Práctico 6 Vídeo juego de naves en micro:bit

Objetivo:

¡Juego de naves!

En este práctico codificaremos las placas micro:bit para simular un juego de naves, donde el jugador moverá su nave con los botones A y B y podrá dispara a sus enemigos para destruirlos.

Pasos del proyecto: NAVE

Paso 1 Crear los sprites

En primer lugar vamos a configurar los Sprites necesarios para el juego. Entre ellos se encuentran los siguientes:

Nave:

Representa la nave que el jugador controlará.

Enemigo:

Representan las naves enemigas que el jugador debe evitar o destruir.

Disparo:

Representan los disparos que la nave del jugador lanza.

Para esto debemos crear las tres variables (Nave, enemigos y disparo), luego nos dirigimos a la categoría "**Juego**" dentro de avanzado y ponemos los bloques "crear sprite …." indicando las coordenadas como se puede apreciar en la imagen.

Paso 2 Pausar y reanudar juego

Utilizaremos agitar la micro:bit para pausar el juego y al presionar el logotipo se reanuda el juego.

Paso 3 Configuración del movimiento de la nave a la izquierda y derecha:

Para esto utilizaremos el botón A y B de la micro:bit y dentro de la categoría **juego** pondremos el bloque "**Nave desplazar 1**" y "**Nave desplazar -1**" que permitirán mover la nave de un lugar (Una columna en el led a derecha o izquierda).

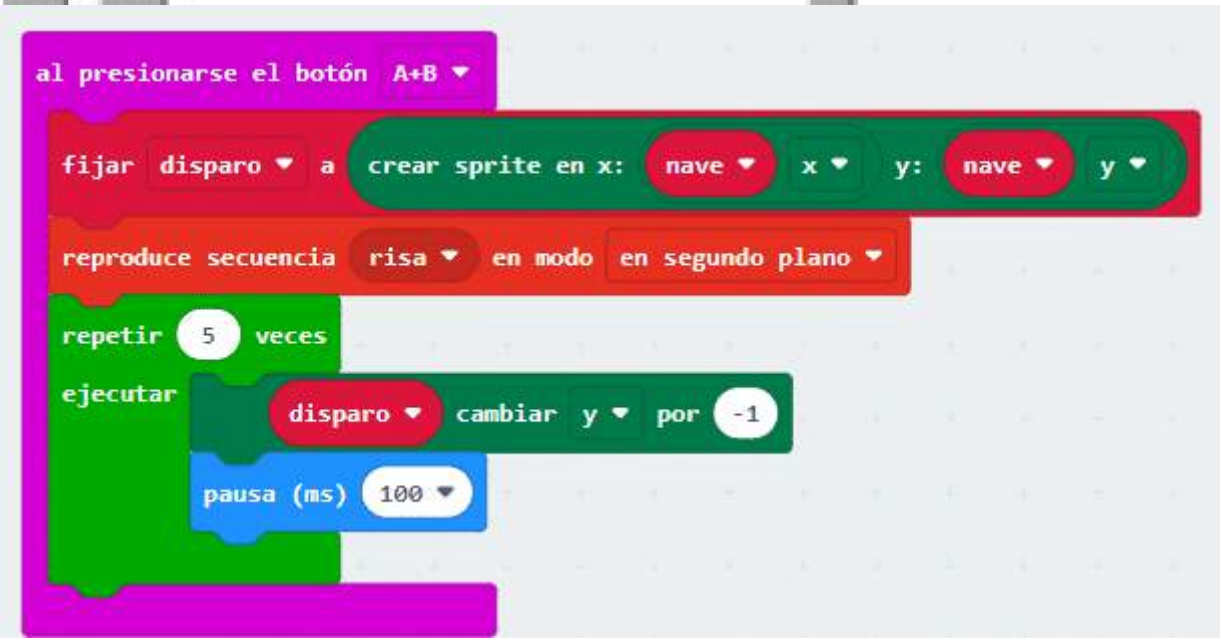
Estas funciones permiten que el jugador mueva la nave hacia la izquierda presionando el botón A y hacia la derecha con el botón B.

El bloque "desplazar"acepta un número que indica cuántas posiciones debe moverse la nave o cualquier sprite; -1 para izquierda y 1 para derecha.

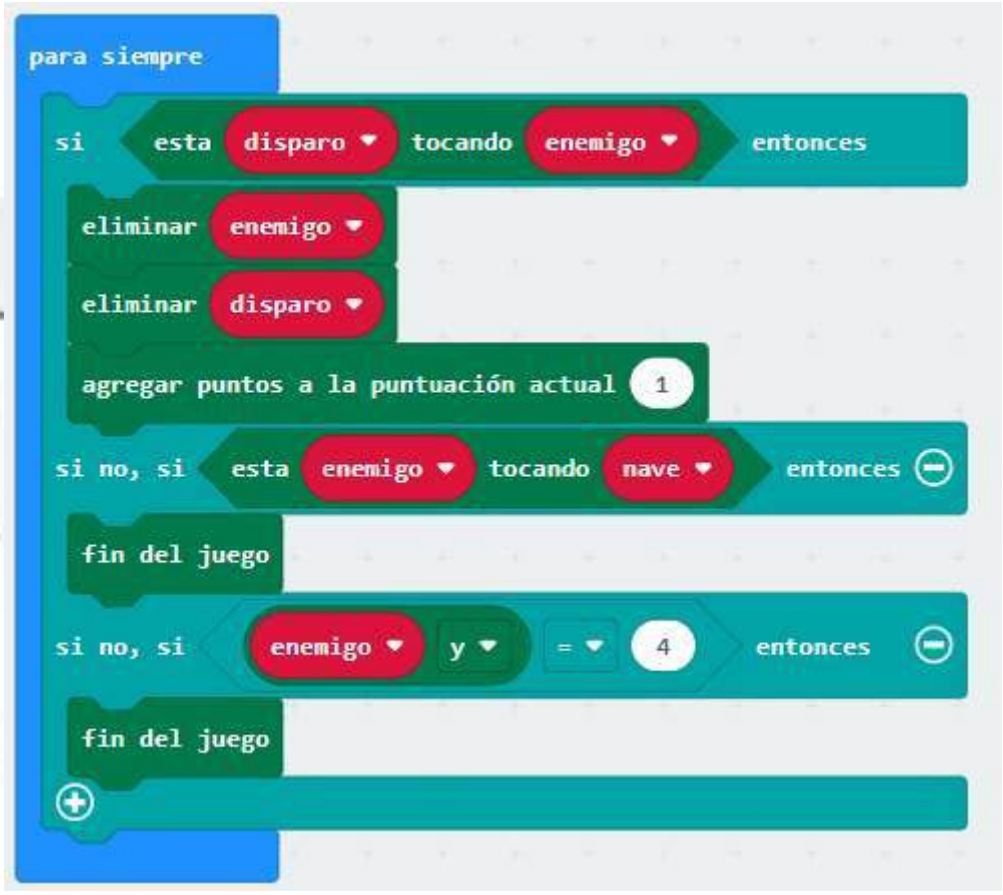
Paso 4 Disparar un proyectil

Utilizaremos el botón A+B para disparar nuestro proyectil. Al presionar ambos botones (A y B) en las primeras dos líneas de bloques, se crea un sprite para el disparo en la misma

posición X de la nave y se mueve hacia arriba (Y -1) repetidamente. También se reproduce un efecto de sonido como se puede ver en la imagen (Bloque de “MÚSICA”).



Paso 5 Manejar movimiento y reaparición del enemigo (Respawning).



En un bloque **Por SIEMPRE,** se estará evaluando si el disparo toca al enemigo, en caso afirmativo, tanto el proyectil como el enemigo desaparecerán y se agregará una puntuación a la que tengamos.

También evaluaremos si el enemigo choca contra la nave o llega al final, en este caso finalizará el juego.

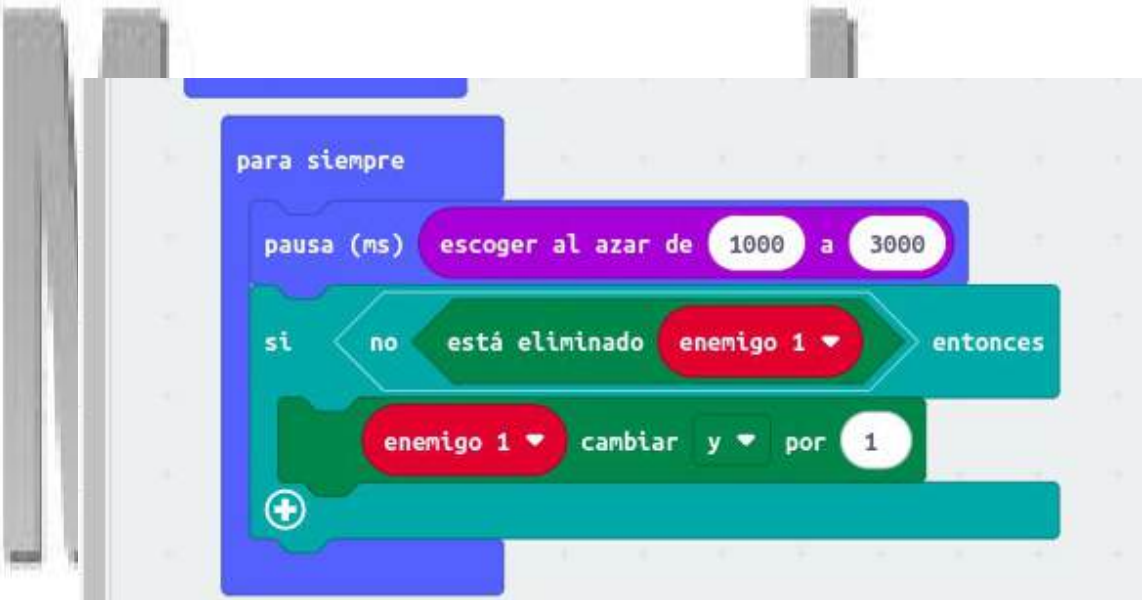
Paso 6 Aparecer enemigos cuando se lo destruye

En un bloque **Por SIEMPRE,** se estará evaluando si el enemigo es eliminado. De ser verdad, aparece un nuevo enemigo en una posición aleatoria en el eje X después de una pausa aleatoria. Como se puede apreciar la posición horizontal es aleatoria pero la posición vertical es en la parte superior “0” en el led.

Paso 7 Eliminar disparo al alcanzar el borde superior

En otro bloque “**PARA SIEMPRE”**, estaremos evaluando si el disparo llega al borde superior, o sea, su posición en el eje de las Y es =0, se elimina el sprite “disparo”.

Paso 8 Movimiento de enemigo



El algoritmo mostrado anteriormente, utiliza otro bloque PARA SIEMPRE, dentro hacemos una pausa al azar entre 1 y 3 segundos y luego evaluamos si no esta eliminado el enemigo, que se mueva en el eje Y (Hacia abajo) una posición en el led.

¿Por qué múltiples “Para Siempre"?

Al separar diferentes funcionalidades en múltiples bloques Para Siempre, cada ciclo puede operar independientemente. Por ejemplo, uno puede manejar la animación de sprites, otro la lógica de colisión, y otro puede controlar la entrada del usuario, sin que uno interfiera directamente con otro.

Esto evita que el programa se quede esperando que una tarea termine para empezar otra, permitiendo que todas las tareas importantes se ejecuten en tiempo real y de manera fluida.

En programación, un "hilo" es una **secuencia de instrucciones** de un programa que puede ser ejecutada de manera independiente de otras secuencias.

Los hilos son utilizados para realizar múltiples operaciones dentro de un solo proceso, permitiendo una ejecución más eficiente y mejor organizada en entornos multitarea. Cada Para Siempre() se puede considerar como un "hilo" que realiza una tarea específica, como mover enemigos, verificar colisiones, o responder a entradas del usuario.

Video construcción del juego:
https://www.youtube.com/watch?v=iVubYJ8isAE

Actividad

Tu tarea es refactorizar y mejorar el código del juego de naves, aplicando principios de modularidad con el empleo de funciones. **El objetivo es hacer el código más legible, mantenible y eficiente.**

Deberás modificar el código atendiendo las consignas siguientes:

1) Crear **Funciones** para Acciones Específicas:

•Define una función **disparar()** para manejar la creación del sprite de disparo y su animación.

•Define una función **aparecer_enemigo()** para gestionar la creación de enemigos de manera aleatoria.

•Define una función **ocultar_sprite()** que acepte un sprite y un número como parámetros, y elimine el sprite si se encuentra en la posición especificada.

2) Mejora la jugabilidad

Definición de Fin de Juego cuando el Jugador Gana:

•Implementa una condición de victoria para el jugador, como alcanzar una puntuación específica (por ejemplo, 5 puntos y decir Ganaste).

Manejo de Puntuación en Rangos:

•Define rangos de puntuación que, al ser alcanzados, incrementan la dificultad del juego. Ejemplo de rangos: 0-2 puntos y 3-5 puntos.

•A medida que el jugador alcanza el segundo rango de puntuación, incrementa la velocidad de los enemigos.

Ejemplo: hasta 2 velocidad de 2 a 5 segundos, mayor a 2 velocidad de 1 a 2 segundos.