

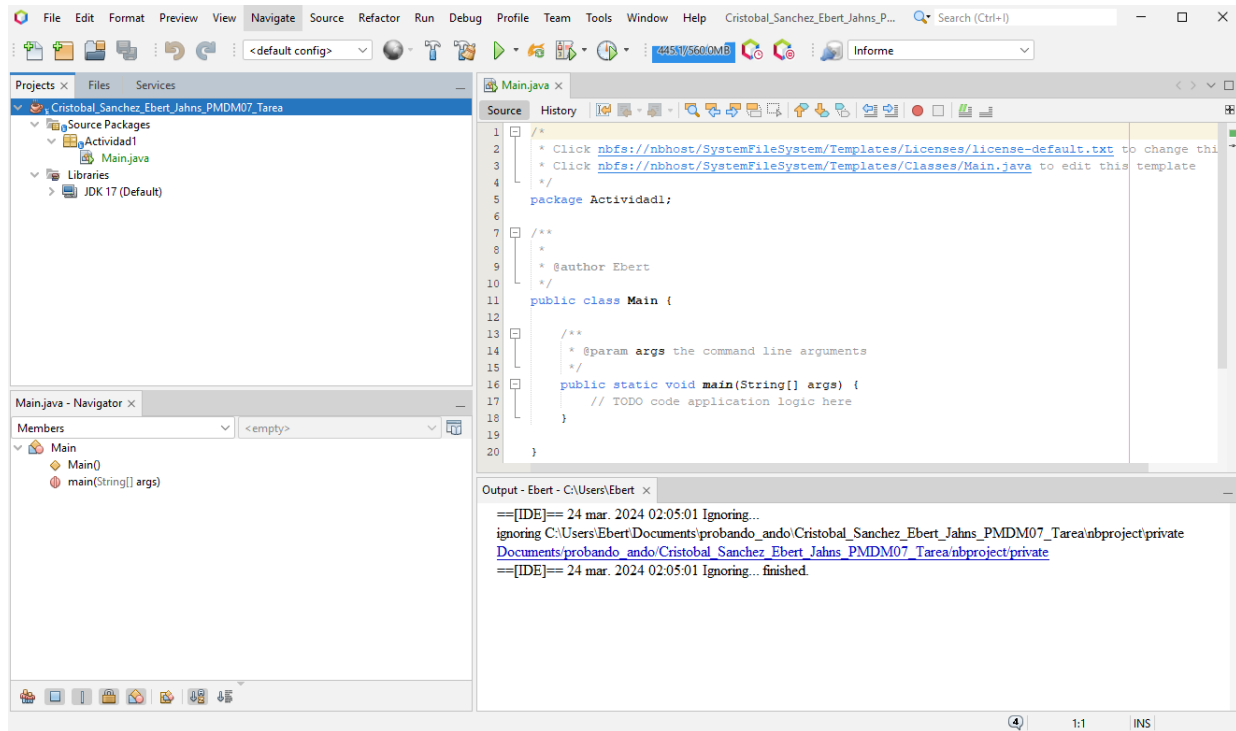
TABLA DE CONTENIDOS

Actividad 1	3
Configuración del entorno de desarrollo	3
Probando código proporcionado.	4
Clases creadas para desarrollar la tarea.....	5
- Clase Bolos.java	6
- Clase Bolo.java	6
- Clase Bola.java.....	6
- Main.java.....	6
Funcionamiento del juego.....	7
Inconvenientes de la actividad 1	9
Actividad 2	11
Preparando entorno de desarrollo.	11
Paso 0:	12
Paso 1:	13
Paso 2:	13
Paso 3:	14
Paso 4:	15
Paso 5:	15
Como se puede ver en la imagen anterior se han colocado 10 objetos. Y también se han creado los Scripts ContadorObjetos y InteraccionObjetos.	16
ContadorObejtos	16
Paso 6:	17
Funcionalidad del juego:	18
Inconvenientes de la actividad 2	19
Anexos Actividad 1	20
- Código bola.java	20
- Código bolos.java	20
- Código bolo.java	21
- Código Main.java.....	21

Actividad 1

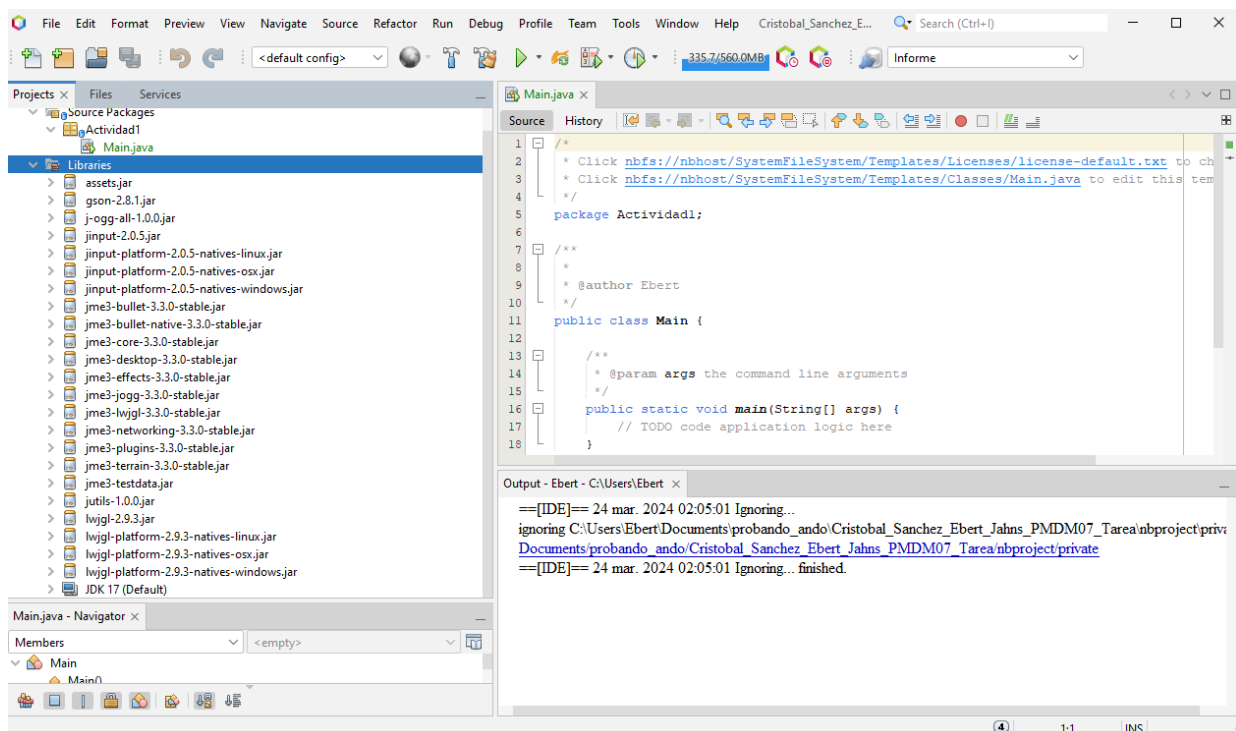
Para realizar esta tarea se ha utilizado **NetBeans 15**, **JDK 17** y **SO Windows 10**

Primero se ha creado el proyecto **Cristobal_Sanchez_Ebert_Jahns_PMDM07_Tarea**.



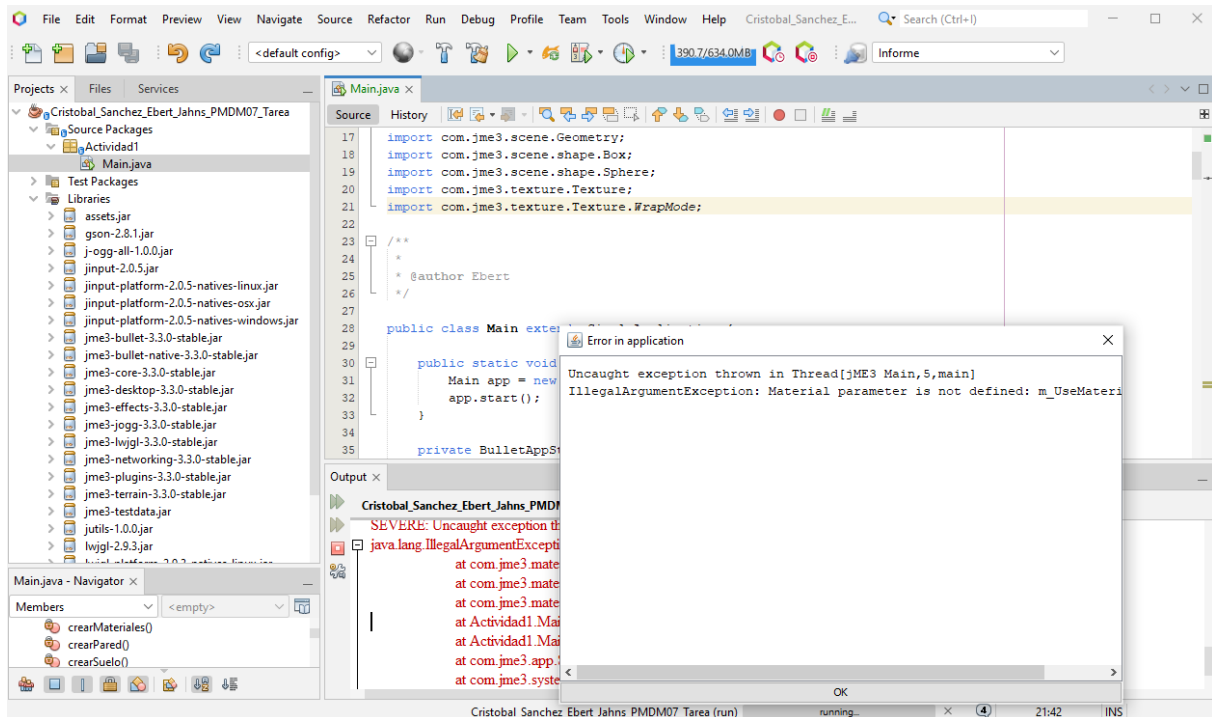
Configuración del entorno de desarrollo

Luego se ha descargado y se han agregado las librerías necesarias en el proyecto para Configurar **jme3** en NetBeans

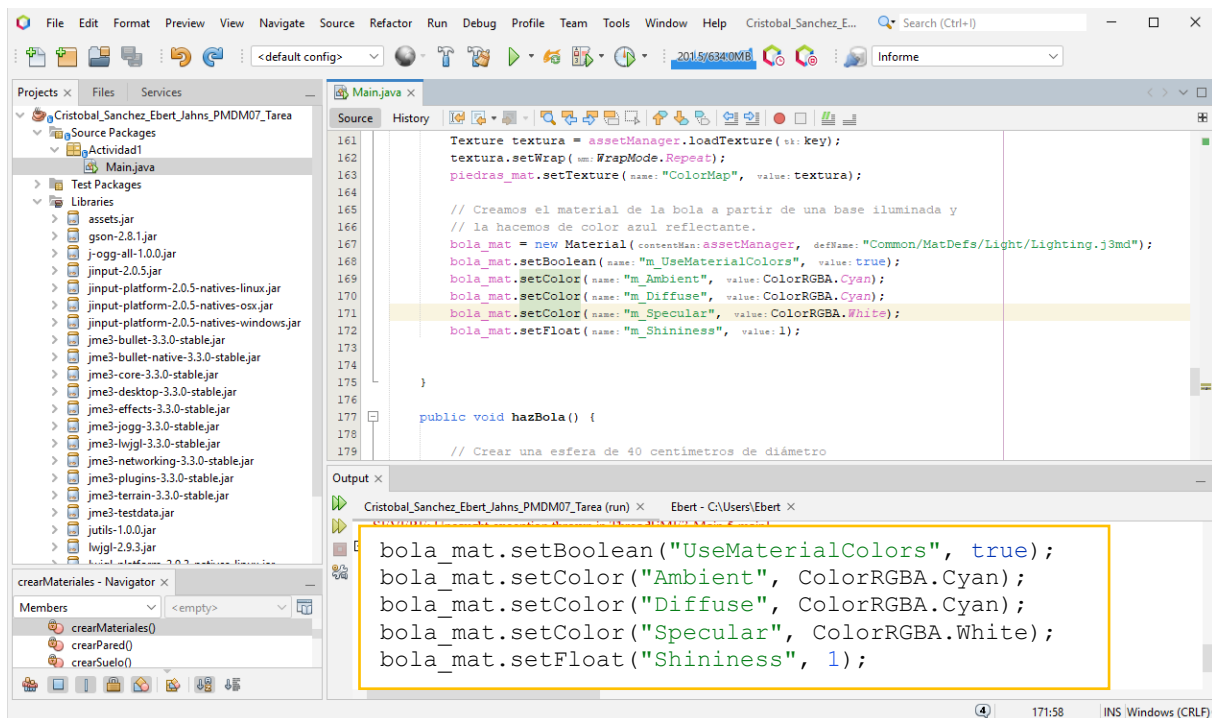


Probando código proporcionado.

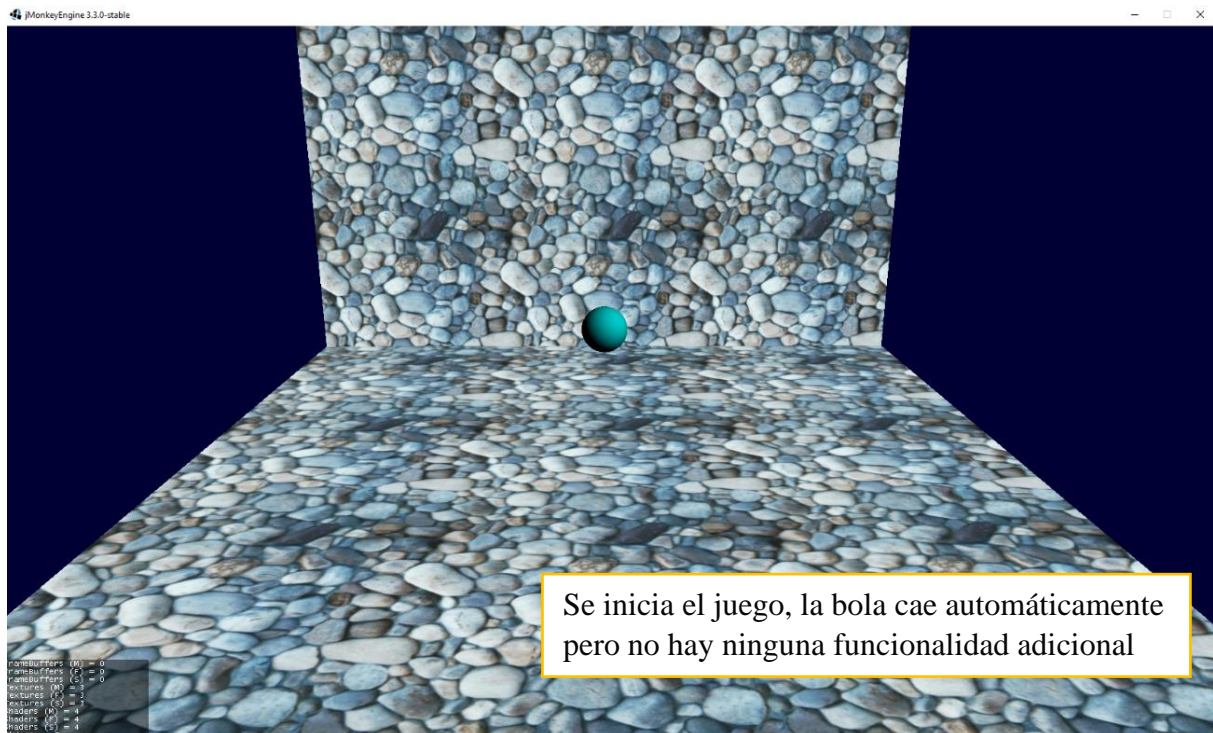
Como punto de partida se ha utilizado el código proporcionado en los recursos de esta tarea, se copió el código en la clase **Main.java**, al ejecutar el código nos muestra el siguiente error:



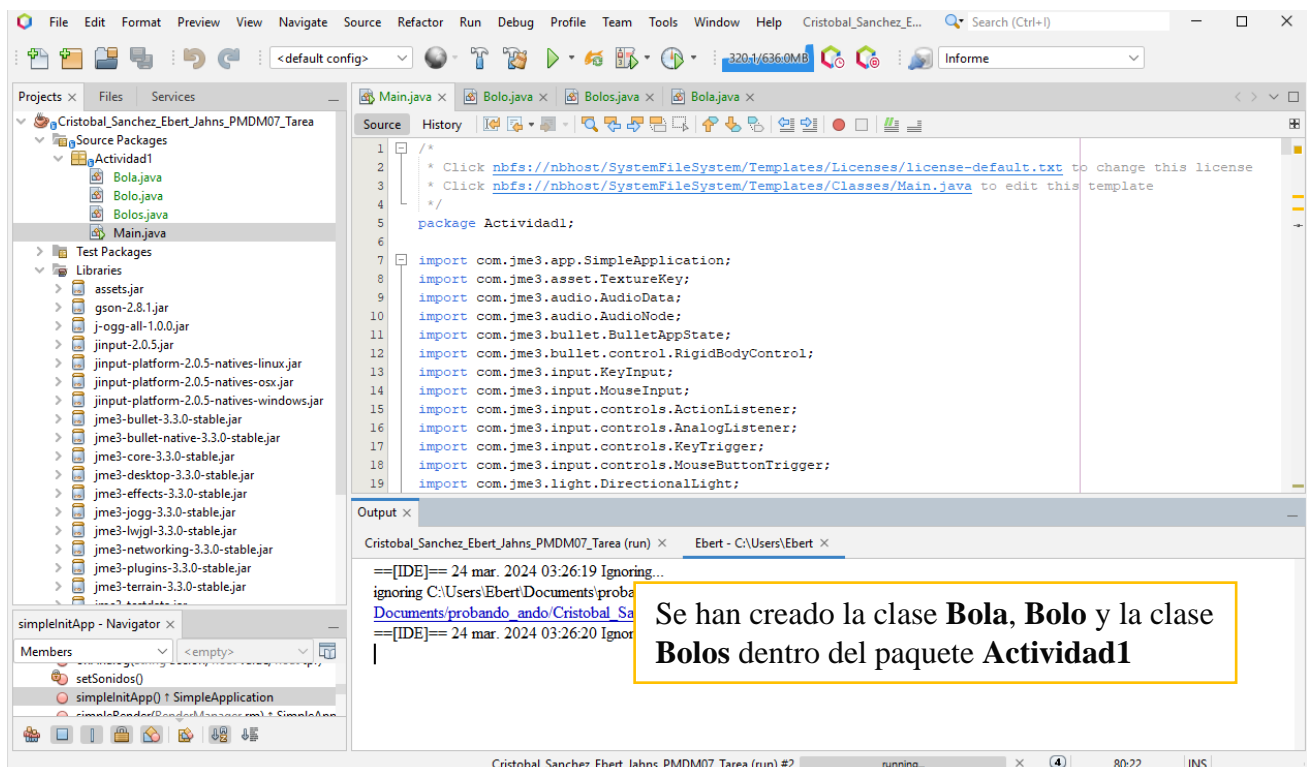
Se logra ver que el error indica que el parámetro `m_UseMaterialColors` no está definido, para solucionar esto solo se eliminó “`m_`” como se muestra en el recuadro de siguiente:



Al ejecutar el juego se nos abre primero una pantalla para elegir la resolución de pantalla de juego y posteriormente se abre la imagen que se muestra a continuación:



Clases creadas para desarrollar la tarea



- **Clase Bolos.java**

Este código define una clase llamada **Bolos** que indica la disposición y características físicas de los bolos en el juego. Utiliza coordenadas tridimensionales para representar la posición de los bolos en el espacio, con distintas filas y separaciones entre ellos. Además, establece propiedades físicas como la masa, altura y diámetro de los bolos.

- **Clase Bolo.java**

Este código define una clase llamada **Bolo** que extiende la clase **Geometry**, lo que sugiere que representa un objeto gráfico en el juego. El constructor de la clase **Bolo** recibe un **AssetManager** como parámetro para poder asignar un material al objeto. Dentro del constructor, se crea un cilindro con un diámetro y una altura determinados según los valores definidos en la clase **Bolos**. Luego, se rota el cilindro para que quede en posición vertical y se le asigna un material que define el aspecto visual del objeto.

- **Clase Bola.java**

Este código define una clase llamada **Bola** que encapsula las propiedades estáticas de una bola, como su masa, diámetro y velocidad media. Estas constantes son útiles en situaciones donde se requiere modelar o simular el comportamiento de una bola en un juego.

- **Main.java**

Este código Java implementa un simulador básico de bolos. La clase **Main** extiende de **SimpleApplication**, que proporciona un marco para desarrollar aplicaciones gráficas en 3D. La aplicación crea un entorno de juego que incluye un suelo, una pared, luces y un conjunto de bolos. Los controles de entrada están configurados para lanzar la bola con la tecla de espacio o el clic del ratón. La física del juego se gestiona utilizando **BulletAppState**, que simula la interacción entre los objetos en el entorno.

Una bola se crea y se lanza cuando se activa la acción correspondiente mediante la tecla de espacio o el clic del ratón. La bola se mueve con una velocidad determinada en función de la dirección de la cámara.

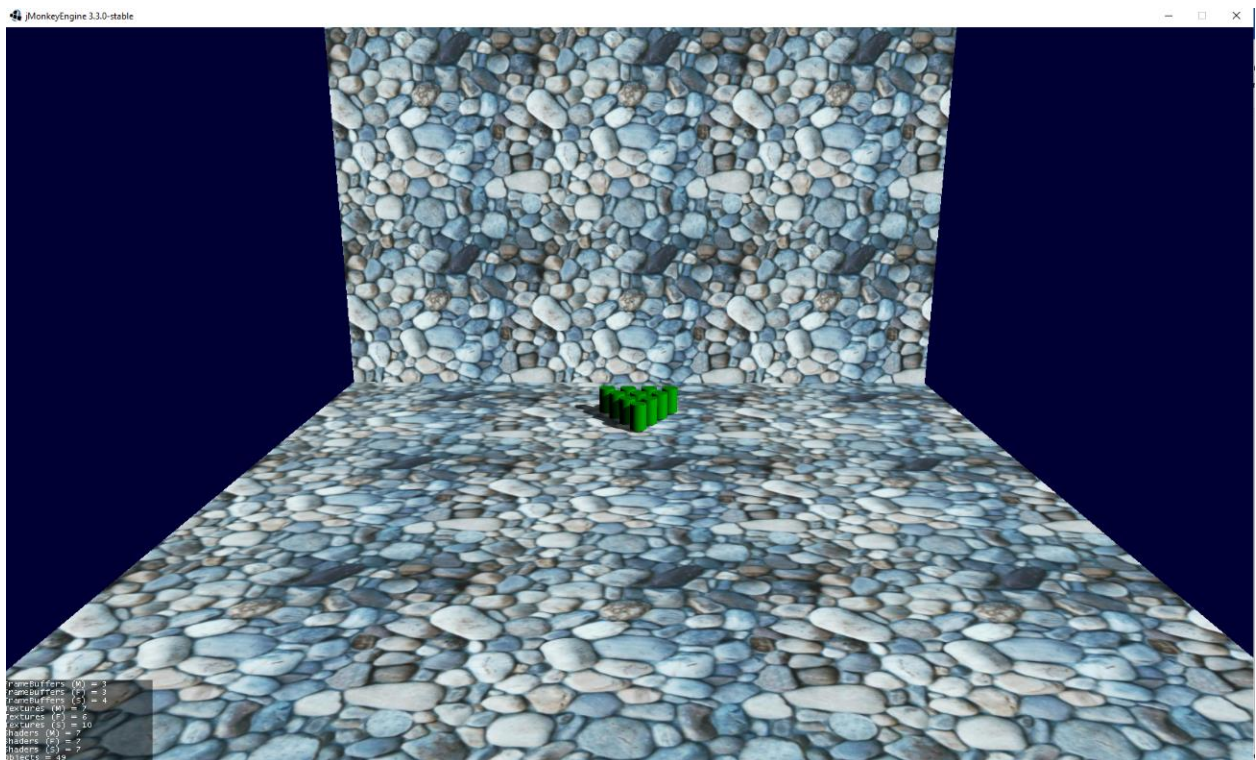
Además, se crean y colocan en la escena diez bolos, cada uno con su propia física para simular su comportamiento cuando son derribados por la bola.

El código también maneja la reproducción de sonidos, con efectos de sonido que se activan cuando se lanza la bola.

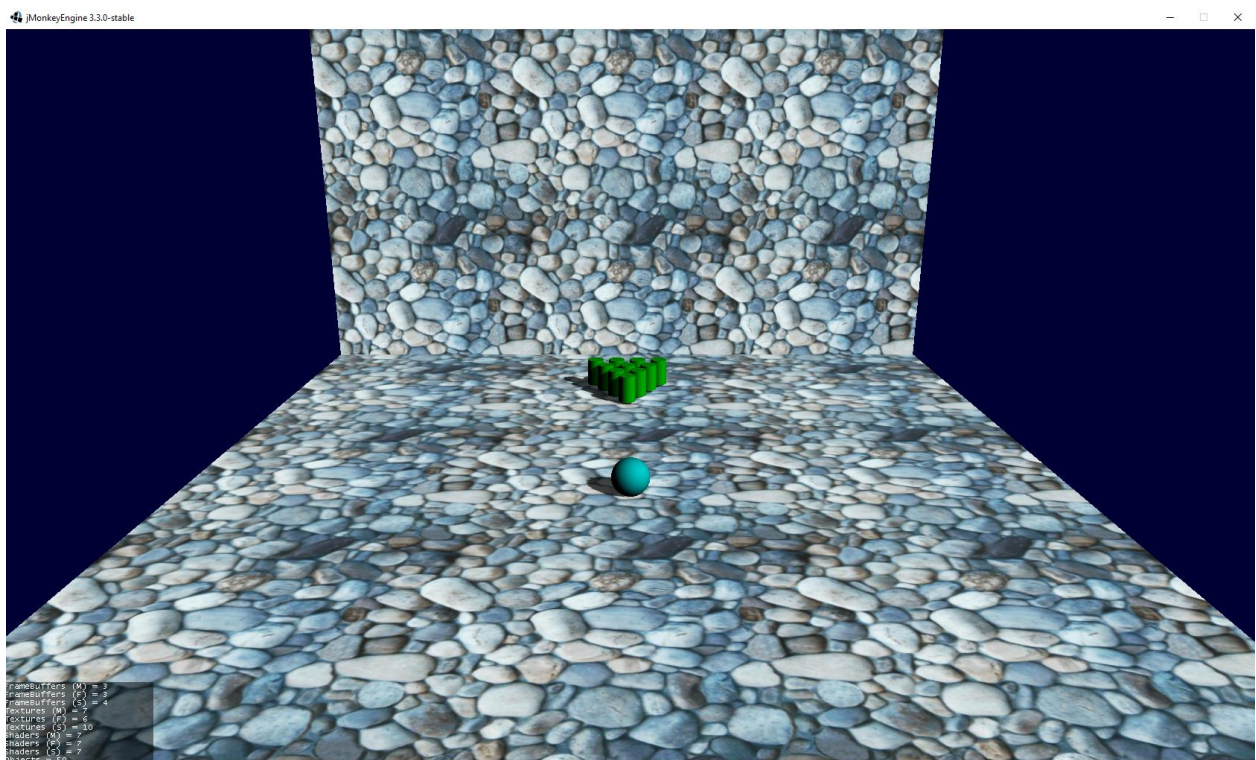
Por último, el código permite activar y desactivar la gravedad dentro del juego para ellos solo es necesario presionar la **tecla G**. Tenemos el método **onAction** que utiliza la biblioteca Bullet Physics. Se encarga de desactivar o activar la gravedad en el juego dependiendo de si se presiona un botón asociado a la acción "Desactivar Gravedad". Si la gravedad está activada, la desactiva y viceversa, cambiando el estado de un objeto **bulletAppState**. Además, muestra un mensaje en la consola indicando si la gravedad ha sido activada o desactivada.

Funcionamiento del juego

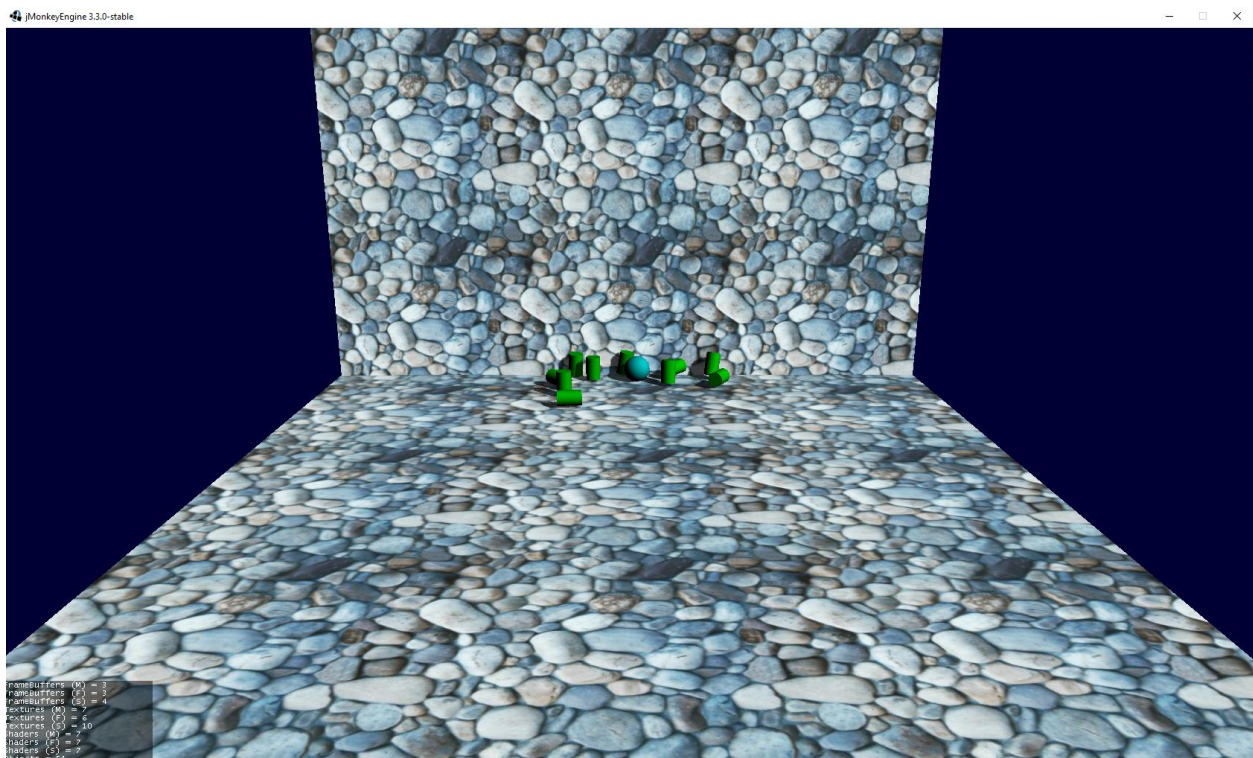
El juego inicia con los bolos colocados en forma de cuña.



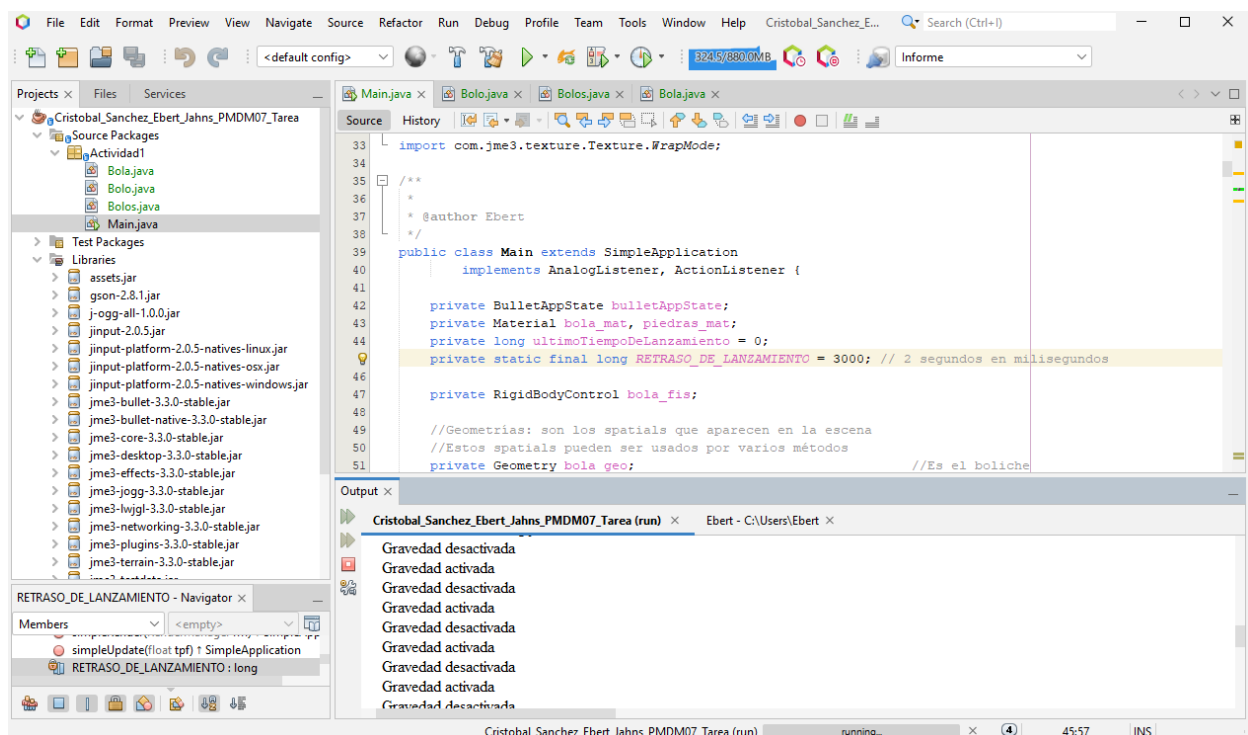
La bola es lanzada presionando la tecla space o presionando clic izquierdo, se pueden lanzar todas las bolas que queramos, pero para ellos hay que esperar 3 segundos antes de poder lanzar una nueva bola. Al momento de lanzar una nueva bola la bola que fue lanzada anterior a esta desaparece. Cuando se lanza una bola se emite un sonido.



Cuando la bola golpea los cilindros que representan los bolos estos caen.

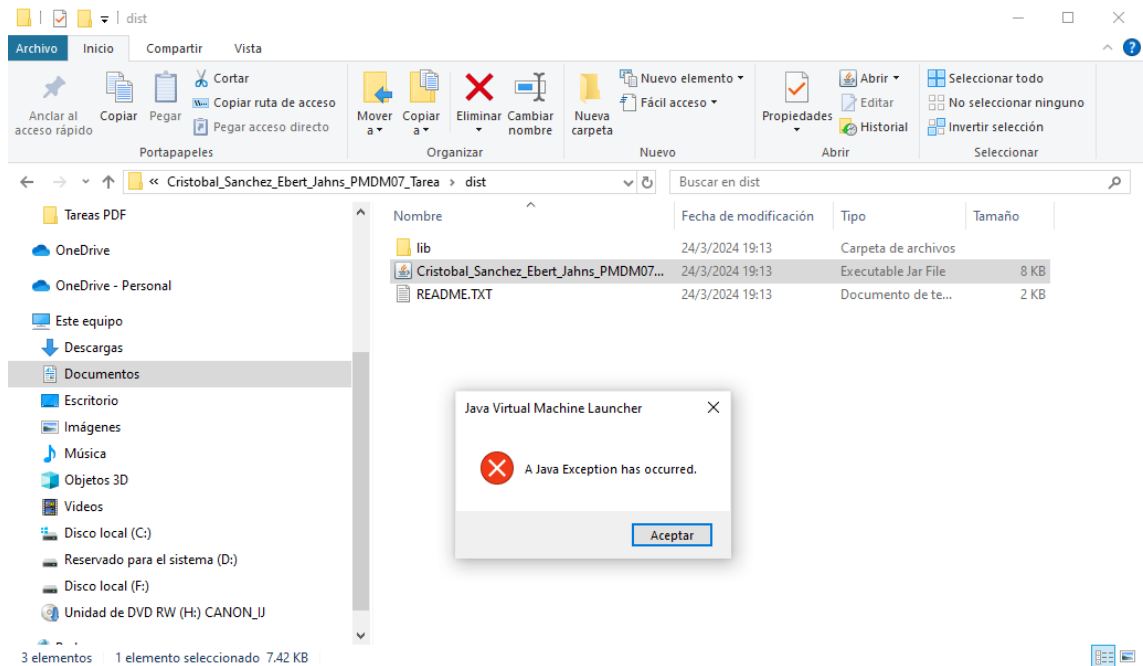


Cuando presionamos la **letra G** todo movimiento dentro del juego queda suspendido y se queda todo inmóvil cuando presionamos la **letra G** nuevamente todo vuelve a funcionar con normalidad. Como esto no se puede apreciar en una imagen dentro del juego, muestro que cada vez que presionamos la letra G en el juego en la consola del entorno de desarrollo nos imprime un mensaje.



Inconvenientes de la actividad 1

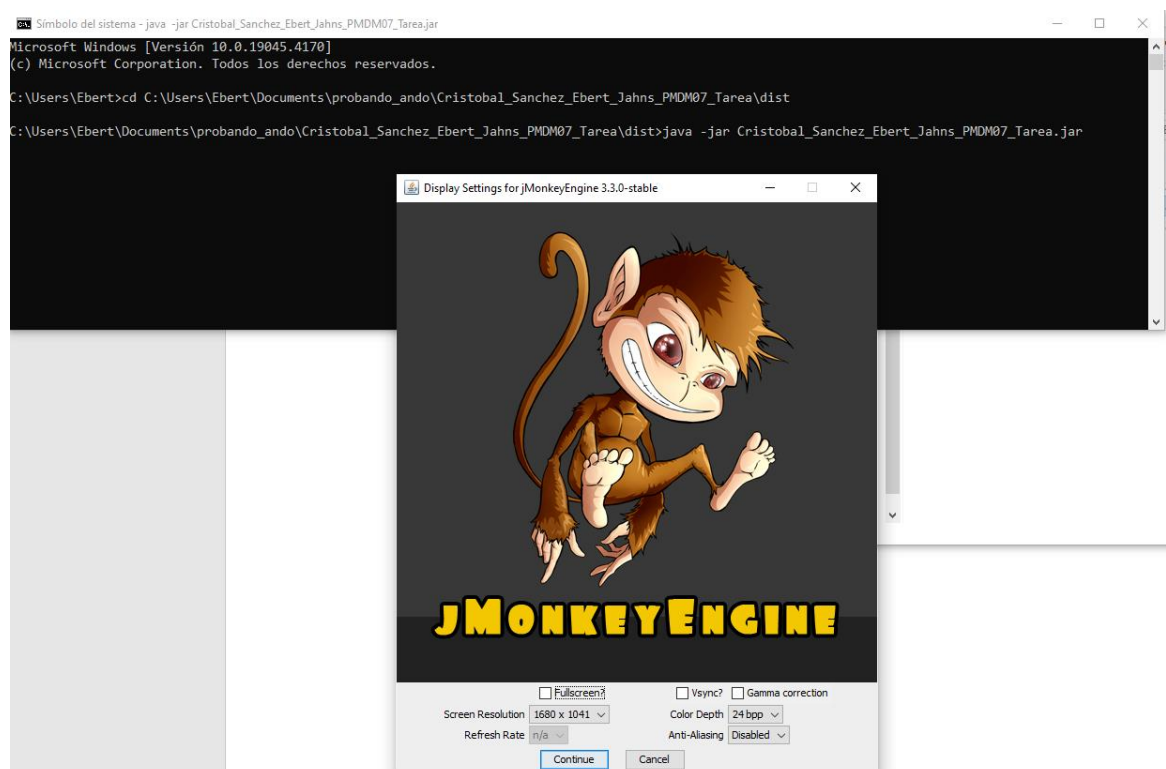
Se ha generado el archivo .jar, pero al hacer doble clic sobre el aparece un error que no he conseguido solucionar. Este error solo aparece al hacer doble clic sobre el fichero.

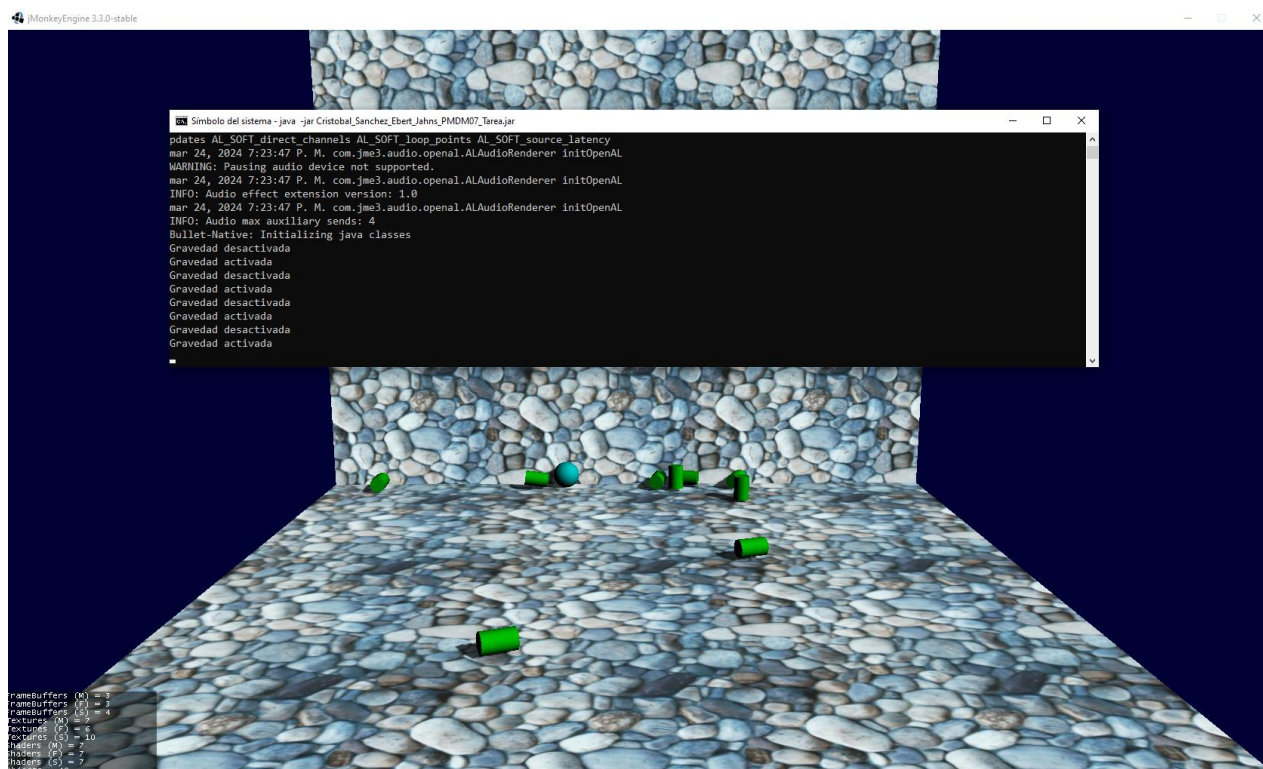


Sin embargo, si se ejecuta el archivo .jar desde consola si funciona correctamente.

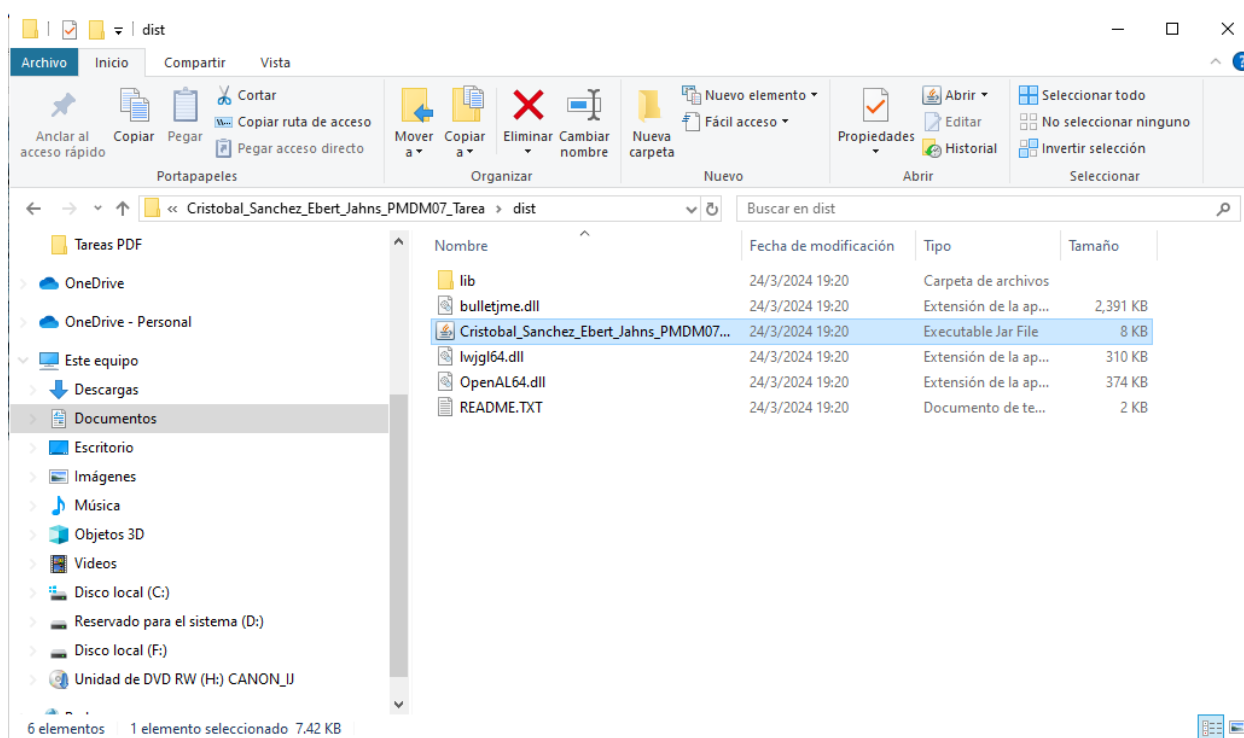
Nos colocamos en la carpeta dist de nuestro proyecto y ejecutamos el comando:

```
java -jar Cristobal_Sanchez_Ebert_Jahns_PMDM07_Tarea.jar
```



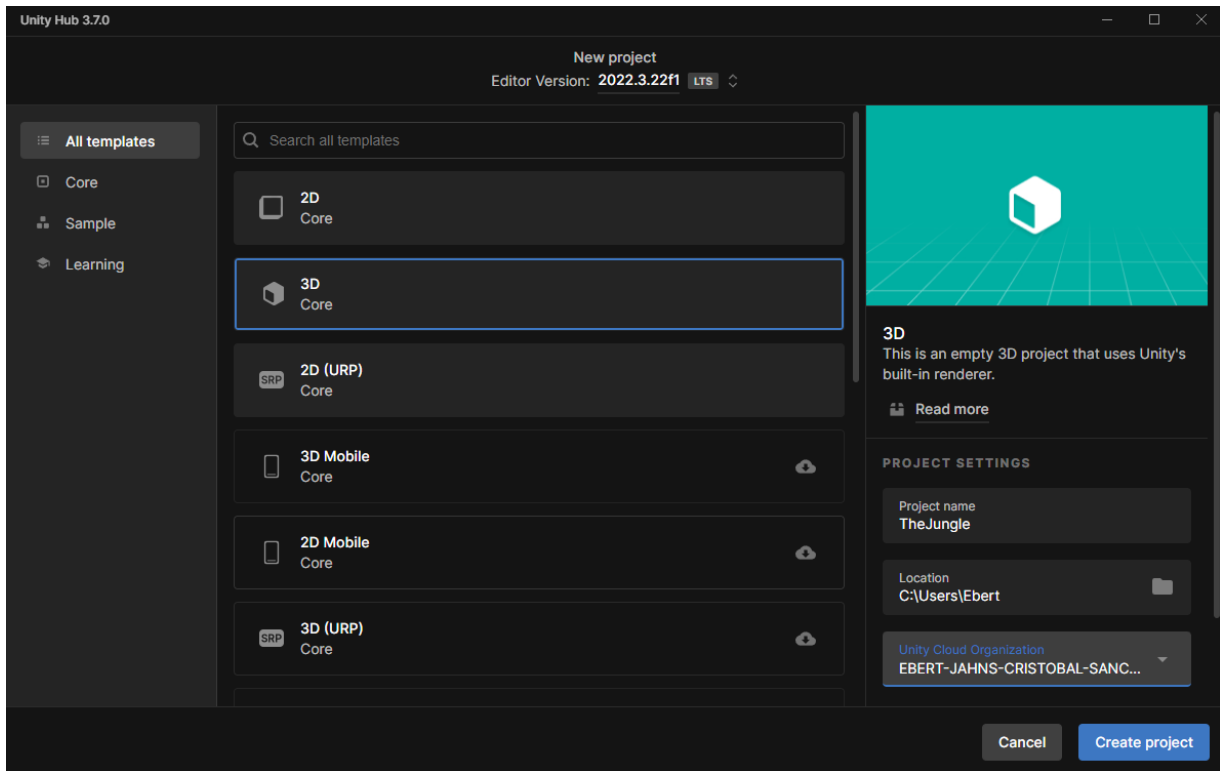


Al ejecutarse la aplicación en la carpeta dist se generan automáticamente nuevos ficheros.

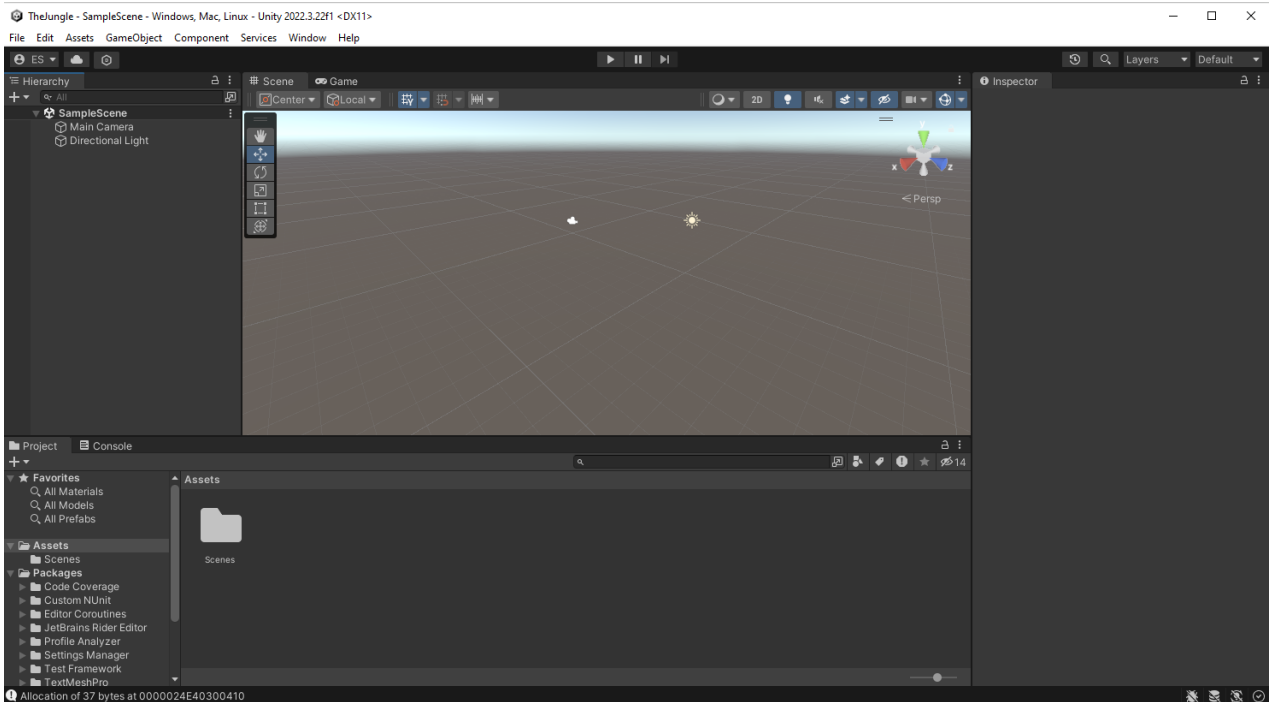


Actividad 2

Preparando entorno de desarrollo.

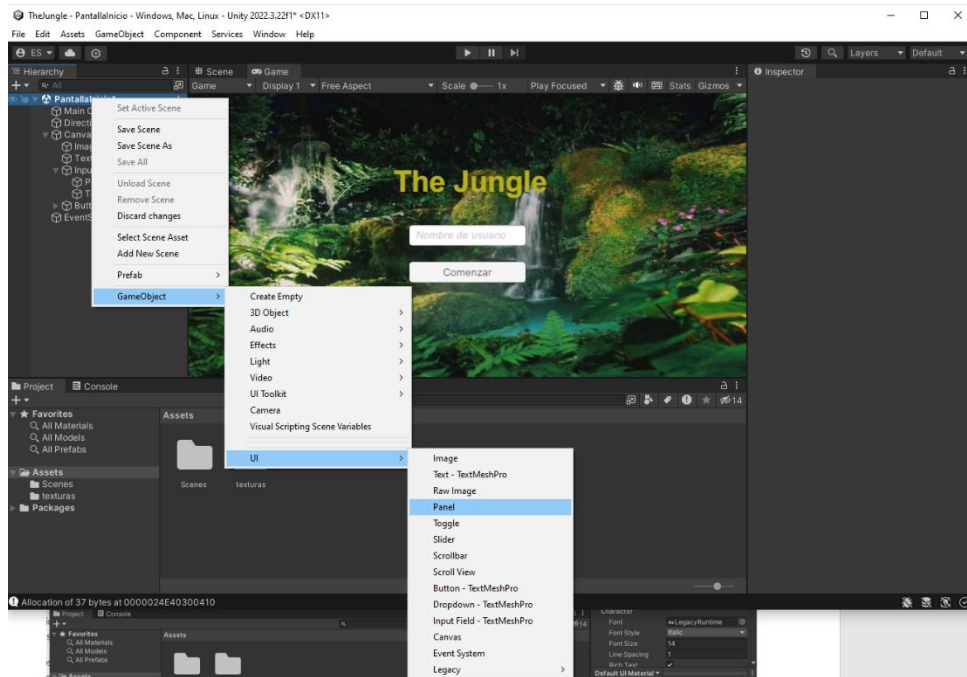


tenemos listo ya el entorno de desarrollo para comenzar a realizar los puntos solicitados en esta tarea



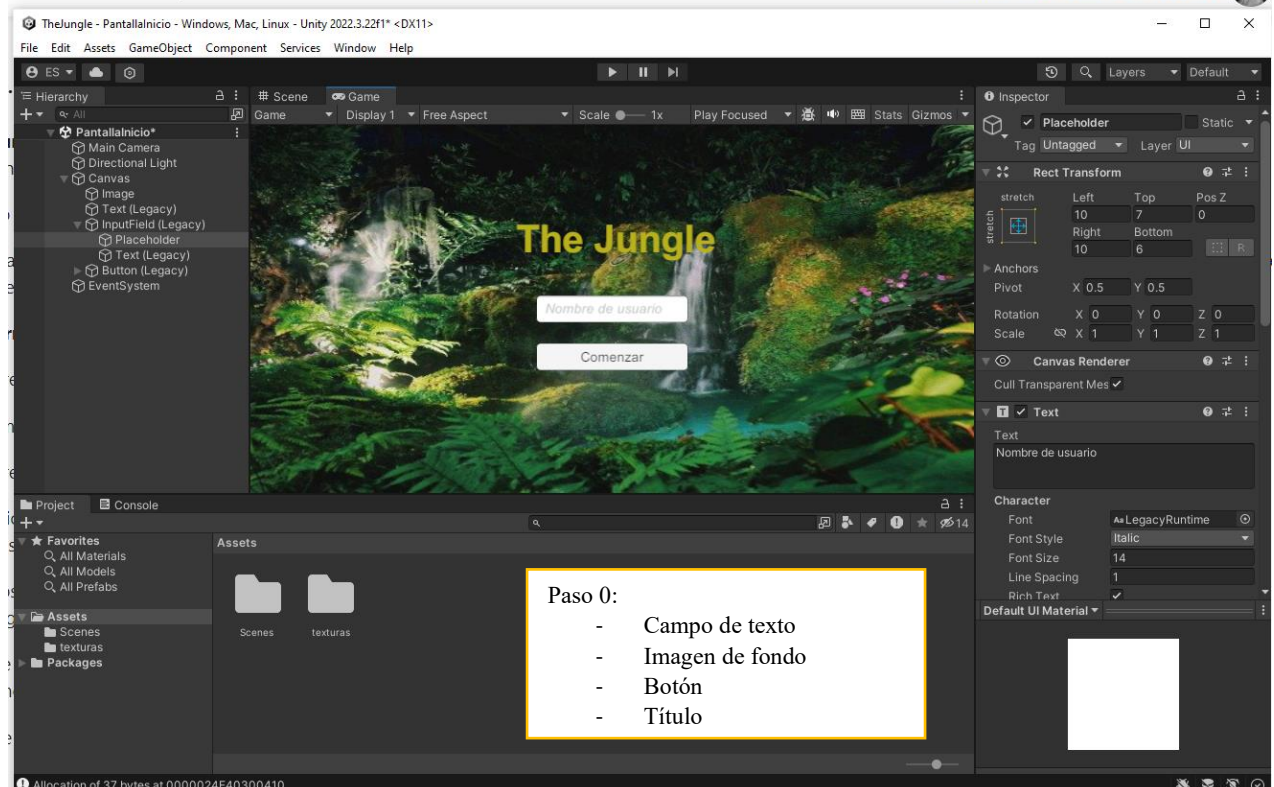
Paso 0: Realiza una pantalla inicial, con el título del juego, una imagen, un campo de texto y un botón para ir a la pantalla de juegos. En el campo de texto se pedirá el nombre de usuario que se pasará a la escena de juego.

Para realizar este paso se ha creado una Escena que le hemos llamado **PantallaInicio**, y dentro de ella se ha entrado en **GameObject** y se ha insertado elementos que he considerado eran necesarios para cumplir con este apartado.



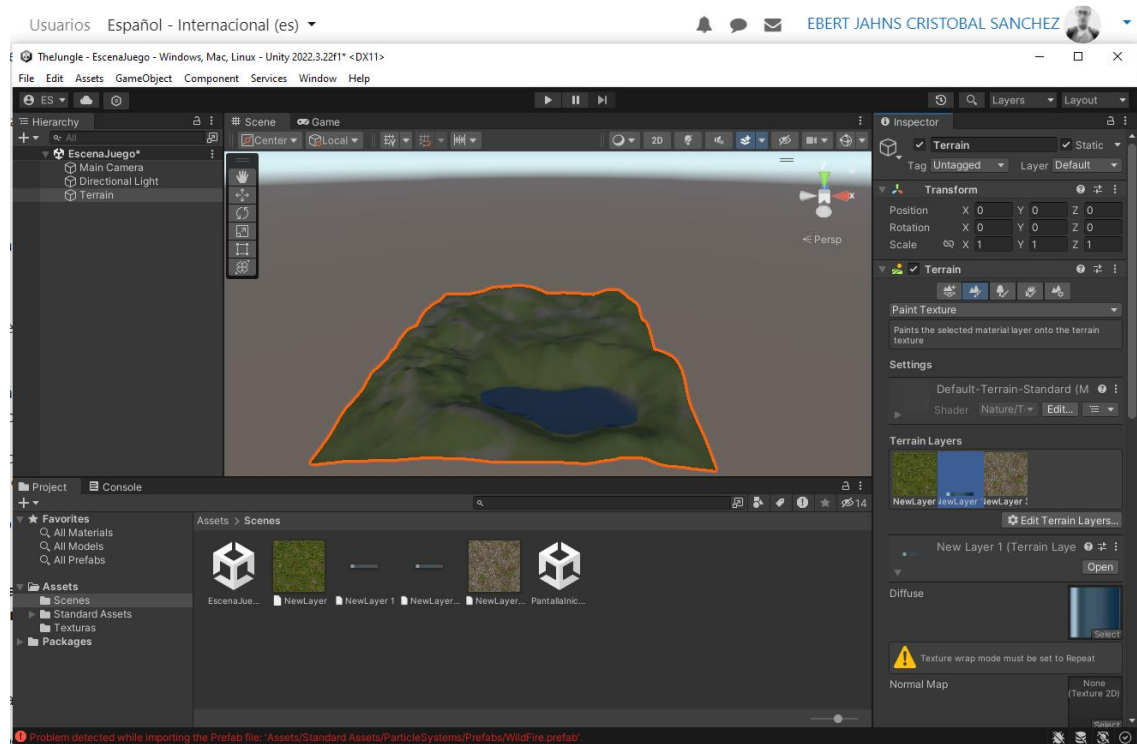
Uso de Usuarios Español - Internacional (es)

EBERT JAHNS CRISTOBAL SANCHEZ



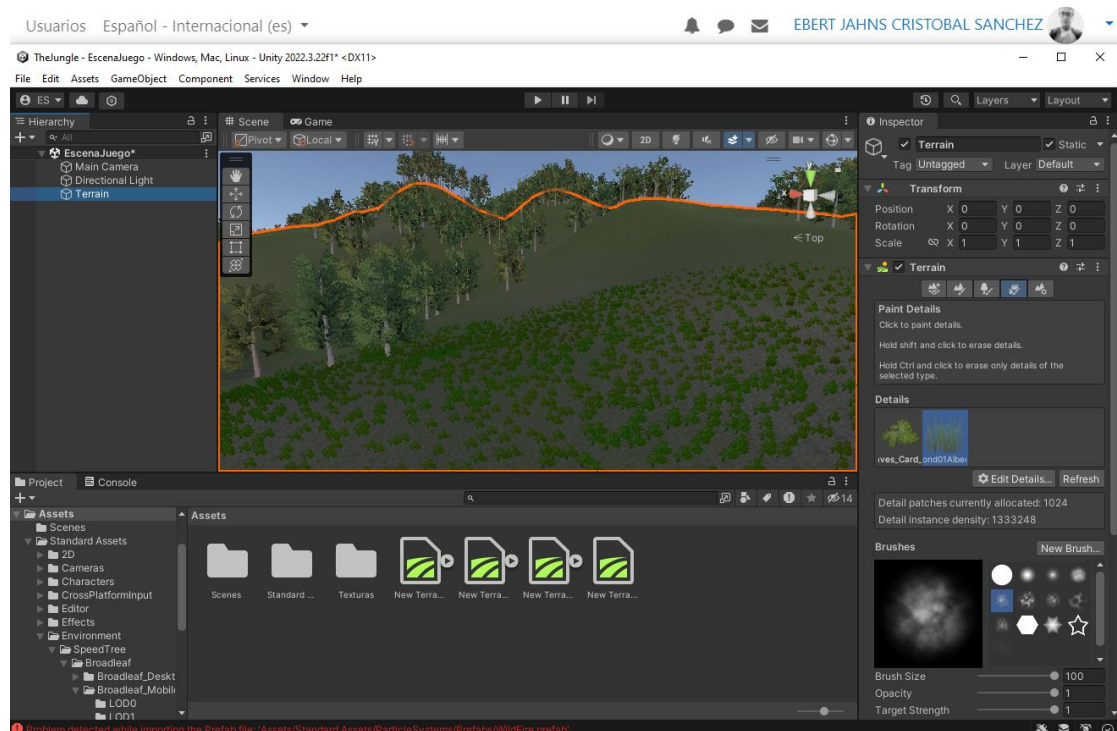
Paso 1: Un **terreno** con al menos tres texturas aplicadas, el terreno tendrá variaciones en altura (colina, montaña, volcán...).

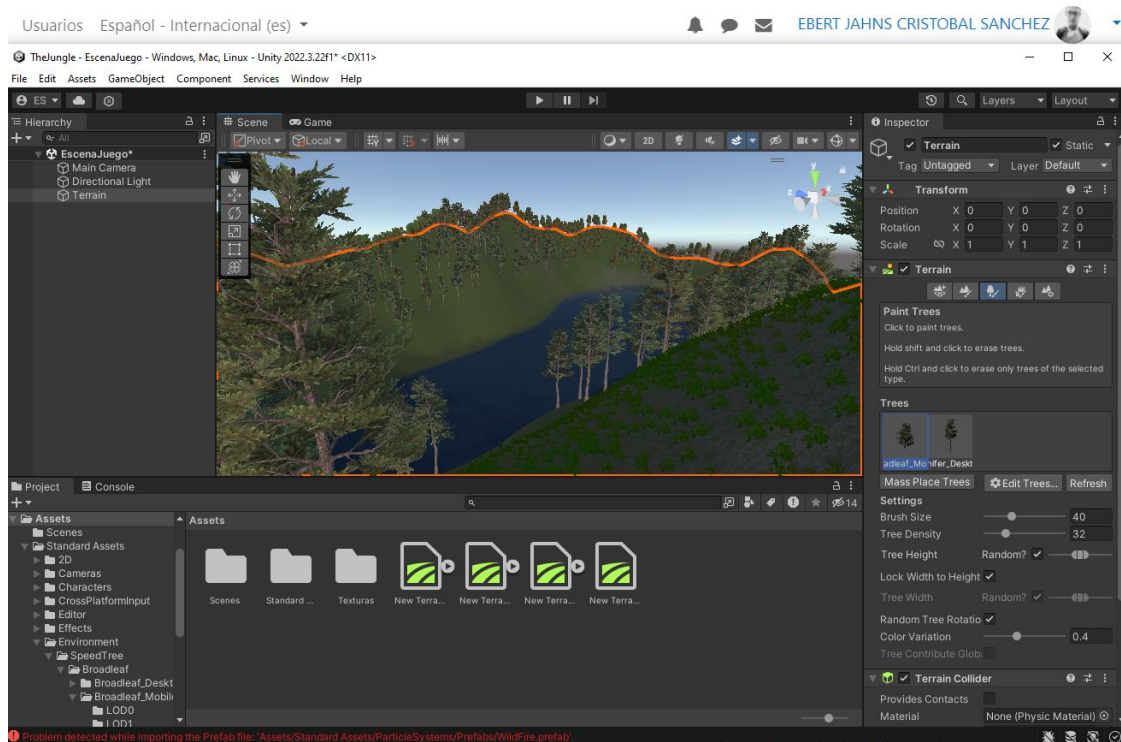
Se ha diseñado un terreno con al menos tres texturas, elevaciones de terrenos y se ha colocado un pequeño lago en medio de la colina.



Paso 2: Incluiremos **vegetación** de forma espaciada y armónica a lo largo del terreno, incluiremos al menos dos tipos de árboles y dos de hierba.

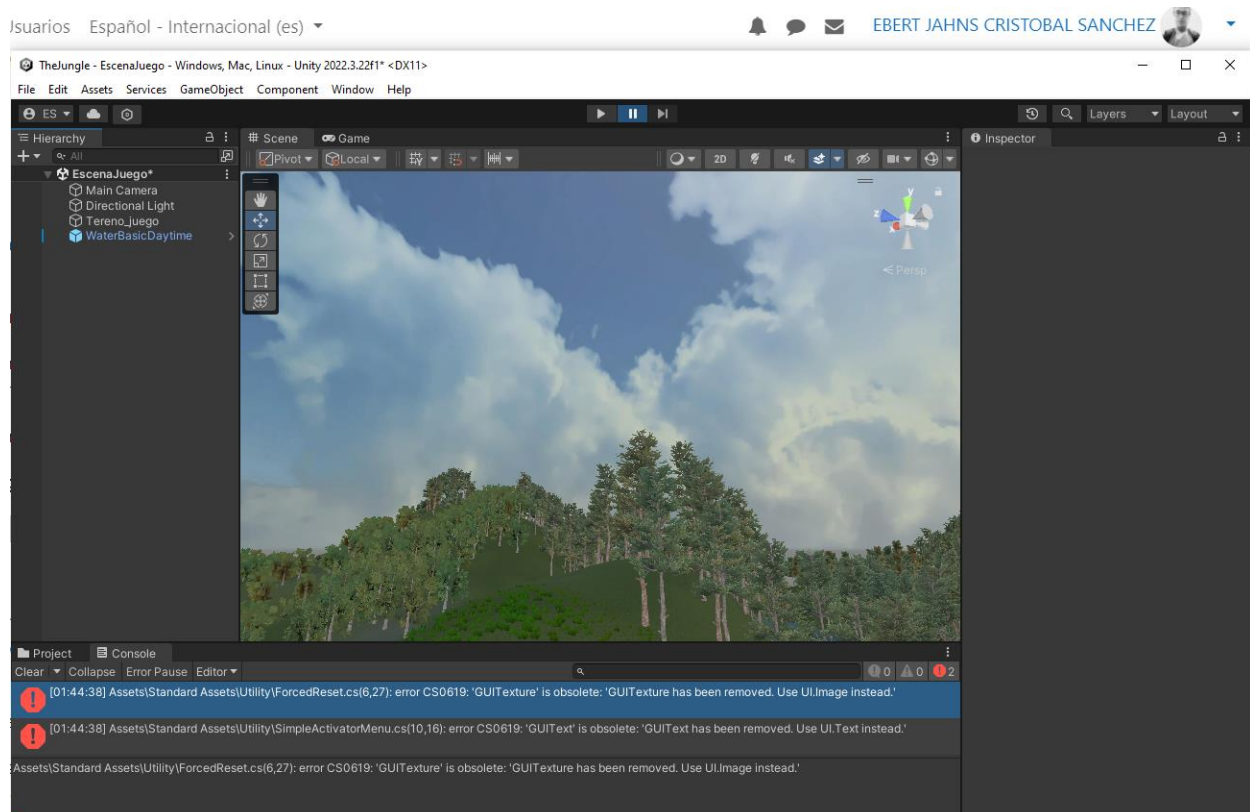
Se ha diseñado vegetación dos tipos de arboles y dos tipos de hierba.





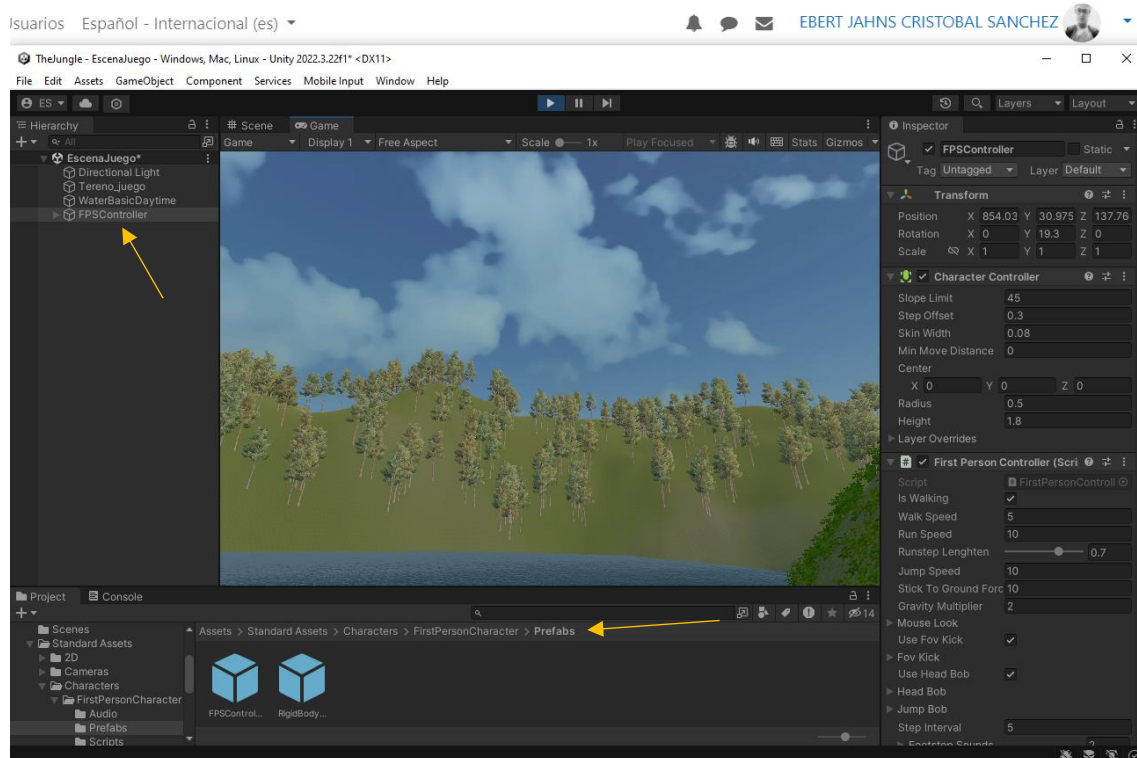
Paso 3: Para mejorar la iluminación incluiremos un cielo (skybox) y luz ambiental (directional light).

Se ha descargado **Classic Skybox** como indica en el tutorial de la unidad y se ha importado a unity y este es el resultado del cielo.

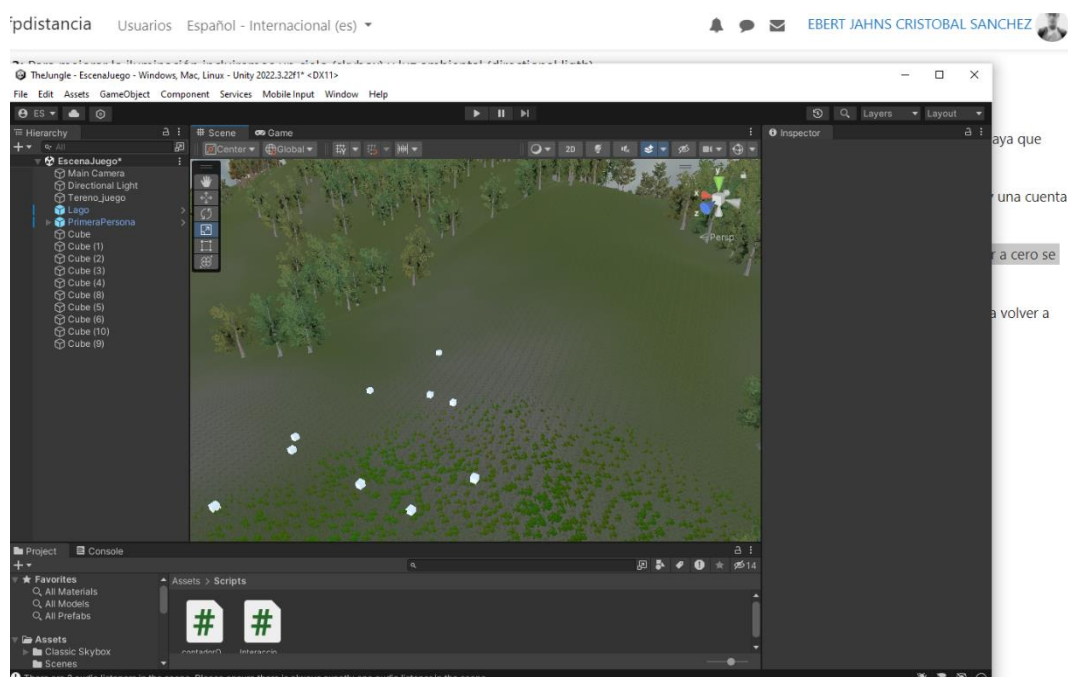


Paso 4: Incluiremos un controlador de forma que nos podremos mover libremente por la escena para su exploración.

Se ha incluido el controlador **FPSController**, este controlador nos permite entrar en el juego en primera persona y así poder explorar el lugar que hemos creado. Como no se puede demostrar por imagen el funcionamiento, he adjuntado un video en la carpeta del proyecto TheJungle.



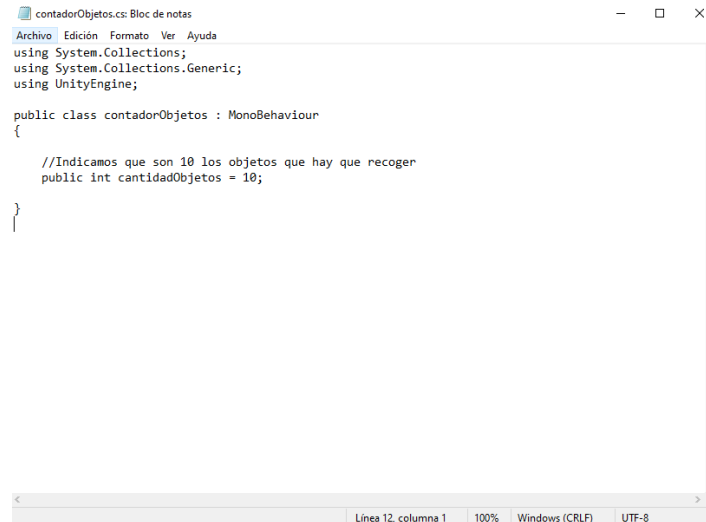
Paso 5: Inclusión de varios objetos 3D texturizados (ejemplo del cubo) a lo largo de la escena, entre 4 y 10 objetos iguales (ligeramente espaciados para que no haya que recorrer demasiado espacio).



Como se puede ver en la imagen anterior se han colocado 10 objetos. Y también se han creado los Scripts ContadorObjetos y InteraccionObjetos.

ContadorObejtos

El código define una clase llamada **contadorObjetos** en C# para Unity, que tiene una variable pública **cantidadObjetos** de tipo entero. Esta variable representa la cantidad de objetos que deben ser recolectados en el juego y se establece en 10 por defecto.

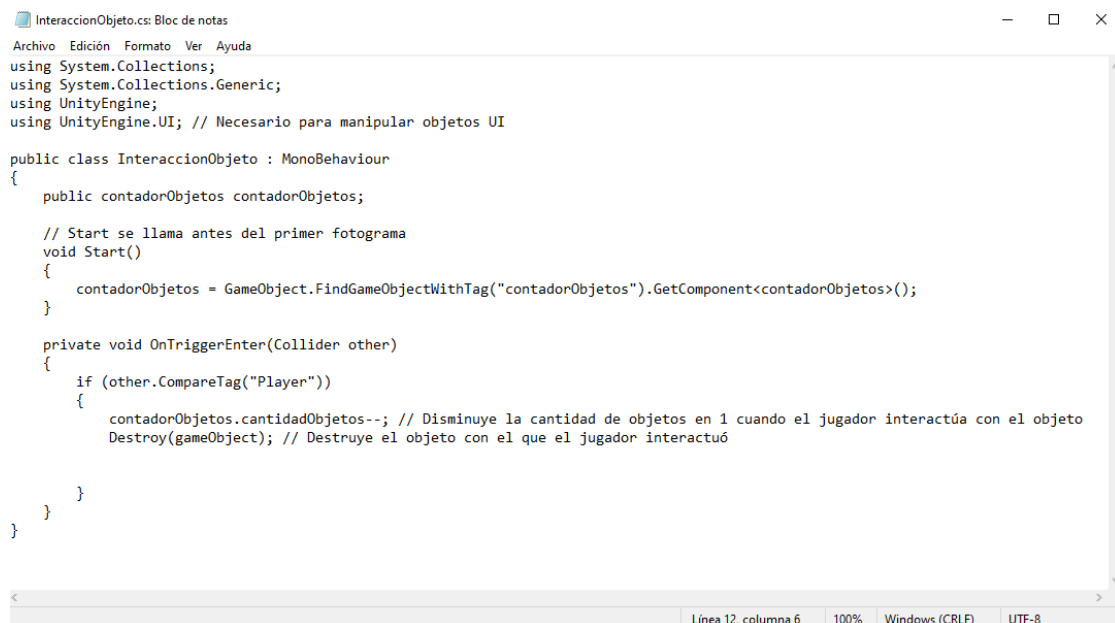


```
contadorObjetos.cs: Bloc de notas
Archivo Edición Formato Ver Ayuda
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class contadorObjetos : MonoBehaviour
{
    //Indicamos que son 10 los objetos que hay que recoger
    public int cantidadObjetos = 10;
}
```

InteraccionObjetos.

Este script llamado **InteraccionObjeto** en C# para Unity maneja la interacción del jugador con los objetos en el juego. Al comienzo (**Start()**), busca un GameObject con el tag "contadorObjetos" y obtiene el componente **contadorObjetos**, que rastrea la cantidad de objetos en la escena. Cuando un collider entra en contacto con el collider de este GameObject, si el collider tiene la etiqueta "Player", disminuye en uno la variable **cantidadObjetos** del contador de objetos y destruye el GameObject al que está adjunto el script.



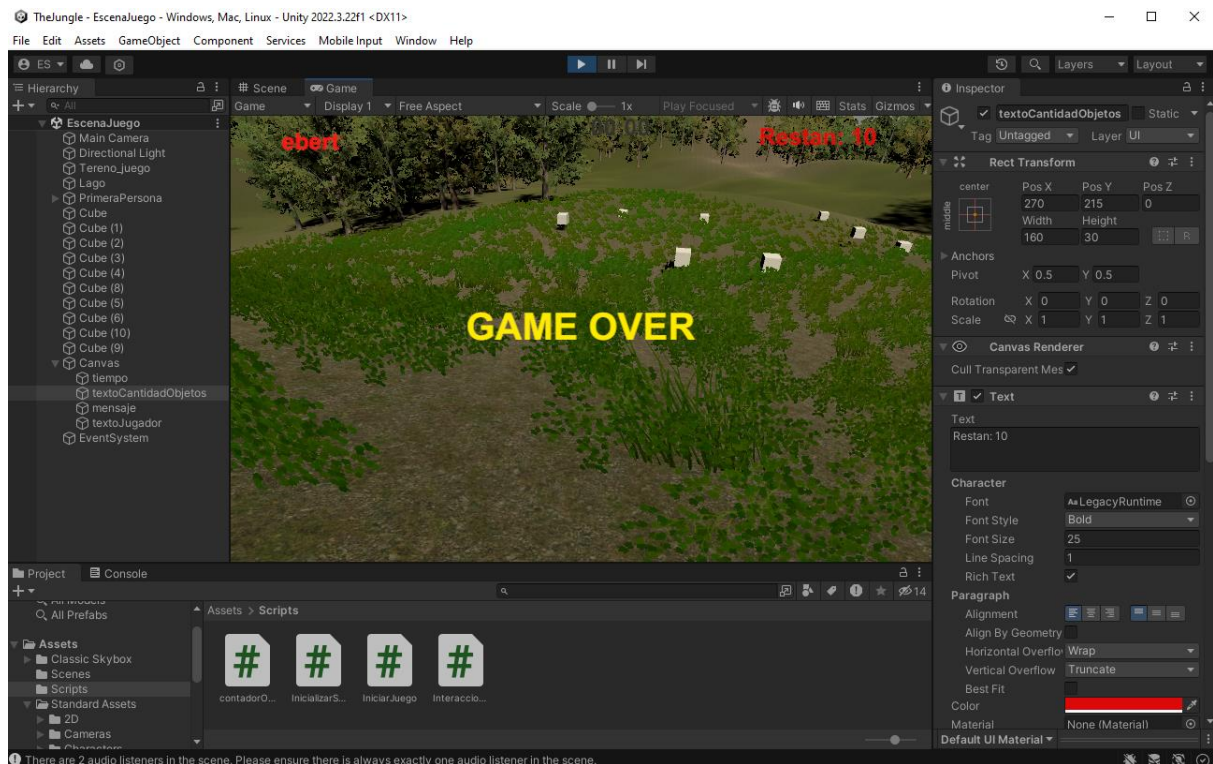
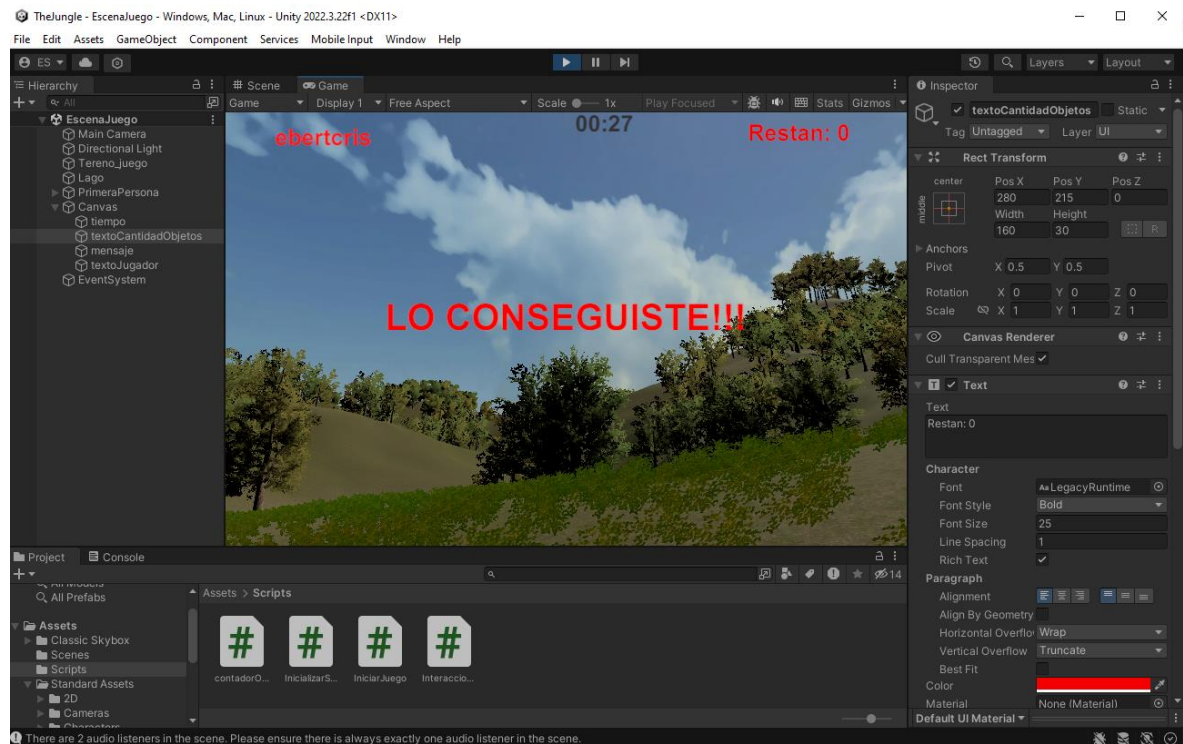
```
InteraccionObjeto.cs: Bloc de notas
Archivo Edición Formato Ver Ayuda
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; // Necesario para manipular objetos UI

public class InteraccionObjeto : MonoBehaviour
{
    public contadorObjetos contadorObjetos;

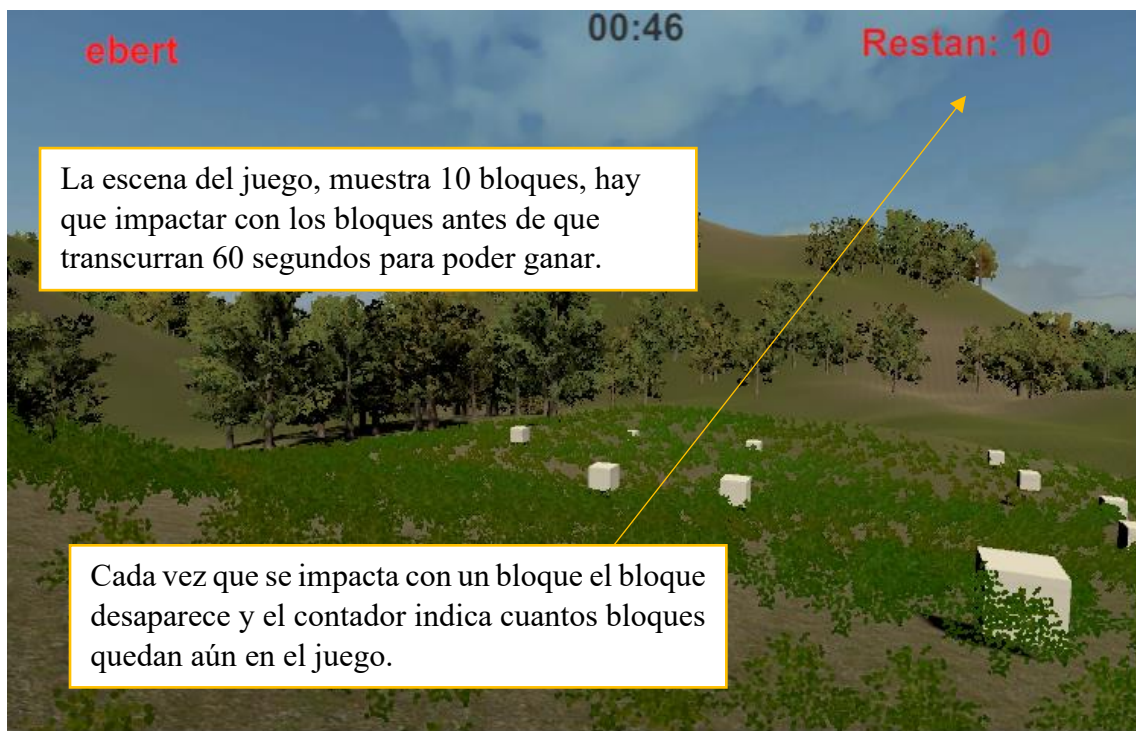
    // Start se llama antes del primer fotograma
    void Start()
    {
        contadorObjetos = GameObject.FindWithTag("contadorObjetos").GetComponent<contadorObjetos>();
    }

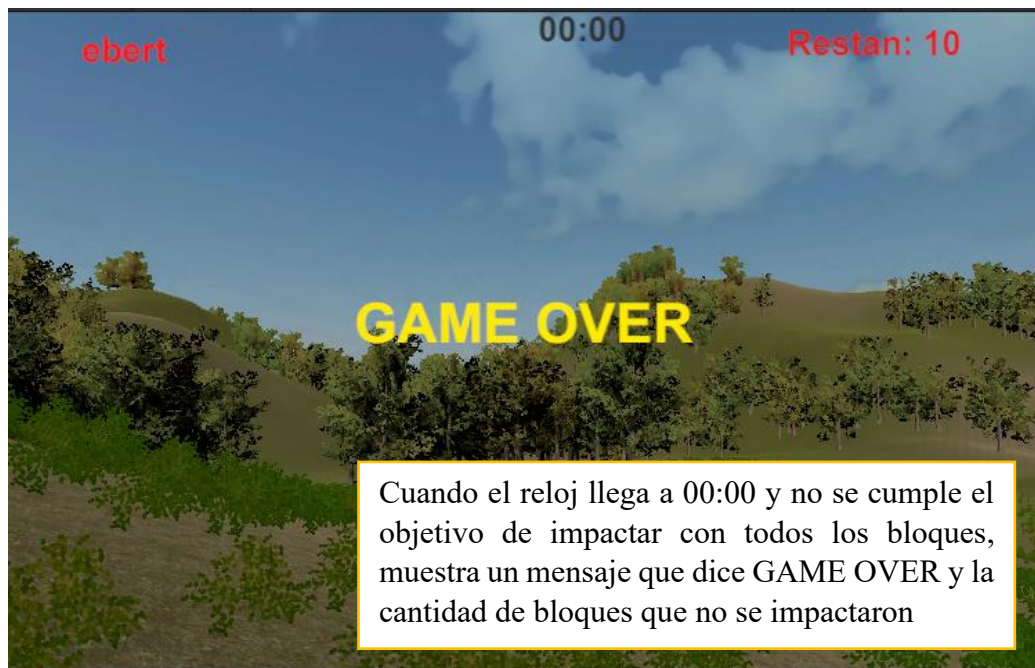
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            contadorObjetos.cantidadObjetos--; // Disminuye la cantidad de objetos en 1 cuando el jugador interactúa con el objeto
            Destroy(gameObject); // Destruye el objeto con el que el jugador interactuó
        }
    }
}
```

Paso 6: Se mostrará en la parte superior de la pantalla un **interfaz de usuario (UI)** donde se indicará el número de objetos en el terreno, el nombre del jugador y una cuenta atrás de 60 segundos (el tiempo será configurable desde el inspector).



Funcionalidad del juego:





Nota: para moverse se puede utilizar las teclas A, W, D y S como también las flechas del teclado y para mover la cámara solo hace falta mover el mouse

Inconvenientes de la actividad 2

Aunque he abordado todos los requisitos de la tarea, he enfrentado dificultades para mostrar el mensaje adecuado que indique si se desea volver a jugar al lograr el objetivo y cuando no se logra. Creo que es lo único que me ha costado y no he conseguido implementar.

Anexos Actividad 1

- Código bola.java

```
/**
 *
 * @author Ebert
 */
public class Bola {

    // Constante que representa la masa de la pelota
    public static final float MASA = 5f;

    // Constante que representa el diámetro de la pelota
    public static final float DIAMETRO = 0.216f;

    // Constante que representa la velocidad de la pelota
    public static final float VELOCIDAD = 12.0f;

}
```

- Código bolos.java

```
/**
 *
 * @author Ebert
 */
public class Bolos {

    //Separación entre los bolos
    private final static float SEPARACION = 0.305f;
    private final static float FILA_2 = -(7 + Bolos.SEPARACION);
    private final static float FILA_3 = -(7 + (Bolos.SEPARACION * 2));
    private final static float FILA_4 = -(7 + (Bolos.SEPARACION * 3));

    public final static Vector3f[] POSICION = {
        //Fila 1
        new Vector3f(0, 0, -7),
        //Fila 2
        new Vector3f((-Bolos.SEPARACION / 2), 0, Bolos.FILA_2),
        new Vector3f((Bolos.SEPARACION / 2), 0, Bolos.FILA_2),
        //Fila 3
        new Vector3f(-Bolos.SEPARACION, 0, Bolos.FILA_3),
        new Vector3f(0, 0, Bolos.FILA_3),
        new Vector3f(Bolos.SEPARACION, 0, Bolos.FILA_3),
        //Fila 4
        new Vector3f(-Bolos.SEPARACION * 1.5f, 0, Bolos.FILA_4),
        new Vector3f((-Bolos.SEPARACION / 2), 0, Bolos.FILA_4),
        new Vector3f((Bolos.SEPARACION / 2), 0, Bolos.FILA_4),
        new Vector3f(Bolos.SEPARACION * 1.5f, 0, Bolos.FILA_4)
    };

    // Constante que representa la masa de algún objeto en kilogramos.
    public final static float MASA = 1.6f;

    // Constante que representa la altura de algún objeto en metros.
    public final static float ALTURA = 0.38f;

    // Constante que representa el diámetro de algún objeto en metros.
    public final static float DIAMETRO = 0.12f;

}
```


- Código bolo.java

```
/**
 *
 * @author Ebert
 */
public class Bolo extends Geometry {

    /**
     * Constructor de la clase Bolo.
     *
     * @param assetManager El AssetManager para cargar recursos.
     */
    public Bolo(AssetManager assetManager) {

        // Llama al constructor de la clase Geometry para crear un cilindro con los
        // parámetros dados
        super("Cylinder", new Cylinder(32, 32, Bolos.DIAMETRO, Bolos.ALTURA, true));

        // Rota el cilindro creado para que esté orientado correctamente
        this.rotate((float) Math.PI / 2.0f, 0, 0);

        // Crea un nuevo material para el bolo
        Material material = new Material(assetManager,
            "Common/MatDefs/Light/Lighting.j3md");
        // Habilita el uso de colores del material
        material.setBoolean("UseMaterialColors", true);
        // Establece el color ambiental del material como verde
        material.setColor("Ambient", ColorRGBA.Green);
        // Establece el color difuso del material como verde
        material.setColor("Diffuse", ColorRGBA.Green);
        // Establece el color especular del material como amarillo
        material.setColor("Specular", ColorRGBA.Yellow);
        // Establece la cantidad de brillo del material
        material.setFloat("Shininess", 1);

        // Asigna el material creado al objeto Bolo
        this.setMaterial(material);
    }
}
```

- Código Main.java

```
package Actividad1;

import com.jme3.asset.AssetManager;
import com.jme3.material.Material;
import com.jme3.math.ColorRGBA;
import com.jme3.scene.Geometry;
import com.jme3.scene.shape.Cylinder;

import com.jme3.app.SimpleApplication;
import com.jme3.asset.TextureKey;
import com.jme3.audio.AudioData;
import com.jme3.audio.AudioNode;
import com.jme3.bullet.BulletAppState;
import com.jme3.bullet.control.RigidBodyControl;
import com.jme3.input.KeyInput;
import com.jme3.input.MouseInput;
import com.jme3.input.controls.ActionListener;
import com.jme3.input.controls.AnalogListener;
import com.jme3.input.controls.KeyTrigger;
import com.jme3.input.controls.MouseButtonTrigger;
import com.jme3.light.DirectionalLight;
import com.jme3.material.Material;
import com.jme3.math.ColorRGBA;
import com.jme3.math.Vector2f;
import com.jme3.math.Vector3f;
import com.jme3.renderer.RenderManager;
import com.jme3.renderer.queue.RenderQueue;
```

```

import com.jme3.renderqueue.RenderQueue.ShadowMode;
import com.jme3.scene.Geometry;
import com.jme3.scene.shape.Box;
import com.jme3.scene.shape.Sphere;
import com.jme3.shadow.DirectionalLightShadowRenderer;
import com.jme3.texture.Texture;
import com.jme3.texture.Texture.WrapMode;

/**
 *
 * @author Ebert
 */
public class Main extends SimpleApplication
    implements AnalogListener, ActionListener {

    // Estado de física de Bullet
    private BulletAppState bulletAppState;

    // Materiales para la bola y los bolos
    private Material bola_mat, piedras_mat;

    // Último tiempo de lanzamiento de la bola
    private long ultimoTiempoDeLanzamiento = 0;

    // Retraso entre lanzamientos de la bola en milisegundos
    private static final long RETRASO_DE_LANZAMIENTO = 3000;

    // Control de física para la bola
    private RigidBodyControl bola_fis;

    // Geometrias para la bola y los bolos
    private Geometry bola_geo;
    private Bolo[] bolo; // Colección de 10 bolos

    // Velocidad inicial de la bola
    private float velocidadDeLaBola;

    // Nodo de audio para reproducir cuando se lance la bola
    private AudioNode audioBola;

    // Método principal de ejecución del programa
    public static void main(String[] args) {
        Main app = new Main();
        app.start();
    }

    // Método de inicialización de la aplicación
    @Override
    public void simpleInitApp() {

        // Inicialización de BulletAppState
        bulletAppState = new BulletAppState();
        stateManager.attach(bulletAppState);

        // Velocidad de la bola obtenida de la clase Bola
        velocidadDeLaBola = Bola.VELOCIDAD;

        // Crear materiales
        crearMateriales();

        // Creamos el suelo
        crearSuelo();

        // Creamos la pared
        crearPared();

        // Crear la luz
        crearLuz();

        // Configuración de la cámara
        cam.setLocation(new Vector3f(0, 4f, 6f));
        cam.lookAt(new Vector3f(0, 2, 0), Vector3f.UNIT_Y);

        // Color de fondo
        viewPort.setBackgroundColor(new ColorRGBA(0f, 0f, 0.2f, 0));

        // Lanzamiento inicial de la bola
    }

```

```

    hazBola();

    //Crear los bolos
    crearBolos();

    //Implementamos las acciones
    initInput();

    //Establecemos los sonidos del juego
    setSonidos();
}

// Método para crear el suelo en la escena
private void crearSuelo() {

    // Crear una caja que represente el suelo con dimensiones 5x0.1x10
    Box suelo = new Box(Vector3f.ZERO, 5f, 0.1f, 10f);

    // Ajustar las coordenadas de textura para evitar deformaciones
    suelo.scaleTextureCoordinates(new Vector2f(6, 3));

    // Crear una geometría para representar el suelo
    Geometry suelo_geo = new Geometry("Floor", suelo);

    // Asignar el material del suelo
    suelo_geo.setMaterial(piedras_mat);

    // Establecer la posición local del suelo
    suelo_geo.setLocalTranslation(0, -0.1f, 0);

    // Adjuntar la geometría del suelo al grafo de la escena
    rootNode.attachChild(suelo_geo);

    // Crear un control físico para el suelo con masa 0 (estático)
    RigidBodyControl suelo_fis = new RigidBodyControl(0.0f);

    // Asociar el control físico a la geometría del suelo
    suelo_geo.addControl(suelo_fis);

    // Agregar el control físico del suelo al espacio de físicas gestionado por
Bullet    bulletAppState.getPhysicsSpace().add(suelo_fis);

    // Establecer el modo de sombra del suelo para recibir sombras
    suelo_geo.setShadowMode(ShadowMode.Receive);
}

// Método para crear la pared en la escena
private void crearPared() {
    // Crear una caja que represente la pared con dimensiones 5x0.1x10
    Box pared = new Box(Vector3f.ZERO, 5f, 0.1f, 10f);

    // Ajustar las coordenadas de textura para evitar deformaciones
    pared.scaleTextureCoordinates(new Vector2f(6, 3));

    // Crear una geometría para representar la pared
    Geometry pared_geo = new Geometry("Wall", pared);

    // Asignar el material de la pared
    pared_geo.setMaterial(piedras_mat);

    // Establecer la posición local de la pared
    pared_geo.setLocalTranslation(0, -0.1f, -10);

    // Rotar la pared 90 grados en el eje X para ponerla en posición vertical
    pared_geo.rotate((float) Math.PI / 2.0f, 0f, 0);

    // Adjuntar la geometría de la pared al grafo de la escena
    rootNode.attachChild(pared_geo);

    // Crear un control físico para la pared con masa 0 (estático)
    RigidBodyControl pared_fis = new RigidBodyControl(0.0f);

    // Asociar el control físico a la geometría de la pared
    pared_geo.addControl(pared_fis);
}

```



```

        // Agregar el control físico de la pared al espacio de físicas gestionado
por Bullet
        bulletAppState.getPhysicsSpace().add(pared_fis);

        // Establecer el modo de sombra de la pared para recibir sombras
pared_geo.setShadowMode(ShadowMode.Receive);
    }

    // Método para crear la luz en la escena
private void crearLuz() {
    // Crear una luz direccional
    DirectionalLight luz = new DirectionalLight();

    // Establecer el color y la intensidad de la luz
luz.setColor(ColorRGBA.White.mult(0.8f));

    // Establecer la dirección de la luz (proveniente de la parte superior
derecha detrás del jugador)
luz.setDirection(new Vector3f(-1, -1, -1).normalizeLocal());

    // Agregar la luz al grafo de la escena
rootNode.addLight(luz);

    // Configurar sombras para la luz direccional
DirectionalLightShadowRenderer dlsr = new
DirectionalLightShadowRenderer(assetManager, 1024, 3);
dlsr.setLight(luz);
viewport.addProcessor(dlsr);
}

    // Método para crear los materiales utilizados en la escena
private void crearMateriales() {
    // Material para el suelo
    piedras_mat = new Material(assetManager,
"Common/MatDefs/Misc/Unshaded.j3md");
    // Cargar una textura predefinida para el suelo y hacerla repetir por su
superficie
    TextureKey key = new TextureKey("Textures/Terrain/Pond/Pond.jpg");
    key.setGenerateMips(true);
    Texture textura = assetManager.loadTexture(key);
    textura.setWrap(WrapMode.Repeat);
    piedras_mat.setTexture("ColorMap", textura);

    // Material para la bola
    bola_mat = new Material(assetManager, "Common/MatDefs/Light/Lighting.j3md");
    // Configurar el material de la bola
    bola_mat.setBoolean("UseMaterialColors", true);
    bola_mat.setColor("Ambient", ColorRGBA.Cyan);
    bola_mat.setColor("Diffuse", ColorRGBA.Cyan);
    bola_mat.setColor("Specular", ColorRGBA.White);
    bola_mat.setFloat("Shininess", 1);
}

    public void hazBola() {
        // Verificar si ya existe una bola y eliminarla antes de crear una nueva
        if (bola_geo != null && rootNode.hasChild(bola_geo)) {
            rootNode.detachChild(bola_geo);
            bulletAppState.getPhysicsSpace().remove(bola_fis);
        }

        // Crear una esfera de 40 centímetros de diámetro
        Sphere esfera = new Sphere(32, 32, Bola.DIAMETRO);

        // Asociar la forma a una geometría nueva
        bola_geo = new Geometry("bola", esfera);
        // asignarle el material
        bola_geo.setMaterial(bola_mat);

        // añadirla al grafo de escena
        rootNode.attachChild(bola_geo);

        // la colocamos en la posición de la cámara
        bola_geo.setLocalTranslation(cam.getLocation());

        // Creamos el objeto de control físico asociado a la bola con un peso
        // de 1Kg.
        bola_fis = new RigidBodyControl(Bola.MASA);
    }

```

```

        // Asociar la geometría de la bola al control físico
        bola_geo.addControl(bola_fis);
        // Añadirla al motor de física
        bulletAppState.getPhysicsSpace().add(bola_fis);

        //La bola emite sombra
        bola_geo.setShadowMode(RenderQueue.ShadowMode.Cast);
    }

    /**
     * añadir las acciones al inputManager
     */
    private void initInput() {

        //Crear la acción "Lanzar"
        inputManager.addMapping("Lanzar",
            new KeyTrigger(KeyInput.KEY_SPACE),
            new MouseButtonTrigger(MouseInput.BUTTON_LEFT));

        //Creamos la acción "DesactivarGravedad" para desactivar la gravedad
        inputManager.addMapping("DesactivarGravedad", new
        KeyTrigger(KeyInput.KEY_G));

        //Añadimos el mapping a la escucha
        //Se puede procesar en los métodos onAnalog y onAction
        //todo depende de qué método implemente la acción
        inputManager.addListener(this, "Lanzar");
        inputManager.addListener(this, "DesactivarGravedad");
    }

    /**
     * Método que implementa los sonidos del juego.
     */
    private void setSonidos() {
        //Añadimos un sonido de ambiente al juego
        AudioNode audio = new AudioNode(assetManager,
            "Sound/Environment/River.ogg", AudioData.DataType.Buffer);
        audio.setVolume(0.3f);
        audio.setPositional(false);
        audio.setLooping(true);
        audio.play();
        rootNode.attachChild(audio);

        //Sonido al lanzar la bola
        audioBola = new AudioNode(assetManager,
            "Sound/Effects/Gun.wav", AudioData.DataType.Buffer);
        audioBola.setVolume(15);
        rootNode.attachChild(audioBola);

        bola_geo.setShadowMode(ShadowMode.CastAndReceive);
    }

    public void simpleUpdate(float tpf) {
        //TODO: add update code
    }

    public void simpleRender(RenderManager rm) {
        //TODO: add render code
    }

    // Método para crear los bolos en la escena
    private void crearBolos() {
        // Inicializar el arreglo de bolos y de controles físicos de los bolos
        bolo = new Bolo[10];
        RigidBodyControl[] bolo_physc = new RigidBodyControl[10];

        // Iterar para crear y configurar cada bolo
        for (int i = 0; i < 10; i++) {
            // Crear un nuevo objeto Bolo
            bolo[i] = new Bolo(assetManager);

            // Agregar el bolo al grafo de la escena
            rootNode.attachChild(bolo[i]);

            // Colocar el bolo en su posición
            bolo[i].setLocalTranslation(Bolos.POSICION[i]);
        }
    }

```

```

        // Crear un control físico para el bolo con la masa especificada en la
clase Bolos bolo_physc[i] = new RigidBodyControl(Bolos.MASA);
        // Agregar el control físico al bolo
        bolo[i].addControl(bolo_physc[i]);
        // Agregar el control físico al espacio de físicas gestionado por Bullet
        bulletAppState.getPhysicsSpace().add(bolo_physc[i]);

        // Configurar el modo de sombra de los bolos para que emitan y reciban
sombbras        bolo[i].setShadowMode(ShadowMode.CastAndReceive);
    }
}

/**
 * Método para procesar las acciones.
 *
 * @param accion nombre de la acción
 * @param value valor dado por la presión de la tecla
 * @param tpf tiempo en segundos desde la última actualización
 */
public void onAnalog(String accion, float value, float tpf) {

    if (accion.equals("Lanzar")) {
        long tiempoActual = System.currentTimeMillis();
        if (tiempoActual - ultimoTiempoDeLanzamiento >= RETRASO_DE_LANZAMIENTO)
        {
            ultimoTiempoDeLanzamiento = tiempoActual;
            audioBola.play();

            float velocidadLanzamiento = 18f; // Velocidad de lanzamiento
deseada
        }
        bola_fis.setLinearVelocity(cam.getDirection().mult(velocidadLanzamiento));
    }
}

public void onAction(String accion, boolean pulsado, float tpf) {
    if (accion.equals("Lanzar") && pulsado) {
        // Crear una nueva bola cuando se active la acción "Lanzar"
        hazBola();
    } else if (accion.equals("DesactivarGravedad") && pulsado) {
        if (bulletAppState.isEnabled()) {
            bulletAppState.setEnabled(false);
            System.out.println("Gravedad desactivada");
        } else {
            bulletAppState.setEnabled(true);
            System.out.println("Gravedad activada");
        }
    }
}
}
}

```