Little Invaders

Introducción

Para el desarrollo del videojuego nos hemos inspirado en el clásico SpaceInvaders ya que nos parece un juego bastante entretenido y a la vez no muy complejo de desarrollar

Herramientas

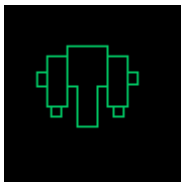
Para desarrollar el juego solamente hemos usando como editor de texto “SublimeText” ya que no existen ningún tipo de sonido o mapa de bits. El juego a sido desarrollado bajo Linux y compilado con el compilador de C++ de GNU, el usado en clase.

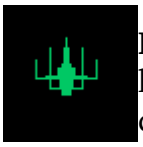
Descripción del juego

El videojuego, como hemos nombrado anteriormente, es un símil del SpaceInvaders por lo que se trata de un juego en 2D en el que el jugador adopta el papel de una nave espacial la cual tiene que acabar con el resto de naves enemigas. Para ello deberá de esquivar todos los proyectiles enemigos y disparar en el momento adecuado para destruirlos.

Lógica del juego

Naves enemigas: Estas naves intentaran acabar contigo disparando,aunque sus disparos están limitados a dos y no podrán volver a disparar hasta que hayan impactado sobre ti o desaparezcan del mapa. Se mueven de un lado a otro de forma oscilante para esquivar tus disparos .





Nave del Jugador: Esta nave es la que controla el jugador, se desplaza horizontalmente a través de las teclas de control y dispara, no puede disparar mas de una bala hasta que no impacte sobre el enemigo o desaparezca del mapa.

Controles

Los controles ofrecidos al jugador por defecto durante la partida son:

Tecla 'A': moverse izquierda.
Tecla 'D': moverse derecha.
Tecla ENTER : Disparar.
Tecla '1': Cerrar Juego.

Interfaz

Al ejecutar el programa aparece el nombre del videojuego junto a un menú formado por:

Jugar: Esta opción ejecuta el videojuego.
Instrucciones: Esta opción muestra unas sencillas instrucciones del funcionamiento del juego para poder jugar.
Salir: Por último esta opción sale del programa.

Little Invaders

Descripción del diseño del videojuego

Lo primero que vamos a explicar son las librerías usadas para el desarrollo del videojuego. Hemos usado :

```
#include "gfx.h"
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <unistd.h>
```

cmath: Se ha utilizado por sus funciones matemáticas como elevar un numero a otro o hacer la raíz cuadrada.
cstdlib: Esta librería nos proporciona la función *rand*().
unistd.h: La función *usleep*() se añade al código a través de esta librería y la usamos para parar el programa durante unos microsegundos.
Por último usamos la librería básica *iostream* y la librería de *gfx* para el juego.

Una vez ya sabemos para que hemos utilizado cada librería vamos a explicar todo el código.

```
const float Kenemy_velocity = 0.5;
```

Al comienzo del código fuente hemos declarado una constante que define la velocidad a la que se mueve el enemigo en horizontal.

A continuación hemos definido los tipos de datos que vamos a usar:

```
typedef struct {
          char type;
          int life;
          float pos_x;
          float pos_y;
          float bala[];
}Tfiguras_str;


typedef Tfiguras_str Tfiguras_list[30];
```

Vamos a analizar cada tipo de dato:

- **type**: en esta variable se guarda el tipo de figura/enemigo se va a guardar. En esta juego solo existe un tipo, “l”, pero se deja a si por si en un futuro se sigue añadiendo

algo mas.

- **life**: en esta variable se guarda la vida de los enemigos, que ira decrementando cada vez que es golpeado por alguna de nuestars balas.
- **pos_x:** esta variable almacena la coordenada X en la que se encuentra el objeto.
- **pos_y**: esta variable almacena la coordenada Y en la que se encuentra el objeto.
- **bala[]**: este array almacena las coordenadas de las balas que tiene cada nave.

bala[0] y bala [1]son las cordenadas x e y respectivamente de la primera bala y bala [2] y bala[3] de la segunda. Por ultimo se crea un tipo de dato array de la estructura anterior.
Seguimos con las funciones:

```
void shot(int x, int y,int mis_balas[], int &n_balas){
     if(n_balas == 0){
          mis_balas[0]=x;
          mis_balas[1]=y;
          n_balas = 1;
     }
}
```

la función shot() se encarga de disparar, es decir, genera una bala en nuestra posición que después sera acelerada hacia los enemigos. Los argumentos de la funcion son las coordenadas x e y del jugador, el array donde se almacenan las coordenadas de las balas y la cantidad de balas.
La función comprueba que no hay ninguna bala en la pantalla y en caso que sea verdad genera una bala en las coordenadas pasadas y aumenta a 1 la cantidad de balas.

Mas funciones :

```
void bullet(Tfiguras_list figuras, int i, int n){
     if(rand()%100 < 5 ){

          int x = figuras[i].pos_x;
          int y = figuras[i].pos_y;

          switch(n){
               case 1:
                              figuras[i].bala[0]=x+5;
                              figuras[i].bala[1]=y-40;
               break;

               case 2:
                              figuras[i].bala[2]=x+30;
                              figuras[i].bala[3]=y-40;
               break;
          }
     }
}
```

Little Invaders

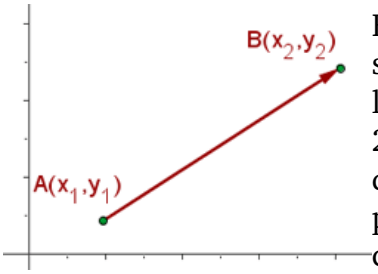
Esta función se ocupa de generar los disparos de los enemigos, para ellos le pasamos el array de las figuras, el tipo de bala que vamos a generar, la bala de la izquierda o la de la derecha,y el numero de la nave a la que estamos haciendo referencia.

La función tiene un 5% de posibilidades de que se genere la bala o no, esto lo hacemos para que no todas salgan a la vez y el juego sea un poco mas dinámico. Por último se generan las balas dependiendo del tipo en las coordenadas respectivas de cada nave.

Función colisión:

```
bool colision(int pos_x,int pos_y, int u, int v){
      if(sqrt(pow(u-pos_x,2) + pow(v - pos_y,2)) < 20 ){
            return(1);
      }else {
            return(0);
      }
}
```

$$|\vec{v}| = \sqrt{x^2 + y^2}$$



Esta función booleana se ocupa de devolver 1 si la distancia que separa dos objetos es inferior a 20px o 0 en caso contrario. Esto lo hacemos calculando el modulo de un vector. La función recibe 2 coordenadas,(pos_x,pos_y)(u,v) y calcula el modulo del vector que forman, si ese modulo es inferior a 20 interpretamos que las posiciones de los objetos se están solapando y por lo tanto están colisionando.

La función draw_world() se ocupa de de dibujar la mayoría de objetos, recibe como parámetros el array de figuras, variables de control y las balas.
El modulo se divide principalmente en dos bucles, el primero se ocupa de dibujar las balas del jugador, para ello incremente a la posición del eje Y 5 y dibuja una linea en las coordenadas de la bala, además comprueba si la bala esta fuera del rango visible y si es así la elimina.

Por otra parte el otro bucle se ocupa de dibujar las naves enemigas y sus balas además de tener unas series de comprobaciones que se ocupan de desplazar las naves en forma horizontal

Además encontramos el modulo play() donde se inicializan la mayoría de datos usados en el juego, se generan las coordenadas de los enemigos y se crea un bucle. Este bucle se ejecuta de manera indefinida hasta que la partida finaliza, de esta forma conseguimos un refresco de la pantalla haciendo el efecto de animación. En cada frame se llama a todas las

funciones anteriores comprobando en tiempo real si cada objeto a colisionado, detectando que tecla ha sido pulsada etc. Hemos usado en conjunto dos funciones que traía GFX para hacerlo:

```
c = gfx_event_waiting();
if(c == 1){
     g=gfx_wait();

     if(g==97) x-=10;
     if(g==100) x+=10;
     if(g==13) shot(x,y, mis_balas, n_balas);
}
```

Por último también encontramos algunos bucles que se ocupan de llamar a la función colision(); para saber si nos han matado o hemos matado a las naves enemigas y darnos la victoria o la derrota. Además también dibuja nuestra nave. También añadir que hemos tenido que añadir la función usleep() para que el juego no se acelerara al máximo del pc.

Por último encontramos las funciones instrucciones() y main(), la primera se ocupa de mostrar una pequeña guía del funcionamiento del juego y la segunda de generar un menú en la consola para poder elegir el juego.

Para compilar el juego usaremos el comando: g++ main.c gfx.c ɵ main łX11 łm