UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Aplicación de Realidad Mixta para la representación visual de datos obtenidos mediante el uso de sensores y dispositivos IoT

Proyecto creado por Carlos Brito Pérez

Supervisado por:
Tutor: Agustín Rafael Trujillo Pino
Co-Tutor de empresa: Antonio José Sánchez López

15/02/2023

ULPGC Universidad de Las Palmas de Gran Canaria | Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D./Dª Carlos Brito Pérez, autor del trabajo de Fin de Título Aplicación de Realidad Mixta para la representación visual de datos obtenidos mediante el uso de sensores y dispositivos IoT, correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa Singular Factory.

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[X] se autoriza / [ ] no se autoriza la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).

[X] No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/la estudiante

Fdo.————————

A rellenar y firmar **obligatoriamente** por el/la/los/las tutores

En relación con la presente solicitud, se informa:(firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (justificación en TFT05)
Fdo.————————	Fdo.————————

DIRECTORA DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

A mi familia, que siempre ha estado ahí para mí, confiando y apoyándome tanto en los buenos como en los malos momentos, dándome las fuerzas para continuar cuando más las necesitaba.

A mis tutores, Agustín y Antonio, por ayudarme y guiarme para realizar este TFG.

A la empresa Singular Factory por ofrecerme hacer este TFG y facilitarme los recursos para realizarlo.

Y a mis compañeros que me han acompañado en esta aventura, con los que he sufrido y reído a partes iguales, sin ellos no hubiese sido lo mismo. En especial a Christian, Jorge, Pablo, Pablo, Pablo y Pablo entre muchos otros.

Abstract

This project consists in the creation of a mixed reality application to observe and monitor different meteorological factors in a specific area.

The application has a user interface to be able to choose if you want to observe the map of the area, the sensors located in it the data collected by each sensor or if you want to observe a heat map created from one of the different types of data (temperature, humidity, pressure...) collected by the sensors. It is also possible to change the type of data used to create the heat map.

The data of the sensors and the heatmap updates in regular intervals with new data.

Resumen

Este proyecto se basa en la creación de una aplicación de realidad mixta, en la que poder observar y monitorizar diferentes factores meteorológicos de una zona específica.

La aplicación posee una interfaz de usuario para poder elegir si se quiere observar el mapa de la zona y los sensores situados en ella, pudiendo observar los datos recogidos por cada sensor, o si quiere observar un mapa de calor creado a partir de uno de los tipos de datos (temperatura, humedad, presión...) recogidos por lo sensores. También se permite cambiar el tipo de dato que se usa para la creación del mapa de calor.

Los datos de los sensores y los mapas de calor se actualizan cada un tiempo determinado con nuevos datos.

Glosario

Hardware. Parte física y componentes del ordenador.

Software. Programa informático.

Script. Archivo de texto con código.

Librería. Archivos de código de terceros que cumplen funciones especificas.

Asset. Los objetos o recursos de un proyecto de Unity

Escena. Espacio en el que se agrupan objetos en Unity

GameObject. Objectos pertenecientes a una escena en Unity

Prefab. Un gameObject creado y almacenado para poder ser usado varias veces.

Grid. Cuadrícula o red.

Json. Archivo de texto formateado para archivar información de manera estandarizada.

C#. Lenguaje de programación de alto nivel orientado a objetos.

Interpolación. Obtención de los valores de unos puntos partiendo de unos ya conocidos.

Índice general

1. Introducción **1**
1.1. Introducción . . . 1
1.2. Estado del arte . . . 1
1.3. Objetivos . . . 3
1.4. Competencias cubiertas . . . 4

2. Herramientas usadas **5**
2.1. Hardware . . . 5
2.1.1. Ordenador personal . . . 5
2.1.2. Gafas de realidad Virtual . . . 5
2.2. Software . . . 6
2.2.1. Unity3D . . . 6
2.2.2. Microsoft Visual Studio . . . 6
2.2.3. TextMeshPro . . . 7
2.2.4. InfinityCode: OnlineMaps . . . 7
2.2.5. MRTK: Mixed Reality Toolkit . . . 7
2.2.6. GitLab . . . 8
2.2.7. GitHub . . . 8
2.2.8. SourceTree . . . 8
2.2.9. Overleaf . . . 9
2.2.10. ClickUp . . . 9
2.2.11. StarUML . . . 10

3. Análisis **11**
3.1. Requisitos . . . 11
3.2. Casos de uso . . . 12
3.3. Historias de usuario . . . 12
3.4. Estudio previo . . . 13

4. Diseño y metodología **14**
4.1. Diseño . . . 14
4.1.1. Diseño del Proyecto . . . 14
4.1.2. Diseño de los botones . . . 14
4.1.3. Vinculación con los sensores . . . 18

4.2. Metodología 19
4.2.1. Metodología Ágil 19

5. Implementación y Modelos 21
5.1. Proyecto en Unity 21
5.1.1. Assets 21
5.1.2. Scripts 23
5.1.3. Escena 24
5.2. Recolección de Datos 25
5.2.1. Tratamiento de los JSONs 25
5.3. Interfaz de Usuario 28
5.3.1. Inicio de la Aplicación 28
5.3.2. Botón de Inicio 28
5.3.3. Menú Principal 30
5.3.4. Menú Mapa de Calor 33
5.4. Representación de Datos 37
5.4.1. Mapa y Sensores 37
5.4.2. Panel con los datos del sensor 40
5.4.3. Representación del Mapa de Calor 41
5.5. Cambios en modelos y diseño 42

6. Cálculos del Mapa de Calor 46
6.1. Introducción 46
6.2. Retos a Afrontar 46
6.3. Cuadrícula de Valores 47
6.4. Cálculo de la Envolvente Convexa 49
6.5. Cálculo de la Cuadrícula de Valores 54
6.6. Cálculo de la Altura y el Color 60

7. Conclusiones y trabajo a futuro 63
7.1. Conclusiones 63
7.2. Trabajo a futuro 64

8. Anexo I: Manual de Instalación 65

9. Anexo II: Manual de Usuario 66

Índice de figuras

1.1. Uso de tecnología VR en el ejercito. 2
1.2. Uso de tecnología VR en como método de enseñanza. 2
1.3. Uso de tecnología AR para reuniones online. 3

2.1. Gafas de realidad aumentada, HoloLens. 6
2.2. Logo Unity. 6
2.3. Logo VisualStudio. 7
2.4. Logo InfinityCode. 7
2.5. Logo Mixed Reality Toolkit. 8
2.6. Logo GitLab. 8
2.7. Logo GitHub. 8
2.8. Logo SourceTree. 9
2.9. Logo Overleaf. 9
2.10. Logo ClickUp. 9
2.11. Logo StarUML. 10

3.1. Diagrama de Casos de Uso. 12

4.1. Jerarquía del MenuButton. 15
4.2. Modelo del botón de Start. 16
4.3. Modelo del botón MenuButton. 16
4.4. Modelo del botón de cambio de modo. 17
4.5. Modelo del botón HeatMapMenuButton. 18
4.6. Organización de las tareas en diferentes grupos usando la aplicación ClickUp. 19

5.1. Todos los assets del proyecto. 22
5.2. Scripts del proyecto. 23
5.3. Jerarquía de la escena de Unity. 25
5.4. Fichero Json con los datos de los sensores. 26
5.5. Modelo de los datos en código. 27
5.6. Tablero con los ejes cardinales. 28
5.7. Botón de Start. 29
5.8. Menú principal de la aplicación. 30
5.9. Mapa topográfico de la isla de Gran Canaria. 31
5.10. Panel de información de los sensores. 31

5.11. Mapa de calor. 32
5.12. Panel de información de una celda del mapa de calor. 32
5.13. Mapa de calor con el tipo de dato temperatura. 33
5.14. Mapa de calor con el tipo de dato humedad. 34
5.15. Menú de cambio de modo del mapa de calor. 35
5.16. Mapa de calor en modo 3D. 36
5.17. Mapa de calor en modo 2D. 37
5.18. Mapa sin los sensores. 38
5.19. Modelo de un sensor. 39
5.20. Jerarquía del modelo del sensor en Unity. 40
5.21. Panel de información de un sensor. 41
5.22. Celda del mapa de calor. 42
5.23. Modelo antiguo del los sensores. 43
5.24. Modelo nuevo de los sensores. 44
5.25. Diseño antiguo del menú. 45
5.26. Diseño nuevo del menú. 45

6.1. Representación visual de la cuadrícula. 48
6.2. Matriz de datos. 48
6.3. Representación de la envolvente convexa del conjunto de puntos (X). 49
6.4. Código del método LeftMostPoint(). 50
6.5. Algoritmo GiftWrapping. 51
6.6. Código del método CalculateConvecHull() con el while() contraído. 52
6.7. Código del método FillConvexHull(). 53
6.8. Código del método CalculateConvecHull() con el for() contraido. 53
6.9. Código del bucle for() del método CalculateConvecHull(). 54
6.10. Código del método CreateGridMap(). 55
6.11. Código del método CalculateGrid(). 55
6.12. Código del método GetSensorData(). 56
6.13. Código del método CalculateConvexHull(). 56
6.14. Código del método SetSensorsOnGrid(). 57
6.15. Código del método SetBaseValues(). 57
6.16. Código del método InterpolateValues(). 58
6.17. Código del método IsBasePoint(). 58
6.18. Código del método IsPointInsideConvexHull(). 59
6.19. Código del método CalculateEuclidianDistance(). 59
6.20. Código del método GetInterpolatedValue(). 60
6.21. Código del método ConsoleMatrix(). 60
6.22. Código del método CalculateHeightScale(). 61
6.23. Código del método CalculateTileHeight(). 62

9.1. Marco del mapa con los ejes cardinales. 66
9.2. Botón de inicio del menú. 67
9.3. Menú principal de la aplicación. 68
9.4. Mapa topográfico de la isla de Gran Canaria. 69

9.5. Panel de información de los sensores. 69
9.6. Mapa de calor con el menú desplegado. 70
9.7. Panel de información de las celdas del mapa de calor. 70
9.8. Menú principal con el menú desplegable del mapa de calor. 71

Índice de cuadros

3.1. Requisitos no funcionales de la aplicación. 11
3.2. Requisitos funcionales de la aplicación. 11

6.1. Tabla de complejidades de los algoritmos 49

Capítulo 1

Introducción

1.1. Introducción

La tecnología, y en especial la informática, siempre ha sido un campo que ha evolucionado a velocidades asombrosas, pero en las últimas décadas no ha hecho más que acelerar.

Como popularizó el filósofo inglés Thomas Hobbes en su obra Leviatán "El conocimiento es poder" y hoy en día esto es más cierto que nunca. Pero el conocimiento, o en este caso, la información no sirve de nada si no se usa, y para usarla hace falta poder acceder a ella de forma cómoda y eficiente. Y no hay forma más inmersiva para interactuar con ella que mediante la Realidad Virtual (VR) y la Realidad Aumentada o Mixta (AR).

Este proyecto nace ante la necesidad de representar de manera accesible y cómoda los datos recogidos por unos sensores repartidos por toda la isla y, como se mencionó antes, no hay nada mejor para esto que la Realidad Mixta.

Estos sensores recogen datos de diferentes factores climáticos, como son la temperatura, la presión o la humedad. Estos datos son interesantes por diversos motivos, y diferentes perfiles de personas pueden estar interesados en ellos, es por eso que este proyecto busca crear un entorno donde todos ellos puedan acceder de forma cómoda a estos.

1.2. Estado del arte

La tecnología de Realidad Virtual y Realidad Aumentada o Mixta lleva existiendo ya una cantidad considerable de años, y aunque no se ha conseguido asentar en las viviendas y en el uso cotidiano, cada día está más presente en diferentes sectores tan diversos como el ejército(Ilustración 1.1) o la enseñanza(Ilustración 1.2).

Ilustración 1.1: Uso de tecnología VR en el ejercito.

Ilustración 1.2: Uso de tecnología VR en como método de enseñanza.

Otro campo en el que la tecnología VR y AR tiene un futuro muy prometedor, es en el de las reuniones, otorgándoles un entorno más inmersivo y personal al poder ver a los demás participantes.

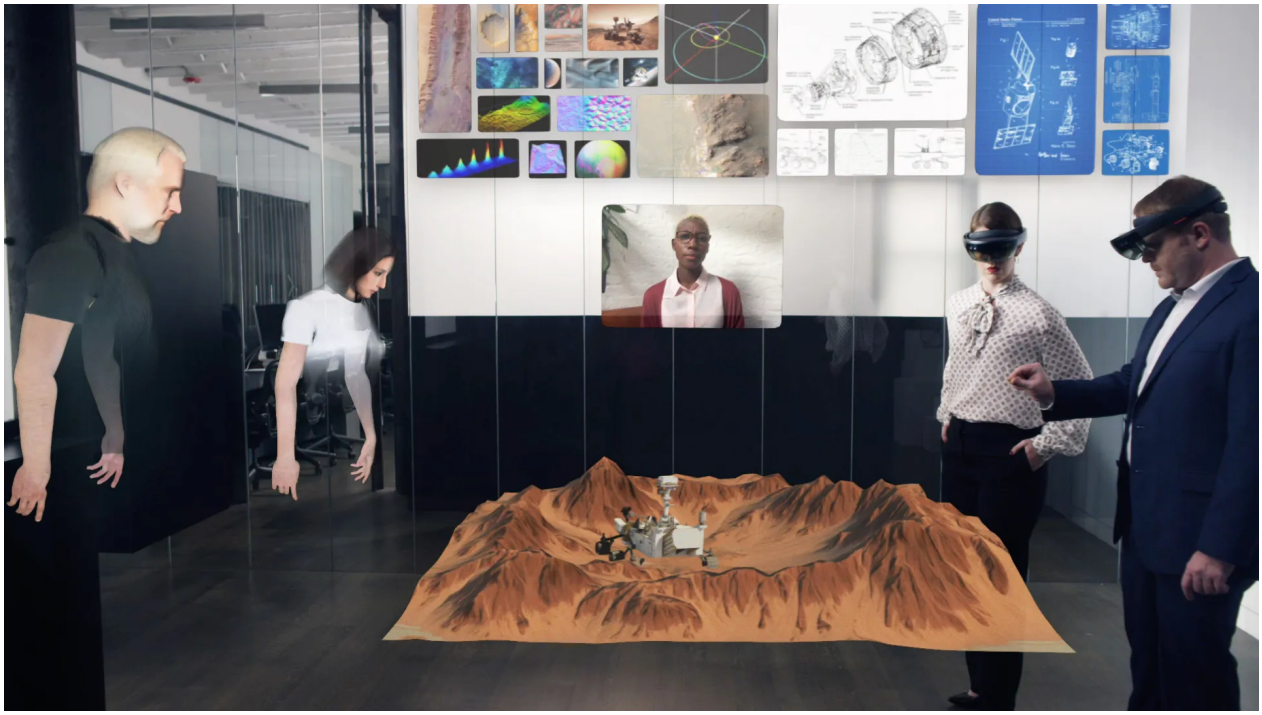

Ilustración 1.3: Uso de tecnología AR para reuniones online.

Los principales motivos que han limitado el crecimiento de esta tecnología han sido una mezcla entre unos precios elevados y un mercado aún en desarrollo que no alienta al usuario a apostar por ella.

Este proyecto intenta explorar las opciones que ofrece la realidad mixta para crear una aplicación que permita de forma interactiva e inmersiva, realizar una tarea como puede ser el control medioambiental de una zona. Este es un campo que ha sido poco explorado y tiene un mercado casi inexistente ya que está orientado a un grupo de personas bastante limitado.

1.3. Objetivos

Este trabajo surge con el objetivo de permitir la monitorización de factores meteorológicos sobre una zona determinada. Esto tiene diferentes utilidades, es un sistema que puede ser empleado tanto en el ámbito de la enseñanza como en el de la seguridad medioambiental.

En el ámbito de la enseñanza, este proyecto aspira a ser una forma de mostrar información de un área de una manera visual para apelar a públicos más jóvenes.

Por otro lado, en el ámbito de la seguridad medioambiental, el objetivo es ofrecer una manera de poder monitorizar de forma cómoda las condiciones meteorológicas, permitiendo de esta manera poder detectar e incluso prevenir incendios u otros tipos de desastres medioambientales.

1.4. Competencias cubiertas

- ***CII07.*** *Conocimiento, diseño y utilización de forma eficiente los tipos y estructuras de datos más adecuados a la resolución de un problema.*

  En el proyecto se han utilizado diferentes tipos de datos y estructuras tales como vectores, matrices y diccionarios entre otros. Buscando siempre el adecuado para cada ocasión.

- ***CII016.*** *Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.*

  La aplicación cuenta con un menú para controlarla y diferentes objetos del entorno con los que el usuario puede interactuar.

- ***CP03.*** *Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.*

  Durante el cálculo de de alguno de los apartados del proyecto hubo que investigar, modificar e implementar algoritmos tales como el GiftWrapping Algorithm para la Envolvente Convexa.

- ***CP06.*** *Capacidad para desarrollar y evaluar sistemas interactivos y de presentación de información compleja y su aplicación a la resolución de problemas de diseño de interacción persona computadora.*

  Como se mencionó en la Competencia CII016, la aplicación cuenta con varios sistemas interactivos, siendo uno de ellos un menú que controla todo el funcionamiento de la aplicación.

Capítulo 2

Herramientas usadas

Para la realización de este proyecto fue necesario el uso de varias herramientas tanto de hardware como de software, las principales fueron las siguientes.

2.1. Hardware

2.1.1. Ordenador personal

El proyecto ha sido realizado casi en su totalidad en mi ordenador personal, el cuál consta de las siguientes especificaciones:

- ***GPU:*** NVIDIA 1080Ti
- ***CPU:*** Intel i7-7700k
- ***RAM:*** 32Gb DDR4
- ***Almacenamiento:*** 2TB SDD

2.1.2. Gafas de realidad Virtual

El proyecto esta desarrollado para ser visualizado con unas gafas de realidad mixta, en este caso para las HoloLens(Ilustración 2.1), las cuales me fueron proporcionadas por la empresa colaboradora The Singular Factory.

Ilustración 2.1: Gafas de realidad aumentada, HoloLens.

2.2. Software

2.2.1. Unity3D

Unity3D[1] es un motor gráfico además de un entorno de desarrollo 3D, en el cual desarrollé la mayor parte de este proyecto. El lenguaje utilizado fue el nativo del entorno, C#.

Ilustración 2.2: Logo Unity.

2.2.2. Microsoft Visual Studio

VisualStudio[2] es un entorno de programación compatible con C# y Unity.

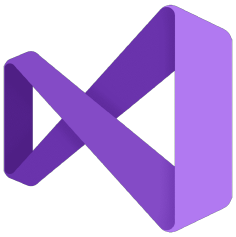

Ilustración 2.3: Logo VisualStudio.

2.2.3. TextMeshPro

TextMeshPro es una herramienta de Unity para la creación de textos en entornos 3D que permite un mayor control y variedad.

2.2.4. InfinityCode: OnlineMaps

OnlineMaps[3] es un software desarrollado por la empresa InfinityCode[4], permite trabajar con mapas 2D y 3D en Unity.

Ilustración 2.4: Logo InfinityCode.

2.2.5. MRTK: Mixed Reality Toolkit

MRTK[5] es un software open-source de Microsoft para trabajar en entornos tanto de realidad mixta como aumentada. Algunas de las funciones que ofrece son de reconocimiento de gestos e interacción con el entorno.

Ilustración 2.5: Logo Mixed Reality Toolkit.

2.2.6. GitLab

Usé GitLab[6] como mi aplicación principal de gestión de versiones, y para tener una copia del código en la nube.

Ilustración 2.6: Logo GitLab.

2.2.7. GitHub

Uno de los servidores de gestión de versiones más populares del mundo. A mitad de proyecto cambié de GitLab a GitHub[7], el GitLab es propiedad de la empresa colaboradora y para la entrega se decidió cambiarlo a mi GitHub personal.

Ilustración 2.7: Logo GitHub.

2.2.8. SourceTree

Utilicé la aplicación SourceTree[8] para realizar los commits, ya que proporciona una interfaz que permite realizarlos de forma mucho más cómoda.

Ilustración 2.8: Logo SourceTree.

2.2.9. Overleaf

Esta memoria fue desarrollada en el lenguaje LaTex, en la web Overleaf[9]. Al usar este lenguaje se consigue un mayor control de personalización de la memoria.

Ilustración 2.9: Logo Overleaf.

2.2.10. ClickUp

ClickUp[10] es una aplicación que permite llevar un seguimiento de las diferentes tareas del proyecto, y el estado en el que se encuentra cada una de ellas.

Ilustración 2.10: Logo ClickUp.

2.2.11. StarUML

StarUML[11] es un software que, entre otras funcionalidades, permite la creación de diagramas de casos de uso.

Ilustración 2.11: Logo StarUML.

Capítulo 3

Análisis

3.1. Requisitos

Los requisitos de usuario del proyecto han sido representados en una tabla para los no funcionales (Tabla 3.1) y otra para los funcionales (Tabla 3.2).

REQUISITOS NO FUNCIONALES	
1	La experiencia debe ser fluida y responder rápidamente a las acciones del usuario.
2	La aplicación debe ser intuitiva, cualquier usuario, independientemente de su conocimiento previo, debería ser capaz de usarla.
3	El código deber permitir añadir nuevas funcionalidades con facilidad.

Cuadro 3.1: Requisitos no funcionales de la aplicación.

REQUISITOS FUNCIONALES	
1	El usuario debe poder acceder al menú de la aplicación.
2	El usuario debe poder cambiar entre el mapa topográfico y el mapa de calor.
3	El usuario debe poder tocar los sensores para activar y desactivar el panel de los datos que está recogiendo.
4	El usuario debe poder tocar las celdas del mapa de calor para activar y desactivar el panel con su valor.
5	El usuario debe poder cambiar el tipo de valor del mapa de calor desde el menú.
6	El usuario debe poder cambiar el modo del mapa de calor desde el menú.

Cuadro 3.2: Requisitos funcionales de la aplicación.

3.2. Casos de uso

Un usuario puede realizar diferentes acciones al interactuar con la aplicación, estas quedan representadas en el diagrama de casos de uso de la (Ilustración 3.1).

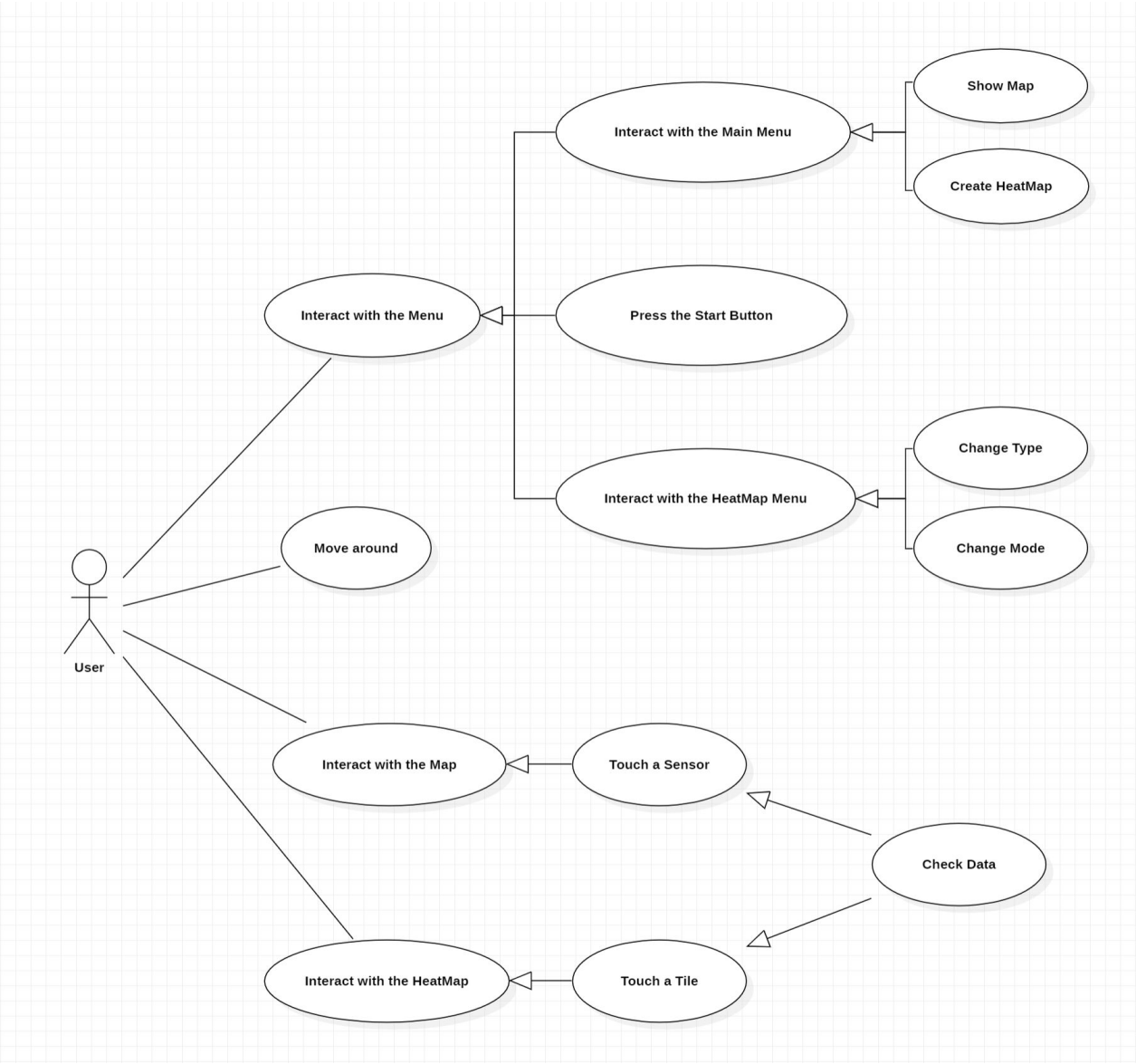

Ilustración 3.1: Diagrama de Casos de Uso.

3.3. Historias de usuario

A continuación se pueden observar varias historias de usuario que escenifican diferentes usos de la aplicación.

- Como Turing, un joven en sus vacaciones de verano, quiero comprobar la temperatura de la isla para decidir si ir a la playa hoy.

- Como Von Neumann, un guarda forestal en activo, quiero vigilar la temperatura y humedad de los diferentes sectores de la isla para prestar especial atención a los más susceptibles a incendios.

- Como Lovelace, una profesora de la ULPGC, quiero mostrarle a mis alumnos una experiencia de la realidad mixta en la que puedan observar los valores del mapa de calor y analizarlos según la zona en la que se encuentran.

- Como Dijkstra, un miembro del tribunal de evaluación de este TFT, quiero experimentar la experiencia de usuario que tendría alguien al usar la aplicación, además quiero comprobar que no hay bugs que se ocultaran en la memoria.

3.4. Estudio previo

Antes de comenzar con el diseño de la aplicación hubo que realizar un proceso de investigación de las diferentes tecnologías que se podrían usar para llevar a cabo los objetivos propuestos.

Primero se eligió Unity3D[1] como motor y entorno de desarrollo 3D, a continuación se investigó cómo adaptarlo para el desarrollo de realidad virtual. Para esto se acabó eligiendo hacer uso del MRTK[5] (Mixed Reality Toolkit) de Microsoft, el cual proporciona varias herramientas que facilitan el desarrollo de aplicaciones de realidad mixta.

La siguiente investigación que se realizó fue la manera de representar un mapa con relieve. Para esto se decidió hacer uso de la herramienta creada por la desarrolladora Infinity Code[4], OnlineMaps[3], una solución multiplataforma para la creación de mapas tanto 2D como 3D.

Por último se investigó las diferentes maneras de recoger los datos meteorológicos, y se acabó decidiendo usar los sensores “2JCIE-BU01”[12] de la compañía Omron[13], los cuales recolectan información sobre diferentes factores medioambientales, se hablará de ello con más detalle en la parte de recolección de datos.

Capítulo 4

Diseño y metodología

4.1. Diseño

4.1.1. Diseño del Proyecto

Al comenzar el proyecto hubo una primera fase en la que se decidió un diseño general de la aplicación. Este diseño constaba de cómo sería el flujo de la aplicación además de el diseño general de sus componentes.

El flujo y diseño original que se planeaba era el siguiente:

Al iniciar la aplicación se muestra un menú que permite elegir entre mostrar el mapa de calor o el mapa topográfico.

En caso de mostrar el mapa topográfico se muestra el mapa con los sensores. Por el contrario, al mostrar el mapa de calor, se crea un nuevo submenú y se muestran unas celdas de diferentes colores y tamaños que enseñan la distribución de los valores.

La mayor parte de esta idea original continuó, pero sufrió algunas modificaciones.

Ahora el menú no se muestra directamente, se accede a él mirándose la muñeca. Se creó un botón para precargar todos los mapas y se diseñó un marco para el mapa para que el usuario tuviera alguna referencia visual.

4.1.2. Diseño de los botones

El proyecto cuenta con un menú que permite interactuar y navegar por toda la aplicación. Para crear este menú hubo que desarrollar y diseñar una serie de botones que cumplieran las diferentes necesidades.

La lógica que controla el funcionamiento de los botones es la misma para todos ellos (Ilustración 4.1), y está explicada en Funcionamiento de los botones. Además la composición

interna de todos ellos es muy similar. Todos ellos presentan un objeto que actúa como el botón, otro como el borde del botón y un último objeto que contiene el texto.

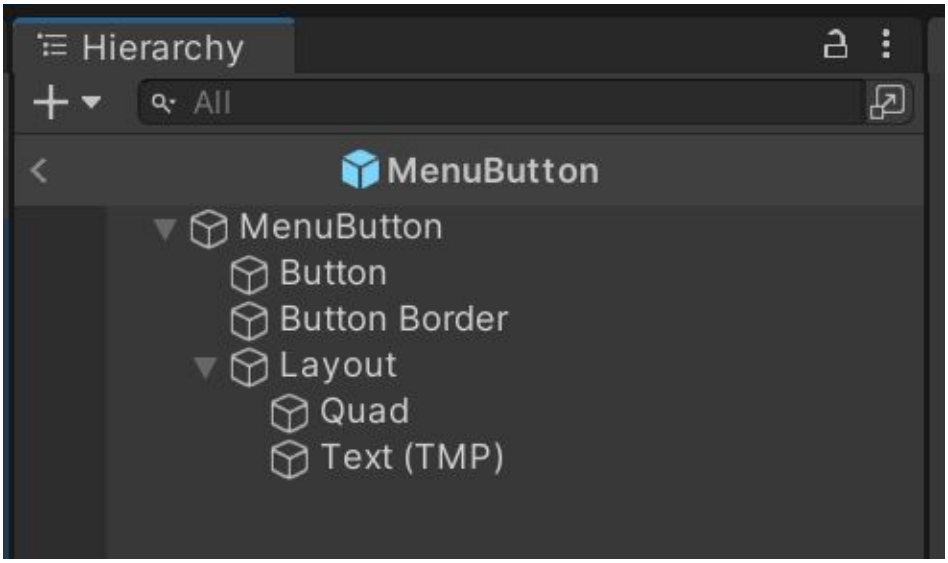

Ilustración 4.1: Jerarquía del MenuButton.

Se decidió crear un diseño único para cada tipo de botón, los diferentes tipos se diferencian según su funcionalidad, sirviendo de esta manera para crear agrupaciones.

El primer tipo de botón (Ilustración 4.2) es usado solo al iniciar la aplicación. Consiste en un botón circular de un tamaño considerable que sirve como inicio del menú.

Ilustración 4.2: Modelo del botón de Start.

El segundo tipo de botón es de menor tamaño que el primero y con forma rectangular. Este tipo (Ilustración 4.3) es utilizado para crear los botones del menú principal, los cuales permiten mostrar y ocultar los mapas.

Ilustración 4.3: Modelo del botón MenuButton.

El tercer tipo es de un tamaño bastante menor que los anteriores y posee una forma cuadrada. Estos botones (Ilustración 4.4) son usados para elegir entre los modos del mapa de calor, siendo parte del submenú del mapa de calor. Tanto este tipo como el siguiente poseen

un menor tamaño que el resto ya que no necesitan un gran espacio para el texto descriptivo del botón.

Ilustración 4.4: Modelo del botón de cambio de modo.

El cuarto y último tipo es muy similar al anterior, pero rotado 45º, dando lugar a una forma de rombo. Este tipo de botón (Ilustración 4.5) es usado para cambiar entre los diferentes tipos de datos que puede mostrar el mapa de calor.

Ilustración 4.5: Modelo del botón HeatMapMenuButton.

4.1.3. Vinculación con los sensores

La intención de este proyecto era hacer uso de los sensores de la empresa Omron, 2JCIE-BU01, pero durante el desarrollo del proyecto se descartó la idea debido a varios motivos. Los dos principales fueron:

- ***El precio:*** El proyecto hace uso de más de 20 sensores para el cálculo del mapa de calor, ya que hay que delimitar la zona a controlar con ellos, y a fecha de desarrollo el precio está situado alrededor de 120€ la unidad, lo cúal lo vuelve inviable económicamente.

- ***El tiempo:*** El plazo máximo para la realización del proyecto son 300 horas, entre el tiempo empleado en el estudio e investigación, además de todo el trabajo diseño e implementación, se acercó demasiado al tiempo límite como para poder llevarlo a cabo.

Son estos los motivos por los que se decidió sustituir los sensores por un fichero Json que simula la obtención de los datos del sensor. Para crear una sensación de variación temporal en el mapa de calor, los datos del Json se auto-modifican cada ciertos intervalos de tiempo. Más información al respecto en Tratamiento de los JSONs.

4.2. Metodología

4.2.1. Metodología Ágil

Para la realización de este proyecto se intentó seguir todo lo posible la metodología ágil SCRUM, ya que hay partes de esta que carecen de sentido cuando se trata de un equipo de un solo integrante.

Debido a estar trabajando durante la realización del proyecto, hubo sprints en los que se pudo avanzar menos de los planeado, pero siempre se intentó tener una versión funcional al final de este.

Se utilizó el programa gratuito ClickUp (Ilustración 4.6) para organizar las tareas dependiendo de su estado actual.

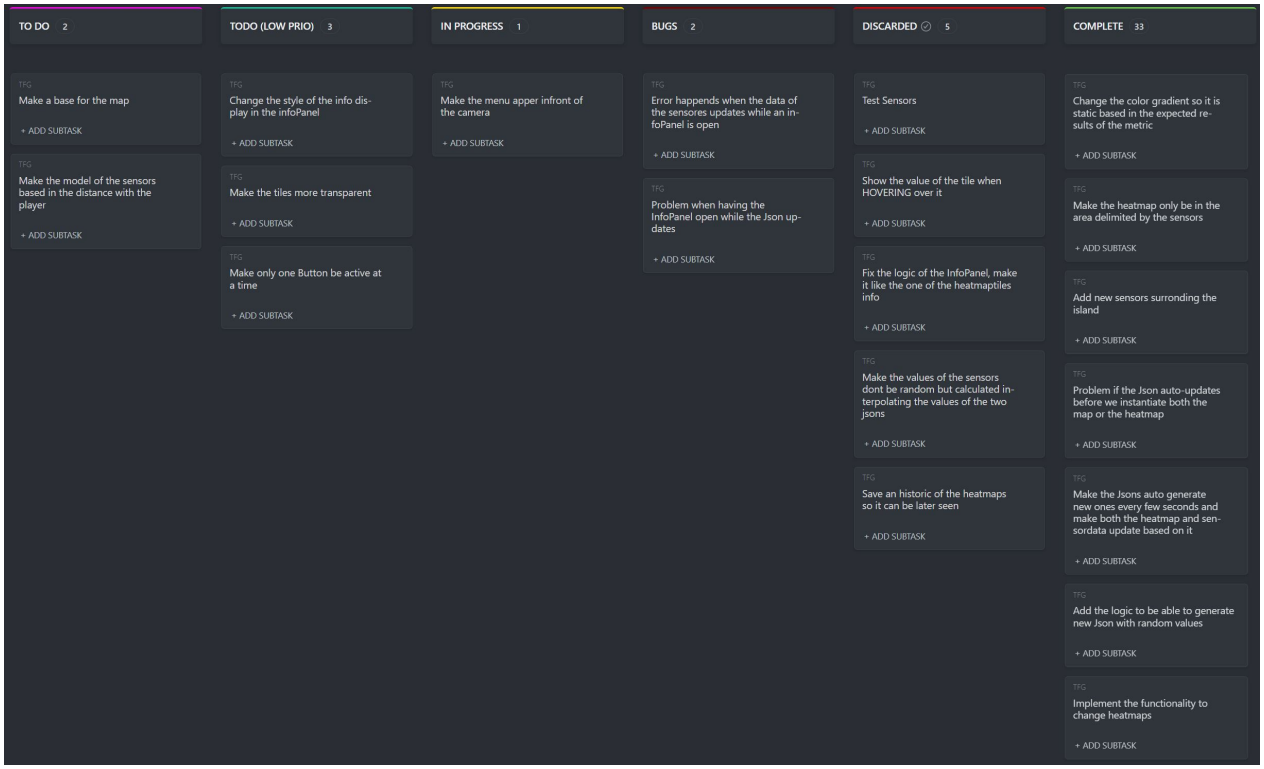

Ilustración 4.6: Organización de las tareas en diferentes grupos usando la aplicación ClickUp.

Como se observa en la (Ilustración 4.6), podemos ver que las tareas se dividen en diferentes grupos según su estado.

- ***TO DO:*** Tareas de código que no se han empezado a realizar pero están pendientes de ello.

- ***TO DO (LOWPRIO):*** Tareas de código que no se han empezado a realizar pero que no son prioritarias, suelen consistir de arreglos visuales o ideas que no han terminado de decidirse si se llevaran acabo.

- ***IN PROGRESS:*** La tarea o tareas que se están realizando actualmente.

- ***BUGS:*** Errores que se han encontrado en el funcionamiento de la aplicación y que están pendientes por resolver.

- ***DISCARDED:*** Tareas o ideas que fueron descartadas por diferentes motivos.

- ***COMPLETED:*** Tareas de cualquiera de los anteriores grupos que ya han sido completadas.

Capítulo 5

Implementación y Modelos

5.1. Proyecto en Unity

5.1.1. Assets

Este proyecto hace uso de una gran cantidad de assets(Ilustración 5.1), pero se pueden dividir en dos grupos principales, los creados por el autor y los de terceros, ya sean de pago o gratuitos.

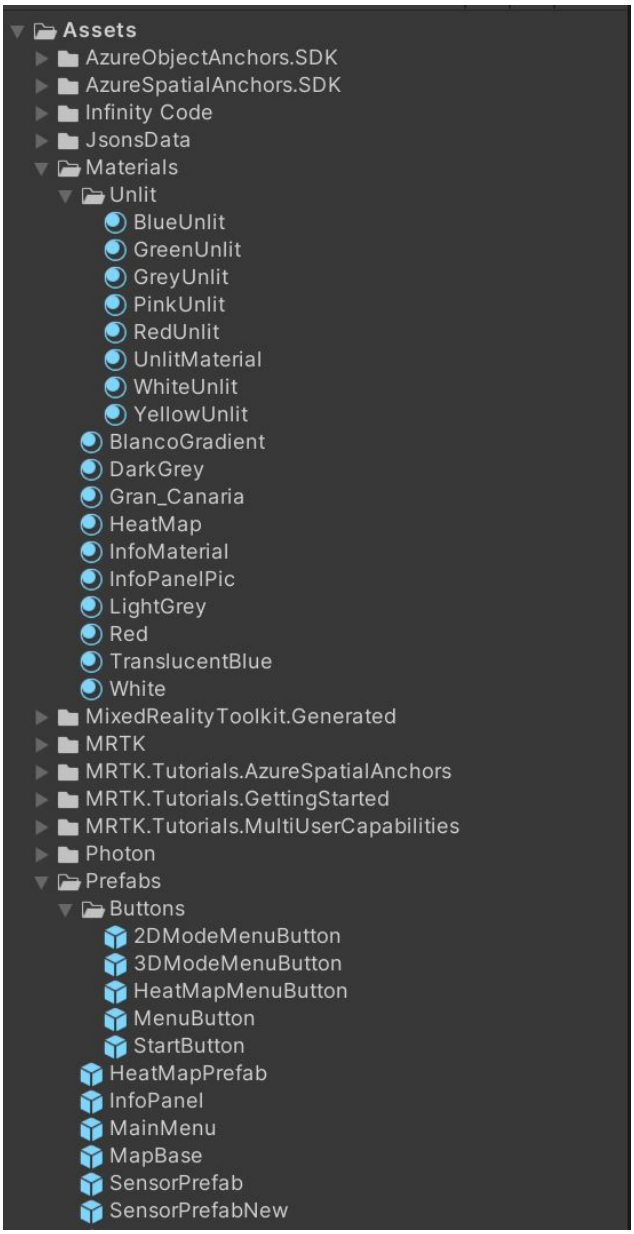

Ilustración 5.1: Todos los assets del proyecto.

Dentro del grupo de los creados por el autor, se encuentran los diferentes materiales creados para el resto de objetos y los prefabs de los diferentes botones, menús, sensores y otros objetos del proyecto.

En cuanto al grupo de assets de terceros, se hallan los pertenecientes a todos los scripts y assets de MRTK y OnlineMaps.

5.1.2. Scripts

Este proyecto hace uso de varios scripts(Ilustración 5.2) para realizar todas sus funciones, estos los podemos dividir en dos grupos, controladores y utilidades.

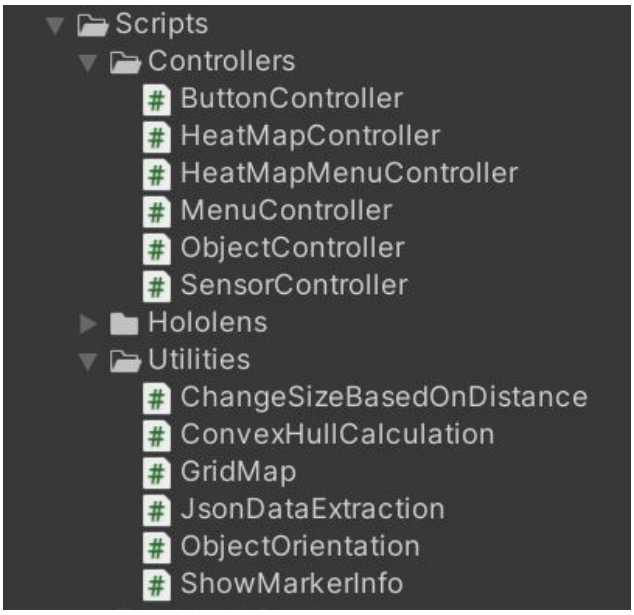

Ilustración 5.2: Scripts del proyecto.

Los controladores son los encargados de la lógica de funcionamiento de diferentes objetos. En cambio las utilidades cumplen funciones concretas y que pueden llegar a ser usadas por diversos controladores.

Controladores:

- ***SensorController:*** Script encargado de instanciar el mapa topográfico y los sensores con sus datos.

- ***HeatMapController:*** Se encarga de instanciar y calcular los parámetros de las celdas del mapa de calor.

- ***MenuController:*** Script principal para el funcionamiento de los menús, contiene todos los métodos relacionados con las acciones de los botones.

- ***HeatMapMenuController:*** Contiene los métodos específicos del menú del mapa de calor.

- ***ButtonController:*** Clase que contiene la lógica de los botones, los métodos que detectan su activación y desactivación, además de sus cambios visuales.

- ***ObjectController:*** Controlador con métodos para los objetos, como puede ser cambiar la visibilidad de estos.

Utilidades:

- ***GridMap:*** Contiene toda la lógica para la creación y cálculo de la cuadrícula de valores.

- ***ConvexHullCalculation:*** Clase dedicada a los cálculos relacionados con la envolvente convexa, tanto su creación como los cálculos para saber si un punto se encuentra dentro de ella.

- ***JsonDataExtraction:*** Encargado de la lectura y actualización de los Json que contienen la información de los sensores.

- ***ObjectOrientation:*** Script otorgado a objetos para que roten y siempre estén orientados al usuario.

- ***ShowMarkerInfo:*** Contiene los métodos y atributos de para mostrar los paneles de información.

- ***ChangedSizeBasedOnDistance:*** Permite a los objetos que lo tengan como componente, aumentar o reducir su tamaño en relación a la distancia con el usuario.

5.1.3. Escena

En la escena(Ilustración 5.3) se encuentran tanto los gameObjects que forman parte del entorno, como pueden ser el marco del mapa, el propio mapa o el menú, como los que instancian los controladores.

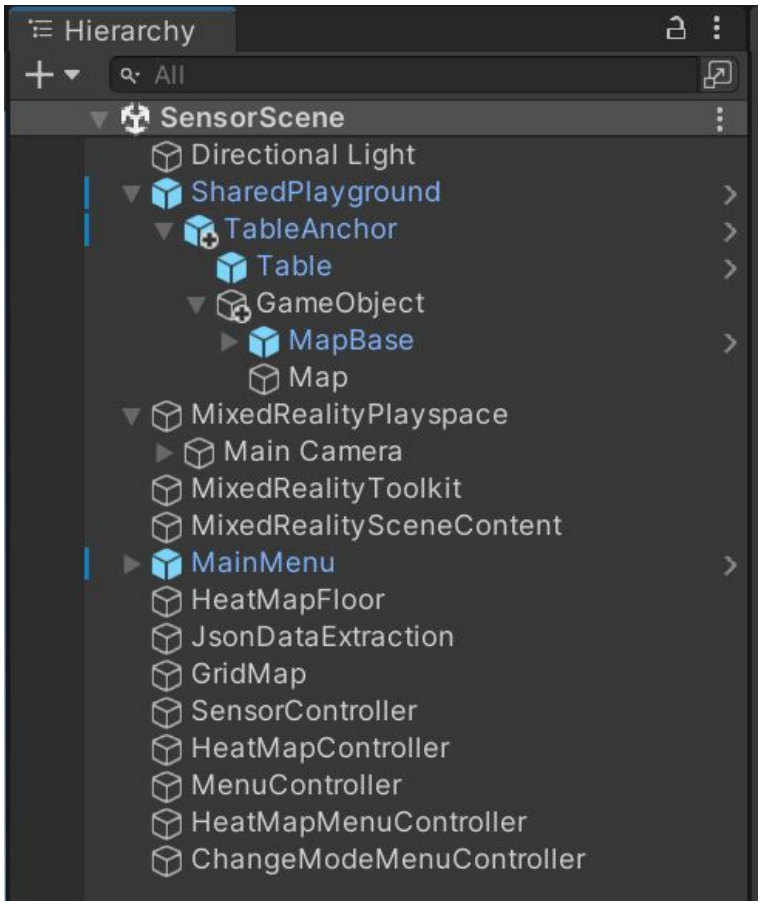

Ilustración 5.3: Jerarquía de la escena de Unity.

5.2. Recolección de Datos

5.2.1. Tratamiento de los JSONs

Como método de obtención de los datos de los sensores se utiliza un archivo JSON (Ilustración 5.4) configurado para almacenar la información de los siguientes campos:

- La fecha y hora a la que se recogió la información de los sensores.
- Un vector que contiene todos los sensores a los que se recoge información.Dentro de este vector, cada sensor posee como parámetros:
    - El ID del sensor para poder identificarlo.

- Las coordenadas en latitud y longitud de su posición actual.

- Un vector con todos los datos que ha recogido el sensor. Los datos son: temperatura, humedad, luminosidad, presión atmosférica, compuestos orgánicos volátiles, C02 en el ambiente e intensidad sísmica.


```
{   "date": "2023-07-05T00:00:00",
    "sensor":
    [{
        "id": 1,
        "location": {"coordinates": [-15.6876, 28.1662]},
        "data": {
            "temperature": 30.0,
            "humidity": 30.0,
            "ambientLight": 50000.0,
            "pressure": 70.0,
            "soundNoise": 70.0,
            "eTVOC": 0.4,
            "eCO2": 400.0,
            "seismicIntensity": 0.0
        }
    },
    {
        "id": 2,
        "location": {"coordinates": [-15.5261, 28.1486]},
        "data": {
            "temperature": 20.0,
            "humidity": 40.0,
            "ambientLight": 50000.0,
            "pressure": 60.0,
            "soundNoise": 70.0,
            "eTVOC": 0.4,
            "eCO2": 400.0,
            "seismicIntensity": 0.0
        }
    },
```


Ilustración 5.4: Fichero Json con los datos de los sensores.

Esto se trató a nivel de código con la clase JsonDataExtraction. Esta clase contiene el modelo de los datos (Ilustración 5.5) y los métodos necesarios para leer y actualizar los valores del Json.

El modelo de los datos está creado a partir de clases que recogen los campos del Json.

```
[System.Serializable]
15 references
public class SensorsList
{
    public string date;
    public Sensor[] sensor;
}

[System.Serializable]
11 references
public class Sensor
{
    public int id;
    public Location location;
    public SensorData data;
}

[System.Serializable]
1 reference
public class SensorData
{
    public float temperature;
    public float humidity;
    public float ambientLight;
    public float pressure;
    public float soundNoise;
    public float eTVOC;
    public float eCO2;
    public float discomfort;
    public float heatStroke;
    public float vibration;
    public float siValue;
    public float pga;
    public float seismicIntensity;
}

[System.Serializable]
1 reference
public class Location
{
    public float[] coordinates;
}
```

Ilustración 5.5: Modelo de los datos en código.

Además de esto la clase tiene métodos para la actualización de los valores. Estos se encargan de actualizar los datos con valores dentro de un rango esperable para simular la recolección de los datos, ya que como se menciona en Vinculación de los Sensores, no se pudo establecer una conexión real con ellos.

Estos métodos no solo se encargan de la actualización de los valores, sino que también llaman a métodos de tanto los sensores como el mapa de calor para que actualicen sus datos.

5.3. Interfaz de Usuario

5.3.1. Inicio de la Aplicación

Al iniciar la aplicación se muestra es un tablero (Ilustración 5.6) sobre el que posteriormente irán situados los mapas. El tablero tiene marcados los ejes cardinales con letras, para facilitar la orientación del usuario.

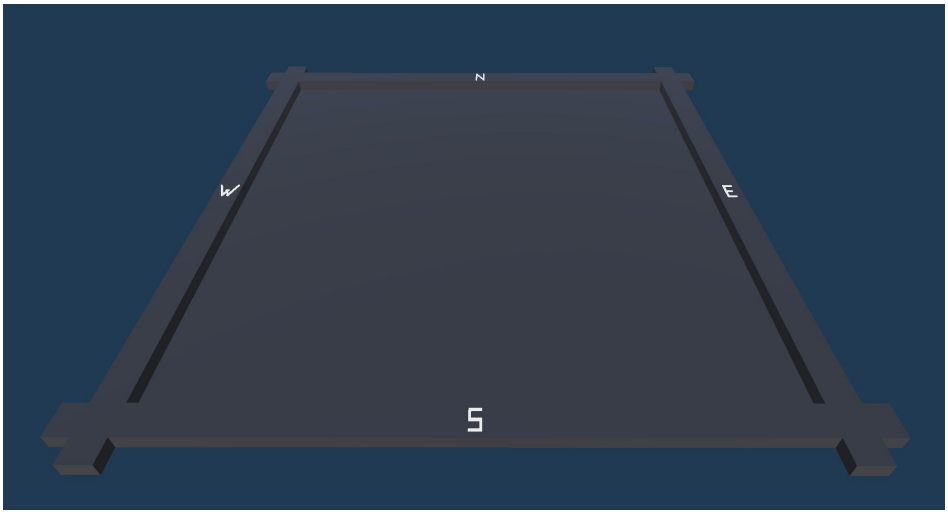

Ilustración 5.6: Tablero con los ejes cardinales.

El usuario podrá acceder al menú mediante un gesto nativo de MRTK, mirar su mano, en concreto su muñeca. Podrá seguir accediendo al menú de esta misma manera durante toda la aplicación.

5.3.2. Botón de Inicio

La primera vez que se acceda al menú al iniciar la aplicación, se mostrará un único botón circular con la palabra “Start” (Ilustración 5.7), este botón posee varias funciones.

Tiene la función de ser la primera toma de contacto con el usuario y ser el punto de partida del proyecto, pero además, a nivel de funcionalidad, también posee la función de

realizar un precarga de los mapas, tanto el topográfico como el de calor, y de activar otra serie de funciones del proyecto. Esto tiene como objetivo reducir la carga de trabajo y mejorar la fluidez del resto del proyecto a cambio de reducirla al iniciar.

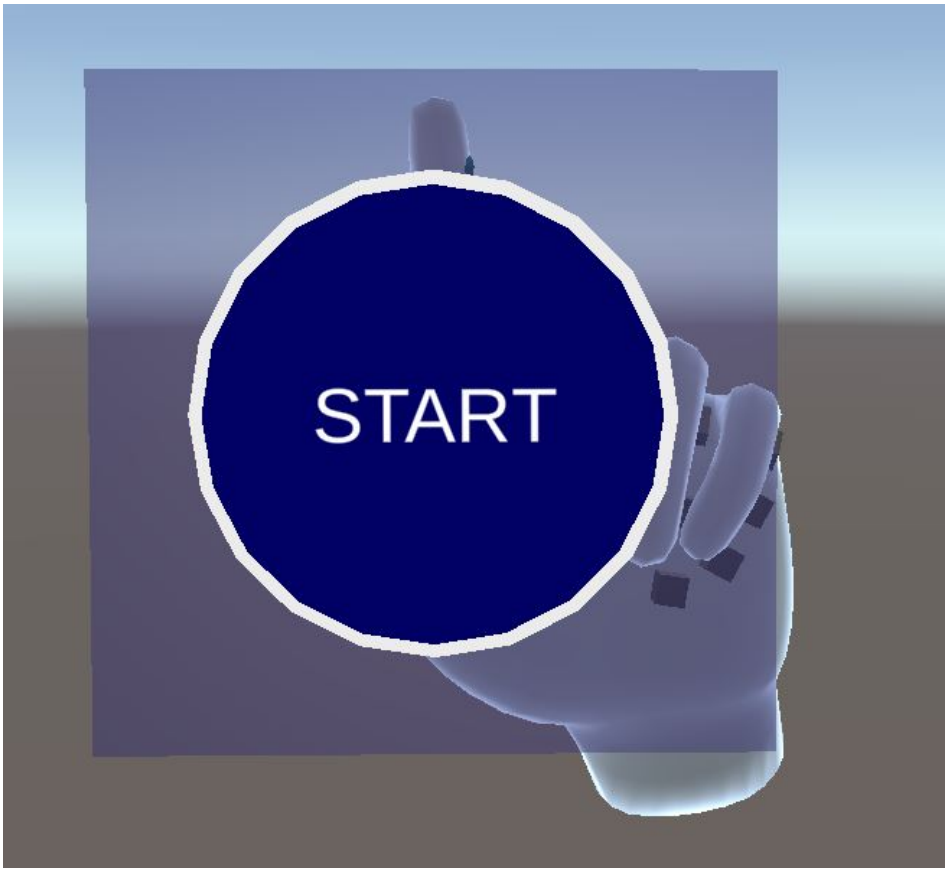

Ilustración 5.7: Botón de Start.

Los pasos que realiza la aplicación en esta fase de precarga son los siguientes:

1. Lee el Json que contiene todos los datos de los sensores.
2. Carga el mapa topográfico de la isla.
3. Sitúa en el mapa unos modelos 3D de los sensores con los datos leídos.
4. Divide el mapa en una cuadrícula y calcula los valores de cada una.
5. Crea un objeto 3D cuadrado para representar cada casilla y usa el valor de dicha casilla en el grid para cambiar su escala y color.
6. Activa la función de actualización de los datos del Json a intervalos fijos
7. Oculta el botón de “Start” y ambos mapas (Durante el proceso de precarga los mapas nunca llegan a ser vistos por el usuario).
8. Muestra el Menú Principal del proyecto.

5.3.3. Menú Principal

El menú principal (Ilustración 5.8) de la aplicación cuenta con dos botones, “Show Map” y “Create HeatMap”, la función de estos botones es mostrar el mapa topográfico de la isla, con los sensores sobre ella, y crear el mapa de calor respectivamente.

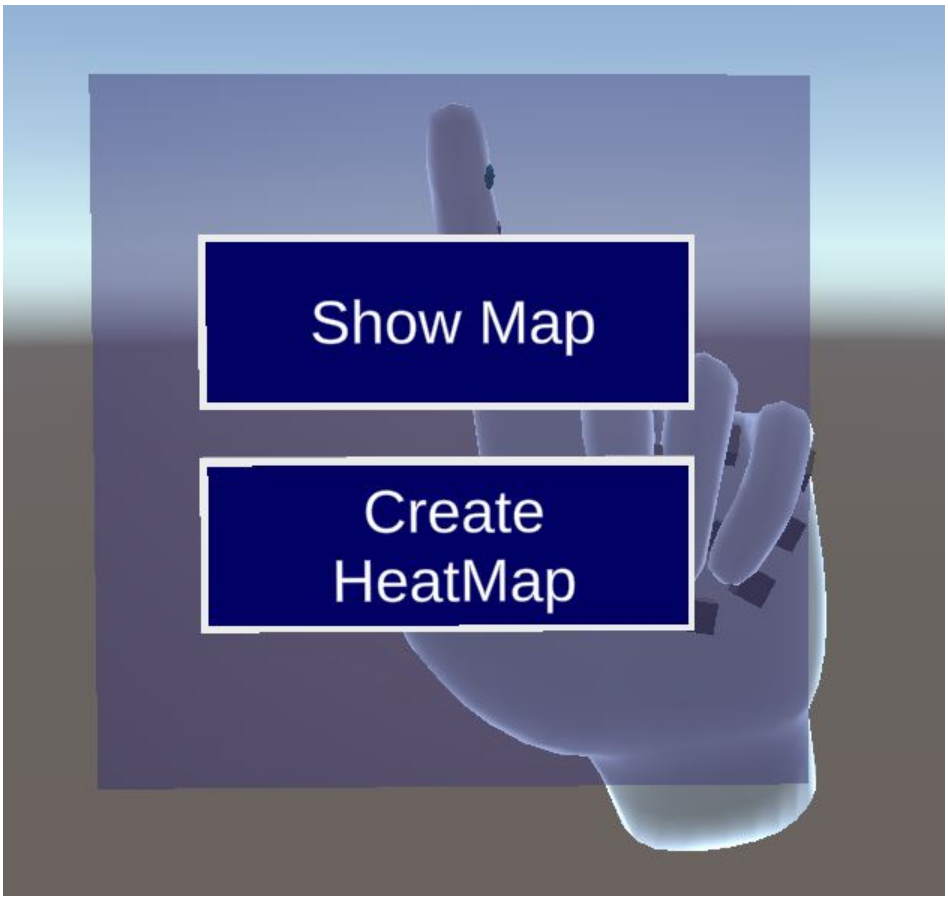

Ilustración 5.8: Menú principal de la aplicación.

En caso de pulsar el botón “Show Map” (Ilustración 5.9) aparecerá el mapa topográfico de la isla de Gran Canaria, sobre él se muestran unos modelos 3D de unas antenas parabólicas, estas antenas son la representación física de los sensores.

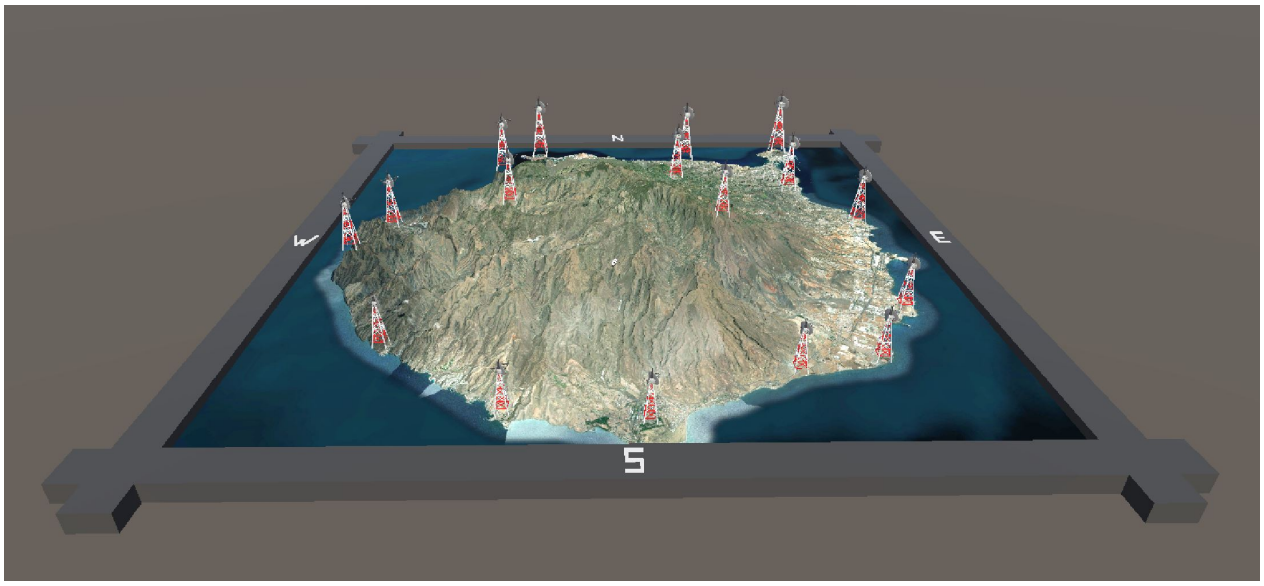

Ilustración 5.9: Mapa topográfico de la isla de Gran Canaria.

En caso de pulsar sobre uno de dichos sensores, se mostrará un panel (Ilustración 5.10) con toda la información a tiempo real que ese sensor está recogiendo.

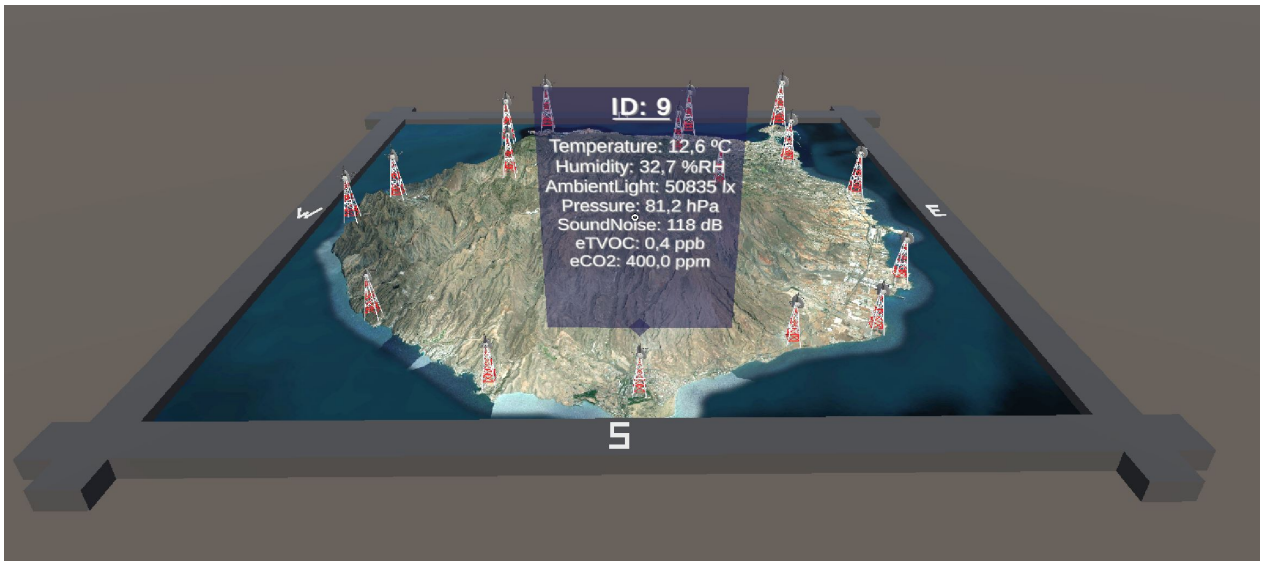

Ilustración 5.10: Panel de información de los sensores.

Por el contrario, en caso de pulsar el botón “Create HeatMap”, sucederán dos cosas, primero se desplegará un nuevo menú, el Menú del Mapa de Calor, el cual contiene las diferentes opciones y funcionalidades para interactuar con el mapa de calor, y seguidamente se activará el mapa de calor (Ilustración 5.11), haciendo uso de unas opciones preestablecidas que podrán ser modificadas posteriormente por el usuario desde el menú.

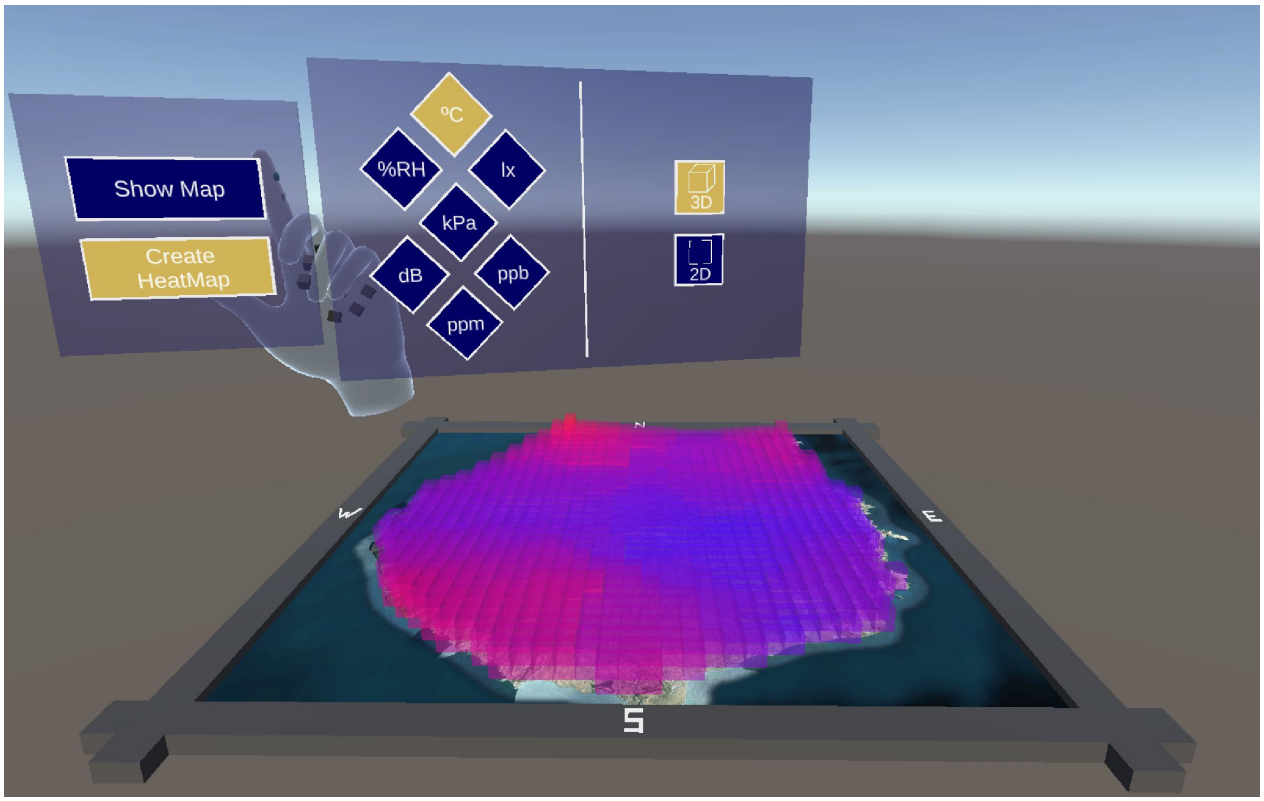

Ilustración 5.11: Mapa de calor.

El mapa de calor está formado por una serie de celdas representadas por una objeto 3D cuadrado, en caso de pulsar sobre una de ellas se abrirá un pequeño panel (Ilustración 5.12) de información mostrando el valor actual de la celda.

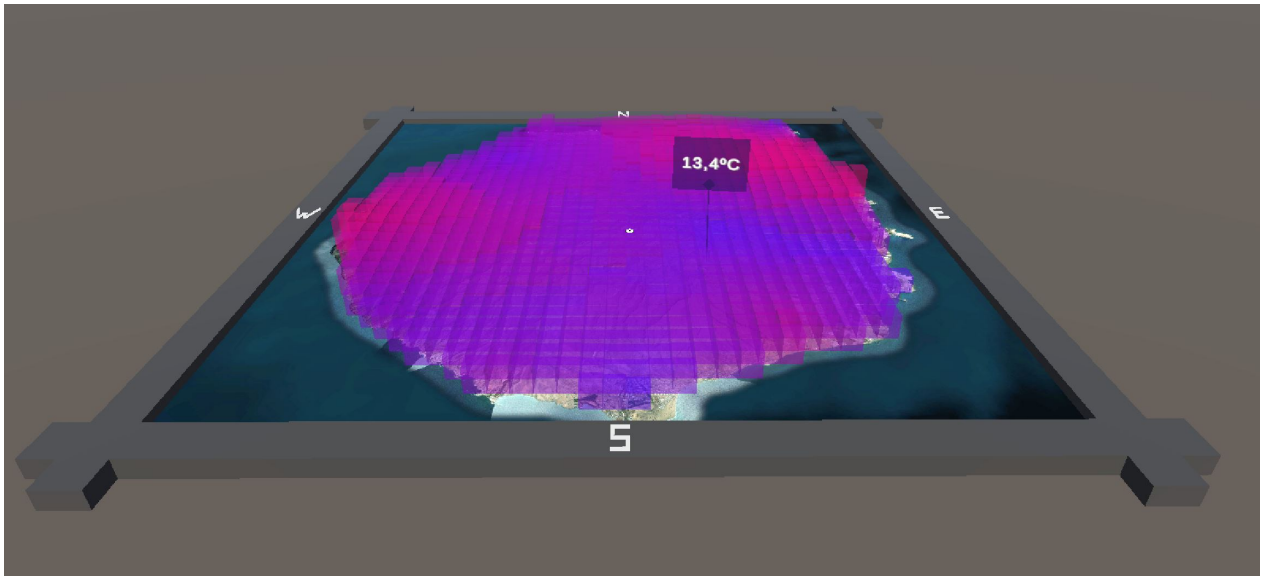

Ilustración 5.12: Panel de información de una celda del mapa de calor.

5.3.4. Menú Mapa de Calor

El menú del mapa de calor presenta una serie de opciones que permiten cambiar la configuración del mapa de calor.

En la parte izquierda hay una serie de botones con forma de rombo, que permiten cambiar el factor meteorológico representado. Los cambios se realizan al instante, provocando una actualización de las casillas del mapa de calor, tanto de su color como de su altura. Esto se puede apreciar en la (Ilustración 5.13) y en la (Ilustración 5.14).

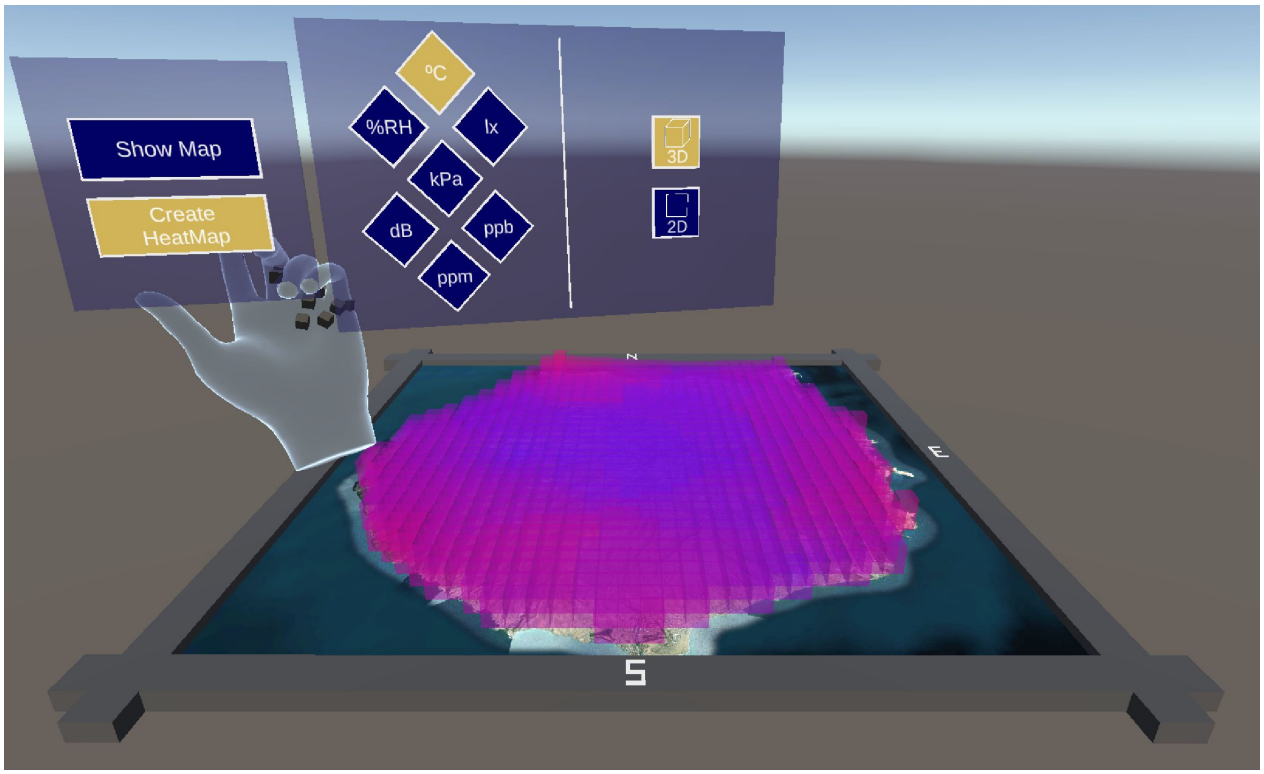

Ilustración 5.13: Mapa de calor con el tipo de dato temperatura.

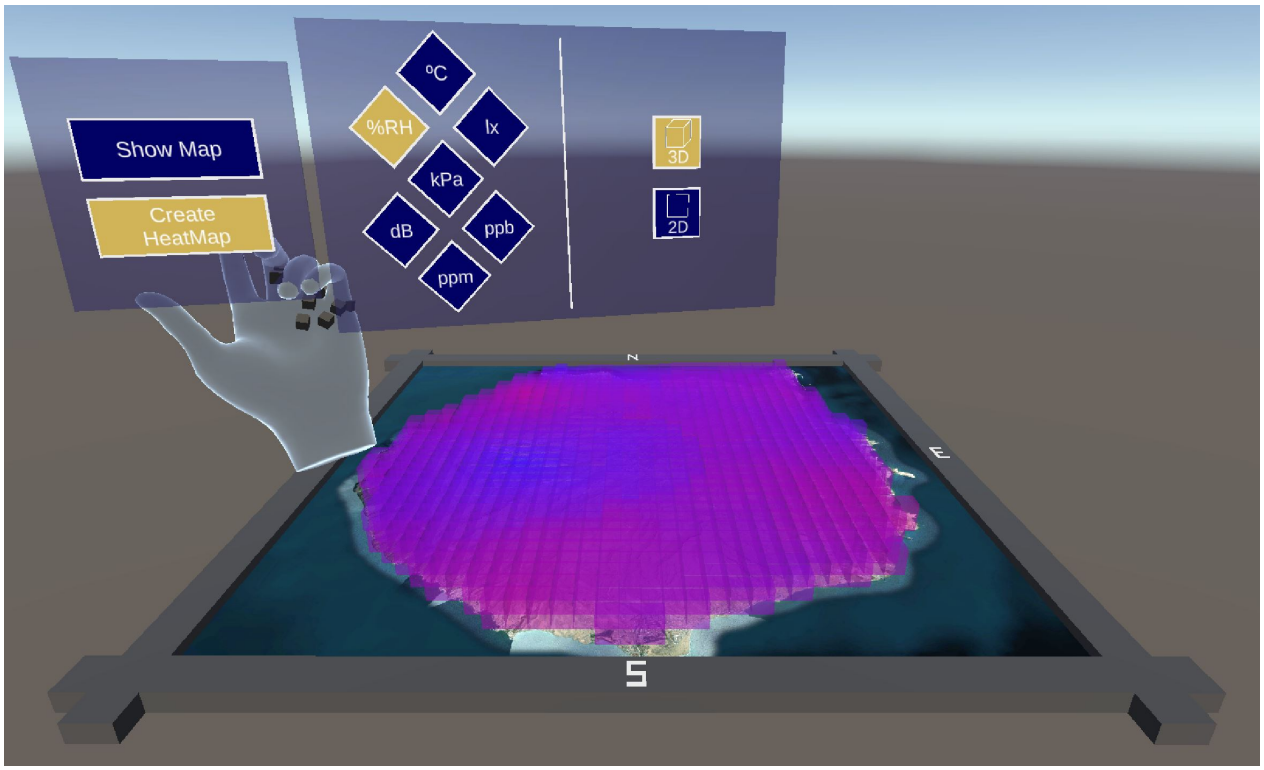

Ilustración 5.14: Mapa de calor con el tipo de dato humedad.

En la parte derecha hay dos botones cuadrados (Ilustración 5.15), estos botones permiten cambiar el modo del mapa de calor entre 3D y 2D.

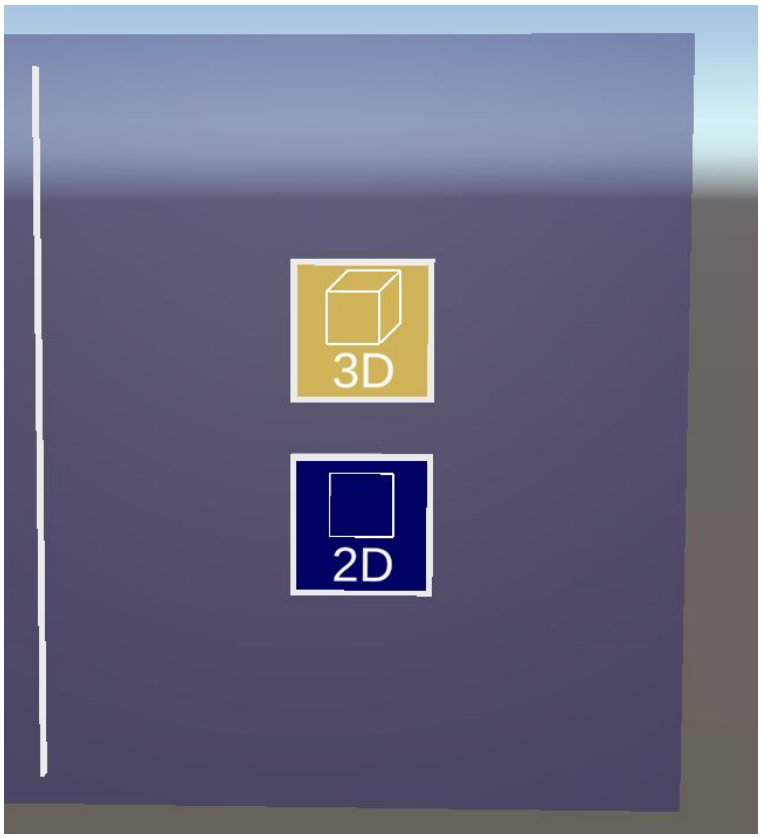

Ilustración 5.15: Menú de cambio de modo del mapa de calor.

En el modo 3D (Ilustración 5.16) el mapa de calor representa el valor de cada casilla mediante un color y una altura, ambos en función del valor de la casilla.

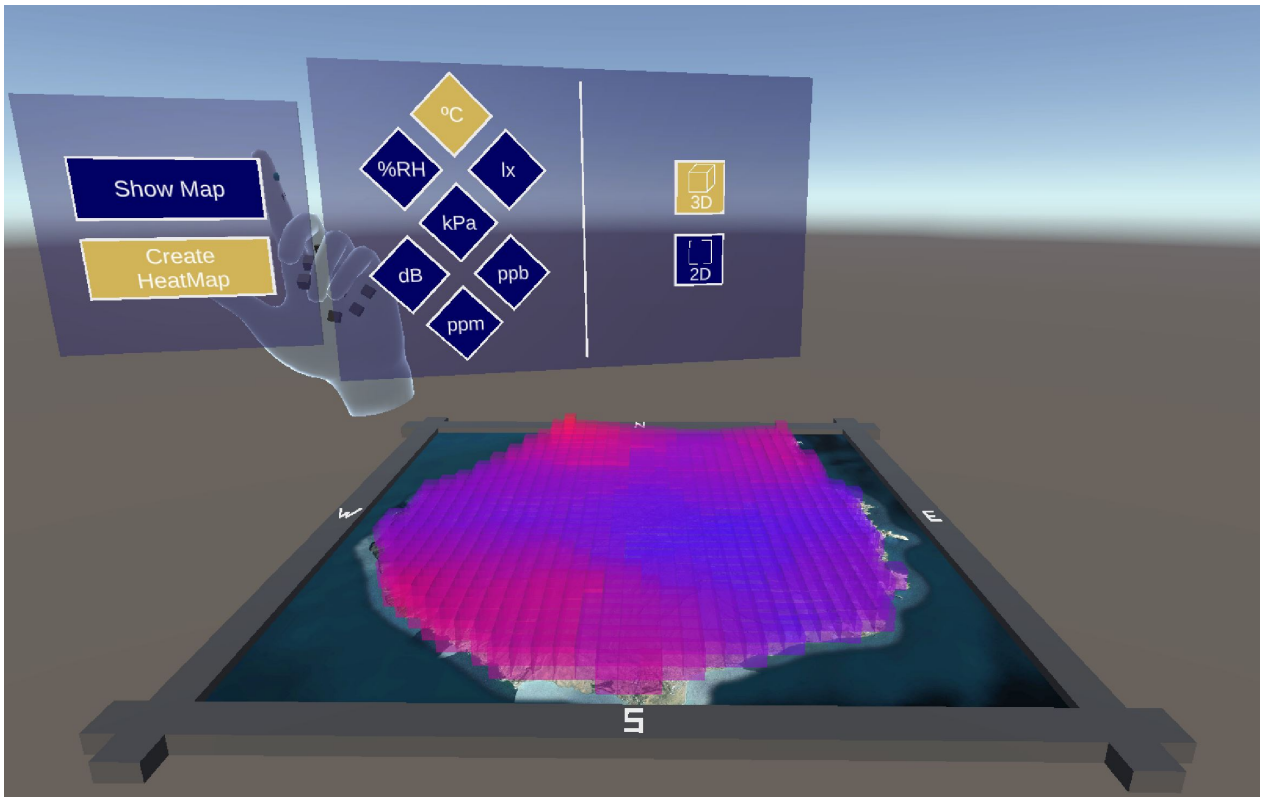

Ilustración 5.16: Mapa de calor en modo 3D.

En el modo 2D (Ilustración 5.17) desaparece la altura de los cuadrados, todos quedan al mismo nivel y cubren el mapa como una capa semitransparente, manteniendo los colores de cada casilla, de esta manera se puede observar la isla a través de él, a la vez que seguir apreciando el color del mapa de calor en cada zona. Esto facilita la observación de las zonas a cambio de perder el factor de la altura como método de monitorización.

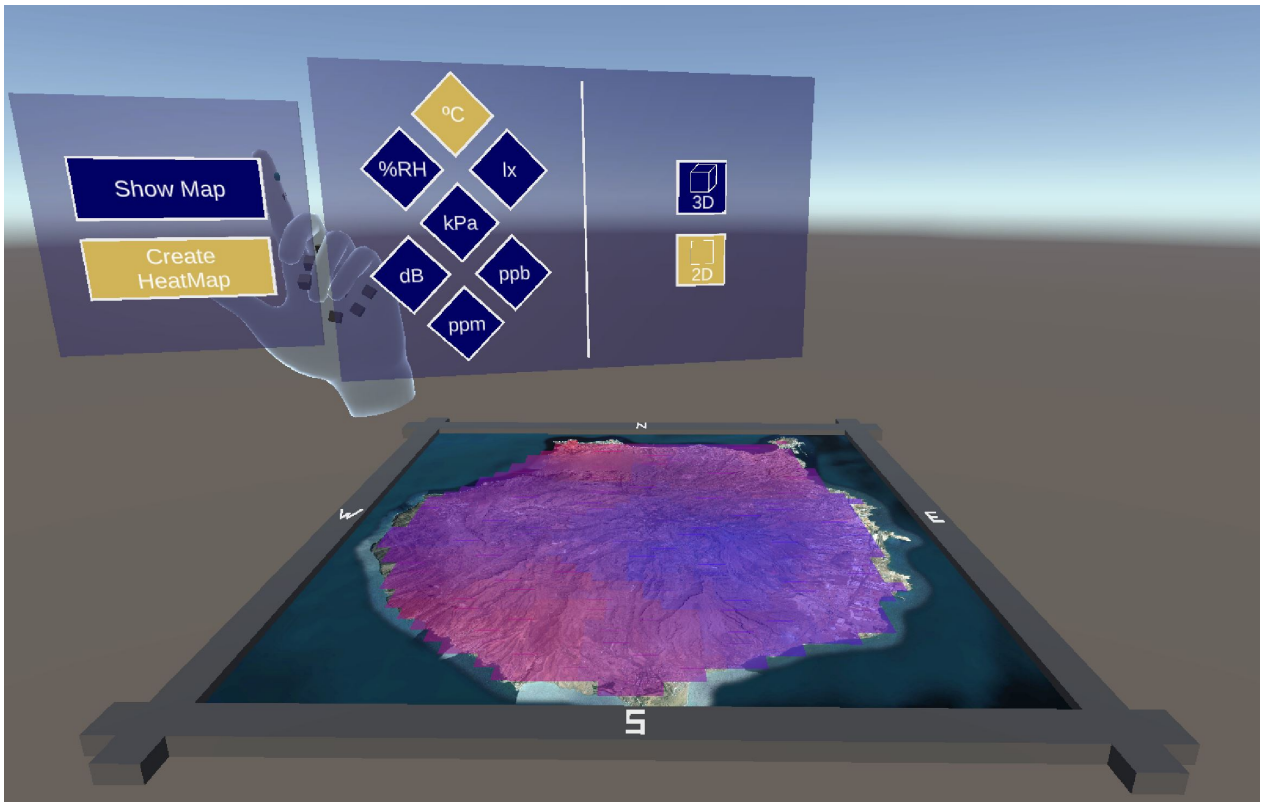

Ilustración 5.17: Mapa de calor en modo 2D.

5.4. Representación de Datos

5.4.1. Mapa y Sensores

La representación del mapa de la isla (Ilustración 5.18) se llevó a cabo mediante el uso del asset de Unity, OnlineMaps, este asset permite la creación de mapas de la superficie terrestre. La zona del mapa se elige proporcionando unas coordenadas de latitud y altitud, las cuales se usan como centro del mapa. Además de esto entre las opciones que ofrece OnlineMaps, también se encuentra la elección de zoom, el tamaño del mapa y la altura del relieve entre otras.

Ilustración 5.18: Mapa sin los sensores.

Sobre este mapa se representan los sensores, cada sensor, entre los datos que envía, manda su ubicación como un valor de latitud y altitud para ser representado en el mapa, OnlineMaps proporciona métodos de creación de objetos a través de estas coordenadas, colocándolos en la posición correspondiente.

Para la representación de los sensores en el mapa, se decidió optar por crear desde cero un modelo de una antena parabólica (Ilustración 5.19) haciendo uso de Unity.

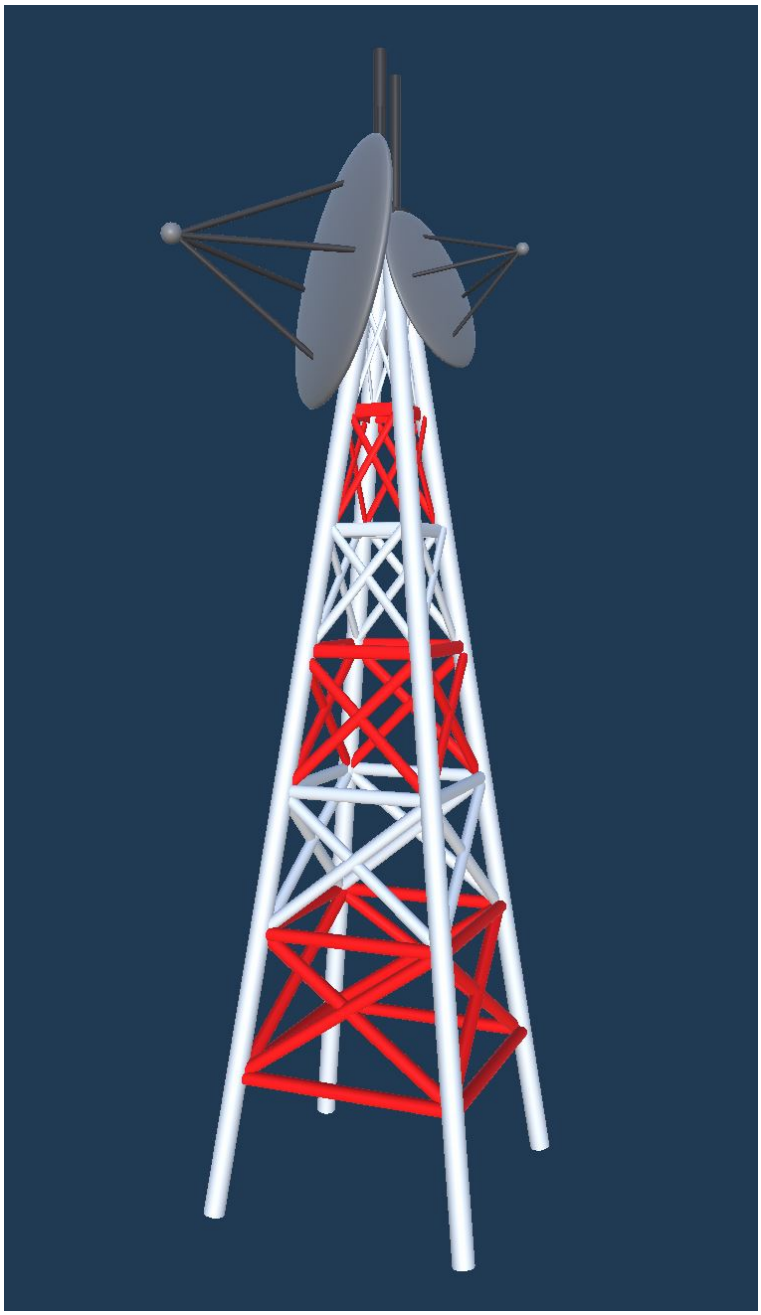

Ilustración 5.19: Modelo de un sensor.

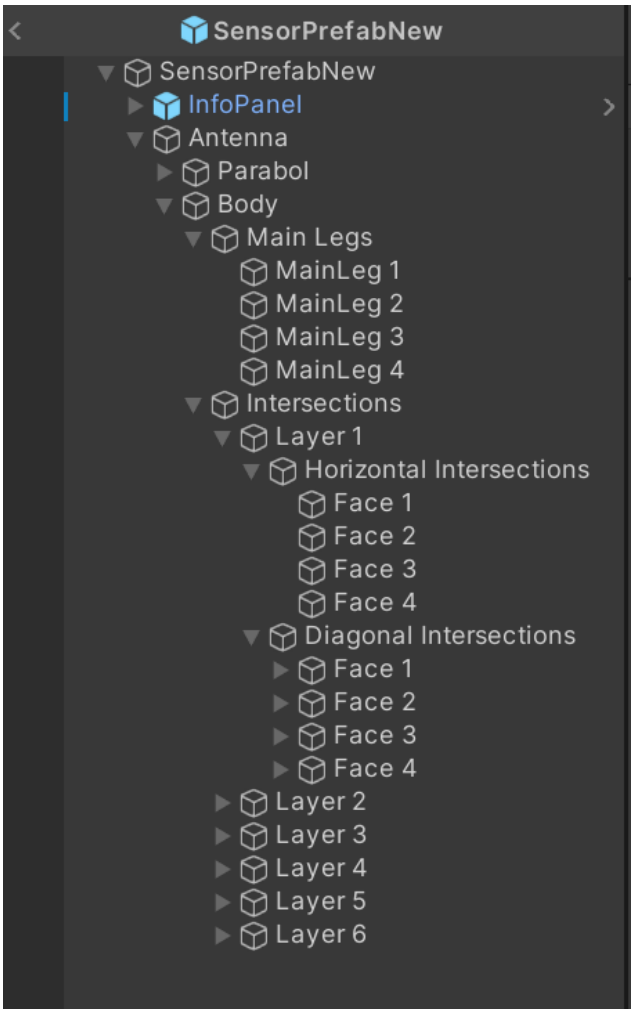

Ilustración 5.20: Jerarquía del modelo del sensor en Unity.

5.4.2. Panel con los datos del sensor

Como se mencionó anteriormente, se puede acceder a un panel con la información que está siendo transmitida por cada sensor pulsando en él.

El panel (Ilustración 5.21) muestra en la parte superior el identificador del sensor, y a continuación, la información que recoge. Esta información la muestra en pares, primero el factor meteorológico, dos puntos, y el valor numérico con su correspondiente magnitud.

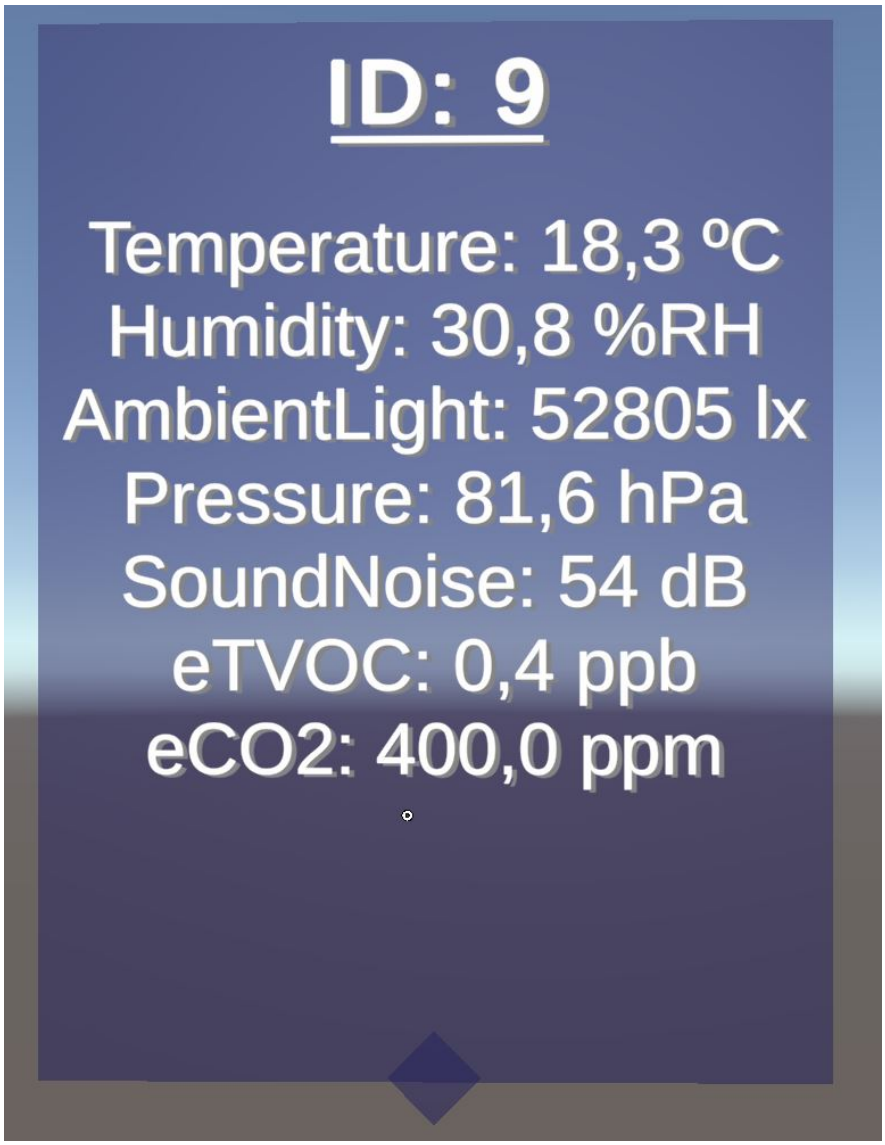

Ilustración 5.21: Panel de información de un sensor.

No pueden haber paneles abiertos simultáneamente, esto fue una decisión de diseño ya que tener varios abierto dificultaba la visibilidad. Para ello se lleva un registro del último panel activado.

5.4.3. Representación del Mapa de Calor

El mapa de calor representa el valor de cada casilla de la cuadrícula de valores mediante un cubo (Ilustración 5.22) de un color y una altura determinada, ambos en función del valor. El color va en un gradiente desde el color azul hasta el rojo, siendo el color azul los valores más bajos y los rojos los más altos. La altura de la misma manera muestra los valores bajos con una menor altura y los altos con una mayor.

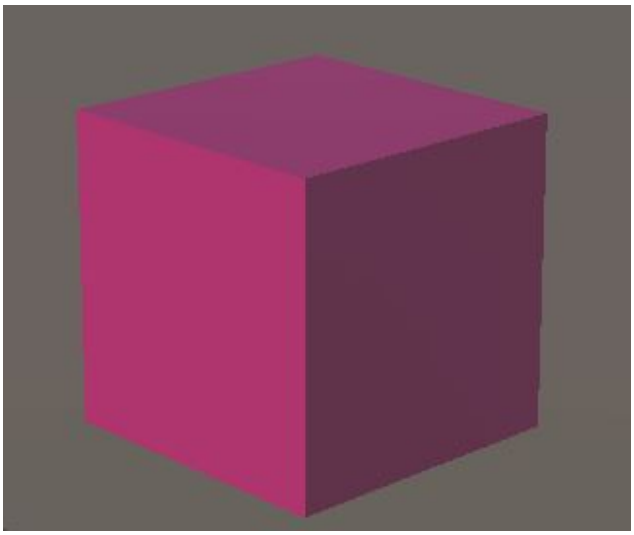

Ilustración 5.22: Celda del mapa de calor.

Tanto el cálculo de los valores de la cuadrícula, como del color y de la altura son explicados en Mapa de Calor.

5.5. Cambios en modelos y diseño

Durante el desarrollo del proyecto muchos de los modelos y diseños sufrieron cambios y mejoras.

Algunos de los más notables fueron el de los modelos del sensor, que pasaron de un modelo simple y minimalista(Ilustración 5.23) de una antena a uno mucho más detallado y complejo(Ilustración 5.24), y el del menú, el cual cambió de gama de colores y se le añadió un fondo semitransparente(Ilustración 5.25)(Ilustración 5.26).

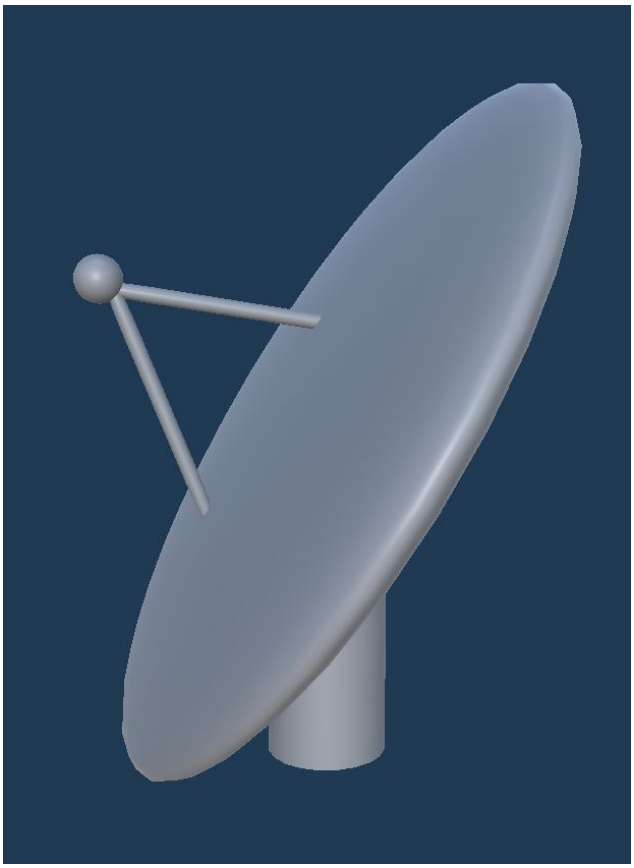

Ilustración 5.23: Modelo antiguo del los sensores.

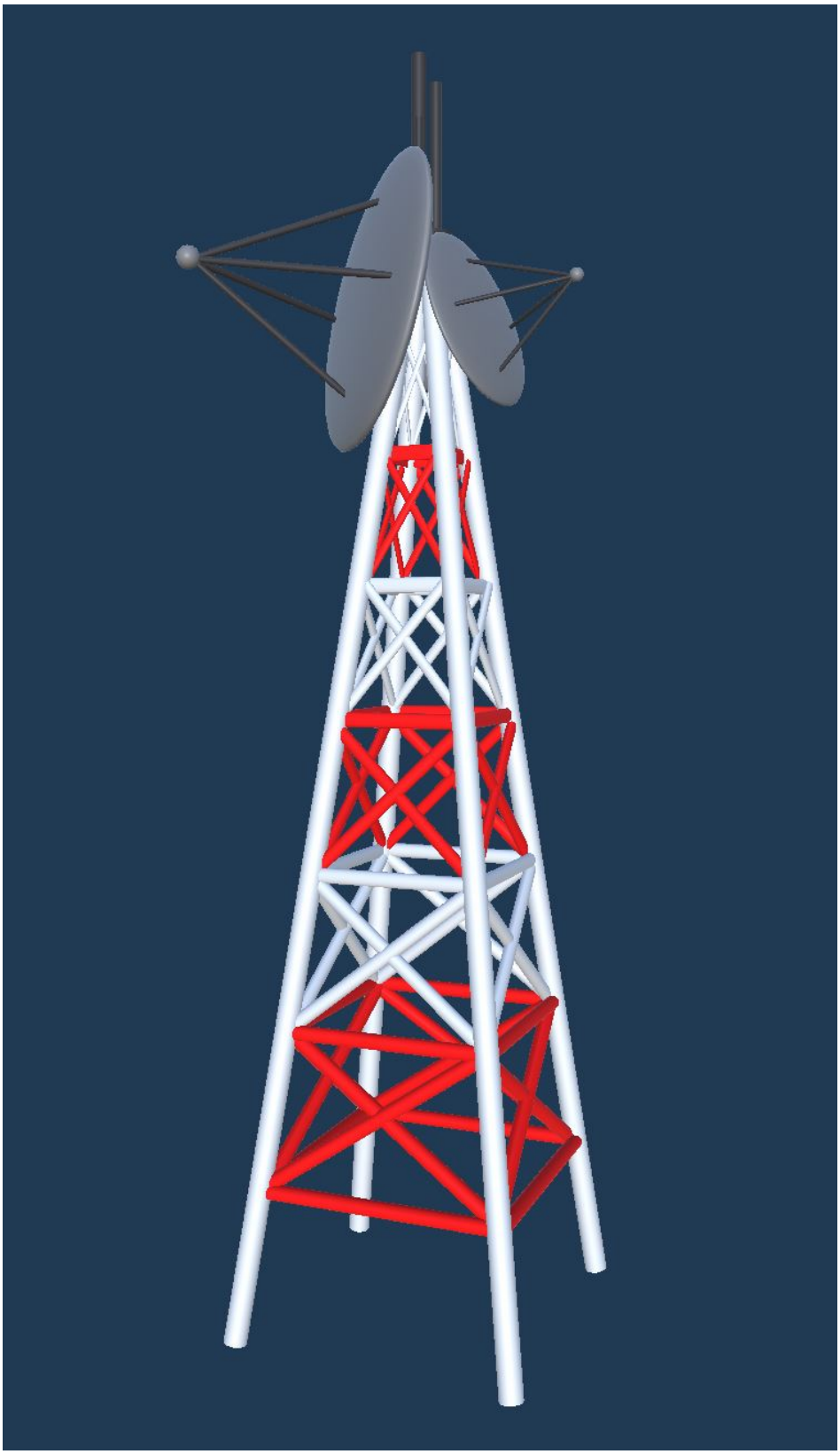

Ilustración 5.24: Modelo nuevo de los sensores.

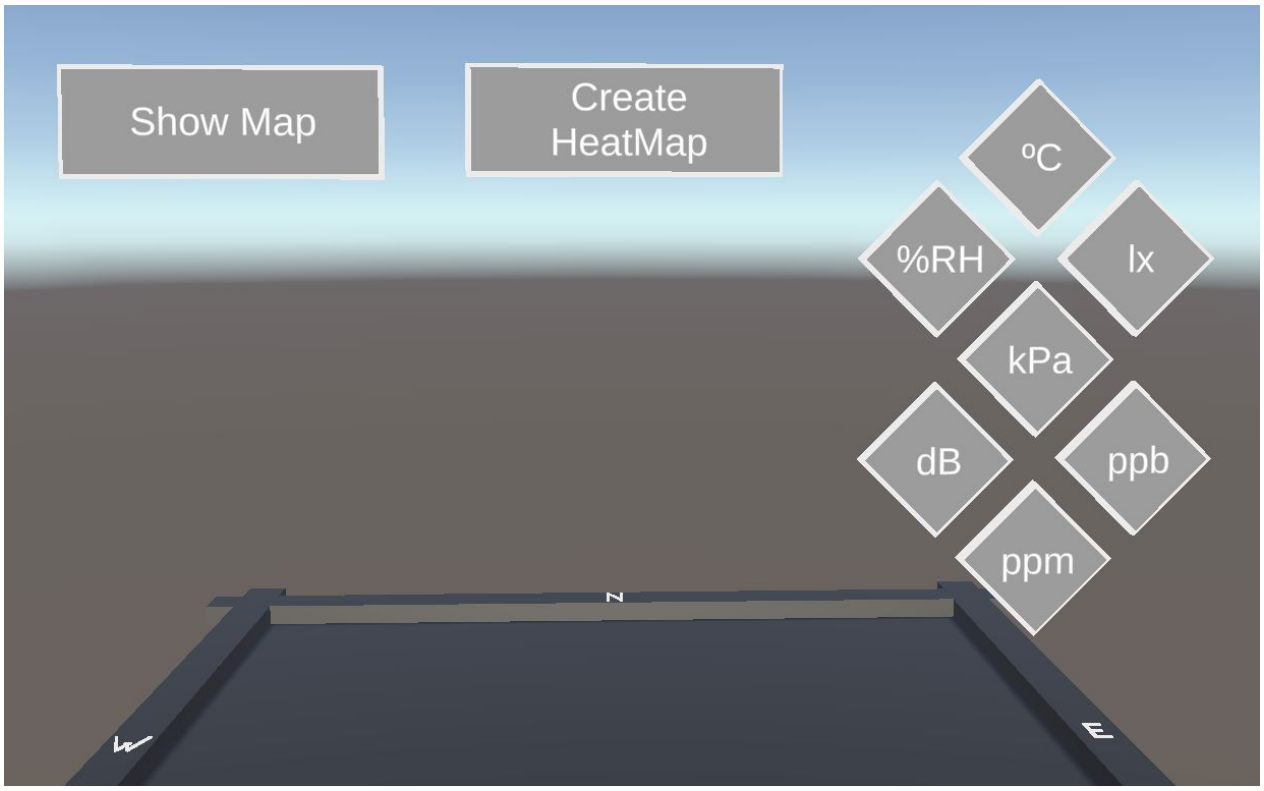

Ilustración 5.25: Diseño antiguo del menú.

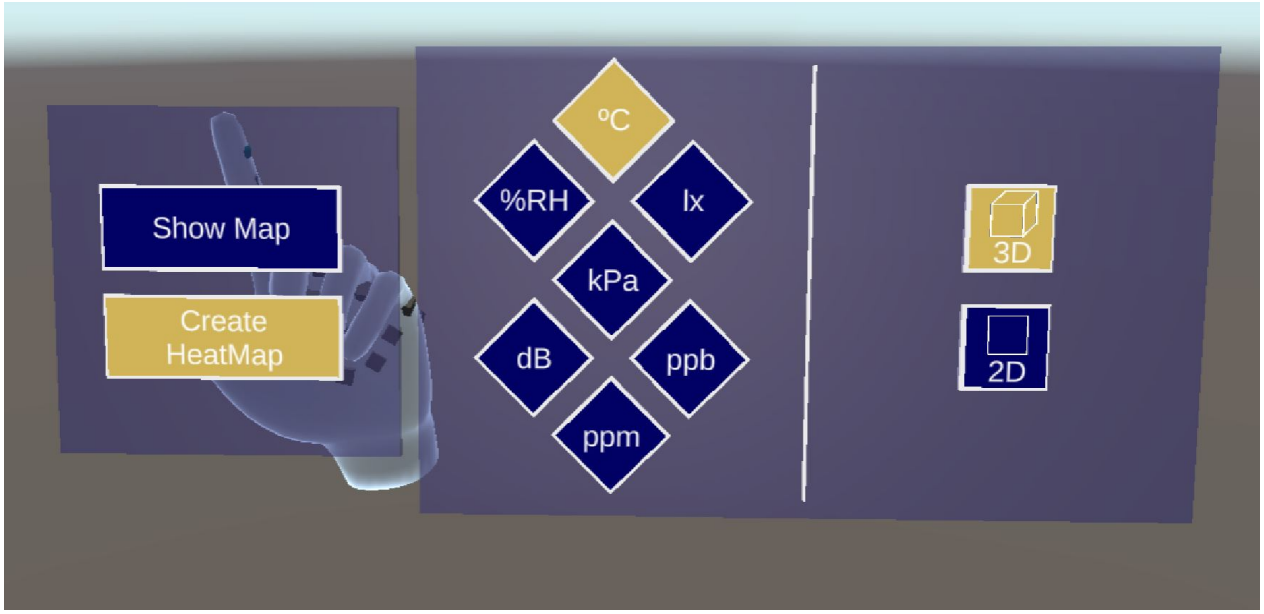

Ilustración 5.26: Diseño nuevo del menú.

Capítulo 6

Cálculos del Mapa de Calor

6.1. Introducción

El objetivo del mapa de calor es claro y conciso, la representación de los datos obtenidos de los sensores, pero a la hora de hacerlo surgen varias dificultades a superar. A continuación se narran, en orden cronológico, los retos a los que el autor tuvo que enfrentarse a la hora de crear el mapa de calor, algo así como su propia Odisea.

6.2. Retos a Afrontar

El primer problema que surge es cómo representar todos los datos de la zona. Para esto se decidió crear una cuadrícula ajustable que contuviera en cada celda el valor de la zona que contiene. Explicación de la cuadrícula en profundidad en el apartado Cuadrícula de Valores.

El segundo problema nace a raíz del primero, cómo conseguir los datos de toda la cuadrícula. Colocar un sensor en cada celda es inviable, tanto económica como materialmente. Como solución a esto se tomó la decisión de colocar el valor de los sensores en la celda a la que pertenecen por su posición en el mapa, y calcular el del resto interpolando los valores de estos. Los métodos usados están explicados en Cálculo de la Cuadrícula de Valores.

El tercer problema aparece al interpolar los datos, para que un dato interpolado tenga un valor con sentido, siendo un valor con sentido aquellos valores similares a lo esperado o a la realidad, debe estar situado entre los puntos que lo interpolan. Entonces el problema es cómo delimitar la zona o cuadrículas a interpolar. La solución que se decidió tomar fue calcular la envolvente convexa formada por la posición de los sensores, y solo calcular las cuadrículas que estuvieran dentro de esta. Para una explicación detallada referirse a la sección Cálculo de la Envolvente Convexa.

El cuarto problema, a diferencia de los anteriores, pertenece al ámbito del diseño. Una vez conseguidos los valores de cada zona hubo que decidir la forma en la que representar los

datos. Para ello se decidió crear un objeto cuadrado de las dimensiones de las celdas en cada una de ellas, al cual se le otorgaría un color y una altura dependiendo del valor de la celda. Más información al respecto en Cálculo de la Altura y el Color.

El quinto y último problema al que el autor tuvo que enfrentarse, surge ante la dificultad de ver el mapa de la isla a través del mapa de calor provocando que no se pudiera distinguir la zona a la que representaba. Se intentaron diferentes estrategías, como aumentar la transparencia de los objetos de las celdas para poder ver a través de ellos, pero esto dificultaba la percepción de la altura de las mismas. Así que se decidió crear un nuevo modo 2D del mapa de calor, en el cual la altura desaparecería y solo estaría representado por el color, al cuál se le aumentó la transparencia.

6.3. Cuadrícula de Valores

Como fue mencionado anteriormente, la cuadrícula de valores fue la estrategia elegida para el almacenamiento y representación de los datos. Se puede visualizar de dos maneras, a nivel de código y a nivel físico.

El nivel físico representado por una cuadrícula (Ilustración 6.1) creada por líneas verticales verdes y horizontales rojas dividen la zona en sectores o celdas.

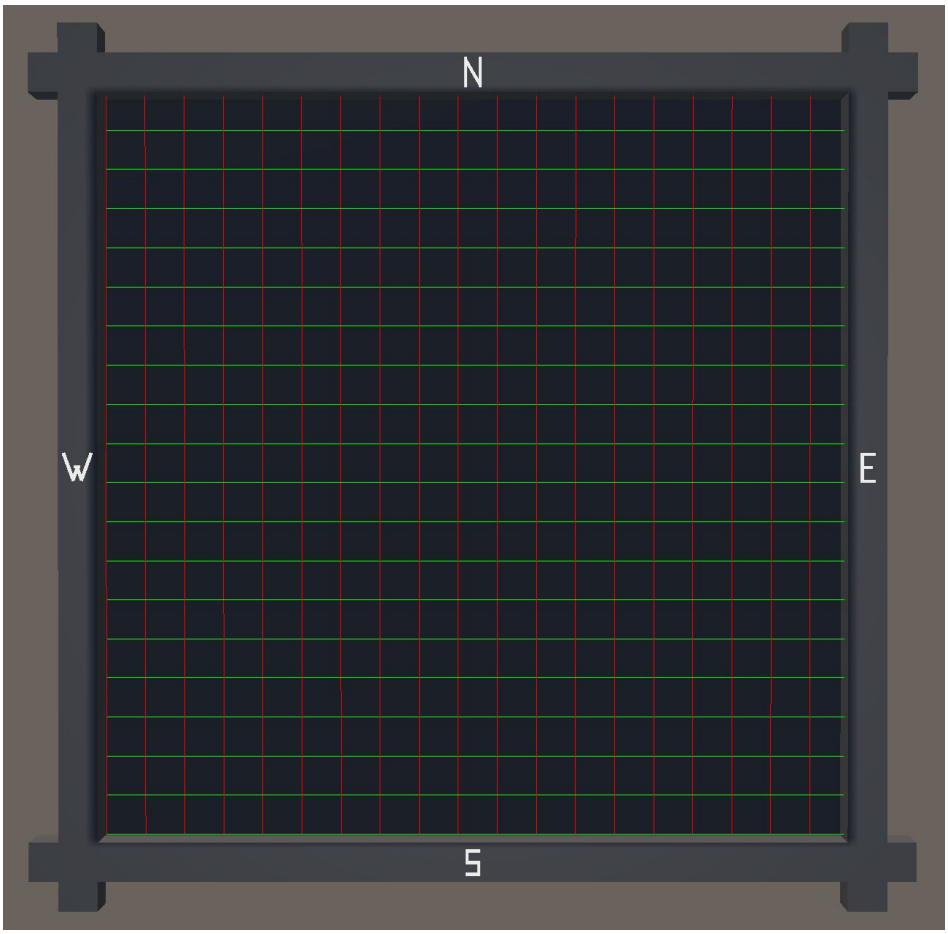

Ilustración 6.1: Representación visual de la cuadrícula.

Por otro lado, a nivel de código, la cuadrícula de valores es una matriz que almacena los valores (Ilustración 6.21). Cada posición de la matriz representa a la correspondiente zona delimitada por la representación física.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	21,29089	0	0	0	21,6945	0	0	0	0		
0	0	0	0	20,91229	20,93747	21,10009	20,15436	19,36237					
0	0	0	0	20,5968	20,66918	18,75121	18,5079	18,14015					
0	0	0	0	19,54802	18,57269	18,14474	17,35647	16,66061					
0	0	0	19,34642	19,36708	18,14301	17,82758	16,60526	14,					
0	0	0	19,1	18,35053	17,92504	17,17205	14,57453	14,33289					
0	0	18,21325	17,40436	16,91725	17,70453	14,44273	13,99685						
20,5136	0												
0	0	0	16,84853	16,15379	15,26935	13,35518	12,93623	12,					
19,92891	0												
0	0	0	19,283	15,34935	14,2288	11,81548	11,32731	11,33416					
0	0	0	16,65502	15,66802	14,17394	12,03433	8,971127	10,					

Ilustración 6.2: Matriz de datos.

6.4. Cálculo de la Envolvente Convexa

La envolvente convexa (Ilustración 6.3) de un conjunto de puntos X es la intersección de todos los conjuntos convexos que contienen a X.

Otra forma de entender el concepto es:

La envolvente convexa de un conjunto de puntos X es el polígono de menor número de aristas que contiene a X.

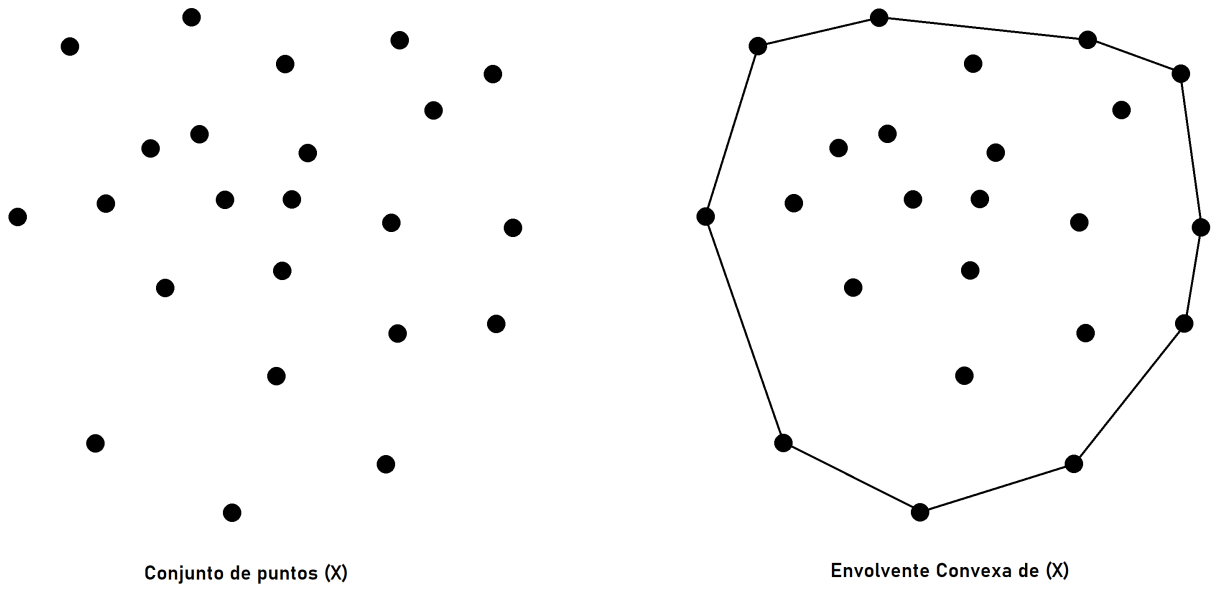

Ilustración 6.3: Representación de la envolvente convexa del conjunto de puntos (X).

Hay diferentes algoritmos (Tabla 6.1) que permiten el cálculo de la envolvente convexa, entre ellos destacan los siguientes.

Algoritmos		
Nombre	Complejidad esperada	Complejidad peor caso
Gift Wrapping	$O(nh)$	$O(n^2)$
Método de Graham	$O(nlogn)$	$O(n)$
Quickhull	$O(nlogn)$	$O(n^2)$
Divide y vencerás	$O(nlogn)$	$O(nlogn)$

Cuadro 6.1: Tabla de complejidades de los algoritmos

En este proyecto se decidió implementar un algoritmo basado en la teoría del Gift Wrapping algorithm pero modificando ligeramente la implementación.

El primer paso para poder poder comenzar a calcular el polígono, es encontrar un punto que pertenezca al mismo. Hay una serie de puntos que podemos estar seguros de que pertenecen a la envolvente convexa, y estos son los puntos más alejados respecto a un eje de coordenadas, es decir, aquellos puntos que tengan la componente X o Y más alta o baja.

Se puede entender también, en los ejes cardinales, como aquel punto que se encuentre más al norte, sur, este u oeste.

En la implementación se decidió buscar el punto con la menor componente X, como se puede apreciar en la (Ilustración 6.4).

```
1 reference
private int LeftMostPoint(Vector2[] allPoints)
{
    int currentMostLeftpoint = 0;
    for (int i = 1; i < allPoints.Length; i++)
    {
        if (allPoints[currentMostLeftpoint].x > allPoints[i].x)
        {
            currentMostLeftpoint = i;
        }
    }
    return currentMostLeftpoint;
}
```

Ilustración 6.4: Código del método LeftMostPoint().

La idea del algoritmo implementado (Ilustración 6.5) es, partiendo del punto ya encontrado, calcular el ángulo que forma la recta, formada por el punto actual y el anterior, y la recta, formada por el punto anterior y el posible siguiente. Se repite este proceso con todos los puntos del conjunto y se elige como siguiente punto el que tenga el mayor ángulo convergente.

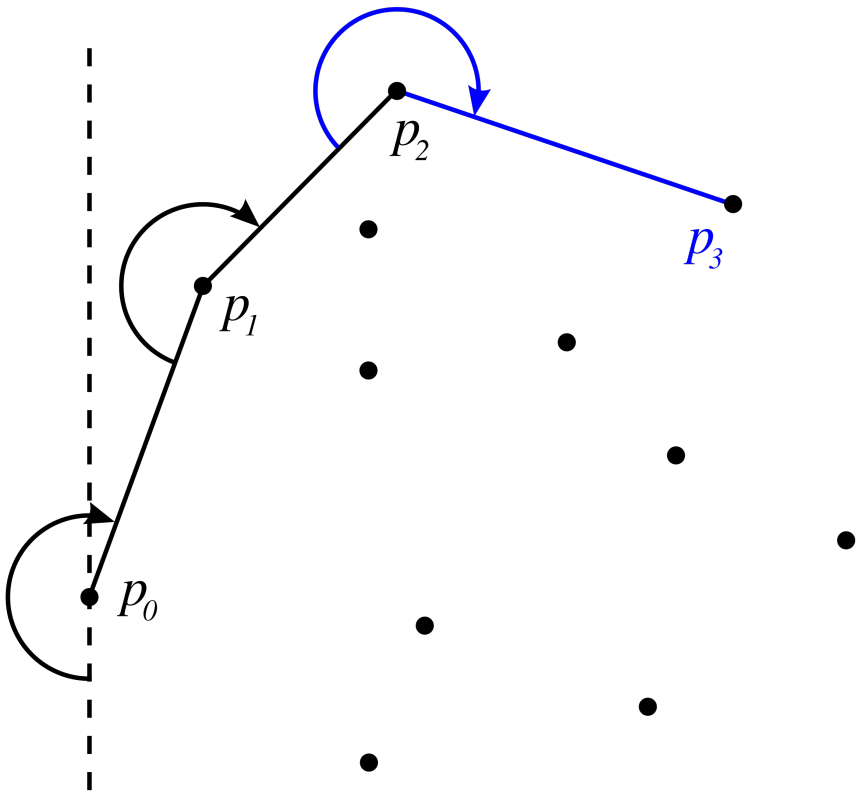

Ilustración 6.5: Algoritmo GiftWrapping.

El algoritmo tiene como condición de parada que el punto final sea igual al inicial. Ya que esto da lugar a un figura cerrada.

Se puede observar el método CalculateConvexHull() (Ilustración 6.6), el cúal se encarga de todo el cálculo mencionado anteriormente.

```
1 reference
public Vector2[] CalculateConvexHull(Vector2[] allPoints)
{
    Vector2 pointOnHull = allPoints[LeftMostPoint(allPoints)];
    Vector2 endPoint = new Vector2(0, 0);

    convexHullPoints = new Vector2[allPoints.Length];

    convexHullPoints[0] = pointOnHull;
    int i = 0;

    while (endPoint != convexHullPoints[0])...

    Vector2[] convexHull = FillConvexHull();
    return convexHull;
}
```

Ilustración 6.6: Código del método CalculateConvecHull() con el while() contraído.

El método recibe un array de vectores de dos dimensiones, estos son el conjunto de puntos del cuál se quiere calcular la envolvente convexa. El método se puede dividir en tres partes diferentes, antes del bucle while(), el propio bucle, y después del bucle.

Antes del bucle se encuentran la mayoría de las declaraciones e inicializaciones. Lo primero que se inicializa es el punto inicial, haciendo una llamada al método LeftMostPoint() que comentamos anteriormente. Seguidamente se hacen una serie de declaraciones, una variable para almacenar el punto que se está probando, un vector para guardar los puntos ya aceptados, una variable para guardar el ángulo y una variable para iterar. Además se guarda el punto ya calculado en la primera posición del resultado.

Después del bucle se hace una llamada a un método llamado FillConvexHull() (Ilustración 6.7), este se encarga de devolver un vector con el resultado con las posiciones justas, esto se debe a que antes de calcular la envolvente no se puede saber el número de vértices que tendrá el resultado, por lo que se declara un vector del tamaño del conjunto de puntos, ya que es el peor caso, y posteriormente se ajusta. Finalmente devuelve el resultado.

```
1 reference
private Vector2[] FillConvexHull()
{
    Vector2[] newConvexHull = new Vector2[CountNumberOfNodesInConvexHull()];
    for (int i = 0; i < newConvexHull.Length; i++)
    {
        newConvexHull[i] = convexHullPoints[i];
    }
    return newConvexHull;
}
```

Ilustración 6.7: Código del método FillConvexHull().

El bucle se encarga de calcular los puntos que pertenecen a la envolvente, esto se lleva a cabo con dos bucles realmente.

Hay un primer bucle while() (Ilustración 6.8) encargado de rellenar el vector resultado y que tiene como condición de parada que el último punto sea igual al primero. Hay un primer if() que comprueba que si se ha pasado por todos los puntos del conjunto, para en ese caso parar. A continuación se crea y se hace reset de una variable que recorre los puntos dentro del segundo bucle y otra que almacena el mayor ángulo de la iteración. Después del segundo bucle se guarda el punto obtenido en el resultado y se itera el bucle.

```
while (endPoint != convexHullPoints[0])
{
    if(i == allPoints.Length)
    {
        return convexHullPoints;
    }

    endPoint = allPoints[0];
    double currentNextPointPolarAngle = 0;

    for (int j = 0; j < allPoints.Length; j++)...

    i++;
    pointOnHull = endPoint;
    convexHullPoints[i] = pointOnHull;
}
```

Ilustración 6.8: Código del método CalculateConvecHull() con el for() contraido.

El segundo bucle es un for() (Ilustración 6.9) que recorre todo el conjunto de puntos y

calcula el ángulo que forman este punto, y los dos últimos puntos añadidos al vector resultado. Primero se comprueba si es la primera iteración, ya que en la primera no existen dos puntos anteriores, solo uno. En caso de que sea la primera, se usa como punto anterior uno igual pero desplazado en el eje Y. Si no se calcula el ángulo con los tres puntos. Entonces se comprueba si el nuevo ángulo calculado es mayor que nuestro mayor ángulo calculado en la iteración. En caso de serlo se guarda como el mayor y se continúa. Este if() también tiene una condición que comprueba que el nuevo ángulo sea mayor a 0, esto sirve para evitar los ángulos negativos ya que no serían convergentes.

```
for (int j = 0; j < allPoints.Length; j++)
{

    float possibleNewPolarAngle;

    if (i == 0)
    {
        possibleNewPolarAngle = CalculatePolarAngle(convexHullPoints[i], convexHullPoints[i] + new Vector2(0, -5), allPoints[j]);
    }
    else
    {
        possibleNewPolarAngle = CalculatePolarAngle(convexHullPoints[i], convexHullPoints[i - 1], allPoints[j]);
    }

    if (possibleNewPolarAngle > currentNextPointPolarAngle && possibleNewPolarAngle > 0)
    {
        currentNextPointPolarAngle = possibleNewPolarAngle;
        endPoint = allPoints[j];
    }
}
```

Ilustración 6.9: Código del bucle for() del método CalculateConvecHull().

6.5. Cálculo de la Cuadrícula de Valores

La cuadrícula de valores, como concepto físico, es una cuadrícula que divide el mapa en diferentes zonas, en cuanto a código, se refiere a una matriz que contiene todos los valores de dichas zonas.

Toda la lógica y cálculos de esta cuadrícula están dentro de la clase de GridMap, y tiene como método principal CreateGridMap() (Ilustración 6.10).

```
1 reference
public void CreateGridMap(float cellSize = 0.1f)
{
    this.cellSize = cellSize;

    Vector2 gridDimensions = CalculateGrid();

    this.height = (int)gridDimensions.x;
    this.width = (int)gridDimensions.y;
    this.gridArray = new float[width, height];

    DrawGrid(Color.green);

    GetSensorData();

    SetBaseSensorsData();
    CalculateConvexHull();
    SetSensorsOnGrid();
}
```

Ilustración 6.10: Código del método CreateGridMap().

Este método realiza la función de punto central y se encarga de llamar a la mayoría de los demás métodos, además de inicializar los objetos de la clase. Recibe como parámetro un número que representa el tamaño de las celdas, en caso de que no fuera especificado, cuenta con un valor predeterminado.

Los métodos a los que llama son los siguientes:

- ***CalculateGrid(Ilustración 6.11):*** Este método se encarga de calcular la cuadrícula a partir de las esquinas del mapa y el tamaño de las celdas. Devuelve un vector de dos dimensiones con el número de celdas en cada eje.

```
1 reference
private Vector2 CalculateGrid()
{
    float heightDistance = Math.Abs((mapTopLeftCorner.z - mapBottomRightCorner.z) / cellSize);
    float widthDistance = Math.Abs((mapTopLeftCorner.x - mapBottomRightCorner.x) / cellSize);

    return new Vector2(widthDistance, heightDistance);
}
```

Ilustración 6.11: Código del método CalculateGrid().

- ***DrawGrid(Ilustración ??):*** Tiene la función de dibujar la cuadrícula sobre el mapa usando líneas verdes y rojas, no afecta al cálculo, es una representación física de la cuadrícula.
- ***GetSensorData(Ilustración 6.12):*** Recoge los datos de los sensores realizando una llamada a la clase JsonDataExtraction.

```
2 references
public void GetSensorData()
{
    sensorList = JsonUtility.FromJson<SensorsList>(jsonDataExtraction.GetJsonData());
}
```

Ilustración 6.12: Código del método GetSensorData().

- ***SetBaseSensorsData(Ilustración ??):*** Se encarga de guardar los sensores en un vector, su posición en la cuadrícula y su valor. Para esto hace uso, primero del método GetWorldPosition() para encontrar la posición de los sensores en relación a la cuadrícula, y a continuación del SetSensorValue() para colocar el valor de un sensor en su celda dependiendo de tipo de dato. Esto lo repite para cada sensor.
- ***CalculateConvexHull(Ilustración 6.13):*** Llama a la clase con la que comparte nombre, CalculateConvexHull, pasándole como parámetros el conjunto de puntos, y guarda el resultado obtenido.

```
1 reference
private void CalculateConvexHull()
{
    ConvexHullCalculation convexHull = new ConvexHullCalculation();
    convexHullFigure = convexHull.CalculateConvexHull(listOfSensorsInScene);
}
```

Ilustración 6.13: Código del método CalculateConvexHull().

- ***SetSensorsOnGrid(Ilustración 6.14):*** Vuelve a funcionar como centralización de las llamadas a otros métodos, para cumplir la función de rellenar la matriz de la cuadrícula de valores, además de activar la creación del mapa de calor mediante la clase HeatMapController. Los métodos que contiene son los siguientes.

```
2 references
private void SetSensorsOnGrid()
{
    SetBaseValues(baseSensors);
    InterpolateValues(baseSensors);
    ShowGridMatrix();

    //Creates the HeatMap
    heatMapController.GetComponent<HeatMapController>().SetParameters(width, height, cellSize, gridArray, sensorDataType);
}
```

Ilustración 6.14: Código del método SetSensorsOnGrid().

- ***SetBaseValues(Ilustración 6.15):*** Añade los valores de los sensores en los puntos de la matriz a los que corresponden.

```
1 reference
private void SetBaseValues(Point[] basePoints)
{
    for (int i = 0; i < basePoints.Length; i++)
    {
        gridArray[basePoints[i].x, basePoints[i].y] = basePoints[i].value;
    }
}
```

Ilustración 6.15: Código del método SetBaseValues().

- ***InterpolateValues(Ilustración 6.16):*** Se encarga de la lógica detrás de la interpolación de los valores y posteriormente los sitúa en la matriz. Para esto hace uso de una serie de métodos con diferentes funciones.

```
1 reference
private void InterpolateValues(Point[] basePoints, int numberOfPoints = 3)
{
    for (int i = 0; i < width; i ++)
    {
        for (int j = 0; j < height; j++)
        {
            if (!IsBasePoint(i, j, basePoints) && IsPointInsideConvexHull(new Vector2(i,j)))
            {
                Point newPoint = new Point();
                newPoint.x = i;
                newPoint.y = j;

                Point[] closestPoints = GetClosestPoints(newPoint, basePoints, numberOfPoints);

                newPoint.value = GetInterpolatedValue(newPoint, closestPoints);
                gridArray[i, j] = newPoint.value;
            }
        }
    }
}
```

Ilustración 6.16: Código del método InterpolateValues().

- ***IsBasePoint(Ilustración 6.17):*** Comprueba si el valor pasado por parámetros en una valor base, es decir, uno de los sensores.

```
1 reference
private Boolean IsBasePoint(int x, int y, Point[] basePoints)
{
    foreach (Point point in basePoints)
    {
        if(point.x == x && point.y == y)
        {
            return true;
        }
    }
    return false;
}
```

Ilustración 6.17: Código del método IsBasePoint().

- ***IsPointInsideConvexHull(Ilustración 6.18):*** Tiene la función de comprobar si el punto pasado por parámetros está dentro de la envolvente convexa. Esto se lleva a cabo en dos pasos, primero comprueba que el punto se encuentra dentro con respecto al eje Y, comprobando si existe un punto de

la envolvente mayor y menor respecto a este eje. Una vez comprobado esto, comprueba si el número de aristas a su izquierda es par o impar, si es impar está dentro y si es par no.

```
1 reference
private bool IsPointInsideConvexHull(Vector2 point)
{
    bool result = false;
    int j = convexHullFigure.Length - 1;
    for (int i = 0; i < convexHullFigure.Length; i++)
    {
        if (convexHullFigure[i].y < point.y && convexHullFigure[j].y >= point.y ||
            convexHullFigure[j].y < point.y && convexHullFigure[i].y >= point.y)
        {
            if (convexHullFigure[i].x + (point.y - convexHullFigure[i].y) /
                (convexHullFigure[j].y - convexHullFigure[i].y) *
                (convexHullFigure[j].x - convexHullFigure[i].x) < point.x)
            {
                result = !result;
            }
        }
        j = i;
    }
    return result;
}
```

Ilustración 6.18: Código del método IsPointInsideConvexHull().

- ***GetClosestPoints(Ilustración ??):*** Calcula los X sensores más cercanos al punto pasado por parámetros. Para realizar este cálculo se recorre la lista de sensores y se hace uso de un método llamado CalculateEuclidianDistance() (Ilustración 6.19), el cuál calcula la distancia euclídea entre el punto y un sensor. El método devuelve un vector de puntos, conteniendo los X sensores con la menor distancia euclídea, X es un valor elegido en la llamada al método.

```
1 reference
private float CalculateEuclidianDistance(Point point1, Point point2)
{
    double x = Math.Pow(Math.Abs(point1.x - point2.x), 2);
    double y = Math.Pow(Math.Abs(point1.y - point2.y), 2);
    return (float)Math.Sqrt(x + y);
}
```

Ilustración 6.19: Código del método CalculateEuclidianDistance().

- ***GetInterpolatedValue(Ilustración 6.20):*** Recibe como parámetros un

punto y un vector con los puntos más cercanos a este, a partir de esto calcula el valor mediante interpolación.

```
1 reference
private float GetInterpolatedValue(Point point, Point[] closestPoints)
{
    float numerator = 0;
    float denominator = 0;

    foreach (Point closePoint in closestPoints)
    {
        numerator += (float)(closePoint.value / closePoint.distance);
        denominator += (float)(1 / closePoint.distance);
    }

    return numerator / denominator;
}
```

Ilustración 6.20: Código del método GetInterpolatedValue().

- ***ShowGridMatrix(Ilustración 6.21):*** Muestra en consola la matriz con los valores para facilitar el proceso de debuggear.

```
0       |0       |0       |0       |0       |0       |0       |0       |0       |0       |0       |0       |0       |0
0       |0       |0       |21,29089       |0       |0       |0       |21,6945       |0       |0       |0       |0
0       |0       |0       |0       |20,91229       |20,93747       |21,10009       |20,15436       |19,36237
0       |0       |0       |0       |20,5968       |20,66918       |18,75121       |18,5079       |18,14015
0       |0       |0       |0       |19,54802       |18,57269       |18,14474       |17,35647       |16,66061
0       |0       |0       |19,34642       |19,36708       |18,14301       |17,82758       |16,60526       |14,
0       |0       |0       |19,1       |18,35053       |17,92504       |17,17205       |14,57453       |14,33289
0       |0       |18,21325       |17,40436       |16,91725       |17,70453       |14,44273       |13,99685
|20,5136       |0       |
0       |0       |0       |16,84853       |16,15379       |15,26935       |13,35518       |12,93623       |12,
|19,92891       |0       |
0       |0       |0       |19,283 |15,34935       |14,2288       |11,81548       |11,32731       |11,33416
0       |0       |0       |16,65502       |15,66802       |14,17394       |12,03433       |8,971127       |10,
```

Ilustración 6.21: Código del método ConsoleMatrix().

6.6. Cálculo de la Altura y el Color

Cada celda del mapa de calor tiene una altura y un color en relación al valor de la misma, por lo que es necesario realizar una serie de cálculos para hallarlos.

En un primer momento se creó un método llamado CalculateHeightScale() (Ilustración 6.22) que creaba una escala tomando como referencia el valor más alto y bajo de la cuadrícula de valores.

```
0 references
private Vector2 CalculateHeightScale()
{
    float[] arrayOfMinimuns = new float[gridArray.Length];
    float[] arrayOfMaximuns = new float[gridArray.Length];

    int i = 0;
    foreach (float line in gridArray)
    {
        float min = Mathf.Min(line);
        arrayOfMinimuns[i] = min;
        float max = Mathf.Max(line);
        arrayOfMaximuns[i] = max;
        i++;
    }

    float minValue = Mathf.Min(arrayOfMinimuns);
    float maxValue = Mathf.Max(arrayOfMaximuns);
    Vector2 heightScale = new Vector2(minValue, maxValue);

    return heightScale;
}
```

Ilustración 6.22: Código del método CalculateHeightScale().

Pero esto fue posteriormente descartado y sustituido por GetHeightScale().

Este nuevo método lo primero que hace es establecer una escala base para los valores, esta escala sirve para dar un máximo y un mínimo al tamaño y color. Como cada tipo de dato tiene valores diferentes, por ejemplo, los valores de la temperatura suelen rondar entre los 0ºC y 40ºC mientras que los de la luminosidad pueden estar sobre los 50000 lx, se creó una escala personalizada para cada uno, la cuál posteriormente se mapea sobre la escala base.

Al crear la celda del mapa de calor, se crea con un largo y ancho ya definido en la creación de la cuadrícula de valores, el alto se calcula individualmente para cada una llamando al método CalculateTileHeight() (Ilustración 6.23), al cuál se le pasa el valor de la celda.

Este método comprueba si el mapa de calor se encuentra en modo 2D o 3D. En caso de estar en 2D devuelve un valor predefinido. En cambio, si es 3D, devuelve el valor mapeado en la escala base. La variable “slope” es calculada en el constructor debido a que es un cálculo constante para todas las celdas.

```
2 references
private float CalculateTileHeight(float value)
{
    if (mode2DActive)
    {
        return height2DMode;
    }

    return baseScale.x + slope * (value - heightScale.x);
}
```

Ilustración 6.23: Código del método CalculateTileHeight().

Capítulo 7

Conclusiones y trabajo a futuro

7.1. Conclusiones

La realización de este proyecto me ha dado la oportunidad de ampliar mis conocimientos y habilidades usando el motor Unity, en concreto para aplicaciones de realidad mixta. Antes del comienzo del proyecto mis conocimientos sobre ellos eran prácticamente inexistentes, pero ahora tengo confianza de que, aunque siempre hay espacio para aprender y mejorar, ahora poseo habilidad suficiente como para realizar proyectos más ambiciosos de manera competente.

Al comenzar el proyecto me encontraba realizando una beca en la empresa colaboradora, esto provocó que tuviera que buscar el modo de compaginar las horas de trabajo con el desarrollo del proyecto, lo que dio lugar a avanzar el proyecto por las tardes. Aunque esto aumentaba la dificultad del desarrollo considerablemente, me permitió aprender a organizarme de manera más eficiente.

Por suerte, la empresa me permitía avanzar el proyecto durante el horario laboral cuando la carga de trabajo era menor. Además, aunque no avanzara en el proyecto, el trabajo que realizaba en la empresa consistía en el desarrollo de aplicaciones de realidad mixta en Unity, por lo que era conocimiento y práctica que ganaba con los programas y herramientas que usaba posteriormente en el proyecto.

A mitad del desarrollo se terminó el periodo de la beca y pude centrar todo mi tiempo en el proyecto.

Este proyecto también me ha permitido practicar mis conocimientos matemáticos y algorítmicos.

Aunque no fueran de gran dificultad, el proyecto tiene bastantes cálculos matemáticos, como el tamaño de la cuadrícula, la interpolación de valores, la alturas y colores del mapa de calor o el mapeo de datos en las escalas.

También hubo que investigar, modificar e implementar algoritmos para el cálculo de la

envolvente convexa y para comprobar si un punto se encontraba dentro de la misma.

Por último mencionar que el proyecto ha terminado siendo una aplicación que aunque tiene bastantes cosas a mejorar, ya sea por el límite de tiempo o por mi inexperiencia con el entorno y herramientas, ha acabado siendo una aplicación funcional con claros usos en la vida real, ya sea como herramienta de enseñanza o como método de control y prevención de incendios u otros desastres naturales.

7.2. Trabajo a futuro

Este proyecto, aunque totalmente funcional, podría ser ampliado y mejorado teniendo más tiempo y recursos.

Debido al límite de aproximadamente 300 horas para la realización de este proyecto y a mi inexperiencia hubieron ideas que no pudieron ver la luz del día. Entre ellas algunas de las que se plantearon realizar fueron:

- La creación de un histórico para guardar los datos recogidos por los sensores y posteriormente poder consultarlos para su estudio y análisis.

- Haciendo uso de los datos del histórico, poder reflejar los datos en el mapa de calor, ya sea eligiendo una fecha o avanzando/retrocediendo en el tiempo.

- Añadir la capacidad de multiusuario, es decir, poderse conectar varias personas simultáneamente para observar los datos, esto sería un gran añadido especialmente para su uso lectivo.

- Añadir más opciones que permitan mayor control y personalización, como cambiar el color del mapa de calor o el tamaño del mapa.

- Tener distintas localizaciones a las que acceder, el proyecto está preparado solo para la isla de Gran Canaria pero me hubiera gustado añadir otras localizaciones.

- Crear el mapa de calor 2D como una textura, creándolo a partir de la cuadrícula de valores pero usando como celdas los píxeles, y de esta manera poder aplicarlo al mapa topográfico de la isla con el relieve, no solo a su versión en forma de plano.

Capítulo 8

Anexo I: Manual de Instalación

Lamentablemente el proyecto completo no está disponible para la descarga e instalación, esto se debe a que hace uso de varios assets y librerías de pago, propiedad de la empresa colaboradora del proyecto, Singular Factory. Además de esto el proyecto hace uso de algunas librerías que pesan más del límite permitido en GitHub.

Pero en caso de desearse, puede acceder a todo los scripts y assets desarrollados durante el proyecto en mi página personal de GitHub[14], en el proyecto TFG-Sensors[15].

Capítulo 9

Anexo II: Manual de Usuario

Al iniciar la aplicación se encontrará con un marco(Ilustración 9.1) con los ejes cardinales para facilitar la orientación.

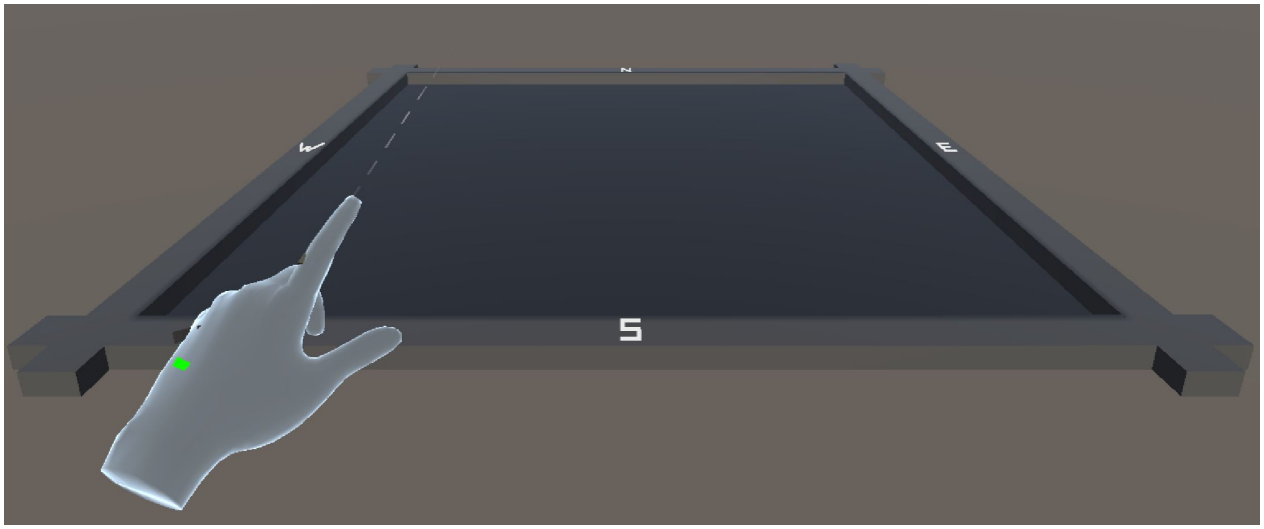

Ilustración 9.1: Marco del mapa con los ejes cardinales.

Para continuar y acceder al menú(Ilustración 9.2) mire la muñeca de una de sus manos y use la otra para pulsar el botón “Start”. Puede volver a acceder al menú cuando y donde quiera volviendo a mirar su muñeca.

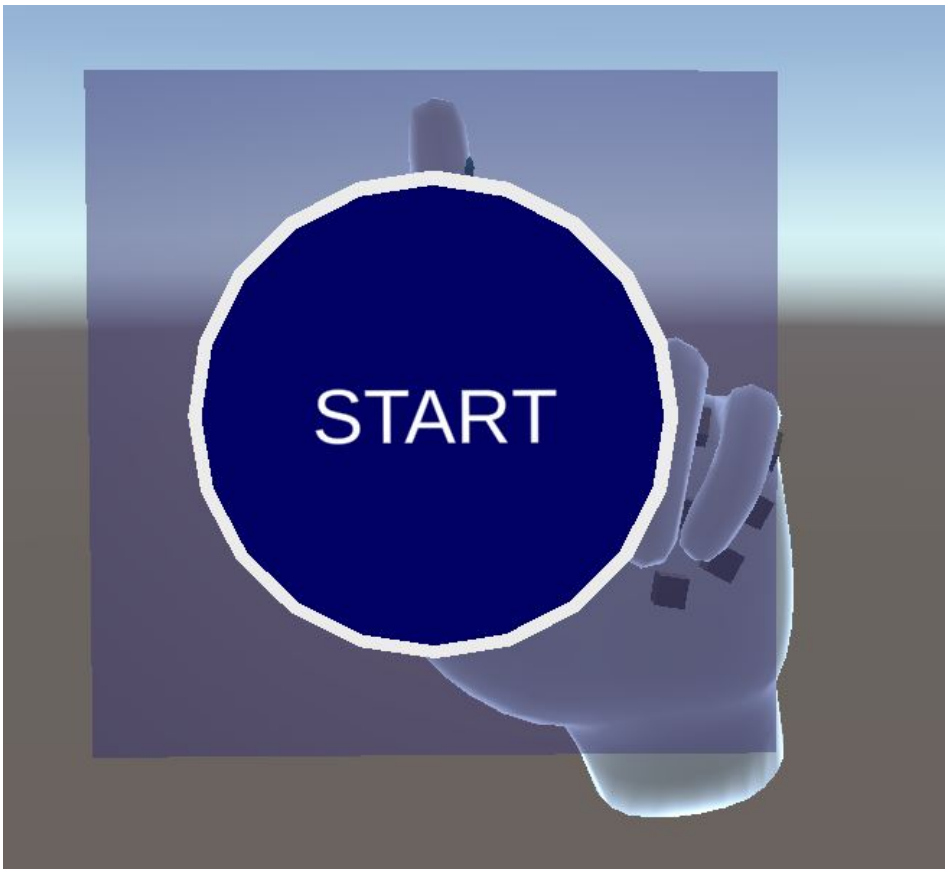

Ilustración 9.2: Botón de inicio del menú.

Tras una pequeña carga el menú cambiará al menú principal(Ilustración 9.3). Desde este nuevo menú puede elegir acceder al mapa o al mapa de calor.

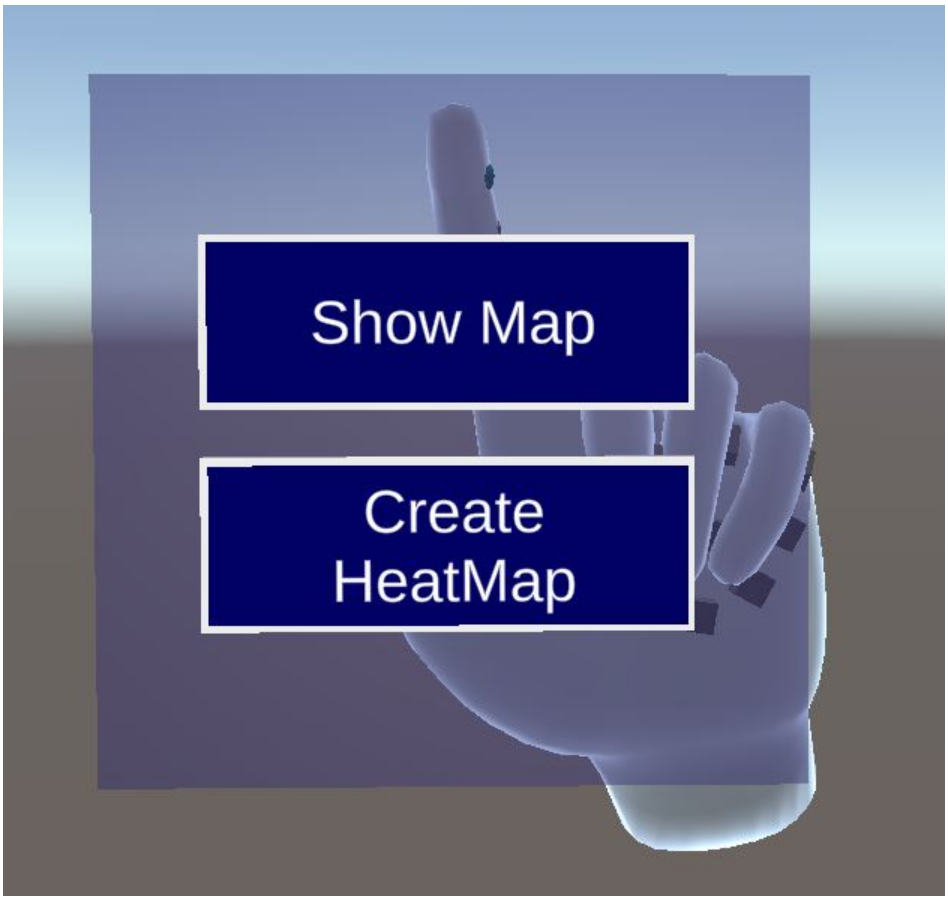

Ilustración 9.3: Menú principal de la aplicación.

Si pulsa el botón “Show Map” se abrirá sobre el marco el mapa topográfico(Ilustración 9.4) de la isla de Gran Canaria con los sensores sobre ella.

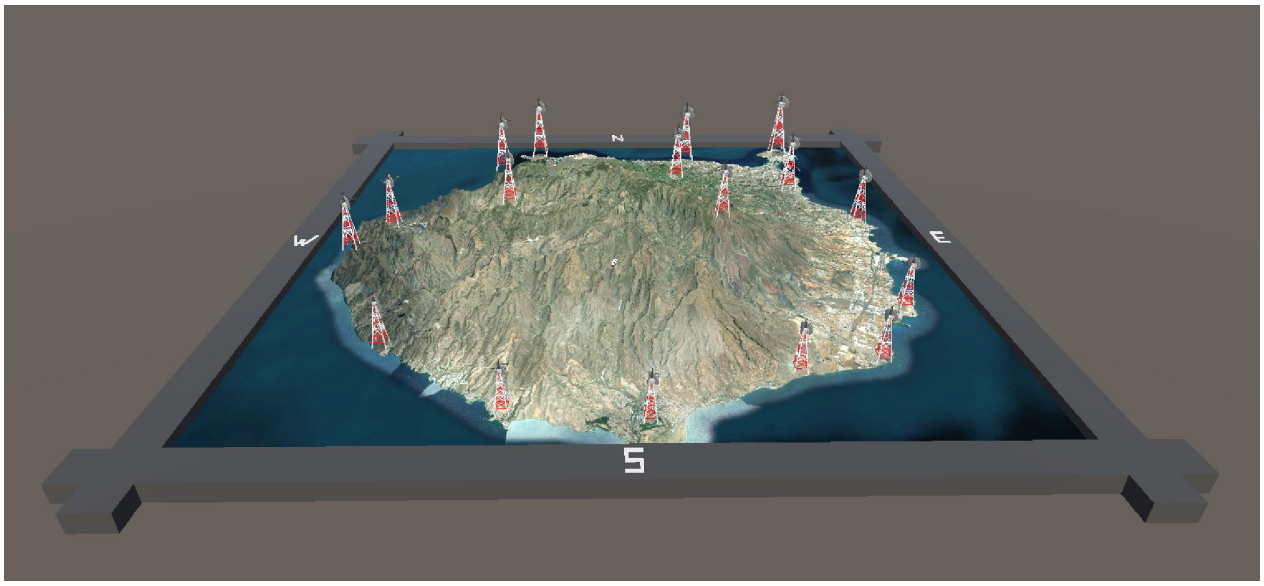

Ilustración 9.4: Mapa topográfico de la isla de Gran Canaria.

Puede acceder a la información del sensor(Ilustración 9.5) tocándolo con la mano.

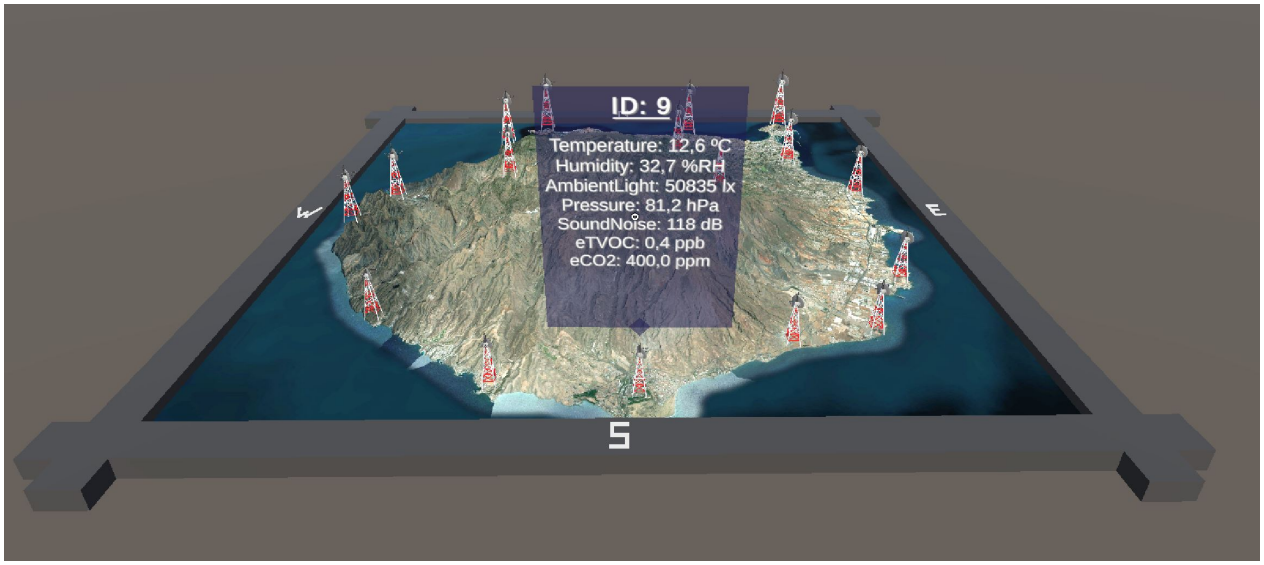

Ilustración 9.5: Panel de información de los sensores.

En cambio, si pulsa el botón “Create HeatMap”, se creará el mapa de calor(Ilustración 9.6) y se le desplegará un nuevo menú para controlar diferentes variables.

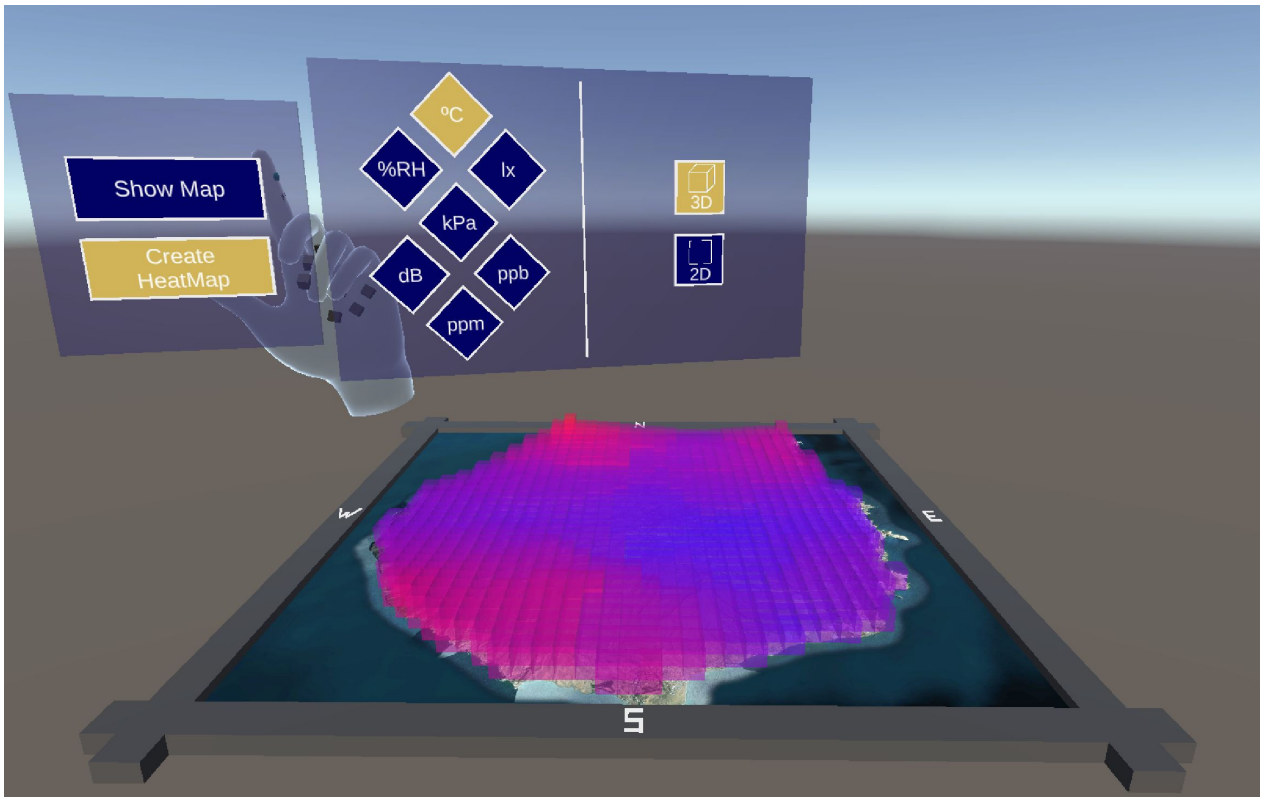

Ilustración 9.6: Mapa de calor con el menú desplegado.

Puede acceder al valor de cada celda(Ilustración 9.7) del mapa de calor tocándola con la mano.

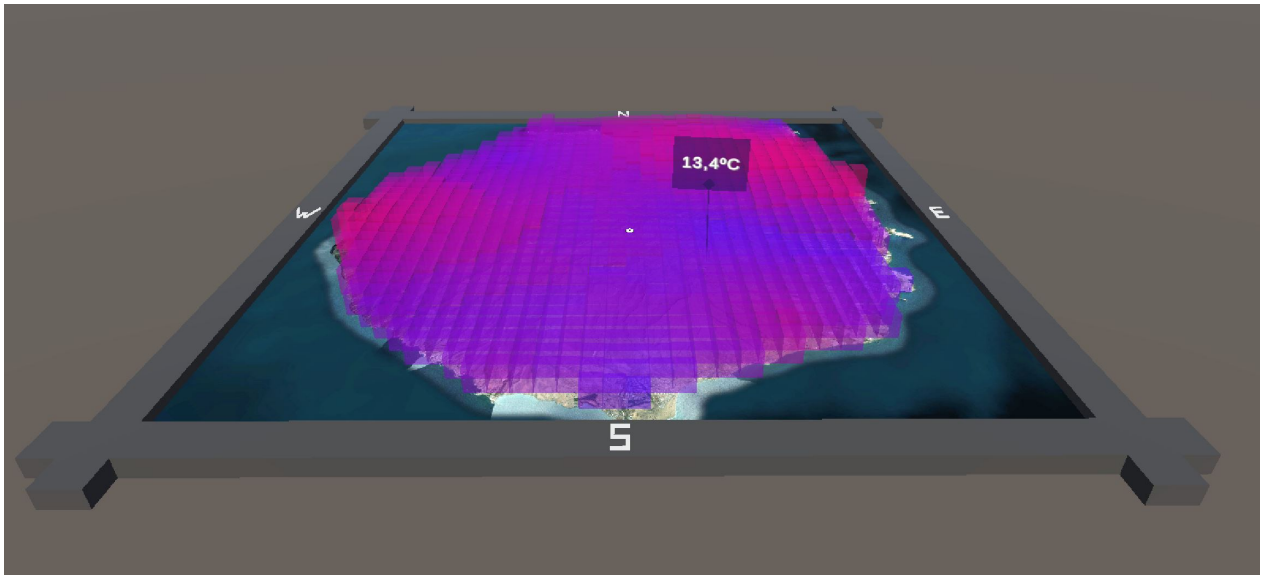

Ilustración 9.7: Panel de información de las celdas del mapa de calor.

Con los botones de la izquierda, los que tienen forma de rombo, puede cambiar el tipo de

datos usado y con los de la derecha, puede cambiar entre modo 2D y 3D(Ilustración 9.8).

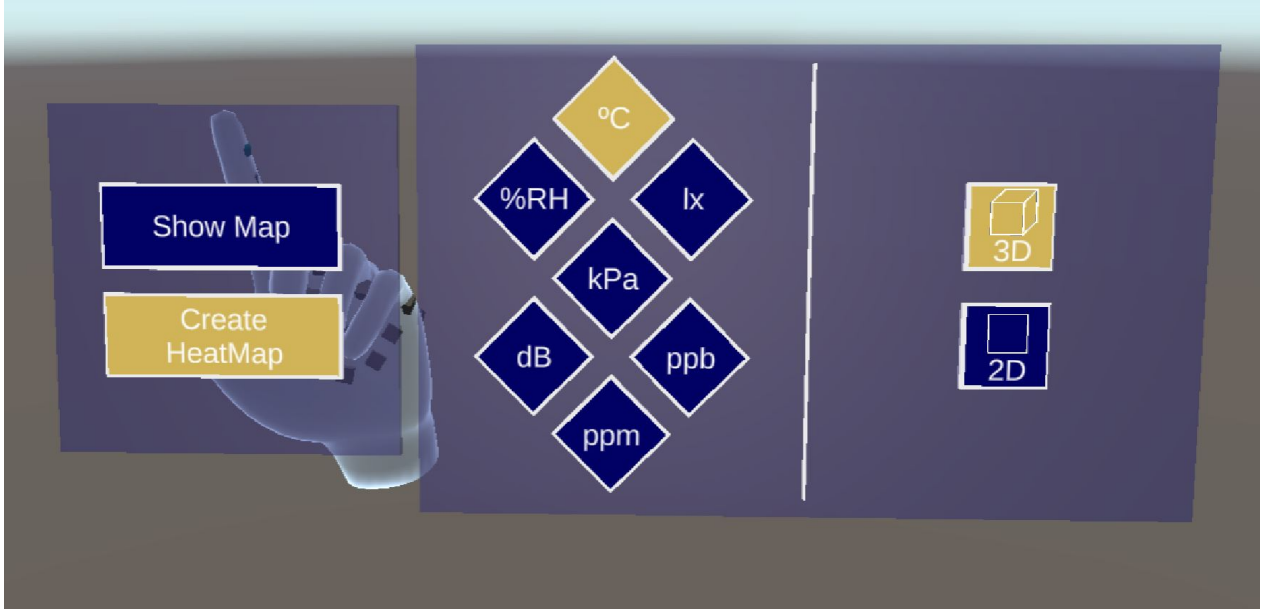

Ilustración 9.8: Menú principal con el menú desplegable del mapa de calor.

Puede desactivar cualquier botón volviendo a pulsar sobre él.

Bibliografía

[1] Unity3D, "Motor y entorno de desarrollo 3d." https://unity.com/es.

[2] VisualStudio, "Entorno de desarrollo." https://visualstudio.microsoft.com/es/.

[3] OnlineMaps, "Asset de unity para trabajar con mapas en realidad virtual." https://infinity-code.com/assets/online-maps.

[4] InfinityCode, "Desarrolladora de el asset onlinemaps." https://infinity-code.com.

[5] MRTK, "Asset de unity para entornos de realidad virtual." https://learn.microsoft.com/es-es/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05.

[6] GitLab, "Aplicación de gestión de versiones." https://about.gitlab.com/.

[7] GitHub, "Alojamiento de proyectos git." https://unity.com/es.

[8] SourceTree, "Aplicación interactuar con git." https://www.sourcetreeapp.com/.

[9] OverLeaf, "Web para la creaciçon de documentos latex." https://www.overleaf.com/.

[10] ClickUp, "Web para la organización de tareas." https://clickup.com/.

[11] StarUML, "Aplicación para la creación de diagramas." https://staruml.io/.

[12] 2JCIE-BU01, "Sensor medidor de factores climáticos." https://components.omron.com/us-en/solutions/sensor/enverioment_seonsor.

[13] Omron, "Empresa productora de los sensores." https://unity.com/es.

[14] G. Carlos, "Página personal de github del autor." https://github.com/CarlosBritoPerez.

[15] G. TFG, "Enlace al proyecto en github." https://github.com/CarlosBritoPerez/TFG-Sensors.