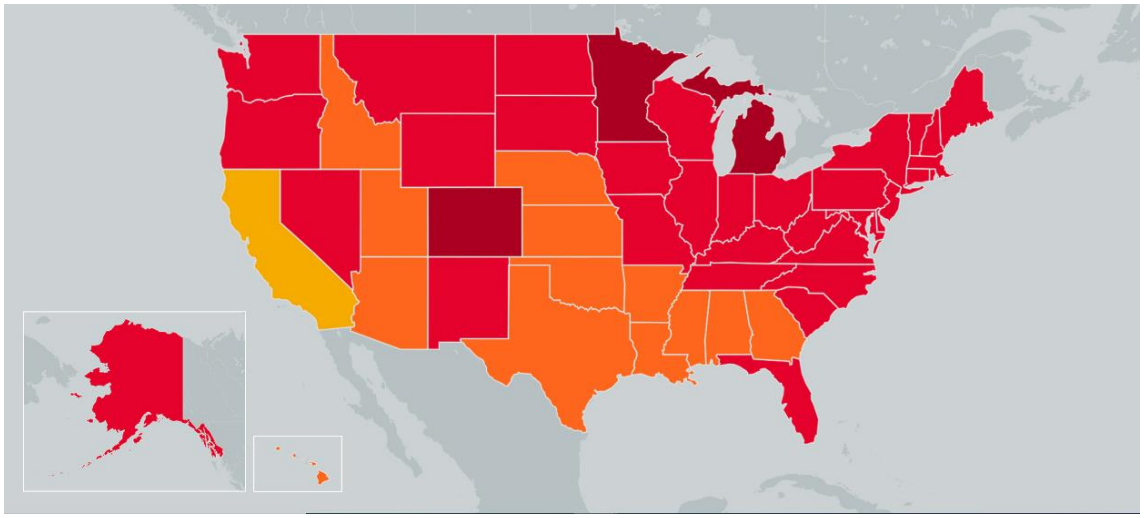
ULPGC
Universidad de Las Palmas de Gran Canaria
Escuela de Ingeniería Informática



GRADO EN INGENIERÍA INFORMÁTICA

Mapa de datos COVID-19 en Estados Unidos

Alumno - Ismael Aarab Umpiérrez
DNI – 45309997C

Tutores:
Agustín Rafael Trujillo Pino
José Miguel Santana Núñez

En Las Palmas de Gran Canaria, a 11 de junio de 2021

Agradecimientos

Agradecer a los tutores D. José Miguel Santana Núñez y D. Agustín Rafael Trujillo Pino por la realización de proyecto con ellos, usar programas que había pedido como el motor Unity para poder aprender y seguir adelante con más proyectos, y por la guía y orientación durante todo el proyecto a realizar.

También dar agradecimientos a mi familia y amigos que me han estado apoyando y dando ánimos durante el progreso del proyecto para poder seguir adelante y desarrollarlo con éxito.

Índice de contenido

Índice de tablas

Índice de ilustraciones

Capítulo 1 Introducción

SARS-CoV-2, también conocido como COVID-19 o Coronavirus, es un virus que afecta al mundo entero en aspectos económicos y sociales a corto y largo plazo. El origen de esta pandemia comenzó en diciembre de 2019 desde Wuhan, China, donde había muy pocos casos, pero no fue hasta febrero de 2020 cuando empezó a extender el virus en varios países, llegando a haber millones de contagios al día. Hoy en día ya hay vacunas que combaten a este virus, ya sea Pfizer, Astrazeneca o Janssen, y estas dosis se están aplicando a las personas de un cierto rango de edad, y cada vez va a otros grupos de edad, solventando esta situación mundial.

El virus se ha convertido en un tema de conversación diario, se puede escuchar su palabra en cualquier lugar, y es tendencia en todas las noticias y redes sociales. Siendo este el primer evento que también se ha expandido a través de estas a escala global. Plataformas como Facebook, Twitter o YouTube se han convertido en portavoces de diversa información sobre el coronavirus.

Durante los meses de confinamiento, Internet y las redes sociales nos ayudaron a mantenernos en contacto con amigos y familiares, pero además nos brindan una fuente de información más directa sobre la pandemia actual. Uno de los efectos que también ha tenido el COVID-19 en Internet ha sido el aumento de usuarios. Al no poder salir de casa por la pandemia, la gente usaba Internet durante todo este tiempo ya sea para usar Netflix para ver películas, realizar retransmisiones o ver estas de otras personas con la plataforma Twitch, expresar nuestros sentimientos por las redes sociales como Twitter, Facebook, Instagram, ver y realizar vídeos en YouTube y un largo etcétera. Todas estas actividades tenían como causa la saturación de tráfico de red por la cantidad de usuarios que estaban usando Internet a la vez.

"DE-CIX, principal operador mundial de puntos de intercambio de Internet con sede en Frankfurt, Alemania, ha comunicado por primera vez el volumen total de datos que han pasado por sus puntos de intercambio de Internet en un año. Así, un total de 32 exabytes de tráfico de datos circularon a través de los nodos de intercambio de Internet de DE-CIX en 2020, lo que supone aproximadamente la misma cantidad de datos que genera una videollamada que dura ocho millones de años." (Data Center Dynamics, 2021)

Por otro lado, la gente para no perder su actividad laboral por estar encerrada en casa, trabajan telemáticamente siguiendo con sus actividades, aumentando el uso de las plataformas de llamada como Skype, Microsoft Teams, Webex o Zoom y realizando reuniones para reanudar su trabajo.

"Otras cifras relevantes registradas durante este tiempo fueron el tráfico relativo a determinadas aplicaciones online. Por ejemplo, el tráfico relativo a videoconferencias -como Skype, Teams y Webex- comparado con las semanas anteriores a la crisis del COVID-19 se duplicó, el tráfico de servicios de gaming online aumentó un 30% y las plataformas de vídeo en streaming doblaron su número de usuarios y su tráfico."
(ticpymes, 2021)

Uno de los países más importantes y donde ha tenido más impacto ha sido Estados Unidos, afirmando tener un gran número de víctimas y provocando el cierre de importantes sectores económicos. El impacto de esta pandemia fue severo. Teniendo el

récord de contagiados de COVID-19, se preveía una gran caída económica en el país, dependiendo de qué medidas se podrían tomar para poder reactivar sus actividades y volver a recuperarse del gran golpe.

Como objetivo de trabajo, se recogen los datos disponibles de Estados Unidos, en el que muestren tanto el número de casos como el número de muertes por COVID-19, y estos datos se verán reflejados con su respectiva fecha en el programa que se vaya a crear para el usuario final. Con el uso de un mapa interactivo dentro del entorno, se puede interactuar en cada región individualmente, mostrando su información relevante, siendo que sea un filtro más fácil para la demostración de dichas zonas y el usuario quiera ir a sitios concretos para su consulta de datos.

Capítulo 2 Estado del arte

Un concepto importante para el entendimiento del trabajo es la *visualización de datos.*
"La visualización de datos es la presentación de datos en formato ilustrado o gráfico. Permite a los tomadores de decisiones ver la analítica presentada de forma visual, de modo que puedan captar conceptos difíciles o identificar nuevos patrones. Con la visualización interactiva, usted puede llevar el concepto un paso adelante utilizando tecnología para profundizar en diagramas y gráficas para observar mayor detalle, cambiando de forma interactiva qué datos ve y cómo se procesan.". (SAS, s.f.)

Las visualizaciones de datos comunes son:

- Gráficos
- Tablas
- Mapas
- Infografías
- Dashboards

Páginas web que usan este tipo de herramientas son, por ejemplo:

- (Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU), 2020) – Tiene una página web donde muestra el mapa mundial con los casos y muertes por COVID-19, añadiendo también la dosis de vacunas administradas en todo el mundo. La representación gráfica es con círculos rojos en las zonas que, según su relación de casos y muertes, es más grande o pequeña. también tiene una serie de gráficas que muestra cómo son los casos, muertes y vacunas diarias.
- *Our World In Data* (Max Roser, 2020) - Esta página web contiene una recolección de datos del COVID-19 diarios de diferentes regiones del mundo, mostrados en forma de gráfica de línea.
- *Google* (Google, 2020) - Ofrece un mapa interactivo de dos dimensiones, con un estilo similar a "Covid19 Dashboard by CSSE" en la demostración de datos, pero la diferencia que hay es que en esta página solo muestra el número de contagios y vacunados, también su forma de desplazarse es más rápida que la otra página, su tiempo de carga es bastante menor y muestra los datos de una forma más estructurada.

También alternativas a la muestra de datos son los Sistemas de Información Geográfica (SIG), donde involucran varios componentes que permiten la organización, almacenamiento, manipulación, análisis y modelado de grandes cantidades de datos del mundo real.
Ejemplos que usan SIG son:

- ArcGIS
- Cesium
- Google Maps

Para nuestro caso, la alternativa que utilizamos fue el motor de Unity, porque nos da más flexibilidad en su uso, pudiendo añadir objetos, materiales, perspectivas de cámara con más facilidad y manejo de recursos, efectos, imágenes y colores., todo en un entorno tridimensional, con la finalidad de dar una comodidad visual al usuario. Además, Unity es eficaz con técnicas que el motor ofrece como *Entity Component System* (ECS), reduciendo el tiempo de CPU y mostrar la interacción de los mapas con más fluidez, en comparación con otros mapas interactivos, también nos permiten añadir efectos de videojuegos y asimismo se puede usar en cualquier plataforma.

Capítulo 3 Motivación del trabajo

La decisión que se ha tomado para este trabajo ha sido con el profesor D. José Miguel Santana Núñez y D. Agustín Trujillo Pino, donde se ha debatido que proyecto de Trabajo Fin de Grado se va a realizar, ya que uno de los lenguajes que yo personalmente quería utilizar era *C#*, utilizando el motor de Unity para la aplicación. El rango de ideas de proyecto a hacer era amplia, desde hacer un juego, hasta aplicar algoritmos para una demostración gráfica.

Finalmente se decidió realizar un proyecto que todo el mundo podría utilizar, usando las herramientas que tenía de preferencia, y con la ayuda del profesor, experto en trabajos de mapas y visualización de datos, tener una orientación para lograr este trabajo. Realizar un mapa de datos de *Estados Unidos*, debido a que es la primera potencia mundial y tiene mayor fuente de información de datos sobre el COVID-19. Los casos de España no están reflejados en todos los sitios, comparado al de Estados Unidos, que tiene tanto de cada estado, como de cada población.

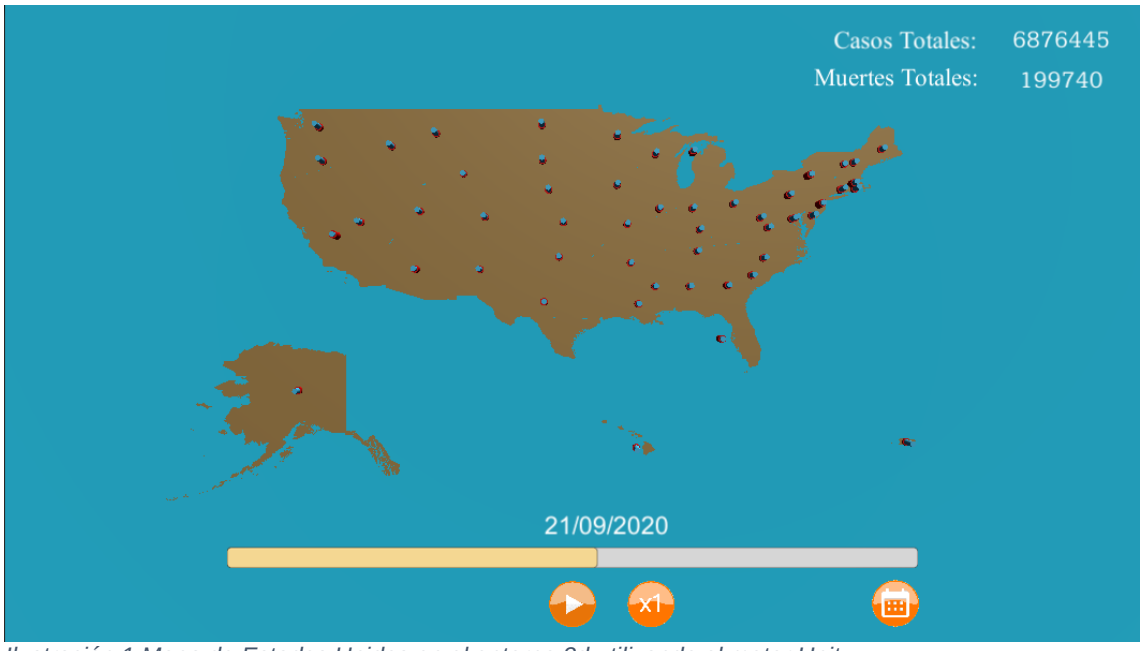


Ilustración 1 Mapa de Estados Unidos en el entorno 3d utilizando el motor Unity

Capítulo 4 Requisitos y herramientas

4.1 Requisitos Hardware

El desarrollo al completo de la herramienta se ha realizado en el dispositivo con las siguientes características:

Tabla 1 Dispositivo donde se desarrolla la herramienta

Modelo	Acer Aspire A515-51G
Sistema operativo	Windows 10 Sistema operativo de 64 bits, procesador x64
Procesador	Intel® Core™ i5-8250U CPU @ 1.6GHz – 1.8 GHz
Memoria	8 GB (7.88 GB utilizable)
Gráficos	NVIDIA GeForce MX130

4.2 Herramientas Utilizadas

4.2.1 Motor Gráfico

Ilustración 2 Logo de Unity

Unity será nuestro motor por utilizar para nuestro proyecto de fin de grado, usando de motor gráfico *OpenGL*, una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Gracias a Unity, se podrá visualizar tanto el mapa de Estados Unidos como sus simbologías y datos necesarios para la realización de este proyecto. La versión que se utilizará de Unity es *2020.2.2*.

4.2.2 Lenguaje de Programación

Ilustración 3 Logo C#

Unity acepta diversos lenguajes de programación, ya sea *UnityScript*, *Boo* o *Monodevelop*, pero para este caso usaremos *C#*, un lenguaje de programación multiparadigma desarrollado y estandarizado por Microsoft en el año 2000, sucesor de C++. Es uno de los lenguajes de programación diseñados para infraestructura de lenguaje común.
Usando C# se podrá implementar la creación de datos y su representación desde aquí para el motor gráfico Unity.

4.2.3 Editor de código

Ilustración 4 Logo Visual Studio Code

Para poder utilizar el lenguaje de programación *C#,* este ha de tener un editor de código para poder escribir en este. Está *Sublime Text, Notepad++, Visual Studio, CODA, ATOM*, etcétera. Pero para este caso se usa *Visual Studio Code*, ya que da muchísimos "plugin" para ayudar al usuario a la facilitación de código como el autocompletado o las sugerencias.

4.2.4 Terminal de descarga de archivos

Ilustración 5 logo cURL

Si queremos descargar los archivos de *GitHub* para nuestro almacenamiento local, *Curl* es nuestra respuesta y amigo. Es un intérprete de comandos orientado a la transferencia de archivos. Con ello, se puede realizar una petición a la página con el archivo que buscamos y nos devolverá este mismo. Usando C#, podemos escribir código para ejecutarlo sin que haya intervención humana.

Capítulo 5 Entity Component System (ECS)

Un patrón de diseño a utilizar en este proyecto es *Entity Component System (ECS)*. Es un paquete dentro de la plataforma Unity con un paradigma diferente de escribir código, donde la estructura del sistema se divide en tres partes:

- *Entidad (Entity)* – Simbologías que tendrán su componente y su sistema para el correcto funcionamiento.
- *Componente (Component)* – Una vez que tenemos la entidad creada, esta tendrá ciertos atributos asignados, ya sea velocidad, tamaño, color, etcétera. Y se almacenarán en la entidad.
- *Sistema (System)* – Se aplicará toda lógica detrás ya una vez se tiene la entidad con sus componentes asignados, como, por ejemplo, incrementar la velocidad, reducir el tamaño, etcétera.

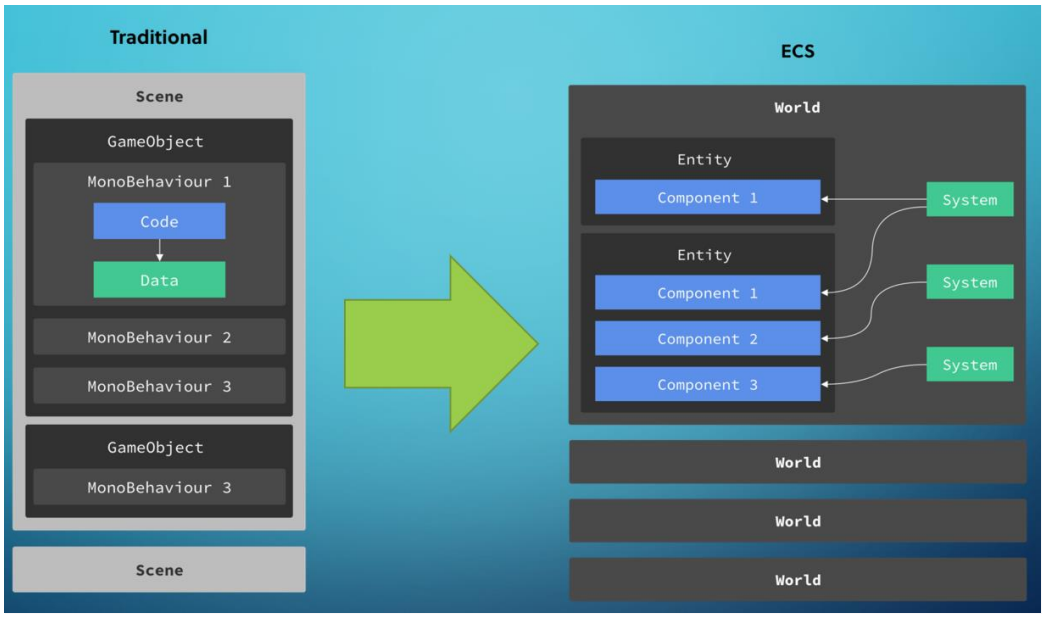


Ilustración 6 Estructura de un programa creando un objecto en Unity de manera general, a la izquierda, en comparación a la creación de un objeto en Unity, utilizando ECS, a la derecha. Fuente: https://www.improbable.io/blog/unity-ecs-1

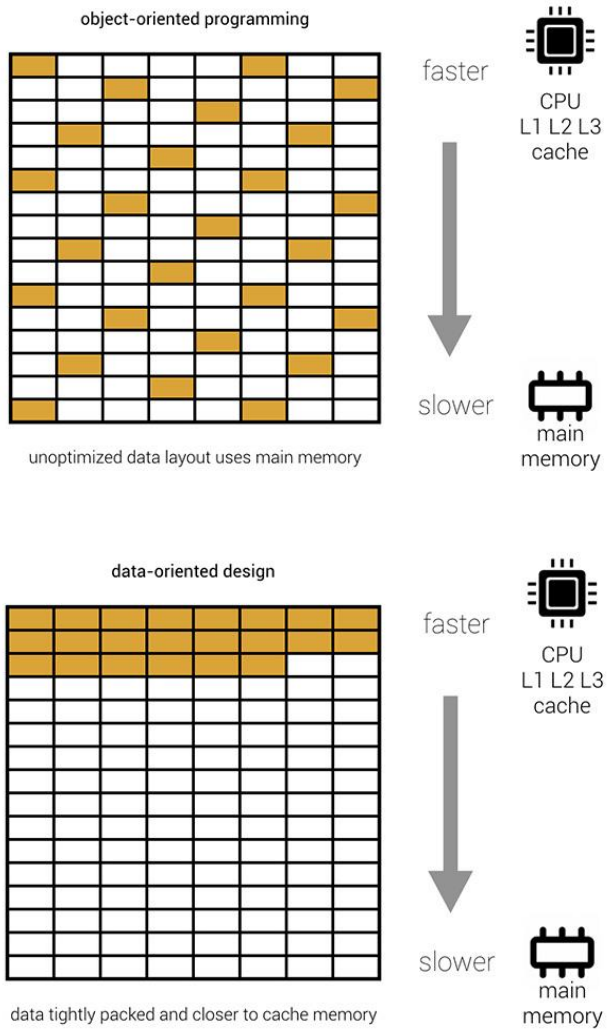


Ilustración 7 Comparación de estructura de datos, sin aplicar ECS y aplicándolo. Fuente: https://www.raywenderlich.com/7630142-entity-component-system-for-unity-getting-started

¿Qué se consigue con esto? Reducir el consumo de CPU, realizar mayor número de acciones y mayor eficacia a la hora de trabajar con múltiples objetos. Gracias a la aplicación de esta técnica, se puede crear una grandísima cantidad de objetos con el menor procesamiento, por ejemplo, generar 10.000 soldados a la vez sin que la CPU haga que ralentice el sistema.

ECS también tiene potencial a la hora de usar operaciones matemáticas, ya que todos los atributos que se reciben son números, en el cuál, se están usando en todo momento para las modificaciones, y con la ayuda de ECS, el tiempo de respuesta será menor.

En la siguiente imagen se muestran las estadísticas de CPU después de generar 10000 objetos y se realiza una comparación de tiempo, uno de ellos *sin aplicar ECS* obtiene un tiempo de *49.6 milisegundos (20.1 Fotogramas Por Segundo)* (foto de la izquierda) y por otro lado, *aplicando ECS*, obtiene *5.7 milisegundos (174.1 Fotogramas Por Segundo)* (foto de la derecha)

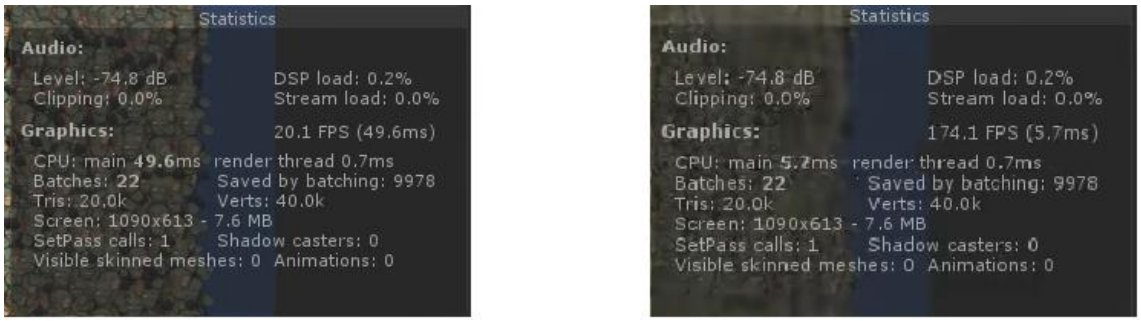


Ilustración 8 Comparación de tiempo de CPU generando 10000 objetos

4.3.1 Ventajas de ECS:

- Está orientado a datos: los datos tienden a almacenarse linealmente, que es la mejor manera para que el sistema acceda a los datos. En una implementación de ECS decente, los datos se almacenan y procesan secuencialmente, casi sin interrumpir ningún sistema dado que procesa un conjunto determinado de componentes.

- Es muy independiente: separa los datos del comportamiento del sistema para una mayor efectividad.

- Es muy amigable con el procesamiento en paralelo y los multihilos: debido a la forma en que están estructuradas las cosas, muchas entidades y componentes pueden evitar conflictos y pueden procesarse en paralelo con otros sistemas.

4.3.2 Desventajas de ECS:

- Aunque es fácil depurar el proceso del sistema, también es más difícil depurar los cambios de un solo componente, y es aún más difícil no ver lo que le sucede a la entidad para todos los componentes de toda la entidad. No estoy seguro de si Unity ha introducido nuevas funciones de depuración para esto.

- Si planea utilizar ECS en su equipo, puede resultar difícil proponer nuevos paradigmas a los desarrolladores que no están familiarizados con ECS. El tiempo de incorporación puede ser más largo y caro.

- Todavía no está lista para la producción, a medida que evoluciona.

Sabiendo lo bueno y lo malo que tiene ECS, procedemos a comenzar el desarrollo del TFG teniendo ya una base puesta a trabajar.

Capítulo 6 Obtención e integración de datos

6.1 Obtención de datos

La primera tarea para realizar es encontrar un *dataset* donde estén disponibles todos los datos de COVID-19 en Estados Unidos en un archivo con la extensión *Comma Separate Values (.CSV)*. Tras probar en varias páginas tales como (Google, s.f.) o (Kaggle, s.f.), se ha encontrado un archivo con los datos de COVID-19 tanto en los estados de Estados Unidos como sus condados en él (New York Times, 2020). Finalmente se decide obtener el *dataset* de todos los estados para tener más información general.

6.2 Integración de datos en Unity

Una vez obtenido los datos desde GitHub, se importan estos por código y se muestra por Unity. Abriendo el archivo que hemos descargado, generamos una estructura donde dividimos los campos que se encuentran disponibles en él con los valores separados por comas, los valores son:

- *Fecha* – Día, mes y año en el que se obtuvieron los casos y muertes en ese estado, será de tipo *DateTime.*
- *Nombre del estado* – Variable de tipo *String*
- *Código del estado* – Este valor no varía para su estado, variable de tipo *String*
- *Número de casos* – Variable de tipo *unsigned int* (uint)
- *Número de muertes* – Variable de tipo *unsigned int* (uint)

Estad dos últimas variables son de tipo uint debido a que, en estos tipos de datos, nunca habrá casos o muertes nuevos con números negativos. Podrá haber un descenso del número, pero el mínimo de es 0 (ninguno).

Por último, almacenamos estas variables en vectores de datos donde se encontrarán sus valores asignados.

```
public struct CovidDataList
{
    public DateTime date;
    public string state;
    public string fips;
    public uint cases;
    public uint deaths;

    public CovidDataList(DateTime date, string state, string fips, uint cases, uint deaths)
    {
        this.date = date;       //Fecha
        this.state = state;     //Estado
        this.fips = fips;       //Código del estado
        this.cases = cases;     //Casos
        this.deaths = deaths;   //Muertes
    }
}
```

```
var reader = new StreamReader(File.OpenRead(Application.dataPath+"/us-states.csv"));
string skip = reader.ReadLine();  //Salta primera linea

while (!reader.EndOfStream)
{
    var line = reader.ReadLine();
    var values = line.Split(',');

    list.Add(new CovidDataList(DateTime.Parse(values[0], new CultureInfo("en-US")), values[1], values[2],
    uint.Parse(values[3]), uint.Parse(values[4])));



}
```

Ilustración 9 Estructuración de datos del archivo .CSV.

Fecha	Estado	Casos	Muertes
12/01/2020	Washington	1	0
13/01/2020	New York	1	0
14/01/2020	Maine	0	0
15/01/2020	Texas	0	0
16/01/2020	California	0	0
17/01/2020	Alaska	0	0
18/01/2020	Florida	0	0
19/01/2020	Arizona	1	0

Ilustración 10 Almacenamiento de los datos obtenidos del CSV en vectores de datos y tabla de datos.

6.3 Obtener los datos de límites fronzerizos de los estados

Al obtener los datos separados en vectores diferentes, se crea el mapa de Estados Unidos cogiendo como base una imagen de este país, y construiremos sus estados con sus dichas colisiones para poder interactuar desde Unity.

Ilustración 11 Mapa de Estados Unidos cogido como base para la implementación de colisiones. Fuente: https://www.mapasparacolorear.com/estados-unidos/mapa-estados-unidos-estados-con-nombres.png

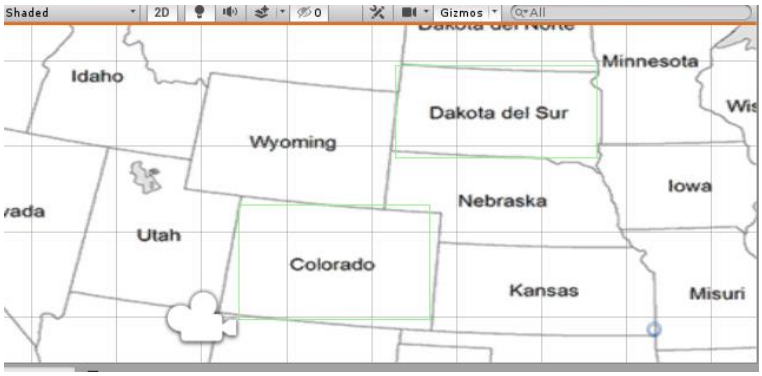


Ilustración 12 Colisiones de los estados para su futura interacción

Claramente esta forma de realizarlo es subóptima, pues el objetivo de un programador es automatizar todo y que lo haga la máquina. Como dijo mi tutor cuando le enseñé esta parte:

"Si tú, como programador estas realizando tareas manuales, tu desempeño como programador no lo estás haciendo de forma correcta, pues el objetivo es automatizar las tareas y que se encargue la máquina de hacer todas estas arduas tareas por ti." (Santana, 2021).

Tras la charla realizada con el profesor, me recomendó usar los datos GeoJSON, en el que tiene ficheros con las coordenadas para la creación de estados. Una página para obtener las coordenadas de Estados Unidos, que se utilizará para Unity, es *eric.clst.org*

(Celeste, 2020) donde se consigue los datos de todos los estados con mayor facilidad, y con diferentes precisiones de coordenadas. Gracias a esta obtención de archivo, se puede crear las colisiones y las figuras deseadas de los estados en el motor Unity.

6.4 Creación de las figuras de los estados

De todos los archivos disponibles de coordenadas que la pagina mencionada anteriormente ofrece, se procede a descargar el archivo "*US States GeoJSON 20m"*, pues este archivo contiene bastantes datos precisos de coordenadas para la generación de los estados.

Desde el editor de código, accederemos al archivo que hemos descargado recientemente, pero, como es un archivo JSON, hay que desestructurarlo y obtener los datos que deseamos. Gracias al plugin SimpleJSON (Unify Community Wiki, 2017) la forma en la que desestructurará el archivo GeoJSON será bastante más agradable para el programador.

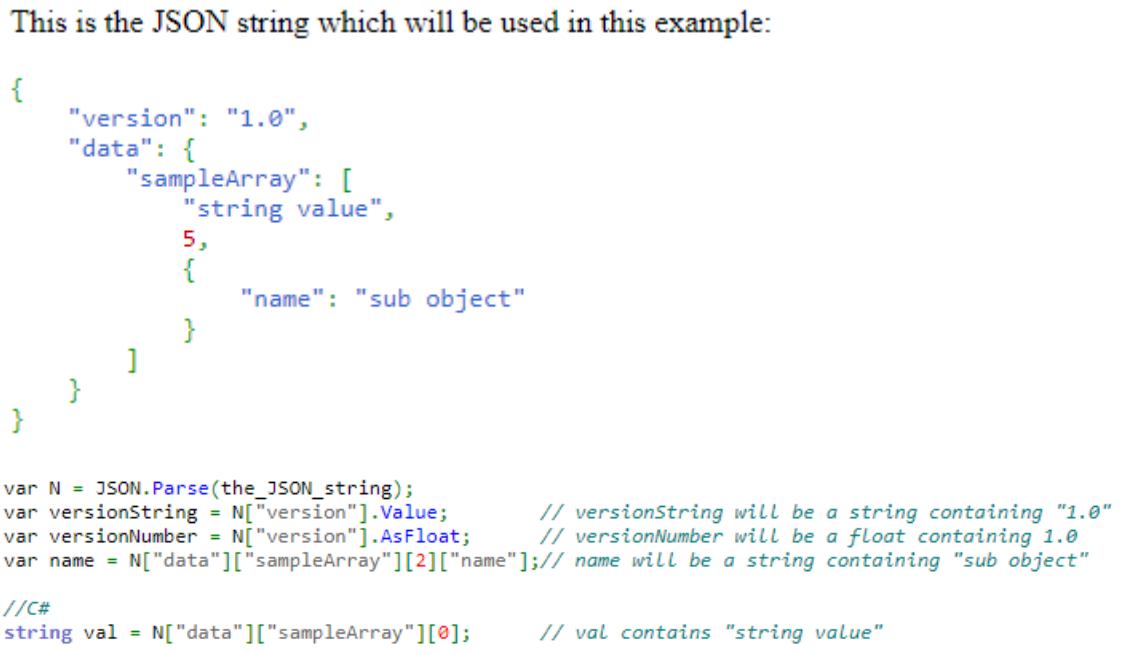


Ilustración 13 Ejemplo de código usando SimpleJSON. Fuente: (Unify Community Wiki, 2017)

Los datos más importantes para obtener del archivo son:

- *Nombre del estado*
- *Coordenadas del estado*
- *Tipo de polígono:* En el archivo JSON hay dos tipos de polígonos diferentes:
    - *Polígono* – Es una figura simple que ha sido generada
    - *Multipoligono* – En este tipo se encuentran múltiples figuras generadas, en el cual pertenece a un mismo estado. Por ejemplo, *Hawái* está compuesto por varias islas pequeñas, en resumen, figuras múltiples a generar. Estos tipos son los que se consideran multipolígonos.

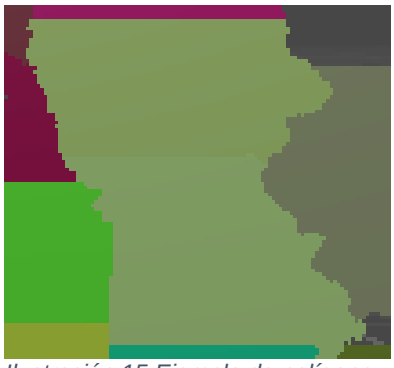


Ilustración 15 Ejemplo de polígono.

Ilustración 14 Ejemplo de multipolígono, hay pequeñas islas alrededor de Alaska.

Conociendo los parámetros deseados, escribimos en código la obtención de estos datos de implementación y generamos las figuras deseadas para la visualización de estas en el motor de Unity.

```
jsonString = File.ReadAllText(Application.dataPath + "/gz_2010_us_040_00_20m.json");
N = JSON.Parse(jsonString);
var features = N["features"];

statesGO = new GameObject[features.Count];
statesMesh = new Mesh[features.Count];
```

Ilustración 16 Abrir fichero JSON y desestructuración de este.

La primera imagen es la transformación del archivo JSON a la cadena String, y, poco a poco, se va accediendo a las secciones de este para poder recoger los datos. También se usa un array de objetos almacenando los estados y su creación, y por último se crea otro array guardando las textura y colisiones de los objetos.

```
for (int i = 0; i < features.Count; i++)
{
    string stateName = features[i]["properties"]["NAME"];
    var coordinates = features[i]["geometry"]["coordinates"];
    statesGO[i] = new GameObject(stateName, typeof(MeshFilter), typeof(MeshRenderer), typeof(MeshCollider),
                                 typeof(StateBehaviour));
    statesMesh[i] = new Mesh();
    if (features[i]["geometry"]["type"] == "Polygon")
    {

        addPolygon(coordinates, i);

        setupGameObject(i);

        statesGO[i].transform.localScale = new Vector3(statesGO[i].transform.localScale.x, statesGO[i].transform.localScale.y, -1);
    }
```

Ilustración 17 Código de generación de polígono

Este código mostrado encima se resume en:

Tabla 2 Variables para la creación de objetos

stateName	Nombre del estado en el objeto
coordinates	Coordenadas del estado en el objeto
statesGO[i]	Objeto del estado con sus tipos
statesMesh[i]	Texturas y colisiones del objeto

```
void addPolygon(JSONNode coordinates, int i)
{
    Vector2[] vertices2D = new Vector2[coordinates[0].Count];
    for (int j = 0; j < coordinates[0].Count; j++)
    {
        vertices2D[j] = new Vector2(coordinates[0][j][0], coordinates[0][j][1]);
    }
    Triangulator tr = new Triangulator(vertices2D);
    int[] indices = tr.Triangulate();
    Vector3[] vertices = new Vector3[vertices2D.Length];
    for (int j = 0; j < vertices.Length; j++)
    {
        vertices[j] = new Vector3(vertices2D[j].x, vertices2D[j].y, 0);
    }

    statesMesh[i].vertices = vertices;
    statesMesh[i].triangles = indices;
    statesMesh[i].RecalculateNormals();
    statesMesh[i].RecalculateBounds();

}
```

Ilustración 18 Transformación de coordenadas a triángulos

Si el tipo de figura que estamos viendo es un polígono, entonces, se creará la figura directamente, gracias a la ayuda de *Triangulator* (Unify Community Wiki, 2018) los triángulos que se van creando, se van uniendo entre ellos, ahorrando el arduo tiempo y código que supone crear los triángulos sin que se generen errores. Después, se ajustan los vértices y los triángulos con su respectivo objeto creado.

Realizamos la primera prueba de generación de polígonos y se ha obtenido el siguiente resultado:

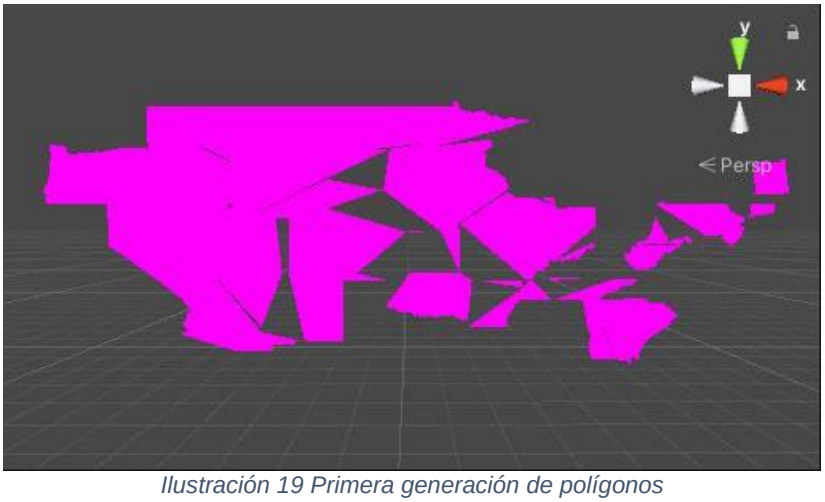


Ilustración 19 Primera generación de polígonos

Como se observa, se han generado los polígonos, pero no de forma correcta, hay escasez de triángulos que se han generado y la figura del estado se encuentra en estado incompleto. Este problema supuso un gran reto debido a que se estaba buscando una manera óptima para generar los triángulos correctamente.

Tras dos semanas buscando el método de generar triángulos sin que cause fallos. Una forma que se ha probado y ha funcionado fue cambiando el archivo JSON por otro, de:

```
gz_2010_us_040_00_20m.json
```

A:

```
gz_2010_us_040_00_500k.json
```

Al parecer, reduciendo el número de coordenadas que tiene el fichero, ayuda a la clase *Triangulator* a generar los triángulos con menos errores, ya que contiene menos información de coordenadas y mejora la creación de estos.

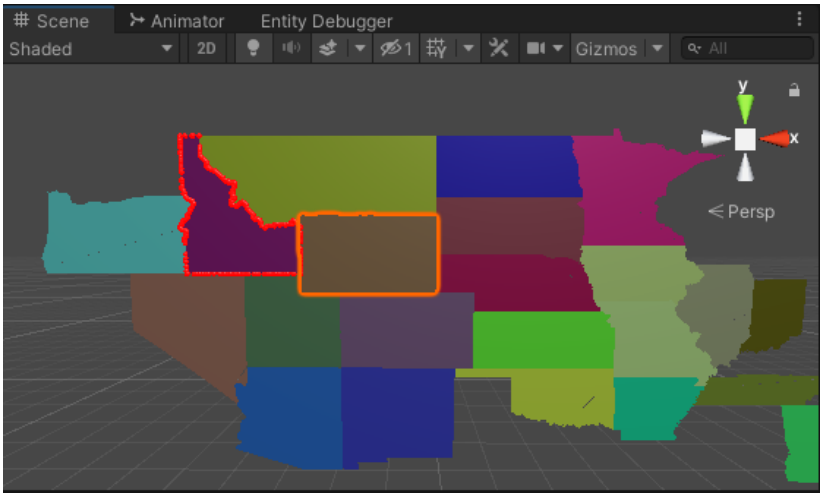


Ilustración 20 Segunda generación de polígonos

Los polígonos creados anteriormente se generan con éxito, ahora falta crear un método para los casos de los multipolígonos.

```
else if (features[i]["geometry"]["type"] == "MultiPolygon")
{
    addMultiPolygon(coordinates, i, stateName);

    switch (stateName)
    {
        case "Alaska":
            statesGO[i].transform.Translate(new Vector3(-55f, -26f, -0f));
            statesGO[i].transform.localScale = new Vector3(0.45f, 0.75f, -1);
            break;
        case "Hawaii":
            statesGO[i].transform.Translate(new Vector3(65f, -2f, -0f));
            statesGO[i].transform.localScale = new Vector3(statesGO[i].transform.localScale.x, statesGO[i].transform.localScale.y, -1);
            break;
        default:
            statesGO[i].transform.localScale = new Vector3(statesGO[i].transform.localScale.x, statesGO[i].transform.localScale.y, -1);
            break;
    }
    setupGameObject(i);
}
```

Ilustración 21 Código de generación de multipolígono

```
void addMultiPolygon(JSONNode coordinates, int i, string stateName)
{
    List<Vector2> vertices2D = new List<Vector2>();
    for (int j = 0; j < coordinates.Count; j++)
    {
        for (int k = 0; k < coordinates[j][0].Count; k++)
        {
            vertices2D.Add(new Vector2(coordinates[j][0][k][0], coordinates[j][0][k][1]));
        }
    }

    Triangulator tr = new Triangulator(vertices2D.ToArray());
    int[] indices = tr.Triangulate();
    Vector3[] vertices = new Vector3[vertices2D.Count];

    for (int j = 0; j < vertices.Length; j++)
    {
        vertices[j] = new Vector3(vertices2D[j].x, vertices2D[j].y, 0);
    }

    statesMesh[i].vertices = vertices;
    statesMesh[i].triangles = indices;
    statesMesh[i].RecalculateNormals();
    statesMesh[i].RecalculateBounds();
    statesMesh[i].Optimize();
}
```

Ilustración 22 Creación de triangulos para multipolígonos

Se realiza la comprobación de que, si es multipoligono la figura que está recibiendo por parámetro, entonces se creará un bucle para crear cada polígono individualmente y luego unir todos estos en las mismas colisiones. Luego hay excepciones como Alaska y Hawái, donde sus figuras creadas están bastantes alejadas de la cámara en Unity, entonces no se aprecia con claridad, con lo cual, hay que trasladarlos a una zona donde se vea estas figuras junto al resto, para ello se usa el método transform.translate(). Reajustamos la escala de los estados y se guarda.

```
void setupGameObject(int i)
{

    statesGO[i].GetComponent<MeshFilter>().mesh = statesMesh[i];
    Color32 newColor = new Color32(0xbc, 0x8e, 0x47, 255);
    statesGO[i].GetComponent<MeshRenderer>().material.color = newColor;
    statesGO[i].GetComponent<MeshCollider>().sharedMesh = statesMesh[i];
    statesGO[i].transform.parent = statesList.transform;
}
```

Ilustración 23 Asignación de materiales para el objeto

Por último, este método coloca los valores de sus respectivas figuras, se ajusta el color y se guarda el objeto en la jerarquía de elementos de Unity, clasificándolo en la lista de "estados" y también ayuda en la visión del desarrollador, ya que no ve todos los objetos desplegados a la vez.

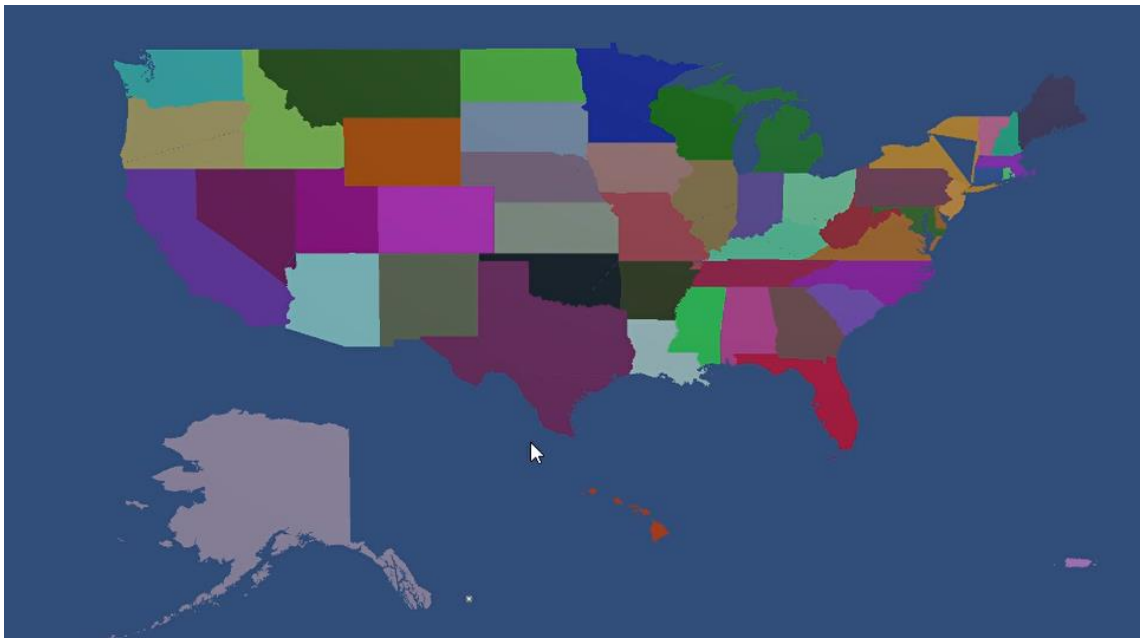


Ilustración 24 Mapa de Estados Unidos en Unity

Este es el resultado de todo el código aplicado anteriormente, como se puede apreciar, Hawái y Alaska fueron trasladados para mejor visión del usuario, otras zonas como *Nueva York* tiene falta de triángulos debido a que *Triangulator* no ha generado correctamente todos los puntos del estado.

6.5 Implementación de datos en el mapa

Las mallas que representan los estados están creados, teniendo sus propias colisiones para su futura interacción en la muestra de información, también los datos recogidos de CSV están almacenados en una lista, ahora toca aunar ambas cosas, donde si hacemos clic, se muestra la información del estado.

Creamos una clase nueva llamada *MapController*, todos los datos a representar en la interfaz estarán involucrados en esta clase.

Primero, inicializamos variables en el método *Awake()* (método que se llama cuando la instancia del "script" está siendo cargado).

```
void Awake()
{
    mapController = this;
    data = new CSVData();
    lastDate = data.GetList()[data.GetList().Count - 1].date;
    firstDate = data.GetList()[0].date;
    currentDate = firstDate;

    Font arial;
    arial = (Font)Resources.GetBuiltinResource(typeof(Font), "Arial.ttf");

    // Set Text component properties.
    dateText = textUI.GetComponent<Text>();
    dateText.font = arial;
    dateText.text = "" + currentDate.ToShortDateString();
    dateText.fontSize = 32;
    dateText.alignment = TextAnchor.MiddleCenter;
}
```

Ilustración 25 Declaración de variables en la clase mapController

Tabla 3 Variables de la clase mapController

Variable	Función
mapController	Es una referencia a la clase en la que estamos trabajando, declarándolo como "this" estamos diciendo que está clase es global, donde el resto de las clases tienen acceso.
Data	Lista de datos del archivo .CSV declarado anteriormente.
lastDate	Última fecha del archivo descargado (que es el día actual).
firstDate	Primera fecha del archivo descargado.
currentDate	Fecha que se está analizando en ese momento.
dateText	Variable para mostrar la fecha *currentDate* en la pantalla.

Segundo, implementamos la consulta de los datos a buscar de la lista *data*.

```
public string getFips(string state)
{
    foreach (CovidList ds in data.GetList())
    {
        if (state == ds.state)
        {
            return ds.fips;
        }
    }
    return "None";
}
```

Ilustración 26 Obtención de código del estado

Para la obtención del código postal de ese estado, pasamos por parámetro la zona a buscar en la lista, una vez encontrado, retorna su código, si no lo encuentra en el archivo CSV, devuelve "None".

```
public uint getCases(string state, DateTime dt)
{

    foreach (CovidList ds in data.GetList())
    {
        if (state == ds.state && ds.date = dt)
        {
            return ds.cases;
        }
    }
    return 0;
}
```

Ilustración 27 Obtención de casos

```
public uint getDeaths(string state, DateTime dt)
{

    foreach (CovidList ds in data.GetList())
    {
        if (state == ds.state && ds.date = dt)
        {
            return ds.deaths;
        }
    }
    return 0;
}
```

Ilustración 28 Obtención de muertes

Tanto el número de casos como el de muertes, pasamos por parámetro la fecha que queremos y el estado a buscar y vamos recorriendo en el bucle *foreach* hasta que encuentre con el dato deseado, entonces retorna su número, en el caso que no encontremos información, devuelve 0.

```
0 references
public DateTime getDate()
{
    return currentDate;
}
```

Ilustración 29 Obtención de fecha reciente

El método *getDate()* simplemente devuelve la fecha en la que se encuentra en ese momento.

```
0 references
public void setDate(DateTime dt)
{
    currentDate = dt;
    dateText.text = "" + currentDate.ToShortDateString();
}
```

Ilustración 30 Método para cambiar la fecha

setDate() cambia la fecha a una deseada por el usuario y cambia el texto de la pantalla por la fecha asignada.

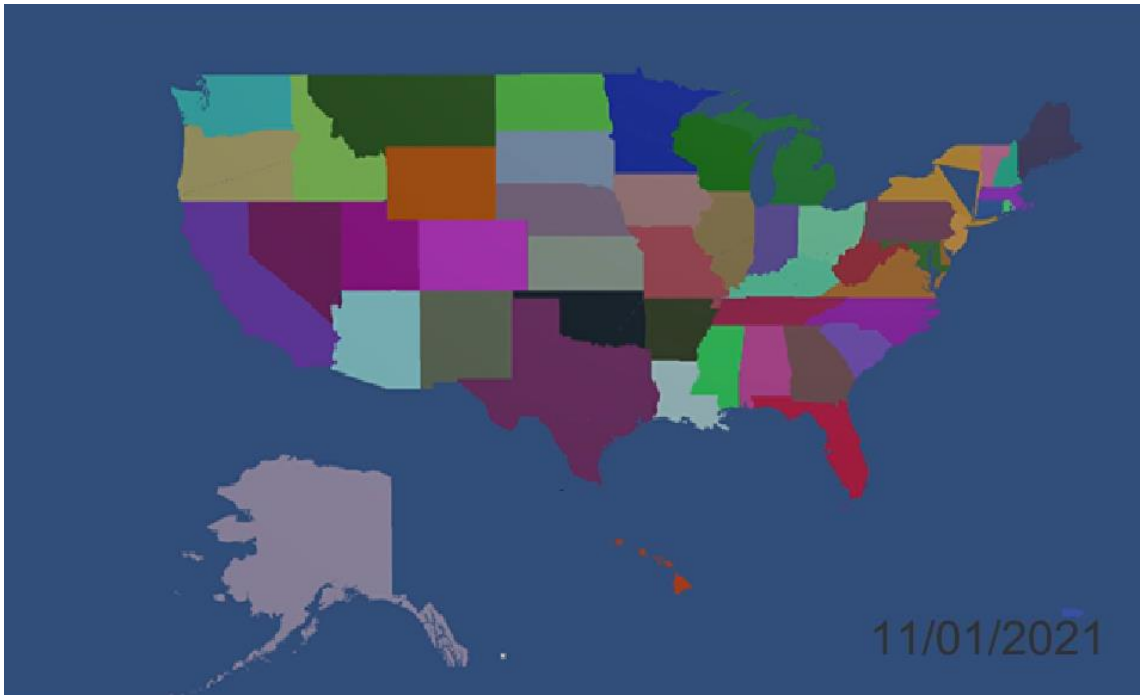


Ilustración 31 Mapa de Estados Unidos con una fecha asignada en pantalla

El mapa ahora tiene la fecha implementada, se sabe que día se encuentra. El siguiente paso es mostrar la información de la región.

Creamos una clase llamada *StateInfo*, mostrará la información del estado en pantalla.

```
public class StateInfo : MonoBehaviour
{
    public static StateInfo stateInfo;
    public Text stateText;
    public Text fipsText;
    public Text xText;
    public Text yText;
    public Text casesText;
    public Text deathsText;
    public GameObject uiInfo;

    MapController mapController;
```

Ilustración 32 Declaración de variables en la clase StateInfo

```
void Start()
{
    stateInfo = this;
}

1 reference
public void showInfo(string state, Vector3 centerPoint)
{
    stateText.text = state;
    fipsText.text = mapController.getFips(state);
    xText.text = "" + centerPoint.x;
    yText.text = "" + centerPoint.y;
    casesText.text = ""+mapController.getCases(state,mapController.getDate());
    deathsText.text = ""+mapController.getDeaths(state,mapController.getDate());

    uiInfo.SetActive(true);
}
```

Ilustración 33 Asignación de variables y muestra de tabla de información en la clase stateInfo

```
public void closeInfo()
{
    uiInfo.SetActive(false);
}
```

Ilustración 34 Llamada al método para ocultar la tabla de información

Tabla 4 Variables en la clase stateInfo

Variables	Función
stateInfo	Esta clase, como *MapController*, serán clases de tipo global, donde se puede llamar desde cualquier script.
stateText	Nombre del estado.
fipsText	Código del estado.
xText	Coordenada X donde se encuentra el estado.

yText	Coordenada Y donde se encuentra el estado.
casesText	Número de casos en ese estado.
deathsText	Número de muertes en ese estado.
uiInfo	Muestra u oculta la información en pantalla.
mapController	Clase que contiene la información del mapa, como la fecha o el código.

Para poder mostrar toda esta información, hay que añadir un evento y que llame a este método para su ejecución en el programa. Creamos otra clase llamada *StateBehaviour*, definirá el comportamiento del mapa, hará que muestre o no la información de la región.

```
private void OnMouseDown()
{
    string state = gameObject.name;
    StateInfo.stateInfo.showInfo(state);

}
```

Ilustración 35 Evento para mostrar la tabla de información

Este método se llama cuando el ratón ha hecho clic sobre una zona que tenga una colisión, en el momento que detecte que hay una región así, se obtiene el nombre del objeto (que es el nombre del estado), y llamará a la clase *StateInfo* para habilitar la interfaz en la pantalla.

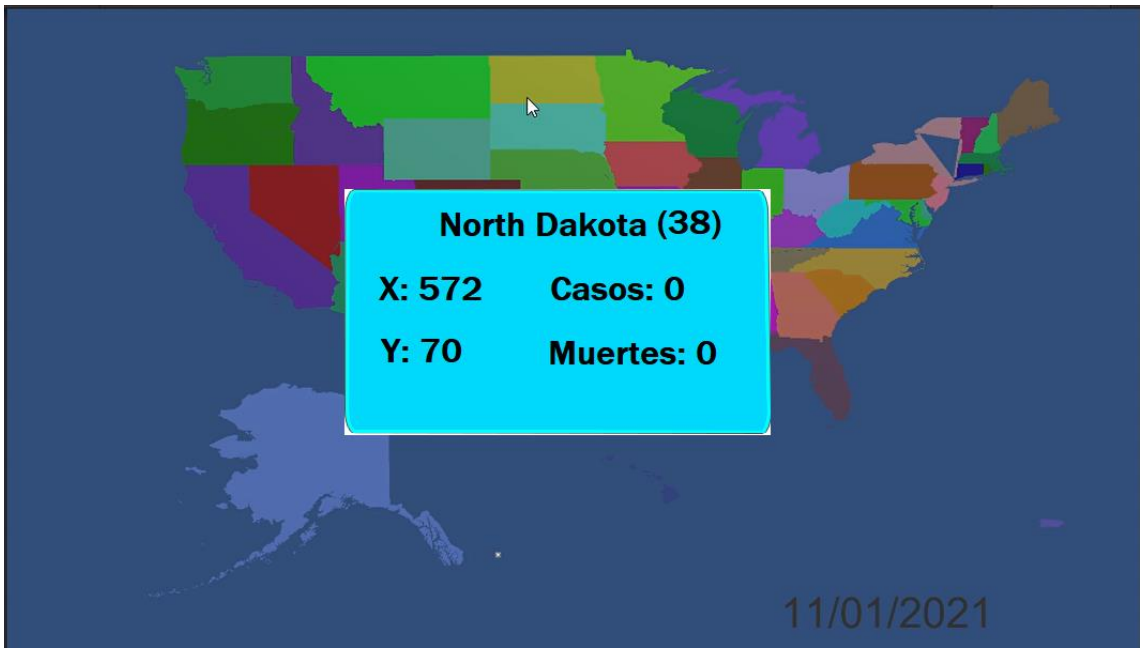


Ilustración 36 Muestra de datos en el mapa

Las coordenadas X e Y son las coordenadas en donde se encuentra el ratón haciendo clic en la pantalla, es un dato innecesario para mostrar al usuario final ya que no indica

nada real ni interesante. Más adelante modificaremos estos parámetros y se añadirán otros.

Capítulo 7 Implementación de ECS

7.1 Instalación ECS en Unity

Para poder usar este paradigma, primero hay que instalar unos paquetes en nuestra versión de Unity para poder utilizarlo.

La importación se descarga desde una página web debido a que Unity no lo tiene como público en su "package manager", pues todavía se encuentra en la versión Preview y aunque pronto será oficializado, la importación se realiza manualmente, colocando la *URL* del paquete en el instalador. Los paquetes de descarga encuentran aquí: (Unity, 2020).
Este proyecto es innovador porque utiliza tecnologías que todavía están en desarrollo y se puede sacar un mayor potencial de cara a su publicación oficial, consiguiendo ahorrar recursos y un menor tiempo en compilación.

Se instalará los siguientes paquetes:

Tabla 5 Paquetes de instalación para ECS

Paquete	Versión
com.unity.entities	0.17.0-preview.41
com.unity.mathematics	1.2.1
com.unity.collections	0.15.0-preview.21

Ya instalados los paquetes, se procede a usarlos en el código para su futura ejecución.

7.2 Aplicación de ECS en Unity

Sabiendo cómo funciona la estructura de ECS teóricamente, podemos construir el programa poco a poco.

Implementamos los paquetes recién descargados.

```
using Unity.Entities;
using Unity.Mathematics;
using Unity.Collections;
```

Ilustración 37 Importación de los paquetes de ECS

Representación general para cada estado.

Creamos el objeto de tipo *Esfera* y lo almacenamos como un *prefab*, o, mejor dicho, un objeto constante, en el cual, siempre tendrá los mismos datos por defecto.

En el script de control de objetos en ECS, creamos una variable para almacenar dicho prefab creado anteriormente.

```
[SerializeField] private GameObject spherePrefab;
```

Ilustración 38 declaración del objeto esfera por código

Creación de entidad

Creamos lo siguiente:

```
EntityManager entityManager;
```

Ilustración 39 Manejador de entidad

EntityManager es una clase para procesar entidades y sus datos.

```
entityManager = World.DefaultGameObjectInjectionWorld.EntityManager;
```

Ilustración 40 Declaración del manejador en el mundo de Unity

Todas las *entidades, componentes y sistemas* existen en un mundo. Cada mundo tiene un *EntityManager.*

```
var settings = GameObjectConversionSettings.FromWorld(World.DefaultGameObjectInjectionWorld, null);

sphereEntityPrefab = GameObjectConversionUtility.ConvertGameObjectHierarchy(spherePrefab, settings);
```

Ilustración 41 Configuración del objeto

Estas líneas solo establecen unas configuraciones por defecto de conversión y lo pasa a la jerarquía de objetos.

Configuración y asignación de los componentes de la entidad

```
private void SpawnSphere()
{
    //1
    NativeArray<Entity> sphereArray = new NativeArray<Entity>(statesGO.Length, Allocator.Temp);

    // 2
    for (int i = 0; i < sphereArray.Length; i++)
    {
        sphereArray[i] = entityManager.Instantiate(sphereEntityPrefab);

        // 3
        entityManager.AddComponentData(sphereArray[i], new Scale
        {});

        //4
        entityManager.SetComponentData(sphereArray[i], new Translation
        {
            Value = new float3(statesGO[i].GetComponent<MeshCollider>().bounds.center.x,
                               statesGO[i].GetComponent<MeshCollider>().bounds.center.y, 0f)
        });

        //5
        entityManager.AddComponentData(sphereArray[i], new DeathComponent
        {
            index = i
        });

    }

    // 6
    sphereArray.Dispose();
}
```

Ilustración 42 Código de creación de entidades

Este código se divide en varias secciones:

1- Creación de una lista de tipo *NativeArray*, un tipo de vector para el caso de las entidades, la longitud de este array será del número de estados que hay en Estados Unidos. *Allocator.Temp* indica que NativeArray no necesitará persistir una vez que se realice la configuración.

2- Creación de un bucle *for* para el acceso de cada estado individualmente.

3- Añadimos a la entidad el component de tipo *Scale*, modificará el tamaño de la esfera según las circunstancias.

4- Otro componente para añadir a la entidad es la posición donde se va a encontrar la esfera cuando se genere los estados. Por defecto estarán en el centro de cada estado.

5- Este componente que se añade a la entidad es creado desde otra clase llamada *DeathComponent*.

```
public struct DeathComponent : IComponentData
{
    public int index;
    public uint deaths;
}
```

Ilustración 43 Creación del componente muertes

Una pequeña observación, no hay “DeathComponent: MonoBehaviour”, en cambio está puesto “DeathComponent: IComponentData”, que es un tipo de comportamiento diferente en el cual solo funciona con ECS, donde se declara a los componentes.
La clase es solo una estructura que tendrá dos variables para inicializarlo más adelante.

Tabla 6 Declaración de variables en DeathComponent

Variable	Función
Index	Identificador del estado
Deaths	Número de muertes en ese estado

Dentro de la entidad que se está añadiendo, se asigna la variable *index* con el valor de la posición del bucle en ese momento.

6- Una vez acabado el bucle, *NativeArray.Dispose()* libera cualquier memoria asignada temporalmente.

Implementando el sistema y sus mecánicas

Teniendo ya la entidad con sus componentes asignados, procedemos a la aplicación de las mecánicas de los componentes.

Al igual que en *DeathComponent*, que tenía su propio comportamiento puesto, esta clase denomindada *DeathSystem*, quién llevará el control de las entidades con sus componentes asignados tendrá también su propio comportamiento llamado *ComponentSystem*.

```
public class DeathSystem : ComponentSystem
{
    3 references
    State states;
    7 references
    float maxSize;
```

Ilustración 44 Declaración de variables en la clase Deathsystem

La variable *States* es de tipo *State*, es una clase creada solo para obtener el nombre del estado y usarlo en el script de ECS.

```
public class State
{

    String[] states;
    string jsonString;

    public State()
    {
        jsonString = File.ReadAllText(Application.persistentDataPath+"/gz_2010_us_040_00_500k.json");
        var N = JSON.Parse(jsonString);
        var features = N["features"];

        states = new String[features.Count];

        for(int i = 0; i < features.Count; i++)
        {
            states[i] = features[i]["properties"]["NAME"];
        }

    }

    public String getStateName(int id)
    {
        return states[id];
    }
```

Ilustración 45 Código de la clase State

Desde el mismo archivo JSON que usamos anterior anteriormente podemos obtener directamente el nombre de los estados. Creamos un Array de tipo *String* con longitud del número de estados. También cada estado tendrá su propio *identificador*.
El método *getStateName()* recibe por parámetro el identificador *(id)* que mandamos y devuelve directamente el nombre del estado con esa id.

```
protected override void OnStartRunning()
{
    states = new State();
    maxSize = 1;

    Entities.ForEach((ref Scale scale, ref DeathComponent sphereComponent) =>
    {
        sphereComponent.deaths = MapController.mInstance.getDeaths(states.getStateName(sphereComponent.index,
        mapController.mInstance.getDate()));
        scale.Value = maxSize;

    });
}
```

Ilustración 46 Inicialización de variables en DeathSystem

Continuando en la clase *DeathSystem*, el método *OnStartRunning()* inicializa todas las variables que se van a usar en esta clase, es como el método *Start()* del resto de scripts, pero la diferencia es que *Start()* no funciona con ECS.

Tabla 7 Variables de la clase DeathSystem

Variables	Función
maxSize	Tamaño máximo que tendrá la esfera.
sphereComponent.deaths	Variable *deaths* cogido de la clase *DeathComponent*, obtiene el número de muertes del estado, recibiendo por parámetro: -El nombre del estado llamado por *getStateName()* y pasando por su parámetro la variable *sphereComponent.index*, que es el identificador del estado. -La fecha en el que encuentra en ese momento el programa.

```
protected override void OnUpdate()
{


    Entities.ForEach((ref Scale scale, ref DeathComponent sphereComponent) =>
    {
        if (sphereComponent.deaths > 10000)
        {
            maxSize = 1.2f;
        }
        else if (sphereComponent.deaths > 5000)
        {
            maxSize = 0.8f;
        }
        else if (sphereComponent.deaths > 1000)
        {
            maxSize = 0.35f;
        }
        else maxSize = 0.2f;


        if(scale < maxSize )
        {
            scale+=0.05f;
        } else {
            scale-=0.05f;
        }
    });

    PlayAdvancingDates();
```

Ilustración 47 Método OnUpdate() de la clase DeathSystem

OnUpdate() será el método principal que llevará a cabo el manejo de los componentes.

Dentro del bucle (*Entities.forEach*, bucle exclusivo para ECS) se pasa por parámetro el tamaño de la esfera (scale) y la entidad del estado. Según el número de muertes que tenga el estado en esa fecha, se modificará la variable *maxSize* para cambiar el valor máximo de esa esfera específica, siendo el tamaño *0.2* el mínimo y *1.2* el máximo, entre ellos se pueden cambiar a 0.35 y 0,8 para los otros casos. Más adelante hay una condición (*if scale < maxSize*) para modificar la escala de forma continua y se aprecie en la interfaz del programa ese cambio.

```
void PlayAdvancingDates()
{

        Entities.ForEach((ref DeathComponent sphereComponent) =>
        {
            sphereComponent.deaths = MapController.mInstance.getDeaths(states.getStateName(sphereComponent.index),
            mapController.mInstance.getDate());
        });
}
```

Ilustración 48 Método PlayAdvancingDates() de la clase DeathSystem

PlayAdvancingDates() crea un bucle similar al anterior, pero la única modificación que hay dentro es el número de muertes, siendo la misma estructura que cuando se declaró en el método *OnStartRunning()*, la diferencia es que este bucle es llamado en el método *OnUpdate()*, es decir, es llamado cada *frame (1/60 segundos)*.

Se realiza la compilación del programa y viene los problemas.

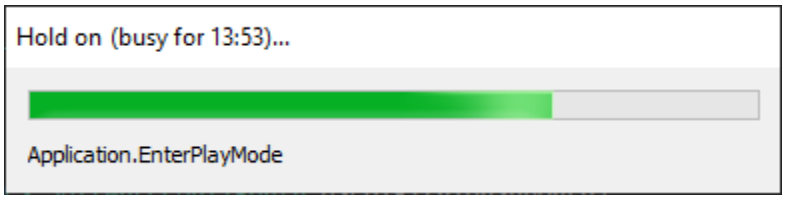


Ilustración 49 Tiempo de espera para la ejecución

El tiempo de ejecución del programa supera los 10 minutos, que es un tiempo demasiado descabellado para este tipo de proyectos, lo normal, estando el ordenador portátil sin enchufar a la toma de corriente y en bajo rendimiento, tomaría unos *50 segundos*, estando enchufado y con un alto rendimiento, *tomaría 30 segundos*. Esto también nos indica que, si el programa tarda esta cantidad exagerada de tiempo en ejecutarse, lo que vaya a mostrar en pantalla también irá muy lento.

Una vez que llega a haber iniciado, el programa muestra la solución.

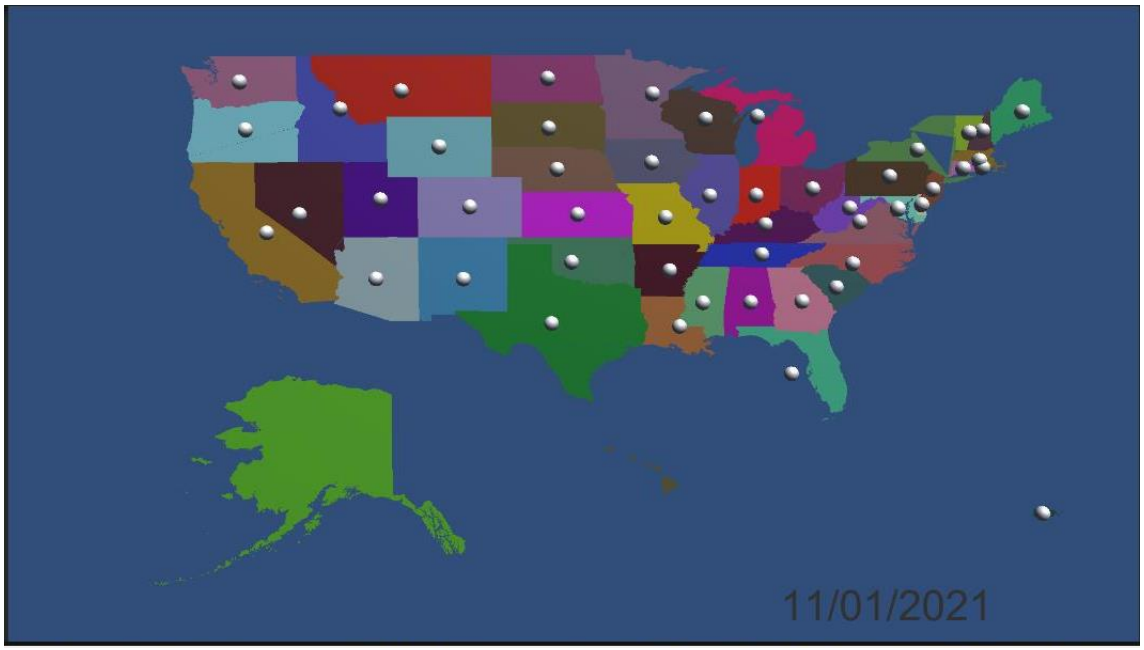


Ilustración 50 Mapa de Estados Unidos con las esferas asignadas en cada estado

Las esferas se han creado correctamente, colocándose en el centro de cada estado, el mapa también se ha generado y la fecha se muestra, pero al comprobar cómo iba la CPU tras ejecutarlo, el programa apenas estaba en funcionamiento.

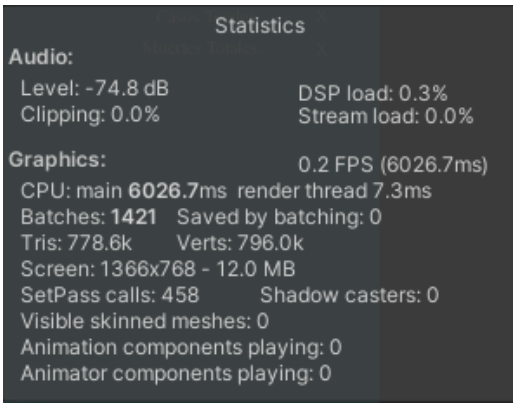


Ilustración 51 Tiempo de CPU en el programa

Simplemente con ver que en los gráficos están ejecutándose en *0.2 FPS* y el tiempo de CPU principal es de 6 segundos, ya dice como está el estado del programa. Por ejemplo, si hay que realizar algún cambio de fecha, se tomaría su tiempo en mostrarse, y que funcione correctamente el cambio de tamaño de la esfera ya es cosa aparte.

¿Por qué va muy lento el programa? Vamos a comprobar los códigos anteriores, se procederá a contar cuantos bucles *for* se ha utilizado para la ejecución del programa.

En la creación de mapas se ha utilizado 5 bucles for y 1 de ellos es anidado, este no supone un gran problema porque es obligatorio el uso de estos bucles para su generación, además que se llama solo una vez, que es cuando inicia el programa.

En *mapController* se llama a 3 bucles for, para obtener el código del estado, el número de casos y de muertes, cada vez que es llamado, el programa tiene que recorrer la lista hasta llegar a ese parámetro exacto, que puede estar al principio o final de la lista, o no estar.

En *deathSystem* llama a 3 bucles for, uno de ellos es llamado al inicializar el programa, otro solo modifica dos variables que son solo números, el último que queda es el crítico, causante del problema de ejecución lenta.

El bucle foreach de las entidades contiene los 52 estados de Estados Unidos, dentro del bucle, se llama a *getDeaths* de la clase controller, que contiene otro bucle, y este tiene una longitud de más de 30000 filas (si encuentra lo deseado antes, pues el rango será menor). Esto hace que el tiempo de complejidad del programa sea muy grande, en este caso, *O* (n^2). Toca optimizar los bucles for para que no nos dé tantos problemas.

Primero, hay que optimizar la búsqueda de datos, creando una base de datos para almacenar las filas que nos pasa y ejecutar sentencias de búsqueda es una opción, pero requeriría de tiempo crear una base de datos y conectarlo al programa y que todo funcione a la perfección.

Una alternativa encontrada en internet es *Linq*, es una biblioteca que tiene C# y permite realizar consultas SQL sin tener que crear una base de datos, perfecto para lo que se necesita en la búsqueda.

```
using System.Linq;
```

Ilustración 52 Implementación de Linq en C#

```
public uint getCases(string state)
{

    var caseQuery =
    from data in data.GetList()
    where data.date == currentDate && data.state == state
    select data;


    foreach (var item in caseQuery)
    {
        return item.cases;
    }

    return 0;
}
```

Ilustración 53 Modificación del método getCases() de la clase mapController

El método *getCases()* de la clase *MapController* se ha modificado enteramente. Ahora recibe un parámetro en vez de dos, que será el nombre del estado, dentro del método se crea una variable llamada *caseQuery*, en ella se genera una consulta en el que, desde la lista *data.GetList()* de la estructura CSV, busca mirando una fecha relacionado con el que se encuentre el programa en ese momento y el nombre del estado con el que se quiere buscar. Como resultado, la variable *caseQuery* recibe una lista de datos buscados. Pese a

que solo recibe un dato encontrado, se realiza un bucle for para devolver el resultado encontrado, esto se hace porque si a *caseQuery* directamente se le añade ".cases" para devolver el resultado, muestra un error en pantalla, entonces desestructurando los datos con foreach ayudaba a tener dicho dato. En el caso que no encuentre ningún resultado devuelve 0.

```
public uint getDeaths(string state)
{
    var deathQuery =
    from data in data.GetList()
    where data.date == currentDate && data.state == state
    select data;


    foreach (var item in deathQuery)
    {
        return item.deaths;
    }

    return 0;
}
```

Ilustración 54 Modificación del método getDeaths() de la clase mapController

GetDeaths() tiene la misma estructura que *GetCases()*, solo que la diferencia es que en vez de devolver el número de casos, devuelve el número de muertes.

Una vez realizado este cambio, ejecutamos el programa de nuevo, y si no tarda en ejecutar, comprobamos como van los gráficos y la CPU.

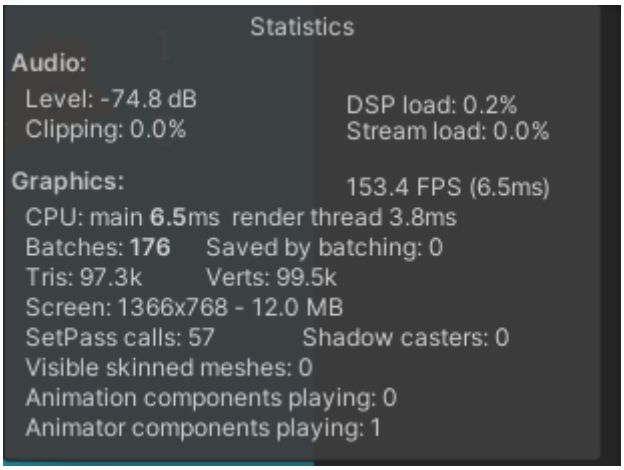


Ilustración 55 Tiempo de CPU después de realizar cambios

El programa está funcionando a la perfección, sin tener lentitud ni tiempos de espera largos.

Capítulo 8 Desarrollo

8.1 Avance de fecha

Para observar que los datos implementados en la esfera están en correcto funcionamiento, hay que crear un método que cambie la fecha actual por su siguiente cada segundo, es decir, si la fecha que está mirando el programa es el 22 de mayo de 2020, que avance al día siguiente, 23 de mayo de 2020.
Para ello, primero hay que saber cómo actualizar la fecha cada segundo, ya que en Unity se está usando el método *Update()* y este se actualiza cada 1/60 segundo.

- La primera manera es usando *UnityEngine.Time.time()*, este método devuelve el tiempo en segundos desde el comienzo de la aplicación.

  Si hacemos una condición en el cual revise que ha pasado un segundo, se puede actualizar el tiempo y por ello, cambiar la fecha al día siguiente.

```
float time = 1.0f;
float nextActionTime = 1.0f;

if (UnityEngine.Time.time() > time){
    MapController.mInstance.setDate(MapController.mInstance.getDate().AddDays(1));
    time += nextActionTime;
}
```

Ilustración 56 Código para aplicar cambio de fecha

MapController.mInstance.setDate(MapController.mInstance.getDate().AddDays (1)) simplemente cambia la fecha del día actual por el siguiente.
Con estas variables y su condición, la fecha cambiará cada segundo al día siguiente. Desgraciadamente, a efectos prácticos, no funciona en su totalidad. Según empieza el programa, la fecha avanza de forma normal, cuando pasa un mes desde el comienzo, el cambio de fecha se vuelve muy lento, y cuando pasa otro mes, el avance que realiza es tan rápido que en tres segundos ha llegado casi al final de su último día de la fecha.

Ilustración 57 Mapa de Estados Unidos con las esferas realizando su funcionamiento

Por otro lado, las esferas cambian de tamaño suavemente, apreciando que cuando el tamaño máximo es modificado, la esfera es cambiada también. Sin embargo, la metodología de avance de tiempo no nos sirve aquí, hay que buscar una alternativa.

- La segunda manera es usando *Time.deltaTime()*, pero esta opción fue descartada rápidamente porque la llamada solo tiene esta función:

> "Delta time (Δt) es el tiempo transcurrido desde el último frame, o usando otras palabras, es el tiempo entre cada frame renderizado.
> Supongamos que queremos dibujar 5 frames por segundo, por lo tanto, vamos a poner un sleep de 0.25 segundos antes del final de la iteración del bucle. 1 segundo / 4 huecos = 0.25 segundos por hueco.

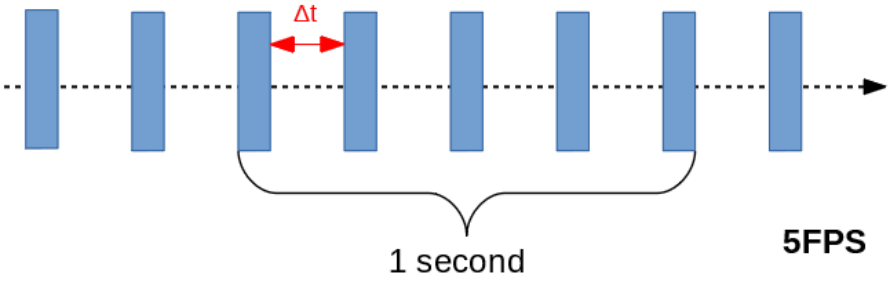


Ilustración 58 Delta time con 5 FPS de ejecución

> Con un valor de fps más bajo el Delta time se incrementa. Como se puede ver en la siguiente imagen, con un valor de 3 FPS el delta

time se incrementaría hasta 0.5 segundos. 1 segundo / 2 huecos = 0.5 segundos por hueco.”

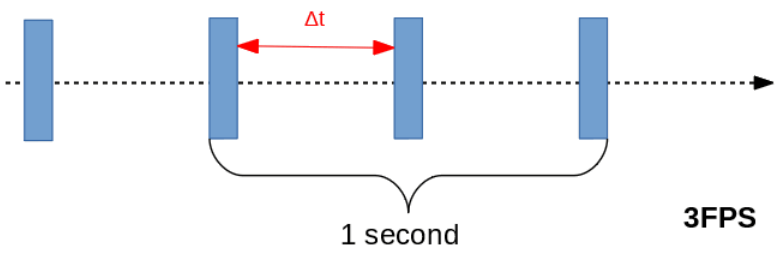


Ilustración 59 Delta time con 3 FPS de ejecución

(BYC, 2017)

Es decir, deltaTime solo mira el flujo del programa y se mantiene estable en un número bajo, asi que esta opción no sirve.

- La tercera opción es utilizando *Corutinas.* La rutina llama al método que tu creas desde el start y ejecuta la función en su totalidad antes de retornar. Una ventaja que tiene la corutina es que puedes estar ejecutando acciones sin parar el progreso del programa. Por ejemplo, puedes hacer que la rutina ejecute la acción de esperar durante un intervalo de tiempo, pero el programa sigue realizando sus acciones sin que se vea afectado. Gracias a esto, podemos hacer que la fecha avance cada segundo y que la rutina pare.

```
public class DateBehaviour : MonoBehaviour
{
    15 references
    public static DateBehaviour mInstance {get; private set;}
    6 references
    public bool timeFlag;
    5 references
    public float timeInterval;

    0 references
    void Awake()
    {
        if(!mInstance) mInstance = this;
    }

    0 references
    void Start()
    {
        timeInterval = 1.0f;
        StartCoroutine(AdvanceTime());
    }
```

Ilustración 60 Declaración de variables de la clase DateBehaviour

Declaramos la clase *DateBehaviour*, en ella se realizará el comportamiento de la corutina. Creamos una variable siendo del tipo de la propia clase, *mInstance* para hacer a esta global y poder usarse en otras clases.

Tabla 8 Variables de la clase DateBehaviour

Variable	Función
timeFlag	Variable de tipo booleano , comprueba si ha pasado el intervalo de tiempo deseado. Si es así, el valor cambia a true.
timeInterval	Tiempo de espera entre intervalos, es decir, un segundo entre cada acción.

```
IEnumerator AdvanceTime()
{
    while (true)
    {
        if (!timeFlag)
        {
            timeFlag = true;
            yield return new WaitForSeconds(timeInterval);
        }
        else
        {
            yield return null;
        }

    }

}
```

Ilustración 61 Método AdvanceTime() de la clase DateBehaviour

Cuando se realiza una corutina, se usa *IEnumerator* para poder llamar a ciertas funciones exclusivas de este. Si la variable timeFlag está en falso, se cambia su valor por verdadero y devuelve la acción de espera durante un segundo (valor asignado en timeInterval), yield devuelve el objeto que implemente en IEnumerator. Si timeFlag es verdadero, devuelve un valor *null*, es decir, no realiza ninguna acción.

Ahora hay una duda, si se ejecuta desde el Start() la corutina, quiere decir que solo se ejecuta una vez, pero, ¿Cómo consigue que pueda ejecutar cada segundo?
El bucle *while(true)* consigue que la corutina sea ejecutada más de una vez, permitiendo que se pueda realizar la acción de espera por cada segundo pasado.

Teniendo esta clase creada, la llamaremos desde donde se está utilizando el avance de fecha, es decir, en la clase *DeathSystem*.

Dentro del método *PlayAdvancingDates()* usamos lo que se creó anteriormente.

```
if (SliderDateBehaviour.mInstance.timeFlag)
{
    if (MapController.mInstance.getDate() < MapController.mInstance.lastDate)
    {
        MapController.mInstance.setDate(MapController.mInstance.getDate().AddDays(1));
    }
    SliderDateBehaviour.mInstance.timeFlag = false;
}
```

Ilustración 62 Modificación del método PlayAdvancingDates() de la clase DeathSystem

Si la variable timeFlag está en verdadero, quiere decir que ha pasado un segundo de intervalo. Como segunda comprobación, hay que revisar si la fecha en la que se encuentra el programa no ha alcanzado al último dia, para evitar errores de muestra de datos en el mapa. Comprobado todo esto, se avanza al día siguiente y así sucesivamente.
Entre las 3 opciones elegidas, las corutinas han dado el mejor resultado para realizar eventos independientes y no causar conflictos, haciendo que avance cada segundo sin que ocurra alguna anomalía en el programa.

8.2 Barra de progreso

Para una mejor visión de avance de fecha, en la interfaz no solo estará la data, también habrá una *barra de progreso* para ubicar al usuario final en que momento de la fecha de datos se encuentra.

Desde Unity creamos un objeto nuevo que será la barra de progreso. Para ello, se hace clic en "+", luego en "UI" y por último en "Slider". En pantalla se mostrará el objeto creado.

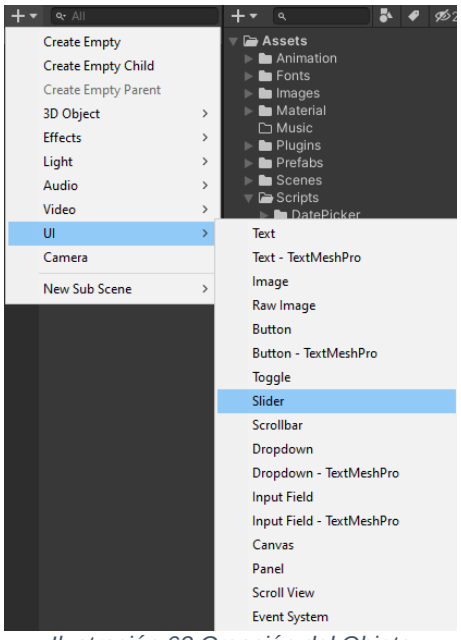


Ilustración 63 Creación del Objeto

Ilustración 64 Objeto en pantalla

En la jerarquía de objetos se aprecia el objeto creado. Dentro de este Iremos a "Slider → Fill Area → Fill" y cambiamos el color de relleno a cualquiera, en este caso puse el color rojo.

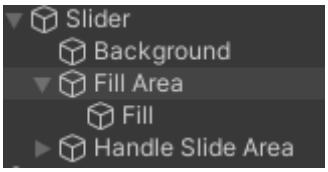


Ilustración 65 Ruta para seleccionar "Fill"

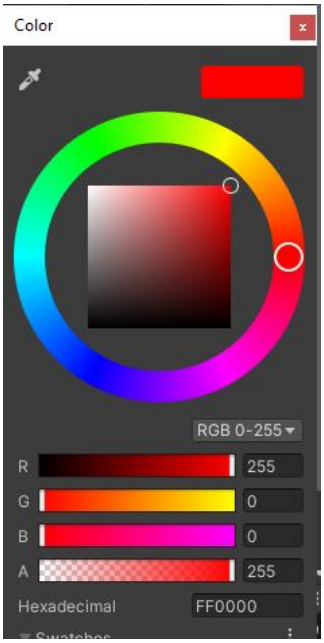


Ilustración 66 Selección de color de la barra de progreso

Ilustración 67 Barra de progreso actualizado

Teniendo la barra creada, se creará su funcionamiento para la aplicación, y lo que mostrará en su progreso será la fecha en la que se encuentra y su posición.

```
using UnityEngine;
using UnityEngine.UI;

0 references
public class ProgressDateController : MonoBehaviour
{
    4 references
    private Slider slider;
    0 references
    void Start()
    {
        slider = GetComponent<Slider>();
        slider.minValue = (float)MapController.mInstance.firstDate.Subtract(System.DateTime.MinValue).TotalSeconds;
        slider.maxValue = (float)MapController.mInstance.lastDate.Subtract(System.DateTime.MinValue).TotalSeconds;
    }

    0 references
    void Update()
    {
        slider.value = (float)MapController.mInstance.getDate().Subtract(System.DateTime.MinValue).TotalSeconds;
    }
}
```

Ilustración 68 Código para la funcionalidad de la barra de progreso en la clase ProgressDateController

Para ello, creamos una clase llamada *ProgressDateController*, y ahí implementamos el objeto "Slider" creado anteriormente. El tipo de objeto es el mismo que su nombre asignado, ya que el tipo "Slider" contiene funciones como su valor mínimo, máximo y actual, que se usará para la interfaz. En el método Start() encontramos:

- slider.minValue: será la fecha mínima de la barra de progreso, el valor que obtiene es la diferencia de valor de la fecha inicial y el valor de la primera fecha que te ofrece el sistema (es decir, el *uno de Enero de Mil novecientos setenta*), esta resta se transforma en segundos, el tipo de variable que te da este resultado es de tipo *double* y se modifica a un valor de tipo *float* para mejor precisión.

- slider.maxValue: será la fecha máxima de la barra de progreso. Al igual que slider.minValue, el valor que obtiene es la diferencia de valor de la fecha final y el valor de la primera fecha que te ofrece el sistema, se transforma en segundos, y la variable de tipo *double* se cambia a una variable de tipo *float*.

Dentro de Update():

- slider.value al igual que slider.minValue y slider.maxValue, se actualiza su valor actual realizando la misma formula: la fecha en la que se encuentra en ese momento (obtenido por MapController.mInstance.getDate()) menos la primera fecha en la que se encuntra el sistema, y transformando el tipo double a float.

Este es el resultado obtenido después de la implementación de la barra de progreso:

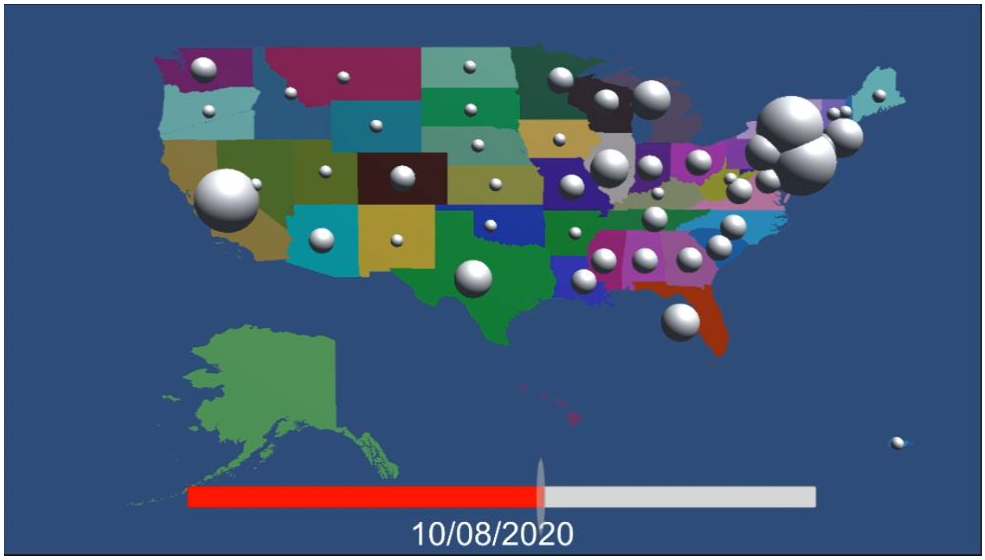


Ilustración 69 Implementación de la barra de progreso en el mapa

8.3 Control de fecha

Ya integrado la fecha y su barra de progreso, toca controlar el paso de estos ya que cuando empieza la ejecución del programa, está avanzando la fecha día por día y no se aprecia con claridad la información del estado en ese momento. Para ello, se introducirán dos botones para sus pruebas:

- Reanudar/Pausa: Este botón simplemente pausa o reanuda el avance de fecha.
- Velocidad: Acelera el tiempo de intervalo entre fechas siendo el doble o el triple de rápido, y también puede volver a su ritmo normal.

Con la integración de los botones, en *DateBehaviour* cambiamos su nombre de clase por *SliderDateBehaviour* debido a que la barra de progreso se verá afectada por esta clase. En ella añadimos una variable booleana más llamada *isPause*, controlando asímismo el comportamiento del programa para que avance o no de fecha dependiendo si pulsamos el botón de pausa o no.

```
isPause = true;
```

```
if (!isPause)
{
    if (!timeFlag)
    {
        timeFlag = true;
        yield return new WaitForSeconds(timeInterval);
    }
    else
    {
        yield return null;
    }

} else yield return null;
```

Ilustración 70 Código para la llamada de intervalo de tiempo en la clase SliderDateBehaviour (llamado anteriormente DateBehaviour)

En la clase *deathSystem* se añade esta función también como una condición para que ejecute su comportamiento:

```
if (!SliderDateBehaviour.mInstance.isPause)
{
    if (SliderDateBehaviour.mInstance.timeFlag)
    {
        if (MapController.mInstance.getDate() < MapController.mInstance.lastDate)
        {
            MapController.mInstance.setDate(MapController.mInstance.getDate().AddDays(1));
        }
        SliderDateBehaviour.mInstance.timeFlag = false;
    }
}
```

Ilustración 71 implementación del intervalo de tiempo en la clase deathSystem

Para modificar el comportamiento del botón de pausa y sus otros botones, la clase *ButtonController* controlará estos eventos.

```
public Text playPauseText;
3 references
public Text speedText;
5 references
int speedStatus;
```

Ilustración 72 inicialización de variables en la clase ButtonController

Tabla 9 Variables de la clase ButtonController

Variables	Función
playPauseText speedText	Contiene el texto del botón, se modificará dependiendo del estado en el que se encuentre.
speedStatus	Velocidad en el que se encuentra en ese momento el cambio de fecha.

```
public void resumeOrPause()
{
    if (SliderDateBehaviour.sliderDate.isPause)
    {
        playPauseText.text = "Pause";
        SliderDateBehaviour.sliderDate.isPause = false;
    }
    else
    {
        playPauseText.text = "Play";
        SliderDateBehaviour.sliderDate.isPause = true;
    }
}
```

Ilustración 73 Método para pausar o continuar el progreso de fecha en la clase ButtonController

```
public void changeSpeed()
{
    speedStatus++;
    if (speedStatus > 2) speedStatus = 0;
    switch (speedStatus)
    {
        case 0:
            speedText.text = "x1";
            SliderDateBehaviour.sliderDate.timeInterval = 1.0f;
            break;
        case 1:
            speedText.text = "x2";
            SliderDateBehaviour.sliderDate.timeInterval = 0.5f;
            break;
        case 2:
            speedText.text = "x3";
            SliderDateBehaviour.sliderDate.timeInterval = 0.3f;
            break;
    }
}
```

Ilustración 74 Método para el cambio de velocidad del progreso de fecha en la clase ButtonController

Estos métodos básicamente cambian el texto de los botones dependiendo del estado en el que se encuentre en ese momento, y en el de velocidad, el tiempo de intervalo de la barra de progreso se verá afectada por el evento botón y cambiando cada vez que es pulsado.

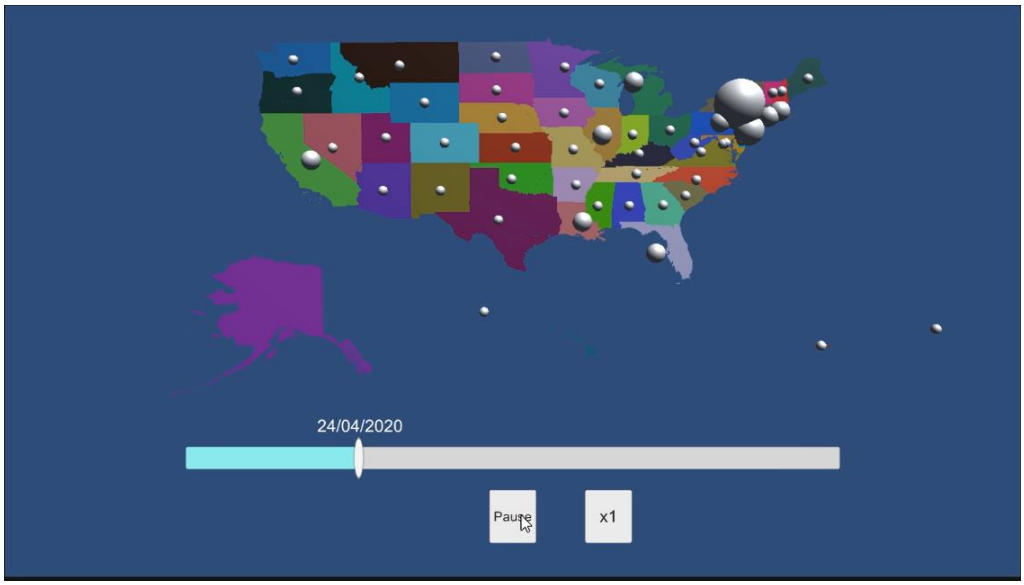


Ilustración 75 Implementación de botones en el programa

Ahora se puede pausar, reanudar y acelerar el avance de fecha, donde se aprecia más las esferas cambiantes. Aunque todavía queda por controlar otro detalle: la fecha también hay que llevarlo a un día exacto confirmando que la funcionalidad de la esfera sea la correcta en el caso que hayamos avanzado tanto en una fecha, retroceder atrás los días y que las esferas encojan también. Un tercer botón es añadido:

- Cambiar: Cambia la fecha actual por alguna puesta de forma predeterminada.

```
public void changeDate()
{
    MapController.mapController.setDate(DateTime.Parse("04/20/2020", new CultureInfo("en-US")));
}
```

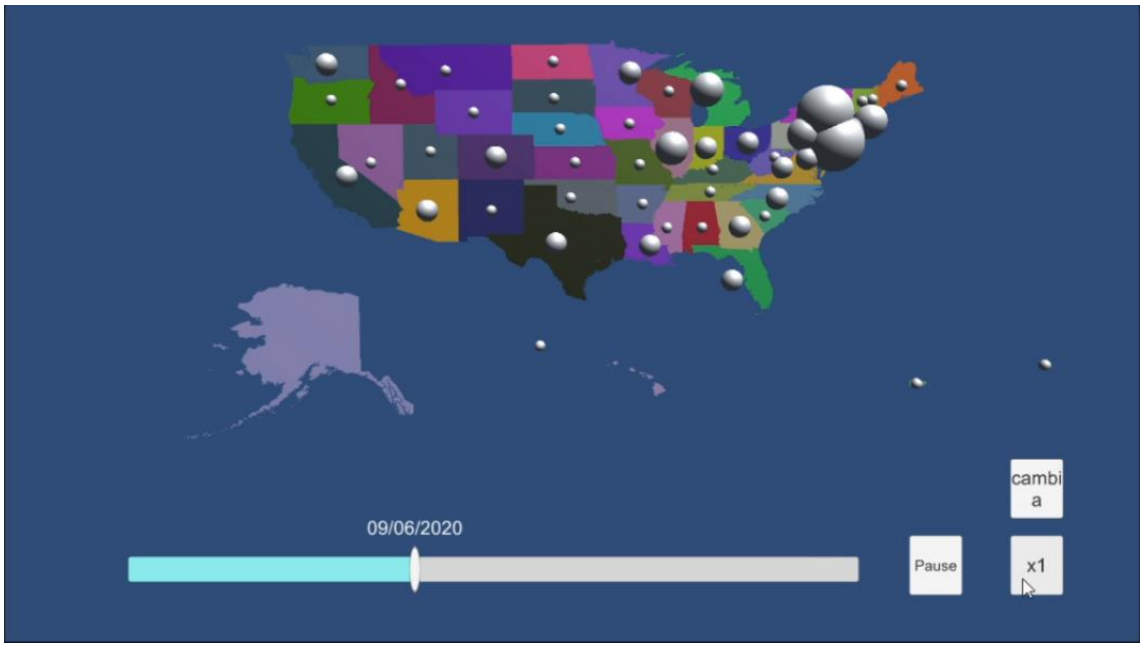
Ilustración 76 Método para cambiar la fecha desde la clase ButtonController

Como detalle, dentro del método *Update()* de la clase *ProgressDateController* se añade una función más:

```
MapController.mInstance.dateText.GetComponent<RectTransform>().transform.position = new Vector3(
    slider.handleRect.transform.position.x + 15, slider.handleRect.transform.position.y + 30);
```

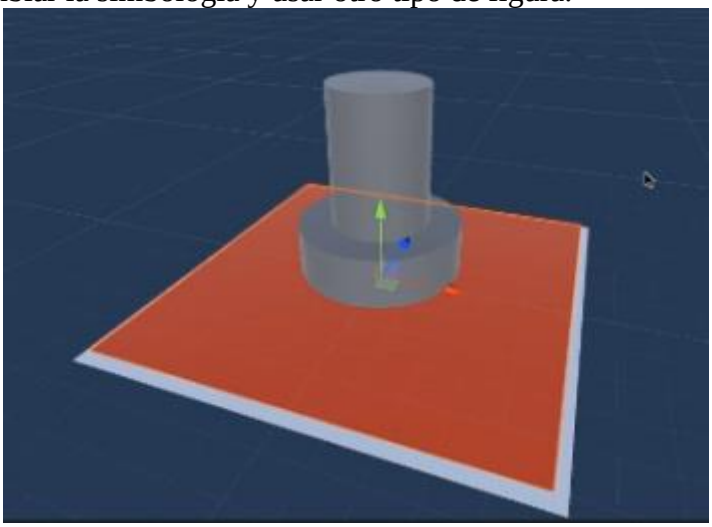
Ilustración 77 Movimiento de texto junto al progeso en la clase ProgressDateController

- El texto mostrando la fecha, que se encuentra en *MapController*, es llamado y modifica su posición en la interfaz, colocandolo justo por encima de donde se encuentra la fecha actual posicionada en la barra de progreso. Y por cada segundo que se actualice, el texto tambíen se verá afectado y se desplazará con el progreso.



Este es el resultado parcial del proyecto en ese momento. Se crean las figuras de los estados con sus esferas (a excepción de algunas), tiene la fecha en la que se encuentra en ese momento mostrándose en pantalla junto a una barra de avance, y con tres botones para el control de la fecha. El funcionamiento principal está integrado en el proyecto, ahora toca mejorar detalles, posiciones, y añadir más simbologías.

Mostrado esto al tutor D. José Miguel Santana Núñez y D. Agustín Trujillo en una reunión, se habla de cambiar la figura de esfera por *dos cilindros concéntricos*, debido a que no solo hay que mostrar el número de muertes gráficamente, también el número de casos en muy importante, y una esfera no puede representar ambas cosas. Por esta razón se procede a cambiar la simbología y usar otro tipo de figura.



8.4 Cambio de figuras e Implementación de número de casos en ECS

El proceso de cambio de figuras no es costoso, pero un problema que hay es que ECS solo trata con un objeto, no con varios, entonces la implementación de figuras es diferente. Para este caso, el prefab que usábamos en la esfera se cambia y se introduce

un objeto vacío y poner dentro de este, ambos cilindros para que el programa lo interprete como un prefab solo.

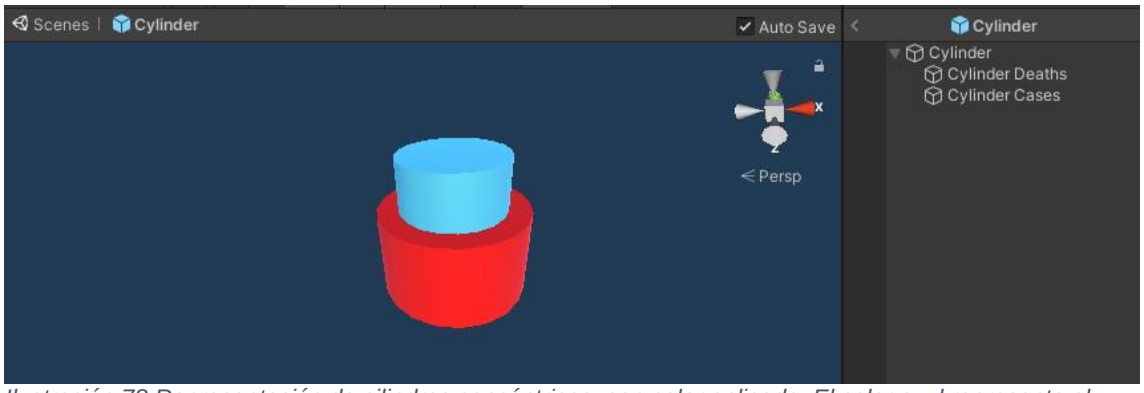


Ilustración 78 Representación de cilindros concéntricos, con color aplicado. El color azul representa el número de casos, y el rojo las muertes.

En la clase principal, llamado *SetupECS*, clase donde en un principio tenemos implementado la generación de estados y la antigua esfera, se realizan par de cambios.

```
[SerializeField] private GameObject cylinderPrefab;
2 references
private Entity cylinderDeathsEntityPrefab;
2 references
private Entity cylinderCasesEntityPrefab;
```

Ilustración 79 Modificación de la clase SetupECS

El nombre pasa de *SpherePrefab* a *cylinderPrefab*, en vez de haber una entidad, habrá dos: uno controla el número de casos y otro el de muertes.

```
private void Start()
{
    GenerateUSA();

    entityManager = World.DefaultGameObjectInjectionWorld.EntityManager;

    var settings = GameObjectConversionSettings.FromWorld(World.DefaultGameObjectInjectionWorld, null);

    cylinderDeathsEntityPrefab = GameObjectConversionUtility.ConvertGameObjectHierarchy
    (cylinderPrefab.transform.GetChild(0).gameObject, settings);

    SpawnCylinderDeaths();

    cylinderCasesEntityPrefab = GameObjectConversionUtility.ConvertGameObjectHierarchy
    (cylinderPrefab.transform.GetChild(1).gameObject, settings);

    SpawnCylinderCases();

}
```

Ilustración 80 Modificación de generación de entidades en la clase setupECS

entityManager y *settings* mantienen sus valores, las configuraciones por defecto de conversión serán de dos en vez de uno como era en la esfera (cylinderDeathsEntityPrefab

y cylinderCasesEntityPrefab) y, por último, se cambia el nombre del método al suyo adecuado actual y se añade un segundo método para la creación del cilindro de casos.

```
private void SpawnCylinderCases()
{
    //1
    NativeArray<Entity> cylinderArray = new NativeArray<Entity>(statesGO.Length, Allocator.Temp);

    // 2
    for (int i = 0; i < cylinderArray.Length; i++)
    {
        cylinderArray[i] = entityManager.Instantiate(cylinderCasesEntityPrefab);
        // 3
        entityManager.AddComponentData(cylinderArray[i], new NonUniformScale
        {});

        entityManager.SetComponentData(cylinderArray[i], new Translation
        {
            Value = new float3(statesGO[i].GetComponent<MeshCollider>().bounds.center.x,
                               statesGO[i].GetComponent<MeshCollider>().bounds.center.y, 0f)
        });

        entityManager.AddComponentData(cylinderArray[i], new CaseComponent
        {
            index = i
        });

    }

    // 5
    cylinderArray.Dispose();
}
```

Ilustración 81 Método de creación de cilindro para los casos, ubicado en la clase stupECS

Contiene las mismas funciones y características que cuando se había creado la esfera para el número de muertes, pero aquí se comprobaría el número de casos. Por último se realiza un cambio muy importante tanto en el método de *SpawnCylinderCases()* como en el de *SpawnCylinderDeaths().*

Antes:

```
// 3
entityManager.AddComponentData(sphereArray[i], new Scale
{});
```

Después:

```
// 3
entityManager.AddComponentData(cylinderArray[i], new NonUniformScale
{});
```

Ilustración 82 Modificación del tipo de escala

¿Qué nos quiere decir esto? La diferencia entre Scale y NonUniformScale es el cómo cambia el tamaño. *Scale hace crecer o encoger las escalas "x", "y" y "z" simultáneamente*, restringiendo la opción de poder agrandar o reducir uno de los tres ejes de escala. *NonUniformScale te permite aumentar o disminuir el tamaño de uno de ellos, siendo independiente cada uno.* Lo más importante ahora es que la figura modifique su escala solo en el eje "y", es decir, la altura, esto es posible con la ayuda de *NonUniformScale.*

Modificado esto, implementamos los mismos componentes y sistema que hay en muertes, pero aplicándolo al número de casos. Las nuevas clases son *CaseComponent* y *CaseSystem.* En ellas se modifica el valor de escala y su tipo de objeto.

```
Entities.ForEach((ref Translation position, ref NonUniformScale scale, ref CaseComponent cylinderComponent) =>
{
    cylinderComponent.cases = MapController.mInstance.getCases(states.getStateName(cylinderComponent.index));
    scale.Value = new Unity.Mathematics.float3(0.5f, 0.25f, 0.5f);
```

Ilustración 83 Declaración de entidades con su escala modificada, para los casos.

```
Entities.ForEach((ref Translation position, ref NonUniformScale scale, ref DeathComponent cylinderComponent) =>
{
    cylinderComponent.deaths = MapController.mInstance.getDeaths(states.getStateName(cylinderComponent.index));
    scale.Value = new Unity.Mathematics.float3(0.7f, 0.2f, 0.7f);
```

Ilustración 84 Declaración de entidades con su escala modificada, para las muertes.

Se declara el valor de escala y declaramos sus valores principales, en la figura de casos, la escala "x" y "z" será menor al de la figura de muertes, pero la escala "y" será mayor a causa de que el número de casos aumentará a una mayor velocidad que el número de muertes, y visualmente se verá con una altura mayor.

Otro detalle importante cambiado, ubicado en la clase de sistemas de estos componentes, es la forma en la que crece y encoge la figura puesta en código.

Antes:

```
if(scale < maxSize )
{
    scale+=0.05f;
} else {
    scale-=0.05f;
}
```

Ilustración 85 Modificador de escala antiguo

Ahora:

```
scale.Value.y = Mathf.Lerp(scale.Value.y, maxSize, 0.005f);
```

Ilustración 86 Modificador de escala nuevo, aplicando Mathf.Lerp

El código de antes no ayudaba al programa a mejorar, estaba cambiando constantemente la escala incluso si había llegado a su tamaño máximo, añadiendo que era un desperdicio de memoria. *Mathf.Lerp()* realiza una interpolación entre la escala en la que se encuentra la figura en ese momento y su tamaño máximo, cambiándolo 0.005 unidades para un efecto gradual y dejando de modificar si ha llegado a su valor máximo de altura. Ahorra tanto el código como su memoria.

Y, por último, pero no el menos importante, es la modificación de la posición de los cilindros tanto en Alaska como en Hawái. En el programa las esferas no se mostraban en los estados debido a que estos se habían trasladado de sitio. Para mover los cilindros también, cuando se inicialicen las figuras en los sistemas, se especifican que para estos dos casos, las figuras se trasladen a donde estén ellos.

```
switch (states.getStateName(cylinderComponent.index))
{
    case "Alaska":
        position.Value = new Vector3(position.Value.x - 68, position.Value.y + 3, 0);
        break;

    case "Hawaii":
        position.Value = new Vector3(position.Value.x + 10, position.Value.y - 4, 0);
        break;
}
```

Ilustración 87 Colocación de cilindros en los estados de Hawái y Alaska

Tras estas modificaciones, el resultado obtenido es el siguiente:

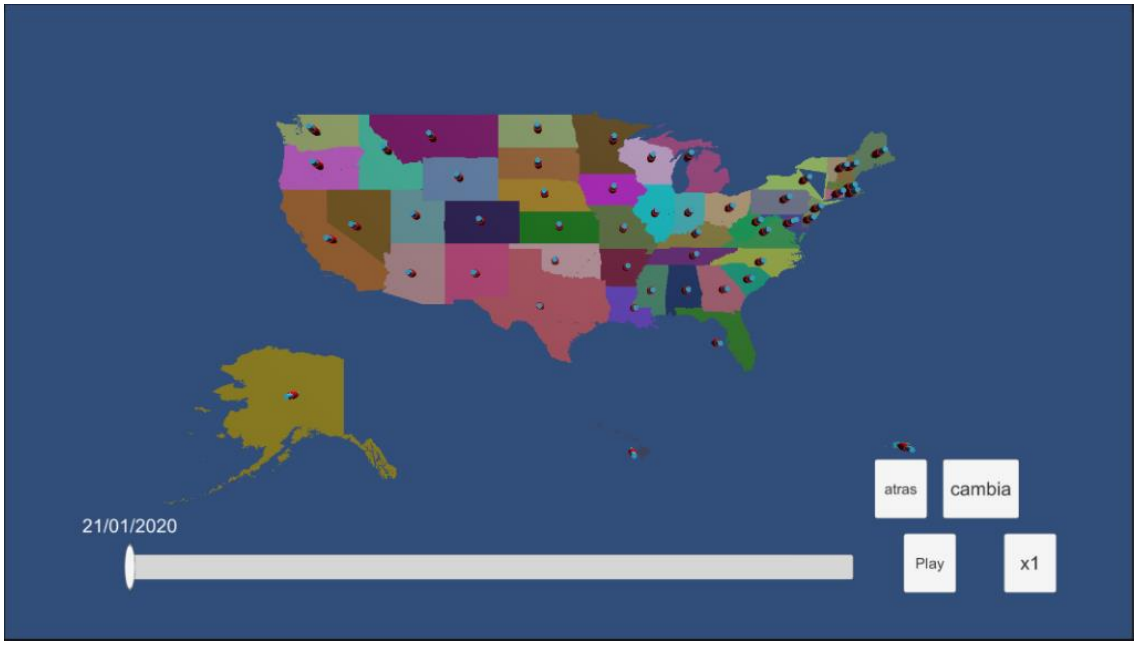


Ilustración 88 Mapa de estados unidos con los cilindros concéntricos aplicados

El cambio se ha aplicado con éxito, y mientras avanza la fecha, los cilindros se modifican también, sin embargo, no se aprecian con mucha claridad porque se encuentran alejado de la cámara.

8.5 Movimiento de cámara

El siguiente objetivo es modificar los valores de la cámara para que pueda estar cerca del estado y muestre los cilindros. A tal efecto, primero comprobaremos cuáles serán los valores adecuados para una buena perspectiva de visión. La cámara principalmente tiene estos valores cuando se ejecuta y muestra el país:

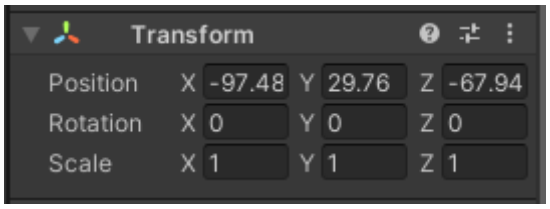


Ilustración 89 Posición de la cámara

Cuando está en ejecución, lo pausamos y movemos la cámara arbitrariamente para acercarlo al estado y anotar sus futuros valores para cuando queramos acercarla.

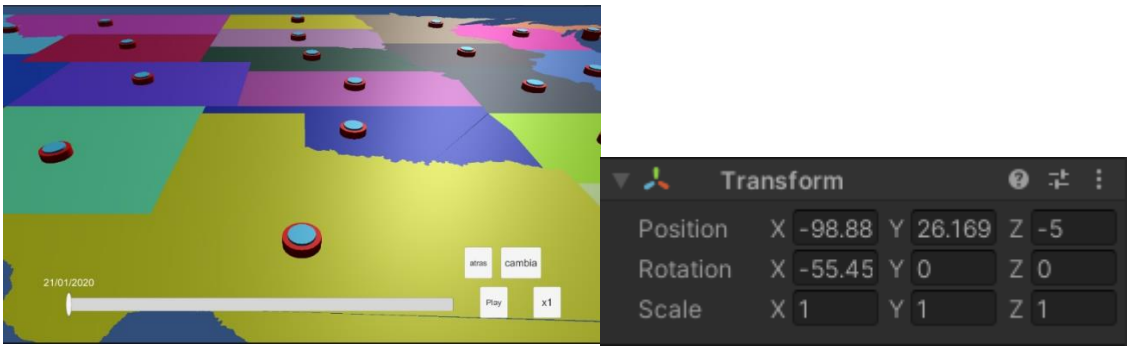


Ilustración 90 Posición de la cámara modificada

Anotamos estos valores y se procede a crear una clase llamada *ZoomCamera*, que realice el movimiento de la cámara desde donde se encuentra por defecto hasta la posición deseada.

```
public class ZoomCamera : MonoBehaviour
{
    6 references
    public static ZoomCamera mInstance { get; private set; }
    4 references
    public bool pressed;
    2 references
    Vector3 target;
    2 references
    Vector3 defaultMap;
    3 references
    Quaternion defaultRotation;
    7 references
    float rotationX;
    1 reference
    private GameObject state;
```

Ilustración 91 Declaración de variables en la clase ZoomCamera

```
void Awake()
{
    if (!mInstance)
    {
        mInstance = this;

    }
}

0 references
void Start()
{
    pressed = false;
    defaultMap = Camera.main.transform.position;
    defaultRotation = Camera.main.transform.rotation;
    rotationX = defaultRotation.x;

}
```

Ilustración 92 Inicialización de variables en la clase ZoomCamera

Tabla 10 Variables de la clase ZoomCamera

Variable	Función
mInstance	Declara que la clase será global.
pressed	Comprueba si se ha realizado click sobre un estado.
target	Objetivo a donde irá la cámara.
defaultMap	Posición por defecto que tiene la cámara cuando está siendo ejecutado.
defaultRotation	Rotación que tiene por defecto la cámara cuando está incializado.
rotationX	Rotación “x” de la cámara.
state	Objeto del estado de Estados Unidos a acercar.

```
void Update()
{
    if (pressed)
    {
        if (rotationX > -55) rotationX -= 0.5f;
        Camera.main.transform.rotation = Quaternion.Euler(rotationX, 0, 0);
        Camera.main.transform.position = Vector3.MoveTowards(Camera.main.transform.position, target, 50 * Time.deltaTime);
    }
    else
    {
        if (rotationX < defaultRotation.x) rotationX += 0.5f;
        Camera.main.transform.rotation = Quaternion.Euler(rotationX, 0, 0);
        Camera.main.transform.position = Vector3.MoveTowards(Camera.main.transform.position, defaultMap, 50 * Time.deltaTime);
    }
}
```

Ilustración 93 Función de la clase ZoomCamera

En cada actualización, se comprueba si se ha pulsado sobre un estado, si fue así cambia los parámetros de la cámara y acércalo al estado poco a poco. Si no, deja los valores de la cámara por defecto y aléjalo si es necesario.

```
public void setTarget(Vector2 stateCoords)
{
    target = new Vector3(stateCoords.x + 1.2f, stateCoords.y - 5, -5);
}

1 reference
public void setState(GameObject s)
{
    state = s;
}

0 references
public void back()
{
    pressed = false;
}
```

Ilustración 94 Métodos para la llamada al objeto y su dirección a él, ubicado en la clase ZoomCamera.

setTarget coloca al objetivo de la cámara para acercarlo con los valores de coordenadas del estado a mirar. *setState()* obtiene el objeto del estado. *Back()* simplemente coloca la cámara a su posición original. Este evento se realiza desde un botón implementado en la interfaz que se llama “atrás”.

Todos estos métodos reciben algo por parámetro, ¿desde dónde vienen? Desde la clase *StateBehaviour*, que contiene la información de los estados ahí para transmitirlos a esta clase.

```
private void OnMouseDown()
{
    if (!EventSystem.current.IsPointerOverGameObject())
    {
        ZoomCamera.mInstance.pressed = true;
        state = gameObject;
        col = state.GetComponent<MeshCollider>();
        switch (state.name)
        {
            case "Alaska":
                centerPoint = new Vector3(col.bounds.center.x - 68, col.bounds.center.y, 0f);
                break;
            case "Hawaii":
                centerPoint = new Vector3(col.bounds.center.x + 10, col.bounds.center.y - 5, 0f);
                break;
            default:
                centerPoint = new Vector3(col.bounds.center.x, col.bounds.center.y, 0f);
                break;
        }
        ZoomCamera.mInstance.setState(state);
        ZoomCamera.mInstance.setTarget(centerPoint);
    }
}
```

Ilustración 95 Modificación de la clase StateBehaviour, añadiendo nuevas funciones al método OnMouseDown()

Cuando se hace clic en un estado, comprobamos si no hay ninguna interfaz por encima que esté estorbando al evento y pueda ejecutarlo sin problemas. Dentro de la condición, se declara que la región ha sido pulsada, se obtiene el centro del objeto para que la cámara se desplace para allá (en los casos de Alaska y Hawai hay que modificar los valores para que la cámara se acercen a ellos, si no, se enfocaría en otro lugar donde no se encuentren), y pasamos a los métodos de *ZoomCamer*a el objeto y su objetivo para la cámara.

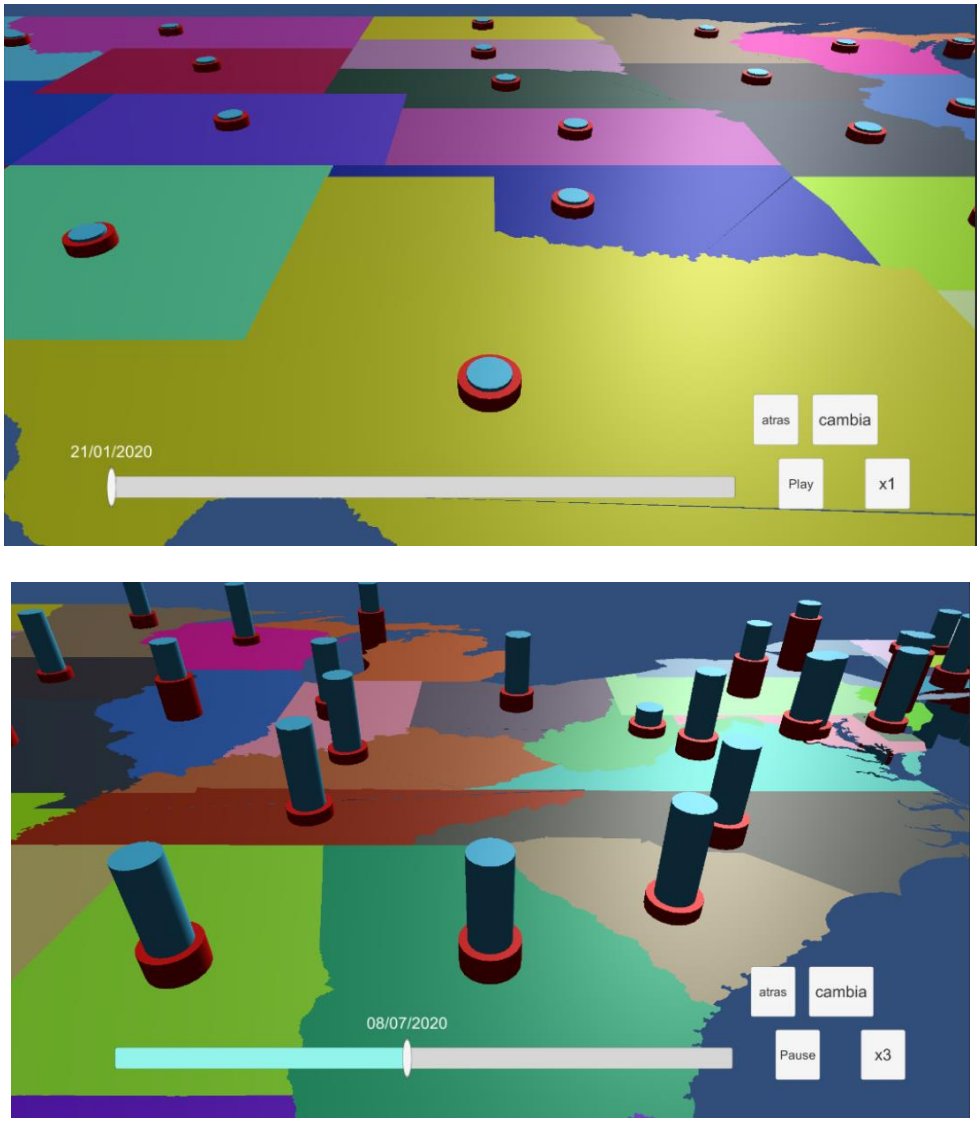


Ilustración 96 Resultado del movimiento de cámara aplicado en el mapa

La cámara está en completo funcionamiento y se puede mover entre los estados, apreciando cómo se modifica la altura de los cilindros con el paso del tiempo. También si se pulsa el botón “atrás” la cámara se aleja del estado y vuelve a su posición original. Pero, el movimiento de la cámara vuelve a ir lento y el tiempo de CPU vuelve a aumentar. Esta vez es debido a que tanto en *caseSystem* como en *deathSystem* se está llamando al bucle for de las entidades cada frame y esto ralentiza el programa. Una solución es obtener la fecha en la que se ha realizado el bucle, una vez recogido, se

comprueba si la fecha obtenida es diferente a la fecha actual, si es así, llama al bucle for y realiza su función.

```
if (isDateChanged != MapController.mInstance.getDate())
{
    Entities.ForEach((ref DeathComponent cylinderComponent) =>
    {
        cylinderComponent.deaths = MapController.mInstance.getDeaths(states.getStateName(cylinderComponent.index));
    });
    isDateChanged = MapController.mInstance.getDate();
}
```

Ilustración 97 Modificación de la clase caseSystem y DeathSystem, añadiendo una condición extra para evitar iteraciones

El tiempo de CPU vuelve a estabilizarse y el programa se ejecuta con fluidez.

8.6 Mostrar información del estado

La información mostrada anteriormente tenía datos muy irrelevantes, y falta de otros importantes, juntando a que la interfaz que había era muy simple, y que además el entorno del programa se ha editado totalmente, entonces la forma de enseñar la información ahora es diferente.

Se ha creado un boceto del proyecto en la página (draw.io, s.f.) para tener una referencia de cómo sería el funcionamiento del programa y que cosas tendría que mostrar por pantalla:

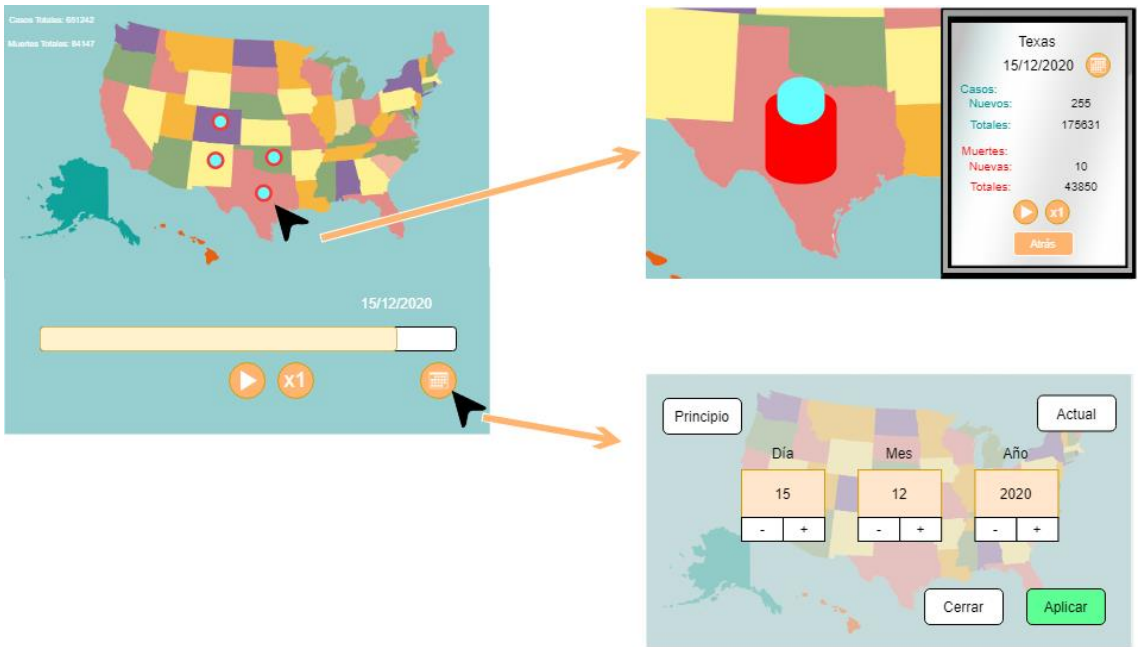


Ilustración 98 Boceto de la aplicación

El usuario tendrá un mapa de Estados Unidos por pantalla, mostrando en ella los cilindros, la fecha con su progreso, tres botones que serán para pausar o continuar el avance la data, la velocidad para cambiar el intervalo de tiempo entre ellos, y el cambio de fecha a uno específico. Y también mostrará una información sobre el número de casos y muertes totales en todo el país.

Si el usuario pone la fecha del ratón encima de un estado y hace clic en él, la cámara ejecutará su movimiento y ser acerca al estado deseado por el usuario final. Mientras se acerca, la barra de progreso, su fecha y sus botones son removidos y una tabla con información del estado es mostrada. En ella, se reflejará:

- Nombre del estado
- Fecha actual
- Número de casos, tanto los nuevos casos, como los totales.
- Número de muertes, tanto las nuevas muertes, como las totales.
- Los botones de pausar o reanudar, el cambio de ritmo de avance, cambio de fecha y el botón de atrás para restablecer la posición original de la cámara y quitar la tabla de información del estado, haciendo volver la barra de progreso y sus botones.

Por otro lado, si el usuario pulsa el botón de cambiar fecha (representado con un icono de calendario), se abrirá una ventana con una selección de fecha, donde el usuario puede modificar tanto el día, como el mes o el año, y también puede volver a la primera fecha contenida en el archivo CSV, o ir al último día en el que se encuentre en el archivo.

Volviendo a la interfaz de Unity, primero modificaremos los colores de los estados, para que no sean aleatorios y formen parte de una paleta predefinido. El color marrón es aplicado en ellos, ya que representa el color de la tierra. Estos valores por modificar se encuentran en la clase *SetupECS*, se cambia los valores de colores aleatorios y aplicando el siguiente color: *0xBC8E47*, siendo este el color marrón claro.

```
Color32 newColor = new Color32(0xbc, 0x8e, 0x47, 255);
statesGO[i].GetComponent<MeshRenderer>().material.color = newColor;
```

Ilustración 99 Asignación de colores a los estados

También el color del fondo que muestra el mapa, se cambia el color a uno que se relacione con el color del mar. Para ello se crea un objeto llamado “Mar” que será un cuadrado grande que simule el color del mar y que ocupe una gran cantidad de tamaño, para cuando se usa el cambio de perspectiva de la cámara, no solo vea la mitad de la figura y la mitad del fondo por defecto. Este cuadrado se coloca justo detrás del mapa para dar la sensación del mar.

Para el color, desde el propio Unity, se crea un material que tenga los colores del mar, el código de color será *0x19DCFF*

Ilustración 100 Asignación de color al mar

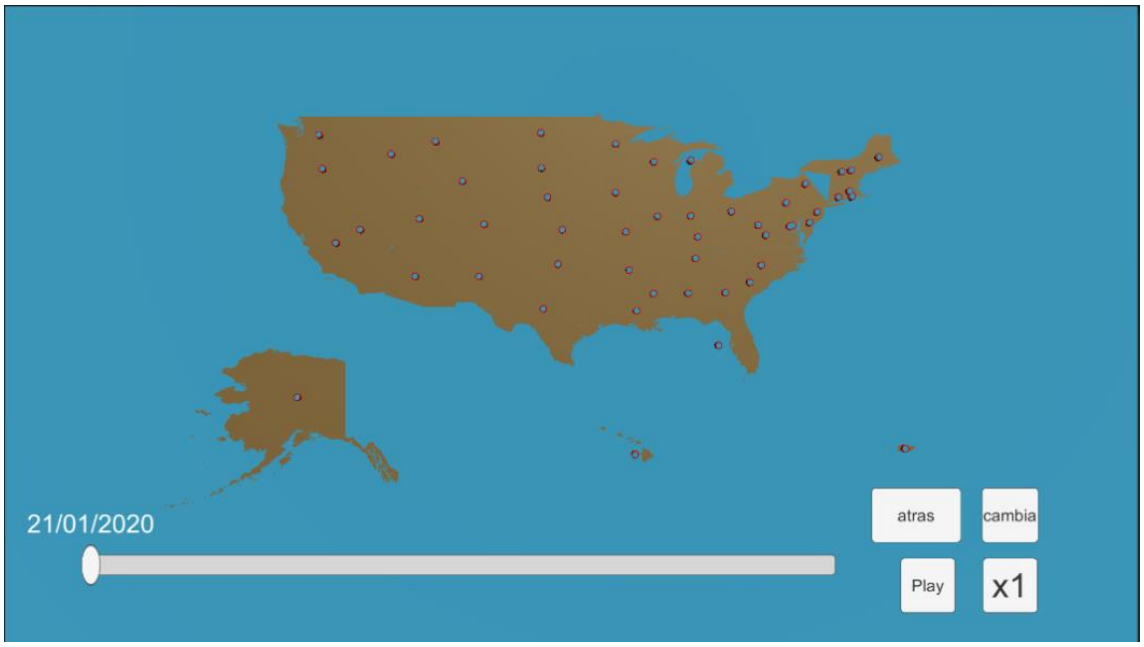


Ilustración 101 Mapa de Estados Unidos con sus colores aplicados

Ya aplicados los colores de los estados y del mar, el siguiente paso es crear un panel que contenga la información del estado.

Este es el panel creado por defecto y que contendrá la información del estado:

Ilustración 102 Panel de información

Una idea para que aparezca este panel en el programa, es, cuando el usuario haga clic en un estado, durante el movimiento de la cámara, el panel será desplazado hacia la pantalla. Para esta función, hay que animar el panel, usando el sistema de animación que ofrece Unity para colocar estados en los que se vaya a encontrar el panel y decir por código que cuando sea verdadero, se dirija al estado que anima el panel para mostrarlo en pantalla, y cuando sea falso, este se desplace a su posición original, es decir, que no se vea en pantalla. El diagrama de animación es el siguiente:

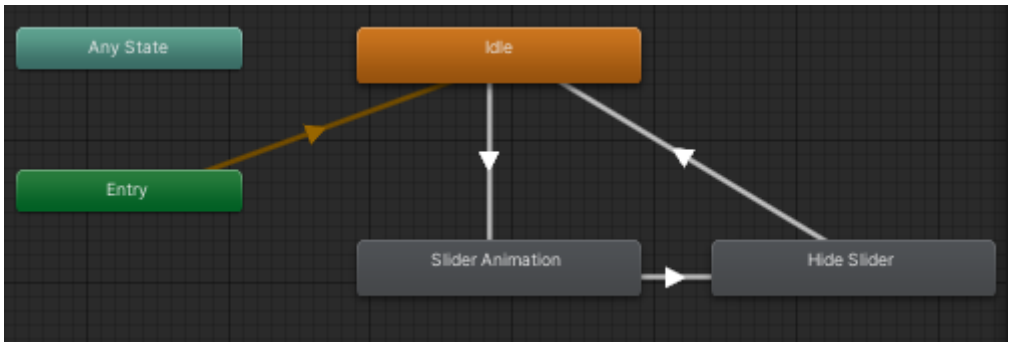


Ilustración 103 Animación de la tabla

El panel se encuentra en un estado estático o “idle”, es decir, no se muestra por pantalla. Si hace clic sobre un estado, se pasa a “Slider Animation” y mueve el panel hacia la pantalla, cuando el usuario le da al botón atrás en el panel, este estado de animación pasará a “Hide Slider” para realizar el movimiento del panel saliendo de la pantalla, y una vez hecho esto, pasa al estado “idle” de nuevo.

Dentro del panel se añaden los textos necesarios de representación de información, y por código se irán modificando estos datos. La clase *StateInfo* contendrá los textos a modificar y las acciones de animaciones.

El método *setStateInfo* recibe por parámetro el nombre del estado. Este método es llamado desde la clase *StateBehaviour*:

```
StateInfo.mInstance.setStateInfo(state.name);
```

Ilustración 104 Método llamado desde StateBehaviour

```
public void setStateInfo(string state)
{
    this.state = state;

}
```

Ilustración 105 Obtención del nombre del estado en la clase StateInfo

```
void Update()
{
    if (state != "")
    {
        uint cases = MapController.mInstance.getCases(state);
        0 references
        uint previousCases = MapController.mInstance.getPreviousCases(state);
        uint deaths =MapController.mInstance.getDeaths(state);
        uint previousDeaths = MapController.mInstance.getPreviousDeaths(state);

        stateText.text = state;
        fecha.text = "" + MapController.mInstance.getDate().ToShortDateString();
        casesText.text = "" + MapController.mInstance.getCases(state);
        deathsText.text = "" + MapController.mInstance.getDeaths(state);
        if((int)(cases - previousCases) <0)
        {
            caseNewText.text = "0";
        } else caseNewText.text = "" + (cases - previousCases);

        if((int)(deaths - previousDeaths) <0)
        {
            deathNewText.text = "0";
        } else deathNewText.text = "" + (deaths - previousDeaths);
    }
}
```

Ilustración 106 Código para aplicar texto de información en pantalla, ubicado en la clase stateInfo

En el Update() si el estado contiene un nombre, que muestre su nombre, los casos y muertes del estado, junto a los casos nuevos y muertes nuevas.

Para la obtención de casos y muertes nuevas, en la clase *MapController*, se añaden dos métodos más: *getPreviousCases() getPreviousCases()* y *getPreviousDeaths()*. En ellos, se obtienen el número de casos y muertes del día anterior al actual.

```
lastCurrentDate = currentDate.AddDays(-1);
```

Ilustración 107 Variable que obtiene el día anterior

```
public uint getPreviousCases(string state)
{

    var caseQuery =
    from data in data.GetList()
    where data.date == lastCurrentDate && data.state == state
    select data;


    foreach (var item in caseQuery)
    {
        return item.cases;
    }

    return 0;
}
```

Ilustración 108 Obtención del número de casos del día anterior

```
public uint getPreviousDeaths(string state)
{
    var deathQuery =
    from data in data.GetList()
    where data.date == lastCurrentDate && data.state == state
    select data;


    foreach (var item in deathQuery)
    {
        return item.deaths;
    }

    return 0;
}
```

Ilustración 109 Obtención del número de muertes del día anterior

Luego en la clase stateInfo se realiza una resta del número de casos actual con el de día anterior, y lo mismo con las muertes. Una vez realizado ello, se muestra por pantalla. Hay algunos casos de error en los datos de entrada en el fichero CSV, donde el día actual tiene menos casos o muertes que el día anterior, entonces al realizar la resta, y para rematar, el tipo de variable es *uint*, entonces el número que se mostraría por pantalla sería "4294967296", representando el valor máximo de uint (2^{32}). Para evitar esto, se crea una condición para cuando ocurra estos casos, que muestren un "0" por pantalla.

```
public void showInfo()
{
    if (panel != null)
    {
        Animator animator = panel.GetComponent<Animator>();

        if (animator != null)
        {
            animator.SetBool("Show", true);
        }
    }
}
```

Ilustración 110 Código para mostrar la tabla de información

```
public void hideInfo()
{
    if (panel != null)
    {
        Animator animator = panel.GetComponent<Animator>();

        if (animator != null)
        {
            animator.SetBool("Show", false);
        }
    }
}
```

Ilustración 111 Código para ocultar la tabla de información

Por últimos, estos dos métodos, activan o desactivan la animación del panel de información.

Este es el resultado obtenido:

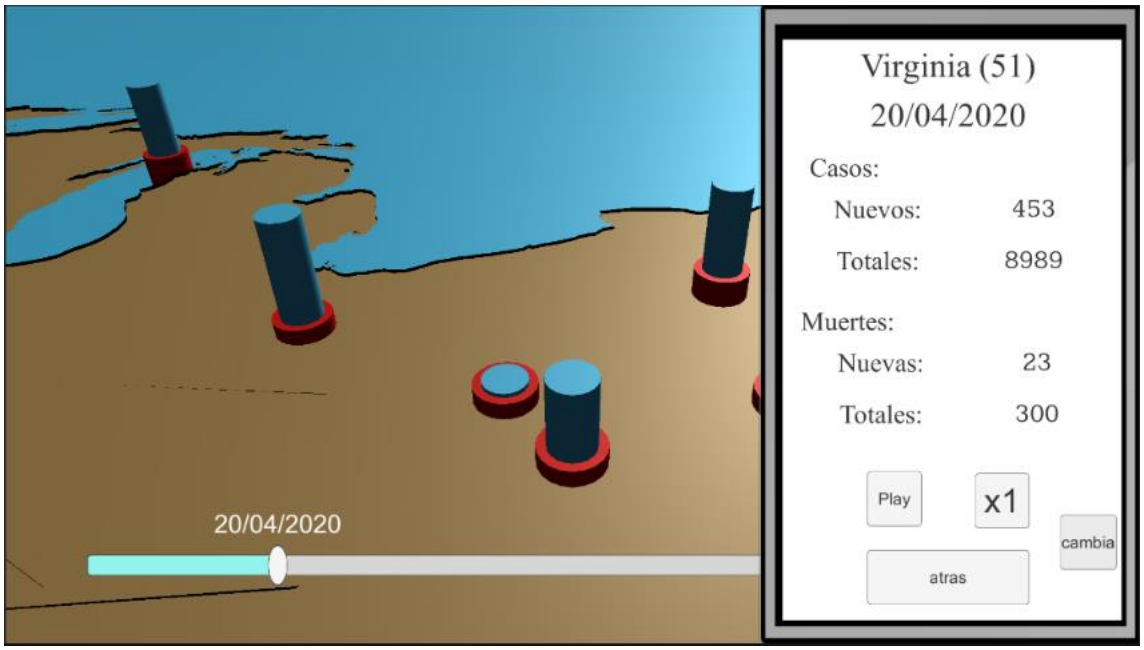


Ilustración 112 Demostación de tabla de información por pantalla

Aquí hay dos problemas de visión:

- No se sabe qué estado ha sido seleccionado
- La tabla de información no es agradable para la vista de usuario, hay que detallar estos elementos de botones e interfaz.

Para el primer problema, si el estado ha sido seleccionado, el objeto que obtenemos en la clase *ZoomCamera* modificará el color del objeto a un marrón más claro, indicando que el estado ha sido seleccionado.

```
state.GetComponent<MeshRenderer>().material.color = new Color32(0xf5, 0xde, 0xb3, 255);
```

Ilustración 113 Cambio de color cuando el estado es seleccionado

Para el segundo problema, se asigna colores correspondientes a los textos según el color de su cilindro (azul para casos, rojo para muertes), la fuente de los textos se cambia a *Times New Roman*, para los números se usan *Kameron-Regular* y añadimos botones adecuados para una mejor visión.

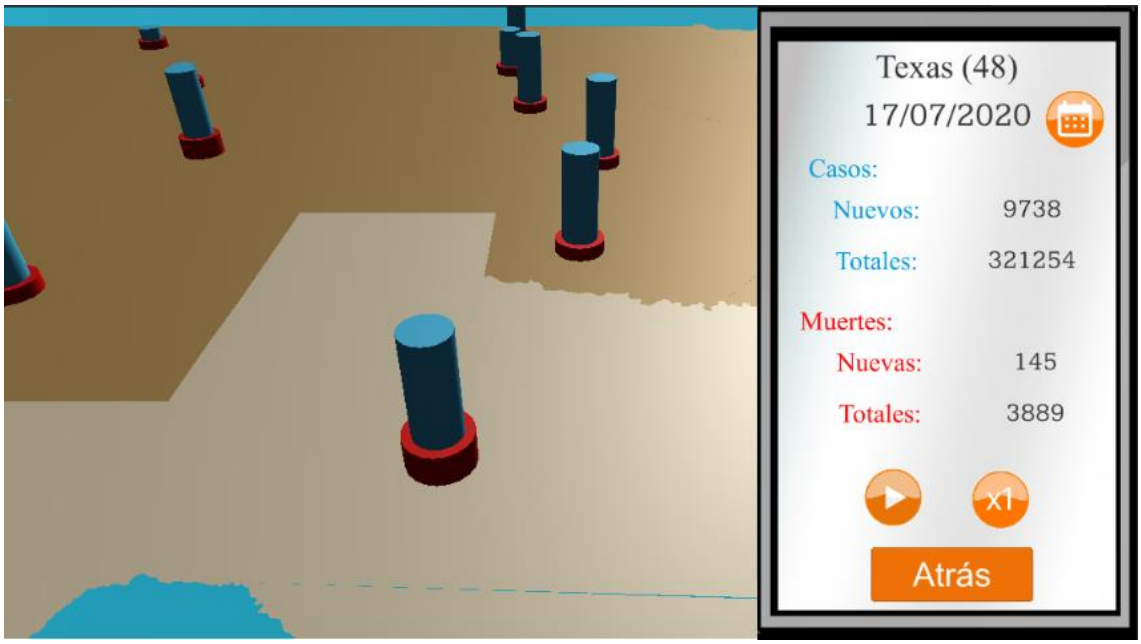


Ilustración 114 Estado con un color más ligero, para diferenciarse de los otros

8.7 Selector de fechas

El botón de calendario, lo único que hace es cambiar de fecha a uno específico, pero el usuario no puede seleccionar otra fecha cualquiera. Se implementará un selector de fechas para elegir cualquier día, mes o año, dentro del rango de las fechas de datos del CSV. Cuando se haga clic al botón del calendario, se creará un objeto donde se mostrará una ventana con los siguientes elementos, mostrados por imagen:

```
public void openDatePicker()
{
    dp = Instantiate(datePicker);
}
```

Ilustración 115 Método para crear el selector de fechas.

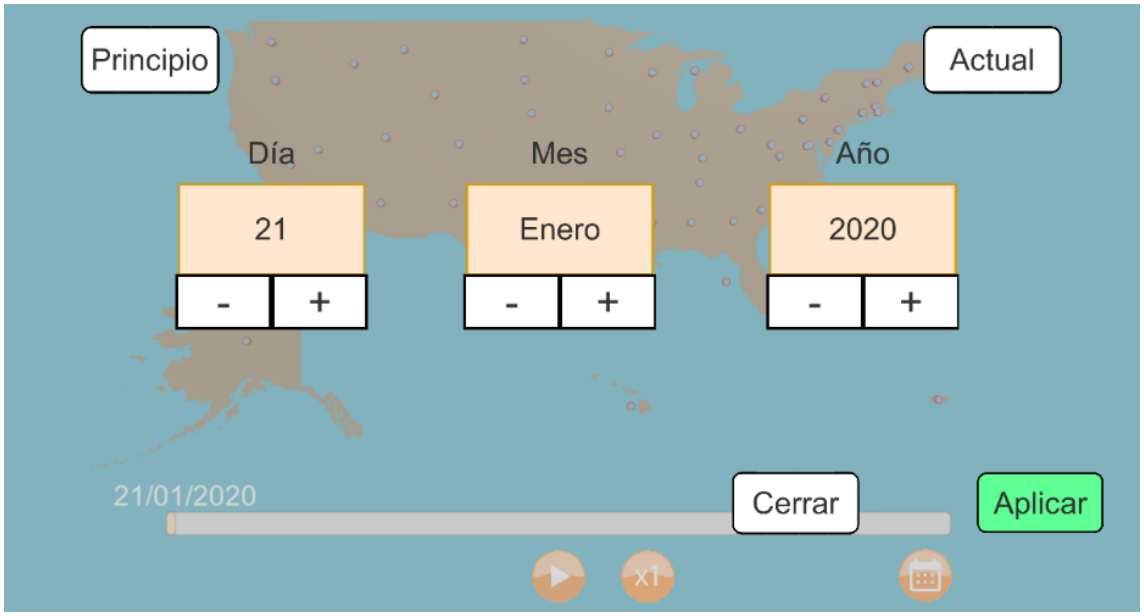


Ilustración 116 Interfaz del selector de fechas

Principio – Te lleva a la fecha inicial del archivo CSV.
Actual – Te lleva al último día del archivo CSV.
Cerrar – Cierra la ventana del selector de fechas.
Aplicar – Aplica los cambios realizados dentro del selector.

La clase *CalendarController* es el que llevará a cabo la acción de estos eventos.

```
void Start()
{
    dt = MapController.mInstance.getDate();
    firstDate = MapController.mInstance.firstDate;
    lastDate = MapController.mInstance.lastDate;
    updateDate();
}
```

Ilustración 117 Inicialización de variables en la clase CalendarController

La variable *dt* obtiene la fecha en la que se encuentra el programa en ese momento, firstDate y lastDate son la primera y última fecha que contiene el archivo CSV.

```
public void updateDate()
{
    dayText.text=""+dt.Day;
    dayMonth.text=""+monthName(dt.Month);
    dayYear.text=""+dt.Year;
}
```

Ilustración 118 Código para actualizar la fecha por pantalla

updateDate() actualiza el texto que muestra por pantallas el día, mes y año. *monthName()* es un método que devuelve el nombre del mes en español.

```
public void addDays()
{
    dt=dt.AddDays(1);
    if(dt >= lastDate)
    {
        dt=lastDate;
    }

    updateDate();
}

0 references
public void substractDays()
{
    dt=dt.AddDays(-1);
    if(dt <= firstDate)
    {
        dt=firstDate;
    }

    updateDate();
}
```

Ilustración 119 Método para pasar al día siguiente o anterior

addDays() y *substractDays()* incrementa y decrementan el día de la fecha la llamada a dt.AddDays() ayuda a que la fecha, si llega al último día del mes, pueda pasar al siguiente sin tener que comprobar estos casos manualmente. Una vez hecho todo eso, actualiza la pantalla de fecha el cambio realizado. Por otra parte, hay cuatro métodos más llamados *addMonths(), substractMonths(), addYears() y substractYears(),* que realizan la misma función que el método addDays() y substractDays(), pero ejecutando AddMonths(1/-1) y AddYears(1/-1) en sus respectivos casos.

```
public void setDate()
{
    MapController.mInstance.setDate(dt);
    DatePickerController.mInstance.closeDatePicker();
}
```

Ilustración 120 Código para aplicar cambios y cerrar ventana

Si le damos al botón aplicar, se realiza los cambios de fecha y se cierra la ventana destruyendo el objeto. Para el caso de cerrar, simplemente cierra la ventana sin realizar ningún cambio.

```
public void closeDatePicker()
{
    Destroy(dp);
}
```

Ilustración 121 Código para cerrar ventana

Hecho esto, se hizo una última reunión con el profesor de coordinación, y se ha hablado del tema de la interfaz del panel, tal como está, informa al usuario, pero sigue sin agradar a la vista, y como propuesta, en vez de usar un panel totalmente cuadrado, que los bordes sean circulares, y sustituir el botón con el texto "atrás" por algún símbolo que lo represente. También, que los textos de casos y muertes serán insertados en una tabla para mejor visualización. Dicho esto, se realizan todos estos cambios necesarios y queda la interfaz y el proyecto de la siguiente forma:

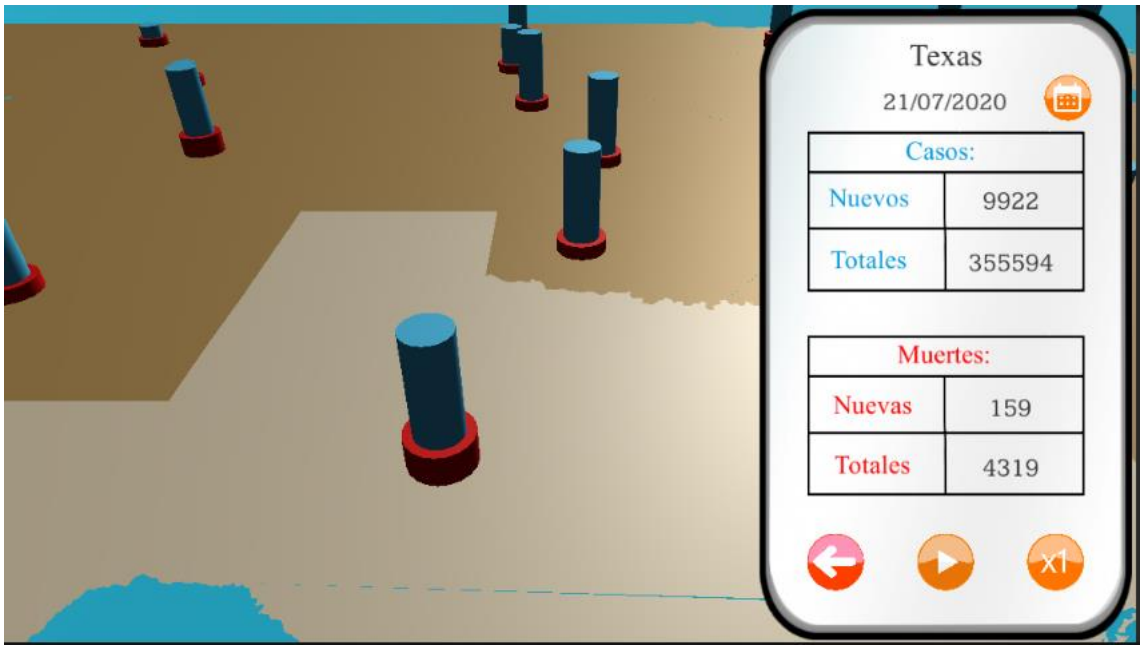


Ilustración 122 Resultado final del mapa con su tabla de información

Por último, para mostrar más información acerca de los casos y muertes, en el mapa se ha añadido el número total de casos y muertes en todo el país. Para ello, se crea una variable llamada *total* en las clases *caseSystem* y *DeathSystem*, y en el bucle for donde llama a los estados, realiza la suma de cada estado y se obtiene su total.

```
if (isDateChanged != MapController.mInstance.getDate())
{
    total=0;
    Entities.ForEach((ref DeathComponent cylinderComponent) =>
    {
        cylinderComponent.deaths = MapController.mInstance.getDeaths(states.getStateName(cylinderComponent.index));
        total+=cylinderComponent.deaths;
    });
    MapController.mInstance.setTotalDeaths(total);
    isDateChanged = MapController.mInstance.getDate();
}
```

Ilustración 123 Código para realizar la suma total

```
public void setTotalCases(uint cases)
{
    totalCases.text = "" + cases;
}

2 references
public void setTotalDeaths(uint deaths)
{
    totalDeaths.text = "" + deaths;
}
```

Ilustración 124 Código para mostrar el total por pantalla

Obtenido estos datos, se muestra por pantalla, ubicándolo en la esquina arriba derecha estos números.

Ilustración 125 Muestra de datos totales por pantalla

8.8 Descarga de archivo de datos

Ya realizado el programa entero se pasa a la última implementación importante: la descarga e inserción de datos.
Usando cURL, podemos descargar el archivo deseado de cualquier página realizando la petición a dicha página. Este comando se puede usar tanto desde Windows como Linux o MacOS, y, dependiendo de su terminal, se llama a este comando desde una forma u otra.
En la clase principal, *CSVData*, antes de inicializar todo, se realiza la implementación de estos comandos por código.

```
string console = "";
string command = "";
if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    console = "C:/WINDOWS/system32/cmd.exe";
    command = @"/c start cd " + Application.streamingAssetsPath +
    " && curl -kLSs https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv -o us-states.csv";
}

System.Diagnostics.Process.Start(console, command).WaitForExit();
```

Ilustración 126 Código para la llamada a la línea de comandos con su argumento

Se comprueba en qué sistema operativo se encuentra el programa en ese momento. En mi caso, uso Windows 10 como sistema operativo, entonces obtengo la ruta de ejecución de la terminal de comandos *"cmd.exe"* con su argumento y se ejecuta.

/c start – da el comienzo al comando de ejecución.
cd + Application.streamingAssetsPath – cambia el destino hacia donde se encuentre el proyecto en ese momento.
&& - es un operador AND para ejecutar otro comando.
Curl -kLSs https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv -o us-states.csv – Este es el comando importante, básicamente realiza la descarga del archivo de GitHub hacia la ruta donde se encuentre la terminal en ese momento y lo guarda con su formato específico.

- -k Permite conectarse a servidores inseguros cuando se usa SSL.
- -L Sigue la redirección.
- -S Muestra el error, aunque se use -s
- -s No muestra ningún mensaje durante la descarga.

Hecho toda su configuración, se llama a System.Diagnostics.Process.Start().WaitForExit() para la ejecución del comando con sus argumentos.
Cuando se ejecuta el programa, se abre el terminal primero y realiza sus comandos automáticamente, después de esto, realiza el resto de sus acciones para el comienzo del programa.

8.9 Construcción del proyecto

Desde Unity, vamos a File →Build settings…

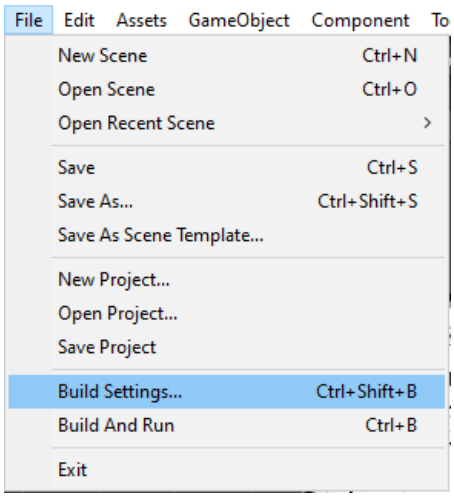


Ilustración 127 Ruta para llegar a "Build Settings..."

Una vez dentro, se selecciona las escenas a exportar y la plataforma será para *PC,Mac & Linux Standalone*, se espera el tiempo necesario hasta que Unity haya hecho sus cambios.

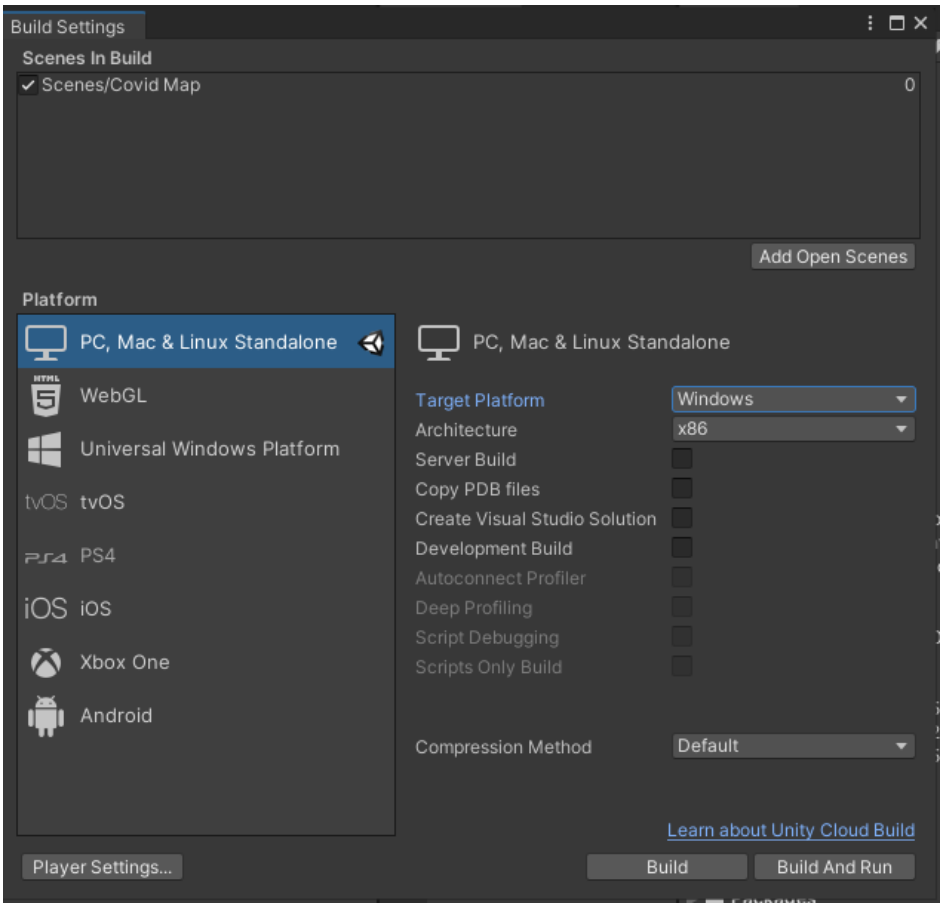


Ilustración 128 Selección de plataforma

Por último, elegimos la opción de “Build and Run”, y elegimos el destino a donde irá el programa. Hecho esto, se espera hasta que Unity haya exportado el archivo en su

totalidad, y ejecutará el programa en funcionamiento una vez terminado. Ahora el programa se puede ejecutar desde otro ordenador.

Capítulo 9 Conclusiones

Podemos concluir que, después de obtener los resultados del desarrollo, se ha logrado implementar el objetivo principal de este proyecto, en el cual cumple con todas sus funcionalidades necesarias para la visualización interactiva y agradable de todos los datos contenidos en el *dataset* de partida.

El programa descarga el archivo CSV desde GitHub usando *cURL*, siendo llamado desde su script. Luego genera un mapa creando sus estados, y colocando algunos en un lugar donde el usuario final pueda apreciar todos los estados. Aplicando ECS, se crea los objetos y se visualizan en el entorno del programa, y con la ayuda de este, los sistemas que usen objetos con mismas características darán menos carga a la CPU, y, por ende, el tiempo de espera de la aplicación será menor.

La muestra de datos que enseña el programa es detallada, desde el número de casos y muertes totales del país entero, hasta el número de casos y muertes actuales y nuevas en una región concreta, en un día especifico. Puede elegir también el día, mes y año deseado estando dentro del rango en el que se encuentre las fechas de estos datos del archivo descargado desde GitHub.

El usuario final, quien interactuará con la aplicación, no notará ralentizaciones en ella y lo verá fluido en su ejecución pese a todos los objetos visibles que tenga el programa, demostrando que es robusto y eficaz.

9.1 Futuras mejoras

Para terminar, Algunas mejoras para implementar al programa o funciones extra serían los siguientes:

- Añadir bordes negros a los estados generados para apreciar las fronteras que hay entre ellas.
- Exportarlo como página web usando WebGL para mejor acceso a la aplicación, así cualquiera podría visualizar los datos sin necesidad de descarga.
- Utilizar comandos de ejecución dependiendo del sistema operativo en el que se encuentre el usuario.
- Aplicar shaders y texturas para una mejor visión de los objetos creados en el programa.

Este proyecto pretende tener continuidad en el tiempo y de que sea puesto a disposición del gran público, puesto que tiene el potencial para ser una buena aplicación.

Capítulo 10 Manual del usuario

Para el usuario final, se dejará un manual de instrucciones para conocer el uso de este programa.

Primero, abrimos el archivo ejecutable que se llama *TFG.exe* dentro de la carpeta *Mapa COVID.*

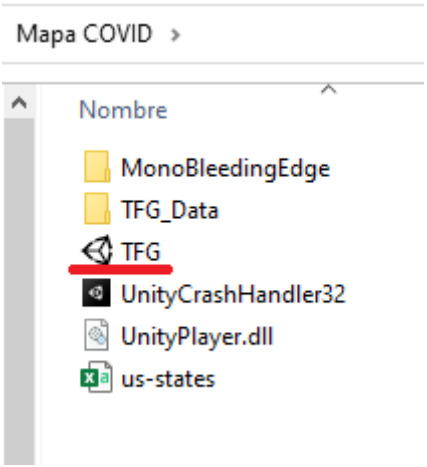


Ilustración 129 Archivo llamado "TFG.exe" que abrirá el programa.

Una vez abierto, aparecerá la pantalla con el logo de Unity y justo al desaparecer esa animación, se abre la *consola de comandos cmd* para realizar su ejecución de descarga. Puede cerrar esa consola cuando quiera, porque ya habrá realizado su descarga.

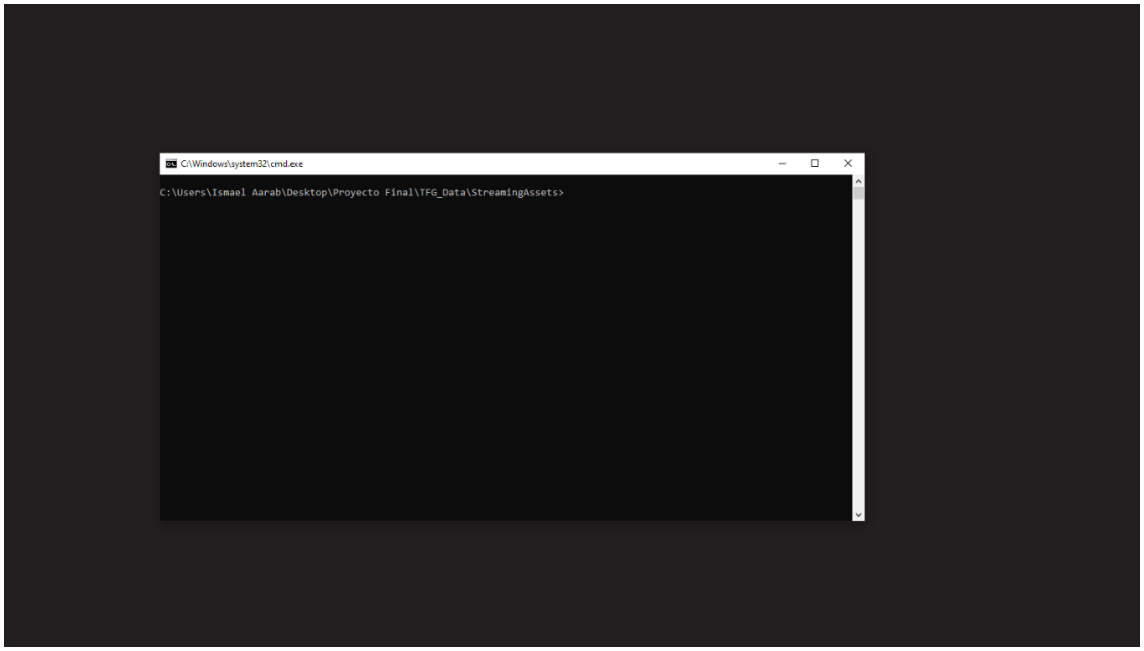


Ilustración 130 Consola de comandos realizando la descarga del archivo.

Una vez el programa ha sido compilado y ejecutado, se mostrará la siguiente pantalla:

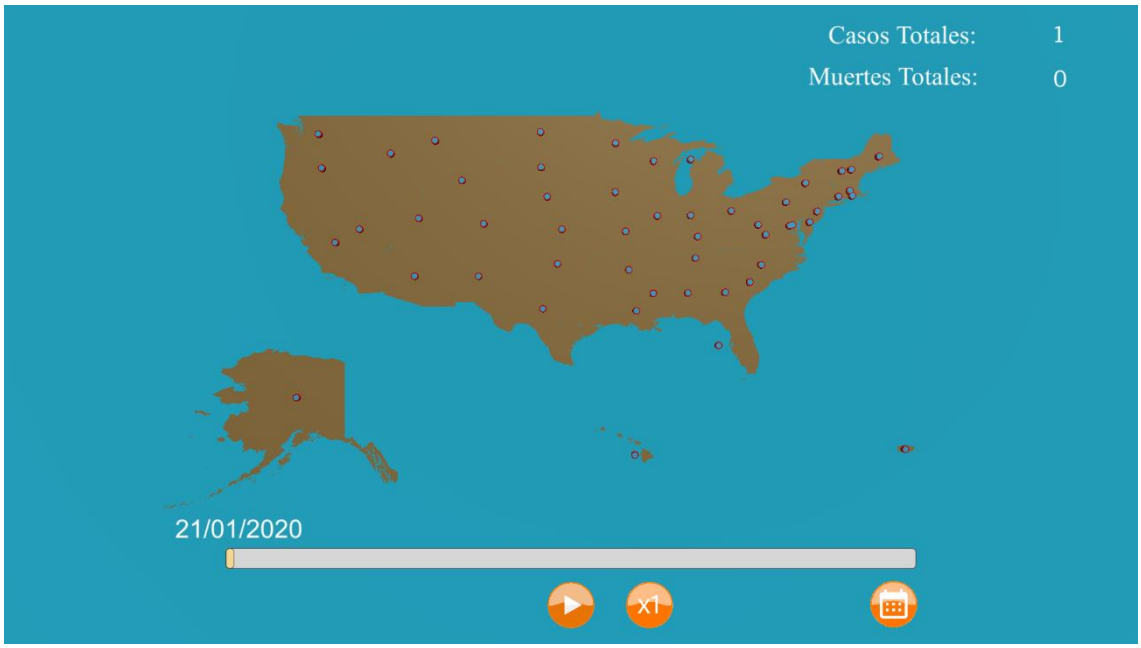


Ilustración 131 Ejecución del programa.

Enumeramos el funcionamiento de cada sección:

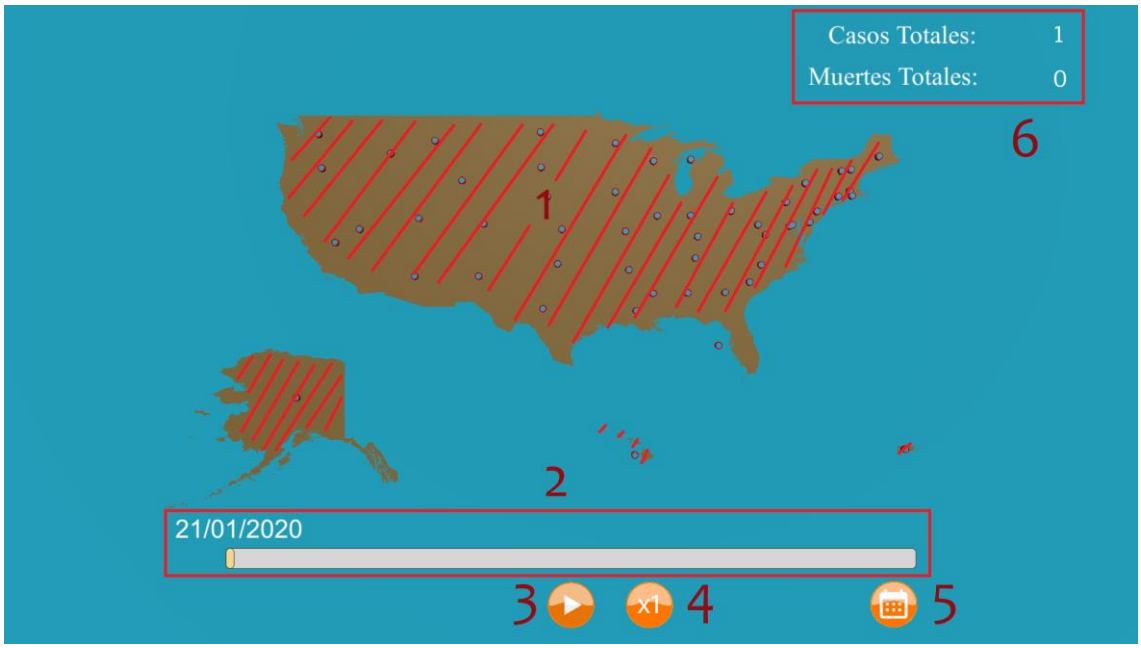


Ilustración 132 Enumeración de los funcionamientos del programa.

1- El área recubierta de líneas rojas es la zona donde se puede hacer clic, si es pulsado, la cámara cambiará su visión y se mostrará un panel de información sobre ese estado.

2- Se muestra la barra de progreso al usuario para saber la fecha y en qué momento se encuentra dicha data en su progreso de muestra de datos.

3- El botón inicio  indica que puede iniciar la reproducción de la barra de progreso para que este pueda avanzar al día siguiente. Si lo pulsa, este botón es sustituido por pausa  , pudiendo pausar la reproducción en cualquier momento.

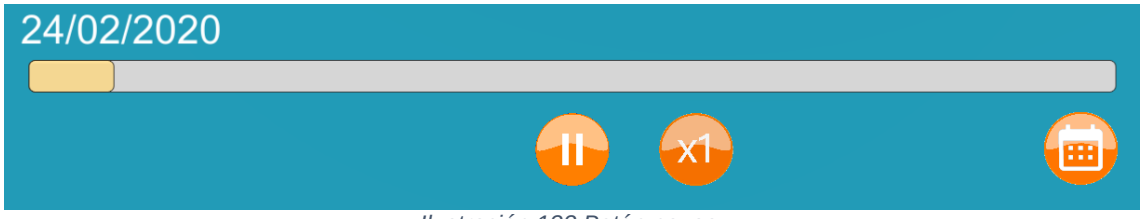


Ilustración 133 Botón pausa.

Si la barra de progreso llega al final de su reproducción, aparecerá el botón de volver al principio 

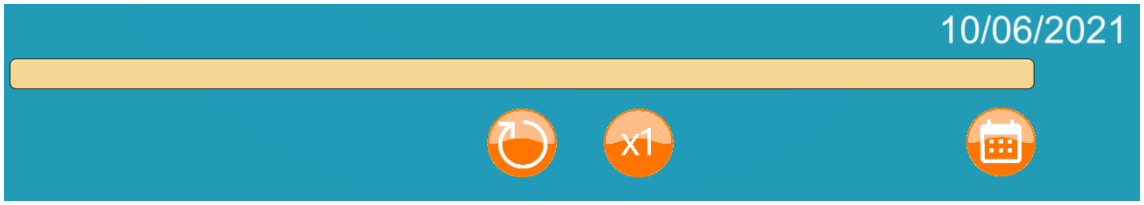


Ilustración 134 Botón volver al principio.

4- El botón de cambio de velocidad  cambia el ritmo del avance de fecha. Por defecto la fecha es cambiada cada segundo, si este botón es pulsado, el cambio de fecha será cada medio segundo y la imagen del botón cambiará a  , si se vuelve a pausar, el cambio será cada tercio de segundo y la imagen del botón cambiará a  . Una vez vuelto a pulsar este botón, se cambiará a su velocidad por defecto, es decir, cada segundo, y la imagen del botón vuelve a ser  .

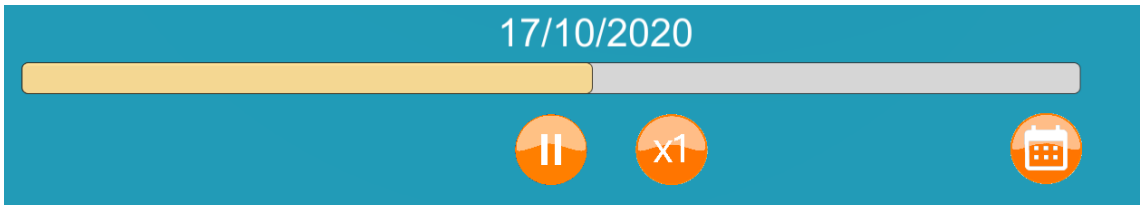


Ilustración 135 Botón x1 de velocidad.

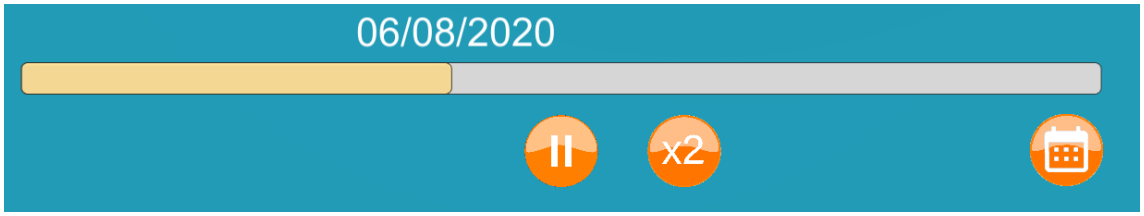


Ilustración 136 Botón x2 de velocidad.

Ilustración 137 Botón x3 de velocidad.

5- El botón de calendario  mostrará un selector de fecha por pantalla, donde el usuario puede elegir una fecha deseada dentro de un cierto rango, que serán las fechas en el que se encuentre dentro del fichero de datos.

   La fecha principal del fichero es el *21 de enero de 2020*
   La fecha final será el día actual, en este manual escrito, el día actual es el *11 de junio de 2021*.

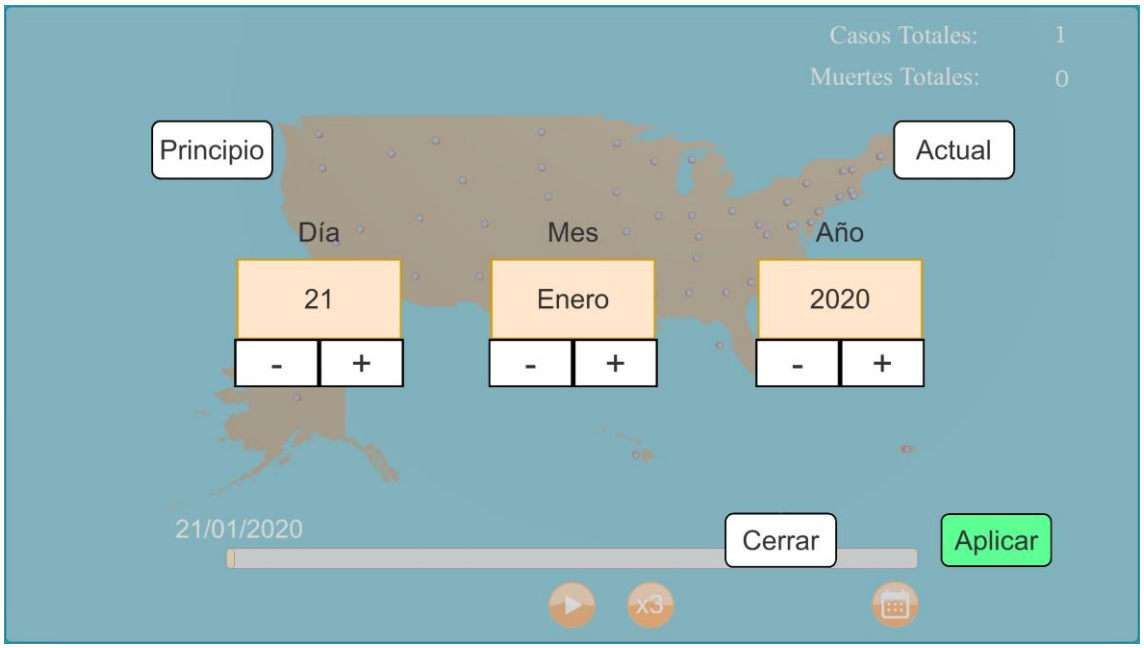


Ilustración 138 Pantalla selector de fecha.

Dentro de esta pantalla de selección de fecha, hay varios botones con sus funcionalidades:

- "+": pasa al día, mes o año siguiente.
- "-": pasa al día, mes o año anterior.
- "Principio": te lleva a la fecha inicial.
- "Actual": te lleva a la fecha final.
- "Cerrar": cierra la pantalla sin realizar ningún cambio.
- "Aplicar": cierra la pantalla guardando los cambios de fecha y aplicándolo al programa.

6- Muestra el número de casos y muertes totales en todo el país según la fecha en la que se encuentre el usuario en ese momento.

Si el usuario decide hacer clic en un estado para ver su información, se muestra lo siguiente:

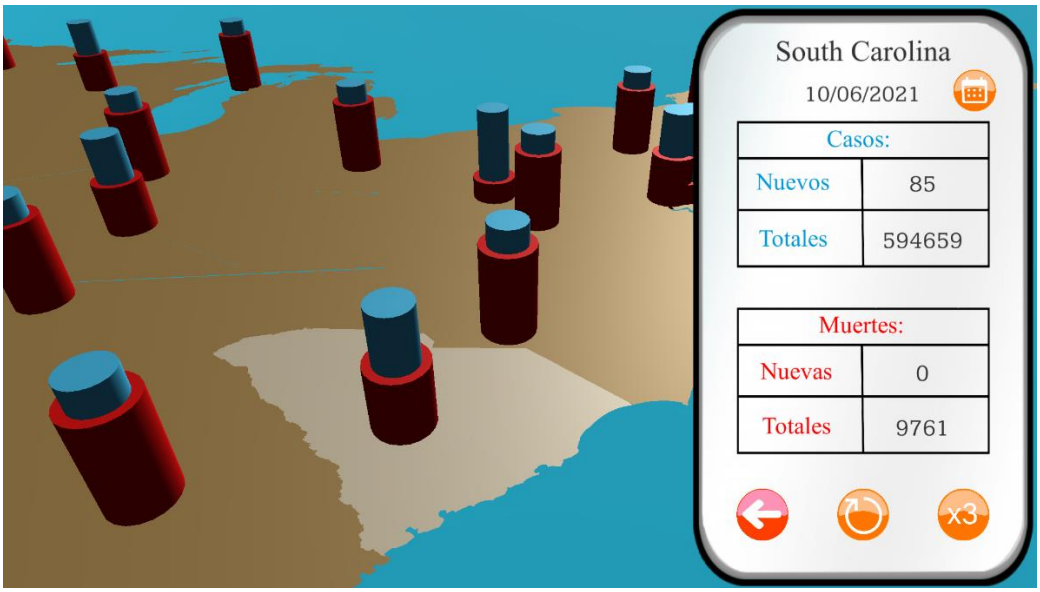


Ilustración 139 Panel de información del estado.

En ella, hay un panel de información que muestra:

- El nombre del estado
- La fecha en el que se encuentra en ese momento, donde puede cambiarla en cualquier momento con el botón .
- Una tabla que muestra los casos nuevos y totales de dicha zona.
- Una tabla que muestra las muertes nuevas y totales de dicha zona.
- El botón / / para poder avanzar o pausar la fecha en la que se encuentra, o volver al principio.
- El botón x1 / x2 / x3 para cambiar los intervalos de tiempo entre cambio de fechas.
- El botón atrás sale del panel y vuelve la cámara al estado original, mostrando el mapa entero.

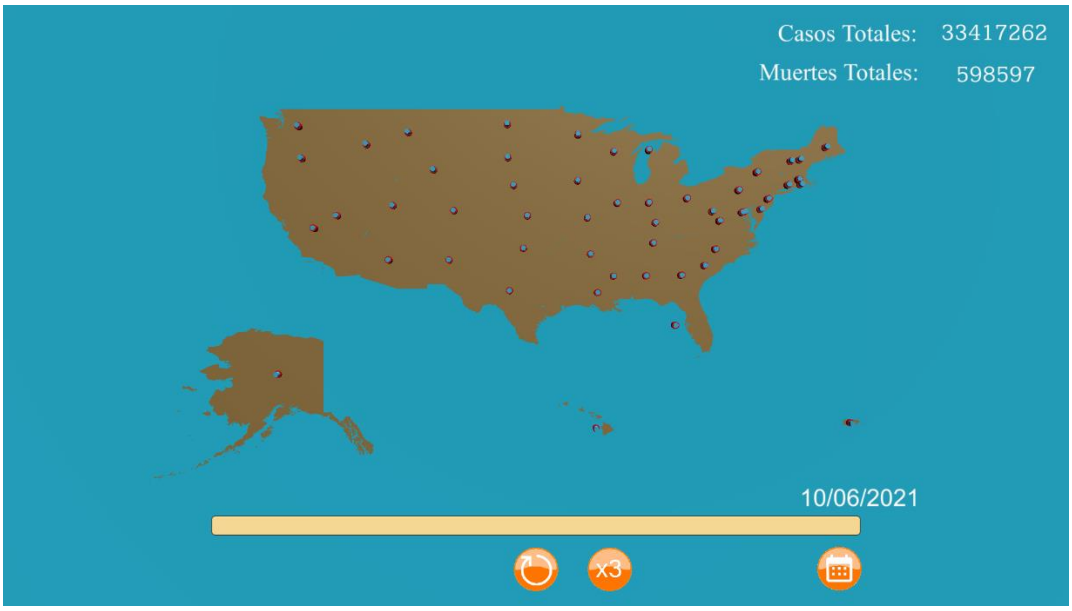


Ilustración 140 Demostración del mapa después de pulsar el botón atrás.

Por último, si desea salir del programa, tiene que pulsar la combinación de teclas *Alt+F4*, este finaliza el progreso y cierra el programa.

Bibliografía

(s.f.).

BYC. (25 de Octubre de 2017). *Parallelcube*. Obtenido de ¿Por qué necesitamos utilizar Delta Time?: https://www.parallelcube.com/es/2017/10/25/por-que-necesitamos-utilizar-delta-time/

Celeste, E. (2020). *Eric Tech*. Obtenido de GEOJSON AND KML DATA FOR THE UNITED STATES: https://eric.clst.org/tech/usgeojson/

Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU). (2020). *Johns Hopkins University of Medicine*. Obtenido de COVID-19 Dashboard: https://coronavirus.jhu.edu/map.html

Data Center Dynamics. (11 de Febrero de 2021). *DCD*. Obtenido de DE-CIX gestionó un tráfico de 32 exabytes de datos en su ecosistema de interconexión en 2020: https://www.datacenterdynamics.com/es/noticias/de-cix-gestion%C3%B3-un-tr%C3%A1fico-de-32-exabytes-de-datos-en-su-ecosistema-de-interconexi%C3%B3n-en-2020/

Google. (2020). *Google Noticias*. Obtenido de Coronavirus (COVID-19): https://news.google.com/covid19/map?hl=es

Google. (s.f.). *Google Dataset Search*. Obtenido de Google Dataset Search: https://datasetsearch.research.google.com/

IMPROBABLE. (24 de Septiembre de 2018). *IMPROBABLE*. Obtenido de Ask the Unity expert: what is the ECS?: https://www.improbable.io/blog/unity-ecs-1

Kaggle. (s.f.). *Kaggle*. Obtenido de Kaggle: https://www.kaggle.com/

Lin, W. (6 de Mayo de 2020). *raywenderlich.com*. Obtenido de Entity Component System for Unity: Getting Started: https://www.raywenderlich.com/7630142-entity-component-system-for-unity-getting-started

Max Roser, H. R.-O. (2020). *Coronavirus Pandemic (COVID-19)*. Obtenido de Our World in Data: https://ourworldindata.org/covid-cases

New York Times. (21 de Enero de 2020). *GitHub*. Obtenido de GitHub: https://github.com/nytimes/covid-19-data

Nuñez, J. M. (2021).

SAS. (s.f.). *sas.com*. Obtenido de Visualización de datos: https://www.sas.com/es_es/insights/big-data/data-visualization.html#:~:text=La%20visualizaci%C3%B3n%20de%20datos%20es,dif%C3%ADciles%20o%20identificar%20nuevos%20patrones.

ticpymes. (12 de Marzo de 2021). *ticpymes*. Obtenido de Un año de pandemia: el tráfico de Internet ha aumentado un 50% desde marzo de 2020: https://www.ticpymes.es/tecnologia/noticias/1124350049504/ano-de-pandemia-trafico-de-internet-aumentado-50-marzo-de-2020.1.html

Unify Community Wiki. (29 de Noviembre de 2017). *SimpleJSON*. Obtenido de SimpleJSON: https://wiki.unity3d.com/index.php/SimpleJSON

Unify Community Wiki. (4 de Diciembre de 2018). *Triangulator*. Obtenido de Triangulator: https://wiki.unity3d.com/index.php/Triangulator

Unity. (12 de Junio de 2020). *Unity*. Obtenido de Visibility changes for preview packages in 2020.1: https://forum.unity.com/threads/visibility-changes-for-preview-packages-in-2020-1.910880/?_ga=2.99023454.1112290768.1595164785-1691868383.1578304221%20.