Занятие 2 . Анимация массива шариков

Задание.

Шарики случайного размера и случайного цвета должны двигаться по экрану.

В этом примере будет использоваться API-интерфейс requestAnimationFrame для анимации всего экрана - вам не нужно заранее знать эти API-интерфейсы.

Обновление данных мяча

Мы можем нарисовать мяч в нужной позиции, но для его перемещения нам нужна какая-то функция обновления. Добавьте следующий код для метода update() в прототип Ball ():

```
Ball.prototype.update = function() {
  if ((this.x + this.size) >= width) {
    this.velX = -(this.velX);
  }

  if ((this.x - this.size) <= 0) {
    this.velX = -(this.velX);
  }

  if ((this.y + this.size) >= height) {
    this.velY = -(this.velY);
  }

  if ((this.y - this.size) <= 0) {
    this.velY = -(this.velY);
  }

  this.x += this.velX;
  this.y += this.velY;
}
```

Первые четыре части функции проверяют, достиг ли шар края холста. Если это так, мы меняем полярность соответствующей скорости, чтобы заставить шар двигаться в противоположном направлении. Так, например, если шар двигался вверх (положительный velY), то вертикальная скорость изменяется так, что вместо этого он начинает двигаться вниз (отрицательный velY).

В четырех случаях мы проверяем, чтобы увидеть:

- если координата x больше ширины холста (мяч уходит от правого края).
- если координата x меньше 0 (мяч уходит с левого края).
- если координата y больше высоты холста (шарик уходит за верхний край).
- если координата y меньше 0 (шарик уходит за нижний край).

В каждом случае мы учитываем размер шара в расчете, потому что координаты x / y находятся в центре шара, но мы хотим, чтобы край шара отскочил от периметра - мы не хотим, чтобы шар отойдите на полпути от экрана, прежде чем он начнет приходить в норму.

Последние две строки добавляют значение velX к координате x, а значение velY к координате y - мяч фактически перемещается при каждом вызове этого метода.

Анимация мяча

Добавьте код:

```
function loop() {
  ctx.fillStyle = 'rgba(0, 0, 0, 0.25)';
  ctx.fillRect(0, 0, width, height);

  for (let i = 0; i < balls.length; i++) {
    balls[i].draw();
    balls[i].update();
  }

  requestAnimationFrame(loop);
}
loop();
```

Функция requestAnimationFrame(`loop`) будет вызываться в зависимости от частоты кадров вашего браузера и компьютера (обычно это 60 кадров в секунду). Ключевым отличием является то, что вы просите браузер выполнить функцию (в нашем примере `loop`) при первой возможности, а не с заданным интервалом.

Все программы, которые анимируют объекты, обычно включают в себя цикл анимации, который служит для обновления информации в программе и последующего отображения результирующего представления в каждом кадре анимации; это основа для большинства игр и других подобных программ. Наша функция loop() выполняет следующие действия:

1. Устанавливает цвет заливки холста на полупрозрачный черный, затем рисует прямоугольник цвета по всей ширине и высоте холста, используя fillRect() для того, чтобы скрыть рисунок предыдущего кадра до того, как будет нарисован следующий. Если вы этого не сделаете, вы просто увидите, как длинные змеи пробираются по холсту вместо движущихся шаров! Цвет заливки установлен на полупрозрачный, rgba (0,0,0,0.25), чтобы позволить нескольким предыдущим кадрам слегка просвечивать, создавая небольшие следы за шарами при их движении. Если вы изменили 0,25 на 1, вы их больше не увидите. Попробуйте изменить это число, чтобы увидеть этот эффект.
2. Перебирает все шары в массиве шаров и запускает функции draw() и update() каждого шара, чтобы нарисовать каждый из них на экране, а затем выполняет необходимые обновления для положения и скорости во времени для следующего кадра.
3. Повторно запускает функцию, используя метод requestAnimationFrame() - когда этот метод многократно запускается и передается одно и то же имя функции, он запускает эту функцию определенное количество раз в секунду, чтобы создать плавную анимацию. Обычно это делается рекурсивно - это означает, что функция вызывает себя каждый раз, когда запускается, поэтому она запускается снова и снова.
4. И последнее: добавьте следующую строку в конец кода - нам нужно вызвать функцию один раз, чтобы запустить анимацию.

```
loop();
```

Вот и все - попробуйте сохранить и обновить, чтобы проверить свои прыгающие шары!

Добавление обнаружения столкновений

Теперь для развлечения давайте добавим в нашу программу обнаружение столкновений, чтобы наши шары знали, когда они попали в другой шар.

1. Сначала добавьте следующее определение метода ниже, где вы определили метод update () (то есть блок Ball.prototype.update).

```
Ball.prototype.collisionDetect = function() {
  for (let j = 0; j < balls.length; j++) {
    if (!(this === balls[j])) {
      const dx = this.x - balls[j].x;
      const dy = this.y - balls[j].y;
      const distance = Math.sqrt(dx * dx + dy * dy);

      if (distance < this.size + balls[j].size) {
        balls[j].color = this.color = 'rgb(' + random(0, 255) + ',' +
random(0, 255) + ',' + random(0, 255) +')';
      }
    }
  }
}
```

Для каждого шара нам нужно проверить каждый второй шар, чтобы увидеть, не столкнулся ли он с текущим шаром. Чтобы сделать это, мы запускаем еще один цикл for для прохождения всех шаров в массиве balls [].

Непосредственно внутри цикла for мы используем оператор if, чтобы проверить, является ли текущий шарик, проходящий через цикл, тем же самым, что и тот, который мы проверяем в данный момент. Мы не хотим проверять, не столкнулся ли мяч с самим собой! Для этого мы проверяем, совпадает ли текущий шар (т. Е. Шар, у которого вызывается метод collisionDetect) с шаром петли (т. Е. Шар, на который ссылается текущая итерация цикла for в collisionDetect метод). Мы тогда используем! отменить проверку, чтобы код внутри оператора if выполнялся, только если они не совпадают.

Затем мы используем общий алгоритм для проверки столкновения двух окружностей. Мы в основном проверяем, перекрываются ли какие-либо области двух кругов. Это объясняется далее в 2D обнаружении столкновений.

Если обнаружено столкновение, выполняется код внутри внутреннего оператора if. В этом случае мы устанавливаем для свойства color обоих кругов только новый случайный цвет. Мы могли бы сделать что-то гораздо более сложное, например, заставить шары реально отскакивать друг от друга, но это было бы гораздо сложнее реализовать. Для такого моделирования физики разработчики, как правило, используют игры или библиотеки физики, такие как PhysicsJS, material.js, Phaser и т. Д

2. Вы также должны вызывать этот метод в каждом кадре анимации. Добавьте следующее ниже шаров [i] .update (); линия:

```
balls[i].collisionDetect();
```

3. Сохраните и обновите демо снова, и вы увидите, как ваши шары меняют цвет при столкновении!