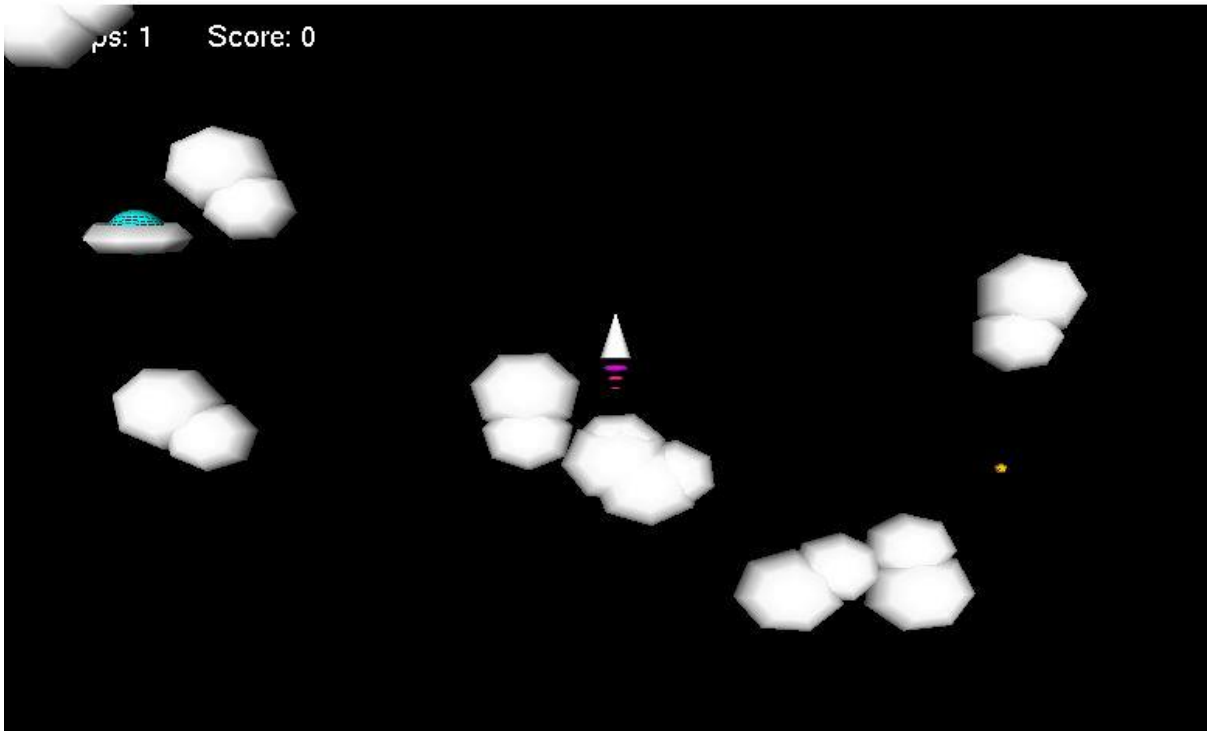
MEMORIA TRABAJO ASTEROIDS

Autores: Celia Ibáñez Martin, Jorge Martínez de la Mata y Adrian Rieker González.

Números de matrícula:18172, 18211 ,18300

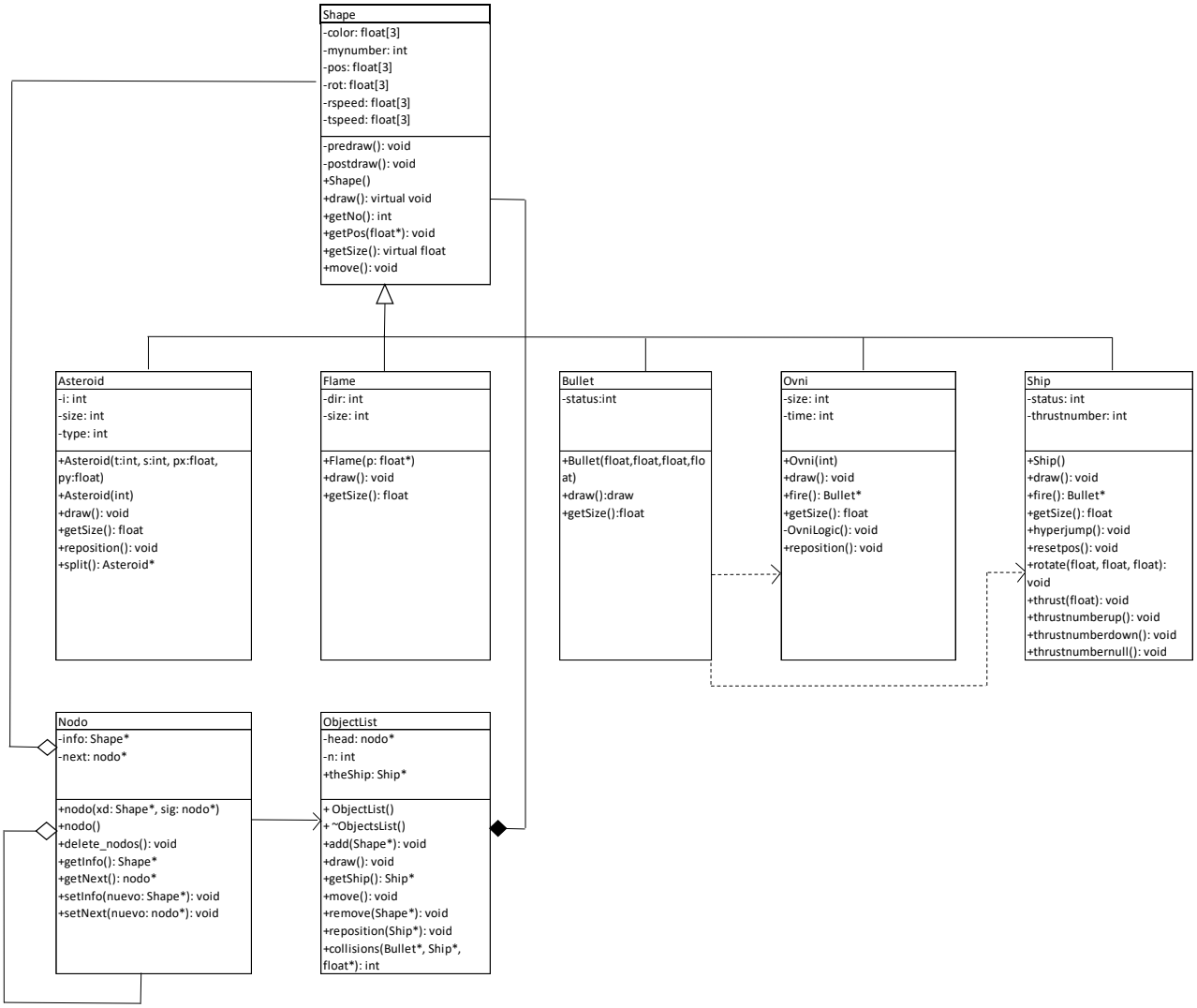
ETSII UPM

20/06/2021

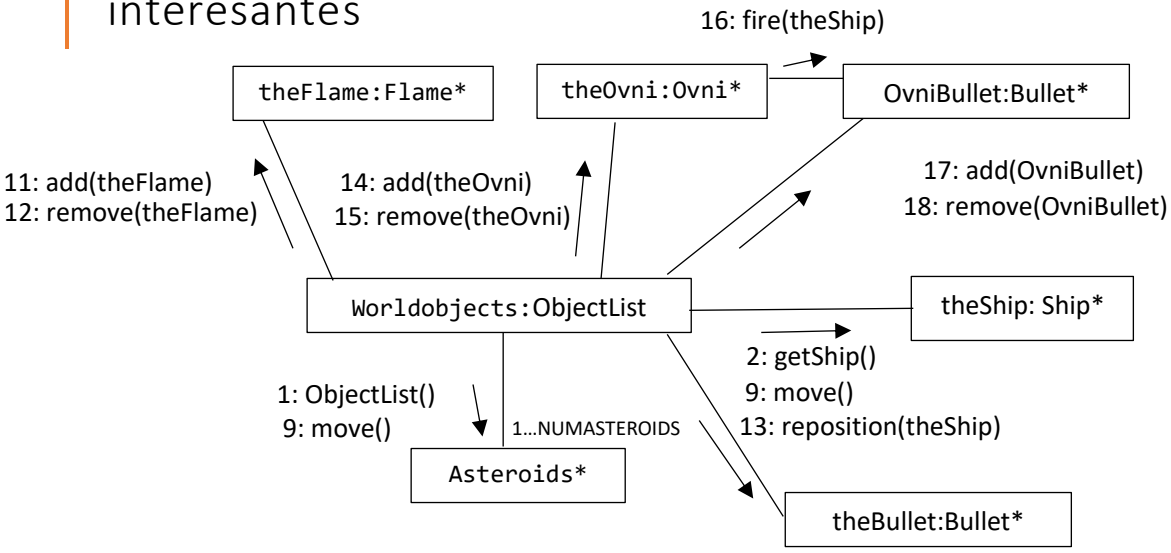


ÍNDICE

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

ObjectSLIst

Para ObjectsList hemos optado por implementar los nodos como una clase aparte. La clase nodo tiene implementados los métodos get y set necesarios para acceder a sus campos, así como un método eliminar_todo(). Este ultimo elimina todos los nodos de forma recursiva. Cada nodo guarda la referencia a un objeto Shape. Respecto a los métodos de ObjectList:

ObjectsList:ObjectsList(): Es el constructor, se instancian los asteroides iniciales y la nave de forma dinámica y se añaden a la lista. Además, se llama a reposition para que no coincida ningún asteroide en la posición de la nave

ObjectsList:: ~ObjectsList(): Es el destructor de ObjectsList. Se manda el mensaje eliminar_todo() al nodo cabeza de la lista y el resto se eliminan recursivamente.

void ObjectsList::move(): Se manda el mensaje move a todos los objetos Shape de la lista. Hay una variable Nodo* aux que va recorriendo la lista y mandando el mensaje.

void ObjectsList::draw(): Funciona igual que el método anterior pero mandando el mensaje draw().

void ObjectsList::remove(Shape*): Elimina una Shape* de la lista. Se han diferenciado los casos en los que la Shape buscada es la primera de la lista y cuando no. Cuando se elimina un objeto se decrementa el contador 'n' que cuenta el número de objetos de la lista

void ObjectsList::add(Shape* nuevo): añade un nuevo objeto.

int ObjectsList::collisions(Bullet*,Bullet*,Ship*,float*): Los dos primeros parámetros son para el proyectil de la nave y para el proyectil del ovni. Hay una variable auxiliar que va recorriendo la lista y detectando uno de los 3 siguientes casos: colisión nave-asteroide, colisión proyectil (de la nave) – asteroide y colisión proyectil(de la nave) contra el ovni. Para saber si un nodo de la lista contiene un asteroide o al ovni hemos usado dynamic_cast. Esta función retornara NULL si el cast no ha sido posible y en caso contrario guardara la referencia del ovni/asteroide en cuestión en una variable auxiliar del tipo de este. Así podemos acceder a los métodos de la subclase de Shape que no están en la interfaz de Shape. Después de estas, si existe el proyectil del ovni se comprobará también la colisión entre el proyectil disparado por el ovni y la nave.

void ObjectsList::reposition(Ship*): Comprueba si hay asteroids cerca de donde reaparece la nave y en caso de que si, los mueve a posiciones aleatorias.

El Ovni

Para la creación del ovni nos hemos inspirado en las interfaces y el código del resto de subclases de Shape. Para el proyectil del ovni hemos reutilizado la clase Bullet.
Los métodos más importantes de Ovni son:

void Ovni::OvniLogic(): Cada vez que se llama a Ovni::draw() se llama a este método. Es un método privado en el que se incrementa el atributo time de ovni. Sirve para que el ovni se mueva de izquierda a derecha y con un movimiento vertical senoidal.

Bullet* Ovni::fire(Ship*): Cuando desde MyLogic se manda este mensaje, se instancia un objeto Bullet en el heap y se devuelve su referencia. El parámetro Ship* sirve para pasar la dirección de la nave para que el proyectil del ovni se dirija hacia su posición. Esto último todavía no funciona del todo bien.

Otros cambios

Asteroid.cpp/.h: Se ha editado el fichero .cpp para que los asteroides cambien de color al mandar el mensaje Split.

Ship.cpp/.h: Se ha añadido a las funciones de rotacion, thrust y hyperjump nuevo código para que la nave pueda frenar en seco, y que la dirección de la velocidad varíe cuando gira, para una mayor facilidad de manejo de la nave.

mainAsteroids.c :

- Se ha quitado la librería unistd.h y se ha cambiado sprintf por sprintf_s (para evitar problemas en visual studio).
- Se han añadido los objetos globales theOvni y OvniBullet, así como las variables globales ovnitime y shottime_ovni.
- En MyLogic() se ha añadido todo lo correspondiente a la aparición/desaparición del ovni, el disparo del ovni y las respectivas colisiones.
- Se han modificado los controles (Esto está más detallado en el apartado de instrucciones de uso)

Al trabajar en Visual Studio 2019 se ha utilizado la función sprintf_s ya que sprintf se considera una función insegura en este entorno y habría que añadir ciertas macros al preprocesador. Para evitar esto y asegurar la compilación del programa en CodeBlocks se ha cambiado por sprintf.

Las explicaciones más concretas de cómo funciona el programa se encuentran en los ficheros de código en forma de comentarios.

4. DESCRIPCIÓN DE PRUEBAS REALIZADAS PARA PROBAR EL CORRECTO FUNCIONAMIENTO DEL ALGORITMO

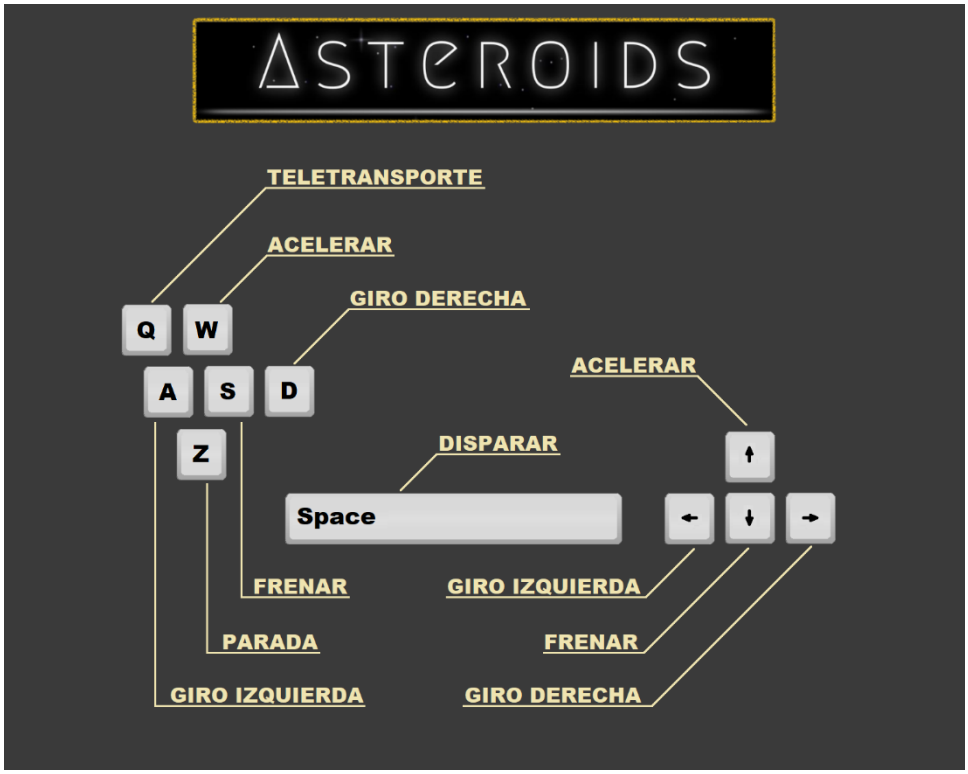
- Prueba Reposition: https://youtu.be/9XjPMbOts8k
  Comprobación del correcto funcionamiento del método reposition que coloca el asteroide en otra posición si coincide con la nave cuando esta debe aparecer en su posición de inicio.

- Llama en movimiento: https://youtu.be/oVpmibDj7bM
  Ejemplo de una mejora estética que llevamos a cabo creando una llama para la nave con movimiento que la sigue en todo momento.

- Bug del Ovni: https://www.youtube.com/watch?v=IhTFCCMw1r0
  Bug solucionado que ocurrió durante el desarrollo del programa en el que los ovnis no se destruían si lograban dar a la nave.

- Juego final: https://youtu.be/8GHqa99his8
  Vídeo de ejemplo de cómo funciona el juego final.

5. GUÍA DE USO DEL PROGRAMA EJECUTABLE



6. DESCRIPCIÓN DEL REPARTO DE ROLES DEL EQUIPO

Coordinación: Jorge Martínez de la Mata
Diseño de los algoritmos: Adrián Rieker y Celia Ibáñez
Programación: Adrián Rieker, Celia Ibáñez y Jorge Martínez de la Mata
Depuración: Jorge Martínez de la Mata, Adrián Rieker y Celia Ibáñez
Diagramas: Celia Ibáñez
Edición de imagen: Adrián Rieker y Jorge Martínez de la Mata

7.- PROPUESTAS DE MEJORA Y VALORACIÓN PERSONAL.

Como propuestas de mejora tenemos que la nave pueda disparar muchas veces de forma automática y animar la llama del propulsor de la nave. Ambas cosas las intentamos implementar, pero no nos dio tiempo. La primera usando un vector dinámico/lista enlazada en el main para guardar los proyectiles en vez de "theBullet" y la segunda usando una nueva subclase de Shape. Además, añadiríamos una pantalla de inicio (Por ejemplo, la imagen anterior). Otras mejoras que se nos ocurren son:

- Añadir tecla de pausa
- Añadir pausa técnica cuando se pierde vida
- Mostrar la puntuación final
- Poder volver a jugar sin abrir de nuevo el programa (y establecer marcas personales)
- Modo 2 jugadores
- Añadir power-ups
- Distintas naves
- Mapa de fondo dinámico con rayas con la misma dirección que la nave para dar sensación de gran velocidad