Universidad de San Carlos de Guatemala

Facultad de Ingeniería, Arquitectura de computadoras y ensambladores 1

Sección N, aux. Ronald Marín

Práctica 1

Carlos Eduardo Soto Marroquín 201902502

Carlos Javier Martinez Polanco 201709282

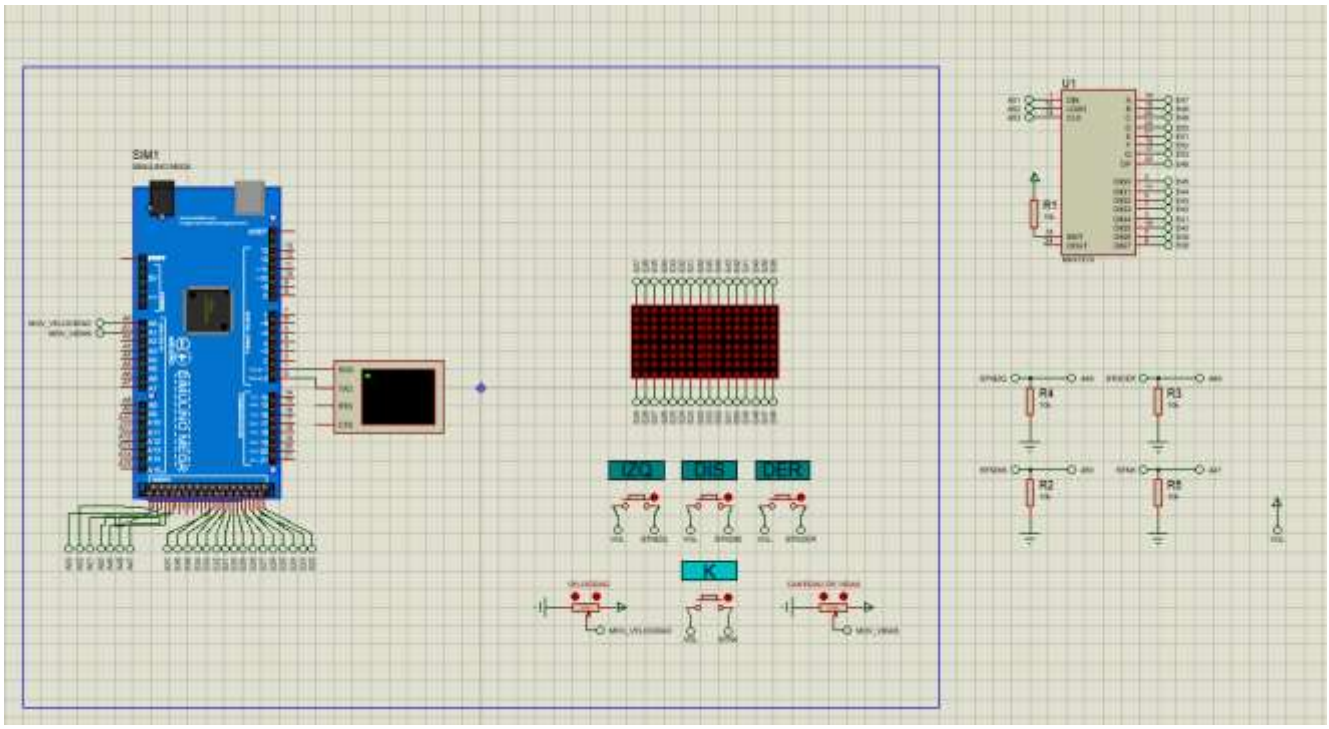
Jefferson Gamaliel Molina Barrios 201945242

Gladys Leticia Ajuchan Vicente 201807389

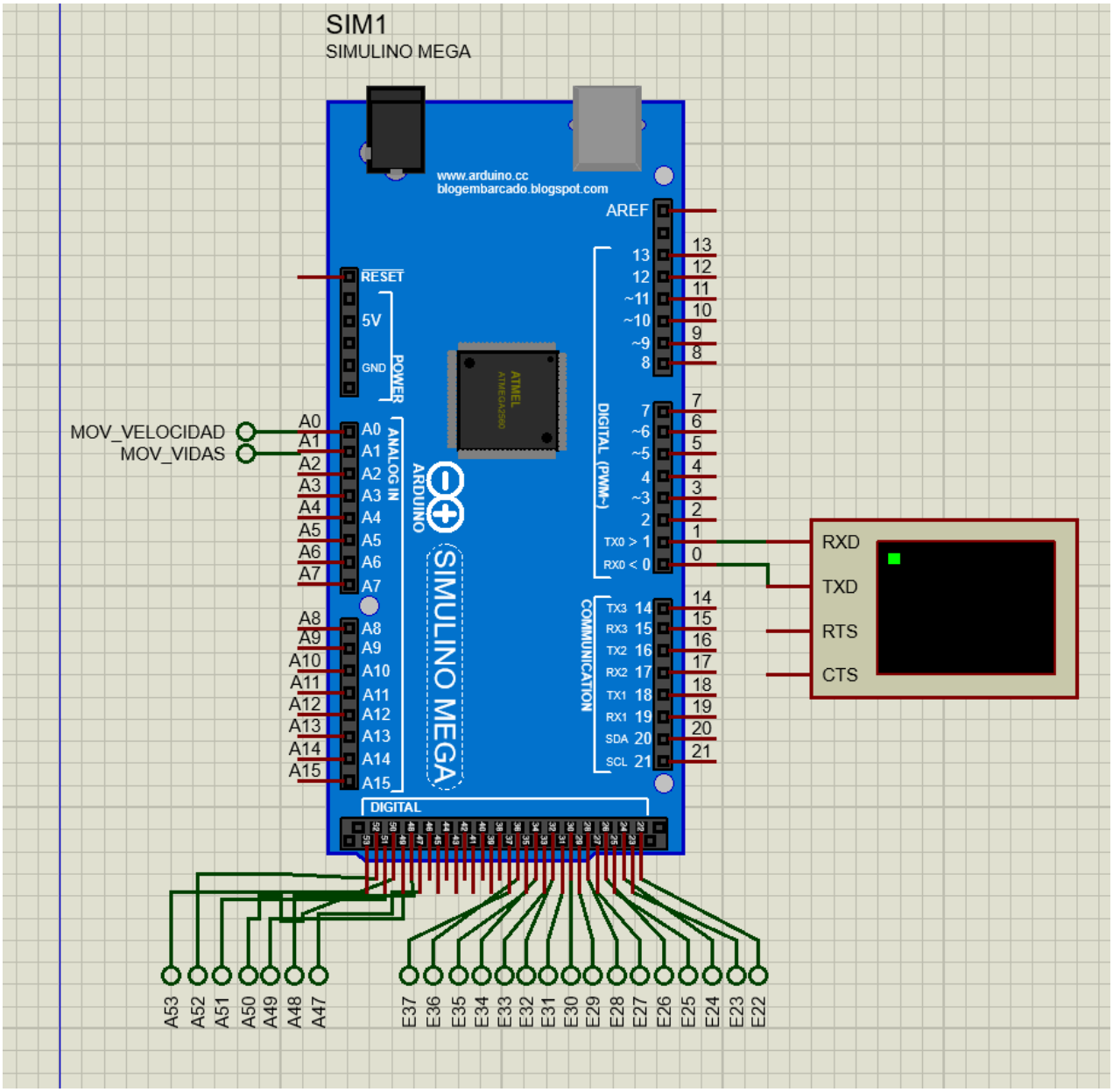
Douglas Alexander Soch Catalán 201807032

Manual Técnico

Visualización general del circuito:



Visualización del Arduino y pines utilizados:



Listado de pines utilizados:

- Digitales (PWM):

Estos pines van conectados a la terminar serial, permiten el este tipo de comunicación con el Arduino.

  - 0: rx
  - 1: tx

- Analógicos:

  Estos pines son conectados a los potenciómetros para controlarlos por medio del Arduino.

  - A0: MOV_VELOCIDAD
  - A1: MOV_VIDAS

- Digitales (I/O):

  Estos pines son conectados a los dispositivos de entradas para el Arduino y salidas del Arduino a otros dispositivos.

  - 47 – 53
  - 22 – 37

Componentes del sistema:

- Arduino UNO (simulino UNO en proteus)
- 2 potenciómetros
- 4 Push buttons.
- 5 resistencias de 10k
- 2 matriz de led de 8x8
- 1 controlador para matriz de led MAX7219

Librerias para Arduino:

- LedControl

Librerias para Proteus:

- Simulino

Estructuras:

// Texto de inicio

```
const short LONGITUD_TEXTO = 24;
bool cadena[8][LONGITUD_TEXTO] = {
  { 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0 },
  { 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 },
  { 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 },
  { 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 },
  { 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0 },
  { 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 },
  { 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 },
  { 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0 },
};

bool tablero[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 }
};

bool menu_principal[8][16] = {
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 },
```

```
  { 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0 },
  { 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0 },
  { 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 }
};
bool numeros[10][8][4] = {
  {
    { 0, 0, 0, 0 },
    { 1, 1, 1, 1 },
    { 1, 0, 0, 1 },
    { 1, 0, 0, 1 },
    { 1, 0, 0, 1 },
    { 1, 0, 0, 1 },
    { 1, 0, 0, 1 },
    { 1, 1, 1, 1 }
  },
```

```
  {
    { 0, 0, 0, 0 },
    { 0, 1, 1, 0 },
```

```
  { 0, 1, 1, 0 },
  { 1, 1, 1, 0 },
  { 0, 1, 1, 0 },
  { 0, 1, 1, 0 },
  { 0, 1, 1, 0 },
  { 1, 1, 1, 1 }

},

{

  { 0, 0, 0, 0 },
  { 1, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 1, 1, 1, 1 },
  { 1, 0, 0, 0 },
  { 1, 0, 0, 0 },
  { 1, 1, 1, 1 }

},

{
  { 0, 0, 0, 0 },
  { 1, 1, 1, 1 },
```

```
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 1, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 1, 1, 1, 1 }
},
{

  { 0, 0, 0, 0 },
  { 1, 0, 0, 1 },
  { 1, 0, 0, 1 },
  { 1, 0, 0, 1 },
  { 1, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 }

},

{

  { 0, 0, 0, 0 },
```

```
  { 1, 1, 1, 1 },
  { 1, 0, 0, 0 },
  { 1, 0, 0, 0 },
  { 1, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 1, 1, 1, 1 }

},

{

  { 0, 0, 0, 0 },
  { 1, 1, 1, 1 },
  { 1, 0, 0, 0 },
  { 1, 0, 0, 0 },
  { 1, 1, 1, 1 },
  { 1, 0, 0, 1 },
  { 1, 0, 0, 1 },
  { 1, 1, 1, 1 }

},

{

  { 0, 0, 0, 0 },
```

```
  { 1, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 0, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 }

},

{

  { 0, 0, 0, 0 },
  { 1, 1, 1, 1 },
  { 1, 0, 0, 1 },
  { 1, 0, 0, 1 },
  { 1, 1, 1, 1 },
  { 1, 0, 0, 1 },
  { 1, 0, 0, 1 },
  { 1, 1, 1, 1 }

},

{

  { 0, 0, 0, 0 },
```

```
  { 1, 1, 1, 1 },
  { 1, 0, 0, 1 },
  { 1, 0, 0, 1 },
  { 1, 1, 1, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 },
  { 0, 0, 0, 1 }

 }

};


bool caracter_nivel[8][8] = {
 { 0, 0, 0, 0, 0, 0, 0, 0 },
 { 0, 1, 0, 0, 0, 0, 1, 0 },
 { 0, 1, 1, 0, 0, 0, 1, 0 },
 { 0, 1, 0, 1, 0, 0, 1, 0 },
 { 0, 1, 0, 0, 1, 0, 1, 0 },
 { 0, 1, 0, 0, 0, 1, 1, 0 },
 { 0, 1, 0, 0, 0, 0, 1, 0 },
 { 0, 0, 0, 0, 0, 0, 0, 0 }
};



struct Bomba {
 short x = -1;
 short y = -1;
```

```
};

Bomba bombas[16];

bool caracter_vidas[8][8]={
 { 0, 1, 1, 0, 0, 1, 1, 0 },
 { 1, 1, 1, 1, 1, 1, 1, 1 },
 { 1, 1, 1, 1, 1, 1, 1, 1 },
 { 1, 1, 1, 1, 1, 1, 1, 1 },
 { 1, 1, 1, 1, 1, 1, 1, 1 },
 { 0, 1, 1, 1, 1, 1, 1, 0 },
 { 0, 0, 1, 1, 1, 1, 0, 0 },
 { 0, 0, 0, 1, 1, 0, 0, 0 }
};

struct Digitos {
 int digito1;
 int digito2;
};
```

Manejo de pines matriz sin controlador:

```
#include <Arduino.h>
```

```
#include "matriz_sin_driver.h"

// Pines que se utilizan en el arduino para manejar la matriz sin controlador
short columnas[] = { 37, 36, 35, 34, 33, 32, 31, 30 };
short filas[] = { 29, 28, 27, 26, 25, 24, 23, 22 };

void inicializarMatrizSinDriver() {
  //Se definen las salidas
  for (int i = 0; i < 8; i++) {
    pinMode(filas[i], OUTPUT);
    pinMode(columnas[i], OUTPUT);
  }
}

void seleccionarFila(int fila) {
  if (fila == 0) digitalWrite(filas[0], LOW);
  else digitalWrite(filas[0], HIGH);
```

```
  if (fila == 1) digitalWrite(filas[1], LOW);
  else digitalWrite(filas[1], HIGH);
  if (fila == 2) digitalWrite(filas[2], LOW);
  else digitalWrite(filas[2], HIGH);
  if (fila == 3) digitalWrite(filas[3], LOW);
  else digitalWrite(filas[3], HIGH);
  if (fila == 4) digitalWrite(filas[4], LOW);
  else digitalWrite(filas[4], HIGH);
  if (fila == 5) digitalWrite(filas[5], LOW);
  else digitalWrite(filas[5], HIGH);
  if (fila == 6) digitalWrite(filas[6], LOW);
  else digitalWrite(filas[6], HIGH);
  if (fila == 7) digitalWrite(filas[7], LOW);
  else digitalWrite(filas[7], HIGH);
}

void setearEstadoEnColumna(int columna, bool estado) {
  if (columna == 0) digitalWrite(columnas[0], estado);
  if (columna == 1) digitalWrite(columnas[1], estado);
  if (columna == 2) digitalWrite(columnas[2], estado);
  if (columna == 3) digitalWrite(columnas[3], estado);
  if (columna == 4) digitalWrite(columnas[4], estado);
  if (columna == 5) digitalWrite(columnas[5], estado);
  if (columna == 6) digitalWrite(columnas[6], estado);
  if (columna == 7) digitalWrite(columnas[7], estado);
}
```

Codigo principal del sistema:

```
#include <LedControl.h>
#include "estructuras.h"
```

```
#include "matriz_sin_driver.h"


// Mensaje
int offset = 0;                 // Indica la posicion actual del movimiento del mensaje
bool msg_hacia_izquierda = false;  // Indica la direccion de recorrido que dara el mensaje inicial
int potenciometro = 0;          // Indica la velocidad del movimiento del mensaje


// Pines generales
short pinPotenciometroVelocidad = A0;               // Pin para manejar el potenciometro que representa la velocidad del mensaje y del avion
short pinPotenciometroVidas = A1;                   // Pin para manejar el potenciometro que representa la cantidad de vidas que se tendra en una partida nueva
short pinDisparo = 50;                              // Pin para manejar el boton de disparo
short pinDerecha = 49;                              // Pin para manejar el boton con movimiento hacia la derecha
short pinIzquierda = 48;                            // Pin para manejar el boton con movimiento hacia la izquierda
short pinIni = 47;                                  // Pin para manejar el boton de inicio
LedControl matriz_driver = LedControl(51, 53, 52, 1);  // Matriz con driver


// Juego
short vidas = 3;                    // Indica la cantidad de vidas que tiene en el juego actual
short nivel = 1;                    // Indica el nivel actual de juego
short edificios_a_destruir = 0;         // Indica la cantidad de edificios que hay que destruir en el nivel actual de juego
short edificios_destruidos = 0;         // Indica la cantidad de edificios que el usuario a destruido desde que ha iniciado el juego
short pos_x_avion = 0;                  // Indica la posicion del avion en el eje x de la matriz
short pos_y_avion = -3;                 // Indica la posicion del avion en el eje y de la matriz
bool avion_hacia_izquierda = false;     // Indica la direccion de movimiento del avion
```

```
String estado_app = "MENSAJE";              // Indica el estado actual en el que esta el programa; Estos pueden ser: MENSAJE, MENU, JUGAR, PAUSA, ESTADISTICA, CONFIGURACION
short estadisticas[5] = { 0, 0, 0, 0, 0 };  // Sirve para almacenar los 5 mejores puntajes

void setup() {
  inicializarMatrizSinDriver();  // Inicializando la matriz sin driver
  Serial.begin(9600);           // Inicializando la comunicacion serial

  // Inicializando la matriz con driver
  matriz_driver.shutdown(0, false);
  matriz_driver.setIntensity(0, 8);
  matriz_driver.clearDisplay(0);
}

long t0 = millis();
long t1 = millis();
void loop() {

  if (estado_app == "MENSAJE") {

    // Definiendo la velocidad de recorrido del mensaje
    t1 = millis();
    if (t1 - t0 >= 200) {
      offset++;
      t0 = millis();
    }

    // Imprimiendo el mensaje en ambas matrices
    imprimirMensajeMatrizSinDriver();
```

```
  imprimirMensajeMatrizConDriver();

 } else if (estado_app == "MENU") {
  imprimirMenuPrincipal();
 } else if (estado_app == "JUGAR") {

  // Muestra el nivel y redibuja los edificios en el tablero en el caso que no halla mas edificios por destruir
  if (edificios_a_destruir == 0) {

    // Obtiene el tiempo actual en milisegundos
    unsigned long tiempoInicio = millis();

    // Muestra el mensaje del nivel durante 2 segundos
    while (millis() - tiempoInicio < 2000) {
     imprimirMensajeNivel();
    }

    // Dibuja los edificios la matriz 'tablero'
    dibujarEdificios();
  }

  // Definiendo la velocidad de recorrido del avion
  t1 = millis();
  if (t1 - t0 >= 200) {
    offset++;
    t0 = millis();

    movimientoBomba();
```

```
    movimientoAvion();
   }

   // Dibujando el avion
   dibujarAvion();
   dibujarBomba();
   imprimirTablero();
  }else if (estado_app == "PAUSA"){
   imprimirMenuPausa(vidas);
  }

  // potenciometro = map(analogRead(A0), 0, 1024, 200, 800);

  // Escuchando los botones
  botonK();
  botonDerecho();
  botonIzquierdo();
  botonDisparo();
}

/**************************/
/********** JUEGO **********/
/**************************/

// Imprime todos los estados de la variable 'tablero'
// en la matriz sin driver y en la matriz con driver
void imprimirTablero() {

  // Matriz sin driver
```

```
  for (int i = 0; i < 9; i++) {
    seleccionarFila(i);
    for (int j = 0; j < 8; j++) {
      setearEstadoEnColumna(j, tablero[i][j]);
    }
    delay(1);
  }

  // Matriz con driver
  for (int i = 0; i < 8; i++) {
    for (int j = 8; j < 16; j++) {
      matriz_driver.setLed(0, i, 15 - j, tablero[i][j]);
    }
  }
}

// Dibuja el avion en el tablero
void dibujarAvion() {
  if (avion_hacia_izquierda) {

    // Imprime la parte alta del avion
    if (pos_y_avion >= 0 && pos_y_avion < 16) {
      tablero[pos_x_avion][pos_y_avion] = 1;
    }

    // Imprime la parte baja del avion
    for (int i = 0; i < 3; i++) {
      if (pos_y_avion - i >= 0 && pos_y_avion - i < 16) {
        tablero[pos_x_avion + 1][pos_y_avion - i] = 1;
```

```
    }
   }

   // Si el avion colisiona contra un edificio se sube 2 unidades hacia arriba
   if (pos_y_avion - 3 >= 0 && pos_y_avion - 3 < 16 && tablero[pos_x_avion + 1][pos_y_avion - 3]
== 1) {
    vidas--;
    if (vidas > 0) {
     movimientoAvion();
     pos_x_avion = pos_x_avion - 3;
    } else {
     reiniciarJuego();
     estado_app = "MENSAJE";
    }
   }

   // Si el avion llega a la parte final del tablero se baja un nivel y se reaparece del lado contrario
   if (pos_y_avion < 0) {
    pos_y_avion = 18;
    pos_x_avion++;
   }
  } else {

   // Imprime la parte alta del avion
   if (pos_y_avion >= 0 && pos_y_avion < 16) {
    tablero[pos_x_avion][pos_y_avion] = 1;
   }

   // Imprime la parte baja del avion
```

```
    for (int i = 0; i < 3; i++) {
      if (pos_y_avion + i >= 0 && pos_y_avion + i < 16) {
        tablero[pos_x_avion + 1][pos_y_avion + i] = 1;
      }
    }

    // Si el avion colisiona contra un edificio se sube 2 unidades hacia arriba
    if (pos_y_avion + 3 >= 0 && pos_y_avion + 3 < 16 && tablero[pos_x_avion + 1][pos_y_avion + 3]
== 1) {
      vidas--;
      if (vidas > 0) {
        movimientoAvion();
        pos_x_avion = pos_x_avion - 3;
      } else {
        reiniciarJuego();
        estado_app = "MENSAJE";
      }
    }

    // Si el avion llega a la parte final del tablero se baja un nivel y se reaparece del lado contrario
    if (pos_y_avion > 16) {
      pos_y_avion = -3;
      pos_x_avion++;
    }
  }
}

// Desplaza el avion en el tablero
void movimientoAvion() {
```

```
  if (avion_hacia_izquierda) {
    if (pos_y_avion >= 0 && pos_y_avion < 16) {
      tablero[pos_x_avion][pos_y_avion] = 0;
    }
    for (int i = 0; i < 3; i++) {
      if (pos_y_avion - i >= 0 && pos_y_avion - i < 16) {
        tablero[pos_x_avion + 1][pos_y_avion - i] = 0;
      }
    }
    pos_y_avion--;
  } else {
    if (pos_y_avion >= 0 && pos_y_avion < 16) {
      tablero[pos_x_avion][pos_y_avion] = 0;
    }
    for (int i = 0; i < 3; i++) {
      if (pos_y_avion + i >= 0) {
        tablero[pos_x_avion + 1][pos_y_avion + i] = 0;
      }
    }
    pos_y_avion++;
  }
}

// Dibuja la bomba en el tablero
void dibujarBomba() {
  for (int i = 0; i < 16; i++) {
    if (bombas[i].x != -1 && bombas[i].y != -1) {
      tablero[bombas[i].x + 1][bombas[i].y] = 1;
    }
```

```
  }
}

// Desplaza la bomba en el tablero
void movimientoBomba() {
  for (int i = 0; i < 16; i++) {
    if (bombas[i].x != -1 && bombas[i].y != -1) {

      // Colosiona contra un edificio
      if (tablero[bombas[i].x + 2][bombas[i].y] == 1) {

        // Se acumula el edificio destruido
        edificios_destruidos++;

        // Se agrega la cantidad de edificios que se ha destruido durante el juego en las estadisticas
        for (int i = 0; i < sizeof(estadisticas) / sizeof(estadisticas[0]); i++) {
          if (edificios_destruidos > estadisticas[i]) {
            estadisticas[i] = edificios_destruidos;
            break;
          }
        }

        // En el caso que halla destruido 5 edificios se le agregara una vida extra
        if (edificios_destruidos % 5 == 0) {
          vidas++;
        }

        // Se borra el edificio
        for (int j = bombas[i].x + 1; j < 8; j++) {
```

```
      tablero[j][bombas[i].y] = 0;
    }

    // Se borra la bala
    tablero[bombas[i].x + 1][bombas[i].y] = 0;
    bombas[i].x = -1;
    bombas[i].y = -1;

    // Se disminuye la cantidad de edificios a destruir
    edificios_a_destruir--;

    // Se pasa al siguiente nivel una vez halla destruido todos los edificios y en el caso que sobrepase el nivel 10,
    // se va a la pantalla principal finalizando asi el juego
    if (edificios_a_destruir == 0) {
      nivel++;
      if (nivel > 10) {
        reiniciarJuego();
        estado_app = "MENSAJE";
      } else {
        movimientoAvion();
        pos_y_avion = avion_hacia_izquierda ? 18 : -3;
        pos_x_avion = 0;
      }
    }

  }
  // Colisiona con la parte baja de la matriz
  else if (bombas[i].x >= 6) {
```

```
      tablero[bombas[i].x + 1][bombas[i].y] = 0;
      bombas[i].x = -1;
      bombas[i].y = -1;
    }
    // Movimiento de la bomba
    else {
      tablero[bombas[i].x + 1][bombas[i].y] = 0;
      bombas[i].x = bombas[i].x + 1;
    }
   }
  }
}

// Dibuja los edificios que hay segun el nivel actual del juego
void dibujarEdificios() {
  short indice = 0;
  short largo = 0;
  short cantidad_edificios = nivel + 2;
  edificios_a_destruir = cantidad_edificios;
  while (cantidad_edificios > 0) {
    indice = random(16);
    largo = random(1, 5);
    if (!tablero[7][indice]) {
      for (int i = 0; i < largo; i++) {
        tablero[7 - i][indice] = 1;
      }
      cantidad_edificios--;
    }
  }
```

```
}

/**************************/
/********* BOTONES *********/
/**************************/

// Reconoce el boton ini presionado
unsigned long tiempo_boton_presionado;
void botonK() {
  int btnK = digitalRead(pinIni);

  if (btnK == HIGH) {
    if (tiempo_boton_presionado == 0) {
      tiempo_boton_presionado = millis();
    }
  } else {
    if (tiempo_boton_presionado != 0) {
      unsigned long lapso_tiempo = millis() - tiempo_boton_presionado;
      tiempo_boton_presionado = 0;

      // Entra al juego
      if (estado_app == "MENSAJE" && lapso_tiempo >= 2000) {
        estado_app = "MENU";
      }
      // Entra a la pausa del juego
      else if (estado_app == "JUGAR" && lapso_tiempo >= 2000) {
        estado_app = "PAUSA";
      }
      // Entra a la pausa del juego
```

```
    else if (estado_app == "PAUSA" && lapso_tiempo >= 2000) {
      estado_app = "JUGAR";
    }
    // Regresar al menu principal
    else if (estado_app == "PAUSA" && (lapso_tiempo >= 2000 && lapso_tiempo < 3000)) {
      estado_app = "MENU";
    }
   }
  }
}

// Reconoce el boton de disparo presionado
bool estado_boton_dis = false;
bool ultimo_estado_boton_dis = false;
unsigned long ultimo_tiempo_rebote_boton_dis = 0;
const unsigned long delay_rebote_boton_dis = 50;
void botonDisparo() {
  int btnDisparo = digitalRead(pinDisparo);

  if (btnDisparo != ultimo_estado_boton_dis) {
    ultimo_tiempo_rebote_boton_dis = millis();
  }

  if ((millis() - ultimo_tiempo_rebote_boton_dis) > delay_rebote_boton_dis) {
    if (btnDisparo != estado_boton_dis) {
      estado_boton_dis = btnDisparo;

      if (estado_boton_dis == LOW) {
        if (estado_app == "MENU") {
```

```
        estado_app = "ESTADISTICA";
      } else if (estado_app == "JUGAR") {
        for (int i = 0; i < sizeof(bombas) / sizeof(bombas[0]); i++) {
          if (bombas[i].x == -1 && bombas[i].y == -1) {
            if (avion_hacia_izquierda) {
              bombas[i].y = pos_y_avion - 1;
            } else {
              bombas[i].y = pos_y_avion + 1;
            }
            bombas[i].x = pos_x_avion + 1;
            break;
          }
        }
      }
    }
  }

  ultimo_estado_boton_dis = btnDisparo;
}

// Reconoce el boton derecho presionado
bool estado_boton_der = false;
bool ultimo_estado_boton_der = false;
unsigned long ultimo_tiempo_rebote_boton_der = 0;
const unsigned long delay_rebote_boton_der = 50;
void botonDerecho() {
  int btnDerecha = digitalRead(pinDerecha);
```

```
 if (btnDerecha != ultimo_estado_boton_der) {
  ultimo_tiempo_rebote_boton_der = millis();
 }

 if ((millis() - ultimo_tiempo_rebote_boton_der) > delay_rebote_boton_der) {
  if (btnDerecha != estado_boton_der) {
   estado_boton_der = btnDerecha;

   if (estado_boton_der == LOW) {
    if (estado_app == "MENSAJE") {
     msg_hacia_izquierda = false;
    } else if (estado_app == "MENU") {
     estado_app = "CONFIGURACION";
    } else if (estado_app == "JUGAR") {
     movimientoAvion();
     avion_hacia_izquierda = false;
    }
   }
  }
 }

 ultimo_estado_boton_der = btnDerecha;
}

// Reconoce el boton izquierdo presionado
bool estado_boton_izq = false;
bool ultimo_estado_boton_izq = false;
unsigned long ultimo_tiempo_rebote_boton_izq = 0;
const unsigned long delay_rebote_boton_izq = 50;
```

```
void botonIzquierdo() {
  int btnIzquierda = digitalRead(pinIzquierda);

  if (btnIzquierda != ultimo_estado_boton_izq) {
    ultimo_tiempo_rebote_boton_izq = millis();
  }

  if ((millis() - ultimo_tiempo_rebote_boton_izq) > delay_rebote_boton_izq) {
    if (btnIzquierda != estado_boton_izq) {
      estado_boton_izq = btnIzquierda;

      if (estado_boton_izq == LOW) {
        if (estado_app == "MENSAJE") {
          msg_hacia_izquierda = true;
        } else if (estado_app == "MENU") {
          estado_app = "JUGAR";
        } else if (estado_app == "JUGAR") {
          movimientoAvion();
          avion_hacia_izquierda = true;
        }
      }
    }
  }

  ultimo_estado_boton_izq = btnIzquierda;
}

/**************************/
/********** NIVEL ***********/
```

```
/**************************/

void imprimirMensajeNivel() {

 // Matriz sin driver
 for (int i = 0; i < 9; i++) {
  seleccionarFila(i);
  for (int j = 0; j < 8; j++) {
   setearEstadoEnColumna(j, caracter_nivel[i][j]);
  }
  delay(1);
 }



 // Matriz con driver - Primer digito
 int primer_digito = nivel / 10;
 for (int fila = 0; fila < 8; fila++) {
  for (int columna = 0; columna < 4; columna++) {
   matriz_driver.setLed(0, fila, 7 - columna, numeros[primer_digito][fila][columna]);
  }
 }

 // Matriz con driver - Segundo digito
 int segundo_digito = nivel - primer_digito * 10;
 for (int fila = 0; fila < 8; fila++) {
  for (int columna = 0; columna < 4; columna++) {
   matriz_driver.setLed(0, fila, 3 - columna, numeros[segundo_digito][fila][columna]);
  }
 }
```

```
}

void reiniciarJuego() {
  nivel = 1;
  edificios_a_destruir = 0;
  pos_x_avion = 0;
  pos_y_avion = -3;
  edificios_destruidos = 0;
}

/**************************/
/********** MENU ***********/
/**************************/

void imprimirMenuPrincipal() {

  // Matriz sin driver
  for (int i = 0; i < 9; i++) {
    seleccionarFila(i);
    for (int j = 0; j < 8; j++) {
      setearEstadoEnColumna(j, menu_principal[i][j]);
    }
    delay(1);
  }

  // Matriz con driver
  for (int i = 0; i < 8; i++) {
    for (int j = 8; j < 16; j++) {
      matriz_driver.setLed(0, i, 15 - j, menu_principal[i][j]);
```

```
   }
  }
}


/******IMPRIMIR MENU PAUSA******/
void imprimirMenuPausa(int vidas){
  Digitos digitos = obtenerDigitos(vidas);
  int digito1 = digitos.digito1;
  int digito2 = digitos.digito2;
  Serial.println(digito1);
  Serial.println(digito2);
  for (int i = 0; i < 9; i++) {
    seleccionarFila(i);
    for (int j = 0; j < 8; j++) {
      setearEstadoEnColumna(j, caracter_vidas[i][j]);
    }
    delay(1);
  }

  for (int fila = 0; fila < 8; fila++) {
    for (int columna = 0; columna < 4; columna++) {
      matriz_driver.setLed(0, fila, 7 - columna, numeros[digito1][fila][columna]);
    }
  }

  // Matriz con driver - Segundo digito
  for (int fila = 0; fila < 8; fila++) {
    for (int columna = 0; columna < 4; columna++) {
```

```
    matriz_driver.setLed(0, fila, 3 - columna, numeros[digito2][fila][columna]);
   }
  }

}


Digitos obtenerDigitos(int numero) {
  Digitos digitos;
  digitos.digito1 = numero / 10;
  digitos.digito2 = numero % 10;
  return digitos;
}

/**************************/
/********* CADENA **********/
/**************************/

void imprimirMensajeMatrizSinDriver() {
  for (int i = 0; i < 9; i++) {
    seleccionarFila(i);
    for (int j = 0; j < 8; j++) {
      offset = offset >= LONGITUD_TEXTO ? 0 : offset;
      if (msg_hacia_izquierda) {
        setearEstadoEnColumna(j, cadena[i][(j + offset) % LONGITUD_TEXTO]);
      } else {
        setearEstadoEnColumna(j, cadena[i][(j - offset < 0 ? LONGITUD_TEXTO + j - offset : j - offset)]);
      }
    }
```

```
   delay(1);
  }
}

void imprimirMensajeMatrizConDriver() {
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      if (msg_hacia_izquierda) {
        short offset_aux = 8 + offset + j;
        matriz_driver.setLed(0, i, 7 - j, cadena[i][offset_aux >= LONGITUD_TEXTO ? offset_aux - LONGITUD_TEXTO : offset_aux]);
      } else {
        short offset_aux = 8 - offset + j;
        matriz_driver.setLed(0, i, 7 - j, cadena[i][offset_aux < 0 ? LONGITUD_TEXTO + offset_aux : offset_aux]);
      }
    }
  }
}
```

MANUAL DE USUARIO

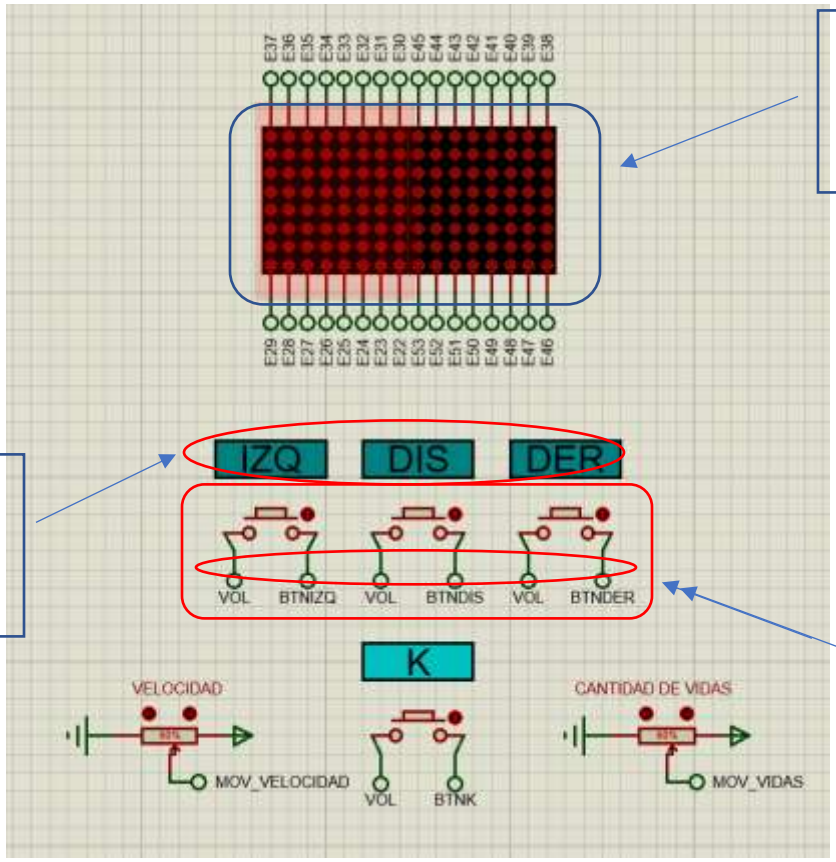
Manual de Usuario

Descripción del Juego:

El juego a desarrollar consiste en un pequeño avión cuya meta es destruir una serie de objetivos que se presentarán. se mostrará el arreglo inicial del juego. El avión destruirá los objetivos lanzando un proyectil que descenderá poco a poco hasta destruir un objetivo o chocar con el suelo en el caso de que se falle el tiro.

Panel de control del juego:

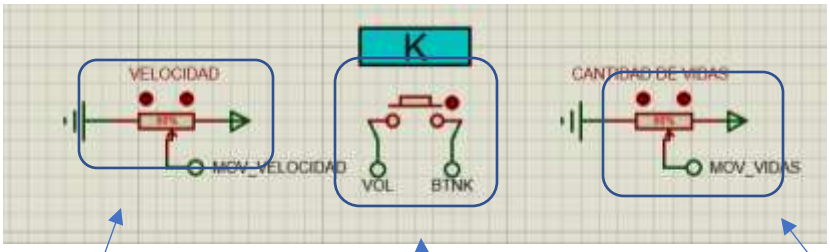
A continuación se les mostrara las partes principales del juego:



Pantalla Principal del Juego

Label del Juego

Botones principales del juego,
botón de la izquierda, centro y derecha del juego



Boton para la velocidad del juegos

Para que el usuario tenga la opción de salir de esta sección hacia el menú principal se deberá presionar el botón "K" por tres segundos.

Cantidad de vidas

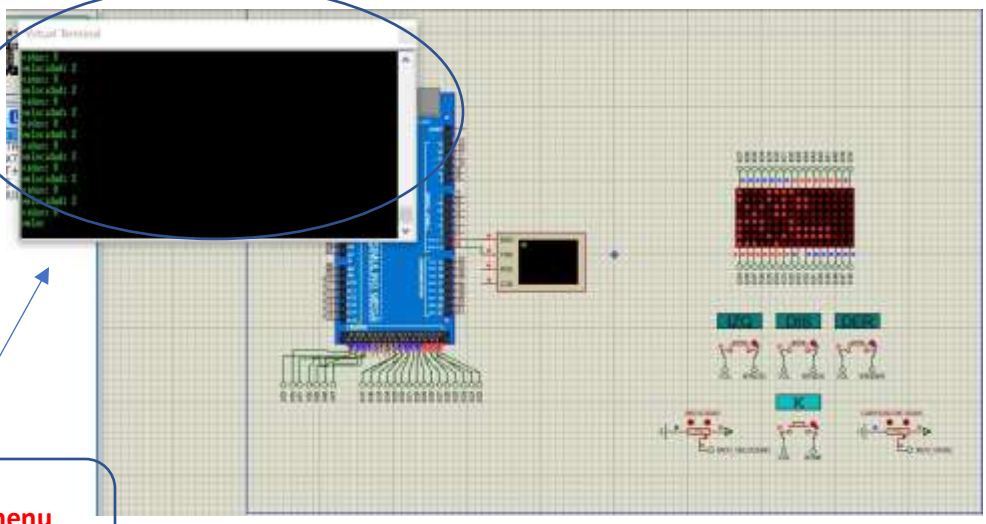
Menú Principal

Se mostrará un menú principal desde donde se podrá acceder a tres secciones:

- Juego,
- Estadísticas
- Configuración.

Menú de Configuración

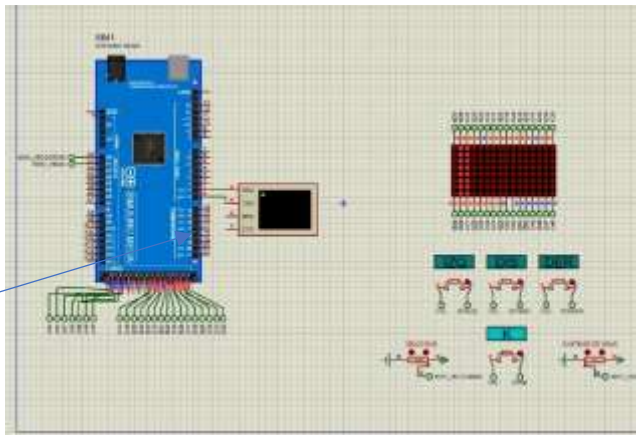
Será la parte donde se permitirá modificar parámetros esenciales del funcionamiento del sistema y del juego principalmente.



Visualization del menu de configuracion

Estadísticas

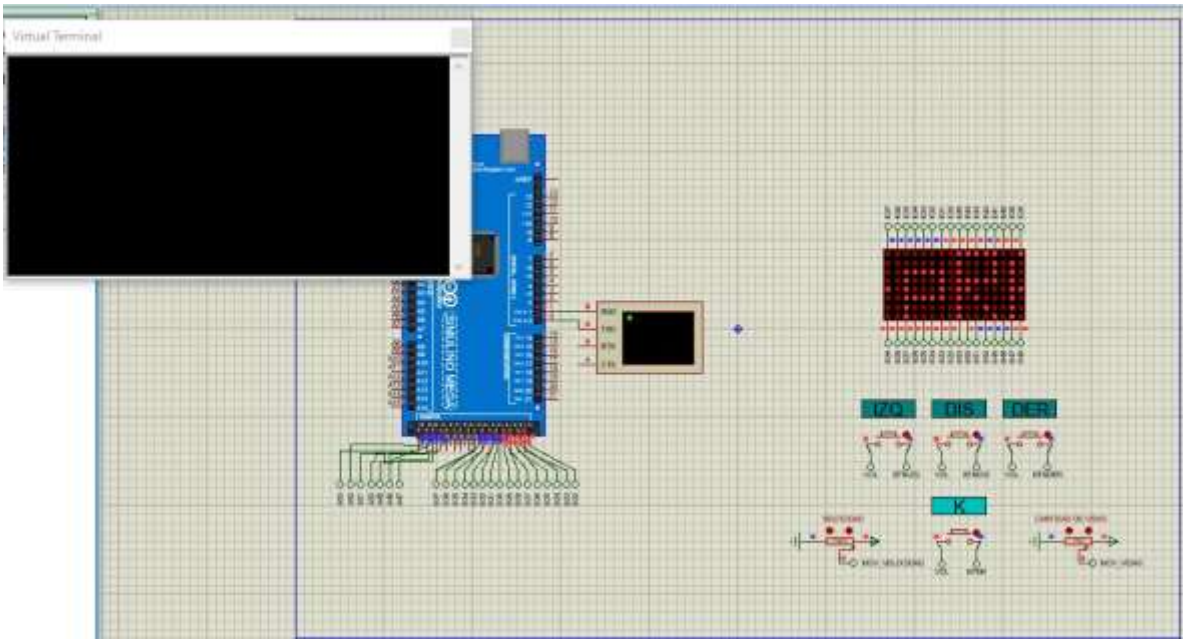
El usuario podrá tendrá un espacio específico para almacenar los últimos cinco puntajes obtenidos en el juego. Con estos datos se generará un gráfico de barras que muestre el valor de los puntajes más recientes.



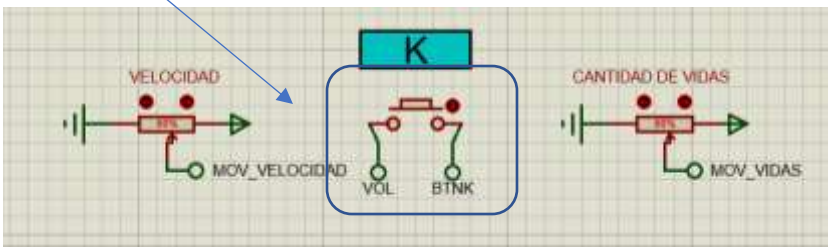
visualización de los puntuajes obtenidos por el usuario

Modalidad del Juego

- Al momento de ingresar, se le mostrara un mensaje inicial:

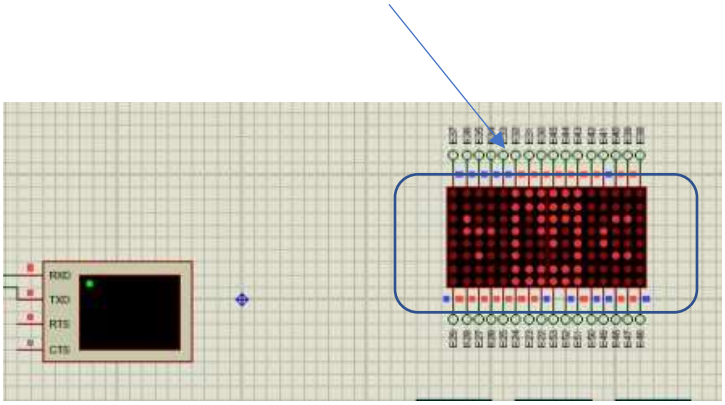


- El usuario debería mantener presionado mas de 2 segundos el botón **“K”** para inicializar el juego



EJECUCION DEL JUEGO

- Ya inicializado el juego, el objetivo principal es destruir una serie de obstáculos.

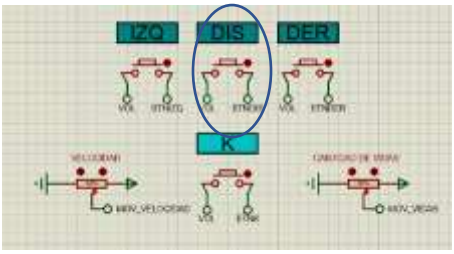


- El avión destruirá los objetivos lanzando un proyectil que descenderá poco a poco hasta destruir un objetivo o chocar con el suelo en el caso de que se falle el tiro.

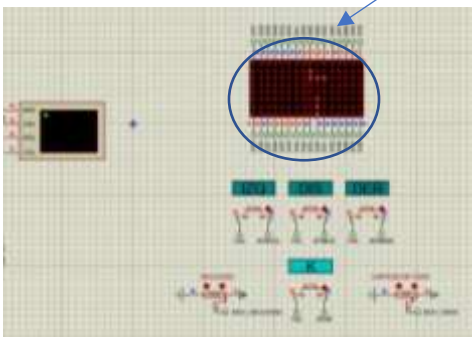
- El avión podrá avanzar en cualquier dirección horizontal, si se llega al límite de las matrices de LEDs el avión simplemente aparecerá del otro lado de forma automática.

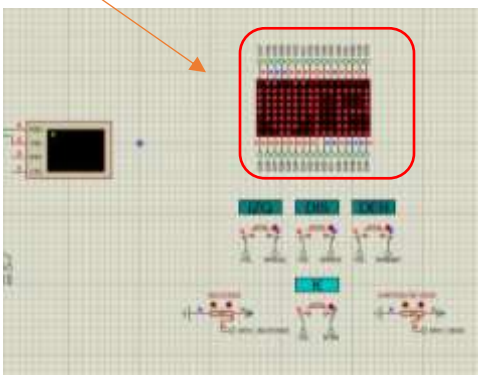
- La altura del avión no podrá ser controlada por el jugador, ésta se modifica automáticamente siguiendo la regla de que cada que pasen aproximadamente un segundo el avión descenderá en una unidad.

- el botón “D” tendrá la función de disparar el proyectil. Los botones de dirección cambiarán la dirección del avión



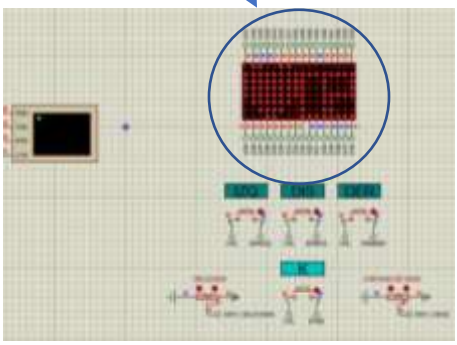
- Si cualquier parte del avión llega a chocar con un objetivo el jugador perderá una vida.





REINICIO DEL JUEGO

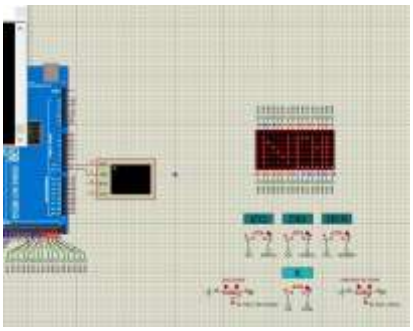
Para el reinicio del juego luego de perder una vida, el avión deberá ser colocado dos posiciones arriba de donde perdió. Originalmente contará con tres, aunque este valor podrá ser modificado como se describirá más adelante.



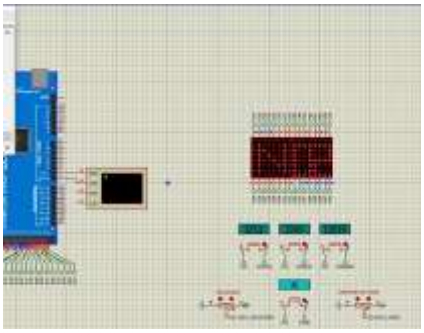
SUBIR DE NIVEL

Cuando el jugador logre destruir todos los objetivos que se le presenten avanzará de nivel. Tal hecho provocará que se incremente el número de torres a destruir

NIVEL 1



NIVEL 2



Acceso al menú Pausa

El usuario tendrá la opción para ingresar al menú pausa, En este menú solamente será necesario que se muestre el número de vidas que le quedan al jugador. Para volver al juego desde este menú se deberá presionar el botón “K” por 2 segundos.