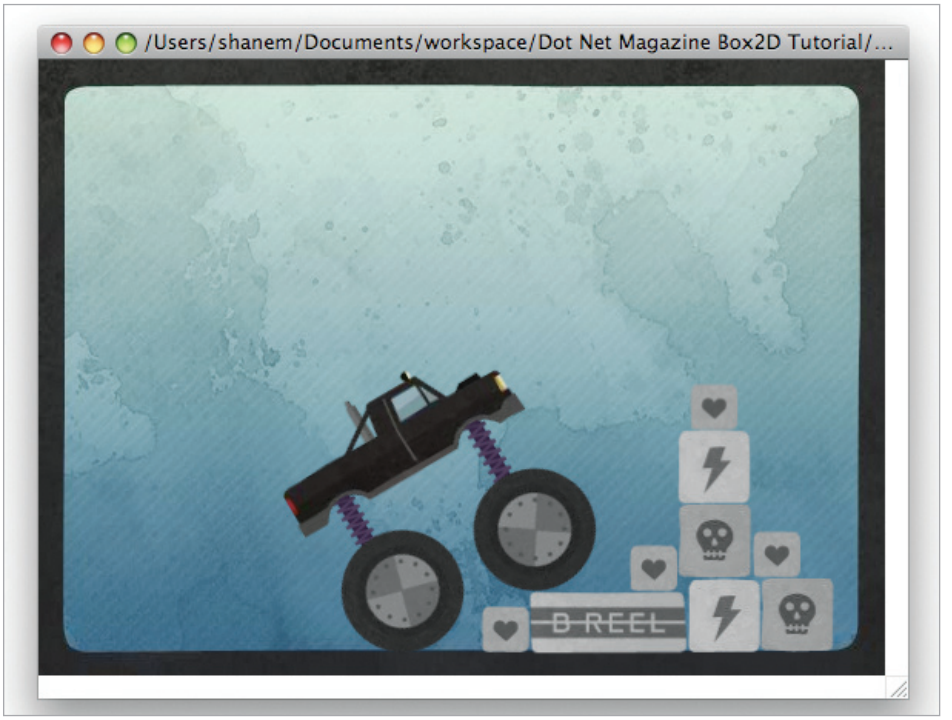
/Flash/create a physics game with Box2D

What could be more fun than creating your own monster truck game? Let **Shane McCartney** of top digital production company B-Reel show you how

Your essential CD
All the files needed for this tutorial can be found on this issue's CD.



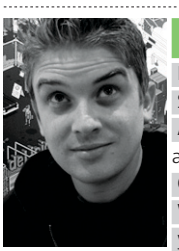
What you'll learn We'll show you how to use QuickBox2D to create a physics-based game. Learn how to make a physics scene, basic obstacles and the 'monster truck'. We'll finish this tutorial by adding interaction and provide tips/techniques to add your own killer graphics.

Knowledge needed	Flash, ActionScript, Photoshop
Requires	Flash, ActionScript, Photoshop
Project time	8 hours

With Flash CS5 on the horizon, there's never been a better time to get involved with physics for animations and games. Although this tutorial will cover physics for real-time user interaction, physics can also be used for life-like improvisational animations.

At B-Reel we've recently been hard at work on a physics-based game for Cheestrings.co.uk with FallonLondon. We'll share some inside knowledge, tips and tricks to make your own amazing physics game using the open source physics engine Box2D. We'll toe-dip you into becoming comfortable with Box2D by using another ActionScript library, QuickBox2D. This library is not a bare-bones version of Box2D but instead interfaces with Box2D at a higher and more comfortable level. This means functionality that ordinarily requires many lines of code is neatly provided in a syntax more suited to ActionScripters.

The end product is a small game that could be developed into a fully fledged driving game. You'll also become familiar with the terminology and practices used in physics engines like Box2D and be prepared for the new features in Flash CS5.

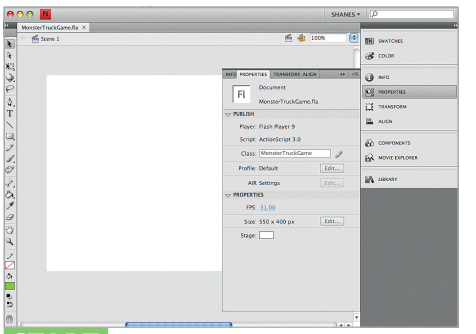


About the author

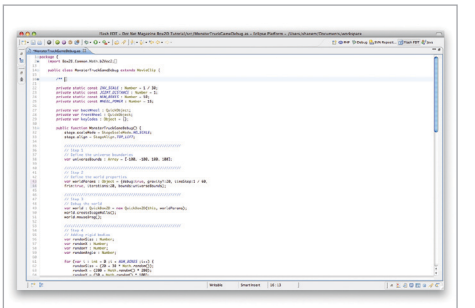
Name	Shane McCartney
Site	b-reel.com
Areas of expertise	Flash, AS, animation and design
Clients	Doritos, Cheestrings
What was the last 3D movie you saw?	*Avatar*

Expert tip Parameters

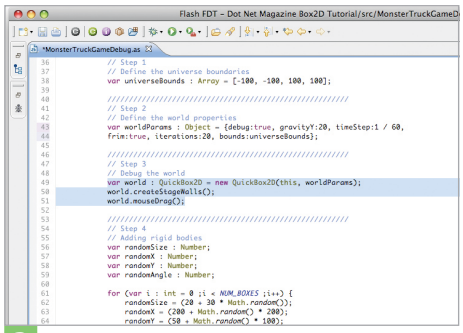
- A time step is a unit of time applied each time the physics engine is updated. Larger steps make the engine run faster. It's vital to keep your time step at a constant rate and not have it based to a frame rate.
- Iterations define the number of times the engine tries to solve a physics simulation. Sometimes by solving one collision, an object can be placed into collision with a different object, so iterations may eventually resolve this. Higher iterations mean a greater chance of a simulation resolving correctly but at the expense of unnecessarily achieving this.



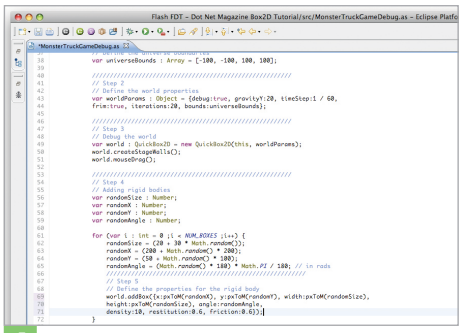
START **Boundary** Create a new document class extending **MovieClip**. Start by constructing an **Array** that defines the universe boundary. The unit for these values is physical metres not screen pixels. We find it's easiest to use a ratio for the conversion of screen to physical values.



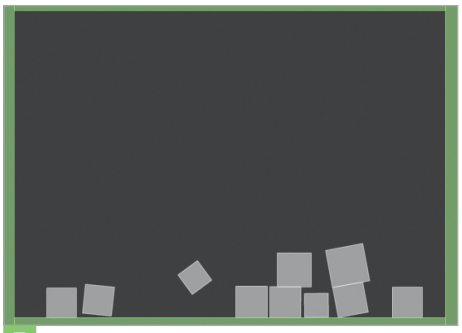
2 **Define properties** Next, define the properties of your world with gravity in m/s. Box2D allows for gravity in both an x and y direction but we tend to always use y. Next set the **timestep** to **1/60** and **iterations** to **20** and make sure **frame rate independent motion** is **true**.



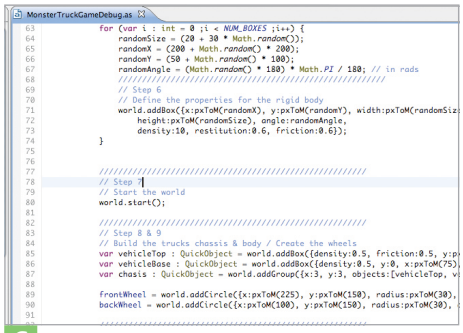
3 Create world Construct an instance of QuickBox2D with the properties of the world in its constructor. Use **create StageWalls()** to define temporary boundary walls fixed to the size of your Flash movie . This is different from the universe boundary (the area objects will be simulated within).



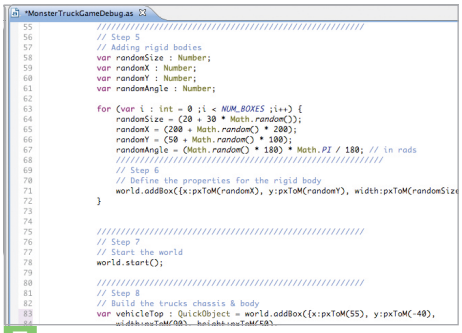
4 Debugging Further to the world parameters we've already defined, set **debug** to **true**. This allows for wireframe rendering of a physics simulation without the need for custom graphics. Also trigger the function **mouseDrag()** to allow for mouse-dragging of world objects.



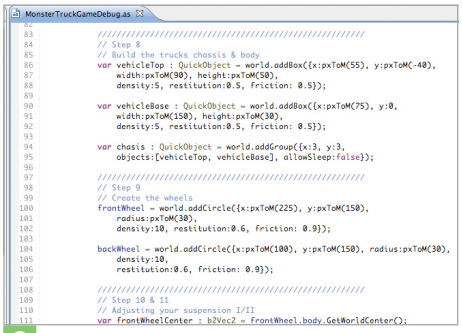
5 Rigid bodies All objects of mass are referred to as rigid bodies within physics simulations and do not deform on collision. Let's add some rigid body boxes with the function **addBox**. We can define the initial position and dimensions and even an initial angle in the unit radians.



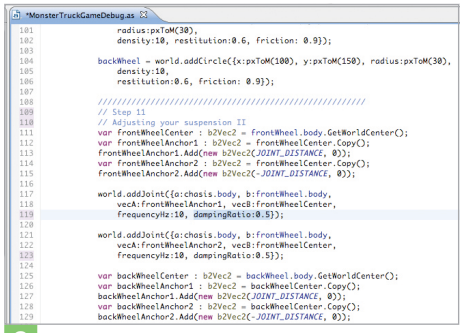
6 Defining properties All rigid bodies have a mass value calculated by **density**. Density determines the body's weight: 0 is weightless. Another parameter is **restitution**, which defines the amount of energy in a body after collision. So 0 resolves in no bounce and 1 is very bouncy.



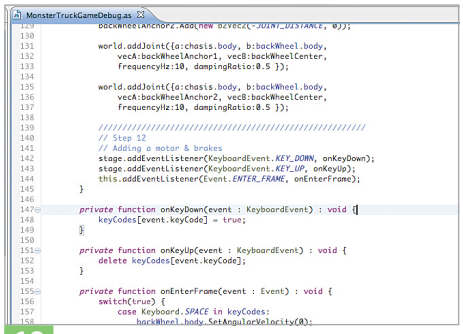
7 Start Start the world by triggering the **start** function. On publish, you can see the world in wireframe debug mode. We can see something in action so it's a good time to start tweaking parameters. This is often the case with physics simulations and it's good to be experimental.



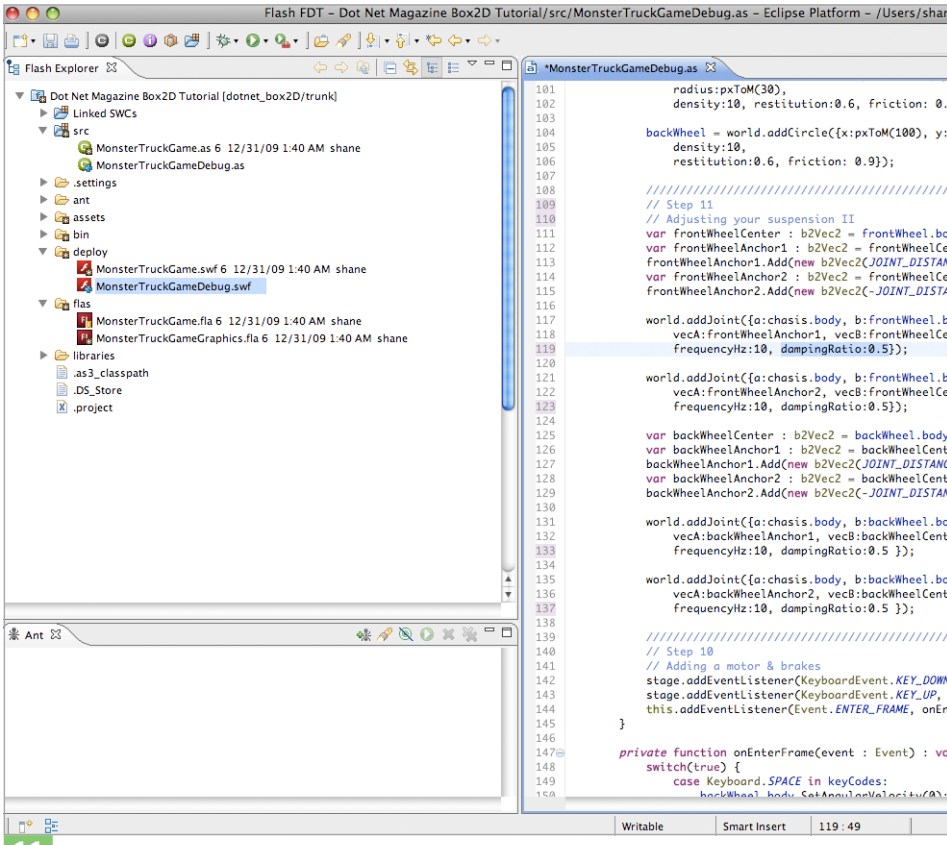
8 Body Create the truck's body with two boxes. To these we set a **friction** value. Friction is used to simulate objects sliding alongside each other. 0 is no friction; 1 is very strong. Next we group these boxes together using **addGroup()**. These boxes are now a compound shape.



9 Wheels Now add wheels using the **addCircle** function. Make sure you pick the correct radius and friction. Once both wheels are created, add two distance-based joints to simulate a suspension system. Each is applied left and right to a wheel's centre point to keep it in position.

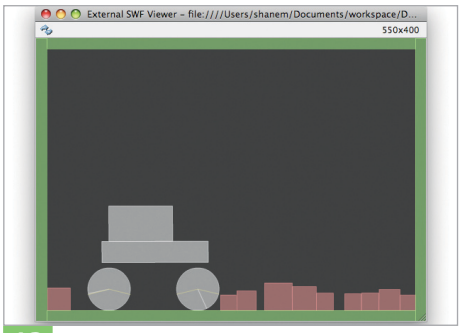


10 Suspension 1 When defining the joints, think about the distance between the wheels and the truck's body, as this defines suspension distance. Also important is **frequencyKhz**, which defines how springy the suspension is. A higher value makes it stiffer, a lower one looser.

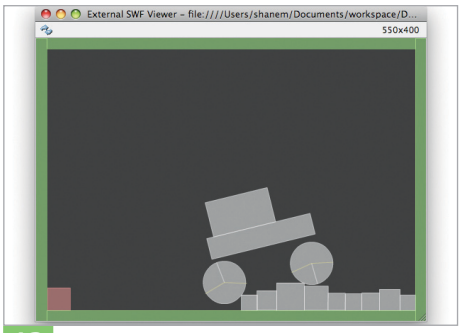


11 Suspension 2 The last important property is the **damping** ratio, which is like **friction**; however, it applies constantly. This allows for our suspension to ease into a resting position. We prefer really springy suspension systems to make the gameplay more fun, but you can try your own values to find something you prefer.

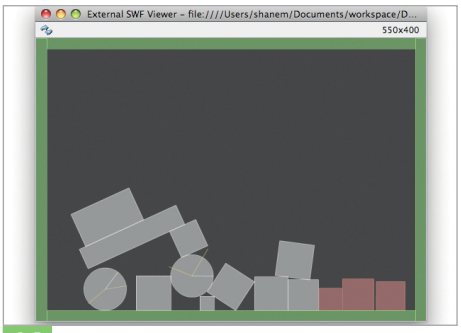
>>



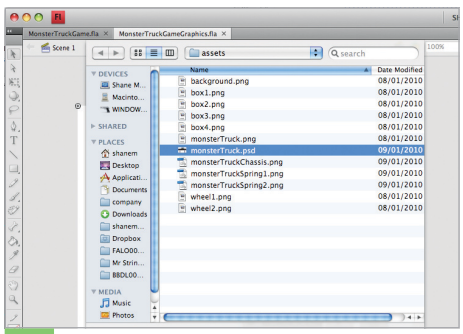
12 Motor To add keyboard interaction, create a stage event listener for the **KEY_DOWN** and **KEY_UP** events. Within each of the keyboard event listener functions, store and delete the ASCII keycode to an **Object** to later retrieve these interactions within an **ENTER_FRAME** listener.



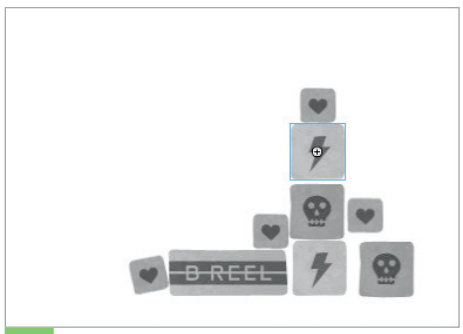
13 Brakes The key presses we're interested in are **KEY_LEFT**, **KEY_RIGHT** & **SPACE**. The **ENTER_FRAME** event listener checks which key is pressed. If it's an arrow key, we add/remove **angularVelocity** to the wheels: this turns them either clockwise or anti-clockwise. Define brakes by setting **angularVelocity** to **0**.



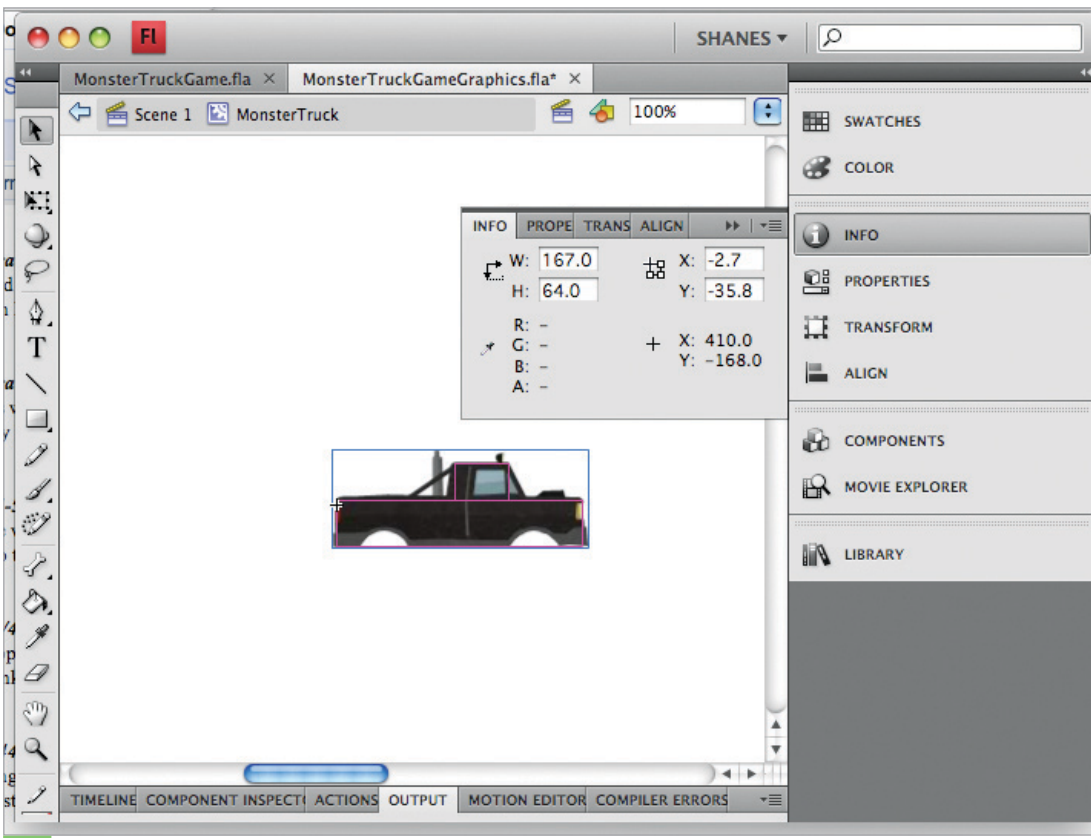
14 Ready to play We're now at a stage where you can drive around the monster truck and crash it into boxes but the artwork is totally lame. Adding custom artwork with QuickBox2D is easy. There are a few techniques to do this and we'll now show you the best approaches to use.



15 Graphics In QuickBox2D to apply your own custom artwork you use the **skin** property when adding a rigid body to the world using either **addCircle**, **addBox** or **addGroup**. The **skin** property can either be a **class** reference, a constructed **DisplayObject** or a linkage name string.



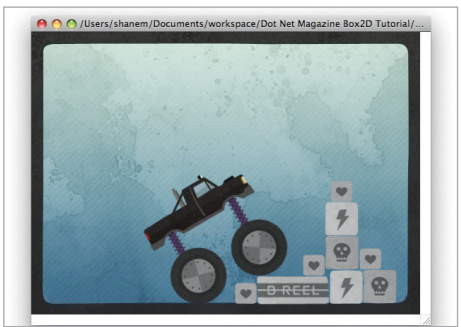
16 Skinning We want a big stack of boxes for our truck to crash through. Using QuickBox2D you can lay out as many boxes as you like, precisely within Flash's design view. QuickBox2D will then use the co-ordinates, dimensions and graphics automatically within the physics system.



17 Skinning truck 1 If you remember, when constructing the truck's body it was made from two boxes, combined to make a compound shape. Next we need to carefully position the monster truck artwork within a containing Sprite in your Flash library. When defining the compound shape, we set the skin to this linkage class name.

```
switch(true) {
    case Keyboard.SPACE in keyCodes:
        backWheel.body.SetAngularVelocity(0);
        frontWheel.body.SetAngularVelocity(0);
        return;
        break;
    case Keyboard.LEFT in keyCodes:
        backWheel.body.SetAngularVelocity(-WHEEL_POWER);
        frontWheel.body.SetAngularVelocity(-WHEEL_POWER);
        break;
    case Keyboard.RIGHT in keyCodes:
        backWheel.body.SetAngularVelocity(WHEEL_POWER);
        frontWheel.body.SetAngularVelocity(WHEEL_POWER);
        break;
}

//////////////////////////////////////////////////////////
// Step 18
// Skinning the truck II
var frontSpringPoint : Point = frontWheelSpring.parent.localToGlobal(new Point(frontWheelSpr
var backSpringPoint : Point = backWheelSpring.parent.localToGlobal(new Point(backWheelSpring

var frontWheelPoint : Point = new Point(Sprite(frontWheel.userData).x, Sprite(frontWheel.use
var backWheelPoint : Point = new Point(Sprite(backWheel.userData).x, Sprite(backWheel.userDa

var frontWheelDistance : Number = Point.distance(frontWheelPoint, frontSpringPoint);
var backWheelDistance : Number = Point.distance(backWheelPoint, backSpringPoint);

backWheelSpring.scaleY = backWheelDistance / 77;
frontWheelSpring.scaleY = frontWheelDistance / 77;
```

18 Skinning truck 2 Skinning wheels is easy but the springs are a bit trickier: they need to y-scale as the wheels rotate. First add the spring graphics inside the truck body. Next calculate the distance from the wheel to the spring position. Normalise this value as the spring's y-scale.



FINISH Wrap it up Set the default rendering for QuickBox2D to not draw lines or fills and add a background graphic to the **DisplayList**. Lastly, tweak the values for **friction**, **density**, **restitution** and **frequencyKhz** to make the physics simulation look right for the materials we've used.

Expert tip Concave or convex

The current ported AS3 version of Box2D only supports convex shapes. In this tutorial we show how to combine these shapes to create complicated compound shapes. In most cases this might be more than enough. However, when the shape is organic in its definition then the use of a concave shape is very handy. Thankfully QuickBox2D allows for this functionality built-in. The **addPoly** function within QuickBox2D allows for definition of the concave shape by a series of points and it will do the otherwise complex triangulation for you. Wowzas!