Learning 2D Game Programming: Basic FrameWork Part 7 - Adding Explosion

Table of Contents

Introduction

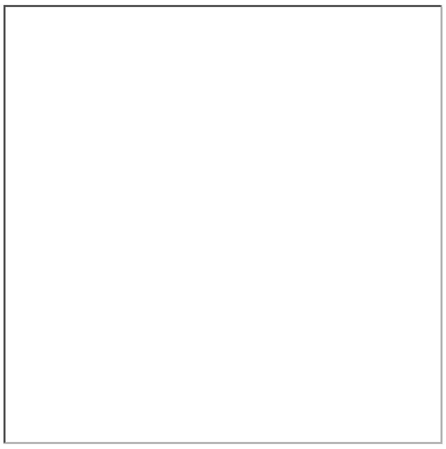
This tutorial shows you how to effect an explosion when your missiles collide with the AlienShip. I have also taken the liberty to re-arrange the code from the previous tutorial to make it more streamlined. Plus I have also added a new way to handle the collision.

Since the complete program is getting longer, I'm not presenting it in it's totality, just the changes. The code can be downloaded here.

Firing enough missiles into the AlienShip will result in the explosion:-

The Explosion Type

One of the new feature of our new TExplosion Type is that it uses a special type of image called an AnimImage (Animation Image) . The one we will be using looks like this:-



As you can see, the AnimImage contain a series of images which when we cycle through the frames (each 64x64 image is called a frame) will give the illusion of an explosion.

The AnimImage is loaded into our program using the following command (Let us not worry too much about the intricacies of this statement for the moment):-

```
Global
ExpImage:Timage=LoadAnimImage(LoadBank(URL+"exp1.png"),64,64,0,16)
```

As you will see below, the UpdateState method of TExplosion object is all about handling the animation. Once the Animation is done (at Frame 16), a new AlienShip is created and the explosion object removes itself from the GameObjectList:-

```
Type TExplosion Extends TGameObject
   Field AnimDelay:Int=10

   Function Create:TExplosion(Image:TImage,xstart:Int,ystart:Int)
      Local obj:TExplosion=New TExplosion
      CreateObject(Obj,Image,xstart,ystart,2)
      Return obj
   End Function

   Method UpdateState()

     If AnimDelay<0 Then
       Frame :+ 1
       AnimDelay=10
       If Frame>15 Then
         TAlienShip.Create(AlienImage,X,Y)
         ListRemove(GameObjectList,Self)
         Return
      EndIf
      EndIf

      AnimDelay :- 1

  End Method

End Type
```

-.
The TGameObject type has also been changed in this tutorial to accomodate the addition of the Frame as well as scale. The Scale is required as we want to increase the size of our explosion to match the size of the AlienShip.

```
Type TGameObject

   Field X:Int = 320
```

```
    Field Y:Int = 420
    Field Speed:Int=3
    Field Image:TImage
    Field Frame:Int=0
    Field XScale:Float=1.0
    Field YScale:Float=1.0

    Method DrawSelf()
        SetColor 255,255,255
        SetScale XScale,YScale
        DrawImage Image,X,Y,Frame
    End Method

     Method UpdateState() Abstract

End Type
```

Initiating the explosion

If you have not done so, you may want to visit the Collision tutorial to get a better understanding of the Collision detection technique that we are using below:-

This loop iterates through every TMissile object in our GameObjectList and writes the missile image into the Alien Collision Layer, ready for collision detection using the CollideImage function in write mode:-

```
    Method CheckCollision()

      For Local g:TMissile=EachIn GameObjectList
            CollideImage(g.Image,g.X,g.Y,0,0,AlienLayer,g)
      Next
```

This next block of code checks for collision using the CollideImage function in read mode. The missiles colliding with the AlienShip (could be more than 1) each causes some damage to the AlienShip. Each Colliding missile is also removed from the GameObjectList as it is destroyed upon contact.

```
      Local p:Object[]= CollideImage(Image,X,Y,0,AlienLayer,0)
      For Local k:TMissile=EachIn p
          Explosion :+ k.Damage
           ListRemove(GameObjectList,k)
      Next
```

We then check whether the damage done is great enough to cause our AlienShip to explode. If so, we remove the AlienShip from the GameObjectList and creates the explosion object which will go through the animation cycle. Once the animation cycle is completed, the explosion object destroys itself and creates a new AlienShip ready for us to start shooting at.

```
      If Explosion > 15
          ListRemove(GameObjectList,Self)
          TExplosion.Create(ExpImage,X,Y)
       EndIf
```

And lastly we resets our collision layer, ready for the next round of collision detection.

```
       ResetCollisions(AlienLayer)
```

Other Modifications done

I have also pre-loaded all the images that we are going to use to speed up the creation of explosions and alienships.

```
Global URL:String="http::www.2dgamecreators.com/tutorials
/gameprogramming/basic/"
Global
ExpImage:Timage=LoadAnimImage(LoadBank(URL+"exp1.png"),64,64,0,16)
Global
AlienImage:TImage=LoadImage(LoadBank(URL+"cartoonufo_1-1.png"))
Global
PlayerImage:TImage=LoadImage(LoadBank(URL+"blobship_1-1.png"))
Global MissileImage:TImage=LoadImage(BlitzMaxPath()+"\samples\firepaint
\bullet.png")
```

Lastly I have placced the CreateObject function outside of each type to streamline the program a bit.

```
Function CreateObject(Obj:TGameObject,
Image:TImage,xstart:Int,ystart:Int,scale:Float=1.0)

     Obj.X=xstart
     Obj.Y=ystart
     Obj.XScale=scale
     Obj.YScale=Scale
     Obj.Image=Image

    If Obj.Image=Null
       Print "Not able to load image file. Program aborting"
       End
    EndIf

     ListAddLast GameObjectList, Obj

End Function
```

The full source code can be downloaded from here.

Summary

There we have it, a very rudimentary "game" built using BlitzMax. It's not very exciting at the moment but the whole idea now is to start learning the necessary concepts of 2D programming where we can then use the framework we have built over the last few tutorials to design better, more complex and fun games.

Blitzmax commands introduced in this tutorial

Four new functions introduced in this tutorial:-

```
Function LoadAnimImage:TImage(
url:Object,cell_width,cell_height,first_cell,cell_count,flags=-1 )
```

Load a multi-frame image.

```
Function CollideImage:Object[]
(image:TImage,x,y,frame,collidemask%,writemask%,id:Object=Null)
```

Pixel accurate collision testing between transformed Images.

```
Function ResetCollisions(mask%=0)
```

Clears collision layers specified by the value of **mask**, mask=0 for all layers.

Function SetScale(scale_x#,scale_y#)

scale_x and **scale_y** multiply the width and height of drawing commands where 0.5 will half the size of the drawing and 2.0 is equivalent to doubling the size

Back to the index, Previous Tutorial