



**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE
MONTERREY**

Diseño de lógica programable

Profesores:

Raúl Peña Ortega y Enrique González Guerrero

Etapa #3. Entrega Final

Equipo #

Integrantes:

Alfonso Solis Diaz A00838034

Felipe de Jesús García García A01705893

Eugenio Javier Martinez Gastelum A0721946

3 de mayo del 2024

Índice

Índice.....	1
Introducción.....	2
Justificación.....	2
Metodología.....	3
3.1 Herramientas y Tecnologías.....	4
3.2 Conceptualización y Diseño de Niveles.....	4
3.3 Desarrollo Iterativo y Sprints.....	4
3.4 Pruebas Continuas.....	4
Juego.....	4
Interfaz física.....	5
Características de placa.....	5
--Figura 1	
Objetivo.....	6
Explicación de Escenas/mundos.....	6
--Figura 2	
Personajes.....	7
--Figura 3	
Reglas del juego.....	8
Dinámica (Relación del juego con la placa).....	8
--Figura 4	
Máquina de estados.....	9
--Figura 5	
5.1.Estado Inicial.....	10
5.2.Valores Iniciales.....	10
5.3.Descripción de Variables.....	10
5.4.Menú Principal.....	10
5.5.Selección de nivel.....	10
5.6.Pausa.....	10
5.7.Niveles.....	10
5.8.Final.....	12
Programación Placa.....	13
6.1 Descripción del Hardware.....	13
6.2 Switches.....	13
--Figura 6	
--Figura 7	
-- Figura 8	
-- Figura 9	
6.2.1 Explicación Detallada de los Switches (con código).....	13
--Figura 10	

6.3 Botones.....	13
--Figura 11	
6.3.1 Explicación Detallada de los Botones (con código).....	13
--Figura 12	
6.4 Displays de 7 Segmentos.....	13
--Figura 13	
6.4.1 Explicación Detallada de los Displays de 7 Segmentos (con código).....	14
--Figura 14	
--Figura 15	
--Figura 16	
--Figura 17	
Conexión Serial.....	23
--Figura 18	
--Figura 19	
--Figura 20	
--Figura 21	
--Figura 22	
--Figura 23	
--Figura 24	
--Figura 25	
--Figura 26	
--Figura 27	
Bibliografía.....	33

1. Introducción

John Deere es una empresa multinacional estadounidense que se destaca como uno de los líderes mundiales en la fabricación de maquinaria agrícola, equipos para la construcción, equipos forestales, motores diesel y equipamiento para cuidado del césped. Fundada en 1837 por John Deere, la compañía ha sido un pionero en la innovación y desarrollo de tecnología avanzada para la industria agrícola y de construcción.

Los tractores actuales de John Deere presentan diversas características diseñadas para aumentar la eficiencia y la productividad en el campo. Entre estas características se incluyen potentes motores diésel o de gasolina, transmisiones automáticas o de cambio de marchas, sistemas de dirección asistida y suspensión, integración de sistemas de navegación por satélite (GNSS) y tecnología de control de implementos agrícolas.

Recientemente, se ha observado un aumento en los desafíos dentro de los campos agrícolas, especialmente en terrenos no tan uniformes, lo que ha resultado en un uso ineficiente de fertilizantes y recursos. Ante esta problemática, John Deere ha tomado la iniciativa de abordar estos obstáculos de manera innovadora. Invitándonos a participar en un emocionante juego de simulación de tractores, la compañía no solo nos permite experimentar virtualmente los desafíos del campo, sino que también nos brinda la oportunidad de encontrar soluciones efectivas. A través de esta plataforma interactiva, podemos explorar diferentes estrategias para optimizar el rendimiento agrícola, minimizar el desperdicio de recursos y mejorar la productividad en condiciones de terreno difíciles.

2. Justificación del Problema

El objetivo principal de 'Aventura en el Campo de Maíz' es enfrentar los desafíos de la agricultura en terrenos irregulares. Este juego simula la gestión de recursos en dichas condiciones, fomentando el desarrollo de estrategias eficaces en un entorno interactivo.

En la actualidad, el entretenimiento interactivo es fundamental en la vida cotidiana de muchas personas. Nuestro videojuego propone una emocionante aventura agrícola que sumerge al jugador en la cabina de un tractor, dentro de un vasto campo de maíz. Con el doble objetivo de recolectar objetos y encontrar combustible, el juego impone un desafío contra el tiempo, empujando al jugador a una experiencia intensa y estratégica. La destreza en la conducción y la capacidad para diseñar estrategias de recolección eficientes son clave, ya que el jugador debe alcanzar un número determinado de objetivos en un tiempo limitado.

El juego se desarrolla en niveles progresivamente desafiantes, donde cada fase introduce nuevos obstáculos y reduce el tiempo disponible para cumplir las tareas, incentivando al jugador a mejorar constantemente su habilidad y estrategia. Este diseño no sólo proporciona entretenimiento, sino que también sumerge al jugador en una experiencia donde la rapidez y la táctica son esenciales para el éxito. Además, se ha incluido una condición crítica: si no se cumplen los objetivos en el tiempo asignado, el juego termina, añadiendo una tensión emocionante a cada nivel y forjando un entorno donde cada segundo y cada decisión cuentan.

La integración de controles físicos y lógica programable en la interfaz del juego busca expandir los límites de la inmersión y la interactividad, proporcionando una conexión tangible entre el jugador y el mundo virtual del cultivo de maíz. Con "Aventura en el Campo de Maíz", establecemos un nuevo estándar en el diseño de juegos, donde el entretenimiento y el desarrollo de habilidades prácticas se entrelazan de manera cohesiva, ofreciendo una experiencia que es tanto desafiante como gratificante.

3. Metodología

El desarrollo de "Aventura en el Campo de Maíz" adoptó una metodología ágil, centrada en la iteración rápida, la flexibilidad y la respuesta oportuna a los cambios. Este enfoque se complementa con la decisión de construir el juego en un entorno 3D, utilizando Unity como el motor de desarrollo principal debido a su amplio soporte para gráficos tridimensionales, física realista y herramientas de animación avanzadas.

3.1 Herramientas y Tecnologías

Unity fue utilizado como el motor de desarrollo principal, seleccionado por su amplio soporte para gráficos tridimensionales, física realista y herramientas avanzadas de animación. Unity facilitó la construcción de un entorno 3D robusto y permitió la integración de efectos visuales y de audio avanzados que eran fundamentales para la inmersión del juego. Además, se implementó nuestra tarjeta DE 10 lite como el mando para nuestro tractor, integrando así hardware específico en el desarrollo.

3.2 Conceptualización y Diseño de Niveles

La fase inicial se centró en la conceptualización de los niveles. Este proceso involucró la creación de bocetos y prototipos utilizando herramientas de diseño en Unity, lo cual permitió un diseño detallado de cada mundo y personaje para aprovechar al máximo las capacidades 3D del motor.

3.3 Desarrollo Iterativo y Sprints

Las fases del desarrollo se dividieron en sprints, cada uno enfocado en lograr un conjunto específico de características o mejoras. Este enfoque permitió al equipo evaluar el progreso y hacer ajustes según fuera necesario, manteniendo una respuesta oportuna a los cambios y desafíos que surgían.

3.4 Pruebas Continuas

Se realizaron pruebas frecuentes para garantizar la calidad del juego, con especial atención en la jugabilidad, los gráficos y la interacción del usuario con el entorno 3D. Estas pruebas ayudaron a identificar áreas de mejora y a asegurar que el juego cumpliera con los estándares de calidad establecidos.

4.Juego

4.1 Interfaz Física

La destreza en la conducción y la estrategia de recolección se ponen a prueba en "Aventura en el Campo de Maíz", donde la interactividad alcanza otro nivel gracias a la incorporación de lógica programable y controles basados en hardware. La interfaz física proporciona una experiencia de juego más táctil e inmersiva

4.2 Características de la Placa FPGA

Se utilizará la placa FPGA "DE-10 Lite" de Intel para la interacción entre el juego y los controles físicos, procesando las señales de entrada y traduciéndose en comandos dentro

del juego. La programación de la placa es clave para garantizar una experiencia de juego sin interrupciones y altamente responsiva:

- Steering (Acelerómetro): La dirección del tractor en el juego se controla mediante un acelerómetro, permitiendo a los jugadores navegar por el campo de maíz inclinando físicamente el controlador, replicando la sensación de conducir un tractor real.
- Gear Selection Forward/Reverse (interruptores): Los jugadores cambian las marchas del tractor usando interruptores reales, lo que afecta la velocidad del vehículo en el juego.
- Throttle (botón): Un botón actúa como el acelerador, crucial para la recolección de objetos y la optimización del tiempo de juego.
- Brake (botón): Un botón dedicado se utiliza para frenar, proporcionando a los jugadores una respuesta inmediata y precisa en situaciones que requieren una desaceleración rápida.

La inclusión de estos controles en el diseño del juego no solo aumenta la inmersión sino que también plantea interesantes desafíos en cuanto a la lógica programable y la integración de hardware.



*Figura 1
Menú principal*

4.3 Objetivo del Juego

El objetivo de "Aventura en el Campo de Maíz" es manejar un tractor a través de diferentes niveles representados por campos de maíz, recolectando objetos estratégicamente colocados y buscando combustible para mantener el tractor en funcionamiento, todo dentro de un tiempo limitado.

4.4 Explicación de Mundos / Escenas

El juego se desarrolla en un mundo tridimensional vibrante, compuesto por diversos niveles que representan campos de maíz con creciente complejidad. Cada nivel está diseñado con una estética única y presenta variaciones en la topografía, condiciones climáticas y obstáculos naturales que afectan la jugabilidad.

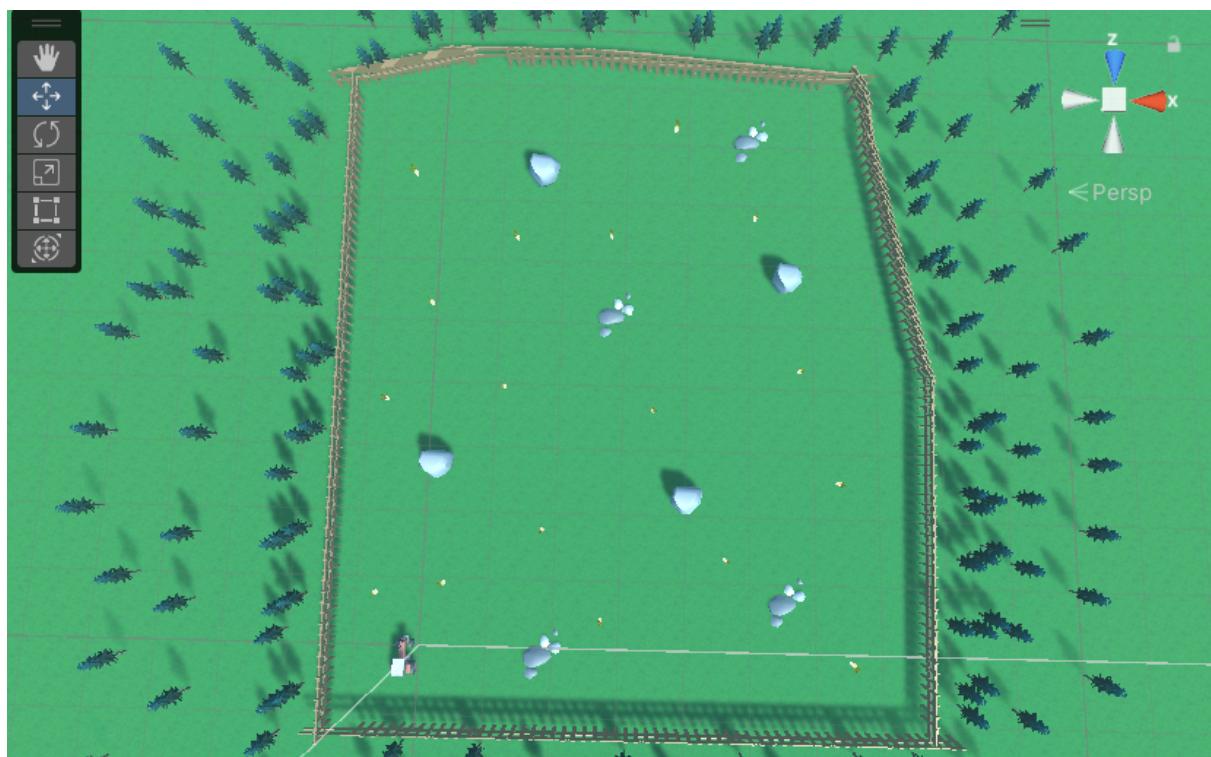


Figura #2

Escenario del nivel 1(todos los niveles se basan de este, nadamas se le agregan más obstáculos a los siguientes niveles)

4.5 Personajes

El personaje principal es el tractor, que el jugador controla. Aunque no hay personajes en el sentido tradicional, el tractor se caracteriza por su diseño y capacidad para interactuar con el entorno y cumplir las tareas requeridas.

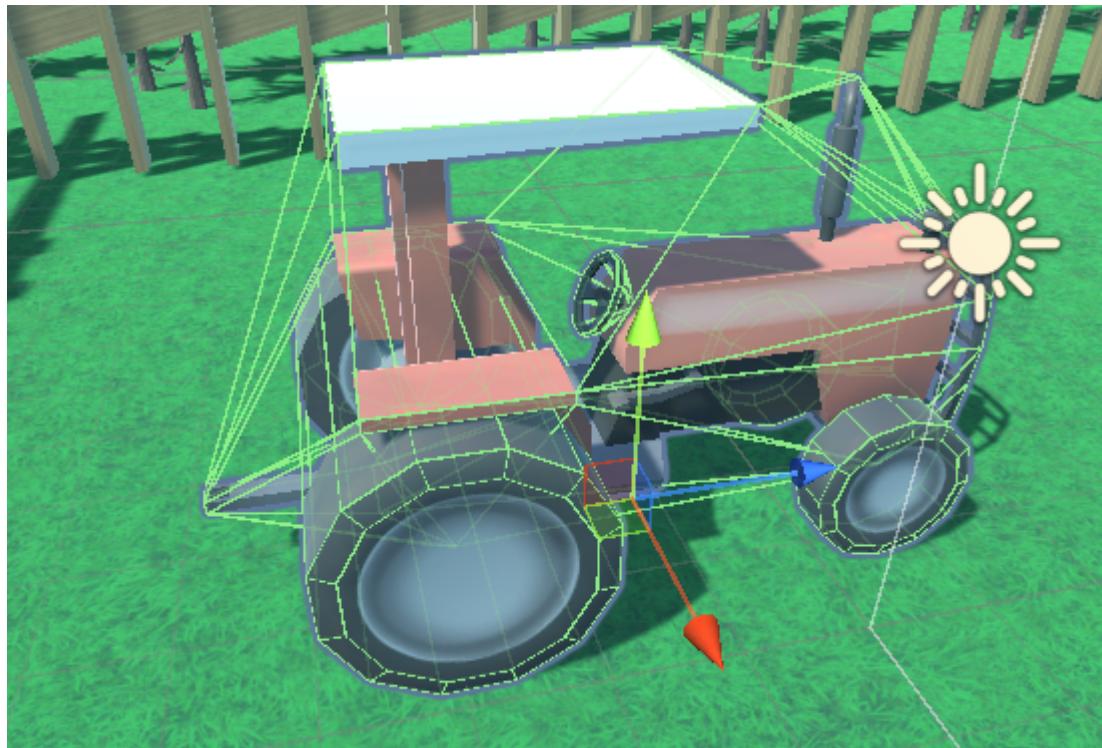


Figura 3
Imagen del tractor(personaje principal)

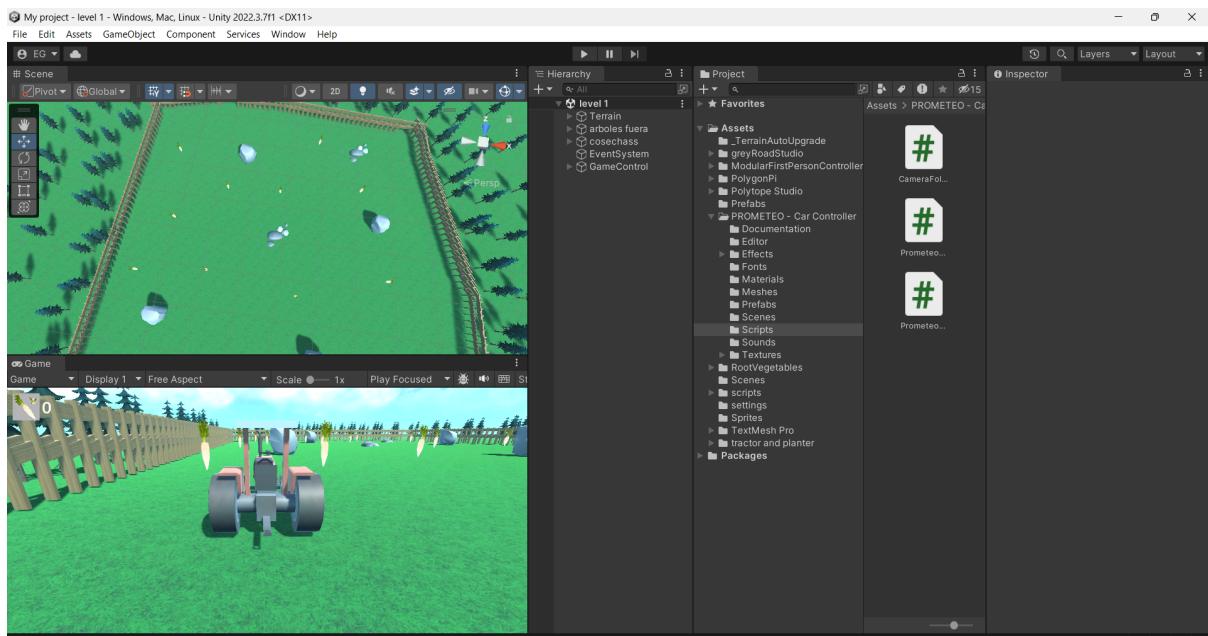
4.6 Reglas del Juego

Las reglas son claras y concisas:

- ❖ El jugador debe recolectar una serie de objetos esparcidos por el campo de maíz.
- ❖ El combustible es limitado y debe ser reabastecido buscando bidones escondidos.
- ❖ Los objetos y el combustible deben ser recolectados antes de que se acabe el tiempo.
- ❖ Chocar contra obstáculos o salirse del camino puede resultar en perdida de tiempo.

4.7 Dinámica (Relación del juego con la placa)

El juego se vincula con una placa base física, que puede incluir controles como un volante, pedales y otros interruptores que el jugador usa para interactuar con el juego. Esta interfaz mejora la experiencia de inmersión y proporciona una respuesta táctil que imita la conducción de un tractor real.



*Figura 4
Layout del primer nivel*

5. Máquina de estados

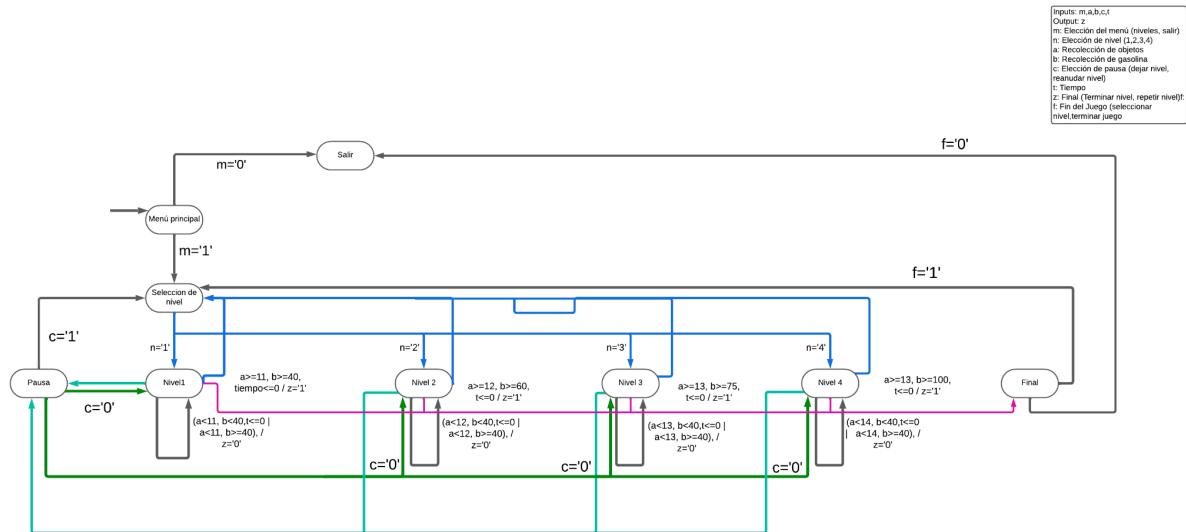


Figura 5
Foto de la máquina de estados

5.1. Estado Inicial: Menú

5.2. Valores Iniciales:

m = 0 (Elección de menú)

c = 0 (Elección en Pausa)

a = 0 (Recolección de objetos)

b = 0 (Recolección de gasolina)

t = 100-50 (Tiempo, va a ir bajando conforme avanza el nivel)

5.3. Descripción de Variables:

m: Bit que indica si el jugador desea salir del juego (m = 0) o ingresar a los niveles (m = 1).

c: Bit que durante la pausa indica si se sale a la selección de niveles (c = 1) o reanuda el nivel en el que estaba (c = 0).

a: Variable multi-bit que representa la cantidad de objetos recolectados.

b: Variable multi-bit que representa la cantidad de gasolina recolectada.

t: Variable multi-bit que representa el tiempo restante.

z: Bit que indica si se debe terminar el nivel y e ir a la selección de nivel (z = 1) o repetir el nivel (z = 0).

n: Variable multi-bit que representa al nivel que quieras ir.

f: Bit que indica si quieres ir a la selección de nivel (f= 1) o salir del juego (f=0).

Aquí está la descripción del comportamiento de la máquina de estados:

5.4.Menú Principal:

El usuario puede seleccionar un nivel (c = '1') o salir del juego (c = '0').

5.5.Selección de nivel:

El usuario puede seleccionar el nivel que él quiera (n= '1','2','3','4').

5.6.Pausa:

Esta parte del programa permite al usuario pausar el juego, dejando el nivel actual (c = '0') y regresando a la selección de niveles o reanudándolo (c = '1'). No afecta directamente el estado del juego.

5.7.Niveles:

Cada nivel tiene criterios de finalización:

- Nivel 1: Se requiere recolectar al menos 11 objetos y recolectar gasolina para que el tractor se siga moviendo antes de que se acabe el tiempo.
- Nivel 2: Se requiere recolectar al menos 12 objetos y recolectar gasolina para que el tractor se siga moviendo antes de que se acabe el tiempo.
- Nivel 3: Se requiere recolectar al menos 13 objetos y recolectar gasolina para que el tractor se siga moviendo antes de que se acabe el tiempo.
- Nivel 4: Se requiere recolectar al menos 14 objetos y recolectar gasolina para que el tractor se siga moviendo antes de que se acabe el tiempo.

Si se cumplen los criterios de finalización del nivel actual (es decir, a, b y z son mayores o iguales a los valores requeridos al momento de que t sea 0), la máquina de estados devuelve z = '1', lo que significa que se avanza al siguiente nivel.

Si alguno de los criterios de finalización no se cumple y el tiempo se agota, la máquina de estados también devuelve z = '0', lo que significa que el jugador debe repetir el nivel.

Estado Actual	m	c	a	b	t	z	Próximo Estado
MENUPRINCIPAL	0	X	X	X	X	X	SALIR
	1	X	X	X	X	X	SELECCION_NIVEL
SALIR	0	X	X	X	X	X	-
SELECCION_NIVEL	X	X	X	X	X	X	-
PAUSA	X	0	X	X	X	X	PAUSA_DEJAR

	X	1	X	X	X	X	PAUSA_REANUDAR
PAUSA_DEJAR	X	X	X	X	X	X	SELECCIÓN_NIVEL
PAUSA_REANUDAR	X	X	X	X	X	X	NIVEL (Depende del nivel en el que este)
NIVEL1	X	X	a+10	b+15	t-1	z=1	NIVEL1_FIN
NIVEL1_FIN	X	X	X	X	X	X	SELECCIÓN_NIVEL
NIVEL2	X	X	a+10	b+15	t-1	z=1	NIVEL2_FIN
NIVEL2_FIN	X	X	X	X	X	X	SELECCIÓN_NIVEL
NIVEL3	X	X	a+10	b+15	t-1	z=1	NIVEL3_FIN
NIVEL3_FIN	X	X	X	X	X	X	SELECCIÓN_NIVEL
NIVEL4	X	X	a+10	b+15	t-1	z=1	NIVEL4_FIN
NIVEL4_FIN	X	X	X	X	X	X	SELECCIÓN_NIVEL

En la tabla de asignaciones, "X" indica que la entrada o variable no se modifica en esa transición. Las expresiones aritméticas se utilizan para actualizar las variables de acuerdo con el estado actual y las acciones del jugador. Las transiciones se activan cuando se cumplen las condiciones de igualdad y relaciones especificadas en los estados.

5.8.Final:

Al completarse cualquier nivel, el jugador puede decidir si irse al menú principal ($f='1'$) o salir del juego ($f = '0'$).

4. Video Demostrativo

liga:https://www.youtube.com/watch?v=Y7WADsOAQw4&ab_channel=eugeniomartinez

6. Programación Placa

6.1 Descripción del Hardware

En este proyecto, se han utilizado componentes específicos de hardware integrados en la FPGA para facilitar el control de funcionalidades interactivas del juego:

- Switches
- Botones
- Displays de 7 Segmentos

6.2 Switches

En este proyecto se han asignado funciones específicas a cuatro switches en la FPGA, los cuales son fundamentales para la interactividad dentro del juego. Cada switch permite al usuario controlar movimientos y acciones clave del personaje o vehículo en el entorno del juego. A continuación se describe el propósito y la implementación de cada uno de estos switches:

- Drive

Switch 0 - Avanza: Este switch se utiliza para hacer avanzar al personaje o vehículo directamente hacia adelante. Su funcionalidad está codificada de la siguiente manera:

```
ELSIF SW(0) = '1' THEN
    tx_ena_de10 <= '0';
    tx_data_de10 <= "00000010"; -- Switch 0 activated
```

Figura 6

Funcionalidad de Drive en VHDL

Cuando el Switch 0 es activado ($SW(0) = '1'$), se prepara el sistema para enviar datos a través de UART desactivando el transmisor ($tx_ena_de10 \leq '0'$). La señal específica enviada ("00000010") indica la acción de avanzar, lo cual es manejado por el componente o sistema que recibe estos datos para ejecutar el movimiento hacia adelante.

- Girar a la izquierda

Este switch permite girar el vehículo o personaje hacia la izquierda dentro del juego.

```

ELSIF SW(1) = '1' THEN
    tx_ena_de10 <= '0';
    tx_data_de10 <= "00000011"; -- Switch 1 activated

```

Figura 7

Funcionalidad de giro a la izquierda en VHDL

Al activar el Switch 1, se emite la señal "00000011" que está codificada para representar la acción de girar a la izquierda. La desactivación del transmisor (`tx_ena_de10 <= '0'`) asegura que la señal puede ser enviada sin interferencia, dirigida a controladores o módulos que realizan la acción física o virtual de girar.

- Girar a la derecha

Switch 2 - Girar a la derecha: Este switch permite girar a la derecha.

```

ELSIF SW(2) = '1' THEN
    tx_ena_de10 <= '0';
    tx_data_de10 <= "00000100"; -- Switch 2 activated

```

Figura 8

Funcionalidad de giro a la Derecha en VHDL

Similar al funcionamiento del switch para girar a la izquierda, al activarse el Switch 2 se envía la señal "00000100". Esta señal le indica al sistema receptor que ejecute un giro a la derecha, controlando así la dirección de movimiento del personaje o vehículo según la lógica del juego.

- Reversa

Similar a los switches para girar, este control permite al jugador mover el vehículo o personaje en reversa.

```

ELSIF SW(3) = '1' THEN
    tx_ena_de10 <= '0';
    tx_data_de10 <= "00000101"; -- Switch 3 activated

```

Figura 9

Funcionalidad de reversa en VHDL

Cuando se activa el Switch 3, se transmite la señal "00000101", codificada para indicar un movimiento en reversa. Al igual que en los otros casos, el transmisor se desactiva (`tx_ena_de10 <= '0'`) para permitir la transmisión clara y precisa de esta instrucción al sistema que controla la dinámica de movimiento.

Estas funciones permiten una experiencia de juego más inmersiva y contribuyen al aprendizaje sobre la utilización de FPGA y la programación en VHDL.

6.2.1 Explicación Detallada de los Switches (con código)

```
-- Process to transmit data according to the interface inputs
PROCESS( CLOCK_50, SW )
BEGIN
    IF rising_edge(CLOCK_50) THEN
        IF key1_db = '1' and key1_db_past = '0' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000001"; -- Button pressed signal
        ELSIF SW(0) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000010"; -- Switch 0 activated
        ELSIF SW(1) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000011"; -- Switch 1 activated
        ELSIF SW(2) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000100"; -- Switch 2 activated
        ELSIF SW(3) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000101"; -- Switch 3 activated
        ELSE
            tx_ena_de10 <= '1';
            tx_data_de10 <= "00000000"; -- No action
        END IF;
        key1_db_past <= key1_db;
    END IF;
END PROCESS;
```

Figura 10

Código switches VHDL

- PROCESS(CLOCK_50, SW): Este proceso se activa con cada ciclo de reloj (CLOCK_50) y cuando hay un cambio en el estado de los switches (SW). Es sensible a estos dos señales para asegurar que las acciones se tomen en el momento preciso, en sincronía con el reloj del sistema.
- Condiciones de Activación de los Switches:
 - IF `SW(0) = '1' THEN`: Si el Switch 0 está activo, se envía el código "00000010". Este código está predefinido para que el sistema o el juego ejecute la acción de avanzar hacia adelante.
 - `ELSIF SW(1) = '1' THEN`: Si el Switch 1 está activo, se envía el código "00000011". Este código indica que se debe ejecutar una acción de giro hacia la izquierda.

- ELSIF SW(2) = '1' THEN: Similarmente, la activación del Switch 2 envía el código "00000100", que corresponde a la acción de girar hacia la derecha.
- ELSIF SW(3) = '1' THEN: La activación del Switch 3 manda el código "00000101", que se utiliza para indicar que el vehículo o personaje debe moverse en reversa.
- ELSE: En caso de que ninguno de los switches esté activado, se reactiva el transmisor UART (tx_ena_de10 <= '1') y se envía una señal de no acción (tx_data_de10 <= "00000000"), lo que indica que no se debe realizar ninguna operación específica hasta que se active algún switch.

6.3 Botones

Se utilizó un botón específico en la FPGA para actuar como freno del tractor, permitiendo al jugador detener completamente el movimiento del vehículo en situaciones críticas.

A continuación, se explica la configuración y el uso de este botón en el código VHDL del sistema.

```
-- Declaración del componente debounce
COMPONENT debounce IS
  PORT(
    Clock      : IN std_logic;
    button     : IN std_logic;
    debounced  : OUT std_logic
  );
END COMPONENT;

-- Instancia del componente debounce para el botón de freno
button_0  : debounce PORT MAP(
  CLOCK_50, KEY(1), key1_db
);
```

Figura 11
Código VHDL para la funcionalidad de los botones

El botón utilizado para el freno está conectado al segundo pin del vector KEY, manejado por el proceso de debounce para limpiar la señal. El proceso principal observa esta señal debounced (key1_db) y actúa en consecuencia.

Esta funcionalidad se implementa a través de un proceso en el código VHDL que monitorea el estado del botón y reacciona en consecuencia.

6.3.1 Explicación Detallada de los Botones (con código)

```
-- Proceso para transmitir datos según las entradas de interfaz, incluyendo el botón de freno
PROCESS(CLOCK_50, SW, KEY)
BEGIN
    IF rising_edge(CLOCK_50) THEN
        -- Verificación del botón de freno (debounced)
        IF key1_db = '1' AND key1_db_past = '0' THEN
            tx_ena_de10 <= '0'; -- Desactiva el transmisor UART para enviar datos
            tx_data_de10 <= "00000001"; -- Código para activar el freno del tractor
        ELSE
            tx_ena_de10 <= '1'; -- Reactiva el transmisor UART si no se presiona el botón
            tx_data_de10 <= "00000000"; -- Código de no acción
        END IF;
        key1_db_past <= key1_db; -- Actualiza el estado pasado del botón para detectar flancos
    END IF;
END PROCESS;
```

Figura 12

Explicacion de codigo VHDL

El código VHDL descrito maneja la funcionalidad de un botón en una FPGA, utilizado específicamente como freno para detener el movimiento de un tractor en un juego. Utiliza un componente de bronce para estabilizar la señal del botón y evitar activaciones erróneas por rebote. El proceso principal detecta un flanko ascendente de esta señal estabilizada (key1_db), y cuando el botón es presionado, desactiva temporalmente el transmisor UART para enviar un código específico ("00000001"), que es interpretado por el sistema del juego para ejecutar un freno inmediato del tractor. Si el botón no está presionado, el transmisor se reactiva y se envía una señal de 'no acción' ("00000000"), permitiendo que el tractor continúe su movimiento. Esta implementación garantiza una respuesta precisa y confiable del botón de freno, crucial para la interacción del usuario en situaciones críticas del juego.

6.4 Displays de 7 Segmentos

El display de 7 segmentos se emplea para mostrar el contador de los objetos recolectados por el tractor, ofreciendo una retroalimentación visual directa y efectiva al jugador sobre su progreso en la recolección de los objetos.

```

COMPONENT display_control IS
  PORT(
    hex_input : in std_logic_vector(3 downto 0);
    seg_output: out std_logic_vector(6 downto 0)
  );
END COMPONENT;

COMPONENT DisplayController IS
  Port (
    clk      : in STD_LOGIC;
    number   : in STD_LOGIC_VECTOR(7 downto 0); -- Input number 0-999
    onesOut  : out STD_LOGIC_VECTOR(6 downto 0);
    tensOut  : out STD_LOGIC_VECTOR(6 downto 0);
    hundredsOut: out STD_LOGIC_VECTOR(6 downto 0)
  );
END COMPONENT;

```

Figura 13

Código display 7 segmentos

- Controlador de Display (DisplayController): Este componente se encarga de recibir un número en forma de vector de bits y convertirlo en señales para los displays de 7 segmentos. El número total de objetos recolectados se recibe del sistema de comunicación UART (rx_data_de10), que puede ser actualizado en tiempo real basado en la actividad del juego.
- Salidas a los Displays (onesOut, tensOut, hundredsOut): Las salidas de este controlador se mapean a tres displays de 7 segmentos individualmente designados para mostrar las unidades, decenas y centenas del contador. Cada salida (HEX0, HEX1, HEX2) controla un display diferente, permitiendo representar números desde 0 hasta 999.

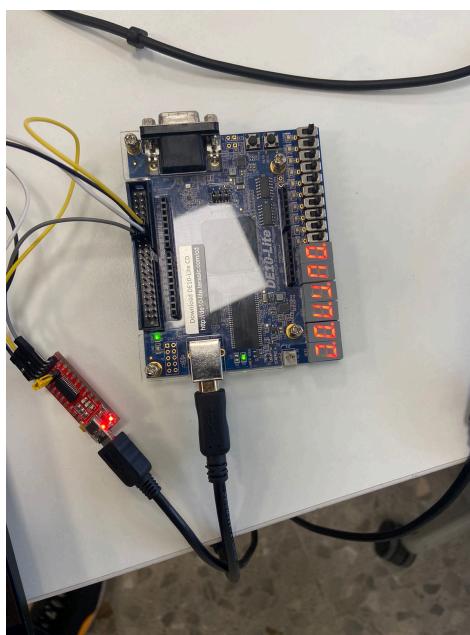


Figura evidencia Display dañado

Desafortunadamente recibimos la FPGA con algunos segmentos de los display dañados, por lo que los números no pueden ser expresados de manera correcta.

6.4.1 Explicación Detallada de los Displays de 7 Segmentos (con código)

Para proporcionar una retroalimentación visual efectiva y directa sobre el progreso de recolección en el juego, los displays de 7 segmentos son empleados para mostrar el contador de objetos recolectados por el tractor. Aquí se presenta un fragmento del código VHDL que maneja la actualización de estos displays, con comentarios detallados para explicar cada paso del proceso.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Main module to control the displays
entity DisplayController is
  Port (
    clk      : in STD_LOGIC;
    number   : in STD_LOGIC_VECTOR(7 downto 0);  -- Input number 0-999
    onesOut  : out STD_LOGIC_VECTOR(6 downto 0);
    tensOut  : out STD_LOGIC_VECTOR(6 downto 0);
    hundredsOut: out STD_LOGIC_VECTOR(6 downto 0)
  );
end DisplayController;

architecture Behavioral of DisplayController is
  signal ones, tens, hundreds : STD_LOGIC_VECTOR(3 downto 0);
  signal num_int : integer range 0 to 999;
  signal uart_rx_data : std_logic_vector(8 downto 0); -- Declare UART RX data signal

begin
  -- Convert STD_LOGIC_VECTOR to integer
  num_int <= to_integer(unsigned(number));

  -- Conversion to hundreds, tens, and ones
  hundreds <= std_logic_vector(to_unsigned((num_int / 100), 4));
  tens <= std_logic_vector(to_unsigned(((num_int mod 100) / 10), 4));
  ones <= std_logic_vector(to_unsigned((num_int mod 10), 4));

  -- Instantiate Decoder for each digit
  Decoder_ones: entity work.display_control
    port map (
      hex_input => ones,
      seg_output => onesOut
    );

  Decoder_tens: entity work.display_control
    port map (
```

Figura 14

Código explicación detallada display

```

]begin
    -- Convert STD_LOGIC_VECTOR to integer
    num_int <= to_integer(unsigned(number));

    -- Conversion to hundreds , tens , and ones
    hundreds <= std_logic_vector(to_unsigned((num_int / 100), 4));
    tens <= std_logic_vector(to_unsigned(((num_int mod 100) / 10), 4));
    ones <= std_logic_vector(to_unsigned((num_int mod 10), 4));

    -- Instantiate Decoder for each digit
Decoder_ones: entity work.display_control
    port map (
        hex_input => ones,
        seg_output => onesOut
    );

Decoder_tens: entity work.display_control
    port map (
        hex_input => tens,
        seg_output => tensOut
    );

DecoderHundreds: entity work.display_control
    port map (
        hex_input => hundreds,
        seg_output => hundredsOut
    );
end Behavioral;

```

Figura 15

Código explicación detallada display(2)

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity display_control is
6      port(
7          hex_input : in std_logic_vector(3 downto 0);
8          seg_output: out std_logic_vector(6 downto 0)
9      );
10 end entity display_control;
11
12 architecture rtl of display_control is
13 begin
14     process(hex_input)
15     begin
16         case hex_input is
17             when "0000" =>
18                 seg_output <= "0000001"; -- 0
19             when "0001" =>
20                 seg_output <= "1001111"; -- 1
21             when "0010" =>
22                 seg_output <= "0010010"; -- 2
23             when "0011" =>
24                 seg_output <= "0000110"; -- 3
25             when "0100" =>
26                 seg_output <= "1001100"; -- 4
27             when "0101" =>
28                 seg_output <= "0100100"; -- 5
29             when "0110" =>
30                 seg_output <= "0100000"; -- 6
31             when "0111" =>
32                 seg_output <= "0001111"; -- 7
33             when "1000" =>
34                 seg_output <= "0000000"; -- 8
35             when "1001" =>
36                 seg_output <= "0000100"; -- 9
37             when "1010" =>
38                 seg_output <= "0001000"; -- A

```

Figura 16

Codigo explicación detallada display(3)

```
when "000" =>
    seg_output <= "1100000"; -- b
when "1100" =>
    seg_output <= "0110001"; -- c
when "1101" =>
    seg_output <= "1000010"; -- d
when "1110" =>
    seg_output <= "0110000"; -- E
when others =>
    seg_output <= "0111000"; -- F
end case;
end process;
end architecture rtl;
```

Figura 17

Codigo explicación detallada display(4)

1. Componente DisplayController:

- Este componente define las entradas y salidas necesarias para controlar los displays de 7 segmentos. Recibe un número en formato binario (number) y divide este número en unidades, decenas y centenas, enviando cada parte a un display específico.

2. Mapeo del Componente:

- El DisplayController se configura con la señal de reloj de la FPGA (CLOCK_50) y los datos de entrada (rx_data_de10), que son actualizados mediante la comunicación UART. Cada salida del controlador está conectada a un display de 7 segmentos específico para mostrar las unidades (HEX0), las decenas (HEX1), y las centenas (HEX2).

6.4.2 Explicación Detallada del Acelerómetro (con código)

Para hacer que el tractor se mueva de derecha a izquierda, los LED son utilizados, y para ello se usa un acelerometro.

```

COMPONENT reset_delay IS
  PORT(
    iRSTN : IN std_logic;
    iCLK : IN std_logic;
    oRST : OUT std_logic
  );
END COMPONENT;

COMPONENT spi_pll IS
  PORT(
    areset : IN std_logic;
    inclk0 : IN std_logic;
    c0 : OUT std_logic;
    c1 : OUT std_logic
  );
END COMPONENT;

COMPONENT spi_ee_config IS
  PORT(
    iRSTN : IN std_logic;
    iSPI_CLK : IN std_logic;
    iSPI_CLK_OUT : IN std_logic;
    iG_INT2 : IN std_logic;
    oDATA_L : OUT std_logic_vector(7 DOWNTO 0);
    oDATA_H : OUT std_logic_vector(7 DOWNTO 0);
    SPI_SDIO : INOUT std_logic;
    oSPI_CSN : OUT std_logic;
    oSPI_CLK : OUT std_logic
  );
END COMPONENT;

COMPONENT led_driver IS
  PORT(
    iRSTN : IN std_logic;
    iCLK : IN std_logic;
    iDIG : IN std_logic_vector(9 DOWNTO 0);
    iG_INT2 : IN std_logic;
    OLED : OUT std_logic_vector(9 DOWNTO 0)
  );
END COMPONENT;

reset : reset_delay
  PORT MAP(
    iRSTN => KEY(0),
    iCLK => CLOCK_50,
    oRST => dly_rst
  );

pll : spi_pll
  PORT MAP(
    areset => dly_rst,
    inclk0 => CLOCK_50,
    c0 => spi_clk,
    c1 => spi_clk_out
  );

spi_config : spi_ee_config
  PORT MAP(
    iRSTN => NOT dly_rst,
    iSPI_CLK => spi_clk,
    iSPI_CLK_OUT => spi_clk_out,
    iG_INT2 => GSENSOR_INT(1),
    oDATA_L => data_x(7 DOWNTO 0),
    oDATA_H => data_x(15 DOWNTO 8),
    SPI_SDIO => GSENSOR_SDI,
    oSPI_CSN => GSENSOR_CS_N,
    oSPI_CLK => GSENSOR_SCLK
  );

```

```

led : led_driver
  PORT MAP(
    iRSTN => NOT dly_rst,
    iCLK => CLOCK_50,
    iDIG => data_x(9 DOWNTO 0),
    iG_INT2 => GSENSOR_INT(1),
    OLED => LEDR(9 DOWNTO 0)
  );

```

Figura 27
Código acelerómetro

Componentes y su función

- reset_delay:** Este componente parece manejar las señales de reinicio del sistema, proporcionando un reinicio sincronizado con el reloj del sistema para asegurar un inicio limpio de todos los componentes.
- spi_pll:** Es un generador de reloj basado en un PLL (Phase-Locked Loop), que produce señales de reloj estabilizadas para otros componentes, asegurando que la sincronización del sistema se mantenga precisa y coherente.

3. **spi_ee_config**: Configura la interfaz SPI para la comunicación con dispositivos externos, posiblemente un acelerómetro o otros sensores. Maneja la transmisión de datos de entrada y salida, así como la selección de chip y el reloj SPI, lo que es crítico para la correcta comunicación en sistemas que dependen de la precisión del tiempo.
4. **led_driver**: Controla un conjunto de LEDs, probablemente usados para indicar estados o alarmas del sistema. Las señales de este componente se configuran en función de entradas digitales, que podrían estar relacionadas con la información procesada desde el acelerómetro o otros sensores.

Mapeo de puertos:

Ccada componente está conectado a través de señales específicas que permiten la interacción entre ellos. Por ejemplo, el reset_delay proporciona su salida oRST a otros componentes como dly_RST para asegurar que todos los componentes se reinician sincronizadamente. El spi_pll suministra relojes a spi_ee_config, garantizando que la comunicación SPI se maneje con la precisión del reloj necesaria.

7. Conexión Serial

Se debe incluir pruebas de funcionamiento y explicaciones detalladas con funciones importantes de código (con 3 pruebas de funcionamiento que incluyan es suficiente).

Primero, necesitas configurar tu FPGA para enviar datos a través de un puerto serial, para esto se debe escribir un módulo en VHDL

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY uart IS
  GENERIC(
    clk_freq : INTEGER := 50_000_000; --frequency of system clock in Hertz
    baud_rate : INTEGER := 115_200; --data link baud rate in bits/second
    os_rate : INTEGER := 16; --oversampling rate to find center of receive bits (in samples per baud period)
    d_width : INTEGER := 8; --data bus width
    parity : INTEGER := 0; --0 for no parity, 1 for parity
    parity_eo : STD_LOGIC := '0'); --'0' for even, '1' for odd parity
  PORT(
    clk : IN STD_LOGIC; --system clock
    reset_n : IN STD_LOGIC; --asynchronous reset
    tx_ena : IN STD_LOGIC; --initiate transmission
    tx_data : IN STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --data to transmit
    rx : IN STD_LOGIC; --receive pin
    rx_busy : OUT STD_LOGIC; --data reception in progress
    rx_error : OUT STD_LOGIC; --start, parity, or stop bit error detected
    rx_data : OUT STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --data received
    tx_busy : OUT STD_LOGIC; --transmission in progress
    tx : OUT STD_LOGIC); --transmit pin
END uart;

```

Figura 18
Código conexión serial

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY de10_lite IS
  PORT(
    CLOCK_50      : IN std_logic;
    KEY           : IN std_logic_vector(1 DOWNTO 0);
    SW             : IN std_logic_vector(9 DOWNTO 0);
    GPIO_24        : IN std_logic;      -- RX
    GPIO_25        : OUT std_logic;     -- TX
    LEDR          : OUT std_logic_vector(9 DOWNTO 0);
    HEX0           : OUT std_logic_vector(6 DOWNTO 0); -- 7-segment display output fc
    HEX1           : OUT std_logic_vector(6 DOWNTO 0); -- 7-segment display output fc
    HEX2           : OUT std_logic_vector(6 DOWNTO 0); -- 7-segment display output fc
    V              : IN std_logic
  );
END de10_lite;

ARCHITECTURE Structural OF de10_lite IS

  COMPONENT debounce IS
    PORT(
      Clock       : IN std_logic;
      button      : IN std_logic;
      debounced   : OUT std_logic
    );
  END COMPONENT;

```

Figura 19
Codigo conexion serial(2)

```

COMPONENT uart IS
  GENERIC(
    clk_freq : integer := 50_000_000; -- System clock frequency in Hertz
    baud_rate : integer := 115_200; -- UART baud rate in bits per second
    os_rate : integer := 16; -- Oversampling rate (samples per baud period)
    d_width : integer := 8; -- Data bus width
    parity : integer := 0; -- 0 for no parity, 1 for parity
    parity_eo : std_logic := '0' -- '0' for even, '1' for odd parity
  );
  PORT(
    clk      : IN std_logic; -- System clock
    reset_n : IN std_logic; -- Asynchronous reset
    tx_ena  : IN std_logic; -- Initiate transmission
    tx_data : IN std_logic_vector(d_width-1 DOWNTO 0); -- Data to transmit
    rx      : IN std_logic; -- Receive pin
    rx_busy : OUT std_logic; -- Data reception in progress
    rx_error: OUT std_logic; -- Start, parity, or stop bit error detected
    rx_data : OUT std_logic_vector(d_width-1 DOWNTO 0); -- Data received
    tx_busy : OUT std_logic; -- Transmission in progress
    tx      : OUT std_logic -- Transmit pin
  );
END COMPONENT;

COMPONENT display_control IS
  PORT(
    hex_input : in std_logic_vector(3 downto 0);
    seg_output: out std_logic_vector(6 downto 0)
  );
END COMPONENT;

```

Figura 20
Codigo conexion serial(3)

Luego en Unity, usarías un script para leer el puerto serial donde el FPGA envía los datos. Este script interpretaría los datos como comandos para mover un objeto en la escena.

Prueba 1.

Acción: Rotación a la derecha.

Descripción: Para la interacción de rotación a la derecha

Para la interacción de rotación a la derecha en el juego, se utiliza una señal específica enviada desde la FPGA a través de la comunicación serial. Esta señal es interpretada por Unity para activar el movimiento correspondiente del vehículo o personaje en el juego.

```
-- En VHDL, enviar el código correspondiente al movimiento a la derecha  
tx_data_de10 <= "00000100"; -- Código para rotación a la derecha
```

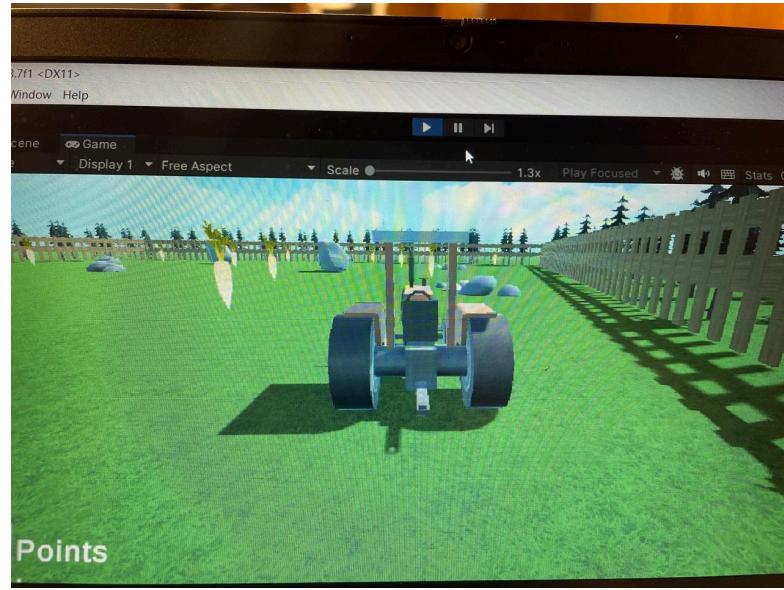
Figura 20
Código prueba 1 VHDL

Este fragmento de código establece el valor transmitido (tx_data_de10) a "00000100", que representa la acción de girar a la derecha en el sistema de la FPGA. Este código se envía cuando el switch adecuado es activado.

```
else if (value == 0x04)  
{  
|   Right = 1;  
|}  
|}
```

Figura 21
Código Unity de prueba 1

Este fragmento detecta la recepción del código "00000100" (interpretado como 0x04 en hexadecimal) y activa el estado Right a 1, que podría estar vinculado a una función o método que maneje la rotación del objeto en la escena hacia la derecha.



Prueba 2.

Acción: Rotación a la izquierda.

Descripción: Para la interacción de rotación a la izquierda

```
-- Enviar el código para rotación a la izquierda
tx_data_de10 <= "00000011"; -- Código para girar a la izquierda
```

Figura 22
Código VHDL de prueba 2

Este comando establece el valor de transmisión a "00000011" para indicar un giro a la izquierda, transmitido cuando el usuario activa el switch correspondiente en la interfaz de la FPGA.

```
else if (value == 0x03)
{
    Left = 1;
}
```

Figura 23
Código unity prueba 2

Este código en Unity escucha la señal 0x03 (el valor hexadecimal de "00000011") y activa el estado Left a 1. Esto se traduce en la aplicación como un comando para girar el objeto a la izquierda.

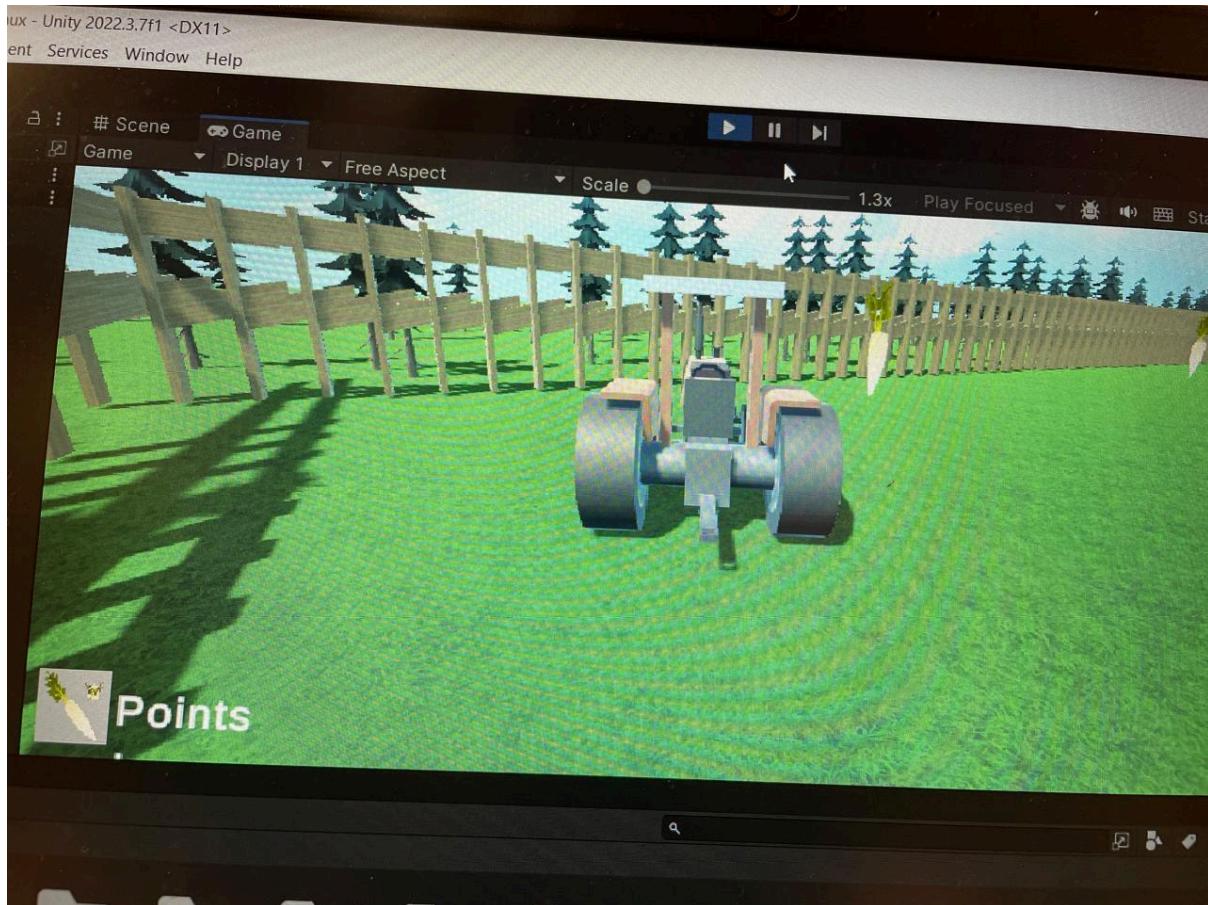


Figura 24
Prueba 1 en unity

Prueba 3.

Acción: Brake.

Para la interacción del botón de freno, se usan algunos códigos en VHDL para pasar la señal del botón de la FPGA a través de la comunicación serial, que luego es captada por Unity para ejecutar una acción de frenado inmediato.

```
-- Enviar el código para activar el freno
tx_data_de10 <= "00000001"; -- Código para activar el freno
```

Figura 24
Código VHDL prueba 3

Este código prepara la FPGA para enviar "00000001" cuando el botón de freno es presionado. Esta señal indica a Unity que active la funcionalidad de freno en el juego.

```
if (value == 0x01)
{
    Brake = 1;
    Left = 0;
    Right = 0;
    Drive = 0;
    Reverse = 0;
}
```

Figura 25
Código unity prueba 3

Al recibir la señal 0x01 (equivalente a "00000001" en VHDL), Unity configura el estado Brake a 1, lo que puede detener el movimiento del objeto en el juego, simulando un freno efectivo.

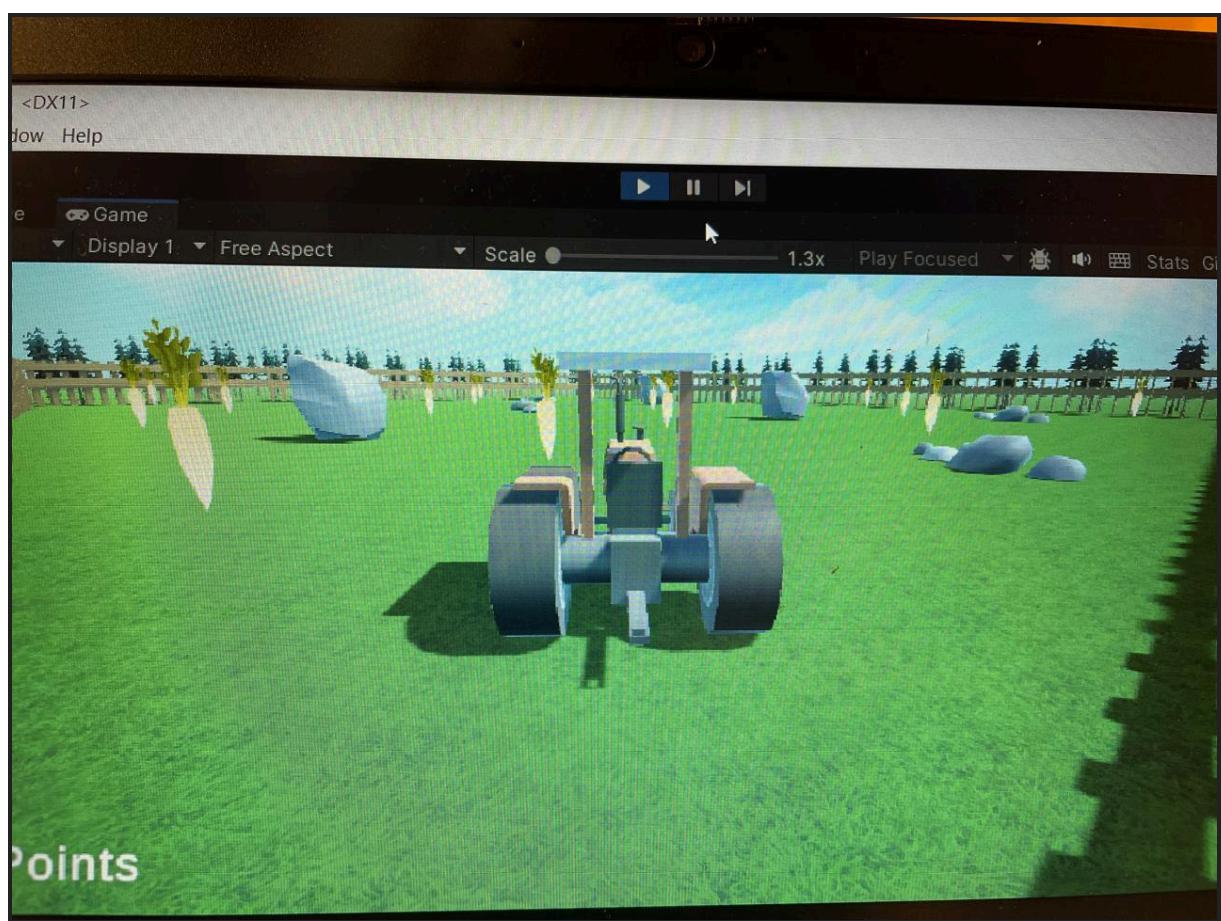


Figura 26
Prueba 3 en unity

8.Code unity

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using System.IO.Ports;
using TMPro;

public class UIController : MonoBehaviour
{
    public int Reverse = 0;
    public int Drive = 0;
    public int Brake = 0;
    public int Right = 0;
    public int Left = 0;
    public SerialPort serialPort = new SerialPort("COM3", 115200);

    public TextMeshProUGUI amountPoints;
    string amountText = "Points: ";
    int currentScore = 0;

    void OnEnable()
    {
        Cosecha.cosechaCollectedEvent += AddPoints; // Subscribe to the cosecha collected event
    }

    void OnDisable()
    {
        Cosecha.cosechaCollectedEvent -= AddPoints; // Unsubscribe from the cosecha collected event
    }

    public void ActiveScore()
    {
        amountPoints.text = amountText + "--";
    }

    public void printScore()
```

```

    {
        amountPoints.text = amountText + currentScore.ToString();
        GameControl.Instance.checkGameOver(currentScore);
    }

    public void AddPoints()
    {

        currentScore += 1; // Assuming each cosecha collection adds 10
points

        if (!serialPort.IsOpen)
        {
            serialPort.Open();
            serialPort.ReadTimeout = 1;
        }

        if (serialPort.IsOpen)
        {
            //var dataByte = new byte[] { 0x00 };
            byte[] data = BitConverter.GetBytes(currentScore);
            serialPort.Write(data, 0, 1); // Clear FPGA's LEDs at start
        }
        printScore();
    }

    void Start()
    {
        ActiveScore();
        if (!serialPort.IsOpen)
        {
            serialPort.Open();
            serialPort.ReadTimeout = 1;
        }

        if (serialPort.IsOpen)
        {
            var dataByte = new byte[] { 0x00 };
            serialPort.Write(dataByte, 0, 1); // Clear FPGA's LEDs at
start
        }
    }
}

```

```

void Update()
{
    if (!serialPort.IsOpen)
    {
        serialPort.Open();
        serialPort.ReadTimeout = 1;
    }

    if (serialPort.IsOpen)
    {
        try
        {
            int value;
            if (serialPort.BytesToRead > 0) // Verificar si hay
nuevos datos recibidos
            {
                value = serialPort.ReadByte(); // Leer el dato de 8
bits
                serialPort.DiscardInBuffer();
            }
            else
            {
                value = 0;
            }

            if (value == 0x01)
            {
                Brake = 1;
                Left = 0;
                Right = 0;
                Drive = 0;
                Reverse = 0;
            }
            else if (value == 0x02)
            {
                Drive = 1;
            }
            else if (value == 0x03)
            {
                Left = 1;
            }
            else if (value == 0x04)
            {

```

```

        Right = 1;
    }
    else if (value == 0x05)
    {
        Reverse = 1;
    }
    else if (value == 0x06)
    {
        Left = 1;
        Drive= 1;
    }
    else if (value == 0x07)
    {
        Right = 1;
        Drive= 1;
    }
    else
    {
        Brake = 0;
    }
}
catch { }
}
}

```

Figura 27
Código unity conexión serial

El propósito de este script es actuar como un puente entre el hardware de la FPGA y la lógica del juego en Unity, permitiendo que acciones físicas como presionar un botón en la FPGA resulten en cambios en el estado del juego. Esto no solo hace que la experiencia del juego sea interactiva y dinámica, sino que también utiliza tecnologías híbridas (hardware y software) para enriquecer la jugabilidad y la interacción del usuario.

9. RTL del prototipo

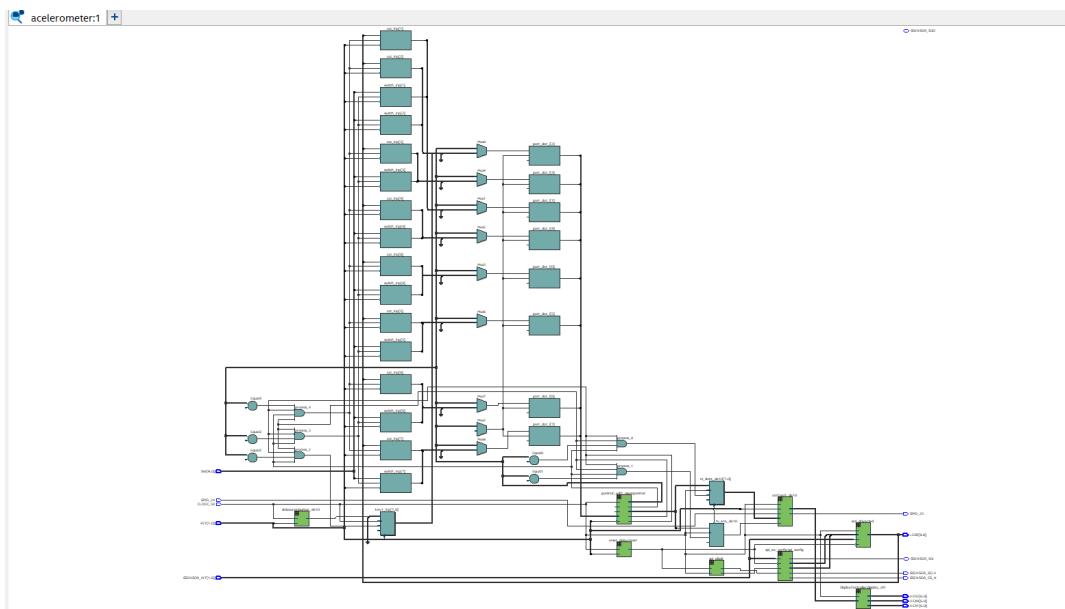


Figura 28

10. Procesador

```

; Data memory layout
    ...           data
TX_DATA:    bss 1
TX_START:   bss 1
KEY1:       bss 1
SW:         bss 1
LED:        bss 1
Cosechas:   bss 1
display:    bss 1
jump_data:  byte 0x01
jump_data2: byte 0x02
jump_data3: byte 0x03
jump_data4: byte 0x04
jump_data5: byte 0x05
tx_disable: byte 1

main:
    inp r1, KEY1
    and r1, r1, 1
    bnz boton

boton:
    ldm r2, jump_data      ; Load the data to be sent if KEY1 is pressed
    out r2, TX_DATA        ; Output the data to TX_DATA
    out r0, TX_START       ; Trigger the transmission
    ldm r2, tx_disable     ; Load the transmission disable value
    out r2, TX_START       ; Disable transmission
    jmp main               ; Return to start of polling loop

```

Figura 29.A

- Se define una serie de variables (como TX_DATA, TX_START, KEY1, LED, entre otros) con valores específicos o reservando espacio con bss.
- Se utilizan instrucciones de manipulación directa de puertos o registros (inp, out, ldm, etc.), comunes en la programación a nivel de hardware.
- El código en la sección main parece estar diseñado para manejar una entrada específica (KEY1) y, basado en esa entrada, manipular datos y controlar el flujo de un programa (como iniciar una transmisión con TX_START).
- El segmento titulado botón incluye instrucciones para manejar acciones cuando se presiona un botón, cargando datos y controlando el envío de estos.

```

inp r1, LED
and r0,r1, 16
bnz boton
and r0,r1, 48
bnz boton
and r0, r1, 32
bnz boton
and r0, r1, 64
bnz acc_left
and r0, r1, 96
bnz acc_left
and r0, r1, 128
bnz acc_left
and r0, r1, 192
bnz acc_left

```

Figura 29.B

- Para el acelerómetro, se hicieron todas las combinaciones posibles de los leds prendidos, ya que eso indicaría para donde gira. Ciertas combinaciones serían para el lado izquierdo, otras para el lado derecho y otra para mantenerse en el centro.

```

]ARCHITECTURE Structural OF de10litedram IS

]  component gumnut_with_mem IS
]    generic(
]      IMem_file_name : string      := "gumnutprojeto_text.dat";
]      DMem_file_name : string      := "gumnutprojeto_data.dat";
]      debug         : boolean     := false
]    );
]    port(
]      clk_i          : in  std_logic;
]      rst_i          : in  std_logic;
]      -- I/O port bus
]      port_cyc_o    : out std_logic;
]      port_stb_o    : out std_logic;
]      port_we_o     : out std_logic;
]      port_ack_i    : in  std_logic;
]      port_adr_o    : out unsigned(7 downto 0);
]      port_dat_o    : out std_logic_vector(7 downto 0);
]      port_dat_i    : in  std_logic_vector(7 downto 0);
]      -- Interrupts
]      int_req        : in  std_logic;
]      int_ack        : out std_logic
]    );
]  end component gumnut_with_mem;

PROCESS (CLOCK_50, rst_i)
BEGIN
  IF rst_i = '1' THEN
    key1_inp <= (OTHERS => '0'); -- Reset data input to 0 on reset
  ELSIF rising_edge(CLOCK_50) THEN
    IF port_adr_o = "00000010" AND -- Address for the second port
       port_cyc_o = '1' AND      -- Control signals for I/O
       port_stb_o = '1' AND      --
       port_we_o = '0'           -- 'read' operation
    THEN
      key1_inp <= "0000000" & key1_db;
    END IF;
  END IF;
END PROCESS;

      WITH port_adr_o SELECT
        port_dat_i <= switch_inp WHEN "00000011",
                           key1_inp WHEN "00000010",
                           acc_inp WHEN "00000100",
                           UNAFFECTED WHEN OTHERS;

```

Figura 30
código vhdl gumnut

Ahora se utilizan estas señales y al igual que los files de texto y de data que salen del código de ensamblador y el proceso de arriba muestra cómo se hacen los procesos de los switches y de los leds del acelerómetro. Lo único que cambia es el port address y la señal dummy que en este caso es key1_inp. Que en el caso del botón equivale port_dat_i, pero eso cambia con un select dependiendo si son el botón, los switches o los leds del acelerómetro.

11. Conclusiones

Eugenio: A través de este proyecto, no solo mejoré mi comprensión técnica sobre la programación de microcontroladores, sino que también descubrí la importancia de la precisión en la programación a nivel de hardware. Aprendí cómo las señales se transmiten entre el microcontrolador y los dispositivos externos, y cómo pequeños errores en el código pueden tener grandes impactos en el funcionamiento del sistema. Esta experiencia me enseñó a ser más detallado y cuidadoso al escribir código, asegurándome de verificar cada línea para evitar errores que podrían llevar horas solucionar.

Felipe: El desafío de hacer que el software interactúe correctamente con el hardware fue una de las partes más difíciles pero educativas del proyecto. Inicialmente, me enfrenté a varios problemas donde el sistema no respondía como esperaba. Esto me llevó a investigar más a fondo y a realizar múltiples pruebas para identificar dónde estaban los errores y cómo corregirlos. Finalmente, conseguir que todo funcionara juntos sin problemas fue enormemente gratificante y me demostró la importancia de la perseverancia y la prueba constante en la ingeniería.

Alfonso: Este proyecto me permitió desarrollar y afinar mis habilidades de resolución de problemas de una manera que la teoría sola nunca podría haberlo hecho. Cada vez que encontraba un error o un problema inesperado, tenía que pensar de manera crítica y creativa para encontrar una solución efectiva. Esto no solo mejoró mi habilidad para tratar con problemas técnicos, sino que también me enseñó valiosas lecciones sobre la gestión del tiempo y la priorización de tareas, ya que tenía que decidir qué problemas abordar primero y cómo distribuir mi tiempo de manera efectiva.

12. Bibliografía

Video link:

https://drive.google.com/drive/folders/1OUa02FUy_BriMw1mEzknjAkIGysMZxoH?usp=sharing

1 .Conectando Arduino y Unity a través del puerto serie | Sensorización. (s. f.).

<https://sensorizacion.blog.tartanga.eus/conectividad/conectando-arduino-y-unity-a-traves-del-puerto-serie/>

2. Editor Web1. (s. f.). Implementación del Protocolo de Comunicación SPI en VHDL para el Modulo “PmodTC1”.

<https://www.boletin.upiita.ipn.mx/index.php/ciencia/745-cyt-numero-64/1453-implementacion-del-protocolo-de-comunicacion-spi-en-vhdl-para-el-modulo-pmodtc1>

3. Información sobre la tarjeta DE10-Lite — documentación de Electrónica Digital - 1.00. (s. f.). <https://www.iuma.ulpgc.es/roberto//ed/practicas/DE10-Lite-info.html>

Anexos

VHDL

de10_lite // MANEJO PRINCIPAL DE LA TARJETA SIN GUMNUT

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY de10_lite IS
PORT(
    CLOCK_50 : IN std_logic;
    KEY      : IN std_logic_vector(1 DOWNTO 0);
    SW       : IN std_logic_vector(9 DOWNTO 0);
    GPIO_24  : IN std_logic; -- RX
    GPIO_25  : OUT std_logic; -- TX
    LEDR     : OUT std_logic_vector(9 DOWNTO 0);
    HEX0     : OUT std_logic_vector(6 DOWNTO 0); -- 7-segment display output for
hundreds
    HEX1     : OUT std_logic_vector(6 DOWNTO 0); -- 7-segment display output for
tens
    HEX2     : OUT std_logic_vector(6 DOWNTO 0); -- 7-segment display output for
units
    V        : IN std_logic
);
END de10_lite;
```

ARCHITECTURE Structural OF de10_lite IS

```
COMPONENT debounce IS
PORT(
```

```

        Clock      : IN std_logic;
        button     : IN std_logic;
        debounced  : OUT std_logic
    );
END COMPONENT;

COMPONENT uart IS
    GENERIC(
        clk_freq : integer := 50_000_000; -- System clock frequency in
Hertz
        baud_rate : integer := 115_200;   -- UART baud rate in bits per
second
        os_rate   : integer := 16;       -- Oversampling rate (samples per baud
period)
        d_width   : integer := 8;        -- Data bus width
        parity    : integer := 0;        -- 0 for no parity, 1 for parity
        parity_eo : std_logic := '0'    -- '0' for even, '1' for odd parity
    );
    PORT(
        clk      : IN std_logic; -- System clock
        reset_n : IN std_logic; -- Asynchronous reset
        tx_ena  : IN std_logic; -- Initiate transmission
        tx_data : IN std_logic_vector(d_width-1 DOWNTO 0); -- Data to
transmit
        rx      : IN std_logic; -- Receive pin
        rx_busy : OUT std_logic; -- Data reception in progress
        rx_error : OUT std_logic; -- Start, parity, or stop bit error detected
        rx_data : OUT std_logic_vector(d_width-1 DOWNTO 0); -- Data
received
        tx_busy : OUT std_logic; -- Transmission in progress
        tx      : OUT std_logic -- Transmit pin
    );
END COMPONENT;

COMPONENT display_control IS
    PORT(
        hex_input : in std_logic_vector(3 downto 0);
        seg_output: out std_logic_vector(6 downto 0)
    );
END COMPONENT;

COMPONENT DisplayController is

```

```

Port (
    clk      : in STD_LOGIC;
    number   : in STD_LOGIC_VECTOR(7 downto 0); -- Input number
0-999
    onesOut  : out STD_LOGIC_VECTOR(6 downto 0);
    tensOut  : out STD_LOGIC_VECTOR(6 downto 0);
    hundredsOut: out STD_LOGIC_VECTOR(6 downto 0)
);
END COMPONENT;

SIGNAL tx_ena_de10  : std_logic := '1';
SIGNAL tx_data_de10 : std_logic_vector(7 DOWNTO 0);
SIGNAL rx_busy_de10 : std_logic;
SIGNAL rx_error_de10 : std_logic;
SIGNAL rx_data_de10 : std_logic_vector(7 DOWNTO 0);
SIGNAL tx_busy_de10 : std_logic;
SIGNAL key1_db      : std_logic;
SIGNAL key1_db_past : std_logic := '0';
SIGNAL hundreds_digit, tens_digit, ones_digit : STD_LOGIC_VECTOR(3 downto
0);
SIGNAL display_number : std_logic_vector(7 downto 0);
SIGNAL rx_data_buffer : std_logic_vector(23 downto 0) := (others => '0');

BEGIN

-- Instance of DisplayController
display_ctrl : DisplayController
    PORT MAP(
        clk      => CLOCK_50,
        number   => rx_data_de10,
        onesOut  => HEX0,
        tensOut  => HEX1,
        hundredsOut=> HEX2

        --
    );
    uart_0      : uart PORT MAP(
        CLOCK_50, KEY(0), tx_ena_de10, tx_data_de10, GPIO_24, rx_busy_de10,
        rx_error_de10, rx_data_de10, tx_busy_de10, GPIO_25
    );
    button_0    : debounce PORT MAP(

```

```

CLOCK_50, KEY(1), key1_db
);

-- Process to transmit data according to the interface inputs
PROCESS( CLOCK_50, SW )
BEGIN
    IF rising_edge( CLOCK_50 ) THEN
        IF key1_db = '1' and key1_db_past = '0' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000001"; -- Button pressed signal
        ELSIF SW(0) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000010"; -- Switch 0 activated
        ELSIF SW(1) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000011"; -- Switch 1 activated
        ELSIF SW(2) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000100"; -- Switch 2 activated
        ELSIF SW(3) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000101"; -- Switch 3 activated
        ELSE
            tx_ena_de10 <= '1';
            tx_data_de10 <= "00000000"; -- No action
        END IF;

        key1_db_past <= key1_db;
    END IF;
END PROCESS;

```

END Structural;

de10_lite // MANEJO PRINCIPAL DE LA TARJETA CON GUMNUT

```

LIBRARY      ieee;
USE         ieee.std_logic_1164.all, ieee.numeric_std.all;

```

ENTITY de10litegum IS

```

PORT(
    CLOCK_50      : IN      std_logic;
    KEY           : IN      std_logic_vector( 1 DOWNTO 0 );
    SW             : IN      std_logic_vector( 9 DOWNTO 0 );
    GPIO_24        : IN      std_logic;          --RX
    GPIO_25        : OUT     std_logic;          --TX
    LEDR          : BUFFER   std_logic_vector( 9 DOWNTO 0 );
    HEX0          : OUT     std_logic_vector(6 DOWNTO 0); -- 7-segment display output for
hundreds
    HEX1          : OUT     std_logic_vector(6 DOWNTO 0); -- 7-segment display output for tens
    HEX2          : OUT     std_logic_vector(6 DOWNTO 0); -- 7-segment display output for units
    GSENSOR_CS_N : OUT     std_logic;          -- Chip select for G-Sensor, output
    GSENSOR_INT  : IN      std_logic_vector(1 DOWNTO 0); -- G-Sensor interrupt lines, input
    GSENSOR_SCLK : OUT     std_logic;          -- Serial Clock for G-Sensor, output
    GSENSOR_SDI  : INOUT   std_logic;          -- Serial Data In/Out for G-Sensor, bidirection
    GSENSOR_SDO  : INOUT   std_logic
);
END de10litegum;

```

ARCHITECTURE Structural OF de10litegum IS

```

component gumnut_with_mem IS
    generic(
        IMem_file_name      : string := "gumnutproyecto_text.dat";
        DMem_file_name : string := "gumnutproyecto_data.dat";
        debug            : boolean := false
    );
    port(
        clk_i              : in      std_logic;
        rst_i               : in      std_logic;
-- I/O port bus
        port_cyc_o         : out     std_logic;
        port_stb_o         : out     std_logic;
        port_we_o          : out     std_logic;
        port_ack_i         : in      std_logic;
        port_addr_o        : out     unsigned(7 downto 0);
        port_dat_o         : out     std_logic_vector(7 downto 0);
        port_dat_i         : in      std_logic_vector(7 downto 0);
-- Interrupts
        int_req             : in      std_logic;
        int_ack             : out     std_logic
    );
end component gumnut_with_mem;

```

COMPONENT uart IS

```
    GENERIC(
```

```

clk_freq      : integer  := 50_000_000;      --frequency of system clock
in Hertz
baud_rate     : integer  := 115_200;        --data link baud rate in
bits/second
os_rate       : integer  := 16;           --oversampling rate to find center of
receive bits (in samples per baud period)
d_width       : integer  := 8;            --data bus width
parity        : integer  := 0;            --0 for no parity, 1 for parity
parity_eo     : std_logic := '0'          --'0' for
even, '1' for odd parity
);
PORT(
    clk      : IN std_logic;             --system clock
    reset_n   : IN std_logic;            --asynchronous reset
    tx_ena    : IN std_logic;            --initiate transmission
    tx_data   : IN std_logic_vector(d_width-1 DOWNTO 0); --data to
transmit
    rx       : IN std_logic;            --receive pin
    rx_busy   : OUT std_logic;          --data reception in
progress, LEDR(9)
    rx_error  : OUT std_logic;          --start, parity, or stop bit
error detected
    rx_data   : OUT std_logic_vector(d_width-1 DOWNTO 0); --data
received
    tx_busy   : OUT std_logic;          --transmission in progress,
LEDR(8)
    tx       : OUT std_logic;
--transmit pin
);
END COMPONENT;
```

```

COMPONENT debounce IS
    PORT(
        Clock      : IN std_logic;
        button     : IN std_logic;
        debounced  : BUFFER std_logic
    );
END COMPONENT;
```

```

COMPONENT display_control IS
    PORT(
        hex_input : in std_logic_vector(3 downto 0);
        seg_output: out std_logic_vector(6 downto 0)
    );
END COMPONENT;
```

```

COMPONENT DisplayController is
    Port (
        clk      : in STD_LOGIC;
        number   : in STD_LOGIC_VECTOR(7 downto 0); -- Input number 0-999
        onesOut  : out STD_LOGIC_VECTOR(6 downto 0);
        tensOut  : out STD_LOGIC_VECTOR(6 downto 0);
        hundredsOut: out STD_LOGIC_VECTOR(6 downto 0)
    );
END COMPONENT;

COMPONENT reset_delay IS
    PORT( iRSTN : IN std_logic;
          iCLK  : IN std_logic;
          oRST  : OUT std_logic
    );
END COMPONENT;

COMPONENT spi_pll IS
    PORT( areset  : IN std_logic;
          inclk0 : IN std_logic;
          c0     : OUT std_logic;
          c1     : OUT std_logic
    );
END COMPONENT;

COMPONENT spi_ee_config IS
    PORT( iRSTN         : IN std_logic;
          iSPI_CLK       : IN std_logic;
          iSPI_CLK_OUT   : IN std_logic;
          iG_INT2        : IN std_logic;
          oDATA_L        : OUT std_logic_vector(7 DOWNTO 0);
          oDATA_H        : OUT std_logic_vector(7 DOWNTO 0);
          SPI_SDIO       : INOUT std_logic;
          oSPI_CSN        : OUT std_logic;
          oSPI_CLK        : OUT std_logic
    );
END COMPONENT;

COMPONENT led_driver IS
    PORT( iRSTN : IN std_logic;
          iCLK   : IN std_logic;
          iDIG   : IN std_logic_vector(9 DOWNTO 0);
          iG_INT2 : IN std_logic;
          oLED   : OUT std_logic_vector(9 DOWNTO 0)
    );
END COMPONENT;

```

```

SIGNAL clk_i, rst_i : std_logic;
SIGNAL port_cyc_o, port_stb_o : std_logic;
SIGNAL port_we_o, port_ack_i: std_logic;
SIGNAL int_req, int_ack : std_logic;
SIGNAL port_adr_o : unsigned(7 DOWNTO 0);
SIGNAL port_dat_o, port_dat_i : std_logic_vector(7 DOWNTO 0);

SIGNAL tx_ena_de10 : std_logic := '1';
SIGNAL tx_data_de10, rx_data_de10 : std_logic_vector(7 DOWNTO 0);
SIGNAL rx_busy_de10, rx_error_de10 : std_logic;
SIGNAL tx_busy_de10 : std_logic;
SIGNAL dly_RST: std_logic;
SIGNAL spi_clk: std_logic;
SIGNAL spi_clk_out: std_logic;
SIGNAL data_x: std_logic_vector(15 DOWNTO 0);

SIGNAL key1_db : std_logic;
SIGNAL switch_inp : std_logic_vector(7 DOWNTO 0);
SIGNAL key1_inp : std_logic_vector(7 DOWNTO 0);
SIGNAL led_inp : std_logic_vector(7 DOWNTO 0);
SIGNAL acc_inp : std_logic_vector(7 DOWNTO 0);
SIGNAL dec_inp : std_logic_vector(3 DOWNTO 0);
CONSTANT ADDR_TX_DATA: unsigned(7 DOWNTO 0) := "00000000";
CONSTANT ADDR_TX_START: unsigned(7 DOWNTO 0) := "00000001";

```

BEGIN

```
rst_i <= not KEY( 0 );
```

```
gumnut : gumnut_with_mem
```

PORT MAP(

```

CLOCK_50,
rst_i,
port_cyc_o,
port_stb_o,
port_we_o,
'1',
port_adr_o,
port_dat_o,
port_dat_i,
int_req,
int_ack
);
```

```
uart_de10 : uart
```

```

PORT MAP(
    CLOCK_50,
    KEY(0),
    tx_ena_de10,
    tx_data_de10,
    GPIO_24,
    rx_busy_de10,
    rx_error_de10,
    rx_data_de10,
    tx_busy_de10,
    GPIO_25
);

button_de10 : debounce
PORT MAP(
    CLOCK_50,
    KEY(1),
    key1_db
);

reset : reset_delay
PORT MAP(
    iRSTN => KEY(0),
    iCLK => CLOCK_50,
    oRST => dly_RST
);

pll : spi_pll
PORT MAP(
    areset => dly_RST,
    inclk0 => CLOCK_50,
    c0 => spi_clk,
    c1 => spi_clk_out
);

spi_config : spi_ee_config
PORT MAP(
    iRSTN => NOT dly_RST,
    iSPI_CLK => spi_clk,
    iSPI_CLK_OUT => spi_clk_out,
    iG_INT2 => GSENSOR_INT(1),
    oDATA_L => data_x(7 DOWNTO 0),
    oDATA_H => data_x(15 DOWNTO 8),
    SPI_SDIO => GSENSOR_SDI,
    oSPI_CSN => GSENSOR_CS_N,
);

```

```

          oSPI_CLK    => GSENSOR_SCLK
        );
      );

-- Instance of DisplayController
display_ctrl : DisplayController
PORT MAP(
  clk      => CLOCK_50,
  number   => rx_data_de10,
  onesOut  => HEX0,
  tensOut  => HEX1,
  hundredsOut=> HEX2
);

led : led_driver
PORT MAP(
  iRSTN  => NOT dly_rst,
  iCLK   => CLOCK_50,
  iDIG   => data_x(9 DOWNTO 0),
  iG_INT2 => GSENSOR_INT(1),
  oLED   => LEDR(9 DOWNTO 0)
);
);

-- Output => TX_DATA -> data memory address: 0x00
PROCESS (CLOCK_50, rst_i)
BEGIN
  IF rst_i = '1' THEN
    tx_data_de10 <= (OTHERS => '0');
  ELSIF rising_edge(CLOCK_50) THEN
    IF port_adr_o = ADDR_TX_DATA AND port_cyc_o = '1' AND port_stb_o = '1' AND
port_we_o = '1' THEN
      tx_data_de10 <= port_dat_o;
    END IF;
  END IF;
END PROCESS;

-- Triggering TX_START at address 0x01
PROCESS (CLOCK_50, rst_i)
BEGIN
  IF rst_i = '1' THEN
    tx_ena_de10 <= '1';
  ELSIF rising_edge(CLOCK_50) THEN

```

```

        IF port_adr_o = ADDR_TX_START AND port_cyc_o = '1' AND
port_stb_o = '1' AND port_we_o = '1' THEN
            tx_ena_de10 <= port_dat_o(0);
        END IF;
    END IF;
END PROCESS;

```

```

PROCESS (CLOCK_50, rst_i)
BEGIN
    IF rst_i = '1' THEN
        key1_inp <= (OTHERS => '0'); -- Reset data input to 0 on reset
    ELSIF rising_edge(CLOCK_50) THEN
        IF port_adr_o = "00000010" AND -- Address for the second port
           port_cyc_o = '1' AND      -- Control signals for I/O
           port_stb_o = '1' AND
           port_we_o = '0'          -- 'read' operation
        THEN
            key1_inp <= "0000000" & key1_db;
        END IF;
    END IF;
END PROCESS;

```

```

PROCESS (CLOCK_50, rst_i)
BEGIN
    IF rst_i = '1' THEN
        switch_inp <= (OTHERS => '0'); -- Reset data input to 0 on reset
    ELSIF port_adr_o = "00000011" AND -- Address for the switches input
          port_cyc_o = '1' AND      -- Control signals for I/O
          port_stb_o = '1' AND
          port_we_o = '0'
    THEN
        switch_inp <= SW(7 DOWNTO 0); -- Reading
switch SW(0)
    END IF;
END PROCESS;

```

```

PROCESS (CLOCK_50, rst_i)
BEGIN
    IF rst_i = '1' THEN
        acc_inp <= (OTHERS => '0'); -- Reset data input to 0 on reset
    ELSIF port_adr_o = "00000100" AND -- Address for the switches input
          port_cyc_o = '1' AND      -- Control signals for I/O

```

```

        port_stb_o = '1' AND
        port_we_o  = '0'

    THEN
        acc_inp <= LEDR(7 DOWNTO 0);
    END IF;
END PROCESS;

PROCESS(clk_i)
BEGIN
    IF rising_edge( clk_i ) THEN
        IF (port_cyc_o = '1' and port_stb_o = '1' and port_we_o = '1' and
port_adr_o = "00000101") THEN
            -- Se actualiza el display en función de los datos recibidos del
componente
            CASE port_dat_o(3 DOWNTO 0) IS
                WHEN "0000" => HEX0(7 DOWNTO 0)
                <= "00000001";
                WHEN "0001" => HEX0(7 DOWNTO 0)
                <= "01001111";
                WHEN "0010" => HEX0(7 DOWNTO 0)
                <= "00010010";
                WHEN "0011" => HEX0(7 DOWNTO 0)
                <= "00000110";
                WHEN "0100" => HEX0(7 DOWNTO 0)
                <= "01001100";
                WHEN "0101" => HEX0(7 DOWNTO 0)
                <= "00100100";
                WHEN "0110" => HEX0(7 DOWNTO 0)
                <= "00100000";
                WHEN "0111" => HEX0(7 DOWNTO 0)
                <= "00001111";
                WHEN "1000" => HEX0(7 DOWNTO 0)
                <= "00000000";
                WHEN "1001" => HEX0(7 DOWNTO 0)
                <= "00000100";
                WHEN OTHERS => HEX0(7 DOWNTO
0) <= "11111111";
            END CASE;
        END IF;
    END IF;
END PROCESS;

PROCESS( clk_i )
BEGIN
    IF rising_edge( clk_i ) THEN

```

```

        IF (port_adr_o = "000000110" and port_cyc_o = '1' and port_stb_o = '1' and
port_we_o = '0') THEN
            dec_inp <= rx_data_de10(3 DOWNTO 0);
        END IF;
    END IF;
END PROCESS;

WITH port_adr_o SELECT

    port_dat_i <= switch_inp WHEN "00000011",
                                key1_inp WHEN "00000010",
                                acc_inp WHEN "00000100",
                                UNAFFECTED WHEN OTHERS;

END Structural;
```

Ensamblador-Gumnut

```

text
org 0x000 ; start here on reset
jmp main

; Data memory layout
data

TX_DATA:    bss 1
TX_START:   bss 1
KEY1:       bss 1
SW:          bss 1
LED:         bss 1
Cosechas:   bss 1
display:    bss 1
jump_data:  byte 0x01
jump_data2: byte 0x02
jump_data3: byte 0x03
jump_data4: byte 0x04
jump_data5: byte 0x05
tx_disable: byte 1

; Main program
text
org 0x010

main:
    inp r1, KEY1
    and r1, r1, 1
    bnz boton
        inp r1, SW
        and r1, r1, 1
    bnz switch0
```

```

inp r1, SW
and r1, r1, 2
bnz switch1
inp r1, LED
and r0,r1, 16
bnz boton
and r0,r1, 48
bnz boton
and r0, r1, 32
bnz boton
and r0, r1, 64
bnz acc_left
and r0, r1, 96
bnz acc_left
and r0, r1, 128
bnz acc_left
and r0, r1, 192
bnz acc_left
inp r1, LED
and r0, r1, 1
bnz acc_right
and r0, r1, 2
bnz acc_right
and r0, r1, 4
bnz acc_right
and r0, r1, 6
bnz acc_right
and r0,r1, 8
bnz acc_right
inp r1, Cosechas
jmp DisCos

```

boton:

```

ldm r2, jump_data      ; Load the data to be sent if KEY1 is pressed
out r2, TX_DATA        ; Output the data to TX_DATA
out r0, TX_START       ; Trigger the transmission
ldm r2, tx_disable     ; Load the transmission disable value
out r2, TX_START       ; Disable transmission
jmp main               ; Return to start of polling loop

```

switch0:

```

ldm r2, jump_data2    ; Load the data to be sent if SW0 is pressed
out r2, TX_DATA        ; Output the data to TX_DATA
out r0, TX_START       ; Trigger the transmission
ldm r2, tx_disable     ; Load the transmission disable value
out r2, TX_START       ; Disable transmission

```

```

        jmp main      ; Return to start of polling loop

acc_left:
        ldm r2, jump_data3    ; Load the data to be sent if SW1 is pressed
        out r2, TX_DATA       ; Output the data to TX_DATA
        out r0, TX_START      ; Trigger the transmission
        ldm r2, tx_disable    ; Load the transmission disable value
        out r2, TX_START      ; Disable transmission
        jmp main      ; Return to start of polling loop

acc_right:
        ldm r2, jump_data4    ; Load the data to be sent if SW2 is pressed
        out r2, TX_DATA       ; Output the data to TX_DATA
        out r0, TX_START      ; Trigger the transmission
        ldm r2, tx_disable    ; Load the transmission disable value
        out r2, TX_START      ; Disable transmission
        jmp main      ; Return to start of polling loop

switch1:
        ldm r2, jump_data5    ; Load the data to be sent if SW3 is pressed
        out r2, TX_DATA       ; Output the data to TX_DATA
        out r0, TX_START      ; Trigger the transmission
        ldm r2, tx_disable    ; Load the transmission disable value
        out r2, TX_START      ; Disable transmission
        jmp main      ; Return to start of polling loop

DisCos:   ldm r2, Cosechas
          out r2, display

delay:    add r7, r0, 0
again3:   add r6, r0, 0
again2:   add r5, r0, 0
again1:   add r5, r5, 1
          sub r0, r5, 0xFF
          bnz again1
          add r6, r6, 1
          sub r0, r6, 0xFF
          bnz again2
          add r7, r7, 1
          sub r0, r7, 0x0C
          bnz again3
          ret

```



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CollisionTest : MonoBehaviour
6  {
7      void OnTriggerEnter(Collider other)
8      {
9          if (other.gameObject.tag == "Tractor")
10         {
11             print("ENTER");
12         }
13     }
14
15     void OnTriggerStay(Collider other)
16     {
17         if (other.gameObject.tag == "Tractor")
18         {
19             print("STAY");
20         }
21     }
22
23     void OnTriggerExit(Collider other)
24     {
25         if (other.gameObject.tag == "Tractor")
26         {
27             print("EXIT");
28         }
29     }
30 }
31
```



```
1  using System.Collections;
2  using UnityEngine;
3
4  public class Cosecha : MonoBehaviour
5  {
6      public delegate void OnCosechaCollected(); // Delegate for cosecha collected event
7      public static event OnCosechaCollected cosechaCollectedEvent;
8
9      private void OnTriggerEnter(Collider other)
10     {
11         PlayerInventory playerInventory = other.GetComponent<PlayerInventory>();
12
13         if (playerInventory != null)
14         {
15             playerInventory.CosechaCollected();
16             gameObject.SetActive(false);
17             Debug.Log("Cosecha collected and deactivated.");
18
19             // Trigger event when cosecha is collected
20             cosechaCollectedEvent?.Invoke();
21
22             Invoke("ReactivateObject", 5); // Use Invoke to delay the call to ReactivateObject
23         }
24     }
25
26     private void ReactivateObject()
27     {
28         Debug.Log("ReactivateObject called. Reactivating object.");
29         gameObject.SetActive(true); // Reactivate the game object
30     }
31 }
32
33 }
```

```
● ● ●
```

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using UnityEngine.SceneManagement;
6  using UnityEngine.VFX;
7
8  public class GameControl : MonoBehaviour
9  {
10     static public GameControl Instance;
11     public GameObject birdBehaviour;
12     public UIController UIController;
13     public int pointsToWin = 15;
14
15     // Start is called before the first frame update
16     void Start()
17     {
18
19     }
20
21     // Update is called once per frame
22     void Update()
23     {
24
25     }
26
27     private void Awake()
28     {
29         PlayerPrefs.SetInt("PointsToWin", pointsToWin);
30         Instance = this;
31         DontDestroyOnLoad(gameObject);
32     }
33
34     public void ActiveEndScene()
35     {
36         SceneManager.LoadScene("Perder");
37     }
38
39     public void checkGameOver(int _currentScore)
40     {
41         PlayerPrefs.SetInt("score", _currentScore);
42         if (_currentScore >= pointsToWin)
43         {
44             ActiveEndScene();
45         }
46     }
47 }
48 }
```

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using UnityEngine.SceneManagement;
using UnityEngine.VFX;

public class GameControl : MonoBehaviour
{
    static public GameControl Instance;
    public GameObject birdBehaviour;
    public UIController UIController;
    public int pointsToWin = 15;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    private void Awake()
    {
        PlayerPrefs.SetInt("PointsToWin", pointsToWin);
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }

    public void ActiveEndScene()
    {
        SceneManager.LoadScene("Perder");
    }

    public void checkGameOver(int _currentScore)
    {
        PlayerPrefs.SetInt("score", _currentScore);
        if (_currentScore >= pointsToWin)
        {
    }

```

```
        ActiveEndScene();
    }
}
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameInput : MonoBehaviour
{
    private PlayerInputActions playerInputActions;
    public UIController controller;

    private void Awake()
    {
        playerInputActions = new PlayerInputActions();
        playerInputActions.Player.Enable();
    }

    public Vector2 GetMovementVectorNormalized()
    {
        Vector2 inputVector =
playerInputActions.Player.Move.ReadValue<Vector2>();
        return inputVector.normalized;
    }

    public bool IsBraking()
    {
        return playerInputActions.Player.Brake.ReadValue<float>() > 0;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
```

```
public class InventoryUi : MonoBehaviour
{
    private TextMeshProUGUI CosechaText;

    // Start is called before the first frame update
    void Start()
    {
        CosechaText = GetComponent<TextMeshProUGUI>();
    }

    public void UpdateCosechaText(PlayerInventory playerInventory)
    {
        CosechaText.text = playerInventory.NumberOfCosechas.ToString();
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelMenu : MonoBehaviour
{
    public void OpenLevel(int levelid)
    {
        stringlevelname = "level " + levelid;
        SceneManager.LoadScenelevelname;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
```

```
{  
public void PlayGame()  
{  
    SceneManager.LoadSceneAsync(1);  
}  
}
```

```
using UnityEngine;  
  
public class Movement : MonoBehaviour  
{  
    [SerializeField] private float maxMoveSpeed = 7f;  
    [SerializeField] private float acceleration = 0.5f;  
    [SerializeField] private float deceleration = 1f;  
    [SerializeField] private float maxRotateSpeed = 200f;  
    [SerializeField] private GameInput gameInput;  
  
    private float currentSpeed = 0f;  
    private Rigidbody rb;  
    private float currentTurnSpeed = 0f;  
    private bool isReversing = false;  
  
    public UIController controller;  
  
    private void Awake()  
    {  
        rb = GetComponent<Rigidbody>();  
    }  
  
    void Update()  
    {  
        HandleMovementInput();  
    }  
  
    void HandleMovementInput()  
    {  
        Vector2 inputVector = gameInput.GetMovementVectorNormalized();  
        bool isBraking = gameInput.IsBraking();  
  
        // Update reversing state  
        isReversing = inputVector.y < 0 || controller.Reverse == 1;  
    }  
}
```

```

        // Handle acceleration, braking, and reversing
        if (isBraking || controller.Brake == 1)
        {
            currentSpeed = Mathf.MoveTowards(currentSpeed, 0,
deceleration * Time.deltaTime);
        }
        else if (isReversing)
        {
            currentSpeed = Mathf.MoveTowards(currentSpeed,
-maxMoveSpeed, acceleration * Time.deltaTime);
        }
        else if (inputVector.y > 0 || controller.Drive == 1)
        {
            currentSpeed = Mathf.MoveTowards(currentSpeed,
maxMoveSpeed, acceleration * Time.deltaTime);
        }
        else
        {
            currentSpeed = Mathf.MoveTowards(currentSpeed, 0,
acceleration * Time.deltaTime);
        }

        // Adjust turning speed based on horizontal input, keyboard,
and UART commands
        if (Input.GetKey(KeyCode.A) || controller.Left == 1)
        {
            currentTurnSpeed = -maxRotateSpeed;
        }
        else if (Input.GetKey(KeyCode.D) || controller.Right == 1)
        {
            currentTurnSpeed = maxRotateSpeed;
        }
        else
        {
            currentTurnSpeed = 0;
        }
    }

    void FixedUpdate()
    {
        // Move the object forward or backward based on 'currentSpeed'
        rb.MovePosition(rb.position + transform.forward * currentSpeed
* Time.fixedDeltaTime);
    }
}

```

```
// Rotate the object based on 'currentTurnSpeed'
if (currentTurnSpeed != 0)
{
    rb.MoveRotation(rb.rotation * Quaternion.Euler(0,
currentTurnSpeed * Time.fixedDeltaTime, 0));
}
}
```

```
using UnityEngine;

public class PlayerCollision : MonoBehaviour
{
    void OnCollisionEnter(Collision collisionInfo)
    {
        if (collisionInfo.collider.tag == "pared")
        {
            Debug.Log("we hit the tower");
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class PlayerInventory : MonoBehaviour
{
    public int NumberOfCosechas { get; private set; }

    public UnityEvent<PlayerInventory> OnCosechaCollected;
    public void CosechaCollected()
    {
        NumberOfCosechas++;
        OnCosechaCollected.Invoke(this);
    }
}
```

