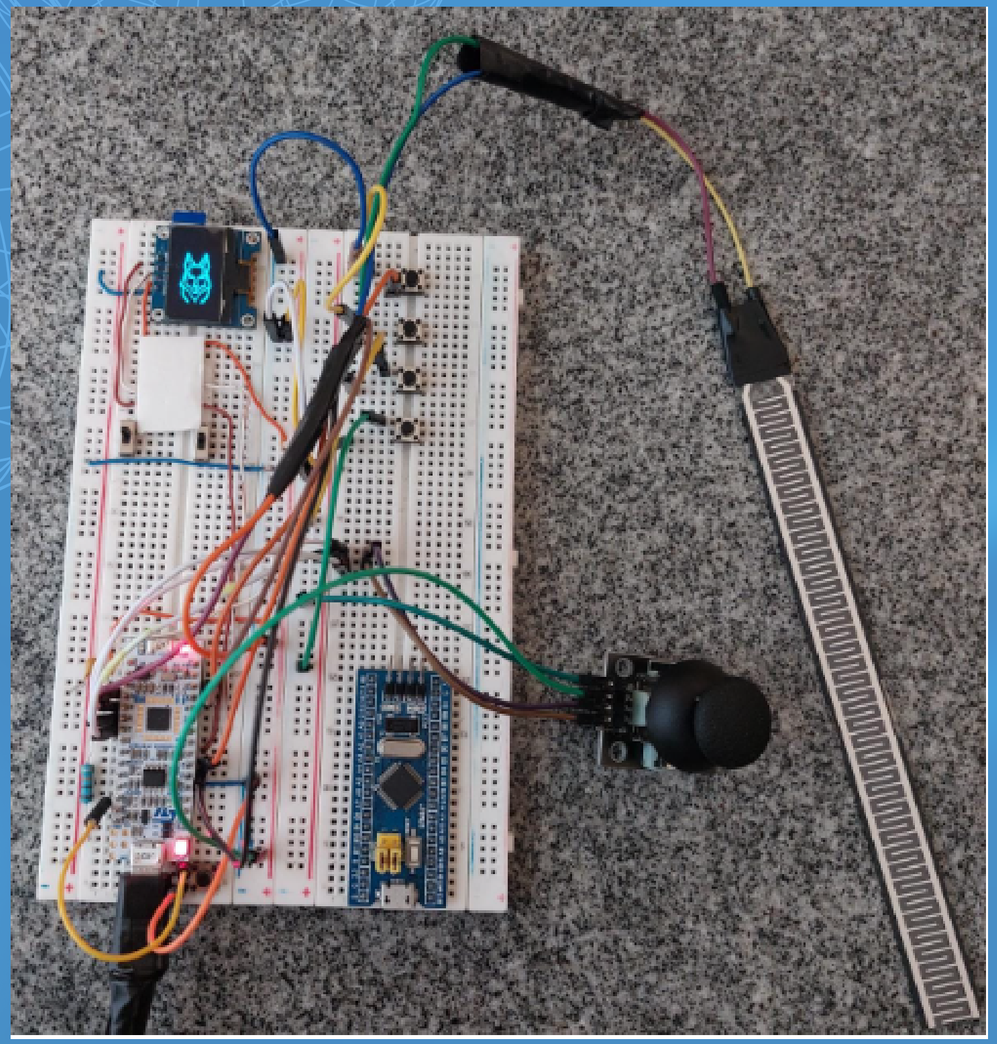
```
void drawNextPiece(){
  short nPiece[2][4];
  copyPiece(nPiece, nextType, 0);
  for(short i = 0; i < 4; i++)
    display.fillRect(50 + 3*nPiece[0][i], 4 + 3*nPiece[1][i], 2, 2, WHITE);
}
```

PROYECTO EN FISICO

PANTALLA OLED



CONEXION



RESULTADOS
Y
CONCLUSIONES