

# Build a 2D Game with Javascript

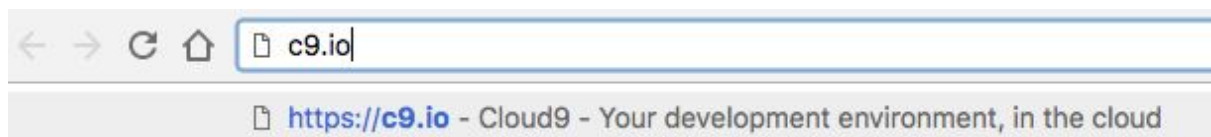
Source: (kode.farm Computer School - Computer programming courses for 5 to 16 year olds in Twickenham - [www.kode.farm](http://www.kode.farm))

In this activity you will learn how to:

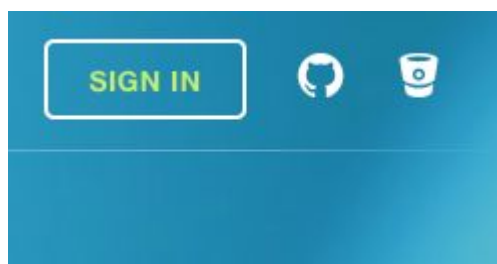
- **Build a complete 2D game (Breakout) from scratch using Javascript;**
- **Listen and react to user interaction in your code;**
- **Using a physics engine to simulate physical behaviour; and**
- **Publish your game to the web for free using GitHub**

## Step 1: Open your online code editor

Type `c9.io` into your browser address bar:



Now click on Sign in:





Enter the 1) username and 2) password that were handed to you:

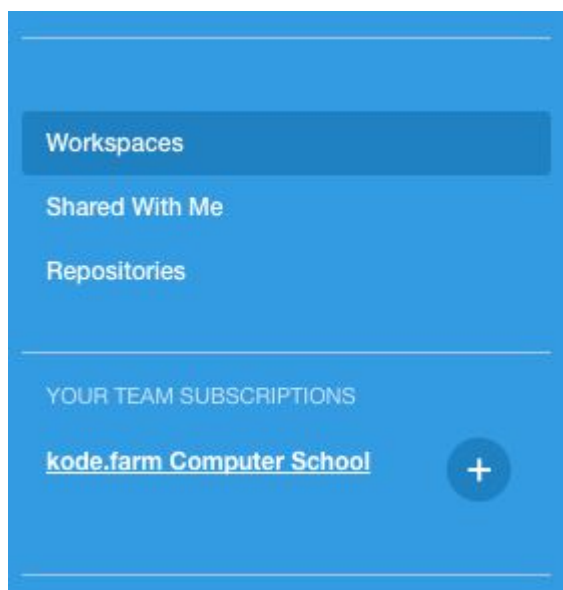
## Sign in to Cloud9

Sign in

Forgot password? - Didn't receive activation email?



On the left-hand side, click on `kode.farm` Computer School:



Select the workspace that has been assigned to you by clicking on its related 'Open' button.

Updated a day ago.

Clone

Open

1 CPU

512MB RAM

2GB HDD

Your code editor will open in a new tab.

## Step 2: Set up the game

Double-click the file called index.html on the left-hand side to open it. Look at how we structure the game:

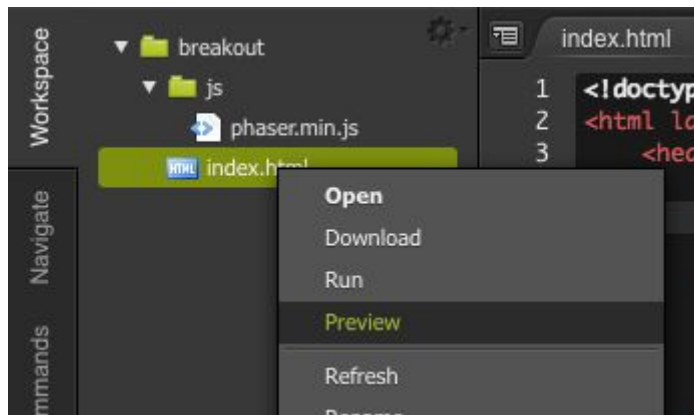
```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>code.farm Breakout Game Challenge</title>
6   </head>
7   <body>
8     <script type="text/javascript" src="js/phaser.min.js"></script>
9     <script type="text/javascript">
10
11      // Create the state that will contain the whole game
12      var mainState = {
13        preload: function() {
14          // Here we preload the assets
15        },
16        create: function() {
17          // Here we create the game
18        },
19        update: function() {
20          // Here we update the game 60 times per second
21        },
22      };
23
24      // Initialize the game and start our state
25      var game = new Phaser.Game(400, 450);
26      game.state.add('main', mainState);
27      game.state.start('main');
28
29    </script>
30  </body>
31 </html>
```

Look at the numbers shown on the image and read the matching explanation here:

- 1) All the files and folders are listed here
- 2) We wrap our game in an HTML page. The HTML page has a `<head>` tag containing the title of the page and some styling, and a `<body>` tag containing 2 script tags: one for our game engine, Phaser, and one for our own script which you are about to write. The basic structure is already there to get you started quickly.
- 3) In our activity today, we will create 3 functions: preload, create and update. The preload function will load everything we need to use in the game, for example our images.

- 4) The create function sets up all the elements in our game so that they behave correctly.
- 5) This function is called many times a second like an infinite loop throughout the game, and will change the values of everything as needed and display it correctly.
- 6) This part sets up the main game, its size and tells the browser to start the game.

To preview the game, right-click on the index.html file in the left-hand side panel, and select 'Preview'.



Do that now, and you'll notice your game at the moment is nothing but a big, black rectangle. This is about to change. Let's change the colour of the background:

Inside your create function, add the following line:

```
game.stage.backgroundColor = '#3598db';
```

```
create: function() {  
    // Here we create the game  
    game.stage.backgroundColor = '#3598db';  
},
```

Now, save your work by pressing Ctrl + S, and then preview it again.

Now, let's enable physics in our game! We do this with 2 more lines inside the create function:

```
game.physics.startSystem(Phaser.Physics.ARCADE);  
game.world.enableBody = true;
```

```
create: function() {
  // Here we create the game
  game.stage.backgroundColor = '#3598db';
  game.physics.startSystem(Phaser.Physics.ARCADE);
  game.world.enableBody = true;
},
```

You won't see any changes here when you preview now, because the physics is there, but *no objects* have been added to the world yet.

## Step 3: Add the paddle

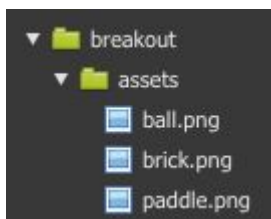
The paddle is the bar at the bottom that can be moved left and right by pressing the arrow keys in order to hit the ball.

First, we preload it. Add the following line to your preload function:

```
game.load.image('paddle', 'assets/paddle.png');
```

```
preload: function() {
  // Here we preload the assets
  game.load.image('paddle', 'assets/paddle.png');
},
```

You will see a folder called **assets** containing images:



That's why, in that line, we describe the *path* to the image from the game:

```
'assets/paddle.png'
```

Next, we create objects that listen to key presses on the left and right arrow keys. Add these two lines to your *create* function:

```
this.left = game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
this.right = game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);
```

```

create: function() {
  // Here we create the game
  game.stage.backgroundColor = '#3598db';
  game.physics.startSystem(Phaser.Physics.ARCADE);
  game.world.enableBody = true;

  // Create the left/right arrow keys
  this.left = game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
  this.right = game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);
}

```

You'll see very shortly how we'll use these objects in the update function.

We add it to the world with this line inside the create function:

```
this.paddle = game.add.sprite(200, 400, 'paddle');
```

```

create: function() {
  // Here we create the game
  game.stage.backgroundColor = '#3598db';
  game.physics.startSystem(Phaser.Physics.ARCADE);
  game.world.enableBody = true;

  // Create the left/right arrow keys
  this.left = game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
  this.right = game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);

  // Add the paddle at the bottom of the screen
  this.paddle = game.add.sprite(200, 400, 'paddle');
}

```

The 200 stands for 200 pixels from the left edge (or on the x-axis), and 400 from the top (y-axis).

We want to prevent the paddle from shifting (due to the physical impact) when the ball hits it with this line:

```
this.paddle.body.immovable = true;
```

And then, we want to prevent it from leaving the game's edges with this line:

```
this.paddle.body.collideWorldBounds = true;
```

```

create: function() {
  // Here we create the game
  game.stage.backgroundColor = '#3598db';
  game.physics.startSystem(Phaser.Physics.ARCADE);
  game.world.enableBody = true;

  // Create the left/right arrow keys
  this.left = game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
  this.right = game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);

  // Add the paddle at the bottom of the screen
  this.paddle = game.add.sprite(200, 400, 'paddle');

  // Make sure the paddle won't move when it hits the ball
  this.paddle.body.immovable = true;
  this.paddle.body.collideWorldBounds = true;
},

```

When you preview it now, you will see a paddle, but it doesn't move! Let's make it move. In the update function, we now add the following lines:

```

if (this.left.isDown)
    this.paddle.body.velocity.x = -300;
else if (this.right.isDown)
    this.paddle.body.velocity.x = 300;
else
    this.paddle.body.velocity.x = 0;

```

So here we see what we do with the left and right key objects that we created earlier.

These lines above are an example of an if/else statement. It's very easy to see what it does:

```

if (some condition)
    Then do this
else if (another condition)
    Then do this
else
    None of the previous conditions were met, rather do this

```

In this case, the condition was: if the left arrow key is being held down, etc.



```

update: function() {
    // Here we update the game 60 times per second
    // Move the paddle left/right when an arrow key is pressed
    if (this.left.isDown)
        this.paddle.body.velocity.x = -300;
    else if (this.right.isDown)
        this.paddle.body.velocity.x = 300;
    // Stop the paddle when no key is pressed
    else
        this.paddle.body.velocity.x = 0;
}

```

## Step 4: Add the bricks

In this step, we add the bricks that will be hit by the ball. We will add 25 of them. First, we need to preload the asset that we will be using. Add the following line to the preload function:

```
game.load.image('brick', 'assets/brick.png');
```

```

preload: function() {
    // Here we preload the assets
    game.load.image('paddle', 'assets/paddle.png');
    game.load.image('brick', 'assets/brick.png');
},

```

Now, in the create function, we create a group to keep all the bricks in:

```
this.bricks = game.add.group();
```

Then, we add the bricks into the group, spacing them out in 5 rows like this:

```

for (var i = 0; i < 5; i++) {
    for (var j = 0; j < 5; j++) {
        var brick = game.add.sprite(55+i*60, 55+j*35, 'brick');
        brick.body.immovable = true;
        this.bricks.add(brick);
    }
}

```

The above is an example of a *for loop*. 2 *for loops* in fact, the one inside the other.

A for loop works like this:

```
for(1. initialise a variable; 2. the loop condition; 3. what happens after every iteration)
{
    Do this every time until the stop condition is met.
}
```

In our first case, we initialise the variable like this:

```
var i = 0;
```

This means, we now have a container called *i* and its value is 0 the first time the loop runs.

We run this loop until *i* is not less than 5 anymore, because:

```
i < 5;
```

is our loop condition. And every time the loop is done, we increase *i*'s value with 1:

```
i++
```

which is shorthand for *i = i + 1*;

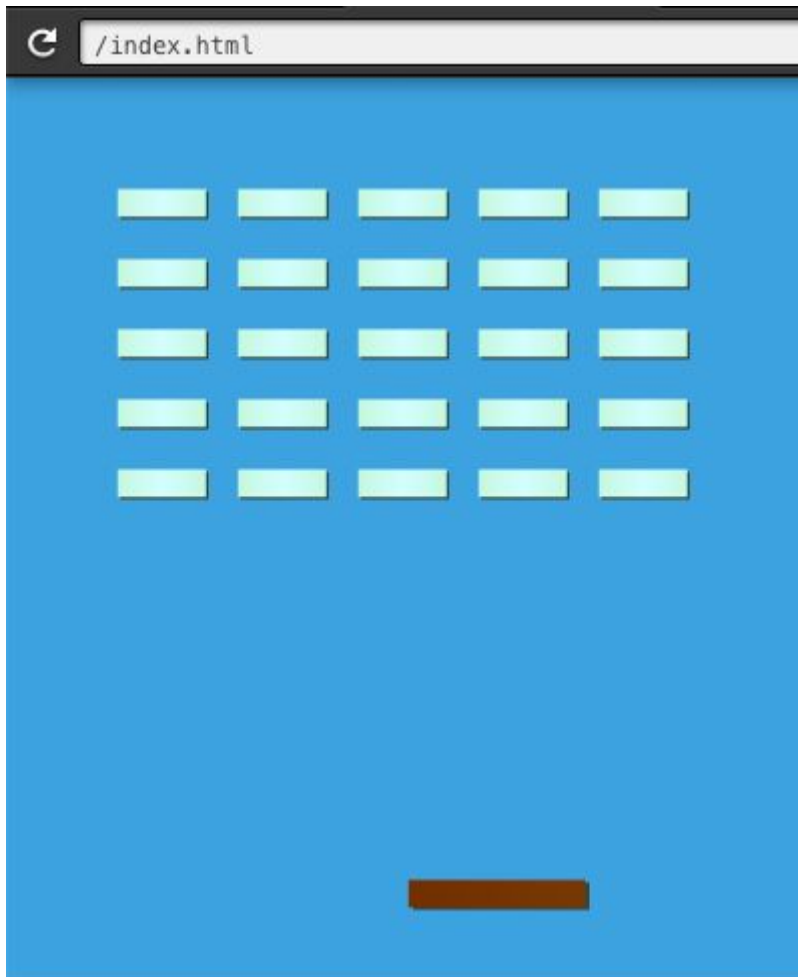
```
this.bricks = game.add.group();

for (var i = 0; i < 5; i++) {
    for (var j = 0; j < 5; j++) {
        // Create the brick at the correct position
        var brick = game.add.sprite(55+i*60, 55+j*35, 'brick');

        // Make sure the brick won't move when the ball hits it
        brick.body.immovable = true;

        // Add the brick to the group
        this.bricks.add(brick);
    }
}
```

Preview your work! You should see the following at the moment:



You'll notice the paddle moving left and right when you press the left and right arrow keys, and the bricks are there, waiting to be hit by the ball. So, that's exactly what we will add next!

## Step 5: Add the ball

Once again, inside the preload function, we preload the asset we will use for the ball first:

```
game.load.image('ball', 'assets/ball.png');
```

```
preload: function() {  
    // Here we preload the assets  
    game.load.image('paddle', 'assets/paddle.png');  
    game.load.image('brick', 'assets/brick.png');  
    game.load.image('ball', 'assets/ball.png');  
},
```

In the create function, we add the ball to the world with this line:

```
this.ball = game.add.sprite(200, 300, 'ball');
```

We give the ball some speed, or velocity:

```
this.ball.body.velocity.x = 200;  
this.ball.body.velocity.y = 200;
```

The x and the y mean that we add speed to the ball going left and right and also up and down.

Next, we give the ball it's bounciness! Add this line:

```
this.ball.body.bounce.setTo(1);
```

And finally, we tell it not to bounce so much that it leaves the game edges:

```
this.ball.body.collideWorldBounds = true;
```

```
// Add the ball  
this.ball = game.add.sprite(200, 300, 'ball');  
  
// Give the ball some initial speed  
this.ball.body.velocity.x = 200;  
this.ball.body.velocity.y = 200;  
  
// Make sure the ball will bounce when hitting something  
this.ball.body.bounce.setTo(1);  
this.ball.body.collideWorldBounds = true;
```

If you preview your game now, you will see a moving ball, rows and columns of bricks and a moving paddle. But there is one thing wrong, isn't there? Do you know what it is missing?

This is what we call *collision handling* in game development. So let's handle those collisions.

## Step 6: Handle collisions

Let's first prevent the ball from passing through the paddle, and instead bounce off it. Add this line to the update function:

```
game.physics.arcade.collide(this.paddle, this.ball);
```

```
update: function() {
    // Here we update the game 60 times per second
    // Move the paddle left/right when an arrow key is pressed
    if (this.left.isDown)
        this.paddle.body.velocity.x = -300;
    else if (this.right.isDown)
        this.paddle.body.velocity.x = 300;
    // Stop the paddle when no key is pressed
    else
        this.paddle.body.velocity.x = 0;

    // Add collisions between the paddle and the ball
    game.physics.arcade.collide(this.paddle, this.ball);
}
```

Next, we handle the collision between the bricks and the ball, and this takes two steps. First, add this line (also in the update function):

```
game.physics.arcade.collide(this.ball, this.bricks, this.hit, null,
this);
```

And then we create our own function below the update function:

```
hit: function(ball, brick) {
    brick.kill();
}
```

```

update: function() {
    // Here we update the game 60 times per second
    // Move the paddle left/right when an arrow key is pressed
    if (this.left.isDown)
        this.paddle.body.velocity.x = -300;
    else if (this.right.isDown)
        this.paddle.body.velocity.x = 300;
    // Stop the paddle when no key is pressed
    else
        this.paddle.body.velocity.x = 0;

    // Add collisions between the paddle and the ball
    game.physics.arcade.collide(this.paddle, this.ball);

    // Call the 'hit' function when the ball hits a brick
    game.physics.arcade.collide(this.ball, this.bricks, this.hit, null, this);
},

hit: function(ball, brick) {
    brick.kill();
}

```

Notice the comma after the update function:

```

update: function() {
    ...
},

hit: function(ball, brick) {
    ...
}

```

The very final step is to start the game over when the ball passes below the paddle (that is, when you miss the ball). In the update function, add the following:

```

if (this.ball.y > this.paddle.y)
    game.state.start('main');

```

That's it! Your game is now complete. Play it and give your friends a go!