**Juego/Lazarus dinosaurio**
**==========**

Hice este programa para enseñarle a un niño de 10 años algo de programación. Y lo mejor es que fuese un juego. Así que este proyecto es acerca de una versión muy ligera del juego del dinosaurio de Chrome. Ver en wikipedia si no sabe cuál es el juego https://es.wikipedia.org/wiki/Dinosaur_Game

Como la idea es aprender **Object Pascal** como lenguaje de programación (usando **Lazarus** https://www.lazarus-ide.org/ Delphi debería servir igual).

**Los juegos 2D en general**
**==========**

El juego original es un "endless runner de scroll lateral" (textual de wikipedia) lo que significa que se corre por siempre de forma lateral. Sólo se puede saltar o agacharse (en esta versión solo saltar) para esquivar obstáculos, en este caso un **cactus**. La idea es aguantar durante el mayor tiempo posible para conseguir la puntuación más alta.

El juego cada 250 puntos se acelera un poco, como hasta llegar a 1000 puntos, luego de allí no acelera más.

Este juego es 2D, hay un fondo que se mueve hacia la izquierda, y el dinosaurio da saltos, solo eso, esquivando un **cactus**

**Recursos básicos requeridos**
**==========**

En cuanto a recursos, Internet nos ha provisto de la imagen del dinosaurio, una de un **cactus** y un sonido para cuando se choca con el **cactus** (algo exagerado el sonido, pero sirve).

En todo caso tenemos una carpeta img con las imágenes en archivo png. Y un archivo wav para el sonido.

"uplaysound", esto es un componente para poder escuchar archivos de sonido wav (onda), es el sonido al chocar con el **cactus**). En la página de **lazarus** hay toda una "breve" explicación sobre como reproducir sonidos, alguien se la leyó y creó una rutina simple y la metió en este componente.

| NOTA: |
| --- |
| A veces hay que descargar los componentes y hacer todo un proceso de instalación, que buenos a veces se complica, pero **lazarus** de hace algún tiempo cuenta con una herramiente bajo el menu paquete llamada "**Online Package Manager**". La abrimos y luego de segundos |

vemos una lista de paquetes disponible. Buscar "**playsound**", se selecciona y se manda a instalar, mientras se puede cantar o escuchar algo de música, el proceso es complejo pero lazarus lo hace sin problema.

**Lazarus**
**=======**

Lazarus es un **IDE** de desarrollo sobre Object Pascal, utiliza **freepascal**. Es multiplataforma (e incluso genera código estando en una plataforma para otra). Esta modelado teniendo en cuenta Delphi con una versión propia de VCL. Como son bastantes compatibles es común encontrar bibliotecas de código abierto que compilan tanto en **Delphi** como en **Lazarus**. **Lazarus** es ***Open Source***.

La mayoría de estos proyectos nos envían a descargar desde algún sitio todo el código fuente, puede hacerlo si lo desea, pero la idea es aprender así que vamos paso a paso.

Arrancando el **IDE**, el primer paso sería **crear un nuevo proyecto de tipo aplicación**.

Un proyecto de este tipo tiene un archivo de proyecto, un par de archivo para el formulario principal y algunos otros de configuración. El **IDE** nos coloca por defecto ante el formulario principal.

El formulario principal, y cualquier otro se compone de una parte gráfica y código (pascal). Alternamos entre ellos por medio de la tecla **F12** (también hay un menú de contexto y en el formulario principal bajo el menú "Ver").

La estructura de un programa en pascal. Es algo parecido a

```
Programa XXX;

begin
  ..... una serie de instrucciones.

end
```

Y de hecho es más o menos así el proyecto, pero el IDE lo gestiona bastante bien por lo que no solemos tener que modificar nada allí. El formulario principal ya no es tanto pascal puro y si mucho Object Pascal. Como tal es una unidad (comienza como "**unit**" ya no como "**program**"). Por lo general estas unidades de formulario comienzan así:

```
unit umain;
```

```
{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls,
ExtCtrls, uplaysound;

type

  { TForm1 }

  TForm1 = class(TForm)
  :
  :
  :
```

Se define nombre de la unidad, se le da una instrucción al compilador (modo de compilación que no nos interesa mucho, pero tiene que ver si se mejora o no la compatibilidad con Delphi). La "interface" que simplemente es declarar lo que vamos a hacer (luego implementamos el código más abajo). El "uses" es una lista de otras unidades que requerimos, y aquí hacemos unos comentarios.

La lista luego del **uses** son los nombres de unidades donde reside el código de las componentes que utilizaremos, por ejemplo **Classes** y **Sysutils** para tipos de datos o funciones de conversión de unidades. **Forms**, por ejemplo lo relativo a formularios y así seguimos. Nótese que usamos "uplaysound" que NO viene por defecto en lazarus.

Uno mismo puede escribir el nombre de las unidades (en particular cuando son creadas por nosotros), pero en cada ocasión que colocamos controles en el formulario y compilamos, **lazarus** agrega las unidades que sean necesarias.

Nuestro formulario principal es como pueden notar una clase (**class**) y hereda de una clase ya definida de nombre "TForm". En este punto le diría que revise en internet que son clase, pero no, no lo voy a decir. Explicaré lo básico.

Una clase es la definición de un paquete que contiene funciones y procedimientos (los que llamamos métodos) y campos que nos dan o almacenan valores (básicamente  variables, pero puede variar un poco esa definición). Esta clase además desciende de una previa y nos hereda muchas funciones y métodos útiles.

NOTA

Por lo general programamos con nombres en inglés, y los comentarios en el idioma al que queremos llegar, pero en este

caso traté de dejar los nombre en castellano porque creo que facilita explicar varias cosas.

En este proyecto debemos llegar a algo como esto.

```
  TForm1 = class(TForm)
    Img2Pie: TImage;
    ImgPieIzq: TImage;
    ImgPieDer: TImage;
    ImgCactus: TImage;
    LblPaisaje: TLabel;
    LblPaisaje1: TLabel;
    LblScore: TStaticText;
    playsound1: Tplaysound;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: char);
    procedure Timer1Timer(Sender: TObject);
  private
    Img :  TImage;
    posicion, puntos, Salto : integer;
    ind_pie : integer;

    procedure PasoIzq;
    procedure puntaje;
    procedure EstablecerImagenDinosaurio;
    procedure ElCactusViene;
  public
```

Nótese que luego de declarar **TForm** hay una lista de controles y componentes que han sido colocados visualmente, esto es se seleccionan de la paleta de controles del IDE y se llevan al formulario principal a la posición que conviene.

**Eventos**
**=======**

Los formularios, controles y componente suelen responder a eventos. Típico evento **OnClick**, **OnShow**, **OnConnect** y son rutinas que se disparan o inician cuando ocurren ciertas situaciones como un Clic del ratón o que se muestre un formulario.

Seleccionado el formulario, hay en lazarus (y delphi) un inspector de proyecto. El inspector nos muestra propiedades y eventos. Entre los eventos buscamos "**FormCreate**" y luego de doble clic se crea la rutina donde va el código. Algo como esto.

```
    procedure TForm1.FormCreate(Sender: TObject);
    begin
    end;
```

que corresponde a la entrada

```
    procedure FormCreate(Sender: TObject);
```

en la declaración de la clase del formulario **Form1.**

Lo que está en la secciones **private** y **public**, no se genera automáticamente, las escribimos nosotros.

**El algoritmo**
**==========**

Este tipo de juegos, y la mayoría trabajan en un ciclo ejecutándose unas 60 veces por segundo que actualiza todos los elementos de la pantalla. Así que en algún momento vamos a requerir un ***Timer***. Bien, lazarus ya nos provee esto como un componente de arrastrar y soltar sobre el formulario principal.


El timer (timer1) tiene pocas propiedades fijamos el lapso 100. Esto implica que cada 100 milisegundos disparara un evento. Doble clic sobre el icono del timer en el formulario nos lleva a la implementación del evento (también por inspector de propiedades donde esta el Tab de Eventos).

El evento ejecutará 4 procedimientos. Quedará algo como esto:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  PasoIzq;
  puntaje;
  EstablecerImagenDinosaurio;
  ElCactusViene;
end;
```

La rutinas PasoIzq, puntaje, etc las estableceremos luego.

**Los elementos del fondo**
**==========**

Nuestro fondo es bien simple, Algo como una texto

“_____--____-________----___”

El texto lo colocamos en una etiqueta (para eso son) y lo desplazamos a la izquierda cada ciclo.

La etiqueta tiene un tamaño finito, de hecho el ancho de nuestra ventana (o más). Cuando se mueve a la izquierda deja espacio libre a la derecha, entonces se dibuja otra etiqueta similar. Una vez que la primera etiqueta se pierde de vista, la dibujamos al final

y con eso damos la ilusión de continuidad.

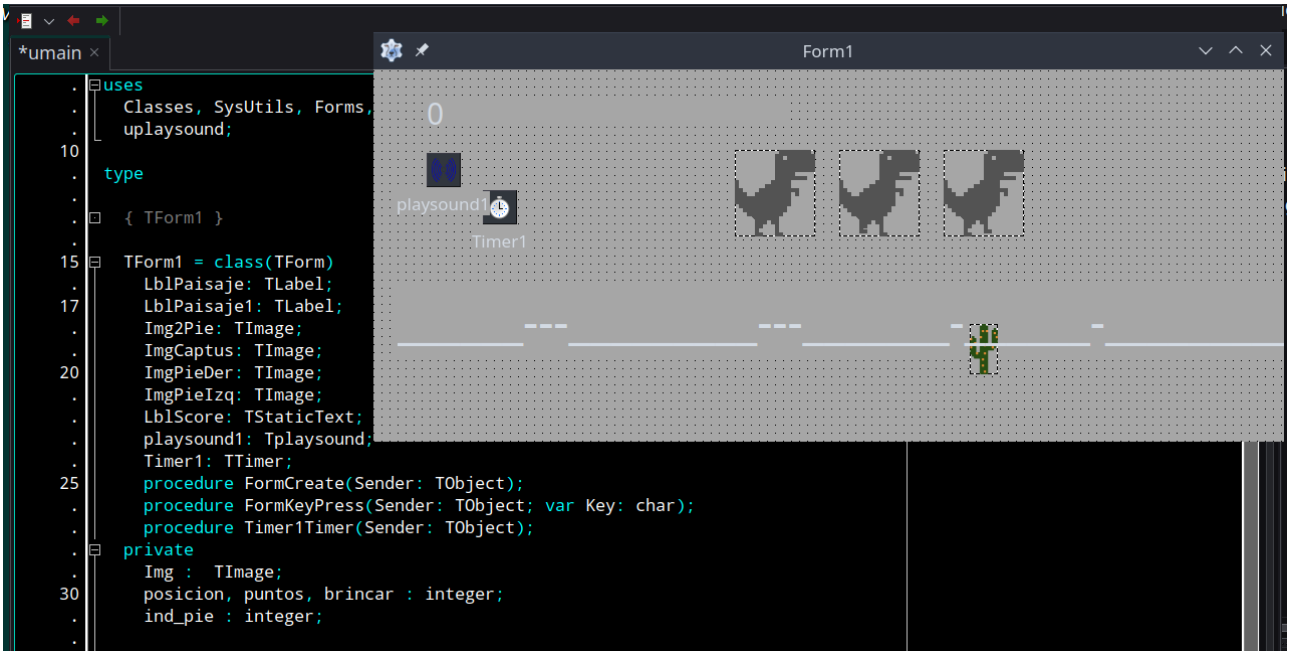
Las etiquetas tiene como nombre LblPaisaje y LblPaisaje1 ambas de tipo Tlabel (Parte de la paleta Standard de controles).

En alguna ocasión dibujamos el **cactus**. Y verificamos si hay choque. Este es un puto algo álgido, podríamos usar una biblioteca de funciones que maneje sprites y colisiones por nosotros, pero haremos algo rudimentario que funcione.

Para esto se ha colocado un control **Timage** al formulario. En la propiedades doble clic en el campo "**Picture**". Mostrará un formulario para cargar archivos gráficos. En el proyecto están dentro de la carpeta img

Bien, volviendo al ciclo, en cada ciclo se requiere:

- Dibujar el fondo un poco más a la izquierda que en el ciclo anterior.
- Establecer el puntaje y presentarlo en la pantalla.
- La imagen del dinosaurio (un pie o el otro o saltando).
- Verificar si el cactus viene y si hay choque.




**Z-order**
**=======**

Los controles, visualmente están unos atrás y unos adelante. Si creamos primero las etiquetas quedan atrás, y los controles más recientes adelante. También se puede alterar el orden por clic derecho sobre el control buscar Orden-Z y allí traer al frente o llevar al fondo según se requiera.

Colocado el control de timer en el formulario le establecemos intervalo en 100 (miliegundos). Es decir que el timer disparará un

evento unas 10 veces cada segundo.

**Rutinas a escribir**
**==========**

En la clase **TForm**, debajo del **private** deberemos agregar esto

```
procedure PasoIzq;
```

Lo que corresponde a una rutina simple llamada PasoIzq, colcoando el ratón sobre esta declaración presionamos **<Shift-Ctrl-C>** con lo que el IDE de lazarus generará la implementación pro nosotros, si ya la tenemos hecha **<Shift-Ctrl-↓>** no llevará a allí.

Este sería la implementación vacía:

```
procedure TForm1.PasoIzq;
begin
end;
```

Allí debemos calcular la posición actual y ajustar las etiquetas conforme a ello. Como vamos a necesitar la posición varias veces, por ello la almacenamos en un campo (variable) llamado posicion que hemos declarado en la sección **private** de la clase **TForm**.

```
  private
    Img :  TImage;
    posicion, puntos, Salto : integer;
    ind_pie : integer;
```

También declaramos **puntos** y **Salto** como enteros la primera para llevar la cuenta del score, la segunda cuando se hace el saldo.

| NOTA |
| --- |
| Tendríamos que considerar **puntos** como una propiedad y no como un campo entero que es un variable con alcance dentro de la clase. Siendo una propiedad la podemos establece de solo lectura de forma que ninguna rutina externa la altere, podría actualizar la etiqueta del score directamente. |

La rutina PasoIzq hace dos cosas:

Le resta a la variable posición un paso (que son como 25 pixeles). Se le resta así:

```
posicion := posicion - paso;   // Un paso a la izquierda
```

y luego a cada etiqueta de paisaje se le ubica en posición

```
LblPaisaje.Left:= posicion;
LblPaisaje1.Left:= LblPaisaje.Left + LblPaisaje.Width + 1;
```

La segunda etiqueta comienza un pixel más allá que la primera, por eso se inicia en el punto izquierdo + ancho de la primera etiqueta.

**Variables**
**=========**

Por el bien del encapsulamiento tratamos de mantener todo dentro de la clase y necesitamos 5 variables enteras que deben subsistir fuera de las rutinas donde se les llama.

| Identificador de variable | descripción |
|---|---|
| posicion | Aunque no le hemos definido valor por defecto, se inicia en 0. Es la posición donde se coloca la etiqueta con el paisaje de fondo. Luego cada ciclo se mueve a la izquierda (por lo que tiene un valor negativo fuera del formulario). |
| puntos | Un contador de puntos de score, a veces se reinicia en 0. |
| Salto | Un salto tiene una serie de posiciones desde 0, abajo a 6 en la cúspide. |
| ind_pie | Va de 0 a 3, según pie abajo, arriba izq... |
| Img | Esto es un apuntador en el sentido más estricto. Es decir nos señala que imagen estamos usando o mostrando (tenemos 3 imágenes que varían por los pies del dinosaurio) |

Para no tener que recordar cuanto se mueven las cosas (25 en este caso) definimos una constante de nombre **paso** con valor 25, y una de nombre **base** con valor 200 (tope de la imagen del dinosaurio). Cada ocasión que se mueve algo lo movemos un paso. Si queremos de pronto trabajar con una resolución de monitor de vídeo muy baja, podemos llevarlo a un valor como de 8 por ejemplo.

**+ Rutinas**
**=========**

Puntaje, incrementa los puntos

```
inc(puntos);
```

Además, escribe ese valor en la etiqueta de score

```
LblScore.Caption := puntos.ToString;
```

La versión actual de este programa tiene dos cosas mas en cuenta.

Si el **score** es igual a 20 lo coloca en el color por defecto. La razón que es que al chocar se coloca en rojo, así que durante los primeros 20 puntos luego de un choque se ve en rojo.

En valores de **score** de 250, 500 y 100 se modifica el **Interval** del **timer**. Así que mientras más alto el puntaje se generan ciclos en tiempo más cortos, esto hace que el juego se acelere.

La siguiente rutina o proceso es EstablecerImagenDinosaurio establece que imagen del dinosaurio se muestra, tenemos 3 imágenes con diferencia entre los pies. La variable ind_pie tiene cuatro valores posibles. Tendría que mostrarnos en valor 0, ambos pies abajo, valor 1 un pie arriba (izq), valor 2 ambos abajo, valor 3 el otro pie arriba y ya.

Para incrementar ind_pie tendríamos que sumarle 1,

```
ind_pie:= (ind_pie + 1)
```

podríamos luego preguntar si el número es mayor a 3 y convertirlo en 0, pero en las computadores es muy común tener un operador de modulo. Esto nos permite escribir:

```
ind_pie:= (ind_pie + 1) mod 4;
```

mod 4 es el resto de dividir entre 4, así que si ind_pie+1 vale 4, el resto es 0 y hemos vuelto a iniciar el contador.

La variable img está apuntando a la imagen que se muestra, la ocultamos apagando su visibilidad.

```
img.Visible:= false;
```

Luego vemos según el valor de img_pie cuál es la siguiente imagen, por ejemplo si ind_pie es 1, entonces

```
img := ImgPieIzq;
```

Podemos usar un par de if y resolver esta lógica o usar una estructura case que pascal nos facilita y quedaría así:

```
    Case ind_pie of // base, sube izq,  base, sube der
    0: img := Img2Pie;
    1: img := ImgPieIzq;
    2: img := Img2Pie;
    3: img := ImgPieDer;
    end;
```

Ya tenemos la imagen se puede hacer entonces visible.

```
img.Visible:= true;
```

Ya en este punto tenemos que imagen del dinosaurio mostrar, pero es posible que esté saltando. Definimos una variable para anotar que que posición va el salto.

El salto, tiene posición en el suelo, 6 comenzando a subir, 5 subiendo a medio camino, 4 y 3 en el tope, 2 bajando, 1 mas abajo, 0 en el suelo.

Este trozo de código cambia la altura del dinosaurio según la variable de salto.

```
    if Salto>0 then
      begin
        Salto := Salto - 1;

        Case Salto of
        6: Img.Top := base - 2*paso;
        5: Img.Top := base - 3*paso;
        4: Img.Top := base - 4*paso;
        3: Img.Top := base - 4*paso;
        2: Img.Top := base - 3*paso;
        1: Img.Top := base - 2*paso;
        0: Img.Top := base;
        end;

      end
    else
      Img.Top := base; // No quedamos en el piso.
```

Simple.

**Colisión**
**========**

Por último ElCactusViene para cerrar labores en el ciclo que generó el timer.

El cactus esta como imagen en un control de nombre **ImgCactus**. Este control normalmente es invisible. Cuando es invisible debemos decidir si lo mostramos, por ello tiramos los dados, en nuestro caso una dado de 40 números  (podríamos aumentar la dificultad colocándolo a 20 números en lugar de 40 por ejemplo). **Random**(40) nos da un numero al azar de 0 a 39, si preguntamos **Random**(40) = 5 o **Random**(40) = 17 es lo mismo, ocurrirá en **promedio** una vez de cada 40.

Cuando decidimos hacer el cactus visible lo colocamos al lado derecho de la pantalla y lo hacemos visible (aunque no se verá hasta que comience a moverse en el siguiente ciclo).

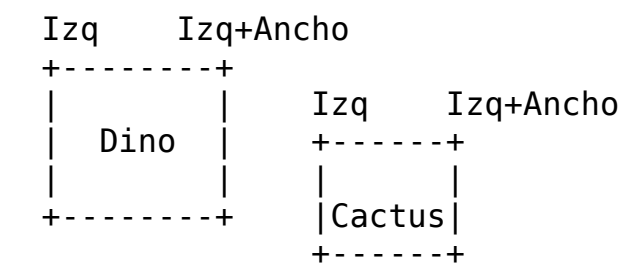
Si el cactus es visible, lo movemos a la izquierda, de la forma usual restando un paso:

```
ImgCactus.Left:=  ImgCactus.Left – paso;
```

Hay que ver si el dinosaurio tiene colisión con él.

Para hacer simple el asunto, podemos asumir el cactus y el dinosaurio son cuadrados. Así por ejemplo si la izquierda del cactus está más atrás de la derecha de dinosaurio y la derecha del cactus es mayor que la izquierda del dinosaurio, es posible que haya contacto (nos falta verificar la vertical).

Sentido horizontal

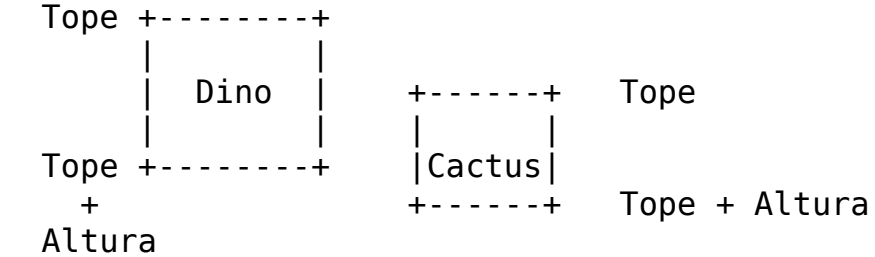



Si la izquierda del cactus es menor que la derecha (izquierda + ancho) del dinosaurio

y además la derecha del cactus (izquierda + ancho) es mayor que la izquierda del dinosaurio

ambas figuras se están cruzando horizontalmente

En el sentido vertical hay que recordar que el 0 es arriba y 100 más abajo. Así que tope + altura es más abajo que tope




Si el tope del cactus es menor que la parte de abajo del dinosaurio (tope + altura) hay colisión (no estamos considerando que el cactus salte al dinosaurio)

En la práctica izquierda + ancho de dinosaurio queda muy adelante del dinosaurio (quedaría bien si choca con la cara, pero es con la parte de abajo del cuerpo) asi que en lugar del ancho se usa ⅔ del

ancho.

```
        if (  ImgCactus.Left <=  (Img.Left + (2*Img.Width div 3)))
            AND
           ( (ImgCactus.Left + ImgCactus.Width)
              >=  (Img.Left ))
            AND
           ( (Img.Top + Img.Height) >= ImgCactus.Top)  then
             begin
               LblScore.Caption:= 'Choque';
               puntos := 0;
               Timer1.Interval:= 100;
               LblScore.Font.Color:= clRed;

               playsound1.Execute;

             end;
```

Como se ve, si hay choque ocurren 4 cosas:

En lugar de score se escribe “Choque”
Los punto acumulados bajan a 0
El intervalo de tiempo de ciclo vuelve a 100 milisegundos (en la versión que acelera cada tanto)
EL fuente de texto del Score hace hace rojo.
Se ejecuta un sonido (este es el componente de sonido que se comentó al inicio).

| NOTA |
| --- |
| El control de sonido. Requiere “expl.wav” como SoundFile, es decir el archivo de sonido. Esta con el proyecto en la descarga en la carpeta de fuentes. |

Luego de verifica si hay colisión se verifica si el cactus ya salió del alcance de la vista, esto es asi que el lado derecho e menor a 0 o el zquierdo menos al ancho con valor negativo.

```
  if ImgCactus.Left< -ImgCactus.Width then
     ImgCactus.Visible := false;
```

**Últimos toques**
**==========**

El dinosaurio salta si se presiona la barra espaciadora. El formulario dispara un evento cuando se presiona una tecla FormKeyPress. En esa rutina se confirma si la tecla presionada fue espacio y si es asi se define el salto en 7 (para que inicie su

```
conteo)

procedure TForm1.FormKeyPress(Sender: TObject; var Key: char);
begin
  if key = ' ' then
    Salto := 7;
end;
```

El formulario tiene un evento FormCreate que es ideal para inicializar valores. Por ejemplo Img debe ser Img2Pie y las imagenes y etiquetas deben ser colocadas en posición inicial.