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
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 a 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.
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();
}
Q: With this basic loop, how fast will the game state advance?

```java
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...

```java
while (true) {
    processInput();
    update();
    render();
}
```

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

```java
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

```java
while (true) {
    processInput();
    update();
    render();
}
```

Slower hardware will run slower and faster

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

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

   ```javascript
   function mainLoop() {
       console.log("calling mainLoop");
       update();
       draw();
       window.requestAnimationFrame(mainLoop);
   }
   ```

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

2. Add `mainLoop()` as a callback once

   ```javascript
   window.requestAnimationFrame(mainLoop);
   ```
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.


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 to 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...

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

```javascript
/**
 * The core game loop.
 *
 * @method Phaser.Game#update
 * @protected
 * @param {number} time - The current time as provided by RequestAnimationFrame.
 * @/ 
 * @method Phaser.Game#update
 * @param {number} time
 * @returns {void}
 */

function (time) {
    this.time.update(time);

    if (this._kickstart) {
        this.updateLogic(this.time.desiredFpsMult);
        this.updateRender(this.time.slowMotion * this.time.desiredFps);
        this._kickstart = false;
    }

    return;
}

// if the logic time is spiraling upwards, skip a frame entirely
if (this._spiral > 1 && !this.forceSingleUpdate) {
    // cause an event to warn the program that this CPU can't keep up with the current desiredFps rate
    if (this.time.time > this._nextFpsNotification) {
        // only permit one fps notification per 10 seconds
        this._nextFpsNotification = this.time.time + 10000;
    }
}
```
Phaser's logic update sequence:

```javascript
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.
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
CMPM 120

High Score (OS)

Boot → Menu → Pre-Game → Game → Game Over
Credits

Title

Spawn

Play

Modal Menu

Game Over

Legacy
```
1  // the simplest Phaser game object instance
2  var game = new Phaser.Game(800, 600, Phaser.AUTO, '', { preload: preload, create: create });
```
An alternate way to define Phaser's game object

```javascript
var config = {
    width: 800,
    height: 600,
    renderer: Phaser.AUTO,
    antialias: true,
    multiTexture: true,
    state: {
        preload: preload,
        create: create,
        update: update
    }
};

var game = new Phaser.Game(config);
```
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
    }
}
Boot
- init
- preload
- create

Menu
- preload
- create
- update

Pre-Game
- init
- preload
- create
- update

Game
- init
- preload
- create
- update
- shutdown

Game Over
- preload
- create
- update
- shutdown
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
Solution
Add some simple input logic.
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
Welcome to A Very Capable Game

Press SPACEBAR to begin

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
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

- Games made with Phaser
- Requirements
- Download Phaser
- Getting Started
- Building Phaser
- Support Phaser
- Phaser World Newsletter
- Contributing
- Change Log

Made With Phaser

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

Phrase Smalls

https://github.com/photonstorm/phaser-examples
https://codepen.io/collection/AMbZqY/
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:

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

ROM: i.e. the ultimate preload
Load minimal assets, enough to start loading...

...then load the rest of the assets
Other games might need to load per level, per world, or on the fly.
Phaser requires us to **load** assets before we can use them in the game.

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

<table>
<thead>
<tr>
<th>type</th>
<th>key</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.
Load before adding

```javascript
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

```javascript
// 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

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?

Reference name (you provide this)

Where is the file located?
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

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

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

- Loaded as an image
- 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.
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.

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.

```javascript
// 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
leshylabs.com/apps/sstool/
The texture atlas needs both an image and a JSON data file.

```javascript
// 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');
```
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

```json
{
    "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.)

```javascript
// 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/

Tiled is a free software level editor. It supports editing tile maps in various projections (orthogonal, isometric, hexagonal) and also supports building levels with freely positioned, rotated or scaled images or annotating them with objects of various shapes.

Even though Tiled is available for free, I accept voluntary payments in order to be able to spend more time on it. I'm currently spending two full days/week on Tiled, which is possible thanks to people choosing to pay for Tiled here as well as those supporting me on a recurring basis through Patreon.

I did not quite reach my funding goal yet, so if you enjoy using Tiled and are able to chip in, please set up a small monthly donation through Patreon. Thanks!

More information →

Download

Click download now to get access to the following files:

- Tiled for Windows (32-bit) Installer  Version 1.0.2  34 MB
- Tiled for Windows (64-bit), Installer (without Python)  Version 1.0.2  12 MB
- Tiled for macOS  Version 1.0.2  17 MB
- Tiled for Linux (64-bit), release  Version 1.0.2  24 MB
- Tiled for Windows (32-bit), snapshot  Version 2017.06.27  16 MB
- Tiled for Windows (64-bit), snapshot  Version 2017.06.27  15 MB
- Tiled for Windows XP, snapshot  Version 2017.06.27  28 MB
- Tiled for Linux (64-bit), snapshot  Version 2017.06.27  24 MB
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.*

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

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

An array of fallback options
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.