UNIVERSIDAD MILITAR NUEVA GRANADA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE TECNOLOGÍAS DEL CONOCIMIENTO
COMPUTACIÓN GRÁFICA
DOCENTE GABRIEL ÁVILA

Daniela Rivera Chavarro, 6000365
Diego Alejandro Niño Calderon, 6000381

PROPUESTA DE DISEÑO

1.RESUMEN

En el documento presentado a continuación, se mostrará la idea de proyecto que se quiere llevar a cabo, a partir de la explicación y el análisis (ventajas y desventajas) del formato de texto JSON y sus derivadas, incluyendo la comprensión de los grafos y cómo, a partir de ellos, se puede realizar el proyecto esperado.

Palabras clave: Grafos, formato de texto, formato de lenguaje.

2.INTRODUCCIÓN.

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Es uno de los más potentes e importantes lenguajes de programación en la actualidad, por tres enfoques claros: es útil, práctico y está disponible en cualquier navegador web.

En JavaScript, un objeto es un entidad independiente con propiedades y tipos. Compáralo con una taza, por ejemplo. Una taza es un objeto con propiedades. Una taza tiene un color, un diseño, tiene peso y un material con la que fue hecha, etc. De la misma manera, los objetos de JavaScript pueden tener propiedades, las cuales definen sus características.

Aquí podemos entrar a hablar sobre la notación de objeto de JavaScript, JSON, el cual, es un formato de texto sencillo para el intercambio de datos. Este formato tiene diferentes ventajas y desventajas respecto a otros, dándose a conocer como uno de los formatos más sencillo de trabajar y comprender.

Por ello, en el siguiente texto, se hablará sobre las características de éste tipo de formato y de cómo, a partir de él, se puede construir la base de un proyecto y el proyecto en sí.

3.MARCO TEÓRICO.

JSON

A principios de la década de los 90 surgió el problema de que las máquinas pudieran entenderse entre sí. Entonces utilizaban diferentes sistemas operativos y sus programas estaban escritos en diferentes lenguajes de programación. Una de las soluciones fue crear el estándar XML**.**

Sin embargo, XML presentaba problemas sobre todo cuando se trataba de trabajar con gran volumen de datos, puesto que el procesamiento se volvía lento. Surgieron entonces intentos para definir formatos que fueran más ligeros y rápidos para el intercambio de información. Uno de ellos fue JSON, promovido y popularizado a principios de los 2000 por Douglas Crockford, un programador conocido como el 'gurú' de JavaScript.

JSON es el acrónimo para JavaScript Object Notation, aunque su nombre lo diga no está necesariamente ligado a JavaScript; Este es un estándar creado para el intercambio de información. Es más usado en sistemas que requieren mostrar y enviar información.

Una de sus mayores ventajas es que JSON, al ser un formato que es independiente de cualquier lenguaje de programación, sus servicios comparten información por éste método, no necesitan hablar el mismo idioma.

JSON puede representar cuatro tipos primitivos (cadenas, números, booleanos, valores nulos) y dos tipos estructurados: objetos y arreglos).

La mayor utilidad de JSON, es que puede ser usado para el intercambio de información entre lenguajes lo que cual lo transforma en el puente entre varias de las tecnologías existentes en la actualidad.

Ventajas de JSON:

•Es autodescriptivo y fácil de entender.

•Su sencillez le ha permitido posicionarse como alternativa a XML.

•Es más rápido en cualquier navegador.

•Es más fácil de leer que XML.

•Es más ligero (bytes) en las transmisiones.

•Se parsea más rápido.

•Velocidad de procesamiento alta.

Características principales:

•JSON soporta dos tipos de estructuras, una de ellas son objetos que contienen una colección de pares clave-valor y el otro tipo se trata de arrays de valores. Esto proporciona una gran sencillez en las estructuras.

•JSON no tiene espacios de nombres, cada objeto es un conjunto de claves independientes de cualquier otro objeto.

•JSON no necesita ser extensible por que es flexible por sí solo. Puede representar cualquier estructura de datos pudiendo añadir nuevos campos con total facilidad.

•JSON es mucho más simple que XML, el cual proporciona pesadas tecnologías que le avalan (Scheme, XSLT, XPath).

•JSON es optimista y no requiere de este tipo de tecnologías, confía en el desarrollador.

Un **grafo** es un tipo abstracto de datos (TAD), que consiste en un conjunto de nodos (también llamados vértices) y un conjunto de arcos (aristas) que establecen relaciones entre los nodos.

En los grafos, como en todas las estructuras de datos, las dos operaciones básicas son insertar y borrar. En este caso, cada una de ellas se desdobla en dos, para insertar/eliminar vértices e insertar/eliminar aristas.

Ya que JSON es un estándar creado para el intercambio de información, se puede relacionar con la utilidad de los grafos, ya que a partir de ellos se puede realizar el intercambio de información que nos permite hacer JSON.

En base a ésta idea, podemos realizar variedad de programas o aplicaciones y podemos representar varios componentes, debido a que, los campos de utilización de los grafos son muy variados, ya que los vértices pueden representar cualquier elemento (ciudades, aeropuertos, etc...), y las aristas serían la conexión entre esos elementos (carreteras, pasillos

aéreos, redes...). Por lo tanto, los grafos son muy usados en la modelización de sistemas reales.

PROPUESTA DE PROYECTO

Con la información obtenida y mediante el documento presentado, se quiere realizar un juego de memoria, en donde la información recolectada del usuario será transportada por medio de la construcción de los grafos. Cada grafo figura un estado, y cada interacción del usuario representa el cambio de un grafo a otro, mediante las aristas, lo cuales llevan la información que el usuario (jugador) ingresa. También se utilizará JSON para optimizar la transferencia de información entre los grafos y mejorar el rendimiento del programa.

Inicialmente, se quiere lograr crear, mediante los grafos, unas piezas, estas representaran las fichas básicas de un juego de memoria. Serán en total 12 fichas, 6 de ellas tendrán su respectiva pareja. A estas se les quiere implementar imágenes de un juego llamado brawl stars, las imágenes serán importadas hacia el código desde la web.

En la imagen se puede apreciar las fichas, estas fueron creadas con nodos, a los cuales se les dio una figura, en este caso, un circulo Se trasladaron y se posicionaron de tal manera que creara una figura circular. Más adelante se sabrá porqué.

```
20
21          //NODOS
22          var nodes = [
23                  { id: 1, value: 1, x:0, y:-9, z:10, label: "bibi" },
24                  { id: 2, value: 1, x:0, y:12, z:-5.5, label: "bibi" },
25                  { id: 3, value: 2, x:12, y:1, z:2, label: "nita" },
26                  { id: 4, value: 2, x:-10, y:7, z:-2, label: "nita" },
27                  { id: 5, value: 3, x:6, y:10.5, z:-4.5, label: "poco" },
28                  { id: 6, value: 3, x:6, y:-8, z:9, label: "poco" },
29                  { id: 7, value: 4, x:-6, y:-8, z:9, label: "primo" },
30                  { id: 8, value: 4, x:-6, y:10.5, z:-4.5, label: "primo" },
31                  { id: 9, value: 5, x:-12, y:1, z:2, label: "crow" },
32                  { id: 10, value: 5, x:10, y:7, z:-2, label: "crow" },
33                  { id: 11, value: 6,x:10, y:-4, z:6, label: "mortis" },
34                  { id: 12, value: 6, x:-10, y:-4, z:6, label: "mortis" },
35              ];
36
```

Imagen 1. Creación de nodos, a los que se les da un id, un valor, las posiciones, y un nombre.

```
198
199         //CREANDO LOS NODOS
200         var sphereGeometry2 = new THREE.SphereGeometry( nodes[1].value, 10, 10 );
201
202         b1.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[0].x, nodes[0].y, nodes[0].z));
203         b2.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[1].x, nodes[1].y, nodes[1].z));
204         b3.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[2].x, nodes[2].y, nodes[2].z));
205         b4.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[3].x, nodes[3].y, nodes[3].z));
206         b5.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[4].x, nodes[4].y, nodes[4].z));
207         b6.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[5].x, nodes[5].y, nodes[5].z));
208         b7.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[6].x, nodes[6].y, nodes[6].z));
209         b8.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[7].x, nodes[7].y, nodes[7].z));
210         b9.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[8].x, nodes[8].y, nodes[8].z));
211         b10.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[9].x, nodes[9].y, nodes[9].z));
212         b11.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[10].x, nodes[10].y, nodes[10].z));
213         b12.applyMatrix(new THREE.Matrix4().makeTranslation(nodes[11].x, nodes[11].y, nodes[11].z));
214
215
```

Imagen 2. Realización de nodos. Acá se crean los nodos, se les da matriz y se trasladan.

```
var linematerial = new THREE.LineBasicMaterial({ color: 0x0000ff, transparent: true, opacity: 0.02 });
var texture = new THREE.TextureLoader();
var texture2 = new THREE.TextureLoader();
var texture3 = new THREE.TextureLoader();
var texture4 = new THREE.TextureLoader();
var texture5 = new THREE.TextureLoader();
var texture6 = new THREE.TextureLoader();
var brawl = texture.load('https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQJaizcdtN6IcZVe9jiQBi--5IVmjYl7eNvpSmryPEQODMdzwnf&usqp=CAU')
var brawl2 = texture2.load('https://i2.wp.com/gamerempire.net/wp-content/uploads/2019/03/Nita-Brawl-Stars.png?resize=512%2C346&ssl=1')
var brawl3 = texture3.load('https://www.meme-arsenal.com/memes/87af9852024b6296b85e9b1f4c6bfaa2.jpg')
var brawl4 = texture4.load('https://static1.millenium.gg/entity_articles/7/20/7/8/23649-16254-brawl-stars-el-primo-1280x720-orig-1-article_m-1.png')
var brawl5 = texture5.load('https://vignette.wikia.nocookie.net/brawlstarschile/images/d/d3/Crow.jpg/revision/latest/scale-to-width-down/340?cb=20190404195350&path-prefix=
var brawl6 = texture6.load('https://cdn131.picsart.com/298500255273201.jpg?type=webp&to=min&r=640')
var material1 = new THREE.MeshStandardMaterial( {color: 0x009888,roughness: 10} );
var material2 = new THREE.MeshStandardMaterial( { color: 0xffffff, metalness: 0.5, roughness: 0.1 } );
var material3 = new THREE.MeshStandardMaterial( { color: 0xffffff, side: THREE.DoubleSide, metalness: 0.5, roughness: 0.1, map: brawl } );
var material4 = new THREE.MeshStandardMaterial( { color: 0xffffff, side: THREE.DoubleSide, metalness: 0.5, roughness: 0.1, map: brawl2 } );
var material5 = new THREE.MeshStandardMaterial( { color: 0xffffff, side: THREE.DoubleSide, metalness: 0.5, roughness: 0.1, map: brawl3 } );
var material6 = new THREE.MeshStandardMaterial( { color: 0xffffff, side: THREE.DoubleSide, metalness: 0.5, roughness: 0.1, map: brawl4 } );
var material7 = new THREE.MeshStandardMaterial( { color: 0xffffff, side: THREE.DoubleSide, metalness: 0.5, roughness: 0.1, map: brawl5 } );
var material8 = new THREE.MeshStandardMaterial( { color: 0xffffff, side: THREE.DoubleSide, metalness: 0.5, roughness: 0.1, map: brawl6 } );
var material9 = new THREE.MeshStandardMaterial( { color: 0x009888, metalness: 0.5, roughness: 0.1, transparent:true, opacity: 0.5  } );
```

Imagen 3. Creación de texturas y materiales. Se crearon las texturas con un “TextureLoader” y se cargaron desde la web (Son links de imágenes sacados de internet). Se les especificó un material, para posteriormente cargarlo en “mapa”.

```
173         b1.material = material3;
174         b2.material = material3;
175         b3.material = material4;
176         b4.material = material4;
177         b5.material = material5;
178         b6.material = material5;
179         b7.material = material6;
180         b8.material = material6;
181         b9.material = material7;
182         b10.material = material7;
183         b11.material = material8;
184         b12.material = material8;
185         parent.add(b1)
186         parent.add(b2);
187         parent.add(b3)
188         parent.add(b4);
189         parent.add(b5)
190         parent.add(b6);
191         parent.add(b7)
192         parent.add(b8);
193         parent.add(b9)
194         parent.add(b10);
195         parent.add(b11)
196         parent.add(b12);
197
```

Imagen 4. Llamado de materiales para cada ficha e implementación en la escena.

En cuanto a los grafos, la idea es que cada pareja esté conectada por “links”, así pues, se tendrá en total 6 parejas. Esto nos permitirá que, por medio, del lenguaje JSON, se extraerá la información de un grafo a otro para saber y comparar, mediante condiciones, si dichos grafos son iguales o no. En caso de ser iguales, la respuesta será correcta y podrá continuar el juego, en caso de que no sea correcta, se reiniciará el juego.

```
36
37          //CONEXIONES
38          var edges = [
39                  { from: 0, to: 1, value: 3 },
40                  { from: 2, to: 3, value: 3 },
41                  { from: 4, to: 5, value: 3 },
42                  { from: 6, to: 7, value: 3 },
43                  { from: 8, to: 9, value: 3 },
44                  { from: 10, to: 11, value: 3 },
45
46              ];
47          init();
48          animate();
```

Imagen 5. Creación de conexiones/links.
"from", desde donde va, "to", hacía donde va.
Aquí se puede observar como solo hay parejas de nodos conectadas.

```
217
218
219             //CREANDO LAS ARISTAS
220             for(var i=0; i<edges.length; i++){
221                 var points = [];
222                     points.push( new THREE.Vector3( nodes[edges[i].from].x,
223                                                     nodes[edges[i].from].y,
224                                                     nodes[edges[i].from].z ) );
225                     points.push( new THREE.Vector3( nodes[edges[i].to].x,
226                                                     nodes[edges[i].to].y,
227                                                     nodes[edges[i].to].z ) );
228                 var geometry = new THREE.BufferGeometry().setFromPoints( points );
229                 var line = new THREE.Line( geometry, linematerial );
230                 parent.add(line);
231             }
232
```

Imagen 6. Realización de las conexiones. Básicamente, en este punto, además de que se les da una geometría de "línea", también se consolida su estructura y funcionamiento, para que concuerde con las conexiones establecidas en la Imagen 3.

Por otro lado, se quiere implementar una función "aleatoria", donde el programa, ya teniendo una estructura o geometría definida, pueda, mediante los links, cambiar la posición de los grafos. Para acceder a esta función, se implementará un botón en pantalla.

Por último, se quiere recrear una interfaz con CSG, que simule un tablero, el cual tendrá como función "tapar" la configuración de las fichas y su posición.

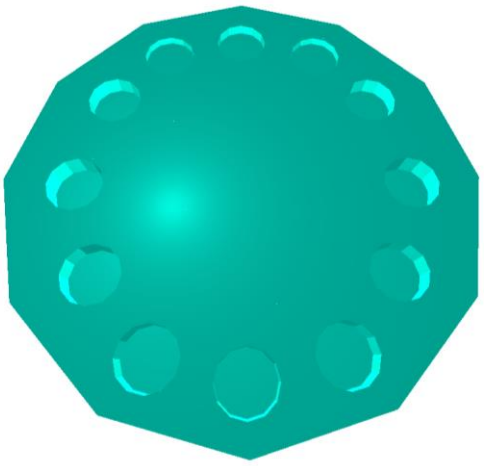

Imagen 7. Aquí ya se puede apreciar el porqué las fichas tienen esa "composición" (En forma circular), ya que el tablero tiene esa forma. Es una figura en 3D, diseñada a partir de CSG, la cual se puede manipular con el mouse.

```
96
97          //CREAR LAS GEOMETRÍAS
98          var figuras = new THREE.CircleGeometry( 2, 10);
99          var cubo = new THREE.CircleGeometry(5,32);
100         var Cilindro = new THREE.CylinderGeometry(17,17,6,12 );
101         var Hueco = new THREE.CylinderGeometry(2,2,2,12 );
102         var Hueco1 = Hueco.clone();
103         var Hueco2 = Hueco.clone();
104         var Hueco3 = Hueco.clone();
105         var Hueco4 = Hueco.clone();
106         var Hueco5 = Hueco.clone();
107         var Hueco6 = Hueco.clone();
108         var Hueco7 = Hueco.clone();
109         var Hueco8 = Hueco.clone();
110         var Hueco9 = Hueco.clone();
111         var Hueco10 = Hueco.clone();
112         var Hueco11 = Hueco.clone();
113
```

Imagen 8. Creación de geometrías. Como se puede apreciar, toda la geometría de la interfaz se creó a partir de cilindros. Las clonaciones hacen referencia a los diferentes “espacios” o “huecos”, implementados en la estructura base.

```
390         //CONVERTIR A CSG
391         var CinCSG = THREE.CSG.fromMesh( Cilin );
392         var HuecoCSG = THREE.CSG.fromMesh( Hueco_ );
393         var Hueco1CSG = THREE.CSG.fromMesh( Hueco_1 );
394         var Hueco2CSG = THREE.CSG.fromMesh( Hueco_2 );
395         var Hueco3CSG = THREE.CSG.fromMesh( Hueco_3 );
396         var Hueco4CSG = THREE.CSG.fromMesh( Hueco_4 );
397         var Hueco5CSG = THREE.CSG.fromMesh( Hueco_5 );
398         var Hueco6CSG = THREE.CSG.fromMesh( Hueco_6 );
399         var Hueco7CSG = THREE.CSG.fromMesh( Hueco_7 );
400         var Hueco8CSG = THREE.CSG.fromMesh( Hueco_8 );
401         var Hueco9CSG = THREE.CSG.fromMesh( Hueco_9 );
402         var Hueco10CSG = THREE.CSG.fromMesh( Hueco_10 );
403         var Hueco11CSG = THREE.CSG.fromMesh( Hueco_11);
404
405
406         //APLICAR LAS OPERACIONES
407         var result = CinCSG.subtract( HuecoCSG ).subtract( Hueco1CSG ).
408         subtract( Hueco2CSG ).subtract( Hueco3CSG ).subtract( Hueco4CSG ).
409         subtract( Hueco5CSG ).subtract( Hueco6CSG ).subtract( Hueco7CSG ).
410         subtract( Hueco8CSG ).subtract( Hueco9CSG ).subtract( Hueco10CSG ).
411         subtract( Hueco11CSG )
412
413
414         //CONVERTIR A THREE
415         Cilindro_ = THREE.CSG.toMesh( result );
416         Cilindro_.material = material1;
417
418
419         //AGREGAR A LA ESCENA LOS DIFERENTES ELEMENTOS
420         scene.add( parent );
421         scene.add( Cilindro_ );
422         //scene.add( cubo_ );
423
```

Imagen 9. Realización del CSG para terminar la estructura de la interfaz.

Cada “hueco” fue modificado, trasladado y rotado de tal manera como se ve en la figura final.

A la figura base, evidentemente se le aplicó un material, y se creó de tal manera que se viera como un decágono.

```
233
234         //TRASLACIONES Y ROTACIONES
235
236         Hueco_.translateX( -10 );
237         Hueco_.translateY( 7 );
238         Hueco_.translateZ( -2.0 );
239
240         Hueco_1.translateX( -10 );
241         Hueco_1.translateY( -4 );
242         Hueco_1.translateZ( 6 );
243
244         Hueco_2.translateX( -6 );
245         Hueco_2.translateY( 11 );
246         Hueco_2.translateZ( -4 );
247
248         Hueco_3.translateX( -0.1 );
249         Hueco_3.translateY( 12 );
250         Hueco_3.translateZ( -5 );
251
252         Hueco_4.translateX( 6 );
253         Hueco_4.translateY( 11 );
254         Hueco_4.translateZ( -4 );
255
256         Hueco_5.translateX( 10 );
257         Hueco_5.translateY( 7 );
258         Hueco_5.translateZ( -2.0 );
259
260         Hueco_6.translateX( -0.1 );
261         Hueco_6.translateY( -9 );
262         Hueco_6.translateZ(  10 );
263
```

```
291
292         Hueco_.rotation.x = 180;
293         Hueco_.rotation.y = 0;
294         Hueco_.rotation.z = 0;
295
296         Hueco_1.rotation.x = 180;
297         Hueco_1.rotation.y = 0;
298         Hueco_1.rotation.z = 0;
299
300         Hueco_2.rotation.x = 180;
301         Hueco_2.rotation.y = 0;
302         Hueco_2.rotation.z = 0;
303
304         Hueco_3.rotation.x = 180;
305         Hueco_3.rotation.y = 0;
306         Hueco_3.rotation.z = 0;
307
308         Hueco_4.rotation.x = 180;
309         Hueco_4.rotation.y = 0;
310         Hueco_4.rotation.z = 0;
311
312         Hueco_5.rotation.x = 180;
313         Hueco_5.rotation.y = 0;
314         Hueco_5.rotation.z = 0;
315
316         Hueco_6.rotation.x = 180;
317         Hueco_6.rotation.y = 0;
318         Hueco_6.rotation.z = 0;
```

Imagen 10. Traslación y rotación de los elementos.

El resultado final, implementando grafos y figura 3D, sería el siguiente.

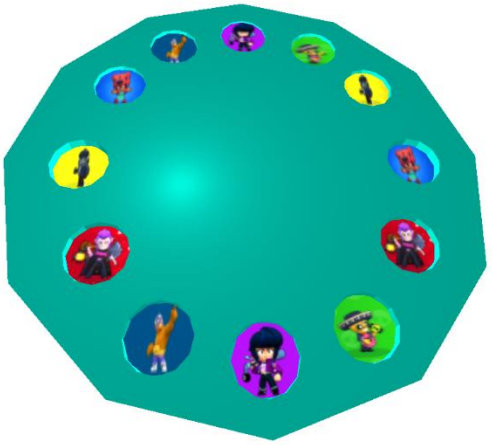

Imagen 8. Resultado final. Implementación de las estructuras y ordenamiento.

4. CONCLUSIONES.

La componente JSON trabaja, principalmente, con una colección de pares clave-valor o con un array de valores. Eso causa una mayor sencillez en su construcción. Además, la forma de utilizarse y su capacidad de ser entendido con más facilidad, vuelven a este estándar uno de los de mejores para entender mejor las estructuras del código.

A partir de los grafos, de su concepto y su estructura, se puede relacionar con la información recolectada, enviada y mostrada de la componente JSON, ya que, ésta información puede ser remitida y recibida con una estructura tipo grafo.

5. BIBLIOGRAFÍA.

- clcanela (23 de Abril, 2013). ¿Qué es JSON, para qué sirve y dónde se usa? Recuperado de https://canela.me/articulo/JSON-JavaScript-jQuery/%C2%BFQu%C3%A9-es-JSON-para-qu%C3%A9-sirve-y-d%C3%B3nde-se-usa

- Wikipedia, La enciclopedia libre (22 de Diciembre, 2019) JSON, Recuperado de https://es.wikipedia.org/wiki/JSON

- Victor Garibay Cadenas (17 de Julio, 2016) "JSON, marcando tendencias" Recuperado de https://medium.com/@victor.garibayy/qu%C3%A9-es-y-para-qu%C3%A9-sirve-json-be05fe02e67d

- Anónimo (1 de marzo, 2010) "Concepto de grafos". Recuperado de http://decsai.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/grafos.htm