Q:

What is one problem you had (or are having) during the programming assignment?

One quiz question was wrong

```
let counter = 4;
for( ; counter < 15; counter = counter + 1) {
   if(0 == counter % 4) {
      break
   }
}
console.log(counter)
```

This was *supposed* to be a 3

JavaScript Notes: Numbers

- Unlike C and Java, JavaScript doesn't have integers.
  - All numbers are floating point
    - `0.1 + 0.2 == 0.30000000000000004`
  - Though integer values are respected unless they're added to an non-integer
    - `1 + 2 == 3`
  - And be careful with strings, because with strings + means concatenate:
    - `1 + "2" == "12"`
  - Use parseInt() instead: `parseInt("2", 10) == 2`
  - For advanced math functions, you can use the `Math` object:
    - `Math.Sin();`
    - `Math.PI`
    - `Et cetera`

What makes videogames different is

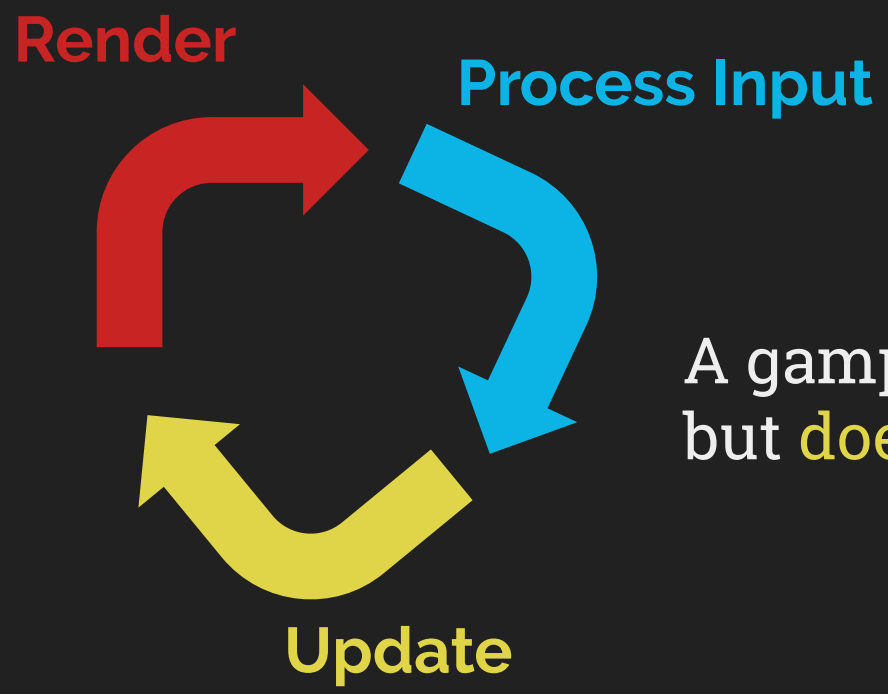
Loops and States

Almost every game has one, no two are exactly alike, and relatively few programs outside of games use them.

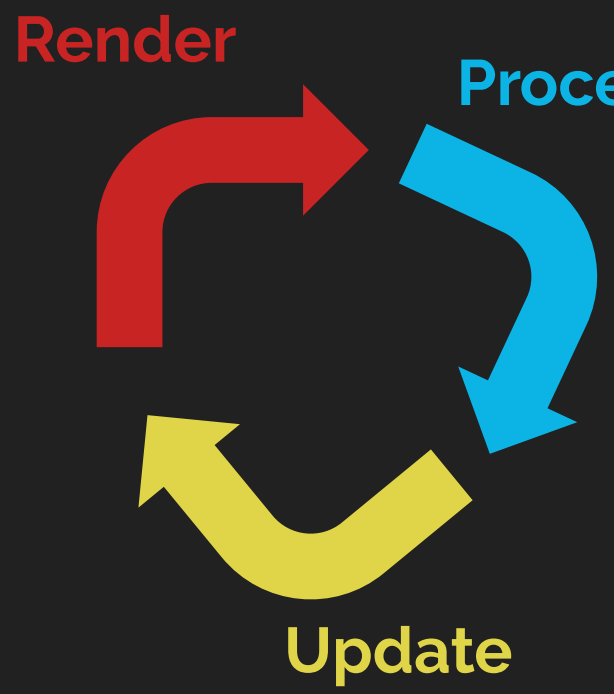
Game Programming Patterns, p. 304

Why do we need a game loop?

Games keep updating even when the user isn't providing input



A gamp loop processes user input but doesn't wait for it.




```
while (true) {
    processInput();

    update();

    render();

}
```

← repeat forever

← handle user input since the last call

← advance the game simulation one step

← draw the game so the player can see it

Q: With this basic loop, how fast will the game state advance?

```
while (true) {
    processInput();

    update();

    render();

}
```

Or in other words, what is the game's frame rate?

A: It depends on how much work each step is doing...

```
while (true) {
    processInput();

    update();

    render();

}
```

...and on the thing doing the work

A: It depends on how much work each step is doing...

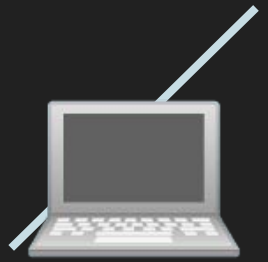
```
while (true) {
    processInput();

    update();

    render();

}
```

...and on the thing doing the work

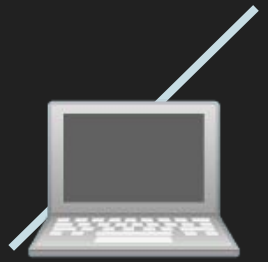


How much work needs to be done each frame?

Physics, on-screen objects, collisions, simulation, etc.

What is the speed of the underlying platform?

CPU speed, memory resources, screen refresh rate, operating system preemption, etc.



How much work needs to be done each frame?

Physics, on-screen objects, collisions, simulation, etc.

For some videogames, this is a constant

For example, games that run on consoles have predictable resource constraints.

How much work needs to be done each frame?

Physics, on-screen objects, collisions, simulation, etc.

On the web, this changes

Not only will different devices have different resources, but the amount of processing time available for the game can change!

This basic loop doesn't handle time

```
while (true) {
    processInput();

    update();

    render();

}
```

This is a big problem when emulating older games that assumed a fixed amount of time per frame!

Slower hardware will run slower and faster hardware will run faster

If you're building your game on top of an OS or platform that has a graphic UI and an event loop built in, then you have two application loops in play. They'll need to play nice together.

Game Programming Patterns, p. 315

If we're using just JavaScript and the browser...

...we can update our loop with a callback function.

The **`window.requestAnimationFrame()`** method tells the browser that you wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint. The method takes as an argument a callback to be invoked before the repaint.

https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

If we're using just JavaScript and the browser...

...we can update our loop with a callback function.

1. Declare a function called "mainLoop"

3. mainLoop() gets called by the window

```
1 function mainLoop() {
2     console.log("calling mainLoop");
3     update();
4     draw();
5     window.requestAnimationFrame(mainLoop);
6 }
7
8 window.requestAnimationFrame(mainLoop);
```

⇦ **Why is update() before draw()?**

4. at the end of mainLoop(), add mainLoop() as a callback again!

2. Add mainLoop() as a callback once

Callback Function

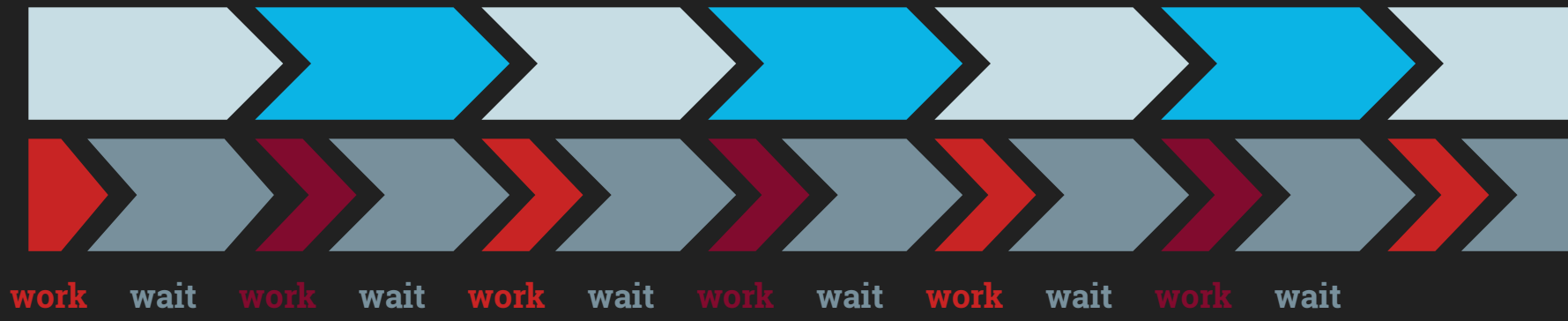
A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

This is a very common pattern in web development, because it is a good way to create an API for an event-driven program.

They let programs call code that hasn't been written yet.

A fast loop needs to wait until the next update is ready



A slow loop also needs track the time elapsed and run updates until it can 'catch up'

Which is a problem if it gets too far behind...

There are solutions for this you can explore in depth…

Home

A Detailed Explanation of JavaScript Game Loops and Timing

Sun, Jan 18, 2015 - 1:31am -- Isaac Sukin

javascript games programming Tip/Tutorial

The **main loop** is a core part of any application in which state changes over time. In games, the main loop is often called the *game loop*, and it is typically responsible for computing physics and AI as well as drawing the result on the screen. Unfortunately, the vast majority of main loops found online - especially those in JavaScript - are written incorrectly due to timing issues. I should know; I've written my fair share of bad ones. This post aims to show you why many main loops need to be fixed, and how to write a main loop correctly.

If you'd rather skip the explanation and just **get the code to do it right**, you can use my **open-source** MainLoop.js **project**.

Table of contents:

https://isaacsukin.com/news/2015/01/detailed-explanation-javascript-game-loops-and-timing

...but Phaser takes care of the game loop for us.

```
34792    /**
34793    * The core game loop.
34794    *
34795    * @method Phaser.Game#update
34796    * @protected
34797    * @param {number} time - The current time as provided by RequestAnimationFrame.
34798    */
34799    update: function (time) {
34800
34801        this.time.update(time);
34802
34803        if (this._kickstart)
34804        {
34805            this.updateLogic(this.time.desiredFpsMult);
34806
34807            // call the game render update exactly once every frame
34808            this.updateRender(this.time.slowMotion * this.time.desiredFps);
34809
34810            this._kickstart = false;
34811
34812            return;
34813        }
34814
34815        // if the logic time is spiraling upwards, skip a frame entirely
34816        if (this._spiraling > 1 && !this.forceSingleUpdate)
34817        {
34818            // cause an event to warn the program that this CPU can't keep up with the
                 current desiredFps rate
34819            if (this.time.time > this._nextFpsNotification)
34820            {
34821                // only permit one fps notification per 10 seconds
34822                this._nextFpsNotification = this.time.time + 10000;
34823
```

Phaser's logic update sequence:

```
this.debug.preUpdate();
this.world.camera.preUpdate();
this.physics.preUpdate();
this.state.preUpdate(timeStep);
this.plugins.preUpdate(timeStep);
this.stage.preUpdate();

this.state.update();
this.stage.update();
this.tweens.update(timeStep);
this.sound.update();
this.input.update();
this.physics.update();
this.particles.update();
this.plugins.update();

this.stage.postUpdate();
this.plugins.postUpdate();
```

Cleanup and preparation for updating

Our code is run here

Rest of the logic updating

Post-update cleanup

Break

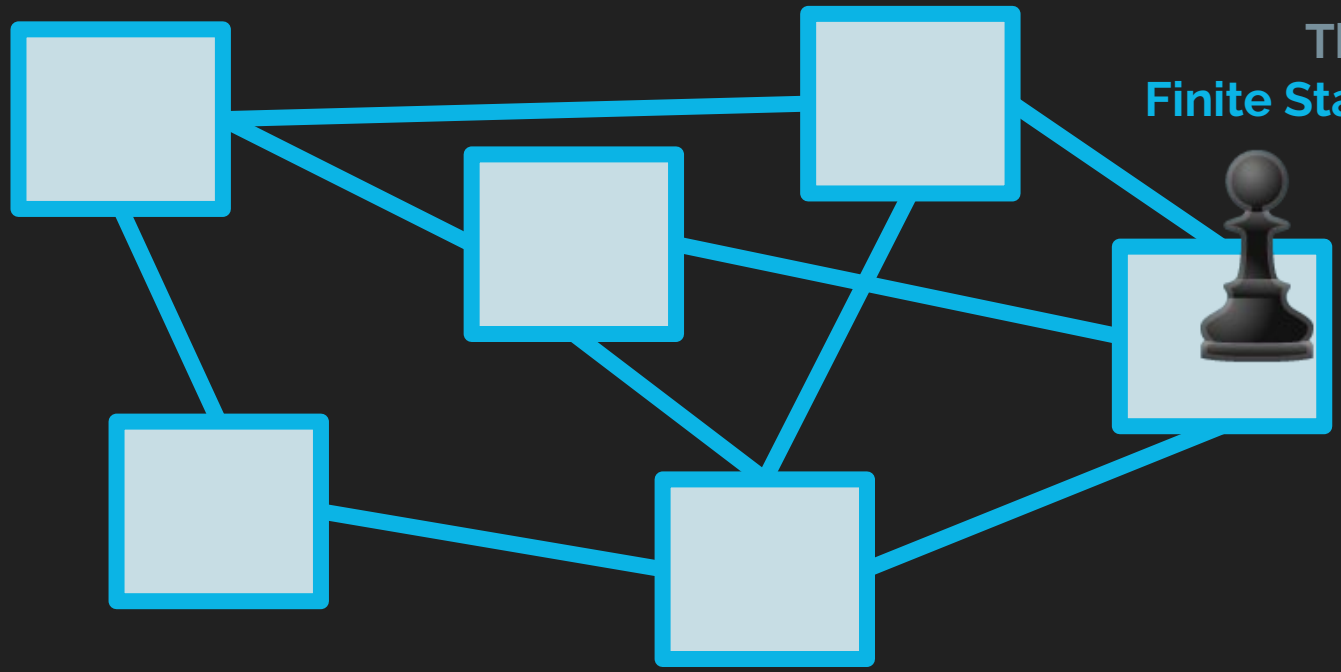
Managing

States

States bundle up a series of methods that help get the program into and potentially out of a section of gameplay.

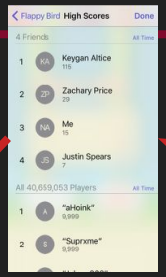
An Introduction to HTML5 Game Development with Phaser.js, p.58

You can think of states like spaces on a game board...



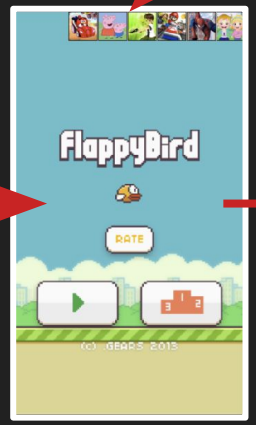
This is also called a
Finite State Machine (FSM)

...where your game piece can only be in one space at a time

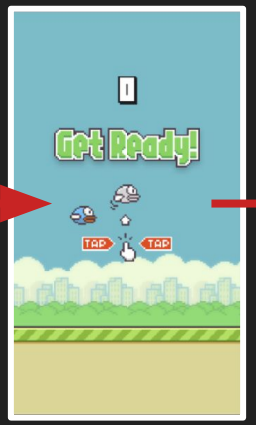


High Score
(OS)

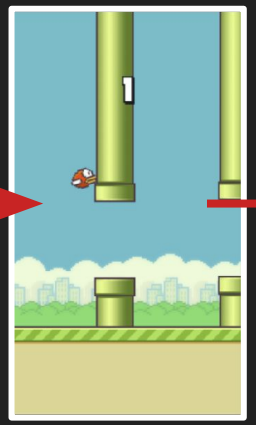
Boot



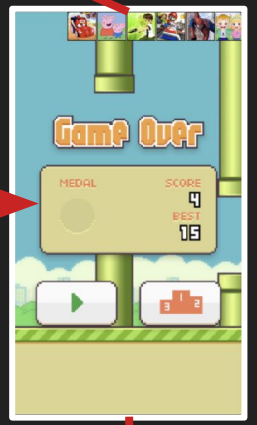
Menu



Pre-Game



Game



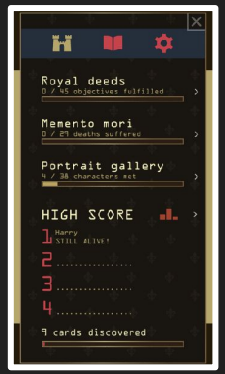
Game Over

Credits

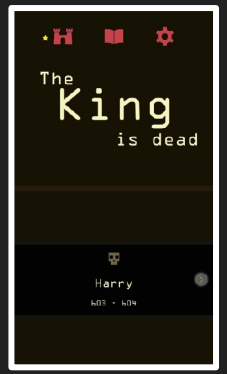
Title

Spawn

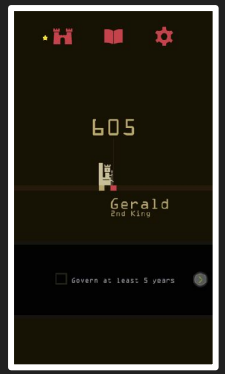
Play



Modal Menu



Game Over



Legacy

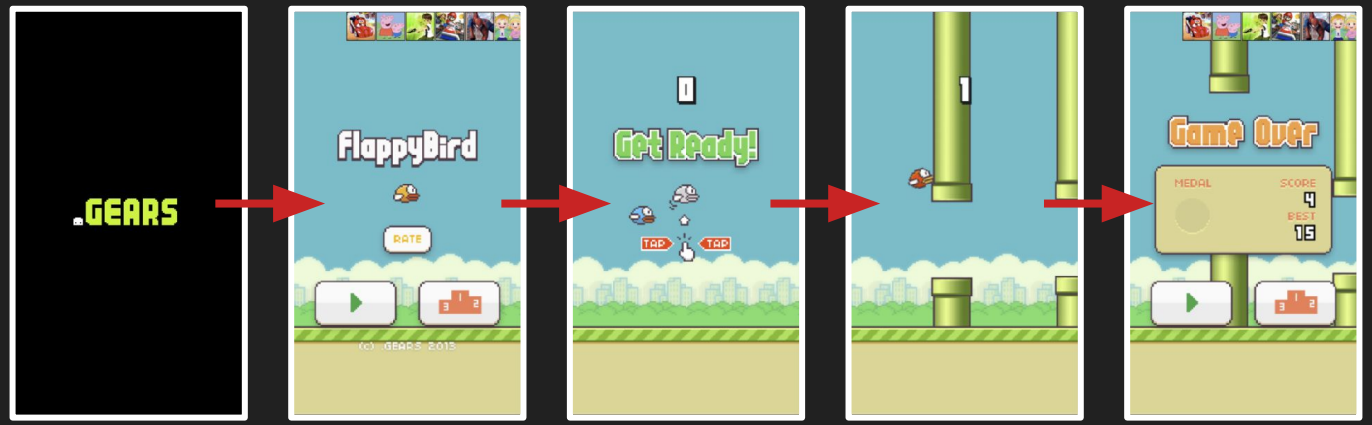
```
1
2  // the simplest Phaser game object instance
3  var game = new Phaser.Game(800, 600, Phaser.AUTO, '', { preload: preload, create: create });
4
```

```
1
2  // passing a single object instead of arguments
3  var config = {
4      width: 800,
5      height: 600,
6      renderer: Phaser.AUTO,
7      antialias: true,
8      multiTexture: true,
9      state: {
10         preload: preload,
11         create: create,
12         update: update
13     }
14 }
15
16 var game = new Phaser.Game(config);
```

An alternate way to define Phaser's game object

```
var exampleState = {
    init: function() {
        // do any setup necessary before the state begins to run
    },
    preload: function() {
        // preload any assets necessary for the game
    },
    create: function() {
        // setup the state with game objects
    },
    update: function() {
        // game code
    },
    shutdown: function() {
        // any cleanup necessary before the state ends
    }
}
```

Phaser states have other optional functions you can use



Boot	Menu	Pre-Game	Game	Game Over
init preload create	preload create update	init preload create update	init preload create update shutdown	preload create update shutdown

```
5   // define game
6   var game = new Phaser.Game(800, 600, Phaser.AUTO);
7
8   // define MainMenu state and methods
9   var MainMenu = function(game) {};
10  MainMenu.prototype = {
11      preload: function() {
12          console.log('MainMenu: preload');
13      },
14      create: function() {
15          console.log('MainMenu: create');
16      },
17      update: function() {
18          // main menu logic
19      }
20  }
21
22  // define GamePlay state and methods
23  var GamePlay = function(game) {};
24  GamePlay.prototype = {
25      preload: function() {
26          console.log('GamePlay: preload');
27      },
28      create: function() {
29          console.log('GamePlay: create');
30      },
31      update: function() {
32          // game logic
33      }
34  }
35
36  // define GameOver state and methods
37  var GameOver = function(game) {};
38  GameOver.prototype = {
39      preload: function() {
40          console.log('GameOver: preload');
41      },
42      create: function() {
43          console.log('GameOver: create');
44      },
45      update: function() {
46          // GameOver logic
47      }
48  }
49
50  // add states to StateManager and start MainMenu
51  game.state.add('MainMenu', MainMenu);
52  game.state.add('GamePlay', GamePlay);
53  game.state.add('GameOver', GameOver);
54  game.state.start('MainMenu');
55
```

Define Phaser game object

Main Menu

Game Play

Game Over

Add states to StateManager

Problem

Currently, there is no way to move from state to state

See states01.js

```
5   // define game
6   var game = new Phaser.Game(800, 600, Phaser.AUTO);
7
8   // define MainMenu state and methods
9   var MainMenu = function(game) {};
10  MainMenu.prototype = {
11      preload: function() {
12          console.log('MainMenu: preload');
13      },
14      create: function() {
15          console.log('MainMenu: create');
16          game.stage.backgroundColor = "#facade";
17      },
18      update: function() {
19          // main menu logic
20          if(game.input.keyboard.isDown(Phaser.Keyboard.SPACEBAR)) {
21              game.state.start('GamePlay');
22          }
23      }
24  }
25
26  // define GamePlay state and methods
27  var GamePlay = function(game) {};
28  GamePlay.prototype = {
29      preload: function() {
30          console.log('GamePlay: preload');
31      },
32      create: function() {
33          console.log('GamePlay: create');
34          game.stage.backgroundColor = "#ccddaa";
35      },
36      update: function() {
37          // GamePlay logic
38          if(game.input.keyboard.isDown(Phaser.Keyboard.SPACEBAR)) {
39              game.state.start('GameOver');
40          }
41      }
42  }
43
44  // define GameOver state and methods
45  var GameOver = function(game) {};
46  GameOver.prototype = {
47      preload: function() {
48          console.log('GameOver: preload');
49      },
50      create: function() {
51          console.log('GameOver: create');
52          game.stage.backgroundColor = "#bb11ee";
53      },
54      update: function() {
55          // GameOver logic
56          if(game.input.keyboard.isDown(Phaser.Keyboard.SPACEBAR)) {
57              game.state.start('MainMenu');
58          }
59      }
60  }
61
62  // add states to StateManager and start MainMenu
63  game.state.add('MainMenu', MainMenu);
64  game.state.add('GamePlay', GamePlay);
65  game.state.add('GameOver', GameOver);
66  game.state.start('MainMenu');
```

Solution

Add some simple input logic.

See states02.js

```
// define MainMenu state and methods
var MainMenu = function(game) {};
MainMenu.prototype = {
    init: function() {
        this.level = 1;
    },
    preload: function() {
        console.log('MainMenu: preload');
    },
    create: function() {
        console.log('MainMenu: create');
        game.stage.backgroundColor = "#facade";
        console.log('level: ' + this.level);
    },
    update: function() {
        // main menu logic
        if(game.input.keyboard.isDown(Phaser.Keyboard.SPACEBAR)) {
            // pass this.level to next state
            // .start(key, clearWorld, clearCache, parameter)
            game.state.start('GamePlay', true, false, this.level);
        }
    }
}
```

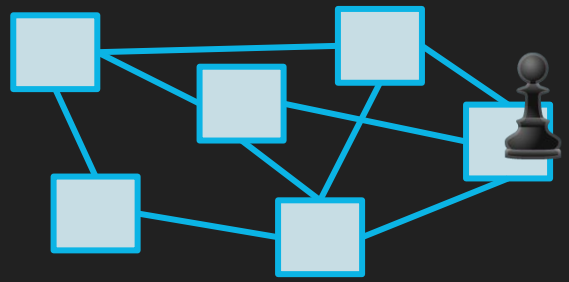
Passing Data

We can also use Phaser's State Manager to pass data between states

This helps us keep things from cluttering up the global scope

https://photonstorm.github.io/phaser-ce/Phaser.StateManager.html#start

See states03.js



Branching

With states and variables, we can make branching decisions.

https://photonstorm.github.io/phaser-ce/Phaser.StateManager.html#start

See states03.js

Resources for Phaser and JavaScript

README.md

Phaser CE (Community Edition)

Phaser CE is a fast, free, and fun open source HTML5 game framework. It uses a custom build of Pixi.js for WebGL and Canvas rendering, and supports desktop and mobile web browsers. Games can be compiled to iOS, Android and native desktop apps via 3rd party tools. You can use JavaScript or TypeScript for development.

Phaser v2 was built and maintained by Photon Storm and turned over to the community (as Phaser CE) in November 2016. Phaser v3 is in active development.

The current Phaser CE release is 2.13.2.

- **Visit:** The Phaser website and follow on Twitter (#phaserjs)
- **Learn:** API Docs, Support Forum and StackOverflow
- **Code:** 700+ Examples (source), new Phaser CE examples
- **Read:** Weekly Phaser World Newsletter
- **Chat:** Slack and Discord
- **Extend:** Phaser plugins - Shop, GitHub, NPM
- **Be awesome:** Support the future of Phaser

Grab the source and join in the fun!

Contents

Made With Phaser

https://github.com/photonstorm/phaser-ce

README.md

Phaser 2 Examples

Looking for Phaser 3 Examples? They are in their own repo.

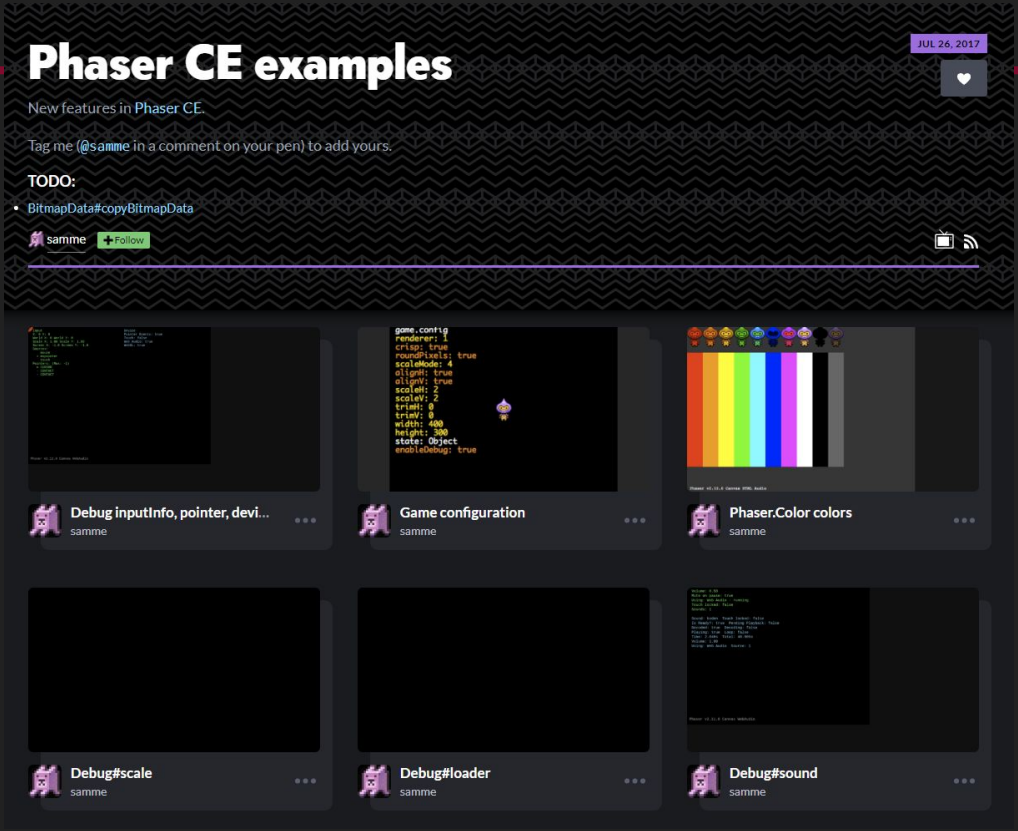
Phaser v2 is a fast, free and fun open source HTML5 game framework. It uses Pixi.js for WebGL and Canvas rendering across desktop and mobile web browsers. Games can be compiled to iOS and Android apps via 3rd party tools.

Along with the fantastic open source community Phaser is actively developed and maintained by Photon Storm Limited. As a result of rapid support and a developer friendly API Phaser is currently one of the most starred game frameworks on Github.

Thousands of developers worldwide use it. From indies and multi-national digital agencies to schools and Universities. Each creating their own incredible games. Grab the source and join in the fun!

Visit: The Phaser website and follow on Twitter (#phaserjs)
Learn: API Docs, Support Forum and StackOverflow
Code: 700+ Examples (source available in this repo)
Read: Weekly Phaser World Newsletter
Chat: Slack and IRC
Extend: With Phaser Plugins
Be awesome: Support the future of Phaser

https://github.com/photonstorm/phaser-examples



https://codepen.io/collection/AMbZqY/

https://mozdevs.github.io/html5-games-workshop/en/guides/platformer/start-here/

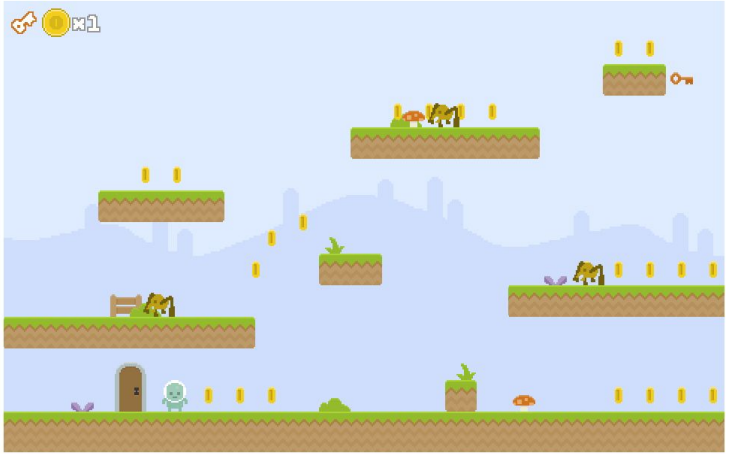
moz://a

HTML5 Games Workshop

Start here

Guide written by Belén "Benko" Albeza" (@ladybenko).

We are going to create a classic **one-screen platformer** game! It will feature a main character, who can run and jump to platforms. There will also be enemies that this character will have to avoid –or kill! The goal of the game is to fetch the key and open the door that leads to the next level.



You can play the game here.

We will be implementing the following game development concepts:

https://mozdevs.github.io/html5-games-workshop/en/guides/platformer/start-here/

GAMEDEV.JS WEEKLY

Subscribe to the weekly, free newsletter all about HTML5 Game Development.

Sent every Friday, managed by Andrzej Mazur from Enclave Games, creator of the js13kGames competition.

email address

SUBSCRIBE

Join over 6500 gamedev subscribers. No spam. Open for related sponsorship opportunities and job listings.

If you have anything you want to share with the HTML5 Game Development community - please let me know.

WEEKLY NEWSLETTER
ABOUT HTML5 GAME DEVELOPMENT

See the archive list below for the ten recent issues.

Help me keep this free newsletter going by sending a donation, thank you!

GAMEDEV.JS WEEKLY NEWSLETTER ARCHIVE

You can view the recent 10 entries:

28/06/2019 - Issue #286: Floppy, SuperwebApp, and DroidScript

21/06/2019 - Issue #285: Azerion, Noa, and Phaser 3.18

14/06/2019 - Issue #284: W3C workshop, E3 news, and multiplayer mini-degree

07/06/2019 - Issue #283: Baldur's Gate 3, Lance.gg, and Dab Motors

31/05/2019 - Issue #282: W3C with WHATWG, loot box odds, and Zdog

24/05/2019 - Issue #281: Playdate, Tipsy Tower, and WebGL fluid simulation

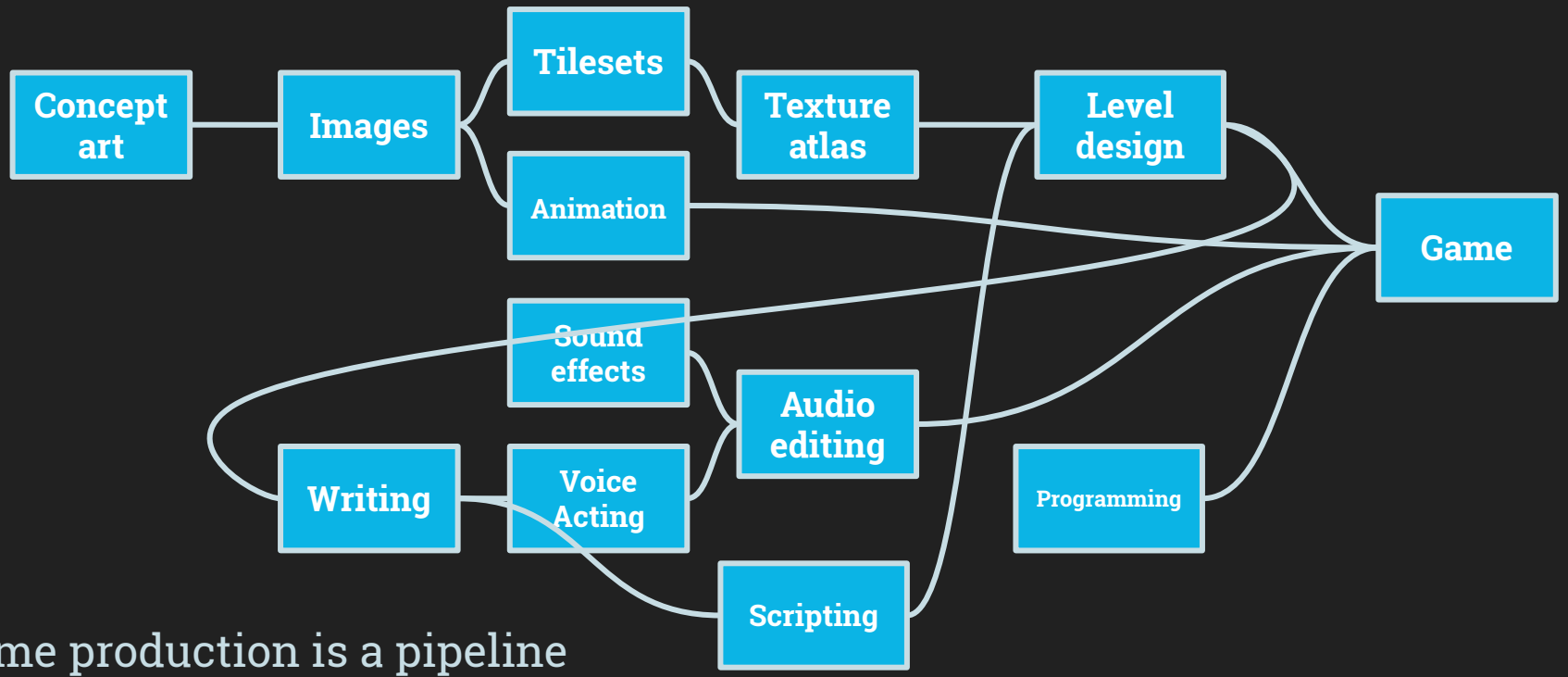
17/05/2019 - Issue #280: Phaser MMORPG, Spoke, and Proxy

http://gamedevjsweekly.com/

Break

What goes into a game?

Asset Management

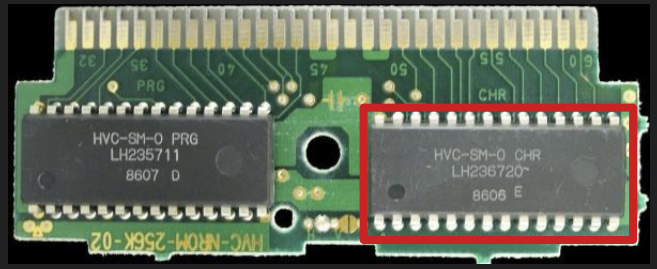


Game production is a pipeline

Assets have to be put into the game

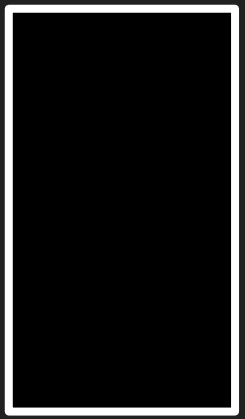
...that is, they need to be loaded before we can use them

- This is especially important with a web game, since your assets will be downloaded by the browser
- But it has always been a consideration for most kinds of games



ROM: i.e. the ultimate preload

In the before-times, assets were burnt directly into Read Only Memory



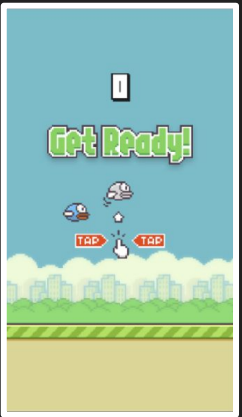
Boot

Load minimal assets, enough to start loading...

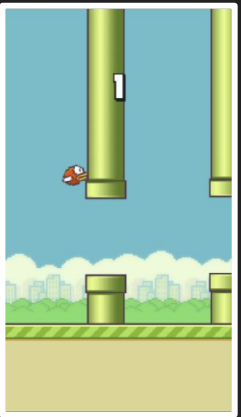
Load

...then load the rest of the assets

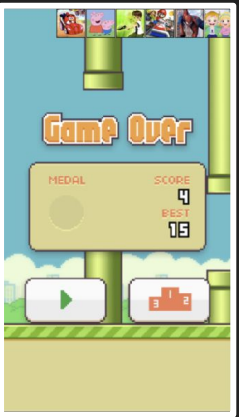
Menu



Pre-Game



Game

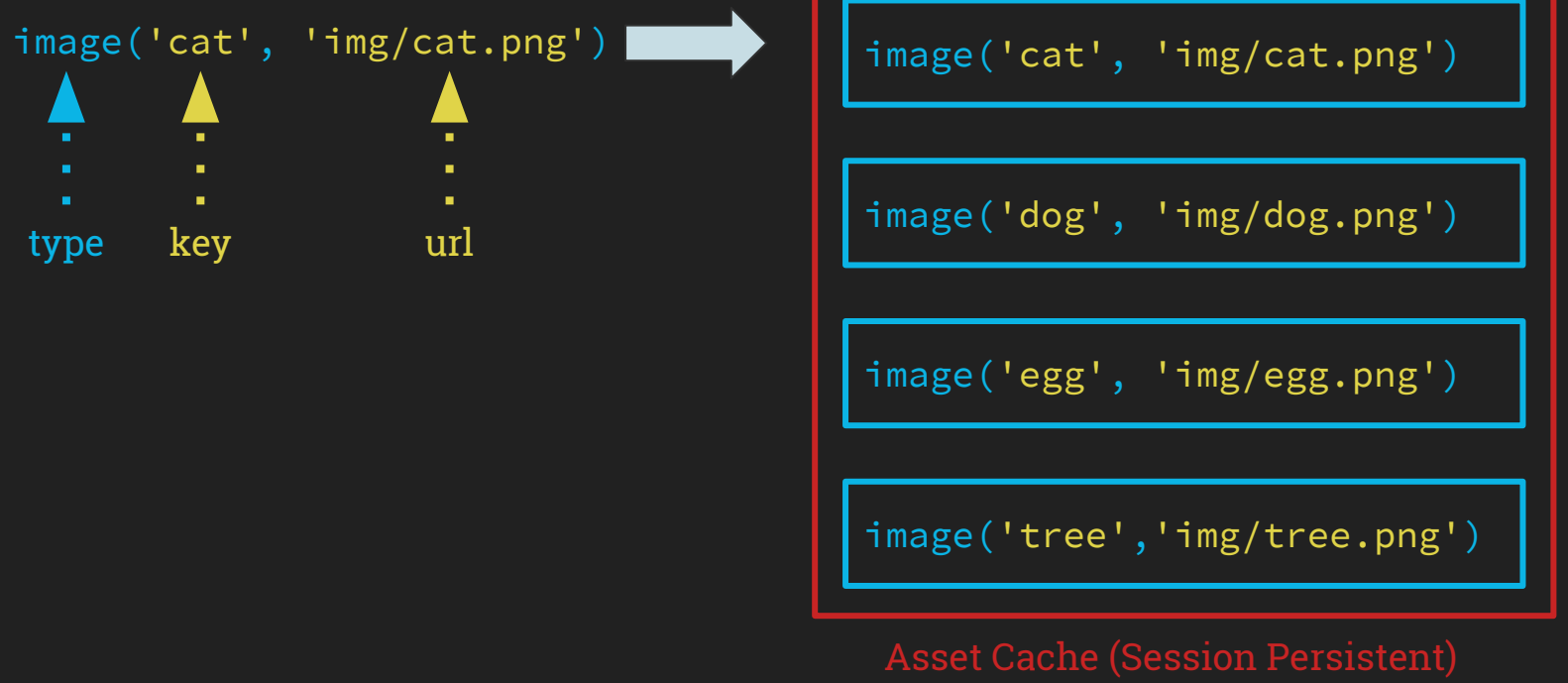


Game Over

Other games might need to load per level, per world, or on the fly

```
image('cat', 'img/cat.png')
```

Phaser requires us to load assets before we can use them in the game



Phaser requires us to load assets before we can use them in the game

```
image('cat', 'img/cat.png')
```

```
image('dog', 'img/dog.png')
```

```
image('egg', 'img/egg.png')
```

```
image('tree','img/tree.png')
```

add

Asset Cache (Session Persistent)

Phaser requires us to load assets before we can use them in the game

Load before adding

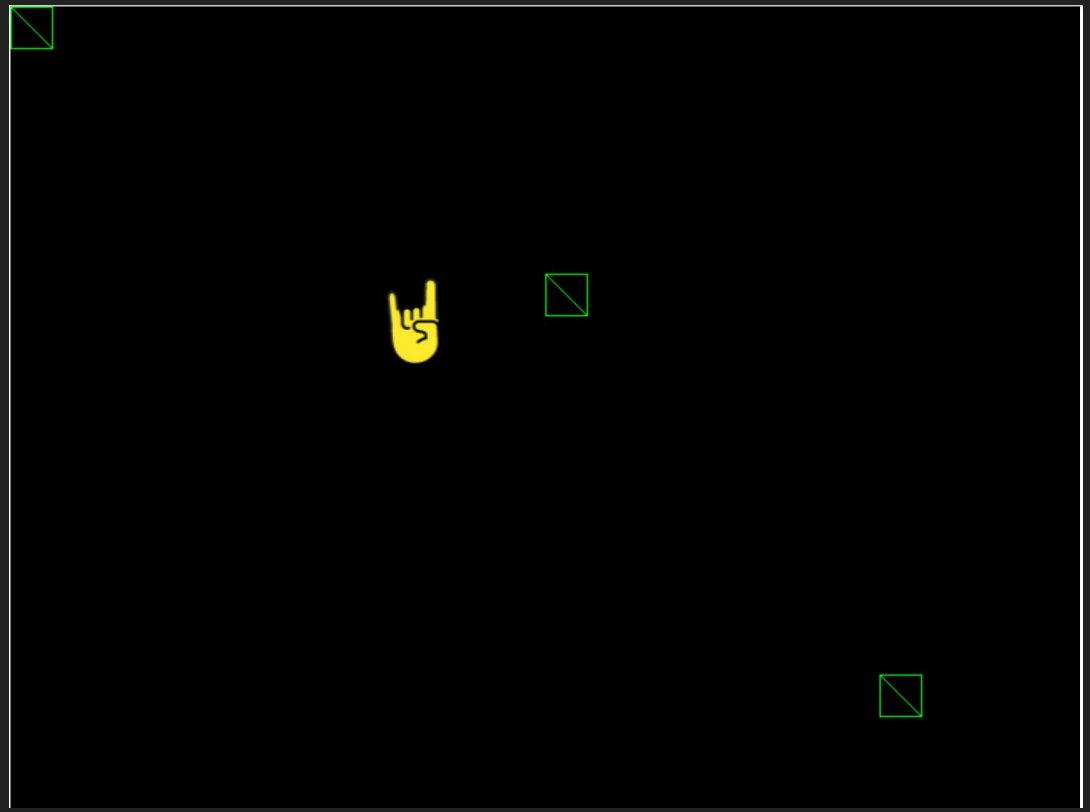
```
preload: function() {
    console.log('Boot: preload');
    // ready the asset we need to display during preload
    this.load.path = '../assets/img/';
    this.load.image('bar', 'bar.png');
    this.load.image('bg1', 'bg1.jpg');
},
```

Images can be loaded individually or in a batch

```
// load image assets
this.load.images(['diamond', 'firstaid', 'star', 'poo', 'bg2', 'leaf'],
    ['diamond.png', 'firstaid.png', 'star.png', 'poo.png', 'bg2.jpg', 'leaf.png']);
this.load.spritesheet('dude', 'dude.png', 32, 48);
```

Note the use of an array [ ]

Asset Cache Demo



What kind of assets can I use?

How do we load things into Phaser?

https://photonstorm.github.io/phaser-ce/

- ➔ Learning how to read the documentation is powerful
- ➔ Programming documentation follows standard conventions

Let's look up the things we suggested:
Slide 48: Q: No, really: What does go into a game?

Q: No, really: What does go into a game?

Images

Sounds

UI

Textures

Sprites / Models

Fonts

Story

Asset Types (In Phaser)

- Images
  - Sprites
  - Sprite Sheets
  - Texture Atlases
  - Tile Sprites
- Tile Maps
- Audio
  - Audio Sprite
- bitmapFont
- Video
- Shader
- Data
  - XML
  - Text
  - JSON
  - Binary
  - Physics
- Resource Pack
- JavaScript

Loading Images

Reference name
(you provide this)

Where is the file located?

```
1 // load an image
2 game.load.image('key', 'path/file.png')
3
4 // add an image
5 game.add.image(x, y, 'key');
6
```

Where do you want
to put the image?

Sprites

A Sprite is a moveable image

Back in the ancient times, game devices had specific hardware support for sprites.

They kind of float on top of the background images.

In Phaser, sprites give us a way to add:

- Motion
- Physics
- Input handling
- Events
- Animation
- Camera culling
- Etc.

Most of your on-screen visuals will be sprites.

Loading Sprites

Loaded as an image

```
// load a sprite
game.load.image('key', 'path/file.png')

// add a sprite
game.add.sprite(x, y, 'key');
```

added as a sprite

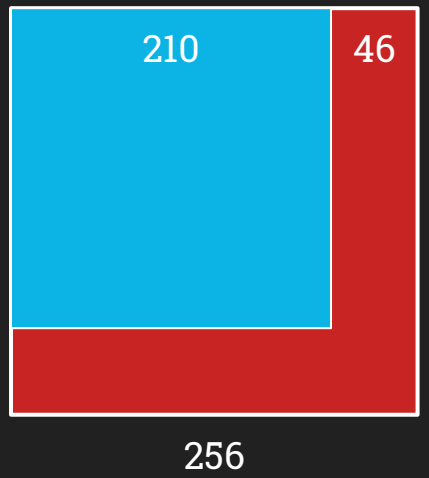
Which image format should I use?

For web games, mostly .png

- ➔ Lossless compression
- ➔ Transparency
- ➔ Widely supported

In your future career you'll also encounter things like DDS (to store compressed textures at multiple scales) or EXR (to store High Dynamic Range image data) but that's less relevant here.

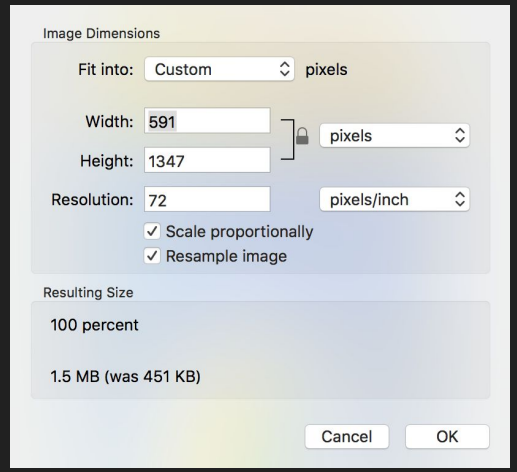
8
16
32
64
128
256
512
1024
2048
4096
...



Images sizes should be in powers of 2

Because of the way computers store things in memory, they will be padded or stretched to fit, or will otherwise be slower to load.

https://www.katsbits.com/tutorials/textures/make-better-textures-correct-size-and-power-of-two.php



32 x 73
8 KB

64 x 146
28 KB

150 x 342
144 KB

250 x 570
384 KB

Having your assets prepared before loading will improve speed and use less memory.

Sprite Sheets

Uniform grid layout of sprite frame data.

Putting many animations into one image reduces load time.

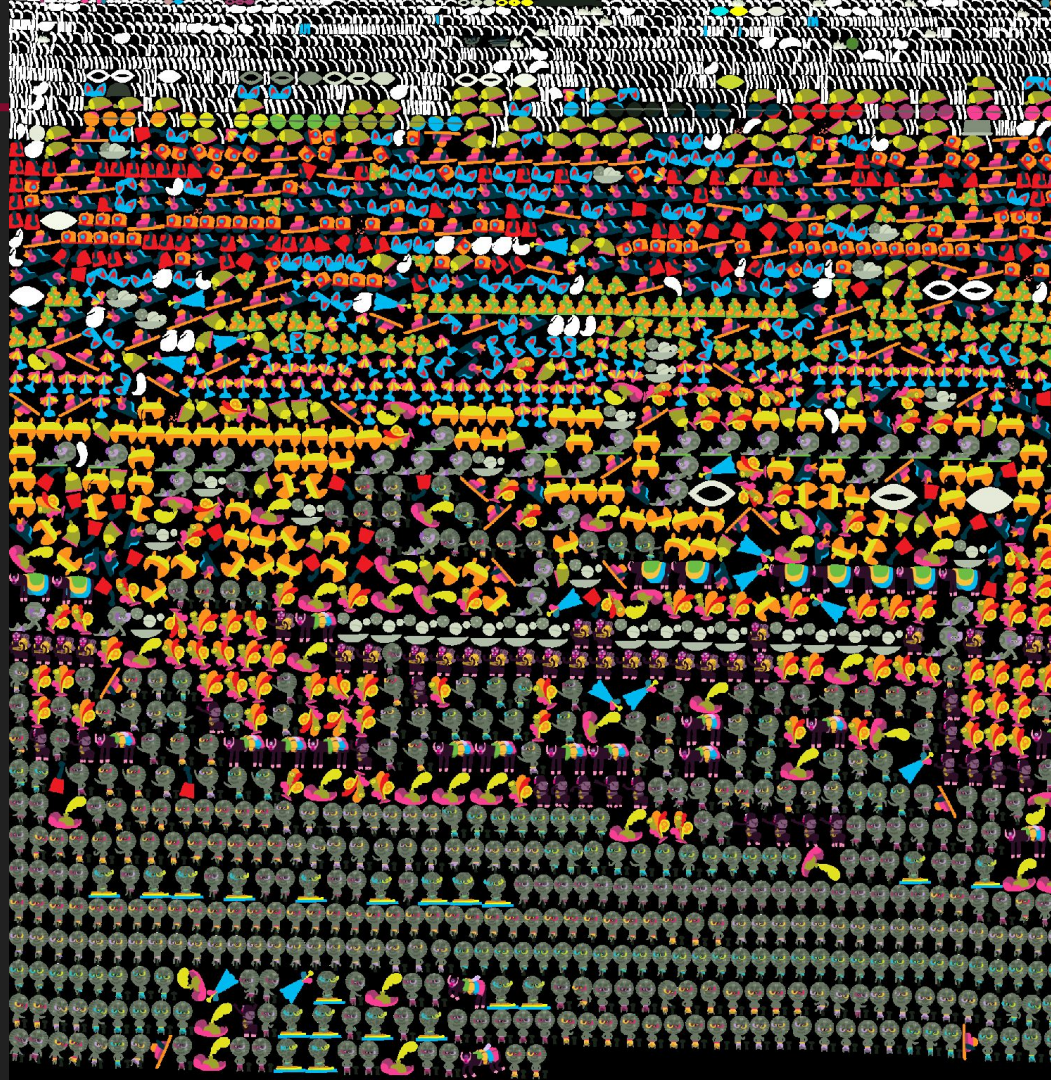
```
// load a sprite sheet
game.load.spritesheet('key', 'path/file.png', frameW, frameH);

// add a sprite (default frame)
game.add.sprite(x, y, 'key');
```

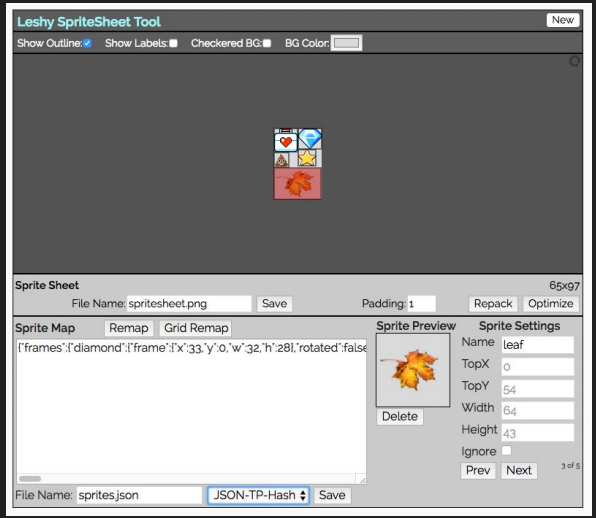
Texture Atlas

Non-uniform arrangement of sprite frame data

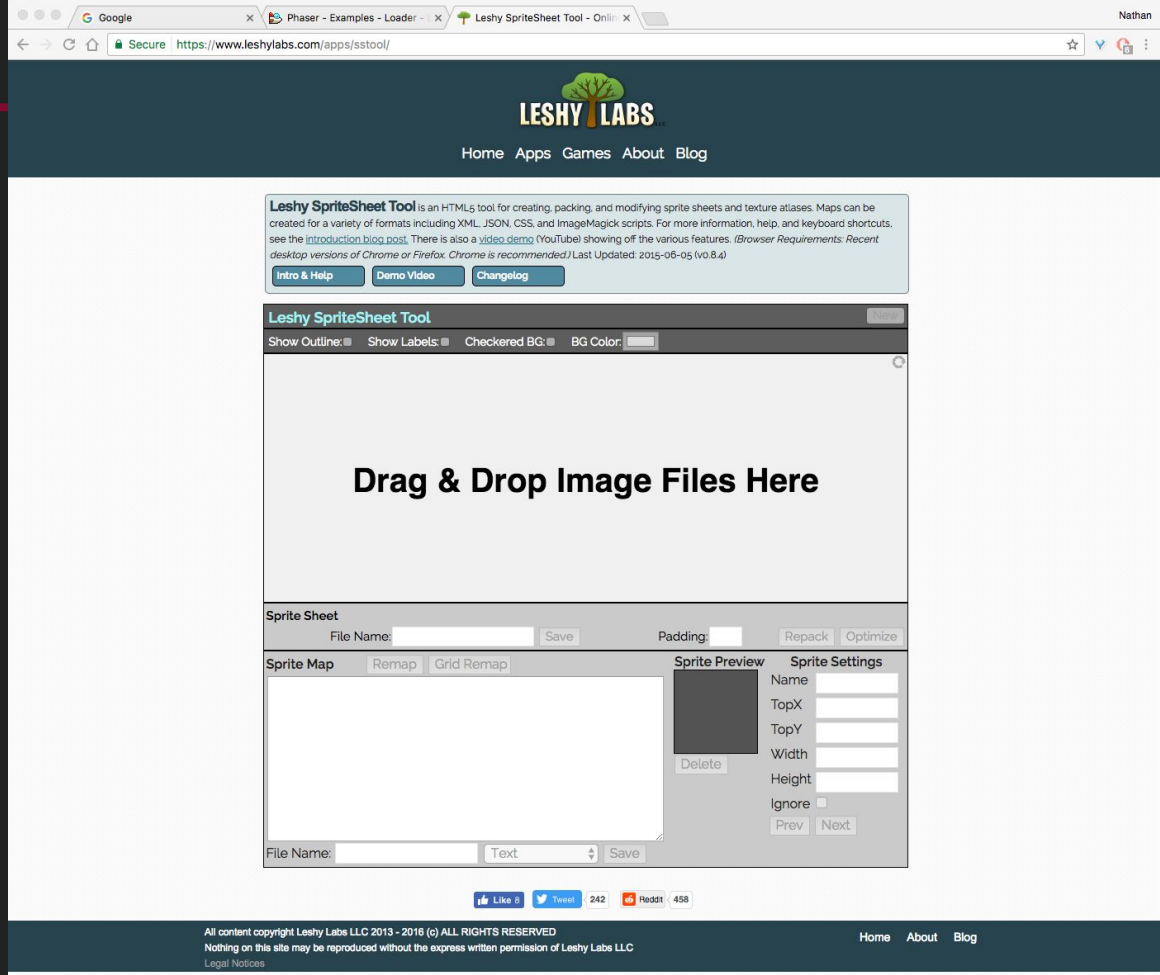
- Less memory and bandwidth
- Each element only drawn once
- Not all frames need to be the same size
- Refer to frames by name rather than index



CMPM 120



[leshylabs.com/apps/sstool/](https://www.leshylabs.com/apps/sstool/)



+

JSON File

```
// load a texture atlas
game.load.atlas('key', 'path/file.png', 'path/file.json');

// add a sprite
game.add.sprite(x, y, 'key', 'frame_name');
```

The texture atlas needs both an image and a JSON data file

JavaScript Object Notation (JSON)

- A file format that is human readable
- Built out of…
  - Attribute/Data pairs
  - Arrays
- Lots of things other than JavaScript can read it
- Common for communicating data on the web

```
{
    "attribute": "data",
    "also": ["can", "be", "nested"]
}
```

Tile Sprites

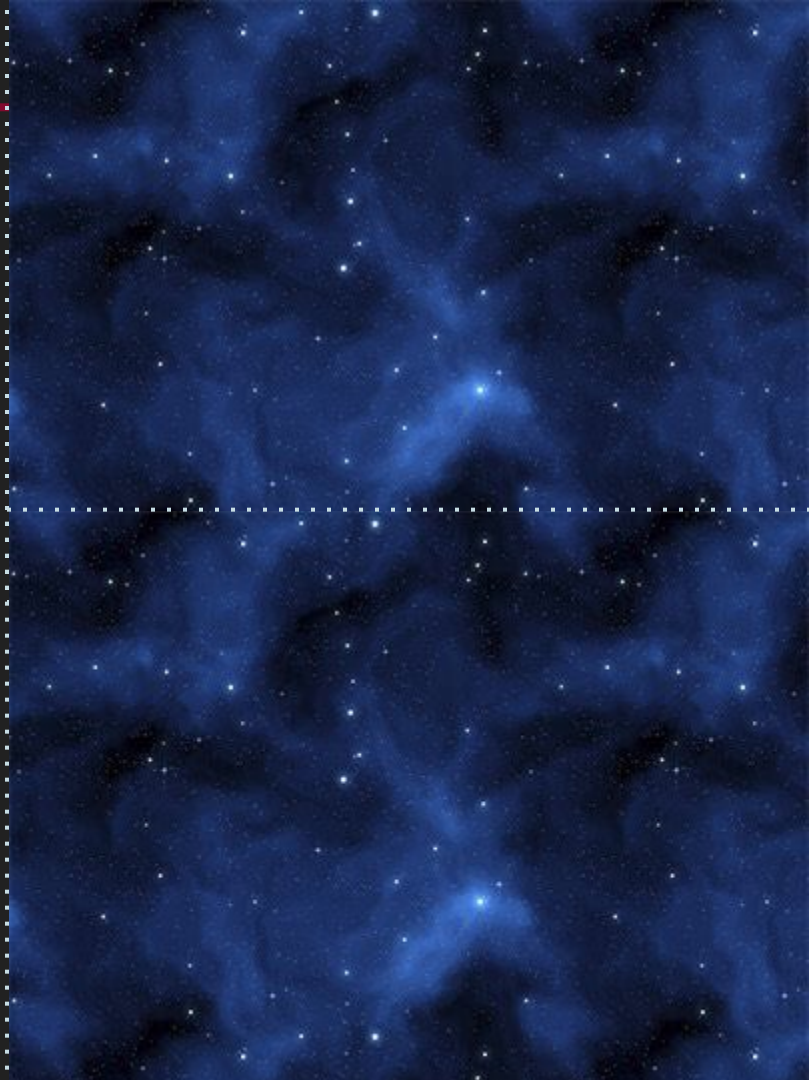
A sprite with a repeating texture that can be scrolled and scaled independently of the sprite itself.

Good for seamlessly looping backdrops.
(endless runner, scrolling shooter, etc.)

```
// load a tile sprite
game.load.image('key', 'path/file.png');

// add a tile sprite
game.add.tileSprite(x, y, w, h, 'key');
```

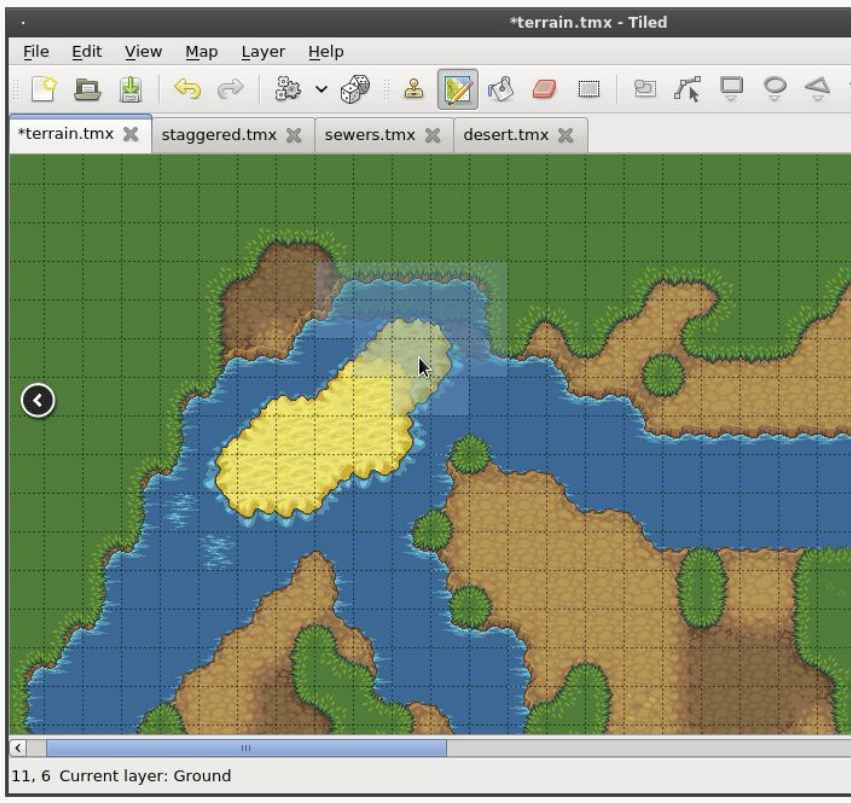
It's a good idea to match the dimensions of your tiled image



Tile Maps

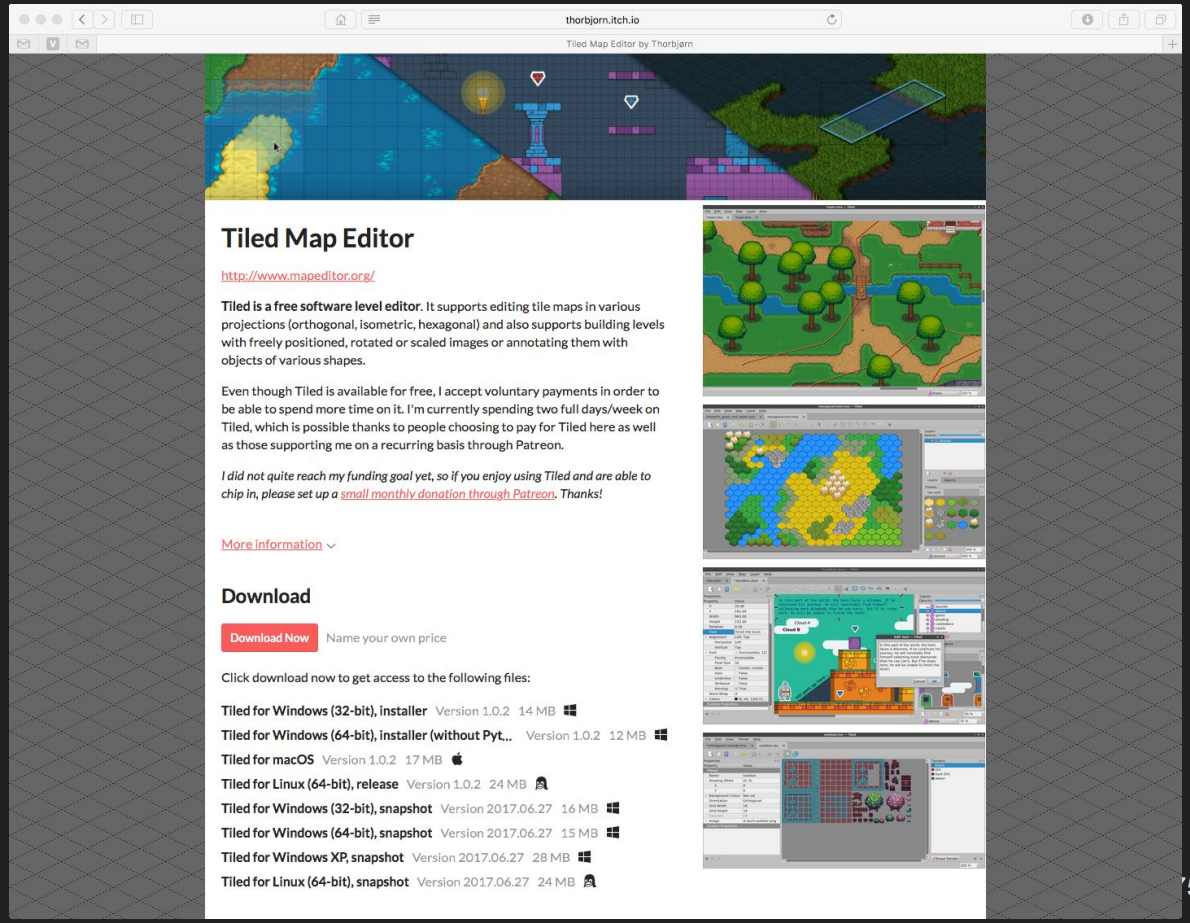
A popular technique in 2D game development that builds larger structures using grid-based elements called tiles.

Good for fast and memory efficient world building.



Tiled Map Editor

https://www.mapeditor.org/



Audio

All the sound data your game will need, whether for one-off sound effects or background music.

Sound adds texture and depth to your games. *Don't neglect it.*

An array of fallback options

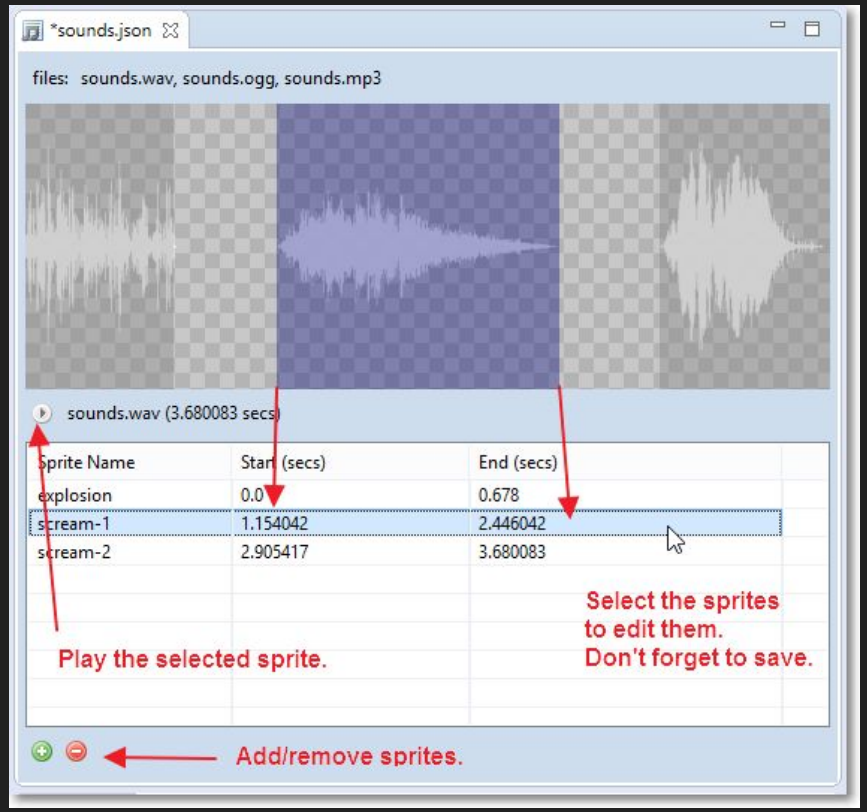
```
// load audio files
game.load.audio('key', ['file.mp3', 'file.ogg']);

// add audio
game.add.audio('key');
```

Audio Sprites

A single audio file that can be split into individual sound “sprites,” as in a sprite sheet. Requires a separate file.

Good for cross-browser compatibility.



Endless Runner

Your big individual project will be to make an endless runner.

It's not officially announced yet, but I wanted to give you a chance to start thinking about it.