Learning 2D Game Programming: Max 2D Collision Part 2

Part 1, Part 2, Part 3

Table of Contents

Introduction

We saw in the previous tutorial how to check for collisions using our built function RectOverlaps and CircRectOverlaps. We also saw the more powerful (pixel perfect) built-in collision functions ImagesCollide and ImagesCollide2.

In this tutorial we are going to look at two othe built-in collision function which employs a different technique to the ones we have seen before.

CollideImage

But in setting the scene, let us introduce this collision function using the scaling problem we saw earlier and see how to do that using this technique. The syntax for CollideImage can cause a lot of confusion. I'm presenting it below but let's ignore it for a minute and see the changes we had to make to our previous program to solve the collision detection problem using the CollideImage function:-

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

Note the changes as highlighted (as per usual, the grey text are unchanged from the previous tutorial)

```
Graphics 640,480
Local URL:String="http::www.2dgamecreators.com/tutorials
/gameprogramming/basic/"
Local Player:TImage=LoadImage(LoadBank(URL+"blobship_1-1.png"))
Local
Alien:TImage=LoadAnimImage(LoadBank(URL+"exp1.png"),64,64,0,16)
Local Frame:Int=0
Local AnimDelay:int=10
Local PlayerSize:Int=1
Local AlienSize:Int=2

While Not KeyHit(key_escape) Or AppTerminate()
  Cls
  ResetCollisions()
  SetScale AlienSize,AlienSize
  DrawImage Alien,150,100,Frame
  CollideImage(Alien,150,100,Frame,0,2)

  SetScale PlayerSize,PlayerSize
  DrawImage Player,MouseX(),MouseY()
  If CollideImage(Player,MouseX(),MouseY(),0,2,0)
      SetClsColor 255,0,0
  Else
      SetClsColor 0,0,0
  EndIf
  Flip
  If AnimDelay<0 Then
     Frame :+ 1
     If Frame>15 Then Frame=0
     AnimDelay=10
  EndIf
  AnimDelay :- 1

Wend
End
```

Running the above code, you will see that the collision behaves exactly the same as per the previous tutorial.

Collision Layer

This new BlitzMax collision system introduces the concept of collision layers. The steps to use the collision detection mechanism would be as follows:-

- first ensure that the collision layer is cleared
- write to the collision layer the image you want to check collision against
- compare the image that you want to check collision against the image which is in the collision layer

There are actually 32 collision layers that we can use. In the example above we are using collision layer #1.
In the program above, the 3 steps are accomplished as follows:-

ResetCollisions() clears all the Collision Layer. We could have use ResetCollision(2) as well to just clear layer #2

```
ResetCollisions()
```

This first CollideImage statement; writes the Alien Image onto Collision Layer 2

```
CollideImage(Alien,150,100,Frame,0,2)
```

This CollideImage statement reads from Collision Layer 2 and checks collision between the Player and Alien Images

```
If CollideImage(Player,MouseX(),MouseY(),0,2,0)
```

This is what Mark Sibly, the creator of BlitzMax has to say about layers:-

The basic idea is that the collidemask and writemask parameters are bitmasks (eg: 1,2,4,8,16 etc) giving you up to 32 'layers' to write/collide to/with. For example, you could use the following layers:

background=0, monsters=1, pickups=2, player=3.

The bitmasks for these would be 1,2,4,8 respectively (bitmask=1 Shl layer).

Using a bitmask instead of just a layer number allows you to specify multiple layers, eg: bitmask 3 would mean both layer 0 and layer 1.

CollideImage simultaneously writes an image to 0-32 layers (depending on the writemask) and checks for collisions with 0-32 layers (dependingon the collidemask).

By 'writes an image', all it really does internally is store the image, and current rot/scale etc. However, you can think of it as physically 'writing' the image.

What Mark is talking about is the collidemask% and writemask% parameter

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

To write an image to Collision Layer 1, we should provide the **writemask** parameter as follows:-

```
CollideImage(Alien,150,100,Frame,0,1)
```

We can also write to both Collision Layer 1 and 2 as follows:-

```
CollideImage(Alien,150,100,Frame,0,3)
```

Similar, to read (check for a collision on layer 1) we need to provide the collidemask% value:-

```
If CollideImage(Player,MouseX(),MouseY(),0,1,0)
```

And similarly check for collision on 2 layers at the same time we use:-

```
If CollideImage(Player,MouseX(),MouseY(),0,3,0)
```

The beauty of Collision layers is that you can then check collision only againts the layers that matters. Before we look at a simple example lets take a look at another concept first:-

Collision returning Object Arrays

So far what we have been using to check for collision is the if <Collision Function> returns a true then a collision has happened type statement.

The BlitzMax CollideImage function actually has a much more powerful way of dealing with collision checking. We are now going to solve the mystery of the **Object[]** and **id:Object** part of the CollideImage syntax.

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

Remember that in the above introduction on CollideImage we said that there were 2 version of CollideImage,

- the write version with writemask
- and the read version with collidemask

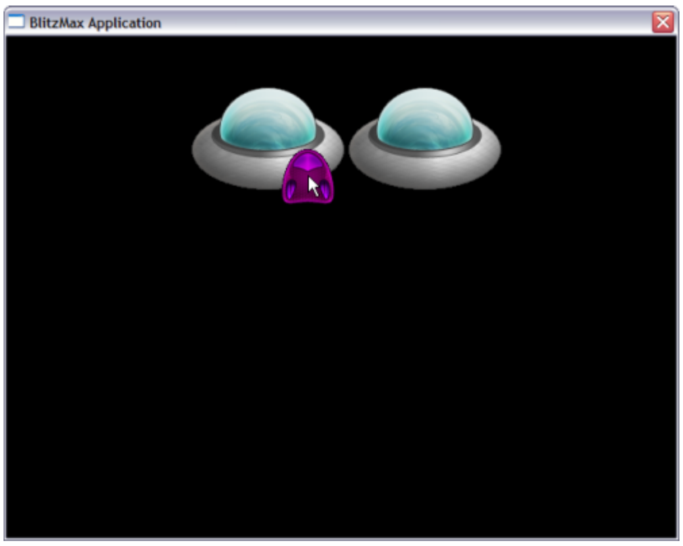
When we are writing to the collision layer, the id:Object parameters allows us to identify the image we are writing to the Collision layer.

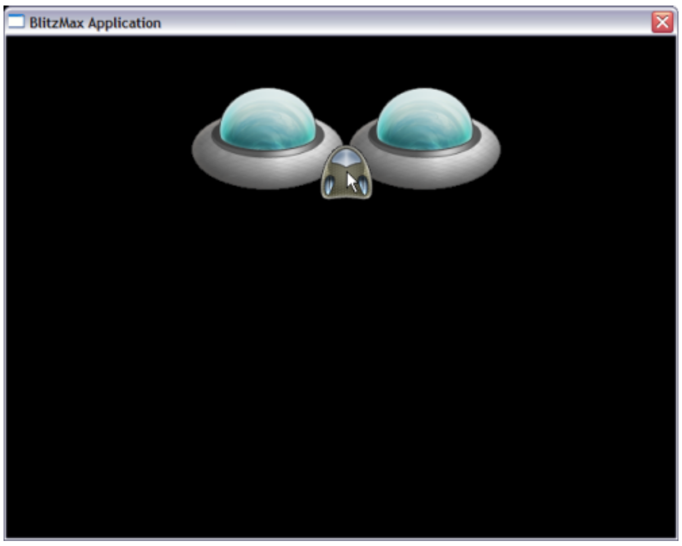
Once the identities have been written, when we read from the collision layer, the objects involved in the collision will be returned in the Object array.

Let us illustrate this with a very simple example:-

```
Strict
Graphics 640,480
AutoMidHandle True

Local URL:String="http::www.2dgamecreators.com/tutorials
/gameprogramming/basic/"
Local SpaceShip:TImage=LoadImage(LoadBank(URL+"blobship_1-1.png"))
Local AlienShip1:TImage=LoadImage(LoadBank(URL+"cartoonufo_1-1.png"))
Local AlienShip2:TImage=LoadImage(LoadBank(URL+"cartoonufo_1-1.png"))

Repeat
 Cls
 Local R:Int=0
 Local G:Int=0
 Local B:Int=255
 ResetCollisions()
 SetColor 255,255,255
 DrawImage AlienShip1, 250,100
 CollideImage(AlienShip1,250,100,0,0,1,AlienShip1)

 DrawImage AlienShip2, 400,100
 CollideImage(AlienShip2,400,100,0,0,1,AlienShip2)
 Local p:Object[]=CollideImage(Spaceship,MouseX(),MouseY(),0,1,0)
 For Local i:TImage=EachIn p
    Select i
    Case AlienShip1
        R=255
    Case AlienShip2
        G=255
    End Select
 Next
 SetColor R,G,B
 DrawImage SpaceShip,MouseX(),MouseY()
 Flip

Until KeyDown(KEY_ESCAPE) Or AppTerminate()
```

Note that the color changes depending on the collision state. We can also detect collisons with both ships with just a single **CollideImage** statement as all collided objects are returned into the Object Array





If you were to not write the AlienShip1 object to the last CollideImage parameter, you cannot detect collision by its Object Name. Try it out and see

```
 DrawImage AlienShip1, 250,100
 CollideImage(AlienShip1,250,100,0,0,1,Null)
```

Note that if you were to also write the SpaceShip image to the collision layer, the SpaceShip object will also be returned in the Object Array. So if you are checking for collision you will need to take care of this, otherwise you will think a collision has occured whereas actually you are just colliding with yourself! In my examples so far, I did not write the player image to the collision layer so this is not a problem.

Collision Layer Example

Below (in highlight) is an example with detecting collisions at different layers

```
Strict
Graphics 640,480
AutoMidHandle True

Local URL:String="http::www.2dgamecreators.com/tutorials
/gameprogramming/basic/"
Local SpaceShip:TImage=LoadImage(LoadBank(URL+"blobship_1-1.png"))
Local AlienShip1:TImage=LoadImage(LoadBank(URL+"cartoonufo_1-1.png"))
Local AlienShip2:TImage=LoadImage(LoadBank(URL+"cartoonufo_1-1.png"))

Repeat
 Cls
 Local R:Int=0
 Local G:Int=0
 Local B:Int=255
 ResetCollisions(3)
 SetColor 255,255,255
 DrawImage AlienShip1, 250,100
 CollideImage(AlienShip1,250,100,0,0,1,AlienShip1)

 DrawImage AlienShip2, 400,100
 CollideImage(AlienShip2,400,100,0,0,2,AlienShip2)
 Local Collision_Layer:int=2
 Local
p:Object[]=CollideImage(Spaceship,MouseX(),MouseY(),0,Collision_Layer,0)
 For Local i:TImage=EachIn p
    Select i
    Case AlienShip1
        R=255
    Case AlienShip2
        G=255
    End Select
 Next
 SetColor R,G,B
 DrawImage SpaceShip,MouseX(),MouseY()
 Flip

Until KeyDown(KEY_ESCAPE) Or AppTerminate()
```

If you run the above code, you will see that our spaceship will only get a positive collision when the same Collision Layer is read. To detect ship 1, use Collision_Layer:int=1. To detect both ships, change the Collision_Layer variable to 3.

Summary

The objective of this second part was to show you how to use the more powerful CollideImage collision function compared with the more traditional image by image collision method like the one shown in part 1 of the tutorial.

The thing to remember is that the collision comparison is not done on the drawing canvas but on a separate collision layer therefore allowing for a more selective (and faster) collision checking. BlitzMax has 32 Collision layers that we can use.

So to reiterate

- first clear the collision layer by using the ResetCollisions(layer#) function.
- write to the collision layer(s) the images we want to check for collision
- read from the collision layer(s) to perform the actual collision check

** Don;t forget that the layers are maskable.

Back to the index, Previous Tutorial, Next Tutorial