

Torque Game Builder – Fish Game Tutorial - Part 2

2. Moving our Fish

2.1 Add our fish

Before we can move a fish, we must first have a fish. Fortunately, this is a simple step. Click and drag the fish image from the *Static Sprites* library (as shown in Figure 2.1.1) into the center of our level (as shown in Figure 2.1.2).

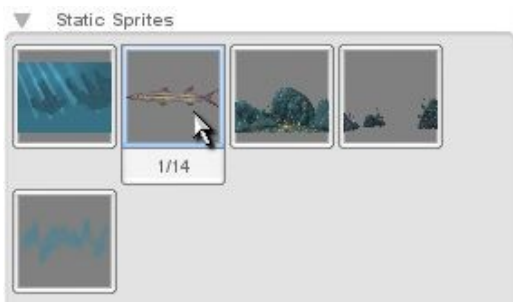


Figure 2.1.1

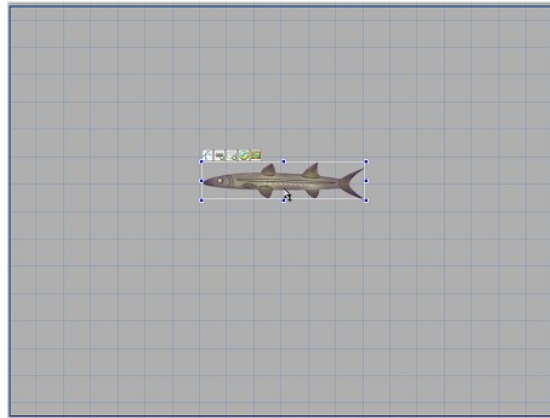


Figure 2.1.2



Figure 2.1.3

Now that we have our fish in our level, our next step will be to get it moving. This requires some scripts, so we are going to go over how to easily link up scripting. We can do this, by giving our object a *class* in the *Level Builder*. Select the fish if you need to, (it should be automatically selected after adding it). Now, click the *Edit* tab (as shown in Figure 2.1.3). This is where we can change the settings of our fish. We want to click the *Scripting* option, to expand the panel (as shown in Figure 2.1.4). You should see the *Class* field (as shown in Figure 2.1.5).

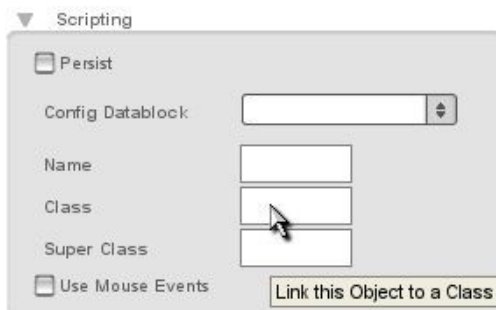


Figure 2.1.5

We can enter a name for whatever class we want our fish object to be. This name is what we will reference in our scripts. Type in "PlayerFish" (as shown in Figure 2.1.6), and press *Enter* to make sure it applies the class to the object. This is all the work we can do in the *Level Builder*. The rest needs to be done in script. Because of this ability to link a class in the *Level Builder*, this next step is very easy.



Figure 2.1.6

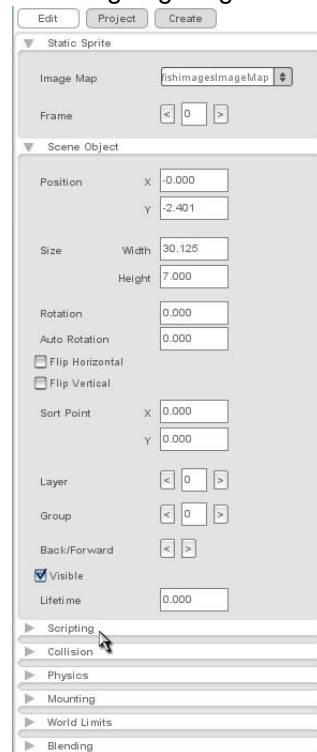


Figure 2.1.4

Torque Game Builder – Fish Game Tutorial - Part 2

2.2 Saving out our level

Before we exit, we want to make sure we save our level properly. That way when we load it up, our fish is already placed with its class already set. To do this, we need to go to the *File* menu and then click the *Save as...* option (as shown in Figure 2.2.1). Then browse out to the *MyFishGame/data/levels* folder so we can save our level in the proper place (as shown in Figure 2.2.2). Type in the name “myFishlevel” and then click the *Save File* button.



Figure 2.2.1

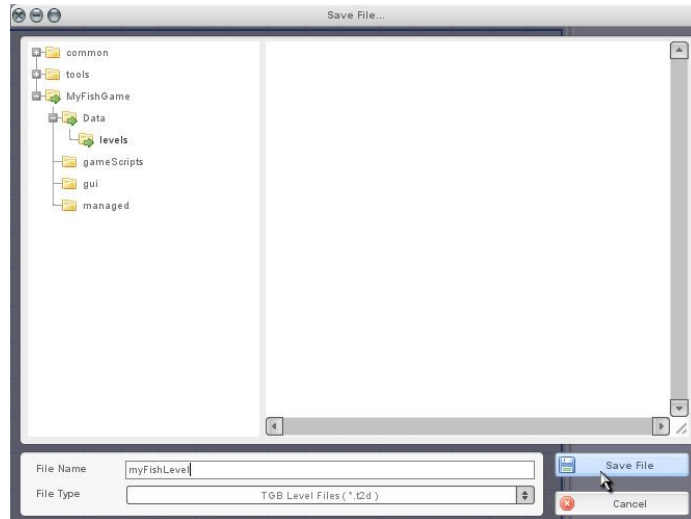


Figure 2.2.2

2.3 Creating our movement script file

Now we can exit the *Level Builder* (by using *File->Quit*), and begin our scripting. Browse out to your *games/MyFishGame/gameScripts* folder. This is where your game scripts should be located. Presently, you should only see a *game.cs* file in this location. (.cs is the *Torque Script* file extension). In this folder we can create additional script files to build up our game; we will begin with movement. Create a text file in this folder and name it “player.cs”. Be sure you add the .cs extension so *TGB* will recognize it as a script file. Now you can open your newly created *player.cs* in any text editing program. (Windows users can right-click and choose *Open With* to use either Notepad or Wordpad. Mac users can control-click and choose *Open With* to use TextEdit). If you remember, we set our fish's class to *PlayerFish*. So we are going to add the following function to our *player.cs*.

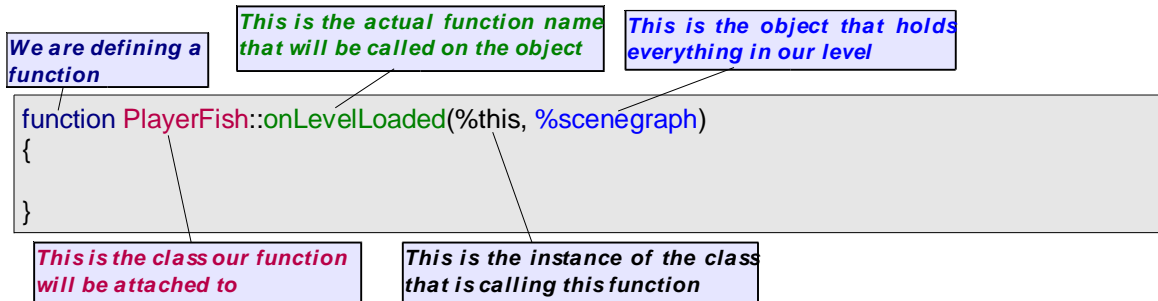
```
function PlayerFish::onLevelLoaded(%this, %scenegraph)
{
}
}
```

Code Sample 2.3.1

As you may notice, we start with the keyword *function*, which tells *TGB* that we are beginning a function declaration. Then we follow with our class name *PlayerFish*. This attaches our function to the *PlayerFish* class. Since our fish is using the *PlayerFish* class, our fish will now have access to this function. Then we get to the actual function name (*onLevelLoaded*), which you might have guessed gets called when our fish gets loaded into the level. After the function name, we have two comma separated values inside of parenthesis. These are values that will be

Torque Game Builder – Fish Game Tutorial - Part 2

passed to this function, and which could be useful. The `%this` value represents the object that this function is being called on. That value is useful when we have multiple objects using the same class. It represents the specific *instance* of the class calling this function. The `%scenegraph` value is useful as well, since it represents our level object. Everything in our level is inside of the *scenegraph* object. Here is a breakdown of what our function script means.



Now it's time to do something inside of the script. Since it gets called when our fish is loaded into our level, we can store this instance of our fish to be used in our key responses. Add this line in between the curly braces (`{ }`).

```
$FishPlayer = %this;
```

Code Sample 2.3.2

What this does is grab the instance we are loading and store it in the `$FishPlayer` variable. In *Torque Script* a "\$" before text means it is a **global variable**, and a "%" before text means it is a **local variable**. A local variable (like `%this`) will only exist within the calling function. Once the function ends, the variable gets destroyed. Think of it as a temporary variable. On the other hand, a global variable will persist outside of the function, and can be accessed in any script or anywhere in the game.

Now that we have stored the fish object, which will represent our player, we need the commands to *bind* our keys for movement. We will use the `w`, `s`, `a`, and `d` keys, so we need to make four separate script lines. Add these lines after our "`$FishPlayer = %this;`" line.

```
moveMap.bindCmd(keyboard, "w", "fishPlayerUp();", "fishPlayerUpStop();");
moveMap.bindCmd(keyboard, "s", "fishPlayerDown();", "fishPlayerDownStop();");
moveMap.bindCmd(keyboard, "a", "fishPlayerLeft();", "fishPlayerLeftStop();");
moveMap.bindCmd(keyboard, "d", "fishPlayerRight();", "fishPlayerRightStop();");

```

Code Sample 2.3.3

There is already an object called `moveMap`, and this object is what handles our key events when running our level. We use a function called `bindCmd` on our `moveMap`, and we pass it a few values. First, we specify that we are binding a command to our keyboard. We then specify which key we are binding. After that, we specify what function to call when the key is pressed, and our final value is what function to call when the key is released. So, for example, we bind the keyboard key `w` and when we press it `fishPlayerUp()` is called. Then, when we release the key, `fishPlayerUpStop()` is called to stop our action.

Now when our fish gets loaded, it will be stored in `$FishPlayer` and the `w`, `s`, `a`, and `d` keys will get bound. Next we need to create the functions that are called when we press the keys.

Torque Game Builder – Fish Game Tutorial - Part 2

2.4 Creating our movement functions

We are now finished putting script lines inside of our *onLevelLoaded()* function, but we still need some others. As we create these new functions, be sure not to put one function inside another one accidentally. The following functions should be placed after the ending curly brace (}) of our *onLevelLoaded* function.

```
function fishPlayerUp()
{
    $FishPlayer.setLinearVelocityY( -15 );
}
```

Code Sample 2.4.1

Here is our *fishPlayerUp()* function. As you can see, we reference the fish by the stored value in *\$FishPlayer* and call *setLinearVelocityY()* on it. This function will basically set its Y direction speed. Since we are setting it negative, it will go upward (since numbers on the Y axis get lower as you go higher up). We now need to add the next three functions for movement.

```
function fishPlayerDown()
{
    $FishPlayer.setLinearVelocityY( 15 );
}

function fishPlayerLeft()
{
    $FishPlayer.setLinearVelocityX( -30 );
}

function fishPlayerRight()
{
    $FishPlayer.setLinearVelocityX( 30 );
}
```

Code Sample 2.4.2

These functions basically do the same thing. The down function sets the Y velocity as a positive value. The left and right functions set the X velocity instead of the Y velocity, to move horizontally. We are nearly done. We now need to add the stop functions. The stop functions are very similar to our movement functions, they simply set the appropriate velocity to zero. So, add these four functions.

Torque Game Builder – Fish Game Tutorial - Part 2

```
function fishPlayerUpStop()
{
    $FishPlayer.setLinearVelocityY( 0 );
}

function fishPlayerDownStop()
{
    $FishPlayer.setLinearVelocityY( 0 );
}

function fishPlayerLeftStop()
{
    $FishPlayer.setLinearVelocityX( 0 );
}

function fishPlayerRightStop()
{
    $FishPlayer.setLinearVelocityX( 0 );
}
```

Code Sample 2.4.3

As you can see, the up and down functions set the Y velocity to zero and the left and right functions set the X velocity to zero. This will cause our fish to stop when the appropriate key is released.

Now your *player.cs* script file should look like this.

```
function PlayerFish::onLevelLoaded(%this, %scenegrph)
{
    $FishPlayer = %this;

    moveMap.bindCmd(keyboard, "w", "fishPlayerUp();", "fishPlayerUpStop();");
    moveMap.bindCmd(keyboard, "s", "fishPlayerDown();", "fishPlayerDownStop();");
    moveMap.bindCmd(keyboard, "a", "fishPlayerLeft();", "fishPlayerLeftStop();");
    moveMap.bindCmd(keyboard, "d", "fishPlayerRight();", "fishPlayerRightStop();");
}

function fishPlayerUp()
{
    $FishPlayer.setLinearVelocityY( -15 );
}

function fishPlayerDown()
{
    $FishPlayer.setLinearVelocityY( 15 );
}

function fishPlayerLeft()
{
    $FishPlayer.setLinearVelocityX( -30 );
}
```

(code sample continued on next page)

Torque Game Builder – Fish Game Tutorial - Part 2

```
function fishPlayerRight()
{
    $FishPlayer.setLinearVelocityX( 30 );
}

function fishPlayerUpStop()
{
    $FishPlayer.setLinearVelocityY( 0 );
}

function fishPlayerDownStop()
{
    $FishPlayer.setLinearVelocityY( 0 );
}

function fishPlayerLeftStop()
{
    $FishPlayer.setLinearVelocityX( 0 );
}

function fishPlayerRightStop()
{
    $FishPlayer.setLinearVelocityX( 0 );
}
```

Code Sample 2.4.4

2.5 Almost ready to test

Now we are almost ready to test. We have one final step before we can move our fish around. The *player.cs* script file has all of the appropriate functions, but it isn't being referenced anywhere right now. Without referencing it, the program won't know to use the code inside it. *TGB* doesn't automatically reference every *.cs* file in your folders. This gives you complete control and you can choose whether or not you want a script file referenced. When *TGB* references a script file it will "exec" (short for execute) the file. So we need to add a proper *exec* command to reference the *player.cs* file we created. Open the *game.cs* file, which should be located in the same folder as *player.cs* (in your *gameScripts* folder).

In there you should find a function called *startGame* which starts like this.

```
function startGame(%level)
{
    // Set The GUI.
    Canvas.setContent(mainScreenGui);
    Canvas.setCursor(DefaultCursor);
```

Code Sample 2.5.1

We are going to add this line right after the curly brace (*}*).

```
exec("./player.cs");
```

Code Sample 2.5.2

So your modified *game.cs* file should look like this.

Torque Game Builder – Fish Game Tutorial - Part 2

```
//-----  
// Torque 2D.  
// Copyright (C) GarageGames.com, Inc.  
//-----  
  
//-----  
// startGame  
// All game logic should be set up here. This will be called by the level builder when you  
// select "Run Game" or by the startup process of your game to load the first level.  
//-----  
function startGame(%level)  
{  
    exec("./player.cs");  
  
    // Set The GUI.  
    Canvas.setContent(mainScreenGui);  
    Canvas.setCursor(DefaultCursor);  
  
    moveMap.push();  
    if( isFile( %level ) || isFile( %level @ ".dso" ) )  
        sceneWindow2D.loadLevel(%level);  
}  
  
//-----  
// endGame  
// Game cleanup should be done here.  
//-----  
function endGame()  
{  
    sceneWindow2D.endLevel();  
    moveMap.pop();  
}
```

Code Sample 2.5.3

This *startGame()* function gets called when we test our level in the *TGB Level Builder*. As you can see, we now reference the *player.cs* file so when *TGB* runs it will include it.

Torque Game Builder – Fish Game Tutorial - Part 2

2.6 Time to test it!

Now before we run *TGB* again we need to be sure we save our changes to these files. Then, after you save, fire up *TGB*. You should see the level we saved with the fish sitting in the middle (as shown in *Figure 2.6.1*).

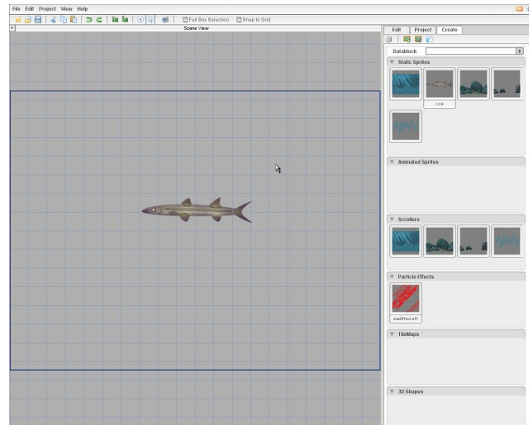


Figure 2.6.1

Now all we have to do is press the *Play Level* button in the top button bar (as shown in *Figure 2.6.2*).



Figure 2.6.2

You should see all of the tool bars and *Level Builder* menus disappear, leaving only our fish. Now, try pressing our movement keys (*a*, *s*, *w*, and *d*). You should see the fish being moved in the appropriate direction! (As shown in *Figure 2.6.3*.)

Congratulations! You have your fish moving around based on your keys. You should be getting a beginning feel of *TGB* scripting. When you're ready, move on to Part 3 and our Fish game takes another step.



Figure 2.6.3