INSTRUCCIONES:

- Con las teclas se podra mover la camara en las direcciones como si fuera en primera persona. Flecha izquierda, derecha (giro de la camara respecto a un punto), arriba y abajo (giro vertical de la camara respecto a un punto). Tecla + acercarse y tecla – es alejarse
- Con la tecla “F” el publico hace las olas

Todos los objetos en la escena son colocados en un **Grupo** denominado como “**ground”** definido en **main.js:**

```
34  // Ground is a group that contains all other stuff in the scene
35  var ground = new THREE.Group();
```

Este grupo contiene un plano largo con dimensiones definidas de la siguiente manera:

```
26  const ground_width = 3000;
27  const ground_height = 2000;
```

El plano es creado con esta funcion:

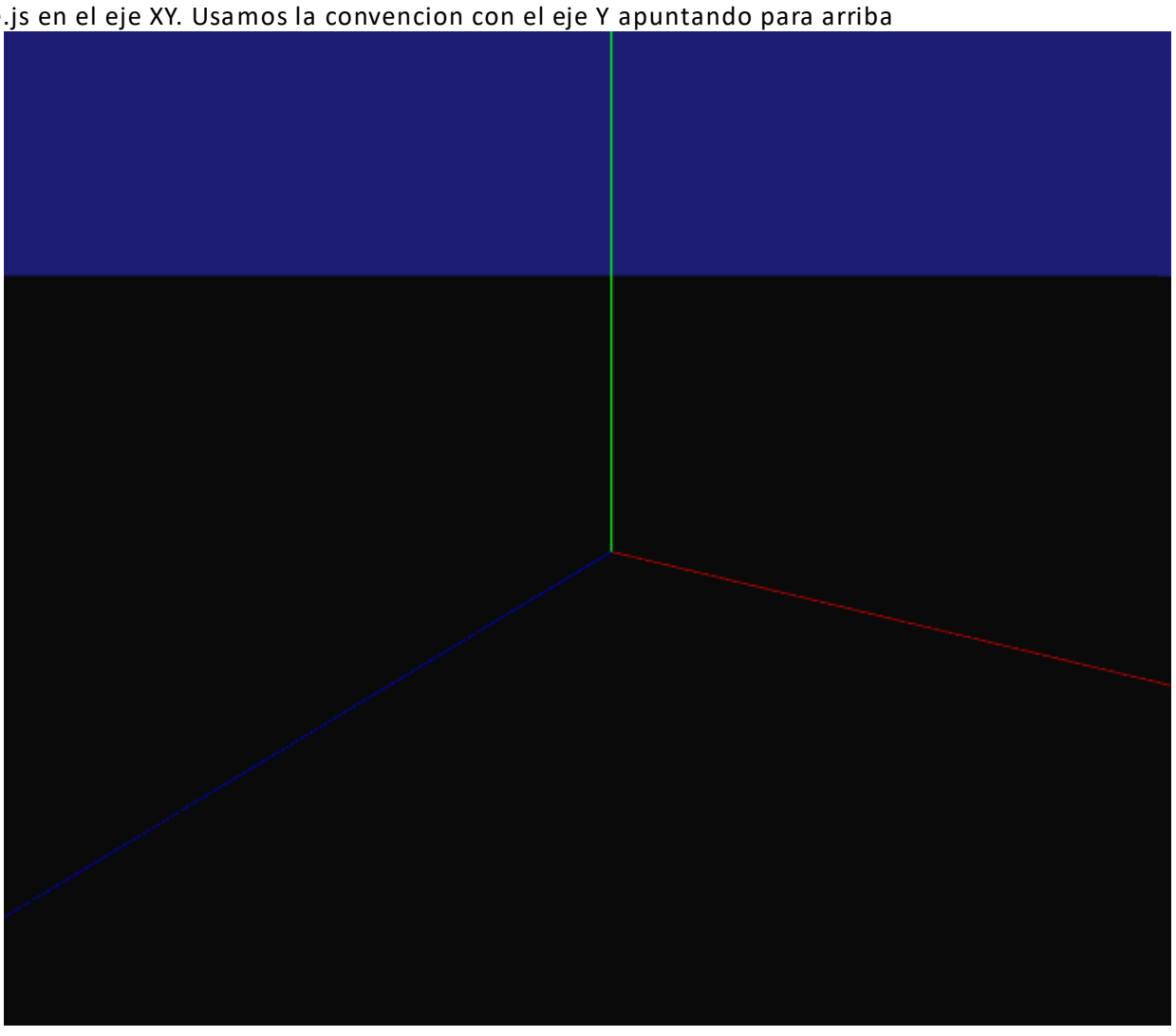
```
38  function create_ground() {
39    const geometry = new THREE.PlaneGeometry(ground_width, ground_height);
40    const material = new THREE.MeshPhongMaterial({ color: 0x202020 , side: THREE.DoubleSide});
41    const plane = new THREE.Mesh(geometry, material);
42    ground.add(plane);
43
44    // Rotate the ground -90 degrees around the X-axis, so that the positive Y-axis points upward
45    ground.rotation.x = - Math.PI / 2;
46
47    // Add the ground to the scene
48    scene.add(ground);
49  }
```

El constructor para el plano geometrico **PlaneGeometry** (llamado en la linea 39):

PlaneGeometry(width : Float, height : Float)

Pasamos a ese constructor la dimension del ground que definimos arriba.

El plano es creado en Three.js en el eje XY. Usamos la convencion con el eje Y apuntando para arriba

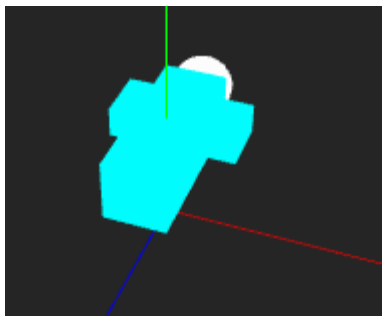


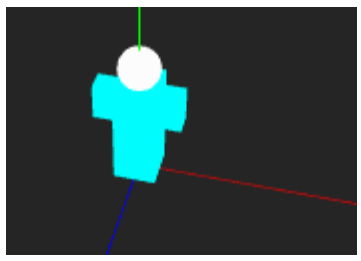
(Linea Roja: Eje X, linea verde: Eje Y, linea azul: Eje Z). Estas lineas son dibujadas al final del codigo en **main.js**, la cual fue usada para ayudar a posicionar el escenario correctamente, lo cual fue comentado luego del diseño.

Luego, para hacer que ese plano mire hacia arriba, rotamos 90 grados alrededor del eje-X (línea 45)
(Las rotaciones son especificadas en radianes)
Todo lo resto en esta escena es afectado por esta transformación, ya que todos los objetos son agregados en el grupo de **ground**.
Así que, si un objeto es creado necesitamos aplicar rotaciones para revertir el efecto de rotación.
En una instancia, en **person.js**:

```
42  // Make it stand on the ground
43  person.rotation.x = Math.PI/2;
```

Sin esa rotacion, la persona estaria acostado en el suelo:





Esta variable define cuantas franjas de pasto claras/oscuras se divide el campo:

```
31  // Number of grass stripes in the field
32  const num_stripes = 12;
```

5 franjas:

25 franjas:

La relación (ancho de campo/número de franja) se usa para ayudar a posicionar otros objetos en todo el código. Para un número distinto de 12, será necesario ajustar la posición de otros objetos en la escena.

Estos valores definen el ancho y alto del campo en **main.js**

```
28  const field_width = 200;
29  const field_height = 150;
```

En el siguiente ejemplo ponemos un valor de 800x600 unidades. El campo y los arcos se vuelven mas largos, mientras que las proporciones son mantenidas:

El campo es un **Grupo** de Threejs que contiene un plano para cada franja de césped. También en ese grupo están las marcas, que son varias líneas, un círculo y dos curvas elípticas.

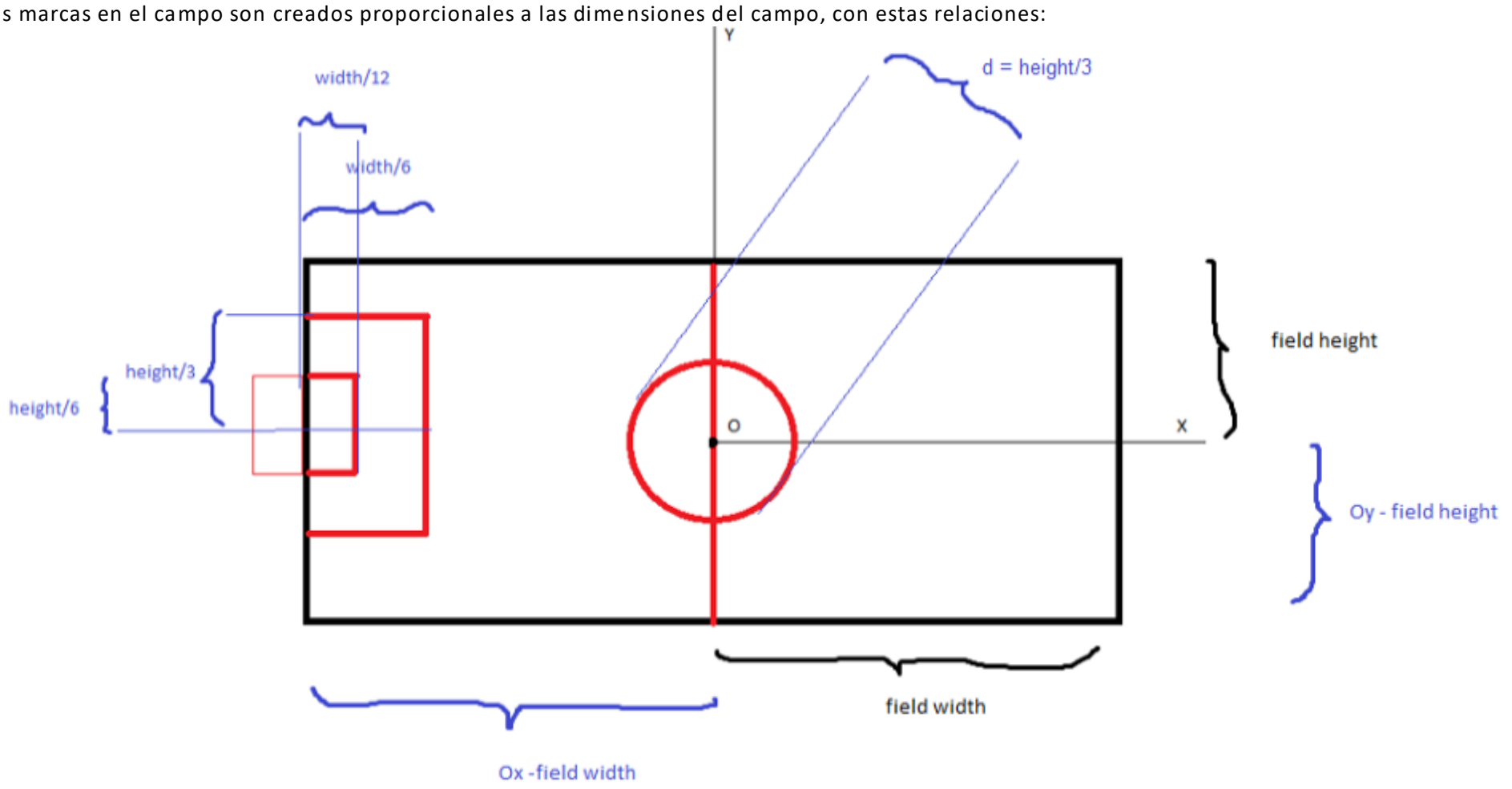
En **field.js:**

```
30    const stripe_width = field_width / num_stripes;
31    for (var i = 0; i < num_stripes; i++) {
32      const geometry = new THREE.PlaneGeometry(stripe_width, field_height);
33      const plane = new THREE.Mesh(geometry);
34
```

Esto crea un plano para cada franja, a lo largo de la altura del campo. No se aplica rotación a estos planos ya que se encuentran planos sobre el suelo.
Un material para cada uno de los tipos distintos de pasto (claro y oscuro) son aplicados a cada franja. Estos materiales usan las texturas 'grass1.jpg' y 'grass2.jpg' del directorio “textures”.
Se distinguen las franjas por par e impar para la franja clara y oscura:

```
41        if (i % 2 == 0)
```

Las marcas en el campo son creados proporcionales a las dimensiones del campo, con estas relaciones:



El campo esta exactamente centrado en el origen (0,0,0) asi que, cuando se crean las franjas, su valor en X es compensado por la mitad del ancho del campo:

```
// Start at minus half of the field's width, so that the origin is at the center;
plane.position.x = (-field_width / 2) + i * stripe_width;
```

This is drawn in the XY axis, lying on the field and plane. The same function is used for the penalty arcs, but with different parameters, making a half-circle with 180 degrees – from pi/2 to 3 x pi/2. The left arc is drawn in clockwise direction, the right one in counter-clockwise.

The field lies slightly above the ground, and the markings lie slightly above the field:

El campo tiene varias líneas de marcado. Todos los rectángulos están hechos usando líneas también.
Las marcas del lado derecho son las mismas que las del lado izquierdo pero invertidas: en vez de usar **-field_width** para el componente X, es usado el valor positivo.
En el centro del campo hay un circulo, el cual es dibujado usando la funcion **EllipseCurve** in Three.js:

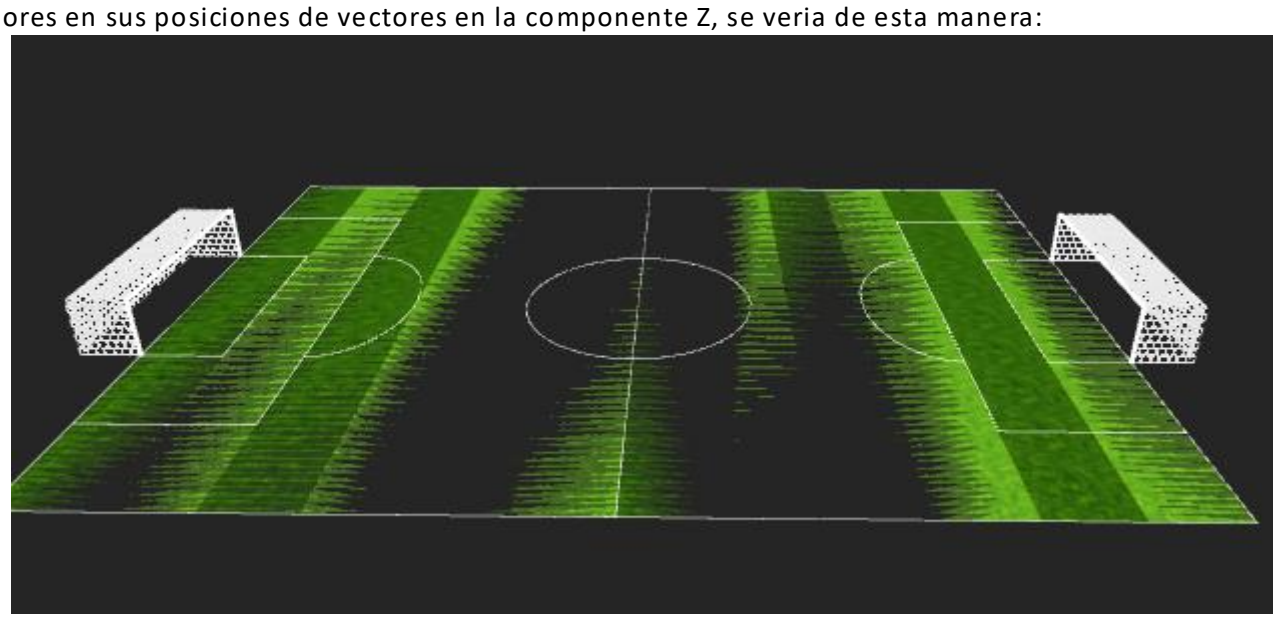
```
EllipseCurve( aX : Float, aY : Float
        , xRadius : Float, yRadius : Float
        , aStartAngle : Radians, aEndAngle : Radians
        , aClockwise : Boolean, aRotation : Radians )
```

Esto se dibuja en el eje XY, acostado en el campo y el plano. La misma función se utiliza para los arcos de penalty, pero con diferentes parámetros, haciendo un semicírculo con 180 grados – desde pi/2 a 3*pi/2. El arco izquierdo es dibujado en sentido horario, el derecho por el sentido opuesto.

El campo se encuentra ligeramente por encima del suelo y las marcas se encuentran ligeramente por encima del campo:

```
// put the field slightly above the ground, to prevent z-fighting
const field_z = 0.3;
// markings slightly above the field
const markings_z = 0.4;
```

Si tuvieran los mismos valores en sus posiciones de vectores en la componente Z, se veria de esta manera:



En **goal.js**, la funcion **create_goals** dibuja los arcos en los 2 lados:

```
6   // Creates both goals
7   export function create_goals(ground, ground_width, ground_height, field_width, field_height, num_stripes) {
8     create_goal(ground, ground_width, ground_height, field_width, field_height, num_stripes, Goals.LEFT);
9     create_goal(ground, ground_width, ground_height, field_width, field_height, num_stripes, Goals.RIGHT);
10  }
```

Esto llama a dos funciones internas, el ultimo parametro define el arco derecho o izquierdo, que es hecho por un enumerador con estos 2 valores.
Este parametro es usado en la funcion **create_goal** para invertir la posicion de las partes de los arcos cuando es igual a **Goals.RIGHT:**

```
26    var multiplier = (goal == Goals.LEFT) ? 1 : -1;
27
28    const goal_x = (multiplier * field_width)/2
```

Con el ultimo parametro configurado como **Goals.LEFT,** el valor **goal_x** sera **field_width/2.** En otras palabras, **-field_width/2;**
Estas variables define las dimenciones de los arcos. El ancho es el mismo que la marca enfrente de esta.

```
16    const goal_post_thickness = 0.5;
17
18    const goal_width = field_height / 3   - goal_post_thickness*2;
19    const goal_height = field_height / 12;
```

Está compuesto por tres cilindros para los postes y cuatro planos para la red detrás de él.
Los postes son hechos con la funcion **CylinderGeometry** en Three.js:

```
CylinderGeometry(radiusTop : Float, radiusBottom : Float
      , height : Float, radialSegments)
```

Ambos parámetros de radio se establecen en el valor de **goal_post_thickness**. Los cilindros se dividen en 8 subdivisiones a lo largo de la altura

```
const geometry = new THREE.CylinderGeometry(goal_post_thickness, goal_post_thickness, goal_height, 8 );
```

Los postes de la izquierda y derecha son rotados para quedarse firme arriba del piso:

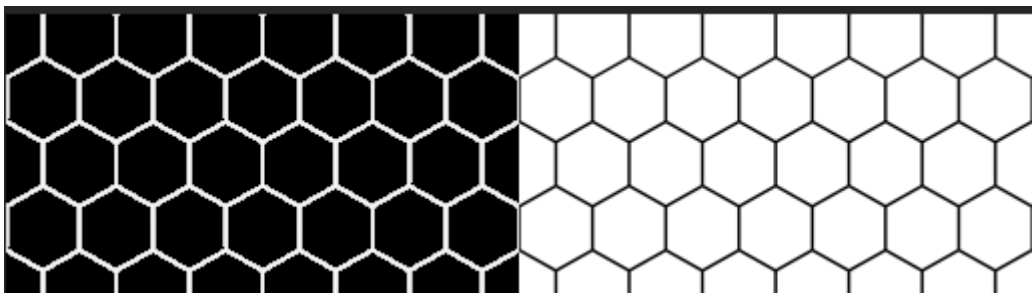
```
cylinder.rotation.x = -Math.PI/2;
```

El poste superior ya se encuentra paralelo al suelo cuando se crea, por lo que no se gira.

La red se compone de cuatro planos: dos a los lados, uno arriba, uno atrás. Se giran para mirar en la dirección correcta, según la rotación del suelo.
Para los planos izquierdo y derecho, simplemente giramos 90 grados para deshacer la rotación del suelo y volver a hacerlos paralelos al eje XY.

```
69    const geometry = new THREE.PlaneGeometry(goal_depth, goal_height);
70    const plane = new THREE.Mesh(geometry);
71    plane.material = new THREE.MeshStandardMaterial({map: netDiff, alphaMap: netAlpha, side: THREE.DoubleSide, alphaTest: 0.1});
72
73
74    plane.position.set (goal_x + (multiplier * goal_depth)/2, goal_y, goal_height/2);
75    plane.rotation.x = Math.PI/2;
```

El material de la red utiliza un mapa difuso y un mapa alfa. Las texturas se ven asi:



A la izquierda está el mapa alfa; todos los píxeles negros en esa textura se descartarán, por lo que la geometría será completamente transparente donde la imagen es negra.
A la derecha está el mapa difuso, que define los colores de la geometría, que se ven afectados por las luces en la escena; la parte blanca se descartará utilizando el mapa alfa de la izquierda, por lo que solo se verán las partes en negro.

```
58  // Textures and material used left and right parts
59  const netDiff = new THREE.TextureLoader().load('./textures/net_map.png');
60  const netAlpha = new THREE.TextureLoader().load('./textures/net_alpha.png');
61  netDiff.wrapS = netDiff.wrapT = THREE.RepeatWrapping;
62  netAlpha.wrapS = netAlpha.wrapT = THREE.RepeatWrapping;
63  netDiff.repeat.set (1, 1);
64  netAlpha.repeat.set (1, 1);
```

Las texturas están configuradas para repetirse una vez en todo el plano. Para las redes superior y trasera, que tienen un ancho mayor, repetimos el patrón cuatro veces:

```
96      netDiff.repeat.set (4, 1);
97      netAlpha.repeat.set (4, 1);
```

En **lightpost.js:**
Estos constants define las propiedades de los postes de luz:

```
6   const lightpost_height = 50;
7   const lightpost_width = 1;
8   const lamp_size = 9.75;
9   const light_intensity = 0.95;
```

El poste de luz es un grupo que contiene un cilindro gris alto y delgado y una pirámide blanca brillante en la parte superior.
Este es el poste:

```
13      const geometry = new THREE.CylinderGeometry(lightpost_width, lightpost_width, lightpost_height, 8);
14      const material = new THREE.MeshPhongMaterial({color: 0x303030, side: THREE.DoubleSide });
15      const cylinder = new THREE.Mesh(geometry, material);
16      cylinder.position.set(0, 0, lightpost_height / 2);
17      cylinder.rotation.x = -Math.PI / 2;
18      lightpost.add(cylinder);
```

Girado en el eje X para que quede de pie. Su color es RGB (30, 30, 30), un gris oscuro.
Usamos **ConeGeometry** de Three.js con los siguientes parametros

ConeGeometry(radius : Float, height : Float, radialSegments : Integer, heightSegments : Integer)

```
22      const geometry = new THREE.ConeGeometry(lamp_size, 8, 4, 4);
23      const material = new THREE.MeshPhongMaterial({color: 0x202020, emissive: 0xFFFFFF, side: THREE.DoubleSide });
24      const cone = new THREE.Mesh(geometry, material);
25      cone.position.set(0, 0, lightpost_height);
26      cone.rotation.y = Math.PI / 4;
27      cone.rotation.x = Math.PI / 12;
28      lightpost.add(cone);
```

Se gira ligeramente hacia abajo (hacia el suelo), 15 grados (pi / 12).
El material de la pirámide usa el parámetro de material 'emisivo' para emitir una luz blanca brillante que es claramente visible, ya que es mucho más fuerte que la luz ambiental de la escena, definida de esta manera, como una luz blanca con una intensidad del 24%, en main.js :

```
22  const ambientLight = new THREE.AmbientLight(0xffffff, 0.24);
23  scene.add(ambientLight);
```

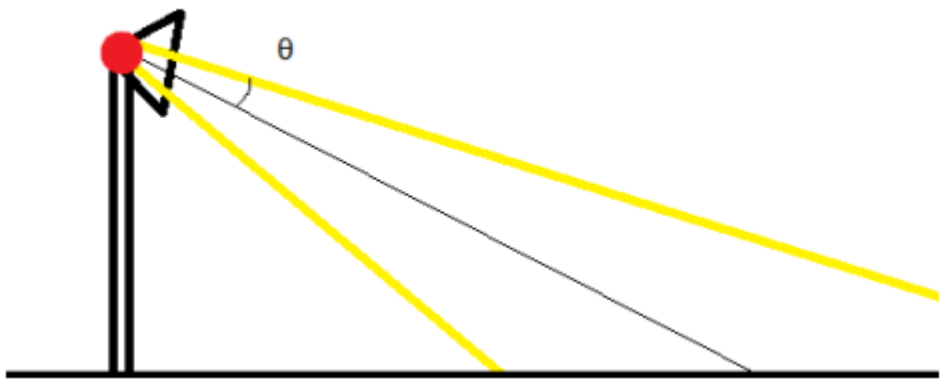
La luz es creada aquí:

```
30      const light = new THREE.SpotLight(0xffffff, light_intensity, 100, Math.PI/3);
31      light.position.set(cone.position.x, cone.position.y, cone.position.z);
32      light.distance = 350;
33      lightpost.add(light);
```

Un Spot Light emite luz a lo largo de un cono, definido por un punto, una dirección, una distancia y un ángulo. Esta es la function signature de Three.js **SpotLight**:

SpotLight(color : Integer, intensity : Float, distance : Float, angle : Radians, penumbra : Float, decay : Float)

Creamos una luz blanca, con un alcance de 100 unidades, con la intensidad definida anteriormente (0,95 en la línea 9). El ángulo del cono es de 60 grados – pi/3.
Dejamos los dos últimos parámetros con sus valores predeterminados.
En el dibujo a continuación, el punto rojo es la posición de la luz, las líneas amarillas representan el cono de luz, θ es el ángulo del cono, la línea negra delgada es la dirección de la luz, su longitud es 100 en este caso, que se extiende a lo largo del suelo.



Three.js establece la dirección de la luz en (0, 0, 0) de forma predeterminada, por lo que no cambiamos ese valor; ese es el centro mismo de nuestra escena, que es hacia donde apuntan todos los postes de luz.
El poste de luz se ve aquí desde un lado; el poste es creado por esa función que mira hacia nosotros, cuando el parámetro de rotación dado es igual a 0.
La función create_lightpost tiene un parámetro que indica la rotación, en radianes, alrededor del poste.

```
4  export function create_lightpost(ground, position, rotation) {
```

En **seats.js:**

La función exportada create_seats dibuja las cuatro gradas alrededor del campo, por lo que esa función solo necesita llamarse una vez en la escena.

Los soportes se crean dibujando tres cajas, con el mismo ancho y alto, pero diferentes profundidades, una encima de la otra.

```
21    // bottom seats
22    { ... }
34    // middle seats
35    { ... }
47    // top seats
48    { ... }
```

Estos usan la funcion **BoxGeometry** en Three.js, con estos parametros:

BoxGeometry(width : Float, height : Float, depth)

Estas dimensiones están definidas por los argumentos dados a create_seats, que se dan a las otras dos funciones.
Los asientos se crean en pares alrededor del campo (arriba y abajo, izquierda y derecha):

```
3  export function create_seats (ground, seat_height, seat_depth, seat_dist_x, seat_dist_y, field_width, field_height, num_stripes) {
4    create_top_bottom_seats (ground, seat_height, seat_depth, seat_dist_y, field_width, field_height, num_stripes, false);
5    create_top_bottom_seats (ground, seat_height, seat_depth, seat_dist_y, field_width, field_height, num_stripes, true);
6    create_left_right_seats (ground, seat_height, seat_depth, seat_dist_x, field_width, field_height, num_stripes, false);
7    create_left_right_seats (ground, seat_height, seat_depth, seat_dist_x, field_width, field_height, num_stripes, true);
8  }
```

Cada función puede dibujar cualquiera de cada par.
El último parámetro de las funciones mencionadas anteriormente en las líneas 4-7 indica si estamos dibujando la parte inferior o la derecha:

```
10  function create_top_bottom_seats(ground, seat_height, seat_depth, seat_dist, field_width, field_height, num_stripes, is_bottom) {

73  function create_left_right_seats(ground, seat_height, seat_depth, seat_dist, field_width, field_height, num_stripes, is_right) {
```

cuando **is_bottom** y **is_right** sea verdadero, gira los asientos 180 grados y los coloca en una posición diferente:

```
61    // if drawing the bottom part, rotate it 180 degrees, and place it in a different position
62    if (is_bottom) {
63      seats.position.y = -field_height/2 - (seat_depth*3)/2 - seat_dist;
64      seats.rotation.z = Math.PI;
65    } else {
66      seats.position.y = field_height/2 + (seat_depth*3)/2 + seat_dist;
67    }
```

Se hacen cosas similares en la función create_left_right_seats.
Usamos estos parámetros, como en la función de dibujo de arcos, por lo que no necesitamos duplicar nuestros códigos.
Las gradas superiores e inferiores se crean a lo largo del ancho del campo; izquierda y derecha a lo largo de la altura.

En **person.js:**

```
3   // Creates a person, returns the person group
4   export function create_person(ground, person_width, person_height, head_color, shirt_color, position, rotation = 0) {
```

Esta función dibuja una persona en la escena. El último parámetro, **rotation**, se utiliza al colocarlos en las gradas: en las gradas izquierda y derecha se giran 90 grados. El ancho, la altura, el color y el color de la camisa de la persona están controlados por estos parámetros.
Una persona está formada por un grupo que contiene la cabeza, que es una esfera, y el cuerpo y los brazos, que son cajas.

```
6     const person = new THREE.Group();
7
8     // materials
9     const headMaterial = new THREE.MeshPhongMaterial({ color: head_color });
10    const shirtMaterial = new THREE.MeshLambertMaterial({ color: shirt_color });
11
12    // head
13    const head = new THREE.Mesh(new THREE.SphereGeometry(person_width / 2, 16, 8), headMaterial);
14    head.position.y = person_height - person_width;
15    person.add(head);
16
17    // create body and arms with some proportions to the person's size
18    const arm_width = person_width/2;
19
20    const body = new THREE.Mesh(new THREE.BoxGeometry(person_width, person_height, person_width), shirtMaterial);
21    const l_arm = new THREE.Mesh(new THREE.BoxGeometry(arm_width, person_height/3, person_width/2), shirtMaterial);
22    const r_arm = new THREE.Mesh(new THREE.BoxGeometry(arm_width, person_height/3, person_width/2), shirtMaterial);
23
24    l_arm.position.x = -person_width/2 - arm_width/1.75;
25    r_arm.position.x = person_width/2 + arm_width/1.75;
26
27    l_arm.position.y = person_height/4;
28    r_arm.position.y = person_height/4;
```

Algunas proporciones se definen arriba: el radio de la cabeza es la mitad del ancho de la persona (línea 13); el tamaño del brazo es la mitad del ancho de la persona (línea 18), etc.
La función **create_person** devuelve el objeto Three.js **Group** con toda la geometría de la persona. Esto se hace para que podamos mantener un array de personas que creamos:

```
139   // Rows of people in each part of the seats, used in the wave animation
140   var people_row_1 = []
141   var people_row_2 = []
142   var people_row_3 = []
```

Estos estan en **main.js.** Son usados para la animacion de olas.

Aca los asientos son llenados con personas:

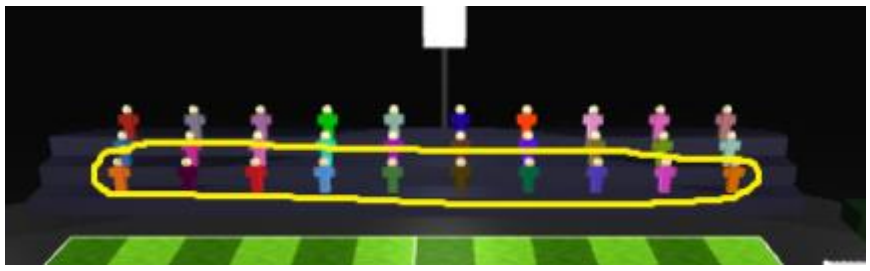
```
156     const num_people = 10; // number of people in this set of seats
157     const x = -field_width/2; // initial X position, first person in the row
158     const dist = (7*person_width); // spacing between people
159     var y = field_height/2 + (seat_depth*3)/2 + seat_dist_y; // initial Y position, starting by the bottom row
160
161     // We loop backwards to add the people to the row array in the right order, in both the top and left seats
162     // People in the lower seats
163     for (var i = num_people - 1; i >= 0; i--) {
164       const person = create_person(ground, person_width, person_height, 0xE2C597, random_color(), new THREE.Vector3(x + i * dist, y, seat_height));
165       people_row_1.push(person);
166     }
```

El loop en la línea 163 llama a **create_person**, dando a cada persona un color de camisa aleatorio y un vector 3D para la posición. Para los asientos medio y superior, el componente Y de este vector cambia, según la profundidad del asiento, y el componente Z también cambia, a un punto más alto:

```
170     y += seat_depth*2;
171     for (var i = num_people - 1; i >= 0; i--) {
172       const person = create_person(ground, person_width, person_height, 0xF7E9A7, random_color(), new THREE.Vector3(x + i * dist, y, seat_height*2));
173       people_row_2.push(person);
174     }
```

Se hace lo mismo para llenar las otras gradas.

Las personas en cada parte de todos los stands (inferior, medio, superior) se colocan en una fila, se almacenan en las arrays visto arriba; por ejemplo, en la parte inferior de la grada superior:



Estos marcados en amarillo son parte del array **people_row_1** mencionado anteriormente. Los del medio están en people_row_2, y los de arriba están en people_row_3.
Cada array contiene personas de las cuatro gradas.

El código para llenar las gradas es similar para las cuatro gradas, solo cambian las posiciones y las rotaciones. Se agrupan en ámbitos anónimos, en **main.js**:

```
152   // -- seats
153   {
154     // top seats
155     { ... }
183
184     // left seats
185     { ... }
210
211     // bottom seats
212     { ... }
236
237     // right seats
238     { ... }
263   }
```

Para hacer la animacion de las olas:

Hay una bandera llamada **making_wave** cuyo valor es **verdadero** durante el transcurso de la animación. Se restablece a **falso** al final.

La variable **current_col** indica qué columna de personas en las gradas se pondrán de pie en un momento dado. Una columna está marcada en amarillo a continuación:

Esta columna es aumentado pasando el tiempo, hasta que la ultima columna es alcanzada.

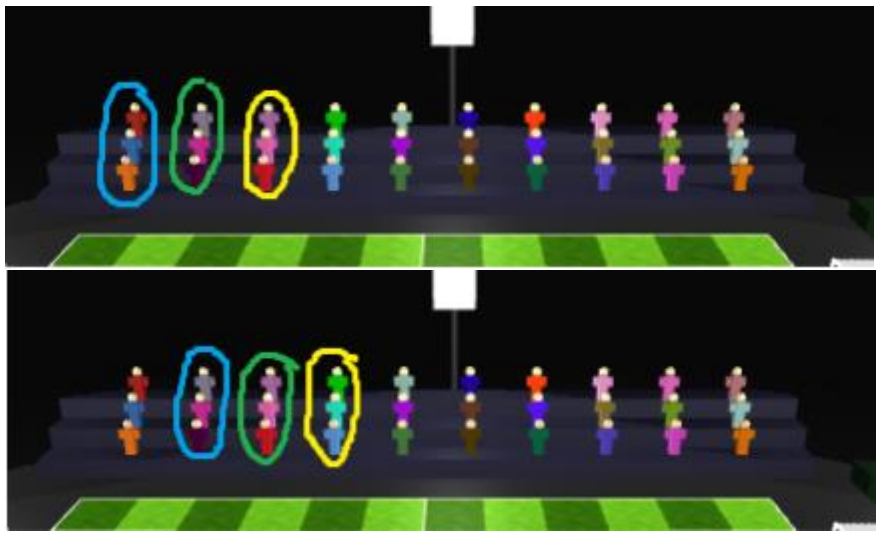
```
279   if (making_wave) {
280     if (delta > 0.035) {
281
282       // number of 'columns' of people: 10 at the top, 10 at the bottom, 9 in the left seats, 9 in the right seats
283       if (current_col < 38) {
```

El número total de columnas depende de cuántas personas haya; configuramos esto a los valores que usamos actualmente.
La variable **delta** almacena cuánto tiempo ha pasado desde la última vez que se llamó a esta función.
El valor comparado con **delta** controla qué tan rápido ocurre la animación; un valor más bajo significa una animación más rápida.

La animación en sí se realiza iterando sobre cada columna de las tres filas y alterando la posición Z de esa columna de personas.

```
285        // current person stands
286        if (people_row_1[current_col] !== undefined) {
287          people_row_1[current_col].position.z += 7;
288          people_row_2[current_col].position.z += 7;
289          people_row_3[current_col].position.z += 7;
290        }
291
292        // previous person lowering
293        if (people_row_1[current_col-1] !== undefined) {
294          people_row_1[current_col-1].position.z -= 3;
295          people_row_2[current_col-1].position.z -= 3;
296          people_row_3[current_col-1].position.z -= 3;
297        }
298
299        // the one before the last one returns to his place in the seat
300        if (people_row_1[current_col-2] !== undefined) {
301          people_row_1[current_col-2].position.z = seat_height + person_height/2;
302          people_row_2[current_col-2].position.z = seat_height*2 + person_height/2;
303          people_row_3[current_col-2].position.z = seat_height*3 + person_height/2;
304        }
```

La columna anterior recibe un valor Z más bajo, y la anterior vuelve a la posición inicial.
La altura de la ola está controlada por estos números 7 y 3 en las líneas 278-289 y 294-296 anteriores.



En algún momento, la fila de personas marcadas en amarillo se pondrá de pie, las de la marca verde, que antes estaban de pie , bajarán un poco, y las marcadas en azul volverán a su posición inicial.
En la próxima iteración, las columnas aumentarán, como se muestra en la segunda imagen, y lo mismo sucederá con cada grupo.
Esto continúa hasta la última columna.