Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Programación de Sistemas

Departamento de Electrónica y Automática

Práctica 2: Programación de un videojuego

Programación en ANSI C++ de una *lista enlazada* (ObjectList) para la gestión de un videojuego (estilo *Asteroids*)

Celia Ramos Ramírez (18295)

Gonzalo Quirós Torres (17353)

Josep María Barberá Civera (17048)

3° GITI

19 de junio de 2021

Resumen

Índice

Introducción

En esta memoria se presenta una posible implementación en ANSI C++ para gestionar un videojuego estilo *Asteroids* mediante el uso de la librería gráfica multiplataforma OpenGL.

El entorno de programación utilizado ha sido Visual Studio Code. Se ha gestionado el trabajo mediante el uso de GitHub y la extensión *Live Share* que ofrece VSCode.

El videojuego elegido es el famoso juego arcade *Asteroids* de Atari de 1979 [1]. Se han implementado las clases con sus atributos y métodos necesarios así como la lógica del juego para su correcto funcionamiento. Los requisitos cubiertos a grandes rasgos han sido:

- Creación de una clase *ObjectList* que gestiona una lista enlazada de los objetos del juego.

- Creación de la clase Alien, necesaria para instanciar el objeto *theUFO*.
- Integración del Ovni en la lógica del juego.
- Ajuste del sistema de puntuaciones.

Además de estos requisitos, se ha implementado alguna característica extra como que de un Ovni aparezca otro algunas veces cuando este sea destruido, o una nueva clase llamada #angel# que otorga vidas al ser capturado por la nave. Dichos extras otorgan emoción al juego.

Diagramas

2.1 Diagramas de Clases (UML)

Para entender correctamente la relación entre las clases, tanto de las ya dadas en el fichero original como de las creadas posteriormente, se ha realizado un diagrama de clases en formato *UML* para ello se ha empleado el programa StarUML®.

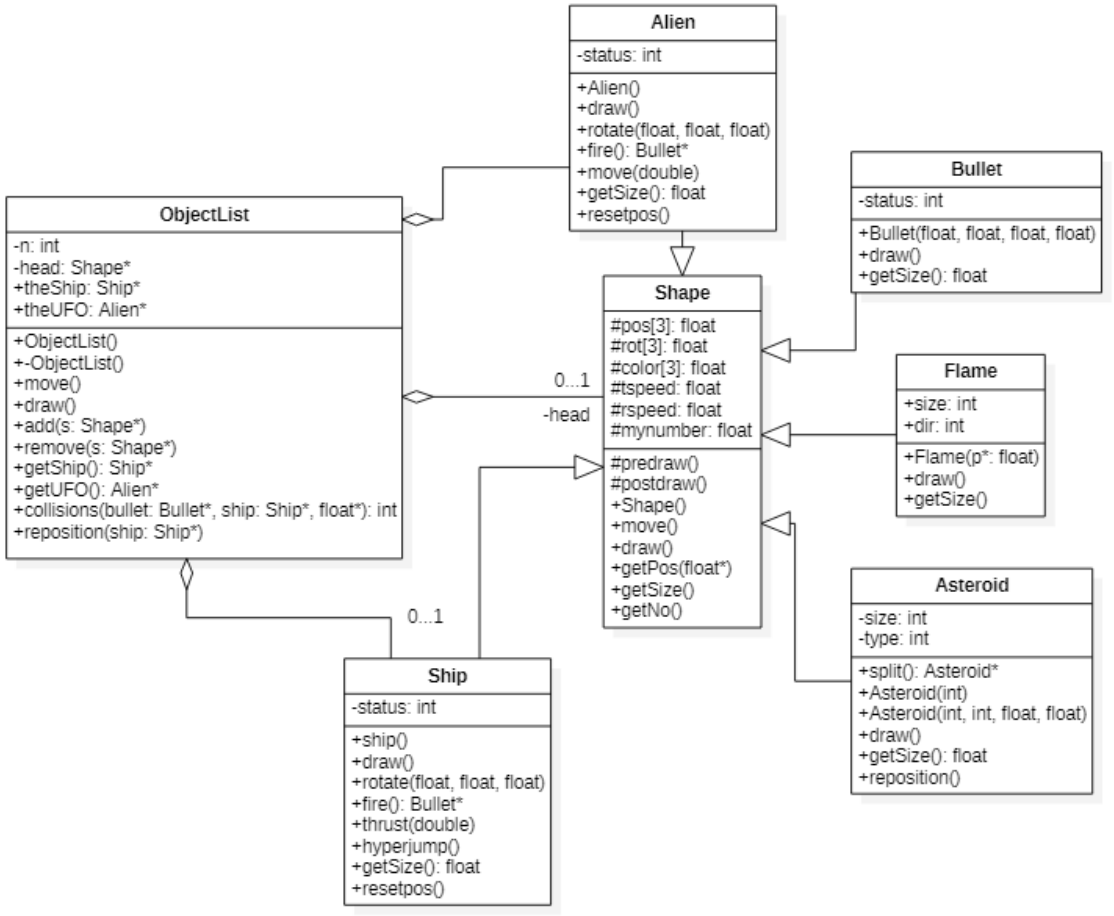

Figura 1: *Diagrama de clases del juego mediante* StarUML

Como puede verse en la Ilustración 1, la clase *ObjectList* agrega punteros a las clases *Alien*, *Ship* y *Angel*, además de contener un puntero tipo *Shape* llamado *head* que apunta a la cabeza de la lista (pero que en la implementación realizado no se utiliza). Al igual que la clase *Ship*, las clases *Alien*, *Angel*, *Bullet*, *Flame* y *Asteroid* son hijas de la clase *Shape*. La clase *Alien* tiene mucho en común con la clase *Ship*, pues realizan prácticamente lo mismo, con la diferencia de que el OVNI (instancia de la clase *Alien*) se mueve automáticamente y de forma errática sin ser necesaria la interacción con el usuario. Por comodidad se han añadido dos punteros, uno al OVNI y otro a la astronave (*theShip*), que simplificarán mucho el código tanto en la lógica del juego como en la implementación de la clase *ObjectList*. A parte de las clases aquí representadas, existe otro archivo de cabecera llamado *commonstuff*, que almacena funciones cortas, parámetros y llamadas a librerías usadas en todos los archivos fuente del programa.

2.2 Diagramas de comunicación

2.2.1. Creación de los objetos

El siguiente diagrama ilustra desde dónde se crean los objetos que se van a usar en el resto del programa:

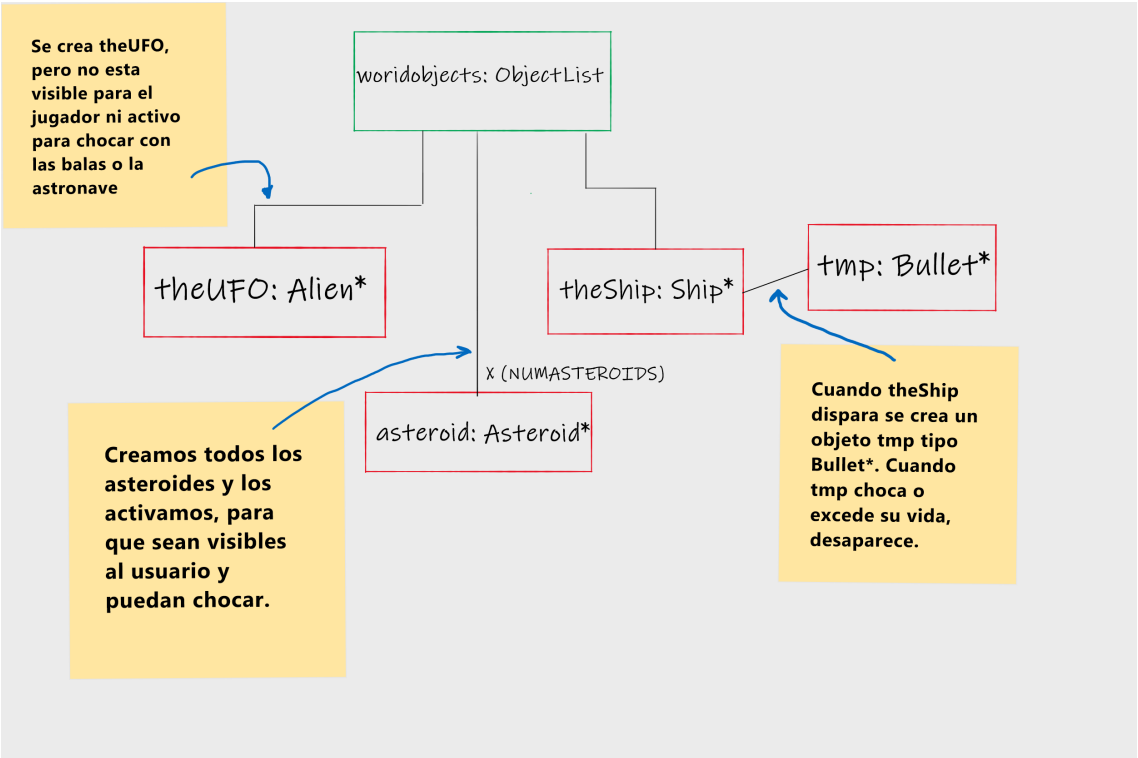

Figura 2: *Diagrama de creación de los objetos*

El diagrama de la Ilustración 2 representa el mecanismo de creación de los objetos más importantes que se lleva a cabo en el constructor del objeto *ObjectList* y la creación de la bala a través del método FIRE() del objeto *theShip*. Se crean también

todos los asteroides, donde su número concreto viene definido por el parámetro NUMASTEROIDS definido en *commonstuff.hpp*. Los asteroides son inmediatamente cargados en la interfaz gráfica mediante la función *push_front*() de la plantilla <list> a diferencia del OVNI (*theUFO*), que será cargado desde la lógica del juego cuando se cumplan las condiciones especificadas en el algoritmo.

A continuación, en otro pequeño diagrama de objetos, se verá el comportamiento de uno de los objetos *Asteroid* cuando es impactado por una bala o por la astronave.

2.2.2. Comportamiento de un asteroide tras una colisión

En el siguiente diagrama (ver Ilustración 3) se observa el comportamiento de un objeto *Asteroid* tras una colisión, ya sea con una bala o con la astronave.

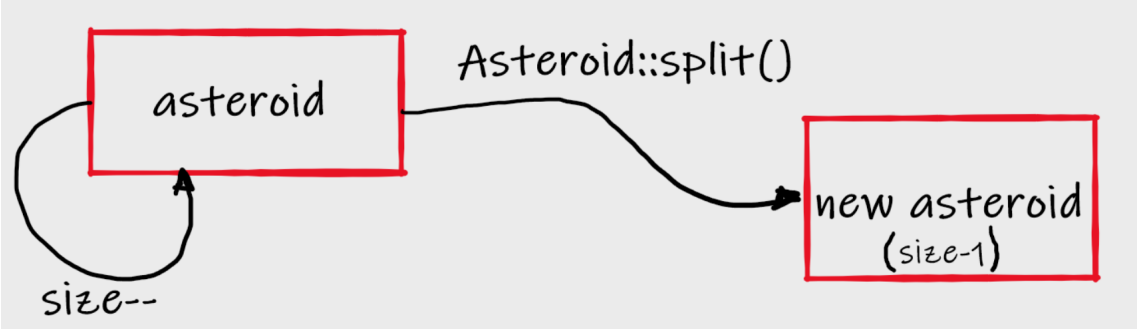

Figura 3: *Asteroide tras colisión*

Cuando un asteroide colisiona, surgen dos asteroides donde antes había uno con un tamaño disminuido en una unidad respecto al asteroide original. En realidad, lo que sucede, como se ve en el diagrama es que el asteroide original disminuye su tamaño en una unidad y en el mismo lugar surge otro con el tamaño del original también disminuido en una unidad. Evidentemente si el tamaño es 1 (esto es, SMALL), la función *Split()* no llega a ejecutarse porque el asteroide es destruido antes en la función *collision* de *ObjectList*.

Descripción de funciones y estructuras del juego

Pruebas

Guía de uso

Reparto de Roles

Propuestas de mejora y valoración personal

Referencias

[1] Wikipedia contributors. Asteroids (video game), 06 2021.