How to Make a Slingshot Game with Matter.js

Getting Started (creating the pen)

Head over to codepen.io and start a brand new pen. We will need to add the matter.js library which can be found on matter.js website. To add the library, we need to click on the settings button and under “js”, we will find the place to add the library. We can add the library by entering in this url under the “Add External Scripts/Pens”. Don’t forget to save and close when done. Now that the pen is ready, we can start adding the code.

Starting code for engine

Now that the pen is ready, we can start adding the code. First, lets create a function which will run the code for the slingshot. Next, we will add the variables. Below is the list of all the variables needed to be defined. For each variable, set it equal to “Matter.” and adding the name of the variable after the dot. Below are the variables that need to be defined.

Engine, Render, Runner, Bodies, Body, Events, Composite, Composites, Constraint, Common, MouseConstraint, Mouse, Vector, Vertices

Starting code for engine

After creating the variables, we need to create an engine. To create the engine, we need to create a variable called “engine” and set it equal to “Engine.create();”. We also need a create another variable called “world” and set it equal to “engine.world;”. Next, we need to add the renderer and can do so by creating an variable called “render” and will set this variable equal to “Render.create({“. Inside the brackets, we will set “element” equal to “document.body” and “engine” equal to “engine”. Don’t forget to close the bracket and the parentheses.

Starting code for engine (what it should look like)

```
function start(){

    var Engine = Matter.Engine,
    Render = Matter.Render,
    Runner = Matter.Runner,
    Bodies = Matter.Bodies,
    Body = Matter.Body,
    Events = Matter.Events,
    Composite = Matter.Composite,
    Composites = Matter.Composites,
    Constraint = Matter.Constraint,
    MouseConstraint = Matter.MouseConstraint,
    Mouse = Matter.Mouse;
```

```
// create an engine
var engine = Engine.create();
var world = engine.world;

// create a renderer
var render = Render.create({
    element: document.body,
    engine: engine
});
```

Adding the obstacles and slingshot

Now that the engine and the renderer have been created, we can start adding the obstacles and slingshot. We will create variables for each of the obstacles. For this example, we will add four, but feel free to add as many obstacles needed for your game. Let's start with the first obstacle by creating a variable called "boxA" and setting it equal to "Bodies.rectangle(" with the inside containing number values for the x-coordinate, y-coordinate, length, and width. After adding the width, add a bracket and set "isStatic: true". In the end, the line of code should look something like this, "Bodies.rectangle(105, 448, 80, 210 { isStatic: true })". We will follow these steps again for the next variables. Don't forget to change the name of the variable each time.

Adding the obstacles

We can also add a ground by creating a variable called “ground” and setting it equal to “Bodies.rectangle(0, 0, ” with the rest of the number values representing the length of canvas, and width from 50-100. After adding the width, add a bracket and set “isStatic: true” so that the ground doesn’t move around. In the end, the line of code should look something like this, “Bodies.rectangle(0, 0, 600, 100 { isStatic: true })”. We can also do the same thing to make walls, but what is important to remember is that the objects that are launched from the slingshot will start to pile up and may take up space within the game.

Adding the obstacles (what it should look like)

```
// create two boxes and a ground
var boxA = Bodies.rectangle(105, 250, 80, 200, { isStatic: true });
var boxB = Bodies.rectangle(440, 320, 80, 80, { isStatic: true });
var boxC = Bodies.rectangle(240, 460, 80, 80, { isStatic: true });
var boxD = Bodies.rectangle(200, 80, 120, 80, { isStatic: true });
var ground = Bodies.rectangle(400, 610, 810, 60, { isStatic: true });
var ground2 = Bodies.rectangle(420, 250, 260, 20, { isStatic: true });
```

Adding the slingshot

Now we will add the slingshot. For the slingshot, we will need to add a couple of variables. First we will add the variable called “rock” and set it equal to the x-coordinate and the y-coordinate of the slingshot. Then we will add the variable called “rockOptions” and we will set it equal to density with a value of 0.004. The next variable we will add will be called “rock” and this will be the object that gets launched with the slingshot. We will set this variable equal to “Bodies.polygon(” and with the inside containing number values for the x-coordinate, y-coordinate, number of sides, size and density which will equal “rockOptions”. Feel free to add as many sides you want or whatever size you prefer. The x-coordinate and y-coordinate will determine the position in where the slingshot will appear.

Adding the slingshot

The next variable will be called “anchor” and set it equal to an “x” and “y” value. The x and y values should be the same as the x-coordinate and y-coordinate of “rock”. For the last variable for the slingshot, we will call it “elastic” and we will set it equal to “Constraint.create({”. Inside it, we will set “pointA” equal to “anchor” and set “bodyB” equal to “rock”. We will also set the “stiffness” equal to “0.05”. The last variable we will add will be for the targets for the slingshot. We will call the variable “pyramid” and set it equal to “Composites.pyramid(“ with the inside containing number values for the x-coordinate, y-coordinate, number of columns, number of rows, column gap size, row gap size, and the callback function. In the callback function, add the parameters, “x, y” and inside the function we will “return Bodies.rectangle(x, y” and also add the length and width of the rectangle. In the end the line of code should look something like this:

```
- var pyramid = Composites.pyramid(300, 50, 10, 20, 0, 0, function(x, y){
-   return Bodies.rectangle(x, y, 25, 25)
- })
```

Adding the obstacles (what it should look like)

```
var rockOptions = { density: 0.004 }
var rock = Bodies.polygon(rock.x, rock.y, 8, 30, rockOptions)
var anchor = {x: 105, y: 450}
var elastic = Constraint.create({
    pointA: anchor,
    bodyB: rock,
    stiffness: 0.05
});

var pyramid = Composites.pyramid(300, 50, 10, 20, 0, 0, function(x, y){
  return Bodies.rectangle(x, y, 25, 25)
})
```

Setting up the slingshot

Before we set up the slingshot, we will need to add all our variables into the engine. We can do this by writing the code, “Composite.add(engine.world, [boxA, box,B, boxC, boxD, ground, rock, elastic, pyramid]);”. Now we will add the code needed to make this slingshot shoot “rock”. To do this, we need to write the following piece of code:

“Events.on(engine, ‘afterUpdate’, function() {

if(mouseConstraint.mouse.button === -1 && (rock.position.x > 425 || rock.position.y < 400)){

rock = Bodies.polygon(405, 420, 7, 30, rockOptions);

Composite.add(engine.world, rock);

elastic.bodyB = rock

}

Setting up the slingshot (what it should look like)

```
// add all of the bodies to the world
Composite.add(engine.world, [boxA, boxB, boxC, boxD, ground, ground2,
rock, elastic, pyramid]);

   Events.on(engine, 'afterUpdate', function() {
     if (mouseConstraint.mouse.button === -1 && (rock.position.x > 125 ||
rock.position.y < 430)){
       rock = Bodies.polygon(105, 450, 7, 30, rockOptions);
       Composite.add(engine.world, rock);
       elastic.bodyB = rock
       pentagonCounter += 1
       }
     })
```

Adding Mouse Controls

Now we will add the mouse controls to the game. We need to create a variable called “mouse” and set it equal to “mouse.create(render.canvas). We also need to add another variable called “mouseConstraint” and set it equal to “mouseConstraint.create(engine, {”. Inside the brackets, we will set “mouse” equal to “mouse”. We will also set “constraint” equal to “stiffness”, which will be equal to 0.2, and “render”, which will be equal to “visible” set equal to “false”. Next we will add, “Composite.add(world, mouseConstraint);”. To keep mouse in sync with rendering, we will set “render.mouse” equal to “mouse”.

Adding Mouse Controls (what it should look like)

```
// add mouse control
var mouse = Mouse.create(render.canvas),
    mouseConstraint = MouseConstraint.create(engine, {
        mouse: mouse,
        constraint: {
            stiffness: 0.2,
            render: {
                visible: false,
            }
        }
    });

Composite.add(world, mouseConstraint);

// keep the mouse in sync with rendering
render.mouse = mouse;
```

Finishing Touches

To make the obstacles bouncy, that way objects can bounce off them, we will need to set the restitution value for each obstacle. We can do this by taking one of the obstacles, we will use boxA for the first one, and set the restitution value of boxA equal to 1.5. The line of code should look something like this, “boxA.restitution = 1.5”. You can make the set value equal to any number from 0 to 2. Anything above two will make it way too bouncy and cause problems for your game. We can use the above steps for the rest of the obstacles. We also can change the appearance of the game and take out the wire framework by using the code “render.options.wireframes = false”. This will allow you to add color to your game.

Creating the Runner

Now that the game is done, we can run the renderer. We can do this by using the line of code, “Render.run(render);”. We will also need to create the runner by creating a variable called runner and setting it equal “Runner.create();”. Finally, we will run the engine by using the line of code “Runner.run(runner, engine);”. Now for the last and final step, we will call the function to display the game on the screen.

The game is now complete and we can start testing it out. Feel free to click on this link if need more reference for the code.

Finishing Touches Creating the Runner (what it should look like)

```
boxA.restitution = 1.5
boxB.restitution = 1.5
boxC.restitution = 2
boxD.restitution = 1.5
ground2.restitution = 1.5
render.options.wireframes = false
```

```
// run the renderer
Render.run(render);

// create runner
var runner = Runner.create();

// run the engine
Runner.run(runner, engine);
```

Thanks for Watching!