Лабораторная работа №2
«Анимация с помощью элемента <canvas>»

Оглавление

Введение

Создание анимации с помощью элемента `<canvas>` в JavaScript похоже на покадровую мультипликацию – мы рисуем фигуру определенного размера в определенной позиции, потом изменяем этот размер или позицию, а потом очищаем холст и перерисовываем фигуру. Если обновление картинки происходит очень быстро, то получается плавная анимация.

Движение по странице

Давайте воспользуемся элементом `<canvas>` и функцией `requestAnimationFrame`, чтобы изобразить плавно двигающийся по странице квадрат (см. программу `2.js`).

В программе `2.js` создается холст и контекст рисования. Затем мы создаем переменную `position` и присваиваем ей значение `0` (`let position = 0`). Эта переменная понадобится, чтобы управлять перемещением квадрата слева направо.

Затем вызывается функция `animate`.

В этой функции, вызывается метод `clearRect`, который очищает прямоугольную область на холсте.

После очистки холста, вызывается метод `fillRect(position, 0, 20, 20)`, чтобы изобразить квадрат со стороной 20 пикселей в точке (position,0). Сразу после старта программы квадрат будет нарисован в позиции (0,0), поскольку `position` в начале равна 0.

Далее командой `position++` позиция увеличивается на `1`.

Затем с помощью команды `if(position > 400)` проверяется, не стала ли переменная `position` больше 400, и если стала, сбрасываем ее в 0.

Далее нужно потребовать от браузера, чтобы он циклически выполнял эти действия. Для этого можно использовать функцию `requestAnimationFrame()`, которая вызывает функцию, указанную в первом параметре некогда в будущем. После вызова функции необходимо вновь потребовать от браузера вызвать функцию в будущем, потому что предыдущее требование автоматически аннулируется после его выполнения.

В результате квадрат плавно движется по экрану. Когда он, пройдя 400 пикселей, достигнет края холста, его позиция обнулится.

Изменение размера квадрата

Немного изменив последний пример, можно изобразить квадрат, который не движется, а увеличивается в размере.

Во-первых, вместо переменной `position` теперь используется переменная `size`, предназначенная для управления размером квадрата. Во-вторых, вместо того чтобы при помощи этой переменной установить горизонтальную позицию, мы командой `ctx.fillRect(0, 0, size, size)` задаем ширину и высоту. В результате в верхнем левом углу холста рисуется квадрат со стороной `size`.

Поскольку изначально `size` равен 0, квадрат невидим. При следующем вызове функции `size` будет равна 1, что даёт квадрат со стороной в 1 пиксель. И так при каждом новом вызове будет рисоваться квадрат на 1 пиксель больше предыдущего. Таким образом, в левом верхнем углу страницы возникает квадрат, который растет до тех пор, пока не заполнит собой весь холст. Затем, когда выполнится условие `size > 400`, квадрат исчезнет и снова начнет расти из левого верхнего угла.

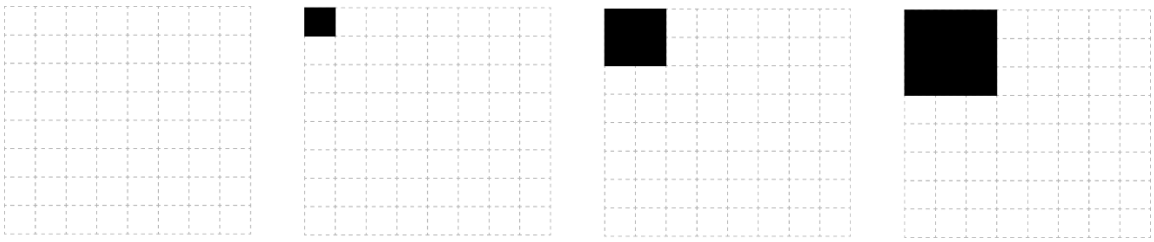

Рис. 1. На каждом шаге анимации `size` увеличивается на 1, и также на 1 увеличиваются ширина и высота квадрата

Случайная пчела

Теперь, когда мы знаем, как двигать объекты по экрану и увеличивать их, давайте изобразим кое-что забавное – а именно пчелу, которая случайным образом перемещается по холсту! Мы сделаем пчелу из нескольких окружностей, вот так:

Работать эта анимация будет примерно так же, как в примере с квадратом: задаем начальную позицию и далее для каждого шага анимации очищаем холст, рисуем пчелу в текущей позиции и обновляем позицию. Однако, чтобы пчела двигалась случайным образом, нам придется использовать более сложную логику изменения позиции, чем для квадрата. Несколько следующих разделов будут посвящены созданию кода для этой анимации.

Функция `circle`

Изображение нашей пчелы состоит из нескольких окружностей, поэтому напишем функцию, рисующую заполненные или обведенные окружности.

Рисуем пчелу

Теперь создадим функцию `drawBee`, которая будет рисовать пчелу центр тела которой задается координатами (*x*, *y*), используя для этого функцию `circle`.

Вначале, мы устанавливаем нужные для рисования свойства `lineWidth`, `strokeStyle` и `fillStyle`.

Для `lineWidth` задаем значение 2 пикселя, а для `strokeStyle` цвет `Black` (черный). Это значит, что окружности, которые мы используем для рисования тела пчелы, ее крылышек и глаз, будут с жирной черной обводкой. Для `fillStyle` зададим значение `Gold` (золотой), чтобы закрасить тело пчелы приятным желтым цветом.

Далее нарисуем набор окружностей, из которых состоит наша пчела. Давайте рассмотрим каждую окружность по отдельности.

Первая окружность – это заполненное желтым цветом тело пчелы с центром в точке (x, y), радиусом 8 пикселей:

Вторая окружность – это черный контур вокруг тела пчелы, его размер и позиция такие же, как у первой окружности. Вместе с первой окружностью получится вот что:

Теперь рисуем крылышки. Первое крыло – это окружность-контур с радиусом 5 пикселей, центр которой на 5 пикселей левее и на 11 пикселей выше центра тела пчелы. Второе крыло точно такое же, однако его центр на 5 пикселей правее центра тела.

Вместе с этими окружностями наша пчела выглядит так:

И наконец, рисуем глаза. Первый на 2 пикселя левее и на 1 пиксель выше центра тела, с радиусом 2 пикселя. Второй глаз такой же, но на 2 пикселя правее центра тела.

Изменение позиции пчелы

Для случайного изменения *x* и *y* координат пчелы – чтобы создавалось впечатление, будто она летает туда-сюда по холсту, – мы создадим функцию `update`. Эта функция принимает единственную координату. Мы будем обновлять по одной координате за раз, чтобы пчела двигалась случайным образом вправо-влево и вверх-вниз.

Изменяем координату на величину смещения

В первой строке функции `update` создадим переменную `offset` – это смещение, определяющее, на сколько нужно изменить текущую координату. Мы вычисляем его как `Math.random( ) * 4 - 2`, получая случайное число в диапазоне от -2 до 2. Логика такая: сам по себе вызов `Math.random()` вернет случайное значение от 0 до 1, следовательно, `Math.random()*4` даст число от 0 до 4. И вычтя 2, мы получим наше случайное число от -2 до 2.

Далее мы используем команду `coordinate += offset`, чтобы изменить координату на величину смещения `offset`. Если смещение положительное, координата увеличится, если отрицательное – уменьшится.

Проверка выхода за границу

Затем мы проверяем, не вылетела ли пчела за границу холста, то есть не стала ли координата больше 400 или меньше 0. Если она стала больше 400, мы присваиваем ей значение 400, а если стала меньше 0, сбрасываем ее в 0.

Возвращаем обновленную координату

В конце мы возвращаем координату с помощью `return`.

Анимируем пчелу

Теперь, когда у нас есть функции `circle`, `drawBee` и `update`, можно написать код для анимации нашей неугомонной пчелы.

Как обычно, сначала мы получаем холст `canvas` и контекст рисования `ctx`. Затем создаем переменные *x* и *y*, задавая им обеим значение 200. Таким образом, начальная позиция нашей пчелы – точка (200, 200), то есть центр холста, как показано на рис. 2.

Рис. 2. Пчела в позиции (200, 200}

Внутри функции `animate`, мы первым делом очищаем холст вызовом `clearRect`. Затем мы рисуем пчелу в позиции (*x*, *y*). При первом вызове функции пчела будет нарисована в позиции (200, 200), как на рис. 2.

Далее, мы обновляем значения *x* и *y*. Функция `update` принимает число, добавляет к нему случайное значение в диапазоне от -2 до 2 и возвращает обновленное число. А код `x = update(x)` фактически означает «изменить x на небольшую случайную величину».

Запустив этот код, вы увидите желтую пчелу, которая случайным образом перемещается по холсту. На рис. 3 показаны несколько кадров анимации.

Рис. 3. Случайная анимация пчелы

Отскакивающий мяч

Теперь давайте изобразим летающий по холсту мяч. При столкновении с одной из границ он будет отскакивать назад, меняя направление, словно резиновый.

Мы создадим для нашего мяча класс `Ball`. Класс будет хранить скорость мяча и направление его движения с помощью двух свойств, `xSpeed` и `ySpeed`. Горизонтальная скорость мяча будет определяться значением `xSpeed`, а вертикальная – `ySpeed`.

Класс `Ball`

Конструктор класса `Ball` довольно простой: он задает начальную позицию мяча (`this.x` и `this.y`), его горизонтальную скорость (`this.xSpeed`) и скорость вертикальную (`this.ySpeed`). Начальной позицией будет точка (200, 200) – это центр нашего 400×400-пиксельного холста.

Начальное значение `this.xSpeed` равно -2, это значит, что мяч будет смещаться на 2 пикселя влево на каждом шаге анимации. Начальное значение `this.ySpeed` равно 3, то есть на каждом шаге анимации мяч будет также смещаться на 3 пикселя вниз. Следовательно, мяч будет двигаться по диагонали (3 пикселя вниз и 2 влево).

На рис. 4 показано начальное положение мяча и направление движения.

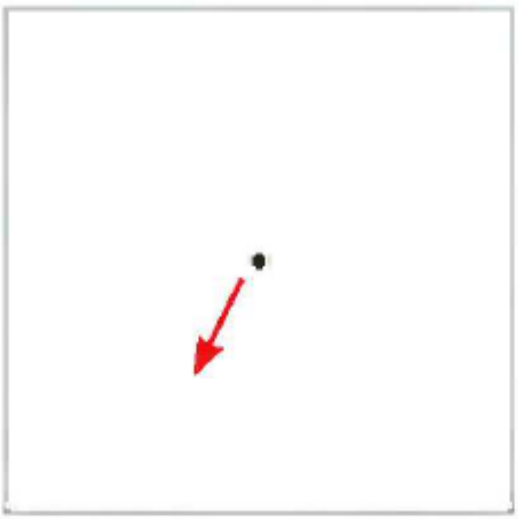

Рис. 4. Начальное положение мяча, направление движения показано стрелкой

Рисуем мяч

Далее напишем метод `draw` для отрисовки мяча. Этот метод рисует окружность с центром в точке (`this.x`, `this.y`). Радиус окружности – 6 пикселей, а в качестве последнего аргумента передается `true`, чтобы окружность была заполненной.

Перемещение мяча

Чтобы мяч двигался, требуется изменять значения свойств *x* и *y* в соответствии с текущей скоростью. Это делается в методе `move`.

Мы используем команду `this.x += this.xSpeed`, чтобы прибавить значение горизонтальной скорости к `this.x`. Аналогично `this.y += this.ySpeed` прибавляет вертикальную скорость к `this.y`. Например, в самом начале анимации мяч находится в позиции (200, 200), `this.xSpeed` равняется -2, `this.ySpeed` равняется 3. Метод `move` при его вызове вычтет 2 из значения *x* и прибавит 3 к значению *y*, в результате чего мяч окажется в позиции (198, 203), то есть переместится на 2 пикселя влево и на 3 пикселя вниз, как показано на рис. 5.

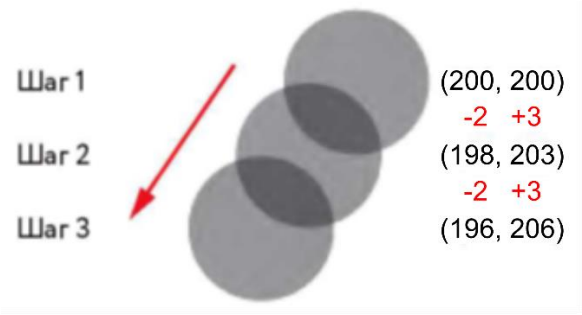

Рис. 5. Первые три шага анимации. Показано, как меняются значения свойств *x* и *y*

Отскоки мяча

На каждом шаге анимации нужно проверять, не столкнулся ли мяч с границей холста. Если столкнулся, следует обновить свойство `xSpeed` или `ySpeed`, инвертировав его значение (то есть умножив на -1).

Проверка столкновения осуществляется в методе `checkCollision`. Мы выясняем, не столкнулся ли мяч с левой или правой границей, сравнивая свойство *x* с 0 (если *x* меньше 0, мяч столкнулся с левой границей) и с 400 (если *x* больше 400, мяч столкнулся с правой границей). Если любая из этих проверок даст `true`, значит мяч начал выходить за пределы холста и его горизонтальное направление нужно инвертировать. Мы делаем это, задавая свойству `this.xSpeed` значение `-this.xSpeed`. Например, если `this.xSpeed` равняется -2 и мяч столкнулся с левой границей, `this.xSpeed` примет значение 2.

Далее мы выполняем аналогичную проверку для верхней и нижней границ. Если `this.y` меньше 0 или больше 400, значит мяч столкнулся или с верхней, или с нижней границей соответственно. В этом случае мы задаем `this.ySpeed` значение `-this.ySpeed`.

На рис. 6 показано, что произойдет при столкновении мяча с левой границей. Сначала значение `this.xSpeed` равняется -2, но после столкновения оно меняется на 2. Однако в `this.ySpeed` по-прежнему остается значение 3.

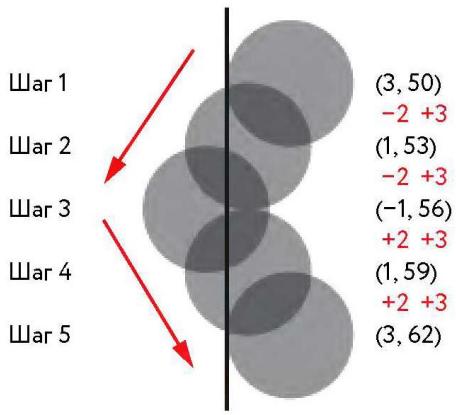

Рис. 6. Так меняется `this.xSpeed` после столкновения с левой границей

Как видно из рис. 6, на третьем шаге мяч сталкивается с границей – его центральная точка уходит за границу холста. При этом часть мяча станет невидимой, однако этот кадр промелькнет столь быстро, что во время анимации будет едва заметным.

Анимация мяча

Создадим объект-мяч вызовом `new Ball()` и сохраним его в переменной `ball`.

В функции `animate` выполняется несколько задач. Сначала она очищает холст командой `ctx.clearRect(0, 0, 400, 400)` . После этого вызываются методы объекта `ball`: `draw`, `move` и `checkCollision`. Метод `draw` рисует мяч в его текущей позиции `this.x, this.y`. Метод `move` обновляет позицию мяча на основе значений `xSpeed` и `ySpeed`. И, наконец, метод `checkCollision` меняет направление движения мяча, если тот столкнулся с границей холста.

Когда вы запустите этот код, мяч сразу же начнет двигаться вниз и влево. Затем он врежется в нижнюю границу и отскочит вверх и влево. И так он будет летать по холсту, отскакивая от границ, пока вы не закроете окно браузера.

Задания для самостоятельной работы

- Изменение размеров «холста»
  Вместо того чтобы вводить числовые размеры холста в тексте программы, можно создать переменные для ширины и высоты и брать их значения из объекта `canvas`. Используйте следующий код:
  ```
  const width = canvas.width;
  const height = canvas.height;
  ```
  Если везде в программе использовать эти переменные вместо чисел, вам останется лишь изменить атрибуты элемента `canvas` в HTML-коде, чтобы опробовать новый размер холста.
  Измените размеры холста, чтобы он был шириной в 500 пикселей и высотой в 300. Удостоверьтесь, что мяч по-прежнему появляется в центре холста и отскакивает от его новых краев.
- Случайные значения `this.xSpeed` и `this.ySpeed`
  Чтобы анимация была интереснее, в конструкторе `Ball` задайте свойствам `this.xSpeed` и `this.ySpeed` разные случайные значения в диапазоне от -5 до 5.
- Много мячей
  Вместо анимации одного мяча создайте пустой массив и в цикле `for` добавьте в него 10 мячей. В функции `animate` используйте цикл `for` для перемещения и проверки столкновений каждого из мячей.
- Цветные мячи
  Задайте в конструкторе `Ball` новое свойство `color` (цвет) и используйте его для задания цвета в методе `draw`. Присвойте случайный цвет каждому из мячей из такого массива:
  ```
  const colors = ["Red", "Orange", "Yellow", "Green", "Blue", "Purple"];
  ```