Alberto Robles Enciso

Tecnologías de la Programación (TP)

Grupo 2.2 - Gracia Sánchez Carpena

Junio – 2016

Índice:

Contenido

Descripción de la aplicación

La aplicación es el típico juego de asteroides, además de tener un minijuego del buscaminas.

Este se ha desarrollado siguiendo el modelo de TDA y haciendo uso de la librería SDL y SDL_mixer

Manual de usuario

La aplicación comienza pidiendo por el terminal el nombre el jugador (este se usara para guardar la puntuación), a continuación se accede al menú, para cambiar entre las distintas opciones que tiene se usa las flechas arriba y abajo, y para seleccionar una opción se hace con la tecla “Espacio”.

La primera opción (“Jugar”) es para iniciar el juego principal, este consiste en el típico juego de asteroides, en este caso el jugador maneja una nave que puede mover con las flechas o con WASD, esta nave tiene un número finito de vidas que pierde cada vez que choca con un asteroide y si las pierde todas pierde el juego. Hay cuatro tipos de asteroides que ordenados por probabilidad de aparición tenemos: el marrón, este el tipo que sale con una dirección fija y debes esquivar o romper; el morado, este el igual que el marrón con la diferencia de que este persigue al jugador; el azul, este es un pequeño asteroide con dirección fija que si te chocas con él te da una vida; y por último el amarillo, este es como el azul pero el recarga las bombas.

En el juego el jugador tiene distintas habilidades:

- Explosión: Se usa con la Q, cada uso gasta una bomba, genera una pequeña explosión de disparos.
- Megaexplosión: Se usa con la E, cada uso gasta una megabomba, genera una gran explosión.
- Escudo: Se usa con el click derecho, este genera un escudo que protege de los asteroides mientras consume la energía del escudo, la cual se recarga automáticamente (el escudo requiere un mínimo para poder funcionar).
- Disparo normal: Se usa con el click izquierdo, tiene cadencia 0 y munición infinita.

Según aumenta la puntuación del jugador la dificultad del juego aumenta (aparecen más asteroides y son más rápidos) y finalmente cuando el jugador alcanza los 200 puntos gana

El juego tiene unas teclas para trucar la partida, con la Z te sumas 5 puntos, con la X ganas una bomba, con la C aumentas una vida y con la V te añades una megabomba.

La segunda opción del menú (“Lateral”) consiste en básicamente el mismo juego pero limitado solo al movimiento hacia arriba y abajo, y los asteroides solo aparecen por el lado derecho de la pantalla.

La tercera opción del menú (“Sandbox”) es el juego principal pero sin enemigos y con explosiones infinitas.

La cuarta opción (“Buscaminas”) es un minijuego del buscaminas, este se juega igual que uno normal, se selecciona una casilla con click izquierdo y se ponen las banderas con click derecho, se puede reiniciar la partida con R, con las flechas arriba y abajo aumentas/disminuyes el número de minas de la partida (requiere reiniciar) de uno en uno y con las flechas de izquierda y derecha lo haces de 5 en 5. Además con la tecla W ganas la partida al instante y con la S la pierdes.

Finalmente con la tecla “Escape” cierras el juego o seleccionando la última opción del menú (“Salir”).

Organización del proyecto

Nivel de aplicación:

La aplicación aparte de tener los .dll correspondientes a las librerías requiere una carpeta “Datos” que contiene:

- Un fichero de texto (“Puntuaciones.txt”) que será el que guardara las puntuaciones.
- Una carpeta “Imagenes” con las imágenes que el juego usa (“bandera.bmp”, “mina.bmp”, “mina_b.bmp”, “mina_`p.bmp”).
- Una carpeta “Musica” con las músicas que el juego usa (“buscaminas.wav”, “main.wav”, “menu.wav”).
- Una carpeta “Sonidos” con los sonidos que el juego usa (“beep5.wav”, “beep38.wav”, “impact_wrench_removing_nut.wav”, “knife_impact_stab_head.wav”, “metal_impact.wav”, “snowball_hit_wall.wav”).

Nivel de código:

- Main.c: Este el fichero principal el cual depende de:

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <SDL2/SDL.h>
#include "SDL2/SDL_mixer.h"
#include <sys\timeb.h>

// TDAs
#include "Pantalla.h"
#include "Vector.h"
#include "Listas.h"
#include "Puntuaciones.h"
#include "Particulas.h"
#include "Sierpinski.h"
#include "Buscaminas.h"
```

  Este fichero agrupa todos elementos para el funcionamiento general de la aplicación, es el que almacena la información del jugador, tiene el bucle principal de dibujado y se encarga de inicializar todo.

  En la parte mas superior del codigo están todas las variables globales, acontinuacion le siguen las funciones del juego:

  - “inicializar_juego”, inicializa las variables del juego.
  - “cerrar_juego”, libera memoria.
  - “draw_prota”, dibuja el triangulo que es el jugador.
  - “disparos”, es la encargada de recorrer la lista de los disparo, dibujarlos y detectar las colisiones.
  - “draw_enemigos”, se encarga de recorrer la lista de los enemigos, dibujarlos, actualizarlos y detectar las colisiones.
  - “crear_disparo”, función para crear un disparo del jugador.
  - “crear_enemigo_pos”, función para crear un enemigo.
  - “draw_info”, dibuja en pantalla la información.
  - “crear_enemigo”, función para crear un enemigo (condicional).
  - “draw_mouse”, dibuja el cursor.
  - “habilidades”, función que administra las habilidades.
  - “escudo_run”, función que controla el escudo

    - “main”, función principal, al inicio tiene una pequeña inicialización, después comienza un bucle infinito que será el del juego, este usando la variable “juego” separa el código en distintos dibujados(menú, juego principal y buscaminas), en el juego el bloque contiene las funciones de control, actualización y dibujado. Finalmente como bloque común de todos los “juego” al final del bucle principal esta el cronometro que calcula el tiempo de ejecución de ese ciclo para indicar los FPS del juego y también poder regular la función “Pantalla_Espera()” para mantenerse constante en 15ms. Y después de este bucle están las funciones para liberar memoria.

- Buscaminas.c/.h: TDA con todo lo necesario para hacer el minijuego del buscaminas, este requiere el TDA “Pantalla”, solo posee funciones públicas para inicializar, dibujar y de avance(“loop”), internamente funciona principalmente con una matriz de 22x22.

- Listas.c/.h: TDA lista, el cual agrupa todo lo necesario para hacer funcionar una estructura enlazada lineal simple, a la cual desde fuera se accede de forma abstracta sin saber cómo funciona, ni su estructura interna

- Pantalla.c/h: TDA de dibujado de SDL, depende de la librería SDL

- Particulas.c/.h: TDA para generar partículas, este depende de “Vector” y de “Pantalla”, en esencia se trata de un TDA que públicamente solo se le puede ordenar que cree una partícula y la dibuje, el internamente trabaja con una estructura enlazada lineal y con la función de dibujado se actualiza comiendo todas las partículas según su velocidad y cuando vea conveniente eliminara las partículas que se salgan de la pantalla

- Puntuaciones.c/.h: TDA con la función de controlas el sistemas de puntos en fichero, externamente solo se le puede ordenar que lea las puntuaciones, diga cuantas hay, devuelva el nombre o los puntos de una determinada posición y finalmente que escriba una puntuación en el fichero. Internamente trabaja con un array el cual crea primero leyendo las líneas que tiene el fichero y con eso sabe cuántos jugadores tiene el fichero de puntuaciones, una vez creado almacena en el los nombres de los jugadores y sus puntuaciones y después se ordena de mayor a menor según las puntuaciones y ese será el array final.

- Sierpinski.c/.h: TDA encargado de generar este famoso fractal triangular, este depende del TDA “Pantalla”, Públicamente solo se le puede pedir que cree un punto (usado para definir los primeros vértices), también que cree un triángulo y por último que lo haga “crecer”. Internamente trabaja con un árbol ternario ya que un principio se crea una raíz que tendrá tres hijos (los tres triángulos interiores) y así sucesivamente hasta que se llega a un límite de profundidad declarado en el TDA

- Vector.c/.h: TDA que agrupa todo lo relacionado con vectores, este sirve para crear vectores bidimensionales y tiene funciones para poder rotar, escalar, hacer unitarios los vectores o para hacer apartir de dos puntos un vector.

Estructuras de datos

Los TDAs ya los he indicado antes, respecto a las estructuras de he usado dos:

Estructura enlazada lineal simple: La he usado para en el TDA Listas y es la encargada de almacenar los enemigos y las balas, al igual, en el TDA Particulas tengo otra similar. La estructura consiste en un apuntador a esta estructura:

```
struct Entidad {
    double x,y,vel_x,vel_y,radio,dat;
    int estado;
};

struct Nodo {
    Lista_Entidad Elemento;
    struct Nodo* Sig;
};
```

Básicamente son nodos encadenados haciendo uso de un apuntador al siguiente (Sig), como prerrequisito se necesita un nodo cabecera que será por el que se comienza a listar. Además del apuntador cada nodo posee un elemento en su interior que es el que almacena los datos, en este caso es una estructura de tipo entidad.

En el TDA partícula tengo la misma lista.

En el TDA Sierpinski tengo un árbol ternario que almacenara los sucesivos triángulos del fractal(es ternario porque por en cada iteración cada triangulo del fractal se divide en tres).

```
struct Sierpinski_TrianguloPtr {
    int count,level;
    Sierpinski_Punto a,b,c;
    struct Sierpinski_TrianguloPtr* padre;
    struct Sierpinski_TrianguloPtr* ap;
    struct Sierpinski_TrianguloPtr* bp;
    struct Sierpinski_TrianguloPtr* cp;
};
```

En esa estructura arbolescente cada nodo puede tener tres hijos(denotados como ap, bp y cp) y tiene un apuntador al padre de ese nodo para poder recorrer la estructura a la inversa, a su vez tiene tres elementos del tipo Sierpinski_punto que son los que almacenan las coordenadas de los tres puntos que componen el triángulo, y también tiene dos enteros, count indica el nivel de desarrollo de ese nodo(a partir de un cierto punto se permite la generación de los hijos) y level que almacena el nivel de profundidad de ese nodo.

El TDA Puntuaciones hace uso un vector que a priori no se sabe su tamaño para almacenar las puntuaciones y los nombres (esos dos--------) el cual asigna su tamaño contando el número de líneas que tiene el fichero de puntuaciones.

El TDA buscaminas funciona exclusivamente con dos matrices de 22x22, una para las minas y los números y otra para guardar si se ha descubierto o no una parte del mapa y también para guardar las banderas. Para descubrir el mapa y todas las posiciones anexas que no sean números se usa recursividad.

Conclusiones

En general ha sido divertido el proyecto, me ha permitido probar multitud de métodos como la programación en TDAs, la generación de fondos con ecuaciones (el fondo del menú esta generado con la ecuación x^2*y^2 para la opacidad), el manejo de estructuras enlazadas, árboles para generar fractales, etc…

Me he encontrado con diversas dificultades que han hecho más interesante el proyecto, comento algunas de ellas:

Al generar muchas balas en pantalla me percaté de que el juego se ralentizaba y para solucionarlo pensé en optimizar el recorrido de la lista enlazada pero no pude, ya que además no tardaba mucho tiempo en recorrerla, el problema estaba en que para mantener unos 66.7 FPS puse el argumento de la función Pantalla_Espera a 15ms, entonces cada vez que llegaba al final del bucle paraba 15ms antes de volver a empezar, si a esto le sumamos el tiempo de ejecución del ciclo en el peor de los casos pasaba a ser 25-30ms por lo que funcionario el doble de lento, para solucionarlo puse un contador de tiempo al principio del bucle (usando las librerías time de C) y al final del bucle compruebo cuanto tiempo ha tardado en ejecutarlo y le paso a la función Pantalla_Espera como dato 15 menos ese tiempo, así logro hacer dinámico la espera entre ciclos ajustándose al tiempo que tarde el ejecutarlo.

Otro problema que tuve fue conseguir hacer un barrido completo de los colores en RGB, para eso lo solucione parametrizando los cambios que se hacen de HSV a RGB.

```
int red = 0;
int green = 0;
int blue = 0;
if(estado <= 255) {
    red = 255;
    green = 0;
    blue = estado;
}else if(estado <= 510) {
    red = 765-estado;
    green = 0;
    blue = 255;
}else if(estado <= 765) {
     ...
```