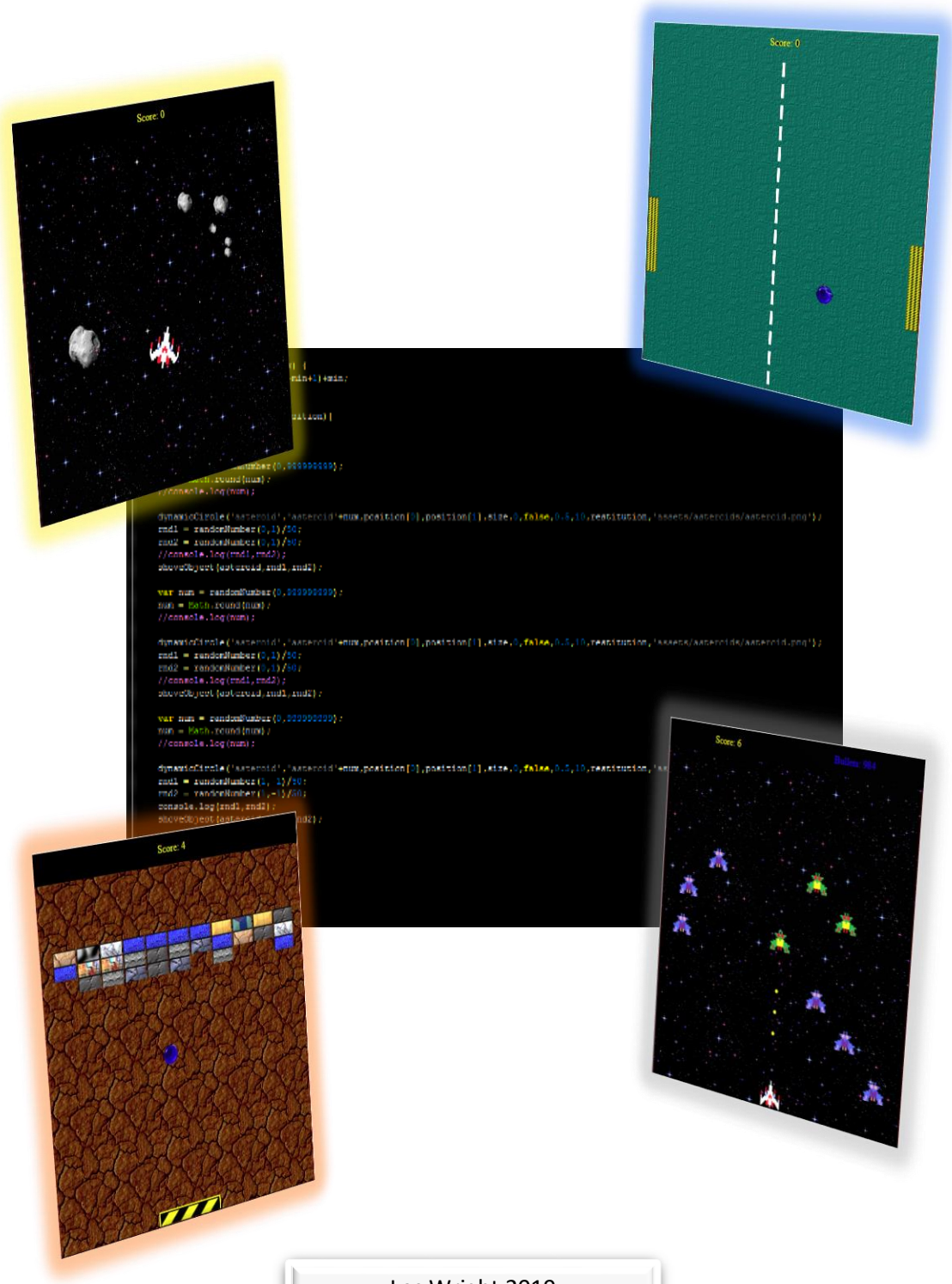
Giftwrap 3.0 CSS



Les Wright 2019

Contents

Giftwrap 3.0 CSS

Les Wright 08/11/2019

What is it??

Giftwrap is a thin wrapper or API for Box2D and CSS, to allow new programmers to produce a web game with minimal coding skills using JavaScript.

This work was inspired by the teaching of Dr Russell Hunter of Perth College UHI whose 4th year Web Programming module was awesome!

Box2d is very complex and is difficult to master, so all of the hard work is done for you.
All of the things required to draw objects, apply sprites, apply sounds and handle collisions are wrapped up in easy to call functions, no object oriented programming is required.

The game design and logic will be all up you.

Simple good looking interactive games can be easily implemented such as:
(In order of difficulty)
Pong
Breakout
Galaga
Frozen Bubble
Asteroids

Giftwrap whilst very functional, could still be considered Alpha. A lot of great functionality is built in, but some Box2d things are not implemented, and some are partially implemented.

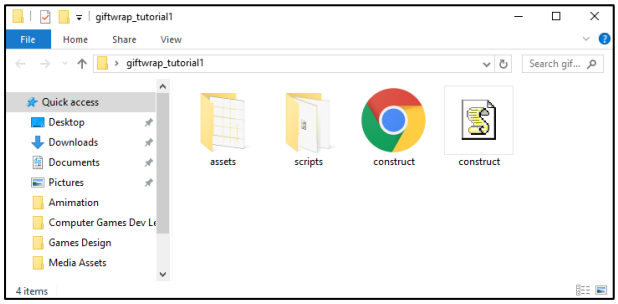
Because this is vastly simplified, there is a lack of granular control over objects that you would normally have in Box2D

Bugs have mostly been ironed out, or dealt with in some way, chances are you might well discover one somewhere!

If you really want to develop Box2D games, go and learn Box2D!!

Setting up

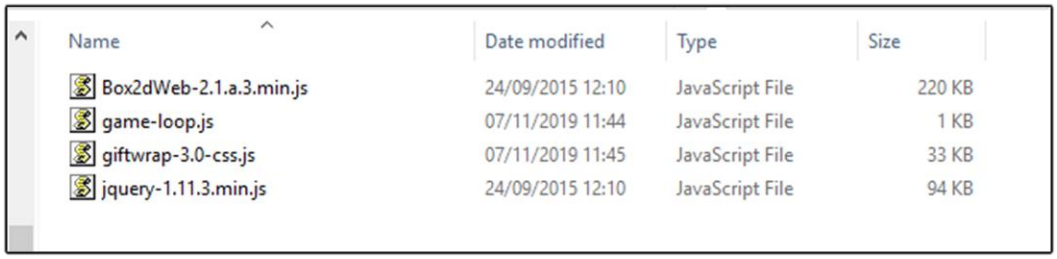
Download giftwrap_tutorial1.zip and unzip it:



There are 4 items in this folder:

- assets A folder for any assets i.e. images & sounds, currently this contains grid.png
- scripts A folder containing all the libraries required
- construct.html An HTML file to load the libs and display our output and get our input
- construct.js Our work area!

Inside the 'scripts' folder, you should see the following items:

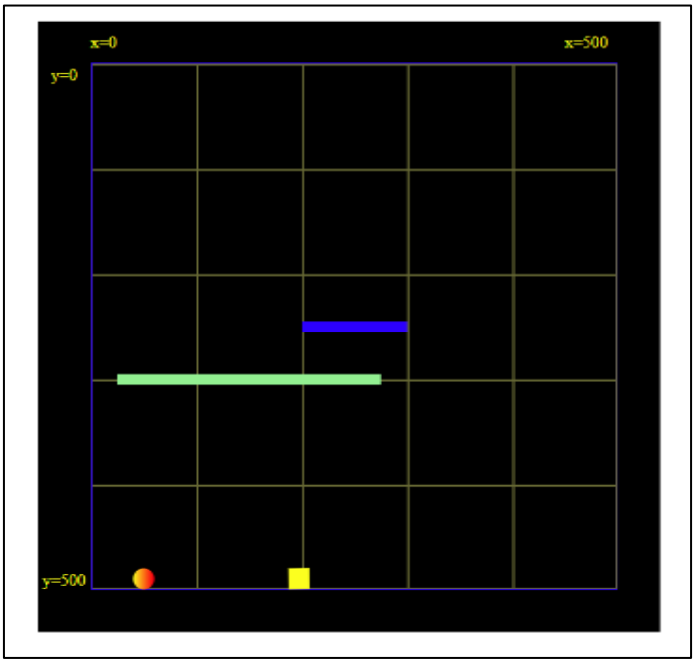


There are 4 items in here, do NOT edit these! But feel free to take a look!

- Box2dWeb-2.1.a.3.min.js The Box2d Physics engine (required by giftwrap)
- jquery-1.11.3.min.js jquery library for getting user input (required by giftwrap)
- giftwrap-3.0-css.js The giftwrap API
- game-loop.js Animation loop for the game

Open ‘construct.html’ with CHROME web browser. We use Chrome for all these tutorials, NOT Internet Explorer, Edge, Firefox or anything else!

You should see a scene similar to the one on the left.



An Orange ball will fall from the top left hand corner to the bottom.

A yellow Square will fall from the left of the page.

Hit Refresh (F5) if you missed it!

The Orange ball is called a ‘Dynamic Object’, i.e. it is one that can move.

The yellow Square is also a Dynamic Object.

The narrow blue box in the centre of the screen is a ‘Static Object’ i.e. it does not move.

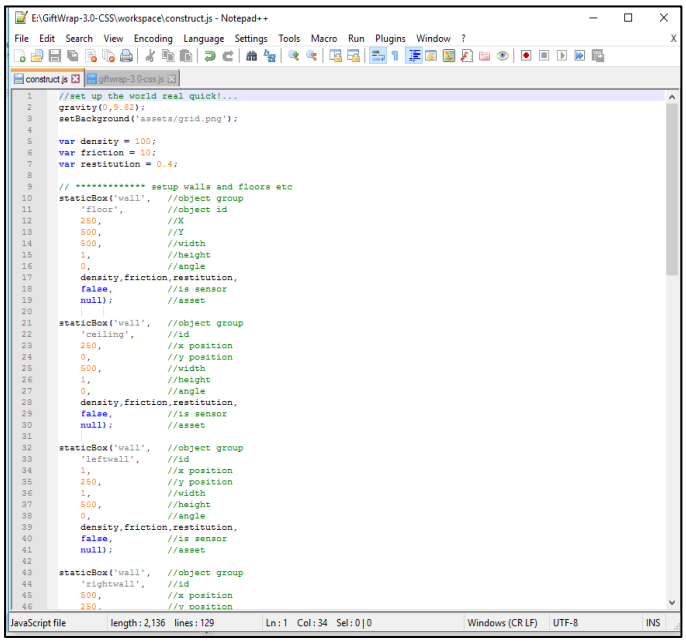
When the square hits it, it bounces off.

The long green box, allows objects like the ball to pass through it. This is called a “Sensor” more about those later!

A grid is provided with an X and Y axis to help guide you with positioning objects, and estimating what size they should be. All measurements are in pixels.

Open up 'construct.js' with Notepad++, and take a look at the code required to create this setup:

We have programmed using JavaScript before, so this should seem very familiar to you by now!



The first line of code reads:

```
gravity(0,9.8);
```

As we can see, gravity is a function that takes two parameters X,Y

Currently X is zero

Y is 9.8 (Earth's gravity will accelerate you toward the ground at 9.8m/s^2)

If you make this value negative, what do you predict will happen to the ball?

If you give X some value, what do you suppose will happen?

Try it!

Next we have another short function call: `setBackground('assets/grid.png');`

The function setBackground(), takes a filename and path as its argument, and displays the image for you. The image should be 500x500px

There are some other functions built in that handle background that allow you to scroll the background, and change the background.

```
scrollBackground(X,Y,S);
```

X direction
Y direction
Speed

If you give the Y axis a negative value, the background image will scroll downwards, giving the illusion of the game area moving upwards.

```
scrollBackground(0,-1,0);
```

It is possible to change the background at any point in your program using changeImage() like this.

```
changeImage('canvas','assets/yourimage.png');
```

The first argument is what we are changing (in this case canvas), and the second argument is the path to the image.

NOTE! This also works with any object in the game!

```
changeImage('player1','assets/ball.png');
```

Static Objects:

The next 4 function calls build static boxes for the 4 walls of the game area. We will look at the first, and the rest should make sense, but before we do we have a few variables declared that need some explanation:

```
var density = 100;
var friction = 10;
var restitution = 0.4;
```

Density: Density of the objects, the higher the denser. Think cork ball vs lead ball.

Friction: The higher, the more friction an object has. 0 would be like wet ice on wet ice.

Restitution: Roughly bounciness. A value of 0 means when an object hits another, it does not bounce. A value of 1 would mean a ball dropped from a height would return to its original height (Impossible in the real world!) and would bounce forever! Over unity, i.e. a number greater than 1 will cause an object to gain momentum during a collision, also impossible in the real world!

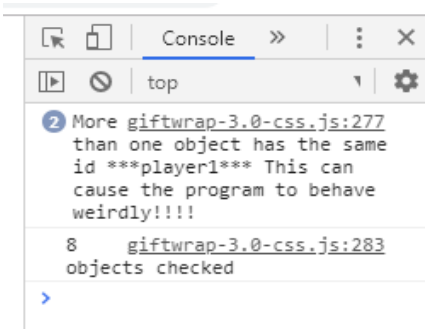
So now let's examine the first function:

```
staticBox('wall',        //Object Group
      'floor',           //Object id
      250,               //X Position
      500,               //Y Position
      500,               //Width
      1,                 //Height
      0,                 //Angle
      density,friction,restitution,
      false,             //is sensor
      null);             //Asset
```

Object Group: An object can belong to a 'class' or 'group' of objects. This object is therefore in the group 'wall' and many other objects can also be in the group 'wall'

Object ID: Each object in the world MUST have a UNIQUE id! If they do not, then inexplicable things will happen within your world (objects can appear to pass through others, and just generally behaving strangely!), it generally does not crash the program, so an error checker is built in for this!

If you do have two objects with the same ID, the following error message will be displayed in the console to remind you, and will give the name of the offending objects in asterisks:



X position and Y Position: Just as they say! All measurements reference the centre of a box. So in this example the box is positioned in the middle on the X axis, and all the way at the bottom on the Y axis. (The floor, the next three functions build a ceiling and two walls. We need these as a boundary for our objects, else they can escape from the canvas!)

Width and height: Once again, just as it says on the tin. This one is 500px wide, and 1px high.

Angle: Angle in degrees. In this example it is 0, so level.

Density, Friction and Restitution: You can give each object any combination of these properties. In this example I have used the ones defined at the top of the script. You should too, to save you making mistakes, except where you want a specific object to have special properties.

Sensor: Either true of false. A sensor will allow objects to pass through it, but we can still detect it! Useful for checkpoints for example! Only static boxes can be sensors, and by default will show as green in the game area.

Asset: in this object the asset is defined as: `null` this means do not load an Asset, and defaults to a plain blue object in the case of a static object.

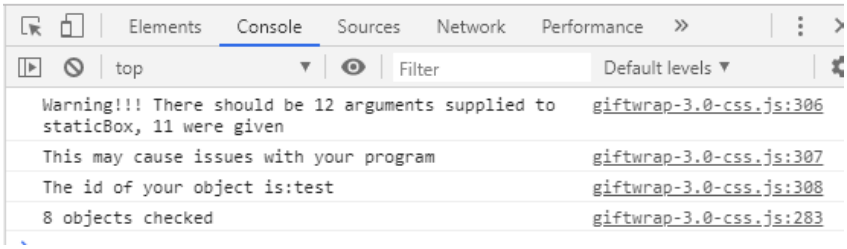
If you wish to load an asset you do it in single quotes like this: `'assets/yourimage.png'`

****Note: Images should be correctly sized ahead of time! If your object is 40*40 pixels, make your asset 40*40!**

If your object is a circle and is 40 px wide, create a 40*40pc image. giftwrap will fit a square image into a circular object, as long as it is sized correctly!

In total, 12 arguments should be supplied to staticBox(), if you miss one, or provide too many, strange things can happen, so be careful!

As always, have the console window open when running your code. If you have supplied the wrong number of arguments, you will see a message like this:



Note: this provides you with the ID of the object, to help you find it in your code!

Static Circles are built with the following properties:

staticCircle(group,id,X,Y,S,angle,density,friction,restitution,asset)

S is Size, and corresponds to with/height in pixels

Angle: this is not a typo! If the angle is set to something other than zero, your asset should exhibit a corresponding rotation!

Dynamic objects:

Dynamic objects are coloured Orange (actually a gradient, so we can see if it rotates!), if no asset is supplied.

```
dynamicCircle('ball',          //object group
      'player1',               //id
      50,                      //x position
      20,                      //y position
      20,                      //width
      0,                       //height
      false,                   //fixed rotation?
      0.2,friction,restitution,
      null);                   //asset
```

*Note how above I have individually tweaked 'density'

The properties here are almost the same as for static circles, with one extra property 'fixed' which affects rotation.

This can be either set to **true** or **false**

true: The objects rotation is fixed, and cannot rotate.

false: The object is free to rotate.

Dynamic boxes are again, almost the same as for static boxes, except they have a 'fixed' property, and as with dynamic circles, determine whether the object can rotate or not.

dynamicBox(group,id,X,Y,W,H,angle,fixed,density,friction,restitution,asset)

```
dynamicBox(
      'dbox',            //object group
      'player2',         //id
      300,               //x position
      50,                //y position
      20,                //width
      20,                //height
      46,                //angle note, this is a perfect world!
      false,             //fixed rotation
      density,friction,restitution,
      null);             //asset
```

In my example here, the Dynamic box that falls is set to an angle of 46 degrees and has a note saying: `this is a perfect world!`

Box2d physics is unlike the real world, in that objects created are **perfect**. There are no surface imperfections, air resistance and level surfaces are perfectly level.

If I make a box with an angle of 45 degrees and drop it onto a level surface, it will land precisely on its corner and balance!

Functions that affect objects:

Move objects

The following functions allow us to move dynamic objects:

**NOTE, you can try these in the console ☺

```
shoveObject('player1',0,10);
```

```
pushObject('player1',0,10);
```

They take the following arguments:

- Object id
- X vector
- Y vector

In the above examples the objects would be moved straight up.

The differences between the two functions are:

- shoveObject: Behaves as if you whacked the object to set it in motion.
- pushObject: Behaves as if you pushed the object to set it in motion.

Rotate Objects

```
rotateObject('player1',45);
```

Rotate object with id ‘player1’ 45 degrees.

Stop Objects

The following function allows us to dead stop an object in motion, and takes just the ID as an argument:

```
deadStop('player1');
```

This is a very handy function for controlling an object, especially if we turn off friction!

For example if we set a keyDown to move an object, we could set a keyUp to stop the object. Think about it!

Position Objects

The following function can arbitrarily position any object:

```
positionObject'player1',250,490);
```

It takes the arguments:

- Object id
- X position
- Y Position

Note: This function works for ALL objects, dynamic or not!

Get object position

The following function can GET position any object:

```
getPosition('player1');
```

Think of a use for that!

Get Size of an object

The following function can GET position any object:

```
getSize('player1');
```

Destroy an object

The following will destroy ANY object. Simply pass it the objects id:

```
destroy('player1');
```

Change the asset of an object

We can change the asset of any object in the game with:

```
changeImage('player1','assets/newimage.gif');
```

Generate a random integer

```
randomNumber(0,10);
```

First argument is the minimum value, the second is the maximum value.

This example generates a random int between 0 and 10

Dealing with collisions

By Default, when an object hits another object, they bounce off each other.

To DO something based on a collision the following function is built in:

superCollider();

It is designed to take the following arguments (5) in the following order:

```
superCollider(
      'group',   //collide by group or id
      'bullet',  //objectA
      'enemy',   //objectB
      'enemy',   //object to destroy
      'hitEnemy' //function to run
);
```

- Collide by: this can either be ‘group’ or ‘id’
  If we use ‘group’ all collisions within that group are handled in the same way, so if we wanted all bullets to destroy all enemies, this would be what to use right?

  If we use ‘id’ only when specific items with specific ids collide to we do something, so for example we might have a ‘checkpoint’ that only does something if player1 touches it, but does nothing if the enemy touches it.

- Object A: the first object. Use the group name if you have already specified group, or it’s id if you specified id

- Object B: the second object. Use the group name if you have already specified group, or it’s id if you specified id

- Object to destroy: The name of the object you want to destroy. This assumes you want to destroy something! If you do NOT want to destroy something, use `null` with no quotes!

- Function to run: A user defined function to run. Use `null` with no quotes, if you have no function to call, else use the function name. What function might you use? Up to you! You might want to make a function that increments score when a hit is made right?

This function is special. You may remember from the earlier tutorials that JavaScript does NOT bomb out if you pass a function too many arguments!

This function is designed to accept as many arguments as you want as long as they follow the rules that I have built in (groups of 5 arguments, see above). If you want to know how this hack is done, take a look at the code!

We can do it like this:

```
superCollider(
      'group',    //collide by
      'bullet',   //objectA
      'enemy',    //objectB
       null,      //object to destroy
      'hitEnemy', //function to run

      'id',       //collide by
      'player',   //objectA
      'door',     //objectB
       null,      //object to destroy
      'opendoor', //function to run

      'group',    //collide by
      'bullet',   //objectA
      'wall',     //objectB
      'wall',     //object to destroy
      'explode'   //function to run
);
```

NOTE: The very last argument does NOT have a comma after it! ALL other items do have.

Experimental functions

The following functions are experimental! That is to say, I have not thought very hard about designing them, nor is testing fully complete! They are just quick hacks!

Simple timer

This can be used to spawn objects using a timer.

It takes 3 arguments:

- Delay
- Counter start number
- Name of a function to run

You must declare a function to run FIRST! Perhaps like this one:

```
var ball_count = 0;

function ballfunction(ball_count){
	if(ball_count < 1000){
		dynamicCircle('ball',
			'player'+ball_count,
			50,
			20,
			15,
			0,
			false,
			0.2,friction,restitution,
			null);
		ball_count++;
	};
};
```

Then call the clock like this:

```
clock(700,0,'ballfunction');
```

Static polygons

Currently, this works!

It may behave strangely when objects interact with it! This is a Box2d issue, not a gift-wrap issue.

If you want complex terrain, you are better creating one out of static boxes or static circles.

Good Luck! ☺

```
staticPolygon('rhombus',        //class
      'rhombus',                //id
      250,                      //X pos
      150,                      //Y Pos
      [
            {x: 0, y: 0},       //vectors
            {x: 100, y: 0},
            {x: 0, y:400},
            {x: -1, y:200}
      ],
      null                      //asset
);

staticPolygon('triangle',       //class
      'triangle',               //id
      250,                      //X pos
      150,                      //Y pos
      [
      {x: 0, y: 0},             //vectors
      {x: 100, y: 0},
      {x: 0, y:200}
      ],
      null                      //asset
);
```

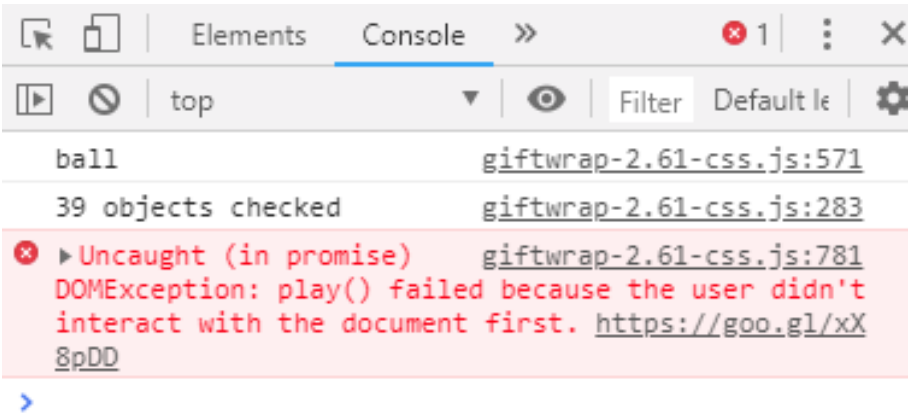
TODO

1. Add SENSORS (Non collidable objects)
    a. fixtureDef.isSensor = true;
    b. fixtureDef.filter.maskBits = 0xffff;
2. Properly test clock()
3. ~~Find a solution for CSS for the staticPolygon() *~~Done!
4. Add a function to getAngle() ball.getAngle returns radians
5. ~~Refactor the code. After the latest iteration, there are a number of for loops which are now redundant, and just eating clock cycles.~~ Done!
6. Implement impact strength (add property to object. During collision we can set this property, after collision we can get this property…) **In progress….
7. ~~Continue work on welds~~ Done!
8. ~~Perhaps introduce motors~~Done!

Known Issues

Latest versions of browsers, will no longer auto play media unless the user has interacted with the document!

So if you call playSound() for example before the user has interacted with the document (perhaps to play a background tune), you may end up with the following console error:



https://developers.google.com/web/updates/2017/09/autoplay-policy-changes

If you want background music, you can either:

- Add a button
- Wait for the user to press a key or move the mouse

How you program that is up to you!