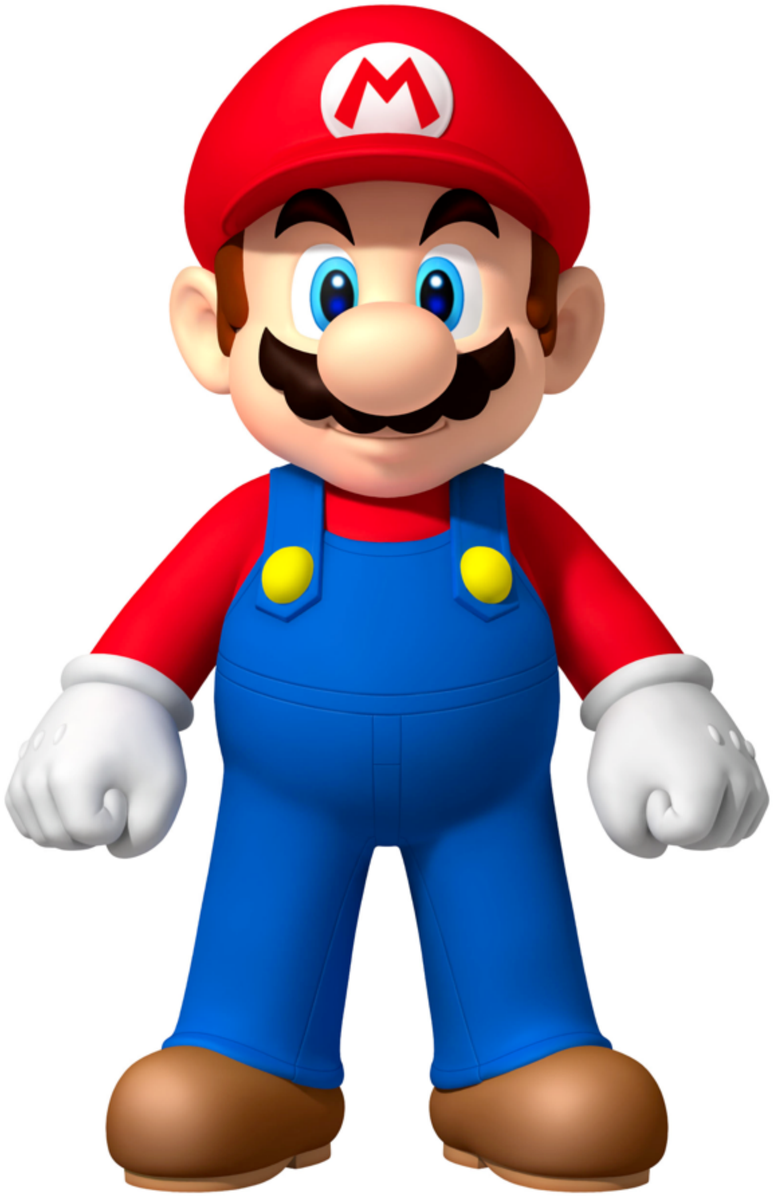
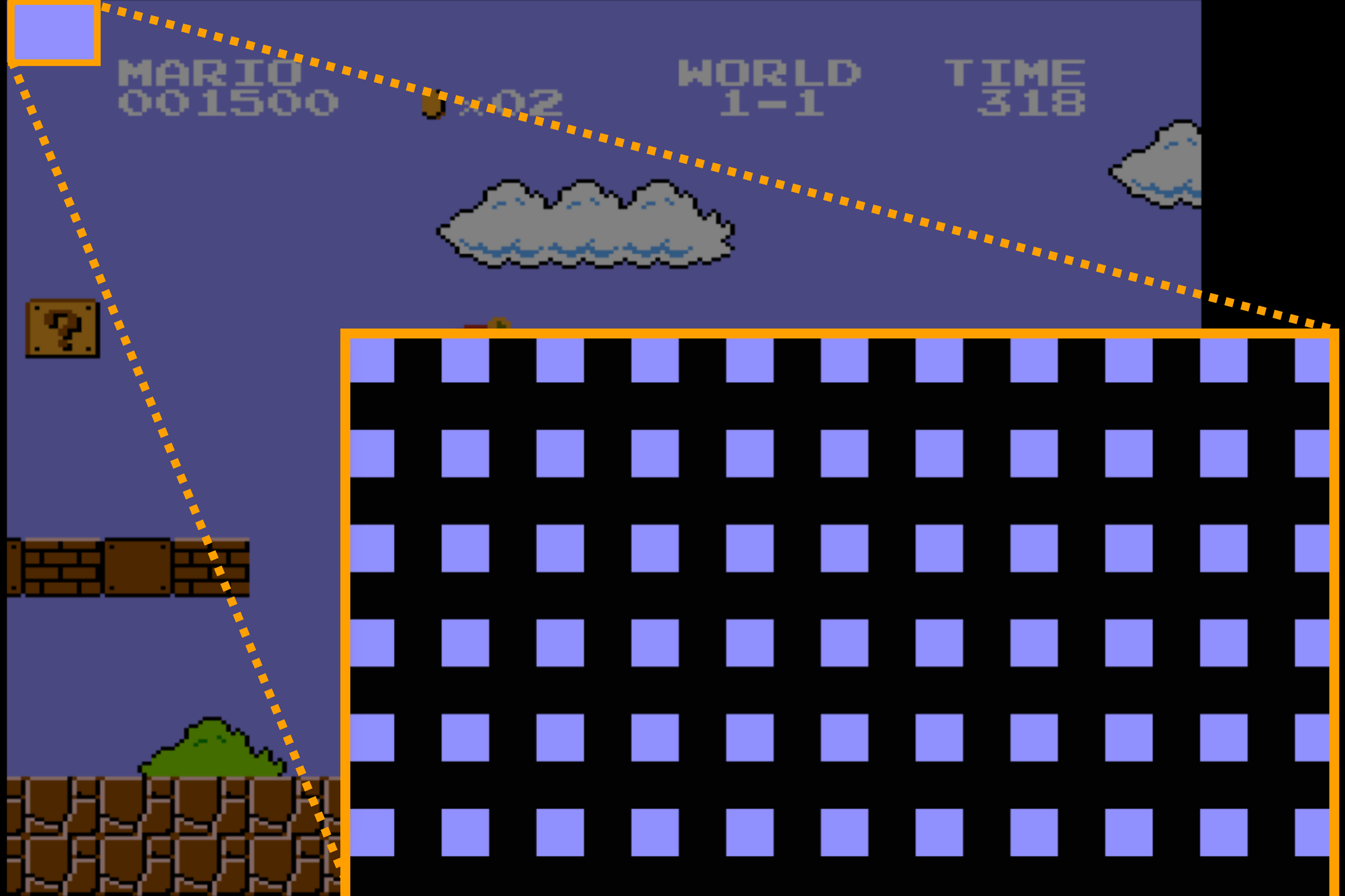
Graphics for Web Devs

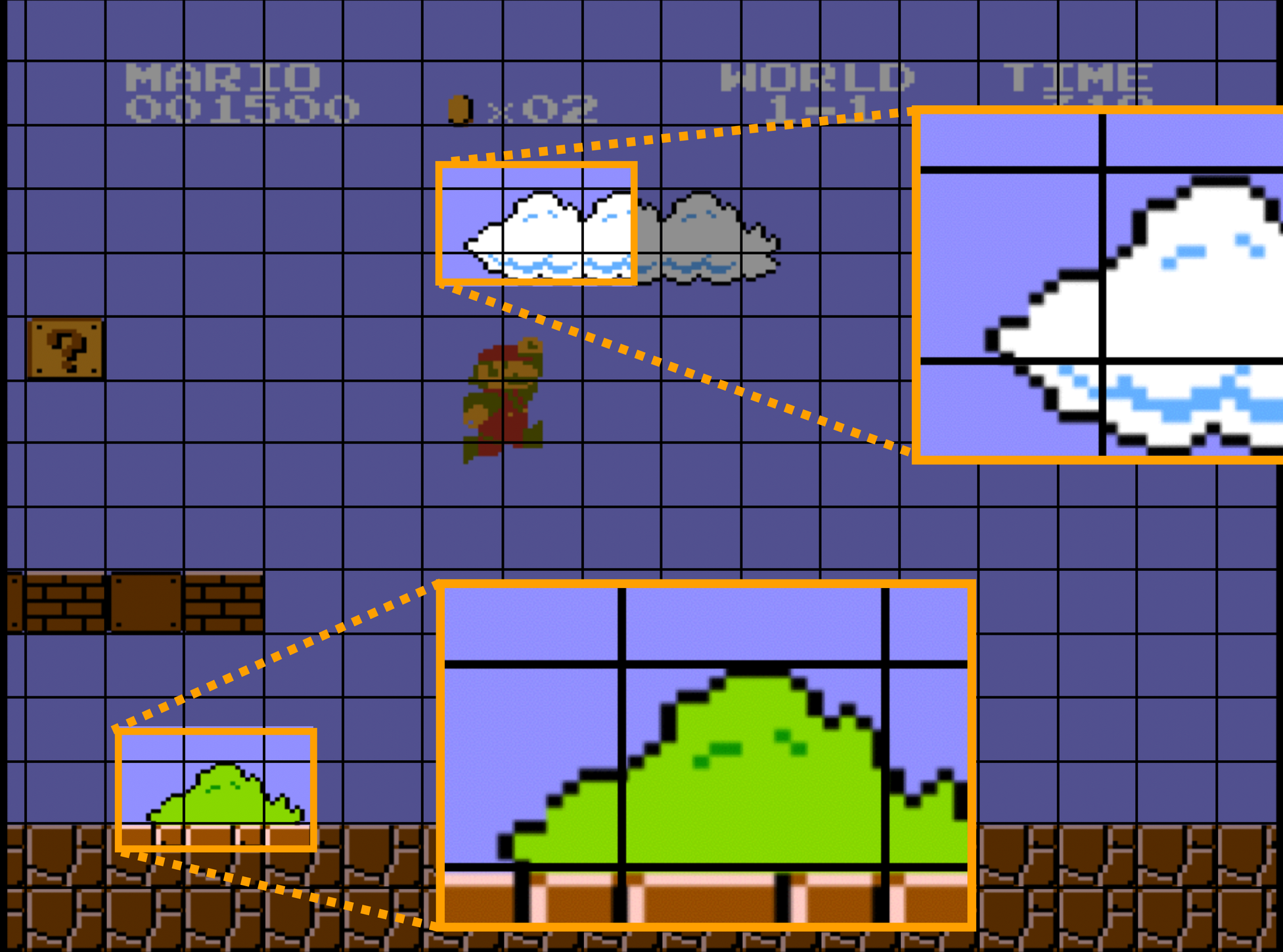
Jarrod Overson - Shape Security
@jsoverson

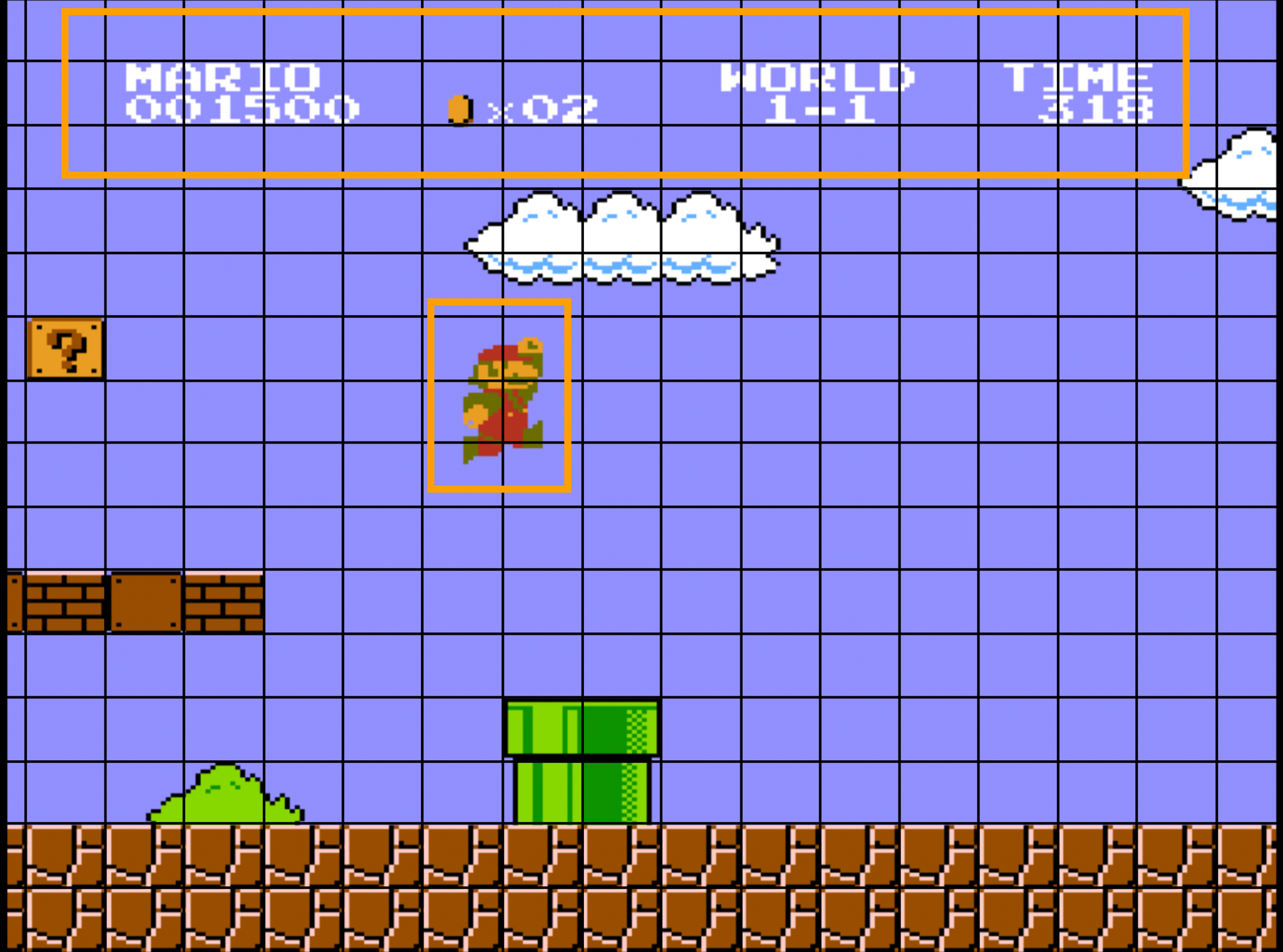


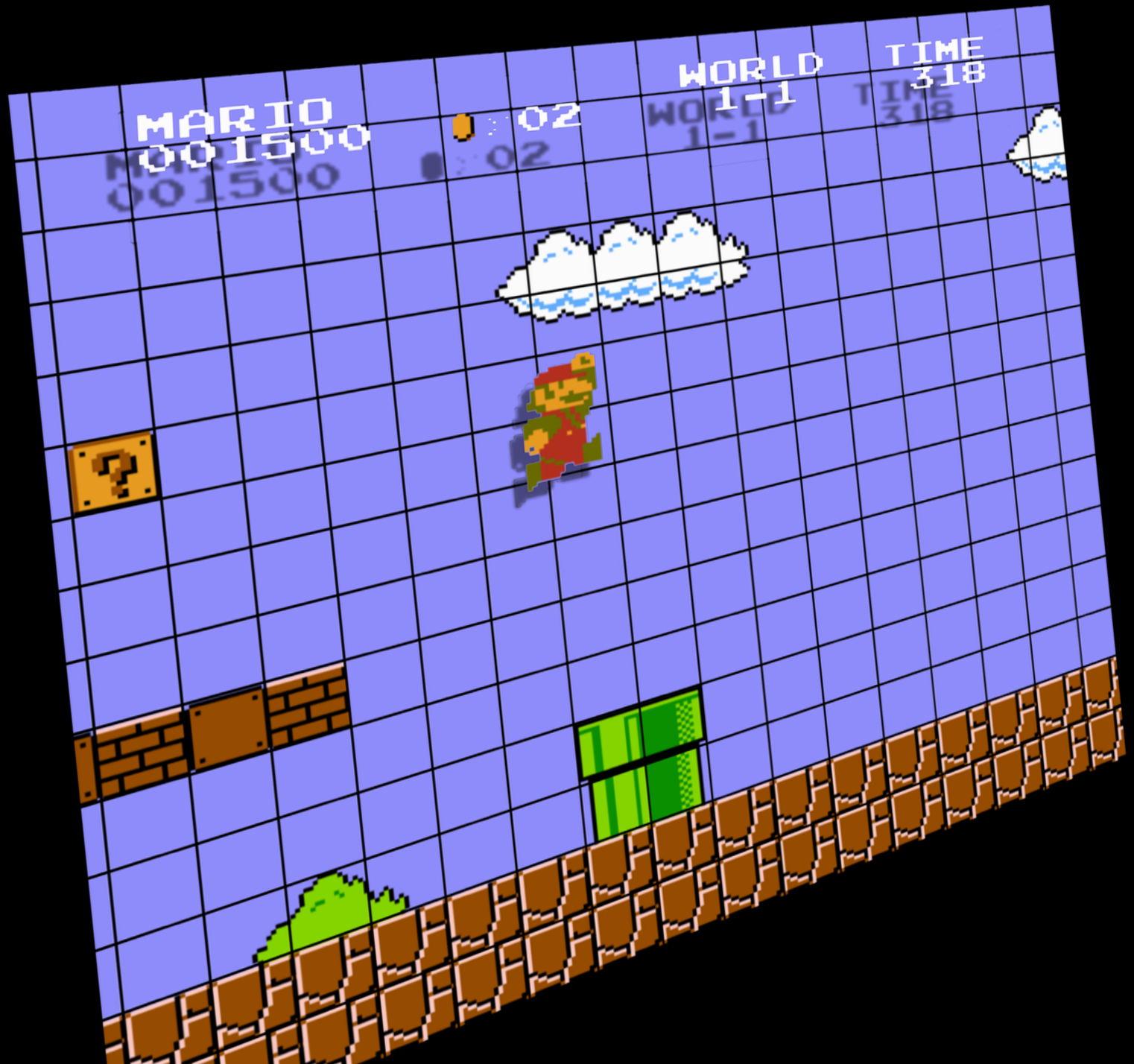

MARIO
001500
×02
WORLD
1-1
TIME
318

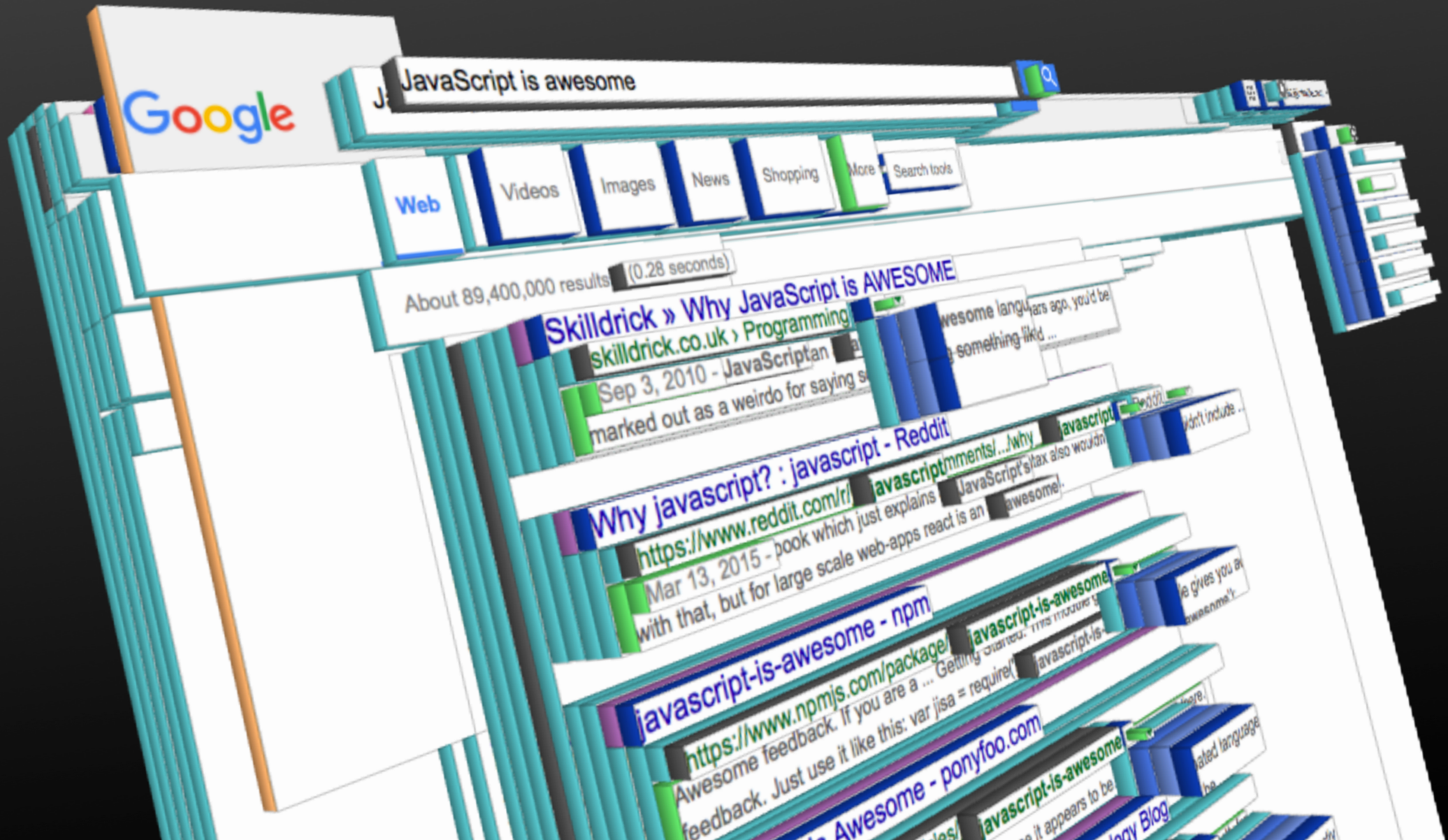

MARIO
001500
×02
WORLD
1-1
TIME
318


MARIO
000500
×02
WORLD
1-1
TIME
318


MARIO
000150O
×02
WORLD
1-1
TIME


MARIO
001500
×02
WORLD
1-1
TIME
318


MARIO
001500
02
WORLD
1-1
TIME
318


Google
JavaScript is awesome
Web
Videos
Images
News
Shopping
More
Search tools
About 89,400,000 results (0.28 seconds)
Skilldrick » Why JavaScript is AWESOME
skilldrick.co.uk › Programming
Sep 3, 2010 - JavaScriptan
marked out as a weirdo for saying s
Why javascript? : javascript - Reddit
https://www.reddit.com/r/
avascript
mments/.../why_
Mar 13, 2015 - book which just explains
JavaScript's
with that, but for large scale web-apps react is an
awesome.
javascript-is-awesome - npm
https://www.npmjs.com/package/
javascript-is-awesome
Awesome feedback. If you are a ... Getting Started. This module
feedback. Just use it like this: var jisa = require('
javascript-is-
Awesome - ponyfoo.com
javascript-is-awesome
javascript-is-awesome

Yes, we're not game developers.

But what are you looking for in your applications?

- 60 FPS
- Beautiful graphics
- Responsive Interaction
- Smooth animations

When I say "video games"

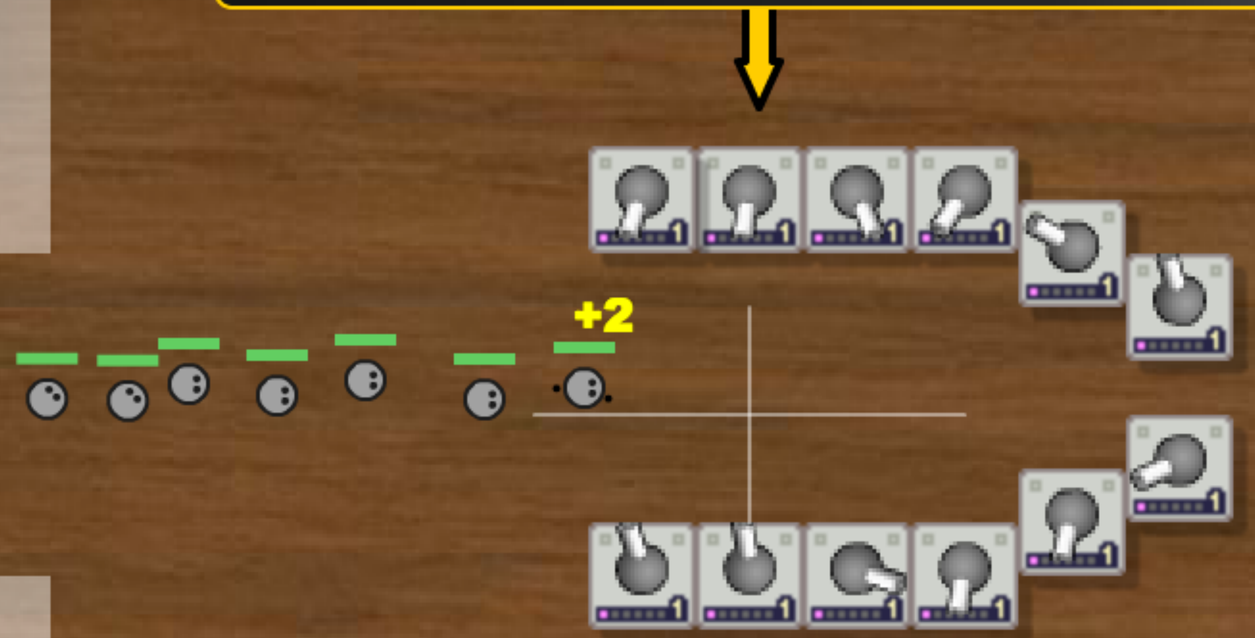
think: "performance critical graphic applications"

The web is one of the only computer platforms that hasn't grown with video games pushing the edge.

Why?

1996

Towers can be upgraded to make them more powerful. First, click on a tower to select it...

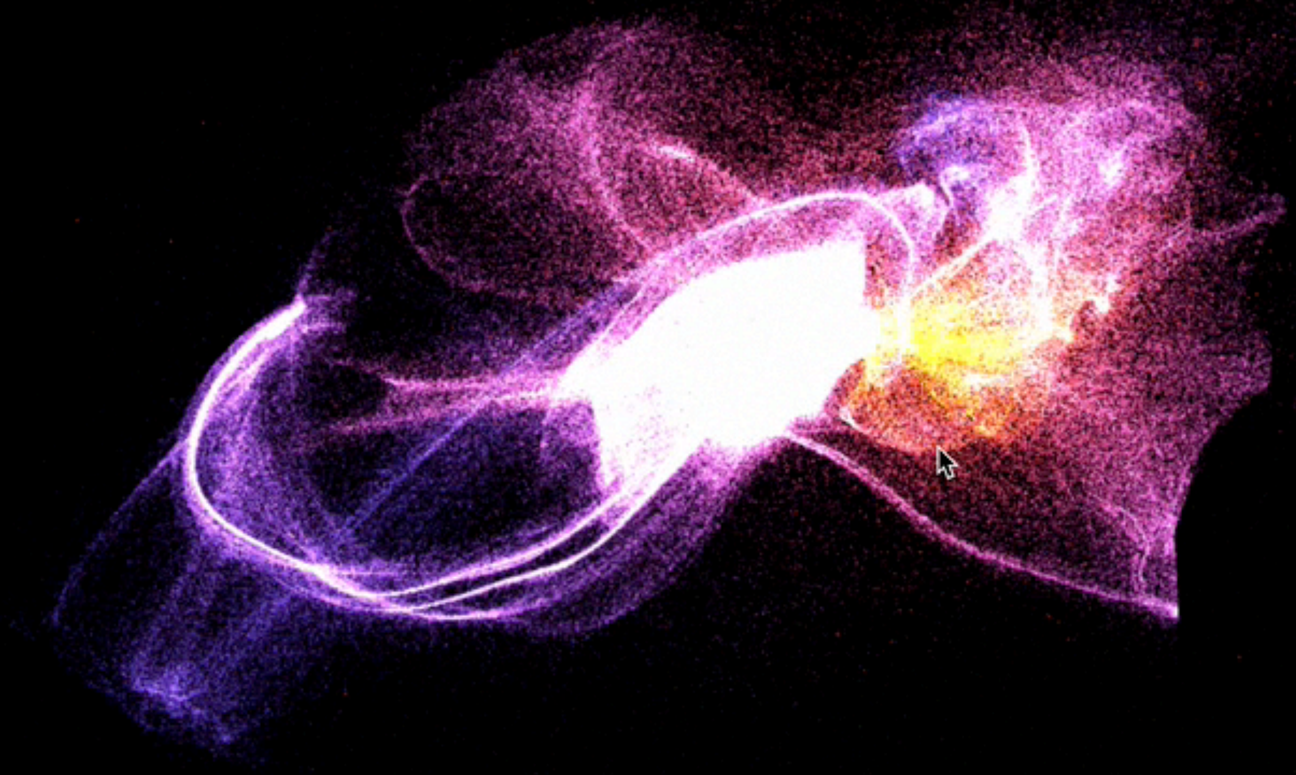


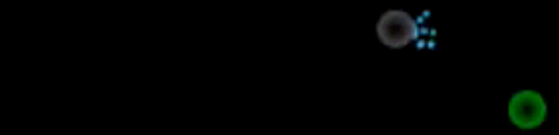
Towers


BEJEWELED
SCORE
00000000
NEW GAME
SIMPLE
TIMED
TAKE A HINT
UPGRADE
BONUS

The web may not need games.

But losing that discipline has handicapped us.





So where do we start?

<canvas>

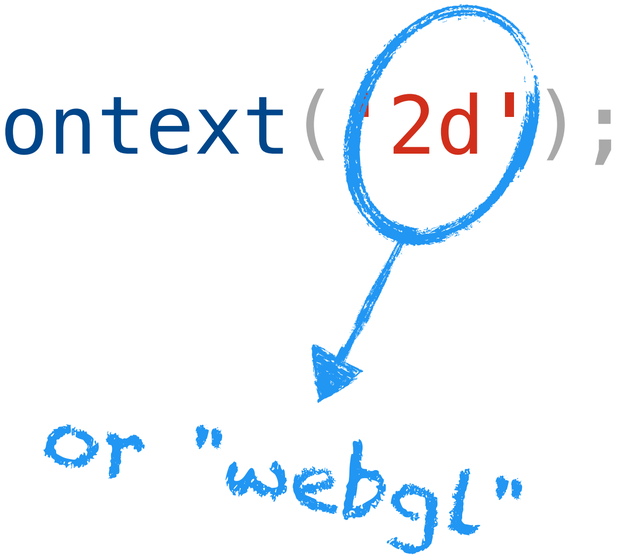
```
<canvas>
```

or

```
document.createElement('canvas');
```

```
canvas.width = 1280;
canvas.height = 720;
```

```
var context = canvas.getContext('2d');
```



```
context.fillStyle = 'MediumAquaMarine';
context.fillRect(10, 10, 100, 100);
```

```
for (var i = 0; i < 500; i++) {
 context.fillRect(10 + i, 10 + i, 100, 100);
}
```

```
for (var i = 0; i < 500; i++) {
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillRect(10 + i, 10 + i, 100, 100);
}
```

```
setTimeout(draw, 1000/60)
```

```
requestAnimationFrame(draw)
```

```
var i = 0;

function draw() {
  if (i < 500) i++;

  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillRect(i + 10, i + 10, 100, 100);

  requestAnimationFrame(draw);
}

draw();
```

First we update our state.

```
var i = 0;

function draw() {
  if (i < 500) i++;

  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillRect(i + 10, i + 10, 100, 100);

  requestAnimationFrame(draw);
}

draw();
```

Then we render our current state.

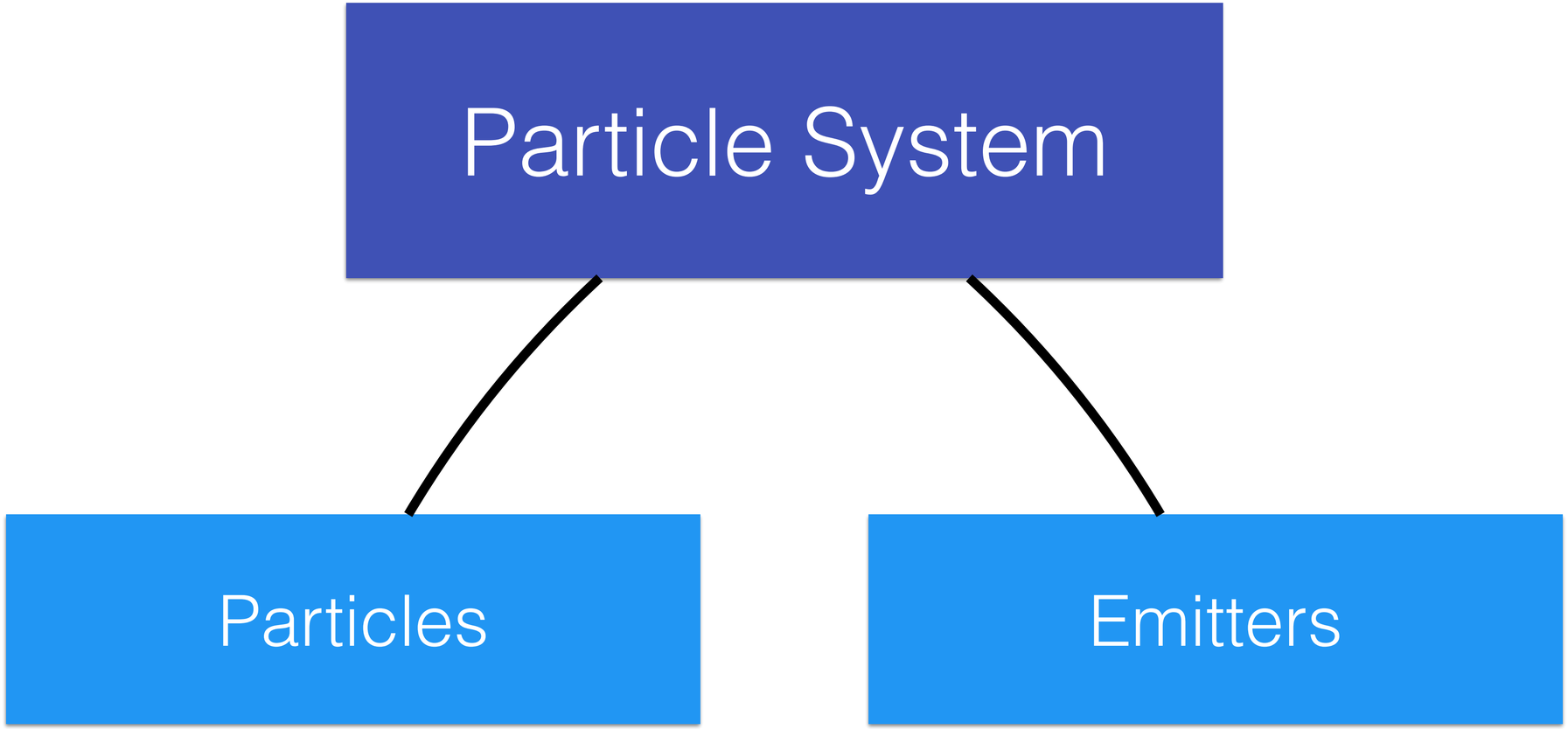
```
var i = 0;

function draw() {
  if (i < 500) i++;

  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillRect(i + 10, i + 10, 100, 100);

  requestAnimationFrame(draw);
}

draw();
```

And we do it all over again.

```
var i = 0;

function draw() {
  if (i < 500) i++;

  context.clearRect(0, 0, canvas.width, canvas.height);
  context.fillRect(i + 10, i + 10, 100, 100);

  requestAnimationFrame(draw);
}

draw();
```

Our loop

```
function loop() {

  update();
  render();

  requestAnimationFrame(loop);
}

loop();
```


Particle System
Particles
Emitters

```
class ParticleSystem {

  constructor() {
    this.particles = [];
    this.emitters = [];
  }

}
```

```
class Particle {

  constructor(position, velocity, acceleration) {
    this.position = position;
    this.velocity = velocity;
    this.acceleration = acceleration;
  }

}
```

```
class Particle {

  constructor(position, velocity, acceleration) {
    this.position = position || {x: 0, y: 0};
    this.velocity = velocity || {x: 0, y: 0};
    this.acceleration = acceleration || {x: 0, y: 0};
  }

}
```

```
class Vector {

  constructor(x, y) {
    this.x = x || 0;
    this.y = y || 0;
  }

}
```

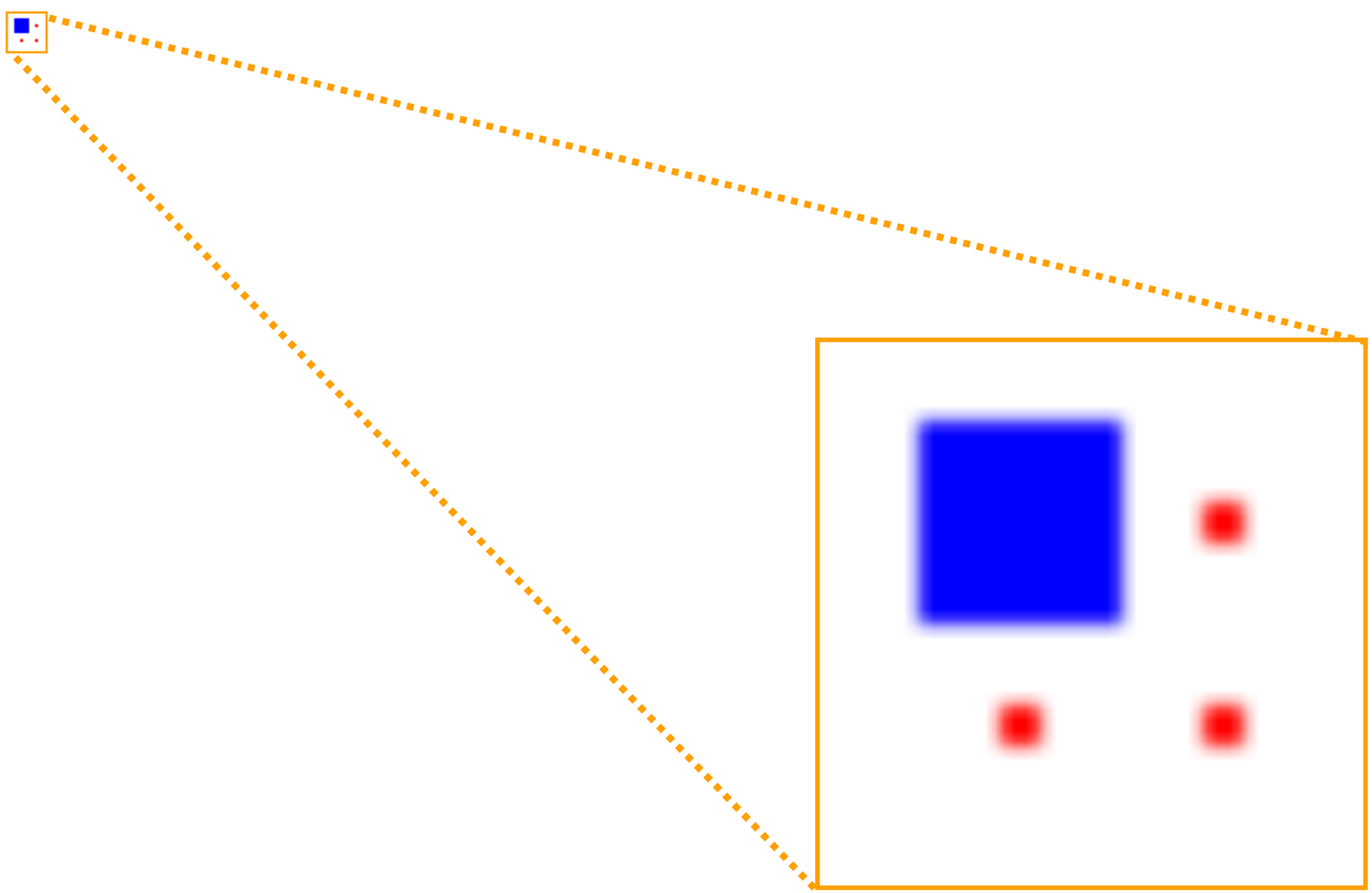
```
class Particle {

  constructor(position, velocity, acceleration) {
    this.position = position || new Vector(0, 0);
    this.velocity = velocity || new Vector(0, 0);
    this.acceleration = acceleration || new Vector(0, 0);
  }

}
```

```
class Emitter {

  constructor(point, velocity) {
    this.position = point || new Vector(0, 0);
    this.velocity = velocity || new Vector(0, 0);
  }

}
```

```
class ParticleSystem {

  constructor(width, height) { /* …snipped… */ }

  render(context) {
    context.clearRect(0, 0, this.width, this.height);

    this.emitters.forEach(emitter => emitter.render(context));

    this.particles.forEach(particle => particle.render(context));
  }

}
```

```
class Particle {
  constructor(position, velocity, acceleration) { /* … */ }

  render(context) {
    context.fillStyle = 'red';
    context.fillRect(this.position.x, this.position.y, 2, 2);
  }

}
```

```
var system = new ParticleSystem(width, height);

system.emitters.push(new Emitter(new Vector(10, 10)));
system.particles.push(new Particle(new Vector(20, 20)));
system.particles.push(new Particle(new Vector(10, 20)));
system.particles.push(new Particle(new Vector(20, 10)));

function loop() {
  system.render(context);
  window.requestAnimationFrame(loop);
}
loop();
```



```
var system = new ParticleSystem(width, height);

system.addEmitter(320, 180);

function loop() {
  system.update();
  system.render(context);
  window.requestAnimationFrame(loop);
}

loop();
```

```
class Emitter {

  constructor(position, velocity) { /* … */ }

  render(context) { /* … */ }

  emitParticle() {
    return new Particle(this.position.clone(), this.velocity.clone());
  }

}
```

```
class Particle {

  constructor(position, velocity, acceleration) { /* … */ }

  render(context) { /* … */ }

  update() {
    this.velocity.add(this.acceleration);
    this.position.add(this.velocity);
  }

}
```

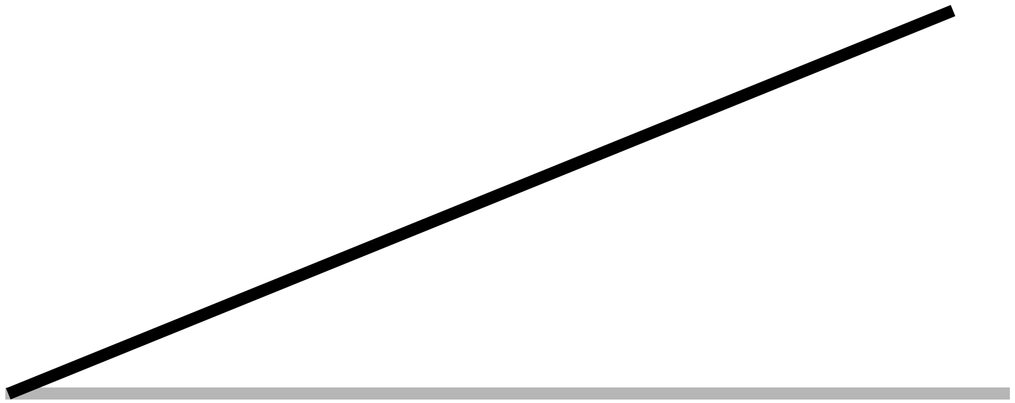
```
class ParticleSystem {
  constructor(width, height) { /* … */ }

  addEmitter(x, y, velX, velY) { /* … */ }

  update() {
    this.emitters.forEach(
      emitter => this.particles.push(emitter.emitParticle())
    );

    this.particles = this.particles.filter(particle => {
      particle.update();
      let pos = particle.position;
      let outOfBounds = pos.x > this.width || pos.y > this.height ||
                        pos.x < 0 || pos.y < 0
      return !outOfBounds;
    })
  }
}
```

```
emitParticle() {
  // Define a maximum angle range in radians.
  let spread = Math.PI / 32;

  // Use an angle randomized over a spread so we have more of a "spray"
  var angle = this.velocity.rad() + spread - (Math.random() * spread * 2);

  // The magnitude of the emitter's velocity
  var magnitude = this.velocity.length();

  // New velocity based off of the calculated angle and magnitude
  var velocity = Vector.fromAngle(angle, magnitude);

  // return our new Particle
  return new Particle(this.position.clone(), velocity);
}
```

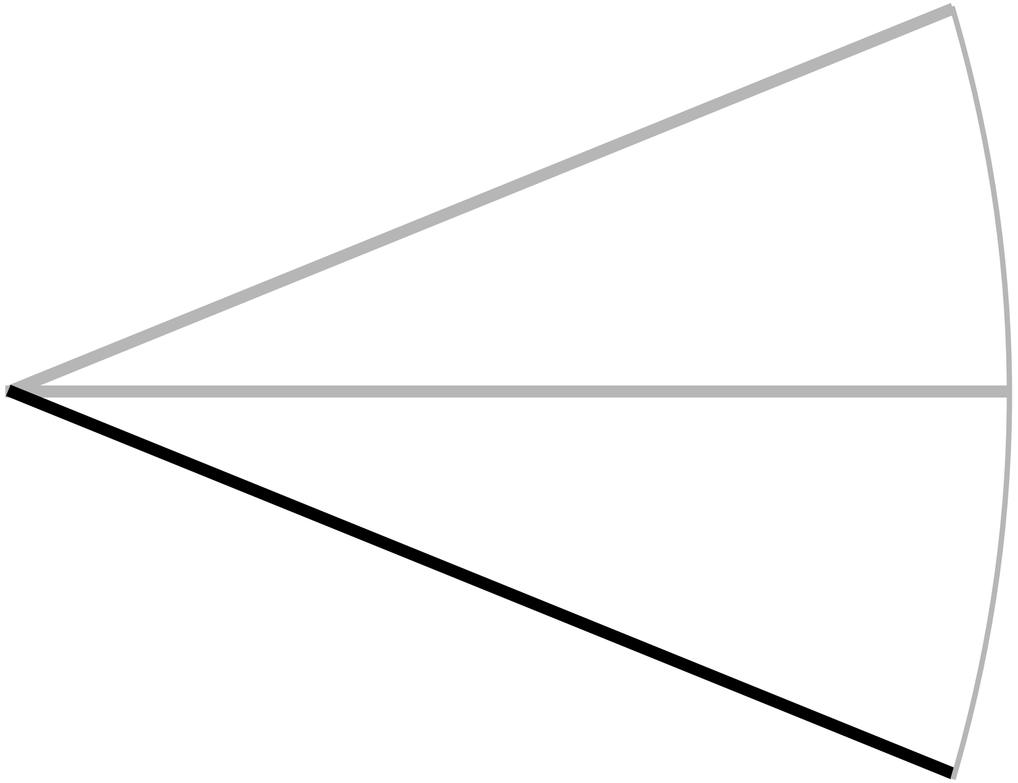
```
emitParticle() {
  // Define a maximum angle range in radians.
  let spread = Math.PI / 32;

  // Use an angle randomized over a spread so we have more of a "spray"
  var angle = this.velocity.rad() + spread - (Math.random() * spread * 2);

  // The magnitude of the emitter's velocity
  var magnitude = this.velocity.length();

  // New velocity based off of the calculated angle and magnitude
  var velocity = Vector.fromAngle(angle, magnitude);

  // return our new Particle
  return new Particle(this.position.clone(), velocity);
}
```

```
emitParticle() {
  // Define a maximum angle range in radians.
  let spread = Math.PI / 32;

  // Use an angle randomized over a spread so we have more of a "spray"
  var angle = this.velocity.rad() + spread - (Math.random() * spread * 2);

  // The magnitude of the emitter's velocity
  var magnitude = this.velocity.length();

  // New velocity based off of the calculated angle and magnitude
  var velocity = Vector.fromAngle(angle, magnitude);

  // return our new Particle
  return new Particle(this.position.clone(), velocity);
}
```

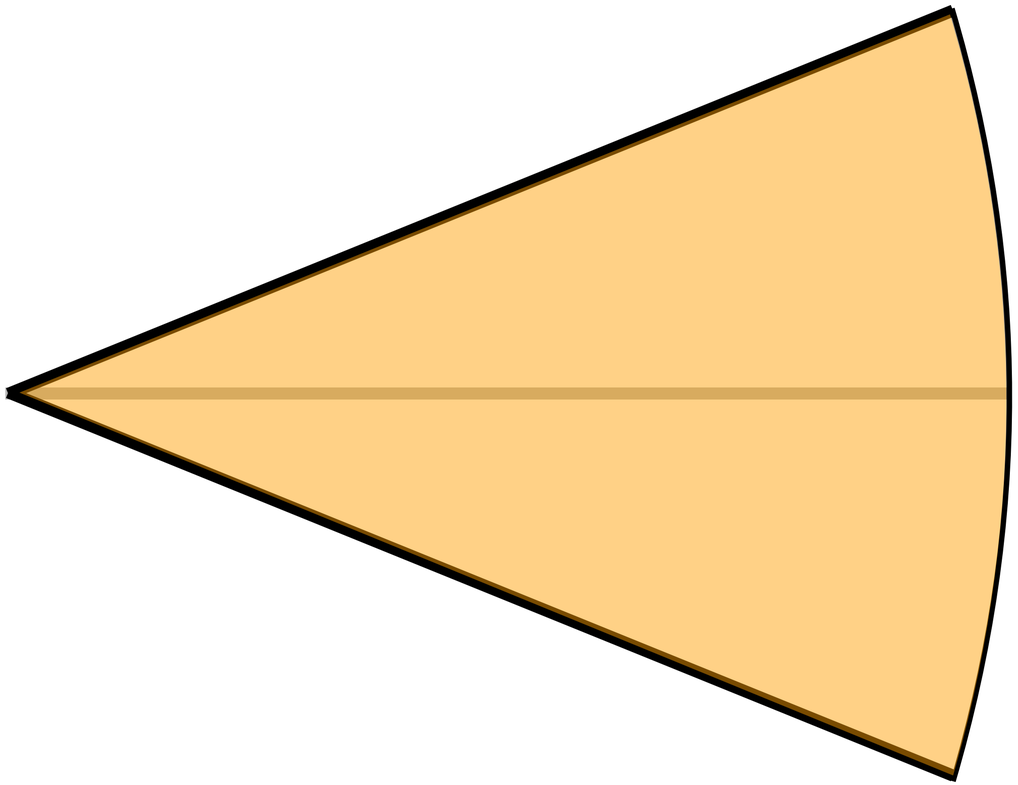
Velocity



+ Spread



- (2 * spread)



- (2 * spread) * rand

```
emitParticle() {
  // Define a maximum angle range in radians.
  let spread = Math.PI / 32;

  // Use an angle randomized over a spread so we have more of a "spray"
  var angle = this.velocity.rad() + spread - (Math.random() * spread * 2);

  // The magnitude of the emitter's velocity
  var magnitude = this.velocity.length();

  // New velocity based off of the calculated angle and magnitude
  var velocity = Vector.fromAngle(angle, magnitude);

  // return our new Particle
  return new Particle(this.position.clone(), velocity);
}
```

```
emitParticle() {
  // Define a maximum angle range in radians.
  let spread = Math.PI / 32;

  // Use an angle randomized over a spread so we have more of a "spray"
  var angle = this.velocity.rad() + spread - (Math.random() * spread * 2);

  // The magnitude of the emitter's velocity
  var magnitude = this.velocity.length();

  // New velocity based off of the calculated angle and magnitude
  var velocity = Vector.fromAngle(angle, magnitude);

  // return our new Particle
  return new Particle(this.position.clone(), velocity);
}
```

```
emitParticle() {
  // Define a maximum angle range in radians.
  let spread = Math.PI / 32;

  // Use an angle randomized over a spread so we have more of a "spray"
  var angle = this.velocity.rad() + spread - (Math.random() * spread * 2);

  // The magnitude of the emitter's velocity
  var magnitude = this.velocity.length();

  // New velocity based off of the calculated angle and magnitude
  var velocity = Vector.fromAngle(angle, magnitude);

  // return our new Particle
  return new Particle(this.position.clone(), velocity);
}
```

1. Application state modified and rendered independently.
2. Expensive mutations performed as little as possible.
3. Physics based animations.

We're getting there. You can do this now, sort of.

```
var Greets = React.createClass({

  getInitialState: function() {
    return { name: "Portland" };
  },

  render: function() {
    return <div>Hello {this.state.name}</div>;
  }

});
```

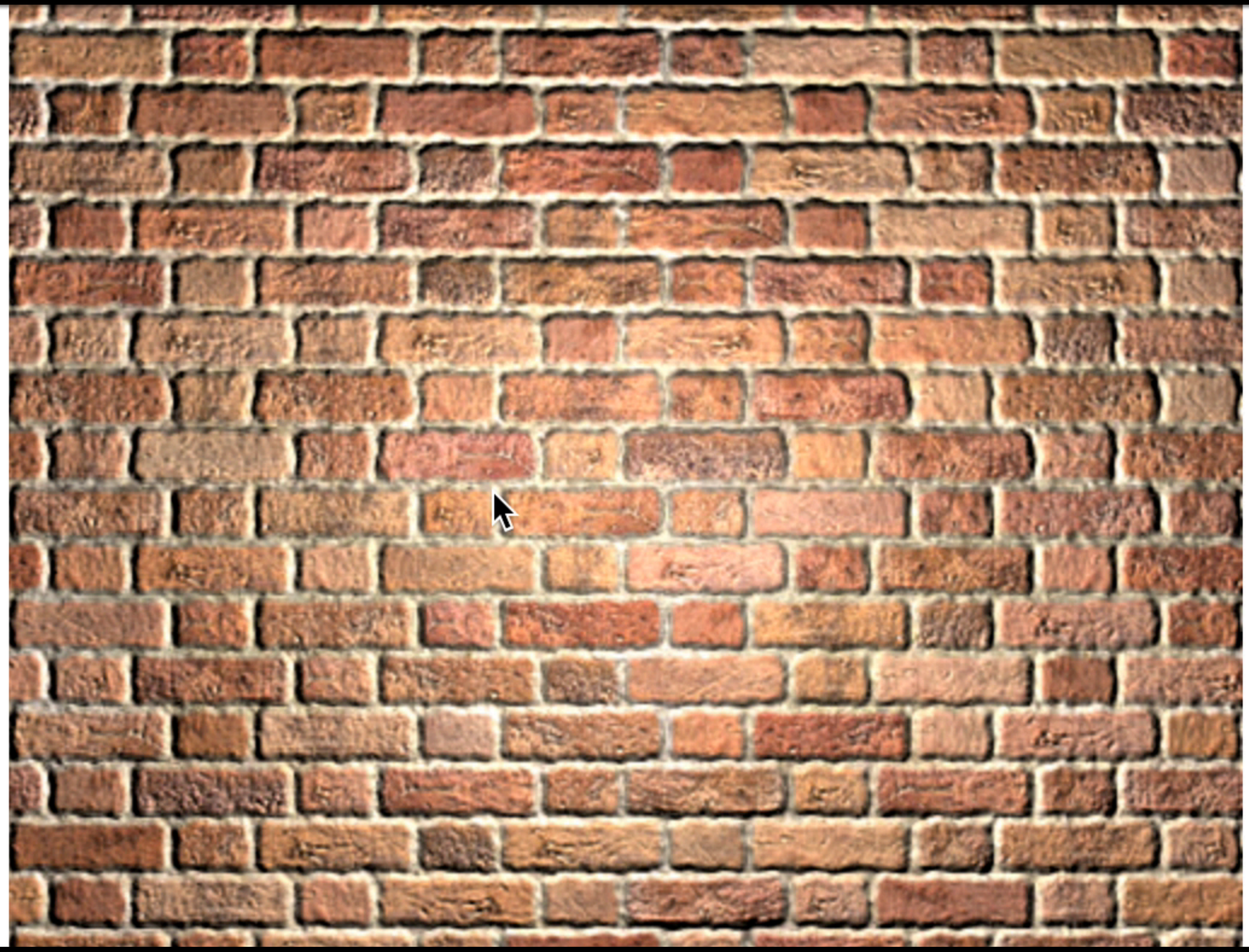
The virtual DOM is an abstraction that saves you the cost of expensive render-based mutation in a familiar package.

It's better, but a virtual DOM is not our application state.

Go, do neat things with canvas!

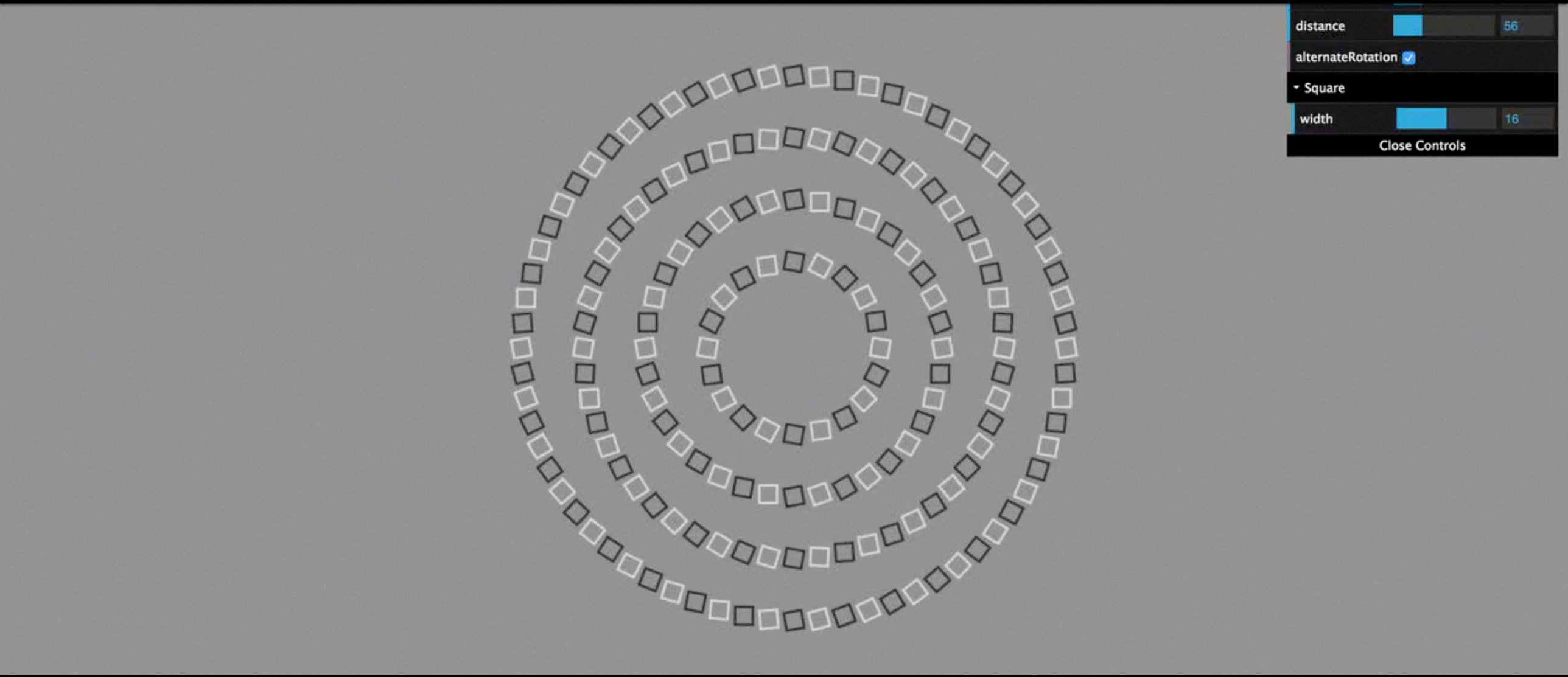
While you're exploring JavaScript, explore graphics & games.

Maybe you'll find a use for normal mapped backgrounds.



http://jsoverson.github.io/texture.js/demo/bricks.html

Or maybe you'll use it to understand the bounds of an optical illusion.



http://jsoverson.github.io/concentric-circle-illusion/

Or maybe you'll make some really awesome games.

https://ga.me/games/polycraft (turbulenz.com)

Or maybe you'll try…

phaser.io

Graphics for Web Devs

Jarrod Overson - Shape Security
@jsoverson