

Torque Game Builder – Fish Game Tutorial - Part 4

4. Adjusting Movement

4.1 Making our fish face the proper direction

One of the initial problems you have probably noticed, is that when we move our fish left and right it still faces the same direction. Fixing this is quite simple, and will require only a couple of lines of script.

Browse out to your *MyFishGame/gameScripts* folder to find the *player.cs* script we created in part 2. Open *player.cs* in any text editor (Notepad or TextEdit works just fine). We want to alter the facing of the fish object when we move left and right. Start by locating the *fishPlayerLeft()* function. It should look like this.

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

Code Sample 4.1.1

To handle issues like this, *TGB* has a nice set of methods called *setFlip*. These methods flip the image in either the X and/or Y direction by passing it either a “true” or “false” value. If you pass it false, then it disables the flip and restores the image to it's default direction. If you pass it true, then it flips the image in whichever direction you specify (by calling *setFlipX(true/false)* or *setFlipY(true/false)*). When we move left we want the fish to be facing its default direction, so we can call the flip function with “false” to ensure it is flipped back to its default direction. Replace the *fishPlayerLeft()* function with this.

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

Code Sample 4.1.2

As you can see, we set its *flipX* to “false”, so that it will face the default direction of left when we move left. This means that we need to add a *setFlipX(true)* to the function called when we move right. So replace your *fishPlayerRight()* function with this.

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

Code Sample 4.1.3

Torque Game Builder – Fish Game Tutorial - Part 4

Now, when we move left or right our fish should be facing the appropriate direction. Be sure to save your script file, then fire up *TGB* and play the level to test it. Press the *d* key to move right, and our fish should now face the proper direction (as shown in *Figure 4.1.1*). Press *a* to move left, and our fish should be flipped back to its default direction, facing left.

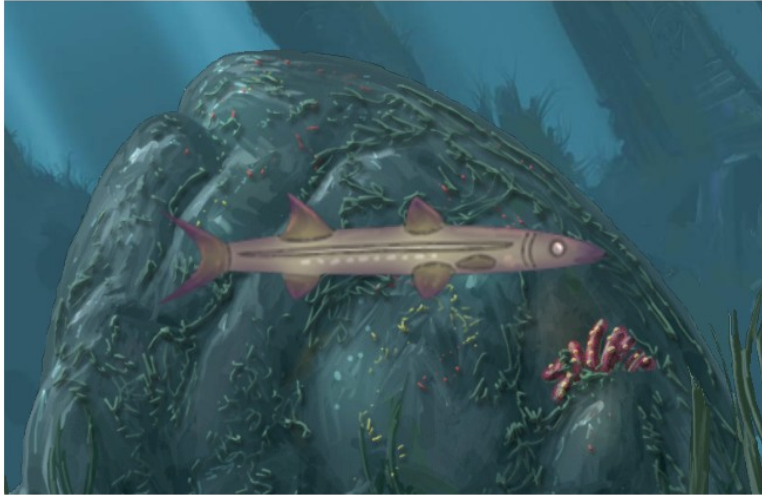


Figure 4.1.1

4.2 Cleaning up movement integration

Now is a good time to clean up some of our movement integration since we are already working on movement script. The main thing that will make our game easier to tweak, is making our fish's speed configurable in the *Level Builder*. There are two steps in this process. First, we must replace all of our calls that set the velocity. Instead of setting a hard coded number, we need to reference a value stored on the fish object. Second, we need to set the speed on the object in the *Level Builder*.

If you look at our four move functions, you will see that when moving horizontally we set the speed to either 30 or -30, and when moving vertically we set the speed to half that at 15 and -15. So instead of calling the velocity functions with those values, let's create two new values that reside on the object. Start by replacing your *fishPlayerLeft()* function with this.

```
function fishPlayerLeft()
{
    $FishPlayer.setFlipX(false);
    $FishPlayer.setLinearVelocityX( -$FishPlayer.hSpeed );
}
```

Code Sample 4.2.1

As you can see, we replaced the hard coded call that set the velocity to a "30" speed value, with the reference "\$FishPlayer.hSpeed". This is a reference to the horizontal speed we will store on the fish object.

Repeat this step to replace your right, up, and down movement functions with the following.

Torque Game Builder – Fish Game Tutorial - Part 4

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

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

function fishPlayerRight()
{
    $FishPlayer.setFlipX(true);
    $FishPlayer.setLinearVelocityX( $FishPlayer.hSpeed );
}
```

Code Sample 4.2.2

As you can see, we reference a *hSpeed* and *vSpeed* value on the fish object itself. Note that we still keep the negative sign to invert the speed for moving in the opposite directions.

Our final step is to create the *hSpeed* and *vSpeed* variables, and to set their values in the Level Builder. This is extremely easy and will only take a few clicks. Open up your *TGB*, and you should be presented with your level. Now select your fish and hit the *Edit* tab. You should now see properties of your fish (as shown in Figure 4.2.1).

We want to click the last group of options called *Dynamic Fields* (you may need to scroll down to find it). Once clicked, it should expand and show you a few text fields (as shown in Figure 4.2.2).

The *Field Name* is the name of the value we want to set, and the *Field Value* is the actual value we want to store on the field. What this does is allow us to store values on our object. This is how we can add an *hSpeed* and *vSpeed* value. First, enter “hSpeed” into the *Field Name* section, then enter “30” into the *Field Value* section (as shown in Figure 4.2.3).

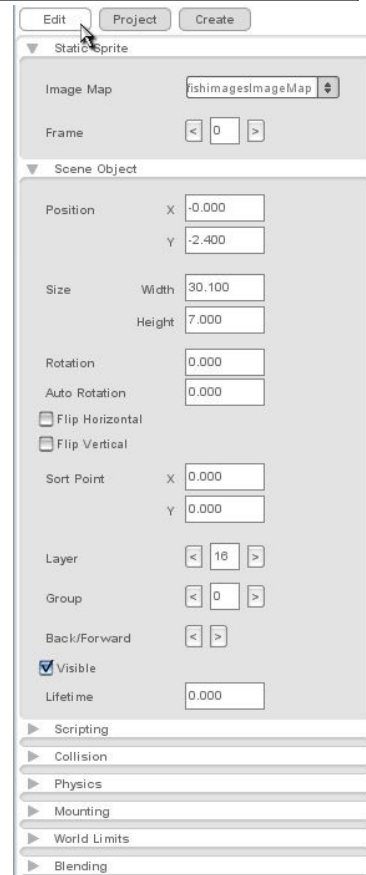


Figure 4.2.1



Figure 4.2.2



Figure 4.2.3

Torque Game Builder – Fish Game Tutorial - Part 4

Then click the little green plus sign button to confirm that you want to add it. You should now see “hSpeed” as a non editable text, and the value as editable with a little red minus sign instead of the plus sign (as shown in Figure 4.2.4). Now lets add the *vSpeed*. So type “vSpeed” into the *Field Name* and “15” into the *Field Value*. Then click the little green plus sign, and the *vSpeed* should appear just like the *hSpeed* (as shown in Figure 4.2.5). Now save and play your level, and your fish should move the same speed it did before.

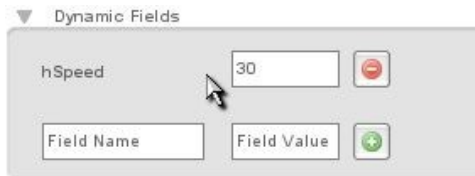


Figure 4.2.4

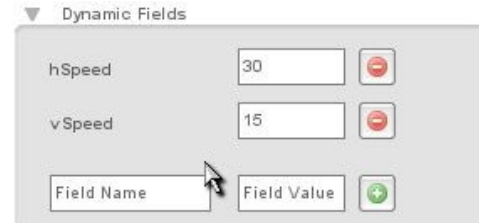


Figure 4.2.5

4.3 Adding a movement boost

Now that we have our fish moving properly, as well as moving off of values set on the fish object let's add a movement boost to our fish. When we press a certain key, let's increase our fish's speed three-fold. To do this, we are going to repeat the same process of adding our movement keys. We need to add a *bind* script line to bind our key to the boost function. Then we need to add the function that gets called when we press and release the key, which will toggle the speed on and off between three times normal and the normal amount.

Browse to your *MyFishGame/gameScripts* folder again and open up your *player.cs* script file. Right now the first function in this file (the *onLevelLoaded()* function) 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();");
}
```

Code Sample 4.3.1

We need to add another *moveMap.bindCmd()* function call to the end. Add this line of script after the last *moveMap.bindCmd()* call and before the ending curly brace (}).

```
moveMap.bindCmd(keyboard, "space", "fishPlayerBoost();", "fishPlayerBoostStop();");
```

Code Sample 4.3.2

This is set up the same way the previous binds were set up. We are binding the keyboard “space” key (space bar) to call *fishPlayerBoost()* when pressed, and *fishPlayerBoostStop()* when released. Now, let's create the functions called *fishPlayerBoost()*, and *fishPlayerBoostStop()*. First, add the boost function to the end of the file, as shown here.

Torque Game Builder – Fish Game Tutorial - Part 4

```
function fishPlayerBoost()
{
    %flipX = $FishPlayer.getFlipX();

    if(%flipX)
    {
        %hSpeed = $FishPlayer.hSpeed * 3;
    } else
    {
        %hSpeed = -$FishPlayer.hSpeed * 3;
    }

    $FishPlayer.setLinearVelocityX(%hSpeed);
}
```

Code Sample 4.3.3

We do a couple of things in this function. First, we grab the *flipX* of our fish player. This is how we determine which direction the player is facing. If its facing right, then the *flipX* will be true. If it is facing left, then the *flipX* will be false. If we are facing right, we store an *%hSpeed* (remember that the % means this is a local variable and will be destroyed at the end of the function) of three times the value. If *%flipX* is false and we are facing left, then we store a speed at negative three times the value. Then we set the x velocity according to the value. This way when you press down on the space bar, your fish should zip at three times the speed in the direction your facing. Next we have our stop function.

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

Code Sample 4.3.4

All this does is set the fish's linear velocity x to zero. That way when we release the boost key, the velocity of the fish is canceled and it will stop in its tracks. This gives you a balance to the extra speed the boost gives; whenever you release it you will be halted.

Now fire up *TGB* and play your level. Move around a little and then try pressing the space bar. Your fish should speed off with a quick boost (try not to run him too far off the screen). Then let go and your fish will stop! We now have a boost feature in our game.

4.4 Fixing our move key functions

You may notice a couple of more issues. One is that the fish can go as far off screen as it wants. This could potentially be very bad. There is also a bit of quirkiness with pressing the movement keys. If you hold down *a* to move left, then also hold down *d* to move right, and let go of even one of the keys, all movement stops. Even if you are still holding down the other key. We can fix this by slightly restructuring the way our keys work. Right now the response makes sense, since when we let go of a movement key it zeros out the velocity in the appropriate direction, whether or not we are holding down another movement key.

The first step is to make a single update function for our *PlayerFish*. This function will check four different values - one for each move direction - and if a single one is true, it will set the

Torque Game Builder – Fish Game Tutorial - Part 4

appropriate velocity. This way we only need one check to see if either move left or move right is true, and if not, to then zero out the velocity. This will stop it from zeroing out at every key release. Add this function to your *player.cs* after your first (*onLevelLoaded()*) function.

```
function PlayerFish::updateMovement(%this)
{
    if(%this.moveLeft)
    {
        $FishPlayer.setFlipX(false);
        $FishPlayer.setLinearVelocityX( -$FishPlayer.hSpeed );
    }
}
```

(code continued on next page)

```
    if(%this.moveRight)
    {
        $FishPlayer.setFlipX(true);
        $FishPlayer.setLinearVelocityX( $FishPlayer.hSpeed );
    }

    if(%this.moveUp)
    {
        %this.setLinearVelocityY( -$FishPlayer.vSpeed );
    }

    if(%this.moveDown)
    {
        %this.setLinearVelocityY( $FishPlayer.vSpeed );
    }

    if(!%this.moveLeft && !%this.moveRight)
    {
        %this.setLinearVelocityX( 0 );
    }

    if(!%this.moveUp && !%this.moveDown)
    {
        %this.setLinearVelocityY( 0 );
    }
}
```

Code Sample 4.4.1

This function does a few things, so let's look at it piece by piece. First we have:

```
function PlayerFish::updateMovement(%this)
{
    if(%this.moveLeft)
    {
        $FishPlayer.setFlipX(false);
        $FishPlayer.setLinearVelocityX( -$FishPlayer.hSpeed );
    }
}
```

Code Sample 4.4.2

Torque Game Builder – Fish Game Tutorial - Part 4

What this does is check if our fish has a value *moveLeft* set to true. If so, it then does our flip operation (just like it did in our move left function), and then sets the appropriate velocity. As you can see, the next three checks are the same, except they are each for a different direction. The code in each of these checks is the same code as in our move functions.

Next we get to the last two checks of this function:

```
if(!%this.moveLeft && !%this.moveRight)
{
    %this.setLinearVelocityX( 0 );
}

if(!%this.moveUp && !%this.moveDown)
{
    %this.setLinearVelocityY( 0 );
}
```

Code Sample 4.4.3

Here we first check to see if both move left and move right are false. If so, we set the velocity in the X direction to 0. We then do the same check for both up and down. Now that we have this function added we need to change both our move functions (called when a key is pressed) and our stop move functions (called when a key is released). Replace those functions with these (a big code snippet).

Torque Game Builder – Fish Game Tutorial - Part 4

```
function fishPlayerUp()
{
    $FishPlayer.moveUp = true;
    $FishPlayer.updateMovement();
}

function fishPlayerDown()
{
    $FishPlayer.moveDown = true;
    $FishPlayer.updateMovement();
}

function fishPlayerLeft()
{
    $FishPlayer.moveLeft = true;
    $FishPlayer.updateMovement();
}

function fishPlayerRight()
{
    $FishPlayer.moveRight = true;
    $FishPlayer.updateMovement();
}

function fishPlayerUpStop()
{
    $FishPlayer.moveUp = false;
    $FishPlayer.updateMovement();
}

function fishPlayerDownStop()
{
    $FishPlayer.moveDown = false;
    $FishPlayer.updateMovement();
}

function fishPlayerLeftStop()
{
    $FishPlayer.moveLeft = false;
    $FishPlayer.updateMovement();
}

function fishPlayerRightStop()
{
    $FishPlayer.moveRight = false;
    $FishPlayer.updateMovement();
}
```

Code Sample 4.4.4

You may notice a pattern in all of these functions. In the move call we set the proper direction value (such as *\$FishPlayer.moveLeft*) to “true” and call the update. Also in the stop call we set the same value to “false” and call the update. That way all movement settings are handled by the update function, while these key press events simply set values and call it. This will help keep our keys from canceling each other out. Time to test it. Be sure to save your script file and then open up *TGB*. Play the level and try moving your fish around. You'll notice it has much smoother movement now!

Torque Game Builder – Fish Game Tutorial - Part 4

4.5 Giving our fish a world limit

Our final issue is that our fish can go far beyond the view of our level. Since our level is small, and limited to just the camera size, we don't want the player to do that. We may want the fish to be able to go somewhat off the screen, but we want to prevent it from going too far. *TGB* makes this a very easy thing, since every object can have a *World Limit*. This world limit can be defined in the *Level Builder*, and we can specify a few different responses when the object hits it. Some of the most useful ones can be *CLAMP*, *BOUNCE*, and *KILL*. *CLAMP* will prevent it from moving outside the world limit. So basically you can think of it as a world bounds for the object. The *BOUNCE* response will cause the object to bounce off of the world limit, and *KILL* will delete the object. In our case we just want *CLAMP*, so the player cannot move far beyond our view.



Figure 4.5.1

Open up *TGB* and you should see your level with your fish in the center. Select your fish and hover over it. You should see the quick menu on the top left bounds of the fish (as shown in Figure 4.5.1). If you hover over these buttons, a tooltip will come up with a title for each of the tools. We want to use the second button from the right, the *World Limit* tool. So click that button (as shown in Figure 4.5.2).

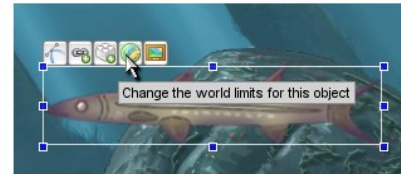


Figure 4.5.2

After clicking the *World Limit* tool, your view will zoom far out and show a really big, grey box around the level screen. This box has draggable corners (as shown in Figure 4.5.3), and represents the *World Limit*. Resize it to fit just beyond our level. If necessary, you may find it useful to scroll-zoom in. It is also a good idea to bring it up just inside the bottom, so the fish can't move below the ground (as shown in Figure 4.5.4).

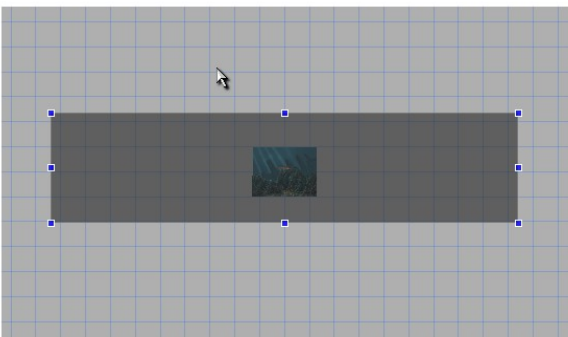


Figure 4.5.3

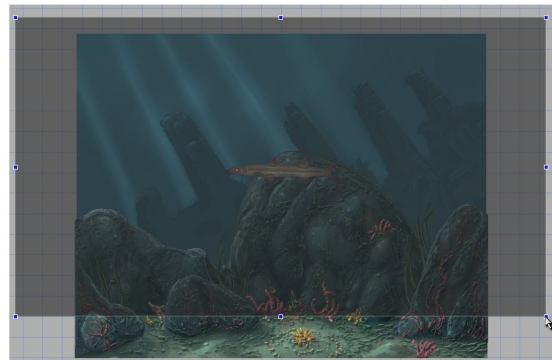


Figure 4.5.4

Now, to apply this world limit, simply click on the *Selection Tool* icon in the top toolbar (as shown in Figure 4.5.5).



Figure 4.5.5

Torque Game Builder – Fish Game Tutorial - Part 4

Now that we have applied the world limit we need to set the world limit mode, or nothing special will happen. Click the *Edit* tab (make sure our fish is still selected). In our properties look for the *World Limits* section. Click the text to expand the options (as shown in Figure 4.5.6). As you can see it defaults to "OFF" so simply click the dropdown and select "CLAMP" (as shown in Figure 4.5.7). Now our fish should respond properly to the borders. Time to test it! Be sure to save your level then hit the play button. Try to move your fish to the rightmost border, where it would previously go off screen. Now it gets stopped properly (as shown in Figure 4.5.8). Our movement is now pretty well cleaned up, time to move on to some gameplay action in part 5.



Figure 4.5.6



Figure 4.5.7



Figure 4.5.8