Интерактивное программирование

До сих пор JavaScript начинал работу сразу же после загрузки страницы, приостанавливаясь лишь при вызове некоторых функций, таких как alert или confirm. Однако порой не нужно выполнять весь код сразу — что если мы хотим запустить фрагмент кода спустя какое-то время или в ответ на действие пользователя?

На этом занятии мы изучим разные способы управлять тем, когда именно выполняется наш код. Это называется *интерактивным программированием*. Оно позволяет создавать интерактивные веб-страницы, которые могут изменяться со временем и реагировать на действия пользователей.

Отложенное выполнение кода и setTimeout

Вместо того чтобы вызывать функцию сразу, можно попросить JavaScript сделать это спустя определенное время. Такого рода отложенное выполнение называется *запуском по таймеру*, и для этого в JavaScript есть функция setTimeout. Данная функция принимает два аргумента: функцию, которую надо будет вызвать при срабатывании таймера, и само время ожидания в миллисекундах. Следующий пример показывает, как открыть диалог alert через setTimeout.

```
var timeUp = function ()
{
  alert("Время: вышло!");
};
setTimeout(timeUp, 3000);
3
```

Мы создали функцию timeUp, открывающую диалог alert с сообщением "Время вышло!". Затем вызвали setTimeout с двумя аргументами: функцией, которую нужно запустить (timeUp), и числом миллисекунд (3000), которые должны пройти перед ее запуском. По сути, мы говорим: «Подожди три секунды и вызови timeUp». Сразу после вызова setTimeout(timeUp, 3000) ничего не произойдет, однако через три секунды сработает функция timeUp, открыв диалог alert.

Обратите внимание — вызов setTimeout вернул число 3. Это значение называют идентификатором (ID) таймера, который обозначает этот конкретный таймер (отложенный вызов функции). Заметим, что возвращаемое setTimeout значение может быть любым числом, ведь это просто идентификатор. Вызовите setTimeout снова, и он вернет другой ID таймера:

```
setTimeout(timeUp, 3000);
7
```

Полученный ID можно передать функции clearTimeout, чтобы отменить этот конкретный таймер. Об этом я расскажу ниже.

Отмена действия таймера

После задания отложенного вызова функции с помощью setTimeout может выясниться, что вызывать эту функцию больше не нужно. Представьте, что вы поставили будильник, чтобы он напомнил вам о домашнем задании, однако в итоге сделали все заранее и теперь хотите отключить будильник. Для отмены действия таймера используется функция clearTimeout с ID таймера (полученным ранее от setTimeout) в качестве аргумента. Предположим, вы установили таймер «сделай домашку» следующим образом:

```
var doHomeworkAlarm = function()
{
  alert("Эй! Пора делать домашку!");
};
var timeoutId = setTimeout(doHomeworkAlarm, 60000);
```

Функция doHomeworkAlarm создает диалог alert, напоминающий о домашке. Вызов setTimeout(doHomeworkAlarm, 60000) сообщает JavaScript, что функцию doHomeworkAlarm нужно вызвать через 60 000 миллисекунд (то есть 60 секунд). Затем мы вызвали setTimeout и сохранили ID таймера в новой переменной timeoutID.

Теперь, чтобы отменить действие таймера, достаточно передать его ID функции clearTimeout, вот так:

```
clearTimeout(timeoutId);
```

Теперь setTimeout не будет вызывать функцию doHomeworkAlarm.

Многократный запуск кода и setInterval

Функция setInterval похожа на setTimeout, однако она вызывает переданную ей функцию повторно через определенные промежутки (интервалы) времени. Скажем, если вы хотите с помощью JavaScript обновлять показания часов, используйте setInterval, чтобы функция обновления вызывалась раз в секунду. SetInterval принимает два аргумента: функцию и интервал времени в миллисекундах.

Например, так можно раз в секунду выводить в консоль сообщение:

```
var counter = 1;
var printMessage = function()
{
  console.log("Ты смотришь в консоль уже " + counter + " сек");
  counter++;
};
var intervalId = setInterval(printMessage, 1000);
```

```
Ты смотришь в консоль уже 1 сек
Ты смотришь в консоль уже 2 сек
Ты смотришь в консоль уже 3 сек
Ты смотришь в консоль уже 4 сек
Ты смотришь в консоль уже 5 сек
Ты смотришь в консоль уже 6 сек
Ты смотришь в консоль уже 7 сек
```

```
clearInterval(intervalId);
```

В первой строке мы создали новую переменную counter и присвоили ей значение 1. С помощью этой переменной мы будем вести учет времени (в секундах).

Затем мы создали функцию printMessage, которая выполняет две задачи. Во-первых, она печатает сообщение о том, сколько секунд вы уже смотрите в консоль. Во-вторых, она увеличивает переменную counter на единицу.

Наконец, мы вызвали setInterval, передав ей функцию printMessage и число 1000, что означает «вызывай printMessage каждые 1000 миллисекунд». Так же как setTimeout возвращает ID таймера, setInterval возвращает *ID интервала*, который мы сохранили в переменной intervalId. Далее этот ID можно использовать для отмены периодического вызова функции printMessage — что мы и сделали с помощью функции clearInterval.

Анимация элементов с помощью setInterval

Отложенный вызов через setInterval можно использовать для анимации элементов в браузере. По сути, для этого нужно создать функцию, которая слегка сдвигает элемент, и затем передать ее setInterval, установив небольшое время повтора. При условии, что каждый сдвиг будет достаточно мал и величина интервала тоже, анимация получится очень плавной.

Давайте анимируем положение фрагмента текста в HTML-документе, двигая его по горизонтали. Создайте файл interactive.html с таким содержимым:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Интерактивное программирование</title>
</head>
<body>
<h1 id="heading">Привет, мир!</h1>
<script src="query-3.1.1.js"></script>
<script>
// Скоро здесь будет JavaScript-код
</script>
</body>
```

```
</html>
```

Теперь перейдем к JavaScript-коду. Как и прежде, поместите его в HTML-документ между тегами <script>.

```
var leftOffset = 0;
var moveHeading = function()
{
 $("#heading").offset({left: leftOffset});
 leftOffset++;
 if(leftOffset > 200)
  leftOffset = 0;
}
setInterval(moveHeading, 30);
```

Открыв наш документ в браузере, вы увидите, как элемент заголовка плавно сдвигается вправо, пока не пройдет расстояние в 200 пикселей. Затем он резко вернется на свое место, и все начнется снова. Разберемся, как это работает.

В первой строке мы создали переменную leftOffset (отступ слева), которой далее воспользуемся для задания позиции заголовка «Привет, мир!». Ее начальное значение — 0. Это значит, что заголовок начнет свое движение с левого края страницы.

Во второй строке мы создали функцию moveHeading, чтобы вызывать ее через setInterval. В теле moveHeading, мы используем $("#heading") для поиска элемента с id "heading" (это элемент h1) и вызываем метод offset, чтобы задать смещение заголовка от левого края экрана, сдвигая его вправо.

Метод offset принимает объект, который может содержать свойство left для задания смещения от левого края или свойство top для смещения от верха страницы. В данном случае мы выбрали свойство left и присвоили ему значение переменной leftOffset. Если бы требовалось задать неизменное смещение, мы могли бы указать для свойства числовое значение. Скажем, вызов $("#heading").offset({left: 100}) поместит заголовок на расстоянии 100 пикселей от левого края страницы.

Затем мы увеличили leftOffset на 1. Чтобы убедиться, что заголовок не уполз слишком далеко, выполняется проверка — leftOffset больше 200? Если это так, сбрасываем значение до 0. Наконец, мы вызвали setInterval, передав ей в качестве аргументов функцию moveHeading и число 30 (что означает 30 миллисекунд).

Этот код вызывает moveHeading каждые 30 миллисекунд, то есть примерно 33 раза в секунду. При каждом вызове moveHeading переменная leftOffset увеличивается, и далее значение этой переменной используется, чтобы задать положение заголовка. Поскольку функция вызывается периодически, а leftOffset каждый раз увеличивается на 1, заголовок плавно движется по странице, смещаясь на 1 пиксель каждые 30 миллисекунд.

Реакция на действия пользователя

Как мы выяснили, один из способов управления временем запуска кода — применение функций setTimeout и setInterval, которые вызывают функцию через фиксированный промежуток времени. Другой способ — выполнять код при определенных действиях пользователя, таких как клик мышкой, нажатие клавиши или просто перемещение мышки. Тогда пользователи смогут взаимодействовать с вашей страничкой, получая соответствующий отклик.

Каждый раз, когда вы совершаете действие — кликаете, вводите текст или двигаете мышку, — в браузере возникает нечто под названием событие. Это способ, которым браузер сообщает «случилось вот это». На события можно подписываться, добавляя обработчик события к элементу, в котором это событие происходит. Добавляя обработчик, вы говорите JavaScript: «Если произойдет это событие, вызови эту функцию». Например, если вы хотите вызывать функцию при клике по заголовку, добавьте к элементу заголовка обработчик события. Сейчас мы разберемся, как это делается.

Реакция на клики

Когда пользователь кликает по элементу в браузере, возникает событие *click*. С помощью jQuery ничего не стоит задать этому событию обработчик. Откройте созданный ранее файл

interactive.html, выберите Файл ► Сохранить как..., чтобы сохранить его под именем clicks.html, и замените содержимое второго элемента script таким кодом:

```
var clickHandler = function(event)
 {
  console.log("Клик! " + event.pageX + " " + event.pageY);
 }
 $("h1").click(clickHandler);
```

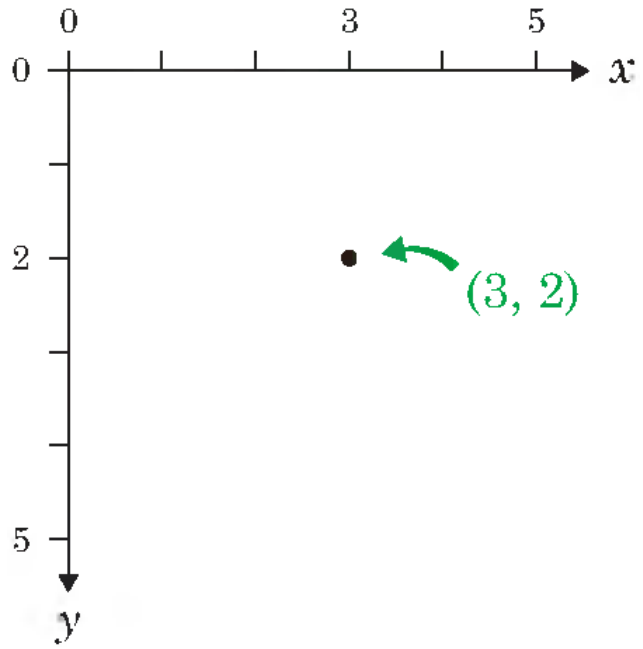
В первой строке мы создали функцию clickHandler с единственным аргументом event. При ее вызове в аргументе event будет передан объект, содержащий информацию о событии, например о том, в каком месте был сделан клик. В коде функции-обработчика, мы использовали console.log для вывода свойств pageX и pageY объекта event. Эти свойства хранят x- и y-координаты события — иными словами, они сообщают, где именно на странице произошел клик.

Наконец, в последней строке мы активировали обработчик кликов. Код $("h1") находит элемент h1, а вызов $("h1").click(clickHandler) означает: «В случае клика по элементу h1 вызови функцию clickHandler и передай ей объект события». В данном случае обработчик извлекает из объекта event информацию об x- и y-координатах клика.

Перезагрузите нашу модифицированную страницу в браузере и кликните по элементу заголовка. При каждом клике в консоли должна появиться новая строка, как показано ниже. Каждая из строк оканчивается двумя числами: x- иу-координатами клика.

```
Клик! 144 52
Клик! 144 41
Клик! 177 42
Клик! 20 38
Клик! 48 37
Клик! 71 24
Клик! 56 49
```

КООРДИНАТЫ В БРАУЗЕРЕ



В веб-браузере, как и в большинстве других сред графического программирования, нулевая позиция x- и y-координат находится в верхнем левом углу экрана. По мере роста x-координаты точка смещается к правому краю страницы, а по мере роста y-координаты — к низу страницы.

Событие mousemove

Событие mousemove возникает всякий раз при перемещении мышки. Создайте файл с именем mousemove.html и введите туда следующий код:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Движение мышью</title>
</head>
<body>
<h1 id="heading">Привет, мир!</h1>
<script src="jquery-3.1.1.js"></script>
<script>
 $("html").mousemove(function(event)
 {
  $("#heading").offset({left: event.pageX, top: event.pageY});
 })
</script>
```

```
</body>
</html>
```

В первой строке мы вызовом $("html").mousemove(обработчик) добавили обработчик события mousemove. В данном случае аргумент обработчик — это функция целиком. Она начинается после mousemove и продолжается до тега </script>. Мы использовали $("html"), чтобы найти элемент html, поэтому обработчик будет вызван при перемещении мышки в любом месте страницы: функция, которую мы передали в скобках после mousemove, будет вызываться всякий раз, когда пользователь передвинет мышку.

В этом примере вместо того, чтобы создать обработчик отдельно и передать методу mousemove имя функции (как мы это делали ранее с функцией clickHandler), мы передали mousemove непосредственно саму функцию. Это очень распространенный подход к написанию обработчиков, поэтому освоиться с такой записью весьма полезно.

В коде функции-обработчика, мы нашли элемент заголовка и вызвали для него метод offset. Как я уже говорил, у объекта, который передается в качестве аргумента offset, могут быть свойства left и top. В данном случае мы присваиваем свойству left значение event.pageX, а свойству top — event.pageY. Теперь каждый раз при передвижении мышки заголовок будет перемещаться в позицию, где произошло событие. Иными словами, куда бы вы ни передвинули мышь, заголовок будет следовать за ней.

Пишем игру «Найди клад!»

Давайте опробуем полученные знания в деле и напишем игру. Цель игры — найти клад. Веб-страница будет отображать карту, на которой программа случайным образом выбирает точку, где спрятаны сокровища. Каждый раз, когда игрок кликает по карте, программа сообщает, насколько он близок к кладу. При клике по точке с кладом (или очень близко к ней) игра выводит поздравление и сообщает, сколько кликов ушло на поиски.

Проектирование игры

Перед тем как писать код, давайте разберем общую структуру этой игры. Вот список задач, которые нужно выполнить для того, чтобы игра адекватно реагировала на клики по карте.

1. Создать страницу игры с картинкой (картой сокровищ) и местом, куда будут выводиться сообщения для игрока.
2. Выбрать на карте случайную точку, где спрятан клад.
3. Создать обработчик кликов. Каждый раз, когда игрок кликает по карте, обработчик кликов должен:
   - Увеличить счетчик кликов на 1.
   - Вычислить, насколько далеко место клика от места, где спрятан клад.
   - Отобразить на странице сообщение для игрока — «горячо» или «холодно».
   - Поздравить игрока, если он кликнул по кладу или вблизи него, и сообщить, сколько кликов ушло на поиски.

Я расскажу, как запрограммировать каждую из этих функций, а затем мы рассмотрим код игры целиком.

Создаем веб-страницу с HTML-кодом

Давайте рассмотрим HTML-код игры. Мы воспользуемся новым элементом *img* для отображения карты клада, а для вывода игровых сообщений добавим на страницу элемент *p*. Введите следующий код в новый файл под названием *treasure.html.*

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Найди клад!</title>
</head>
<body>
  <h1 id="heading">Найди клад!</h1>
  <img id="map" width="400" height="400" src="treasuremap.png">
  <p id="distance"></p>
  <script src="jquery-3.1.1.js"></script>
  <script>
    // Здесь будет код игры
  </script>
</body>
</html>
```

Элемент *img* служит для добавления изображений в HTML-документ. В отличие от прочих известных нам элементов, закрывающий тег *img* не нужен — потребуется лишь открывающий тег, который может содержать различные атрибуты. Мы добавили элемент *img* с *id "map"*. С помощью атрибутов *width* и *height* мы задали ширину и высоту соответственно — и то и другое по *400*, то есть наше изображение будет занимать *400* пикселей в ширину и *400* пикселей в высоту.

Чтобы указать, какое именно изображение нам нужно, мы использовали атрибут *src*, задав ему значение — веб-адрес картинки или локальный адрес в файловой системе. В данном случае это ссылка на изображение *treasuremap.png*, которое находится в той же папке, где и HTML-страница *treasure.html*.

После *img*, мы добавили пустой элемент *p*, задав ему *id "distance"* (расстояние). В этот элемент мы будем с помощью JavaScript выводить текст подсказок, сообщающих игроку, насколько он близок к цели.

Выбор случайного места для клада

Теперь давайте писать JavaScript-код игры. Первая наша задача — выбрать на карте случайное место для клада. Поскольку размер карты *400 x 400* пикселей, координаты ее верхнего левого угла равны *{ x: 0, y: 0 }*, а координаты нижнего правого угла — *{ x: 399, y: 399 }*.

Получение случайных значений

Чтобы указать на карте сокровищ случайную точку, нам нужно выбрать случайное значение в диапазоне от *0* до *399* для координаты *x* и случайное значение в том же диапазоне для координаты *y*. Для этого напишем функцию, которая принимает размер в качестве аргумента и возвращает случайное число от *0* до этого размера (но не включая его):

```
var getRandomNumber = function(size)
{
  return Math.floor(Math.random() * size);
}
```

Примерно такой же код мы использовали для получения случайных значений на предыдущих занятиях. Мы генерируем случайное число от *0* до *1* с помощью *Math.random*, умножаем его на аргумент *size* и затем используем *Math.floor* для округления до ближайшего снизу целого числа. Далее мы возвращаем полученный результат из функции. Вызов *getRandomNumber(400)* вернет случайное число от *0* до *399*, что нам и требуется.

Задаем координаты клада

Теперь используем функцию *getRandomNumber* для задания координат клада:

```
var width = 400;
var height = 400;
var target = {
  x: getRandomNumber(width),
  y: getRandomNumber(height)
};
```

В этом фрагменте кода задаются переменные *width* и *height*, соответствующие ширине и высоте элемента *img*, который мы используем в качестве карты. В следующей строке мы создали объект под названием *target* с двумя свойствами *x* и *y*, обозначающими координаты клада. Значения *x* и *y* мы получаем из функции *getRandomNumber*. Каждый раз при запуске этого кода мы получим новую случайную позицию на карте и координаты этой позиции будут сохранены в свойствах *x* и *y* переменной *target*.

Обработчик кликов

Обработчик кликов — функция, которая будет вызываться каждый раз, когда игрок кликнет по карте. Начнем писать эту функцию со следующего кода:

```
$("#map").click(function(event)
{
  // Здесь будет код обработчика
});
```

Сначала мы используем *$("#map")*, чтобы найти карту (поскольку *"map"* — это *id* элемента *img*), а затем указываем обработчик кликов. Всякий раз, когда игрок кликнет по карте, начнется выполнение тела функции между фигурных скобок. Информация о клике будет передана в функцию через аргумент *event*.

В обработчике нужно выполнить несколько действий: увеличить счетчик кликов, вычислить, насколько точка клика отстоит от координат клада, и отобразить сообщения. Перед тем как писать код обработчика, мы создадим переменные и функции, которые помогут нам запрограммировать нужные действия.

Подсчет кликов

Первое, что должен делать обработчик, — отслеживать число кликов. Для этого нам понадобится переменная *clicks*, которую мы создадим в начале программы (за пределами кода обработчика) и присвоим ей значение *0*:

```
var clicks = 0;
```

В код обработчика кликов мы включим команду *clicks++*, чтобы увеличивать счетчик каждый раз, когда игрок кликнет по карте.

Вычисляем расстояние от клика до клада

Чтобы выяснить, «горячо» или «холодно» (вблизи клада сделан клик или далеко от него), нужно найти расстояние между точкой клика и местом, где лежит клад. Для этого создаем функцию *getDistance*, вот такую:

```
var getDistance = function(event, target)
{
  var diffX = event.offsetX - target.x;
  var diffY = event.offsetY - target.y;
  return Math.sqrt((diffX * diffX) + (diffY * diffY));
}
```

Функция *getDistance* принимает два аргумента: *event* и *target*. Объект *event* — тот же самый, что передается обработчику кликов, и в нем содержится информация о событии. В частности, это свойства *offsetX* и *offsetY*, хранящие *x*- и *y*-координаты клика — как раз они нам и нужны.

В коде функции переменная *diffX* хранит горизонтальное расстояние между кликом и кладом, которое мы получаем, вычитая *target.x* (*x*-координата клада) из *event.offsetX* (*x*-координата клика). Тем же образом мы находим вертикальное расстояние, сохраняя его в переменной *diffY*.

И наконец, чтобы найти расстояние между двумя точками в коде функции *getDistance*, используется теорема Пифагора. В функции *getDistance* переменная *diffX* — это длина горизонтальной стороны треугольника, а *diffY* — длина вертикальной стороны.

Найти нужное нам расстояние — значит найти длину гипотенузы, зная длины *diffX* и *diffY*. Чтобы найти гипотенузу, сначала нужно возвести *diffX* и *diffY* в квадрат. Затем мы складываем эти значения и извлекаем из суммы квадратный корень с помощью JavaScript-функции *Math.sqrt*. Целиком формула для вычисления расстояния между кликом и кладом выглядит так:

```
Math.sqrt((diffX * diffX) + (diffY * diffY))
```

Функция *getDistance* вычисляет это выражение и возвращает результат.

Сообщаем игроку; насколько он близок к цели

Зная расстояние между кликом и кладом, остается отобразить подсказку, которая сообщала бы игроку, насколько близко он подошел, — но без конкретных цифр. Для этого создадим следующую функцию *getDistanceHint:*

```
var getDistanceHint = function(distance)
{
  if(distance < 10)
    return "Обожжешься!";
  else
    if(distance < 20)
      return "Очень горячо";
    else
      if(distance < 40)
        return "Горячо";
      else
        if(distance < 80)
          return "Тепло";
        else
          if(distance < 160)
            return "Холодно";
          else
            if(distance < 320)
              return "Очень холодно";
            else
              return "Замерзнешь!";
}
```

Эта функция возвращает одну из строк в зависимости от переданного ей расстояния до клада. Если это расстояние меньше *10*, она вернет «Обожжешься!». Для расстояния от *10* до *20* функция вернет «Очень горячо». По мере увеличения расстояния функция будет сообщать о все большем холоде, вплоть до расстояния в *320* пикселей, начиная с которого функция возвращает строку «Замерзнешь!».

Отображать эти сообщения мы будем, задавая их как текстовое содержимое элемента *p* на нашей странице. Следующий код вычислит расстояние, получит нужную строку с сообщением и отобразит эту строку:

```
distance = getDistance(event, target);
distanceHint = getDistanceHint(distance);
$("#distance").text(distanceHint);
```

Как видите, сначала мы вызываем *getDistance*, сохраняя возвращенное значение в переменной *distance*. Затем мы передаем *distance* в функцию *getDistanceHint*, чтобы получить соответствующую строку подсказки и сохранить ее в переменной *distanceHint*.

Код *$("#distance").text(distanceHint);* находит элемент, *id* которого равен "*distance*" (в нашем случае это элемент *p*), и меняет его текст на значение *distanceHint*, так что всякий раз при клике по карте наша веб-страничка сообщает игроку, насколько близко он подошел к цели.

Проверка на выигрыш

И наконец, наш обработчик кликов должен проверить, не попал ли игрок в цель. Поскольку один пиксель очень мал, мы не будем вынуждать игрока кликать в точности по месту с кладом, а засчитаем за победу клик на расстоянии менее восьми пикселей.

Этот код проверяет расстояние до клада, в случае победы сообщая об этом игроку:

```
if(distance < 8)
  alert("Клад найден! Сделано кликов: " + clicks);
```

Код игры

Теперь, когда у нас есть все части кода, объединим их в программу.

```
// Получить случайное число от 0 до size-1
var getRandomNumber = function(size)
{
  return Math.floor(Math.random() * size);
}
// Вычислить расстояние от клика (event) до клада (target)
var getDistance = function(event, target)
{
  var diffX = event.offsetX - target.x;
  var diffY = event.offsetY - target.y;
  return Math.sqrt((diffX * diffX) + (diffY * diffY));
}
// Получить для расстояния строку подсказки
var getDistanceHint = function(distance)
{
  if(distance < 10)
    return "Обожжешься!";
  else
    if(distance < 20)
      return "Очень горячо";
    else
      if(distance < 40)
        return "Горячо";
      else
        if(distance < 80)
          return "Тепло";
        else
```

```
        if(distance < 160)
          return "Холодно";
        else
          if(distance < 320)
            return "Очень холодно";
          else
            return "Замерзнешь!";
    }
    // Создаем переменные
    var width = 400;
    var height = 400;
    var clicks = 0;
    // Случайная позиция клада
    var target = {
      x: getRandomNumber(width),
      y: getRandomNumber(height)
    };
    // Добавляем элементу img обработчик клика
    $("#map").click(function(event)
    {
      clicks++;
      // Получаем расстояние от места клика до клада
      distance = getDistance(event, target);
      // Преобразуем расстояние в подсказку
      distanceHint = getDistanceHint(distance);
      // Записываем в элемент #distance новую подсказку
      $("#distance").text(distanceHint);
      // Если клик был достаточно близко, поздравляем с победой
      if(distance < 8)
      alert("Клад найден! Сделано кликов: " + clicks);
    });
```

Начинается код с функций *getRandomNumber*, *getDistance* и *getDistanceHint*, о которых мы уже говорили. Затем мы создали необходимые переменные: *width*, *height* и *clicks*. Далее задается случайная позиция для клада. Потом мы добавили элементу карты обработчик кликов. Первым делом этот обработчик увеличивает на *1* переменную *clicks*. Затем он вычисляет расстояние между *event* (местом клика) и *target* (позицией клада). Мы использовали функцию *getDistanceHint* для преобразования этого расстояния в строку ("Холодно", "Тепло" и т.д.). Далее мы обновляем подсказку на экране, чтобы игрок видел, насколько он близок к цели. И наконец, проверяем, уложился ли игрок в расстояние меньше *8* пикселей от клада, и если уложился, мы сообщаем ему о победе и количестве затраченных кликов.

Это весь JavaScript-код игры. Добавив его ко второму элементу *script* в файле *treasure.html*, вы сможете запустить игру в браузере! За сколько кликов вам удастся найти клад?