Universidad ORT Uruguay

Facultad de Ingeniería Bernard Wand Polak
Entregado como requisito para la obtención del crédito
de la materia Diseño de Aplicaciones 1

Estudiantes:
Sebastian Cuneo - 267584
Gonzalo Gularte - 270959

Docente: Nicolas Elizabelar
Tecnologías: Sebastian Sebastián Bañales y Guzman Viglieca

Junio 2023

Índice

1) Link al repositorio de la entrega: https://github.com/ORT-DA1-2023/267584_270959

Comentario sobre commits en el repositorio: Al igual que en la entrega anterior los commits en el repositorio de nuestro obligatorios fueron hechos por un integrante del grupo, ya que tuvimos varios problemas por trabajar con Windows y Mac. Para este segundo obligatorio intentamos descargar una máquina virtual (Oracle VM y Parallels) para poder correr correctamente Visual Studio y Microsoft SQL Server Express 2019, pero nos topamos con muchos errores y problemas para instalar por falta de archivos de Windows o funciones premium que ofrece las VM. Aun así nunca dejamos de trabajar juntos comunicándonos ya sea vía discord, juntándose en las casas o de manera presencial en la facultad.

2) Descripción general del trabajo y del sistema:

Este obligatorio se centra en la creación de figuras utilizando técnicas de “3D renderer con ray rendering”. El objetivo principal del proyecto es ayudar a un matemático que tuvo problemas con su startup cuando los clientes empezaron a aumentar y es así cómo se decide diseñar un sistema que permita a los usuarios crear figuras tridimensionales y visualizarlas utilizando ray tracing para generar imágenes realistas y detalladas. El sistema ha sido diseñado para que sea fácil de usar y accesible para los usuarios, a continuación, se proporcionará una descripción más detallada del trabajo y del sistema, incluyendo información sobre las funcionalidades implementadas y los errores conocidos. Los requerimientos funcionales se pueden dividir en 3 grupos; *gestión de usuarios*, la *creación de figuras, materiales, modelos y escenas* y finalmente el *motor gráfico* el cual se encarga de renderizar las escenas creadas.

Empezaremos mencionando las funcionalidades que se presentan en el grupo “*Gestion de usuarios*”: se define al “usuario” como “cliente” el cual va a ser capaz de registrarse, loguearse, crear sus propias escenas y a su vez renderizarlas. De esta manera el cliente puede tener un entorno único y personalizado para crear sus figuras, modelos, materiales y escenas.

A continuación seguiremos mencionando las funcionalidades que se presentan en “*creación de figuras, materiales, modelos y escenas*”: el cliente podrá configurar elementos a través de la user interface y así será posible la creación de escenas. Cada objeto que cree dicho cliente es visible y modificable por el mismo, lo que significa que no comparten datos entre clientes. Haremos una subdivisión y explicaremos las funcionalidades de “figura”, “material”, “modelo” y “escena” por separado para poder así tener una idea más detallada de cada uno de ellos. Comenzaremos por “figura”; la cual se define como *“una entidad geométrica que describe la forma de un objeto”*, en este caso la única figura que se usará será la esfera, el cliente podrá crear una figura dirigiéndose a la ventana correspondiente y agregando una “nueva figura” en la cual se le asignará un nombre y un radio, así como el propietario de dicha figura (el cliente). Seguiremos con “Material” que se puede definir como *“lo que describe como se ve una figura”* contamos con dos tipos de material; el lambertiano el cual se caracteriza por su color opaco y

la capacidad de reflejar la luz la cual le dará un efecto sombra a la esfera y el material metálico el cual se caracteriza por ser reflectivo, contar con un color y un difuminado propio. Al igual que en figura al material se le asigna un nombre, un color y un propietario y para el material metálico se le agrega un difuminado. El “Modelo” se define como *“una figura con un material asignado”*, un material se le asigna un propietario, un nombre, una figura, un material y por último un “preview” el cual el sistema debe generar si el cliente lo desea. Por último tenemos “escena” la cual es una *“composición de modelos posicionados (modelo que cuenta con una posición (x,y,z)) en el espacio”*, al modelo se le asigna un propietario, un nombre, una fecha de creación, la lista de modelos posicionados, la fecha desde la última modificación, la fecha desde la última renderización, la posición de la cámara, la posición del objetivo, el campo de visión, el desenfoque de la cámara (el cual se obtiene a partir de los datos de distancia focal y la apertura del lente) y por último un preview.

A partir de lo mencionado anteriormente el cliente es capaz de usar un entorno intuitivo para la creación de figuras personalizadas desde el nombre, color y tamaño hasta la ubicación de dicha figura en un espacio.
Para finalizar mencionaremos al “motor gráfico” el cual recibe una escena y ejecuta el renderizado de la imagen de la escena, el motor gráfico funciona a partir de scripts JS que describen las operaciones necesarias para renderizar una escena, fue necesario traducir dichos scripts a C# para que funcione en nuestro obligatorio. El motor gráfico cuenta con los siguientes datos: resolución, muestreos por pixel y profundidad máxima. Al renderizar una escena (sea un preview o una escena creada) el motor gráfico es capaz de producir dicha imagen y la devuelve como una lista de valores RGB en formato PPM, PNG o JPG según cómo elija el usuario en la pantalla donde se visualiza la escena.

3) Descripcion y justificacion de diseño:

Cuando hablamos sobre el diseño del obligatorio nos referimos a cómo está organizado y qué decisiones fueron tomadas para que este esté organizado y sea mucho más agradable y fácil de entender para alguien que no fue el creador. Para justificar el diseño y las decisiones tomadas nos apoyaremos en el “diagrama de paquetes” el cual nos muestra la organización general de nuestra aplicación, a su vez usaremos el diagrama de clases el cual nos da una vista “de arriba” de cómo está organizado nuestro obligatorio; mostrando así las distintas clases, los métodos y atributos de las mismas y qué relaciones tienen unas con las otras.

La aplicación ha sido diseñada con el propósito de proporcionar una interfaz de usuario fácil de usar para interactuar con un conjunto de funcionalidades especificadas anteriormente. Para lograr esto, se ha organizado la aplicación en diferentes paquetes que agrupan funcionalidades similares. A continuación, se describe cada uno de estos paquetes.

- Domain
- Repositories
- Divers
- User interface (UI)

3a) Diagrama de paquetes (Anexo - 1):

Como se muestra en el anexo 1 en nuestro obligatorio contamos con 4 “namespaces” (paquetes); estos agrupan clases que hacen posible que el proyecto compile y corra correctamente y las mismas contienen el código con las funcionalidades requeridas, los repositorios, los conductores y la inferzas de usuario. En nuestro diagrama se pueden notar las dependencias entre los paquetes (Anexo - 2). Esto ocurre cuando un paquete depende de otro, o sea “lo usa”, en el caso de arriba “UI” está usando a “Domain” (Anexo - 3), y esto ocurre a lo largo del proyecto por ejemplo; con Drivers que usa Repositories, Drivers que usa a Domain, UI que usa a Drivers, Repositories que usa a Domain.

Ahora nos detendremos en cada paquete para poder explicar las responsabilidades de cada uno, y así entender mejor qué tipo de funcionalidades agrupan.

- Domain (Anexo - 4):

  El paquete dominio es uno de los paquetes más importantes en nuestro obligatorio, este se encarga de implementar la lógica del programa y las reglas del dominio. En el dominio se encuentran las clases del programa con sus respectivos constructores y métodos, así como en algunos casos también encontramos con las validaciones que requiere el sistema. A su vez el motor gráfico como los scripts del matemático también se encuentran dentro del dominio. Dentro del Dominio nos encontramos con la carpeta “Exceptions” que dentro contiene clases de excepción personalizadas (Las excepciones se utilizan en la programación para manejar errores y situaciones excepcionales que pueden surgir durante la ejecución del código). Estás clases están diseñadas para ser ejecutadas cuando se detecta un error en los inputs del usuario. Las clases heredan de la clase base "Exception", que es una clase predeterminada en C# para definir excepciones. Utiliza constructores sin parámetro para cuando no se desea dar un mensaje, y a su vez hay constructores que nos permitieron crear una instancia de la excepción (se utiliza para proporcionar más información sobre el origen del error) y proporcionar un mensaje que describa el error que se ha producido. Por último, hay un constructor protegido que se utiliza para la serialización de la excepción.

- Divers (Anexo - 5):

  El paquete drivers es el intermediario entre el paquete Repositories y la interfaz, este paquete facilita la conexión con la base de datos Microsoft SQL Server Express 2019. Su función es llamar a los métodos de la clase Repositories. (Anexo - 8)

- Repositories (Anexo - 6):

  El paquete repositories es el encargado de guardar la información del sistema, en él se encuentran los métodos de agregación, modificación eliminación y

obtención para cada clase. A su vez dentro del paquete repositories se encuentran las interfaces vinculadas a cada clase, estas llaman a los métodos de dichas clases.

- User interface (UI) (Anexo - 7):

  El paquete UI se enfoca en la interacción entre el usuario y la aplicación. En la UI se encuentran todos los diseños de ventanas que el usuario visualizará y usará a la hora de ejecutar el proyecto. También la clase UI se encarga de captar todos los errores que el usuario pueda cometer al ingresar datos, esto es posible ya que en el código de cada ventana a la hora de agregar o crear un elemento hay un catch que se encarga de "atrapar" el error y así llamar a la excepción correspondiente, entonces se podría decir que la UI es responsable también de recopilar información que ingresa el usuario, validar dicha información y en los casos que sea necesario comunicarse con otros paquetes.

3b) Diagrama de clases (Anexo - 9):

En el Anexo - 9.1 se muestra parte del UML completo de nuestro proyecto, por motivos de comodidad se decidió dividir el UML por partes, en el anexo anterior se muestran solo las clases y las relaciones entre ellas, iremos por "porciones" añadiendo las clases completas con atributos y métodos así como sus relaciones. Para tener un orden lógico empezaremos por la descripción de la clase más importante, y luego continuaremos con las demás clases en orden de importancia.

- Cliente (Anexo - 10): La clase Cliente, cuenta con atributos de username, password, la fecha de registro y las respectivas colecciones para figura, material, modelo y escena. Los métodos de dicha clase se encuentran en el repositorio correspondiente. Notar que Figura, Modelo, Escena y Material dependen del cliente mediante la relación de agregación. También notar que exista una asociación entre el cliente y su repositorio, en el cual se guarda toda la información, y donde se encuentran los métodos de agregación y obtención. También podemos observar la interfaz y el manejador de cliente.

- Figura (Anexo - 11): La clase Figura cuenta con atributos del dueño de la figura, Id del dueño de la figura, el nombre y el radio. Los métodos de dicha clase se encuentran en el repositorio correspondiente, aunque podemos encontrar 2 métodos de validación para el nombre de la figura y el radio de la figura. Notar que exista una asociación entre la figura y su repositorio, en el cual se guarda toda la información, y donde se encuentran los métodos de agregación, eliminación y obtención. También podemos observar la interfaz y el manejador de figura.

- Material (Anexo - 12): La clase Material cuenta con atributos del dueño de la figura, el Id del dueño del material, el nombre del material y el color del material. Los métodos de dicha clase se encuentran en el repositorio correspondiente, aunque podemos encontrar

un método de validación para el nombre. Existen las clases “Lambertian Material” y “Metallic Material” que heredan de la clase material. Notar que exista una asociación entre el Material y su repositorio, en el cual se guarda toda la información, y donde se encuentran los métodos de agregación, eliminación y obtención. También podemos observar la interfaz y el manejador de Material.

    - Metallic Material: cuenta con el atributo de difuminado perteneciente únicamente al tipo de material metálico. También nos encontramos con un método de validación para el difuminado.

- Modelo (Anexo - 13): La clase Modelo cuenta con atributos del dueño del modelo, el Id del dueño del modelo, el nombre del modelo, el nombre de la figura del modelo, el Id del dueño de la figura, la figura, el nombre del material, el Id del dueño del material, el material y un preview. Los métodos de dicha clase se encuentran en el repositorio correspondiente, a su vez podemos encontrar un método de validación para el nombre. Notar que exista una asociación entre el Modelo y su repositorio, en el cual se guarda toda la información, y donde se encuentran los métodos de agregación, eliminación y obtención. También podemos observar la interfaz y el manejador de Modelo.

- Modelo Posicionado (Anexo - 14): La clase Modelo Posicionado cuenta con atributos de id, la posición de x, la posición de y, la posición de z, el nombre del modelo, el Id del dueño del modelo y el modelo por el cual está compuesto el modelo posicionado. Notar que existe una composición entre el Modelo Posicionado y la Escena y otra composición entre Modelo Posicionado y Modelo.

- Escena (Anexo - 15): La clase Escena cuenta con atributos del dueño de la escena, el Id del dueño de la escena, el nombre de la escena, la fecha de creación, la fecha de modificación, la última vez que se renderizo, los distintos puntos de vista para X,Y,Z para look at y para look from, el fov que hace referencia al campo de visión, el path para el preview, el radio del lente, el difuminado y la colección de los modelos posicionados. Los métodos de dicha clase se encuentran en el repositorio correspondiente, aunque podemos encontrar un método de validación para el nombre. Notar que exista una asociación entre la escena y su repositorio, en el cual se guarda toda la información, y donde se encuentran los métodos de agregación, modificación, eliminación y obtención. También podemos observar la interfaz y el manejador de la escena. Escena implementa la clase de motor gráfico que se explicará a continuación

- Motor Gráfico (Anexo - 16): La case Motor gráfico está compuesta por distintas clases que se analizan a continuación:
    - Cámara: La clase cámara tiene como métodos: origin, cornerLowerLeft, horizontal, vertical y el lensRadius
    - HitRecord: La clase HitRecord tiene como métodos: t, intersection, normal, attenuation, material y roughness

    - Sphere: La clase Sphere tiene como métodos: center, radius, attenuation, material y roughness

- Motor Gráfico: Clase con atributo random Provider, agregación de la clase randomProvider. Tiene los siguientes métodos: SavePixel, ShootRay, IsShereHit,GetRandomUnitShere, LambertianScatter, MetalicScatter y Reflect.
- Random Provider: Clase randomProvider con atributo random. Cuenta con el método getRaondom().
- Vector 3D: Clase Vector3D que tiene como atributos los valores de x, y, z y como métodos tiene: Add, Subtract, Multiply, Divide, AddTo, SubstractFrom,ScaleUpBy, ScaleDownBy, GetUnit, Dot, Cross, Length, SqueredLenght.
- Ray: La clase Ray tiene como atributos origin y direction. Y cuenta con los métodos de PointAt.

- Repository Context (Anexo - 28): Esta clase representa una base de datos “virtual” de objetos. Cada vez que queremos interactuar con la base de datos lo hacemos a través de esta clase, y luego Entity Framework se encarga de realizar el “mapeo” y modificar la base de datos real

3c) Diagramas de Interacción:

Se crean tres diagramas de interacción (secuencia) enfocados en los procesos de: registrar un cliente y que este sea guardado en la base de datos del sistema y que por ende persista en el sistema. Se crea el diagrama que muestra como es el proceso de crear una redirección de un modelo en el cual fue seleccionado realizar su respectivo preview y que usara el motor grafico y todas sus respectivas clases y métodos y por último se crea el diagrama de los dos tipos de materiales distintos.

1. Funcionalidad: Cliente en base de datos
2. Funcionalidad: Render de modelo
3. Funcionalidad: Tipos de materiales

3d) Modelo de Tablas:

Se adjuntan los modelos de tablas así como los scripts con y sin datos de prueba en la entrega de la documentación.

3e) Mecanismos generales y descripción de las principales decisiones de diseño tomadas:

Los mecanismos generales de nuestro proyecto están compuestos por una serie de elementos que trabajan de forma coordinada para lograr un objetivo específico. Cada uno de estos mecanismos está diseñado para cumplir una función específica y es crucial para el correcto funcionamiento del sistema en su conjunto.

En primer lugar, la interfaz de usuario es la forma en que los usuarios interactúan con el sistema. Esta interfaz es intuitiva y fácil de usar para que el usuario pueda realizar las tareas de forma eficiente. Para lograr esto, se consideraron factores como la usabilidad, la accesibilidad y la estética. Además, se diseñaron las ventanas y los elementos de la interfaz de usuario de tal manera que se ajusten a las necesidades del usuario y sean fáciles de entender y utilizar. En

este sentido, es importante tener en cuenta el diseño centrado en el usuario, que se enfoca en las necesidades y expectativas del usuario final.

En segundo lugar, el almacenamiento de datos es otro mecanismo importante en nuestro proyecto. Fue necesario diseñar una estructura de datos adecuada para que la información pueda ser almacenada y recuperada de forma eficiente. La elección fue crear una clase “repositorio”, donde el sistema encuentre los métodos de agregación, modificación eliminación y obtención para cada clase. Se agrega una base de datos (Microsoft SQL Server Express 2019) que permite que haya persistencia en los datos, esto permite que la siguiente vez que se ejecute la aplicación se comenzará con dichos datos cargados con el último estado guardado antes de cerrar la aplicación.
En tercer lugar, el manejo de errores y excepciones es fundamental para garantizar la fiabilidad del sistema. En nuestro proyecto se creó una carpeta en el paquete dominio llamada “Exceptions” la cual contiene clases que contienen excepciones para que cuando sean llamadas estas alerten al usuario. A su vez se utilizó “try/catch” para atrapar los errores y así poder llamar a las excepciones. Otro método que decidimos utilizar fue el “KeyPress” el cual verifica cuando se presiona una tecla, por ejemplo si en una caja de texto solo pueden haber números, también queremos que solo haya una coma, y asegurarnos de que la coma no sea el primer carácter, realizamos el código correspondiente para que el método reconozca cuando se teclea algo que no cumple con dichas condiciones y no deje al usuario seguir escribiendo, para así poder evitar errores de escritura. (Anexo - 17).

El diseño del proyecto se asegura de que en la posibilidad de que ocurran errores y excepciones, el mismo cuente con un “plan de contingencia” para hacer frente a estas situaciones. Es importante que el sistema tenga un mecanismo para detectar y notificar los errores, así como para corregirlos o tomar medidas para mitigar sus efectos.

Para el proyecto, aplicamos la programación orientada a objetos (POO) como metodología de programación. La POO nos permite diseñar el sistema en términos de objetos, encapsulando tanto los datos como el comportamiento del sistema, esto nos facilitó el desarrollo del proyecto al permitirnos centrarnos en el diseño de los objetos y sus interacciones en lugar de tener que preocuparnos por detalles técnicos.

La principal decisión de diseño que tomamos fue la creación de la jerarquía de las clases que representarán las diferentes entidades del sistema. Cada una de estas clases tenía atributos específicos y métodos que permitían manipularlos, y estaban interconectados de tal manera que reflejaban la lógica del sistema. Además, utilizamos patrones de diseño como el patrón de métodos abstractos para crear objetos de diferentes tipos de forma dinámica. Y se implementó el concepto de polimorfismo en las clases Metallic Material y Lambertian Material que heredan ambas de la clase material.

SOLID:
En el diseño del software, se aplicaron parcialmente los principios de SOLID. Se hizo un esfuerzo por seguir el principio de responsabilidad única y el principio de abierto/cerrado,

adaptando el código para que las clases tuvieran una única responsabilidad y fueran extensibles sin modificarlas directamente. Además, se implementaron interfaces cohesivas y se realizó una inversión parcial de dependencia en ciertos módulos, lo que permitió una mayor flexibilidad y reutilización del código. Aunque no se cumplió en su totalidad el principio de sustitución de Liskov, se hizo lo posible para mantener la integridad del programa al reemplazar objetos de una clase derivada por objetos de su clase base.

GRASP:
En la arquitectura del software, se aplicaron de manera parcial los patrones de GRASP. El controlador asumió la responsabilidad de manejar las solicitudes de entrada y coordinar las acciones entre los objetos, mejorando así la estructura y modularidad del sistema. Además, se utilizó el patrón Creator en algunos casos para la creación de objetos, facilitando la creación y gestión de instancias. También se asignaron responsabilidades a los objetos según el principio de Information Expert, asegurando que cada objeto tuviera la información necesaria para cumplir con sus funciones. Aunque el acoplamiento no se mantuvo en su nivel más bajo en todos los casos, se logró un nivel aceptable de bajo acoplamiento entre los objetos en general.

Se utilizó EF (Entity Framework) para lograr la persistencia de datos solicitada en la parte dos del obligatorio, a partir de esto se realizó la siguiente reflexión sobre el cambio hacia otra implementación concreta de EF: El cambio hacia EF para poder lograr la persistencia en nuestro proyecto ofrece varias ventajas. Al implementar EF, se introduce una capa de persistencia que permite almacenar y recuperar datos de manera eficiente desde una base de datos. Esto mejora la estructura del programa y facilita el mantenimiento. EF proporciona una interfaz sencilla y orientada a objetos para interactuar con la base de datos, lo que simplifica el código y mejora la legibilidad. Además, EF es compatible con múltiples proveedores de bases de datos, lo que brinda flexibilidad en la elección del motor de base de datos y ofrece funcionalidades adicionales, como el seguimiento de cambios y consultas LINQ, que mejoran el rendimiento y la productividad del desarrollo. En resumen, el cambio a EF agrego una capa de persistencia que fue lo pedido en la letra, simplificar el acceso a la base de datos, ofrece flexibilidad en la elección del proveedor y proporcionar funcionalidades adicionales para mejorar el desarrollo del proyecto.

Para mostrar como es la respectiva interacción entre la UI y el dominio usaremos la clase “AddNewFigureView” para demostrarlo. Mostraremos un UserControl que usa la clase del dominio Figura y cómo se comunican entre ellas según las reglas que nos han decretado. En el UserControl se tiene un botón que se encarga de agregar la figura, al darle click se agrega dicha figura para el cliente que está logueado. En este ejemplo la clase UserControl interactúa con la clase Figura del dominio para agregar una nueva figura para posteriormente nuestro otro userControl ListOfFigures obtenga la lista y la actualice.. El evento del boton “Agregar figura” por ejemplo crea una instancia de la clase “Figura” con el nombre que se ingresó en la UI y luego se llamó al método “addFigure()” del driver FigureDriver para agregar dicha figura al cliente que está iniciado sesión y que después pueda ser persistente en la base de datos.

3f) Análisis de los criterios para las responsabilidades:

Dominio:

- **Client:** La clase "Cliente" se creó para representar a los usuarios del sistema, su responsabilidad puntual es almacenar y gestionar la información de los usuarios, como su nombre de usuario, contraseña, fecha de registro y las listas de figuras, materiales, etc. En el obligatorio 1 esta clase no contaba con la apropiada responsabilidad y cohesión, contaba con operaciones que no eran apropiada tenerlas en la clase, para el obligatorio 2 esto se modificó asignando dichas operaciones al repositorio correspondiente y quitandolos de la clase por lo tanto ahora esta clase cuenta con la apropiada responsabilidad y cohesión.

- **Figure**: La clase "Figura" se creó para representar diferentes tipos de figuras geométricas, y su responsabilidad puntual es almacenar y gestionar la información relacionada con las figuras, como su nombre, radio y dueño. A su vez, la clase "Figura" incluye la validación y verificación de la información de figuras, como el nombre y el radio, para asegurarse de que sean válidos y cumplan con los requisitos necesarios.

- **Graphic Engine:** La clase "Graphic Engine" nos fue brindada por el cliente, se utiliza para renderizar las figuras que el cliente cree.

- **Material:** La clase "Material" se creó para representar diferentes tipos de materiales que se pueden utilizar en la construcción de modelos, y su responsabilidad puntual es almacenar y gestionar la información relacionada con los materiales, como su nombre, color y dueño. También incluye la validación y verificación de la información relacionada con los materiales, esta valida el nombre, para asegurarse de que sea válido y cumpla con los requisitos necesarios.

- **Model:** La clase "Modelo" se creó para representar diferentes tipos de modelos que se pueden utilizar en la construcción de modelos, y su responsabilidad puntual es almacenar y gestionar la información relacionada con los modelos, como su nombre, las figuras, materiales, un preview y dueño. También incluye la validación y verificación de la información relacionada con los modelos esta valida el nombre, para asegurarse de que sea válido y cumpla con los requisitos necesarios.

- **Positioned Model:** La clase "Modelo Posicionado" se utiliza para almacenar y gestionar las distintas posiciones “x,y,z”.

- **Ray:** La clase "Ray" nos fue brindada por el cliente, se utiliza para darle origen y dirección al vector.

- **Scene:** La clase "Escena" se creó para representar diferentes tipos de escenas, y su responsabilidad puntual es almacenar y gestionar la información relacionada con dichas

escenas, como su nombre, la fecha de creación, fecha de modificación, los distintos puntos de vista, y dueño. También incluye la validación y verificación de la información relacionada con la escena, esta valida el nombre, para asegurarse de que sea válido y cumpla con los requisitos necesarios.

- **Vector 3D:** La clase "Vector" nos fue brindada por el cliente, se utiliza para almacenar y gestionar la información relacionada con los vectores, como es los valores x,y,z, los colores, operaciones como sumar, restar, entre otras.

Drivers (uno para cada clase del Dominio menos scripts):

- Las clases “Driver” son el intermediario entre el paquete Repositories y la interfaz, estas clases fueron añadidas para facilitar la unión con la base de datos. Su responsabilidad es llamar al repositorio de cada clase para acceder a los métodos necesarios de cada clase. La clase "Driver" encapsula la lógica necesaria para, por ejemplo, devolver una figura de un cliente específico, y delega la tarea de devolver los datos a una clase especializada en la gestión de los datos de los clientes (Anexo - 18).

Repositories:

- **Repositories (uno para cada clase del Dominio menos scripts):**
  Las clases “Repositories” son las encargadas de guardar la información del sistema, en ellas se encuentran los métodos de agregación, modificación eliminación, obtención y algunos otros métodos, para cada una de las clases. Su responsabilidad es llamar a los métodos de cada clase para que el programa funcione correctamente. Por ejemplo, dentro del repositorio de figura tenemos el método de Get Figura.(Anexo - 19).

- **Interface(uno para cada clase del Dominio menos scripts):**
  Las clases “Interfaces” son las encargadas de llamar a dichos métodos.

Tests:

Las clases “Test” contienen las pruebas unitarias correspondientes para cada clase del sistema, las mismas nos aseguran que el código funcione correctamente y que las excepciones se lancen en el momento indicado. Esas clases tienen cada una un porcentaje de cobertura individual, pero que al mismo tiempo todas juntas nos dan la cobertura global del proyecto que debe superar el 90% para que el código sea aceptable.

4) Cobertura de las pruebas unitarias (análisis y justificaciones):

Se aplicó TDD para este obligatorio logrando aplicar las técnicas aprendidas en clase y reafirmar la importancia de realizar las pruebas primero. Se utilizó el ciclo de TDD, el cual se refiere a primero a crear un test que falle para después poder realizar los cambios necesarios para que dicho test pase, y por último realizar el refactoreo necesario y volver a repetir el proceso.

En nuestro proyecto se logró alcanzar una cobertura de código del 92,23%(Anexo - 20). Se crearon un total de 183 pruebas de las cuales ninguna produce errores o queda suspendida. Para generalizar pruebas, en la mayoría de las clases se nos pedía asegurarnos que el nombre no tuviera espacios y que no fuera repetido; es así cómo podemos reutilizar test variando según en qué clase fueran a hacerse las pruebas (Anexo - 21). También hemos creado ciertas pruebas para el motor gráfico y para los repositorios y manejadores, así como para las interfaces. Y así poder lograr un buen porcentaje de cobertura. En la parte de anexos se agregan más ejemplos de Pruebas unitarias realizadas en nuestro proyecto (Anexo - 22).

5) Reporte de evidencia de los resultados de los casos de prueba:

Al no haber implementado TDD en la primera parte del obligatorio hay ciertas clases que no llegan al 90%, pero al igual que el obligatorio pasado lo requerido es llegar al 90% del trabajo global, que fue logrado a partir de realizar distintos test y de implementar en esta segunda parte del obligatorio el TDD, para así asegurándonos de cubrir los casos bordes y los casos evidentes. A continuación se adjunta un desglose de la cobertura de código de cada clase test con la cantidad de pruebas que tiene cada una (Anexo - 23).

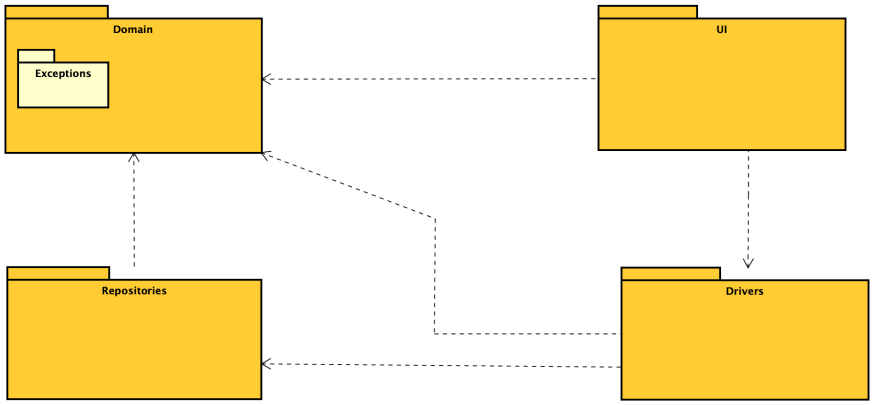
6) Instalación:

Para la correcta instalación de nuestro obligatorio se proporciona la ubicación de los ejecutables en una carpeta en la raíz del proyecto con el nombre release, si bien no se necesitan librerías es necesario tener el paquete NuGet Entity Framework instalado. Por otro lado los string de conexión a modificar se encuentran en el archivo App.config ubicado en cada proyecto de la solución (UI, domain, drivers, repositories, tests siendo este el único proyecto que posee un string de conexión diferente a los demás). A continuación se detalla la ubicación de los backups y los scrips de la base de datos, dicha ubicación se encuentra un una carpeta ubicada en la raíz del proyecto llamada “scripts” y por último en el caso que se recupere la base de datos con datos existen tres distintos usuarios:

- User: Carlos – Pass: Carlos4567
- User: Seba – Pass: Sc1234
- User: Gonza – Pass: 2617Gg

7) Anexos:

Anexo 1:

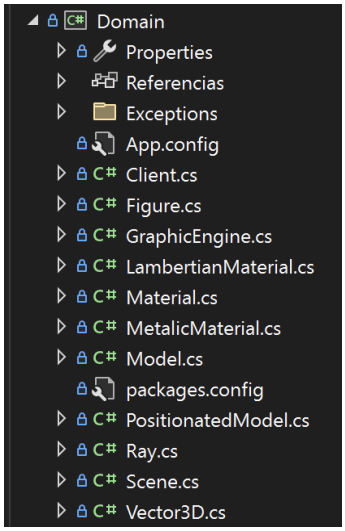


Anexo 2:

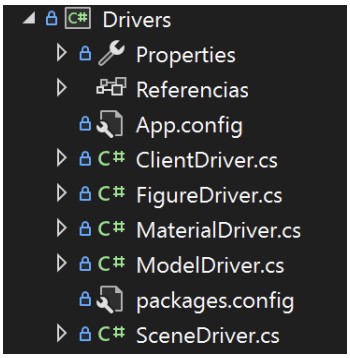
Anexo 3:

```
using Domain;

namespace UI
{
```

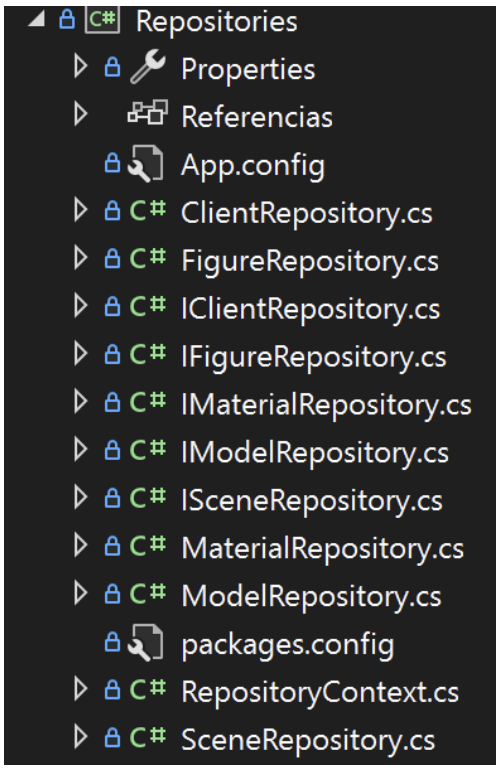
Anexo 4:



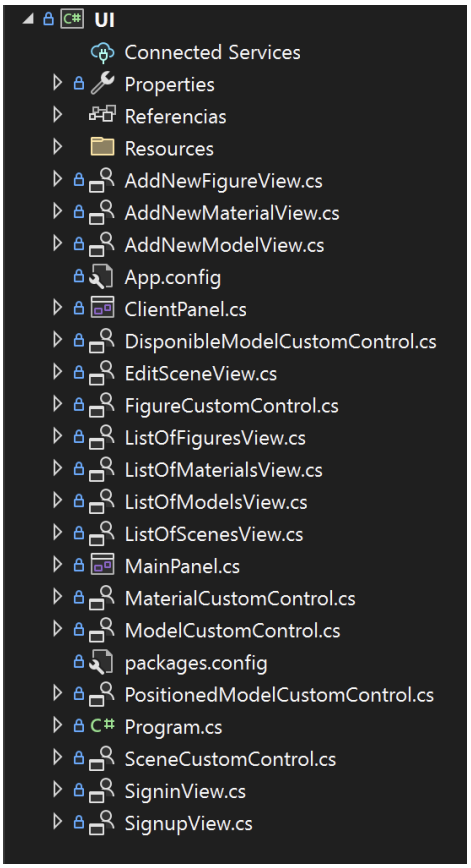
Anexo 5:



Anexo 6:



Anexo 7:

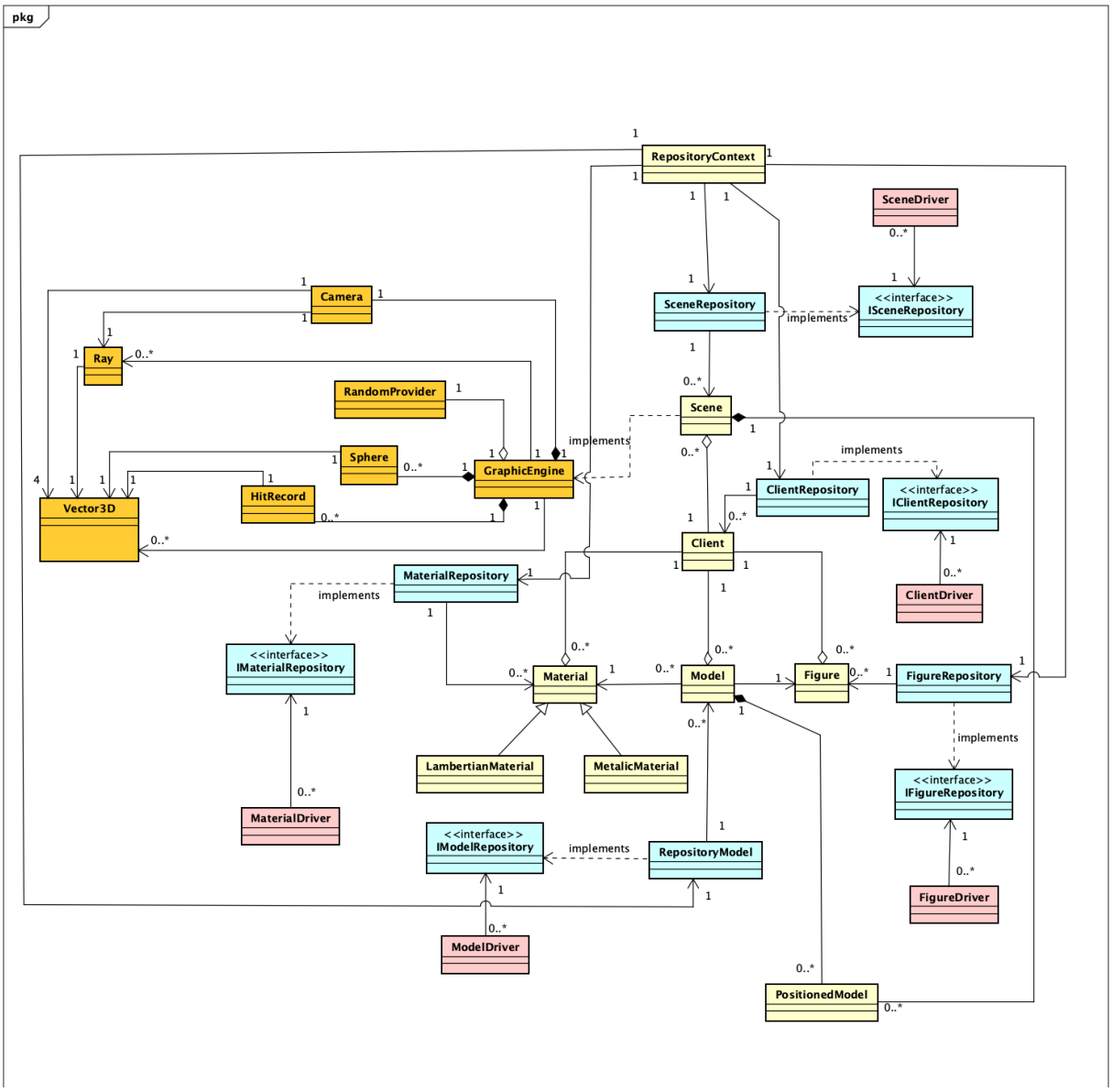


Anexo 8:

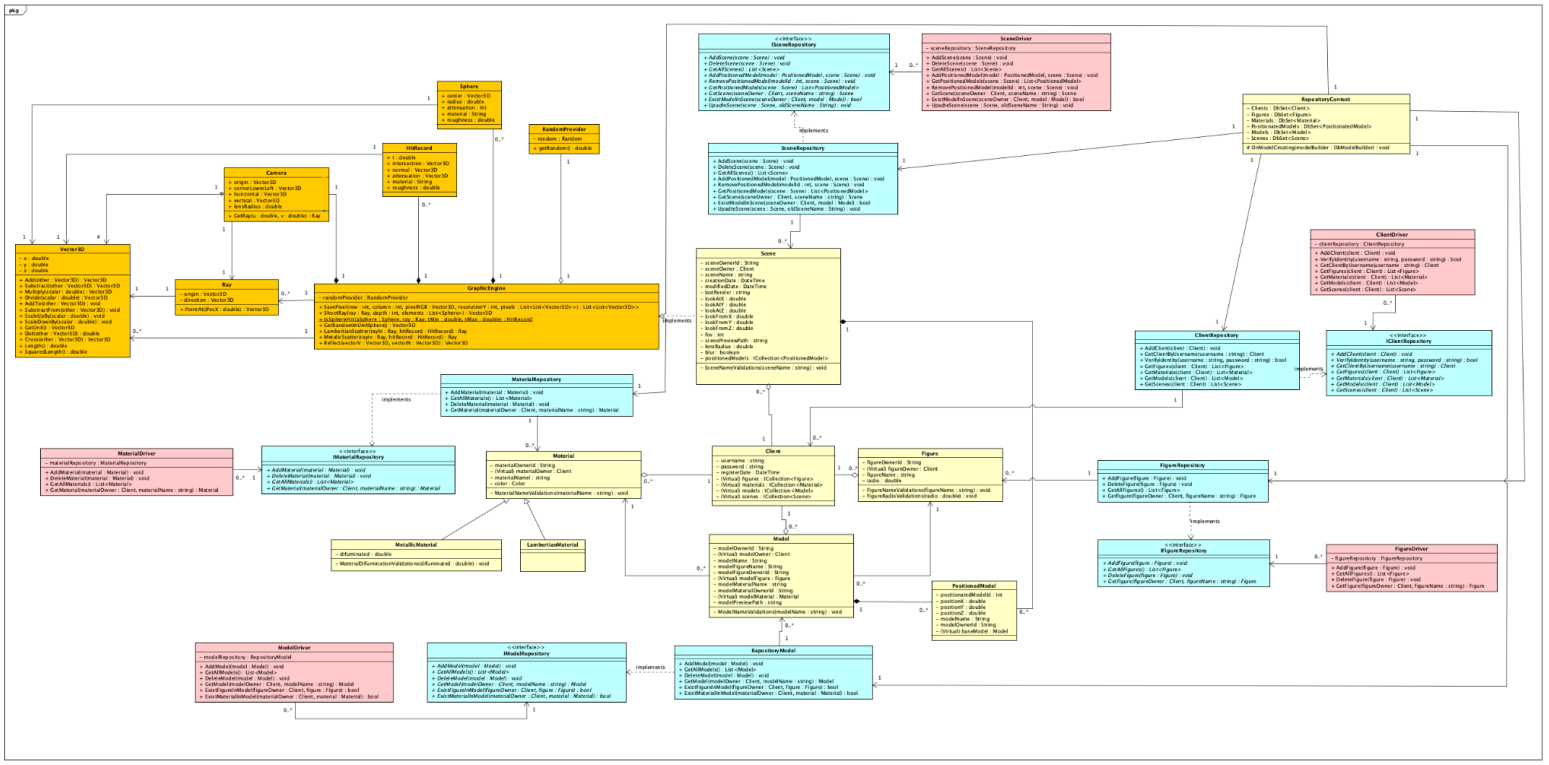
```
1 referencia | 0 cambios | 0 autores, 0 cambios
public void AddClient(Client client)
{
    clientRepository.AddClient(client);
}
```

Anexo 9:

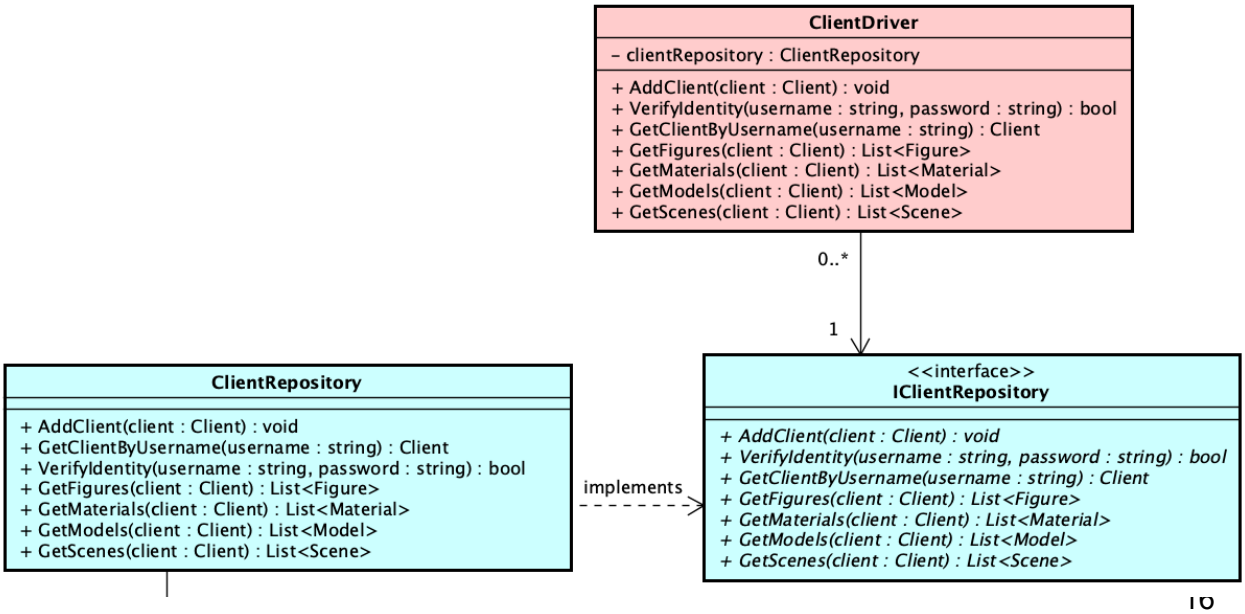
UML (diagrama general con las clases y las relaciones entre ellas):



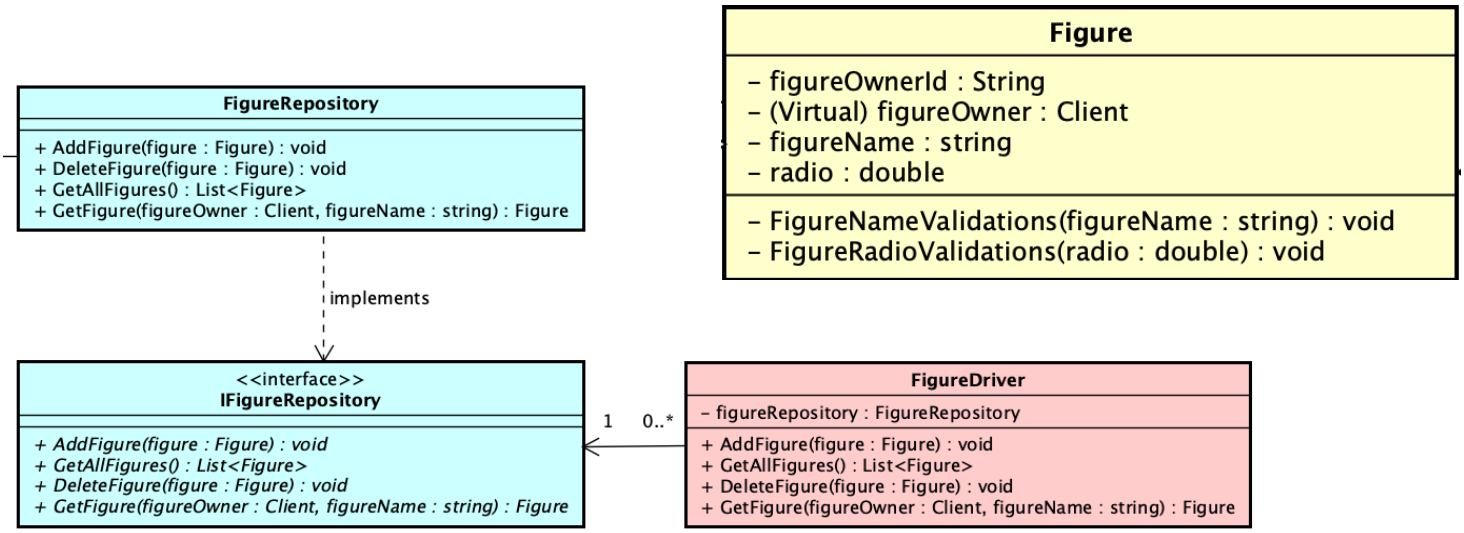
Anexo 9.1:



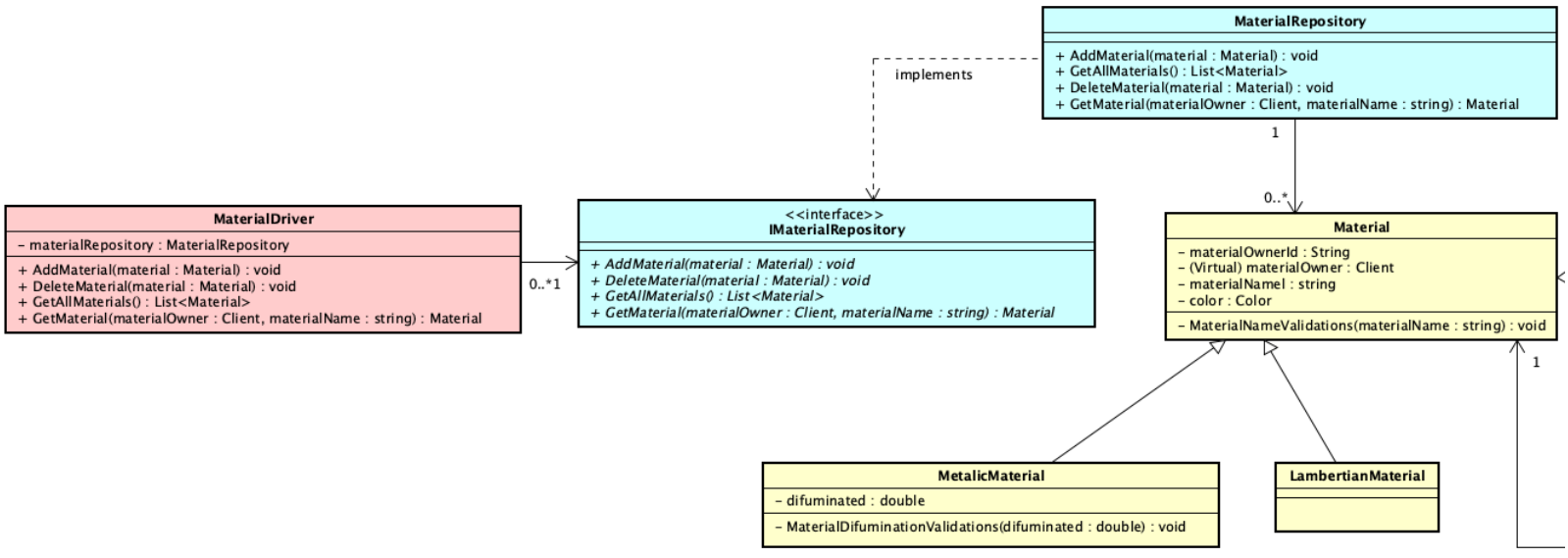
Anexo 10:



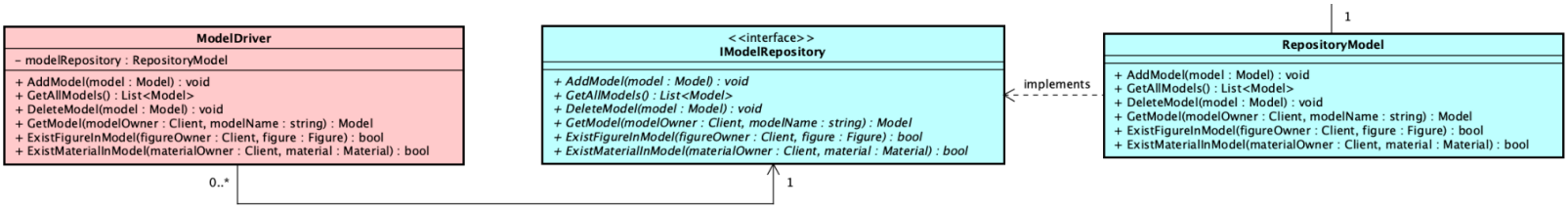
Anexo 11:

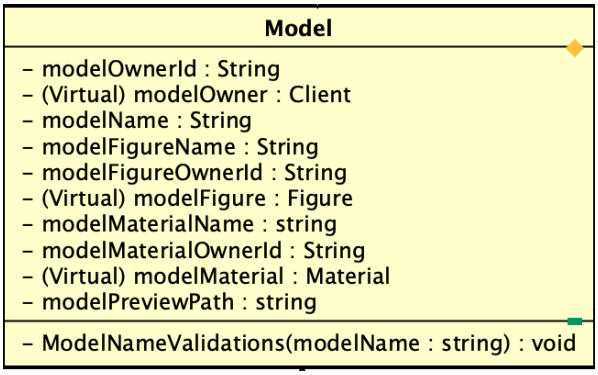


Anexo 12:

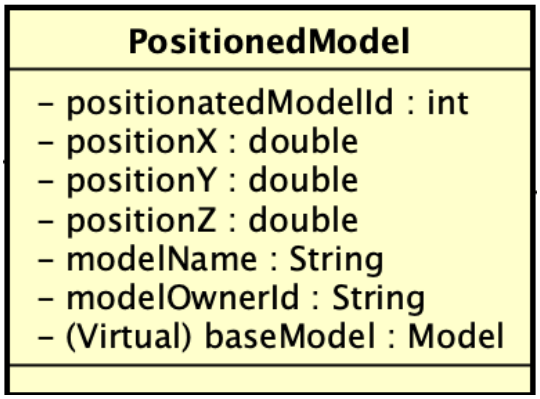


Anexo 13:

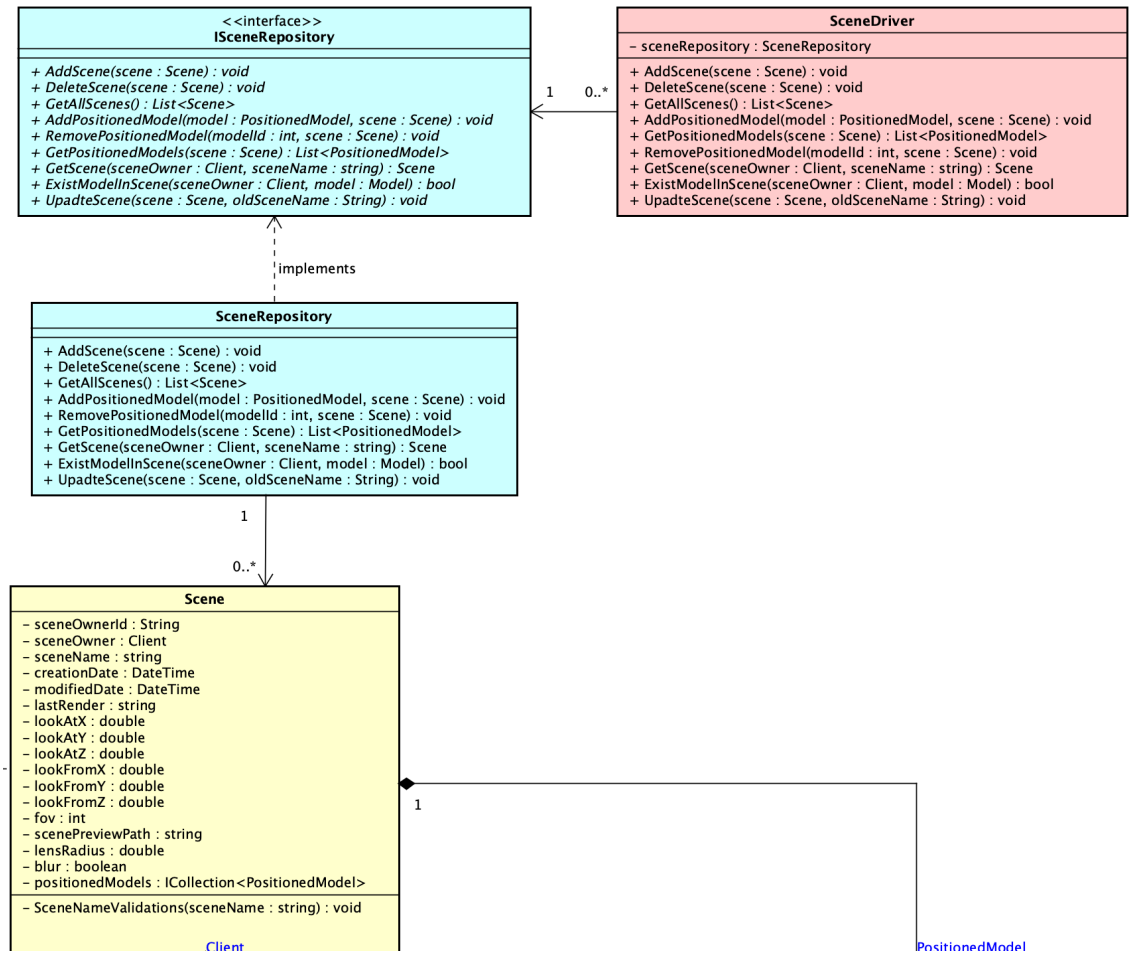




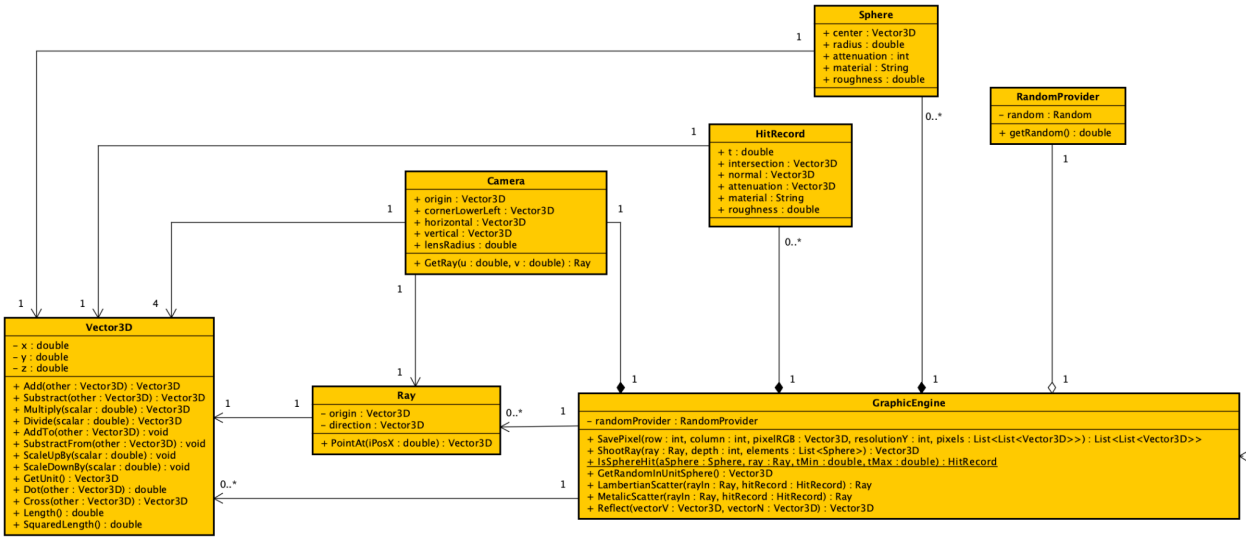
Anexo 14:



Anexo 15


<<interface>>
ISceneRepository
+ AddScene(scene : Scene) : void
+ DeleteScene(scene : Scene) : void
+ GetAllScenes() : List<Scene>
+ AddPositionedModel(model : PositionedModel, scene : Scene) : void
+ RemovePositionedModel(modelId : int, scene : Scene) : void
+ GetPositionedModels(scene : Scene) : List<PositionedModel>
+ GetScene(sceneOwner : Client, sceneName : string) : Scene
+ ExistModelInScene(sceneOwner : Client, model : Model) : bool
+ UpadteScene(scene : Scene, oldSceneName : String) : void
1
0..*
SceneDriver
- sceneRepository : SceneRepository
+ AddScene(scene : Scene) : void
+ DeleteScene(scene : Scene) : void
+ GetAllScenes() : List<Scene>
+ AddPositionedModel(model : PositionedModel, scene : Scene) : void
+ GetPositionedModels(scene : Scene) : List<PositionedModel>
+ RemovePositionedModel(modelId : int, scene : Scene) : void
+ GetScene(sceneOwner : Client, sceneName : string) : Scene
+ ExistModelInScene(sceneOwner : Client, model : Model) : bool
+ UpadteScene(scene : Scene, oldSceneName : String) : void
implements
SceneRepository
+ AddScene(scene : Scene) : void
+ DeleteScene(scene : Scene) : void
+ GetAllScenes() : List<Scene>
+ AddPositionedModel(model : PositionedModel, scene : Scene) : void
+ RemovePositionedModel(modelId : int, scene : Scene) : void
+ GetPositionedModels(scene : Scene) : List<PositionedModel>
+ GetScene(sceneOwner : Client, sceneName : string) : Scene
+ ExistModelInScene(sceneOwner : Client, model : Model) : bool
+ UpadteScene(scene : Scene, oldSceneName : String) : void
1
0..*
Scene
- sceneOwnerId : String
- sceneOwner : Client
- sceneName : string
- creationDate : DateTime
- modifiedDate : DateTime
- lastRender : string
- lookAtX : double
- lookAtY : double
- lookAtZ : double
- lookFromX : double
- lookFromY : double
- lookFromZ : double
- fov : int
- scenePreviewPath : string
- lensRadius : double
- blur : boolean
- positionedModels : ICollection<PositionedModel>
- SceneNameValidations(sceneName : string) : void
1
Client
PositionedModel

Anexo 16:



Anexo 17:

```
1 referencia | 0 cambios | 0 autores, 0 cambios
private void txtFigureRadio_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!char.IsDigit(e.KeyChar) && e.KeyChar != ',' || e.KeyChar == ',' && txtFigureRadio.TextLength == 0)
    {
        e.Handled = true;
    }
    if (e.KeyChar == ',' && (sender as System.Windows.Forms.TextBox).Text.IndexOf(',') > -1)
    {
        e.Handled = true;
    }
}
```

Anexo 18:

```
3 referencias | 0 cambios | 0 autores, 0 cambios
public List<Figure> GetFigures(Client client)
{
    return clientRepository.GetFigures(client);
}
```

Anexo 19:

```
11 referencias | 0 cambios | 0 autores, 0 cambios
public Figure GetFigure(Client owner, string figureName)
{
    using (RepositoryContext dbContext = new RepositoryContext())
    {
        Figure returnFigure = null;
        Client dbClient = dbContext.Clients.Find(owner.Username);
        if (dbClient != null)
        {
            returnFigure = dbContext.Figures.FirstOrDefault(f => f.FigureName == figureName && f.FigureOwner.Username == owner.Username);
        }
        return returnFigure;
    }
```

Anexo 20:

Resultados de la cobertura de código

scuneo_CUNEO2786 2023-05-11 15_33_44.

Hierarchy	Covered (%Blocks)	Not Covered (%Blocks)	Covered (%Lines)
scuneo_CUNEO2786 2023-05-11 15_33_44.coverage	92,84%	7,16%	93,42%

Anexo 21:
//En figura

```
[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void FigureNameValidations_ThrowsException_WhenFigureNameHasLeadingSpaces()
{
    // Arrange
    string figureName = " Circle";
    Figure figura = new Figure();

    // Act and Assert
    Assert.ThrowsException<InvalidFigureNameException>(() => figura.FigureName = figureName);
}

[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void FigureNameValidations_ThrowsException_WhenFigureNameHasTailSpaces()
{
    // Arrange
    string figureName = "Circle ";
    Figure figura = new Figure();

    // Act and Assert
    Assert.ThrowsException<InvalidFigureNameException>(() => figura.FigureName = figureName);
}
```

//En Material

```
[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void SetName_WhenNameStartsWithSpace_ShouldThrowInvalidMaterialNameException()
{
    // Arrange
    var material = new Material();

    // Act
    Action setNameAction = () => material.MaterialName = " Material";

    // Assert
    Assert.ThrowsException<InvalidMaterialNameException>(setNameAction);
}

[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void SetName_WhenNameEndsWithSpace_ShouldThrowInvalidMaterialNameException()
{
    // Arrange
    var material = new Material();

    // Act
    Action setNameAction = () => material.MaterialName = "Material ";

    // Assert
    Assert.ThrowsException<InvalidMaterialNameException>(setNameAction);
}
```

Anexo 22:

```
[TestMethod]
    | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void AddClient_ShouldAddNewClient()
{
    // Arrange
    var repository = new ClientDriver();
    var client = new Client
    {
        Username = "username"
    };

    // Act
    repository.AddClient(client);

    // Assert
    Client dbClient = repository.GetClientByUsername("username");
    Assert.IsTrue(client.Username == dbClient.Username);
}
```

```
[TestMethod]
    | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void VerifyIdentity_EmptyUsername_ReturnsFalse()
{
    // Arrange
    string username = "";
    string password = "5678";
    var repository = new ClientDriver();

    // Act
    bool result = repository.VerifyIdentity(username, password);

    // Assert
    Assert.IsFalse(result);
}
```

```
[TestMethod]
    ) referencias | 0 cambios | 0 autores, 0 cambios
public void AddsMultipleFigures()
{
    var client = new Client
    {
        Username = "username1"
    };
    var figure1 = new Figure
    {
        FigureName = "Test1",
        FigureOwnerId = client.Username,
        FigureOwner = client,
        Radio = 1
    };
    var figure2 = new Figure
    {
        FigureName = "Test2",
        FigureOwnerId = client.Username,
        FigureOwner = client,
        Radio = 1
    };
    var repository1 = new ClientDriver();
    var repository2 = new FigureDriver();

    repository1.AddClient(client);
    repository2.AddFigure(figure1);
    repository2.AddFigure(figure2);

    Figure dbFigure1 = repository2.GetFigure(client, figure1.FigureName);
    Assert.IsTrue(figure1.FigureName == dbFigure1.FigureName);
    Figure dbFigure2 = repository2.GetFigure(client, figure2.FigureName);
    Assert.IsTrue(figure2.FigureName == dbFigure2.FigureName);
}
```

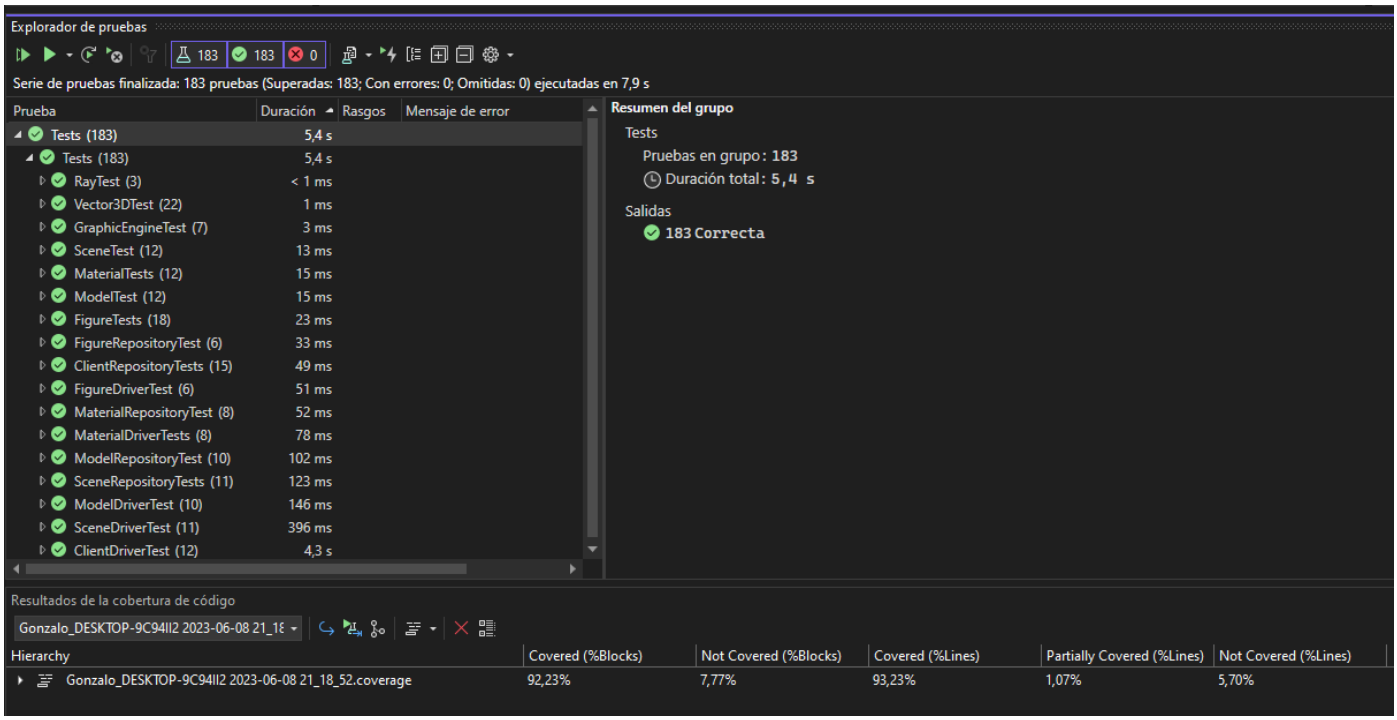
```
[TestMethod]
    0 referencias | 0 cambios | 0 autores, 0 cambios
public void TestDeleteFigure()
{
    var client = new Client
    {
        Username = "username1"
    };
    var figure = new Figure
    {
        FigureName = "Test",
        FigureOwnerId = client.Username,
        FigureOwner = client,
        Radio = 1
    };
    var repository1 = new ClientDriver();
    var repository2 = new FigureDriver();

    repository1.AddClient(client);
    repository2.AddFigure(figure);

    Figure dbFigureAdd = repository2.GetFigure(client, figure.FigureName);
    Assert.IsTrue(figure.FigureName == dbFigureAdd.FigureName);

    repository2.DeleteFigure(figure);
    Figure dbFigureDelete = repository2.GetFigure(client, figure.FigureName);
    Assert.IsTrue(dbFigureDelete == null);
}
```

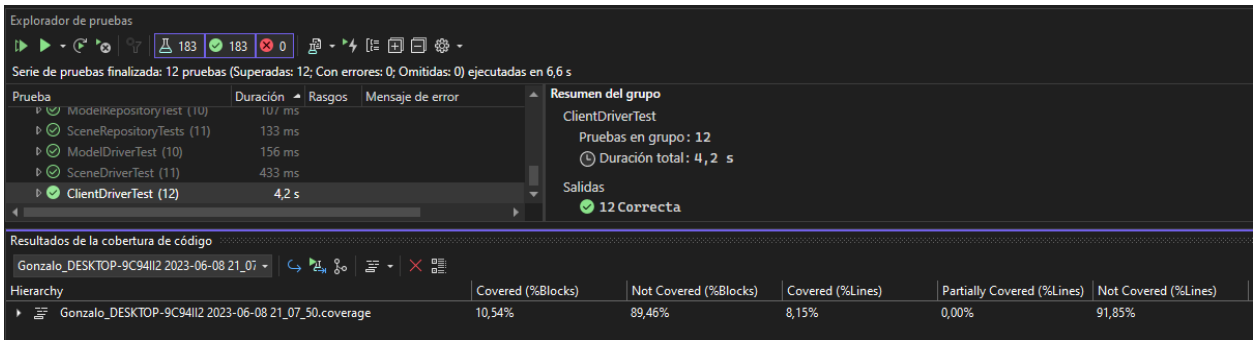
```
[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void GetRandom_ReturnsValueBetweenZeroAndOne()
{
    // Arrange
    Random random = new Random();
    RandomProvider randomProvider = new RandomProvider(false, random);

    // Act
    double result = randomProvider.getRandom();

    // Assert
    Assert.IsTrue(result >= 0 && result <= 1);
}
```

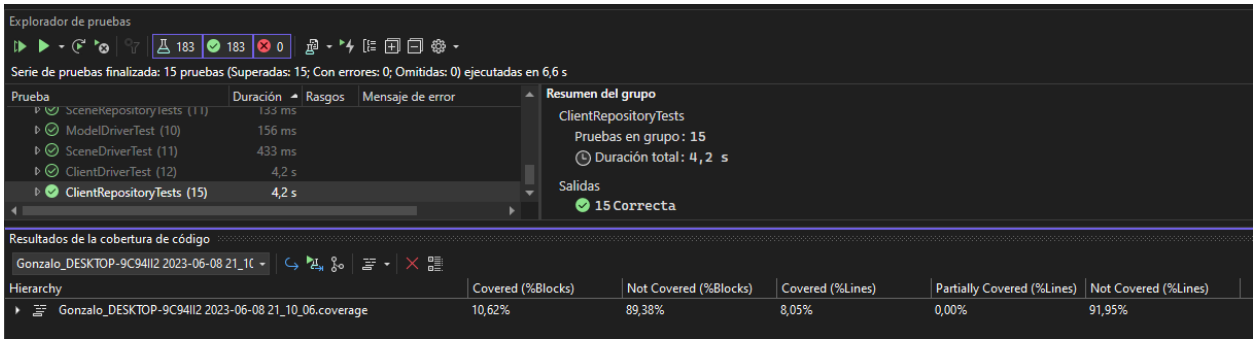
```
[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void IsSphereLambertianHit_RayDoesNotHitSphere_ReturnsNull()
{
    // Arrange
    Sphere sphere = new Sphere(new Vector3D(0, 0, -1), 0.5, new Vector3D(0.1, 0.2, 0.3), "lambertian");
    Ray ray = new Ray(new Vector3D(0, 0, 0), new Vector3D(0, 1, 0));

    // Act
    var result = GraphicEngine.IsSphereHit(sphere, ray, 0, double.MaxValue);

    // Assert
    Assert.IsNull(result);
}

[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void IsSphereMetalicHit_RayDoesNotHitSphere_ReturnsNull()
{
    // Arrange
    Sphere sphere = new Sphere(new Vector3D(0, 0, -1), 0.5, new Vector3D(0.1, 0.2, 0.3), "metalic");
    Ray ray = new Ray(new Vector3D(0, 0, 0), new Vector3D(0, 1, 0));

    // Act
    var result = GraphicEngine.IsSphereHit(sphere, ray, 0, double.MaxValue);

    // Assert
    Assert.IsNull(result);
}
```

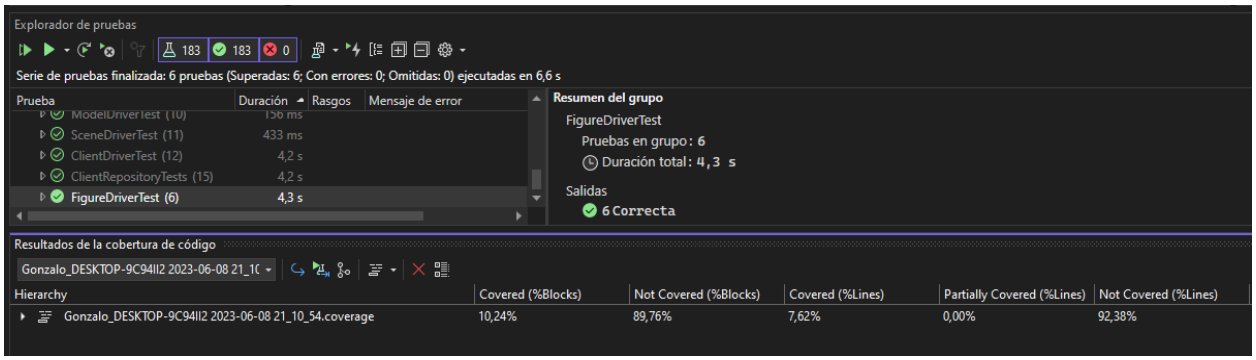
```
[TestMethod]
| 0 referencias | 0 cambios | 0 autores, 0 cambios
public void Direction_Setter_ModifiesDirection()
{
    // Arrange
    var origin = new Vector3D(1, 2, 3);
    var direction = new Vector3D(2, 0, 1);
    var ray = new Ray(origin, direction);
    var newDirection = new Vector3D(1, 1, 1);

    // Act
    ray.Direction = newDirection;

    // Assert
    Assert.AreEqual(ray.Direction, newDirection);
}
```

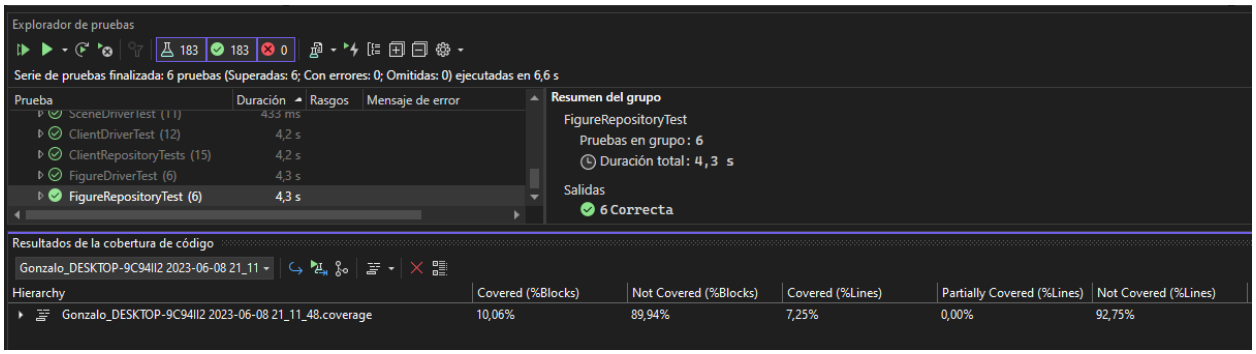
Anexo 23:

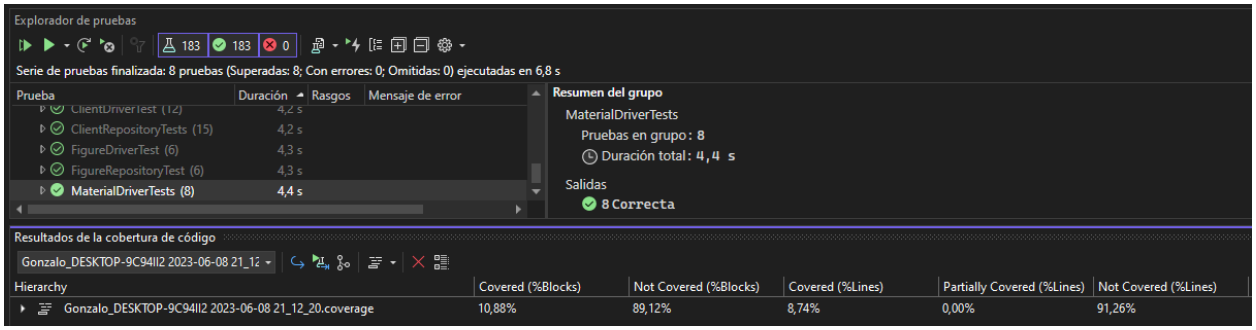


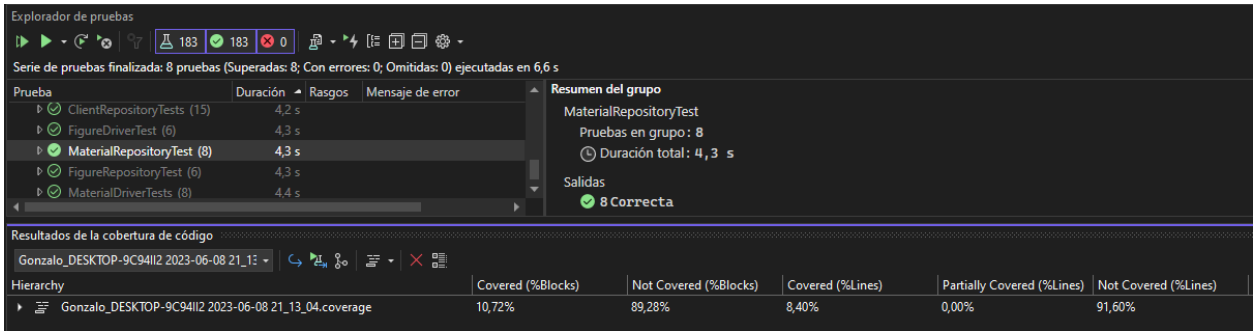

Explorador de pruebas
183
183
0
Serie de pruebas finalizada: 12 pruebas (Superadas: 12; Con errores: 0; Omitidas: 0) ejecutadas en 6,6 s
Prueba
Duración
Rasgos
Mensaje de error
SceneRepositoryTests (11) 133 ms
ModelDriverTest (10) 156 ms
SceneDriverTest (11) 433 ms
ClientDriverTest (12) 4,2 s
Resumen del grupo
ClientDriverTest
Pruebas en grupo: 12
Duración total: 4,2 s
Salidas
12 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_07
Hierarchy
Covered (%Blocks)
Not Covered (%Blocks)
Covered (%Lines)
Partially Covered (%Lines)
Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_07_50.coverage
10,54%
89,46%
8,15%
0,00%
91,85%

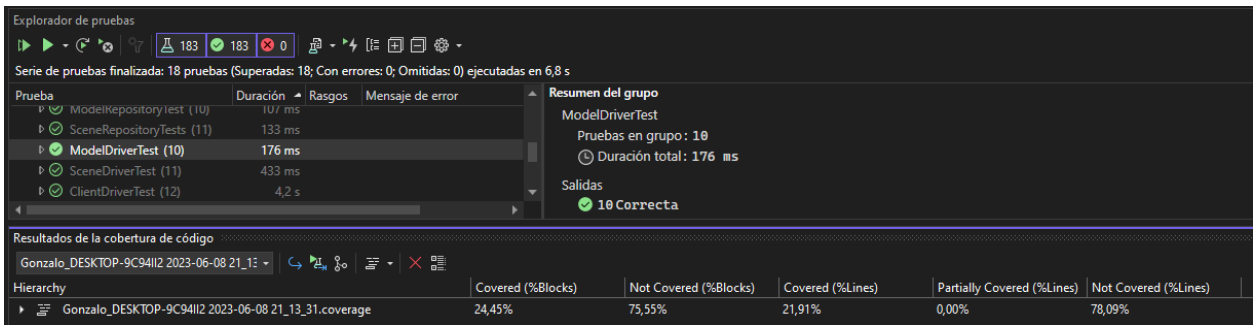

Explorador de pruebas
183
183
0
Serie de pruebas finalizada: 15 pruebas (Superadas: 15; Con errores: 0; Omitidas: 0) ejecutadas en 6,6 s
Prueba
Duración
Rasgos
Mensaje de error
ModelDriverTest (10) 156 ms
SceneDriverTest (11) 433 ms
ClientDriverTest (12) 4,2 s
ClientRepositoryTests (15) 4,2 s
Resumen del grupo
ClientRepositoryTests
Pruebas en grupo: 15
Duración total: 4,2 s
Salidas
15 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_10
Hierarchy
Covered (%Blocks)
Not Covered (%Blocks)
Covered (%Lines)
Partially Covered (%Lines)
Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_10_06.coverage
10,62%
89,38%
8,05%
0,00%
91,95%

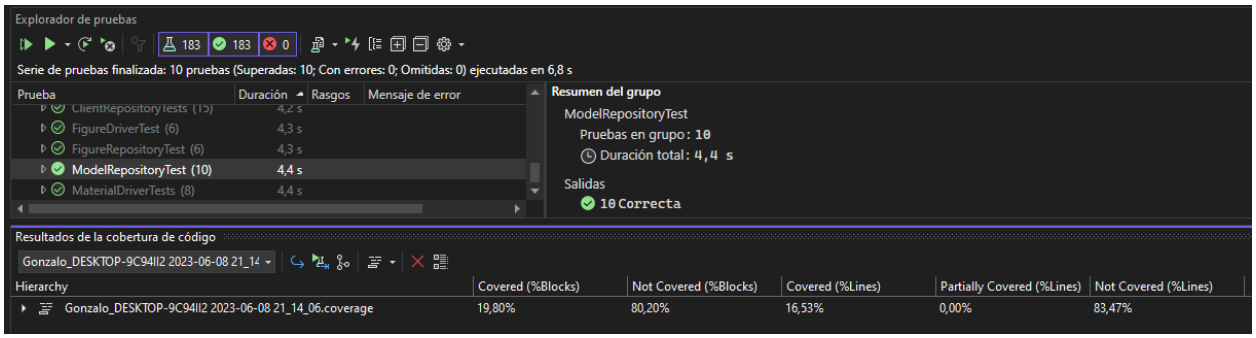

Explorador de pruebas
183
183
0
Serie de pruebas finalizada: 6 pruebas (Superadas: 6; Con errores: 0; Omitidas: 0) ejecutadas en 6,6 s
Prueba
Duración
Rasgos
Mensaje de error
SceneDriverTest (11) 433 ms
ClientDriverTest (12) 4,2 s
ClientRepositoryTests (15) 4,2 s
FigureDriverTest (6) 4,3 s
Resumen del grupo
FigureDriverTest
Pruebas en grupo: 6
Duración total: 4,3 s
Salidas
6 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_10
Hierarchy
Covered (%Blocks)
Not Covered (%Blocks)
Covered (%Lines)
Partially Covered (%Lines)
Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_10_54.coverage
10,24%
89,76%
7,62%
0,00%
92,38%

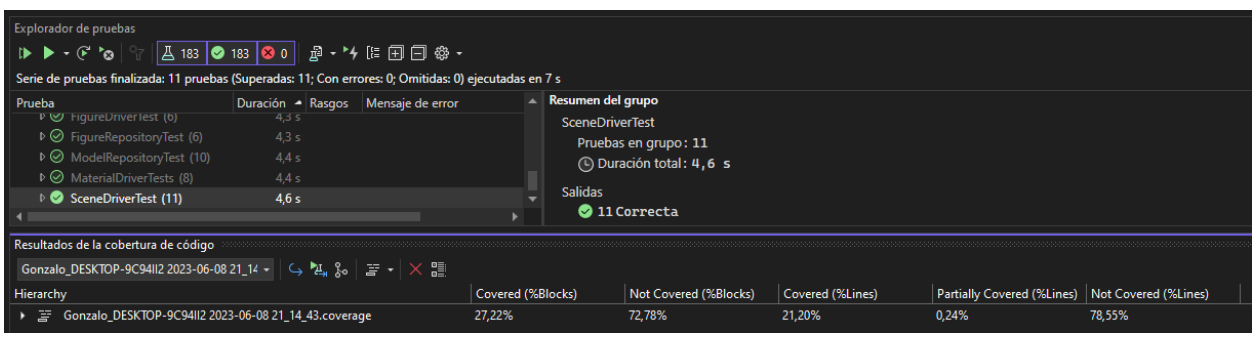

Explorador de pruebas
183
183
0
Serie de pruebas finalizada: 6 pruebas (Superadas: 6; Con errores: 0; Omitidas: 0) ejecutadas en 6,6 s
Prueba
Duración
Rasgos
Mensaje de error
ClientDriverTest (12) 4,2 s
ClientRepositoryTests (15) 4,2 s
FigureDriverTest (6) 4,3 s
FigureRepositoryTest (6) 4,3 s
Resumen del grupo
FigureRepositoryTest
Pruebas en grupo: 6
Duración total: 4,3 s
Salidas
6 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_11
Hierarchy
Covered (%Blocks)
Not Covered (%Blocks)
Covered (%Lines)
Partially Covered (%Lines)
Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_11_48.coverage
10,06%
89,94%
7,25%
0,00%
92,75%

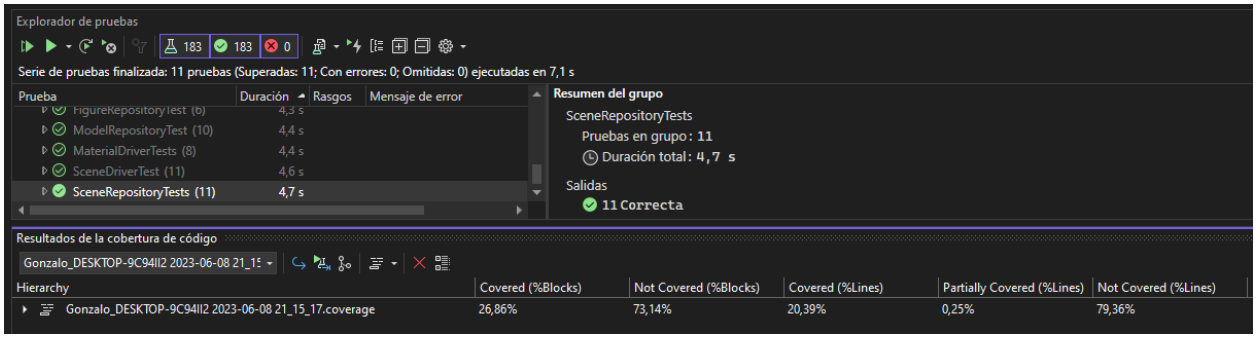

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 8 pruebas (Superadas: 8; Con errores: 0; Omitidas: 0) ejecutadas en 6,8 s
Prueba Duración Rasgos Mensaje de error
ClientRepositoryTests (15) 4,2 s
FigureDriverTest (6) 4,3 s
FigureRepositoryTest (6) 4,3 s
MaterialDriverTests (8) 4,4 s
Resumen del grupo
MaterialDriverTests
Pruebas en grupo: 8
Duración total: 4,4 s
Salidas
8 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_12
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_12_20.coverage 10,88% 89,12% 8,74% 0,00% 91,26%

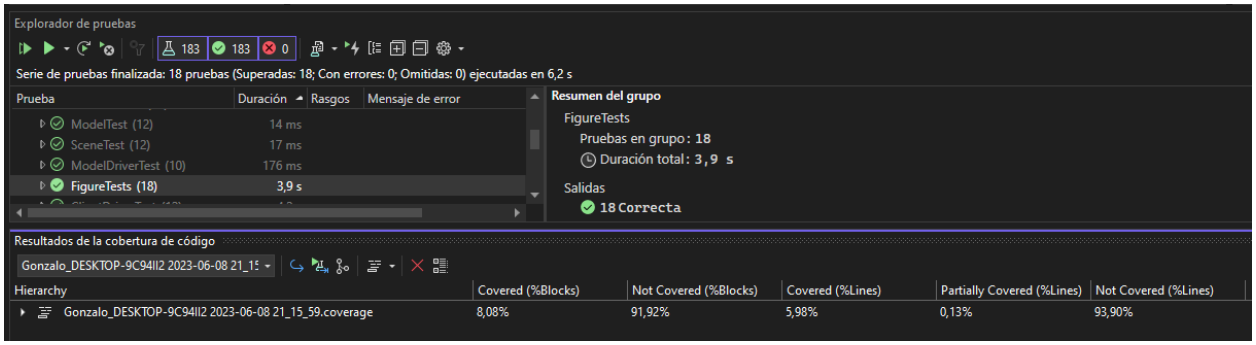

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 8 pruebas (Superadas: 8; Con errores: 0; Omitidas: 0) ejecutadas en 6,6 s
Prueba Duración Rasgos Mensaje de error
ClientRepositoryTests (15) 4,2 s
FigureDriverTest (6) 4,3 s
MaterialRepositoryTest (8) 4,3 s
FigureRepositoryTest (6) 4,3 s
MaterialDriverTests (8) 4,4 s
Resumen del grupo
MaterialRepositoryTest
Pruebas en grupo: 8
Duración total: 4,3 s
Salidas
8 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_13
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_13_04.coverage 10,72% 89,28% 8,40% 0,00% 91,60%

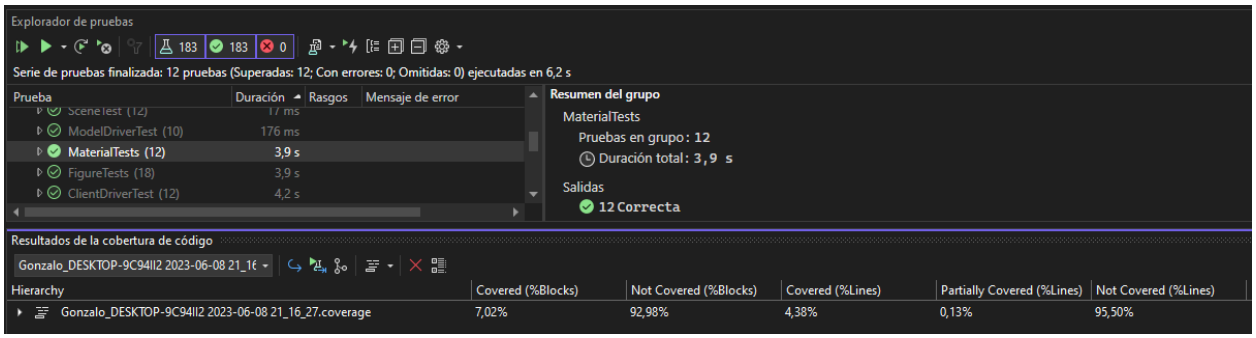

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 18 pruebas (Superadas: 18; Con errores: 0; Omitidas: 0) ejecutadas en 6,8 s
Prueba Duración Rasgos Mensaje de error
SceneRepositoryTests (11) 133 ms
ModelDriverTest (10) 176 ms
SceneDriverTest (11) 433 ms
ClientDriverTest (12) 4,2 s
Resumen del grupo
ModelDriverTest
Pruebas en grupo: 10
Duración total: 176 ms
Salidas
10 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_13
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_13_31.coverage 24,45% 75,55% 21,91% 0,00% 78,09%

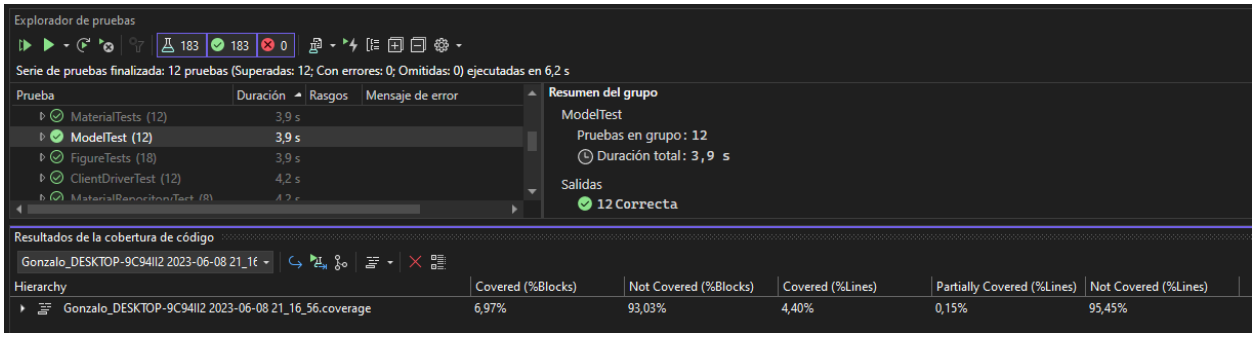

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 10 pruebas (Superadas: 10; Con errores: 0; Omitidas: 0) ejecutadas en 6,8 s
Prueba Duración Rasgos Mensaje de error
FigureDriverTest (6) 4,3 s
FigureRepositoryTest (6) 4,3 s
ModelRepositoryTest (10) 4,4 s
MaterialDriverTests (8) 4,4 s
Resumen del grupo
ModelRepositoryTest
Pruebas en grupo: 10
Duración total: 4,4 s
Salidas
10 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_14
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_14_06.coverage 19,80% 80,20% 16,53% 0,00% 83,47%

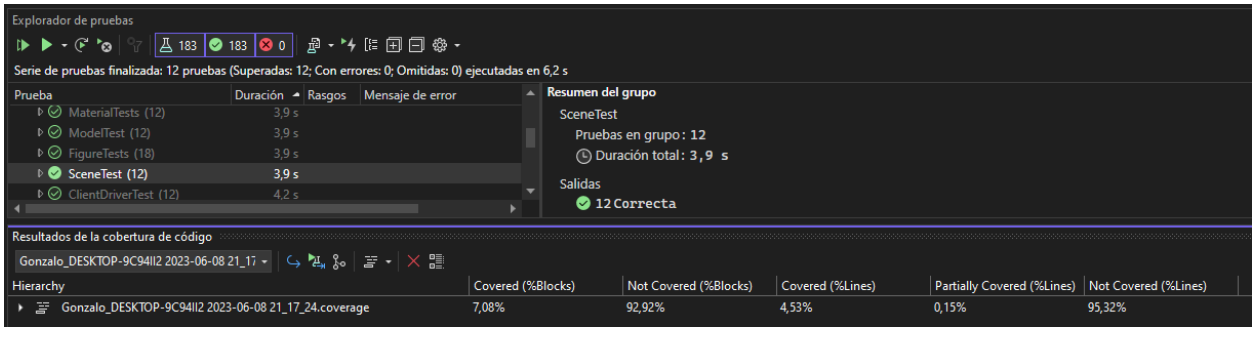

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 11 pruebas (Superadas: 11; Con errores: 0; Omitidas: 0) ejecutadas en 7 s
Prueba Duración Rasgos Mensaje de error
FigureRepositoryTest (6) 4,3 s
ModelRepositoryTest (10) 4,4 s
MaterialDriverTests (8) 4,4 s
SceneDriverTest (11) 4,6 s
Resumen del grupo
SceneDriverTest
Pruebas en grupo: 11
Duración total: 4,6 s
Salidas
11 Correcta
Resultados de la cobertura de código
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_14
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_14_43.coverage 27,22% 72,78% 21,20% 0,24% 78,55%

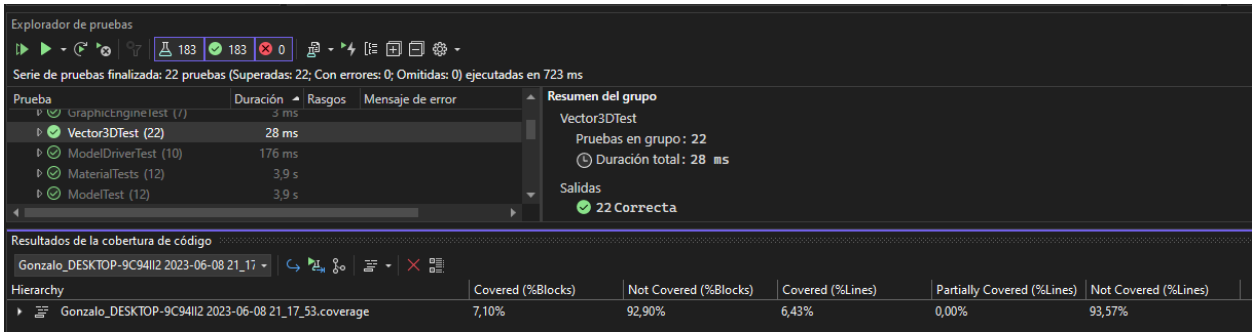

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 11 pruebas (Superadas: 11; Con errores: 0; Omitidas: 0) ejecutadas en 7,1 s
Prueba Duración Rasgos Mensaje de error
ModelRepositoryTest (10) 4,4 s
MaterialDriverTests (8) 4,4 s
SceneDriverTest (11) 4,6 s
SceneRepositoryTests (11) 4,7 s
Resumen del grupo
SceneRepositoryTests
Pruebas en grupo: 11
Duración total: 4,7 s
Salidas
11 Correcta
Resultados de la cobertura de código
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_15_17.coverage 26,86% 73,14% 20,39% 0,25% 79,36%

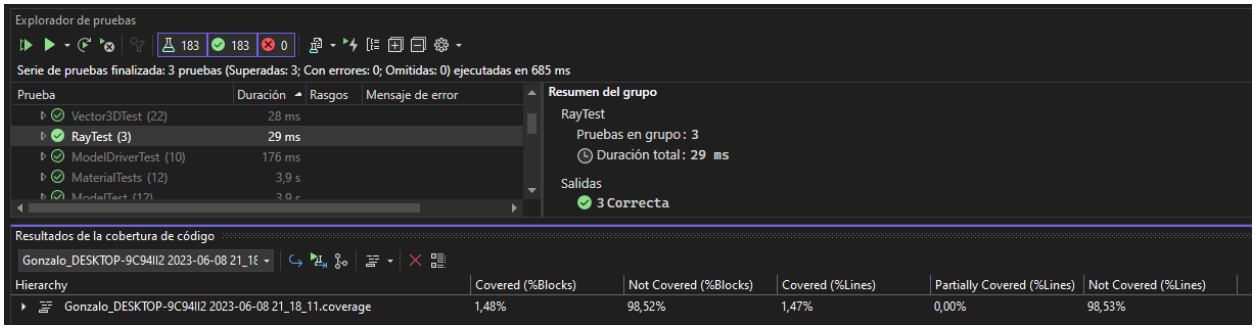

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 18 pruebas (Superadas: 18; Con errores: 0; Omitidas: 0) ejecutadas en 6,2 s
Prueba Duración Rasgos Mensaje de error
ModelTest (12) 14 ms
SceneTest (12) 17 ms
ModelDriverTest (10) 176 ms
FigureTests (18) 3,9 s
Resumen del grupo
FigureTests
Pruebas en grupo: 18
Duración total: 3,9 s
Salidas
18 Correcta
Resultados de la cobertura de código
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_15_59.coverage 8,08% 91,92% 5,98% 0,13% 93,90%

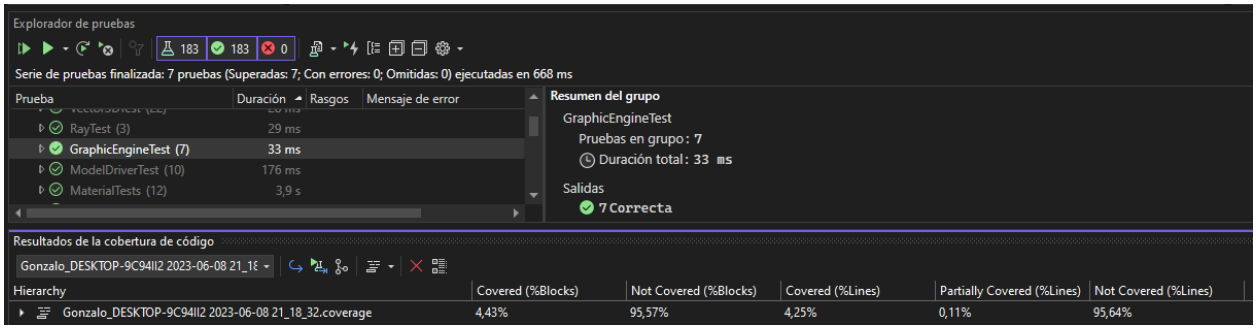

Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 12 pruebas (Superadas: 12; Con errores: 0; Omitidas: 0) ejecutadas en 6,2 s
Prueba Duración Rasgos Mensaje de error
ModelDriverTest (10) 176 ms
MaterialTests (12) 3,9 s
FigureTests (18) 3,9 s
ClientDriverTest (12) 4,2 s
Resumen del grupo
MaterialTests
Pruebas en grupo: 12
Duración total: 3,9 s
Salidas
12 Correcta
Resultados de la cobertura de código
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_16_27.coverage 7,02% 92,98% 4,38% 0,13% 95,50%


Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 12 pruebas (Superadas: 12; Con errores: 0; Omitidas: 0) ejecutadas en 6,2 s
Prueba Duración Rasgos Mensaje de error
MaterialTests (12) 3,9 s
ModelTest (12) 3,9 s
FigureTests (18) 3,9 s
ClientDriverTest (12) 4,2 s
Resumen del grupo
ModelTest
Pruebas en grupo: 12
Duración total: 3,9 s
Salidas
12 Correcta
Resultados de la cobertura de código
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_16_56.coverage 6,97% 93,03% 4,40% 0,15% 95,45%


Explorador de pruebas
183 183 0
Serie de pruebas finalizada: 12 pruebas (Superadas: 12; Con errores: 0; Omitidas: 0) ejecutadas en 6,2 s
Prueba Duración Rasgos Mensaje de error
MaterialTests (12) 3,9 s
ModelTest (12) 3,9 s
FigureTests (18) 3,9 s
SceneTest (12) 3,9 s
ClientDriverTest (12) 4,2 s
Resumen del grupo
SceneTest
Pruebas en grupo: 12
Duración total: 3,9 s
Salidas
12 Correcta
Resultados de la cobertura de código
Hierarchy Covered (%Blocks) Not Covered (%Blocks) Covered (%Lines) Partially Covered (%Lines) Not Covered (%Lines)
Gonzalo_DESKTOP-9C94II2 2023-06-08 21_17_24.coverage 7,08% 92,92% 4,53% 0,15% 95,32%



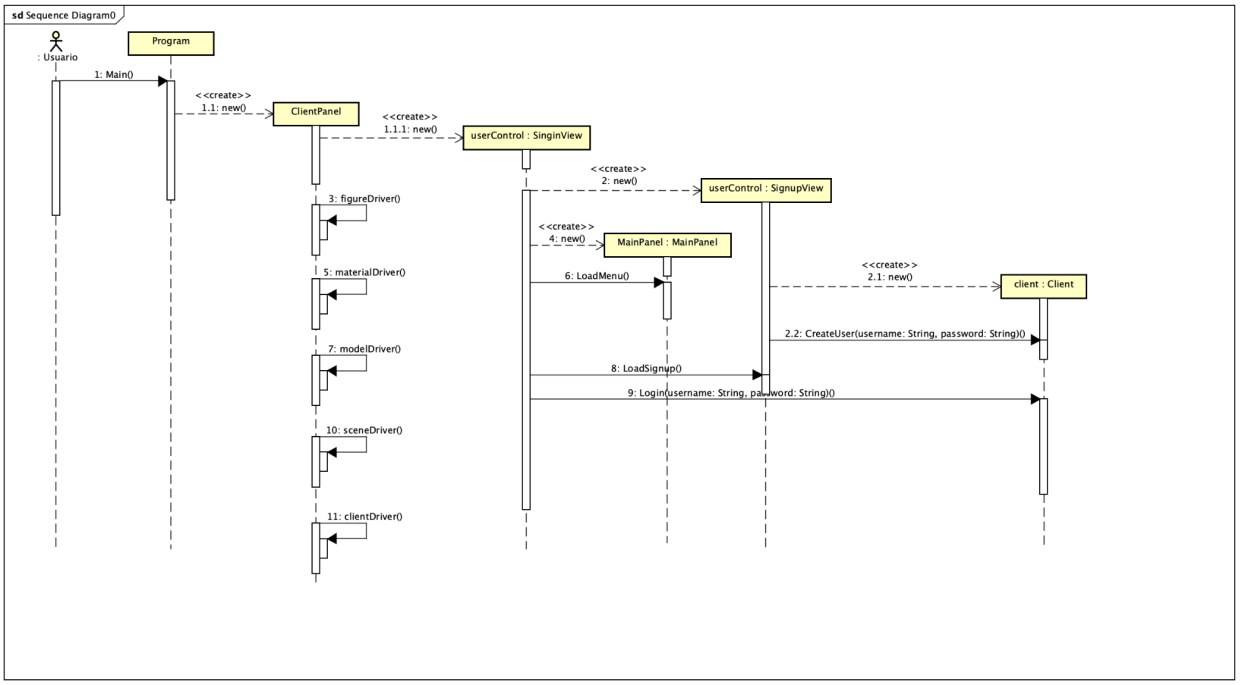




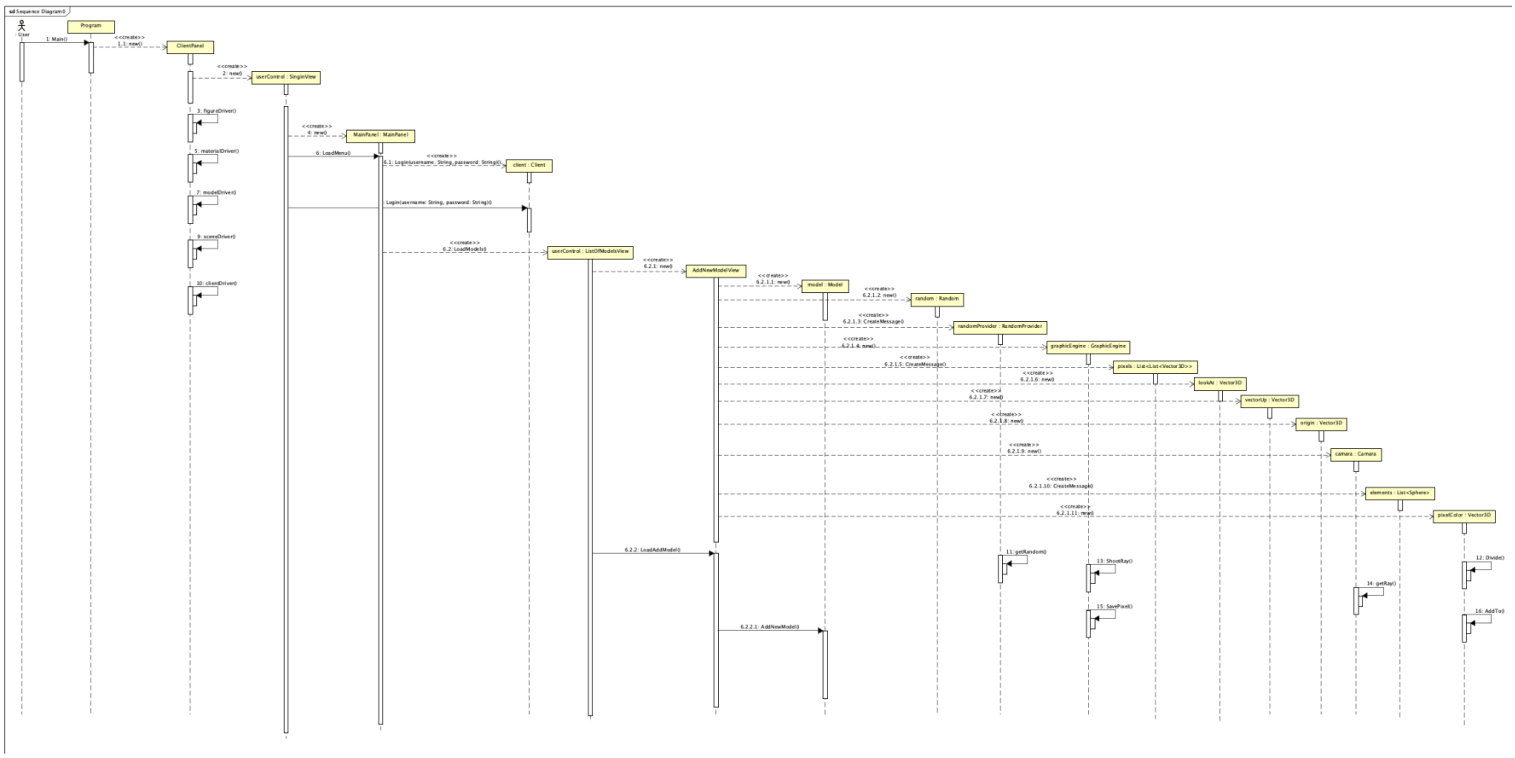
Anexo 24:

```
1 referencia | 0 cambios | 0 autores, 0 cambios
private void AddNewFigure()
{
    if (figureDriver.GetFigure(ClientPanel.usernameClient,txtFigureName.Text) == null)
    {
        try
        {
            Figure figure = new Figure()
            {
                FigureName = txtFigureName.Text,
                Radio = Double.Parse(txtFigureRadio.Text),
                FigureOwnerId = ClientPanel.usernameClient.Username,
                FigureOwner = ClientPanel.usernameClient
            };
            figureDriver.AddFigure(figure);
            MessageBox.Show("Figura agregada", "Operacion realizada", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (InvalidFigureNameException ex)
        {
            MessageBox.Show(ex.Message, "Ha ocurrido un error en el nombre de la figura", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (InvalidFigureRadioException ex)
        {
            MessageBox.Show(ex.Message, "Ha ocurrido un error en el radio de la figura", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else
    {
        MessageBox.Show("Ya existe una figura con el mismo nombre.", "Ha ocurrido un error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

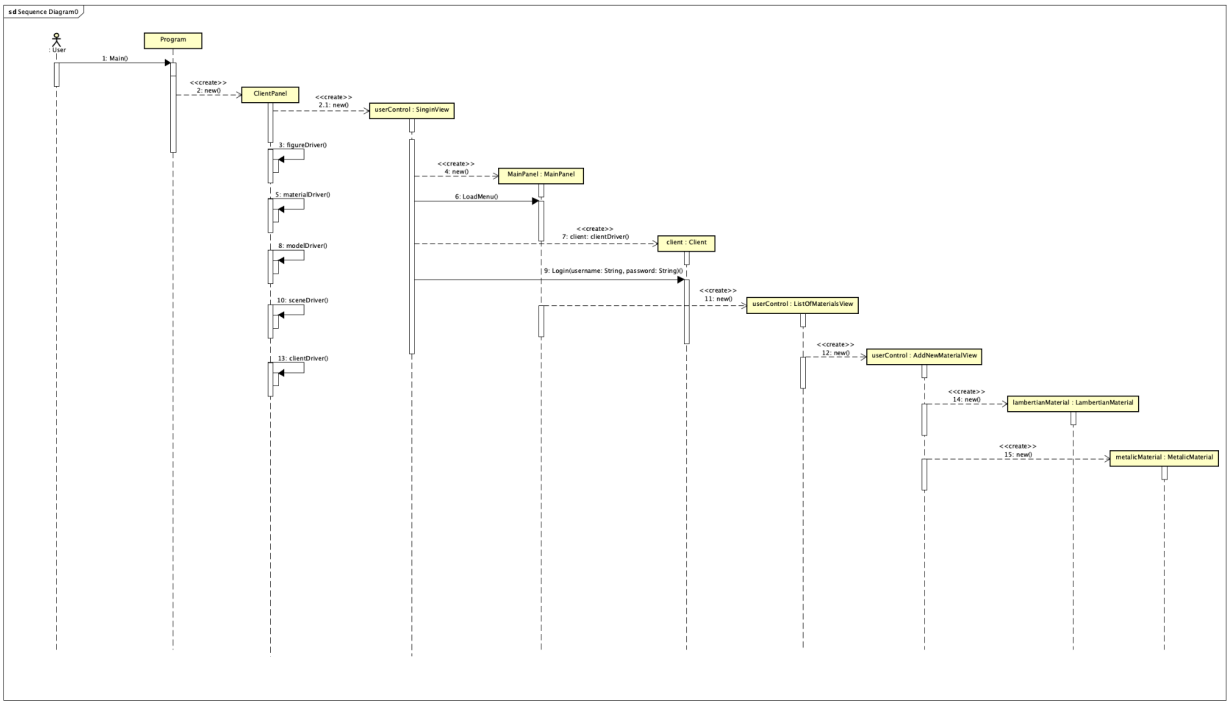
Anexo 25:



Anexo 26:



Anexo 27:



Anexo 28: