DIVISIÓN DE INGENIERÍA DE
SISTEMAS Y AUTOMÁTICA
ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID

PROGRAMACIÓN DE SISTEMAS

CURSO 2020/21

GUIÓN DEL SEGUNDO TRABAJO VERSION 2

1. Objetivos

El objetivo del trabajo de programación de la asignatura es completar, programando en lenguaje ANSI C++, un sencillo videojuego (imitación del celebre "Asteroids" de Atari de 1979[1,2]). El programa (incompleto) proporcionado se basa en el uso de la librería gráfica multi-plataforma OpenGL.

[1] http://www.ataritimes.com/index.php?ArticleIDX=174,
[2] https://en.wikipedia.org/wiki/Asteroids_(video_game)

Concretamente, se trata de implementar la clase que gestiona los elementos que aparecen en el juego. Para ello se deberá seguir un el siguiente procedimiento:

- Dado un conjunto de ficheros de código fuente, identificar las clases presentes y sus relaciones, y crear el **diagrama de clases** correspondiente.
- Implementar la clase "**ObjectsList**", cuya **interfaz** se indica en la siguiente figura:

ObjectsList
-n: int -head: nodo* +theShip: Ship*
+ObjectsList() +~ObjectsList() +move(): void +draw(): void +add(s: Shape*): void +remove(s: Shape*): void +getShip(): Ship* +collisions(buller: Bullet*, ship: Ship*, float *): int +reposition(ship: Ship*): void

- Esta clase contendrá y gestionará una **lista enlazada** de objetos `Shape` del juego (Nave, asteroides, proyectil,…), según la lógica del juego.
- El elemento "`nodo`" de la lista enlazada podrá estar implementado como "`struct`" o como objeto.

Los atributos `n` y `head` son el número de objetos que contiene la lista, y el puntero a la cabeza de la lista. Por comodidad, el atributo `theShip` contiene un puntero a la nave (que también se guardará en el árbol).

A continuación se describen las funcionalidades que tienen que implementar los método de dicha clase.

- **ObjectsList (constructor)**
  - Crea un objeto astronave (Ship) y lo añade a la lista
  - Crear NUMASTEROIDS asteroides y los añade a la lista

- **~ObjectsList (destructor)**
  - Destruye todos los elementos de la lista

- **move**
  - Simula un paso de ejecución. Concretamente, manda el mensaje "move" a todos los objetos de la lista

- **draw**
  - Dibuja los objetos del mundo. Concretamente, manda el mensaje "draw" a todos los objetos de la lista

- **getShip**
  - Retorna un puntero a la astronave

- **add**
  - Añade un objeto a la lista

- **remove**
  - Borrar un objeto de la lista

- **reposition**
  - Tras una colisión con un asteroide, la nave se reposiciona en el centro. Hay que comprobar que ningún asteroide esté cerca del centro, a través de este método, que cambiará la posición del Asteroide (random).
  - La función `mydistance` de `commonstuff.h` calcula distancias entre dos puntos

- **collisions**
  - Este método comprueba si hay colisión entre un asteroide y el proyectil, y entre la astronave y un asteroide

  - Hay que recorrer la lista de objetos y, si son asteroides, calcular sus distancias con el proyectil y con la astronave. Se pueden dar dos casos:

  - *Asteroid vs. Ship*. Si la distancia entre un asteroide y la nave es < de la suma de los tamaños de la nave y del asteroide es que hay colisión. En este caso se borra la nave del árbol y se retorna 1. (El main restará una nave del numero de naves disponibles, si este llega a cero el juego termina.

  - *Asteroid vs. Bullet*. Si hay proyectil, y si la distancia entre un asteroide y el proyectil es inferior a la suma de los tamaños del proyectil y del asteroide es que el proyectil ha dado con un asteroide. En este caso:
    1. Se borra el proyectil del árbol.
    2. Si el tamaño del asteroide es SMALL se borra el asteroide del árbol, y se retorna 4.
    3. Si el tamaño del asteroide es MEDIUM o BIG, se manda el mensaje "split" al asteroide. Este reduce el tamaño del asteroide, y retorna otro asteroide del mismo tamaño, que hay que añadir a la lista.. El método retorna 2 o 3 según el nuevo tamaño es MEDIUM o BIG.

2. Compilación

Para compilar correctamente hay que linkar las librerias de OpenGL. Para Linux, usar la siguiente línea de comandos:

```
g++ -o Asteroids mainAsteroids.cpp … -lGL -lGLU -lglut
```

Se proporciona un makefile que puede ser usado tanto en osX como en Linux (en ambos sistemas las librerías openGL y GLUT deberían estar instaladas por defecto).

Para Windows, se proporcionan 3 ficheros: glut.h, glut32.lib y glut.dll.

En Microsoft Visual C++, poner el .h en la carpeta con los demás ficheros de código fuente, el .lib en las carpeta “release” o “debug” (que es donde el sistema pone los códigos objeto) y la dll en la misma carpeta del proyecto en donde se genera el ejecutable. Al crear el proyecto, elegir la opción “console application” y “empty project”.

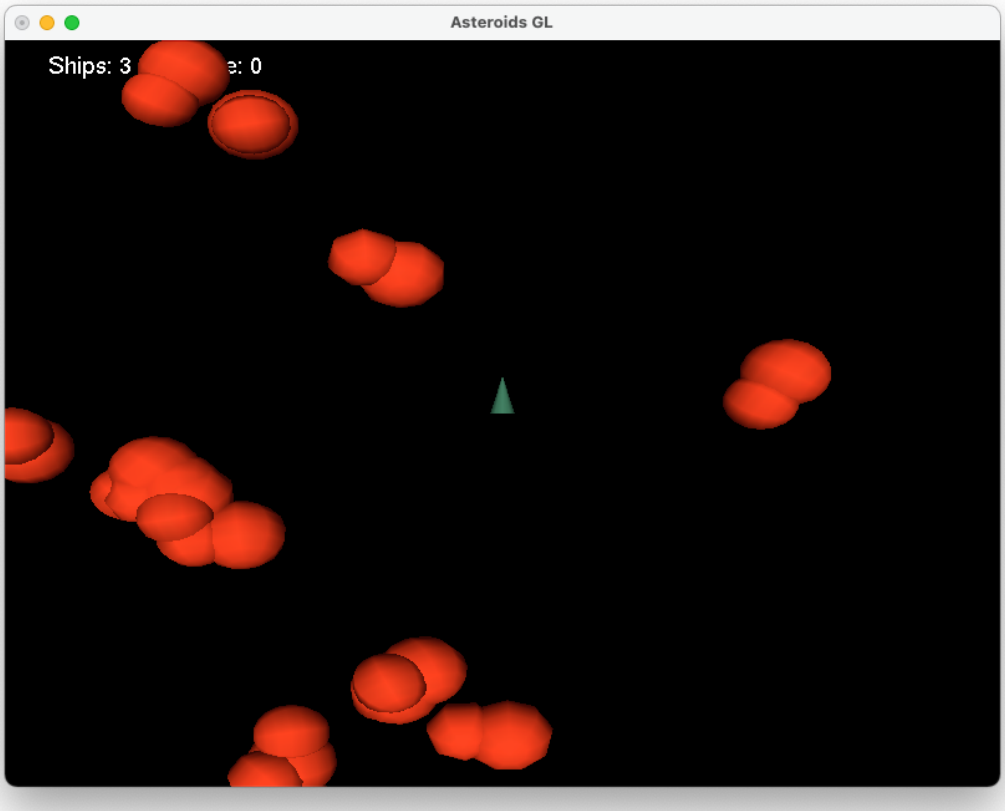
Para Codeblocks mirar, por ejemplo, esta pagina:

https://www.codewithc.com/how-to-setup-opengl-glut-in-codeblocks/

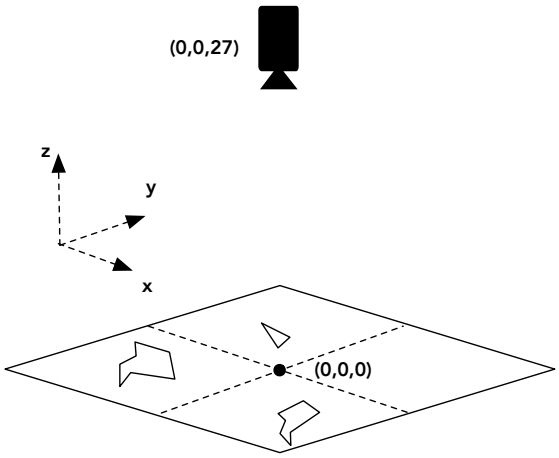
Nota: En Windows, la instrucción `#include <windows.h>` tiene que ir antes de la `#include <glut.h>`.

3. Resultado

Vuestro juego aparecerá como en la siguiente imagen:



Nota: Aunque en OpenGL el mundo es 3D, el juego es bidimensional: todos los objetos tienen coordenada z=00, y la cámara está colocada en la coordenada (0,0,27), mirando hacia 0,0,0:



4. Extensiones obligatorias

Ovni. En el juego original, de vez en cuando aparece un objeto (“el ovni”), que recorre la pantalla de derecha a izquierda o viceversa, con un movimiento vertical oscilante bastante errático. Darle a este objeto da puntos extra al jugador. El ovni

puede tener 3 tamaños (grande, mediano, pequeño) y los puntos que otorga dependen de su tamaño (50,100,200). La esxtensión consiste en:

- Definir e implementar la clase correspondiente
- Añadir a la lógica del programa la gestión del ovni
- Modificar l método “collision” para gestionar las colisiones proyectil/ovni

5. Extensiones opcionales

Disparo Ovni. En el juego original, el ovni también dispara en dirección de la nave.

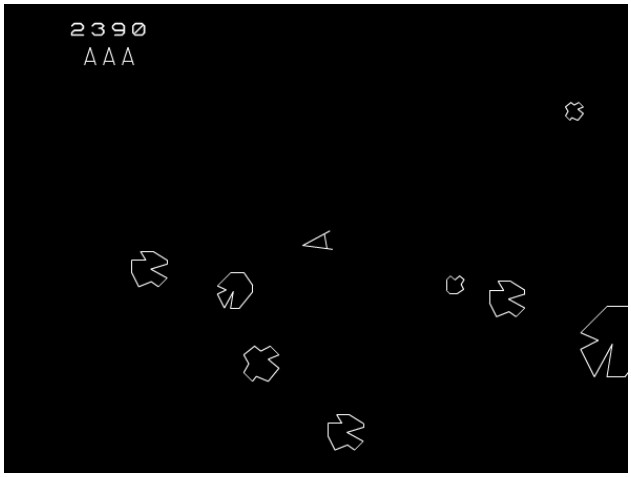
Múltiples proyectiles. El juego original permite disparar varios proyectiles, siempre y cuando no haya más de 4 ya disparados (en la implementación actual, un proyectil que no da a nada desaparece después `MAXSHOTTIME` ciclos. Hasta este momento no es posible disparar otro).

Dos jugadores. Y ¿por qué no añadir otra nave controlada por otro jugador?

Grafica mejorada. Si os gusta la gráfica en 3D, podéis aprovechar de las potencialidad de OpenGL para mejorar la gráfica del juego dibujando objetos más sofisticados con “mesh” triangulares (ver, por ejemplo, http://www.onemotion.com/flash/asteroids-game/).

Efectos especiales. Por ejemplo, en el destructor de un asteroide o de la nave, se puede implementar una secuencia de imágenes para simular una explosión. Se pueden añadir otros efectos especiales.

Aspecto “retro”. OpenGL permite dibujar polígonos (2D). Podéis animaros a recrear el aspecto original del juego (ver, por ejemplo, http://my.ign.com/atari/asteroids



6. Requisitos

Los siguientes aspectos son requisitos indispensables para que se corrija el ejercicio:

- Debe ser realizado en equipo. Puede consultarse a quien se desee (preferiblemente al profesor), pero cada alumno que forme parte del equipo debe ser capaz de demostrar que es capaz de volverlo a programar, si así se le requiere para ello.

- El programa debe estar escrito según metodología de programación secuencial, debe compilar en ANSI C++, y su ejecutable debe funcionar tanto en entornos unix como en Windows. El alumno podrá usar el compilador que desee para desarrollarlo, y el profesor el que desee para evaluarlo.

7. Desarrollo del trabajo

Deberá demostrarse que el trabajo se ha desarrollado en equipo. Cada equipo establecerá distintos roles para cada uno de sus miembros, los cuales deberán indicarse en la documentación. Algunos ejemplos de roles son, pero no tienen porqué limitarse a, los siguientes:

- Coordinación de la planificación y desarrollo del trabajo.
- Diseñar los algoritmos a emplear para resolver el problema planteado.
- Programar y depurar el código básico de los algoritmos.
- Documentación del trabajo.
- Etc.

Algunos de estos roles pueden ser compartidos, siempre que se justifique adecuadamente.

Además, cada fichero fuente que se entregue deberá tener asociado un alumno responsable de la elaboración del código que en él aparece, cuyo nombre deberá indicarse en su cabecera.

8. Entrega del trabajo

En ningún caso se entregará documentación en papel. La entrega se realizará exclusivamente mediante un único fichero zip o rar en Moodle (sección de trabajos) que deberá incluir todos los ficheros a entregar.

Los ficheros comprimidos contendrán los siguientes tipos de ficheros:

- Ficheros fuentes con el código desarrollado en C++.
- Fichero con la documentación explicativa del trabajo realizado (memoria del trabajo).

y no deberán contener ninguno de los siguientes tipos de ficheros:

- Ficheros objetos o ejecutables.
- Ficheros de salida generados en tiempo de ejecución.

Será el profesor quien se encargue de generar el fichero ejecutable.

La fecha tope de recepción por parte del profesor será el día de la celebración del examen final de la convocatoria correspondiente (Junio o Julio). En caso de que los ficheros se encuentren defectuosos, se avisará al miembro del equipo que lo envió para que pueda subsanar el problema y volverlo a entregar.

9. Documentación

El fichero con la documentación al que hace referencia el apartado anterior deberá ser lo más breve y conciso posible (se recomienda no exceder 5 páginas). El documento deberá contener:

1. Diagrama de clases del juego.
2. Diagramas de comunicación de las fases del juego más interesantes
3. Descripción general de funciones y estructuras de datos más representativas.
4. Descripción de pruebas realizadas para probar el correcto funcionamiento del algoritmo.
5. Guía de uso del programa ejecutable.
6. Descripción del reparto de roles del equipo.
7. Propuestas de mejora y valoración personal.

10. Evaluación

Se valorarán principalmente los siguientes aspectos:

- Aprovechamiento de lo aprendido en la asignatura.
- Documentación auxiliar acompañada.
- Planteamiento del algoritmo.
- Comentarios incluidos en el código del programa.
- Legibilidad del código fuente.
- Portabilidad a otras plataformas.
- Estilo de programación, uso adecuado de las sentencias y operadores, eficiencia, belleza.

En caso de duda, y dado que quien lo va a corregir es el profesor de la asignatura, la mejor forma de obtener una buena nota es preguntarle a él cualquier cuestión que surja.