

95.11 – Trabajo Práctico N°2

Asteroids 40º aniversario

1. Objetivo del TP

El objetivo del presente trabajo consiste en la implementación de una reversión del juego Asteroids en lenguaje ISO-C99 utilizando la biblioteca gráfica SDL2. El trabajo consiste en la resolución del problema mediante el diseño y uso de TDAs y en la reutilización de las rutinas desarrolladas a lo largo del curso en trabajos anteriores.

2. Alcance del TP

Mediante el presente TP se busca que el estudiante adquiera y aplique conocimientos sobre los siguientes temas:

- Listas enlazadas,
- Encapsulamiento de TDAs,
- Módulos singleton,
- Punteros a funciones,
- Archivos binarios,
- Técnicas de abstracción,

además de los temas ya evaluados en trabajos anteriores.

3. Introducción

3.1. Atari Asteroids

El arcade Asteroids fue desarrollado por Atari en el año 1979. El juego consiste en eliminar todos los asteroides que se mueven por la pantalla y evitar las colisiones con los mismos.

El Asteroids está montado sobre la misma plataforma del Lunar Lander. Al iniciar su producción la demanda fue tal que debió discontinuarse el Lunar Lander para producir Asteroids. El juego se convirtió en el 6º juego de arcade más vendido de la historia, con 100.000 unidades.

Entre alguna de las novedades introducidas por el Asteroids está la de que el jugador ingrese sus iniciales al final del juego para conformar el ranking de los mejores jugadores.

En el juego (figura 1) el usuario controla una nave que puede girar, acelerarse y disparar. Se empieza con 3 o 4 vidas, y cada vez que la nave colisiona con algún objeto se destruye y se continúa con el juego.

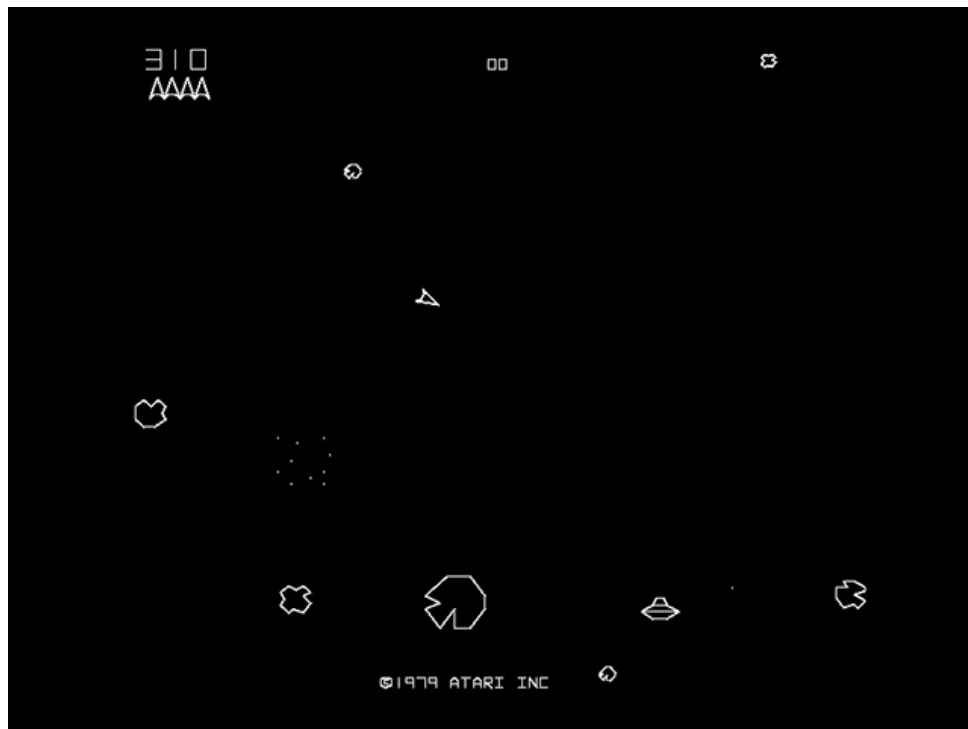


Figura 1: Atari Asteroids.

En el espacio hay asteroides de diferentes tamaños que se desplazan por la pantalla. Cuando un asteroide recibe un disparo o choca con la nave se parte en dos asteroides más pequeños y de mayor velocidad. Cuando se rompe un asteroide muy pequeño el mismo desaparece.

El nivel se completa cuando se logra destruir todos los asteroides del juego y el mismo vuelve a recomenzar con nuevos asteroides.

La pantalla del juego es esférica, es decir, lo que sale por debajo reaparece por arriba y lo que sale por la derecha reaparece por la izquierda y viceversa.

En el juego original había naves que atacaban con disparos a la nave del jugador y la nave podía teletransportarse a una ubicación aleatoria de la pantalla para escapar de una colisión segura.

4. Preliminares

4.1. Definición de sprites

Todos los sprites (representación gráfica de los objetos) del juego se distribuyen en un archivo binario y deben ser levantados dinámicamente de dicho archivo. Es decir que si cambiara la biblioteca de sprites deberían actualizarse todos los gráficos del juego.

Cada sprite se representa como una secuencia de:

0	
9	<code>char nombre[10]</code>
10	
11	<code>uint16_t n</code>
12	
12+2×4n	<code>float coords[n][2]</code>

Donde `nombre` es el nombre del sprite, `n` es la cantidad de coordenadas y `coords` son los pares de coordenadas que dibujan el objeto.

El archivo de sprites contiene una secuencia desconocida de sprites con nombres desconocidos (desconocidos para el graficador).

4.2. Diseño del programa principal

A diferencia de en el TP1 se da cierta flexibilidad para construir el programa y, por ejemplo, no se entrega un archivo de configuración general, etc. Sin embargo se impone utilizar el `main.c` del TP1 con los mismos bloques delimitados que tenía el mismo. Los mismos son más que suficientes como para resolver todo lo pedido en este trabajo y respetar esa lógica va a ordenar el diseño general dado que el mismo delimita con efectividad las distintas etapas. El archivo `config.h` puede utilizarse de base para definir los parámetros que hagan falta.

5. Diseño de módulos

5.1. Singleton graficador

El patrón singleton¹ consiste en un módulo del cual hay una única instancia en toda la ejecución del proceso. Para manejar los sprites se requiere implementar este patrón: Los sprites se cargan desde el módulo principal al comienzo una única vez, todas las funciones pueden pedirle cosas al módulo (sin necesitar una referencia a su instancia) y al finalizar el módulo principal libera los recursos.

Se da completa la interfaz del módulo, como parte de este trabajo se debe desarrollar una implementación capaz de poder cumplir con lo pedido:

- `bool graficador_inicializar(const char *fn, int ancho, int alto);`

Inicializa el módulo. Recibe `fn` que es el nombre del archivo de sprites y el `ancho` y `alto` de la pantalla. Esta función debe ser llamada antes de poder acceder a las demás funciones del módulo.

- `void graficador_finalizar();`

Finaliza el módulo. Deben liberarse todos los recursos asociados. Después de llamar a esta función el módulo debe quedar como antes de llamar a `graficador_inicializar()`.

- `bool graficador_dibujar(SDL_Renderer *r, const char *nombre, float escala, float x, float y, float angulo);`

Dibuja el sprite de nombre `nombre` en el renderer `r` escalado según `escala`, rotado según `angulo` en la posición `x`, `y` de la pantalla (tomando como (0, 0) la esquina inferior izquierda).

- `void graficador_ajustar_variables(float *x, float *y);`


Esta función es de asistencia a otros módulos, recibe dos variables `x` e `y` y las ajusta dentro del `ancho` y `alto` de la pantalla. De esta forma los módulos que trabajan con coordenadas no tienen necesidad de conocer las dimensiones del espacio.

¹<https://es.wikipedia.org/wiki/Singleton>

5.2. TDA Lista enlazada

Se debe implementar un TDA Lista enlazada genérica con las primitivas mínimas que permitan resolver el TP.

Para interfaz de la lista enlazada se utilizará la interfaz que se encuentra en la guía de trabajos prácticos de la materia. No se permite la adición de primitivas nuevas por fuera de las de la guía.

Para la realización del trabajo será necesario implementar o bien las primitivas que permitan recorrer y filtrar los elementos o implementar el TDA ador. Si bien se considera que es más cómodo implementar el iterador, el trabajo puede resolverse completo (pero complicado) con recorrer y filtrar.

5.3. TDAs objetos del juego

Se deben implementar TDAs para los 3 objetos más importantes del juego: Nave, asteroides, disparos.

Siendo que los 3 objetos tanto evolucionan en el tiempo como tienen que ser graficados en la pantalla deberán contener las siguientes primitivas:

- `void xxx_mover(xxx_t *x, float dt);`²

Le avisa al objeto `x` que pasó un instante `dt` de tiempo y que debe actualizar su posición y su estado.

- `bool xxx_dibujar(const xxx_t *x, SDL_Renderer *r);` Le pide al objeto `x` que se dibuje sobre el renderer `r`.

Los TDAs deberán modelar el siguiente comportamiento:

Nave: Representa a la nave que el usuario controla y viaja por el espacio.

Tiene una posición, velocidad, ángulo y potencia. Debe tener una interfaz para incrementar/decrementar su ángulo. Debe tener una interfaz para aplicarle un impulso. Al aplicarle un impulso se incrementa la potencia en 1000 unidades. Por cada centésima de segundo la potencia debe reducirse un 10 % y la velocidad un 1 %, es decir, hay una componente friccional en el movimiento y tanto potencia como velocidad decaen exponencialmente³.

La nave se grafica mediante el sprite “SHIP” mientras que el chorro según el sprite “THURST”. El chorro debe dibujarse mientras la potencia sea mayor a la unidad.

Asteroide: Representa a un asteroide que viaja por el espacio.

Tiene una posición, velocidad, ángulo y radio. Cuando se crea un asteroide su velocidad y su ángulo son aleatorios. La velocidad estará en el rango $\frac{1000}{\text{radio}} \pm 100$. Hay 4 clases de asteroides, que se dibujan con diferentes sprites, la clase del asteroide debe ser generada aleatoriamente al crearlo. El asteroide debe tener una interfaz para testear si un par de coordenadas dado colisiona con él o no, esto servirá para saber si la nave o el disparo han chocado con un asteroide.

El asteroide, según su clase, se grafica mediante los sprites “ROCK1”, “ROCK2”, “ROCK3” o “ROCK4”. (Los sprites provistos tienen radio 1, por lo que deben ser escalados por el radio del asteroide.)

²`xxx` representa al TDA en cuestión, sea nave, asteroide o disparo.

³Por simplicidad, si bien no es matemáticamente correcto puede asumirse que Δt estará cerca de 0,01 s y ajustar la cuenta para que dé esos factores con ese valor y aproximados con valores cercanos.

Disparo: Representa a una bala que viaja por el espacio.

Tiene una posición, ángulo y tiempo de vida. Las balas se desplazan a una velocidad de 1000.

La bala se grafica mediante el sprite “SHOT”.

Pueden agregarse a los TDAs las primitivas necesarias para consultar los parámetros (posición, ángulo, etc.) de los mismos. (Particularmente para el disparo en caso de implementar los recorridos de lista con recorrer y filtrar puede ser conveniente definir una primitiva para forzar el tiempo de vida a un valor alto.)

Si bien se mencionó previamente, se reitera que estos módulos no tienen la necesidad de conocer las características de la pantalla del juego. Tanto la graficación de los sprites como la puesta en rango de las posiciones deben ser delegados al graficador.

5.4. Listas de TDAs

Los TDA asteroides y los TDA disparo deben ser contenidos en TDA listas. Está prohibido indexar estas estructuras en un contenedor que no sea una lista. Todas las funcionalidades del juego deben resolverse a través del manejo de estas listas.

5.5. Otros TDAs y módulos

Se permite la confección de otros TDAs que sean de utilidad, pero en principio no hacen falta más que los descriptos.

Además está permitido el reaprovechamiento (y modificación y extensión) de módulos ya desarrollados para el TP1, como por ejemplo los módulos de física, manejo de vectores, impresión de caracteres, etc.

5.6. Tests

Se recomienda testear los TDAs individualmente con pruebas unitarias de sus funcionalidades.

6. Funcionamiento del juego

El programa funcionará continuamente mientras no se cierre la ventana. Durante la vida del mismo se desarrollarán partidas del juego.

Cada partida del juego empezará con el marcador de puntos en cero y con 4 vidas disponibles. Habrá 4 asteroides de radio 32 distribuidos aleatoriamente sobre los bordes de la pantalla (esto es sobre los ejes $(0, y)$ y $(x, 0)$). La nave se inicializará en el centro de la pantalla con ángulo 0. A partir de ahí el jugador podrá mover la nave para esquivar asteroides y podrá dispararles.

La nave se comandará con las teclas izquierda y derecha para girar, arriba para darle un impulso y la barra espaciadora para disparar⁴.

Cuando se dispare los disparos se originarán en las coordenadas de la nave y con el ángulo que tenga la misma. Los disparos deberán “morir” cuando lleguen a una edad de 0,7 s.

Cuando la nave o un disparo impacten un asteroide el mismo se destruirá. Si el asteroide tuviera radio 32 se crearán dos asteroides de tamaño 16 en la misma posición, si tuviera radio

⁴La barra espaciadora se representa en SDL2 por `SDLK_SPACE`.

16 se crearán dos de tamaño 8, si fuera de tamaño 8 no se hará nada. La destrucción de un asteroide grande entrega 20 puntos, la de uno mediano 50 y la de uno pequeño 100 puntos.

Si no quedaran más asteroides en la pantalla se generarán nuevos asteroides con las características de la generación inicial. Cada vez que no queden asteroides se crearán 2 asteroides más que la vez anterior.

Si la nave recibiera el impacto de un asteroide se destruirá y el usuario perderá una vida. Si quedaran vidas restantes se creará una nave nueva en la posición y coordenadas del inicio del juego⁵. Para crear la nueva nave se esperará en principio 1 s, pero si al cumplirse el segundo hubiera un asteroide que colisione con el sitio de aparición de la nave se esperará una décima de segundo más y se continuarán sumando décimas hasta que la ubicación de la nave sea segura.

Una partida finaliza cuando al usuario no le quedan vidas restantes. Al finalizar la partida se imprimirá la leyenda “GAME OVER” junto con el puntaje de esa partida y el juego estará listo para iniciar una nueva partida.

Las partidas no empezarán hasta tanto el usuario no presione la barra espaciadora.

Durante el tiempo que dure la partida se deberán mostrar en la pantalla la cantidad de vidas disponibles, el puntaje del jugador y el mejor puntaje de todos los juegos (de esa ejecución del programa) en ubicaciones y con tamaños similares a los del juego original⁶.

(Toda la dinámica descrita puede resolverse sencillamente en el mismo mainloop. Se recomienda diseñar cuidadosamente el invariante del ciclo principal para saber qué significa exactamente que no haya nave, o que no haya asteroides, etc. según las diferentes fases del juego.)

7. Consideraciones generales

- Se recomienda releer el enunciado las veces que sean necesarias para extraer toda la información contenida en el mismo.
- Es imprescindible entender la dinámica de los TDAs antes de empezar la etapa de diseño.
- Es imprescindible diseñar el trabajo antes de empezar la etapa de programación del mismo.
- El programa debe estar modularizado, las acciones tienen que estar delegadas para ser ejecutadas por el módulo en el que más natural sea hacerlo.
- Se recomienda sistematizar el trabajo para poder implementar pequeñas partes que puedan ser probadas. El trabajo tiene muchísimas partes independientes entre sí que pueden abordarse de a una por vez.

Se deben entregar:

- Los códigos fuentes del proyecto completo.
- El archivo Makefile para compilarlo.
- Un pequeño informe que explique sintéticamente cómo se diseñó la solución, de qué forma se diseñó, consideraciones que se tomaron, etc.

⁵Curiosidad: En el juego original la nave se crea en el ángulo que haya estado la última nave, incluso al empezar un juego nuevo.

⁶Para dibujar las vidas puede utilizarse el sprite “SHIP”.

8. Entrega

Fecha de entrega: Domingo 30 de junio.

La entrega se realiza por correo a la dirección `algoritmos9511entregas` en `gmail.com` (reemplazar `en` por `@`).

Se permiten grupos de máximo dos integrantes.