D:\UCC\React\PortafolioUCC\Project\_v3\_multiplayer\_final\backend\controllers\blockController.js

// Obtener bloques sin \_id

exports.getBlocks = async (req, res) => {

    const blocks = await Block.find({}, { name: 1, x: 1, y: 1, z: 1, \_id: 0 })

    res.json(blocks)

}

// Agregar un nuevo bloque

exports.addBlock = async (req, res) => {

    const { x, y, z } = req.body

    const newBlock = new Block({ x, y, z })

    await newBlock.save()

    res.status(201).json({ message: 'Bloque guardado', block: newBlock })

}

# Ajustar de esta forma

exports.getBlocks = async (req, res) => {

    try {

        const level = parseInt(req.query.level) || 1;

        const blocks = await Block.find({ level: level }).select('name x y z level -\_id'); // 👈 selección limpia

        res.json(blocks);

    } catch (error) {

        res.status(500).json({ message: 'Error al obtener bloques', error });

    }

};

exports.addBlock = async (req, res) => {

const { name, x, y, z, level } = req.body;

const newBlock = new Block({ name, x, y, z, level });

await newBlock.save();

res.status(201).json({ message: 'Bloque guardado', block: newBlock });

}

D:\UCC\React\PortafolioUCC\Project\_v3\_multiplayer\_final\backend\models\Block.js

const mongoose = require('mongoose')

const blockSchema = new mongoose.Schema({

    name: String,

    x: Number,

    y: Number,

    z: Number

})

module.exports = mongoose.model('Block', blockSchema)

# Cambiar de esta forma

const mongoose = require('mongoose');

const blockSchema = new mongoose.Schema({

name: String,

x: Number,

y: Number,

z: Number,

level: {

type: Number,

required: true,

default: 1

}

});

module.exports = mongoose.model('Block', blockSchema);

Con estos ajustes vamos a renovar el archivo Text\_3\_cero.txt D:\UCC\React\PortafolioUCC\Project\_v3\_multiplayer\_final\backend\scripts\Text\_3\_cero.txt, para que incluya el nivel del cual vamos a exportar el nivel de blender.

```js

import bpy

import os

import json

from mathutils import Vector

# -------- Configuraciones --------

nivel = 2 # 🔵 Puedes cambiar a 2, 3, etc. para otros niveles

# Rutas de exportación

export\_path = r"D:/UCC/React/PortafolioUCC/Project\_v3\_multiplayer\_final/game-project/public/models/toycar2"

json\_path = os.path.join(export\_path, "toy\_car\_blocks1.json")

os.makedirs(export\_path, exist\_ok=True)

positions = []

# Configuración de Blender

depsgraph = bpy.context.evaluated\_depsgraph\_get()

bpy.ops.object.mode\_set(mode='OBJECT')

bpy.context.view\_layer.update()

# -------- Recorrer objetos seleccionados --------

for obj in bpy.context.selected\_objects:

if obj.type != 'MESH':

continue

obj\_eval = obj.evaluated\_get(depsgraph)

matrix\_world = obj\_eval.matrix\_world

loc = matrix\_world.to\_translation()

# 🔵 Normalizar el nombre

name = obj.name.lower().replace(" ", "\_").replace(f"\_lev{nivel}", "")

# 🔵 Asignar nuevo nombre con nivel

name\_with\_level = f"{name}\_lev{nivel}"

obj.name = name\_with\_level

# 🔵 Conversión de coordenadas para Three.js

pos = {

"name": obj.name, # Nombre ya corregido

"x": loc.x, # X real

"y": loc.z, # Z de Blender pasa a Y en Three.js

"z": -loc.y, # Invertir Y de Blender para ser Z en Three.js

"level": nivel

}

positions.append(pos)

print(f"📦 {obj.name}: ({pos['x']}, {pos['y']}, {pos['z']}, nivel {pos['level']})")

# -------- Exportar modelo individual --------

bpy.ops.object.select\_all(action='DESELECT')

obj.select\_set(True)

bpy.context.view\_layer.objects.active = obj

filepath = os.path.join(export\_path, f"{obj.name}.glb")

bpy.ops.export\_scene.gltf(

filepath=filepath,

use\_selection=True,

export\_format='GLB',

export\_apply=False

)

# -------- Guardar archivo JSON --------

with open(json\_path, "w") as f:

json.dump(positions, f, indent=4)

print("✅ Exportación finalizada con sistema de coordenadas compatible con Three.js.")

```

Despues de esto exportamos el archivo que genera los glb y el json.

Luego el archivo toy\_car\_blocks.json, lo ubicamos en : backend\data

El archivo coin.json, hay que agregar o ajustar su estructura para que tambien tenga level: 1, para esto te puedes ayudar de una IA para que haga la reparación del archivo.

Luego ajustamos la parte de: backend\scripts\sync\_blocks.js, para enviar la informacion de forma correcta a la base de datos de mongodb, aunque si los dos archivos a cargar en mongodb tienen la misma estructura no deberías ajustar nada en este script.

Vaciamos la base de datos y cargamos la informacion con el manejo de level.

node scripts/sync\_blocks.js

Verificar con Postman la informacion de los endpoints ajustados

| **Acción** | **Nueva estructura esperada** |
| --- | --- |
| **GET /api/blocks?level=1** | Trae solo bloques del nivel 1 |
| **POST /api/blocks** | Agrega un bloque con name, x, y, z, level |
| **POST /api/blocks/batch** | Agrega varios bloques cada uno con name, x, y, z, level |

Se trabaja el segundo mundo y como se dejaran algunos elementos del anterior escenario iguales, solo que rotados, estos se dejaran en otro directorio para que sean leidos para cargar.

Para esto hay que modificar el nivel en la exportación de blender, ajustar el archivo backend\scripts\sync\_blocks.js para que inserte el archvo de bloques pero utilizándolos de esta forma:

* data\toy\_car\_blocks1.json
* data\coin1.json

Los cuales internamente deben estar apuntando a nivel 2

La carpeta donde se encontraran estos recursos será : game-project\public\models\toycar2

Para trabajar el archivo generate\_sources.js realizar los ajustes necesarios para crear archivos nuevos que no vayan a reescribir los elementos del primer escenario.

Ahora tambien es necesario que en los archivos coin el final del nombre del archivo quede apuntando a \_lev1 o \_lev2 porque el archivo en : scripts\Text\_3\_cero.txt se ajustaron los nombres.

Con esto ya se puede dejar todos los recursos en un solo directorio y estará listo para funcionar de forma estándar en el nivel 1