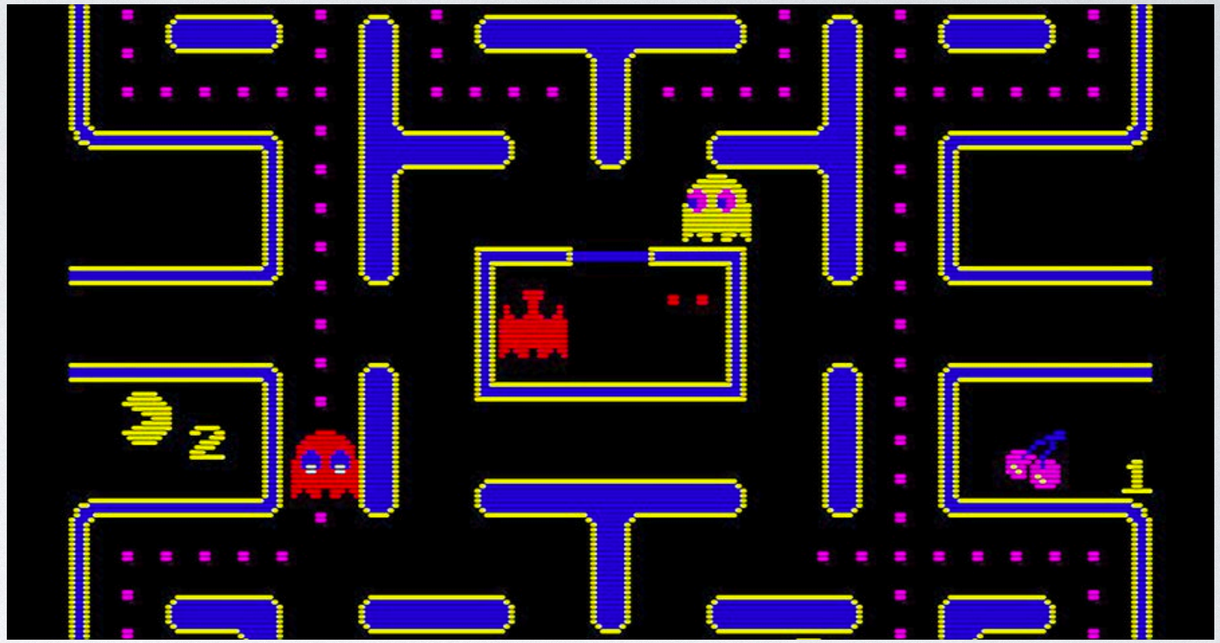
2D GAME PROGRAMMING FOR THE WEB

Nicholas Zinn, September 2020

TABLE OF CONTENTS



- Introduction to 2D Games
- What is a Scene, Sprite, Sprite Sheet, Tile?
- The Game Loop
- 2D Graphics Canvas
- 2D Math
- User Input
- Collisions
- Circles Dev Environment

INTRODUCTION TO 2D GAMES

Common Features

- Sprite Sheets
- Animation
- Sound Effects
- Input, gamepad, multi-touch
- Parallax Scrolling
- Collision Detection
- Mobile and Desktop Support
- Image filters and effects
- Physics System
- Particle System

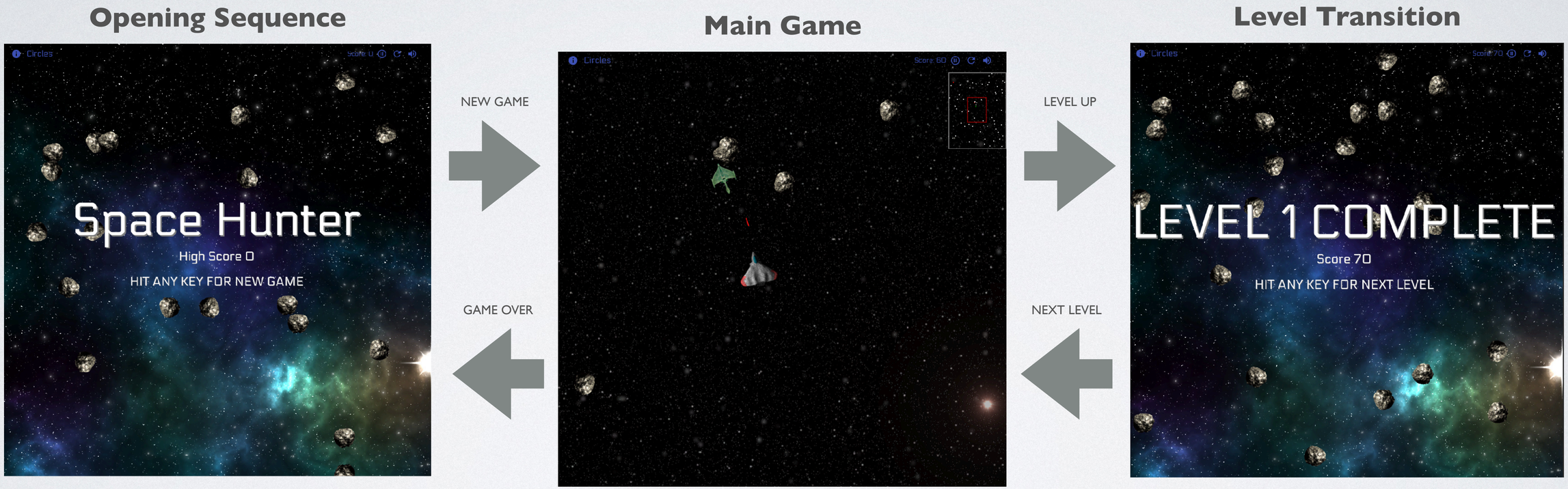
Popular Web Graphics and Game Libraries

PixiJS

unity

To teach concepts, we rolled our own, Circles:
https://nickzinn.github.io/circles/

WHAT IS A SCENE?



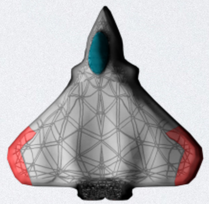
Scenes allows you to storyboard a game.

WHAT IS A SPRITE?

2D Game Visual Object

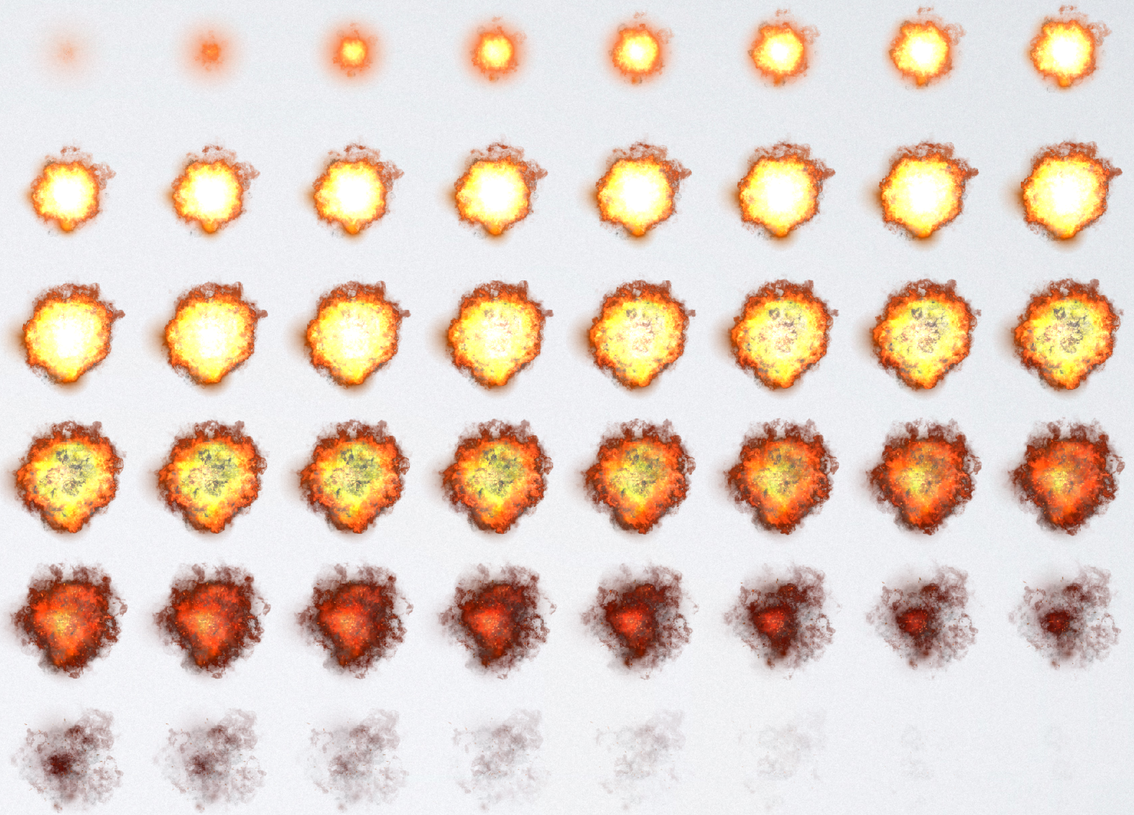
Properties
- Position (x,y)
- Size (width, height)
- Vector (speed, angle in radians)

Methods
- paint()
- updateModel()



SPRITE SHEET

- One image file divided into smaller images.
- Used for either animation scenes for one sprite or many individual sprites.
- For animation, need to sequence the flipping of scenes to the time clock.



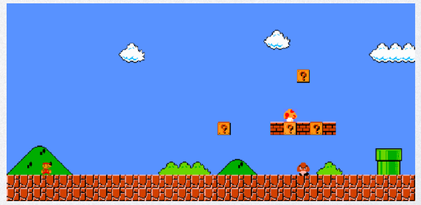
WHAT IS A TILE?

- Tiles are regular shaped images that used to build the game world or level map.

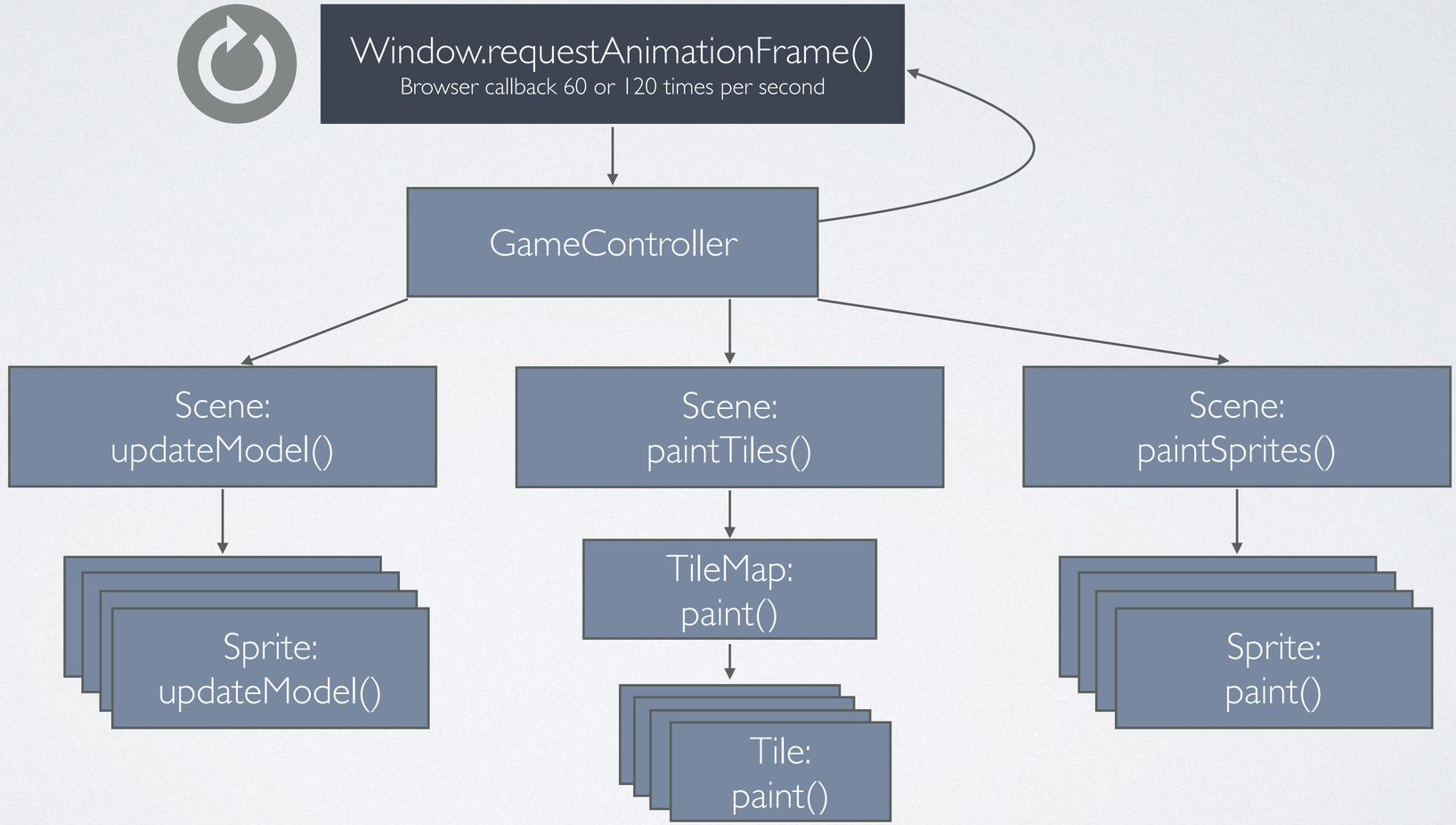
SpriteSheet of Tiles

Level Map

Painted Screen



THE GAME LOOP



2D GRAPHICS CANVAS

- HTML **<canvas>** element allows 2D graphics and 3D graphics(WebGL) in the browser.

Code

HMTL:

```
<canvas id="my-canvas" width="300" height="300"></canvas>
<div style="display:none;">
  <img id="source" src="sampleImage.jpg">
</div>
```

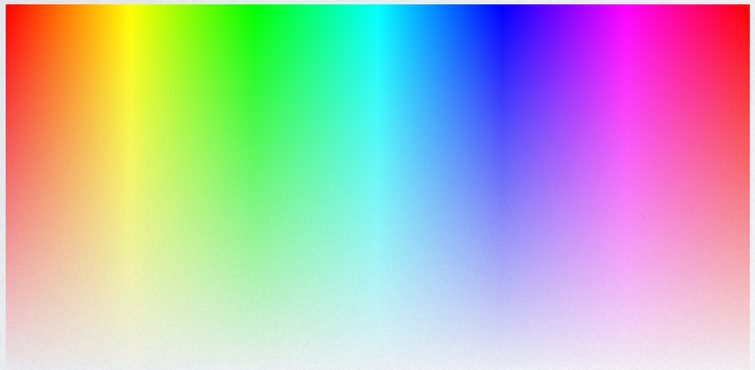
Javascript:

```
const canvas = document.getElementById('my-canvas');
const ctx = canvas.getContext('2d');
const image = document.getElementById('source');
//Clear Canvas
ctx.fillStyle = 'red';
ctx.fillRect(0, 0, canvas.width, canvas.height);
//Draw Image Scaled to half size in center.
const w = image.width/2, h = image.height/2;
const dX = canvas.width/2 - w/2, dY = canvas.height/2 - h/2;
ctx.drawImage(image,0, 0, image.width, image.height, dX, dY, w, h);
```

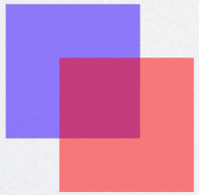
2D GRAPHICS CANVAS

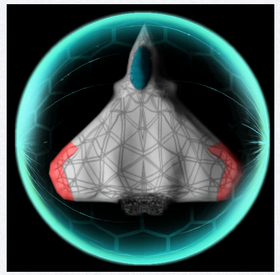
Transparency

- pixel = (r, g, b, a)



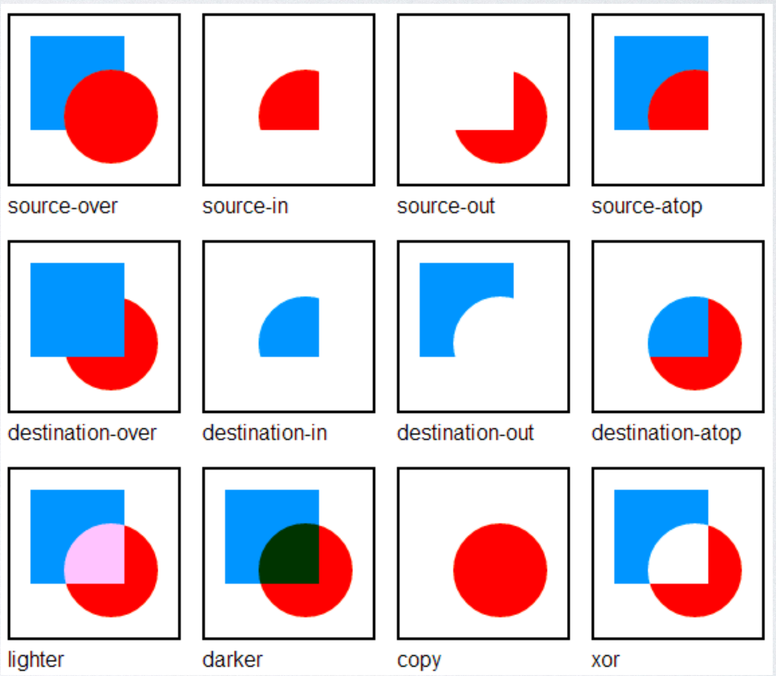
- Global Alpha





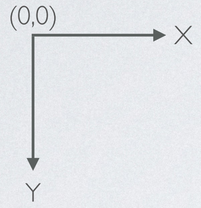
Compositing

globalCompositeOperation



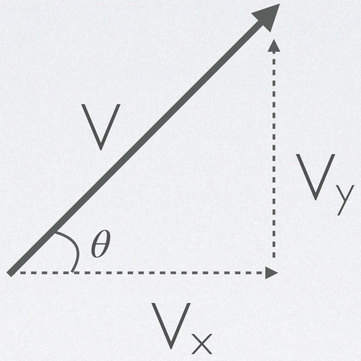
2D GRAPHICS MATH

Canvas Rendering Context 2D



All 2D computer graphics are done on an upside-down cartesian coordinate system.

Vector to X, Y



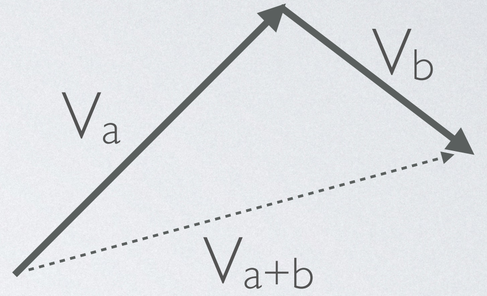
$$V = hypot(x, y)$$

$$\theta = atan2(y, x)$$

$$V_x = V \cdot \cos\theta$$

$$V_y = V \cdot \sin\theta$$

Vector Addition



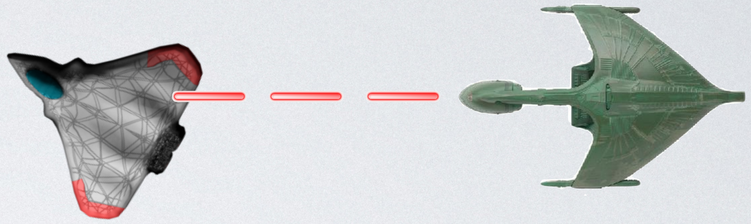
Convert each vector to x,y and add together.

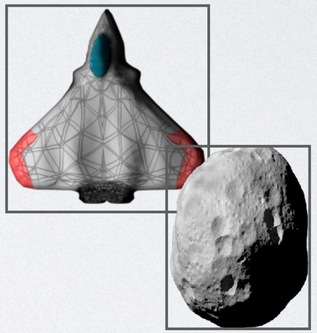
USER INPUT

- Input is event based (can't query state, so need to keep track of all input state).
  - Keyboard: keydown, keyup. Need to keep track of key presses to handle key being held down.
  - Mouse: mousedown, mouseup, mousemove. Need to keep track of mouse button to handle mouse being held down.
  - Touch: touchmove, touchend, touchcancel, touchstart. Need to keep track of touch events.
- Event Handling: Scene.updateModel() triggers event handlers, once per game loop.

COLLISIONS

- Detecting sprite collisions is done in the game model update step.
- Must be done efficiently. Naive implementation result in game lag and low frame rate.
- Easiest computation is to assume objects are rectangular and only perform one-way check.



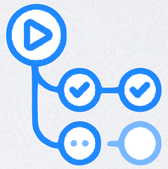


CIRCLES DEV ENVIRONMENT

TypeScript

GitHub

Visual Studio Code



GitHub Actions

Gitpod

GitHub Pages

APPENDIX: REFERENCES

- https://github.com/nickzinn/circles
- MDN Techniques for game development
- Canvas API
- Window.requestAnimationFrame()