UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Manual Técnico

Nombres: Herrera Argumedo Luis Diego

Materia: Temas Selectos de Ingeniería en Computación III - Realidad Virtual y Aumentada

Grupo: 2

Profesor: Ing. Arturo Pérez De La Cruz

Semestre: 2025-2

Fecha de Entrega: 18 de mayo de 2025

Índice

Descripción Técnica

Futigol es una aplicación de realidad aumentada desarrollada en Unity con el motor Vuforia. Su objetivo es simular una experiencia de juego en la que el usuario asume el rol de portero y debe atajar penales virtuales que se dirigen hacia la cámara del dispositivo móvil.

La aplicación utiliza un marcador físico (ImageTarget) como punto de referencia para ubicar el lanzador de pelotas. Cada pelota se instancia como un objeto físico con Rigidbody y se lanza con una trayectoria dinámica hacia la ARCamera. El sistema detecta si el disparo fue atajado o terminó en gol mediante colisiones con colliders específicos. La interfaz gráfica (Canvas) muestra mensajes en pantalla y un contador de atajadas, además de botones para lanzar pelotas y reiniciar la escena.

Lenguaje: C#
Plataforma: Android (APK)

Tecnologías Utilizadas

1. Unity (versión 2022.3.48f1)
 — Entorno principal de desarrollo. Se utilizó para crear y organizar la escena, los objetos 3D, las físicas, los scripts en C# y la interfaz gráfica.

2. Vuforia Engine
 — Motor de realidad aumentada integrado en Unity. Permite el reconocimiento del marcador (ImageTarget) en el entorno real, utilizado como punto de referencia para colocar objetos virtuales.

3. C# (lenguaje de programación en Unity)
 — Se empleó para desarrollar la lógica del juego, como la generación y disparo de pelotas, detección de colisiones, control de eventos y reinicio de la escena.

4. TextMeshPro
 — Herramienta de Unity para mostrar texto de alta calidad en pantalla. Se utilizó para los mensajes (“¡Gol!”, “¡Atajada!”) y el contador de atajadas.

5. Sistema de físicas de Unity (Rigidbody, Collider, Trigger)
 — Se aplicaron para manejar el comportamiento físico de las pelotas, las colisiones con el portero y la detección de goles o atajadas.

6. Canvas (UI de Unity)
 — Sistema de interfaz gráfica donde se colocaron los botones de “Disparar”, “Reiniciar” y los textos informativos.

7. Build Settings para Android
— Utilizado para compilar y exportar la aplicación en formato APK para su instalación en dispositivos móviles Android.

Cronograma de Actividades

FASE	ACTIVIDAD	FECHA DE FINALIZACIÓN
Investigación y Planificación	Análisis de tecnologías y recursos necesarios para el proyecto	8 de abril
Diseño y Conceptualización	Diseño de la interfaz del juego, creación del concepto de la mascota	17 de abril
Desarrollo Inicial	Programación de la escena básica y los objetos virtuales	24 de abril
Animación y VFX	Creación de animaciones para la mascota y efectos visuales	30 de abril
Integración de Contenido	Integración de los modelos 3D, efectos y programación de interacciones	11 de mayo
Pruebas y Ajustes	Pruebas de jugabilidad, ajuste de errores y optimización del rendimiento	1 de mayo
Entrega Final	Preparación para la presentación y distribución del proyecto	18 de mayo

Estructura del Proyecto

- **Assets**
- **Modelos**
    - Aquí se encuentran los modelos usados con sus respectivos materiales y texturas.
- **Scripts**
    - Aquí se encuentran todos los scripts utilizados.

- **Scenes**
    - Aquí se encuentran las escenas creadas para el proyecto.

Jerarquía de Objetos

- **Directional Light**
- **Global Volume** - Iluminación general de la escena y efectos gráficos.
- **ARCamera** - Cámara principal con capacidades de realidad aumentada (Vuforia).
    - **Collider** – Área de colisión para detectar pelotas "atajadas" por el jugador.
    - **Gol** – Zona trasera para detectar goles (si la pelota la atraviesa).
    - **tripo_convert...** – Modelo 3D de guantes de portero anclados a la cámara, simulan manos reales.
- **ImageTarget** – Marcador AR que da origen al contenido digital.
    - spawnPoint – Punto donde aparece (spawnea) la pelota al presionar el botón
- **EventSystem** – Sistema de eventos necesario para la interacción con la interfaz UI.
- **Canvas** – Contenedor general de la interfaz de usuario.
    - **Boton** – Agrupa todos los elementos de la UI interactiva.
        - **Text (TMP)** – Texto visible del botón de disparo ("RECIBIR TIRO").
    - **GameMessage** – Texto dinámico que muestra “¡Gol!” o “¡Atajada!”.
    - **SaveCounterText** – Texto que muestra cuántas atajadas ha realizado el jugador.
    - **Reset** – Botón que reinicia la escena para jugar otra vez.
- **UIManager** – Objeto con el script GameManager.cs que controla la UI y el contador.
- **SceneReset** – Objeto con el script ResetScene.cs que reinicia toda la escena.

Variables Principales

GameManager.cs

- GameMessage (TextMeshProUGUI) → Muestra “¡Gol!” o “¡Atajada!” en pantalla.
- SaveCounterText (TextMeshProUGUI) → Muestra el número total de atajadas acumuladas.
- saveCounter (int) → Contador que se incrementa cada vez que el jugador ataja una pelota.

BallShooter.cs

- ballPrefab (GameObject) → Prefab de la pelota que se instancia al disparar.
- spawnPoint (Transform) → Punto de origen desde donde se lanza la pelota.

- force (float) → Intensidad con la que se aplica la fuerza hacia la cámara.
- spread (float) → Rango de desviación aleatoria para que los disparos no sean perfectamente rectos.
- cameraTransform (Transform) → Referencia a la ARCamera; sirve como dirección objetivo del disparo.

GoalkeeperCollider.cs
- gameManager (GameManager) → Referencia al GameManager para actualizar marcador y mensajes.
- goalMessageShown (bool) → Indica si ya se mostró el mensaje de gol para evitar repeticiones.
- goalTextDuration (float) → Tiempo que el mensaje se muestra antes de desaparecer.

TargetTracker.cs
- isTargetVisible (bool) → Referencia para verificar si el ImageTarget es visible.
- uiBotton (GameObject) → Referencia para que aparezca el botón según el estado de isTargetVisible.

ResetScene.cs
- Ninguna variable pública. El script usa SceneManager.LoadScene() para reiniciar la escena.

Procedimiento Técnico

Inicio y reconocimiento del marcador

- Al ejecutar la aplicación, la ARCamera inicia el reconocimiento de entorno mediante Vuforia.
- Cuando se detecta el ImageTarget (marcador impreso), Unity posiciona en su superficie el objeto SpawnPoint, que servirá como punto de disparo de las pelotas, al igual que al botón para realizar el tiro.

Interfaz de usuario

- La interfaz gráfica (Canvas) muestra dos botones principales: “Disparar” y “Reiniciar”.
- También se muestran los textos informativos: el mensaje de estado ("¡Gol!" / "¡Atajada!") y el contador de atajadas.

Disparo de la pelota

- Al presionar el botón “Recibir Tiro”, el script BallShooter instancia una pelota desde el prefab.
- La pelota es posicionada en el SpawnPoint y se le aplica una fuerza (AddForce) hacia la dirección de la ARCamera.

- Se introduce un pequeño componente aleatorio para que la trayectoria no sea perfectamente recta.

Detección de colisión

- La ARCamera tiene un collider (trigger) que simula la “zona de portero”.
- Si la pelota colisiona con esta zona, el script GoalkeeperCollider lo detecta como una atajada.
- Si no colisiona con esta zona y pasa fuera del campo de visión, se considera un gol.

Resultados

- El GameManager actualiza el mensaje ("¡Atajada!" o "¡Gol!") y modifica el color del texto.
- Si es una atajada, se incrementa el contador y se actualiza el texto del marcador.
- En ambos casos, la pelota se destruye inmediatamente después de la detección.

Reinicio del juego

- Al presionar el botón “Reiniciar”, el script ResetScene recarga la escena actual usando SceneManager.LoadScene().
- Esto reinicia los contadores, mensajes y limpia cualquier pelota en escena.

Interfaz de Usuario

La interfaz gráfica de usuario fue diseñada para ser clara, funcional y compatible con dispositivos móviles. Está implementada mediante un Canvas en Unity.

Componentes principales del Canvas:

1. Canvas
Tipo: Screen Space.

2. Botón: Recibir Tiro

- Función: Lanza una pelota desde el marcador hacia la cámara.
- Script asociado: BallShooter.cs → Llama al método ShootBall().
- Diseño: Botón visible en pantalla con texto "Recibir Tiro".
- Interacción: Se activa con un toque en pantalla.

3. Botón: Reiniciar (ResetButton)

- Función: Reinicia la escena actual, borrando pelotas y reseteando el contador.
- Script asociado: ResetScene.cs → Llama a SceneManager.LoadScene().
- Diseño: Botón con texto "Reiniciar" o ícono de recarga.

4. Texto: Mensaje de resultado (MessageText)

- Función: Muestra el resultado de cada intento ("¡Gol!" o "¡Atajada!").
- Script asociado: GameManager.cs
- Estilo: Texto grande con color dinámico (rojo para gol, verde para atajada).
- Temporización: Se oculta después de unos segundos.

5. Texto: Contador de atajadas (CounterText)

- Función: Muestra la cantidad acumulada de atajadas exitosas.
- Script asociado: GameManager.cs
- Actualización: Incrementa automáticamente tras cada atajada.