https://github.com/
kittykatattack/dwc2014

Rex van der Spuy

- Foundation Game Design with Flash
- Advanced Game Design with Flash
- Foundation game Design with AS3.0
- Foundation Game Design with HTML5 and JavaScript
- Advanced Game Design with HTML5 and JavaScript
- The Advanced HTML5 Game Developers Toybox (2015)

How do you make a video game?

What you'll learn

- Technologies you can use
- Low-level code for canvas-based games
- High-level code for making games quickly
- Playtime

CHOOSE YOUR OWN ADVENTURE ® · 3

YOU'RE THE STAR OF THE STORY!
CHOOSE FROM 40 POSSIBLE ENDINGS.

BY BALLOON TO THE SAHARA

BY D. TERMAN

ILLUSTRATED BY PAUL GRANGER

27189-9 • $1.95 • A BANTAM BOOK

Most routes lead to a certain and terrifying death.

Which things?

Unity
iOS (SpriteKit/SceneKit)
Unreal engine
Monogame (XNA)
Flash (OpenFL/Haxe)
HTML5/JavaScript

HTML5?

Free

Everywhere

Easy

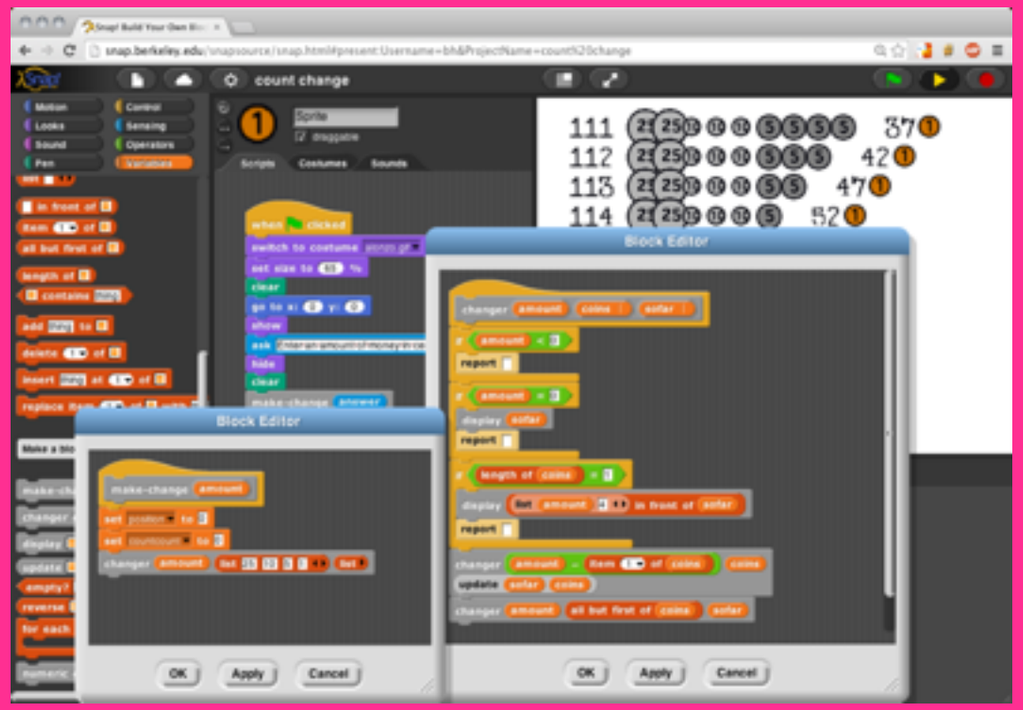
Fun

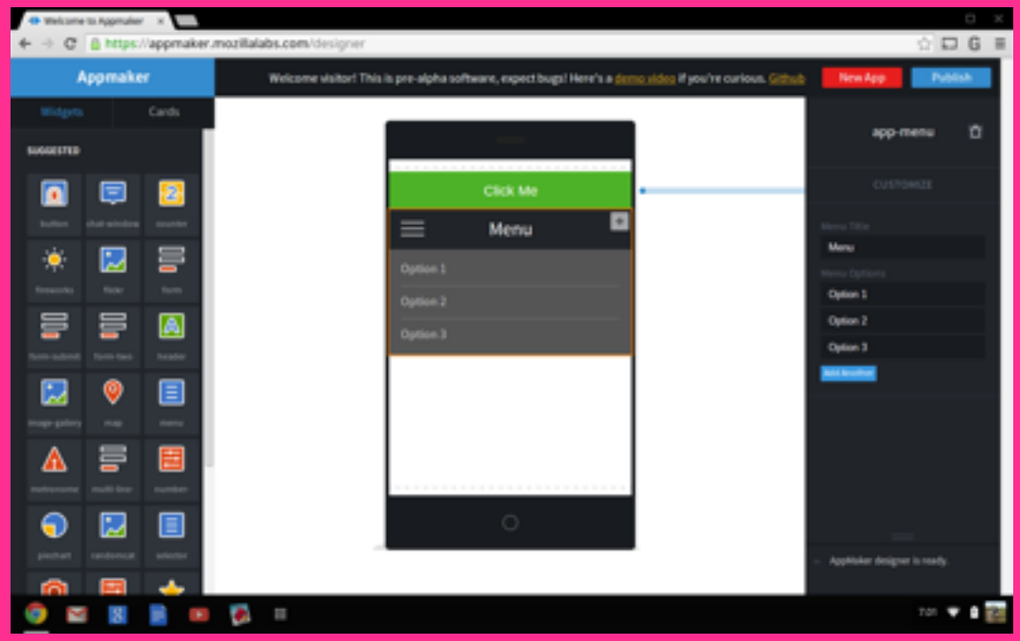
Fast (enough)

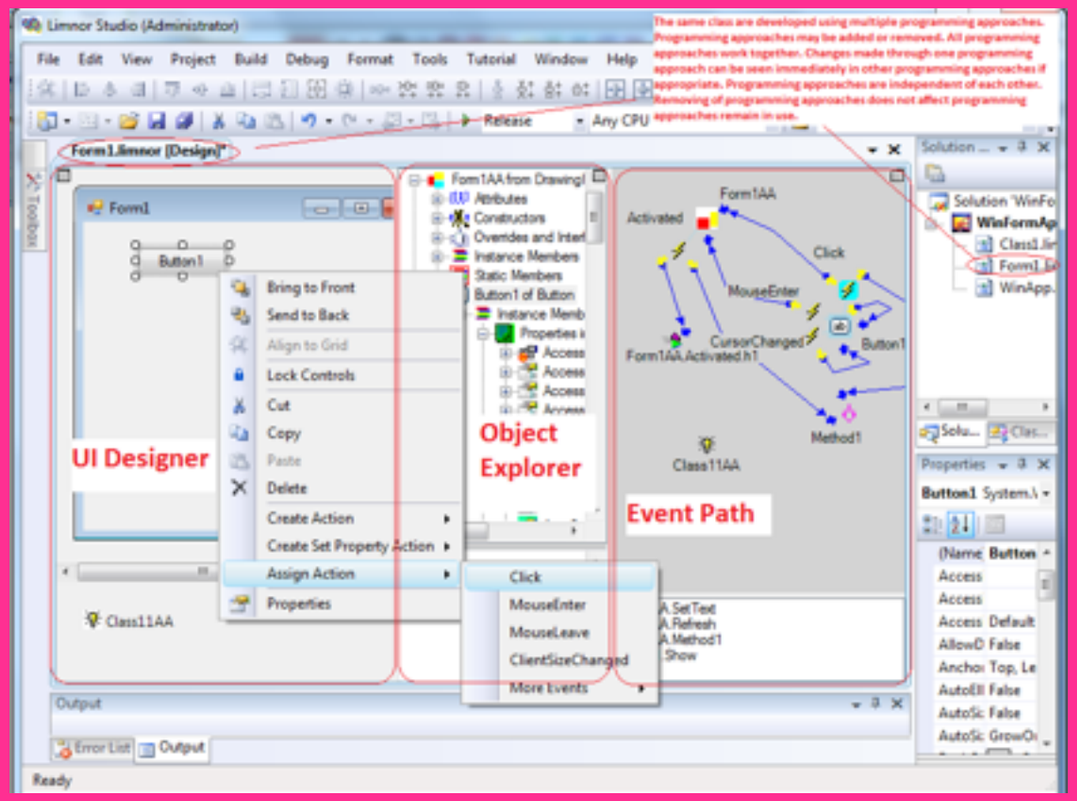
It’s 2014.
HTML5 is not a development platform.
It’s a *compile-to* platform.

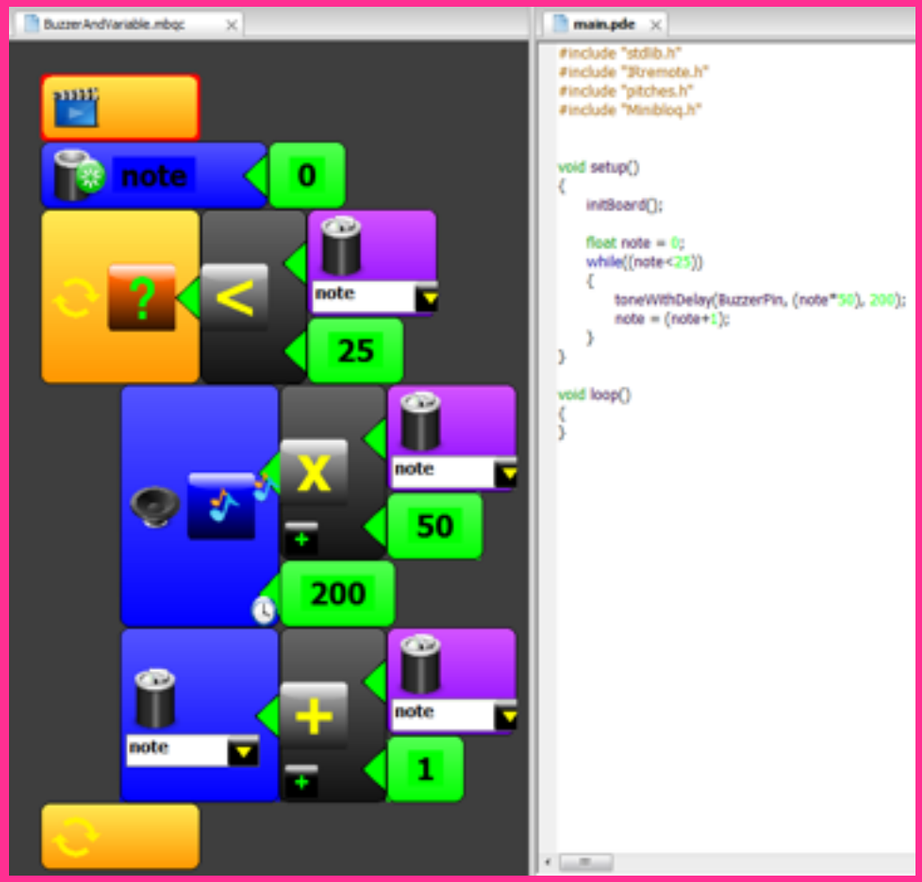
Which things?

Unity
Monogame (XNA)
Flash (OpenFL/Haxe)
Live Code
Play Canvas
Goo Engine
Raw HTML5/JavaScript









http://blog.interfacevision.com/design/design-visual-progarmming-languages-snapshots/

“Programming is about typing”

“Learn ideas, not platforms”

Raw HTML5/JS

Accessible
Flexible
Easy
Infinitely configurable
Reassuringly low-level
Plugin anything
Non-proprietary
Typing!
Platform and paradigm agnostic

What HTML5 stuff?

HTML?
CSS?
WebGL?
Canvas?
JavaScript?
SVG?

Canvas +

WebGL +

JavaScript +

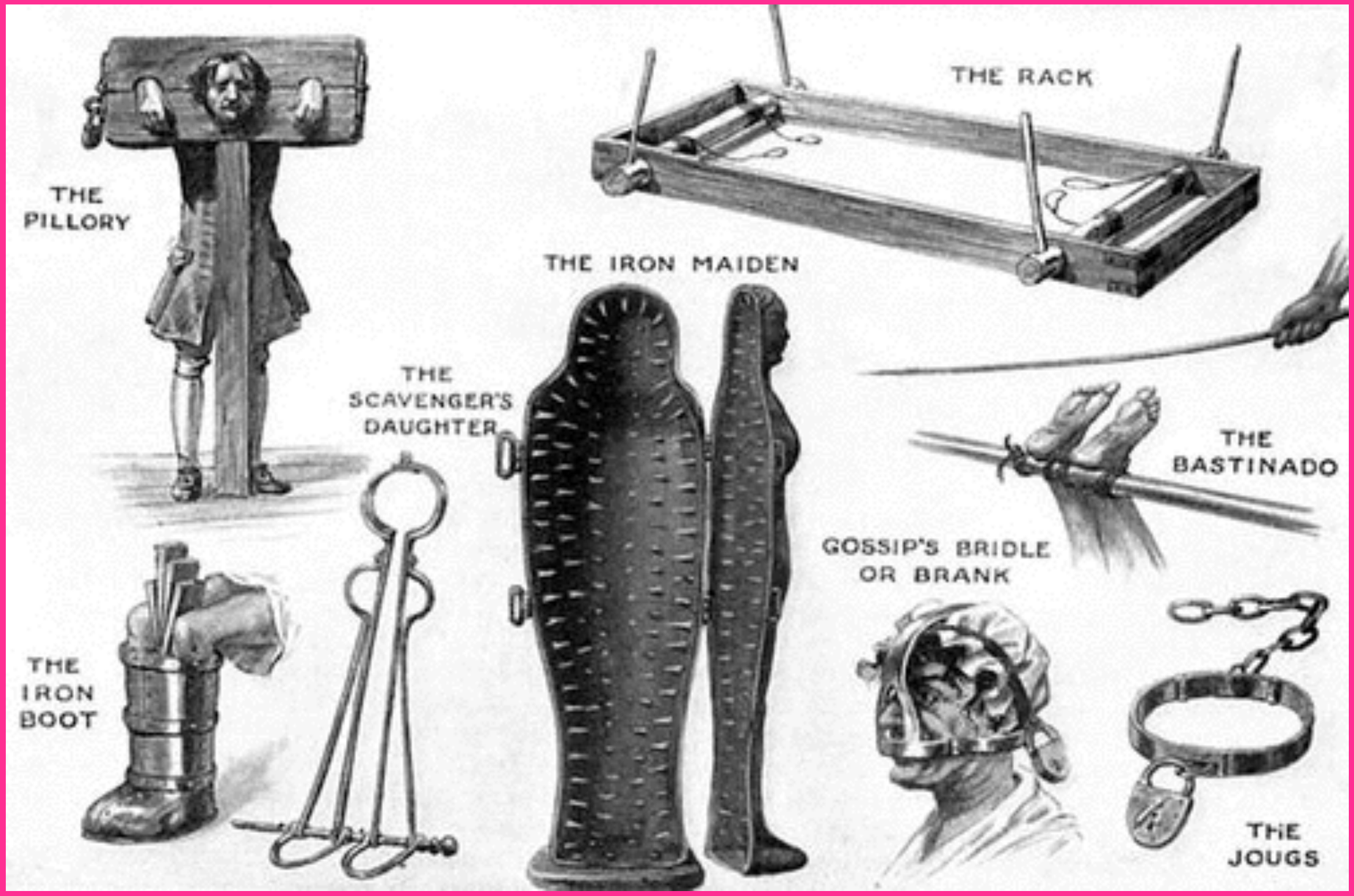
= Yay!

HTML +

CSS +

SVG +

= ???


THE PILLORY
THE RACK
THE IRON MAIDEN
THE SCAVENGER'S DAUGHTER
THE BASTINADO
GOSSIP'S BRIDLE OR BRANK
THE IRON BOOT
THE JOUGS

Game engines?

Phaser, Game Closure, Goo, Impact, Quintus, Enchant, Panda, Grapefruit, MelonJS, Turbulenz, Play Canvas, Game Maker, Construct2, Spell, CreateJS, Kiwi Cocos2D-X, Crafty, LimeJS, FrozenJS…

* forever

Choice Paralysis

Which way?

- **The low road:**
  Code everything from scratch

- **The middle road:**
  Code most things from scratch

- **The high road:**
  Use a game engine

The middle road

Write most of your code from scratch. Drop in a third-party library for any highly specialized code when it stops becoming fun, educational or productive.

This is the best long-term strategy for developing a deep understanding of the art of game design.

Now we can start making things!

But what?

1. Make a drawing surface.
2. Draw lines, shapes and images.
3. Turn them into reusable components called **Sprites**.
4. Make the sprites interactive.

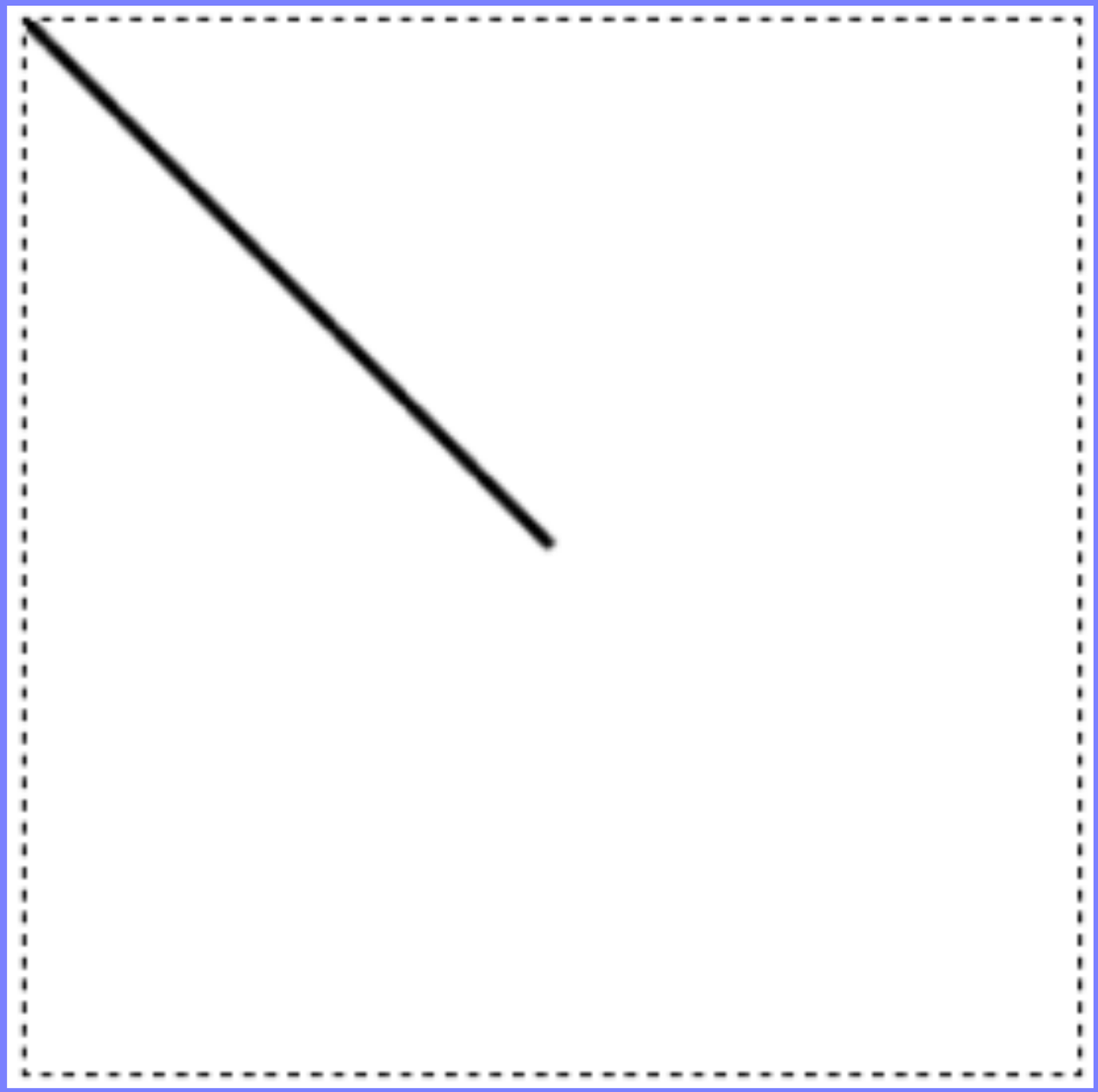
Make a canvas

```
var canvas = document.createElement("canvas");
canvas.width = 512;
canvas.height = 512;
canvas.style.border = "1px dashed black";
document.body.appendChild(canvas);
var ctx = canvas.getContext("2d");
```

```
<canvas id=“canvas" width=“256” height=“256”></canvas>
```

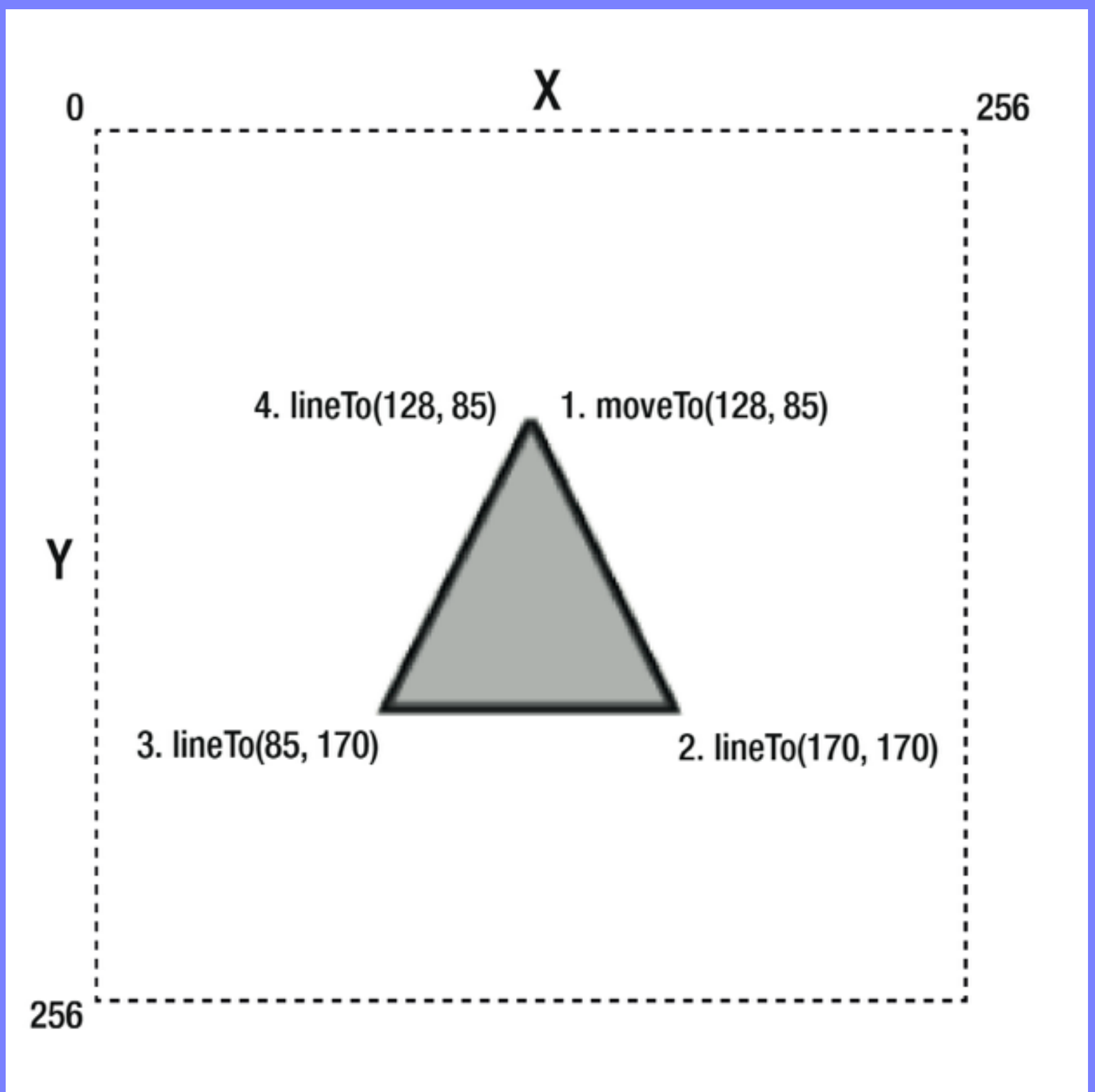
Draw a line

```
//1. Set the line style options
ctx.strokeStyle = "black";
ctx.lineWidth = 3;

//2. Draw the line
ctx.beginPath();
ctx.moveTo(0, 0);
ctx.lineTo(128, 128);
ctx.closePath();
ctx.stroke();
```



Connect lines

```
//Set the line and fill style options
ctx.strokeStyle = "black";
ctx.lineWidth = 3;
ctx.fillStyle = "rgba(128, 128, 128, 0.5)";

//Connect lines together to form a
//triangle in the center of the canvas
ctx.beginPath();
ctx.moveTo(128, 85);
ctx.lineTo(170, 170);
ctx.lineTo(85, 170);
ctx.lineTo(128, 85);
ctx.closePath();
ctx.stroke();
ctx.fill();
```


0
X
256
Y
4. lineTo(128, 85)
1. moveTo(128, 85)
3. lineTo(85, 170)
2. lineTo(170, 170)
256

Shapes

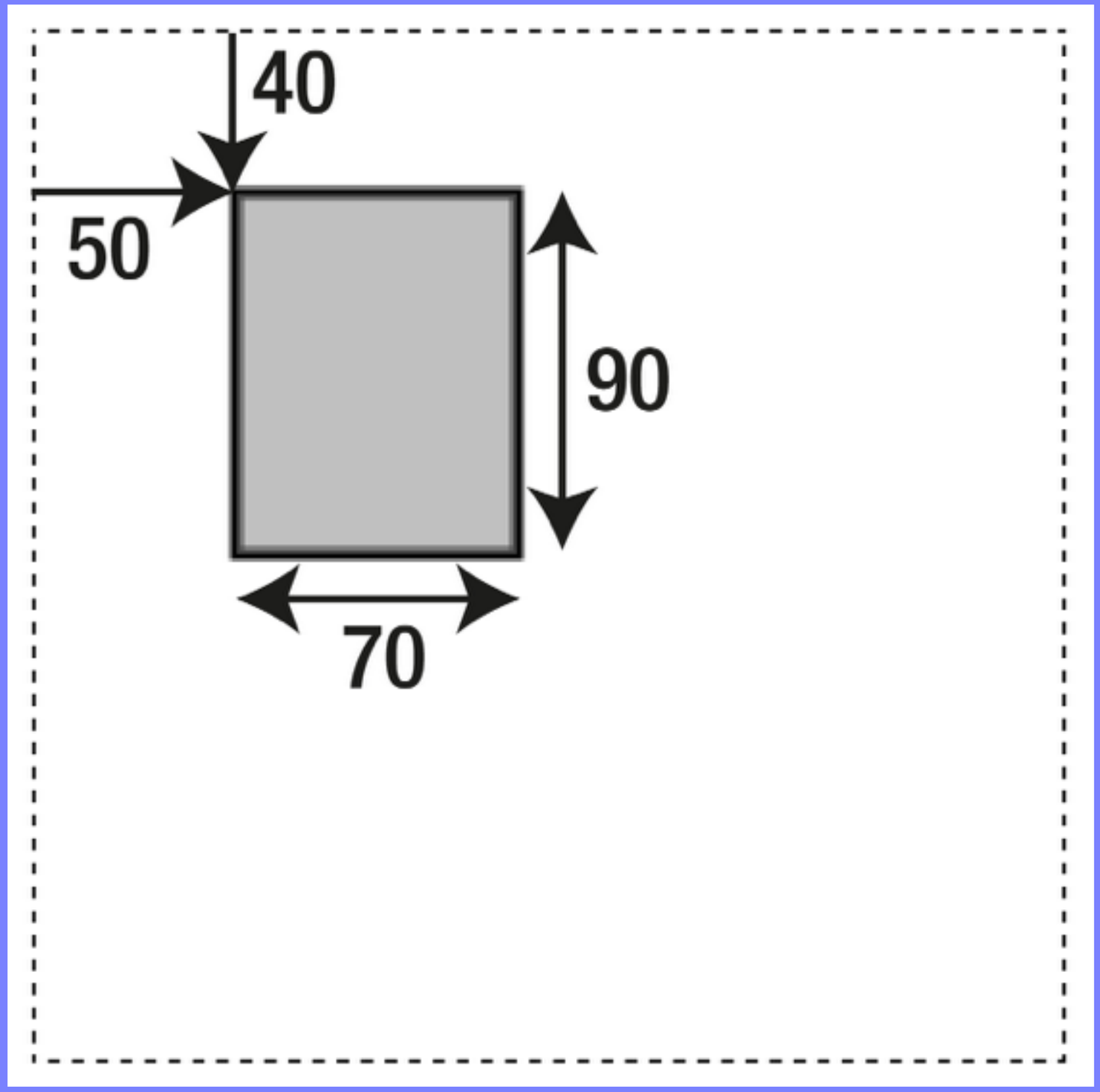
```
//1. Define the shape

var triangle = [
  [128, 85],
  [170, 170],
  [85, 170]
];
```

```
//3. Draw the shape

ctx.beginPath();
drawPath(triangle);
ctx.stroke();
ctx.closePath();
ctx.fill();
```

```
//2. Plot the lines

function drawPath(shape) {
  //Start drawing from the last point
  var lastPoint = shape.length - 1;
  ctx.moveTo(
    shape[lastPoint][0],
    shape[lastPoint][1]
  );
  //Use a loop to plot each point
  shape.forEach(function(point) {
    ctx.lineTo(point[0], point[1]);
  });
}
```

Rectangles

```
//Set the line and fill style options
ctx.strokeStyle = "black";
ctx.lineWidth = 3;
ctx.fillStyle = "rgba(128, 128, 128, 0.5)";

//Draw the rectangle
ctx.beginPath();
ctx.rect(50, 40, 70, 90);
ctx.stroke();
ctx.closePath();
ctx.fill();
```

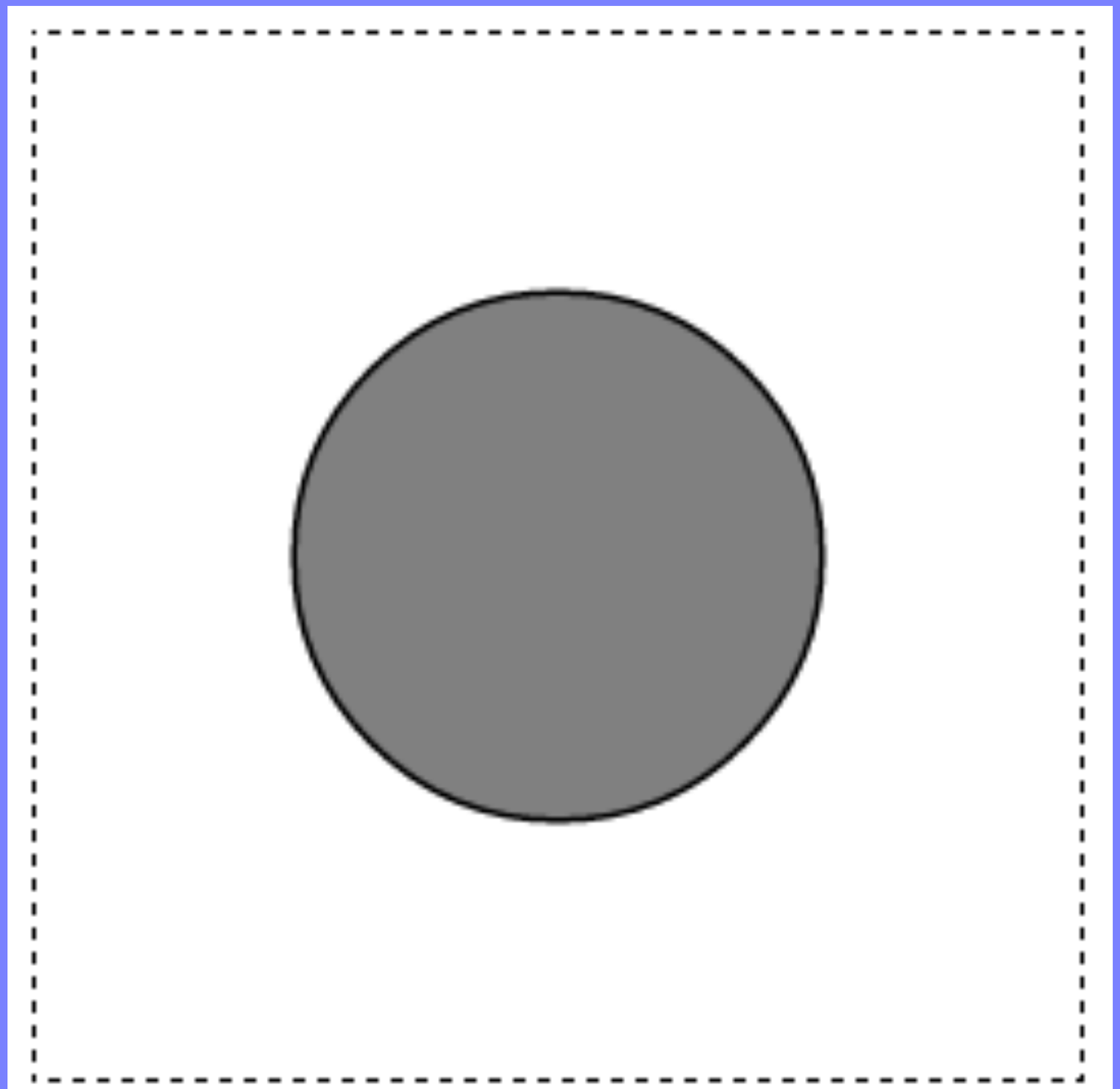



```
ctx.rect(50, 40, 70, 90);
```

Circles

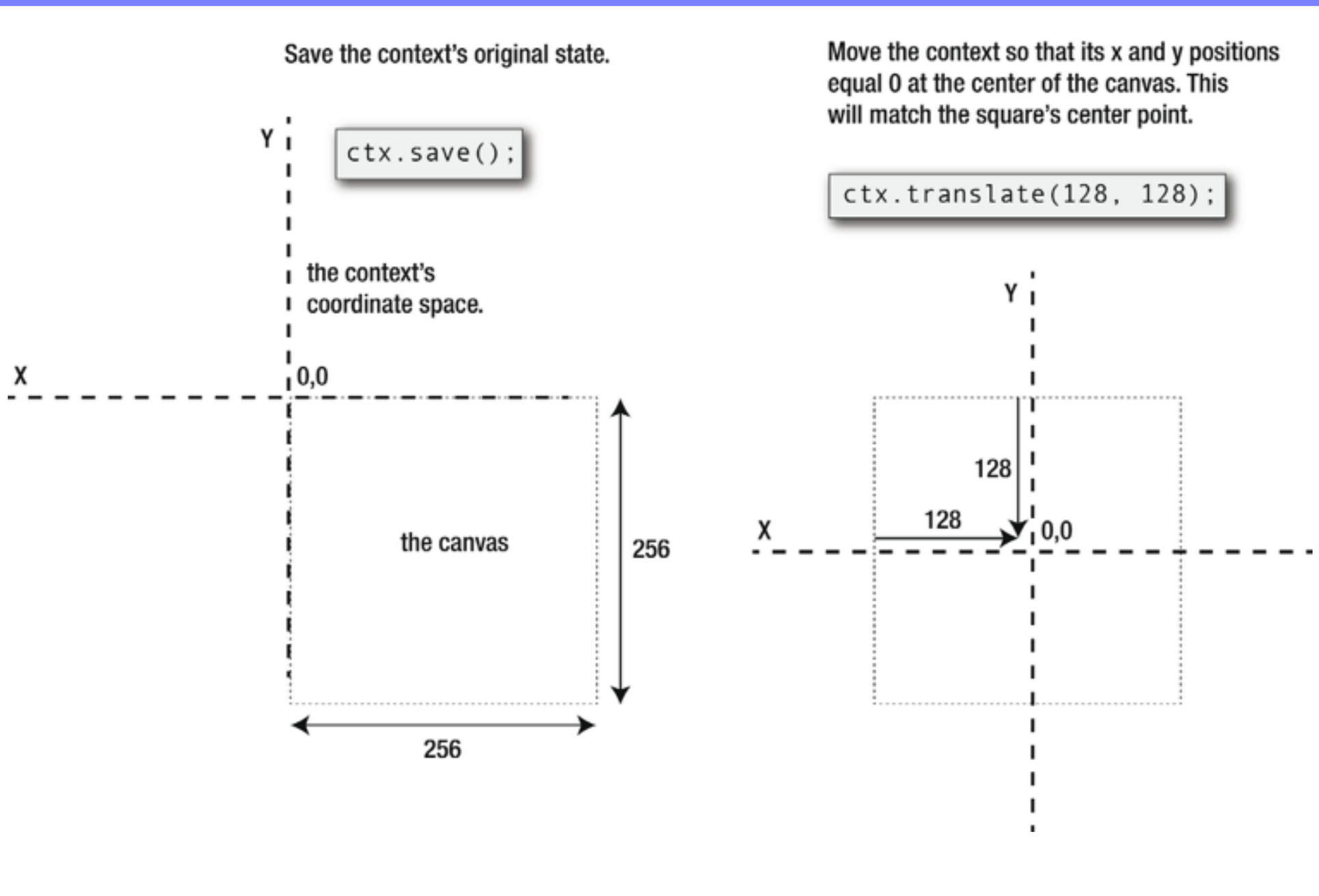
```
arc(centerX, centerY, circleRadius, startAngle, endAngle, false)
```

```
//Set the line and fill style options
ctx.strokeStyle = "black";
ctx.lineWidth = 3;
ctx.fillStyle = "gray";

//Draw the circle
ctx.beginPath();
ctx.arc(128, 128, 64, 0, 2*Math.PI, false);
ctx.closePath();
ctx.stroke();
ctx.fill();
```

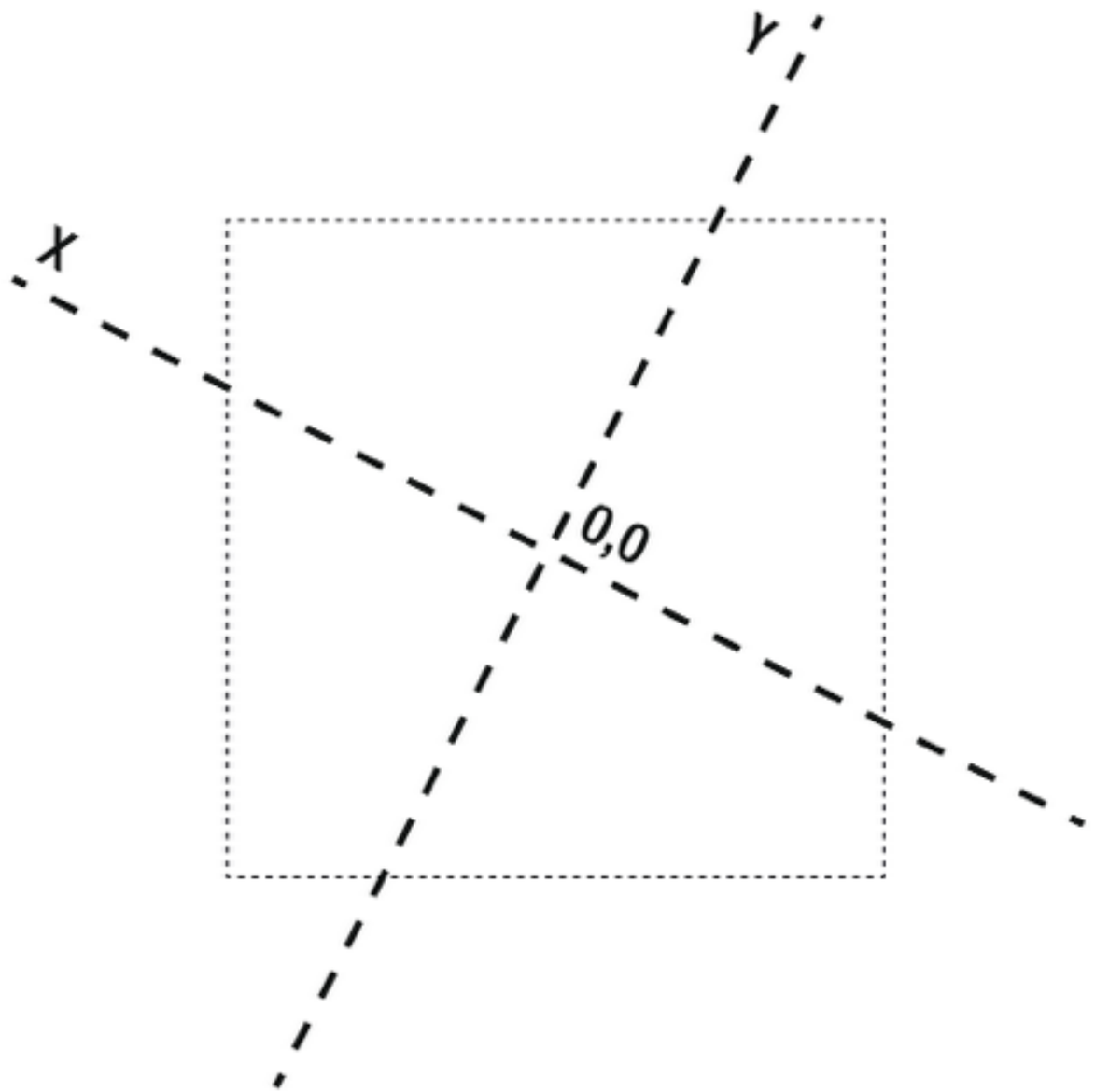


```
ctx.arc(128, 128, 64, 0, 2*Math.PI, false)
```

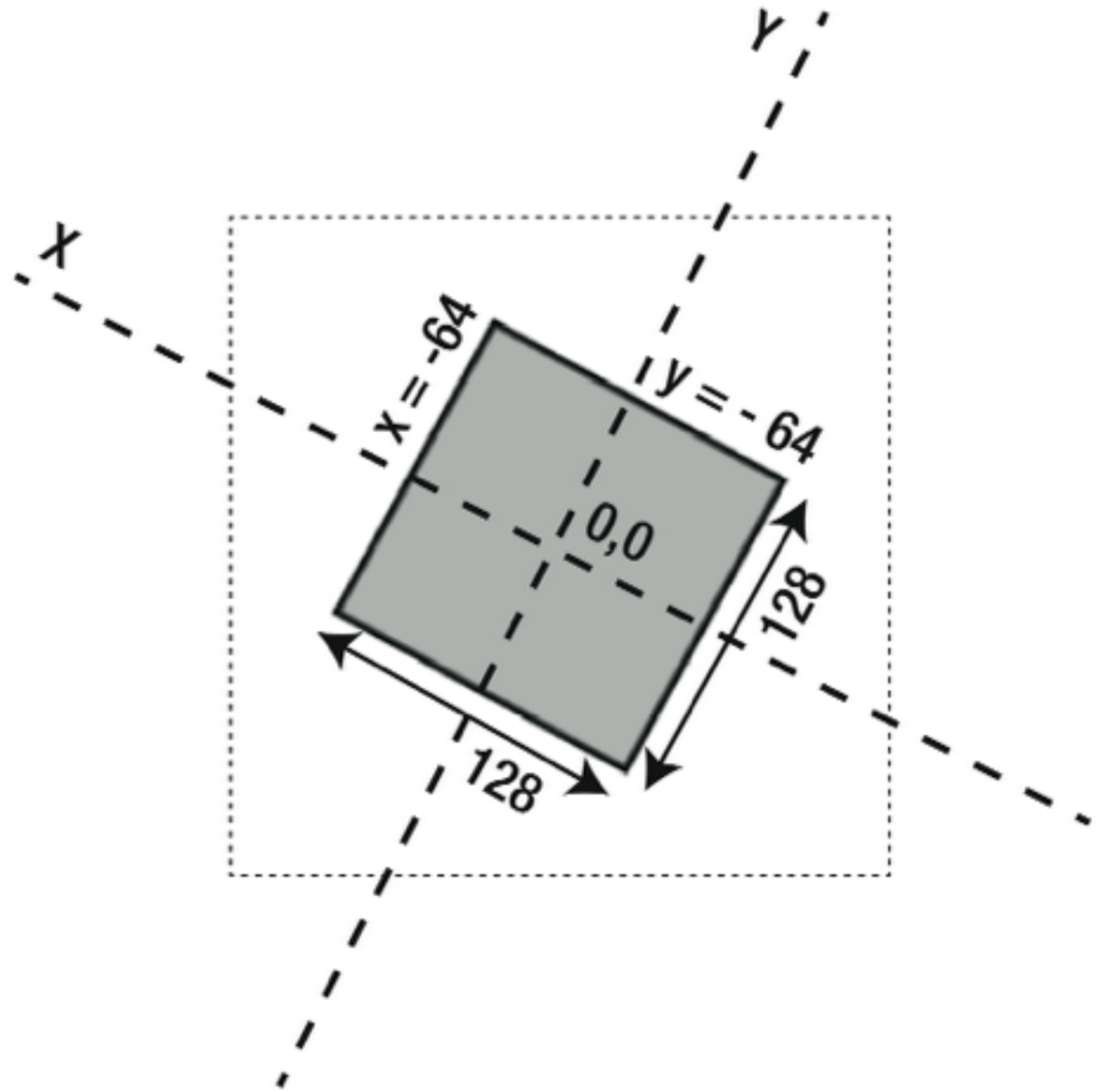
Rotation


Save the context's original state.
ctx.save();
Y
X
0,0
the context's
coordinate space.
the canvas
256
256
Move the context so that its x and y positions
equal 0 at the center of the canvas. This
will match the square's center point.
ctx.translate(128, 128);
Y
X
128
128
0,0

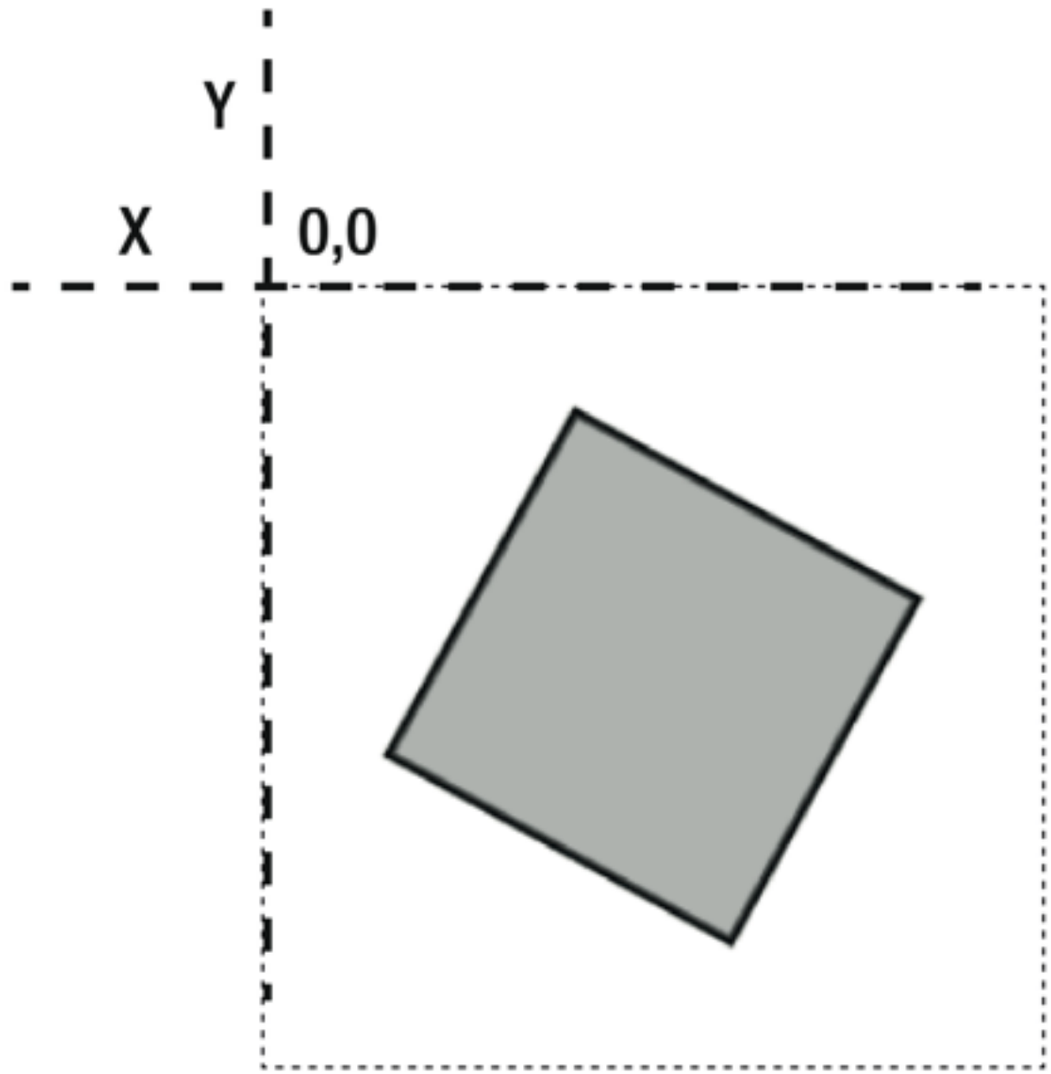
```
ctx.rotate(0.5);
```



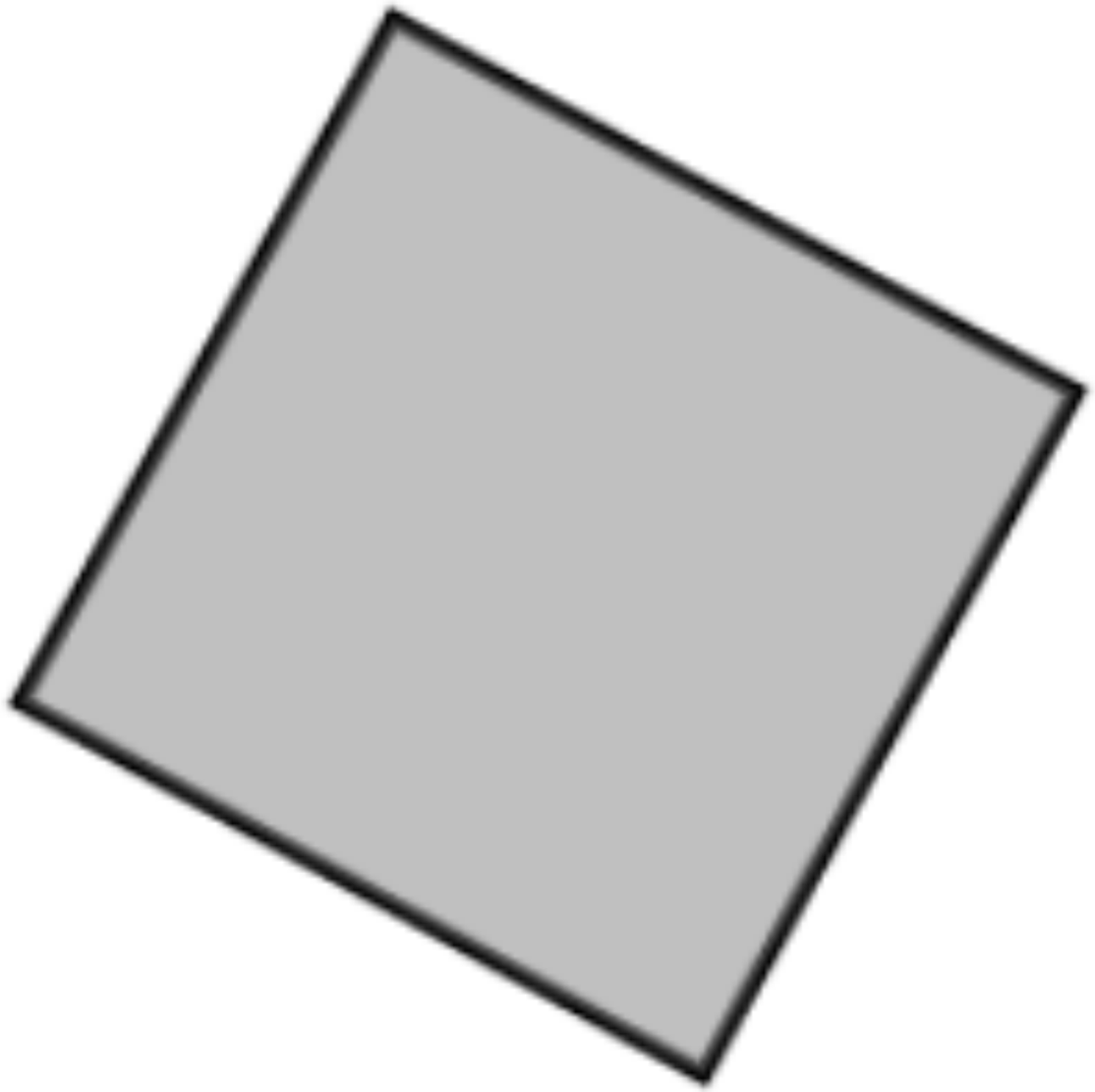
```
ctx.rect(-64, -64, 128, 128);
```



```
ctx.restore();
```



```
//Set the line and fill style options
ctx.strokeStyle = "black";
ctx.lineWidth = 3;
ctx.fillStyle = "rgba(128, 128, 128, 0.5)";

//Save the current state of the drawing context before it's rotated
ctx.save();

//Shift the drawing context's 0,0 point from the canvas's top left
//corner to the center of the canvas. This will be the
//square's center point
ctx.translate(128, 128);

//Rotate the drawing context's coordinate system 0.5 radians
ctx.rotate(0.5);

//Draw the square from -64 x and -64 y. That will mean its center
//point will be at exactly 0, which is also the center of the
//context's coordinate system
ctx.beginPath();
ctx.rect(-64, -64, 128, 128);
ctx.closePath();
ctx.stroke();
ctx.fill();

//Restore the drawing context to
//its original position and rotation
ctx.restore();
```



Transparency

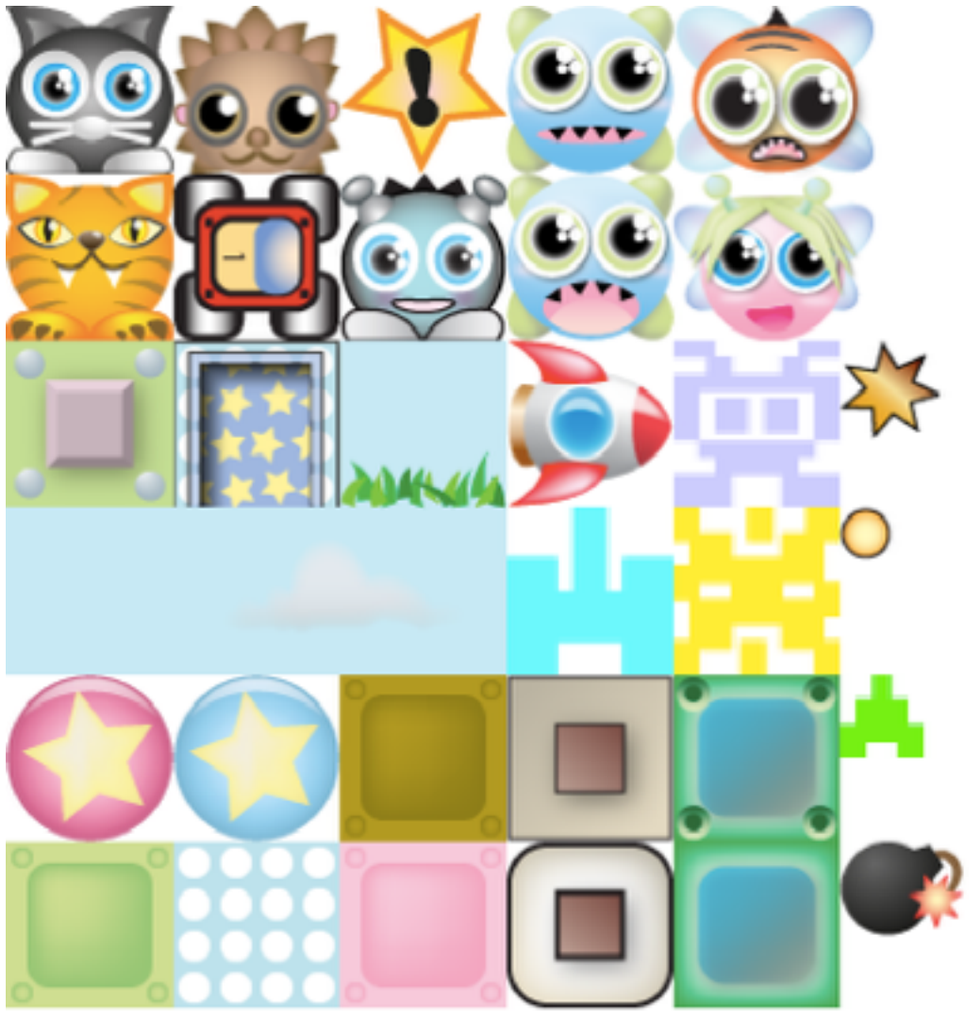
```
//Set the fill style options
ctx.fillStyle = "black";

//Draw the rectangle
ctx.save();
ctx.beginPath();
ctx.globalAlpha = 0.6;
ctx.rect(32, 32, 128, 128);
ctx.stroke();
ctx.fill();
ctx.restore();

//Draw the circle
ctx.save();
ctx.beginPath();
ctx.globalAlpha = 0.3;
ctx.arc(160, 160, 64, 0, 6.28, false)
ctx.fill();
ctx.restore();
```

Display an image

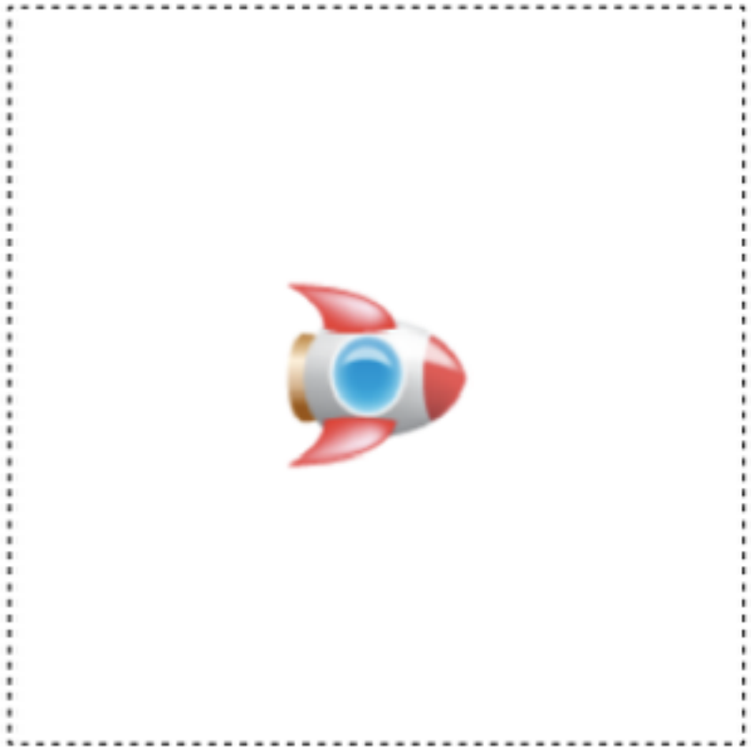
```
ctx.drawImage(imageObject, xPosition, yPosition);
```

```
//Load an image
var catImage = new Image();
catImage.addEventListener("load", loadHandler, false);
catImage.src = "images/cat.png";

//The loadHandler is called when the image has loaded
function loadHandler() {
  ctx.drawImage(catImage, 64, 64);
}
```

Blitting



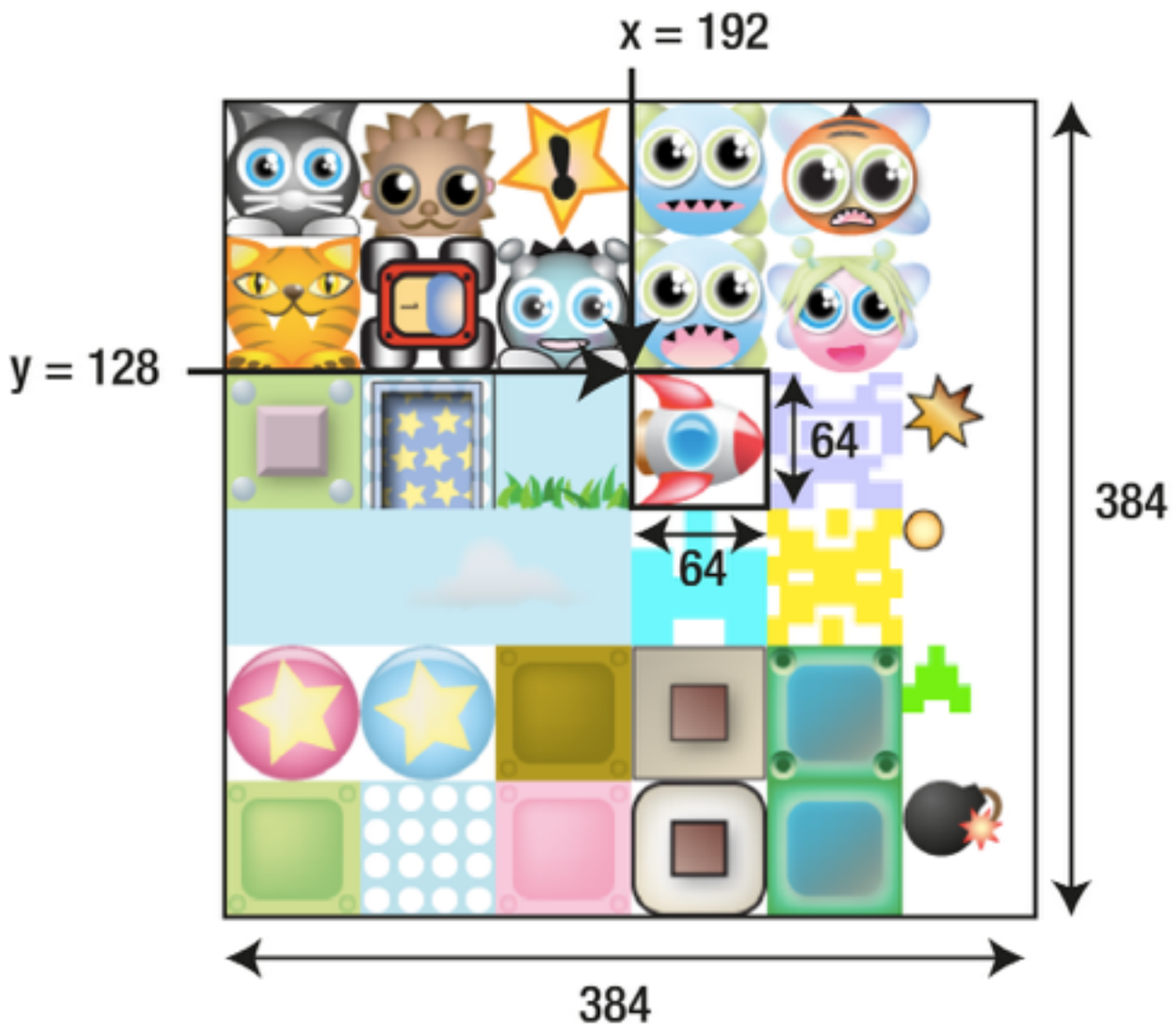
tileset.png



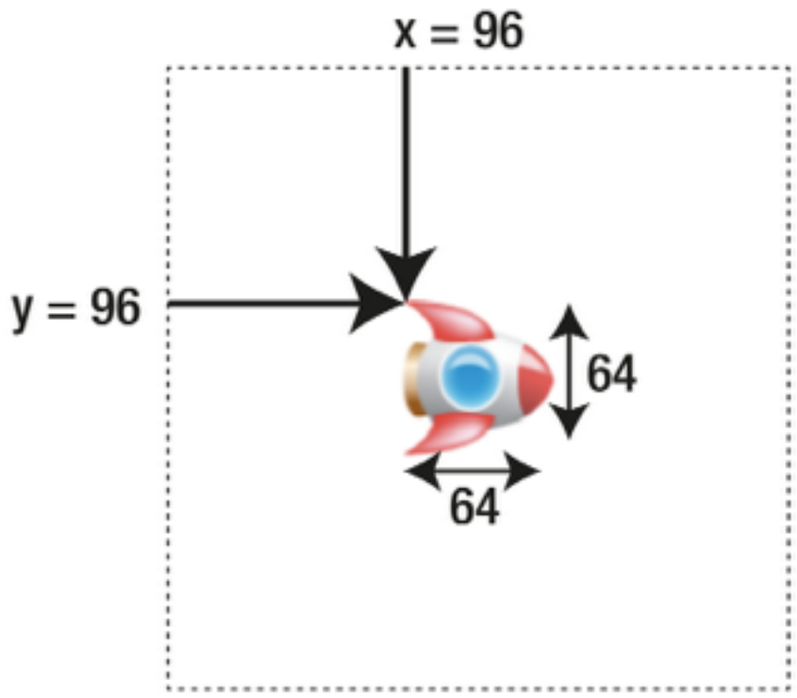
The canvas

```
//Load the tileset image
var tileset = new Image();
tileset.addEventListener("load", loadHandler, false);
tileset.src = "images/tileset.png";

//The loadHandler is called when the image has loaded
function loadHandler() {
  ctx.drawImage(
    tileset,      //The image file
    192, 128,     //The source x and y position
    64, 64,       //The source height and width
    96, 96,       //The destination x and y position
    64, 64        //The destination height and width
  );
}
```

source



destination



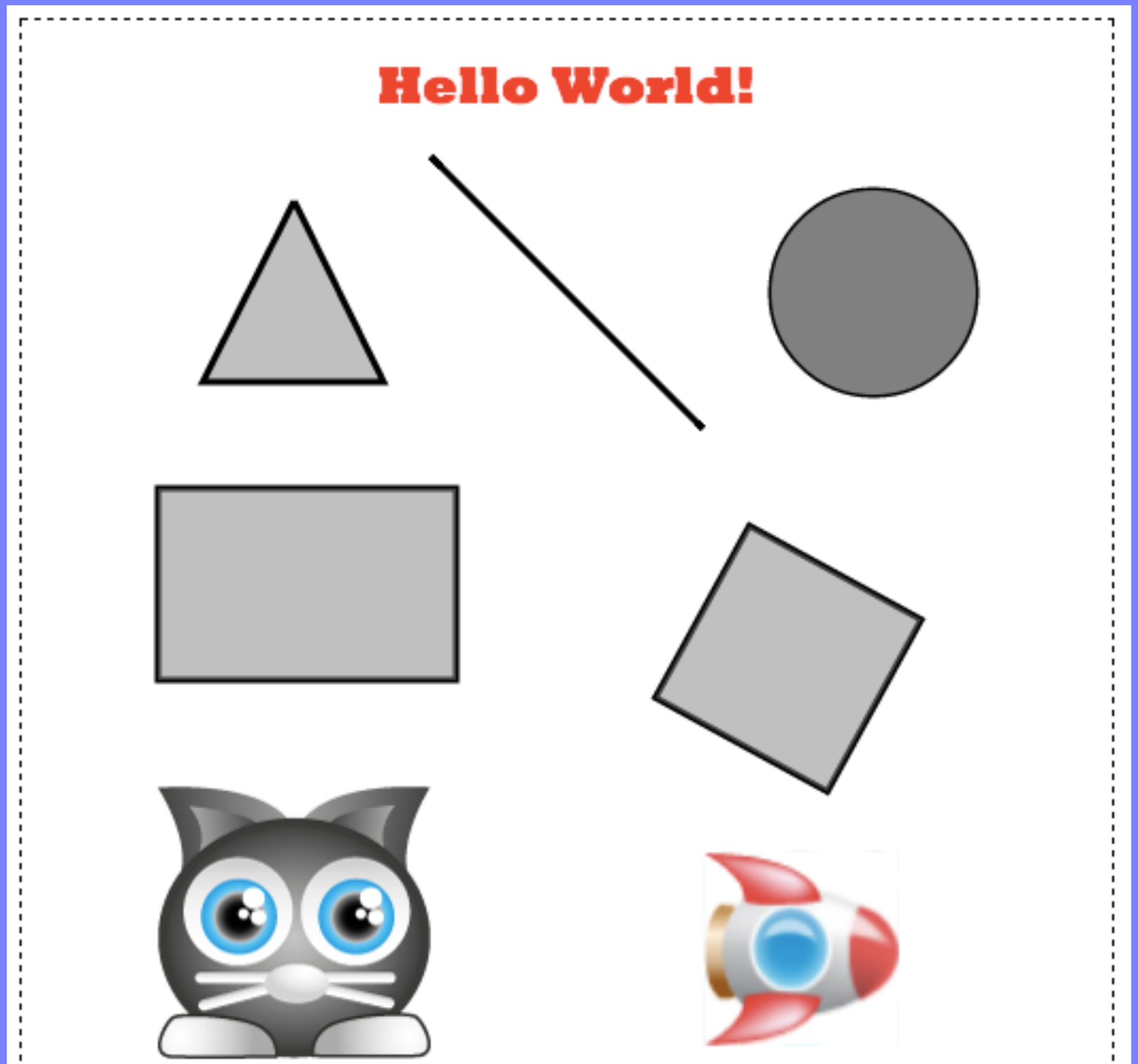

```
ctx.drawImage(
  tileset,
  192, 128, 64, 64,
  96, 96, 64, 64
);
```

```
ctx.drawImage(
  tileset,
  192, 128, 64, 64,
  96, 96, 64, 64
);
```

Text

```
//Create a text string defines the content you want to display
var content = "Hello World!";

//Assign the font to the canvas context
ctx.font = "24px 'Rockwell Extra Bold', 'Futura', sans-serif";

//Set the font color to red
ctx.fillStyle = "red";

//Figure out the width and height of the text
var width = ctx.measureText(content).width,
    height = ctx.measureText("M").width;

//Set the text's x/y registration point to its top left corner
ctx.textBaseline = "top";

//Use `fillText` to Draw the text in the center of the canvas
ctx.fillText(
  content,                          //The text string
  canvas.width / 2 - width / 2,     //The x position
  canvas.height / 2 - height / 2    //The y position
);
```

Hello World!



examples/canvasPlayground.html

More canvas stuff

Scale
Blend modes
Compositing
Shadows
Curves

You can’t make a game like this.

The code will crumble in your hands.

Then how?

Sprites

The LEGO bricks of game design.

Make sprites using canvas components.
Snap sprites together to make a game.

Making sprites

1. An array to store all the game sprites

```
var sprites = [];
```

1. A function that returns a sprite object

```
function sprite (property) {

  //Create an object
  var o = {};

  //...Assign properties to the object
  //that refer to canvas properties…
  o.property = property;

  //Add a `render` method that explains how to draw the sprite
  o.render = function(ctx) {
    //...canvas drawing operations...
  };

  //Push the sprite object into the `sprites` array
  sprites.push(o);

  //Return the object
  return o;

}
```

2. A function to render the sprite on the canvas

```
function render() {

  //Clear the canvas
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  //Loop through each sprite in the sprites array
  sprites.forEach(function(sprite){

    //Save the canvas state
    ctx.save();

    //Shift the canvas to the sprite's position and set
    //Rotation, alpha and scale values

    //Use the sprite's own `render` method to draw the sprite
    sprite.render(ctx);

    //Restore the canvas state before rendering the next sprite
    ctx.restore();
  });
}
```

3. Make a sprite and render it

```
var newSprite = sprite(“propertyValue”);

render();
```

1. Make a rectangle sprite function

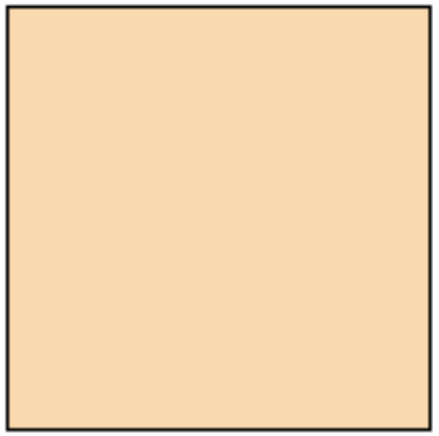
```
var rectangle = function(width, height, fillStyle, strokeStyle, lineWidth) {

  //Create an object
  var o = {};

  //Assign properties to the object that refer to canvas properties
  o.x = 0;
  o.y = 0;
  o.width = width;
  o.height = height;
  o.fillStyle = fillStyle;
  o.strokeStyle = strokeStyle;
  o.lineWidth = lineWidth;

  //Optional rotation and alpha
  o.rotation = 0;
  o.alpha = 1;

  //Add `vx` and `vy` (velocity) variables that will help us move the sprite
  o.vx = 0;
  o.vy = 0;

  //Add a `render` method that explains how to draw the sprite
  o.render = function(ctx) {
    ctx.strokeStyle = o.strokeStyle;
    ctx.lineWidth = o.lineWidth;
    ctx.fillStyle = o.fillStyle;
    ctx.beginPath();
    ctx.rect(-o.width / 2, -o.height / 2, o.width, o.height);
    ctx.closePath();
    ctx.stroke();
    ctx.fill();
  };

  //Push the sprite object into the `sprites` array
  sprites.push(o);

  //Return the object
  return o;
};
```

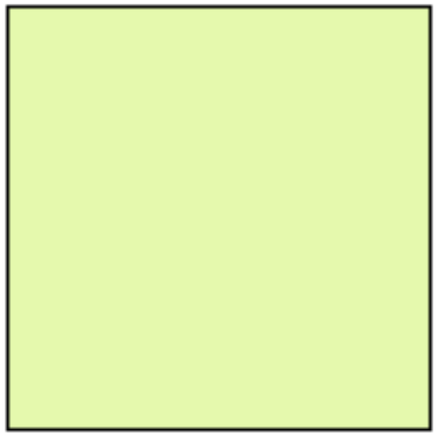
2. A function to render sprites

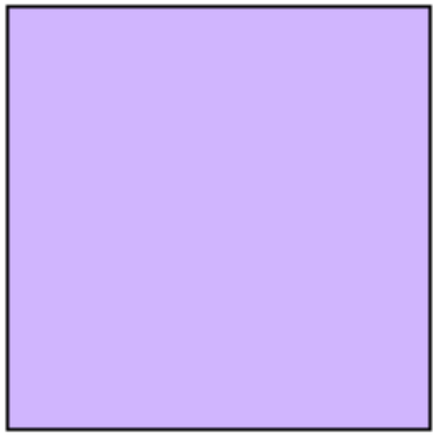
```
function render() {

  //Clear the canvas
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  //Loop through all the sprites in the `sprites` array
  sprites.forEach(function(sprite){

    //Save the canvas context's current state
    ctx.save();

    //Shift the canvas to the sprite's position
    ctx.translate(
      sprite.x + sprite.width / 2,
      sprite.y + sprite.height /2
    );

    //Set the sprite's `rotation` and `alpha`
    ctx.rotate(sprite.rotation);
    ctx.globalAlpha = sprite.alpha;

    //Use the sprite's own `render` method to draw the sprite
    sprite.render(ctx);

    //Restore the canvas for the next sprite
    ctx.restore();
  });

}
```

3. Make some sprites and render them

```
var boxOne = rectangle(128, 128, "#FFDAAB", "black", 2);
boxOne.x = 64;
boxOne.y = 64;

var boxTwo = rectangle(128, 128, "#DDFFAB", "black", 2);
boxTwo.x = 300;
boxTwo.y = 200;

var boxThree = rectangle(128, 128, "#D9ABFF", "black", 2);
boxThree.x = 64;
boxThree.y = 300;

render();
```







Moving sprites

1. Assign the sprites' velocities

```
var boxOne = rectangle(128, 128, "#FFDAAB", "black", 2);
boxOne.x = 64;
boxOne.y = 64;
boxOne.vx = 1;
boxOne.vy = -1;

var boxTwo = rectangle(128, 128, "#DDFFAB", "black", 2);
boxTwo.x = 300;
boxTwo.y = 200;
boxTwo.vx = -1;
boxTwo.vy = 1;

var boxThree = rectangle(128, 128, "#D9ABFF", "black", 2);
boxThree.x = 64;
boxThree.y = 300;
boxThree.vx = -1;
boxThree.vy = -1;
```

2. Create a **game loop**

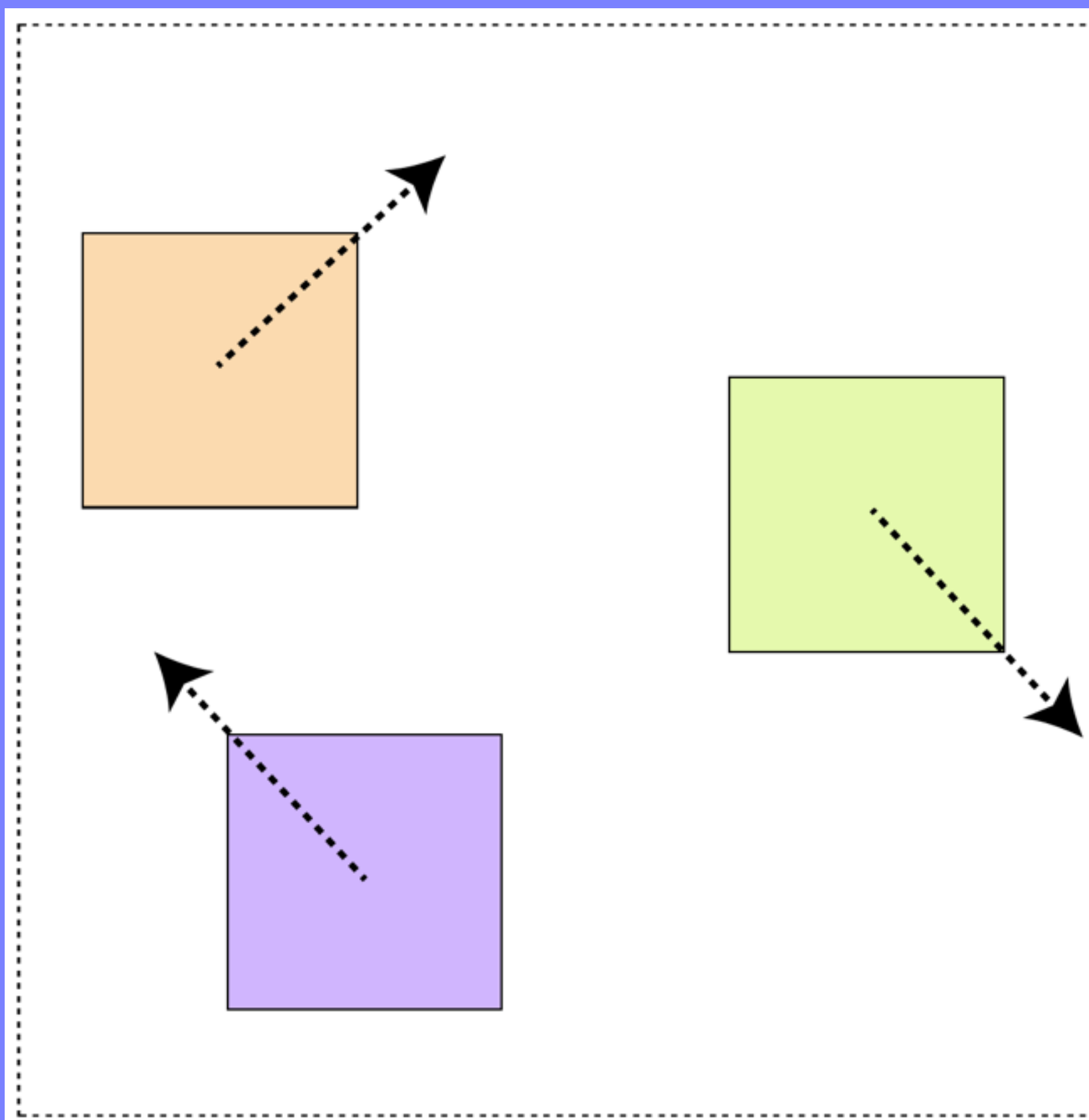
```
function gameLoop() {
  //Create a loop
  requestAnimationFrame(gameLoop, canvas);

  //Update the game logic and render
  //the sprites 60 times per second:
  update();
  render();
}

//Start the game loop
gameLoop();
```

3. Update the game logic and renderer

```
//The game logic (called 60 times per second)
function update() {

  //Loop through all the sprites
  sprites.forEach(function(sprite){

    //a. Change the sprite's velocity in some way
    //... ??? ...

    //b. Add the new velocity to the sprite's current position
    //to make the sprite move

    sprite.x += sprite.vx;
    sprite.y += sprite.vy;

  });
}

//The renderer (called 60 times per second)
function render() {
  //... Render all the sprites ...
}
```

Make sprites bounce off the canvas edges

```
function update() {
  sprites.forEach(function(sprite){
    sprite.x += sprite.vx;
    sprite.y += sprite.vy;

    //Screen boundaries
    //Left
    if (sprite.x < 0) {
      //Position the sprite inside the canvas
      sprite.x = 0;
      //Reverse its velocity to make it bounce
      sprite.vx = -sprite.vx;
    }
    //Right
    if (sprite.x + sprite.width > canvas.width) {
      sprite.x = canvas.width - sprite.width;
      sprite.vx = -sprite.vx;
    }
    //Top
    if (sprite.y < 0) {
      sprite.y = 0;
      sprite.vy = -sprite.vy;
    }
    //Bottom
    if (sprite.y + sprite.width > canvas.height) {
      sprite.y = canvas.height - sprite.width;
      sprite.vy = -sprite.vy;
    }
  });
}
```



Interactivity

Use **event listeners**

```
window.addEventListener(“keydown", function(event) {
  //do something when a key is pressed down
}, false);

window.addEventListener("keyup", function(event) {
  //do something when a key is released
}, false);
```

1. Assign key codes and direction trackers.

```
//Arrow key codes
var UP = 38,
    DOWN = 40,
    RIGHT = 39,
    LEFT = 37;

//Directions
var moveUp = false,
    moveDown = false,
    moveRight = false,
    moveLeft = false;
```

2. Program the keyboard even listeners

```
window.addEventListener(
  "keydown",
  function(event) {
    switch(event.keyCode){
      case UP:
        moveUp = true;
        break;

      case DOWN:
        moveDown = true;
        break;

      case LEFT:
        moveLeft = true;
        break;

      case RIGHT:
        moveRight = true;
        break;
    }
  },
  false
);
```

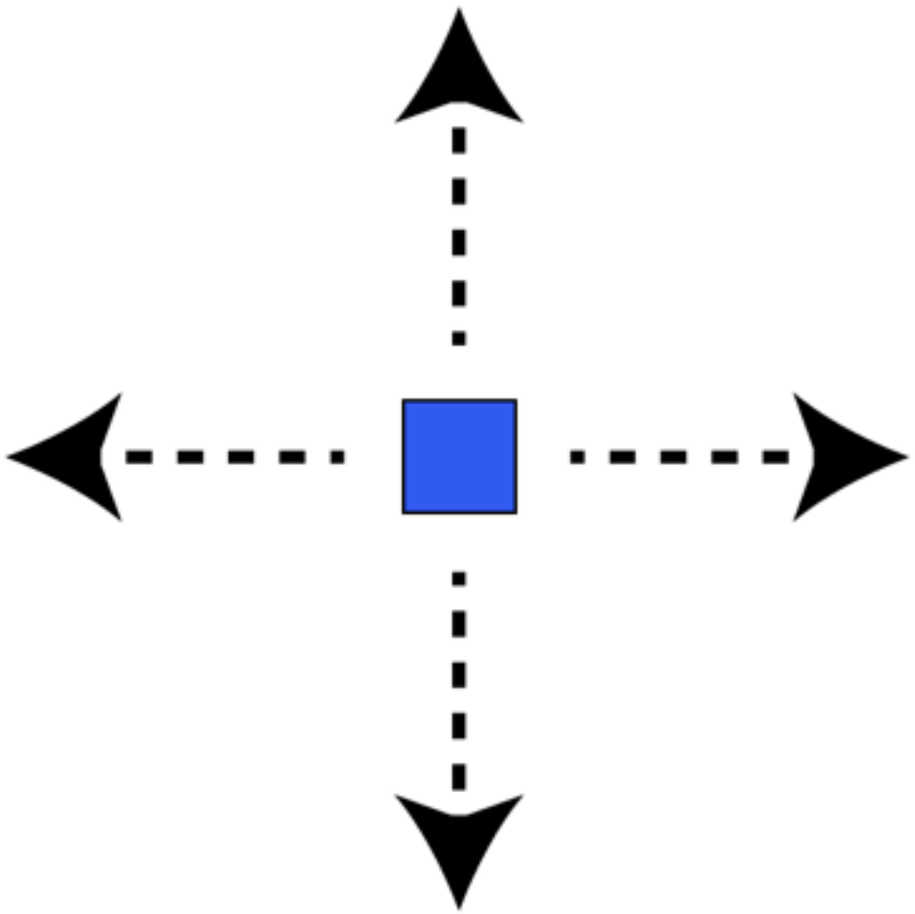
```
window.addEventListener(
  "keyup",
  function(event) {
    switch(event.keyCode){
      case UP:
        moveUp = false;
        break;

      case DOWN:
        moveDown = false;
        break;

      case LEFT:
        moveLeft = false;
        break;

      case RIGHT:
        moveRight = false;
        break;
    }
  },
  false
);
```

3. Use the directions to update the sprite's velocity in the game loop.

```
function update() {
  //Up
  if(moveUp && !moveDown) {
    box.vy = -5;
  }
  //Down
  if(moveDown && !moveUp) {
    box.vy = 5;
  }
  //Left
  if(moveLeft && !moveRight) {
    box.vx = -5;
  }
  //Right
  if(moveRight && !moveLeft) {
    box.vx = 5;
  }

  //Set the box's velocity to zero if none
  //of the keys are being pressed
  if(!moveUp && !moveDown) {
    box.vy = 0;
  }
  if(!moveLeft && !moveRight) {
    box.vx = 0;
  }

  //Move the box
  box.x += box.vx;
  box.y += box.vy;
}
```

3. Set the screen boundaries

```
//Left
if (box.x < 0) {
  //Position the box inside the canvas
  box.x = 0;
}
//Right
if (box.x + box.width > canvas.width) {
  box.x = canvas.width - box.width;
}
//Top
if (box.y < 0) {
  box.y = 0;
}
//Bottom
if (box.y + box.width > canvas.height) {
  box.y = canvas.height - box.width;
}
```



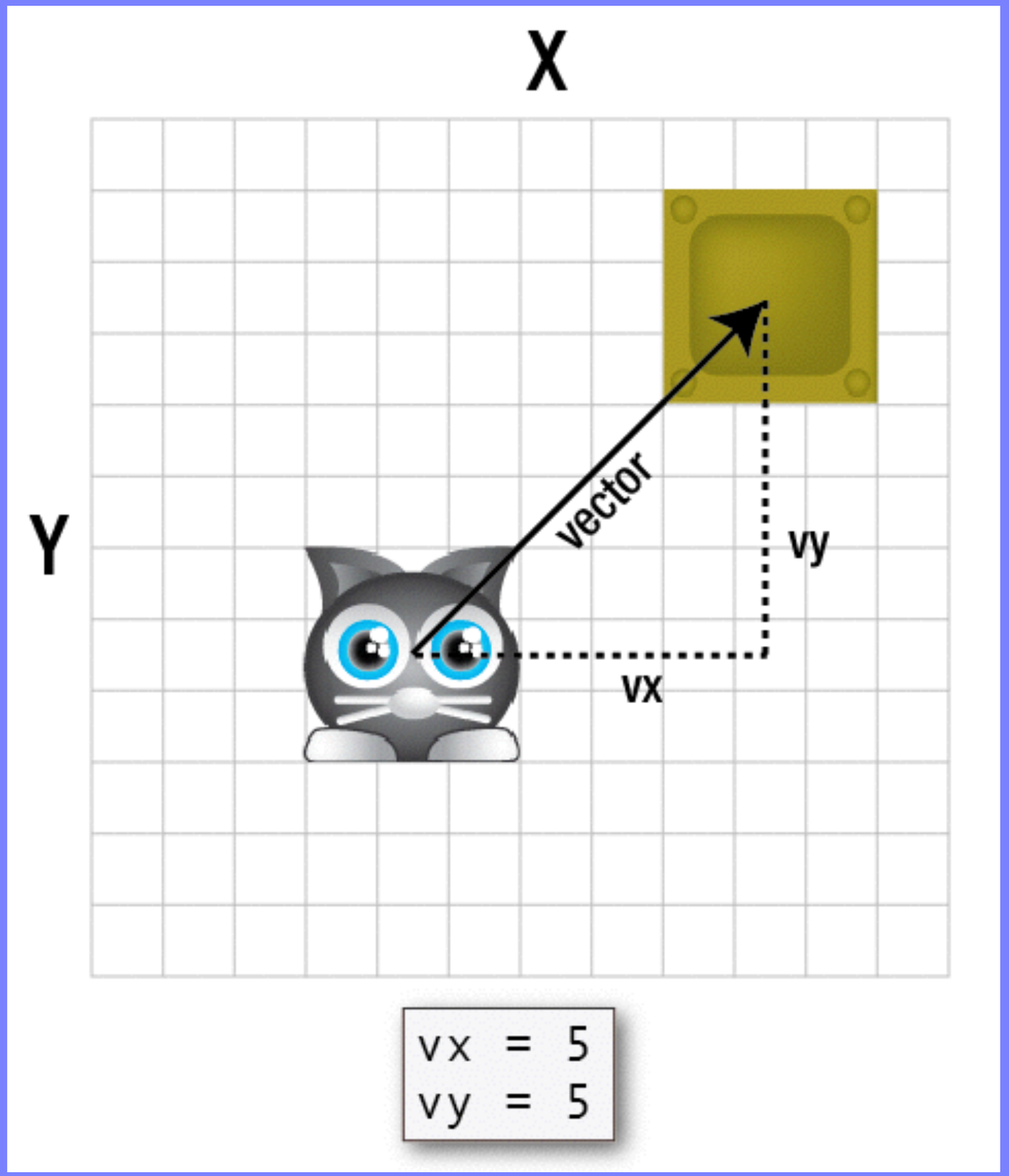
Collision detection

Check whether rectangles are overlapping

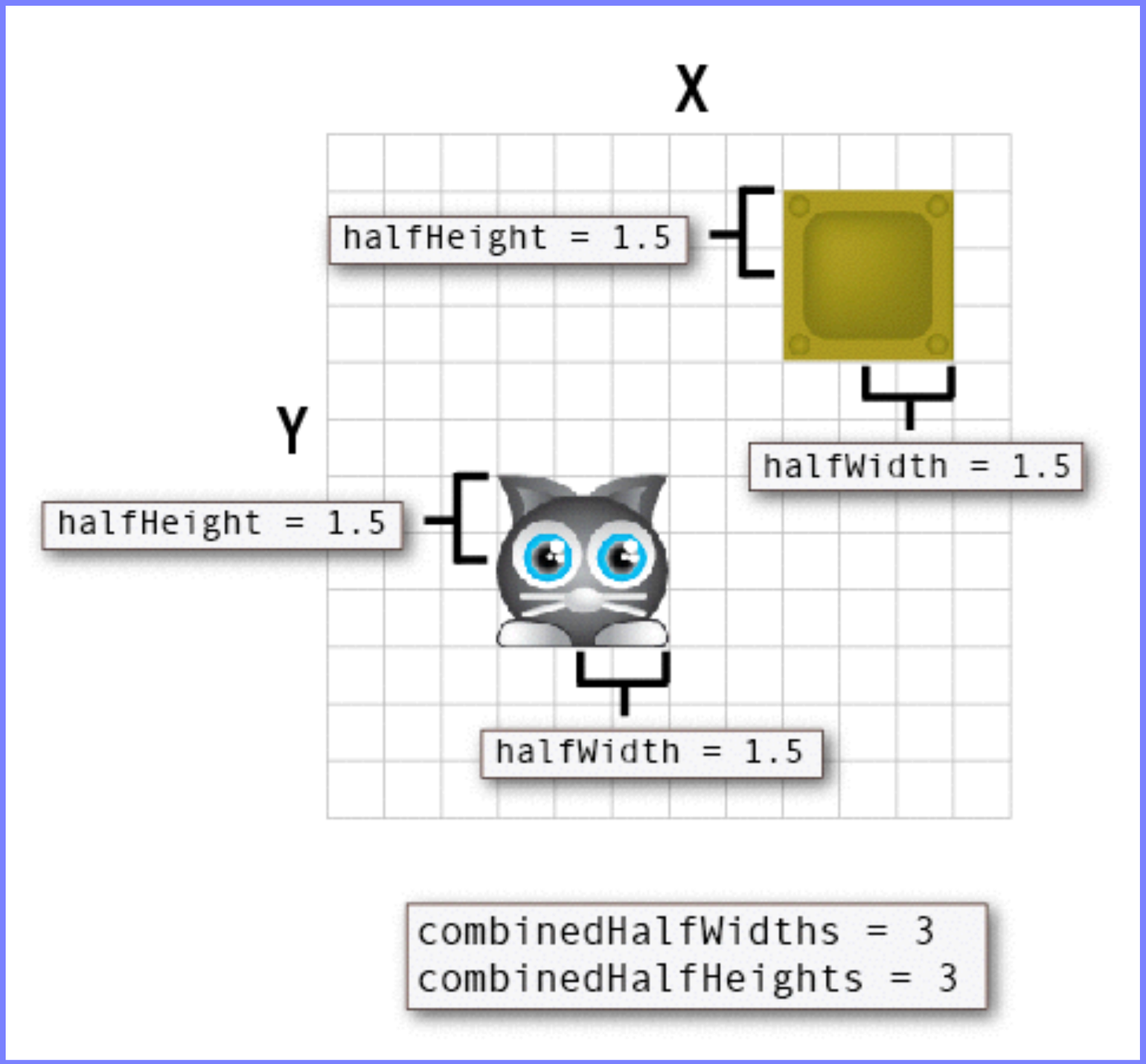
No collision...

hit!

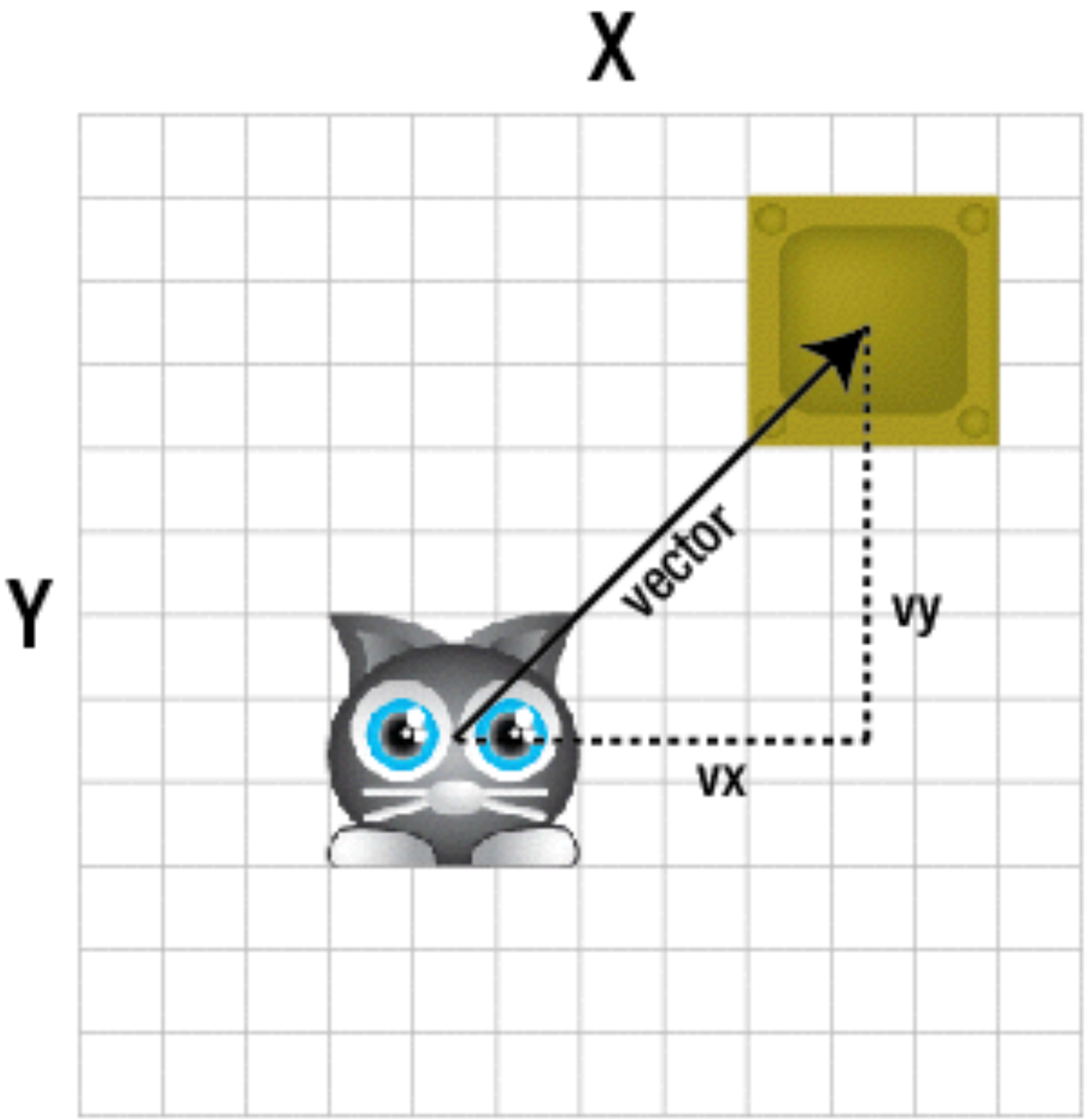
1. Figure out the vector.



2. Figure out the combined half-widths and half-heights.



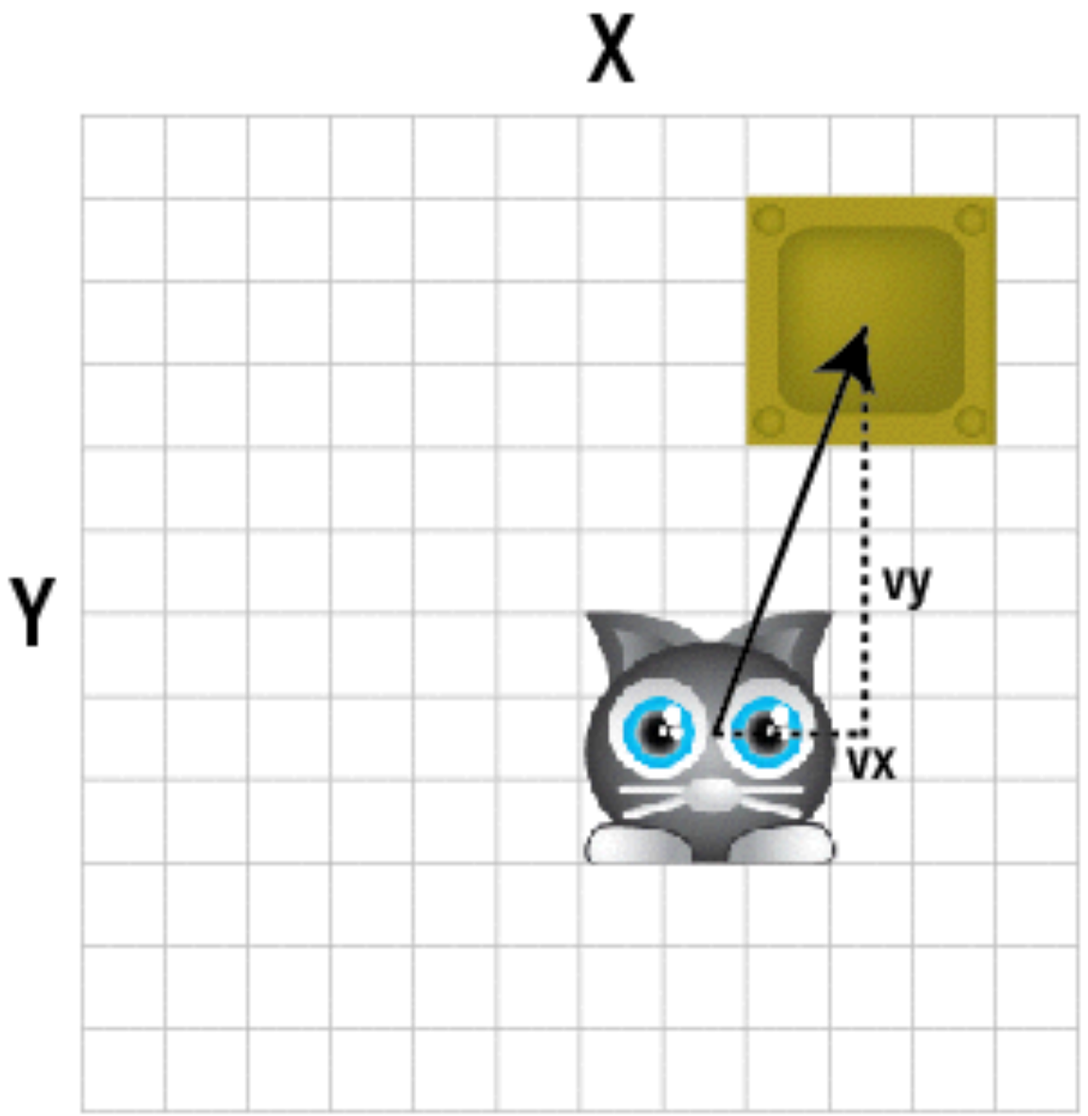
If the combined half-widths and half-heights are less than the vx and vy variables, then the sprites are touching.




```
combinedHalfWidths = 3
combinedHalfHeights = 3
```

```
vx = 5
vy = 5
```

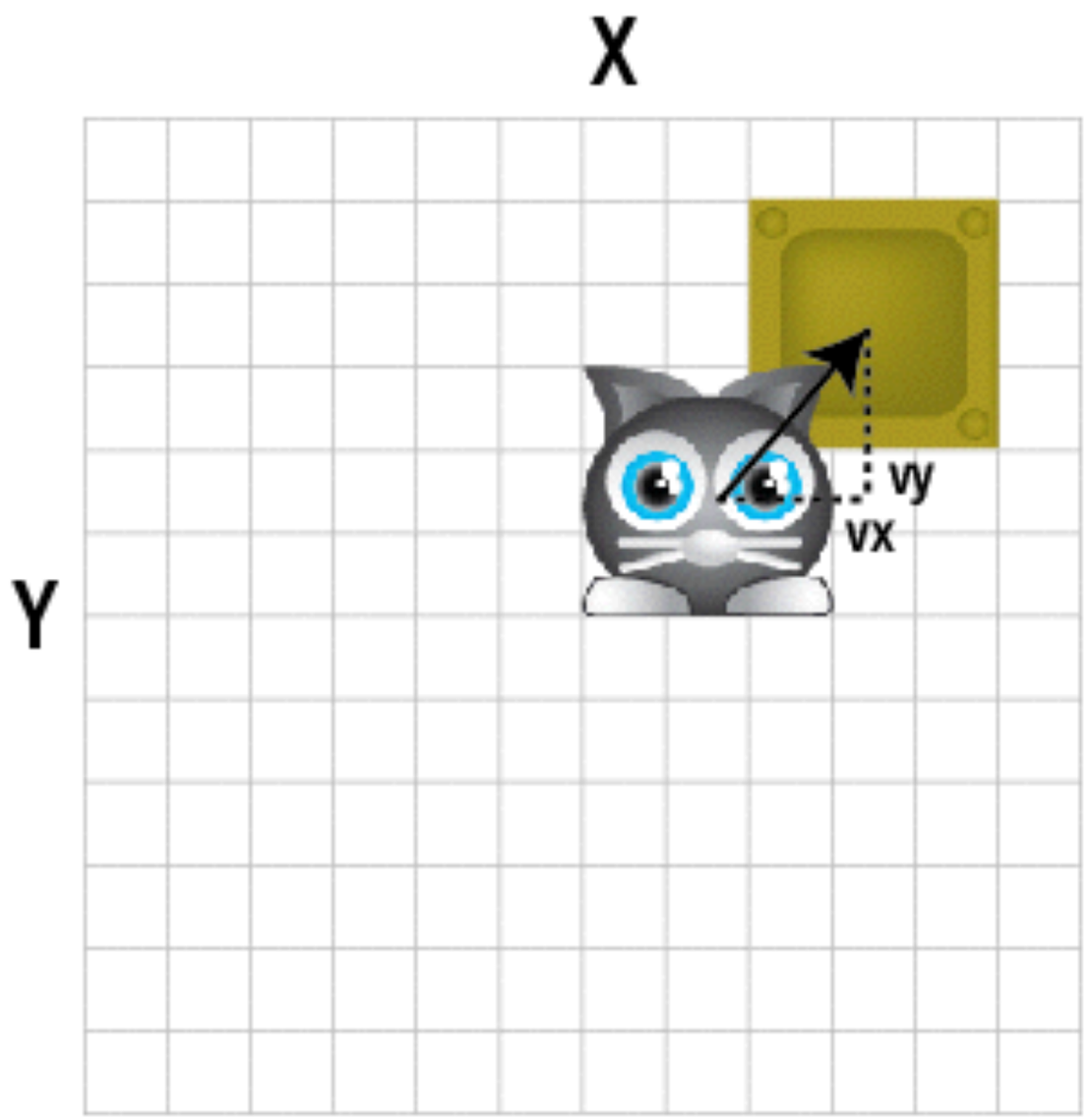
There's no collision




```
combinedHalfWidths = 3
combinedHalfHeights = 3
```

```
vx = 2
vy = 5
```

The vx value is less than the combinedHalfWidths.
A collision might be approaching!




```
combinedHalfWidths = 3
combinedHalfHeights = 3
```

```
vx = 2
vy = 2
```

The vx and vy values are less than the combinedHalfWidths and combinedHalfHeights. A collision is definitely occuring!

The code: `hitTestRectangle`

```
function hitTestRectangle(r1, r2) {
  //Find the half-width values and center x/y points of two rectangles
  r1.halfWidth = r1.width / 2;
  r1.halfHeight = r1.height / 2;
  r1.centerX = r1.x + r1.halfWidth;
  r1.centerY = r1.y + r1.halfHeight;

  r2.halfWidth = r2.width / 2;
  r2.halfHeight = r2.height / 2;
  r2.centerX = r2.x + r2.halfWidth;
  r2.centerY = r2.y + r2.halfHeight;

  //A variable to determine whether there's a collision
  var hit = false;

  //Calculate the distance vector
  var vx = r1.centerX - r2.centerX;
  var vy = r1.centerY - r2.centerY;

  //Figure out the combined half-widths and half-heights
  var combinedHalfWidths = r1.halfWidth + r2.halfWidth;
  var combinedHalfHeights = r1.halfHeight + r2.halfHeight;

  //Check for a collision on the x axis
  if(Math.abs(vx) < combinedHalfWidths) {
    //A collision might be occurring. Check for a collision on the y axis
    if(Math.abs(vy) < combinedHalfHeights) {
      //There's definitely a collision happening
      hit = true;
    }
    else {
      //There's no collision on the y axis
      hit = false;
    }
  }
  else {
    //There's no collision on the x axis
    hit = false;
  }
  return hit;
}
```

Use `hitTestRectangle` like this:

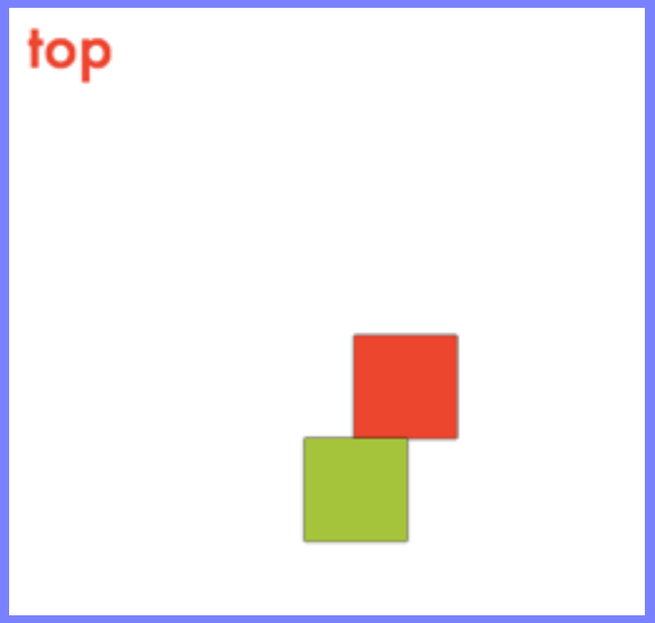
```
if(hitTestRectangle(rectangleOne, rectangleTwo)) {
  //Collision!
} else {
  //No collision.
}
```

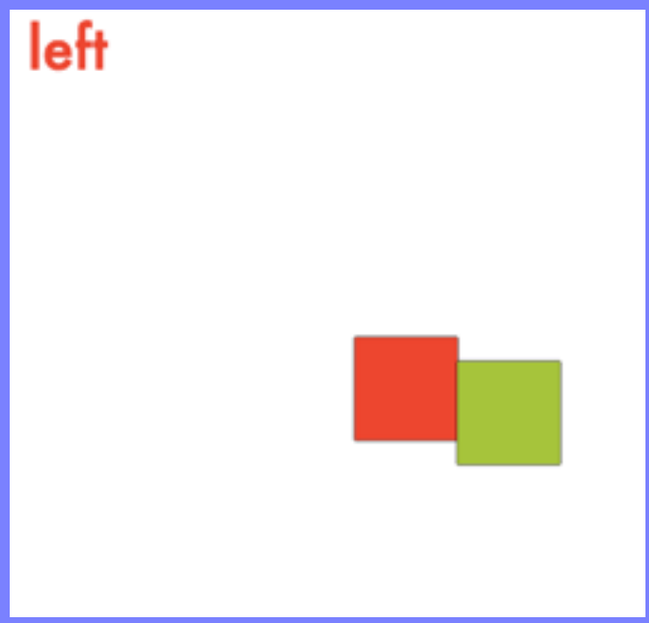
No collision...

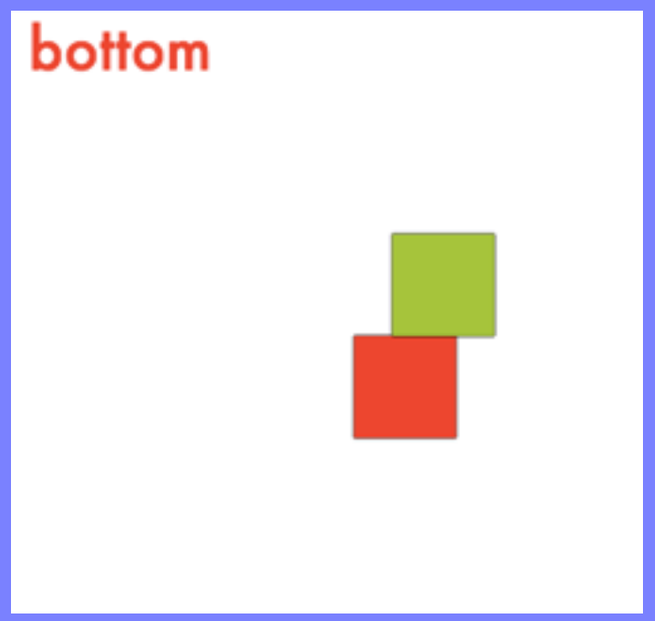
hit!

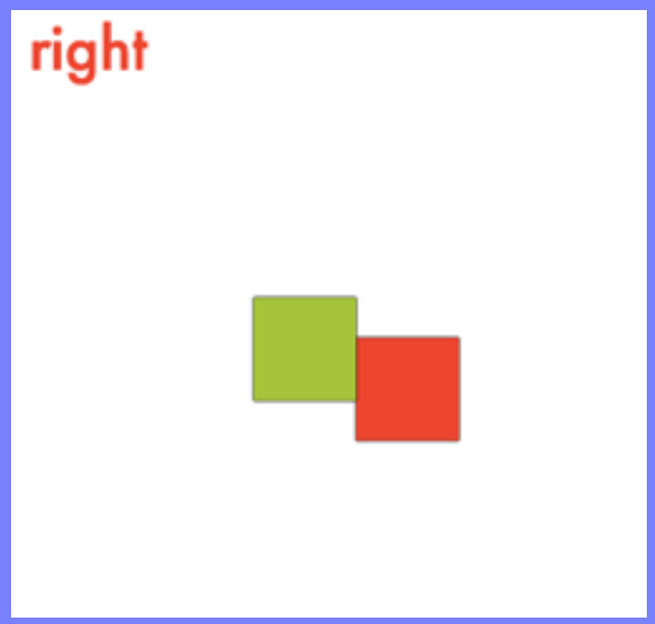
```
if(hitTestRectangle(blueBox, redBox)) {
  message.content = "hit!"
  blueBox.fillStyle = "yellowGreen";
} else {
  message.content = "No collision...";
  blueBox.fillStyle = "blue";
}
```

Preventing overlaps

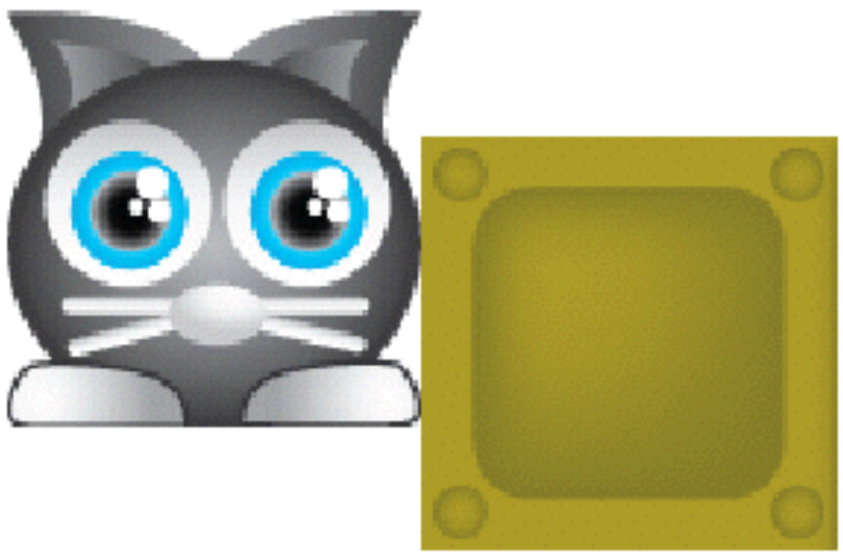




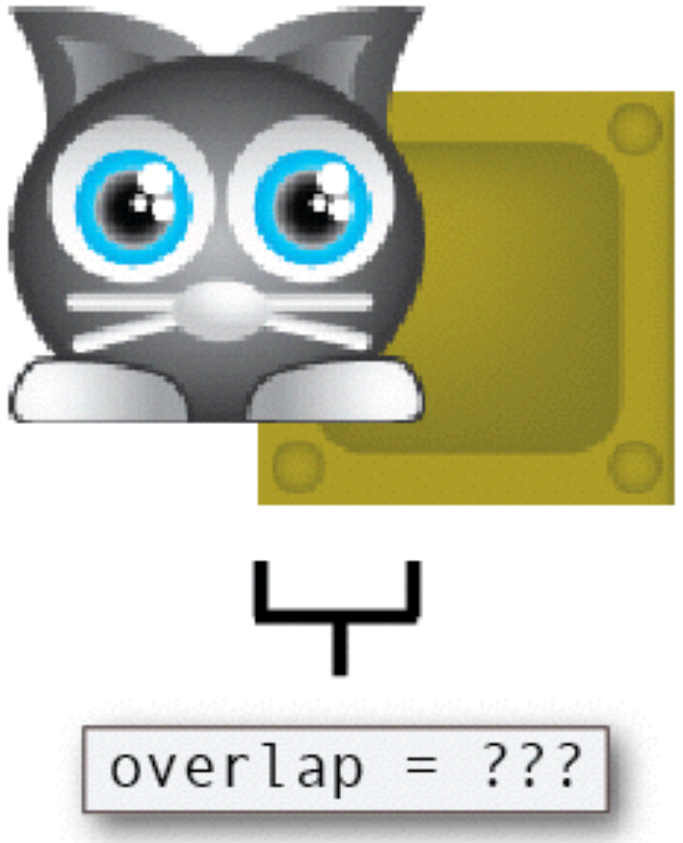


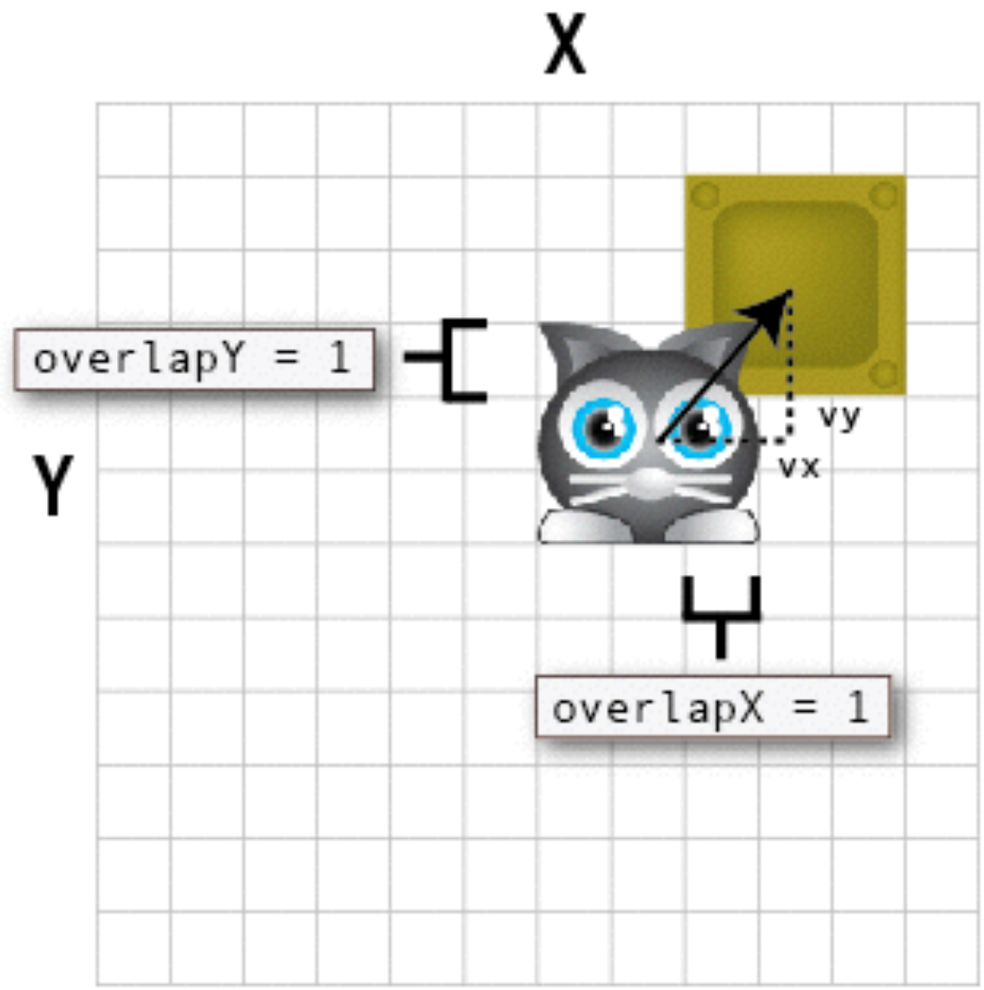


The box should block
the cat's movement
when they touch.



You can only do that
if you know by how much
the sprites overlap
when they first collide.

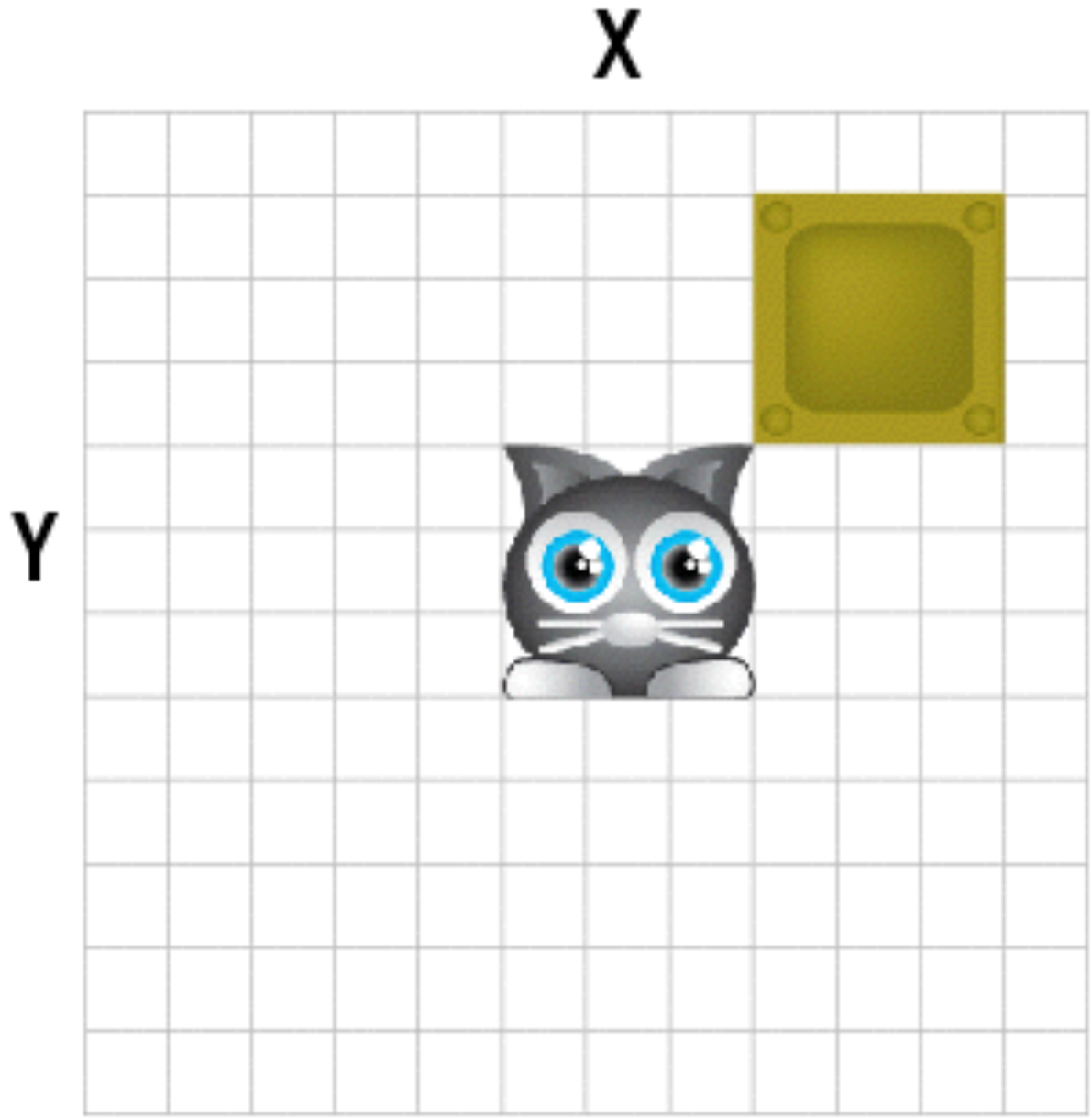





```
combinedHalfWidths = 3
combinedHalfHeights = 3
```

```
vx = 2
vy = 2
```

```
overlapX = combinedHalfWidths - vx
overlapX = 3 - 2
overlapX = 1

overlapY = combinedHalfHeights - vy
overlapY = 3 - 2
overlapY = 1
```




```
overlapX = 1
overlapY = 1
```

```
sprite.x = sprite.x - overlapX;
sprite.y = sprite.y - overlapY;
```

```
function rectangleCollision(r1, r2, bounce) {
  //... abridged: calculate the sprites' properties ...
  //Check whether vx is less than the combined half widths
  if (Math.abs(vx) < combinedHalfWidths) {
    //Check whether vy is less than the combined half heights
    if (Math.abs(vy) < combinedHalfHeights) {

      //A collision is occurring! Calculate the overlap
      overlapX = combinedHalfWidths - Math.abs(vx);
      overlapY = combinedHalfHeights - Math.abs(vy);

      //The collision has occurred on the axis with the
      //*smallest* amount of overlap.
      if (overlapX >= overlapY) {
        if (vy > 0) {
          collision = "top";
          r1.y = r1.y + overlapY;
        } else {
          collision = "bottom";
          r1.y = r1.y - overlapY;
        }
        //Bounce
        if (bounce) {
          r1.vy *= -1;
        }
      } else {
        if (vx > 0) {
          collision = "left";
          r1.x = r1.x + overlapX;
        } else {
          collision = "right";
          r1.x = r1.x - overlapX;
        }
        //Bounce
        if (bounce) {
          r1.vx *= -1;
        }
      }
    } else {
      //No collision
    }
  } else {
    //No collision
  }
  //Return the collision string. it will be either "top", "right",
  //"bottom", or "left" depending on which side of r1 is touching r2.
  return collision;
}
```

No collision...

right

```
var collision = rectangleCollision(blueBox, redBox);
if (collision) {
   message.content = collision;
   blueBox.fillStyle = "yellowGreen";
}
else {
  message.content = "No collision...";
  blueBox.fillStyle = "blue";
}
```

Playing sounds

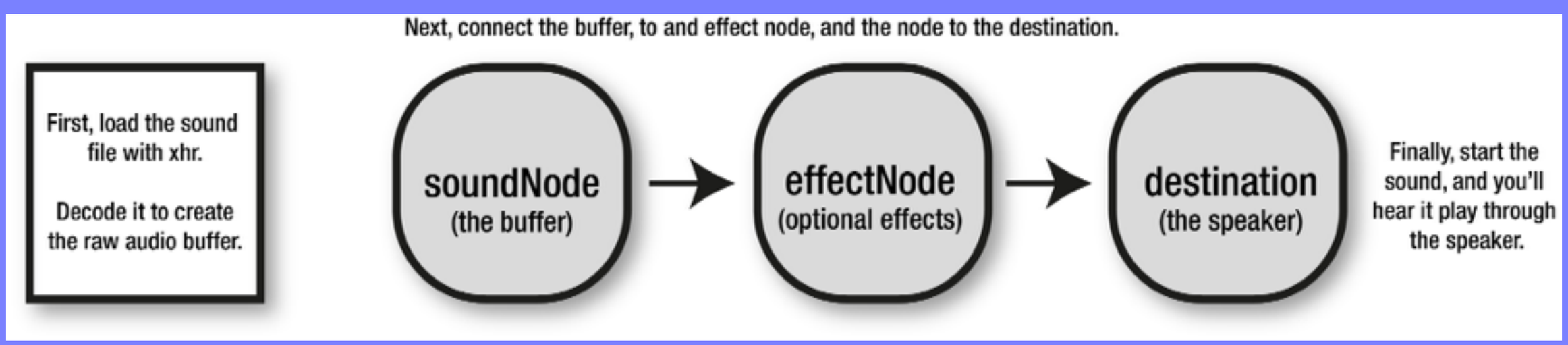
With the HTML Audio element…?

```
var sound = new Audio();
sound.addEventListener(
  "canplaythrough", playSoundHandler, false
);
sound.src = "sounds/music.wav";

function playSoundHandler(event) {
  sound.play();
  sound.volume = 0.5;
  sound.loop = true;
}
```

- Imprecise playback control
- Latencty
- No concurrency without dodgy hacks
- Quirky browser implementations

Web Audio API

Extremely powerful, low-level control over sounds.


Next, connect the buffer, to and effect node, and the node to the destination.
First, load the sound file with xhr.
Decode it to create the raw audio buffer.
soundNode
(the buffer)
effectNode
(optional effects)
destination
(the speaker)
Finally, start the sound, and you'll hear it play through the speaker.

“Web Audio API”

by Boris Smus

But!

You have to write about 20 lines of code just to play a single sound…

```
//1. Create an audio context
var actx = new AudioContext();

//2. Declare a variable to hold the sound we'll load
var soundBuffer;

//3. Load the sound
//a. Use an XMLHttpRequest object to load the sound
var xhr = new XMLHttpRequest();

//b. Set properties for the file we want to load.
//Use GET and set the path to the sound file.
//`true` means that the file will load
//asynchronously and will create an event when the
//file has finished loading
xhr.open("GET", "sounds/shoot.wav", true);

//c. Set the `responseType`, which is the file format
//we're expecting to load. Sound files should
//be loaded as  binary files, so the responseType
//needs to be `arraybuffer`
xhr.responseType = "arraybuffer";

//d. Load the sound into the program
xhr.send();

//e. Create a `loadHandler` that runs when
//the sound has been loaded
xhr.addEventListener("load", loadHandler, false);

function loadHandler(event) {
  //console.log("music loaded")
  //f. Decode the audio file and store it in a
  //the `music` sound variable. The `buffer` is the
  //raw audio data
  actx.decodeAudioData(
    xhr.response,
    function(buffer) {
      //g. Copy the audio file into the
      //`soundBuffer` variable
      soundBuffer = buffer;
    },
    //Optionally throw an error
    function(error) {
      throw new Error("Audio could not be decoded: " + error);
    }
  );
}
```

```
//f. Play a sound when a key is pressed
window.addEventListener("keydown", keydownHandler, false);

function keydownHandler(event) {
  switch (event.keyCode) {
    case 49:
      /*
      //4. Play the sound (without volume and pan control)
      //a. Create a new `soundNode` variable and tell
      //it to use the sound that we loaded as its audio source
      var soundNode = actx.createBufferSource();
      soundNode.buffer = soundBuffer;
      //b. Connect the sound to the desination
      soundNode.connect(actx.destination);
      //c. Finally, actually play the sound. Use the
      //`start` method to set the sound's start position time to
      //0 (the beginning of the sound)
      soundNode.start(0);
      */

      //4. Play the sound (with volume, pan and loop)
      var soundNode = actx.createBufferSource();
      soundNode.buffer = soundBuffer;
      //Create volume and pan nodes
      var volumeNode = actx.createGain();
      var panNode = actx.createPanner();
      //Connect the sound source to the pan node, the pan node to
      //volume node, and the volume node to the destination
      soundNode.connect(panNode);
      panNode.connect(volumeNode);
      volumeNode.connect(actx.destination);
      //Set the volume
      volumeNode.gain.value = 0.5;
      //Set left/right panning
      var x = -1,
          y = 0,
          z = 1 - Math.abs(x);
      panNode.setPosition(x, y, z);
      panNode.panningModel = "equalpower";
      //Optionally loop the sound
      //soundNode.loop = true;
      //Finaly, play the sound
      soundNode.start(0);

      break;
  }
}
```

And you have to do this every time you want to play the sound.

Abstract it

- Spend and afternoon writing your own high level API wrapper so that you can make easy-to-use sound sprites.
- Use someone else's wrapper: Howler, Theresa's Sound World, or WAD.

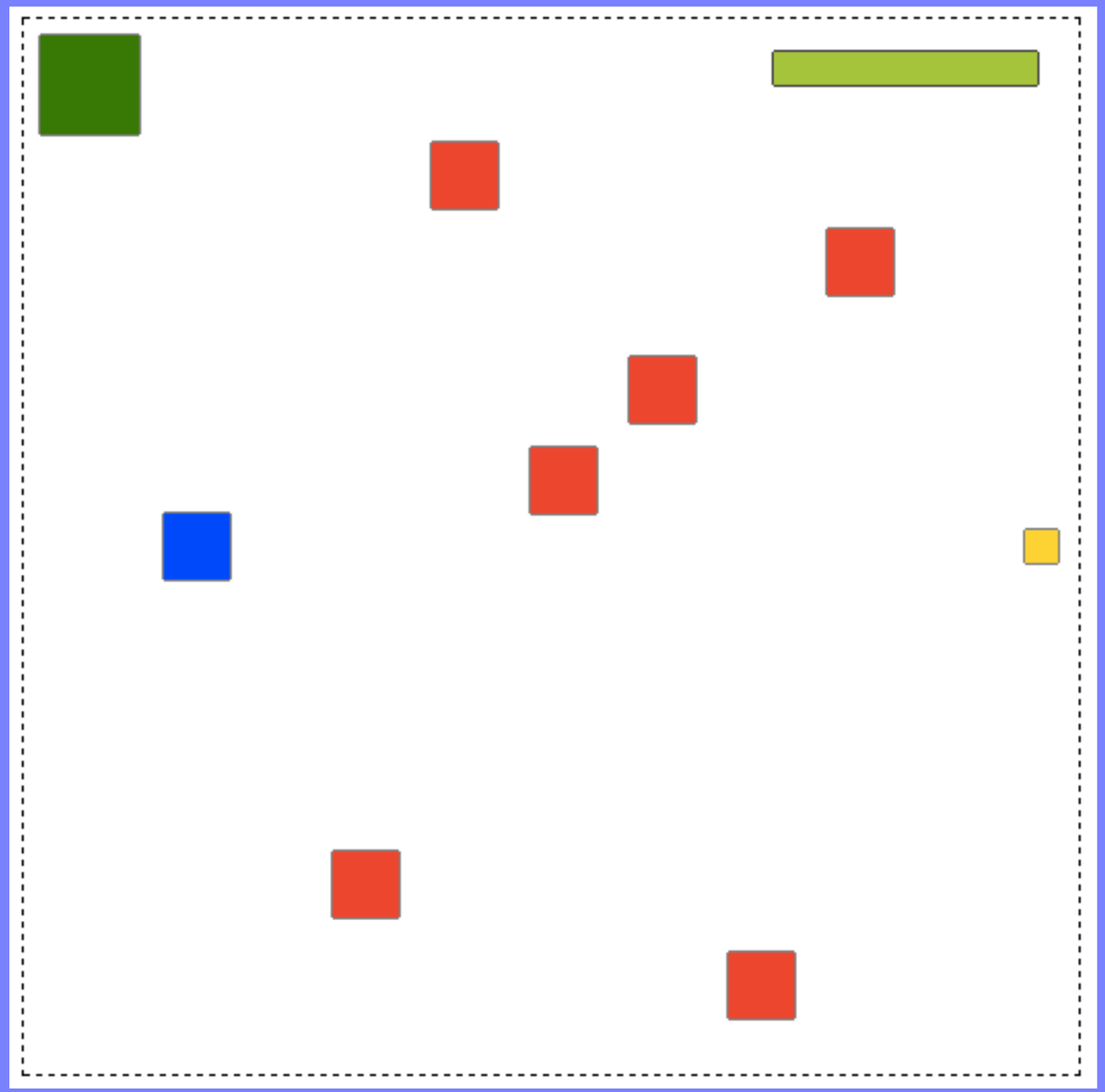
`makeSound`

A function that uses the Web Audio API to create sound sprites.

```
//Define the audio context
window.AudioContext = window.AudioContext || window.webkitAudioContext;
var actx = new AudioContext();

//Create the sound
var shoot = makeSound("sounds/shoot.wav", actx, soundLoadHandler);

//An optional callback that runs when the sound is loaded
function soundLoadHandler() {
  console.log("sound loaded");
}

//A keyboard event listener that plays the sound when
/the `1` key is pressed is pressed
window.addEventListener("keydown", function(event) {
  if(event.keyCode === 49) {
    //Play the sound
    shoot.play();
  }
}, false);
```

Now you can start making games!

Treasure Hunter



Treasure Hunter

Everything a complete game needs:

- Sprites
- An update/render loop
- Interactivity
- Collision
- Game states
- “juice” (sound)

Make the enemies

```
var numberOfEnemies = 6,
    spacing = 48,
    xOffset = 150,
    speed = 2,
    direction = 1,
    enemies = [];

//Make as many enemies as there are `numberOfEnemies`
for (var i = 0; i < numberOfEnemies; i++) {

  //Each enemy is a red rectangle
  var enemy = rectangle(32, 32, "red");

  //Space each enemy horizontally according to the `spacing` value.
  //`xOffset` determines the point from the left of the screen
  //at which the first enemy should be added.
  var x = spacing * i + xOffset;

  //Give the enemy a random y position
  var y = random(0, canvas.height - enemy.height);

  //Set the enemy's position
  enemy.x = x;
  enemy.y = y;

  //Set the enemy's vertical velocity. `direction` will be either `1` or
  //`-1`. `1` means the enemy will move down and `-1` means the enemy will
  //move up. Multiplying `direction` by `speed` determine's the enemy's
  //vertical direction
  enemy.vy = speed * direction;

  //Reverse the direction for the next enemy
  direction *= -1;

  //Push the enemy into the `enemies` array
  enemies.push(enemy);
}
```

Make the health bar

```
var outerBar = rectangle(128, 16, "black"),
    innerBar = rectangle(128, 16, "yellowGreen");

innerBar.x = outerBar.x = canvas.width - 148;
innerBar.y = outerBar.y = 16;
```

Set the game state

```
//Set the `state` function to `play`
var state = play;

//Start the game loop
gameLoop();

function gameLoop() {
  requestAnimationFrame(gameLoop, canvas);
  //Update the current game state
  state();
  //Render the sprites
  render();
}
```

```
function play() {
  //...The main game logic...
  //If the game ends:
  state = end;
}
```

```
function end() {
  //What should happen when the game ends
}
```

Collision between the player and enemies

```
//Set `playerHit` to `false` before checking for a collision
var playerHit = false;

//Loop through all the enemies
enemies.forEach(function(enemy) {

  //Move the enemy
  enemy.y += enemy.vy;

  //Check the enemy's screen boundaries
  checkScreenBoundaries(enemy, true);

  //Test for a collision. If any of the enemies are touching
  //the player, set `playerHit` to `true`
  if(hitTestRectangle(player, enemy)) {
   playerHit = true;
  }
});

//If the player is hit...
if(playerHit) {
  //Make the player semitransparent
  player.alpha = 0.5;
  //Reduce the width of the health bar's inner rectangle by 1 pixel
  innerBar.width -= 1;
} else {
  //Make the player fully opaque (non-transparent) if it hasn't been hit
  player.alpha = 1;
}
```

Collision between the player and the treasure

```
//Create the `chimes` sound
var actx = new AudioContext();
var chimes = makeSound("sounds/chimes.wav", actx);
```

```
//Create the treasure
var treasure = rectangle(16, 16, "gold");

//Position it next to the left edge of the canvas
treasure.x = canvas.width - treasure.width - 10;
treasure.y = canvas.height / 2 - treasure.height / 2;

//Create a `pickedUp` property on the treasure to help us Figure
//out whether or not the treasure has been picked up by the player
treasure.pickedUp = false;
```

```
//In the update loop: Check for a collision between the player and the treasure
if (hitTestRectangle(player, treasure)) {

  //If the treasure is touching the player, center it over the player
  treasure.x = player.x + 8;
  treasure.y = player.y + 8;
  if(!treasure.pickedUp) {
      //If the treasure hasn't already been picked up,
      //play the `chimes` sound
      chimes.play();
      treasure.pickedUp = true;
  };
}
```

Checking for the end of the game

```
//Does the player have enough health? If the width of the `innerBar`
//is less than zero, end the game and display "You lost!"
if (innerBar.width < 0) {
  state = end;
  message.content = "You lost!";
}

//If the player has brought the treasure to the exit,
//end the game and display "You won!"
if (hitTestRectangle(treasure, exit)) {
  state = end;
  message.content = "You won!";
}
```

```
function end() {
  //Make all the game sprites invisible and display
  //the game over message
  sprites.forEach(function(sprite) {
    sprite.visible = false;
  });

  //Display the game over message
  message.visible = true;
}
```

Everything is understandable and under your complete control.

Too much work!

Out of the bosom of the Air,
    Out of the cloud-folds of her garments shaken,
Over the woodlands brown and bare,
    Over the harvest-fields forsaken,
        Silent, and soft, and slow
        Descends the snow.

Even as our cloudy fancies take
    Suddenly shape in some divine expression,
Even as the troubled heart doth make
    In the white countenance confession,
        The troubled sky reveals
        The grief it feels.

This is the poem of the air,
    Slowly in silent syllables recorded;
This is the secret of despair,
    Long in its cloudy bosom hoarded,
        Now whispered and revealed
        To wood and field.

– Henry Longfellow (1807–1882)

No sky
no earth - but still
snowflakes fall.

- Kajiwara Hashin (1864-?)

“Haiku”

Abstract and Simplify

Figure out which code you’re using over and over.
Abstract it into reusable objects and functions.

Enhance the readability of your game code and reduce your mental overhead.

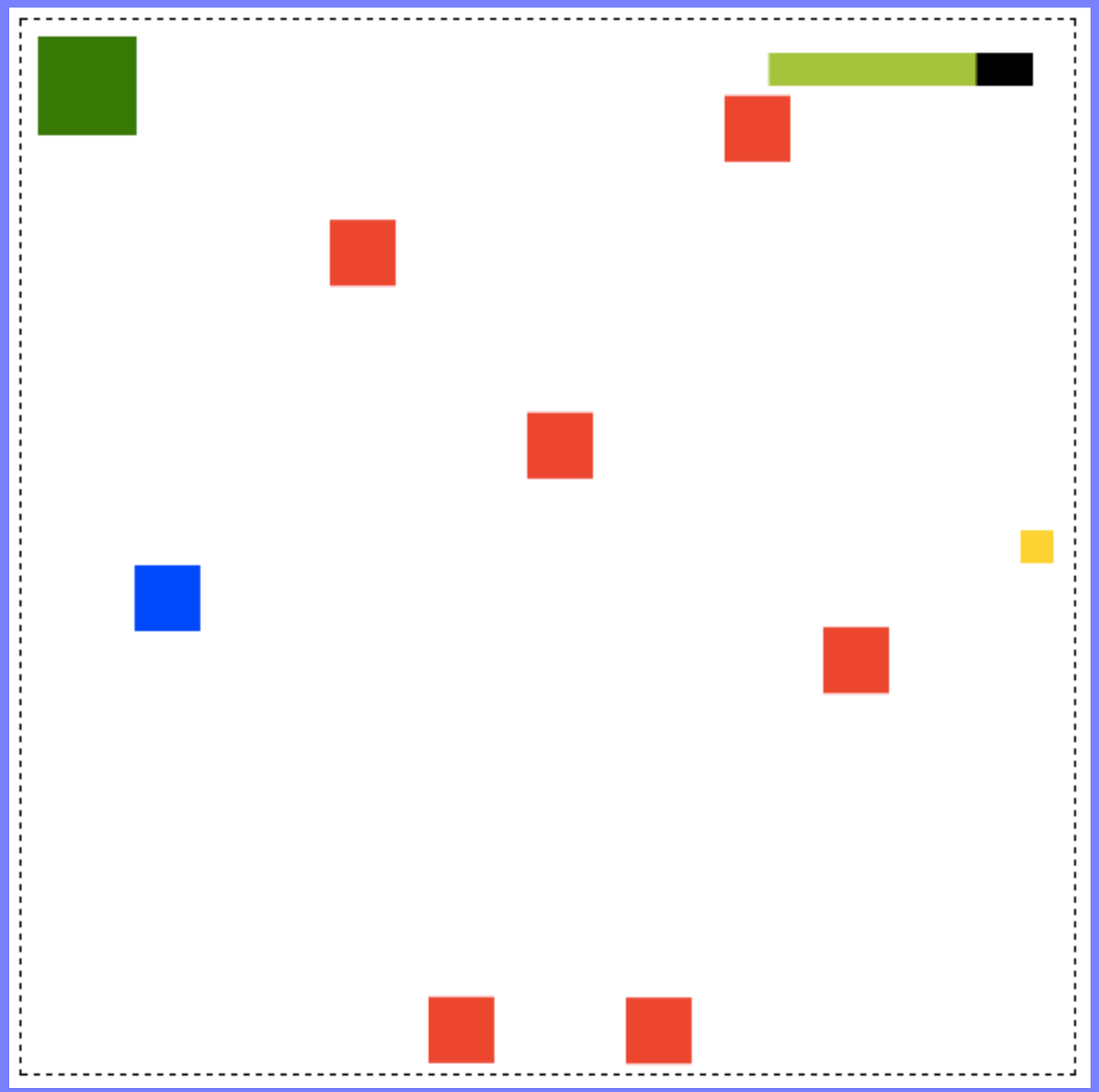
Apply to: Sprites, game loop, bounds checking, preloading, environment configuration, collision, interactivity (keyboard and pointer objects), game state management, special objects like sounds, special functions like building game worlds.

`Ga`

A minimalist game engine for learning

```
<!doctype html>
<meta charset="utf-8">
<title>`Ga` template</title>
<body>
<!-- Import the Ga game engine files -->
<script src="../ga/ga.js"></script>
<script src="../ga/plugins.js"></script>
<script>

//Create a new GA instance, and start it
var g = ga(
  512, 512, setup,
  [
    //List any assets you want to preload
    "sounds/chimes.wav"
  ]
);
g.start();

function setup() {
  //...Initialization tasks that run only once ...
  //Change the game state when setup tasks have finished:
  g.state = play;
}

function play() {
  //Any game logic that runs in a loop goes here
}

</script>
```

Treasure hunter enhanced



Groups

Group sprites together to make compound sprites

```
//Create the health bar
var outerBar = g.rectangle(128, 16, "black"),
    innerBar = g.rectangle(128, 16, "yellowGreen");

//Group the inner and outer bars
healthBar = g.group(outerBar, innerBar);

//Set the `innerBar` as a property of the `healthBar`
healthBar.inner = innerBar;

//Position the health bar sprite group
healthBar.x = g.canvas.width - 148;
healthBar.y = 16;
```

Use groups to create game scenes

```
//Create the `gameScene` group
gameScene = g.group();

//Create sprites and add them to the `gameScene`
player = g.rectangle(32, 32, “blue");
gameScene.addChild(player);
treasure = g.rectangle(16, 16, “gold");
gameScene.addChild(treasure);
healthBar = g.group(outerBar, innerBar);
gameScene.addChild(healthBar);

//Optionally group all sprites with at once like this:
gameScene.add(player, treasure, healthBar);

//Create text and add it to the `gameOverScene`
message = g.text("Game Over!", "64px Futura", "black", 20, 20);
gameOverScene = g.group(message);
gameOverScene.visible = false;
```

Hide or reveal game scenes as you need them

```
//...
//Check for the end of the game

//Does the player have enough health? If the width of the `innerBar`
//is less than zero, end the game and display "You lost!"
if (healthBar.inner.width < 0) {
  g.state = end;
  message.content = "You lost!";
}

//If the player has brought the treasure to the exit,
//end the game and display "You won!"
if (g.hitTestRectangle(treasure, exit)) {
  g.state = end;
  message.content = "You won!";
}
}

function end() {
  gameScene.visible = false;
  gameOverScene.visible = true;
}
```

Convenience methods

`move` and `contain`

```
var playerHit = false;
enemies.forEach(function(enemy) {

  //Move the enemy
  g.move(enemy);

  //Check the enemy's screen boundaries
  var enemyHitsEdges = g.contain(enemy, g.stage.localBounds);

  //If the enemy hits the top or bottom of the stage, reverse
  //its direction
  if (enemyHitsEdges === "top" || enemyHitsEdges === "bottom") {
    enemy.vy *= -1;
  }

  //Test for a collision. If any of the enemies are touching
  //the player, set `playerHit` to `true`
  if(g.hitTestRectangle(player, enemy)) {
   playerHit = true;
  }
});
```

Keyboard controller

```
//arguments: thingToControl, speed, up, right, down, left

g.fourKeyController(player, 5, 38, 39, 40, 37);
```

Adding images

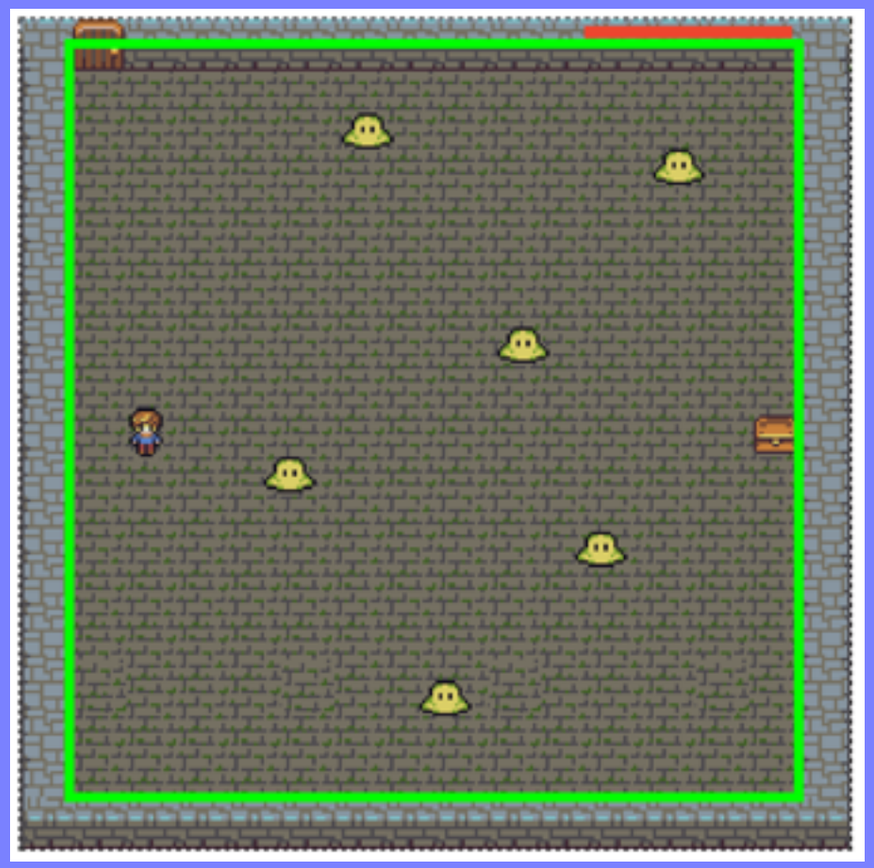
Preload the images you want to use

```
var g = ga(
  512, 512, setup,
  [
    "images/explorer.png",
    "images/dungeon.png",
    "images/blob.png",
    "images/treasure.png",
    "images/door.png",
    "sounds/chimes.wav"
  ]
);
g.start();
```

Use the `sprite` method to create an image sprite

```
//The dungeon background
dungeon = g.sprite("images/dungeon.png");
gameScene.addChild(dungeon);

//The exit door
exit = g.sprite("images/door.png");
exit.x = 32;
gameScene.addChild(exit);

//The player sprite
player = g.sprite("images/explorer.png");
player.x = 68;
player.y = g.canvas.height / 2 - player.height / 2;
gameScene.addChild(player);

//Create the treasure
treasure = g.sprite("images/treasure.png");

//Position it next to the left edge of the canvas
treasure.x = g.canvas.width - treasure.width - 32;
treasure.y = g.canvas.height / 2 - treasure.height / 2;
```

Fine tune the containment area



```
g.contain(
  player,
  {
    x: 32, y: 16,
    width: g.canvas.width - 32,
    height: g.canvas.height - 32
  }
);
```

You’re done!

Where now?

- Decide what kind of games you want to make.
- Explore tools that will help you make that game.
- Use an existing game engine or build your own out of components.
- Start designing a game for next year’s js13K competition.

Useful tools

- **Pixi**: fast and easy 2D rendering engine
- **Babylon**: 3D rendering engine for games
- **Howler**: WebAudio sound library
- **Photon**: Particle effects
- **Tween**.js: Tween animation library
- **Better-than-average game engines**: Phaser, Impact, Goo Engine, Unity, Play Canvas, Monogame

Useful resources

- www.html5gamedevs.com
- opengameart.org
- www.universalsoundfx.com
- **Game development links**: github.com/ellisonleao/magictools

Rich Davey's tips

- Things you're told are not possible today might be by the end of the project.
- Be prepared to un-learn what you know every 6 months.
- Googling a problem? Check the dates of the answers. Disregard anything > 1 year old.

Bye for now!

- www.kittykatattack.com
- github.com/kittykatattack/ga
- @rexvanderspuy
- dandylion13@gmail.com
- … and lots of books.