Tarea 1: OpenGL2D

Universidad de Chile **CC3501**
FCFM - DCC **1 de abril de 2020**

Instrucciones Generales

- La tarea es estrictamente individual.
- Este enunciado presenta varias opciones, escoja e implemente solo una de ellas.
- El plazo de entrega se indica en tareas/u-cursos.
- Esta tarea debe ser trabajada y entregada en un repositorio Git remoto privado, proporcionando acceso al equipo docente. Instrucciones detalladas disponibles en material docente.
- Guarde todo su trabajo para esta tarea en una carpeta de nombre tarea1x, con x indicando la opción que haya escogido.
- DEBE utilizar: Python 3.5 (o superior), Numpy, OpenGL core profile, GLFW.
- *Las imágenes presentadas son solo referenciales, utilice un estilo propio para su trabajo.*
- A pesar de lo dicho en la primera clase, esta tarea no incluye lectura. La tarea 2 sí incluirá.

Entregables

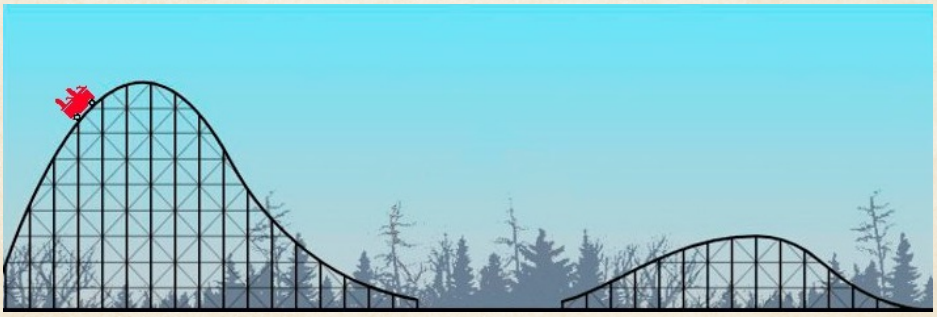
- Código que implemente su solución (4.5 puntos)
  - Su versión final DEBE utilizar archivos *.py, NO jupyter notebooks.
  - Incluya TODO lo que su código necesita, incluyendo archivos proporcionados en cátedras o auxiliares.
  - Al menos 5 commits en su repositorio git remoto ilustrando progreso en su trabajo. Nota 1 si no cumple con este item
  - *Importante: el mínimo de 5 commits deben corresponder a progresos en el código. Agregar reportes o videos al repositorio no contabiliza.*
- Reporte de documentación de entre 1 y 2 planas. Formato pdf. Detalles disponibles en primera clase. (1.2 puntos)
- Video demostrativo de 20-30 segundos. (0.3 puntos)
- Todo lo anterior debe estar disponible en su repositorio Git remoto.

Objetivos

- Ejercitar el uso de OpenGL core profile en una aplicación en 2 dimensiones simple.
- Dominar el uso de matrices de transformación, modelación jerárquica, texturas y curvas.
- Hacer uso del patrón de diseño Modelo-Vista-Controlador.
- Trabajar con interacciones de usuario vía GLFW.
- Implementar una animación simple.
- *Nota: No todas las opciones de tareas ejercitan TODOS los objetivos.*

Opción A: Roller Coaster

Se le pide construir una montaña rusa capaz de cambiar de forma. Primero se le exige hacer una simulación gráfica en 2 dimensiones.



Su programa se debe ejecutar con la siguiente llamada:

```
python roller-coaster.py track.csv
```

Donde el archivo track.csv posee 2 números separados por comas en cada línea, con un ejemplo de archivo csv a continuación:

```
0,4
1,6
2,4
3,3
4,1
x5,0
6,1
```

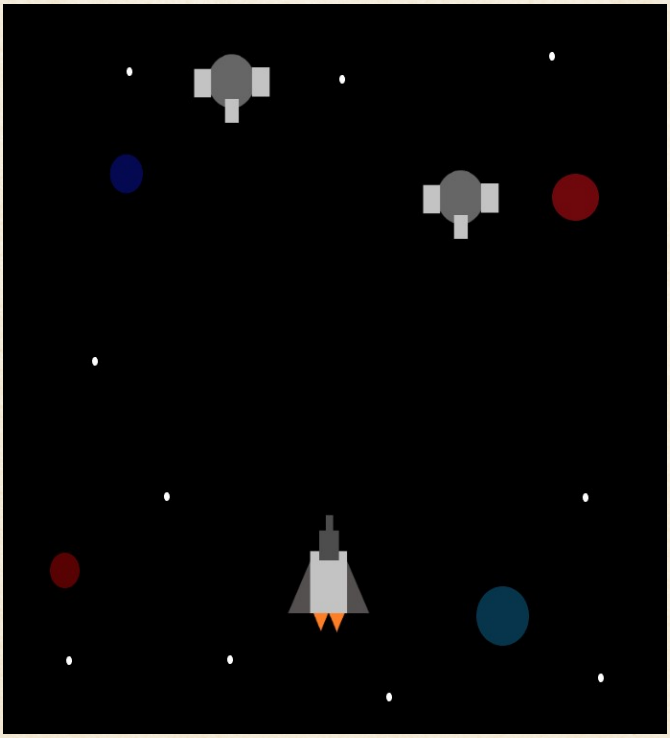
- Su programa debe leer y cargar el archivo track.csv con la descripción de la pista del roller-coaster, tal que pase por esos puntos en los ejes x e y, respectivamente.
- La pista debe estar modelada con curvas.
- Debe modelar un carrito que navegará por el roller-coaster. Este debe tener un nivel razonable de detalle y moverse a lo largo de la montaña rusa automáticamente sin control.
- Muy posiblemente la pista no cabrá completamente en la escena, por lo tanto debe resolver este problema. Por ejemplo, puede mover la pista hacia atrás, mientras que el carrito se trasladará verticalmente y rotará según la inclinación de la pista.
- Por simplicidad, no se preocupe de la velocidad neta del carro. Suponga que es constante en el eje x. Otros supuestos también son válidos.
- El carrito debe moverse a lo largo del circuito. Si está en una parte plana, entonces el vagón está horizontal. Si está en una parte vertical, entonces está vertical.
- Para agregar más emoción al juego, se han agregado secciones donde la pista está cortada. Estas partes se representarán con una x al inicio de la línea en el archivo track.csv en el punto correspondiente. Cuando esté cortada, usted debe generar un espacio vacío entre ambos puntos.
- El gerente quiere evitar muertes en el juego, así que le pide diseñar un sistema para que el vagón pueda saltar estos espacios y así brindar aún más diversión. Al presionar la barra espaciadora del teclado, el carrito debe saltar para así evitar los vacíos. Usted debe implementar el salto como estime conveniente.
- Si no se salta a tiempo y el vagón cae, se produciría una escena NOT SAFE FOR WORK que preferimos omitir, así que cierre la ventana cuando esto ocurra.
- Si bien la pista es arbitraria, esta no tendrá loops (i.e. vueltas completas).
- Utilice texturas para decorar la escena.

Puntuación

- Pista generada en base a archivo *csv*: 1 punto
- Modelos de escena y del carrito: 1 punto
- Movimiento continuo del carrito por la pista: 1 punto
- Se puede avanzar por la pista más allá de los límites de la ventana: 0.5 puntos
- Salto del carrito al presionar espacio: 1 punto

Opción B: Space War

Eres el piloto de una nave de ataque que participa en una guerra espacial. Tu deber es eliminar a todos los enemigos que se aproximen a tu nave, la cual contiene un cañón de largo alcance. Logra este objetivo y gana la guerra espacial.



Su juego se debe ejecutar con la siguiente llamada:

```
python space-war.py N
```

Siendo N la cantidad de enemigos que van apareciendo en el transcurso del juego.

Considere lo siguiente:

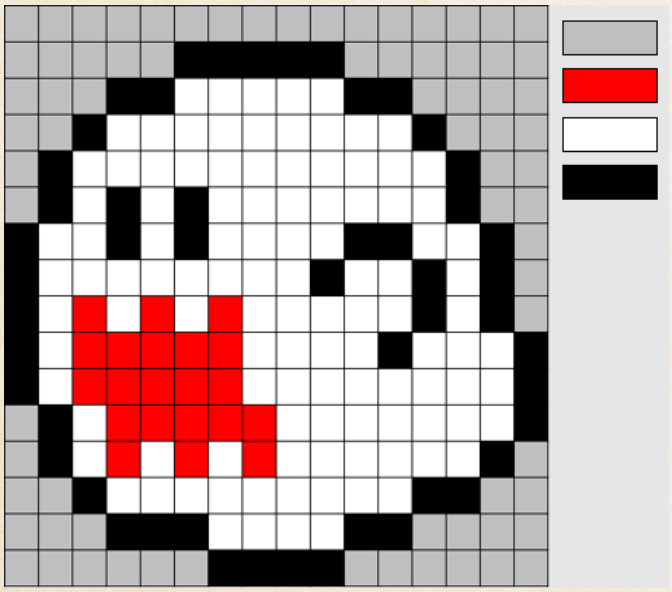
- Debe implementar un diseño de la nave y de las naves enemigas, además de crear un fondo apropiado para el juego.
- La nave debe desplazarse de forma vertical y horizontal al presionar A (izquierda), S (atrás), D (derecha) y W (adelante).
- Añada una animación de traslación al fondo para simular el movimiento de toda la escena.
- Al presionar espacio debe salir un disparo en dirección vertical a los enemigos. Si el disparo toca a los enemigos, estos se destruyen.
- Las naves enemigas deben aparecer por la parte superior de la pantalla y realizar desplazamientos horizontales y verticales, siempre disparando.
- El juego debe liberar las naves enemigas gradualmente como usted se estime conveniente.
- Si la nave recibe 3 disparos, el juego se termina, y se debe mostrar una animación básica de GAME OVER. La única forma de ganar el juego es destruyendo a todos los enemigos.

Puntuación

- Modelos: 1 punto
- Generación, movimiento y disparos de las N naves enemigas: 2 puntos
- Control de nave (movimiento y disparo): 1 punto
- Animación de GAME OVER: 0.5 puntos

Opción C: Pixel-Paint

En esta tarea desarrollaremos un software estilo paint enfocado principalmente en imágenes de baja resolución. De esta forma, será idóneo para Pixel Art.



Su programa se debe ejecutar con la siguiente llamada:

```
python pixel_paint.py 16 pallete.json boo.png
```

- El primer argumento, 16, corresponde al tamaño de la grilla cuadrada de píxeles a utilizar. En este caso generará una grilla de 16×16.
- El segundo argumento, *pallete.json* corresponde a un archivo externo donde se especifica la paleta de colores disponible para pintar. El archivo se encuentra en formato json, por lo que puede ser leído fácilmente.
- El tercer y último argumento corresponde al nombre con el cual se guardará el archivo. Siempre será de extensión *png*.

Ejemplo de archivo *pallete.json*:

```
{
    "transparent" : [0.5, 0.5 ,0.5],
    "pallete" :
    [[1, 0, 0], [1, 0, 0], [0, 0, 0]]
}
```

Considere que:

- Los valores dados son de ejemplo. Su programa debe funcionar para cualquier grilla y nombres de archivos.
- Debe utilizar solamente GLFW y OpenGL. No debe utilizar librerías adicionales para manejo de ventanas o botones.
- Debe identificar dónde se hace clic con el mouse. Si es un color, se pondrá como color activo; si es un pixel de la imagen, se pintará del color activo.
- Internamente, debe almacenar la información en una matriz cuadrada numpy.
- Al presionar la tecla s o g se deberá guardar la imagen con el nombre especificado como tercer argumento.
- El tamaño de la ventana puede ajustarse automáticamente según el valor de N.
- Asuma que existirá un máximo de 20 colores, para configurar apropiadamente la distribución de la ventana.
- En la barra de colores, el primer color corresponde al color especificado como transparente en el archivo *.json*. Al momento de guardar la imagen como *png*, este color debe ser escrito como transparente (canal *alpha*).
- Siéntase libre de utilizar otro diseño para el *layout* de su aplicación.

Puntuación

- Tamaño de grilla de píxeles especificada por el usuario: 0.7 puntos
- Paleta de colores especificada por el usuario: 0.8 puntos
- Seleccionar colores y pintar: 2 puntos
- Almacenar imagen, incluyendo transparencia: 1 punto

Opción D: Explorador de mapas

El objetivo de esta tarea es implementar un explorador de mapas que presente funcionalidades básicas de navegación y que nos permita añadir marcadores sobre él.

Su programa se debe ejecutar con la siguiente llamada:

```
python map_explorer.py map.json
```

Ejemplo de archivo *map.json*:

```
{
    "map_name" : "westeros.png",
    "landmarks" :
    [[476,687], [710,555],
    [724,355], [1005,144]]
}
```

Considere lo siguiente:

- Al hacer clic izquierdo y arrastrar el mouse, el mapa debe deslizarse.
- Al hacer clic derecho, se debe agregar un marcador numerado al mapa en la posición especificada. Si en la posición especificada ya existe un marcador previo, digamos, en un radio de 10 píxeles de pantalla, debe eliminar dicho marcador previo.
- Los marcadores numerados deben ser conectados por una línea curva suave (i.e. sin quiebres entre los tramos).
- Al mover la rueda del mouse (*scroll*), hacia adelante o hacia atrás, se debe acercar o alejar el mapa.
- Al presionar *s* se deben almacenar los marcadores en el archivo *json* especificado.
- Puede asumir que no habrán más de 99 marcadores.
- Si se elimina algún marcador, la conexión siempre se realiza en orden creciente. Ejemplo: Si se elimina el 2, luego se conecta el 1 con el 3.
- En el archivo *json*, en *landmarks*, se especifica la posición de los marcadores utilizando coordenadas de píxeles respecto del extremo superior izquierdo de la imagen. Esto es, hacia la derecha aumenta x; y hacia abajo aumenta y.
- Para renderizar los números, puede utilizar texturas, mallas de triángulos u otra solución que le acomode.

Puntuación

- Lectura de mapa y movimiento sobre él: 2 puntos
- Agregar y remover marcadores: 1 punto
- Conexión de marcadores: 0.5 puntos
- Almacenamiento de marcadores: 0.5 puntos
- Solución para dibujar números: 0.5 puntos