Tarea 2: 1080 Snowboarding

Introducción

En esta tarea usted debe implementar una versión simplificada del exitoso juego de Nintendo 64: 1080 Snowboarding. Algunos dirán que se trata de una simple copia 3D de SkiFree, el equipo docente no comentará al respecto. El video en youtube muestra las características de gameplay de este videojuego.

https://www.youtube.com/watch?v=we6_iAGztJE

Deberá elegir entre Ogre[1] y Mona Engine[2] para desarrollar este trabajo. Ogre es un Rendering Engine 3D que ha evolucionado desde el año 2001, tiene muchas características interesantes, como administrador de assets, generación de sombras, partículas, un grafo de escena. Se ha utilizado comercialmente en videojuegos. Por otro lado, Mona Engine fue desarrollado por Byron Cornejo (exalumno dcc) para facilitar la enseñanza de este mismo curso. Tiene menos características visuales, pero añade otras como modelo de objetos, física y audio 3D. Mona es una base de código de menor escala, por lo que puede ser mucho más fácil para trabajar en esta etapa.

Debe organizar su proyecto con CMake y submódulos git para bibliotecas de terceros grandes. Utilice C++20. Su proyecto debe ser capaz de compilarse con Visual Studio Community 2022 y ejecutarse en el sistema operativo Windows.

[1] https://www.ogre3d.org
[2] https://github.com/dantros/MonaEngine

Consideraciones

Su repositorio debe utilizar git-lfs para versionar archivos binarios en su repositorio github privado.

Jugabilidad

El personaje jugador del videojuego original está modelado como un muñeco articulado, aún no revisamos ese contenido, por lo que queda fuera del alcance de esta tarea. Usted debe proponer un reemplazo, por ejemplo: un modelo estático de una caja, un pingüino, *billboards* 2D (en estilo Mario Kart 64).

Defina usted al menos 2 tipos de obstáculos, de un tipo deben ralentizar y del otro acelerar. Ambos modificadores aplican solo por un tiempo.

Se gana la partida al llegar a la meta dentro de determinado tiempo, se pierde si transcurre dicho tiempo y aún no se ha llegado.

El control del personaje se debe realizar en 3era persona vía teclado/mouse y joystick. Si no posee joystick para probar su solución, notifíquelo vía email al profesor lo antes posible (plazo máximo: 1 semana antes de la fecha de entrega). El código relacionado, escríbalo, pero déjelo comentado, solo se revisará el espíritu de la lógica en este caso.

No debe utilizar un framework/biblioteca de física para esta tarea, sino implementar el movimiento dentro del proceso de update o tick del videojuego. Para esto, puede utilizar una función de elevación del suelo $z = f(x, y)$ y así posicionar correctamente el personaje y la cámara.

Controles esperados:

- Teclado y mouse: WASDQE, flechas del teclado. Mouse y sus 2 botones principales.
- Joystick: Análogo izquierdo, botones ABXY (o equivalente según joystick de xbox)

Especifique en el archivo README.md de su repositorio como se utilizan los distintos controles. Intente emular el videojuego original, pero con las simplificaciones que usted estime necesarias. Si utiliza otros botones, o botones adicionales, debe especificarlos claramente en el mismo archivo.

Audio

Debe agregar sonidos para distintas las interacciones, considere al menos 4 para situaciones interesantes que puedan ocurrir (ejemplo: chocar) justo con música de fondo.

En el caso de Ogre, deberá utilizar una biblioteca externa para el manejo de audio, escoja entre Miniaudio[3] y RustyAudio[4].

Escenario

Debe modelar la elevación del terreno utilizando un script Python o Lua, y desde ahí generar un asset en formato compatible. Los scripts y archivos exportados deben ser versionados.

En el caso de Python, pueden serle de utilidad los códigos ex_height_plotter.py y ex_openmesh_pyramid.py del repositorio https://github.com/dantros/grafica , puede ignorar OpenGL y centrarse solo en la generación de vértices y el uso de openmesh.

En cualquier caso, quizá la manera más simple de generar y leer el mapa de elevaciones es utilizando un archivo csv y alguna biblioteca C++ como https://github.com/d99kris/rapidcsv . Si el personaje se encuentra en una celda, la interpolación lineal de los 4 vertices de la misma, le dará la posición x, y, z para el personaje.

[3] https://miniaud.io/
[4] https://github.com/dantros/rusty_audio

Entregables

- Archivo adjunto autoevaluación.txt
    - Comente en una tabla como la mostrada en la sección “puntajes”, que ítems le faltaron y el puntaje que usted cree que tendría. Cada categoría tendrá 0, mitad o puntaje completo según completitud.
    - Si no adjunta autoevaluación, no tendrá derecho a reclamo.
- Enlaces en la descripción de la entrega:
    - Link a repositorio GitHub **privado**.
        - No incluya archivos de Visual Studio en su repositorio, el equipo docente los generará desde CMake.
        - Bibliotecas externas de gran tamaño deben ser incluidas como submodulos git.
        - Deberá invitar como colaboradores a todo el equipo docente. Los usuarios GitHub del equipo docente se encuentran en u-cursos/Novedades.
        - Compruebe que su repositorio puede ser descargado y compilado apropiadamente.
    - Video o link a video de instrucciones y demostración.
        - Recuerde que u-cursos posee un límite al tamaño máximo de archivos.

Importante

- El código de su tarea no debe ser compartido de ninguna forma durante el periodo de desarrollo de esta tarea. Por ejemplo, **no debe almacenar su código en un repositorio GitHub público**. Siempre puede debatir ideas y algoritmos, pero no puede compartir código.
- Tome decisiones razonables para obtener un prototipo funcional, el enunciado no posee letra chica. En cada categoría de puntaje se otorgará 0, mitad o completo, según criterio de completitud.

Puntajes

	Software	**Puntaje**
P1	Control del juego a través de teclado/mouse y joystick.	0.8
P2	Decoración y gráficos de escenario, incluyendo modelo de montaña de descenso texturizado, skybox, sombras (solo Ogre). Videojuego debe verse inmersivo.	0.8
P3	Movimiento de cámara que persigue al personaje.	0.6
P4	Efectos de sonido y música de fondo (solo 2D).	0.4
P5	Organización efectiva del proyecto utilizando git, git-lfs, cmake, y distintos archivos .h y .cpp. Utiliza un modelo de objetos de juego razonable.	0.8
P6	Colisiones con el mapa y modificadores de velocidad/control. Condiciones de inicio y término de juego (ganar y perder).	0.6
P7	Administrador de recursos e instalación. Su programa debe ser portable, es decir, una vez compilado el proyecto, al mover la carpeta ‘install’, todo deberá seguir funcionando.	0.6
P8	Ejercicio 3 implementado en Ogre o MonaEngine (debe usar el sistema distinto del utilizado en esta tarea)	0.6

	Video	**Puntaje**
P9	Se debe apreciar la generación de archivos de Visual Studio vía CMake, la compilación y la ejecución del videojuego, y la presentación de las funcionalidades implementadas. El equipo docente seguirá paso a paso este video para revisar si su tarea funciona. Edite el video para que en ningún caso exceda los 5 minutos, pero mientras más breve, mejor.	0.8

** Los modelos pueden ser hechos en Blender o generados vía código escrito Python, Lua o C++. Debe adjuntar dichos archivos fuente. La expectativa es algo simple y low poly, no se complique con modelos detallados. También puede utilizar modelos disponibles en internet, en dicho caso, debe especificar las licencias y atribuciones correctamente.