**Lesson 6. SVG. Dino (2 lessons)**

Objective

Make a Dinosaur game using SVG graphics and library

Contents

[Preparing the project structure 1](#_Toc72749828)

[Drawing game elements 2](#_Toc72749829)

[Game logic 3](#_Toc72749830)

[**Homework** 10](#_Toc72749831)

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=201293&action=openFolderList&ncc=1)(for the teacher).

[Materials (edit)](https://hwschool.bitrix24.ru/~PgW58)(we send this link to the student at the beginning of the lesson).

[Video presentation of the finished project](https://youtu.be/YLkd0aIKT6o)

Methodical material

Preparing the project structure

Today we will consolidate the skills of using the **SVG.js** library and make a Dinosaur game (similar to the **Google Dinosaur game**). And we will learn how to create animation for our heroes.

First, let's prepare the project structure. Please create a new folder and files: **index.html**, **script.js** and connect our library to our **HTML** document. Do you remember how we do it? (We connect before loading our main script so that all the commands and functions of the library are already ready when we use them in our script)

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

**<! DOCTYPE html>**

**<html>**

**<head>**

**<meta charset = "utf-8">**

**<title> Dino </title>**

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

**</head>**

**<body>**

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

**</body>**

**</html>**

Next, 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.

**<body>**

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

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

**</body>**

Done! Now let's move on to our script - **script.js.**

Drawing game elements

Just like in the last game, let's create 2 variables that will determine the dimensions of the game window - width and height.

**const WIDTH = 480;**

**const HEIGHT = 360;**

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);**

With **addTo**, we tell where we want to add our **SVG** tag (we pass the element **id** selector as an argument to the function).

And we use 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. That's it, our canvas is ready).

Add our pictures.

**1. Background**

The background should cover the entire document, so after loading our image, we set the same dimensions as our document.

**let background = draw.image ('images / desert.png'). size (WIDTH, HEIGHT);**

**2. Dinosaur**

Add our hero - Dinosaur, select the first picture (the rest will be useful to us a little later, when we will do the animation - we will change the pictures to make Dino move), set the size of our Dino (we divide the original sizes in half, for the current size of the playing field 480x360 pixels such numbers are great) and move it so that it is on the ground and not flying in the sky).

**let dino = draw.image ('images / dino1.png'). size (84, 67) .move (0, 180);**

**3. Cactus**

Then add a cactus, through which Dino will jump (remember Dinosaur Google, everything is there too, only ours are much more beautiful - the pictures are colored).

**let cactus = draw.image ('images / cactus2.png'). size (50, 62) .move (400, 180);**

**4. Scoreboard (points, points)**

And the last element is our text board, where we will show how many points we have scored in the game.

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

**size: 40**

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

Done, all the elements are loaded, now we will animate them).

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 movement - animate the cactus**

First, let's make the cactus move to the left and create the illusion of movement - as if Dino is always moving forward and the cactus is approaching him because he is catching up with him.

**function update () {**

**cactus.dx (-4);**

**}**

**setInterval (update, 10);**

So, we see that the cactus flew past us and disappeared. Let's make it so that when the cactus flew over the left edge of our playing field and disappeared, it returned (moved) over the right edge and ended up on the right of us again and began to move again (as if it were a new cactus, and he was left behind).

**function update () {**

**cactus.dx (-4);**

**if (cactus.x () <= 0) {**

**cactus.x (WIDTH);**

**}**

**}**

**setInterval (update, 10);**

Super, already something interesting turns out. Only our Dino is standing in one place and it doesn't look like he is running, right? Let's fix it. We have 2 more pictures of our Dino (dino2, dino3). How are all three pictures different? Let's open them and see. Do you see how he moves his legs?

**2. Dino animation**

Let's create a separate animation function at the very end of our code. And we'll just switch pictures between ourselves. If there is the first picture, we will say to turn on the second, if the second, then turn on the third, if the third, then turn on the first again. And constantly changing the pictures, we will get a cool animation with our dinosaur). You will see now).

Remember how we create conditions in JavaScript? (right, if statement).

And what attribute do img tags have, where do we indicate which image we want to upload? (correct src).

Look, Dino has it too. (We can go to the developer's console and display our **DINO** variable, and there we will find the **src** property, where the path to the picture will be indicated, which was prescribed when it was loaded using the **image()** function)

**// code above unchanged**

**setInterval (update, 10);**

**function animation () {**

**if (dino.src == "images / dino1.png") {**

**dino.load ("images / dino2.png");**

**} else if (dino.src == "images / dino2.png") {**

**dino.load ("images / dino3.png");**

**} else {**

**dino.load ("images / dino1.png");**

**}**

**}**

Graphical user interface, text, application

Description automatically generated

Ok, we've created the function, but if we refresh our page now, nothing works. Why? (Correct, because we haven't run our function yet)

Let's also use the **setInterval** command to tell **JavaScript** to run our function every **100 milliseconds** (or a little more, but not worth less, otherwise it will be Dino Flash).

**function animation () {**

**if (dino.src == "images / dino1.png") {**

**dino.load ("images / dino2.png");**

**} else if (dino.src == "images / dino2.png") {**

**dino.load ("images / dino3.png");**

**} else {**

**dino.load ("images / dino1.png");**

**}**

**}**

**setInterval (animation, 100)**

Wow, look how cool! Dino now seems to be running and blowing away all the cacti with his forehead) (well, or he walks past them) Let's now teach our Dino to jump, because then we will make it so that when he hits the cactus (crashes into him), the game will stop ... (Loss, end of the game).

**3. Jump**

To jump, we need 2 variables - the first variable will be responsible for the distance at which Dino will jump, and the second for enabling / disabling the jump (so that we can only jump once, not 20 so that we do not accidentally fly into space).

Let's create these 2 variables at the very beginning, before the update () function (because then we will simply change the values, and if we create inside the update function - which works for us every 10 milliseconds, then these variables will be recreated by our function every time and the values ​​cannot be changed).

The initial values ​​will be 0 and false, because the jump will be when you press a key (space, for example), and not when you just opened the page with the game.

**let changeY = 0;**

**let isJump = false;**

**function update () {**

**// code below unchanged**

Okay, now let's think about when Dino should jump and when not. That's right when the button was pressed. Let's now write the code that will fire when we click on the buttons. (We do this outside the **update** function, and at the bottom, for example, because we do not need the code to be included every 10 milliseconds, but always work as soon as our page with the game is opened)

**// the code above is unchanged**

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

**if (event.keyCode == 32 && isJump == false) {**

**changeY = -14;**

**isJump = true;**

**}**

**});**

**PS isJump == false can be written as! IsJump (not isJump, not false = true, not false = true)**

So, they did it - they said to reduce the height (to move up -14) and switched on the jump, but nothing happens. Why? And we didn't write the commands for the jump itself in the **update** function).

Let's say move our Dino up (along the **Y**-axis) by the value that was set in our **changeY** variable when the spacebar was pressed.

**function update () {**

**dino.dy (changeY);**

**cactus.dx (-4);**

**if (cactus.x () <= 0) {**

**cactus.x (WIDTH);**

**}**

**}**

**setInterval (update, 10);**

We did it, and now what? Press the space bar and Dino flew up into space). They laughed, now we need to fix it.

Look, you and I have a variable that should enable the jump, and when we press the spacebar, we turn on our variable - instead of **FALSE** (no, off), we set it to **TRUE** (yes, turn on).

And let's add the condition that when Dino flew to a certain height (a certain point), then we turn off the jump, set the value to **FALSE** again (no, off), but tell Dino to fly up - jump when the value is **TRUE** (yes, turn on) ...

**function update () {**

**if (isJump == true) {**

**dino.dy (changeY);**

**if (dino.y () <= 100) {**

**isJump = false;**

**}**

**}**

**cactus.dx (-4);**

**if (cactus.x () <= 0) {**

**cactus.x (WIDTH);**

**}**

**}**

Now Dino no longer flies into space but hangs at a height of Y = 100 pixels. We must lower it back to Earth). And to do this smoothly, you and I will change our **changeY** variable while Dino flies up (we will increase it) - as if the Earth attracts Dino to itself (like you and I when we jump or throw something up, it then flies down, due to the acceleration of free fall, but now not about physics).

Look, our variable is equal to **-14** at the beginning (a negative number) and when we substituted it into the **dy** command for Dino, then **Y** decreases and Dino is therefore at the top, now we need to put a positive number in the variable so that Dino is down flew, right?

But if we put it down right away, it will fly down sharply, so we will do it gradually - let's try to **add 1** each time Dino flies up.

**function update () {**

**if (isJump == true) {**

**dino.dy (changeY);**

**changeY = changeY + 1;**

**if (dino.y () <=100) {**

**isJump = false;**

**}**

**}**

So, now it is hovering again at **100** pixels. Let's change the height, let's say turn off the jump if Dino has reached the top edge - when his **Y** is 0 or less.

**function update () {**

**if (isJump == true) {**

**dino.dy (changeY);**

**changeY = changeY + 1;**

**if (dino.y () <=0) {**

**isJump = false;**

**}**

**}**

Oops, now he jumps and falls down - like he fell through the Earth). Laughed again, let's fix it.

Look how our commands are working now, we told you to turn off the jump - assign **FALSE** if Dino's **Y** is less than or equal to 0. But, our Dino now does not reach 0, but immediately falls down when he rises to some height ... Let's now tell the jump to turn off when Dino returned to Earth - when his Y again became equal to 180 or even more.

**function update () {**

**if (isJump == true) {**

**dino.dy (changeY);**

**changeY = changeY + 1;**

**if (dino.y ()> =180) {**

**isJump = false;**

**}**

**}**

That's better. He jumps, but not high, and very quickly comes back down. Let's change (increase) our variable **changeY** not so fast, 2 times slower, we will add 0.5 each.

**function update () {**

**if (isJump == true) {**

**dino.dy (changeY);**

**changeY = changeY + 0.5;**

**if (dino.y ()> = 180) {**

**isJump = false;**

**}**

**}**

Hurray, it worked. What do you think? Jumping well now? Yes, I also like it, now we have time to jump over the cactus, otherwise it didn't work before. Super, it remains to add the loss and the score.

**4. Losing**

When you hit a cactus, the game turns off.

**function update () {**

**if (isJump) {**

**dino.dy (changeY);**

**changeY + = 0.5;**

**if (dino.y ()> = 180) {**

**isJump = false;**

**}**

**}**

**let collision =**

**dino.x () + dino.width ()> cactus.x () &&**

**dino.x () <cactus.x () + cactus.width () &&**

**dino.y () + dino.height ()> cactus.y ();**

**if (collision) {**

**alert ("The dinosaur has suffered");**

**background.load ("images / desertGO.png");**

**clearInterval (update\_id);**

**}**

**cactus.dx (-4);**

**if (cactus.x () <= 0) {**

**cactus.x (WIDTH);**

**}**

**}**

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

**5. Account**

Add points - count how many cacti passed by us (how many were behind the left edge) /

**let changeY = 0;**

**let isJump = false;**

**let score = 0;**

**function update () {**

**// code unchanged**

**cactus.dx (-4);**

**if (cactus.x () <= 0) {**

**score + = 1;**

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

**cactus.x (WIDTH);**

**}**

**// code unchanged**

**}**

The game is ready).

**Homework**

Try to make the background in the game also move to the left (as the cactus did) Hint: Add a copy of the background - the second picture is the same, and move the pictures to the left, when one is hidden behind the left edge, then we return it to the right edge, while the second passes along game screen.