Box2d demo tutorial

- Download the demo files from: http://github.com/gazpachu/box2d/archive/master.zip

- [optional] Download the box2d manual from: http://box2d.org/manual.pdf

Description

This demo will allow the user to create boxes by clicking the mouse on different parts of the canvas. The boxes will then fall down (if the gravity has been set in the Y axis) and collide with the world boundaries or other box2d bodies.

Tutorial

1. Open js/main.js in your text editor

- If you don't feel like typing, you can copy/paste the code from this document, located at **docs/tutorial.pdf** ;-P

2. Add the box2d objects and the demo variables

```
var world,
b2Vec2 = Box2D.Common.Math.b2Vec2,
b2BodyDef = Box2D.Dynamics.b2BodyDef,
b2Body = Box2D.Dynamics.b2Body,
b2FixtureDef = Box2D.Dynamics.b2FixtureDef,
b2Fixture = Box2D.Dynamics.b2Fixture,
b2World = Box2D.Dynamics.b2World,
b2PolygonShape = Box2D.Collision.Shapes.b2PolygonShape,
b2DebugDraw = Box2D.Dynamics.b2DebugDraw,
worldScale = 30,
canvas = document.getElementById("canvas"),
context = canvas.getContext("2d");
```

Box2d uses meters instead of pixels, that's why we need a variable called worldScale to change values from pixels to meters and viceversa. It's a best practice to assume that 1m = 30px.

Technically speaking, box2d doesn't need a canvas to work, but in this demo, we are using an object called DebugDraw that uses a canvas to paint things in the screen.

3. Set the canvas dimensions

```
canvas.width = 800;
canvas.height = 600;
```

4. Initialize the Box2d world

```
world = new b2World(new b2Vec2(0, 10), true);
```

The first parameter is the amount of gravity in the **x** and **y** axis. The second parameter indicates if the world can be set to sleep when there's nothing moving.

5. Make a function to create a box2d body with the shape of a box every time the user clicks the mouse

```
function createBox(width, height, pX, pY, type, data) {

     var bodyDef = new b2BodyDef();
     bodyDef.type = type;
     bodyDef.position.Set(pX / worldScale, pY / worldScale);
     bodyDef.userData = data;

     var polygonShape = new b2PolygonShape();
     polygonShape.SetAsBox(width / 2 / worldScale, height / 2 /
worldScale);

     var fixtureDef = new b2FixtureDef();
     fixtureDef.density = 2.0;
     fixtureDef.friction = 0.2;
     fixtureDef.restitution = 0.5;
     fixtureDef.shape = polygonShape;

     var body = world.CreateBody(bodyDef);
     body.CreateFixture(fixtureDef);
}
```

Box2d entities are called “bodies”. In this function we are creating a single body with the shape of a box. Every body needs to have a “body definition”, a “shape” and a “fixture”. These objects will define how the body interacts with the world (by defining its physic properties), it's shape, position and size. In this demo we are also using a special container called “userData” that works in a similar way as the “data” containers in jQuery.

6. Next we are going to create the walls that define the boundaries of the box2d world

```
createBox(800, 1, 400, 600, b2Body.b2_staticBody, null);
createBox(800, 1, 400, 0, b2Body.b2_staticBody, null);
createBox(1, 600, 0, 300, b2Body.b2_staticBody, null);
createBox(1, 600, 800, 300, b2Body.b2_staticBody, null);
```

The first call to createBox will create a body with the shape of a thin line because the value of the height property is 1. Bear in mind that a box is created from its center point, not from the top left corner... that's why we use 400 as the X value and 300 as the Y value, because that's half of the width and height of the canvas (800x600).

7. On mouse down, create a new box

```
document.addEventListener("mousedown", function (e) {

     var offset = $('#canvas').offset();
     createBox(32, 32, e.clientX - offset.left, e.clientY -
offset.top, b2Body.b2_dynamicBody,
document.getElementById("crate"));
});
```

Notice that we are passing as the “type” parameter “b2_dynamicBody” as opposed to the type “b2_staticBody” that we were using when creating the world boundaries. The reason behind this is that the boxes will be moving around the world and the walls will be static.

The 32px of width and height correspond to the dimensions of the box image in **img/crate.png** and the canvas offset top and left values are used to get the correct position relative to the canvas position on the page.

8. Initialize the debugDraw object

```
function initDebugDraw() {

     debugDraw = new b2DebugDraw();
debugDraw.SetSprite(document.getElementById("canvas").getContext(
“2d"));
     debugDraw.SetDrawScale(30.0);
     debugDraw.SetFillAlpha(0.5);
     debugDraw.SetLineThickness(1.0);
     debugDraw.SetFlags(b2DebugDraw.e_shapeBit |
b2DebugDraw.e_jointBit);
     world.SetDebugDraw(debugDraw);
}
```

The debugDraw object is a helper to paint bodies on a canvas. This is not required in a production environment (because of its graphic limitations), but can be useful for demos or testing purposes. The debugDraw parameters and methods can be checked in the box2d manual.

9. Update the box2d bodies every frame

```
function update() {

     world.Step(1 / 60, 10, 10);
     world.DrawDebugData();

     for (var b = world.m_bodyList; b !== null; b = b.m_next) {
          if (b.GetUserData()) {
               context.save();
               context.translate(b.GetPosition().x * worldScale,
b.GetPosition().y * worldScale);
               context.rotate(b.GetAngle());
               context.drawImage(b.GetUserData(),
-b.GetUserData().width / 2, -b.GetUserData().height / 2);
               context.restore();
          }
     }

     requestAnimFrame(update);
}
```

The world.step method has 3 parameters. The first one sets how often box2d will run. Generally, physics engines for games like a time step at least as fast as 60Hz or 1/60 seconds. The second and third parameters are used to indicate the velocity and position iterations in the constraint solver. The suggested iteration count for Box2D is 8 for velocity and 3 for position, although you can tune this number to your liking, just keep in mind that this has a trade-off between performance and accuracy.

This concept can be a bit tricky to understand, so I recommend reading more about in the manual (2.4 Simulating the World of Box2D).

The following **for** loop is only required to update the position and rotation of the images attached to the boxes that we created. The actual position and rotation of the bodies is automatically calculated. You can confirm this by commenting out the for loop and see what happens... ;-)

The loop iterates among all the bodies in the world. For every body, it checks if it has an image attached (GetUserData). Then it moves and rotates the canvas context to draw the crate image with its updated position. Notice that the values are being parsed with the worldScale variable to change from px to meters.

Finally we wait for the browser to give us the ready signal to calculate another frame, which is done again with the update function. It's a function that calls itself every time the browser says it can do it.

10. Initialize the demo

```
initDebugDraw();
update();
```

This is the entry point of our application!