Programando en forth/r3

r3 es un lenguaje de programación derivado del ColorForth.
https://github.com/phreda4/r3

Pablo H. Reda

2023

Este manual esta hecho con la colaboración de alumnos:
Vladimir Gomez Perez
Bianca Cipollone Di Croce

Programando un Juego en r3

Para hacer programas con gráficos necesitamos acceso a esta capacidad de nuestra computadora, para esto, vamos a utilizar la biblioteca SDL: https://www.libsdl.org/

Las palabras que manejan esta biblioteca las encontramos en SDL2GFX.r3

Este listado es la base de cualquier programa gráfico.

```
1   ^r3/win/sdl2gfx.r3                  incluye las palabras gráficas
2
3   :juego
4         0 SDLcls                      limpia la ventan con color 0 (negro)
5         SDLredraw                     refresca la ventana
6         SDLkey                        apila la tecla
7         >esc< =? ( exit )             si soltó escape… sale
8         drop ;
9
10  :main
11        "r3sdl" 640 480 SDLinit       inicia ventana de 640 x 480
12        'juego SDLshow                repite palabra juego hasta exit
13        SDLquit ;                     quita la ventana
14
15  : main ;
```

Este código va a crear una ventana de gráficos.
El programa empieza en la línea 15, llama a la palabra main.
En la línea 11 hace una ventana gráfica de 640 por 480 píxeles.
La línea 12 usa **SDLshow** para hacer un loop con esta palabra, se necesita la dirección de la palabra y no la palabra, hasta que, dentro de esta palabra, se llama a **exit**.
La linea 4 limpia la ventana con el color 0, negro, entre esta palabra y la siguiente podemos dibujar los graficos que queramos hasta la linea 5 donde redibuja la ventana.
La línea 6 obtengo la tecla del usuario.
en la línea 7 comparo si se soltó la tecla ESC y ejecutó **exit** si ocurre esto.

Dibujos del juego

Con esta base empezaremos por crear una hoja de sprites, esto es, un archivo con todos los gráficos que usaremos en el juego. Utilizaremos sprites de 16 x 16 píxeles.

Cada dibujo va a numerarse de arriba a abajo y de izquierda a derecha, comenzando de 0, tenemos 9 sprites.

Comenzamos definiendo una variable para guardar los sprites, y las coordenadas para dibujar
En la línea 19 utilizamos para cargar el archivo gráfico:

```
::ssload | w h file -- sprite
```

El ancho (w) y el alto (h) de cada sprite en la hoja y el lugar y nombre del archivo (file) es lo que va a obtener de la pila la palabra **SSLOAD** y el valor que queda es la referencia a este gráfico. Lo guardamos en una variable y lo vamos a utilizar cuando dibujemos en pantalla.
En la línea 8, para dibujar en pantalla usamos

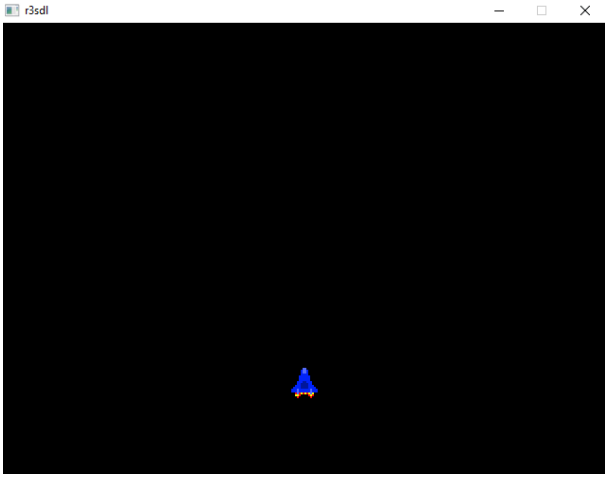
```
::sspritez | x y zoom nrosprite sprite --
```

Aqui X e Y son la posición de píxeles que será el centro de nuestro sprite, el nivel de zoom, aquí un número con punto decimal llamado número de punto fijo..donde podremos achicar o agrandar el dibujo, en nuestro caso lo duplicaremos, luego el número de sprite y por último la hoja donde se encuentra, que es donde lo cargamos.

El código se encuentra en la distribución del lenguaje:

```
1   ^r3/win/sdl2gfx.r3
2
3   #sprites
4   #x 320 #y 380
5
6   :juego
7         0 SDLcls
8         x y 2.0 0 sprites sspritez
9         SDLredraw
10
11        SDLkey
12        >esc< =? ( exit )
13        <le> =? ( -1 'x +! )
14        <ri> =? ( 1 'x +! )
15        drop ;
16
17  :main
18        "r3sdl" 640 480 SDLinit
19        16 16 "media/img/manual.png" ssload 'sprites !
20        'juego SDLshow
21        SDLquit ;
22
23  : main ;
```

r3/curso/manual01.r3



Mejorando el movimiento

En el código, las flechas izquierda y derecha restan y suman a la variable X, donde guardamos la posición de la nave.
Si probamos el programa notamos que el efecto de movimiento no es el deseado, necesitamos que mientras esté presionada la tecla, la nave se mueva y para esto necesitamos tener variables de velocidad y sumar constantemente esta velocidad a la posición.
Para que las velocidad tenga la posibilidad de ser una parte del entero vamos a utilizar números de punto fijo, para esto cuando lo indicamos debemos agregar obligatoriamente el punto decimal, y necesitamos convertirlo a entero cuando indicamos el pixel en pantalla por lo que utilizaremos la palabra

```
::int. | punto.fijo -- entero
```

Utilizamos **<le>** (tecla LEft o flecha a la izquierda) para detectar cuando se oprime la tecla y **>le<** cuando se suelta la tecla, ya que necesitamos cambiar la velocidad en estos dos estados.
Creamos una nueva palabra para separar el código que maneja el jugador, y acá dibujamos el sprite y sumamos las posiciones con las velocidades.
Probando el código notaremos la diferencia en el funcionamiento.

```
1   ^r3/win/sdl2gfx.r3
2
3   #sprites
4   #x 320.0 #y 380.0
5   #xv 0 #yv 0
6
7   :jugador
8         x int. y int. 2.0 0 sprites sspritez
9         xv 'x +! yv 'y +! ;
10
11  :juego
12        0 SDLcls
13        jugador
14        SDLredraw
15
16        SDLkey
```

```
17        >esc< =? ( exit )
18        <le> =? ( -1.0 'xv ! ) >le< =? ( 0 'xv ! )
19        <ri> =? ( 1.0 'xv ! ) >ri< =? ( 0 'xv ! )
20        drop ;
21
22    :main
23        "r3sdl" 640 480 SDLinit
24        16 16 "media/img/manual.png" ssload 'sprites !
25        'juego SDLshow
26        SDLquit ;
27
28    : main ;
```

r3/curso/manual02.r3

Llegaron los extraterrestres

Agregamos otro elemento al programa, el alien, necesitamos la posición en **XA** y **YA**, y moverlo, no con las teclas, sino que sumando constantemente a **YA**, que va a ser que baje en la pantalla.
La línea 11 reinicia el alien, **XA** será un número al azar entre 0 y el ancho de pantalla, en punto fijo.
La línea 16 va a reiniciar el alien cuando salga por abajo, o sea, la coordenada **YA** sea mayor al alto de la ventana.
La línea 24 primero dibuja el jugador y luego el alien.

```
..
7
8     #xa 0.0 #ya 100.0
9
10    :+alien
11        -16.0 'ya ! 640.0 randmax 'xa ! ;
12
13    :alien
14        xa int. ya int. 2.0 2 sprites sspritez
15        2.0 'ya +!
16        ya 480.0 >? ( +alien ) drop ;
..
24        jugador alien
..
```

r3/curso/manual3.r3

Disparos

Agregamos otras dos posiciones para guardar las coordenadas del disparo. Necesitamos indicar cuando está presente el disparo y para esto, en este caso, elegimos utilizar la variable XD, cuando sea negativa será que no hay disparo. Solamente podemos tener un disparo en pantalla.
La palabra disparo va a dibujar la bala y modificar su posición. la línea 26 consulta sobre la variable xd y si esta es negativa, termina la palabra, esto es, no hace mas nada aqui.
La línea 27 se encarga de dibujar el disparo.
La línea 28 resta 4 a la posición en Y, quiere decir que irá subiendo en pantalla.

La línea 29 pregunta si la posición en YD es negativo, o sea, sale de pantalla, si ocurre esto pone en negativo el X para que no entre a esta palabra más, por la línea 26.
La palabra +disparo primero controla que no haya una bala en la pantalla, y si es posible, copiar las coordenadas de nuestra nave en las coordenadas de la bala para que funcione este mecanismo.
La línea 44 pregunta si se presiona la tecla ESCAPE y ejecuta +disparo cuando esto ocurre.

```
..
22    #xd #yd
23
24    :disparo
25          $ffffff SDLColor
26          xd -? ( drop ; ) | si es negativo sale
27          int. 1 - yd int. 3 8 SDLFRect | dibuja un rectangulo relleno
28          -4.0 'yd +!
29          yd -? ( -1.0 'xd ! ) drop ;
30
31    :+disparo
32          xd +? ( drop ; ) drop
33          x 'xd ! y 'yd ! ;
..
37          disparo jugador alien
..
```

r3/curso/manual04.r3

Colisión y contar puntos

Definimos una variable para almacenar los puntos en la línea 35. Vamos a agregar la detección de la colisión entre la bala y el alien.
Para esto, en la línea 41, vamos a calcular la distancia que hay en x entre estas dos variables XA y XD, restando y obteniendo el valor absoluto y calculando que si hay más 16 pixeles, no va a producirse el choque.
Vamos a hacer lo mismo para la coordenada Y,en la línea 42, pero aquí calcularemos que la distancia sea de 8 pixeles, esto es, porque el dibujo del alien es mas ancho que alto, variando estos numeros sera mas dificil o mas facil pegarle.
Esto se realiza en la palabra CHOCO?, donde si se cumplen las condiciones, entonces sumaremos 1 al puntaje, borraremos el disparo y agregaremos un nuevo alien, borrando el anterior.
La línea 48 agrega al bucle principal esta comprobación del choque de la bala y el alien.

```
..
35    #puntos 0
36
37    :+punto
38          1 'puntos +! -1 'xd ! +alien ;
39
40    :choco?
41          xa xd - abs 16.0 >? ( drop ; ) drop
42          ya yd - abs 8.0 >? ( drop ; ) drop
```

```
43          +punto ;
..
48          choco?
..
```

r3/curso/manual5.r3

Perder también hay que programarlo

Para que sea totalmente funcional falta agregar una variable de vidas, en la linea 37 y agregar cuando el alien choca a nuestra nave.
La forma de hacerlo es igual. Definiendo la palabra perdio? en la linea 51 y 52 calculamos que tanto en la coordenadas X como en Y la distancia entre las coordenadas del jugador y el alien sean menores a un umbral definido. Cuando esto ocurre se resta la cantidad de vidas y se comprueba que las vidas sean 0 para salir del programa llamado a la palabra exit.
las líneas 55 y 56 retornan la nave al inicio y agrega un nuevo alien, quitando el anterior para que continúe el juego.

```
..
50    :perdio?
51          xa x - abs 16.0 >? ( drop ; ) drop
52          ya y - abs 16.0 >? ( drop ; ) drop
53          -1 'vidas +!
54          vidas 0? ( exit ) drop
55          320.0 'x !
56          380.0 'y !
57          +alien
58          ;
..
63          choco? perdio?
..
```

r3/curso/manual06.r3

Mostrar texto

Para utilizar texto en una ventana gráfica, tenemos varias posibilidades. Veamos como unas de las más simples. Incluyendo a la biblioteca bfont, tenes dos fuentes de ancho fijo, en la línea 79 elegimos la fuente.
Usamos para ubicar dónde se va a imprimir el texto
bat | x y --

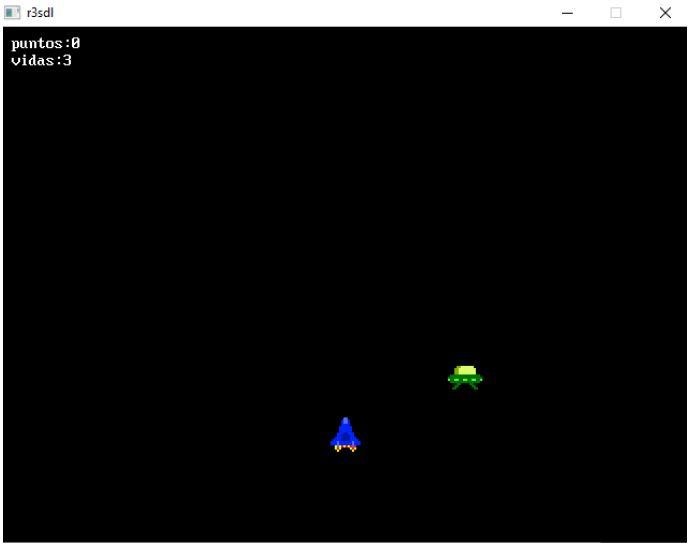
para imprimir un texto
bprint | “” --

tener en cuenta que bprint no produce el texto con los modificadores de %, por eso acá usamos sprint para convertir a un texto con valores de la pila.

Las líneas 66 y 67 preparan el texto para mostrar vidas y puntos y lo imprimen en el lugar indicado por bat

```
..
3       ^r3/util/bfont.r3
..
66            10 8 bat puntos "puntos:%d" sprint bprint
67            10 24 bat vidas "vidas:%d" sprint bprint
..
79            bfont1
..
```

r3/curso/manual7.r3



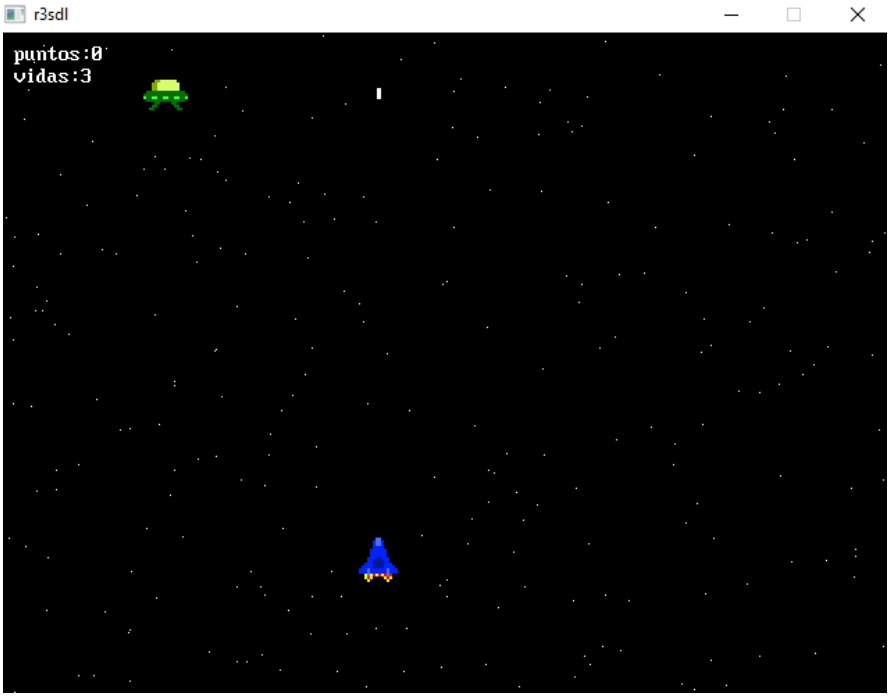
Estrellas de fondo

Vamos a agregar un fondo de estrellas que se mueven al juego utilizando un buffer de memoria. Necesitamos guardar posiciones X e Y para cada estrella y vamos a dibujar un pixel en estos lugares.
Para guardar estas posiciones utilizamos una parte de memoria, y elegimos guardar los números en 16 bits, 2 bytes, por eso utilizamos w!+ y w@+.
La palabra +estrella agrega las coordenadas X e Y en memoria en orden inverso dibuja.estrellas recorre en buffer y extrae estas coordenadas y llama a dibujar cada punto
Para mover las estrellas, en este caso vamos a incrementar en 1 la coordenada Y, cuando supera el límite de la pantalla (variable SH) vuelve a 0 para que aparezca por el otro lado de la ventana.
Por último tenemos una palabra para agregar las estrellas que necesitamos, SW es el ancho de la ventana, SH es el alto, **RANDMAX** produce un número aleatorio entre 0 y el valor de la pila.
Como indicamos un tamaño de 1024 bytes y cada estrella ocupa 4, 2 para x y 2 para Y, podremos guardar 256 coordenadas, una parte de la programar consiste en definir límites y cantidades
Necesitamos agregar el dibujo y movimiento de las estrellas al bucle principal, en la linea 84 y rellenar las coordenadas al inicio del programa

```
..
5       #buffer * 1024                | lugar para guardar las coordenadas
6       #buffer> 'buffer
7
```

```
8     :+estrella | x y -
9           buffer> w!+ w!+ 'buffer> ! ;
10
11    :.estrellas
12    $ffffff sdlColor
13          'buffer ( buffer> <?
14                w@+ swap
15                over 1 + sh >? ( 0 nip )
16                over 2 - w!
17                w@+ rot
18                SDLPoint ) drop ;
19
20    :llenaestrellas
21          256 ( 1? 1 -
22                sw randmax sh randmax +estrella
23                ) drop ;
..
84          .estrellas
..
102         llenaestrellas
..
```

r3/curso/manual08.r3



Animar

Los sprites de la hoja consisten en una imagen con las diferentes dibujos que representan animaciones, generalmente se trata de secuencias que representan caminar, correr, saltar, esperar para personajes o estado para objetos como, vaso lleno o vaso vacío.
La forma de utilizar estos gráficos es, primero, organizar estas animaciones en una grilla regular, es decir, todos los casilleros del mismo tamaño. Este tamaño se indica cuando se carga la hoja.
Tenemos palabras que van a dibujar este sprite, indicando el número de gráfico, este número comienza en 0 y va de izquierda a derecha y de arriba a abajo.

Para animar lo que necesitamos es alternar este número en el tiempo, utilizamos una palabras construidas para este fin

Muchos sprites

A veces necesitamos dibujar muchas instancias de un determinado sprite.
Para esto usaremos una biblioteca que permite manejar una porción de memoria como lista de multi celdas con una capacidad de 16 números.

```
^r3/util/arr16.r3
```

Este módulo está construido para realizar exactamente lo que necesitamos.
Primero, reservar la cantidad de memoria necesaria para contener un máximo indicado por el programador, y necesitamos una variable para indicar a qué lista hacemos referencia.
Se necesita una variable con dos valores para guardar este mecanismo

```
#listalien 0 0
```

para inicializar vamos a definir cuál es la capacidad máxima de esta lista

```
200 ‘listalien p.ini
```

Cada elemento van a ser 16 celdas de memoria, en la primera vamos a poner una dirección de palabra especial que va a dibujar el sprite en pantalla y calcular la próxima posición.
cada vez que se llame a esta palabra, va a recibir de la pila la dirección de memoria donde está el inicio de las 15 celdas restantes y debe quitarla de la pila al finalizar la palabra.
Si queremos que este sprite desaparezca, vamos a dejar en la pila un 0.
En estas celdas podemos guardar las variables que queramos, como la posición, la velocidad de desplazamiento, la animación del sprite, etc.
Estos valores se acceden o modifican por la posición relativa a la dirección que tenemos en la pila, Aunque si queremos podemos definir un palabra que indique a qué nos estamos refiriendo.
Es conveniente definir las posiciones consecutivas de las celdas como datos que vamos a necesitar para, por ejemplo, dibujar nuestro sprite, así, como estamos utilizando:

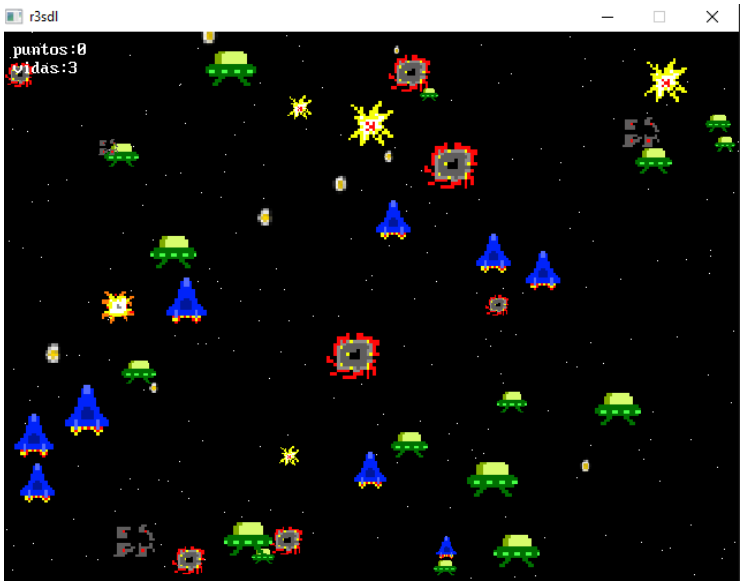
```
sspritez | x y z nro sprite –
```

Podemos usar ese orden para guardar los valores, usando el registro A podemos recorrer y usar estos valores, líneas 26 a 32.
Para dibujar constantemente en pantalla debemos agregar en el bucle de dibujo, antes de SDLredraw, línea 69.
Podríamos agregar estos elementos con f1 para probar, entonces cuando testeamos las teclas presionadas agregamos, línea 80.
Esto debería agregar dibujos a la pantalla

```
..
15    #listalien 0 0 | lista de aliens
..
```

```
25    |-------------- Aliens
26    :alien | adr -
27          >a
28          a@+ int. a@+ int. | x y , convertidos a enteros
29          a@+ | zoom
30          a@+ | nro de sprite
31          a@+ | hoja de sprite
32          sspritez ;
33
34    :+alien
35          'alien 'listalien p!+ >a
36          640.0 randmax a!+ | x
37          480.0 randmax a!+ | y
38          2.0 randmax 1.0 + a!+
39          12 randmax a!+
40          sprites a!+
41          ;
..
69          'listalien p.draw
..
80          <f1> =? ( +alien )
..
88          200 'listalien p.ini
..
```

r3/curso/manual9.r3



Utilizando este recurso vamos a utilizar una lista de sprites para los disparos y otra para los aliens, aunque es posible utilizar una lista para ambos, a la vez, es conveniente separarlos porque cuando comprobemos si una bala chocó con un alien, esta separación nos simplifica el código ya para cada bala vamos a recorrer la lista de aliens. Si no está separado debería comprobar en cada recorrido el tipo de cada sprite.

```
..
```

```
14    #listalien 0 0 | lista de aliens
15    #listshoot 0 0 | lista de disparos
..
42    |-------------- Disparo
43    #hit
44    :choque  | x y i n p -- x y p
45          dup 8 + >a
46          pick4 a@+ - pick4 a@+ -
47          distfast 20.0 >? ( drop ; )   drop
48          dup 'listalien p.del
49          1 'puntos +!
50          0 'hit !
51          ;
52
53    :bala | v --
54          objsprite
55
56          1 'hit !
57          dup @ | x
58          over 8 + @ -20.0 <? ( 3drop 0 ; ) | y ; fuera de pantalla
59          'choque 'listalien p.mapv | 'vector list --
60          2drop
61          hit 0? ( nip ; ) drop
62          drop
63          ;
64
65    :+disparo
66          'bala 'listshoot p!+ >a
67          x a!+ y 16.0 - a!+| x y
68          1.0 a!+     | zoom
69          8 2 10 vci>anim | vel cnt ini
70          a!+ sprites a!+   | anim sheet
71          0 a!+ -3.0 a!+    | vx vy
72          ;
73
74    |-------------- Alien
75    :alien | v --
76          objsprite
77          dup @ x -
78          over 8 + @
79          500.0 >? ( 3drop 0 ; ) | llego abajo?
80          y - distfast
81          30.0 <? ( pierdevida )
82          drop
83          drop
84          ;
85
86    :+alien
87          'alien 'listalien p!+ >a
88          500.0 randmax 70.0 + a!+
89          -16.0 a!+
90          2.0 a!+     | zoom
91          7 4 2 vci>anim | vel cnt ini
92          a!+   sprites a!+             | anim sheet
93          2.0 randmax 1.0 -
```

```
94        a!+ 2.0 a!+ | vx vy
95        ;
96
97    :horda
98        50 randmax 1? ( drop ; ) drop
99        +alien
100       ;
..
127       'listalien p.draw
128       'listshoot p.draw
..
148       200 'listalien p.ini
149       200 'listshoot p.ini
..
```

r3/curso/manual10.r3

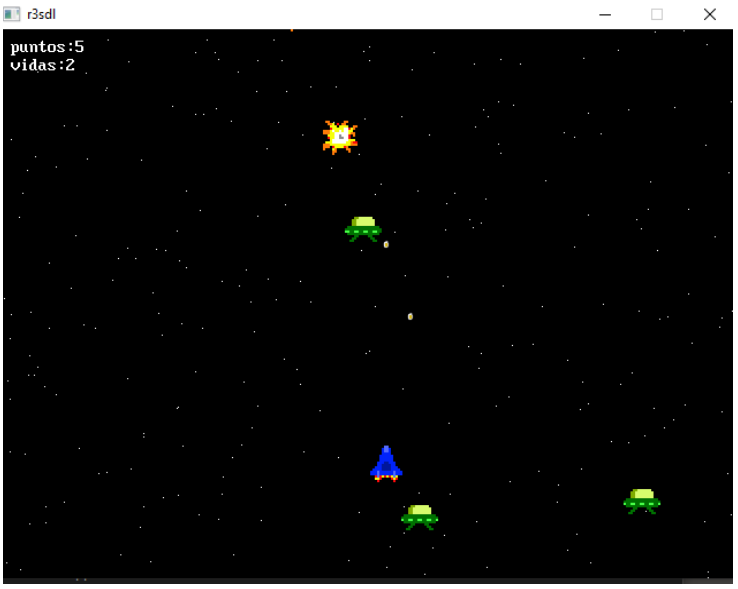
Explosiones y Efectos especiales

Los efectos especiales tienen como objetivo decorar lo que está pasando en la pantalla, la mayoría de las veces no influyen en el funcionamiento del juego, por lo tanto, vamos a construirlo con una lista de sprites que solamente se crean cuando ocurre algo y se destruyen automáticamente.
Vamos a agregar una lista sprites para las explosiones de los aliens, podríamos tener varios tipos de efectos especiales, y aquí si, agruparlos en una sola lista porque no van a tener interacción con el juego, son solo decorativos. Podríamos poner nubes, estrellas, efectos de chispas, quizás necesitaríamos varias listas si utilizamos distintos planos para estos sprites, es decir, que se dibujen antes o después del jugador y de los enemigos.
En la línea 24 preguntamos si estamos en el último cuadro de la animación de la explosion para quitar el sprite.
La línea 26 agrega la explosion cuando se produce la colisión del disparo y el alien utilizando las coordenadas del mismo.

```
..
16    #listfx 0 0 | fx
..
28    |--------------- fx
29    :explosion
30        objsprite
31        24 + @ nanim 10 =? ( drop 0 ; )
32        drop
33        ;
34
35    :+explo     | y x --
36        'explosion 'listfx p!+ >a
37        swap a!+ a!+      | x y
38        2.0 a!+                 | zoom
39        7 5 6 vci>anim          | vel cnt ini
40        a!+   sprites a!+ | anim sheet
41        0 a!+ 0 a!+       | vx vy
42        ;
..
```

```
68          pick4 pick4 +explo
..
178         200 'listfx p.ini
..
```

r3/curso/manual11.r3



Sonidos y Música

El sonido y la música agregan otro nivel de realismo al juego, para esto vamos a utilizar la biblioteca de música y sonidos de SDL2

```
^r3/win/sdl2mixer.r3
```

Al inicio del programa vamos a cargar los archivos de música o sonidos que se van a disparar durante la ejecución.
Para esto vamos a definir variables para guardar las referencias a estos elementos, podemos cargar un sonido o un tema musical completo
Las palabras para cargar los archivos de sonido son:

```
::Mix_loadMus | filename -- referencia
```

Carga en memoria un archivo de sonido que será usado como música, generalmente de larga duración.

```
::Mix_LoadWAV | filename -- referencia
```

Carga un archivo de sonido que será usado como efecto, generalmente de corta duración.
Los principales formatos de sonidos son .wav, .mp3 o .ogg, tanto para música como para sonido.

```
::Mix_PlayMusic | musica loop —
```

Reproduce un archivo de sonido, generalmente largo, música es la referencia que se cargó con Mix_LoadMus y loop será la cantidad de veces que se va a reproducir, con -1 se repetirá infinitamente, hasta que se apague, terminando la reproducción o comenzando otra música.

```
::SNDplay | sonido —
```

Reproduce un sonido, lo que necesita de la pila es la referencia que se cargó con Mix_LoadWAV, esta biblioteca mantiene varios canales de sonido, quiere decir que mientras suene la música pueden aparecer varios efectos de sonido, como explosiones o disparos y sonarán todos a la vez. La biblioteca posee más posibilidades pero con solamente estas palabras es posible hacer lo básico.

```
..
8       #snd_shoot        | sonido de disparo
9       #snd_explode| sonido de explosion
..
45            snd_explode SNDplay | en +explo
..
94            snd_shoot SNDplay | en +disparo
..
172           "media/snd/shoot.mp3" Mix_LoadWAV 'snd_shoot !
173           "media/snd/explode.mp3" Mix_LoadWAV 'snd_explode !
..
```

r3/curso/manual12.r3