








Tutorial para implementar el juego TETRIS minimalista en Arduino

labtec@umce.cl
jonnhatan.garcia@umce.cl

Materiales necesarios

<ul style="list-style-type: none">1 Arduino UNO	
<ul style="list-style-type: none">1 Display 4 Matriz De Puntos Led 8x8 Max7219	
<ul style="list-style-type: none">1 Buzzer	<p>KY-006</p> 
<ul style="list-style-type: none">1 Módulo de joystick analógico de 3 ejes para Arduino	
<ul style="list-style-type: none">Cables varios	

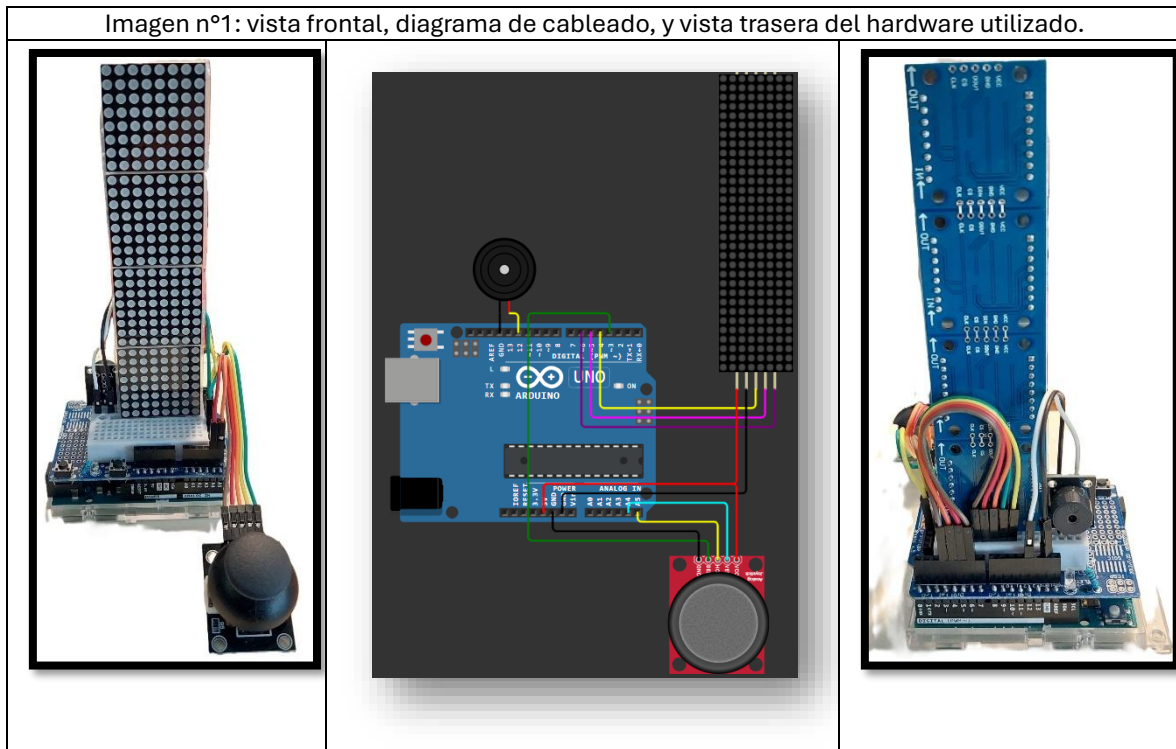
Este es el famoso juego “Snake” para 4 matrices MAX7219 dispuesto para usar la totalidad de la matriz led. La matriz está modelada con 8 columnas y 32 filas. Controlado con joystick analógico. Utiliza la librería LedControl para mostrar la serpiente y la comida. Incluye lógica para movimiento, colisiones, crecimiento de la serpiente y puntuación. Efectos de sonido y animaciones para comer comida y fin de juego.

Características generales

Todos los juegos están diseñados para ser minimalistas y funcionar con el mismo hardware sencillo y de bajo costo. El documento incluye el código fuente completo en Arduino para cada juego, con comentarios explicativos en español. Se detallan las funciones principales, la inicialización de hardware, la gestión de entradas y salidas, y la visualización en matrices LED. Se utilizan animaciones y sonidos para mejorar la experiencia de juego.

Cómo implementar el juego SNAKE minimalista en Arduino

Imagen n°1: vista frontal, diagrama de cableado, y vista trasera del hardware utilizado.



Hardware necesario

Construye el circuito de acuerdo con el diagrama de cableado mostrado en la imagen n°1. Los pines que controlan la matriz led configurada de forma vertical son: DIN_PIN 4, CS_PIN 5, CLK_PIN 6. La matriz fue configurada para que las piezas se muevan de forma vertical, por lo tanto la matriz dispone de 8 columnas y 32 filas para el movimiento de las piezas del juego. Los pines del joystick se conectan a: JOY_X (salida horizontal) al pin analógico A5; y JOY_Y (salida vertical) al pin analógico A4, y el botón de



rotación (botón integrado SW del joystick) se conecta al pin digital 3 del Arduino UNO. El Buzzer se conecta al pin digital 12 de la placa y a GND.

Estructura básica del código del juego SNAKE

Inicialización

- Se usan las librerías LedControl para controlar las matrices LED.
- Se definen los pines para el joystick y el buzzer.
- Se inicializan las matrices LED y la posición inicial de la serpiente y la comida.

Variables principales

- **snakeX, snakeY:** Coordenadas de la cabeza de la serpiente.
- **foodX, foodY:** Coordenadas de la comida.
- **score:** Puntuación acumulada.
- **snakeSize:** Tamaño de la serpiente (número de segmentos).
- **tailX[], tailY[]:** Arrays para guardar las posiciones de la cola.
- **direction:** Dirección actual ('l', 'r', 'u', 'd').
- **isGameOver:** Indicador de fin de juego.

Lógica del juego

Bucle principal (loop)

- Si el juego ha terminado, se muestra la animación de fin y se reinician variables.
- Si no, se muestra una cuenta regresiva antes de empezar y se ejecuta la lógica principal.

Lógica principal

- **Lectura del joystick:** Se detecta la dirección en la que se mueve el joystick para cambiar la dirección de la serpiente.
- **Movimiento:** Se actualiza la posición de la cabeza según la dirección.
- **Colisiones:** Se verifica si la serpiente choca consigo misma (fin de juego) o con los límites (la serpiente aparece por el lado opuesto).
- **Comida:** Si la serpiente come la comida, se incrementa el tamaño y la puntuación, y se genera una nueva comida en posición aleatoria.
- **Dibujo:** Se dibuja la serpiente y la comida en la matriz LED.

Animaciones y sonidos

- Se usan funciones para mostrar animaciones de cuenta regresiva y fin de juego.
- Se reproducen sonidos al comer comida y al terminar el juego usando el buzzer.

Funciones clave

- **setupPins():** Configura los pines del joystick.
- **setupLedBoard():** Inicializa las matrices LED.
- **setupSnakePosition():** Posiciona la serpiente al inicio.
- **setupFoodPosition():** Coloca la comida en una posición aleatoria.
- **setJoystickDirection():** Lee el joystick y actualiza la dirección.



- **changeSnakeDirection():** Cambia la posición de la cabeza según la dirección.
- **manageGameOver():** Verifica si la serpiente colisiona consigo misma.
- **manageSnakeOutOfBounds():** Evita que la serpiente salga de los límites.
- **manageEatenFood():** Verifica si la serpiente comió la comida.
- **manageSnakeTailCoordinates():** Actualiza las coordenadas de la cola.
- **drawSnake():** Dibuja la serpiente y la comida en la pantalla.
- **showCountdown():** Muestra una cuenta regresiva antes de iniciar.
- **playFoodEatenSong()** y **playGameOverSong():** Reproducen sonidos.

Sugerencias para adaptar el código

- Puedes ajustar la velocidad del juego modificando la variable velocidad.
- El tamaño máximo de la serpiente está limitado por el tamaño de los arrays tailX[] y tailY[].
- El código está comentado en español para facilitar la comprensión y modificación.

Código fuente SNAKE minimalista

```
/* Juego SNAKE minimalista para 4 matrices MAX7219 (32x8)
 * Controlado con joystick analógico.
 * Utiliza la librería LedControl para manejar las matrices LED.
 * Adaptado de varios ejemplos de Snake en Arduino.
 * by LabTec
 * octubre/2025
 */

#include <LedControl.h>          // Librería para controlar matrices LED con MAX7219

// Pines para el MAX7219
#define DIN_PIN 4                // Data IN del MAX7219
#define CS_PIN 5                 // Chip Select / LOAD del MAX7219
#define CLK_PIN 6                // Clock del MAX7219
#define NUM_OF_MATRIX 4         // Número de matrices encadenadas (cada una 8x8)

// Pines del joystick y buzzer
#define JOY_X A5                 // Entrada analógica X del joystick
#define JOY_Y A4                 // Entrada analógica Y del joystick
#define JOY_SW 3                 // Botón del joystick (switch)
#define BUZZER 12                // Pin para el buzzer (sonidos)

// Dimensiones de la pantalla total (8 columnas x 32 filas)
const int screenWidth = 8;       // Ancho físico (columnas) de la pantalla total
const int screenHeight = 32;    // Alto lógico combinado (4 matrices * 8 filas)

// Variables del juego
int snakeX, snakeY;              // Coordenadas de la cabeza de la serpiente (x=col, y=row global)
int foodX, foodY;               // Coordenadas de la comida
int score = 0;                  // Puntuación acumulada
int rapidez = 100;              // Ajusta de rapidez del juego
int snakeSize = 1;              // Tamaño inicial de la serpiente (número de segmentos)
int velocidad = 100;            // Ajusta la rapidez del juego
int tailX[100], tailY[100];    // Arrays con coordenadas de los segmentos de la cola
char direction = 'd';           // Dirección actual: 'l','r','u','d' (inicial: 'd' = abajo)
bool isGameOver = false;        // Indicador de fin de juego
bool hasShownCountdown = false; // Controla si ya se mostró la cuenta regresiva

// Instancia del controlador de matrices LED
LedControl lc = LedControl(DIN_PIN, CLK_PIN, CS_PIN, NUM_OF_MATRIX);
```



```
void setup() {
  setupPins();           // Configurar pines de entrada/salida
  setupLedBoard();       // Inicializar las matrices LED
  setupSnakePosition();  // Posicionar la serpiente al inicio
  setupFoodPosition();   // Colocar la primera comida aleatoria
}

void loop() {
  if (isGameOver) {
    playGameOverSong();
    showGameOverScreen();
  } else {
    if (!hasShownCountdown) {
      showCountdown();
      hasShownCountdown = true;
    }
    startGame();
  }
}

// Configura los pines del joystick
void setupPins() {
  pinMode(JOY_SW, INPUT);
  digitalWrite(JOY_SW, HIGH);
}

// Inicializa todas las matrices LED
void setupLedBoard() {
  for (int i = 0; i < NUM_OF_MATRIX; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 8);
    lc.clearDisplay(i);
  }
}

// Posición inicial de la serpiente
void setupSnakePosition() {
  snakeX = 4;
  snakeY = 4;
}

// Genera una nueva posición aleatoria para la comida
void setupFoodPosition() {
  foodX = random(screenWidth);
  foodY = random(screenHeight);
}

// Lógica principal del juego
void startGame() {
  manageGameOver();
  setJoystickDirection();
  changeSnakeDirection();
  manageSnakeOutOfBounds();
  manageEatenFood();
  manageSnakeTailCoordinates();
  drawSnake();
  delay(velocidad);
}

// Verifica si la serpiente colisionó consigo misma
void manageGameOver() {
  for (int i = 1; i < snakeSize; i++) {
    if (tailX[i] == snakeX && tailY[i] == snakeY) {
```



```
isGameOver = true;
hasShownCountdown = false;
}
}
}

// Evita que la serpiente salga de los límites de la pantalla
void manageSnakeOutOfBounds() {
    if (snakeX >= screenWidth) snakeX = 0;
    else if (snakeX < 0) snakeX = screenWidth - 1;
    if (snakeY >= screenHeight) snakeY = 0;
    else if (snakeY < 0) snakeY = screenHeight - 1;
}

// Verifica si la serpiente comió la comida
void manageEatenFood() {
    if (snakeX == foodX && snakeY == foodY) {
        playFoodEatenSong();
        score++;
        snakeSize++;
        setupFoodPosition();
    }
}

// Actualiza las coordenadas de la cola de la serpiente
void manageSnakeTailCoordinates() {
    int previousX = tailX[0];
    int previousY = tailY[0];
    tailX[0] = snakeX;
    tailY[0] = snakeY;
    for (int i = 1; i < snakeSize && i < 100; i++) {
        int prevX = tailX[i];
        int prevY = tailY[i];
        tailX[i] = previousX;
        tailY[i] = previousY;
        previousX = prevX;
        previousY = prevY;
    }
}

// Lee el joystick y actualiza la dirección de movimiento
void setJoystickDirection() {
    char lastDirection = direction;
    if (analogRead(JOY_X) > 600 && lastDirection != 'u') direction = 'd';
    else if (analogRead(JOY_X) < 400 && lastDirection != 'd') direction = 'u';
    else if (analogRead(JOY_Y) > 600 && lastDirection != 'l') direction = 'r';
    else if (analogRead(JOY_Y) < 400 && lastDirection != 'r') direction = 'l';
}

// Cambia la posición de la cabeza según la dirección
void changeSnakeDirection() {
    switch (direction) {
        case 'l': snakeX--; break;
        case 'r': snakeX++; break;
        case 'u': snakeY--; break;
        case 'd': snakeY++; break;
    }
}

// Reproduce sonido al comer comida
void playFoodEatenSong() {
    tone(BUZZER, 500, 100);
}
```



```
// Reproduce secuencia sonora de fin de juego
void playGameOverSong() {
  tone(BUZZER, 100, 100); delay(100);
  tone(BUZZER, 400, 100); delay(100);
  tone(BUZZER, 800, 100); delay(100);
  tone(BUZZER, 1200, 100); delay(100);
}

// Muestra animación de fin de juego encendiendo todos los LEDs
void showGameOverScreen() {
  for (int i = 0; i < screenHeight; i++) {
    for (int j = 0; j < screenWidth; j++) {
      showLed(j, i);
      delay(10);
    }
  }
  fin();
  resetVariables();
}

// Reinicia todas las variables del juego
void resetVariables() {
  setupSnakePosition();
  setupFoodPosition();
  direction = 'd';
  isGameOver = false;
  score = 0;
  snakeSize = 1;
}

// Enciende un LED en la posición (x, y) correspondiente a la matriz
void showLed(int x, int y) {
  int matrixIndex = 3 - (y / 8); // Inversión vertical: matriz 4 arriba
  int localY = y % 8;
  lc.setLed(matrixIndex, x, localY, true);
}

// Apaga un LED en la posición (x, y)
void hideLed(int x, int y) {
  int matrixIndex = 3 - (y / 8);
  int localY = y % 8;
  lc.setLed(matrixIndex, x, localY, false);
}

// Dibuja la serpiente y la comida en la pantalla
void drawSnake() {
  for (int i = 0; i < screenHeight; i++) {
    for (int j = 0; j < screenWidth; j++) {
      if (i == snakeY && j == snakeX) {
        showLed(snakeX, snakeY);
      } else if (i == foodY && j == foodX) {
        showLed(foodX, foodY);
      } else {
        bool isShown = false;
        for (int k = 0; k < snakeSize; k++) {
          if (tailX[k] == j && tailY[k] == i) {
            showLed(j, i);
            isShown = true;
          }
        }
      }
      if (!isShown) {
        hideLed(j, i);
      }
    }
  }
}
```



```
}  
}  
}  
}  
}  
  
// Muestra una cuenta regresiva antes de iniciar el juego  
void showCountdown() {  
    for (int n = 3; n >= 1; n--) {  
        clearAllMatrices();  
        drawDigit(n);  
        delay(1000);  
    }  
    clearAllMatrices();  
}  
  
// Limpia todas las matrices  
void clearAllMatrices() {  
    for (int i = 0; i < NUM_OF_MATRIX; i++) {  
        lc.clearDisplay(i);  
    }  
}  
  
// Dibuja un número grande centrado en las matrices  
void drawDigit(int number) {  
    byte digits[3][5] = {  
        {B111, B100, B111, B100, B111}, // 3  
        {B111, B100, B111, B001, B111}, // 2  
        {B010, B011, B010, B010, B111} // 1  
    };  
  
    int startX = 2; // Posición horizontal inicial  
    int startY = 9; // Posición vertical para centrar en 32 filas  
  
    for (int row = 0; row < 5; row++) {  
        byte line = digits[3 - number][row];  
        for (int col = 0; col < 5; col++) {  
            if (bitRead(line, 2 - col)) {  
                showLed(startX + col, startY + row);  
            }  
        }  
    }  
}  
  
void fin() {  
    for (int i = 0; i < screenHeight; i++) { // Recorrer todas las filas globales  
        for (int j = 0; j < screenWidth; j++) { // Recorrer todas las columnas  
            showLed(j, i); // Encender cada LED (x = columna, y = fila)  
            delay(1); // Pequeña pausa entre LEDs para animación  
        }  
    }  
    clearAllMatrices();  
    const byte F[8] = { // F  
        B00000000, B01000000, B01001000, B01001000, B01111110, B00000000, B00000000, B00000000  
    };  
    const byte I[8] = { // I  
        B00000000, B00000000, B01000010, B01111110, B01000010, B00000000, B00000000, B00000000  
    };  
    const byte N[8] = { // N  
        B00000000, B01111110, B00001100, B00011000, B00110000, B01111110, B00000000, B00000000  
    };  
  
    // Mostrar cada letra en su matriz correspondiente
```




UMCE
el poder transformador de la educación

Laboratorio de Tecnología
Departamento de Física
labtec@umce.cl



```
for (int row = 0; row < 8; row++) {  
  lc.setRow(3, row, F[row]); // Matriz 3: F  
  lc.setRow(2, row, I[row]); // Matriz 2: I  
  lc.setRow(1, row, N[row]); // Matriz 1: N  
}  
  
// Efecto de sonido de game over  
for (int i = 0; i < 3; i++) {  
  tone(BUZZER, 300 - (i * 50), 300);  
  delay(400);  
}  
noTone(BUZZER);  
}
```

Espero que todo funcione.