Obligatorio II de Diseño de Aplicaciones I

Requerimientos:

El matemático quedó muy conforme con la primera versión del renderizador, y ahora quiere agregarle algunas mejoras más. Para ello, se provee nuevamente scripts JS con los detalles de implementación.

Requerimientos para todos los grupos:

RF20 – Desenfoque de la cámara

La cámara de una escena ahora tendrá un desenfoque, el cual se generará simulando el funcionamiento de un lente de una cámara en la vida real.

El desenfoque se da a partir de 2 datos: la distancia focal y la apertura del lente.

En la edición de la escena, el usuario puede configurar la apertura del lente (número con precisión decimal de 1 dígito, mayor estricto a 0.0 y menor o igual a 3.0).
La distancia focal será siempre la distancia entre el look-from y el look-at (el usuario no la ingresa, sino que lo calcula el sistema).

En una escena, debe existir la opción de activar o desactivar el desenfoque.
Si el desenfoque está desactivado, el renderizado de la escena se hace utilizando la lógica de la cámara implementada en la versión anterior del sistema.
Si el desenfoque está activado, el renderizado de la escena se hace con la nueva lógica de la cámara (la cual contiene las operaciones que generan el desenfoque).

Modificar cualquier atributo de la cámara de una escena (look-at, look-from, FOV o apertura del lente) debe actualizar la fecha de última modificación de le escena.

RF21 – Material metálico

En esta versión aparece un nuevo tipo de material: el **metálico**.
Un material metálico se caracteriza por ser reflectivo. Además, el material metálico tiene un color y un difuminado.

Datos de un material metálico:

- **Propietario** (autogenerado)
  - Cliente que generó el material.
  - Autoasignado por el sistema.
- **Nombre** (requerido)
  - Único entre los materiales del cliente propietario.
  - No vacío.
  - Sin espacios al principio ni al final.
- **Color** (requerido)
  - Valores RGB (0 - 255, 0 - 255, 0 - 255).
- **Difuminado (requerido)**
  - Número decimal mayor o igual a 0.0.

Al momento de crear un material, se debe elegir qué tipo de material se está creando: Lambertiano o Metálico.

Todos los usos que se le daba al material lambertiano para la entrega anterior deben poder ser realizados utilizando el material metálico. Por ejemplo, crear un modelo con material metálico, listar los materiales metálicos, crearlos, borrarlos, etc.

En el listado de materiales se muestran ambos tipos de materiales en la misma lista. Se debe distinguir de alguna forma el tipo de cada material del listado. En caso de ser metálico, se debe mostrar además el valor de su difuminado.

En cualquier listado de modelos o materiales, se debe especificar cuál es el tipo de material.

Recuerde que en un futuro se desea seguir agregando nuevos tipos de materiales con el menor impacto en el código posible.

RF22 – Exportar a PNG/JPG/PPM

En la pantalla donde se visualiza la escena, debe existir un botón que permite guardar la escena en cualquier directorio que el usuario elija. Los formatos soportados deben ser PNG, JPG y PPM (el usuario elige el formato al cual quiere exportar la escena).

Persistencia de los datos

En esta nueva versión, se le solicita que todos los datos del sistema sean persistidos en una base de datos. De esta manera, la siguiente vez que se ejecute la aplicación se comenzará con dichos datos cargados con el último estado guardado antes de cerrar la aplicación. Los datos se deben ir guardando en la base de datos durante la operativa de la solución, a medida que el usuario realiza las acciones se debe ir almacenando, y **NO** con un botón que salve todo "en un bloque" o al cerrar la aplicación.

Toda la información contenida en el sistema debe ser persistida en una base de datos Microsoft SQL Server Express 2019. El diseño debe contemplar el modelado de una solución de persistencia adecuada para el problema utilizando Entity Framework (Code First). Se espera que como parte de la entrega se incluya dos respaldos de la base de datos: uno vacío y otro con datos de prueba. Se recomienda entregar el archivo .bak y también el script .sql para ambas bases de datos.

Importante: La entrega final debe poder persistir correctamente al menos una entidad del dominio. Este requerimiento es mandatorio para la corrección del obligatorio.

Requerimientos funcionales adicionales para grupos de tres estudiantes:

RF23 – Reporte de logs de renderizado

El sistema debe registrar un log cada vez que se ejecute un renderizado (ya sea en la generación de un preview o en el renderizado que el usuario ejecuta manualmente).

Debe existir una pantalla donde se muestre una lista con los logs de los renderizados de todos los usuarios. Un log de renderizado tiene los siguientes datos:

- Usuario
- Tiempo de renderizado en segundos (cuánto se demoró en generar la imagen).
- Fecha de renderizado (año, mes, día, hora, minuto y segundo).
- Ventana de tiempo entre la renderización anterior de la escena y la renderización que generó el log. Si es un preview, la ventana de tiempo es 0 segundos.
    - Si es menor estricto a 60 segundos, se muestra en segundos.
    - Si es mayor o igual a 1 minuto y menor estricto a 60 minutos, se muestra en minutos.
    - Si es mayor o igual a 1 hora y menor estricto a 24 horas, se muestra en horas.
    - Si es mayor o igual a 1 día, se muestra en días.
    - En todos los casos, se redondea utilizando la operación *piso*.
- Nombre de la escena renderizada (si es un preview, se pone “preview - ” seguido del nombre del objeto al cual se le está generando el preview).
- Cantidad de elementos siendo renderizados.

La pantalla debe mostrar además:

- El promedio de tiempo de renderizado en segundos (entre todos los usuarios).
- El total de tiempo de renderizado en minutos (entre todos los usuarios).
- El usuario que tiene el mayor tiempo de renderizado acumulado, junto con el valor del tiempo de renderizado acumulado de ese usuario.

Para acceder a la pantalla de logs no hace falta autenticarse.

Nota sobre requerimientos (todos los grupos):

Uso del foro

Nota: Al igual que en la entrega anterior, la totalidad y detalle de los requisitos serán relevados a partir de consultas en el foro correspondiente en aulas, las que deben tener un título descriptivo de su contenido. Tener en cuenta al momento de planificar el trabajo que las respuestas en Aulas pueden demorar hasta 24 horas. Para evitar complejidades innecesarias se realizaron simplificaciones al dominio del problema real.Las consultas en el foro se aceptarán hasta el día 9/6 a las 12:00 hrs.

Entrega del obligatorio:

Se debe entregar una aplicación que contemple toda la funcionalidad descrita en este documento. **Se espera que la aplicación se entregue con una base de datos con datos de prueba**, de manera de poder comenzar las pruebas sin tener que definir una cantidad de datos iniciales. Por este motivo, la fuente de datos de prueba no debe estar embebida en el código.

Considerar que los datos de prueba entregados deben ser completos y permitir ver reportes no triviales. Se espera que la base de datos incluya **como mínimo**:

- 3 usuarios
- 5 figuras por usuario
- 5 materiales lambertianos y 5 materiales metálicos entre todos los usuarios (usando los 3 usuarios).
- 8 modelos entre todos los usuarios (usando los 3 usuarios).
- 12 escenas entre todos los usuarios (usando los 3 usuarios) con al menos 4 modelos posicionados en cada escena, en distintas posiciones.

Instalación

El costo de instalación de la aplicación debe de ser mínimo y documentado adecuadamente. Indicando: ubicación de los ejecutables y librerías requeridas, los strings de conexión a modificar, ubicación de los backups de la base de datos, usuario inicial de la aplicación (si aplica) y cualquier otro cambio y/o artefacto requerido para que la aplicación ejecute adecuadamente.

Mantenibilidad (se mantienen los requerimientos de la primera entrega)

La propia empresa eventualmente hará cambios sobre el sistema, por lo que se requiere un alto grado de mantenibilidad, flexibilidad, calidad, claridad del código y documentación adecuada. Por lo que el desarrollo de todo el obligatorio debe cumplir:

➔ Estar en un repositorio Git.

➔ Haber sido escrito utilizando TDD (desarrollo guiado por pruebas) lo que involucra otras dos prácticas: escribir las pruebas primero (Test First Development) y refactoring. De esta forma se utilizan las pruebas unitarias para dirigir el diseño.

➔ Cumplir los lineamientos de Clean Code (capítulos 1 al 10, y el 12), utilizando las técnicas y metodologías ágiles presentadas para crear código limpio.

➔ Favorecer a los principios y patrones de diseño vistos en el curso (entre ellos SOLID, GRASP, etc).

Pruebas unitarias (se mantienen los requerimientos de la primera entrega)

Se requiere escribir los casos de prueba automatizados con el framework de unit tests visto en el curso, documentando y justificando las pruebas realizadas. Para hacer pruebas F.I.R.S.T. usando BD, y al no conocer herramientas de *mocking* de base de datos, se recomienda contar con una BD vacía y el setup reconstruir el estado para realizar las pruebas.

Como parte de la evaluación se va a revisar el nivel de cobertura de los pruebas sobre el código entregado, por lo que se debe entregar un reporte y un análisis de la cobertura de las pruebas justificando cada uno de los valores obtenidos de acuerdo al diseño elaborado para la solución.

Control de versiones (se mantienen los requerimientos de la primera entrega)

La gestión del código del obligatorio debe realizarse utilizando el mismo repositorio Git de GitHub (github.com) perteneciente a la organización de la cátedra ORT-DA1 (github.com/ORT-DA1), apoyándose en el flujo de trabajo recomendado GitFlow (nvie.com/posts/a-successful-git-branching-model).

Como clientes visuales desktop para el manejo del repositorio, pueden utilizar:

➔ SourceTree (www.atlassian.com/software/sourcetree) el cual incorpora GitFlow (www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow).

➔ Git Bash (https://git-scm.com/downloads).

➔ Github Desktop (https://desktop.github.com/)

➔ Visual Studio with Git (https://docs.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio).

Documentación

La documentación entregada debe ser en **un solo** documento digital (**límite 20 páginas, sin contar anexos)**, que contenga la siguiente información ordenada e indexada:

1. Carátula de documento como indica el documento 302. **Debe incluirse un link al repositorio de la entrega.**

2. Descripción general del trabajo y del sistema: si alguna funcionalidad no fue implementada o si hay algún error conocido (bug) deben ser descritos aquí.

3. Descripción y justificación de diseño. Para ello se debe incluir:
    a. Diagrama(s) de paquetes mostrando la organización general de la aplicación (no es necesario incluir paquetes de pruebas).
    b. Diagramas de clases: **al menos uno** por paquete. Los diagramas deberán ser lo más completos y formales posible respecto a lo que se desea comunicar.
    c. Al menos 3 diagramas de interacción para especificar los principales comportamientos del sistema. La evaluación de este punto incluye la correcta selección de las funcionalidades a mostrar con este tipo de diagramas. Es decir, se deben elegir funcionalidades que sean relevantes al diseño y al sistema.
    d. Modelo de tablas de la base de datos (tal cual lo brinda SQL Server Management Studio). Se debe discutir si hubo alguna decisión particular que hizo que el diseño de las clases difiere de las tablas finales.
    e. **Explicación de los mecanismos generales y descripción de las principales decisiones de diseño tomadas**. A modo de ejemplo, se debería describir la forma en que la interfaz de usuario interactúa con el dominio, cómo se almacenan los datos, el manejo de errores y excepciones, o la utilización de polimorfismo.
    f. Breve **análisis de los criterios seguidos para asignar las responsabilidades**.

4. Cobertura de pruebas unitarias con su debido análisis y justificaciones. En caso de que la cobertura de líneas de código no supere el 90% para alguna funcionalidad, se deberá incluir un análisis sobre porque, explicando que faltó en los test para que no se ejercitara la parte de código no cubierto. (De entregar evidencia la misma puede ir en anexos por fuera del límite de páginas)

5. Se debe **actualizar todos los diagramas que corresponda**, y agregar los que se considere necesario para mostrar los cambios realizados y justificar las decisiones tomadas.

   Para esta segunda entrega es necesario actualizar la documentación de diseño de la primera entrega.

Importante: se espera que la descripción y justificaciones sigan un orden lógico, **intercalando diagramas y explicaciones según sea necesario**. Las condiciones de entrega serán evaluadas como si se le estuviese entregando a un cliente real: **prolijidad, claridad, profesionalismo**, etc.

Entrega:

En el sistema de gestión:

La entrega de la **documentación** se realiza en el sistema de gestion.ort.edu.uy

Se deberá entregar SOLO la documentación digital en formato PDF (incluyendo modelado UML dentro del documento). Si hay diagramas poco legibles en la documentación se puede agregar una carpeta con los diagramas o un archivo Astah que permita a los docentes leer los diagramas con mayor facilidad, así lo desean.

En Github:

El commit final del obligatorio debe estar claramente identificado (mediante un tag con el texto "**entrega_2**") en la rama main en el repositorio del grupo de GitHub dentro de la organización ORT-DA1. Debe incluir:

- Código fuente de la aplicación, incluyendo el proyecto que permita compilar, probar, ejecutar y ejecutar pruebas unitarias.
- Carpeta con la aplicación compilada en release.
- Documentación digital en formato PDF (incluyendo modelado UML).
- Base de datos:
  - Entregar una base de datos vacía y otra con datos de prueba.
  - Scripts de generación de tablas y/o datos.

- No se aceptará que a main se suba un archivo comprimido con la entrega final (ej: Zip, Rar, otros), el contenido de main deberá ser la integración de la rama develop a la main.

Rúbrica de evaluación:

Puntaje total: 30 puntos.

	Evaluación de	Pts.	Criterios de corrección
Funcionalidad	Implementación de la funcionalidad pedida y calidad de la interfaz de usuario.	9	**Excelente:** Se implementa toda la funcionalidad, tiene buena usabilidad y no hay errores importantes de funcionamiento. **Correcto:** Falta algún punto no central de la funcionalidad, existen detalles no bloqueantes de funcionamiento o la aplicación no es fácil de usar. **No suficiente:** Faltan partes importantes (por alcance o dificultad) de la funcionalidad. Existe gran cantidad de errores o funcionalidades no usables.
Diseño y documentación	Diagramas de clases Diagramas de interacción Justificación del diseño Uso adecuado y justificado de principios de diseño Diagramas y justificación de la solución de persistencia. Modelo de tablas. Descripción de cambios importantes sobre el diseño de la primera entrega. Calidad del diseño Informe de cobertura de los casos de prueba Claridad de la documentación Organización de la documentación (debe tener un orden lógico y un índice) Completitud de la documentación Prolijidad de la documentación	11	**Excelente**: Se presenta un documento completo en función de lo que se pide, ordenado y prolijo, que explica la solución entregada en todos sus aspectos. Se utiliza la notación vista en clase, de manera formal y correcta, para presentar los aspectos importantes del diseño. Se intercalan diagramas y explicaciones, que permiten comprender el diseño de la solución, las decisiones tomadas y la motivación detrás de las mismas. **Correcto**: Se presenta un documento prolijo que describe el diseño de la solución. Se describen los aspectos más importantes, pero hay errores en la nomenclatura. Falta describir aspectos importantes. El orden de la documentación no permite seguir un hilo descriptivo. **No suficiente**: Falta detallar parte importante de la solución. No se usa la notación vista en clase, o se usa en forma equivocada o incompleta en repetidas ocasiones. Se presenta la documentación como una sucesión de diagramas sin explicación.
Calidad del código y metodología de desarrollo	Desarrollo guiado por las pruebas (TDD) y técnicas de refactoreo de código Buenas prácticas de estilo y	10	**Excelente:** El código es prolijo, fácil de entender y cumple con los puntos cubiertos en el curso sobre Clean Code. Lo construido coincide con lo detallado en la documentación de diseño. Se evidencia el uso

	codificación y su impacto en la mantenibilidad (Clean Code) Correcto uso de las tecnologías Claridad del código Concordancia con el diseño Implementación de acceso a la base de datos		de TDD mediante los commits en el repositorio. Hay buena cobertura de casos de prueba. El acceso a la base de datos está bien implementado y se evitan las operaciones con grandes volúmenes de datos en memoria. **Correcto:** El código es prolijo en general, aunque presenta detalles a mejorar. Lo construido se corresponde en términos generales con lo presentado en el documento. Hay pruebas unitarias, aunque no se evidencia el uso de TDD. **No suficiente:** El código es desprolijo o difícil de entender. No cumple en repetidos casos lo propuesto por Clean Code. No hay pruebas unitarias.

Demo al cliente:

La demo al cliente del trabajo intenta:

➔ Evaluar el conocimiento general de los integrantes del grupo sobre la solución propuesta. Todos los integrantes deben conocer toda la solución.

➔ Verificar el aporte individual al trabajo por parte de cada uno de los integrantes del equipo y en función de los resultados, se podrán otorgar distintas notas a los integrantes del grupo.

Se espera que cada uno de los integrantes haya participado en la codificación de parte significativa del obligatorio. El mecanismo de defensa consistirá en una demo del producto por parte de los integrantes del equipo y una sección de preguntas y respuestas.

Aspectos importantes sobre la defensa:

- Al momento de la defensa deben tener la **demo preparada**. Funciona como prueba de aceptación por parte del cliente, presentan su producto final al cliente mostrando consideraciones que tuvieron durante la implementación, manejo de errores, diseño de la solución, qué funciona y qué no, etc. Es decir, todas las funcionalidades que no

son demostradas por parte del equipo, se tomarán como **no implementadas**, lo que llevará a la pérdida de esos puntos.

- Se espera que la demo tenga una duración de aproximadamente **15 minutos**.
- Deben usar los **datos de prueba que entregaron**, para que funcionalidades tales como reportes brinden resultados interesantes. Si no entregaron datos de prueba, deben crearlos para la demo y aclararlo en el momento.
- La demo **NO** es para arreglar y ver cómo lo hacen funcionar, sino para mostrar lo que entregaron y funciona. Por lo tanto, se **ejecuta desde el release** con lo que entregaron, y no se permite cambiar código para que funcionen estos casos.
- La **calidad de la demo** afecta el puntaje de la funcionalidad. Una demo con preparación pobre **implicará pérdida de puntos**.
- La mecánica de la demostración al cliente se definirá oportunamente en cada dictado pudiendo ser **en forma virtual o presencial.** La **no presentación a la demo** sin aviso previo implica **pérdida de puntos, pudiendose llegar a perder la totalidad de los puntos del obligatorio**.
- La solución debe estar instalada y funcionando al momento de comenzar la demo. Deben estar listos para la llamada con **al menos 10 minutos de anticipación**. Es decir, si a mi equipo le toca a las 18:30hs, a esta hora **comienza** la defensa. Por lo tanto, deben estar preparados y listos a las 18:20.
- En caso de que algún grupo necesite cambiar el horario, debe coordinar directamente con algún otro grupo y notificarnos antes del día de la demo.
- Es importante que **prueben el software con tiempo y con la copia de lo que entregaron**, así no tienen problemas al momento de la demo.
- Ambos estudiantes deben tener **abierta la solución de su aplicación en Visual Studio** (**NO para ejecutar**, sino para mostrar código cuándo se le solicite).
- Deben estar todos los integrantes del grupo en su horario.

NOTA: El incorrecto funcionamiento de la instalación puede significar la no corrección de la funcionalidad. En el caso de defensa en el laboratorio, durante la defensa cada grupo contará con 15 minutos para la instalación de la aplicación. Luego de transcurridos los mismos se restan puntos al trabajo.

Información Importante

Lectura de obligatorio:	11-05-2023
Plazo máximo de entrega:	14-06-2023
Defensa:	A definir por el docente

Puntaje mínimo / máximo: 10 / 30 puntos

LA ENTREGA DE LA DOCUMENTACION SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR O PDF.

IMPORTANTE:

- Inscribirse.
- Formar grupos de hasta tres personas.
- Subir el trabajo a Gestión antes de la hora indicada, ver hoja al final del documento: “RECORDATORIO”.

RECORDATORIO: IMPORTANTE PARA LA ENTREGA

➢ *Obligatorios (Cap.IV.1, Doc. 220)*

La entrega de los obligatorios será en formato digital online, a excepción de algunas materias que se entregarán en Bedelía y en ese caso recibirá información específica en el dictado de la misma.

Los principales aspectos a destacar sobre la ***entrega online de obligatorios*** *son:*

1. La entrega se realizará desde gestion.ort.edu.uy

2. Previo a la conformación de grupos cada estudiante deberá estar inscripto a la evaluación. ***Sugerimos realizarlo con anticipación.***

3. ***Uno de los integrantes del grupo de obligatorio será el administrador del mismo*** *y es quien formará el equipo y subirá la entrega*

4. Cada equipo debe entregar ***un único archivo en formato zip o rar*** *(los documentos de texto deben ser pdf, y deben ir dentro del zip o rar)*

5. El archivo a subir debe tener ***un tamaño máximo de 40MB***

6. Les sugerimos ***realicen una 'prueba de subida' al menos un día antes****, donde conformarán el 'grupo de obligatorio'.*

7. ***La hora tope para subir el archivo será las 21:00*** *del día fijado para la entrega.*

8. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc)

9. Aquellos de ustedes que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor pasar por la oficina del Coordinador o por Coordinación adjunta antes de las 20:00hs. del día de la entrega.

Si tuvieras una situación particular de fuerza mayor, debes dirigirte con suficiente antelación al plazo de entrega, al Coordinador de Cursos o Secretario Docente.