Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Aplicación de apoyo para el tratamiento de la dislalia

Support application for the treatment of dyslalia

Adán de la Rosa Lugo

La Laguna, 13 de septiembre de 2022

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43826207Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas, como tutor

D. **Vanesa Muñoz Cruz**, con N.I.F. 78698687R profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Aplicación de apoyo para el tratamiento de la dislalia*”

ha sido realizada bajo su dirección por D. **Adán de la Rosa Lugo**, con N.I.F. 43832256Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de septiembre de 2022

Agradecimientos

En primer lugar quiero agradecer a mi tutor Jesús Miguel Torres Jorge por su apoyo durante la realización de este proyecto.

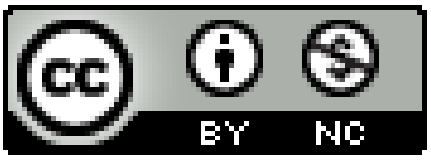
En segundo lugar, agradecer a la profesora Isabel Sánchez Berriel por ayudar en la obtención de las herramientas necesarias para el desarrollo de los juegos.

También quiero agradecer a mis compañeros y amigos del grado por brindarme ánimos en todo momento.

Por último, agradecer a mis padres, ya que sin su sacrificio no hubiese podido estar donde estoy a día de hoy.

Muchas gracias a todos.

Licencia



Resumen

El principal objetivo de este proyecto es desarrollar dos juegos con realidad aumentada que permitan usarse para tratar problemas de dislalia, un trastorno del lenguaje caracterizado por la dificultad de producir ciertos sonidos. Así como, la creación de una aplicación frontend que permita a profesionales de la logopedia realizar un seguimiento de sus pacientes.

El desarrollo del proyecto se ha realizado mediante el motor de videojuegos Unity y la librería de Apple de realidad aumentada ARKit para los juegos. Además, la aplicación SPA se ha desarrollado con React, haciendo uso de storybook, testing y CI.

Palabras clave: Dislalia, Logopedia, Unity, Realidad Aumentada, ARKit, Frontend, React, Storybook.

Abstract

The main goal of this proyect is to create two games developed with Augmented Reality that can be used to treat dyslalia problems, a language disorder characterized by difficulty of being able to produce certain sounds. As well as, the creation of a frontend application that could be use by profesionals of speech therapy to follow up on their patients.

The development of this proyect has been carried out using Unity engine and the Apple AR library called ARKit. In addition, the SPA app has been developed with React, using storybook, testing and CI.

Keywords: Dyslalia,speech therapy, Unity, Augmented Reality, ARKit, Frontend, React, Storybook.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Justificación

El objetivo del proyecto es crear una aplicación para el tratamiento de niños con dislalia. En específico, diseñar juegos con realidad aumentada que les permitan realizar ejercicios de movilidad o pronunciación con el objetivo de que el niño aprenda a articular los sonidos correctamente. Además, desarrollar una interfaz que facilite al logopeda o pediatra el seguimiento de sus pacientes.

La dislalia infantil es un trastorno del lenguaje que se caracteriza por una dificultad para producir ciertos sonidos o grupos de sonidos de la manera que se considera adecuada. Entre los fonemas que se omiten, sustituyen o deforman con mayor frecuencia son: r, s, z, l, k y ch, siendo estos el tipo de dislalia funcional más común. Aunque se diagnostica con facilidad y no representa un riesgo para la salud, es conveniente corregirla lo antes posible para evitar problemas de conducta, comportamiento, inseguridad, baja autoestima, y otras dificultades que pueden alterar su aprendizaje escolar.

Por tanto, es conveniente que el paciente siga un tratamiento orientado y especializado. Actualmente, existen diferentes tratamientos a aplicar por el especialista, con la ayuda de juegos y programas basados en la estimulación de la capacidad del niño para producir sonidos, realizar movimientos y posturas o hacer ejercicios labiales y linguales, etc.

1.2. Antecedentes y estado del arte

La Realidad Aumentada agrupa a aquellas tecnologías que permiten la superposición, en tiempo real, de imágenes, marcadores o información generados de forma virtual, sobre el mundo físico. Por ejemplo, haciendo uso de dispositivos como pueden ser teléfonos móviles o tabletas que permitan generar experiencias aportando conocimiento relevante sobre nuestro entorno.

Las ventajas principales del uso de la realidad aumentada son diversas. Por un lado, contribuye al crecimiento del sector tecnológico. Tanto complementando otras áreas tecnológicas como la IA o la robótica, como siendo aplicado en otros sectores como en el de la medicina. En este último, ha servido en la proyección de órganos y toma de decisiones para el diagnóstico de los pacientes.

Un ejemplo de esto, es que se han publicado diferentes estudios analizando la mejora de la terapia de rehabilitación de articulaciones mediante el uso de la RA.

Por otro lado, en conjunto con el sector de los videojuegos puede servir de entretenimiento. En concreto, existen juegos educativos que combinan tarjetas didácticas con la realidad aumentada que consiguen mantener la motivación y la atención de los niños.

1.3. Objetivos

La idea de este proyecto surgió del trabajo realizado por los alumnos de la asignatura de Videojuegos Accesibles de la Maestría en Arte y Diseño de Videojuegos de la UDEM. Los cuales detectaron la problemática inicial y desarrollaron una serie de mockups de una posible aplicación y realizaron diferentes test con usuarios.

Se realizó una primera reunión con los integrantes y se puso en contexto tanto la problemática a resolver como la solución que se planteaba. Además, se proporcionaron las imágenes con los diferentes prototipos que se desarrollaron en su trabajo e ideas para los juegos.

Una vez conocidos los detalles del proyecto, se procedió a realizar una planificación de los objetivos. Por tanto, se definieron las siguientes tareas a realizar:

- Diseño y prototipo de los juegos con AR.
- Diseño de la aplicación pediatra/paciente.
- Testing y UX.
- Aplicar mejoras en base al test de usuario.

Los juegos implicarán diferentes movimientos de la boca. Para el primer juego, el usuario tendrá que realizar movimientos con la lengua en diferentes direcciones y atrapar los objetos que se desplazan por la pantalla. Mientras que para el segundo juego, el usuario deberá realizar el movimiento de aspiración colocando los labios en forma de "embudo". Además, la aplicación frontend servirá para la gestión de pacientes por parte de los logopedas.

Capítulo 2

Tecnologias y herramientas utilizadas

Inicialmente se realizó un estudio de las principales tecnologías disponibles de realidad aumentada y detección de expresiones faciales para el proyecto. En un principio se planteó el kit de desarrollo ARCore, sin embargo, carecía de la detección de la musculatura lingual. También se observó la plataforma Spark AR y se llegó a implementar un filtro que implicaba la detección de la lengua, pero se descartó esta opción por estar más enfocada a filtros en redes sociales. Finalmente, se llegó a la conclusión de que la mejor opción era la plataforma de ARKit de Apple, ya que permitía realizar el seguimiento de la lengua y podía utilizarse junto con el motor de videojuegos de Unity.

2.1. Herramientas de planificación

2.1.1. Git y GitHub

Para la realización de este proyecto se ha hecho uso de control de versiones mediante *Git*[18] y repositorios remotos en *GitHub*[19]. Concretamente dos repositorios, uno para los juegos en RA y otro para el frontend de la aplicación. Todos ellos se encuentran públicos para su consulta en el siguiente enlace `https://github.com/AdanRL/dizkids`. Se ha utilizado la herramienta de kanban automático de GitHub para la planificación de las tareas a realizar `https://github.com/users/AdanRL/projects/3`.

2.1.2. Notion

Notion[20] es un software para la planificación de proyectos y toma de notas. En el, se han definido los diferentes sprints y los objetivos del desarrollo de la aplicación y los juegos.

2.2. Herramientas de desarrollo

2.2.1. Juegos de RA

Unity

Un motor de videojuegos es un software que ofrece un conjunto de herramientas de desarrollo que permiten crear juegos de manera más sencilla y rápida. El motor *Unity*[28]

ofrece todas estas prestaciones en múltiples sistemas operativos como Windows, Mac OS o Linux.

Las principales razones por las que se ha escogido esta herramienta, es porque su curva de aprendizaje es menor, ya que ofrece una buena documentación y cuenta con una gran comunidad de desarrolladores. Además, Unity integra la herramienta de *AR Foundation*[1], un framework que permite trabajar con las tecnologías de *ARCore*[2] y *ARKit*[3]. Otra de las ventajas es que ofrece servicios como *Unity Cloud Build*[29] que permite compilar y generar builds de manera automática.

Unity Cloud Build

Como el desarrollo del proyecto se realiza con una librería específica para dispositivos iOS, Unity ofrece esta herramienta para automatizar el proceso de compilado y subirlo a la nube. La configuración se realiza en minutos y está enlazada con el repositorio remoto de GitHub, de manera que al realizar un push sobre el repositorio, este se ejecuta automáticamente.

C#

C#[5] es un lenguaje de programación multiparadigma cuya sintaxis desciende de C/C++ pero agregando nuevas funcionalidades de otros lenguajes como Java. Unity permite realizar scripts en este lenguaje para crear objetos, definir interacciones y mecánicas.

ARKit

ARKit es la plataforma de realidad aumentada de Apple para dispositivos iOS. Las principales características de ARKit son:

- Capacidad de detectar hasta tres caras simultáneamente.
- Capacidad de detectar objetos.
- Capacidad de detectar superficies.
- Uso de blendshapes para detectar zonas de la cara especificas.

El principal motivo por el que se ha elegido esta tecnología, es la gran cantidad de documentación relacionada con el reconocimiento facial y reconocimiento de gestos de la cara mediante el uso de *blendShapes coefficients*[4]. Para este caso, son interesantes los que registran la zona de la boca, como pueden ser: *Jaw Forward*[6], *Mouth Left*[8], *Mouth Right*[9], *Jaw Open*[7].

2.2.2. Frontend de la aplicación

React

React[22] es una libreria de Javascript para el desarrollo de aplicaciones móviles y web. Hace uso de fragmentos reutilizables llamados componentes para la construcción de interfaces de usuario. Además, desde las últimas versiones, se ha implementado la programación funcional mediante hooks, lo cual permite una mayor escalabilidad en las aplicaciones.

Se ha elegido esta librería principalmente por el rendimiento que ofrece en las aplicaciones y la velocidad de desarrollo.

React Redux

Redux[24] es una librería de Javascript de código abierto para el manejo de estados de las aplicaciones. Es decir, facilita la comunicación entre componentes y es el principal motivo por el que se ha utilizado.

En el proyecto se ha definido un directorio *Store*[15] con la lógica necesaria para utilizar la librería.

React Router

La librería de *React Router*[23] permite gestionar las rutas, permitiendo así la navegación entre las diferentes partes de la aplicación.

Las rutas de la aplicación se han definido en el fichero de *App.tsx*[14]

Eslint y Prettier

Se ha utilizado el linter de *Eslint*[17] y el formateador de código *Prettier*[21]. Estas herramientas permiten que el código tenga el mismo formato en todos sus ficheros y son una gran herramienta a la hora de aplicar clean code.

Se ha decidido usar estas herramientas para mantener una consistencia en el código y facilitar su lectura a quienes quieran leer el código. Las reglas definidas se encuentran en los ficheros de *package.json*[13] y *.prettierrc*[16]

Storybook

Storybook[26] es una librería de código abierto que permite desarrollar y documentar los componentes aislados sin tener en cuenta las dependencias con otros componentes.

EL principal motivo de utilizar esta herramienta es que permite comprender y reutilizar los componentes de la aplicación. En el proyecto, cada componente tiene definido un fichero *.stories.tsx*[10]

Testing Library

Testing library[27] es una librería de testing para interfaces de usuario. Promueve las buenas prácticas y orienta el testing en el enfoque del usuario.

SmartMock.io

Como no se ha desarrollado ninguna API, se ha utilizado la herramienta *SmartMock*[25] que permite mockear llamadas para simular los diferentes endpoints de la aplicación.

Figma

Se utilizó la plataforma de *Figma*[11] para realizar el prototipado web. Se eligió esta herramienta por ser gratuita y bastante intuitiva de utilizar, además de que no requería ningún tipo de instalación para su uso.

Capítulo 3

Desarrollo de los juegos

En este capítulo se describirá el proceso que se ha realizado desarrollando los juegos y los resultados obtenidos.

3.1. Manejo de la librería ARKit

Como no se encontró la forma de poder depurar los coeficientes de los blendShapes desde el editor de Unity, se realizó un pequeño proyecto que sirvió como aproximación con la librería y así conseguir visualizar en pantalla los valores y ajustarlos para detectar los movimientos de la lengua.

Figura 3.1: Detectando coeficientes

Figura 3.2: Detectando coeficientes.

3.2. Desarrollo del primer juego AR

3.2.1. Prototipado

Uno de los primeros pasos en el desarrollo de una aplicación es realizar un prototipado que te permita tener una idea clara de como estructurarla y que interfaz va a tener. La idea es realizar un boceto de las pantallas que tendrán los juegos en AR. Para el primer juego, se realizó un prototipado en figma.

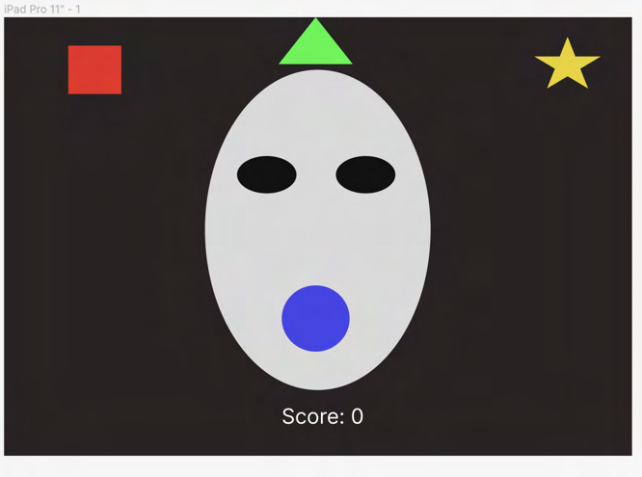


Figura 3.3: Prototipo del primer juego.

De esta manera, queda definida con un boceto la estructura del primer juego, donde tres objetos se desplazarán desde la parte superior de la pantalla hacia la parte inferior y el jugador deberá usar la lengua para atraparlos. Para controlar la colisión con los objetos, se colocará una esfera transparente en la zona de la boca, si el usuario tiene la lengua en una posición determinada y el objeto esta en contacto con la esfera, contará como un punto.

3.2.2. Asset Store

Para poder agregar algún modelado al proyecto, se ha utilizado los assets gratuitos de la store que ofrece Unity. En concreto, se utilizaron *assets*[12] de comida para ambientar el juego y darle más sentido que atrapar objetos sin modelado.

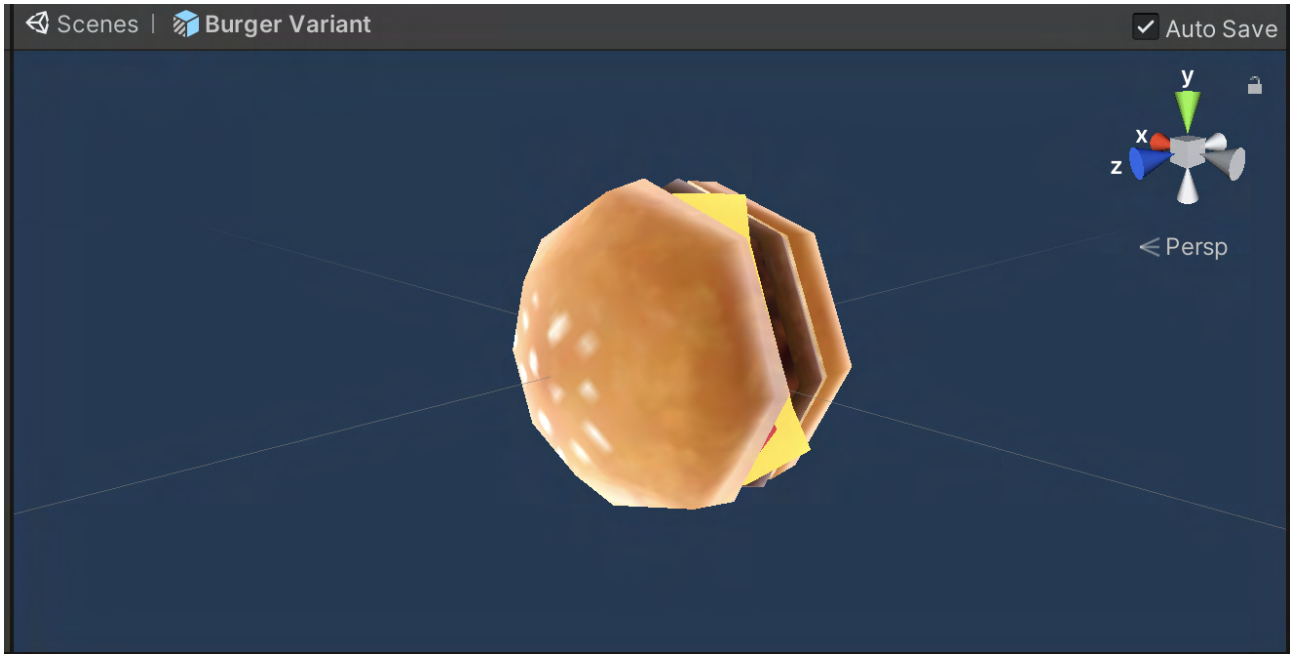


Figura 3.4: Modelo de hamburguesa.

3.2.3. Scaffolding del proyecto

En Unity, los proyectos se estructuran dentro del directorio de Assets. Para este proyecto he seguido la siguiente estructura:

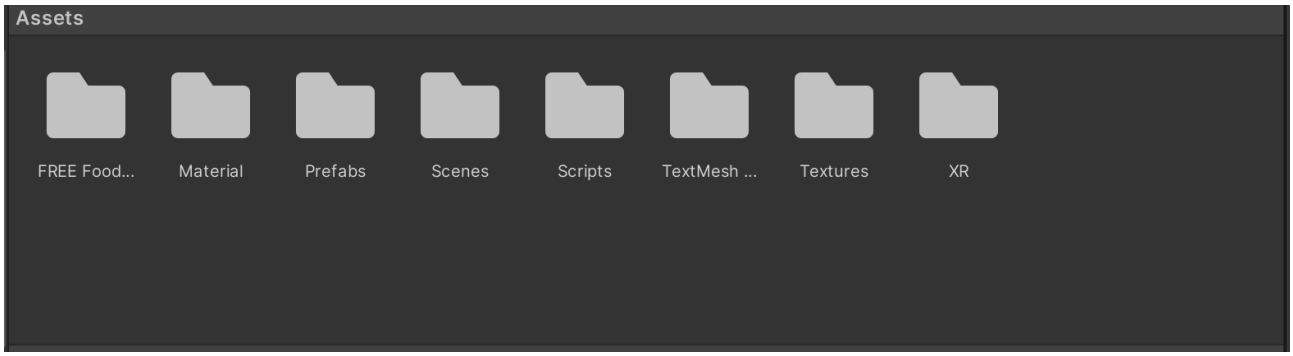


Figura 3.5: Scaffolding Unity.

- **FREE Food Pack:** Contiene todos los modelos de los alimentos.
- **Material:** Contendrá diferentes materiales para aplicar a la máscara de la cara. Aunque en la versión final no se utiliza ningún material ya que interesa poder ver la cara del jugador, es útil para testear que se está aplicando correctamente el seguimiento facial y que se podrá acceder a los indicadores faciales.
- **Prefab:** Los prefabs actúan como plantillas de objetos GameObject de la escena. Modificar un prefab permite aplicar esos cambios a todos los objetos de la escena a la vez. En concreto, se realizaron prefabs de los GameObjects que representan los alimentos y un prefab de la máscara de la cara.
- **Scenes:** Las escenas contienen los entornos y menús del juego, se puede entender como un nivel o pantalla especifico. Para este proyecto solo es necesario dos escenas, una para cada juego.
- **Scripts:** Contendrá los diferentes scripts con las mecánicas y la lógica del juego.
- **TextMeshPro:** Para utilizar elementos de texto renderizados.
- **Textures:** Contiene las texturas especificas para la cara.
- **XR:** Directorio con las dependencias que necesita la librería de ARKit.

3.2.4. Estructura de la primera escena

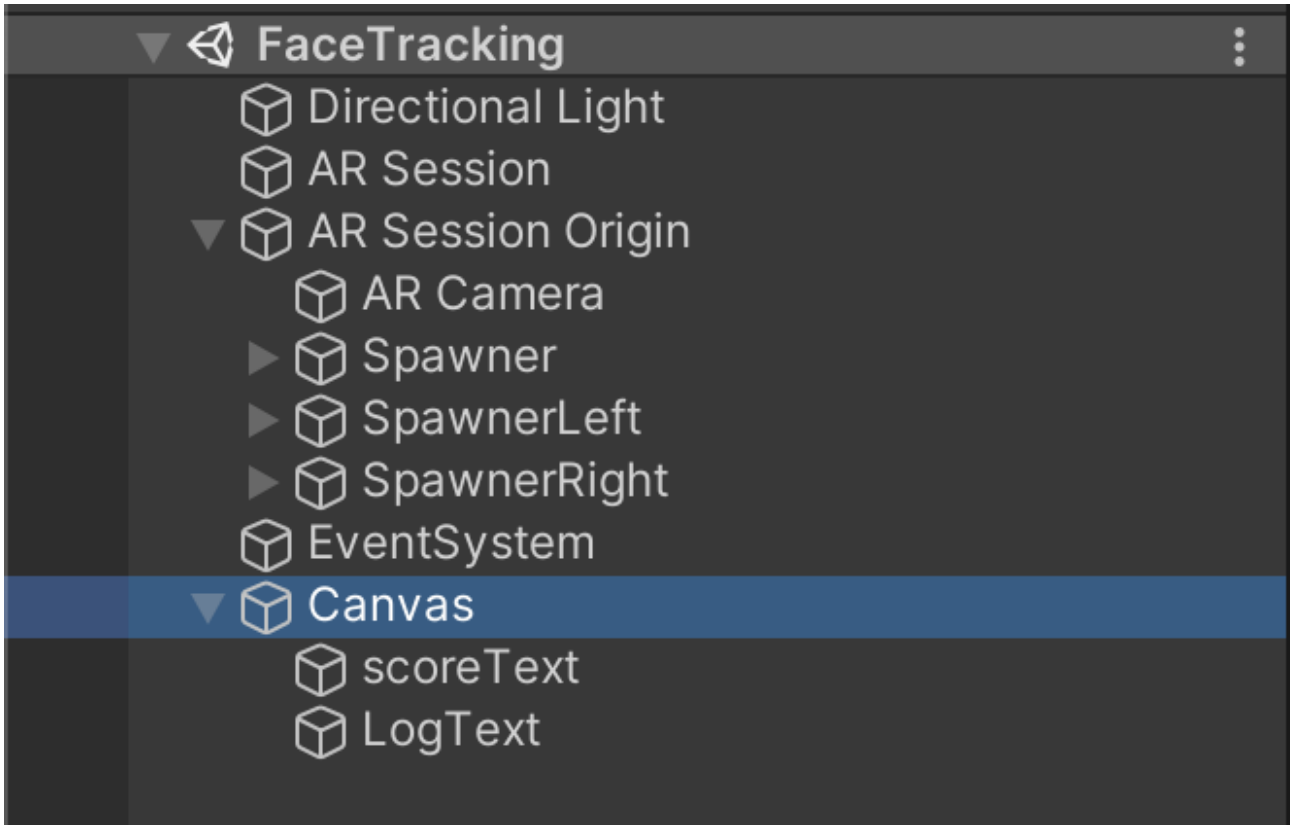


Figura 3.6: Estructura.

La escena del primer juego contiene como elementos más importantes:

- **AR Session:** Controla el ciclo de vida y la configuración de la sesión de realidad aumentada.
- **AR Session Origin:** Contendrá la cámara y todos los GameObjects que aparecerán.
- **Spawners:** Estos GameObjects indicaran los puntos de aparición de los objetos que caerán por la pantalla.
- **Canvas:** Contendrá la interfaz de la pantalla, con la puntuación y un texto para poder depurar los coeficientes de los blendShapes.

3.2.5. Scripts

BlendShape Tracker

Script principal que será el encargado de recoger los coeficientes de los blendshapes. Los principales atributos de esta clase son:

- **collisionArea:** Se inicializará con el gameObject que usaremos para colisionar, es decir, la esfera que se posicionará en la boca. Al indicar que el atributo es un SerializeField nos permite inicializar la variable desde el editor.
- **collisionAreaOffset:** Nos permitirá ajustar la posición de la esfera para posicionarla donde queremos.

- **ARKitBlendShapeLocations:** Recogeremos los diferentes blendShapes que utilizaremos para analizar la lengua y su posición. Como podemos ver en la imagen, se han usado los coeficientes de TongueOut para comprobar que la lengua esta fuera, jawForward para comprobar que la mandíbula esta hacia delante y por último MouthLeft y MouthRight para comprobar si la boca esta orientada hacia la izquierda o hacia la derecha respectivamente.
- **currentBlendShape:** Es un diccionario que contendrá todos los blendShapes que se usarán.
- **faceSubsystem y face:** Nos permite inicializar y actualizar el sistema de tracking de la cara.

```
8   [RequireComponent(typeof(ARFace))]
9   public class BlendShapeTracker : Singleton<BlendShapeTracker>
10  {
11      [SerializeField]
12      public GameObject collisionArea;
13
14      [SerializeField]
15      private Vector3 collisionAreaOffset;
16          You, hace 3 días • first game completed
17      [SerializeField]
18      private ARKitBlendShapeLocation tongue = ARKitBlendShapeLocation.TongueOut;
19      private ARKitBlendShapeLocation jawForward = ARKitBlendShapeLocation.JawForward;
20      private ARKitBlendShapeLocation mouthLeft = ARKitBlendShapeLocation.MouthLeft;
21      private ARKitBlendShapeLocation mouthRight = ARKitBlendShapeLocation.MouthRight;
22
23      private Dictionary<ARKitBlendShapeLocation, float> currentBlendShape;
24
25      private ARKitFaceSubsystem faceSubsystem;
26
27      private ARFace face;
```

Figura 3.7: Atributos de la clase.

También, es necesario guardar el valor de los coeficientes descritos y que estos puedan ser leídos desde otros scripts para poder implementar la lógica de la puntuación.

```
29  public float TongueCoefficient
30  {
31      private set;
32      get;
33  }
34
35  public float jawForwardCoefficient
36  {
37      private set;
38      get;
39  }
40
41  public float MouthLeftCoefficient
42  {
43      private set;
44      get;
45  }
46
47  public float MouthRightCoefficient
48  {
49      private set;
50      get;
51  }
```

Figura 3.8: Atributos de los coeficientes.

La función Awake permite inicializar cualquier variable antes de que se inicie el juego. Se ha utilizado esta función para rellenar el diccionario con lo que necesitamos.

El método onEnable se activa cuando el GameObject que lo contiene este habilitado y activo. Es necesario utilizar esta función para actualizar el sistema de tracking y que todo funcione correctamente.

```
53      void Awake()
54      {
55          face = GetComponent<ARFace>();
56          currentBlendShape = new Dictionary<ARKitBlendShapeLocation, float>();
57          currentBlendShape.Add(tongue, 0);
58          currentBlendShape.Add(mouthLeft, 0);
59          currentBlendShape.Add(mouthRight, 0);
60          currentBlendShape.Add(jawForward, 0);
61      }
62
63      void OnEnable()
64      {
65          var faceManager = FindObjectOfType<ARFaceManager>();
66
67          if (faceManager != null)
68          {
69              faceSubsystem = (ARKitFaceSubsystem)faceManager.subsystem;
70          }
71
72          face.updated += OnUpdated;
73      }
```

Figura 3.9: Funciones Awake y OnEnable.

La función UpdatefaceFeatures es la encargada de recoger los coeficientes de los blendShapes que se necesitan.

```
85
86  void UpdateFaceFeatures()
87  {
88    using (var blendShapes = faceSubsystem.GetBlendShapeCoefficients(face.trackableId, Allocator.Temp)) {
89      foreach( var blendShape in blendShapes ){
90        if(currentBlendShape.TryGetValue(blendShape.blendShapeLocation, out float coefficent )) {
91          if(blendShape.blendShapeLocation == tongue){
92            TongueCoefficient = coefficent;
93          }
94          if(blendShape.blendShapeLocation == jawForward){
95            jawForwardCoefficient = coefficent;
96          }
97          if(blendShape.blendShapeLocation == mouthLeft){
98            MouthLeftCoefficient = coefficent;
99          }
100         if(blendShape.blendShapeLocation == mouthRight){
101           MouthRightCoefficient = coefficent;
102         }
103         currentBlendShape[blendShape.blendShapeLocation] = blendShape.coefficient;
104       }
105     }
106   }
107 }
```

Figura 3.10: Actualizando los coeficientes que necesitamos.

Spawner

Este script es el encargado de crear los objetos que caerán por la pantalla y destruirlos cuando pase un tiempo. Para ello, es necesario el atributo prefab con la opción de SerializeField donde elegiremos el prefab desde el editor.

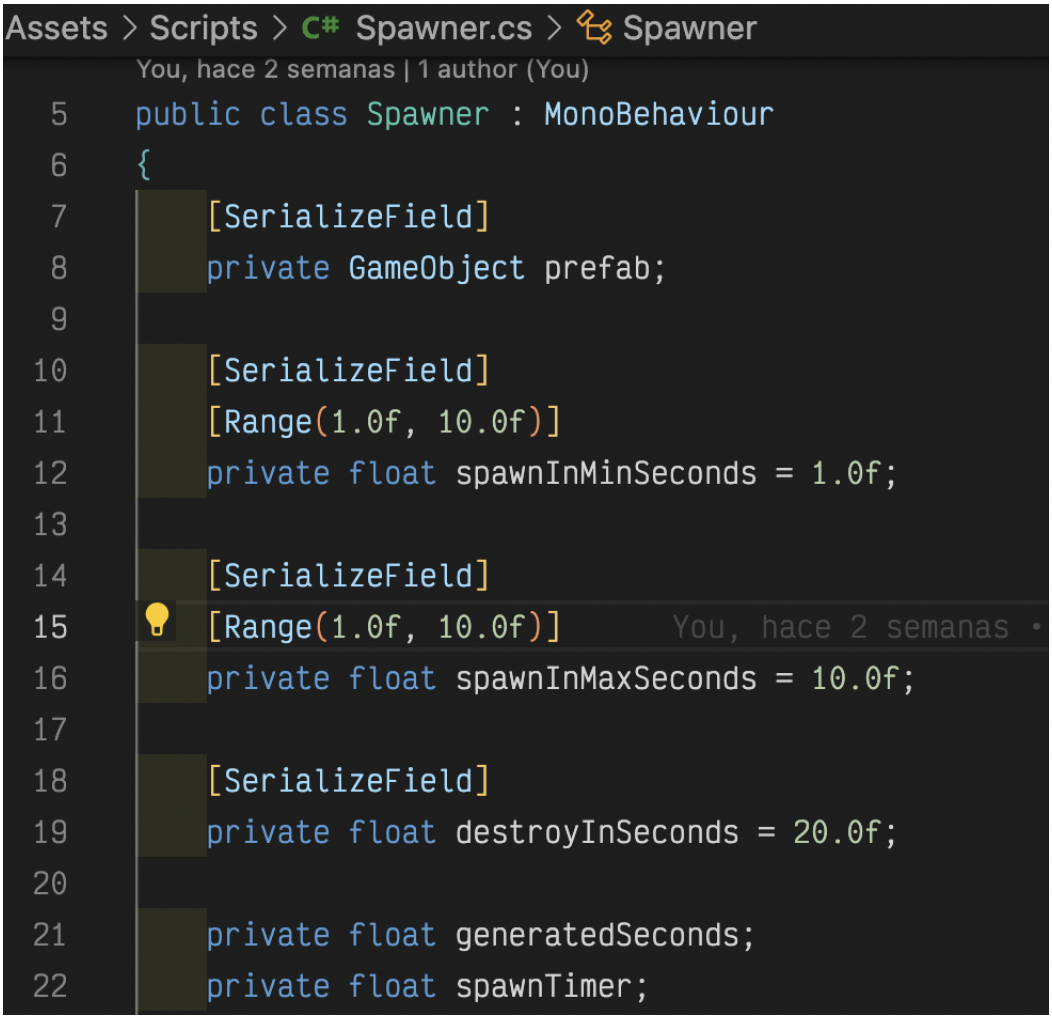


Figura 3.11: Atributos de la clase spawner.

Assets > Scripts > C# Spawner.cs > Spawner

```
24      void Awake()
25      {
26          spawnTimer = generatedSeconds = Random.Range(spawnInMinSeconds, spawnInMaxSeconds);
27      }
28
29      void Update()
30      {
31          if(spawnTimer ⩾ generatedSeconds)
32          {
33              var go = Instantiate(prefab, transform.position, prefab.transform.rotation);
34              Destroy(go, destroyInSeconds);
35
36              spawnTimer = 0;
37              generatedSeconds = Random.Range(spawnInMinSeconds, spawnInMaxSeconds);
38          }
39          else
40          {
41              spawnTimer += Time.deltaTime * 1.0f;
42          }
```

Figura 3.12: Funciones de spawner.

Scale

El script Scale permite escalar mejor los objetos que aparecen en pantalla y agrega un efecto de goma a los objetos mientras caen haciéndolos más animados.

Assets > Scripts > C# Scale.cs > Scale > Update()

```
    You, hace 2 semanas | 1 author (You)
5   public class Scale : MonoBehaviour
6   {
7       [SerializeField]
8       private float scalingSpeed = 0.3f;
9
10      [SerializeField]
11      private float targetScale = 0.01f;
12
13      [SerializeField]
14      private float length = 0.001f;
15
16      float scalingTime;
17
18      void Update()
19      {
20          scalingTime = Time.time * scalingSpeed;
21          transform.localScale = new Vector3(transform.localScale.x, Mathf.PingPong(scalingTime, length) + targetSca
22      }       You, hace 2 semanas • movement, scale, spawner and UIManager script …
```

Figura 3.13: Scale script.

Movement

En total hay tres scripts de movement, cada uno comprueba que la lengua esté en la posición correcta, izquierda derecha o centro. Si se cumple esa condición y además el objeto entra en colisión con la boca, gracias al método OnTriggerEnter, se incrementa la puntuación y se destruye el objeto.

```
3   public class MovementLeft : MonoBehaviour
4   {
5       [SerializeField]
6       private Vector3 speed;
7
8       [SerializeField]
9       private float destroyAfterDead = 1.0f;
10      private bool eaten = false;
11
12      void Update()
13      {
14          transform.position += speed * Time.deltaTime;
15      }
16
17      private void OnTriggerEnter(Collider other)
18      {
19          if(other.name.Contains("Mouth") &&
20          BlendShapeTracker.Instance.TongueCoefficient > 0.0000001f &&
21          BlendShapeTracker.Instance.MouthLeftCoefficient > 0.002f )      You, hace 1 segundo • Uncommitted cha
22          {
23              if(!eaten)
24              {
25                  Logger.Instance.LogInfo("Tongue(M) " + BlendShapeTracker.Instance.TongueCoefficient);
26                  Logger.Instance.LogInfo("Mouth Left(M) " + BlendShapeTracker.Instance.MouthLeftCoefficient);
27                  eaten = true;
28                  UIManager.Instance.AddScore();
29                  Destroy(gameObject, destroyAfterDead);
30              }
31          }
32      }
33  }
```

Figura 3.14: Script de MovementLeft.

UIManager

Controla el aumento de la puntuación y la actualización del texto en el elemento canva.

Assets > Scripts > C# UIManager.cs > ...

You, hace 2 semanas | 1 author (You)

```
1   using UnityEngine;
2   using TMPro;
3   using Singleton;
4
    You, hace 2 semanas | 1 author (You)
5   public class UIManager : Singleton<UIManager>
6   {
7       [SerializeField]
8       private TextMeshProUGUI scoreText;
9       private int score;
10      public void AddScore()
11      {
12          score++;
13          scoreText.text = $"{score}";
14      }
15  }
16
```

Figura 3.15: Script de UIManager.

3.2.6. Configuraciones en el editor de Unity

Malla de la cara

Para añadir la malla de la cara se creó un nuevo prefab agregando un GameObject desde XR/AR Default Face y una esfera como elemento hijo. También se añadió el script blendShapeTracker, un rigidbody y un box Collider a la esfera. Es necesario activar las opciones de is Kinematic e is Trigger para que la función OnTriggerEnter del script se active correctamente.

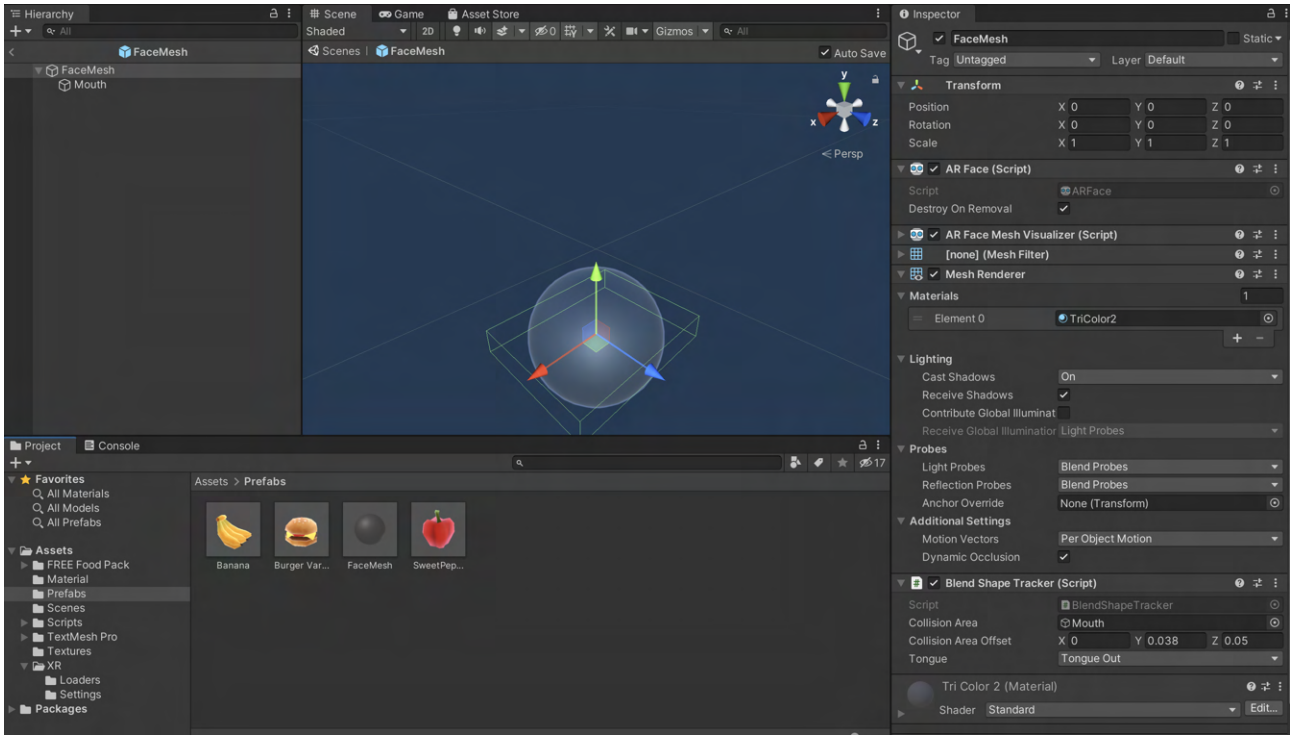


Figura 3.16: Configuración de la malla de la cara.

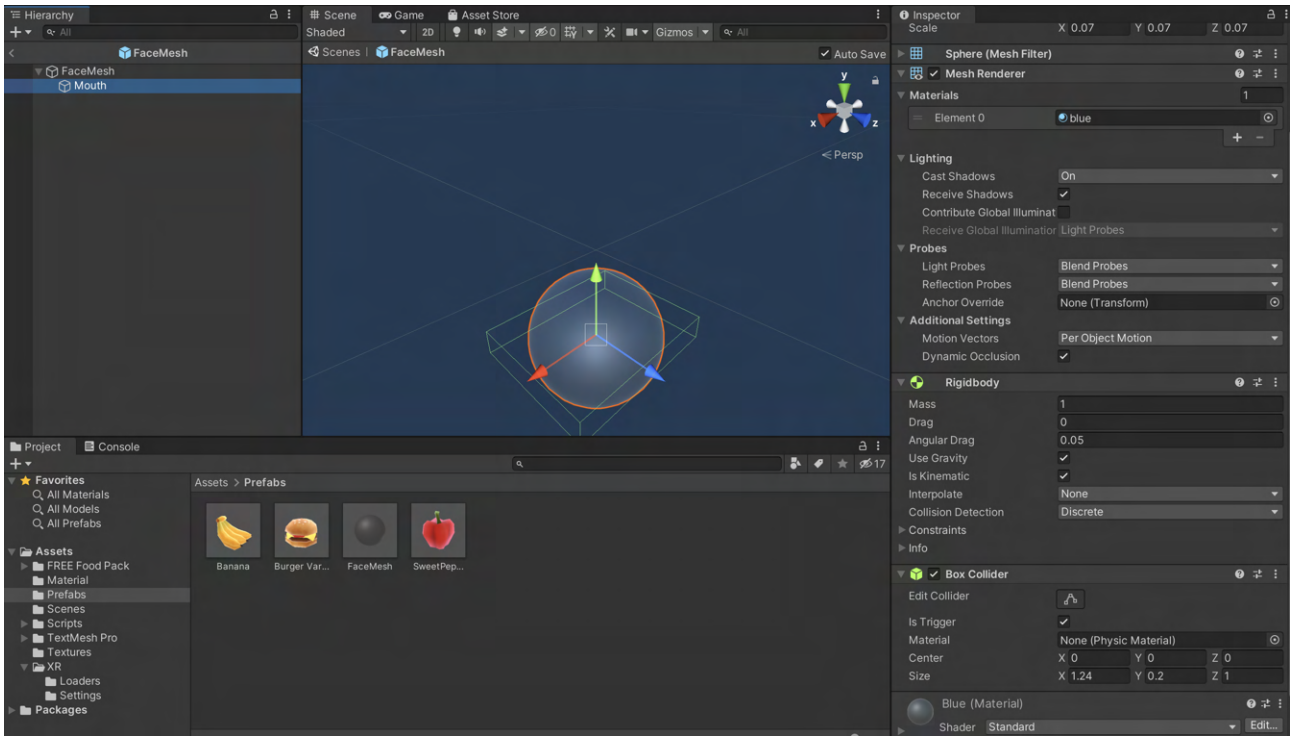


Figura 3.17: Configuración de la esfera de la boca.

Figura 3.18: Resultado final de la malla de la cara y la esfera de la boca.

Configuración de los objetos

Se agregaron los scripts de movement y scale al prefab y al igual que con el anterior elemento se añadió un box collider y un rigidBody.

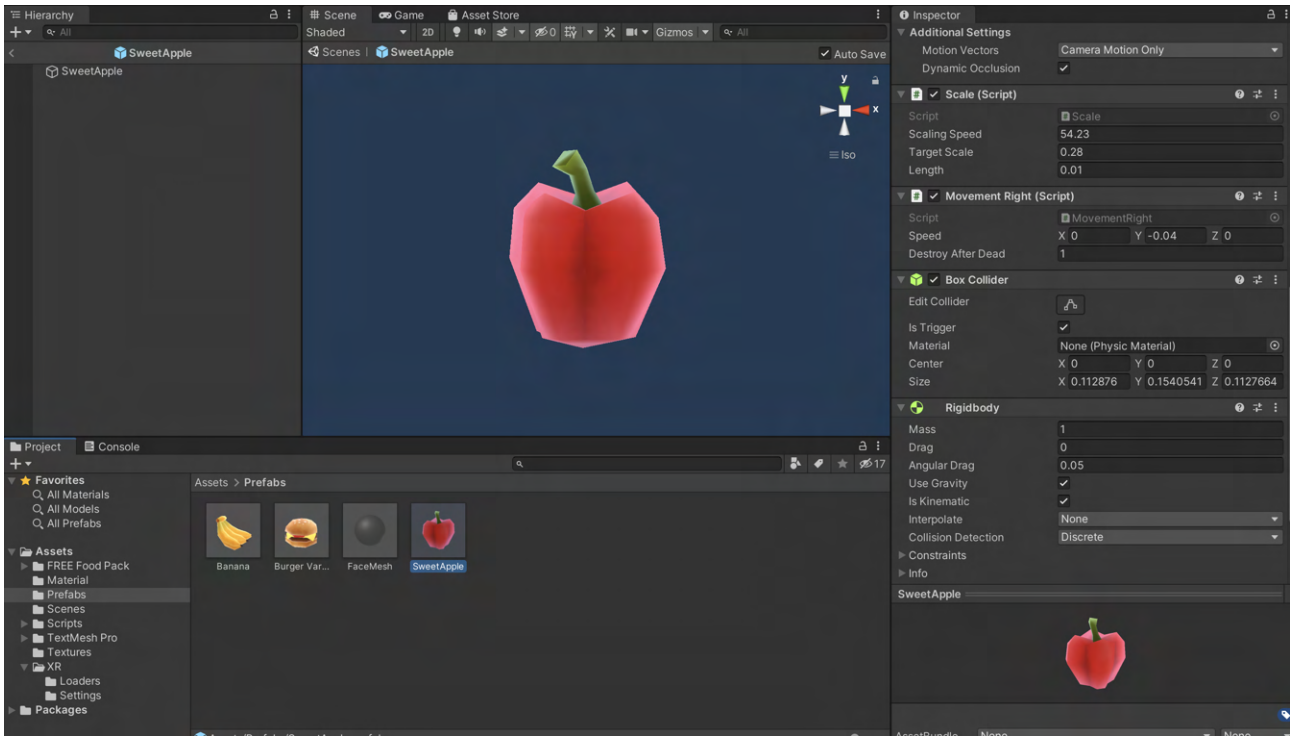


Figura 3.19: Configuración del prefab manzana.

Configuración del spawner

Los GameObjects spawner contendrán una pequeña esfera simple para poder usarlos como referencia y posicionarlos en la pantalla y se les agregará el script spawner con el objeto que se va a invocar.

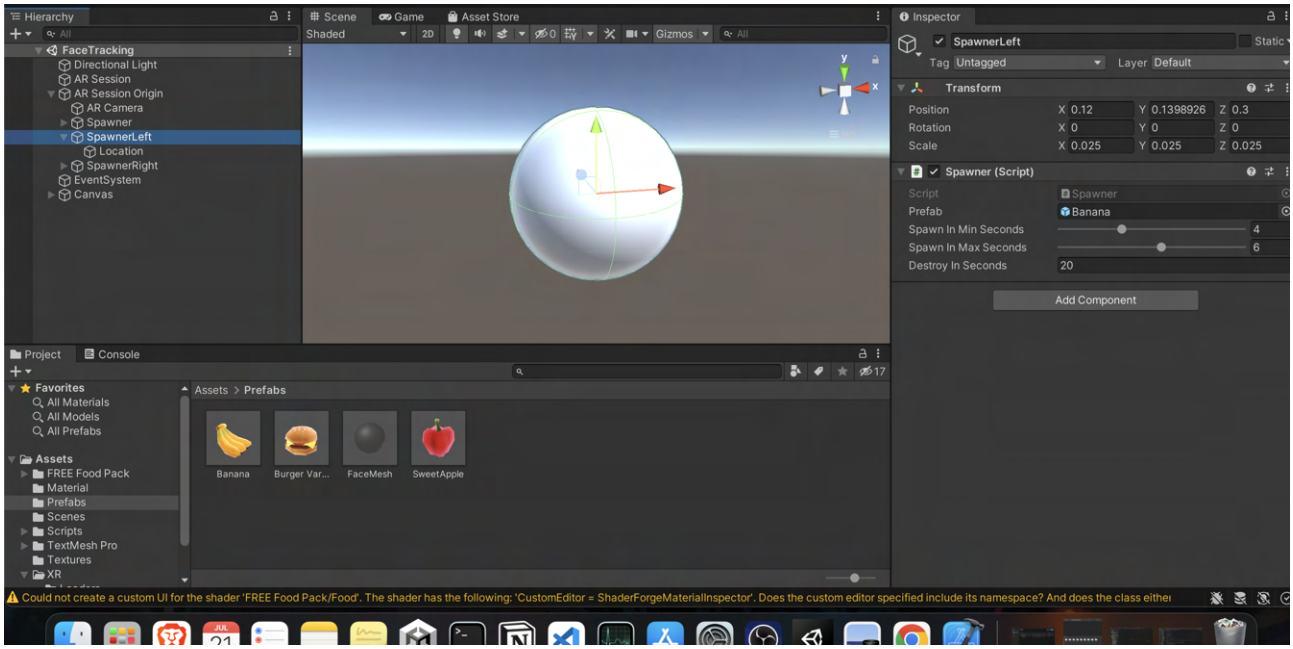


Figura 3.20: Configuración del spawner.

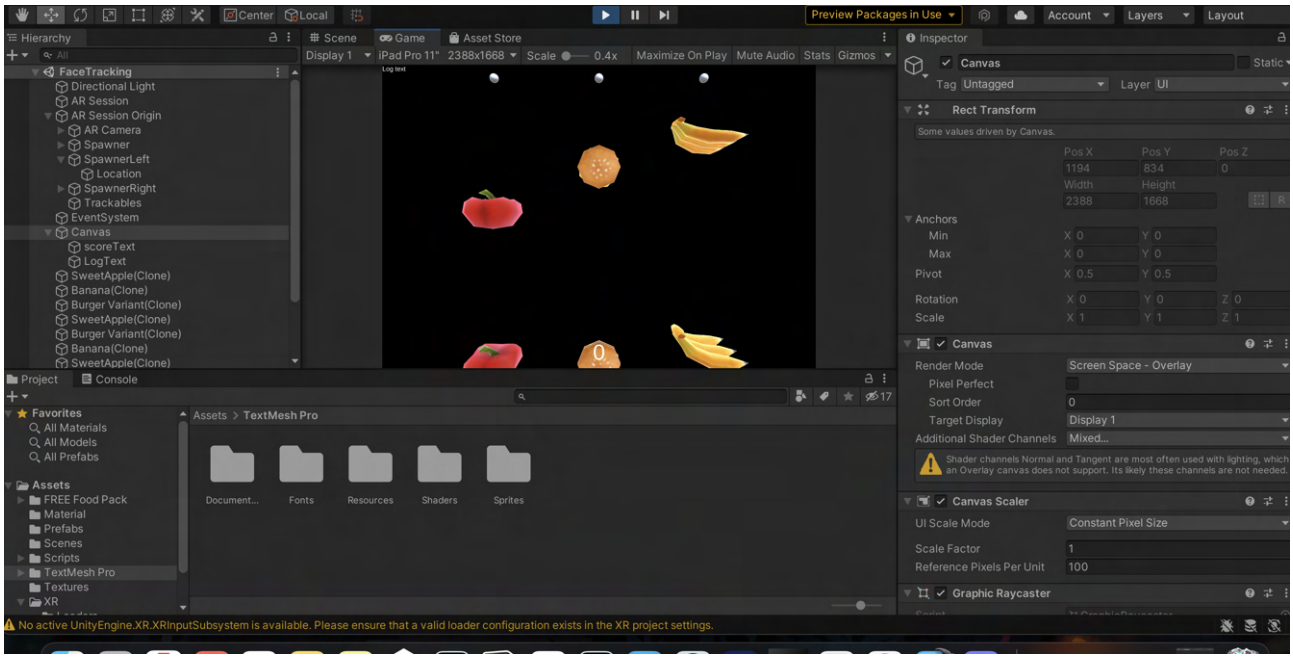


Figura 3.21: Resultado final del spawn de objetos.

Configuración de la interfaz de pantalla

La interfaz de pantalla mostrará la puntuación. Este elemento es bastante simple, consiste en un canvas con el texto posicionado en la parte inferior de la pantalla y con el script UIManager.

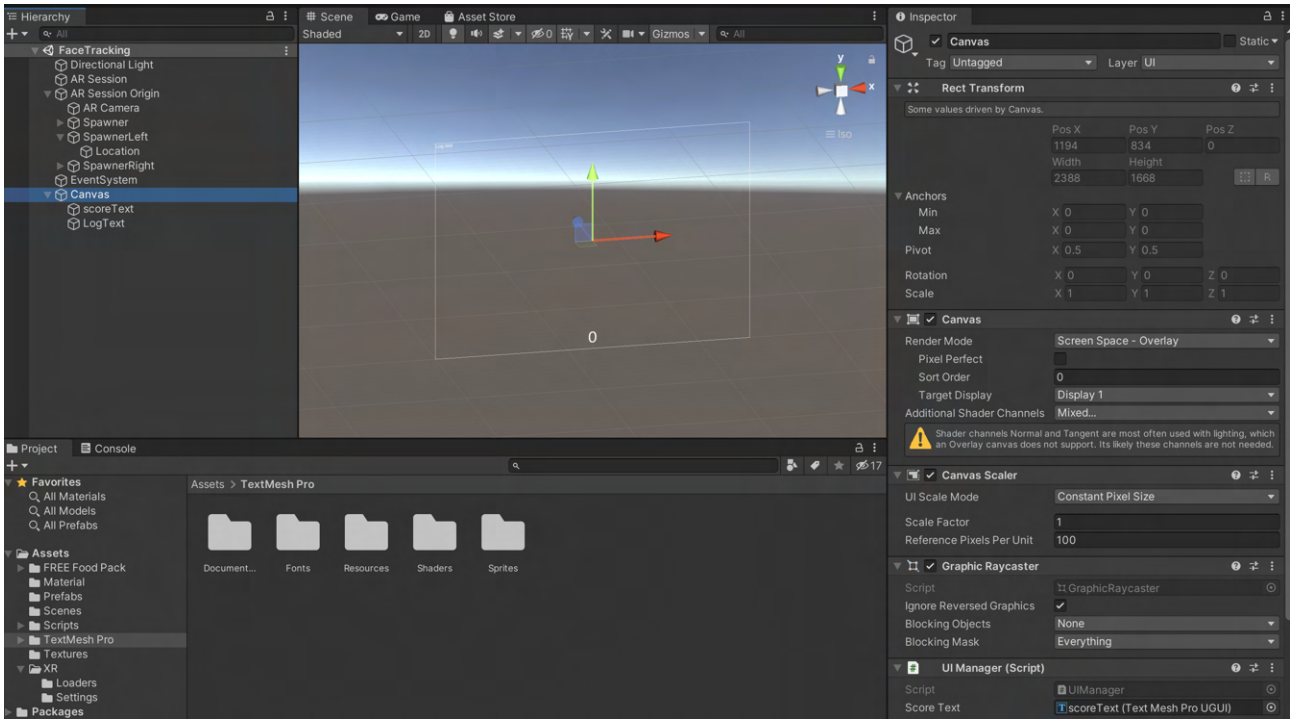


Figura 3.22: Configuración de la interfaz.

3.2.7. Resultado final del juego

Figura 3.23: Capturando objetos sacando la lengua.

Figura 3.24: Capturando objetos con la lengua a la izquierda.

Figura 3.25: Capturando objetos con la lengua a la derecha.

3.3. Desarrollo del segundo juego

3.3.1. Prototipado del segundo juego AR

El prototipo del segundo juego ya había sido creado por los alumnos de la UDEM. La idea principal de este juego es ejercitar la zona labial poniendo la boca en forma de embudo. Como entre los assets gratuitos no se encuentra ningún modelo de tarta o velas, se ha utilizado un modelo provisional de una bebida.

Figura 3.26: Prototipado del segundo juego.

3.3.2. Scripts

Se ha reutilizado la mayoría de lógica del primer juego. La principal diferencia ha sido modificar el comportamiento del spawner. Para este caso, solo aparecerá un objeto a la vez. Solo cuando el objeto es eliminado de la escena, aparecerá otro.

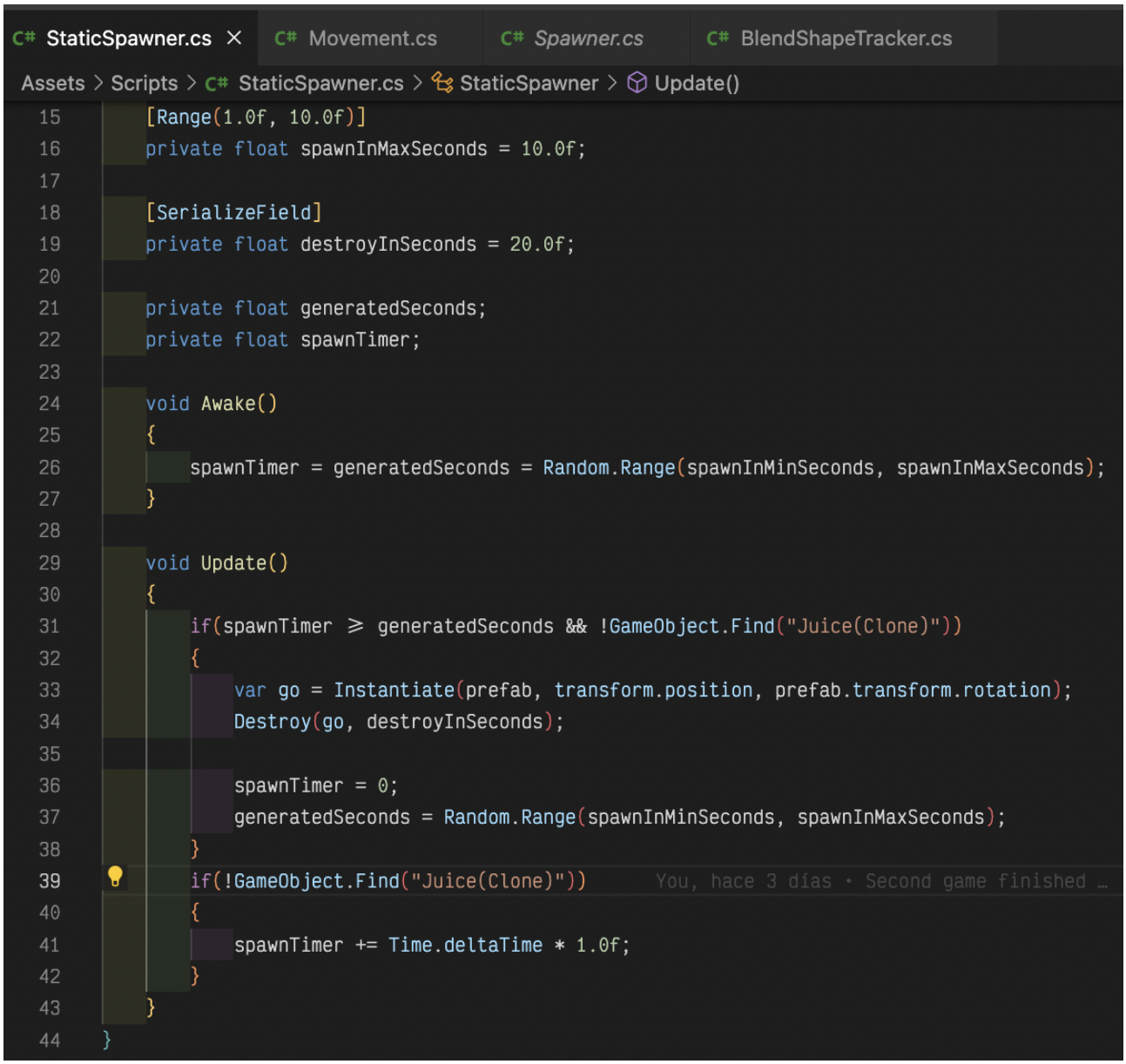


Figura 3.27: Sript Modificado.

Figura 3.28: Pantalla inicial del juego.

Figura 3.29: Capturando el objeto.

Capítulo 4

Desarrollo de la aplicación

En este capítulo se explicarán las diferentes fases de desarrollo del frontend de la aplicación y sus resultados.

4.1. Prototipado

Al igual que con el desarrollo de los juegos, se realizó un prototipado en figma de las principales pantallas para tener una mejor idea de como iba a ser la interfaz.

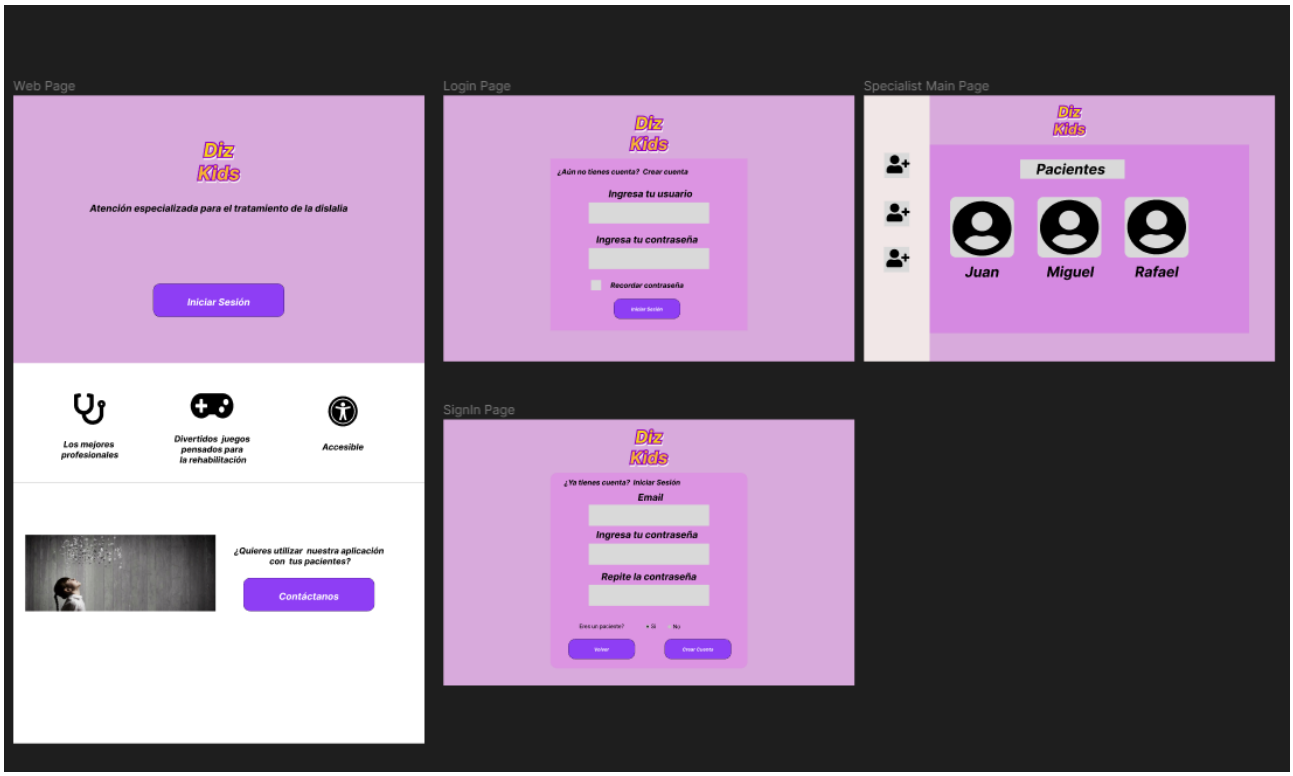


Figura 4.1: Prototipado.

4.2. Scaffolding del proyecto

El proyecto se ha creado con la herramienta de create react app, y se ha seguido la estructura de desarrollo que se muestra a continuación:

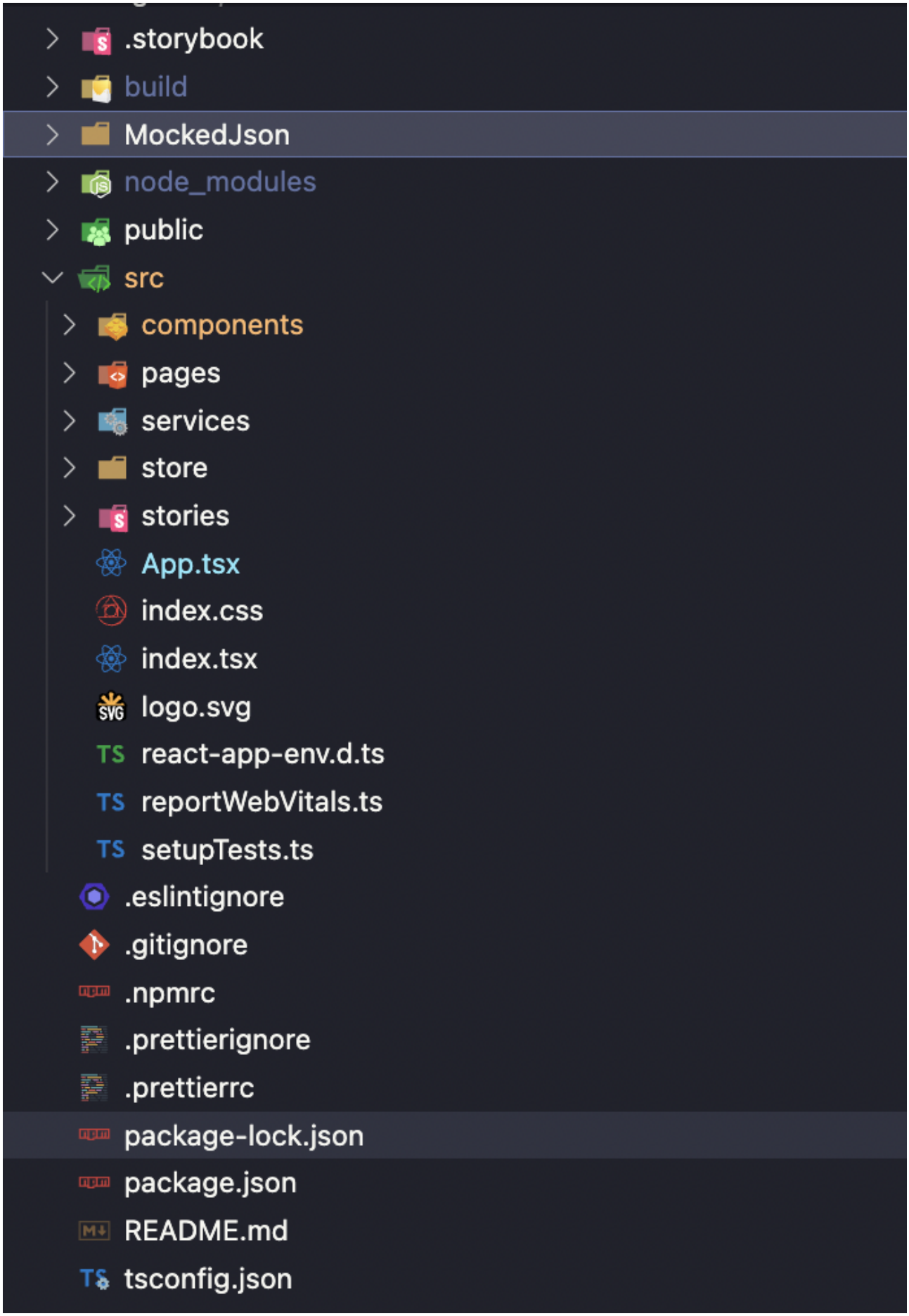


Figura 4.2: Scaffolding frontend.

Cada componente tendrá un directorio de test y los ficheros correspondientes para el storybook, los estilos css y el propio fichero .tsx. Adicionalmente, en services se almacenarán las funciones que realicen llamadas a la API, permitiendo desacoplar esa lógica de los componentes.

4.3. Componentes

Algunos ejemplos de los diferentes componentes que se han desarrollado son:

4.3.1. MarketingCard

Componente de la página principal, hace uso de los props para mostrar la información y el estilo de la carta.

Figura 4.3: Marketing Card.

4.3.2. Login

Componente del formulario de login, si el inicio de sesión tiene éxito, almacena en los estados de Redux la información correspondiente y navega a la página principal del usuario.

Figura 4.4: Login.

4.3.3. Lista de pacientes

Componente que muestra el listado de pacientes. Obtiene información de los estados de Redux.

Figura 4.5: Lista de pacientes.

4.4. Storybook

El uso de Storybook facilitó el desarrollo de cada componente por separado y agilizó dicha tarea. Además, se registraba un histórico de los componentes y páginas desarrolladas.

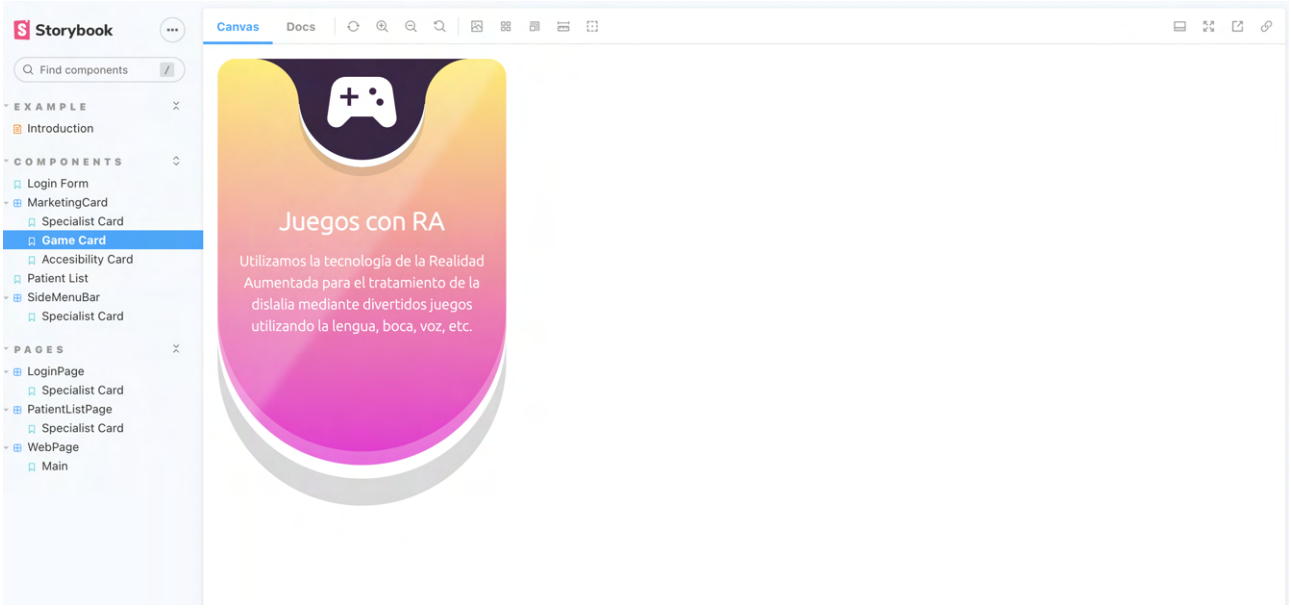


Figura 4.6: Storybook.

4.5. Testing

Aunque en un inicio se planteó realizar una metodología TDD durante el desarrollo, esto no fue posible cumplirlo. Sin embargo, se realizaron test sobre algunos de los componentes probando la librería de testing library.

Figura 4.7: Testing.

4.6. CI

Para la integración continua durante el desarrollo se han utilizado los actions de GitHub. En concreto, se desarrolla un action que se encargara de comprobar que el código subido a la Pull Request pasase los test y cumpliese con las reglas de ESLint y Prettier.

4.7. Smartmock

Como no se contaba con una API, se ha utilizado la herramienta de SmartMock que permite hacer simulaciones. La herramienta es bastante sencilla de utilizar, simplemente hay que crearlos y definir la respuesta que devolverá.

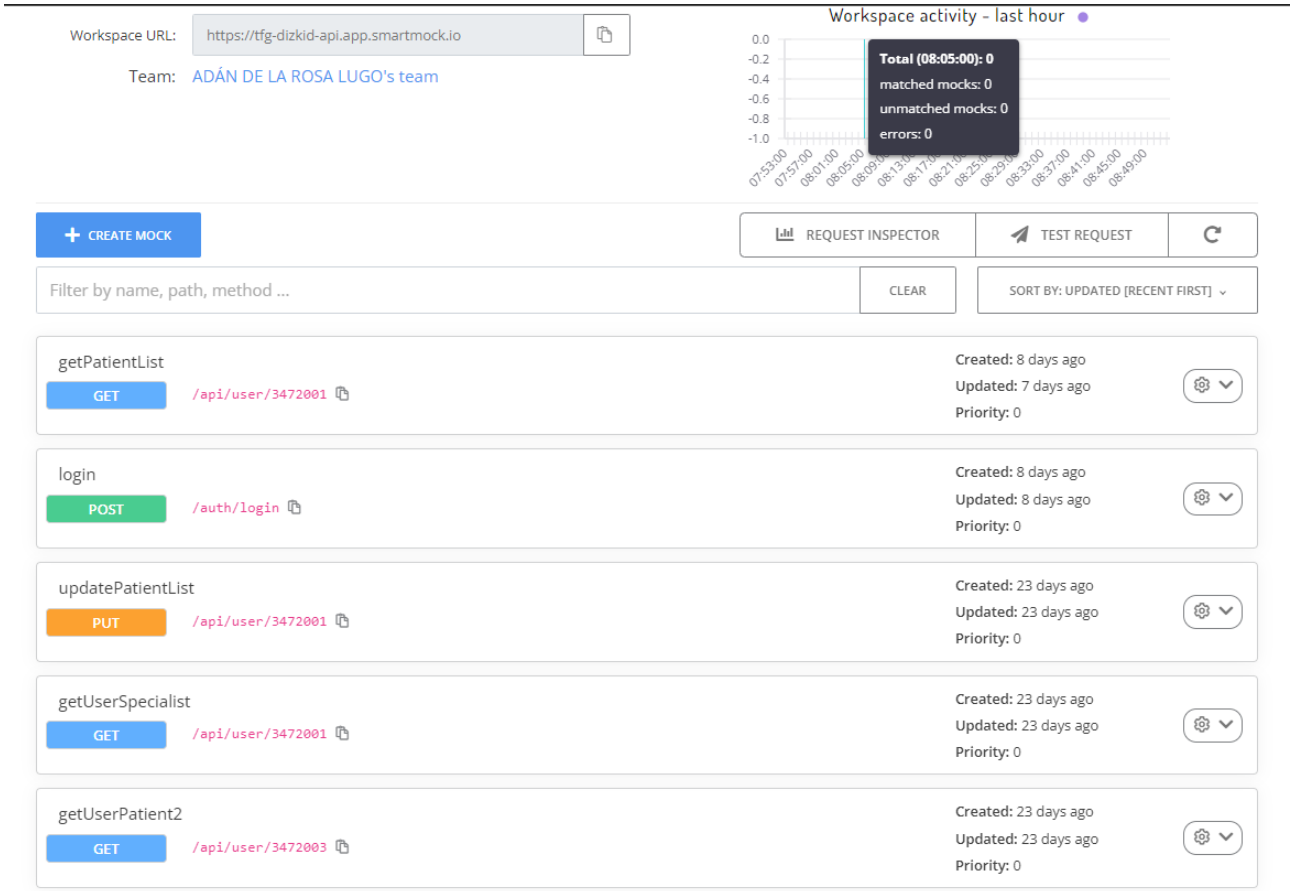


Figura 4.8: Smartmock.

Capítulo 5

Conclusiones y líneas futuras

Para concluir, se han desarrollado dos prototipos de juegos de realidad aumentada con el motor Unity y la librería de ARKit de Apple. Mediante el uso de los diferentes coeficientes de los gestos de la cara y de la boca. Esto da como resultado dos juegos que ayudarán a tratar la dislalia, permitiendo a los usuarios atrapar diferentes objetos representados con comida y ganar el máximo de puntos posible.

Así mismo, se ha desarrollado el frontend de una aplicación web con React, utilizando herramientas de clean code como formateadores y linters que ayudan a mantener una estructura consistente y coherente en el código. También se ha realizado testing y storybook para el desarrollo de los diferentes componentes, agilizando su desarrollo y creando un histórico de cada uno. Además, se ha realizado integración continua mediante GitHub actions. En definitiva, se ha desarrollado una aplicación que ayudará a los especialistas a gestionar la lista de sus paciente y poder ver su progreso.

A pesar de que se ha finalizado el proyecto, se han analizado posibles líneas futuras con las que se puede mejorar y ampliar el mismo. A continuación, se detallarán algunas de ellas:

- Desarrollar un backend y establecer una API a la que realizar las consultas desde el frontend realizado. Se deberán definir los endpoints necesarios y conectar una base de datos donde almacenar la información.
- Definir un sistema de vidas y dificultades para los juegos, en los que los usuarios puedan elegir entre modo fácil, medio o difícil. Para ello, será necesario crear una interfaz y una pantalla de inicio.
- Creación de nuevos juegos para diferentes tipos de dislalia. Este proyecto se ha enfocado principalmente para aquellos pacientes que ejerciten la lengua o los labios con movimientos, sin embargo, se podrían realizar otros que implicaran realizar ciertos sonidos, movimiento de mejillas o soplar.

Capítulo 6

Summary and Conclusions

In conclusion, it has been developed two AR game prototypes with Unity game engine and the Apple library called ARKit by using different face expression coefficients. Specifically combining those of the face area to track the tongue or lips. As a result this two games will help to treat dyslalia allowing users to catch different objects represented as food and win the maximum amount of points.

Furthermore, it has been developed the frontend application made with React, using clean code tools like formatters and linters that help to maintain a consistent structure in the code. Moreover, it has been used techniques of testing and storybook in order to develop differents components, making the process lighter with a history of each one. Besides, continuous integration has been employed with Github actions. In short, the application will help specialists to manage their list of patients and watch their progress.

Despite of having the project ended, new lines have been analyzed to improve and expand itself. Some of them are:

- Develop a backend application and an API to make consults. Endpoints and a database connection must be defined.
- Create a life system and differents difficulties for the games such as easy, normal and hard mode. For this, it’s necessary to create a menu interface and a home screen.
- Make new games for other differents types of dyslalia. This project has been mainly focused on those patients that exercise the tongue and lips with movements, however, others games could be done that require sounds, cheeks movement or blow.

Capítulo 7

Presupuesto

A continuación, se especifica el presupuesto estimado del desarrollo del proyecto realizado. En total se estima que el coste total del proyecto es de 10610€, teniendo en cuenta que el coste de desarrollo y el número de horas es de un desarrollador junior, localizado en Canarias.

Descripción	Cantidad	Coste
Licencia de desarrollador de Apple	1ud	100€
Dispositivo Apple con cámara	1ud	1000€
Software-Licencia de Unity, Assets	1ud	0€
Estudio de tecnologías para el desarrollo	30h	360€
Desarrollo de las aplicaciones de Realidad Aumentada	230h	5750€
Desarrollo de la aplicación web	170h	3400€
Total		10610€

Tabla 7.1: Resumen de presupuesto

Bibliografía

[1] Ar foundation. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html. (Accessed: 01.03.2022).

[2] Arcore. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html. (Accessed: 01.03.2022).

[3] Arkit. https://developer.apple.com/augmented-reality/arkit/. (Accessed: 01.03.2022).

[4] Documentación de blendshapes. https://developer.apple.com/documentation/arkit/arfaceanchor/2928251-blendshapes. (Accessed: 01.03.2022).

[5] Documentación de c. https://docs.microsoft.com/es-es/dotnet/csharp/. (Accessed: 01.03.2022).

[6] Documentación del blendshape jaw forward. https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation/2928229-jawforward. (Accessed: 01.03.2022).

[7] Documentación del blendshape jaw open. https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation/2928236-jawopen. (Accessed: 01.03.2022).

[8] Documentación del blendshape mouth left. https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation/2928228-mouthleft. (Accessed: 01.03.2022).

[9] Documentación del blendshape mouth right. https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation/2928246-mouthright. (Accessed: 01.03.2022).

[10] Enlace a fichero .stories.tsx. https://github.com/AdanRL/dizkids-frontend/blob/main/src/components/MarketingCard/MarketingCard.stories.tsx. (Accessed: 01.08.2022).

[11] Enlace a figma. https://www.figma.com/. (Accessed: 01.08.2022).

[12] Enlace a figma. https://assetstore.unity.com/packages/3d/props/food/free-casual-food-pack-mobile-vr-85884. (Accessed: 01.08.2022).

[13] Enlace a package.json del proyecto. https://github.com/AdanRL/dizkids-frontend/blob/main/package.json. (Accessed: 01.08.2022).

[14] Enlace al directorio con las rutas. https://github.com/AdanRL/dizkids-frontend/blob/main/src/App.tsx. (Accessed: 01.08.2022).

[15] Enlace al directorio store. https://github.com/AdanRL/dizkids-frontend/tree/main/src/store. (Accessed: 01.08.2022).

[16] Enlace al fichero .prettierrc. https://github.com/AdanRL/dizkids-frontend/blob/main/.prettierrc. (Accessed: 01.08.2022).

[17] Eslint. https://eslint.org/. (Accessed: 01.03.2022).

[18] Git. https://git-scm.com. (Accessed: 01.03.2022).

[19] Github. https://github.com. (Accessed: 01.03.2022).

[20] Notion. https://www.notion.so/. (Accessed: 01.03.2022).

[21] Prettier. https://prettier.io/. (Accessed: 01.03.2022).

[22] React. https://es.reactjs.org/docs/getting-started.html. (Accessed: 01.03.2022).

[23] React router. https://reactrouter.com/en/main. (Accessed: 01.03.2022).

[24] Redux. https://es.redux.js.org/. (Accessed: 01.03.2022).

[25] smartmock. https://smartmock.io/. (Accessed: 01.03.2022).

[26] Storybook. https://storybook.js.org/. (Accessed: 01.03.2022).

[27] Testing library. https://testing-library.com/. (Accessed: 01.03.2022).

[28] Unity. https://unity.com/. (Accessed: 01.03.2022).

[29] Unity cloud build. https://unity.com/es/features/cloud-build. (Accessed: 01.03.2022).