An intro to 2-D graphic through PIXI.js

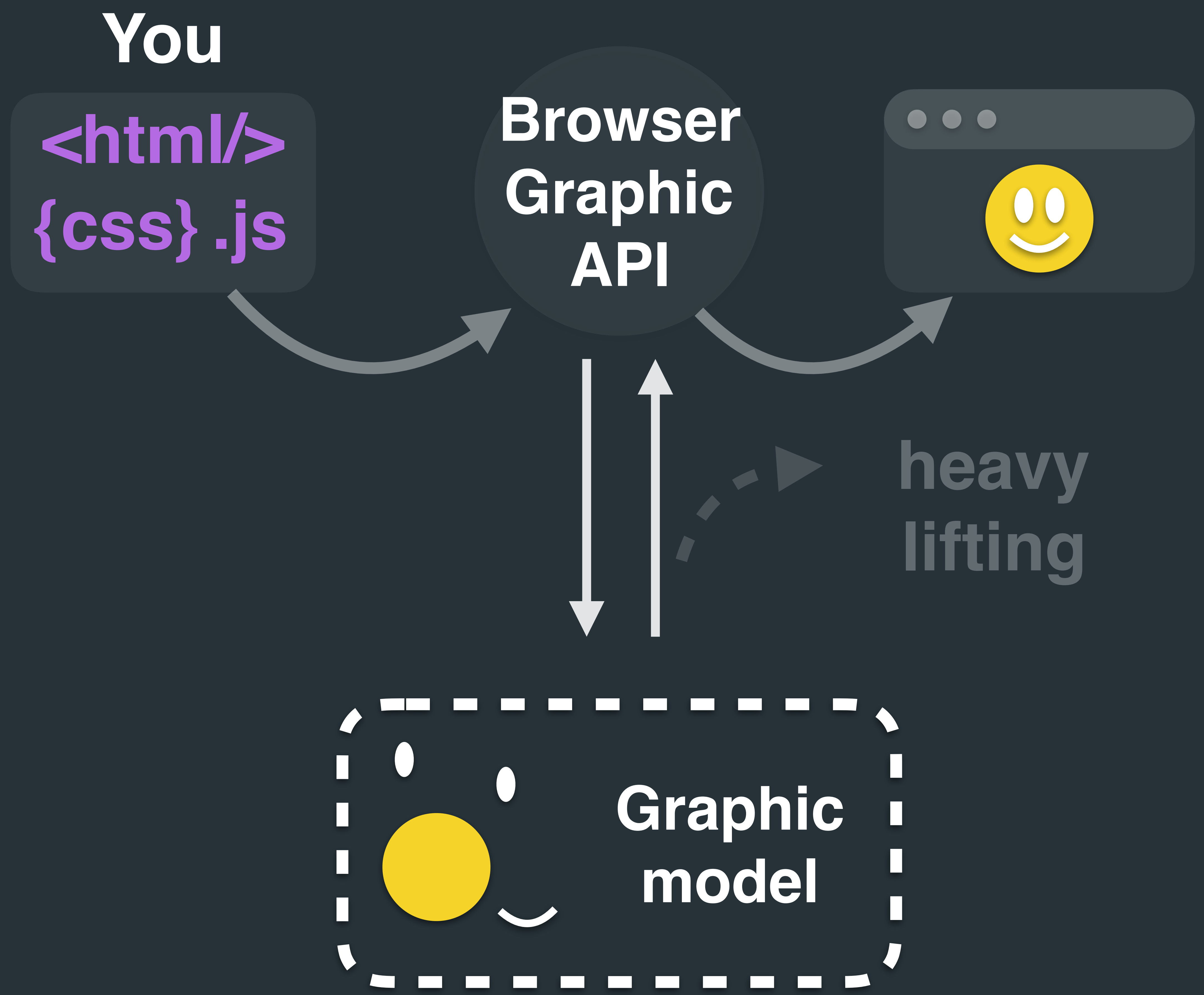
by Minh for FCC Toronto

How do you draw stuff onto the browser?

Two ways

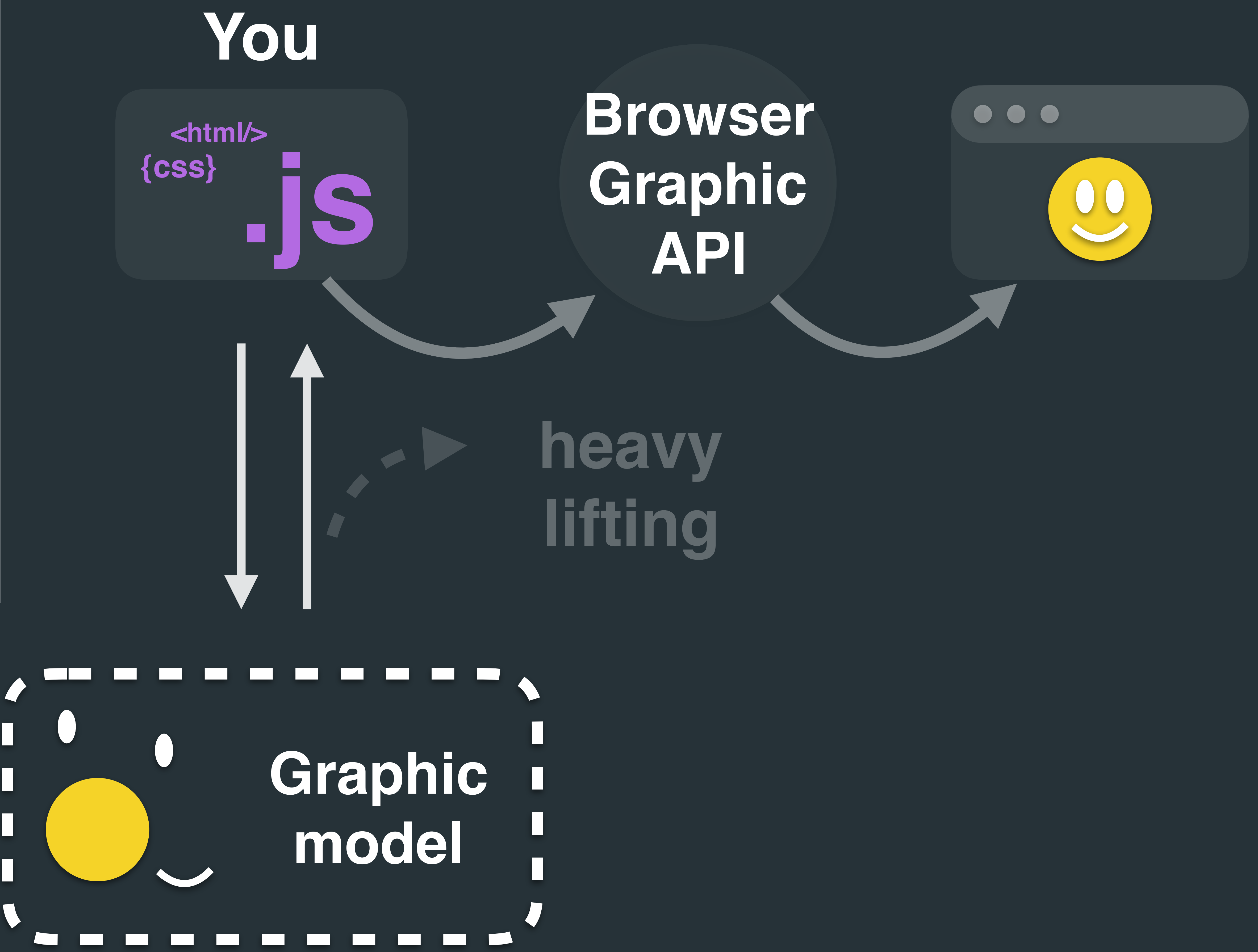
DOM

aka, “Retained Mode”



Canvas

aka, “Immediate Mode”



But Why????

Pros & Cons

DOM

Pros

No “redrawing logic”

CSS styling

Easier

Cons

Memory intensive

Less control over rendering

Best for complex layouts that don’t move much

Canvas

Pros

FAST, like… REALLY FAST!

Flexibility & Control

Cons

Complex

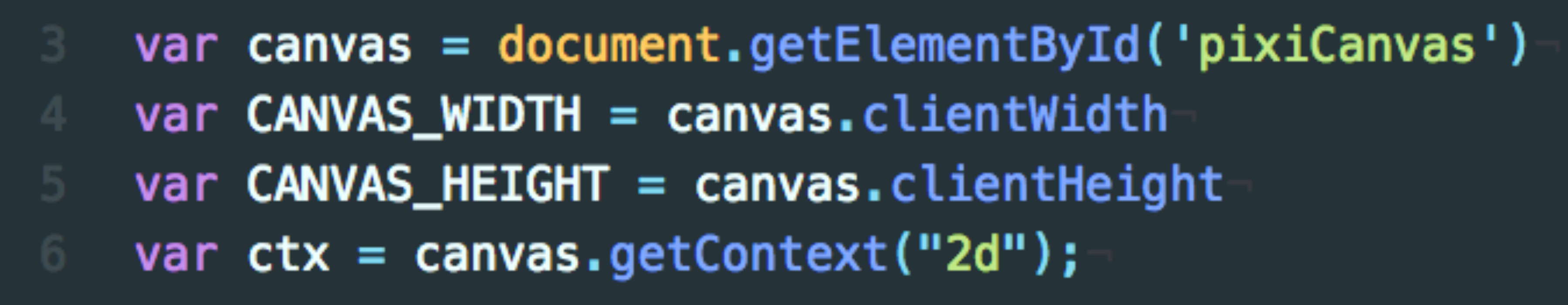
No Devtool support

Easy to shoot yourself in the foot

Best for complex graphics & interactions

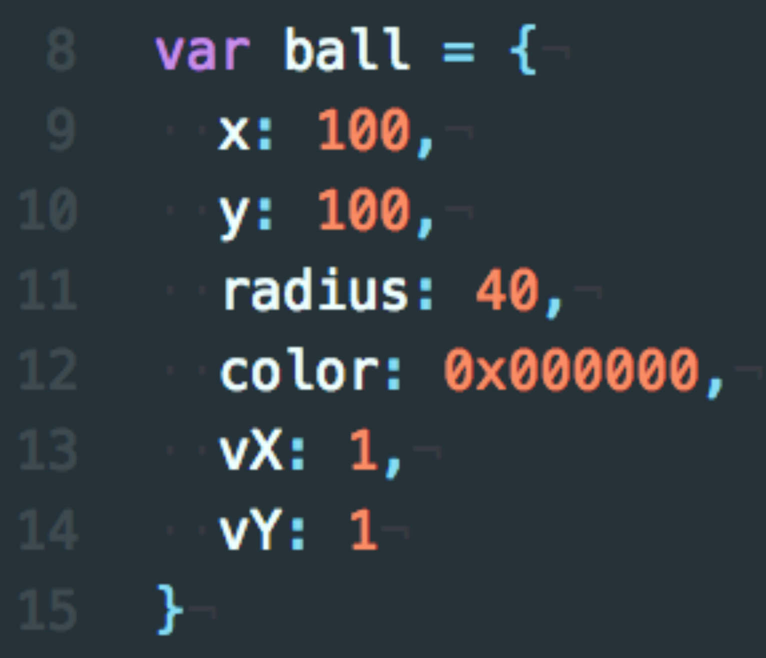
Primitive canvas rendering - bouncing ball

① Setup



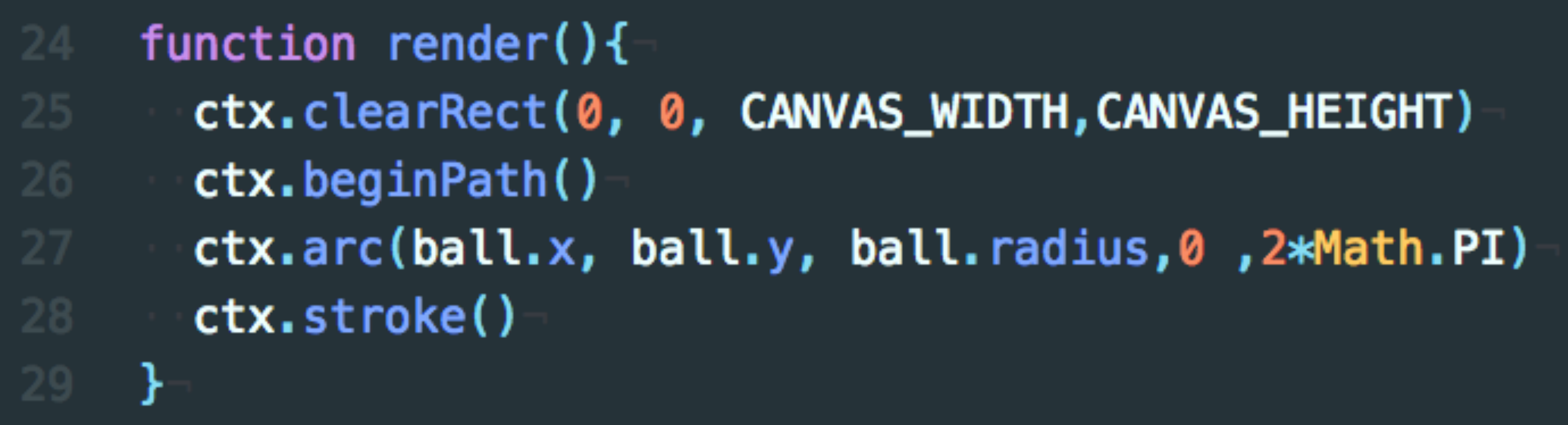
```
var canvas = document.getElementById('pixiCanvas')
var CANVAS_WIDTH = canvas.clientWidth
var CANVAS_HEIGHT = canvas.clientHeight
var ctx = canvas.getContext("2d");
```

② Logic state model



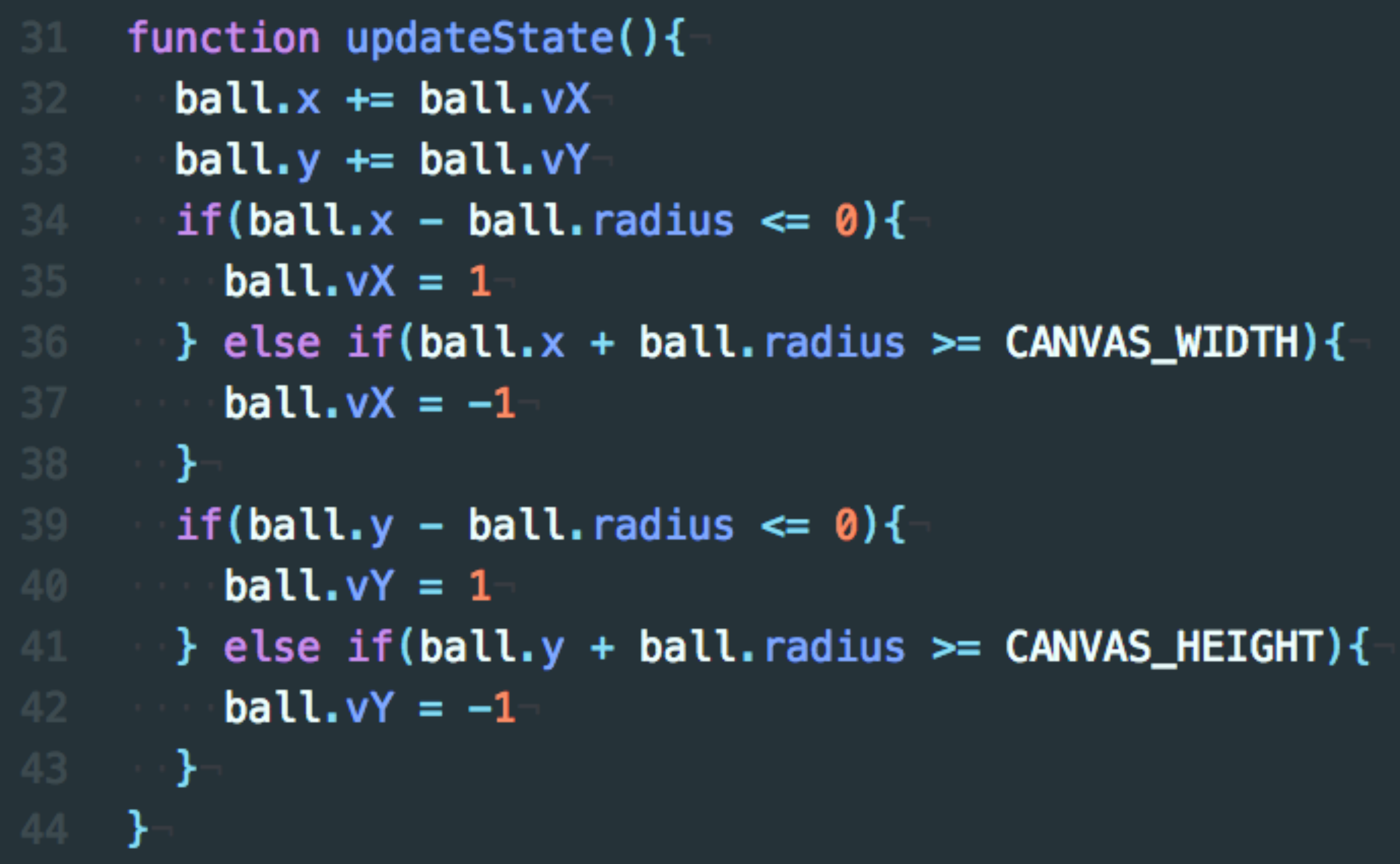
```
var ball = {
  x: 100,
  y: 100,
  radius: 40,
  color: 0x000000,
  vX: 1,
  vY: 1
}
```

③ Render logic



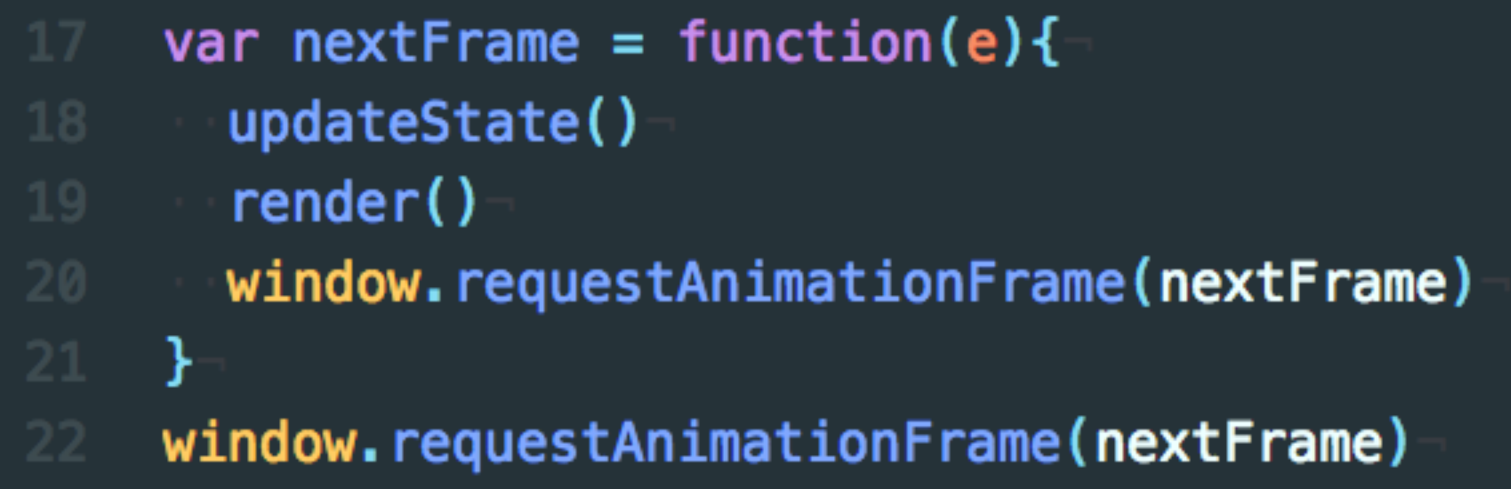
```
function render(){
  ctx.clearRect(0, 0, CANVAS_WIDTH,CANVAS_HEIGHT)
  ctx.beginPath()
  ctx.arc(ball.x, ball.y, ball.radius,0 ,2*Math.PI)
  ctx.stroke()
}
```

④ State update logic



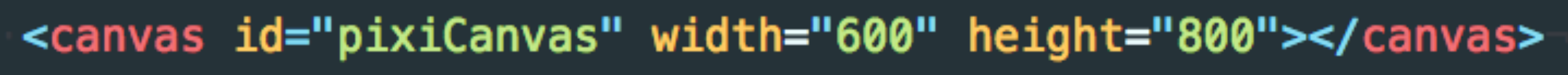
```
function updateState(){
  ball.x += ball.vX
  ball.y += ball.vY
  if(ball.x - ball.radius <= 0){
    ball.vX = 1
  } else if(ball.x + ball.radius >= CANVAS_WIDTH){
    ball.vX = -1
  }
  if(ball.y - ball.radius <= 0){
    ball.vY = 1
  } else if(ball.y + ball.radius >= CANVAS_HEIGHT){
    ball.vY = -1
  }
}
```

⑤ Render loop



```
var nextFrame = function(e){
  updateState()
  render()
  window.requestAnimationFrame(nextFrame)
}
window.requestAnimationFrame(nextFrame)
```

… and the lonely html…



```
<canvas id="pixiCanvas" width="600" height="800"></canvas>
```

The challenge:

1

No coupling of display element with state and functionalities

- well… there’s no such thing as “display elements” to begin with -

2

Hard to handle user interactions

- how can I tell which ball was clicked on? -

3

No display hierarchy

- what if I want a smaller ball inside the big one? -

Enter PixiJS v4

What is it?

A “rendering engine*” so you don’t have to invent your own

Why it’s useful?

It introduces display elements into canvas, aka “Sprites**”

Why it’s popular?

- Fast, REALLY FAST!
- Handles both canvas and WebGL*** rendering from a common API
- Relatively general purpose

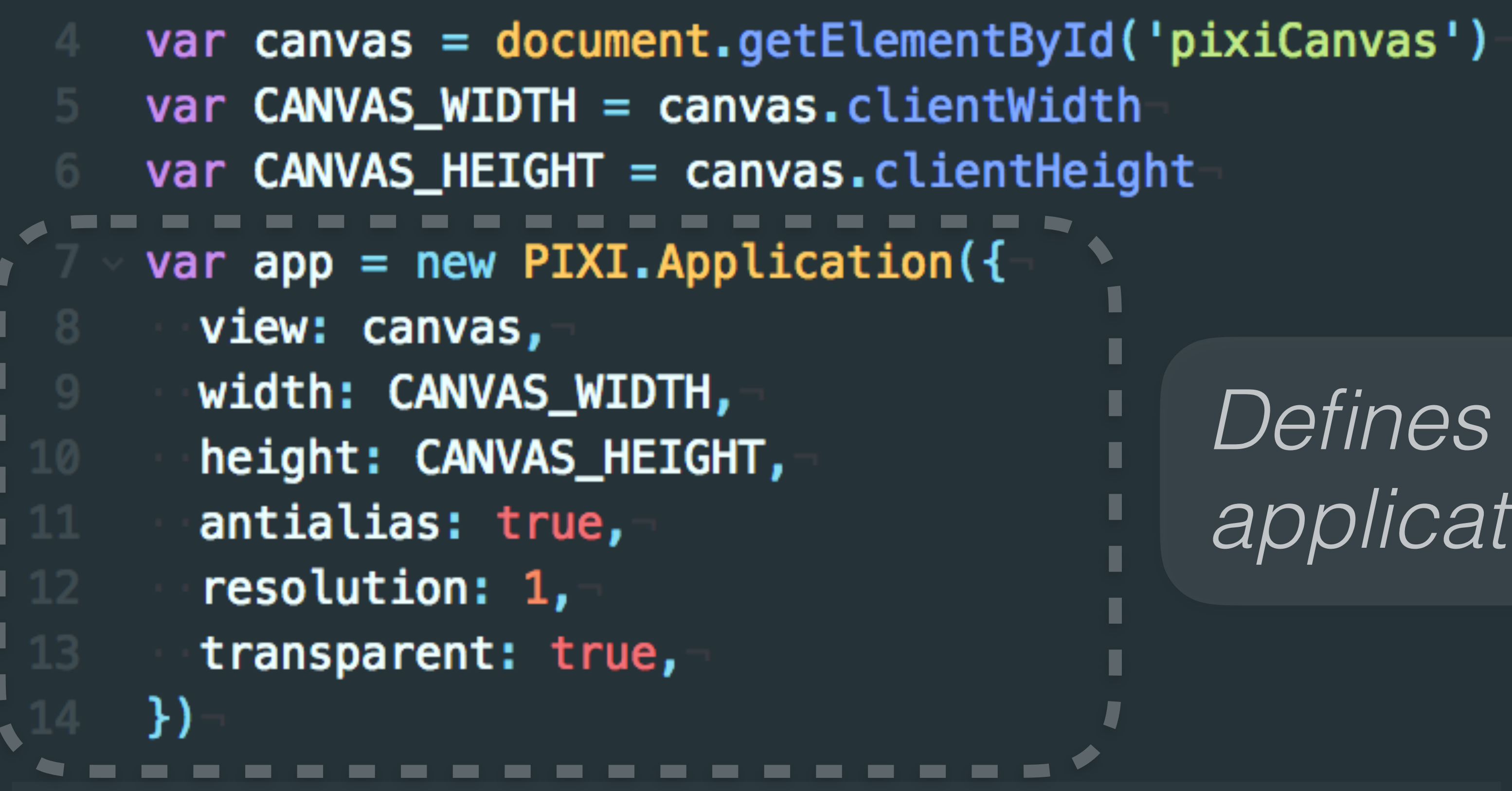
* **rendering engine:** a software that handles putting stuff on the screen

** **not the drink**

*** **WebGL:** a 3D graphic engine for the web that also renders to a <canvas/> tag

PixiJS v4 - bouncing ball

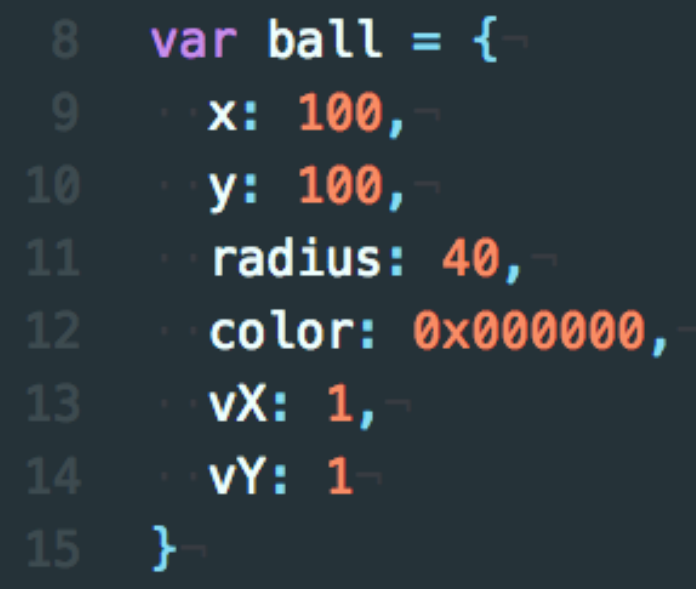
1 Setup



```
var canvas = document.getElementById('pixiCanvas')
var CANVAS_WIDTH = canvas.clientWidth
var CANVAS_HEIGHT = canvas.clientHeight
var app = new PIXI.Application({
  view: canvas,
  width: CANVAS_WIDTH,
  height: CANVAS_HEIGHT,
  antialias: true,
  resolution: 1,
  transparent: true,
})
```

Defines the application

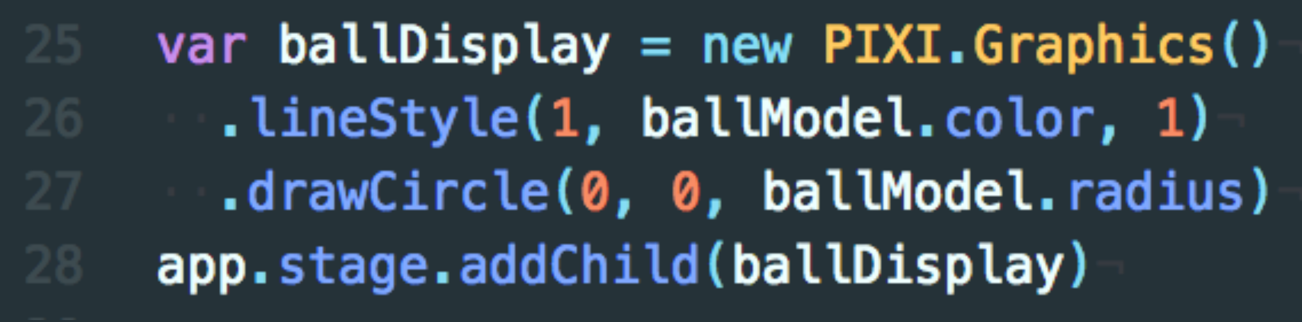
2 Logic model



```
var ball = {
  x: 100,
  y: 100,
  radius: 40,
  color: 0x000000,
  vX: 1,
  vY: 1
}
```

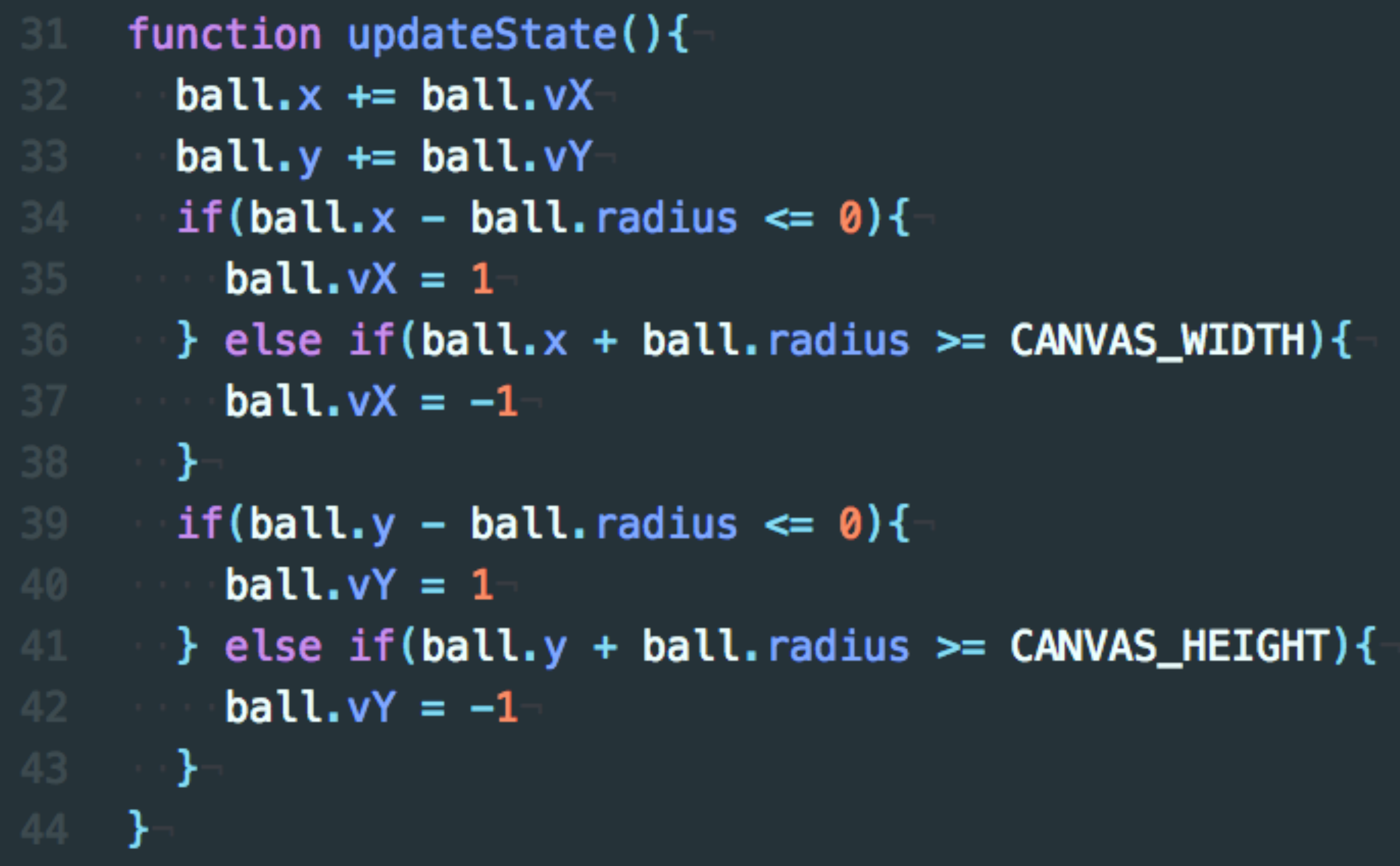
3 Display Model

Object representation of display state



```
var ballDisplay = new PIXI.Graphics()
  .lineStyle(1, ballModel.color, 1)
  .drawCircle(0, 0, ballModel.radius)
app.stage.addChild(ballDisplay)
```

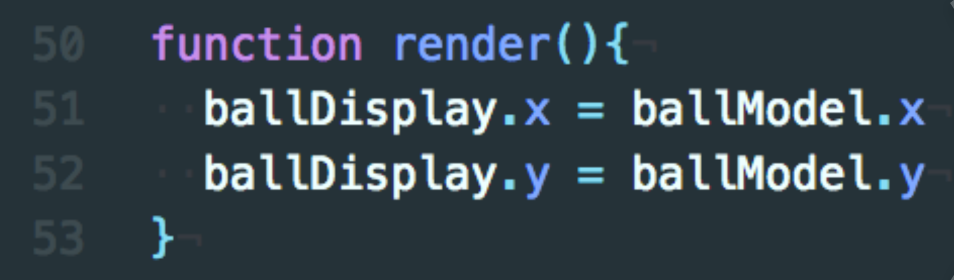
4 State update logic



```
function updateState(){
  ball.x += ball.vX
  ball.y += ball.vY
  if(ball.x - ball.radius <= 0){
    ball.vX = 1
  } else if(ball.x + ball.radius >= CANVAS_WIDTH){
    ball.vX = -1
  }
  if(ball.y - ball.radius <= 0){
    ball.vY = 1
  } else if(ball.y + ball.radius >= CANVAS_HEIGHT){
    ball.vY = -1
  }
}
```

3 Render Logic

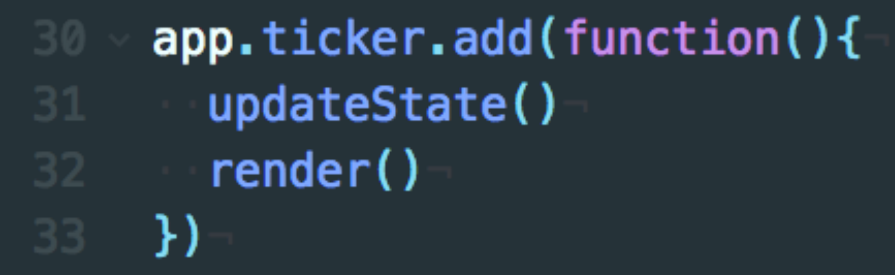
Rendering maps logic model to display model



```
function render(){
  ballDisplay.x = ballModel.x
  ballDisplay.y = ballModel.y
}
```

5 Render loop

Built-in render loop



```
app.ticker.add(function(){
  updateState()
  render()
})
```

Common Components of PixiJS v4

EventEmitter

Anything that emit events, like mouse events. API provides methods like on, once, etc…

PIXI.DisplayObject

The most raw type of objects that can be put on the screen.
API provides properties like: x, y, alpha, visible,… and methods like getBounds

PIXI.Container (the PIXI version of “DOM node”. Every app has at least on instance of this: Pixi.Application.stage, like <body>)

Extension of DisplayObject that supports nesting, so other DisplayObjects can be added to it.
API provides properties like: width, height, children,… and methods like addChild, removeChild,…

PIXI.Graphics

Extension of Container that provides ways to logically draw custom graphics
API provides properties like: lineColor, tint, blendMode,… and methods like drawRect, moveTo,…

PIXI.Sprite

The base for all “textured” objects, to load up images, tiling, etc
Provides the API with methods like: from, fromImage

PIXI.Text

…

Fun fact

What on earth is a “Sprite”?

sprite [noun] /sprīt/

1. an elf or fairy.
   synonyms: fairy, elf, pixie, imp, brownie, puck, peri, leprechaun;
2. a computer graphic that may be moved on-screen and otherwise manipulated as a single entity.

- Google, 2017 -

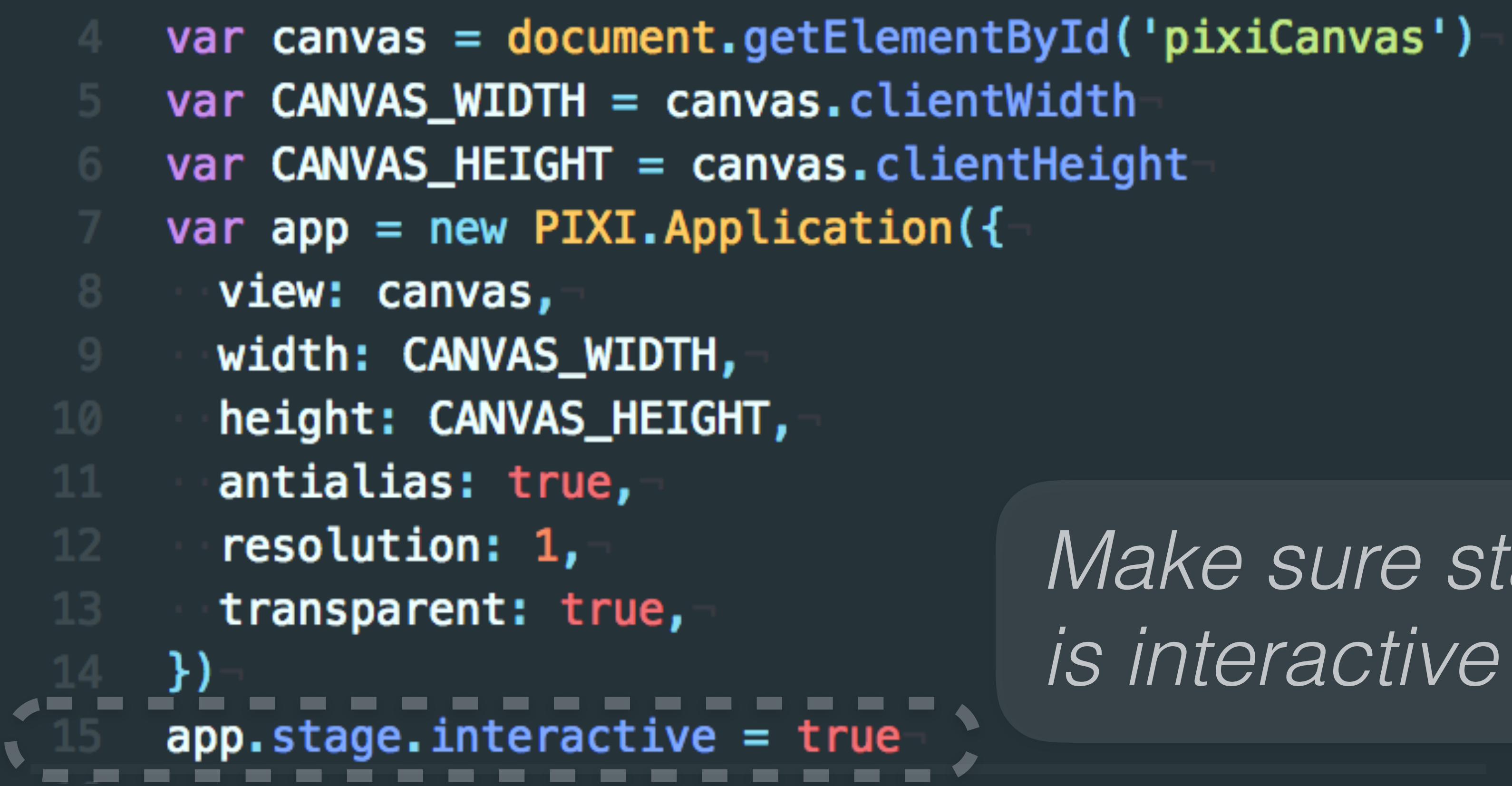
> “ *The term was derived from the fact that sprites, rather than being part of the bitmap data in the framebuffer, instead "floated" around on top without affecting the data in the framebuffer below, much like a ghost or "sprite".* ”

- Wikipedia, 2017 -

What if I told you a div was a Sprite,
and you've been manipulating fairies all this time?

PixiJS v4 - interactive example

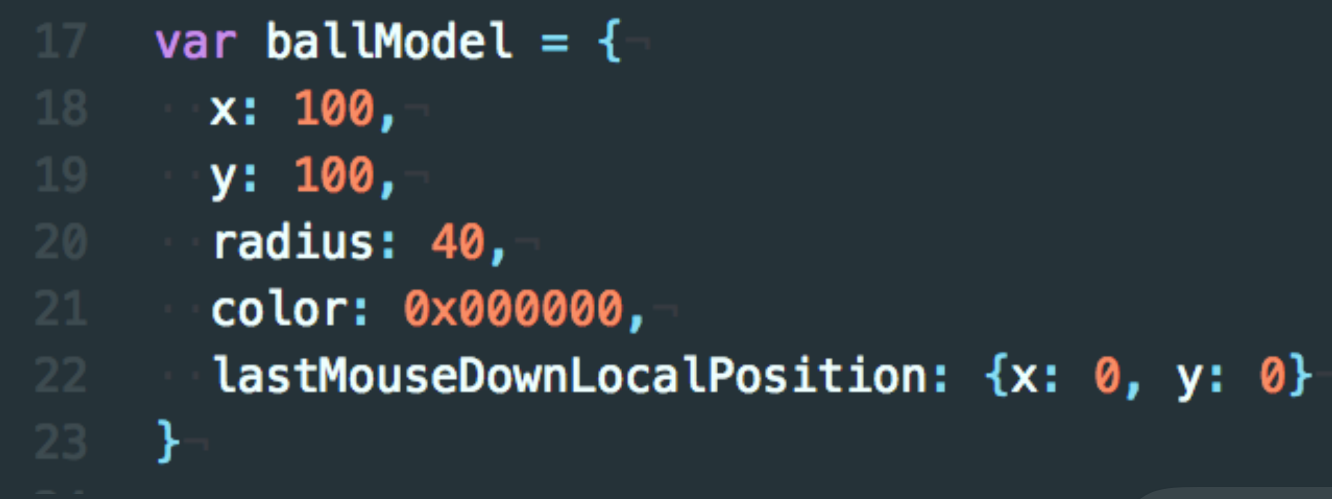
① Setup



```
var canvas = document.getElementById('pixiCanvas')
var CANVAS_WIDTH = canvas.clientWidth
var CANVAS_HEIGHT = canvas.clientHeight
var app = new PIXI.Application({
  view: canvas,
  width: CANVAS_WIDTH,
  height: CANVAS_HEIGHT,
  antialias: true,
  resolution: 1,
  transparent: true,
})
app.stage.interactive = true
```

Make sure stage is interactive

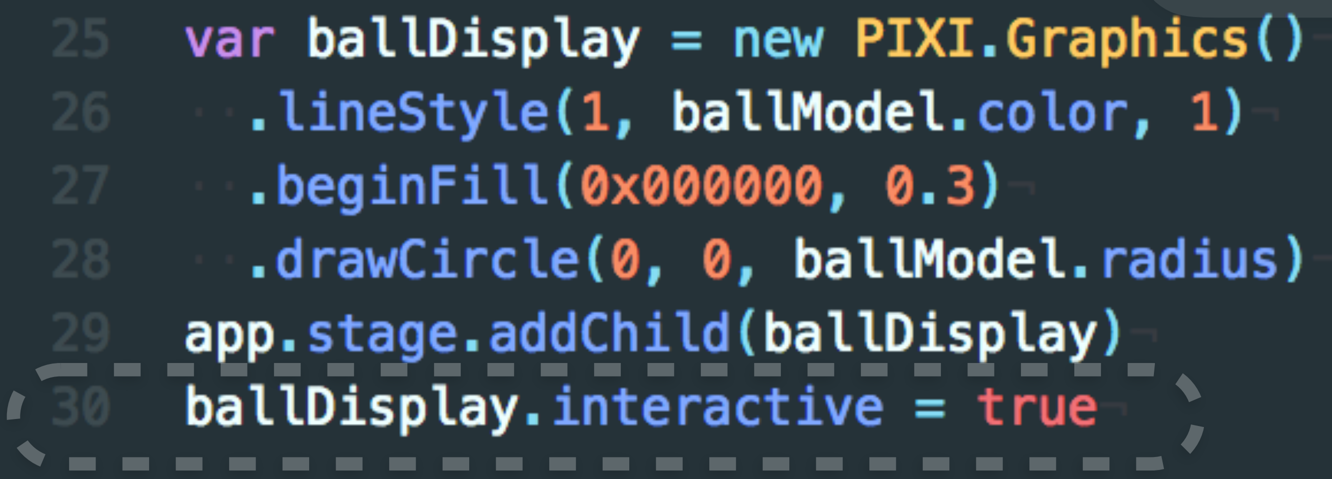
② Logic model



```
var ballModel = {
  x: 100,
  y: 100,
  radius: 40,
  color: 0x000000,
  lastMouseDownLocalPosition: {x: 0, y: 0}
}
```

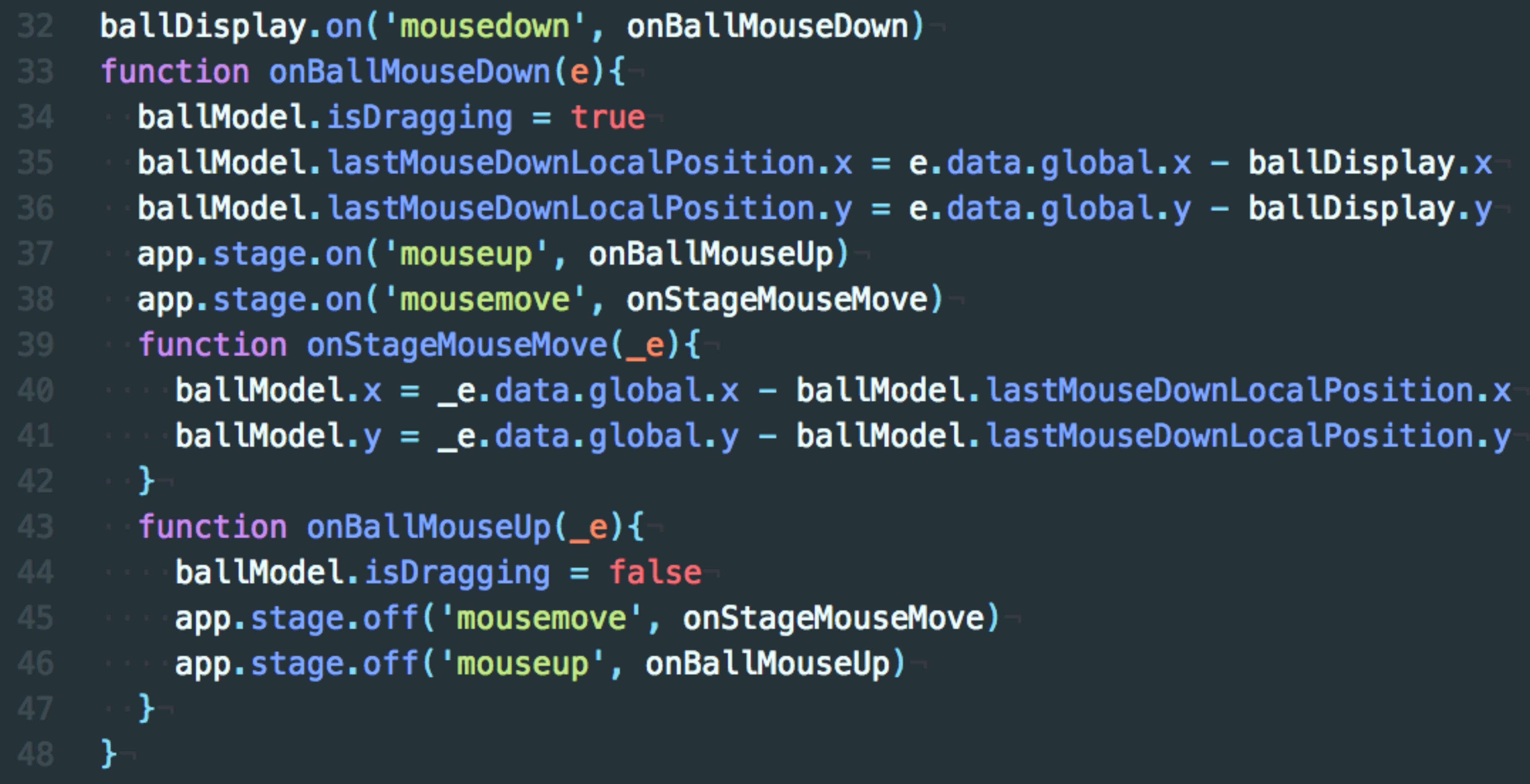
③ Display Model

Make sure display object is interactive



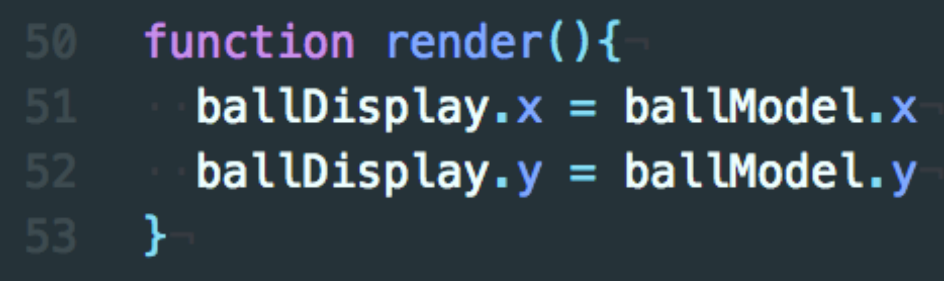
```
var ballDisplay = new PIXI.Graphics()
  .lineStyle(1, ballModel.color, 1)
  .beginFill(0x000000, 0.3)
  .drawCircle(0, 0, ballModel.radius)
app.stage.addChild(ballDisplay)
ballDisplay.interactive = true
```

④ Event listeners to update logic state



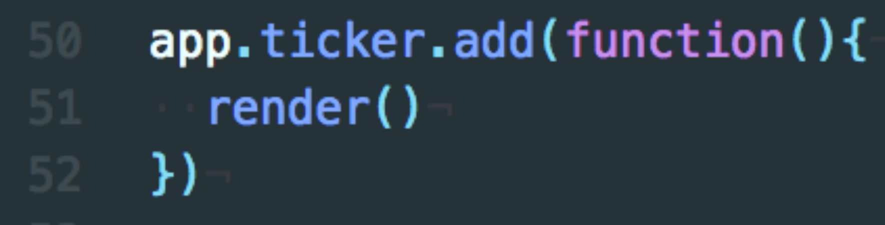
```
ballDisplay.on('mousedown', onBallMouseDown)
function onBallMouseDown(e){
  ballModel.isDragging = true
  ballModel.lastMouseDownLocalPosition.x = e.data.global.x - ballDisplay.x
  ballModel.lastMouseDownLocalPosition.y = e.data.global.y - ballDisplay.y
  app.stage.on('mouseup', onBallMouseUp)
  app.stage.on('mousemove', onStageMouseMove)
  function onStageMouseMove(_e){
    ballModel.x = _e.data.global.x - ballModel.lastMouseDownLocalPosition.x
    ballModel.y = _e.data.global.y - ballModel.lastMouseDownLocalPosition.y
  }
  function onBallMouseUp(_e){
    ballModel.isDragging = false
    app.stage.off('mousemove', onStageMouseMove)
    app.stage.off('mouseup', onBallMouseUp)
  }
}
```

③ Render Logic



```
function render(){
  ballDisplay.x = ballModel.x
  ballDisplay.y = ballModel.y
}
```

⑤ Render loop



```
app.ticker.add(function(){
  render()
})
```

When to consider <canvas> and PixiJS$_{v4}$?

My rules of thumb:

1. Physics simulations
2. Proximity / collision / overlap detection
3. Complex drag and drop interactions
4. Fluid and constantly chaninging UI
5. Complex logical drawings
6. Any other time where easy access to display state is important

Is PixiJS v4 for you?

Reasons to learn

- You want to do web game development
- You want to build complex interactive UI
- Your project requires heavy custom graphics
- “Doing the heavy lifting” makes you think of architecture A LOT

Reasons to skip

- Most web projects don’t need this level of control
- A relatively specialized skill set
- Generally much slower development speed (subjective)
- “Doing the heavy lifting” makes you think of architecture A LOT

PixiJS v4 - Hackathon!

demo: https://codepen.io/hlminh2000/full/veZQbN/

Requirements:

1. Ball must bounce on the ground and ceiling
2. Ball must be draggable
3. Ball must maintain “momentum” when let go
4. Ball must bounce on the walls