Programación de Videojuegos

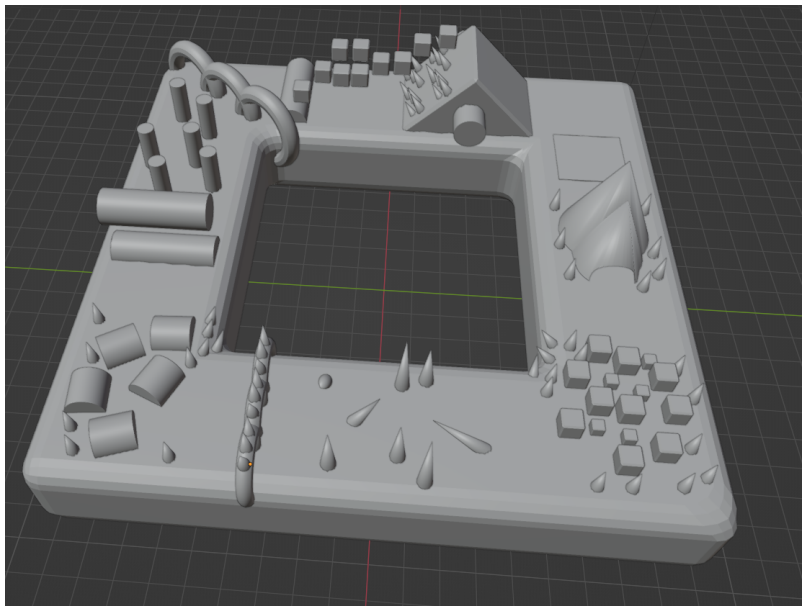
Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Parcial 2
Angel de Jesús Maldonado Juárez
Actividad 2
30 de marzo del 2023

Primer avance del proyecto

El proyecto consta de la creación de un videojuego con vista de primera persona, en donde el jugador deberá superar un mapa de obstáculos. Este mapa deberá basarse en el inflable mostrado en el video *Ultimate Survival - 395ft / 120m Inflatable Obstacle Course*:

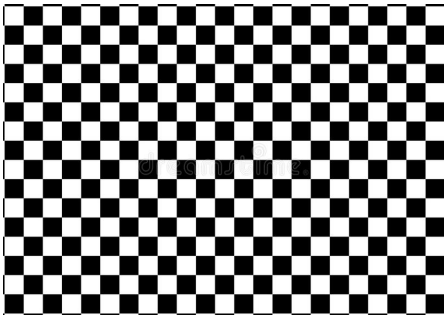
Modelado del mapa

Para el modelado del mapa y los obstáculos se utiliza el programa *Blender*, el avance del mapa sin texturas ni materiales es de la siguiente forma:

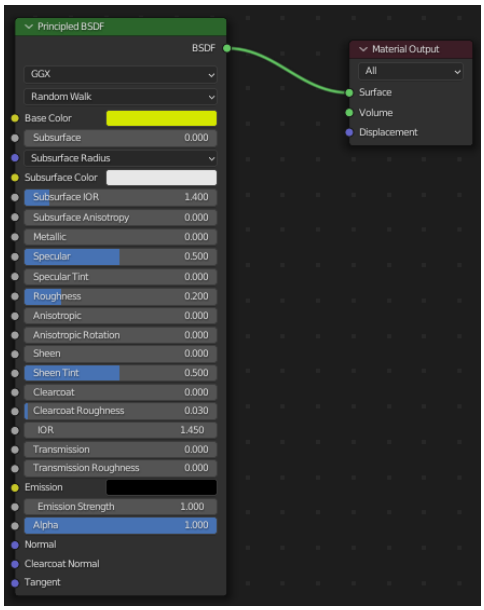


Para la base del mapa se utiliza una imagen que simula la textura y apariencia de la lona del inflable:

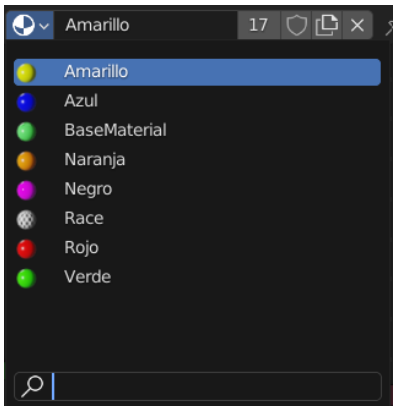
Para la placa de meta también se utiliza una imagen con el patrón de cuadros blanco y negro:



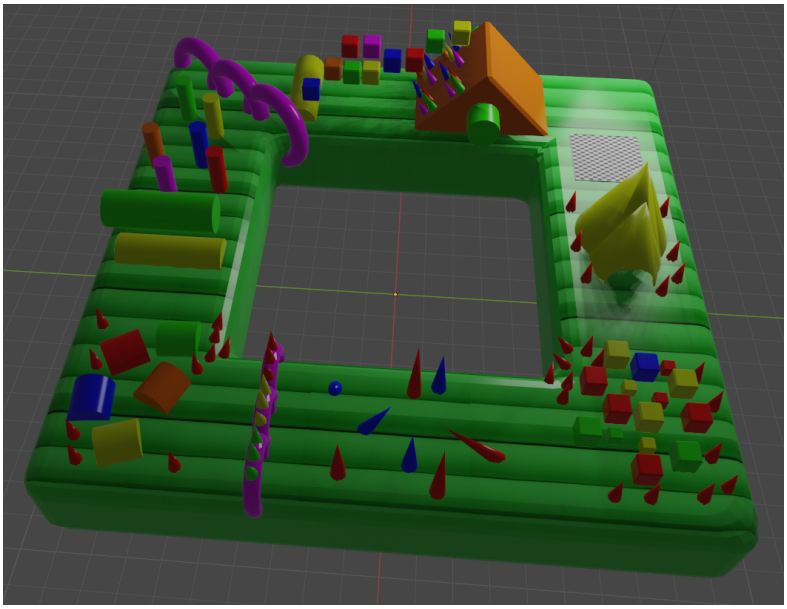
Los picos y obstáculos del mapa no utilizan ninguna textura, sin embargo, se crea un material utilizando un nodo **BSDF** (*Bidirectional Scattering Distribution Function*), cuya configuración queda de la siguiente forma:



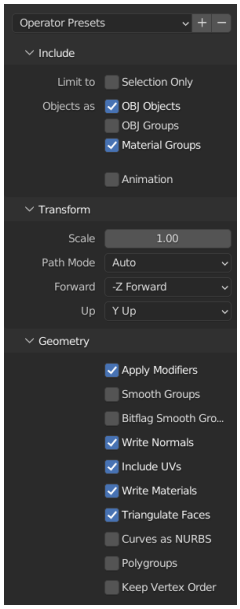
Cada obstáculo será de un color distinto, por lo que simplemente se optó por duplicar el primer material creado la misma cantidad de colores que habrá en el mapa:



Finalmente, al aplicar las texturas y materiales a cada elemento en el mapa, este se visualiza de la siguiente manera:



Para la exportación del modelo se optó por el formato *.obj*, por lo que se generará el archivo del modelo *.obj* junto con el archivo que contiene los materiales *.mtl* la configuración de exportado queda de la siguiente manera:



Lectura del modelo y los materiales en OpenGL

Como segundo avance del proyecto se carga el modelo del mapa del juego utilizando *OpenGL*, la plantilla del proyecto en *Visual Studio*, y el código visto en clases.

Para la carga del modelo simplemente se reemplaza la cadena de la ruta de uno de los modelos que se cargan en el archivo `main.cpp`:

```
    glfwMakeContextCurrent(window);

    if (!gladLoad())
    {
        return -1;
    }

    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);

    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
    glfwSetKeyCallback(window, key_callback);

    glEnable(GL_DEPTH_TEST);

    Shader ourShader("vertexShader.vs", "fragmenShader.fs");

    stbi_set_flip_vertically_on_load(true);
    Model ourModel("Recursos/Modelos/backpack/backpack.obj");

    Model ourModel1("Recursos/Nivel/nivel.obj");

    updateWindow(window, ourShader, ourModel1, ourModel);

    glfwTerminate();
    return 0;
}

void initGLFWVersion()
{
    glfwInit();
```

A manera de prueba, simplemente se modifica el valor de la variable `UseTexture` en el archivo `vertexShader.vs` entre 0 y 1 para visualizar el mapa sin texturas y con color o con texturas y sin color (respectivamente):

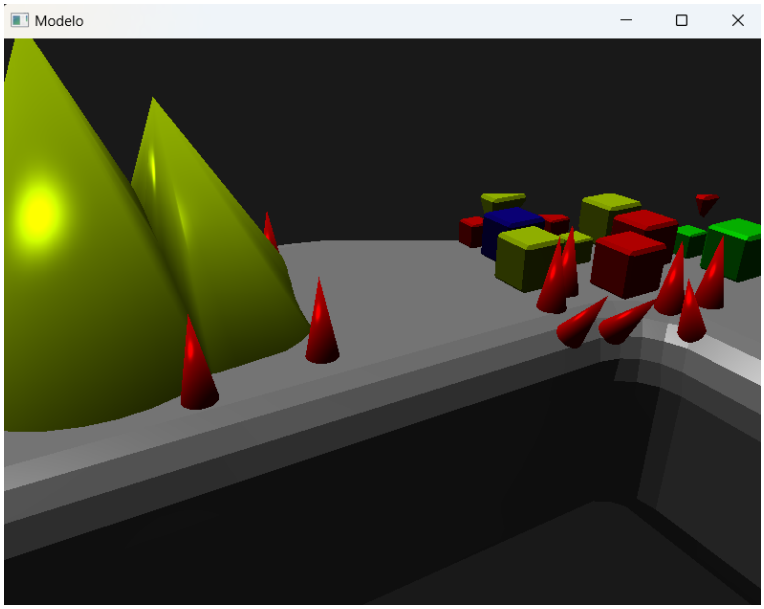
```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec4 aColor;
layout (location = 3) in vec2 aTexCoords;

out vec3 FragPos;
out vec3 Normal;
out vec2 TexCoords;
out float UseTexture;
out vec4 Color;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = mat3(transpose(inverse(model))) * aNormal;
    TexCoords = aTexCoords;
    Color = aColor;
    UseTexture = 1.0f;
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

Mapa sin texturas y con color (en OpenGL)



El mapa queda corto para el área de visualización (renderizado), por lo que el mapa debe escalarse a un tamaño más chico, o debe agrandarse el área de renderizado.

Mapa con texturas y sin color (en OpenGL)