TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

REPORTE DE PRÁCTICA:

RECONOCIMIENTO BASADO EN UBICACIÓN.

MATERIA:

DESARROLLO CON REALIDAD AUMENTADA

PRESENTA:

Darío Sánchez Linton

Edwin Ortiz Cruz

Diego Alexis Carlos Cruz

Irving Fernando Reyes Pacheco

DOCENTE:

Ing. José Alfredo Román Cruz

CARRERA:

Ingeniera en Sistemas Computacionales

Grupo: 8US

Tlaxiaco, Oax. abril de 2024.

"Educación, ciencia y tecnología, progreso día con día"®

ÍNDICE.

ÍNDICE DE FIGURAS.

INTRODUCCIÓN

El siguiente documento muestra cómo comenzar con AR.js v2.0.x y AR basada en ubicación agregando lugares estáticamente solo usando HTML, también a agregar lugares estáticamente usando Javascript, también con una UX/UI mínima y a agregar lugares dinámicamente usando Javascript y API remotas. También verá cómo implementar su primera aplicación Web AR en un servidor remoto. Es preciso tener conocimientos básicos de HTML y Javascript. También es necesario mencionar que para cualquier duda que pueda surgir en el desarrollo de este documento ya sea relacionado con conceptos o metodología que no se aborde de manera detallada el lector proceda a indagar de forma autodidacta para adquirir un panorama más claro para la inmersión en este fascinante mundo de la realidad aumentada.

DESARROLLO

Objetivo de la práctica.
Implementar la realidad aumentada basada en la ubicación mediante el uso de AR.js y una aplicación en la web.

Descripción.
Un tutorial para crear aplicaciones de realidad aumentada basadas en la ubicación que se ejecutan en todos los navegadores, sin ninguna instalación en su teléfono gratis.

AR.js v2 introdujo la RA basada en ubicación en la Web por primera vez. Esto permite nuevas experiencias en AR y una gran oportunidad para los desarrolladores interesados en la Realidad Aumentada.

Básicamente, significa que ahora es posible ofrecer experiencias de RA sin marcadores. Los desarrolladores pueden especificar lugares de interés, representados por coordenadas del mundo real, en los que aparecerá el contenido AR.

Además, esta nueva característica permite combinar tanto la AR basada en marcadores, la forma 'clásica' de AR.js para aumentar la realidad, como la nueva AR basada en la ubicación, basada en datos de GPS.

Material.
- Computadora o teléfono con cámara.
- Conexión a internet
- Editor de texto.
- Nvegador web.

Procedimiento.
Primera forma: agregar lugares solo usando HTML
Archivo de plantilla
Comencemos a crear una nueva carpeta y un nuevo archivo, llamémoslo index.html. Luego copie y pegue las siguientes líneas:

```
<!DOCTYPE
html>
        <html>
        <head>
            <meta charset='utf-8'>
            <meta http-equiv='X-UA-Compatible' content='IE=edge'>
            <title>GeoAR.js demo</title>
```

```
    <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
    <script
src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-
ar.min.js"></script>
    <script src="https://raw.githack.com/donmccurdy/aframe-
extras/master/dist/aframe-extras.loaders.min.js"></script>
    <script>
        THREEx.ArToolkitContext.baseURL =
'https://raw.githack.com/jeromeetienne/ar.js/master/three.js/'
    </script>
</head>

<body style='margin: 0; overflow: hidden;'>
    <a-scene
        vr-mode-ui="enabled: false"
              embedded
        arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960;
displayWidth: 1280; displayHeight: 960; debugUIEnabled: false;'>

        <a-camera gps-camera rotation-reader></a-camera>
       </a-scene>
</body>
```

Este es su punto de entrada, la aplicación de plantilla para una aplicación AR.js basada en ubicación. Dentro de la etiqueta **head** puede ver las importaciones de **aframe, AR.js** (que contiene el código basado en ubicación de v2) y la biblioteca de **donmccurdy** para manejar la animación de modelos 3D. Necesitaremos este último cada vez que queramos animar un modelo 3D.

Añade tu primer lugar

Ahora es el momento de definir un lugar de interés. Hay un consejo importante para recordar:

Este tipo de Realidad Aumentada tiene más sentido en exteriores, donde la señal del GPS es buena y el usuario puede moverse sin restricciones.

Si desea realizar algunos experimentos en interiores, intente abrir una aplicación como Google Maps, active los datos del GPS en su teléfono y vea si la aplicación puede recuperar su posición. Si es así, puedes usarlo dentro de los edificios; de lo contrario, tendrás que moverte al exterior para probar tu código. De todos modos, la señal del GPS se refleja en las paredes e incluso si parece funcionar en interiores, los resultados no deben considerarse confiables.

Volvamos al código. En el fragmento anterior puedes ver el elemento **a-scene**, el contenedor de nuestra aplicación AR, y **gps-camera** el elemento que calcula la rotación, la posición del usuario y la distancia entre el usuario y los lugares.

Al agregar un lugar, decidimos contextualmente qué contenido mostrar cuando movemos nuestra cámara en su dirección. En este tutorial, mostraremos un modelo 3D: usaremos un modelo gratuito que representa al Pokémon Magnemite, puedes encontrarlo en esta dirección (https://raw.githack.com/nicolocarpignoli/location-based-ar-tutorial/master/static-on-html/assets/magnemite.zip) . Se descarga, se descomprime y se crea una carpeta llamada 'assets' en la misma carpeta de tu archivo index.html.

La estructura debería quedar así:

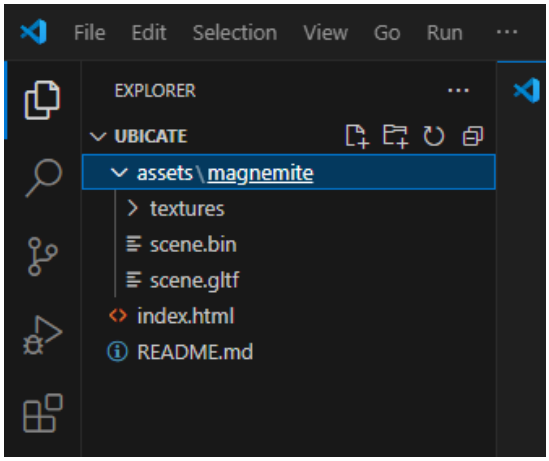

Figura 1 Estructura de archivos.

Dentro de la etiqueta **a-scene**, agregue lo siguiente:

```
<a-entity gltf-model="./assets/magnemite/scene.gltf" rotation="0 180 0" scale="0.15 0.15 0.15" gps-entity-place="longitude: 12.489820; latitude: 41.892590;" animation-mixer/>
```

El atributo **scale** se utiliza porque ese modelo es bastante grande y la rotación personalizada hará que el modelo "mire" hacia el usuario. El atributo **animation-mixer** le dice al modelo que use su animación incorporada.

Usando los atributos **gps-entity-place**, especificamos coordenadas geoespaciales.La recomendación es utilizar las coordenadas que el usuario requiera y eliminar las coordenadas que el código trae por defecto.

Ejecútelo en la Web

Usaremos la función **Github** y **Github Pages**. Es gratis y sencillo. En primer lugar, hay que registrarse en **Github.com**, el sitio web más utilizado que gestiona

repositorios de Código Abierto. Luego, debe crear un nuevo repositorio, podemos llamarlo 'tutorial de arte basado en la ubicación'. Después de eso, debe configurar git (si aún no lo ha hecho) en su máquina para poder enviarlo al repositorio remoto de Github.

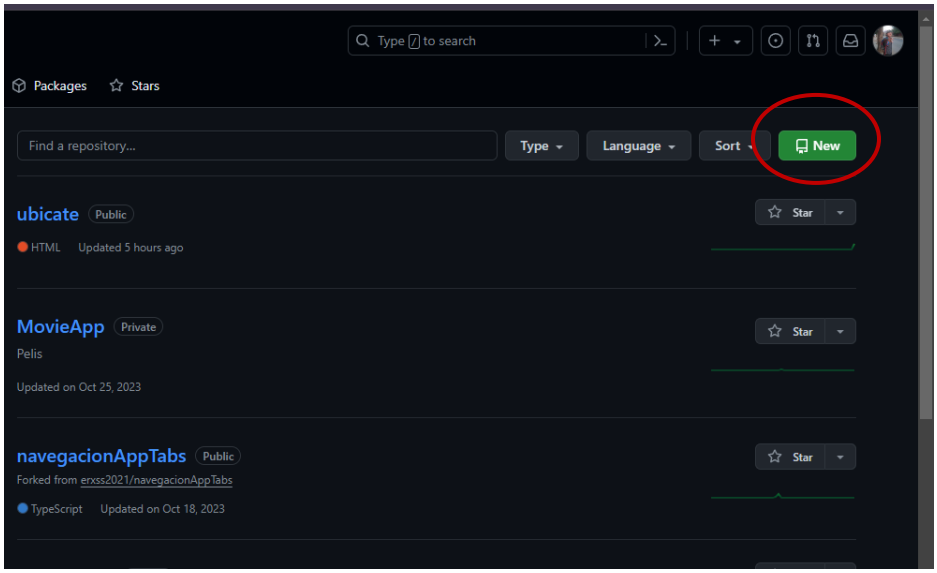

Figura 2 Creación de repositorio.

Para efectos de esta práctica se requiere también configurar git en la computadora y clonar el proyecto para realizar los cambios pertinentes tanto de manera local como remota.

Una vez hecho esto se debe implementar el código en una dirección pública remota. Para llevar esto a cabo utilizaremos **GithubPages**.

Crear un archivo de índice.
Nos dirigimos a GitHub.com y crea un nuevo repositorio o ve a uno existente.

Asignamos un nombre al archivo index.htmly escribimos contenido HTML en el editor.

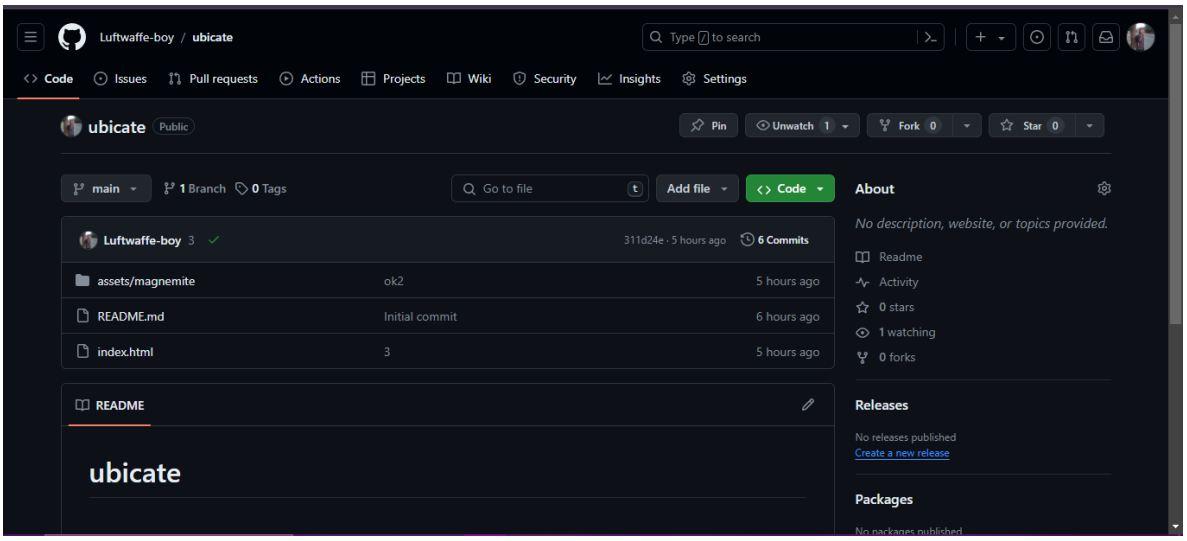

Figura 3 Estructura de archivos en GitHub.

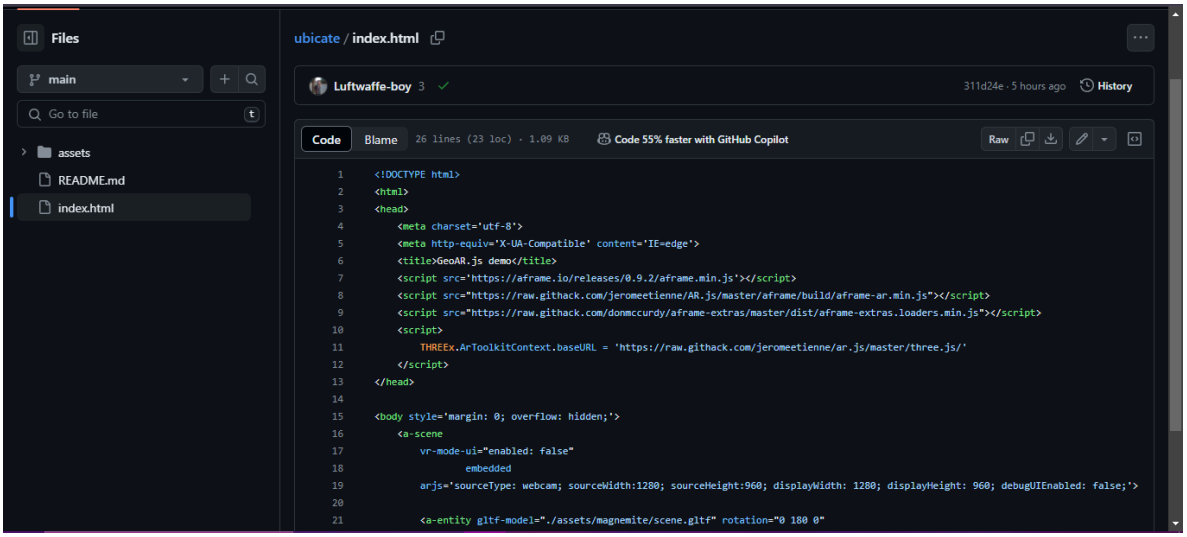

Figura 4 Archivo index.html.

Configuración del repositorio.

Hacemos clic en la pestaña Configuración y nos desplazamos hacia abajo hasta la sección Páginas de GitHub. Luego seleccionamos la fuente **de la rama principal** y haga clic en el botón **Guardar**.

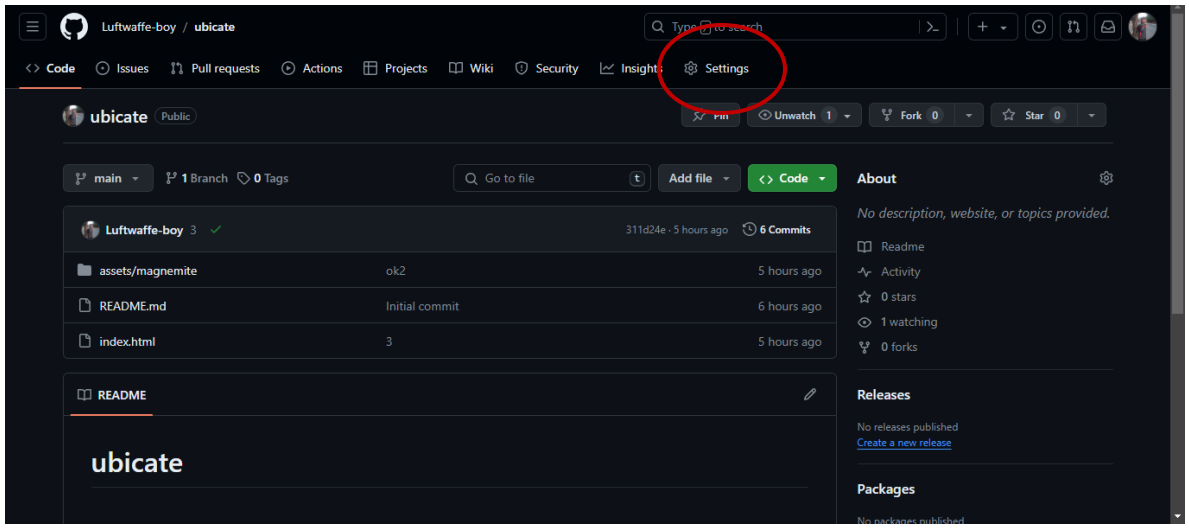

Figura 5 Menú de configuración.

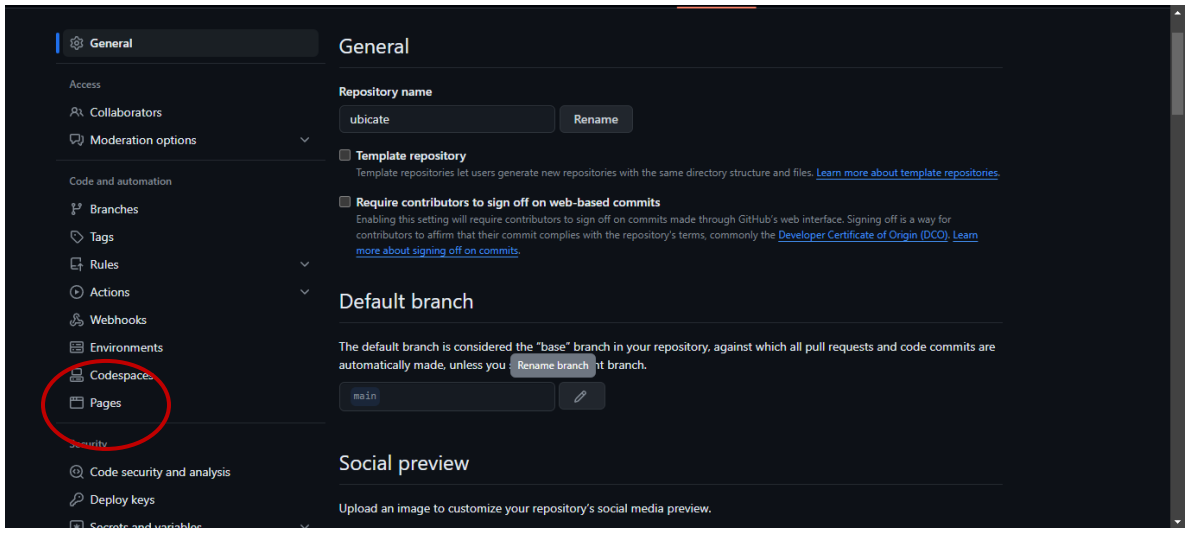

Figura 6 Menú de páginas.

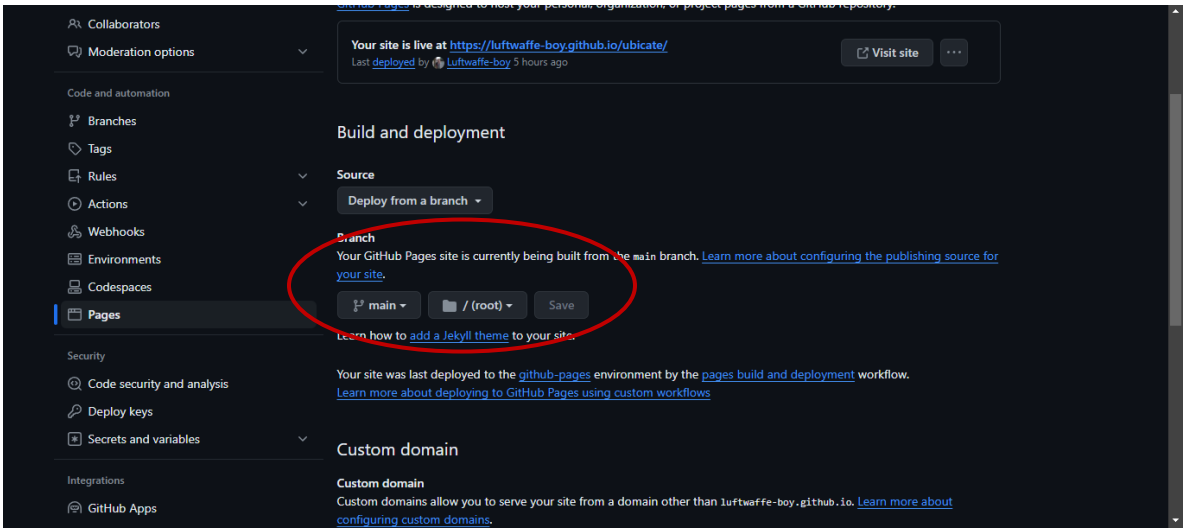

Figura 7 Configuración de la página.

Finalmente iniciamos un navegador y nos dirigimos a **http://** ***nombre_de_usuario*****.**github.io***/repositorio***.

Asegúrate de navegar utilizando el protocolo https para evitar problemas de seguridad con los navegadores.

Debes tener en cuenta que la página web solicita permisos de cámara y permisos de geolocalización, concedemos ambos y deberíamos ver una Magnemite salvaje moviéndose cerca de nuestra posición.

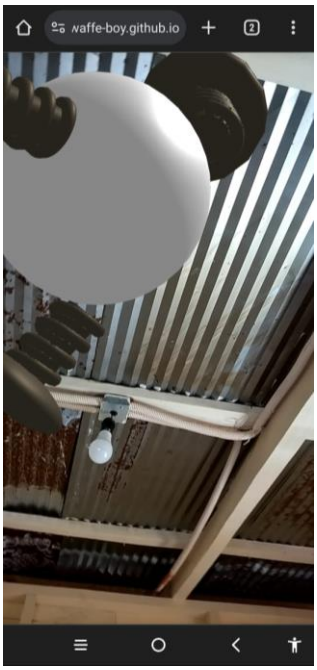

Figura 8 Prueba de ejemplo.

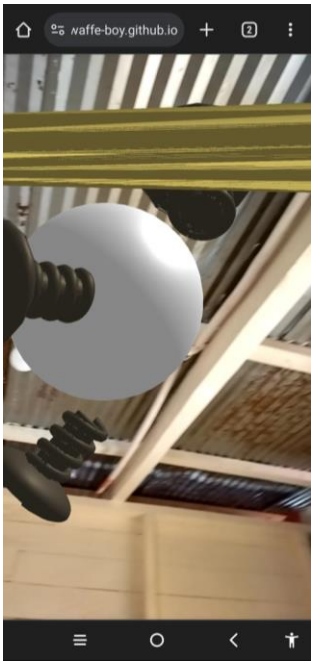

Figura 9 Prueba de ejemplo 2.

Segunda forma: agregar lugares estáticamente usando Javascript.

Al utilizar Javascript, obtendremos un enfoque imperativo, mantendremos el HTML más limpio y podremos hacer mucho más.

Por ejemplo, podríamos cargar muchos lugares iterando a través de una matriz y agregarlos mediante programación, pero lo más importante es que podemos brindar interacción y un poco de UX/UI a nuestra aplicación.

Agregar lugares desde Javascript

Como primer paso, limpiaremos nuestro archivo HTML y agregaremos lugares a través de Javascript. Terminaremos con el mismo comportamiento anterior con los siguientes archivos (siguen index.html y script.js):

Para el archivo index.html usaremos el siguiente código:

```
<!DOCTYPE
html>
        <html>
        <head>
            <meta charset='utf-8'>
            <meta http-equiv='X-UA-Compatible' content='IE=edge'>
            <title>GeoAR.js demo</title>
            <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
            <script
        src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-
        ar.min.js"></script>
            <script src="https://raw.githack.com/donmccurdy/aframe-
        extras/master/dist/aframe-extras.loaders.min.js"></script>
            <script>
                THREEx.ArToolkitContext.baseURL =
        'https://raw.githack.com/jeromeetienne/ar.js/master/three.js/'
            </script>
        </head>

        <script src="./script.js"></script>
        <body style='margin: 0; overflow: hidden;'>
            <a-scene
               vr-mode-ui="enabled: false"
                      embedded
                arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960;
        displayWidth: 1280; displayHeight: 960; debugUIEnabled: false;'>
                <a-camera gps-camera rotation-reader>
```

```
                    </a-camera>
                </a-scene>
            </body>
```

Para el archivo script.js usaremos el siguiente código:

```
window.onload
= () => {
                 let places = staticLoadPlaces();
                 renderPlaces(places);
            };

            function staticLoadPlaces() {
                return [
                    {
                        name: 'Magnemite',
                        location: {
                            lat: 44.496470,
                            lng: 11.320180,
                        }
                    },
                ];
            }

            function renderPlaces(places) {
                let scene = document.querySelector('a-scene');

                places.forEach((place) => {
                    let latitude = place.location.lat;
                    let longitude = place.location.lng;

                    let model = document.createElement('a-entity');
                    model.setAttribute('gps-entity-place', `latitude: ${latitude};
            longitude: ${longitude};`);
                    model.setAttribute('gltf-model', './assets/magnemite/scene.gltf');
                    model.setAttribute('rotation', '0 180 0');
                    model.setAttribute('animation-mixer', '');
                    model.setAttribute('scale', '0.5 0.5 0.5');

                    model.addEventListener('loaded', () => {
                        window.dispatchEvent(new CustomEvent('gps-entity-place-loaded'))
                    });

                    scene.appendChild(model);
```

```
            });
        }
```

Al observar la función **staticLoadPlaces**, podemos ver que los lugares se cargan estáticamente mediante una matriz. Podemos intentar agregar más lugares y se verán mientras se mueve la cámara si no están demasiado lejos de tu posición. Si están lejos pero no demasiado se verán pequeños, si están más cerca se verán más grandes.

Sin embargo, para los efectos de este ejemplo, utilizaremos solo un lugar.

La interacción del usuario.

Ahora, usando **Javascript aframe** podemos agregar interacción básica del usuario.

Primero, tenemos que reemplazar todo el index.html con el siguiente código:

```
<!DOCTYPE
html>
        <html>

        <head>
            <meta charset='utf-8'>
            <meta http-equiv='X-UA-Compatible' content='IE=edge'>
            <title>GeoAR.js demo</title>
            <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
            <script
        src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-
        ar.min.js"></script>
            <script src="https://raw.githack.com/donmccurdy/aframe-
        extras/master/dist/aframe-extras.loaders.min.js"></script>
            <script>
                THREEx.ArToolkitContext.baseURL =
        'https://raw.githack.com/jeromeetienne/ar.js/master/three.js/'
            </script>
            <script src="./script.js"></script>
            <link rel="stylesheet" type="text/css" href="./style.css"/>
        </head>

        <body style='margin: 0; overflow: hidden;'>
            <div class="centered instructions"></div>
            <a-scene
                vr-mode-ui='enabled: false'
                embedded
```

```
        arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960;
displayWidth: 1280; displayHeight: 960; debugUIEnabled: false;'>
    <a-camera gps-camera rotation-reader></a-camera>
</a-scene>
<div class="centered">
    <button data-action="change"></button>
</div>
</body>
```

Importamos un script, una hoja de estilo, agregamos un botón y un **div** vacío. Eso es todo por el lado HTML. También debemos crear la nueva hoja de estilo llamada **'style.css'** en la misma carpeta de index.html.

La idea es agregar un botón con un controlador de eventos de clic, que cambiará nuestro Pokémon cada vez que interactuemos con él. Encontraremos tres modelos 3D gratuitos, uno es el Magnemite que ya usamos y los otros dos se pueden encontrar en el siguiente enlace: https://github.com/nicolocarpignoli/location-based-ar-tutorial/tree/master/static- lugares/activos .

Entonces, antes que nada, descargamos los activos y los colcamos en la carpeta "assets". Terminará con la siguiente estructura de archivos:

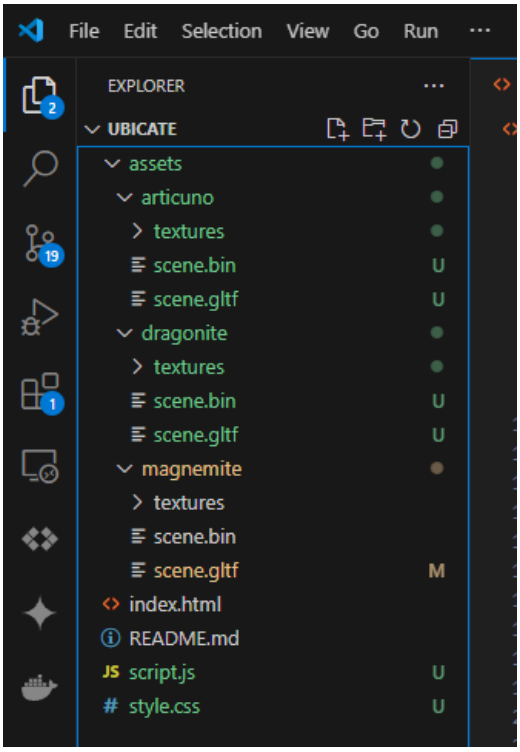

Figura 10 estructura de archivos.

Ahora, completamos la hoja de estilo con las siguientes reglas CSS:

```
.centered
{
            height: 20%;
            justify-content: center;
            position: fixed;
            bottom: 0%;
            display: flex;
            flex-direction: row;
            width: 100%;
            margin: 0px auto;
            left: 0;
            right: 0;
        }

        button {
            display: flex;
            align-items: center;
            justify-content: center;
            border: 2px solid white;
            background-color: transparent;
            width: 2em;
            height: 2em;
            border-radius: 100%;
            font-size: 2em;
            background-color: rgba(0, 0, 0, 0.4);
            color: white;
            outline: none;
        }

        .instructions {
            position: fixed;
            top: 5%;
            font-size: 1.25em;
            color: white;
            z-index: 999999;
        }
```

Y ahora agregamos comportamiento con Javascript. Cosas a tener en cuenta en el siguiente guión:

- Agregaremos el lugar de interés después de cargar la ventana principal.

- Agregaremos un detector de clics en el botón que llama a la función **'setModel'**.
- En la función 'setModel' iteraremos a través de una serie de modelos: cada uno de ellos especifica un contenido textual para el DIV superior, la URL del modelo 3D y propiedades personalizadas, esto porque no todos los modelos que puedes encontrar en línea tendrán la misma rotación, orientación y tamaño. Les damos algún tipo de 'configuración' para tener una vista consistente para cada uno de ellos.

Puedes cambiar muchos atributos: intenta ajustarlos según tu gusto personal. Para cada propiedad, el triple de valores representa la propiedad expresada en las coordenadas X, Y y Z. La rotación se expresa en grados, la posición en metros y la escala es relativa a la escala inicial (el valor predeterminado es 1).

Colocamos el siguiente código en el archivo **script.js**:

```
window.onload
= () => {
                const button = document.querySelector('button[data-action="change"]');
                button.innerText = ' ? ';

                let places = staticLoadPlaces();
                renderPlaces(places);
            };

            function staticLoadPlaces() {
                return [
                    {
                        name: 'Pokèmon',
                        location: {
                            // lat: <your-latitude>,
                            // lng: <your-longitude>,
                        },
                    },
                ];
            }

            var models = [
                {
                    url: './assets/magnemite/scene.gltf',
                    scale: '0.5 0.5 0.5',
                    info: 'Magnemite, Lv. 5, HP 10/10',
                    rotation: '0 180 0',
                },
```

```
    {
        url: './assets/articuno/scene.gltf',
        scale: '0.2 0.2 0.2',
        rotation: '0 180 0',
        info: 'Articuno, Lv. 80, HP 100/100',
    },
    {
        url: './assets/dragonite/scene.gltf',
        scale: '0.08 0.08 0.08',
        rotation: '0 180 0',
        info: 'Dragonite, Lv. 99, HP 150/150',
    },
];

var modelIndex = 0;
var setModel = function (model, entity) {
    if (model.scale) {
        entity.setAttribute('scale', model.scale);
    }

    if (model.rotation) {
        entity.setAttribute('rotation', model.rotation);
    }

    if (model.position) {
        entity.setAttribute('position', model.position);
    }

    entity.setAttribute('gltf-model', model.url);

    const div = document.querySelector('.instructions');
    div.innerText = model.info;
};

function renderPlaces(places) {
    let scene = document.querySelector('a-scene');

    places.forEach((place) => {
        let latitude = place.location.lat;
        let longitude = place.location.lng;

        let model = document.createElement('a-entity');
```

```
        model.setAttribute('gps-entity-place', `latitude: ${latitude};
longitude: ${longitude};`);

        setModel(models[modelIndex], model);

        model.setAttribute('animation-mixer', '');

        document.querySelector('button[data-
action="change"]').addEventListener('click', function () {
            var entity = document.querySelector('[gps-entity-place]');
            modelIndex++;
            var newIndex = modelIndex % models.length;
            setModel(models[newIndex], entity);
        });

        scene.appendChild(model);
    });
}
```

Importante: antes de continuar, en la función **staticLoadPlaces()** del script.js se debe establecer coordenadas de lugar válidas, como se hizo en los ejemplos anteriores.

El resultado debería ser una aplicación Web AR muy básica con una UX/UI mínima con un solo botón, que cambia de Pokémon cuando se hace clic. Las posiciones de los Pokémon deben ser aproximadamente las mismas después de cada cambio. Algunos de ellos se mueven, otros no, de acuerdo con su animación incorporada.

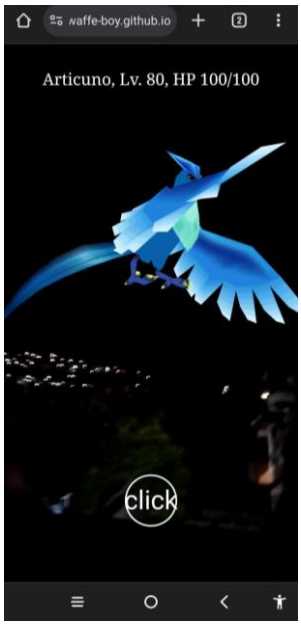

Figura 11 RA Articuno.

Figura 12 RA Magnemite.

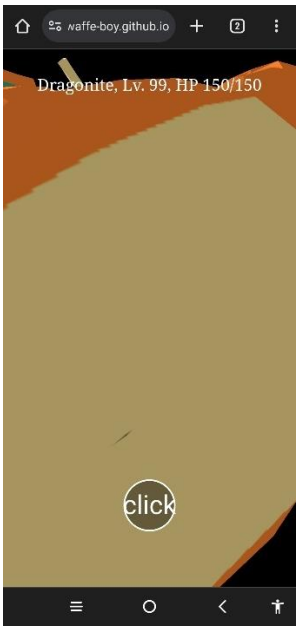

Figura 13 RA Dragonite.

Tercera forma: agregar lugares dinámicamente usando Javascript.

Cuando agrega estáticamente lugares de interés en su aplicación web, **debe conocer su posición en el momento del desarrollo**. ¿Qué sucede si implementas tu aplicación y le das la URL a un amigo o la compartes en las redes sociales? Si los usuarios no están cerca de la posición que usted estableció arbitrariamente, no verán nada. Sólo la transmisión de vídeo de la cámara.

No puede saber mediante programación el lugar donde estarán sus usuarios cuando abran su aplicación web. Y no debería.

La idea es recuperar primero la posición de los usuarios y luego cargar dinámicamente los lugares de interés cercanos a ellos. Para hacer eso, necesitaremos API externas. Existe un buen servicio en la API de Foursquare Places. Para este ejemplo, los usaremos, pero puedes optar por usar también algo más: en ese caso, solo cambiará la función de recuperación de datos.

Regístrese en las API de Foursquare y obtenga credenciales.
Navegue a https://developer.foursquare.com y haga clic en 'Crear cuenta'. Es gratis, por supuesto. Después del proceso, recibirás un correo electrónico de confirmación. Abra el correo electrónico, haga clic en el enlace e inicie sesión nuevamente. Ahora estás listo para crear tu primera 'aplicación'.

Haga clic en 'Crear nueva aplicación'. En el 'Nombre de la aplicación' puede escribir algo como 'ubicación-basada-ar' y en 'URL de la empresa' configuro 'https://localhost:3000', la URL de mi entorno local.

En la página siguiente, tome nota de los valores de ID DE CLIENTE y SECRETO DE CLIENTE. Los necesitarás más tarde. Usaremos, en nuestra aplicación, la 'API de lugares'; de ser necesario puede buscar la documentación.

Busquemos algunos datos

Para este ejemplo, podemos reutilizar nuestro index.html limpio del ejemplo anterior, sin una interfaz de usuario personalizada, y cambiar la forma en que agregamos lugares en el script.

Hay una configuración que te sugiero agregar: en **gps-camera** el elemento, es mejor agregar un atributo llamado **minDistance**. Le dice a AR.js que oculte elementos que están demasiado cerca del usuario. Lo que significa "demasiado cerca" se especifica con el valor del atributo, así: gps-camera="minDistance: 40;"esto significa "no mostrar elementos que estén a 40 metros o menos cerca del usuario".

Cuando nos desplazamos por ubicaciones con muchos lugares de interés, sugiero no tener demasiado contenido en pantalla: si estamos muy cerca de un lugar de interés, es extraño ver el contenido AR tan grande y tan cerca. a nosotros. En tales casos, es mejor ofrecer un tipo diferente de experiencia (mostrar una alerta, un mensaje para el usuario, etc.).

También puedes eliminar la carpeta de activos 3D y su contenido, no los necesitaremos. Finalmente, el index.html tendrá el siguiente aspecto:

```
<!DOCTYPE
html>
        <html>

        <head>
            <meta charset='utf-8'>
            <meta http-equiv='X-UA-Compatible' content='IE=edge'>
            <title>GeoAR.js demo</title>
            <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
            <script
        src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-
        ar.min.js"></script>
            <script>
                THREEx.ArToolkitContext.baseURL =
        'https://raw.githack.com/jeromeetienne/ar.js/master/three.js/'
            </script>
```

```
        <script src="./script.js"></script>
    </head>

    <body style='margin: 0; overflow: hidden;'>
        <a-scene
            vr-mode-ui='enabled: false'
            embedded
            arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960;
    displayWidth: 1280; displayHeight: 960; debugUIEnabled: false;'>
        <a-camera gps-camera="minDistance: 40;" rotation-reader></a-camera>
    </a-scene>
    </body>
```

La idea es mostrar un icono y un nombre para cada lugar de interés cercano al usuario. Los lugares se agregan dinámicamente.

Ahora recuperemos la posición del usuario y busquemos algunas ubicaciones. Mostraremos, para cada lugar, un 'enlace a'. Es una entidad personalizable que muestra un texto y, si se hace clic, puede redirigir al usuario a una página web externa, si se especifica con el atributo 'href'. Puedes personalizar su aspecto visual, el predeterminado es un círculo rojo. Consulte la documentación para saber cómo hacerlo.

```
//
getting
places
from
APIs
    function loadPlaces(position) {
        const params = {
            radius: 300,    // search places not farther than this value (in meters)
            clientId: '<YOUR-CLIENT-ID>',
            clientSecret: 'YOUR-CLIENT-SECRET',
            version: '20300101',    // foursquare versioning, required but unuseful
    for this demo
        };

        // CORS Proxy to avoid CORS problems
        const corsProxy = 'https://cors-anywhere.herokuapp.com/';

        // Foursquare API (limit param: number of maximum places to fetch)
```

```
    const endpoint =
`${corsProxy}https://api.foursquare.com/v2/venues/search?intent=checkin
        &ll=${position.latitude},${position.longitude}
        &radius=${params.radius}
        &client_id=${params.clientId}
        &client_secret=${params.clientSecret}
        &limit=30
        &v=${params.version}`;
    return fetch(endpoint)
        .then((res) => {
            return res.json()
                .then((resp) => {
                    return resp.response.venues;
                })
        })
        .catch((err) => {
            console.error('Error with places API', err);
        })
};


window.onload = () => {
    const scene = document.querySelector('a-scene');

    // first get current user location
    return navigator.geolocation.getCurrentPosition(function (position) {

        // than use it to load from remote APIs some places nearby
        loadPlaces(position.coords)
            .then((places) => {
                places.forEach((place) => {
                    const latitude = place.location.lat;
                    const longitude = place.location.lng;

                    // add place name
                    const placeText = document.createElement('a-link');
                    placeText.setAttribute('gps-entity-place', `latitude:
${latitude}; longitude: ${longitude};`);
                    placeText.setAttribute('title', place.name);
                    placeText.setAttribute('scale', '15 15 15');

                    placeText.addEventListener('loaded', () => {
```

```
                        window.dispatchEvent(new CustomEvent('gps-entity-place-
loaded'))
                    });

                    scene.appendChild(placeText);
                });
            })
    },
        (err) => console.error('Error in retrieving position', err),
        {
            enableHighAccuracy: true,
            maximumAge: 0,
            timeout: 27000,
        }
    );
};
```

Esto no funcionará todavía, deberá agregar su ID DE CLIENTE y sus datos SECRETOS DE CLIENTE en las líneas correspondientes.

Ahora, intenta confirmar, presionar y visitar nuevamente la URL de github.io para tu aplicación. Tenga en cuenta que las URL de las páginas de Github.io siguen la estructura de su proyecto: si ha agregado este nuevo ejemplo en una carpeta diferente, deberá visitarhttps://<your-username>/<your-repository>/<your-folder>/

Finalmente, los lugares de interés podrán apreciarse con puntos rojos en la pantalla.

CONCLUISIÓN.

La práctica realizada nos proporcionó conceptos y métodos que ampliaron nuestro margen de conocimiento relacionado a la realidad aumentada, aportó conceptos útiles incluso sorprendentes y muy interesantes de la implementación de la realidad aumentada utilizando coordenadas geoespaciales con herramientas como el GPS, algunas líneas de código y una agradable experiencia de poder incrustar material de realidad aumentada con figuras acorde al gusto del desarrollador prácticamente en cualquier lugar que se desee. A grandes rasgos resulta sorprendente la cantidad de áreas para la aplicación de la realidad aumentada y la capacidad de reinvención y diversidad de tópicos desde el entretenimiento hasta aspectos científicos, informativos, turísticos y un gran etcétera. Sin lugar a dudas la realidad aumentada es una tecnología cuya tendencia es propia de adquirir una potencial aceptación y desarrollo para el futuro incluso incrustando material más realista que pueda facilitar el interés de público de todas la edades y gustos.