**Objetivo**

Realizar una historia de algún(os) personaje(es), recorrido o juego que contenga interacciones con botones, recorridos(cambio de posiciones de un objeto).Utilizando la herramienta Unity.

**Desarrollo**

**Descripción de la elección:**

Deseo realizar un juego con los personajes e imágenes de “Hora de Aventura”, donde el personaje Jake, debe de atinar a que objeto de los 3 restantes debe de moverse.

Logrando que las 3 veces atine al marcador, para “GANAR”, si falla regresa a la posición original y comienza el Juego.

**Realización en Unity**

**Elección de Marcadores**

**Marcado 1 Marcador 2**

****

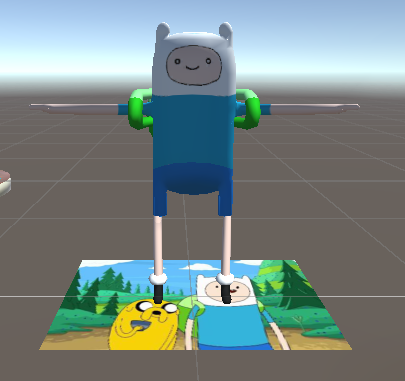
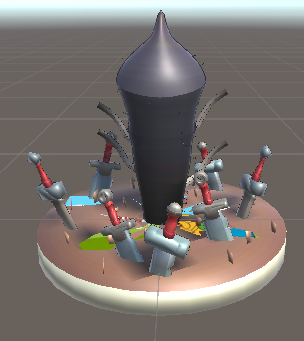
**Marcador 3 Marcador 4**

****

**Objeto 1 Objeto 2**

**** ****

**Objeto 3 Objeto 4**

**Código:**

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

using Vuforia;

using UnityEngine.UI;

public class Move : MonoBehaviour

{

public GameObject model; // El modelo a mover

public ObserverBehaviour[] ImageTargets; // Los 3 marcadores

public Button[] buttons; // Botones para seleccionar marcadores

public Text resultText; // Texto para mostrar resultados

private int currentTargetIndex = -1; // Índice del marcador actual

private Vector3 startPosition;

private bool isMoving = false; // Controlar si el modelo se está moviendo

private Vector3 originPosition; // Posición fija del marcador 1

void Start()

{

originPosition = ImageTargets[0].transform.position; // Asignar la posición del marcador 1 al inicio

model.transform.position = originPosition; // Colocar el modelo inicialmente en la posición del marcador 1

startPosition = model.transform.position;

resultText.text = "Aventure Time";

resultText.gameObject.SetActive(true); // Ocultar texto al inicio

// Asignar la función de clic a cada botón

for (int i = 0; i < buttons.Length; i++)

{

int index = i; // Necesario para evitar el problema de cierre

buttons[i].onClick.AddListener(() => OnButtonClick(index));

}

// Seleccionar un objetivo inicial

SelectRandomTarget();

}

private void SelectRandomTarget()

{

// Seleccionar un marcador aleatorio (0, 1 o 2) que aún no haya sido usado

List<int> availableTargets = new List<int>();

for (int i = 0; i < buttons.Length ; i++)

{

if (buttons[i].interactable) // Solo botones habilitados

availableTargets.Add(i);

}

if (availableTargets.Count > 0)

{

currentTargetIndex = availableTargets[Random.Range(0, availableTargets.Count)];

Debug.Log("Marcador objetivo aleatorio: " + currentTargetIndex);

}

else

{

currentTargetIndex = -1; // No hay marcadores disponibles

}

}

public void OnButtonClick(int buttonIndex)

{

if (!isMoving)

{

if (buttonIndex == currentTargetIndex) // Si seleccionas el marcador correcto

{

resultText.text = "¡Éxito!"; // Mensaje de éxito

resultText.gameObject.SetActive(true); // Mostrar texto

StartCoroutine(MoveToPosition(ImageTargets[buttonIndex + 1].transform.position)); // Moverse al marcador correcto

buttons[buttonIndex].interactable = false; // Desactivar el botón del marcador acertado

// Generar un nuevo marcador aleatorio

SelectRandomTarget();

// Comprobar si se alcanzó el último marcador

if (AreAllTargetsDisabled())

{

resultText.text = "¡GANASTE!"; // Mensaje de victoria

StartCoroutine(RestartGame()); // Reiniciar el juego

}

}

else // Si no es correcto

{

// Si está en la posición de origen, no mostrar mensaje

if (model.transform.position != originPosition)

{

resultText.text = "Regresando al origen.";

resultText.gameObject.SetActive(true);

StartCoroutine(RestartGame());// Regresar al origen

}

else

{

resultText.gameObject.SetActive(false); // No mostrar mensaje

}

}

}

}

private IEnumerator MoveToPosition(Vector3 endPosition)

{

isMoving = true;

Vector3 startPosition = model.transform.position;

float journey = 0.0f;

while (journey <= 1.0f)

{

journey += Time.deltaTime; // Modificar según la velocidad deseada

model.transform.position = Vector3.Lerp(startPosition, endPosition, journey);

yield return null;

}

yield return new WaitForSeconds(1.0f); // Esperar 2 segundos para mostrar el mensaje

resultText.gameObject.SetActive(false); // Ocultar el texto

isMoving = false;

// Reactivar todos los botones después de regresar

if (endPosition == startPosition)

{

foreach (Button button in buttons)

{

button.interactable = true; // Habilitar todos los botones nuevamente

}

}

}

private IEnumerator RestartGame()

{

yield return new WaitForSeconds(1.0f); // Esperar 2 segundos antes de reiniciar

resultText.gameObject.SetActive(false); // Ocultar mensaje anterior

// Mover el modelo de regreso al marcador 1

yield return MoveToPosition(ImageTargets[0].transform.position);

// Reactivar todos los botones para comenzar de nuevo

foreach (Button button in buttons)

{

button.interactable = true; // Habilitar todos los botones nuevamente

}

// Seleccionar un nuevo marcador aleatorio

SelectRandomTarget();

}

private bool AreAllTargetsDisabled()

{

// Comprobar si todos los botones están desactivados

foreach (Button button in buttons)

{

if (button.interactable)

{

return false; // Hay al menos un botón habilitado

}

}

// Si todos los botones están desactivados, reiniciar el juego

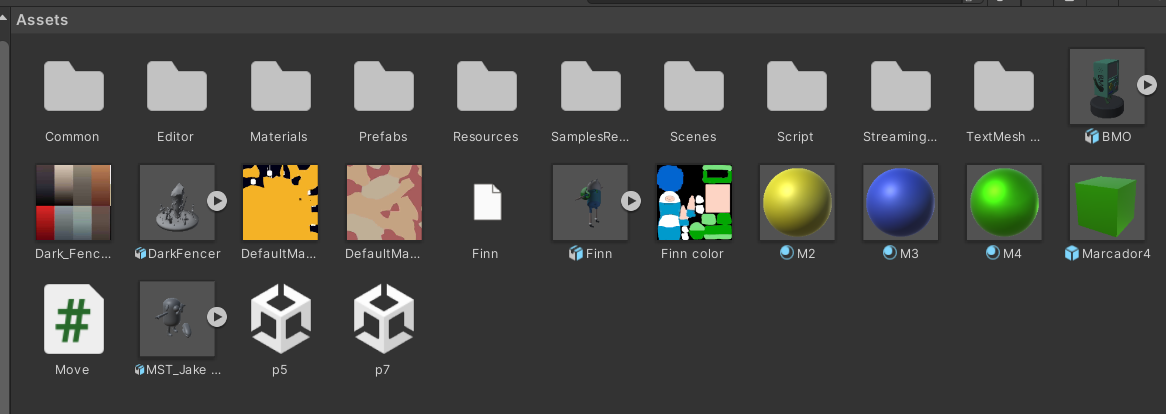
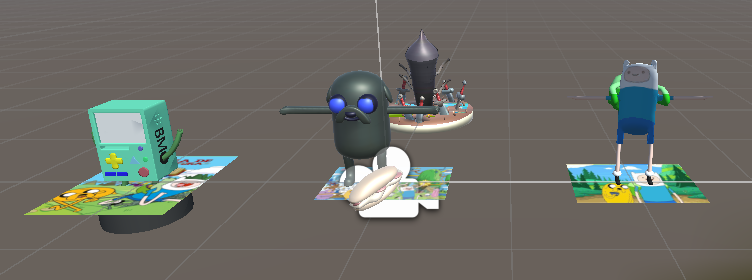
StartCoroutine(RestartGame());

return true; // Todos los botones están desactivados

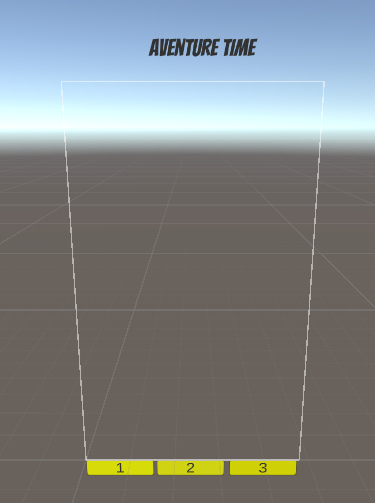
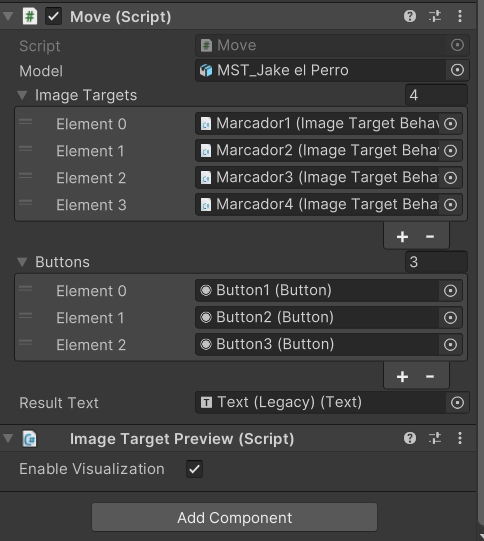
}

}

**Assets**



**Configuración Marcador 1 Canvas Botón y Texto**

****

**Funcionalidad**

**Comienza el Juego**



**Éxito 1**

Es decir, el jugador acertó a que objeto debe moverse.



**Éxito 2 Éxito 3**

Vuelve atínar Atinó 3 veces el jugador, por lo tanto, GANÓ

Como observamos los botones, cada que acertamos se dirige a la posición del objeto y se deshabilita el objeto, esto sucede para que solamente puedas pasar 1 veces por el mismo objeto y tenga fin el juego.

**Regresa al estado original Activa de nuevo los botones**

**Caso Éxito- Regreso**

Consiste cuando el jugador atina y después falla y tiene que regresar al origen para comenzar de nuevo el juego por que falló.

**Éxito Regreso**

**Comienza el juego de Nuevo**

