Luis Pedro González 21513
Gabriel Carrera 21216

Informe del Proyecto: Adaptación del Juego Frog Smashers

Descripción del Juego:

Frog Smashers es un juego de pelea en el que los jugadores controlan. Cada rana puede correr, saltar entre plataformas y golpear a su oponente con un bate. El juego comienza con cada jugador teniendo 10 vidas. El objetivo es reducir las vidas del oponente a cero a través de golpes y tácticas. El jugador que se quede sin vidas primero es el perdedor.

Desarrollo:

Plataforma Central - Tiva C:

La Tiva C actuó como el núcleo del juego, encargándose de la lógica principal, procesando las entradas de los controles y mostrando los gráficos en la pantalla LCD ILI9341.

Controles Inalámbricos - ESP32:

Dos microcontroladores ESP32 fueron empleados, uno para cada control. Estos dispositivos se conectaron a una aplicación de Android mediante Bluetooth clásico, transformando el teléfono móvil en un joystick interactivo para el juego. Los ESP32 están conectados mediante UART a la TIVA. Estaos envían caracteres que se definieron para cada movimiento del personaje de manera serial.

Aplicación de Android:

Diseñada específicamente para este proyecto, la aplicación se encargó de establecer la conexión con los ESP32 y transmitir las acciones del usuario a la Tiva C de manera inalámbrica, proporcionando una experiencia de control intuitiva.

Reproducción de Música - Arduino:

Un Arduino se integró para manejar la reproducción de música y efectos sonoros. Se optó por archivos de tipo MIDI para la música, debido a su calidad y eficiencia en términos de almacenamiento. El Arduino trabajó en sincronía con la Tiva C para asegurar que la música y los efectos sonoros se alinearan con las acciones en el juego.

Almacenamiento - Tarjeta SD:

Una tarjeta SD fue esencial para guardar diversos elementos gráficos del juego, como fondos y sprites de personajes. La Tiva C accedía a estos datos para renderizar los gráficos en la pantalla LCD ILI9341.

Luis Pedro González 21513
Gabriel Carrera 21216

Funciones importantes:

Función de salto:

La función saltar está diseñada para manejar la animación y la lógica de un salto para el personaje Frog1, la lógica es la misma para la función de Frog2. Al llamar esta función, se establece que el personaje está en el aire y se registra su altura inicial. El salto se divide en dos fases: subida y bajada. Durante la fase de subida, el personaje se mueve hacia arriba y horizontalmente (a la izquierda o a la derecha) basado en su dirección actual. Durante esta fase, se muestran distintos sprites para visualizar el movimiento ascendente del salto.

En la fase de bajada, el personaje desciende y, mientras lo hace, el código verifica si ha aterrizado en una plataforma. Si es así, se ajusta su posición para que esté correctamente alineado con la plataforma y se indica que ha aterrizado. De lo contrario, continúa descendiendo hasta que completa su trayectoria de salto. Durante esta fase, se muestran otros sprites para visualizar el movimiento descendente.

Finalmente, se muestra el sprite de Frog en su estado de reposo, en la dirección en la que estaba mirando antes del salto.

```
//-----------------------------------FUNCION DEL SALTO FROG1-------------------------------------------------------
void saltar() {
  enElAire = true; // Al iniciar el salto, el personaje está en el aire
  alturaActual = posYf1;  // Actualiza la altura actual antes de iniciar el salto
  int alturaSalto = 84;  // Define la altura total del salto
  int avanceHorizontalf1 = 1;  // Define cuántos píxeles se moverá horizontalmente en cada paso del salto
  bool landed = false;  // Variable para rastrear si el personaje ha aterrizado en la plataforma
  isJumping = true;  // Activa la bandera de salto

  // Fase de subida del salto
  for (int j = 0; j < alturaSalto / 2; j++) {
    FillRect(posXf1, posYf1, frogswidth+1, frogsheight, fillmovecolor);  // Borra el sprite anterior dibujando un rectángulo del color de fondo
    posYf1--;  // Decrementa la posición vertical para mover el sprite hacia arriba
    if (movimiento == 1 && posXf1 < 320 - 26) {  // Si el movimiento es a la derecha y no se ha llegado al límite derecho
      posXf1 += avanceHorizontalf1;  // Incrementa la posición horizontal
    } else if (movimiento == -1 && posXf1 > 0) {  // Si el movimiento es a la izquierda y no se ha llegado al límite izquierdo
      posXf1 -= avanceHorizontalf1;  // Decrementa la posición horizontal
    }
    for(uint16_t i = 0; i < 3; i++) {
       LCD_Sprite(posXf1, posYf1, frogswidth, frogsheight, jumpfrog1, 5, i, movimiento == -1, 0); // Dibuja el sprite de salto correspondiente
        delay(5);  // Introduce un pequeño retardo para visualizar el sprite
    }
  }
```

Luis Pedro González 21513
Gabriel Carrera 21216

```
  // Fase de bajada del salto
  for (int j = 0; j < alturaSalto / 2; j++) {

    // Verifica la colisión con las plataformas en la fase descendente del salto
    if ((posYf1 + frogsheight >= platformHeight && posYf1 + frogsheight <= platformHeight + 5) ||
        (posYf1 + frogsheight >= platform2Height && posYf1 + frogsheight <= platform2Height + 5)) {
       // Serial.println("Colisión detectada!");
        if (posYf1 + frogsheight >= platformHeight && posYf1 + frogsheight <= platformHeight + 5) {
            posYf1 = platformHeight - frogsheight;  // Ajusta la posición Y del personaje para que esté sobre la primera plataforma
            alturaActual = platformHeight - frogsheight;  // Actualiza la altura actual
        } else {
            posYf1 = platform2Height - frogsheight;  // Ajusta la posición Y del personaje para que esté sobre la segunda plataforma
            alturaActual = platform2Height - frogsheight;  // Actualiza la altura actual
        }
        landed = true;  // Indica que el personaje ha aterrizado en una plataforma
        break;  // Termina el bucle, ya que el personaje ha aterrizado en una plataforma
    }

    FillRect(posXf1, posYf1, frogswidth, frogsheight, fillmovecolor);  // Borra el sprite anterior
    posYf1++;  // Incrementa la posición vertical para mover el sprite hacia abajo
    if (movimiento == 1 && posXf1 < 320-26) {  // Si el movimiento es a la derecha y no se ha llegado al límite derecho
      posXf1 += avanceHorizontalf1;  // Incrementa la posición horizontal
    } else if (movimiento == -1 && posXf1 > 0) {  // Si el movimiento es a la izquierda y no se ha llegado al límite izquierdo
      posXf1 -= avanceHorizontalf1;  // Decrementa la posición horizontal
    }
    for(uint16_t i = 2; i < 5; i++) {
        LCD_Sprite(posXf1, posYf1, frogswidth, frogsheight, jumpfrog1, 5, i, movimiento == -1, 0);   // Dibuja el sprite de bajada correspondiente
        delay(5);  // Introduce un pequeño retardo para visualizar el sprite
    }
  }



  // Finalización del salto
  FillRect(posXf1, posYf1, frogswidth, frogsheight, fillmovecolor);  // Borra el sprite anterior una vez que el salto ha finalizado

  if (landed) {
    enElAire = false; // Si el personaje ha aterrizado en una plataforma, restablece enElAire
  }

  isJumping = false;  // Desactiva la bandera de salto

  // Dibuja el sprite original después de finalizar el salto, con o sin flip dependiendo de la dirección
  if (movimiento == 1) {  // Si el movimiento fue a la derecha
    LCD_Sprite(posXf1+1, posYf1, frogswidth, frogsheight, runf1, 4, 0, 0, 0);  // Dibuja el sprite en reposo sin flip
  } else if (movimiento == -1) {  // Si el movimiento fue a la izquierda
    LCD_Sprite(posXf1, posYf1, frogswidth, frogsheight, runf1, 4, 0, 1, 0);  // Dibuja el sprite en reposo con flip
  }
}
```

Luis Pedro González 21513
Gabriel Carrera 21216

Función de caída de plataforma:

La función chequearPlataformaf() verifica si el personaje Frog se encuentra sobre una de las dos plataformas definidas en el juego. Esta función devuelve true si Frog está en la primera o segunda plataforma y false en caso contrario. La verificación se basa en comparar la posición actual de Frog (tanto horizontal como vertical) con las coordenadas de inicio y fin de ambas plataformas.

La función caerf1() maneja la lógica y animación de la caída de Frog. Al llamar esta función, se establece que el personaje está en el aire y se determina si Frog ha aterrizado en alguna plataforma o ha llegado al suelo. La función utiliza una lógica de colisión para determinar si Frog ha colisionado con alguna de las plataformas durante su caída. Si Frog colisiona con una plataforma, su posición vertical se ajusta para que se alinee con la plataforma y la función termina su ejecución. Si Frog no colisiona con ninguna plataforma y llega al suelo, su altura se actualiza y se establece que ha aterrizado. Durante la caída, se muestran sprites para visualizar la animación de caída de Frog.

```
bool chequearPlataformaf2() {
    return (
        // Chequea la primera plataforma
        (posXf2 + frogswidth >= platformStartX && posXf2 <= platformEndX && alturaActualf2 == platformHeight - frogsheight) ||
        // Chequea la segunda plataforma
        (posXf2 + frogswidth >= platform2StartX && posXf2 <= platform2EndX && alturaActualf2 == platform2Height - frogsheight)
    );
}

void caerf2() {
    enElAiref2 = true;  // El personaje Frog2 está en el aire
    uint8_t altinicial = 193;
    unsigned long currentMillisf2 = millis();  // Obtiene el tiempo actual

    // Si el personaje Frog2 aún no ha llegado al suelo o a una plataforma
    if (posYf2 < altinicial) {
        // Chequear colisión con la primera plataforma
        if (posXf2 + frogswidth >= platformStartX && posXf2 <= platformEndX &&
            posYf2 + frogsheight >= platformHeight && posYf2 + frogsheight <= platformHeight + 5) {
            alturaActualf2 = platformHeight - frogsheight;  // Ajusta la altura actual de Frog2
            posYf2 = alturaActualf2;  // Ajusta la posición Y del personaje Frog2
            enElAiref2 = false;  // El personaje Frog2 ha aterrizado
            return;  // Sale de la función
        }

        // Chequear colisión con la segunda plataforma
        if (posXf2 + frogswidth >= platform2StartX && posXf2 <= platform2EndX &&
            posYf2 + frogsheight >= platform2Height && posYf2 + frogsheight <= platform2Height + 5) {
            alturaActualf2 = platform2Height - frogsheight;  // Ajusta la altura actual de Frog2
            posYf2 = alturaActualf2;  // Ajusta la posición Y del personaje Frog2
            enElAiref2 = false;  // El personaje Frog2 ha aterrizado
            return;  // Sale de la función
        }

        if (currentMillisf2 - lastFrameUpdatef2 >= frameDurationf2) {  // Verifica si ha pasado el tiempo de duración de un frame para Frog2
            posYf2++;  // Incrementa la posición vertical de Frog2 para mover el sprite hacia abajo
            FillRect(posXf2, posYf2 - frogsheight, frogswidth, frogsheight, fillmovecolor);  // Borra el sprite anterior de Frog2
```

Luis Pedro González 21513
Gabriel Carrera 21216

```
        if (currentMillisf2 - lastFrameUpdatef2 >= frameDurationf2) {  // Verifica si ha pasado el tiempo de duración de un frame para Frog2
            posYf2++;  // Incrementa la posición vertical de Frog2 para mover el sprite hacia abajo
            FillRect(posXf2, posYf2 - frogsheight, frogswidth, frogsheight, fillmovecolor);  // Borra el sprite anterior de Frog2

            for (uint16_t i = 2; i < 5; i++) {
                LCD_Sprite(posXf2, posYf2, frogswidth, frogsheight, jumpfrog2, 5, i, movimientof2 == -1, 0);  // Dibuja el sprite de caída correspondiente para Frog2
            }
            lastFrameUpdatef2 = currentMillisf2;  // Actualiza la última vez que se cambió el frame para Frog2
        }
    } else {
        alturaActualf2 = posYf2;  // Actualiza la altura actual de Frog2 una vez que el personaje ha llegado al suelo (solo si no ha aterrizado en una plataforma)
        enElAiref2 = false;  // El personaje Frog2 ha aterrizado
    }
}
```

Función de colisión:

La función Colision() verifica si hay una colisión entre los personajes Frog1 y Frog2. Para cada personaje, se definen los lados de su rectángulo en función de su posición actual y las dimensiones del sprite. Estos lados son: left (izquierdo), right (derecho), top (superior) y bottom (inferior). Luego, la función verifica si hay una superposición entre los rectángulos de los dos personajes. Si los rectángulos se superponen en ambas direcciones (horizontal y vertical), significa que hay una colisión entre los personajes. En tal caso, la función devuelve true. Si no hay superposición, la función devuelve false, indicando que no hay colisión.

```
//--------------------------------FUNCION DE COLISION-----------------------------------

bool Colision() {
    // Lados del rectángulo de frog1
    int left1 = posXf1;
    int right1 = posXf1 + frogswidth;
    int top1 = posYf1;
    int bottom1 = posYf1 + frogsheight;

    // Lados del rectángulo de frog2
    int left2 = posXf2;
    int right2 = posXf2 + frogswidth;
    int top2 = posYf2;
    int bottom2 = posYf2 + frogsheight;

    // Comprueba si hay colisión
    if (left1 < right2 && right1 > left2 && top1 < bottom2 && bottom1 > top2) {
        return true;  // Hay colisión
    }
    return false;  // No hay colisión
}
```

Función de movimiento horizontal y vertical:

Los Frogs tienen la capacidad de moverse tanto a la derecha como a la izquierda en el juego. Cuando se detecta la entrada correspondiente al carácter que se definió en los controles para cada movimiento. La posición horizontal de los Frogs se incrementa o decrementa en 2 píxeles, dependiendo de la dirección, y se verifica que no hayan

excedido los bordes de la pantalla. Luego, se borra el sprite anterior de los Frogs utilizando la función V_line y, a continuación, se muestra el nuevo sprite, ajustado según su nueva posición. La animación de los Frogs al correr se maneja mediante un cálculo que determina qué frame mostrar, dependiendo de la posición actual de cada personaje.

```
// Movimiento hacia la derecha de Frog1
if (t == 'B' && !enElAire) {
    movimiento = 1;

    posXf1 += 2;
    if (posXf1 > 320 - frogswidth)
        posXf1 = 320 - frogswidth;

    // Borra el sprite anterior
    V_line(posX2f1 - 1, alturaActual, 22, fillmovecolor);
    V_line(posX2f1, alturaActual, 22, fillmovecolor);
    V_line(posX2f1 + 2, alturaActual, 22, fillmovecolor);

    uint8_t animrun = (posXf1 / 2) % 4;
    LCD_Sprite(posXf1, alturaActual, frogswidth, frogsheight, runf1, 4, animrun, 0, 0);
    V_line(posXf1 - 1, alturaActual, 22, fillmovecolor);

    posX2f1 = posXf1;
}

// Movimiento hacia la izquierda de Frog1
if (t == 'D' && !enElAire) {
    movimiento = -1;

    posXf1 -= 2;
    if (posXf1 < 0)
        posXf1 = 0;

    // Borra el sprite anterior
    V_line(posX2f1 + 24, alturaActual, 22, fillmovecolor);
    V_line(posX2f1 + 25, alturaActual, 22, fillmovecolor);
    V_line(posX2f1 + 26, alturaActual, 22, fillmovecolor);

    uint8_t animrun2 = (posXf1 / 2) % 4;
    LCD_Sprite(posXf1, alturaActual, frogswidth, frogsheight, runf1, 4, animrun2, 1, 0);
    V_line(posXf1 + 25, alturaActual, 22, fillmovecolor);

    posX2f1 = posXf1;
}
```

```
// Salto de Frog1
if (t == 'J' && millis() - lastJumpTime > jumpDebounceTime) {
    saltar();
    lastJumpTime = millis();
}
```

Luis Pedro González 21513
Gabriel Carrera 21216

Función de bateo:

La función de bateo se activa cuando se detecta la entrada del control. Antes de seguir, se verifica que haya pasado un tiempo mínimo desde el último bateo, evitando así que los jugadores bateen de manera excesiva. Una vez confirmado, se indica que los Frogs están bateando. Luego, se muestra una animación de bateo que consiste en 4 frames. Cada frame se muestra durante un corto período de tiempo para que los jugadores puedan visualizar la animación de bateo completa.

Después de batear, se verifica si ha habido una colisión entre los Frogs, utilizando la función Colision(). Si hay una colisión, se reduce una vida del Frog afectado y se inicia una animación de colisión para ese Frog. Finalmente, se indica que los Frogs han terminado de batear y se actualiza el tiempo desde el último bateo.

```
// Verifica si se ha presionado el botón de bateo y si ha pasado el tiempo mínimo requerido desde el último bateo
if (t == '3' && millis() - lastBatTime > batDebounceTime) {
    isBatting = true;  // Indica que Frog1 está bateando

    // Anima la acción de bateo mostrando 4 frames
    for (uint16_t i = 0; i < 4; i++) {
        LCD_Sprite(posXf1, alturaActual, frogswidth, frogsheight, batfrog1, 4, i, 1, 0);
        delay(100);  // Pequeña pausa para que la animación sea visible
    }

    // Devuelve a Frog1 a su posición inicial después de batear
    LCD_Sprite(posXf1, alturaActual, frogswidth, frogsheight, runf1, 4, 0, 0, 0);

    // Después de batear, verifica si hay colisión con frog2
    if (Colision()) {
        Serial.println(vidasFrog2);

        // Iniciar la animación de colisión para Frog2
        animacionColisionActiva = true;
        tiempoInicioAnimacion = millis();
        frameActualAnimacion = 0;
        LCD_Sprite(posXf2, alturaActualf2, frogswidth, frogsheight, frogcol1, 2, frameActualAnimacion, 1, 0);

        vidasFrog2--;  // Si hay colisión, reduce una vida de frog2
    }

    isBatting = false;  // Indica que Frog1 ha terminado de batear
    lastBatTime = millis();  // Actualiza el tiempo desde el último bateo
}
```