How to write a BreakOut Game: Part 4: The TBrick Object

Table of Contents

The TBrick Object

The final object that we need to get our game going is the TBrick object. This object is a passive object and all it does is to allow itself to be lined up looking pretty and then gets destroyed when the ball hits it.

The declaration of the the TBrick Object looks like this:-

```
Type TBricks Extends TGameObject

    Function Create:TBricks(Image:TImage,xstart:Int,ystart:Int)
        Local B:TBricks=New TBricks
        CreateObject(B,Image,xstart,ystart)
        Return B
    End Function

    Method UpdateSelf()
        'do nothing
    End Method

EndType
```

Notice that we have to create an UpdateSelf method that does nothing otherwise it will break our framework as the framework expects every object stored in the GameObjectList to have an updateSelf method.

The way we declare the TGameObject with an UpdateSelf Abstract method enforces this requirement.

We also need a helper function **CreateBricks** to create the bricks we require on to the screen. 16 columns by 6 rows of bricks.

```
Function CreateBricks()

   For Local x:Int=0 To 15
    For Local y:Int=0 To 5
     TBricks.Create(Bricks,50+x*36,50+y*22)
    Next
   Next

End Function
```

Since the image that we are using for the brick is an Animation Image we need to use the LoadAnimImage function instead of the LoadImage function. Apologies if this is confusing but it also illustrates the fact that we can use an animated image also as a static image :)

```
Global Bricks:TImage=LoadAnimImage(URL+"tiles.png",32,20,0,5)
CreateBricks()
```

Download the full source here. You should see the following without any collision :-

Adding Collision detection between Brick and Ball

We are going to use a slightly different technique for checking collisions between the Ball and the bricks (cf with collison between Ball and Paddle). We are going to use the **ImagesCollide2** collision function as it allow us to get pixel perfect collision between objects which are rotating and scaled like our Ball.

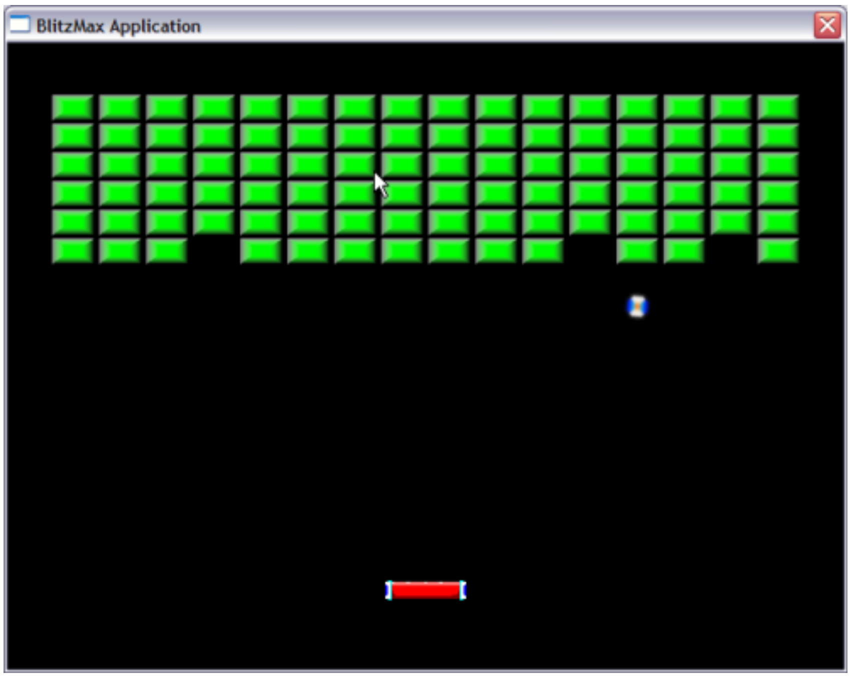
The process again is very straightforward. We loop through every brick currently in the GameObjectList and check whether it is colliding with our ball. If yes, we remove the brick from the GameObjectList and at the same time bounce the ball away from the bricks.

To add the collision detection, we only need to modify the CheckCollision function within the TBall Type Declaration and nothing else, again, demonstrating the power of Object Oriented Programming:-

```
Method CheckCollision()

   SetScale XScale, YScale
   SetRotation Rotation
   If CollideImage(Image,X,Y,0,PLAYER_LAYER,0)
     YSpeed=-YSpeed
   EndIf

   For Local b:TBricks=EachIn GameObjectList
      If ImagesCollide2(Image,X,Y,0,Rotation,XScale,YScale, b.Image, b.X, b.Y, 0, 0,
1.0, 1.0)
         ListRemove(GameObjectList,b)
         YSpeed=-YSpeed
         Exit
      EndIf
   Next

End Method
```

Having done all those, viola, we have a collision between Bricks and Ball. In fact the program now becomes a very rudimentary BreakOut game!

Dowload the source here and build and run the resulting program.



Summary

As we can see from our previous three tutorials, adding a game object into our framework was fairly straight forward. In each tutorial , all we needed to do was define the object, program in the required behaviour in the UpdateSelf method, place the necessary collision detection and subsequent behaviour and then we are done.

The next section of this tutorial series will introduce the concept of Game States. To do this we will be enhancing our framework. This we will do over our next two tutorial.

Goto Next Tutorial or Return to Index.