Getting Started with the Phaser Box2D Plugin

Version 1.0.0
March 11th 2015

By Richard Davey and Chris Campbell

Phaser Box2D Classes

World - Phaser.Physics.Box2D
Body - Phaser.Physics.Box2D.Body
Polygon - Phaser.Physics.Box2D.Polygon
DefaultContactListener - Phaser.Physics.Box2D.DefaultContactListener
DefaultDebugDraw - Phaser.Physics.Box2D.DefaultDebugDraw
PointProxy - Phaser.Physics.Box2D.PointProxy

How to add Box2D to your project

This plugin has been extensively tested against Phaser 2.2.2. It will also work with Phaser 2.3 upon release. We do not guarantee it will work with earlier builds.

You will find the distribution files in the `dist` folder.

Within there are two sub-folders: `minified` and `testing`. During development only you should use the `testing` files, as these will give you accurate line numbers should debugger errors occur. When you're ready to release your game swap to the files found in `minified` to save on bandwidth and download times.

Within each folder you will find three files:

- box2d-html5
- box2d-plugin
- box2d-plugin-full

box2d-html5

This is the Box2D JavaScript port. It was created with Google Closure Compiler and as such uses the `goog` global namespace.

box2d-plugin

This is just the Phaser Box2D Plugin code. It doesn't include the Box2D library itself.

box2d-plugin-full

This file contains both the Phaser Box2D Plugin and the Box2D Library. Depending on how your game is set-up, and what your build process is like, you may find it easier to use this single file alongside `phaser.min.js`. Be sure to include it after Phaser itself has been loaded.

If you have a more complex build process, or are working within an environment such as Angular or AMD you will probably need the split out files. Set-up of this is outside the scope of this manual.

Edit your html file

Assuming you are using the `full` file, your html should contain the following within the `<head>` block:

```
<script src="phaser.js" type="text/javascript"></script>";
<script src="box2d-plugin-full.js" type="text/javascript"></script>";
```

You may need to edit the paths depending on your environment.

Plugin Protection

We spent a lot of time and effort working on this plugin. We would really appreciate it if you didn't upload the raw unminified version of it to your site, or elsewhere. At the very least please use the minified build, but if at all possible we would love it if you could use the scrambled build instead.

How to enable Box2D physics

As with the other Phaser physics systems you need to tell it to start:

```
game.physics.startSystem(Phaser.Physics.BOX2D);
```

How to create a world

By enabling Box2D physics as above, a world is already created. You can access this as `game.physics.box2d` which is an object of type `Phaser.Physics.Box2D`, and wraps the actual Box2D world object to allow integration with Phaser.

If you want to access the actual Box2D world itself, it is accessible as `game.physics.box2d.world` and is an object of type `box2d.b2World`.

How to change the pixel-to-meter ratio

You can do this to change the pixel-to-meter ratio, but you should really do this before you have created any physics bodies, and not change it after that.

```
game.physics.box2d.ptmRatio = 40;
```

Note: the default setting for pixel-to-meter ratio is 50, which means a 50x50 pixel sprite will have physics behavior like a 1x1 meter box. Do not make the pixel-to-meter ratio value too large, otherwise you will have precision problems.

For example with a pixel-to-meter ratio of 1000, the same 50x50 pixel sprite would now be about the size of a matchbox in the physics world (0.05 meters). Making bodies smaller than 0.05 in the physics world will likely result in poor physics behavior.

How to change world characteristics

The main characteristic of the world is gravity, which you can change like this:

```
game.physics.box2d.gravity.y = 500;
```

You can also set the default density, friction and restitution (bounciness) for fixtures when they are created.

```
game.physics.box2d.density = 1;
game.physics.box2d.friction = 0.3;
game.physics.box2d.restitution = 0.2;
```

How to set up walls around the screen area

You can set up walls around the screen area with this function:

```
game.physics.box2d.setBoundsToWorld();
```

Technically, this sets up walls around the world boundary, not the screen. However unless you have changed the world boundary, the screen area and the world boundary will be the same. You can change the world boundary like this, (make sure to do this first if you use it):

```
game.world.setBounds(0, 0, 1600, 1200);
```

How to turn on debug drawing

You can call this in your render() function to draw the debug display for the whole world.

```
game.debug.box2dWorld();
```

By default, this will draw only fixtures. You can change this to show other aspects of the world as well, by setting any of these properties:

```
game.physics.box2d.debugDraw.shapes = true;
```

```
game.physics.box2d.debugDraw.joints = true;
game.physics.box2d.debugDraw.aabbs = true;
game.physics.box2d.debugDraw.pairs = true;
game.physics.box2d.debugDraw.centerOfMass = true;
```

How to draw debug shapes for specific bodies only

To draw debug shapes of a single body only, you can call this in your render() function:

```
game.debug.body(mySprite);
```

You can also use an optional color parameter to specify the color for the body:

```
game.debug.body(mySprite, 'rgb(127,0,192)');
```

How to create a body from a sprite

The simplest way to create a body for a sprite is like this:

```
var mySprite = game.add.sprite(400, 550, 'platform');
game.physics.box2d.enable(mySprite);
```

This will create a dynamic body with a rectangular fixture, having the same dimensions as the sprite texture. The object type of the created body is `Phaser.Physics.Box2D.Body`.

How to create a Physics Group

Rather than enable every single Sprite as a physics object, you can create a Group to contain them that automatically sets them up as physics sprites:

```
var group = game.add.physicsGroup(Phaser.Physics.BOX2D);
group.create(100, 100, 'alien');
group.create(300, 300, 'player');
```

This will create a `Phaser.Group` and add two sprites to it. Both of them will be automatically configured for Box2D physics.

How to create a body without a sprite

You can create a body without a sprite by constructing a Phaser.Physics.Box2D.Body and passing null for the sprite parameter:

```
var myBody = new Phaser.Physics.Box2D.Body(game, null, x, y);
```

This will create a dynamic body with no fixtures. The object type of the created body is `Phaser.Physics.Box2D.Body`.

How to get the sprite of a body, body of a sprite etc

You can access the sprite of a `Phaser.Physics.Box2D.Body` with the `parent` property.

You can access the body of a sprite with the `body` property.

The `Phaser.Physics.Box2D.Body` class wraps the actual Box2D body object to allow integration with Phaser.

If you want to access the actual Box2D body itself, it is accessible as `data` and is an object of type `box2d.b2Body`:

```
var rawb2Body = sprite.body.data;
```

How to change body characteristics

There are a variety of body characteristics you can modify. Please note that these are properties of `Phaser.Physics.Box2D.Body`, so if you have a sprite, you would need to call eg.

```
sprite.body.static = true;
```

```
// Setting one of these three to true will cause the others to become false
myBody.static = true;
myBody.kinematic = true;
myBody.dynamic = true;

// Movement characteristics
myBody.fixedRotation = true;
myBody.bullet = true;
myBody.linearDamping = 0.5;
myBody.angularDampiong = 0.2;
myBody.gravityScale = 0;

// These will affect all fixtures on the body
myBody.sensor = true;
myBody.friction = 0.9;
myBody.restitution = 0.3;
myBody.mass = 1.7;
myBody.collideWorldBounds = false;
```

The properties above are usually set only once after the body is created. While the game is running, you'll likely want to modify other properties as well:

```
myBody.x = 100; // pixels
myBody.y = 200;
myBody.angle = 90; // degrees
myBody.angle = 1.57; // radians
myBody.velocity.x = 30; // pixels/sec
myBody.velocity.y = 40;
myBody.angularVelocity = 1.2; // radians/sec
```

How to change or add fixtures on a body

You can use the `setXXX` or `addXXX` functions listed below to set or add fixtures to an *existing* body.

Using set will clear any existing fixtures first, while add will keep any existing fixtures.

```
myBody.setCircle(40); // radius
myBody.setRectangle(50, 100, 0, 0, 10); // width, height, offsetX, offsetY, angle (radians)
myBody.setEdge(-10, -20, 20, 60); // x, y, x, y of the two end points
myBody.setChain([-10, -20, 20, 60, ...]); // flat array of x,y values
myBody.setPolygon([-10, -20, 20, 60, ...]); // flat array of x,y values
```

Concave polygons will automatically be converted into polygons that Box2D can use (max 8 vertices, convex, CCW winding), so you don't need to worry about whether your polygon is convex or wound correctly. However, you should make sure that none of the polygon edges cross over each other.

Each of these functions will return the created fixture which is a `box2d.b2Fixture`. You may want to keep a reference to this for other features mentioned below.

How to destroy bodies

You can permanently remove a body from the world by calling destroy on it:

```
myBody.destroy();
```

However, if you created the body from a sprite, in most cases you will want to destroy the sprite and the body together, which you can do like:

```
mySprite.destroy();
```

If for some reason you want to destroy only the body and keep the sprite, you should set the body property of the sprite to null after destroying the it:

```
mySprite.body.destroy();
mySprite.body = null;
```

You can temporarily de-activate a body by calling kill on it:

```
myBody.kill();
```

To revive the body later, you can call reset on it, with the new position:

```
sprite.body.reset(pointer.x, pointer.y);
```

Usually you will want to have the sprite and the body both be killed and revived at the same time, in which case you can call these same functions on the sprite instead.

How to find out what is under the mouse pointer

(For mouse location, use game.input.mousePointer.x, game.input.mousePointer.y in the functions below.)

You can get a list of bodies at a specific point like this:

```
var bodies = game.physics.box2d.getBodiesAtPoint(x, y);
```

That will give you an array of `Phaser.Physics.Box2D.Body` objects.

If you have bodies with multiple fixtures and you want to find the individual fixtures at a specific point, you can do this:

```
var fixtures = game.physics.box2d.getFixturesAtPoint(x, y, true);
```

That will give you an array of `box2d.b2Fixture` objects.

Keep in mind that these methods actually work on fixtures, so bodies with no fixtures will not be found. Likewise, fixtures with no area will also not be found, such as edge and chain fixtures.

How to find out what is in an area

There are two ways you can find out what is at an area of the world.

The first is an AABB (axis aligned bounding box) overlap check. This is quite fast, but it only tells you that the AABB of the fixtures are overlapping the test AABB. It does not necessarily mean that the fixtures themselves are actually touching the test AABB:

```
var queryOutput = game.physics.box2d.queryAABB( x, y, width, height );
```

This will give you an array of objects that contain two properties, the body and fixture of each fixture that was found to be overlapping the test AABB. The structure of these objects is like this:

```
{
    body: Phaser.Physics.Box2D.Body,
    fixture: box2d.b2Fixture
}
```

The other method available requires a fixture to be the test shape, and you can check to see what other fixtures in the world overlap with it. This method is not quite so fast because it has to first do the AABB check as above, and then also check if the shapes found in the AABB are actually touching the test fixture:

```
var queryOutput = game.physics.box2d.queryFixture( polygonFixture );
```

How to find out if a point is inside a body

You can find out if a body contains a point like this:

```
myBody.containsPoint(game.input.mousePointer)
```

The parameter given to this function should be an object that has x and y properties, so most objects in Phaser will be suitable, including Point objects and Input Pointers.

Technically, this checks if any of the fixtures of the body contains the point, so if the body has no fixtures, or only has fixtures with no area (edge and chain fixtures) then it will always return false.

How to drag things with the mouse

For most cases, you can enable simple mouse-dragging by adding these three callbacks in your create() function:

```
// Set up handlers for mouse events
game.input.onDown.add(mouseDragStart, this);
game.input.addMoveCallback(mouseDragMove, this);
game.input.onUp.add(mouseDragEnd, this);
```

... and then implement these to simply call the defaults for mouse dragging as provided by the world object:

```
function mouseDragStart() {
    game.physics.box2d.mouseDragStart(game.input.mousePointer);
}

function mouseDragMove() {
    game.physics.box2d.mouseDragMove(game.input.mousePointer);
}

function mouseDragEnd() {
    game.physics.box2d.mouseDragEnd();
}
```

How to call a function to do something when bodies touch each other

You can set up a callback function so that when a body touches another body, your callback function will be called:

```
myBody.setBodyContactCallback(otherBody, myCallback, this);
```

The first parameter here is the body you want the callback to be called for. The second parameter is the callback function itself. The callback function would be implemented like this:

```
function myCallback(body1, body2, fixture1, fixture2, begin) {
    // If 'begin' is true this is a begin contact, otherwise a contact has just ended
    ... do something
}
```

In this example, when myBody begins contact with otherBody, the myCallback function will be called. The parameters passed to myCallback will be:

```
body1 - myBody
body2 - otherBody
fixture1 - the fixture in myBody that is contacting
fixture2 - the fixture in otherBody that is contacting
begin - true if the contact is beginning, false if the contact has just finished
```

As well as setBodyContactCallback, you can also detect when the body touches a specific fixture of another body by setting up a callback like this:

```
myBody.setFixureContactCallback(otherFixture, myCallback, this);
```

... and when the body touches a fixture with a specific category like this:

```
myBody.setCategoryContactCallback(2, myCallback, this);
```

How to use raycasting

To cast a ray between two points and check for intersection with fixtures, you can do this:

```
var raycastOutput = game.physics.box2d.raycast( x1, y1, x2, y2 );
```

This will cast a ray from x1,y1 to x2,y2. The direction of the ray is important because it will only intersect fixtures when cast against them from the outside. The return value from this function is an array of objects containing the intersection location, normal, and which body and fixture it was. The structure of these objects is like this:

```
{
    body: Phaser.Physics.Box2D.Body
    fixture: box2d.b2Fixture,
    point: {x: number, y: number},
    normal: {x: number, y: number}
}
```

How to convert coordinates between pixel and physics

For most usage, you can deal with coordinates as pixels because the wrapped classes do all the conversion between pixels and meters internally. If you need to deal with Box2D physics units coordinates directly, you will need to convert coordinates as below:

```
// Converts 100 pixels to meters
var meters = game.physics.box2d.pxm(100);
```

and the opposite:

```
// Converts 100 meters to pixels
var px = game.physics.box2d.mpx(100);
```

Further Questions?

Please check the Examples provided. They will cover lots of use cases not covered here.

But you're also welcome to ask on the Phaser Forum. Just be sure to mention in your post you are using Box2D and not one of the other physics systems.

If you think you've found a bug in the plugin then please report it. Either via Github Issues or email support@phaser.io.

License

Box2D License