Universidad de san Carlos de Guatemala
Facultad de ingeniería
Ciencias y Sistemas
Arquitectura de computadores y ensambladores I
Sección N

Manual Tecnico

Nombre	Carnet
Gustavo Alejandro Girón Arriola	201900898
Cristian Aramis López Bautista	201904042
Christopher Alexander Acajabon Gudiel	201404278
Erick Estuardo Muñoz Escalante	201602947
Jose Pablo Ceron Urizar	201908251

Al inicio de nuestro programa implementamos la librería LedControl para la manipulacion de multiples leds y se crean variables para el funcionamiento del programa.

```
#include "LedControl.h"

LedControl ledControl = LedControl(51, 53, 49, 1);
int filas[] = {24, 22, 2, 3, 4, 5, 6, 7};         // filas encienden con 0 (--> +y)
int columnas[] = {8, 9, 10, 11, 12, 13, 23, 25}; // columnas encienden con 1 (--> +x)
int DIR_LEFT = 52, DIR_RIGHT = 50, K = 48, DISP = 46;
int velocidad = 200;
boolean direccionLetrero = true; // true -> derecha
int posicionControlador = 123;   // 116 + 8 tablero = 124 => 123 porque empieza en 0

long int t0 = millis();
long int t1 = millis();
```

Se crean varias matrices de tipo byte para poder realizar manipulaciones a la matrices de led.

```
byte tableroJuego[8][16] = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

byte tableroBalas[8][16] = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
```

Para la implementación del juego se crean varias variables y arreglos con diferentes funciones, como puede ser el desplazamiento del avión en el eje X y Y, la velocidad del avión, el punteo de jugador, las vidas, la creación de la bala y las torres por derribar.

```
int tableroTorres[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int puntajes[5] = {-1, -1, -1, -1, -1};

int xAvion = 0;
int yAvion = 0;
long int tAvion0 = 0;          // velocidad avion, bala
long int tAvion1 = 0;          // velocidad avion, bala
long int tAvionDesplazY0 = 0; // desplazamiento de avion (eje y)
long int tAvionDesplazY1 = 0; // desplazamiento de avion (eje y)
bool direccionAvion = true;   // true -> derecha
bool poderDisparar = true;    // true -> puede disparar
int nivelJuego = 1;
int vidas = 3;
int puntos = 0;
bool construirNuevasTorres = true; // true -> construye torres
int cantidadTorresPorDestruir = 3;
bool nuevaPartida = true;
```

Se realiza una función para la inicialización de la matriz con y sin driver, en el caso de la matriz sin driver se declaran los pines a utilizar de tipo salida y encendiendo y apagando dichos pines. En el caso de la matriz con driver no es necesario apagar los pines.

```
void inizializarMatrizSinDriver()
{

    // Inicializando pines
    for (int i = 0; i < 8; i++)
    {
        pinMode(columnas[i], OUTPUT);
    }
    for (int i = 0; i < 8; i++)
    {
        pinMode(filas[i], OUTPUT);
    }

    // Limpiar Matriz
    for (int i = 0; i < 8; i++)
    {
        digitalWrite(columnas[i], LOW);
    }
    for (int i = 0; i < 8; i++)
    {
        digitalWrite(filas[i], HIGH);
    }
}
```

```
void inicializarMatrizDriver()
{
    ledControl.shutdown(0, false);
    ledControl.setIntensity(0, 15);
    ledControl.clearDisplay(0);
}
```

La función pintarLed recibe como parámetro los valores de X y Y y modifica en la matriz el valor de los parámetros a recibir.

```
void pintarLED(int x, int y)
{
    digitalWrite(filas[y], LOW);
    digitalWrite(columnas[x], HIGH);
    delayMicroseconds(1100);
    digitalWrite(filas[y], HIGH);
    digitalWrite(columnas[x], LOW);
}
```

En la función mostrarMatriz2 por medio de ciclos for se realiza la validación de cada posición de led en la matriz y en caso de encuentre un 1 en la matriz se va encender el led en el caso de la matriz con y sin driver.

```
void mostrarMatriz2(byte matrzXd[8][16])
{
    // Con driver
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            ledControl.setLed(0, i, j, matrzXd[i][j] == 1 ? true : false);
    // Sin driver
    for (int k = 0; k < 3; k++)
        for (int i = 0; i < 8; i++)
            for (int j = 8; j < 16; j++)
                if (matrzXd[i][j] == 1)
                    pintarLED(j - 8, i);
                else
                    delayMicroseconds(50);
}
```

La función moverCartelIZQ y moverCartelDCH hacen el fujo de movimiento de las luces de la matrices.

```
void moverCartelDCH()
{

    encenderMatrizDriver(0, posicionControlador - 8);
    encenderMatrizDriver(1, posicionControlador - 7);
    encenderMatrizDriver(2, posicionControlador - 6);
    encenderMatrizDriver(3, posicionControlador - 5);
    encenderMatrizDriver(4, posicionControlador - 4);
    encenderMatrizDriver(5, posicionControlador - 3);
    encenderMatrizDriver(6, posicionControlador - 2);
    encenderMatrizDriver(7, posicionControlador - 1);

    //    delay(1);

    encenderMatriz(posicionControlador, posicionControlador + 7);

    if (posicionControlador == -7)
    {
        posicionControlador = 123;
    }

    t1 = millis();
    if (t1 - t0 >= velocidad)
    {
        posicionControlador--;
        t0 = millis();
    }
}
```

```
void moverCartelIZQ()
{

    encenderMatrizDriver(0, posicionControlador - 8);
    encenderMatrizDriver(1, posicionControlador - 7);
    encenderMatrizDriver(2, posicionControlador - 6);
    encenderMatrizDriver(3, posicionControlador - 5);
    encenderMatrizDriver(4, posicionControlador - 4);
    encenderMatrizDriver(5, posicionControlador - 3);
    encenderMatrizDriver(6, posicionControlador - 2);
    encenderMatrizDriver(7, posicionControlador - 1);

    encenderMatriz(posicionControlador, posicionControlador + 7);

    if (posicionControlador == 126)
    {
        posicionControlador = -7;
    }

    t1 = millis();
    if (t1 - t0 >= velocidad)
    {
        posicionControlador++;
        t0 = millis();
    }
}
```

La validacion al momento de encender los leds de las matriz con y sin driver por medio de las funciones encenderMatrizDriver y encenderMatriz depende de los valores de la posicion de la columna en la que se encuentra.

```
void encenderMatrizDriver(int valorColumna, int valor)
{
    for (int valorFila = 0; valorFila < 8; valorFila++)
    { // recorremos fila de la matriz
        if (letrero[valorFila].charAt(valor) == '1')
        {
            ledControl.setLed(0, valorFila, valorColumna, true); //
        }
        else
        {
            ledControl.setLed(0, valorFila, valorColumna, false);
        }
    }
}

void encenderMatriz(int inicio, int fin)
{
    int aux = 0;
    for (int posFila = 0; posFila < 8; posFila++)
    { // recorremos filas de matriz

        digitalWrite(filas[posFila], LOW); // fila a utilizar

        for (int j = inicio; j <= fin; j++)
        { // de cada fila del letro se van leyendo 8 caraceres a l
            if (letrero[posFila].charAt(j) == '1')
            {
                digitalWrite(columnas[aux], HIGH);
            }
            else
            {
                digitalWrite(columnas[aux], LOW);
            }
            aux++;
        }

        delay(1);
        //    delayMicroseconds(1000);
        digitalWrite(filas[posFila], HIGH);
        aux = 0;
    }
```

El punteo almacenado en el juego por medio de las funciones de mostrarPunteo y mostrarPuntuacion, calculando el porcentaje sobre el total que representa cada uno de los punteos almacenados y los muestra en las dos matrices.

```
void mostrarPunteo(int punteo1, int punteo2)
{

    // MOSTRAR PUNTEO MATRIZ CON DRIVER
    for (int i = 0; i < 8; i++)
    {
        ledControl.setRow(0, i, puntuacion[punteo2][i]);
    }

    // MOSTRAR PUNTEO MATRIZ SIN DRIVER
    for (int i = 0; i < 8; i++)
    {
        if (punteo1 == 0)
        {
            mostrarPuntuacion(numero0);
        }
        else if (punteo1 == 1)
        {
            mostrarPuntuacion(numero1);
        }
        else if (punteo1 == 2)
        {
            mostrarPuntuacion(numero2);
        }
        else if (punteo1 == 3)
        {
            mostrarPuntuacion(numero3);
        }
        else if (punteo1 == 4)
        {
            mostrarPuntuacion(numero4);
        }
        else if (punteo1 == 5)
        {
            mostrarPuntuacion(numero5);
        }
```

```
void mostrarPuntuacion(int numero[8][8])
{ // numero en matriz sin driver
    for (int i = 0; i < 8; i++)
    {
        digitalWrite(columnas[i], HIGH);
        for (int j = 0; j < 8; j++)
        {
            if (numero[j][i] == 1)
            {
                digitalWrite(filas[j], LOW);
            }
        }
        delay(1);
        digitalWrite(columnas[i], LOW);
        for (int j = 0; j < 8; j++)
        {
            digitalWrite(filas[j], HIGH);
        }
    }
}
```

Durante el juego, las funciones para encender los leds para formar un avion en las matrices es pintarAvionDirDerecha y pintarAvionIzquierda el cual la única diferencia es la posición del avión para que el usuario pueda apreciar la parte delantera y trasera del avión y saber a qué dirección está moviéndose.

```
void pintarAvionDirDerecha()
{

    // DIBUJO AVION DERECHA
    //[yAvion]                          [xAvion-2]
    //[yAvion+1][yAvion+1][yAvion+1]    [xAvion-2][xAvio

    if (xAvion < 18)
    {                           // parte final del avion
        if ((xAvion - 2) >= 0) // asignar valores siemp
        {
            tableroJuego[yAvion][xAvion - 2] = 1;
            tableroJuego[yAvion + 1][xAvion - 2] = 1;
        }
    }

    if (xAvion < 17)
    {                           // parte medio del avion
        if ((xAvion - 1) >= 0) // asignar valores siemp
        {
            tableroJuego[yAvion + 1][xAvion - 1] = 1;
        }
    }

    if (xAvion < 16)
    { // trompa del avion
        tableroJuego[yAvion + 1][xAvion] = 1;
    }
}
```

```
void pintarAvionDirIzquierda()
{

    // DIBUJO AVION IZQUIERDA
    //                      [yAvion]
    //[yAvion+1][yAvion+1][yAvion+1]   [xAvion][xAvion-

    if (xAvion > -3)
    { // parte final del avion
        if ((xAvion + 2) <= 15)
        { // asignar valores siempre que esten en el r
            tableroJuego[yAvion][xAvion + 2] = 1;
            tableroJuego[yAvion + 1][xAvion + 2] = 1;
        }
    }

    if (xAvion > -2)
    { // parte media del avion
        if ((xAvion + 1) <= 15)
        { // asignar valores siempre que esten en el r
            tableroJuego[yAvion + 1][xAvion + 1] = 1;
        }
    }

    if (xAvion > -1)
    { // trompa del avion
        tableroJuego[yAvion + 1][xAvion] = 1;
    }
}
```

Cuando se realiza el cambio de la orientación del movimiento del avión es necesario borrar los leds actuales y encenderlos según la dirección seleccionada por el usuario.

```
void borrarAvionDirDerecha()
{

    if (xAvion < 18)
    {                           // parte final del avion
        if ((xAvion - 2) >= 0) // asignar valores siemp
        {
            tableroJuego[yAvion][xAvion - 2] = 0;
            tableroJuego[yAvion + 1][xAvion - 2] = 0;
        }
    }

    if (xAvion < 17)
    {                           // parte media del avion
        if ((xAvion - 1) >= 0) // asignar valores siemp
        {
            tableroJuego[yAvion + 1][xAvion - 1] = 0;
        }
    }

    if (xAvion < 16)
    { // trompa del avion
        tableroJuego[yAvion + 1][xAvion] = 0;
    }
}
```

```
void borrarAvionDirIzquierda()
{

    if (xAvion > -3)
    { // parte final del avion
        if ((xAvion + 2) <= 15)
        { // asignar valores siempre que esten en el ra
            tableroJuego[yAvion][xAvion + 2] = 0;
            tableroJuego[yAvion + 1][xAvion + 2] = 0;
        }
    }

    if (xAvion > -2)
    { // parte media del avion
        if ((xAvion + 1) <= 15)
        { // asignar valores siempre que esten en el ra
            tableroJuego[yAvion + 1][xAvion + 1] = 0;
        }
    }

    if (xAvion > -1)
    { // trompa del avion
        tableroJuego[yAvion + 1][xAvion] = 0;
    }
}
```

El avión al momento de disparar se obtiene la posición del avión actual y también su centro, a partir del centro del avión se realiza el disparo del avión y se va moviendo sobre el eje de la Y hasta tocar suelo.

```
void pintarBala()
{
    if (direccionAvion)
    { // avion va hacia derecha
        if ((xAvion - 1) >= 0 && xAvion <= 16)
        { // Dispara cuando la parte media esta dentro
            tableroJuego[yAvion + 2][xAvion - 1] = 1;
            tableroBalas[yAvion + 2][xAvion - 1] = 1; /
        }
    }
    else
    {
        if ((xAvion + 1) <= 15 && xAvion >= -1)
        { // Dispara cuando la parte media esta dentro
            tableroJuego[yAvion + 2][xAvion + 1] = 1;
            tableroBalas[yAvion + 2][xAvion + 1] = 1; /
        }
    }
}
```

```
void pintarNuevaPosicionBalas()
{

    // Recorremos matriz de balas, cuando encontramos una,
    for (int i = 7; i > (yAvion + 1); i--)
    {
        for (int j = 0; j < 16; j++)
        {
            if (tableroBalas[i][j] == 1)
            { // si hay bala

                if (i == 7)
                { // cuando se este en la ultima fila de la
                    if (tableroTorres[j] == 1)
                    { // si hay torre

                        // torre destruida
                        tableroTorres[j] = 0; // quitamos t
                        cantidadTorresPorDestruir--;

                        // punto
                        puntos++;
                        if (puntos % 5 == 0)
                        {
                            vidas++;
                            Serial.println("VIDA EXTRA");
                        }

                        if (cantidadTorresPorDestruir == 0)
                        {
                            // Serial.println("Siguiente ni
                            nivelJuego++;
                            mostrarNivel();
                            construirNuevasTorres = true;
                            yAvion = 0; // reseteamos altur
                            xAvion = 0;
                            tAvionDesplazY0 = millis(); //
                        }
                    }
                }
```

La función jugar se declaran variables de la posición del avión al momento de iniciar, genera las torres, realiza la acción de disparar, cambia dirección del movimiento, almacena el puntaje y la cantidad de vidas que tiene al momento de jugar.

```
void jugar()
{
    //  Serial.print(xAvion);
    if(nuevaPartida){
      xAvion = 0;
      yAvion = 0;
      nivelJuego = 1;
      mostrarNivel();
      tAvionDesplazY0 = millis();
      nuevaPartida = false;
      construirNuevasTorres = true;
      poderDisparar = true;
      cantidadTorresPorDestruir = 3;
      limpiarTableros();
    }

    // Cambia a direccion izquierda el
    if (digitalRead(DIR_LEFT) == LOW)
    {
        borrarAvionDirDerecha();
        direccionAvion = false;
    }

    // Cambia a direccion derecha el av
    if (digitalRead(DIR_RIGHT) == LOW)
    {
        borrarAvionDirIzquierda();
        direccionAvion = true;
    }

    if (direccionAvion)
    {
        jugarDirDerecha();
    }
    else
    {
        jugarDirIzquierda();
    }
```

Conforme pasa un determinado tiempo durante la ejecución del juego, el avión empezara a descender y en caso de que toque una torre perderá una de sus vidas.

```
void desplazamientoAvionY()
{
    tAvionDesplazY1 = millis();
    if ((tAvionDesplazY1 - tAvionDesplazY0)
    { // se desplaza cada 2 segundos
        tAvionDesplazY0 = millis();
        yAvion++;
    }

    if (yAvion == 7)
    { // llego a la parte baja del tablero
        avionLlegoAlFinal();
    }
}
```

Las torres que debe derribar el avión se generan de forma aleatoria entre 0 – 16 en el eje de las X y de 1- 5 en el eje de las Y.

```
void construirTorres()
{

    for (int i = 0; i < 16; i++)
    {
        tableroTorres[i] = 0; // reseteamos
    }

    cantidadTorresPorDestruir = nivelJuego + 2;

    if (cantidadTorresPorDestruir > 16)
    { // la cantidadmaxima de torres a generar es de 16 (0 al 15)
        cantidadTorresPorDestruir = 16;
    }

    for (int i = 0; i < cantidadTorresPorDestruir; i++)
    {
        int posicionTorreEnTablero = random(0, 16); // (min, max-1)
        int alturaTorre = random(1, 5);             // (min, max-1)

        if (tableroTorres[posicionTorreEnTablero] != 1)
        { // aun no hay torre en esa posicion (ya que random las genera en

            tableroTorres[posicionTorreEnTablero] = 1; // con 1 indicamos
                                                       //      Serial.prin

            for (int j = 0; j < 16; j++)
            { // recorreo matriz de forma horizontal
                if (j == posicionTorreEnTablero)
                {
                    for (int k = 0; k < alturaTorre; k++)
                    {
                        tableroJuego[7 - k][posicionTorreEnTablero] = 1;
                    }
                }
            }
        }
```

Cuando el avión al pasar el tiempo disminuirá su altura y en caso de que choque con una de las torres se valida si la posición del avión es igual a una de las torres generadas aleatoriamente, se genera un mensaje con el texto: choque parte final del avión. Disminuirá su vida pero aumentara un poco la altura del avión.

```
void verificarChoque()
{

    if (direccionAvion)
    {

        if (xAvion < 18)
        {                           // parte final del avion
            if ((xAvion - 2) >= 0) // asignar valores siempre que esten en el rango de la matriz
            {
                if (tableroJuego[yAvion][xAvion - 2] == 1 || tableroJuego[yAvion + 1][xAvion - 2] == 1)
                {
                    Serial.println("choque parte final del avion");
                }
            }
        }

        if (xAvion < 17)
        {                           // parte medio del avion
            if ((xAvion - 1) >= 0) // asignar valores siempre que esten en el rango de la matriz
            {
                if (tableroJuego[yAvion + 1][xAvion - 1] == 1)
                {
                    Serial.println("choque parte medio del avion");
                    avionChoco();
                }
            }
        }

        if (xAvion < 16)
        { // trompa del avion
            if (tableroJuego[yAvion + 1][xAvion] == 1)
            {
                Serial.println("choque trompa del avion");
                avionChoco();
            }
        }
    }
```

```
void avionChoco()
{
    vidas--;
    if (vidas == 0)
    {
        guardarPuntaje();
        resetCuandoNoHayVidas();
        estadoActual = MENSAJE;
    }
    else
    {
        yAvion = (yAvion - 2) < 0 ? 0 : (yAvion - 2);
    }
}
```

La función avionLlegoAlFinal es ejecutado desde el método desplazamientoAvionY, cuando llega hasta abajo del tablero.

```
void avionLlegoAlFinal()
{ // es ejecutado desde el metodo desplazamientoAvionY,
    vidas--;
    if (vidas == 0)
    {
        guardarPuntaje();
        resetCuandoNoHayVidas();
        estadoActual = MENSAJE;
    }
    else
    {
        yAvion = (yAvion - 3) < 0 ? 0 : (yAvion - 3);;
    }
}
```

Cuando el usuario llega a perder un nivel en la partida, se reinicia todas las variables en caso de que el usuario quiere volver a jugar.

```
void resetCuandoNoHayVidas()
{
    puntos = 0;
    vidas = 3;
    xAvion = 0;
    yAvion = 0;
    nivelJuego = 1;
}
```

Una vez que el usuario derribe todas las torres de un nivel, se ejecuta la función mostrarNivel y por dos segundos enseña el nivel siguiente al que gano.

```
void mostrarNivel()
{ // Mostrar nivel por 2 segundos

    unsigned long tiempo1 = millis();
    unsigned long tiempo2 = millis();

    while (true)
    {
        imprimirPuntuacion(nivelJuego); //
        tiempo1 = millis();
        if (tiempo1 >= (tiempo2 + 2000))
        {
            tiempo1 = 0;
            break;
        }
    }
}
```

Se guarda el puntaje por medio de un array que su contenido es de -1, se lee el array y si encuentra un -1 lo sustituye con el puntaje almacenado durante la partida.

```
void guardarPuntaje()
{
    // [posVieja,x,x,x,PosNueva]
    bool flatArrayLleno = true;
    for (int i = 0; i < 5; i++)
    { // recorremos array puntaje
        if (puntajes[i] == -1)
        { // no hay nada en esa posicion
            puntajes[i] = puntos;
            flatArrayLleno = false;
            break;
        }
    }

    if (flatArrayLleno)
    { // array lleno, hay que desplazar posi
        for (int i = 0; i < 4; i++)
        {
            puntajes[i] = puntajes[i + 1];
        }
        puntajes[4] = puntos;
    }

    Serial.println("PUNTAJES");
    for (int i = 0; i < 5; i++)
    {
        Serial.println(puntajes[i]);
    }
}
```

En la visualización de estadísticas reservar un espacio específico para almacenar los últimos cinco puntajes obtenidos en el juego. Con estos datos se generará un gráfico de barras que muestre el valor de los puntajes más recientes. Para generar los valores que se mostrarán se deberán sumar los punteos almacenados. Con ese total se calculará el porcentaje sobre ese total que representa cada uno de los punteos guardados.

```
void visualizarEstadisticas()
{
    // limpiando matriz
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 16; j++)
        {
            tableroEstadisticas[i][j] = 0;
        }
    }

    float promedio = 0;
    float valores[5] = {0, 0, 0, 0, 0};
    int alturaBarra[5] = {0, 0, 0, 0, 0};

    // calculado promedio
    for (int i = 0; i < 5; i++)
    {
        if (puntajes[i] > -1)
        {
            promedio = promedio + (float)puntajes[i];
        }
    }

    // valores para altura de cada barra de grafica
    for (int i = 0; i < 5; i++)
    {
        if (puntajes[i] > -1)
        {
            valores[i] = ((float)puntajes[i] / promedio) * 8;
        }
    }

    // redondeando valores
    for (int i = 0; i < 5; i++)
    {
        if (valores[i] > 0 && valores[i] <= 1)
        {
            alturaBarra[i] = 1;
```

```
        alturaBarra[i] = 3;
    }
    else if (valores[i] > 3 && valores[i] <= 4)
    {
        alturaBarra[i] = 4;
    }
    else if (valores[i] > 4 && valores[i] <= 5)
    {
        alturaBarra[i] = 5;
    }
    else if (valores[i] > 5 && valores[i] <= 6)
    {
        alturaBarra[i] = 6;
    }
    else if (valores[i] > 6 && valores[i] <= 7)
    {
        alturaBarra[i] = 7;
    }
    else if (valores[i] > 7 && valores[i] <= 8)
    {
        alturaBarra[i] = 8;
    }
    else
    {
        alturaBarra[i] = 0;
    }
```

```
void mostrarMatrizEstadistica()
{
    // Con driver
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            ledControl.setLed(0, i, j, tableroEstadisticas[i][j] == 1 ? true : false);
    // Sin driver
    for (int k = 0; k < 3; k++)
        for (int i = 0; i < 8; i++)
            for (int j = 8; j < 16; j++)
                if (tableroEstadisticas[i][j] == 1)
                    pintarLED(j - 8, i);
                else
                    delayMicroseconds(50);
}
```

Por medio de dos potenciómetros se puede modificar la cantidad de vidas al momento de iniciar una partida, la velocidad del mensaje inicial y del avión. Son representadas por dos barras horizontales y por medio de Los potenciómetros que están en un rango de 1 a 10 lo puede modificar dicho valor.

```
void pintarBarra(int posicionY, int longitudX)
{
    for (int i = 0; i < 10; i++)
    {
        if (i <= 4)
        {
            ledControl.setLed(0, posicionY, i + 3, i < longitudX ? true : false);
            ledControl.setLed(0, posicionY + 1, i + 3, i < longitudX ? true : false);
        }
        else
        {
            if (i < longitudX)
            {
                pintarLED(i - 5, posicionY);
                pintarLED(i - 5, posicionY + 1);
            }
            else
                delayMicroseconds(50);
        }
    }
}
```

```
void configuracion()
{
  // DIFICULTAD-VELOCIDAD
  // mapear los valores del potenciometro en un rango de nu
  potenciometroIzq = map(analogRead(A0), 0, 1024, 1, 11);

  // para no cambiar logica de pintado de barras
  if(potenciometroIzq >= 0 && potenciometroIzq <= 2){
    velocidad = 400;
  }
  else if(potenciometroIzq > 2 && potenciometroIzq <= 5){
    velocidad = 300;
  }
  else if(potenciometroIzq > 5 && potenciometroIzq <= 9){
    velocidad = 200;
  }
  else if(potenciometroIzq > 9){
    velocidad = 150;
  }

  // VIDAS
  //  mapear los valores del potenciometro en un rango de n
  potenciometroDer = map(analogRead(A1), 0, 1024, 3, 11);
  vidas = potenciometroDer;

  pintarBarra(1, potenciometroIzq);
  pintarBarra(5, potenciometroDer);
}
```

Al momento de pulsar el botón K, imprime en las matrices de los leds la cantidad de vidas de la partida y también tiene la opción de volver a jugar pulsando el mismo botón por 2 segundos o al menú principal por 3 segundos.

```
void MenuPausa()
{
    //mostrarMatriz2(PausaXd);
    imprimirPuntuacion(vidas);
    if (digitalRead(K) == LOW)
    {
        delay(1000);
        Serial.println("1s");
        if (digitalRead(K) == LOW)
        {
            delay(1000);
            Serial.println("2s");
            estadoActual = JUGAR;
            if (digitalRead(K) == LOW)
            {
                delay(1000);
                Serial.println("3s");
                if (digitalRead(K) == LOW)
                {
                    estadoActual = MENU_PRINCIPAL;
                    nuevaPartida = true; // como reg
                }
            }
        }
    }
}
```

En el menú principal, el usuario tiene las opciones de escoger si quiere ingresar a estadísticas, configuración o jugar, por medio del estado de los botones se distingue la opción del usuario.

```
void MenuPrincipal()
{
    int estadoBotonLeft = digitalRead(DIR_LEFT);
    int estadoBotonDisp = digitalRead(DISP);
    int estadoBotonRight = digitalRead(DIR_RIGHT);
    int estadoBotonK = digitalRead(K);
    switch (estadoActual)
    {
    case MENSAJE:
        if (direccionLetrero)
        {
            moverCartelDCH();
        }
        else
        {
            moverCartelIZQ();
        }

        if (estadoBotonDisp == LOW)
        {
            delay(200);
            if (direccionLetrero)
            {
                direccionLetrero = false;
            }
            else
            {
                direccionLetrero = true;
            }
        }
```

```
case MENU_PRINCIPAL:
    mostrarMatriz2(MenuP);
    Serial.println(".");
    if (estadoBotonLeft == LOW)
    {
        // Botón 1 presionado: ir a OPCION_1
        estadoActual = JUGAR;
        delay(100); // Pequeña pausa para ev
    }
    else if (estadoBotonDisp == LOW)
    {
        // Botón 2 presionado: ir a OPCION_2
        estadoActual = ESSTADISTICAS;
        delay(100);
    }
    else if (estadoBotonRight == LOW)
    {
        // Botón 3 presionado: ir a OPCION_3
        estadoActual = CONFIG;
        delay(100);
    }
    break;
case JUGAR:
    Serial.println("JUGAR");
    jugar();
    if (estadoBotonK == LOW)
    {
        // Botón 3 presionado: volver al MEN
        estadoActual = MENU_PAUSA;
        delay(100);
        while (estadoActual == MENU_PAUSA)
        {
            MenuPausa();
        }

        delay(100);
    }

    break;
```