

Бесплатный урок: рисуем первого героя на JavaScript

Материал создан проектом [Инди Код](#) – обучение программированию через маленькие понятные проекты.

В этом уроке ты нарисуешь первого героя на **JavaScript** с помощью **canvas**.

Сначала нарисуем простое тело, а потом постепенно добавим уши, глаза, усы и другие детали.

Главное здесь сразу сделать код удобным. В процессе мы будем возвращаться к уже написанным частям и менять их, поэтому размер, цвет и позицию героя лучше хранить в одном месте, а не искать числа по всему файлу.

Начинаем с файла `index-start.html`. Внутри него уже есть `canvas` и подготовленное место для твоего кода:

```
// === НАЧАЛО: ваш код ===  
// Стартовое место для урока.  
// Всё из урока пишем между НАЧАЛОМ и КОНЦОМ.  
// Эту подсказку можно удалить перед первым шагом.  
// === КОНЕЦ: ваш код ===
```

Всё, что мы будем писать в уроке, добавляй между `// === НАЧАЛО: ваш код ===` и `// === КОНЕЦ: ваш код ===`. Подсказки посередине можно удалить перед первым шагом.

Код выполняется сверху вниз: браузер читает первую строку, потом вторую, потом третью. Поэтому сначала мы записываем данные героя, а уже потом пишем команды, которые эти данные используют.

Для начала просто перепечатай код из урока в стартовый файл. Так ты быстрее привыкнешь к тому, как пишется JavaScript, и увидишь, как каждая новая строка влияет на результат.

Шаг 1. Сначала описываем героя

Сначала создаём `hero` — место, где будут лежать основные данные нашего персонажа.

Такие места в коде называют переменными. Переменную мы объявляем с помощью `const`, а `hero` — это имя, которое мы сами ей даём.

Внутри `hero` мы положим объект. Объект записывается в фигурных скобках `{}` и может хранить сразу несколько данных.

Данные внутри объекта называют свойствами. Их имена и значения мы придумываем сами:

- `x` и `y` — место героя на холсте. В `canvas` отсчёт начинается от левого верхнего угла: `x` показывает расстояние слева, а `y` — расстояние сверху.
- `radius` — размер тела.
- `color` — цвет тела.

Позже мы сможем обращаться к этим значениям через имя объекта и точку: например, `hero.x`, `hero.radius` или `hero.color`.

```
const hero = {  
  x: 250,  
  y: 150,  
  radius: 45,  
  color: '#111111'  
};
```

Пока на экране ещё ничего нет, но важная часть уже готова: герой описан в одном месте. Теперь его проще двигать, увеличивать и перекрашивать.

Шаг 2. Рисуем тело героя

Теперь берём значения из `hero` и рисуем круг. Это будет тело героя.

Здесь мы впервые используем готовые команды `canvas`. Их не нужно писать с нуля: они уже есть в браузере. В начале стартового файла я уже подготовил `canvas` и положил инструмент для рисования в переменную `ctx`, поэтому дальше мы просто вызываем команды через `ctx`.

Коротко:

- `canvas` — область на странице, где можно рисовать.
- `ctx` — инструмент для рисования внутри `canvas`.
- Команды с круглыми скобками, например `beginPath()` и `fill()`, запускают готовые действия.

- Значения внутри скобок подсказывают команде, что именно нужно сделать.

Теперь добавь код:

```
ctx.beginPath();
ctx.arc(hero.x, hero.y, hero.radius, 0, Math.PI * 2);
ctx.fillStyle = hero.color;
ctx.fill();
```

Разберём, что здесь происходит:

- `ctx.beginPath()` — говорим браузеру: начинаем новую фигуру.
- `ctx.arc(hero.x, hero.y, hero.radius, 0, Math.PI * 2)` — рисуем круг. Первые три значения берём из `hero`: место по `x`, место по `y` и размер.
- `ctx.fillStyle = hero.color` — выбираем цвет заливки. Здесь мы берём цвет из `hero.color`.
- `ctx.fill()` — закрашиваем фигуру выбранным цветом.

На экране появился всего один круг, но это уже не случайная фигура. Это тело героя, связанное с объектом `hero`.

Теперь можно сразу поиграть с цветом. Цвет котика задан как `#111111` — это HEX-запись цвета. В ней используются цифры от 0 до 9 и буквы от A до F. Попробуй заменить любой символ после `#` на другое значение от 0 до F, сохрани файл и посмотри, как изменится котик.

Если хочешь подобрать цвет руками, открой [Google Color Picker](#) и скопируй значение, которое начинается с `#`.

Должно получиться так:



Результат второго шага

Если что-то не совпало, сравни свой код с готовым файлом `index-step2.html`. В нём показано, как должен выглядеть код после второго шага.

Шаг 3. Выносим код в функции и рисуем уши

Дальше деталей станет больше: уши, глаза, нос, рот, усы. Если просто добавлять строки одну за другой, код быстро превратится в длинный список команд.

Чтобы этого не случилось, начнём собирать код в функции.

Функция — это кусок кода с именем. Мы один раз описываем, что она делает, а потом можем вызвать её по имени.

Начнём с кода, который уже рисовал круг. Теперь это будет тело героя, поэтому обернём этот код в функцию `drawBody()`.

Для этого над кодом круга дописываем строку `function drawBody() {`, а снизу закрываем функцию фигурной скобкой `}`:

```
function drawBody() {  
  ctx.beginPath();  
  ctx.arc(hero.x, hero.y, hero.radius, 0, Math.PI * 2);  
  ctx.fillStyle = hero.color;  
  ctx.fill();  
}
```

Обрати внимание: когда мы пишем `function drawBody() { ... }`, мы только создаём функцию. Код внутри неё сам по себе ещё не запускается.

Чтобы функция сработала, её нужно вызвать:

```
drawBody();
```

Теперь сделаем ещё одну функцию — drawHero(). Она будет главной: внутри неё мы будем собирать героя из маленьких частей.

После этого код с телом героя выглядит так:

```
function drawBody() {
  ctx.beginPath();
  ctx.arc(hero.x, hero.y, hero.radius, 0, Math.PI * 2);
  ctx.fillStyle = hero.color;
  ctx.fill();
}

function drawHero() {
  drawBody();
}

drawHero();
```

Смысл такой: в конце файла мы вызываем только drawHero(), а уже внутри неё вызываются все части героя.

Теперь добавим уши. Левое и правое ухо почти одинаковые, только смотрят в разные стороны. Поэтому не будем писать две отдельные функции.

Сделаем одну функцию drawEar(direction). Слово direction означает направление:

- -1 — рисуем влево от центра героя.
- 1 — рисуем вправо от центра героя.

Добавь функцию уха после drawBody():

```
function drawEar(direction) {
  ctx.beginPath();
  ctx.moveTo(hero.x + direction * 35, hero.y - 30);
  ctx.lineTo(hero.x + direction * 20, hero.y - 75);
  ctx.lineTo(hero.x + direction * 5, hero.y - 35);
  ctx.closePath();
  ctx.fillStyle = hero.color;
  ctx.fill();

  ctx.beginPath();
  ctx.moveTo(hero.x + direction * 27, hero.y - 36);
  ctx.lineTo(hero.x + direction * 20, hero.y - 60);
  ctx.lineTo(hero.x + direction * 12, hero.y - 38);
  ctx.closePath();
  ctx.fillStyle = '#ffffff';
  ctx.fill();
}
```

Здесь появились новые команды canvas:

- `moveTo(...)` — ставит точку, откуда начнём рисовать линию.
- `lineTo(...)` — проводит линию к новой точке.
- `closePath()` — закрывает фигуру, соединяя последнюю точку с первой.

Ухо рисуется как треугольник из трёх точек:

```
ctx.moveTo(hero.x + direction * 35, hero.y - 30);  
ctx.lineTo(hero.x + direction * 20, hero.y - 75);  
ctx.lineTo(hero.x + direction * 5, hero.y - 35);
```

Первая строка ставит начальную точку. Вторая строка проводит линию от первой точки ко второй. Третья строка проводит ещё одну линию — от второй точки к третьей.

После этого `ctx.closePath()` закрывает треугольник: соединяет третью точку обратно с первой.

Каждая точка считается от центра героя:

- `hero.x` — центр героя по горизонтали.
- `hero.y` — центр героя по вертикали.
- `+ 35, + 20, + 5` — насколько точка уходит в сторону от центра.
- `- 30, - 75, - 35` — насколько точка поднимается вверх от центра.

`direction` помогает использовать один и тот же код для двух ушей. Когда `direction` равен 1, выражение `direction * 35` даёт 35, и точка уходит вправо. Когда `direction` равен -1, выражение `direction * 35` даёт -35, и точка уходит влево.

Так один и тот же треугольник можно нарисовать с двух сторон от головы.

После большого треугольника мы рисуем ещё один треугольник поменьше. Он работает по той же логике, но точки стоят ближе друг к другу, а цвет заливки белый:

```
ctx.fillStyle = '#ffffff';  
ctx.fill();
```

Так у уха появляется внутренняя светлая часть.

Теперь обнови drawHero():

```
function drawHero() {  
  drawBody();  
  drawEar(-1);  
  drawEar(1);  
}  
  
drawHero();
```

Сначала рисуется тело, потом левое ухо, потом правое. Порядок важен: браузер рисует сверху вниз, как и читает код.

Должно получиться так:



Результат третьего шага

Если что-то не совпало, сравни свой код с готовым файлом index-step3.html. В нём показано, как должен выглядеть код после третьего шага.

Высоту ушек можно менять через вторую точку треугольника:

```
ctx.lineTo(hero.x + direction * 20, hero.y - 75);
```

Чем больше число после минуса, тем выше поднимется верхняя точка уха. Например, hero.y - 90 сделает ушки выше, а hero.y - 60 — ниже.

Шаг 4. Рисуем глаза

Теперь добавим глаза. Они устроены проще, чем уши: каждый глаз — это два круга.

Сначала рисуем большой белый круг, а поверх него маленький чёрный круг. Белый круг будет самым глазом, чёрный — зрачком.

Для этого создадим функцию `drawEye(direction)`. Здесь снова используем `direction`, потому что левый и правый глаз отличаются только положением:

```
function drawEye(direction) {
  ctx.beginPath();
  ctx.arc(hero.x + direction * 16, hero.y - 10, 7, 0, Math.PI * 2);
  ctx.fillStyle = '#ffffff';
  ctx.fill();

  ctx.beginPath();
  ctx.arc(hero.x + direction * 16, hero.y - 10, 3, 0, Math.PI * 2);
  ctx.fillStyle = '#000000';
  ctx.fill();
}
```

Разберём первую команду `arc`:

```
ctx.arc(hero.x + direction * 16, hero.y - 10, 7, 0, Math.PI * 2);
```

Здесь первые два значения задают место глаза:

- `hero.x + direction * 16` — сдвигает глаз влево или вправо от центра героя.
- `hero.y - 10` — поднимает глаз немного выше центра.

Третье значение — это размер круга. У белого круга размер 7, а у зрачка размер 3.

Оба круга рисуются в одном месте. Разница только в размере и цвете: сначала большой белый круг, потом маленький чёрный.

Теперь обновим `drawHero()`:

```
function drawHero() {
  drawBody();
  drawEar(-1);
  drawEar(1);
  drawEye(-1);
  drawEye(1);
}

drawHero();
```

Сначала рисуем тело и уши, а потом глаза. Если поменять порядок, некоторые детали могут оказаться под другими фигурами.

Должно получиться так:



Результат четвёртого шага

Если что-то не совпало, сравни свой код с готовым файлом `index-step4.html`. В нём показано, как должен выглядеть код после четвёртого шага.

Шаг 5. Добавляем нос

Нос — это ещё один маленький круг. Он находится по центру головы, но чуть ниже глаз.

Добавь функцию `drawNose()`:

```
function drawNose() {  
  ctx.beginPath();  
  ctx.arc(hero.x, hero.y + 8, 6, 0, Math.PI * 2);  
  ctx.fillStyle = '#ffffff';  
  ctx.fill();  
}
```

Здесь `hero.x` остаётся без сдвига, потому что нос находится по центру. А `hero.y + 8` опускает нос немного ниже центра головы.

Теперь обнови `drawHero()`:

```
function drawHero() {  
  drawBody();  
  drawEar(-1);  
  drawEar(1);  
  drawEye(-1);  
  drawEye(1);  
  drawNose();  
}  
  
drawHero();
```

Должно получиться так:



Результат пятого шага

Если что-то не совпало, сравни свой код с готовым файлом `index-step5.html`.

Шаг 6. Рисуем рот

Рот нарисуем не заливкой, а линией. Для этого используем `stroke()` — он не закрашивает фигуру, а обводит её.

Добавь функцию `drawMouth()`:

```
function drawMouth() {
  ctx.beginPath();
  ctx.arc(hero.x - 6, hero.y + 14, 6, 0, Math.PI);
  ctx.arc(hero.x + 6, hero.y + 14, 6, 0, Math.PI);
  ctx.strokeStyle = '#ffffff';
  ctx.lineWidth = 2;
  ctx.stroke();
}
```

Здесь два маленьких полукруга стоят рядом:

- `hero.x - 6` — левая часть рта.
- `hero.x + 6` — правая часть рта.
- `hero.y + 14` — рот находится ниже носа.
- `Math.PI` — рисуем не полный круг, а половину.

`strokeStyle` задаёт цвет линии, а `lineWidth` — толщину линии.

Обнови drawHero():

```
function drawHero() {  
  drawBody();  
  drawEar(-1);  
  drawEar(1);  
  drawEye(-1);  
  drawEye(1);  
  drawNose();  
  drawMouth();  
}  
  
drawHero();
```

Должно получиться так:



Результат шестого шага

Если что-то не совпало, сравни свой код с готовым файлом [index-step6.html](#).

Шаг 7. Добавляем усы

Усы тоже рисуются линиями. С каждой стороны будет по три линии.

Снова используем `direction`, чтобы одна функция рисовала и левую, и правую сторону.

Добавь функцию `drawWhiskers(direction)`:

```
function drawWhiskers(direction) {
  ctx.beginPath();
  ctx.moveTo(hero.x + direction * 8, hero.y + 8);
  ctx.lineTo(hero.x + direction * 25, hero.y);
  ctx.moveTo(hero.x + direction * 8, hero.y + 12);
  ctx.lineTo(hero.x + direction * 25, hero.y + 12);
  ctx.moveTo(hero.x + direction * 8, hero.y + 16);
  ctx.lineTo(hero.x + direction * 25, hero.y + 25);
  ctx.strokeStyle = '#ffffff';
  ctx.lineWidth = 2;
  ctx.stroke();
}
```

Здесь каждая пара `moveTo(...)` и `lineTo(...)` рисует один ус:

- первая пара рисует верхний ус;
- вторая пара рисует средний ус;
- третья пара рисует нижний ус.

Когда вызываем `drawWhiskers(-1)`, усы уходят влево. Когда вызываем `drawWhiskers(1)`, усы уходят вправо.

Обнови `drawHero()`:

```
function drawHero() {
  drawBody();
  drawEar(-1);
  drawEar(1);
  drawEye(-1);
  drawEye(1);
  drawNose();
  drawMouth();
  drawWhiskers(-1);
  drawWhiskers(1);
}

drawHero();
```

Должно получиться так:



Результат седьмого шага

Если что-то не совпало, сравни свой код с готовым файлом `index-step7.html`.

Шаг 8. Добавляем хвост и обводку

Финальная деталь — хвост. Его тоже можно нарисовать дугой, только теперь мы используем толстую линию.

Добавь функцию `drawTail()`:

```
function drawTail() {  
  ctx.beginPath();  
  ctx.arc(hero.x + 42, hero.y + 25, 22, -Math.PI / 2, Math.PI / 1.5);  
  ctx.strokeStyle = '#111111';  
  ctx.lineWidth = 10;  
  ctx.stroke();  
}
```

Здесь `arc(...)` рисует не полный круг, а дугу:

- `hero.x + 42` — хвост начинается справа от тела.
- `hero.y + 25` — хвост немного ниже центра.
- `22` — размер дуги.
- `-Math.PI / 2` и `Math.PI / 1.5` — откуда и докуда идёт дуга.

Ещё добавим обводку тела, чтобы котик выглядел аккуратнее:

```
function drawBodyOutline() {  
  ctx.strokeStyle = '#000000';  
  ctx.lineWidth = 4;  
  ctx.stroke();  
}
```

Эту функцию вызываем сразу после drawBody(), потому что она обводит уже нарисованное тело.

Финальный drawHero() выглядит так:

```
function drawHero() {  
  drawTail();  
  
  drawBody();  
  drawBodyOutline();  
  
  drawEar(-1);  
  drawEar(1);  
  
  drawEye(-1);  
  drawEye(1);  
  
  drawNose();  
  drawMouth();  
  
  drawWhiskers(-1);  
  drawWhiskers(1);  
}  
  
drawHero();
```

Должно получиться так:



Финальный результат

Если что-то не совпало, сравни свой код с готовым файлом `index-step8.html`.

Что дальше

В этом уроке ты нарисовал первого героя на JavaScript: начал с круга, вынес код в функции, добавил уши, глаза, нос, рот, усы и хвост.

Это маленький кусок того, что будет в пошаговом курсе по созданию игры.

В игре главный герой окажется в лабиринте. Ему нужно успеть выбраться за отведённое время: найти ключ от двери, добраться до выхода и при этом не попасться в ловушки. Звучит просто, но на деле это уже настоящая игровая задача: движение, стены, столкновения, состояние игры, победа, проигрыш и несколько уровней.

Курс подойдёт тем, кто хочет попробовать программирование и понять, нравится ли разработка в принципе.

Мы будем писать на JavaScript так же, как в текущем уроке: без установки сложных программ. Понадобятся только обычный редактор текста и веб-браузер.

Примерная программа курса:

1. Создаём html-файл с нужной структурой, открываем его в браузере, рисуем холст и первую фигуру по центру экрана. Результат — нарисованный круг.
2. Заставляем круг двигаться по кнопкам, делаем рефакторинг, исправляем баги и собираем структуру из функций. Результат — движение игрока во все стороны и игровой цикл.
3. Добавляем стены, через которые нельзя пройти, создаём коллизии, добавляем ключик и состояние игры. Результат — окружение и базовая игровая логика.
4. Создаём лабиринт: спавн героя, дверь, карту игры, логику появления игрока, взятие ключа и победу после выхода через дверь. Результат — готовая первая версия игры.
5. Добавляем ловушки с анимацией в лабиринте. Результат — игра становится сложнее и интереснее.
6. Добавляем таймер обратного отсчёта, экран Game Over, кнопку перезапуска и экран победы.
7. Добавляем второй уровень и переход на него после победы. Заодно показываем точки расширения, чтобы игру можно было развивать дальше.
8. Бонусный урок: создаём главное меню, с которого игрок сможет начать игру.

Если тебе было интересно пройти этот урок и хочется собрать уже не просто героя, а полноценную игру с движением, лабиринтом, ловушками, ключом и победой — переходи по ссылке ниже.

Там будет подробнее о курсе, программа и возможность оставить заявку, чтобы получить новости и первые материалы.

[Подробнее о курсе](#)