Если верить англо-русскому словарю, то можно узнать что canvas переводится как холст, а если верить википедии, то можно узнать что тег canvas, это элемент HTML 5, который предназначен для создания растрового изображения при помощи JavaScript. Тому как создать это растровое изображение и будет посвящен мой небольшой текст. Прежде чем начинать пробовать свои силы в этом не легком деле рекомендуется уже иметь базовые знания о том что такое HTML и с чем едят JavaScript.

Предварительная «настройка» нашего холста

У нашего подопытного тега есть всего два атрибута — height и width, высота и ширина соответственно, по умолчанию размер холста 150х300 пикселей.  
Стоит отметить что canvas создает область фиксированного размера содержимым которого управляют контексты.  
Элементарный пример:

<!doctype html>

<html>

<head>

<title>canvasExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas height='320' width='480' id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

ctx.fillRect(0, 0, example.width, example.height);

</script>

</body>

</html>

Если сохранить эти несчастные 13 строк в файл и открыть его браузером, то можно будет увидеть область с чёрным прямоугольником, так вот это и есть тот самый холст, на котором нарисован прямоугольник размеры которого равны размерам canvas'а.

Прямоугольники

Самой элементарной фигурой которую можно рисовать является прямоугольник. Предусмотрено три функции для отрисовки прямоугольников.

strokeRect(x, y, ширина, высота) // Рисует прямоугольник

fillRect(x, y, ширина, высота) // Рисует закрашенный прямоугольник

clearRect(x, y, ширина, высота) // Очищает область на холсте размер с прямоугольник заданного размера

Пример иллюстрирующий работу этих функций:

<!doctype html>

<html>

<head>

<title>rectExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.width = 640;

example.height = 480;

ctx.strokeRect(15, 15, 266, 266);

ctx.strokeRect(18, 18, 260, 260);

ctx.fillRect(20, 20, 256, 256);

for (i = 0; i < 8; i += 2)

for (j = 0; j < 8; j += 2) {

ctx.clearRect(20 + i \* 32, 20 + j \* 32, 32, 32);

ctx.clearRect(20 + (i + 1) \* 32, 20 + (j + 1) \* 32, 32, 32);

}

</script>

</body>

</html>

А теперь краткий построчный разбор:  
в строках 10 и 11 мы изменили размер холста — чтоб бы задуманное нами изображение полностью отобразилось,  
в строках 12 и 13 мы нарисовали два не закрашенных прямоугольника которые будут символизировать своеобразную рамку нашей «шахматной доски»,  
в строке 14 отрисовываем закрашенный прямоугольник размеры которого бы позволил вместить в себе 64 квадрата с шириной стороны 32 пикселя,  
в строках с 15 по 19 у нас работает два цикла которые очищают на чёрном прямоугольнике квадратные области в таком порядке что бы в итоге полученное изображение было похоже на шахматную доску

Линии и дуги

Рисование фигур составленных из линий выполняется последовательно в несколько шагов:

beginPath()

closePath()

stroke()

fill()

beginPath используется что бы «начать» серию действий описывающих отрисовку фигуры. Каждый новый вызов этого метода сбрасывает все действия предыдущего и начинает «рисовать» занова.  
closePath является не обязательным действием и по сути оно пытается завершить рисование проведя линию от текущей позиции к позиции с которой начали рисовать.  
Завершающий шаг это вызовом метода stroke или fill. Собственно первый обводит фигуру линиями, а второй заливает фигуру сплошным цветом.  
Те кто когда-то на школьных 486х в былые годы рисовал в бейсике домик, забор и деревце по задумке учителя тот сразу поймёт часть ниже. Итак, существуют такие методы как,

moveTo(x, y) // перемещает "курсор" в позицию x, y и делает её текущей

lineTo(x, y) // ведёт линию из текущей позиции в указанную, и делает в последствии указанную текущей

arc(x, y, radius, startAngle, endAngle, anticlockwise) // рисование дуги, где x и y центр окружности, далее начальный и конечный угол, последний параметр указывает направление

Пример ниже показывает действие всего описанного выше:

<!doctype html>

<html>

<head>

<title>pathExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.height = 480;

example.width = 640;

ctx.beginPath();

ctx.arc(80, 100, 56, 3/4 \* Math.PI, 1/4 \* Math.PI, true);

ctx.fill(); // \*14

ctx.moveTo(40, 140);

ctx.lineTo(20, 40);

ctx.lineTo(60, 100);

ctx.lineTo(80, 20);

ctx.lineTo(100, 100);

ctx.lineTo(140, 40);

ctx.lineTo(120, 140);

ctx.stroke(); // \*22

</script>

</body>

</html>

В строке 14 заливается цветом дуга, в строке 22 обводится контур нашей короны.

Кривые Бернштейна-Безье

Что такое кривые Безье я думаю лучше объяснит Википедия.   
Нам доступно две функции, для построения кубической кривой Бизье и квадратичной, соотвестствено:

quadraticCurveTo(Px, Py, x, y)

bezierCurveTo(P1x, P1y, P2x, P2y, x, y)

x и y это точки в которые необходимо перейти, а координаты P(Px, Py) в квадратичной кривой это дополнительные точки которые используются для построения кривой. В кубическо кривой соответственно две дополнительные точки.  
Пример двух кривых:

<!doctype html>

<html>

<head>

<title>curveExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.height = 480;

example.width = 640;

ctx.beginPath();

ctx.moveTo(10, 15);

ctx.bezierCurveTo(75, 55, 175, 20, 250, 15);

ctx.moveTo(10, 15);

ctx.quadraticCurveTo(100, 100, 250, 15);

ctx.stroke();

</script>

</body>

</html>

Добавим цвета

Что бы наше изображение было не только двух цветов, а любого цвета предусмотрено, два свойства 

fillStyle = color // определяет цвет заливки

strokeStyle = color // цвет линий цвет задается точно так же как и css, на примере все четыре способа задания цвета

Цвет задается точно так же как и css, на примере все четыре способа задания цвета

// все четыре строки задают оранжевый цвет заливки

ctx.fillStyle = "orange";

ctx.fillStyle = "#FFA500";

ctx.fillStyle = "rgb(255,165,0)";

ctx.fillStyle = "rgba(255,165,0,1)"

Аналогично задаётся и цвет для линий.   
Возьмём пример с шахматной доской и добавим в него немного цвета:

<!doctype html>

<html>

<head>

<title>rectExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.height = 480;

example.width = 640;

ctx.strokeStyle = '#B70A02'; // меняем цвет рамки

ctx.strokeRect(15, 15, 266, 266);

ctx.strokeRect(18, 18, 260, 260);

ctx.fillStyle = '#AF5200'; // меняем цвет клеток

ctx.fillRect(20, 20, 256, 256);

for (i = 0; i < 8; i += 2)

for (j = 0; j < 8; j += 2) {

ctx.clearRect(20 + i \* 32, 20 + j \* 32, 32, 32);

ctx.clearRect(20 + (i + 1) \* 32, 20 + (j + 1) \* 32, 32, 32);

}

</script>

</body>

</html>

 научимся использовать изображения. Естественно рисовать на холсте примитивами очень не удобно и требует определённых трудозатрат, и результат иногода явно хромает качеством. Поэтому естественно в canvas api предусмотрено взаимодействие с изображениями. Добавление изображения условно можно разделить на два шага: создание JavaScript объекта Image, а второй и заключительный шаг это отрисовка изображения на холсте при помощи функции drawImage. Рассмотрим оба шага подробнее.  
  
Создание нового графического объекта: 

var img = new Image(); // Создание нового объекта изображения

img.src = 'image.png'; // Путь к изображению которое необходимо нанести на холст

Кстати в качестве источника изображения, можно указать вот такую строку в которой изображение и описанно:

img.src = 'data:image/gif;base64,';

1. drawImage

Теперь мы перейдём к рисованию изображения на холсте. Для этого существует функция drawImage.

drawImage(image, x, y) // Где x и y это координаты левого верхнего угла изображения, а первый параметр это изображение

Стоит отметить что загрузка изображения происходит сразу после присвоения объекту источника изображения, и если оно не загрузится полностью к моменту вызова функции отрисовки, то оно попросту не будет нарисовано на холсте. Для избежания этой ситуации используется такая конструкция: 

var img = new Image(); // Новый объект

img.onload = function() { // Событие которое будет исполнено в момент когда изображение будет загружено

/\*

Какие-либо действия

\*/

}

img.src = 'myImage.png'; // Путь к изображению

Вот мы наверное и дошли до момента когда можно рассмотреть элементарный пример: 

<!doctype html>

<html>

<meta charset='utf-8'>

<head>

<title>imgExample</title>

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d'), // Контекст

pic = new Image(); // "Создаём" изображение

pic.src = 'http://habrahabr.ru/i/nocopypast.png'; // Источник изображения, позаимствовано на хабре

pic.onload = function() { // Событие onLoad, ждём момента пока загрузится изображение

ctx.drawImage(pic, 0, 0); // Рисуем изображение от точки с координатами 0, 0

}

</script>

</body>

</html>

Ссылка на [фидл с примерами](http://jsfiddle.net/shpaker/uwCFS/) для этой стать.

2. Тянем-потянем

Но если бы всё ограничивалось простым рисованием изображением, то отдельную статью можно было бы не писать, а ограничиться подпунктом «Изображения» в предыдущем посте. Итак теперь мы попытаемся масштабировать изображение и для этого существует ещё один способ вызова функции drawImage: 

drawImage(image, x, y, width, height) // параметры width, height меняют ширину и высоту изображения

Возьмём предыдущий пример и внесём в drawImage некоторые изменения: 

ctx.drawImage(pic, 0, 0, 300, 150);

Ссылка на [фидл с примерами](http://jsfiddle.net/shpaker/uwCFS/) для этой стать.

3. Рисуем фрагмент изображения

Третий вызов drawImage с восемью параметрами, выглядит приблизительно так: 

drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);

// Первый параметр указывает на изображение

// sx, sy, sWidth, sHeight указывают параметры фрагмента на изображение-источнике

// dx, dy, dWidth, dHeight ответственны за координаты отрисовки фрагмента на холсте

Возьмём всё тот же пример и подкорректируем функцию drawImage в третий раз: 

ctx.drawImage(pic, 25, 42, 85, 55, 0, 0, 170, 110);

4. Задача

Теперь осталось закрепить всё пройденное на практическом примере. Как и в том примере это будет небольшая карта, только не из какой-то существовавшей игры, а придуманной нами. Для того что бы он нормально работал нам необходимо будет создать в любом графическом редакторе изображение составленное из фрагментов которые нам будут нужны что бы нарисовать дорожку, домик и полянку. Вообще фрагменты карты называются тайлами, а файл в котором они все собраны в одно изображение называется тайлсетом. Вот это изображение я нарисовал собственноручно в программе Pinta под Ubuntu.  
  
Итак, размерность будет 8 на 8 квадратных блоков шириной 32 пиксела. На карте нам необходимо будет изобразить домик и дорожку. Элементы домика нарисованы ручками, можно сказать каждый кирпичик. Траву и песок делалось путём заливки облости соответствующим цветом и добавки шума. Всё конечно представленно очень примитивно, но показательно.  
Рассмотрим с пинцетом такой кусок кода как var map = [[{x:1, y: 4}… значения x и y указывают какой элемент из картинки брать. Т.е. если исходный рисунок разбить на квадрат 32×32 то станет понятней.  
И форэкзампл: 

<!doctype html>

<html>

<meta charset='utf-8'>

<head>

<title>imgExample</title>

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById('example'), // Задаём контекст

ctx = example.getContext('2d'), // Контекст холста

cellSize = 32, // Размер одной ячейки на карте

pic = new Image(), // "Создаём" изображение

map = // Карта уровня двумерным массивом

[

[{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4}], // 1ый ряд

[{x:1,y:4},{x:1,y:1},{x:2,y:1},{x:3,y:1},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4}], // 2ый ряд

[{x:1,y:4},{x:1,y:2},{x:2,y:2},{x:3,y:2},{x:1,y:4},{x:1,y:3},{x:1,y:3},{x:1,y:3}], // 3ый ряд

[{x:1,y:4},{x:3,y:4},{x:2,y:3},{x:3,y:4},{x:1,y:4},{x:1,y:3},{x:1,y:4},{x:1,y:4}], // 4ый ряд

[{x:1,y:4},{x:3,y:4},{x:2,y:4},{x:3,y:4},{x:1,y:4},{x:1,y:3},{x:1,y:4},{x:1,y:4}], // 5ый ряд

[{x:1,y:4},{x:1,y:4},{x:1,y:3},{x:1,y:4},{x:1,y:4},{x:1,y:3},{x:1,y:4},{x:1,y:4}], // 6ый ряд

[{x:1,y:4},{x:1,y:4},{x:1,y:3},{x:1,y:3},{x:1,y:3},{x:1,y:3},{x:1,y:4},{x:1,y:4}], // 7ый ряд

[{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4},{x:1,y:4}] // 8ый ряд

]; // Первая и вторая координата (x и y соответственно) задают фрагмент в исходном изображении

// Размер холста равный 8х8 клеток

example.width = 8 \* cellSize;

example.height = 8 \* cellSize;

pic.src = 'http://habrastorage.org/getpro/habr/post\_images/e85/727/cb1/e85727cb1a88099325eaf5b243d4c41f.png';

pic.onload = function() { // Событие onLoad, ждём момента пока загрузится изображение

for (var j = 0 ; j < 8; j ++)

for (var i = 0; i < 8; i ++)

// перебираем все значения массива 'карта' и в зависимости от координат вырисовываем нужный нам фрагмент

ctx.drawImage(pic, (map[i][j].x-1)\*cellSize, (map[i][j].y-1)\*cellSize, 32, 32, j\*cellSize, i\*cellSize, 32, 32);

}

</script>

</body>

</html>