Advanced animations

← Previous Next →

In the last chapter we made some basic animations and got to know ways to get things moving. In this part we will have a closer look at the motion itself and are going to add some physics to make our animations more advanced.

Drawing a ball

We are going to use a ball for our animation studies, so let's first draw that ball onto the canvas. The following code will set us up.

HTML

```html
<canvas id="canvas" width="600" height="300"></canvas>
```

As usual, we need a drawing context first. To draw the ball, we will create a `ball` object which contains properties and a `draw()` method to paint it on the canvas.

JS

```js
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");

const ball = {
  x: 100,
  y: 100,
  radius: 25,
  color: "blue",
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fillStyle = this.color;
    ctx.fill();
  },
};

ball.draw();
```

Nothing special here, the ball is actually a simple circle and gets drawn with the help of the `arc()` method.

Adding velocity

Now that we have a ball, we are ready to add a basic animation like we have learned in the last chapter of this tutorial. Again, `window.requestAnimationFrame()` helps us to control the animation. The ball gets moving by adding a velocity vector to the position. For each frame, we also clear the canvas to remove old circles from prior frames.

JS

```js
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let raf;

const ball = {
  x: 100,
  y: 100,
  vx: 5,
  vy: 2,
  radius: 25,
  color: "blue",
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fillStyle = this.color;
    ctx.fill();
  },
};

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ball.draw();
  ball.x += ball.vx;
  ball.y += ball.vy;
  raf = window.requestAnimationFrame(draw);
}

canvas.addEventListener("mouseover", (e) => {
  raf = window.requestAnimationFrame(draw);
});

canvas.addEventListener("mouseout", (e) => {
  window.cancelAnimationFrame(raf);
});

ball.draw();
```

Boundaries

Without any boundary collision testing our ball runs out of the canvas quickly. We need to check if the `x` and `y` position of the ball is out of the canvas dimensions and invert the direction of the velocity vectors. To do so, we add the following checks to the `draw` method:

JS

```js
if (
  ball.y + ball.vy > canvas.height - ball.radius ||
  ball.y + ball.vy < ball.radius
) {
  ball.vy = -ball.vy;
}
if (
  ball.x + ball.vx > canvas.width - ball.radius ||
  ball.x + ball.vx < ball.radius
) {
  ball.vx = -ball.vx;
}
```

First demo

Let's see how it looks in action so far.

HTML

HTML

```html
<canvas id="canvas" width="600" height="300"></canvas>
```

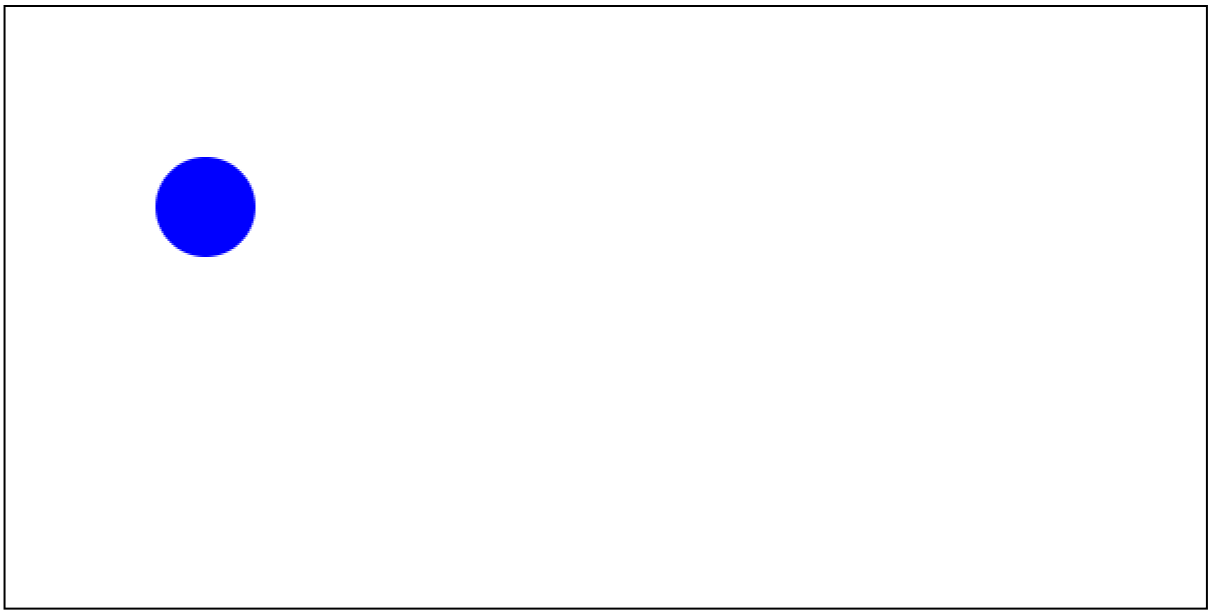
JAVASCRIPT

JS

```js
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let raf;

const ball = {
  x: 100,
  y: 100,
  vx: 5,
  vy: 2,
  radius: 25,
  color: "blue",
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fillStyle = this.color;
    ctx.fill();
  },
};

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ball.draw();
  ball.x += ball.vx;
  ball.y += ball.vy;

  if (
    ball.y + ball.vy > canvas.height - ball.radius ||
    ball.y + ball.vy < ball.radius
  ) {
    ball.vy = -ball.vy;
  }
  if (
    ball.x + ball.vx > canvas.width - ball.radius ||
    ball.x + ball.vx < ball.radius
  ) {
    ball.vx = -ball.vx;
  }

  raf = window.requestAnimationFrame(draw);
}

canvas.addEventListener("mouseover", (e) => {
  raf = window.requestAnimationFrame(draw);
});

canvas.addEventListener("mouseout", (e) => {
  window.cancelAnimationFrame(raf);
});

ball.draw();
```

RESULT

Move your mouse into the canvas to start the animation.



Acceleration

To make the motion more real, you can play with the velocity like this, for example:

JS

```js
ball.vy *= 0.99;
ball.vy += 0.25;
```

This slows down the vertical velocity each frame, so that the ball will just bounce on the floor in the end.

Second demo

HTML

HTML

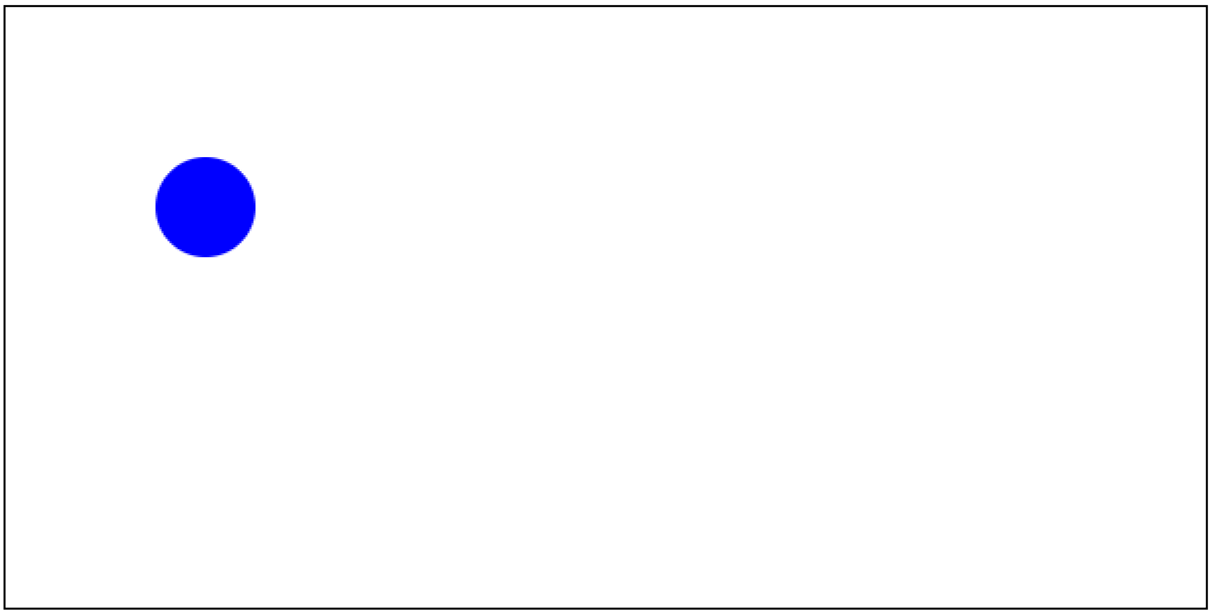
```html
<canvas id="canvas" width="600" height="300"></canvas>
```

JAVASCRIPT

JS

```
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let raf;

const ball = {
  x: 100,
  y: 100,
  vx: 5,
  vy: 2,
  radius: 25,
  color: "blue",
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fillStyle = this.color;
    ctx.fill();
  },
};

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ball.draw();
  ball.x += ball.vx;
  ball.y += ball.vy;
  ball.vy *= 0.99;
  ball.vy += 0.25;

  if (
    ball.y + ball.vy > canvas.height - ball.radius ||
    ball.y + ball.vy < ball.radius
  ) {
    ball.vy = -ball.vy;
  }
  if (
    ball.x + ball.vx > canvas.width - ball.radius ||
    ball.x + ball.vx < ball.radius
  ) {
    ball.vx = -ball.vx;
  }

  raf = window.requestAnimationFrame(draw);
}

canvas.addEventListener("mouseover", (e) => {
  raf = window.requestAnimationFrame(draw);
});

canvas.addEventListener("mouseout", (e) => {
  window.cancelAnimationFrame(raf);
});

ball.draw();
```

RESULT



Trailing effect

Until now we have made use of the `clearRect` method when clearing prior frames. If you replace this method with a semi-transparent `fillRect`, you can easily create a trailing effect.

JS

```js
ctx.fillStyle = "rgb(255 255 255 / 30%)";
ctx.fillRect(0, 0, canvas.width, canvas.height);
```

Third demo

HTML

HTML

```html
<canvas id="canvas" width="600" height="300"></canvas>
```

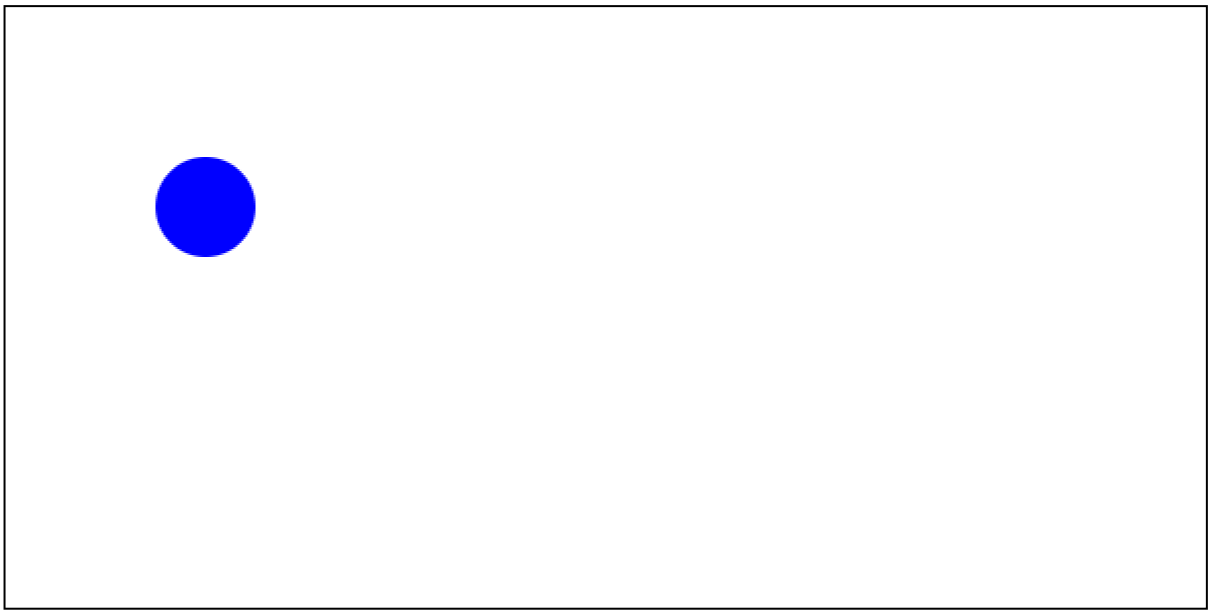
JAVASCRIPT

JS

```
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let raf;

const ball = {
  x: 100,
  y: 100,
  vx: 5,
  vy: 2,
  radius: 25,
  color: "blue",
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fillStyle = this.color;
    ctx.fill();
  },
};

function draw() {
  ctx.fillStyle = "rgb(255 255 255 / 30%)";
  ctx.fillRect(0, 0, canvas.width, canvas.height);
  ball.draw();
  ball.x += ball.vx;
  ball.y += ball.vy;
  ball.vy *= 0.99;
  ball.vy += 0.25;

  if (
    ball.y + ball.vy > canvas.height - ball.radius ||
    ball.y + ball.vy < ball.radius
  ) {
    ball.vy = -ball.vy;
  }
  if (
    ball.x + ball.vx > canvas.width - ball.radius ||
    ball.x + ball.vx < ball.radius
  ) {
    ball.vx = -ball.vx;
  }

  raf = window.requestAnimationFrame(draw);
}
```

```
canvas.addEventListener("mouseover", (e) => {
  raf = window.requestAnimationFrame(draw);
});

canvas.addEventListener("mouseout", (e) => {
  window.cancelAnimationFrame(raf);
});

ball.draw();
```

RESULT



Adding mouse control

To get some control over the ball, we can make it follow our mouse using the `mousemove` event, for example. The `click` event releases the ball and lets it bounce again.

Fourth demo

HTML

HTML

```
<canvas id="canvas" width="600" height="300"></canvas>
```

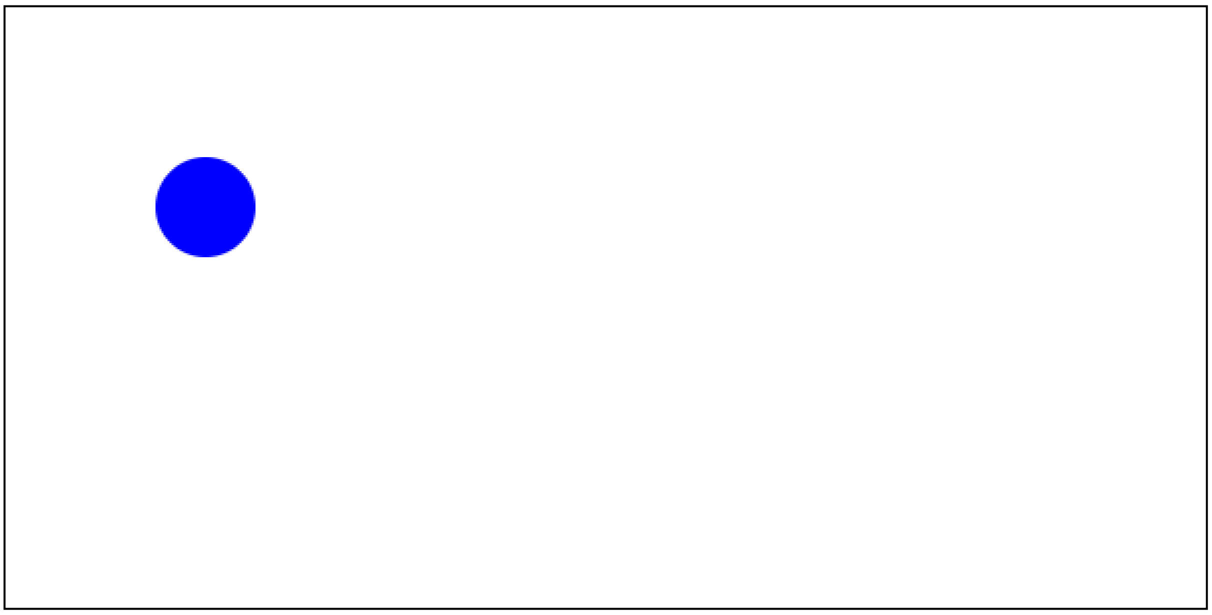
JAVASCRIPT

JS

```js
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let raf;
let running = false;

const ball = {
  x: 100,
  y: 100,
  vx: 5,
  vy: 1,
  radius: 25,
  color: "blue",
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fillStyle = this.color;
    ctx.fill();
  },
};

function clear() {
  ctx.fillStyle = "rgb(255 255 255 / 30%)";
  ctx.fillRect(0, 0, canvas.width, canvas.height);
}

function draw() {
  clear();
  ball.draw();
  ball.x += ball.vx;
  ball.y += ball.vy;

  if (
    ball.y + ball.vy > canvas.height - ball.radius ||
    ball.y + ball.vy < ball.radius
  ) {
    ball.vy = -ball.vy;
  }
  if (
    ball.x + ball.vx > canvas.width - ball.radius ||
    ball.x + ball.vx < ball.radius
  ) {
    ball.vx = -ball.vx;
  }
  raf = window.requestAnimationFrame(draw);
}

canvas.addEventListener("mousemove", (e) => {
  if (!running) {
    clear();
    ball.x = e.clientX;
    ball.y = e.clientY;
    ball.draw();
  }
});

canvas.addEventListener("click", (e) => {
  if (!running) {
    raf = window.requestAnimationFrame(draw);
    running = true;
  }
});

canvas.addEventListener("mouseout", (e) => {
  window.cancelAnimationFrame(raf);
  running = false;
});

ball.draw();
```

RESULT

Move the ball using your mouse and release it with a click.



Breakout

This short chapter only explains some techniques to create more advanced animations. There are many more! How about adding a paddle, some bricks, and turn this demo into a Breakout game? Check out our Game development area for more gaming related articles.

See also

- `window.requestAnimationFrame()`

← Previous Next →