# Tarea 1



## 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 3.7 (o superior), Numpy, OpenGL core profile, GLFW.
- *Las imágenes presentadas son solo referenciales, utilice un estilo propio para su trabajo.*

## 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 ítem.
- Reporte de documentación de entre 1 y 2 planas. Formato pdf. Detalles disponibles en primera clase. (1.2 puntos)
- Vídeo 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: GUI para un Framework

La nueva plataforma de videojuegos **Vapor** le ha pedido que implemente la GUI (Graphical User Interface) de su nuevo sistema operativo que es compatible con todos sus juegos.

La GUI debe cumplir con los siguientes requisitos:

- Debe incluir un modelo gráfico de directorio (carpeta) que sea dibujado puramente con OpenGL, indicando sus vértices y colores. Deben ser al menos 8 vértices.
- Debe permitir tomar elementos (como directorios o aplicaciones) y arrastrarlos a otra ubicación. Esto se realiza con manteniendo presionado el click izquierdo y llevándolo al lugar deseado como se hace en otros sistemas como Windows, Android, Linux y macOS.
- Debe poder importar una única textura de algún icono de programa y desplegarse en la GUI.
- Para facilitar su tarea y limitar las aplicaciones desplegadas, se ha decidido por un modelo de grillado. Este debe ser de un tamaño variable entre 4x4 y 8x8 dependiendo de la llamada del archivo.
- La grilla anterior debe ser manipulada con un arreglo de numpy de tamaño N x N. Se le recomienda mantener una notación clara en esta grilla, tal que su programa la lea y sepa qué desplegar gráficamente. Por ejemplo, puede escribir un 0 para indicar la ausencia de un archivo en esa ubicación, un 1 para el directorio y un 2 para el archivo con textura.
- Para eliminar un archivo se debe presionar click derecho sobre este. Al hacerlo, rotará durante dos segundos y se achicará hasta desaparecer.

- Al presionar **Enter** con el puntero del mouse sobre un directorio, se debe ingresar a este, desplegando los archivos dentro suyo.
- Su programa debe leer un archivo .csv y desplegar los iconos en la gui según se indica ahí.
- No se le pide mostrar los nombres de los archivos. Puede iniciar la posición de los archivos como estime conveniente. Lo importante es que se puedan mover.
- Dos archivos NO se pueden superponer. No es necesaria la capacidad de mover archivos de un directorio a otro.

El programa se correrá con la siguiente llamada:

```
python tarea1_gui.py directories.csv N texture
```

Donde:
* tarea1_gui.py es el nombre de su script que presenta lo pedido.
* directories.csv es el nombre del archivo json que contiene la información de los directorios a desplegar.
* N es el tamaño de la grilla, que va de 4 a 8.
* texture es el nombre de algún ícono de archivo.

Un ejemplo de archivo csv:

```
"file"
"dir": ["dir", "dir"]
"dir"
"dir"
```

Esto significa que en el directorio base tenemos un archivo y tres carpetas, una de ellas tiene otras dos dentro que están vacías. El archivo debe mostrarse con la textura indicada en la llamada.

Un ejemplo de llamada para testear:

```
python tarea1_gui.py directories.csv 5 mozilla.png
```

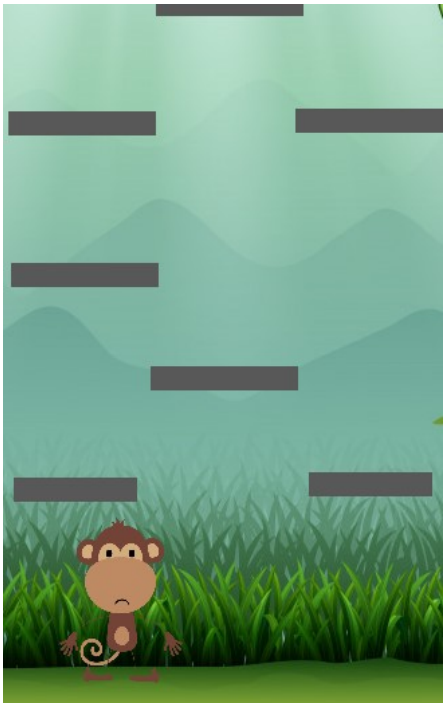
## Puntuación

- Interactividad con los archivos (Trasladar carpetas, eliminarlas, agregar su rotación): 2.2 puntos
- Capacidad de lectura del archivo csv: 0.8 puntos

- Modelo de carpeta: 0.5 puntos
- Capacidad de ajuste al tamaño N indicado: 0.5 puntos
- Capacidad de despliegue de textura: 0.5 puntos

# Opción B: Monkey Jump

Caminando por la selva un monito divisó en la cúspide de una estructura unas bananas. Con mucha hambre se propone a alcanzarlas. Ayuda al monito a subir por las plataformas hasta llegar a las bananas que se encuentran al final del recorrido.



Su juego se debe ejecutar con la siguiente llamada:

```
python monkey_jump.py structure.csv
```

Donde el archivo structure.csv posee 3 números separados por comas en cada línea, los cuales indican el patrón de la estructura. Si es un 0 indica que no existe plataforma. Si es un 1 indica que si existe plataforma.

El siguiente ejemplo de archivo csv se visualiza en la imagen anterior:

```
1,0,1
0,1,0
1,0,0
1,0,1
0,1,0
0,1,1
1,0,1
0,1,0
```

Considere lo siguiente:

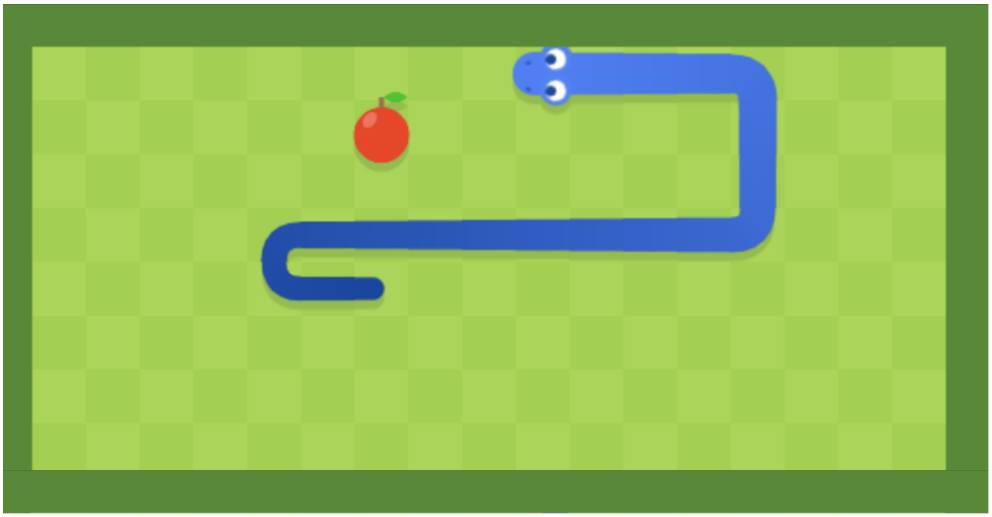
- Su programa debe leer y cargar el archivo structure.csv con la descripción de la estructura del juego. La estructura visualizada debe seguir los mismos patrones mostrados en el archivo.
- El archivo csv no debe tener filas con solo 1, ya que eso no permite el avance del monito.
- Debe implementar un diseño para el monito, para las bananas, para las plataformas y para el fondo. El diseño debe ser adecuado para el juego. (Se puede utilizar texturas)
- El fondo debe tener movimiento para que simule el avance del monito por la estructura.
- El personaje debe realizar movimientos horizontales y saltos verticales, utilizando las teclas A ,W , D.
- El personaje debe tener reacciones cuando realiza algún movimiento. Como por ejemplo al trasladarse mover manos o pies. (En el caso de texturas, puede hacer una animación básica)
- El juego termina cuando el monito llega al final de la estructura y alcanza las bananas. Si el monito cae a la base de la estructura se termina el juego. En ambos casos debe haber una pequeña animación, realizada con texturas, que indique el termino del juego.

## Puntuación

- Lectura y visualización del archivo csv: 0.7 puntos.
- Diseños dentro del juego (monito, bananas, estructura y fondo): 0.8 puntos
- Movimientos del fondo y del personaje: 1.5 puntos
- Control del personaje: 1.0 punto.
- Animación final: 0.5 puntos

# Opción C: Snake

Una serpiente con hambre infinita de manzanas crece cada vez que come una. La serpiente muere al tocarse a sí misma o tocar uno de los bordes del mapa en el cual se mueve.



Su tarea debe cumplir con lo siguiente:

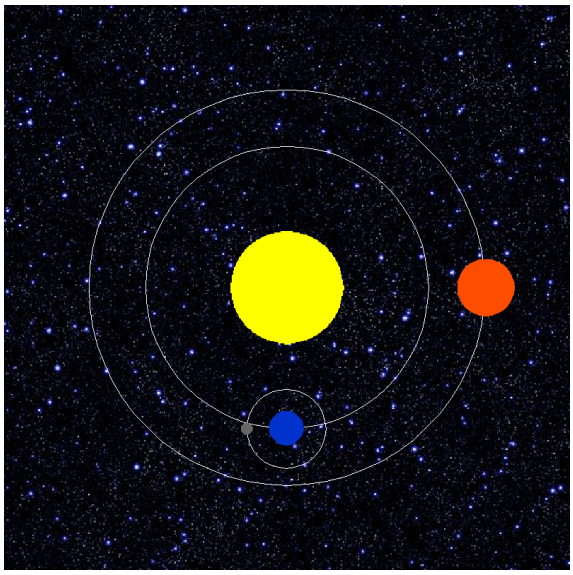
- El borde del mapa debe ser explícito y claramente distinguible del centro, por donde se desplaza la serpiente.
- La serpiente crece en un punto por cada manzana que come. Una vez comida la fruta, inmediatamente aparece otra en algún lugar del mapa donde no esté la serpiente.
- La serpiente se debe mover a una velocidad razonable. No menos de 2 movimientos por segundo y no más de 30 por segundo.
- Agregue alguna textura para la serpiente.
- El modelo de manzana debe ser implementado con OpenGL.
- Al morir la serpiente, se despliega una escena/textura que diga Game Over. Esta escena debe tener un giro parcial.
- Las dimensiones del mapa se dan en la llamada del programa, y pueden ir de un N = 10 a un N = 50. El mapa será una grilla cuadrada.
- La serpiente debe ser controlable con las teclas WASD o las flechas del teclado. Esta siempre está en movimiento, donde al presionar teclas se cambia la dirección hacia donde se dirige.

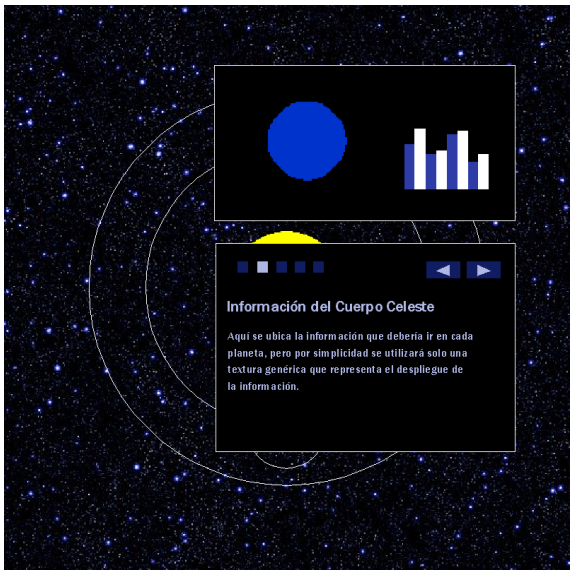
## Puntuación

- Mecánicas del juego: que la serpiente se agrande, aparezcan las manzanas, morir al tocar un borde 1.5 puntos.
- Diseños dentro del juego (serpiente, bordes y fondo): 1.0 puntos
- Movimientos y control de la serpiente: 1.5 puntos
- Animación final: 0.5 puntos

# Opción D: Sistema Planetario

Se le ha pedido crear una visualización en dos dimensiones de un sistema planetario a partir de un archivo que contenga los datos de los cuerpos celeste, además el usuario podrá interactuar con la aplicación.






La aplicación debe cumplir con los siguientes requisitos:

- Debe representar los cuerpos celestes dibujando un círculo con OpenGL con los datos de color y tamaño dado en el archivo json y una línea de la órbita en la que se desplaza.
- Tiene que generar las órbitas y la velocidad de estas según las jerarquías dadas en el archivo json.
- Se tiene que apreciar el movimiento de los planetas de acuerdo a los datos de magnitud y sentido de rotación de cada uno.
- Con las teclas W, A, S, D se debe poder desplazar en la visualización e implementar un zoom al presionar **Z** y **X** para acercar y alejar respectivamente.
- Utilizar una textura para simular el fondo del espacio o dibujar estrellas distribuidas aleatoriamente. Este fondo debe permanecer estático en la navegación.
- La aplicación permite seleccionar un cuerpo, resaltándolo con un círculo brillante detrás de él. El planeta seleccionado se cambia presionando las flechas izquierda y derecha del teclado.
- Al presionar **Enter** se debe desplegar la información del cuerpo celeste seleccionado. Con esta misma tecla se debe volver a ocultar lo desplegado.

- La información desplegada corresponde una textura genérica única para todos los cuerpos, la cual contiene un espacio para colocar el círculo que representa el planeta seleccionado y otro espacio para una pequeña animación generada con texturas.

El programa se correrá con la siguiente llamada:

```
python system_view.py bodies.json
```

Donde system_view.py es el nombre de su script que ejecuta la aplicación y el archivo bodies.json contiene una lista jerarquizada de cuerpos celestes que cumple con:

- Color que corresponde a un arreglo con el color en RGB.
- Radius que es el radio del cuerpo celeste.
- Distance es la distancia al cuerpo padre.
- Velocity es la velocidad de traslación con respecto a su padre, y su signo indica el sentido de rotación (sistema horario)
- Satellites corresponde a una lista de cuerpos celeste que orbitan alrededor, si no tiene se agrega "Null".
- El primer cuerpo debe representar la estrella del sistema que se ubicara en el centro, por lo que los datos de Velocity y Distance se ignoran en este caso.

Un ejemplo de archivo json:

```
[
    {
        "Color": [ 1, 1, 0 ],
        "Radius": 0.1,
        "Distance": 0.0,
        "Velocity": 0.0,
        "Satellites": [
            {
                "Color": [ 0.0, 0.2, 0.8 ],
                "Radius": 0.03,
                "Distance": 0.25,
                "Velocity": 0.4,
                "Satellites": [
                    {
                        "Color": [ 0.4, 0.4, 0.4 ],
                        "Radius": 0.01,
                        "Distance": 0.07,
```

```
                    "Velocity": -0.2,
                    "Satellites": "Null"
                }
            ]
        },
        {
            "Color": [ 1, 0.3, 0.0 ],
            "Radius": 0.05,
            "Distance": 0.35,
            "Velocity": 0.5,
            "Satellites": "Null"
        }
    ]
}
]
```

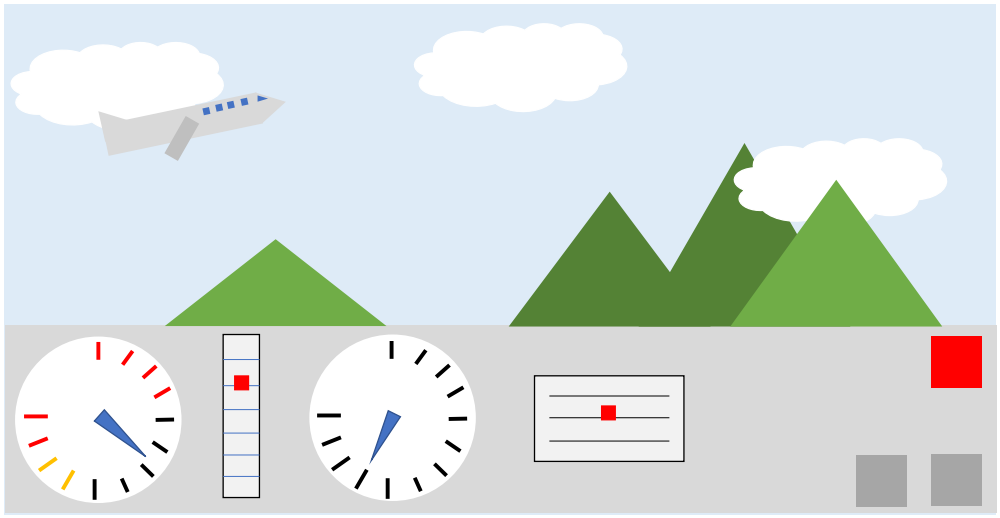
Esto representa una estrella con dos planetas, donde el primero de ellos tiene un satélite.

## Puntuación

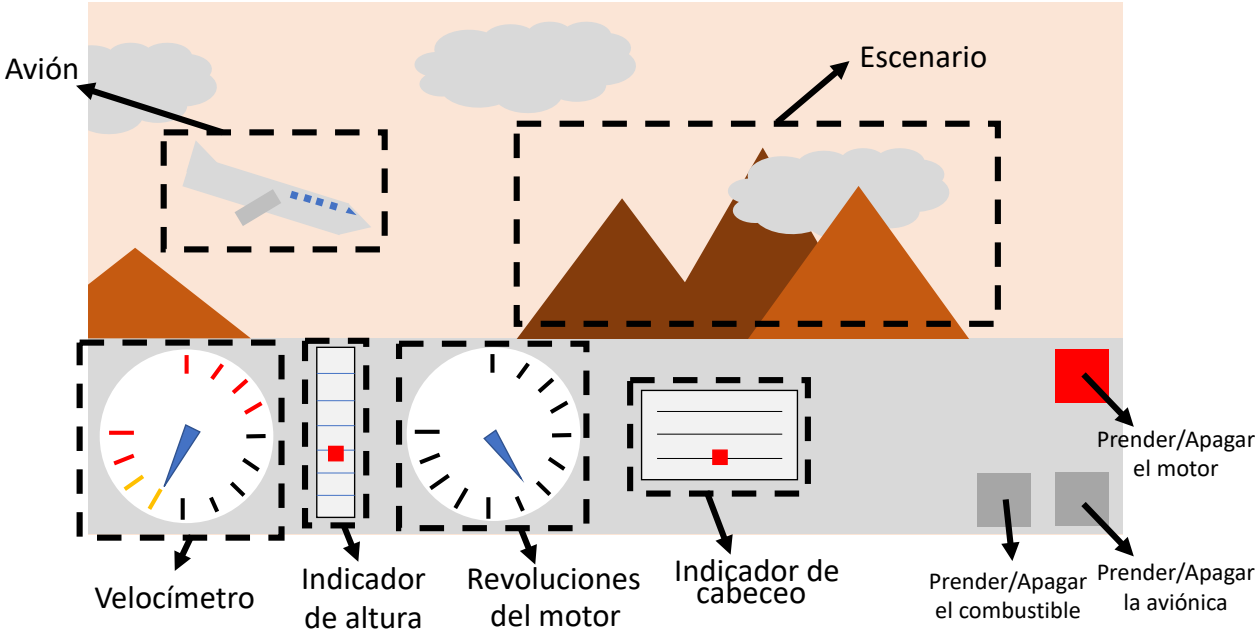
- Capacidad de lectura del archivo json: 0.8 puntos
- Dibujo y movimiento de los cuerpos celestes: 1.5 puntos
- Interactividad con el usuario, navegación, zoom, selección de planetas: 1.2 puntos
- Despliegue de la información con una textura y el cuerpo seleccionado: 0.5 puntos
- Animación en la información desplegada: 0.5 puntos

# Opción E: CC3501 Flight Simulator

A modo de competir con los gigantes de la simulación de vuelo como *Microsoft Flight Simulator* (2020) o *XPlane* en esta tarea se pide implementar su propio simulador de vuelo, un poco más reducido, que le permita controlar un avión en 2D y desplegar la información de vuelo en un panel (aviónica) que incluya la velocidad, altitud, cabeceo (inclinación), velocidad del motor, además de algunos controles para prender y apagar el avión o el panel.



La siguiente figura describe los principales elementos de la escena:



La aplicación debe cumplir con los siguientes requisitos:

- La vista debe estar compuesta por una visualización de la escena y el panel inferior de la aviónica.
- La escena debe estar compuesta al menos de nubes y cerros creados con las primitivas de OpenGL/Texturas. Estas deben moverse hacia la izquierda a medida que el jugador avanza para dar el efecto de movimiento. Note que puede proponer nuevos elementos en la escena como edificios, un sol, o estrellas.
- El avión debe estar dibujado sólo con primitivas de OpenGL, este debe ser capaz de rotar en su conjunto según la posición de cabeceo del avión (giro en el plano para indicar que gana o pierde altitud) y debe poder moverse en el eje z para similar la altura. El modelo de la Figuras anteriores esta compuesto de 9 primitivas (triángulos y cuadrados). La forma del avión la puede proponer ud.
- El panel inferior debe desplegar la aviónica. Esta, al menos, debe incorporar lo siguiente:
  - Velocímetro: Indica la velocidad del avión (*knots*).
  - Indicador de altura: Es una barra segmentada que contiene un punto rojo al medio que indica la altura del avión.
  - Revoluciones del motor: Velocidad en RPM del motor.
  - Indicador de cabeceo: Indica el ángulo del avión con respecto al plano principal de vuelo. Indica si se gana o pierde altura.
  - Botones: Al menos el panel debe ofrecer tres botones, prender y apagar el motor, prender y apagar la aviónica (si está apagado ningún panel funciona), y prender y apagar el combustible.
- Debe presentar unas físicas simples:
  - El avión comienza en el suelo.
  - Al prender el motor el usuario puede acelerar o desacelerar el motor con los botones **A** y **W**. El avión sólo puede despegar si la velocidad es mayor a 50, bajo esta velocidad el avión entra en caída libre (*stall*). Si el avión gana mucha velocidad puede resultar dañado y explotar. Note los límites superior e inferior del velocímetro. Si se choca con el suelo a una velocidad muy alta el avión puede explotar y el juego termina.
  - La velocidad sólo controla la ganancia de altura, si se deja de acelerar el avión debe perder altura de manera gradual (dado el roce con el aire).
  - El cabeceo del avión puede ser controlado con la teclas **FLECHA ARRIBA** y **FLECHA ABAJO**. El cabeceo permite controlar la tasa de ganancia de altura

a velocidad constante. Ésta debe ser reflejada en la aviónica. De igual manera si el avión gana mucha altura (limite superior) el motor pierde empuje con el aire y la velocidad decae drásticamente perdiendo altura.

- El motor puede ser prendido o apagado con el botón del panel. Al estar prendido el motor el jugador puede interactuar con el motor. Además puede prender o apagar la alimentación de combustible (*mixture fuel*) y el panel (*avionics*).

Ud. sin embargo puede proponer sus propias físicas, aunque éstas deben ser consistentes. Puede basar su implementación en los simuladores reales para entender las principales mecánicas de la aviación. Además puede implementar otros botones como subir o bajar las ruedas (*landing gear*), prender o apagar baterías, etc.

- Por último, el juego debe presentar al menos dos paletas de colores para el escenario. En las figuras anteriores se ilustran dos ejemplos (día) y (tarde). Puede proponer cualquier otra como por ejemplo: noche, nublado, etc.

El programa se correrá con la siguiente llamada:

```
python simulador.py
```

El simulador debe elegir la paleta de colores de manera aleatoria, o bien ud. puede proponer algún tipo de menú en consola para realizar todos los ajustes que quiera (elegir el avión, el escenario, etc...). Sea creativo en su solución.

## Puntuación

- Dibujo del escenario de manera estática (sin moverse en la medida que el jugador avanza): 0.8 puntos
- Movimiento del escenario a medida que el jugador avanza: 0.4 puntos
- Dibujo y movimiento del avión (eje Z y rotación para el cabeceo): 1.5 puntos
- Interacción con el usuario (teclas y clicks de los botones): 0.5 puntos
- Dibujo del panel de aviónica (estático): 1.0 puntos
- Física del avión e interacción con la aviónica: 1.8 puntos