**Lesson 5. SVG Racing (4 Lessons)**

Objective

Make a racing game using SVG graphics and library.

Contents

[SVG.js library 1](#_Toc72600318)

[Training 2](#_Toc72600319)

[Drawing game elements 3](#_Toc72600320)

[Game logic 5](#_Toc72600321)

What we repeat

1. SVG coordinates
2. JS methods for working with attributes (setAttribute, getAttribute)
3. Variables and Functions
4. Timer operation

What's new

Links to materials and personal account

[Working materials](https://hwschool.bitrix24.ru/bitrix/tools/disk/focus.php?folderId=201389&action=openFolderList&ncc=1)(for the teacher).

[Materials (edit)](https://hwschool.bitrix24.ru/~2VRmn)(wesend this link to the student at the beginning of the lesson).

[Video presentation of the finished project](https://youtu.be/GxIdI5zGpGg)

Methodical material

SVG.js library

Today you and I will create a Race game using the library we met in the last lesson

**Website - the management of our library (in Russian):**

<https://customizer.github.io/svg.js-ru/svg.js-ru.htm>

Let's start by creating a new HTML document called index.html and a script.js file.

**<! DOCTYPE html>**

**<html>**

**<head>**

**<title> Racing game </title>**

**</head>**

**<body>**

**<script src = "script.js"> </script>**

**</body>**

**</html>**

And first of all, let's link our library to our HTML document. Do you remember how to do it?

Here is the link to the library file I took from the manual site:

<https://github.com/svgdotjs/svg.js/releases/download/3.0.12/svg.js.zip>

We have now downloaded the ZIP archive with the script of our library. Let's unpack the archive and move the svg.js file to our new project folder (where index.html and script.js are).

Or we can use a CDN and just add one of the links to the library file from this site:

<https://cdnjs.com/libraries/svg.js>

And now we just add a new script tag and specify the name of our library file.

**<! DOCTYPE html>**

**<html>**

**<head>**

**<title> Racing game </title>**

**<script src = "svg.js"> </script>**

**</head>**

**<body>**

**<script src = "script.js"> </script>**

**</body>**

**</html>**

We can add it to the **HEAD** tag so that our library is loaded immediately even before the game itself and our file with js code start loading.

Option 2 (when we use CDN)

**<script src = "https://cdnjs.cloudflare.com/ajax/libs/svg.js/3.0.16/svg.min.js"> </script>**

Training

Let's create an empty <**DIV**> that will be used as a container (box) for our **SVG** element. And we will assign it an ID attribute so that later it can be easily found in our document (DOM). Something like **GAME** should work for our project. {PS we are making a game here with you :)}

**<body>**

**<div id = "game"> </div>**

**<script src = "script.js"> </script>**

**</body>**

And let's go to our script - **script.js**.

Let's create two variables (constants) where we will write the width and height of our game window. We will use the **CONST** command, because the width and height of our game will not change (when we play toys on a laptop or phone, you and I do not change the size of the game window, so it will not change in our game either, so we will immediately use a command that will not allow us to change (even accidentally) our sizes while we play).

**Script.js** file

**const WIDTH = 480;**

**const HEIGHT = 360;**

**P.S.** We can use any numbers, but in the beginning, it is better to set values ​​less than 500 to make it easier to customize the game logic and make calculations for collision conditions, etc. Those values ​​will fit perfectly in the beginning. And in the end, we can easily change them and make the game larger (full screen or at the request of the student) - we can leave it as homework.

We do not write the units of measurement, since we use the default pixels.

Next, we use the SVG() function to wrap the SVG document (to create the **svg** tag):

**const WIDTH = 480;**

**const HEIGHT = 360;**

**let draw = SVG (). addTo ('# game'). size (WIDTH, HEIGHT);**

Then we say that we add our **SVG** tag inside our **DIV** element with **id game**, where the argument is the **id** of the element, or the selected element itself. And immediately apply the **SIZE** method to set the dimensions of our **SVG** document. And we will assign all this beauty to the **DRAW** variable, so that it will be more convenient to add our game elements to the **SVG** tag further.

Now we can start creating).

Drawing game elements

**1. Background**

The background should cover the entire document, so we draw a rectangle over the entire playing field (window), use **<rect>** and paint it a neutral grayish color.

**let draw = SVG ('game'). size (WIDTH, HEIGHT);**

**let background = draw.rect (WIDTH, HEIGHT) .fill ('# dde3e1');**

**2. Markup**

We will also need a vertical dashed line in the middle to distinguish between the two sides of the road and side lines to indicate the boundaries of the road.

**let borderLeft = draw.line (30, 0, 30, HEIGHT) .stroke ({**

**width: 10,**

**color: "black"**

**});**

**let borderRight = draw.line (450, 0, 450, HEIGHT) .stroke ({**

**width: 10,**

**color: "black"**

**});**

**let borderMiddle = draw.line (WIDTH / 2, 0, WIDTH / 2, HEIGHT) .stroke ({**

**width: 5,**

**color: "black",**

**dasharray: [10, 10]**

**});**

**3. Machine and obstacle**

Next, we draw a car - send the student a link to the materials for the lesson (if not done at the beginning of the lesson) and save the pictures for the game.

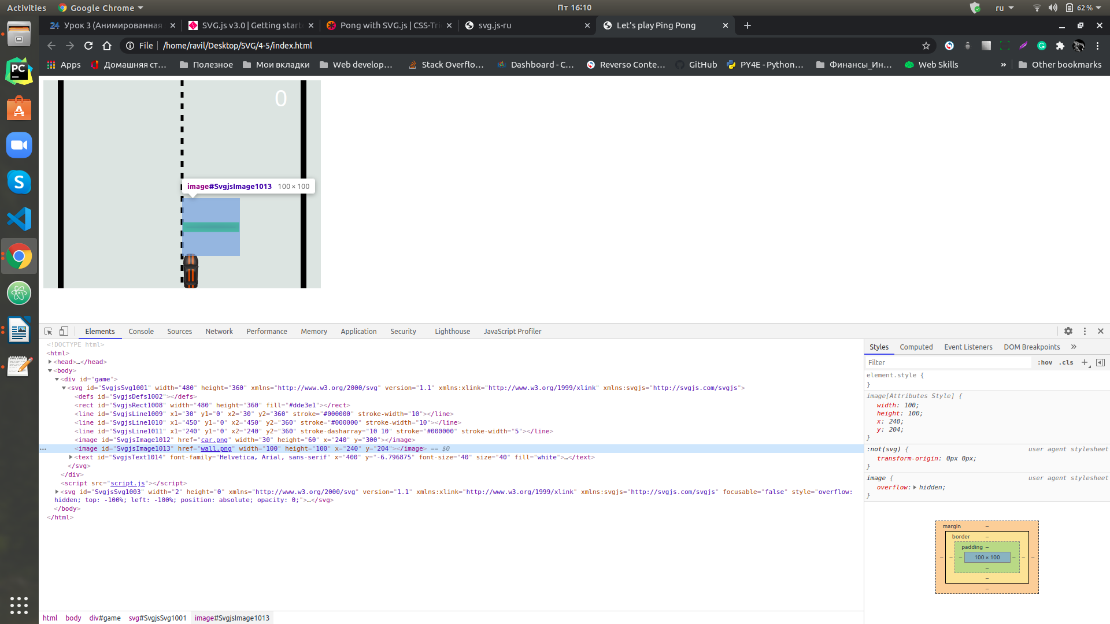
**let car = draw.image ('car.png'). size (30, 60) .move (WIDTH / 2, 300);**

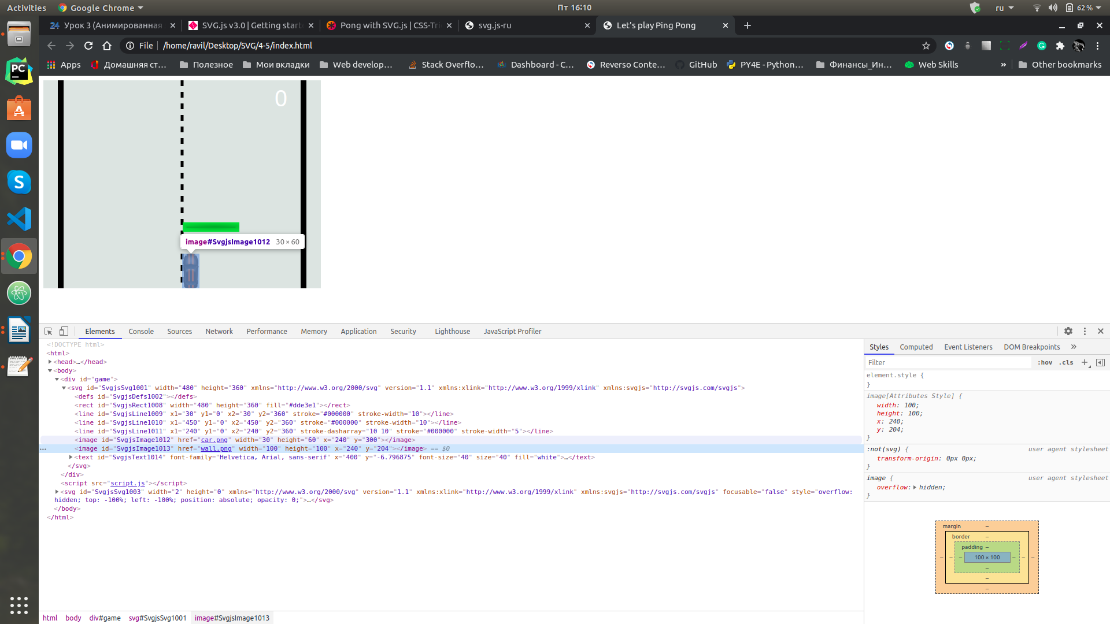
**let wall = draw.image ('wall.png'). size (106, 20) .move (WIDTH / 2, 100);**

!!! Why 30 and 60, or 106 and 20 ??? - depends on the size of the original (at the beginning, we can open the properties of the image in the folder and see together with the student what sizes of pictures are (in pixels, of course)). We can first set it as in the properties - we saw that the big ones turned out, we reduce it proportionally - we divide the first and second numbers by 2, for example.

Why can't we choose any sizes???

If we choose any other size - violating the proportions (in the developer's console we can highlight the picture, or rather what will become of it if we make the dimensions at random, without preserving the proportions of the original), then it will look like this:





The picture will take up more space than it actually is, and then it will be much more difficult to make a collision condition in the game itself (remember when we were making ball hitting in Ping Pong).

**4. Scoreboard**

Finally, we need a scoreboard, which we will add at the top of the box.

**let text = draw.text ("0"). move (400, 0) .font ({**

**size: 40**

**}). fill ("white");**

That's all! Now that we have all the game elements, let's move on to the game logic.

Game logic

We will start by writing an update function that will constantly update the state of our game and game elements at a certain interval of time (it will show the new location of our pictures at a new point in time).

**function update () {**

**}**

**setInterval (update, 10);**

**1. Animation of the road**

And let's first make the line in the middle move (scroll) and create the effect that our car is constantly moving forward (although it actually stands still).

**function update () {**

**borderMiddle.dy (10);**

**}**

**setInterval (update, 10);**

We see that the line ran down and disappeared. Let's make it so that when it moves down a little, then it immediately comes back again - it turns out that it runs here and there, but with our small interval everything will happen very quickly, and it will seem that the road is going down (as if the car is going forward - up).

**function update () {**

**if (borderMiddle.y () === 0) {**

**borderMiddle.dy (10);**

**} else {**

**borderMiddle.dy (-10);**

**}**

**}**

**setInterval (update, 10);**

Hurray, it works - it looks like the car is always going forward and the road does not end.

**2. Car control**

Next, let's make it so that the car can move left and right when you click on the arrows.

First, inside our function, we add a command that will move our car along the X axis to the right or left. If we put some number in the command, then the car will move (a positive number - to the right, with a minus - to the left).

**let stepCar = 0;**

**function update () {**

**car.dx (stepCar);**

But, after all, you and I want to move the car in different directions, so let's create a variable so that we can change it later when we press the buttons. At the beginning, we put 0 in the variable, since the car at the beginning of the game must be stationary. And then, when we click on the buttons, we will set new values ​​for our variable.

**// The code above is unchanged**

**setInterval (update, 10);**

**document.addEventListener ("keydown", function (event) {**

**if (event.keyCode == 37) {**

**stepCar = -2;**

**} else if (event.keyCode == 39) {**

**stepCar = 2;**

**}**

**});**

Is everything okay with our car? When we release the buttons, does it drive smoothly? Nope. Let's fix it. What do we need to do? How do we want it to be controlled? Yes, let’s make it so that when we release the buttons, the car will not move to the right or left, but go straight.

Add another event - **keyup**, which is triggered when the key is released (raised up)

**document.addEventListener ("keyup", function (event) {**

**if (event.keyCode == 37 || event.keyCode == 39) {**

**stepCar = 0;**

**}**

**});**

Hurray, so much better. What do you think about it?

**3. Obstacle movement.**

Now let's set up our obstacle. Let's put the racket towards our car (like a ping pong ball). Let the racket fall from top to bottom - as if we are approaching it and driving along the road.

Add the command to the **update** function - change the height (**Y**) of the paddle

**wall.dy (2);**

And let's bring it back upstairs when it has already flown through our entire game window.

**wall.dy (2);**

**if (wall.y ()> = HEIGHT) {**

**wall.y (-wall.height ());**

**}**

And to make it more interesting to play, we will make it appear in a random place at the top.

**wall.dy (2);**

**if (wall.y ()> = HEIGHT) {**

**let newPlace = getRandomInt (35, WIDTH - wall.width () - 35);**

**// 35 - so that it is not behind the road, but inside - between the side lines - to show separately, at first it is simply possible from 0 to WIDTH, and then we change the values ​​when we see a problem**

**wall.x (newPlace);**

**wall.y (-wall.height ());**

**}**

**}**

**setInterval (update, 10);**

**function getRandomInt (min, max) {**

**return Math.floor (Math.random () \* (max - min + 1)) + min;**

**}**

There are a couple of touches left: first we make it so that when we hit the racket, the game stops.

**car.dx (stepCar);**

**let collision = car.x () + car.width ()> wall.x () && car.x () <wall.x () + wall.width () && car.y () <wall.y () + wall.height ();**

**if (collision) {**

**clearInterval (update\_id);**

**}**

**…………………………….**

**}**

**let update\_id = setInterval (update, 10);**

And the last thing - we do the SCORE.

**let stepCar = 0;**

**let score = 0;**

**function update () {**

**…………………………**

**if (wall.y ()> = HEIGHT) {**

**score + = 1;**

**text.text ("" + score);**

**// add quotes to pass text, not a number - otherwise it won't work, or String (value)**

**////////////////**

**wall.y (-wall.height ());**

**}**

The Race game is ready.